CumulusCI Documentation

Release 3.85.0

Cumulus Suite Development Team

Mar 14, 2024

CONTENTS

1 Introduction **1**
- 1.1 Automation with CumulusCI . . . 1
- 1.2 The Product Delivery Model . . . 2
- 1.3 Anyone Can Use CumulusCI . . . 2
- 1.4 Where Does CumulusCI Fit in the Toolchain? . . . 2
- 1.5 Why Is It Called CumulusCI? . . . 3
- 1.6 Learn More Through Demos . . . 3

2 Key Concepts **5**
- 2.1 Packages . . . 5
- 2.2 Projects . . . 5
- 2.3 Tasks and Flows . . . 6
- 2.4 Project Structure . . . 7
- 2.5 Project Orgs & Services . . . 8

3 Get Started **9**
- 3.1 Install CumulusCI . . . 9
- 3.2 Set Up SFDX . . . 12
- 3.3 Connect to GitHub . . . 13
- 3.4 Work On an Existing CumulusCI Project . . . 14
- 3.5 Start a New CumulusCI Project . . . 15
- 3.6 Convert an Existing Salesforce Project . . . 17

4 The `cci` Command Line **19**
- 4.1 Basic Operation . . . 19
- 4.2 List Tasks, Flows, and Plans . . . 20
- 4.3 Task Info and Options . . . 20
- 4.4 Flow Info and Options . . . 21
- 4.5 Plan Info and Options . . . 22
- 4.6 Run Tasks and Flows . . . 23
- 4.7 Access and Manage Orgs . . . 25
- 4.8 Manage Services . . . 25
- 4.9 Troubleshoot Errors . . . 26

5 Configure CumulusCI **29**
- 5.1 `cumulusci.yml` Structure . . . 29
- 5.2 Task Configurations . . . 29
- 5.3 Flow Configurations . . . 34
- 5.4 Scratch Org Configurations . . . 38
- 5.5 Configuration Scopes . . . 39

5.6 Advanced Configurations . . . 40
5.7 Troubleshoot Configurations . . . 48

6 Manage Scratch Orgs **51**
6.1 What Is an Org in CumulusCI? . . . 51
6.2 Set Up the Salesforce CLI . . . 51
6.3 Predefined Orgs . . . 52
6.4 Create a Scratch Org . . . 52
6.5 List Orgs . . . 53
6.6 Set a Default Org . . . 53
6.7 Open Orgs in the Browser . . . 53
6.8 Delete Scratch Orgs . . . 53
6.9 Configure Predefined Orgs . . . 54
6.10 Import an Org from the Salesforce CLI . . . 54
6.11 Use a Non-Default Dev Hub . . . 55

7 Connect Persistent Orgs **57**
7.1 The `org connect` Command . . . 57
7.2 Verify Your Connected Orgs . . . 58
7.3 Global Orgs . . . 58
7.4 Use a Custom Connected App . . . 59

8 Develop a Project **61**
8.1 Set Up a Dev Org . . . 61
8.2 List Changes . . . 61
8.3 Retrieve Changes . . . 62
8.4 List and Retrieve Options . . . 62
8.5 Push Changes . . . 63
8.6 Run Apex Tests . . . 63
8.7 Set Up a QA Org . . . 63
8.8 Manage Dependencies . . . 65
8.9 Use Tasks and Flows from a Different Project . . . 72

9 Automate Data Operations **73**
9.1 Sample Data . . . 73
9.2 Low level datasets . . . 74
9.3 The Lifecycle of a Dataset . . . 75
9.4 Defining Datasets . . . 75
9.5 Custom Settings . . . 82
9.6 Dataset Tasks . . . 82
9.7 Generate Fake Data . . . 86

10 Acceptance Testing with Robot Framework **89**
10.1 Get Started . . . 89
10.2 Run Your First Test . . . 90
10.3 So Why Robot? . . . 92
10.4 The Robot Framework Advantage . . . 92
10.5 Write a Sample Robot Test Case . . . 93
10.6 Suite Setup and Teardown . . . 96
10.7 Generate Fake Data with Faker . . . 97
10.8 Create Custom Keywords . . . 97
10.9 Create a Resource File . . . 98
10.10 Create a Simple Browser Test . . . 100
10.11 Combine API Keywords and Browser Tests . . . 102
10.12 Run an Entire Test Suite . . . 103

10.13 Learn More About Robot Framework . . . 105

11 Continuous Integration **133**
11.1 CumulusCI Flow . . . 133
11.2 CumulusCI in GitHub Actions . . . 133
11.3 Other CI Systems and Servers . . . 133
11.4 Testing with Second-Generation Packaging . . . 134
11.5 Further Reading . . . 134

12 Release Managed and Unlocked Packages **157**
12.1 Release a First-Generation Managed Package . . . 157
12.2 Release a Second-Generation Managed Package . . . 159
12.3 Release an Unlocked Package . . . 162
12.4 Extend NPSP and EDA with Second-Generation Packaging . . . 165
12.5 Generate Release Notes . . . 166
12.6 Manage Push Upgrades . . . 167

13 Manage Unpackaged Configuration **169**
13.1 Roles of Unpackaged Metadata . . . 169
13.2 Unpackaged Metadata Folder Structure . . . 170
13.3 Namespace Injection . . . 171
13.4 Retrieve Unpackaged Metadata . . . 173
13.5 Customize Config Flows . . . 174

14 Reference **175**
14.1 Cheat Sheet . . . 175
14.2 Tasks Reference . . . 178
14.3 Flow Reference . . . 300

15 About CumulusCI **315**
15.1 History . . . 315
15.2 Contribute to CumulusCI . . . 418

CHAPTER

ONE

INTRODUCTION

CumulusCI helps development teams build great applications on the Salesforce platform by automating org setup, testing, and deployment.

1.1 Automation with CumulusCI

If your product development lifecycle and release process is anything like ours at Salesforce.org, it's complex. You're managing multiple packages, dependencies, orgs, and release versions. Not to mention managing org metadata and all the setup operations that need to run in the right sequence, before or after a package is installed, to create a properly configured org.

For example, Nonprofit Success Pack (NPSP) is one of Salesforce.org's flagship open source products. NPSP is a large, complex application with many different components. It consists of six managed packages (five dependencies plus itself) with multiple dependency relationships. Using automation, all five dependent packages are deployed in the right sequence; the unpackaged record types for the Account and Opportunity objects are delivered; and the final configurations to make the customers' experience better, such as setting up Global Actions and delivering translations, are performed. Biweekly NPSP releases are easy for new customers to install, with all the right configuration and without requiring end users to work through a lengthy setup guide.

The CumulusCI suite of tools is part of the not-so-secret sauce that makes it possible for Salesforce.org to build and release products at high volume, velocity, and quality. CumulusCI automation runs throughout the Salesforce development lifecycle, starting from feature branches through the delivery of the latest release.

- The CumulusCI command-line interface, `cci`, runs single-action tasks and multiple-action flows for development and testing.
- MetaCI uses CumulusCI flows to build Salesforce managed packages from GitHub repositories.
- MetaDeploy automates setup and configuration of customer orgs.

You can use the very same automation used internally by Salesforce.org to quickly:

- Build sophisticated orgs with dependencies automatically installed.
- Load and retrieve sample datasets to make your orgs feel like a production environment.
- Apply transformations to existing metadata to tailor orgs to your specific requirements.
- Run builds in continuous integration systems.
- Create end-to-end browser tests and set up automation using Robot Framework.

The automation defined using CumulusCI is portable. It's stored in a version control repository and can be run from your local command line, from a continuous integration system, or from a customer-facing installer. CumulusCI can run automation on scratch orgs created using the Salesforce CLI, or on persistent orgs like sandboxes, production orgs, and Developer Edition orgs.

Finally, by way of introduction, CumulusCI is more than just a set of tools. It represents our holistic approach to product development. Rather than focusing on just the Org Development Model or the Package Development Model, Salesforce.org has implemented its own *Product Delivery Model* using CumulusCI.

1.2 The Product Delivery Model

The Product Delivery Model focuses on the customer experience, not on the technical artifacts you're delivering. When building a product, there are detailed technical considerations for whether an individual component is best distributed within a package, or as additional unpackaged metadata, or as setup automation that runs before or after a package is installed. It's not uncommon for managed packages that don't use the Product Delivery Model to require customers to perform manual configuration steps that can take hours, or even days, to complete. The Product Delivery Model lets teams develop configurations directly into automated workflows, making it possible to deliver a first-class, fully configured product to the customer.

CumulusCI automation, which makes it easy to create products that span multiple package repositories and include complex setup operations, best implements the Product Delivery Model, along with MetaDeploy and other applications in the CumulusCI suite.

1.3 Anyone Can Use CumulusCI

Salesforce.org uses CumulusCI to develop products for our nonprofit and education constituents — both public, open source products such as NPSP and commercial managed package products developed in private GitHub repositories. But anyone developing on the Salesforce platform can use CumulusCI. It supports both open source and private development, and building managed package products or org implementations.

Automation defined using CumulusCI can support all roles on a project.

- *Developers* can create new development environments for different feature branches.
- *Quality engineers* can create test environments from feature branches and managed package installs.
- *Documentation teams* can create environments to interact with new features and retrieve screenshots.
- *Product managers* can create environments to interact with new features and provide feedback on future work.
- *Release engineers* can create beta and final releases and push them to subscriber orgs.
- *Partners* can create their own projects on top of your package.
- *Customers* can install the product and get set up using the same automation steps used during development and QA.

1.4 Where Does CumulusCI Fit in the Toolchain?

Developers often ask whether CumulusCI competes with or replaces Salesforce DX, the Salesforce command line interface (CLI) for development, testing, and continuous integration. It doesn't. Like Salesforce DX, CumulusCI is designed to maintain the source of truth for a project in a version-controlled repository, and to make it as easy as possible to set up an org from scratch. CumulusCI uses the Salesforce CLI to perform operations such as creating scratch orgs, and is an alternative to bash scripts for running sequences of Salesforce CLI commands.

CumulusCI builds on top of the commands provided by the Salesforce CLI, and helps to manage and orchestrate them into a simple, straightforward user experience. CumulusCI implements a complete development, test, and release process that comes with a standard library of functionality, while the Salesforce CLI is a lower-level toolbelt that drives particular workflows within the overall process.

For non-developers, knowing Salesforce DX isn't a requirement for using CumulusCI. Neither is knowing Python, the language CumulusCI is written in (in the same way that most Salesforce DX users don't need to know Node.js). If you're going to get fancy with CumulusCI customizations, only then does Python come in handy.

1.5 Why Is It Called CumulusCI?

Before there was the toolset known today as CumulusCI, there was a product that would go on to become Nonprofit Success Pack (NPSP). This product had the code name Cumulus. Early on, continuous integration (CI) tools were created for the Cumulus product. This tooling expanded in scope and scale to eventually become CumulusCI. Even though it's used for much more than CI, and for many more products than NPSP, the name has stuck.

1.6 Learn More Through Demos

Love demos? These no-audio screencasts show how to use CumulusCI from a command line.

Initialize a fresh CumulusCI project.

Retrieve metadata from a Salesforce org and save it in GitHub.

Manage sample or test data.

Customize flows and use CumulusCI for QA.

For a narrated demo, see Jason Lantz's PyCon 2020 presentation (00:36 through 00:54).

CHAPTER

TWO

KEY CONCEPTS

Let's review some important concepts when building and testing features using CumulusCI.

2.1 Packages

CumulusCI works well with both managed package projects and org implementations. However, packages always play a role in how projects are built and deployed.

A *package* is a container for something as small as an individual component or as large as a sophisticated application. After creating a package, you can distribute it to other Salesforce users and organizations, including those outside your company.

Unmanaged packages are typically used to distribute open-source (non-proprietary) features or application templates to provide developers with the basic building blocks for an application. After the components are installed from an unmanaged package in a specific org, it's what's known as an *org implementation*. These freshly installed components can be edited by the owners of the implementation. The developer who created and uploaded the unmanaged package has no control over the installed components, and can't change or upgrade them.

Managed packages are typically used by Salesforce partners to distribute and sell applications to customers. They are proprietary code that can be upgraded and deployed only by the developer that built them. To ensure seamless upgrades, managed packages don't allow certain destructive changes, such as deleting objects or fields.

In CumulusCI, packages are built and deployed via projects.

2.2 Projects

When you work with CumulusCI, you do so inside a *project*. A project is an individual Git repository that contains both Salesforce metadata and CumulusCI automation (such as tasks and flows) that builds and releases the project. If you are building multiple packages, we strongly recommend organizing each package as a separate project in its own repository.

Important: CumulusCI's standard library assumes that there is one package per repository, so it will work best if you follow this convention.

It's important to note that a project doesn't have to contain a package. For example, a project can deliver unpackaged metadata, deliver automation but no metadata at all, or provide test data for QA. A project can contain the entirety of a product offered to customers, or be just one of multiple projects that combine to form a complete product.

To sum up, although a project doesn't require a package, a package requires a project to be built and deployed.

2.3 Tasks and Flows

CumulusCI uses a framework of *tasks* and *flows* to organize the automation that is available to each project.

Tasks are units of automation. A task can perform a deployment, load a dataset, retrieve data from an org, install a managed package, or do many other things. CumulusCI ships with scores of tasks in its standard library. You can run `cci task list` to view them all.

Popular task commands include:

- `cci task list`: Review the tasks available in a project.
- `cci task info <name>`: Learn more about a task `<name>` and how to configure its options.
- `cci task run <name> --org <org>`: Run the task `<name>` against the org `<org>`.

For example, the `run_tests` task executes Apex unit tests. If you have an org called `dev`, you can run this task against it with the command `cci task run run_tests --org dev`.

Many operations in CumulusCI, including creating new orgs, use flows. Flows are ordered sequences of tasks (and even other flows!) that produce a cohesive outcome, such as an org that's configured to suit a workflow like development, QA, or product demonstration.

Popular flow commands include:

- `cci flow list`: Review the flows available in a project.
- `cci flow info <name>`: Learn more about the flow `<name>` and the tasks it contains.
- `cci flow run <name> --org <org>`: Run the flow `<name>` against the org `<org>`.

For example, the `dev_org` flow sets up an org for development purposes. If you have an org called `dev`, you can run this flow against it with the command `cci flow run dev_org --org dev`.

Many of the most common flows you'll work with in CumulusCI are designed to build and configure specific orgs for you. Here's a few of the most common flows that build orgs.

- `dev_org`: This flow builds an unmanaged org designed for development use. It's typically used with an org whose configuration is `dev` or `dev_namespaced`.
- `qa_org`: This flow builds an unmanaged org designed for testing. It's typically used with an org whose configuration is `qa`.
- `install_beta`: This flow builds a managed org with the latest beta release installed, for projects that build managed packages. It's typically used with an org whose configuration is `beta`.
- `install_prod`: This flow builds a managed org with the latest release installed, for projects that build managed packages.
- `push_upgrade_org`: This flow builds a managed org that starts with the latest release installed and available for all Profiles. It then upgrades the package dependencies and the package itself to their latest betas, installing upgrades for System Administrators only, and runs `config_qa` to set up the org for testing. This simulates a subscriber push upgrade for non-System Administrator users. It's typically used with an org whose configuration is `release`.

CumulusCI derives the library of tasks and flows available for any project by combining its internal standard library with your customizations in `cumulusci.yml`. Customizations can add new tasks and flows, customize the way tasks behave, and extend, combine, and modify flows to better suit the project's needs. We cover customization in depth in the *Configure CumulusCI* section.

2.4 Project Structure

2.4.1 Project Directory

The project directory is the root of your CumulusCI project. Because each project is linked to a single GitHub repository, CumulusCI knows which project you are working on by the current working directory of your shell.

Tip: Avoid headaches by making sure you're in the correct repository for your project before running project-specific commands. Otherwise, your project produces an error. (**Check your repo first** when troubleshooting in CumulusCI and potentially save yourself an extra trip to this guide.)

In order to be used as a CumulusCI project, a directory must both be a Git repository and contain a `cumulusci.yml` configuration file. We cover how to get set up with a new or existing CumulusCI project in the *Get Started* section.

2.4.2 `cumulusci.yml`

The `cumulusci.yml` file defines a project's automation. It contains all the customizations and configurations that pertain to your project's lifecycle. It can encompass everything from customizing the shapes of scratch orgs to configuring tasks and flows.

Learn more about customizing CumulusCI automation in the *Configure CumulusCI* section.

2.4.3 `force-app` (or `src`)

The main body of the project's code and metadata lives in the default package directory, which is the `force-app` directory for Salesforce DX-format projects and the `src` directory for Metadata API-format projects. `force-app` defines what's included when you release a managed package from your CumulusCI project. (Or when you release an unlocked package, or if you're not releasing a package at all but running the `deploy` task to get the metadata into an org in unmanaged form.)

2.4.4 `orgs` directory

The `.json` files found in the `orgs` directory define the Salesforce DX org configurations that are available to the project. See *Manage Scratch Orgs* for more information.

2.4.5 `datasets`

Each project can have one or more datasets: on-disk representations of record data that can be inserted into Salesforce orgs, and that can also be modified and re-captured during the evolution of the project. Datasets are stored in the `datasets` directory. Learn more about datasets in *Automate Data Operations*.

2.4.6 robot

Robot Framework provides browser automation for end-to-end testing. Each project contains a `robot` directory, which stores the project's Robot Framework test suites. New projects start with a simple Robot test case that creates a Contact record.

While Robot Framework is used primarily for automated browser testing, it can also be harnessed to help configure orgs where other strategies and APIs are insufficient.

See *Acceptance Testing with Robot Framework* for more information.

2.4.7 unpackaged metadata

As we touched upon earlier, a project doesn't just encompass the contents of a managed package or a single deployment. It also includes *unpackaged metadata*: extra bundles of Salesforce metadata that further tailor an org or complete the product.

In a CumulusCI project, all unpackaged metadata is stored in subdirectories within the `unpackaged` directory. Unpackaged metadata plays multiple roles, including preparing an org for installing packages, adding more customization after the package or application is deployed, and customizing specific orgs that are used in the product's development process.

Learn more in the *Manage Unpackaged Configuration* section.

2.5 Project Orgs & Services

Orgs and services are external, authenticated resources that each project uses. CumulusCI makes it easy to connect orgs and services to a single project, or to use them across many projects.

2.5.1 Orgs

Each project has its own set of orgs, including active scratch orgs, persistent orgs like a production or packaging org, and predefined scratch org configurations. CumulusCI securely stores org authentication information in its keychain, making it easy to access connected orgs at any time. The `cci org list` command shows all of the orgs connected to a project. Orgs can also be shared across multiple projects.

Configuring orgs in CumulusCI is powerful, but comes with some complexity. For details, see *Manage Scratch Orgs* and *Connect Persistent Orgs*.

2.5.2 Services

Services represent external resources used by CumulusCI automation, such as access to a GitHub account or a MetaDeploy instance. Services are usually, but not always, connected to CumulusCI across projects as part of the global keychain. The command `cci service list` shows you which services are connected in the context of the current project.

Global services are easy to use and share. We recommend that you use them as much as possible. However, services can also be connected at the project level, which means that they're scoped to a single project and cannot be shared.

For more information on configuring services via the `cci` command line see the *Manage Services* section.

CHAPTER

THREE

GET STARTED

3.1 Install CumulusCI

Tip: These installation instructions assume some familiarity with entering commands into a terminal. If that's completely new to you, we recommend visiting the CumulusCI Setup module on Trailhead for a step-by-step walkthrough.

3.1.1 On macOS

Homebrew is a prerequisite for installing CumulusCI on macOS. Follow the instructions on the Homebrew website to install Homebrew before continuing.

Install via `pipx`

`pipx` ensures that CumulusCI and its dependencies are installed into their own Python environment separate from other Python software on your computer. **We cannot recommend it enough!**

First, install `pipx` with these commands:

```
$ brew install pipx
$ pipx ensurepath
```

After `pipx` installs, install CumulusCI:

```
$ pipx install cumulusci
```

Note: Upgrading Homebrew's Python version can result in a `bad interpreter` error. To resolve this, run `pipx reinstall cumulusci`.

When finished, *verify your installation*.

3.1.2 On Linux

Install via `pipx`

`pipx` ensures that CumulusCI and its dependencies are installed into their own Python environment separate from other Python software on your computer. **We cannot recommend it enough!**

Installation instructions for `pipx` can be found here.

After `pipx` installs, install CumulusCI:

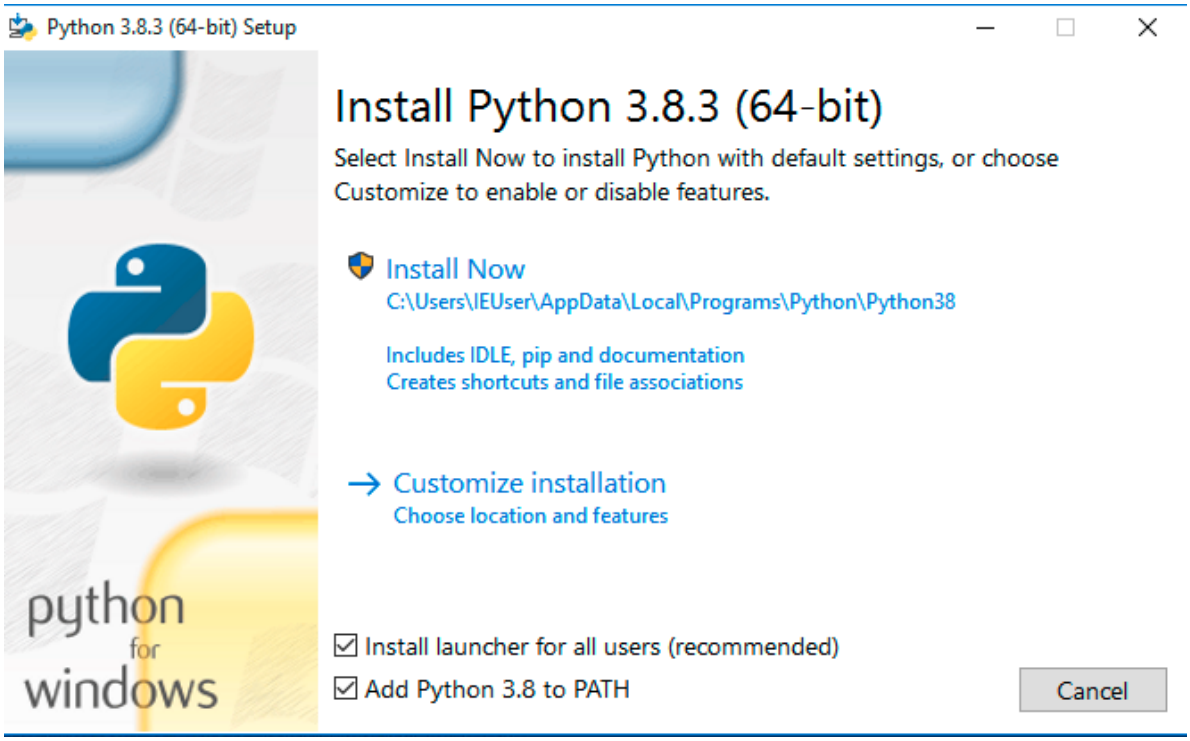
```
$ pipx install cumulusci
```

When finished, *verify your installation*.

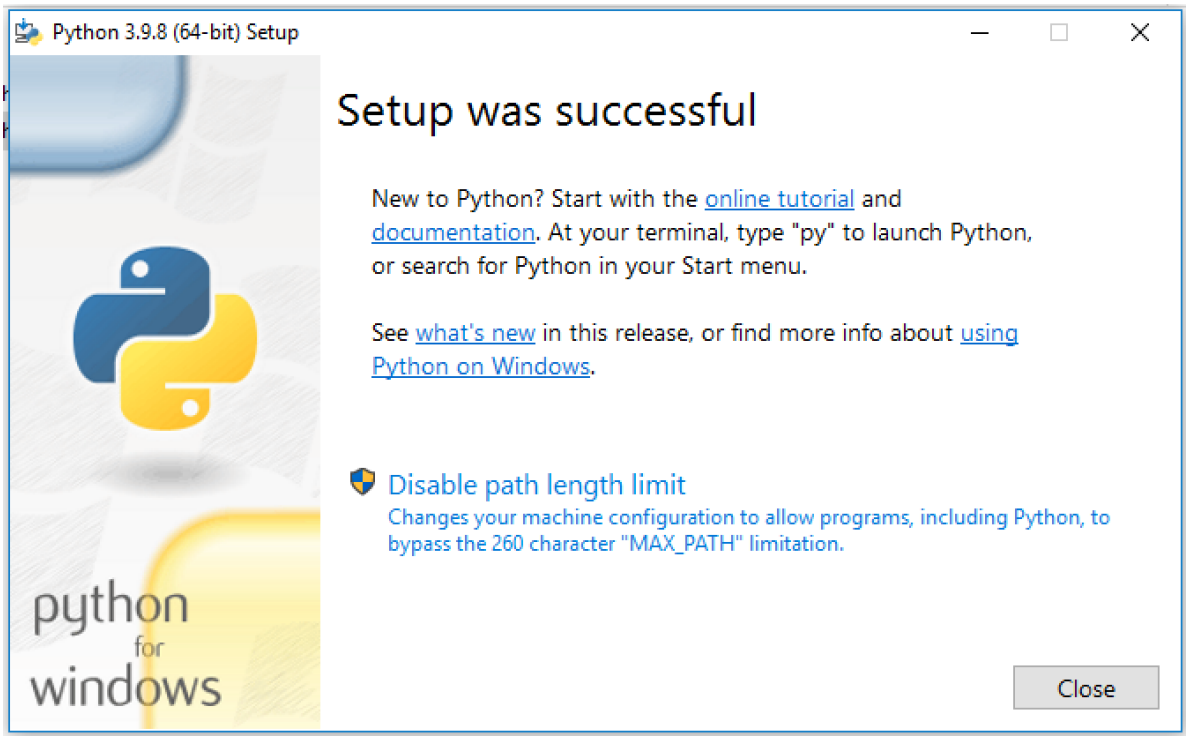
3.1.3 On Windows

Install Python 3

1. Go to the Python downloads page.
2. Download the latest Python 3.9 release. Most users select the "Windows Installer (64-bit)" link, but it depends on your particular computer setup.
3. Install using the installation wizard.
   - Select **Add Python <version> to PATH**.
   - Click "Install Now".



4. On the screen entitled "Setup was successful," click the "Disable path length limit" button (if it's present).



Install via `pipx`

`pipx` ensures that CumulusCI and its dependencies are installed into their own Python environment separate from other Python software on your computer. **We cannot recommend it enough!**

Open your preferred terminal application (such as cmd.exe on Windows). If your terminal is already open, close it and reopen it. Enter the following commands:

```
$ python -m pip install --user pipx
$ python -m pipx ensurepath
```

Close your current command prompt and open a new one to verify that `pipx` was successfully installed:

```
pipx --version
```

Look for a version number after entering this command, such as: `0.12.3.1`.

If you encounter an error, such as `'pipx' is not recognized as an internal or external command, operable program or batch file.`, you can try *updating your environment variables manually*.

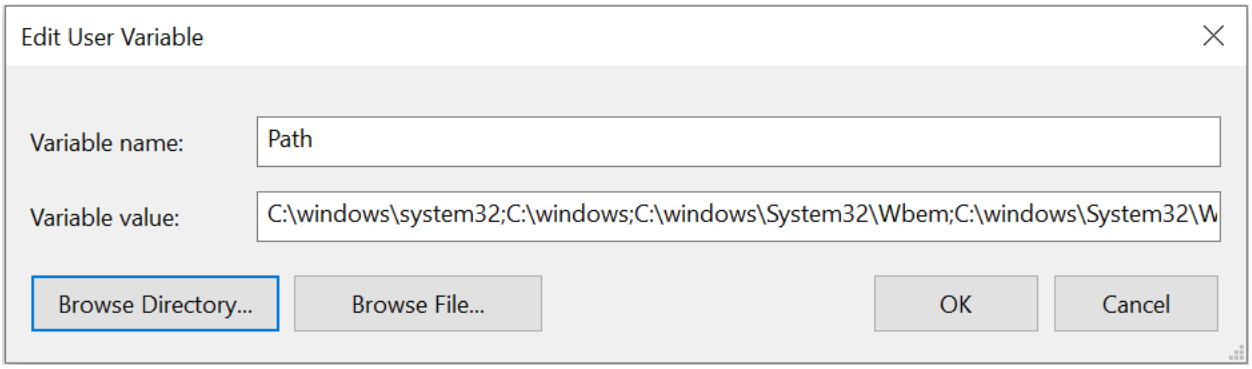
Finally, install CumulusCI with this command:

```
pipx install cumulusci
```

When finished, *verify your installation*.

Update Environment Variables Manually

1. Click Start and search for `edit environment variables`, or open System properties and click `Advanced system settings`.
2. Click the `Environment Variables` button.
3. To change System variables, you need non-restricted access (administrator rights) to your machine. Add these paths to your PATH environment variable:
   - `%USERPROFILE%\.local\bin`
   - `%USERPROFILE%\AppData\Roaming\Python\Python38\Scripts`



Note: Be precise when entering these paths. Add them at the very end of the Variable Value already in place. Separate each path by a semicolon (;) with no space between path names.

3.1.4 Verify Your Installation

In a new terminal window, verify that CumulusCI installed correctly by running `cci version`.

```
$ cci version
CumulusCI version: 3.29.0 (/path/to/bin/cci)
Python version: 3.8.5 (/path/to/bin/python)

You have the latest version of CumulusCI.
```

You can also use this command to check whether your CumulusCI installation is up to date.

Still need help? Feel free to submit a question on our Trailblazer community group.

3.2 Set Up SFDX

Scratch orgs created by Salesforce DX allow teams to work efficiently in individual, fully-configured environments that are easy to create and destroy. We recommend working with scratch orgs throughout the development process.

To set up Salesforce DX:

1. Install Salesforce CLI
2. Enable Dev Hub Features in Your Org
3. Connect SFDX to Your Dev Hub Org - Be sure to use the `--setdefaultdevhubusername` option!

If you have the `sfdx` command installed, are connected to your Dev Hub, and set the `defaultdevhubusername` config setting (use `sfdx force:config:list` to verify), you're now ready to use `cci` with `sfdx` to build scratch orgs.

Important: SFDX supports multiple Dev Hubs, so CumulusCI uses the one set as `defaultdevhubusername` when creating scratch orgs.

Tip: For a detailed introduction on how to set up Salesforce CLI and Visual Studio Code to work with CumulusCI, review the Build Applications with CumulusCI module on Trailhead.

Learn more about Salesforce DX at https://developer.salesforce.com/platform/dx.

3.3 Connect to GitHub

In order to allow CumulusCI to work with your repositories in GitHub, connect GitHub as a service in `cci`. Simply run:

```
$ cci service connect github mygithub
```

to open a browser and authenticate with your GitHub account using the one-time code shown by the CLI. Verify the GitHub service is connected by running `cci service list`:

```
[~/repos/cumulusci]-[feature/doc-reorg__get_started]
$ cci service list
Services
```

Name	Description	Configured
connected_app	A Connected App is required to connect to and run commands against persistent orgs. See http://cumulusci.readthedocs.io/en/latest/tutorial.html#creating-a-connected-app for more info.	
devhub	Configure which SFDX org to use as a Dev Hub for creating scratch orgs	✔
github	Configure connection for github tasks, e.g. Create Release	✔
metaci	Connect with a MetaCI site to run builds of projects from this repository	
metadeploy	Connect with a MetaDeploy site to publish installers from this repository	✔
apextestsdb	Configure connection for ApexTestsDB tasks, e.g. ApextestsdbUpload	
saucelabs	Configure connection for saucelabs tasks.	

After you've configured the `github` service, it's available to *all* CumulusCI projects.

Note: If your GitHub organization has enabled OAuth App access restrictions, CumulusCI will not be able to access organization repositories until it has been approved by an owner. You can use a personal access token instead.

Alternatively, you may create a new personal access token with both **repo** and **gist** scopes specified. (Scopes appear as checkboxes when creating the personal access token in GitHub.) Copy the access token to use when configuring the GitHub service.

Next, run the following command and provide your GitHub username and access token:

```
$ cci service connect github mygithub --username $GITHUB_USERNAME --token $GITHUB_TOKEN
```

Services are stored in the global CumulusCI keychain by default.

3.3.1 GitHub Enterprise Server Support

GitHub Enterprise Server Support is an experimental feature that may change.

As of CumulusCI version 3.66, a new GitHub service type is available named `github_enterprise` to allow connections to a GitHub Enterprise Server (GES). The `github_enterprise` service works like the `github` service but includes a `server_domain` field that must be supplied with the GES server domain.

In some cases, you may receive an error connecting to a GitHub Enterprise Server that states "a self-signed certificate is in the chain". This may happen when you need to validate server TLS certificates using the system's certificate authorities, instead of the set of CA certs that is bundled with requests. You may need to set a ENV variable `CUMULUSCI_SYSTEM_CERTS` to "True". See *`cumulusci-system-certs`*.

Here's an example command to configure a `github_enterprise` service:

```
$ cci service connect github_enterprise enterprise --token <access token> --username
↪<username> --email <email> --server_domain <repo url>
```

Limitations of *GitHub Enterprise Server Support*

The GitHub Enterprise Server service only allows one `github_enterprise` service to be configured per server domain. This means the access token you are using must have permission to access all required resources at a given repository location.

3.4 Work On an Existing CumulusCI Project

If you'd like to work on an existing CumulusCI project on GitHub, these are the prerequisites.

1. *Install CumulusCI*
2. Install Git
3. Clone the Project's GitHub Repository

Note: CumulusCI does not support projects stored on other Git hosts such as BitBucket or GitLab at this time.

You can change directories into the project's root directory and begin executing `cci` commands.

For example, `cci project info` shows information about the project:

```
$ cd cumulusci-test

$ cci project info
name: CumulusCI Test
package:
    name: CumulusCI Test
    name_managed: None
    namespace: ccitest
    install_class: None
    uninstall_class: None
    api_version: 33.0
git:
    default_branch: main
```

(continues on next page)

(continued from previous page)

```
    prefix_feature: feature/
    prefix_beta: beta/
    prefix_release: release/
    release_notes:
        parsers:
            1:
                class_path: cumulusci.tasks.release_notes.parser.GithubLinesParser
                title: Critical Changes
            2:
                class_path: cumulusci.tasks.release_notes.parser.GithubLinesParser
                title: Changes
            3:
                class_path: cumulusci.tasks.release_notes.parser.GithubIssuesParser
                title: Issues Closed
            4:
                class_path: cumulusci.tasks.release_notes.parser.GithubLinesParser
                title: New Metadata
            5:
                class_path: cumulusci.tasks.release_notes.parser.GithubLinesParser
                title: Deleted Metadata
    repo_url: https://github.com/SFDO-Tooling/CumulusCI-Test
test:
    name_match: %_TEST%
```

3.5 Start a New CumulusCI Project

If you'd like to start a new CumulusCI project, these are the prerequisites.

1. *Install CumulusCI*
2. Install Git
3. Install the Salesforce CLI

Run the following commands to create a directory with your project's name, navigate to said directory, and initialize it as a Git repository:

```
$ mkdir cci_project
$ cd cci_project
$ git init
```

Then, initialize the project as a CumulusCI project.

3.5.1 Project Initialization

Use the `cci project init` command within a Git repository to generate the initial version of a project's `cumulusci.yml` file. CumulusCI creates a customized `cumulusci.yml` file by first asking questions about your project.

Prompt	What's it for?
Project Info	The name is usually the same as your repository name.NOTE: Do **not** use spaces in the project name.
Package Name	CumulusCI uses an unmanaged package as a container for your project's metadata.Enter the name of the package you want to use.
Is this a managed package project?	Yes, if this project is a managed package.
Salesforce API Version	Which Salesforce API version does your project use? Defaults to the latest API version.
Which source format do you want to use? [sfdx | mdapi]	DX source format (aka "SFDX Format") stores data under the `force-app` directory.Metadata API format is the "older" format and stores data under the `src` directory.
Are you extending another CumulusCI project such as NPSP or EDA?	CumulusCI makes it easy to build extensions of other projects configured for CumulusCI like Salesforce.org's NPSP and EDA. If you are building an extension of another project using CumulusCI and have access to its GitHub repository, use this section to configure this project as an extension.
Default Branch	Your project's main/master branch in GitHub. Defaults to the branch that is currently checked out in your local repository.
Feature Branch Prefix	Your project's feature branch prefix (if any). Defaults to `feature/`.
Beta Tag Prefix	Your project's beta branch prefix (if any). Defaults to `beta/`.
Release Tag Prefix	Your project's release branch prefix (if any). Defaults to `release/`.
Test Name Match	The CumulusCI Apex test runner uses a SOQL `WHERE` clause to select which tests to run. Enter the SOQL pattern to match test class names. Defaults to match classes ending in `TEST`.
Do you want to check Apex code coverage when tests are run?	If yes, checks Apex code coverage when tests are run.
Minimum code coverage percentage	Sets the minimum allowed code coverage percentage for your project.

3.5.2 Verify Project Initialization

Verify successful project initialization by verifying that `cumulusci.yml` exists and has contents.

```
$ cat cumulusci.yml
project:
    name: SampleProjectName
    package:
        name: SamplePackageName
        namespace: sampleNamespace
    .
    .
    .
```

The `cumulusci.yml` file configures your project-specific tasks, flows, and customizations.

Once you're satisfied, add and commit it to your Git repository.

```
$ git add cumulusci.yml
$ git commit -m "Initialized CumulusCI Configuration"
```

3.5.3 Add Your Repo to GitHub

With your `cumulusci.yml` file committed, create a repository on GitHub for your new project and push your changes there, using whichever method you prefer.

- Our Community Project Development with CumulusCI module covers GitHub Desktop in the "Set Up the Project" section.
- If you prefer the command line, GitHub has excellent guides on both git remote and git push.

3.6 Convert an Existing Salesforce Project

Converting an existing Salesforce project to use CumulusCI can follow a number of different paths, depending on whether you're practicing the Org Development Model or the Package Development Model; whether or not you're already developing in scratch orgs; and the complexity of your project's dependencies on the org environment.

If you're developing in persistent orgs and not already using source control, you'll also need to retrieve the existing metadata from the org. Note that the processes of adopting source control and moving from persistent orgs to scratch orgs tend to look different for everyone, and these transitions sometimes require additional work.

You're welcome to discuss project conversion in the CumulusCI (CCI) Trailblazer group.

3.6.1 Project Setup

Before retrieving your project's metadata, set up the project's root directory.

- Create a project directory and initialize it as a Git repository as outlined in *start a new CumulusCI project*.
- Initialize the directory as a CumulusCI project as outlined in *project initialization*.

3.6.2 Retrieve Metadata from the Persistent Org

This documentation assumes that your project currently lives in a persistent org, such as a Developer Edition org or a Sandbox. We recommend retrieving metadata via the Metadata API (using the Salesforce CLI), followed by converting the source from Metadata API to SFDX format.

1. If the metadata you want to retrieve isn't already in a package, create one. See creating packages for more info.

Note: If your project contains a managed package, ensure that the package namespace matches the namespace you entered when running `cci project init`.

---

2. Run the retrieve command to extract your package metadata.

```
$ sfdx force:source:retrieve -n package_name /path/to/project/
```

That's it! You now have all of the metadata you care about in a single Git repository configured for use with CumulusCI. At this point *add your repo to GitHub*, or perhaps begin to *configure CumulusCI*.

3.6.3 Setup Scratch Orgs

To see how to use scratch orgs with CumulusCI see the *Manage Scratch Orgs* section.

3.6.4 Other Conversion Considerations

- If you or your team have been working with scratch org definition files for use with `sfdx`, see *Manage Scratch Orgs* for details on using them with CumulusCI.
- If you have metadata that you would like deployed pre- or post-deployment, see *Roles of Unpackaged Metadata*.
- If you have data that you need to include for testing or production purposes, see *Automate Data Operations*.

CHAPTER

FOUR

THE CCI COMMAND LINE

4.1 Basic Operation

Tip: If you're new to working with command line interfaces, the Install Visual Studio Code Trailhead module covers installing and opening a terminal window in Visual Studio Code.

After *installing CumulusCI*, use the `cci` command in your terminal or command prompt to interact with it.

On any platform, you can use the integrated terminal in Visual Studio Code. Alternately, on macOS, access the terminal via `Terminal.app`; on Windows, open `cmd.exe`; or on Linux, use your preferred terminal application.

To see all available commands, type `cci` in your terminal.

```
$ cci
Usage: cci [OPTIONS] COMMAND [ARGS]...

Options:
--help  Show this message and exit.

Commands:
error    Get or share information about an error
flow     Commands for finding and running flows for a project
org      Commands for connecting and interacting with Salesforce orgs
plan     Commands for getting information about MetaDeploy plans
project  Commands for interacting with project repository configurations
service  Commands for connecting services to the keychain
shell    Drop into a Python shell
task     Commands for finding and running tasks for a project
version  Print the current version of CumulusCI
```

To retrieve information on a specific command, type `cci <command>`.

Let's examine the `cci task` command:

```
$ cci task
Usage: cci task [OPTIONS] COMMAND [ARGS]...

Commands for finding and running tasks for a project

Options:
--help  Show this message and exit.
```

(continues on next page)

(continued from previous page)

```
Commands:
doc   Exports RST format documentation for all tasks
info  Displays information for a task
list  List available tasks for the current context
run   Runs a task
```

We can see that the `cci task` command has many useful subcommands, such as `cci task info`.

4.2 List Tasks, Flows, and Plans

CumulusCI ships with many standard tasks and flows. In addition, your project might have one or more MetaDeploy plans. The following commands list all available tasks, flows, and plans for a project:

```
$ cci task list
$ cci flow list
$ cci plan list
```

The tasks, flows, and plans listed are specific to the project directory that you're in when you run the command. For example, if you have a custom flow defined in your `cumulusci.yml` file for Project A, it will only be listed if you run `cci flow list` in Project A's root directory.

The tasks and flows are listed by their `group` attribute as specified in the `cumulusci.yml` file. It's easy to edit these groups as you see fit. Any modifications will be reflected in the `list` commands.

4.3 Task Info and Options

For additional information on task <name>, run either command:

```
$ cci task info <name>
$ cci task run <name> --help
```

Information about specific tasks includes:

- A description of the task.
- The Python class associated with this task.
- The syntax for running the command.
- Any options accepted or required by the task.

Each option available for a given task also lists:

- The syntax for the option (`--<name> value`).
- Whether the option is required or optional.
- A description of the option.

Let's examine the `util_sleep` task:

```
$ cci task info util_sleep
util_sleep

Description: Sleeps for N seconds

Class: cumulusci.tasks.util.Sleep

Command Syntax

    $ cci task run util_sleep

Options

    --seconds SECONDS
    Required
    The number of seconds to sleep
    Default: 5
```

4.4 Flow Info and Options

For additional information on flow <name>, run either command:

```
$ cci flow info <name>
$ cci flow run --help
```

Information about specific flows includes:

- A description of the flow.
- The ordered steps (and substeps) of a flow.

For example, listing the info for the `dev_org` flow shows that it's composed of three subflows: `dependencies`, `deploy_unmanaged`, and `config_dev`, and one task: `snapshot_changes`. The tasks and flows making up the three subflows are also listed.

```
$ cci flow info dev_org
Description: Set up an org as a development environment for unmanaged metadata
1) flow: dependencies [from current folder]
    1) task: update_dependencies
    2) task: deploy_pre
2) flow: deploy_unmanaged
    0) task: dx_convert_from
    when: project_config.project__source_format == "sfdx" and not org_config.scratch
    1) task: unschedule_apex
    2) task: update_package_xml
    when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3) task: deploy
    when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3.1) task: dx_push
    when: project_config.project__source_format == "sfdx" and org_config.scratch
    4) task: uninstall_packaged_incremental
    when: project_config.project__source_format != "sfdx" or not org_config.scratch
```

(continues on next page)

(continued from previous page)

```
3) flow: config_dev
    1) task: deploy_post
    2) task: update_admin_profile
4) task: snapshot_changes
```

4.5 Plan Info and Options

Your project may have one or more defined MetaDeploy plans, though none come preconfigured with CumulusCI. If you have plans, for additional information on plan <name>, run the following command:

```
$ cci plan info <name>
```

Information about specific plans includes:

- Configuration settings (slug, tier, etc)
- Messages
- Plan preflight checks
- Step preflight checks
- An ordered list of steps

By default all of the above information is displayed. You can display only the list of messages by using the command line option `--messages`

The following example shows the output of a typical plan, in this case a plan named 'config'.

```
$ cci plan info config
        Config

 Key        Value
 ──────────────────────
 YAML Key   config
 Slug       config
 Tier       secondary
 Hidden?    No

                                  Messages

 Type           Message
 ──────────────────────────────────────────────────────────────────────────
 Title          Express Setup Configuration Plan
 Preflight      This will install metadata configurations into your org.
 Post-install   Thanks for installing Advisor Link. Visit the Trailblazer Community
                for any questions about Advisor Link.
 Error          If you experience an issue with the installation, please post
                in the Trailblazer Community.

                              Plan Preflights

 Action   Message                            When
 ──────────────────────────────────────────────────────────────────────────
```

(continues on next page)

(continued from previous page)

```
error    My Domain must be enabled in       '.my.' not in
         your org before installation.      org_config.instance_url
error    Chatter must be enabled in your    not
         org before installation.           tasks.check_chatter_enabled()
error    Enhanced Notes must be enabled     not
         in your org before installation.   tasks.check_enhanced_notes_enab...

                              Step Preflights

Step   Action   Message   When
-------------------------------------------------------------------------------
4      skip              'PID_Customer_Community_Plus' not in
                         tasks.get_available_licenses()
5      skip              'PID_Customer_Community_Plus_Login' not in
                         tasks.get_available_licenses()

                                    Steps

Step   Name                                          Required   Recommended
-------------------------------------------------------------------------------
1      Express Setup - Additional Unpackaged         No         Yes
       Metadata
2      Express Setup - Sample Reports and            No         Yes
       Dashboards
3      Express Setup - Lightning App and Advisor     No         Yes
       Profile
4      Express Setup - Advisee Profile               No         Yes
5      Express Setup - Advisee Portal Profile        No         Yes
6      Express Setup - Permission Sets               No         Yes
7      Express Setup - Advisor Sharing Metadata      No         Yes
```

4.6 Run Tasks and Flows

Execute a specific task or flow with the `run` command.

```
$ cci task run <name> --org <org> [options]
$ cci flow run <name> --org <org> [options]
```

This command runs the task or flow <name> against the org <org>.

Tip: You can see a list of available orgs by running `cci org list`.

For example, the `run_tests` task executes Apex unit tests in a given org. Assuming there exists an org named `dev`, you can run this task against it with the command `cci task run run_tests --org dev`.

4.6.1 Get Help Running Tasks

If you're not certain about what a specific command does, use the `--help` flag to get more information.

```
$ cci task info <name> --help
```

When the `--help` flag is specified for a command, the output includes:

- A usage statement featuring the syntax that executes the command.
- A description of the command.
- The list of available options for use with the command.

```
$ cci task --help
Usage: cci task [OPTIONS] COMMAND [ARGS]...

Options:
--help  Show this message and exit.

Commands:
doc   Exports RST format documentation for all tasks
info  Displays information for a task
list  List available tasks for the current context
run   Runs a task
```

If you're just getting started with CumulusCI and aren't sure which of the many tasks and flows to use, don't worry. We show you specific tasks and flows in later sections of the documentation.

4.6.2 Specify Task Options When Running Flows

When executing a flow with `cci flow run`, you can specify options on specific tasks in the flow with the following syntax:

```
$ cci flow run <flow_name> -o <task_name>__<option_name> <value>
```

<flow_name> is the name of the flow to execute, <task_name> is the name of the task you wish to specify an option for, <option_name> is the option on the task you want to specify, and <value> is the actual value you want to assign to the task option.

For example, in the above output from `cci flow info dev_org` if we wanted to set the `allow_newer` option on the `update_dependencies` to `True`, we would use the following:

```
$ cci flow run dev_org --org dev -o update_dependencies__allow_newer True
```

Note: If the specified task executes more than once in the flow, it uses the given option value *each time it executes*.

If you want to configure specific task options on flows without explicitly listing them see *Configure Options on Tasks in Flows*.

4.7 Access and Manage Orgs

CumulusCI makes it easy to create, connect, and manage orgs. The `cci org` top-level command helps you work with orgs.

To learn about working with orgs in detail, read *Manage Scratch Orgs* and *Connect Persistent Orgs*.

4.8 Manage Services

Services represent external resources used by CumulusCI automation, such as access to a GitHub account or a MetaDeploy instance.

4.8.1 List Services

You can have CumulusCI show you a list of all possible services supported. Services that are not currently configured will be displayed in a dimmed row.

```
$ cci service list
```

4.8.2 Connect A Service

To connect a service to the global keychain (which we recommend for almost all situations) you can use:

```
$ cci service connect <service_type> <service_name>
```

If you wanted to connect to your personal GitHub account as a service you could use:

```
$ cci service connect github personal
```

CumulusCI will prompt you for the required information for the given service type.

If you want a service to onlye be available to a given project you can pass the `--project` flag.

```
$ cci service connect <service_type> <service_name> --project
```

4.8.3 Set a Default Service

The first service connected for a given service type is automatically set as the default service for that type. If you have multiple services connected for a given type and would like to set a new default use:

```
$ cci service default <service_type> <service_name>
```

4.8.4 Rename a Service

To rename a service use:

```
$ cci service rename <service_type> <old_name> <new_name>
```

4.8.5 Remove a Service

To remove a service use:

```
$ cci service remove <service_type> <service_name>
```

4.9 Troubleshoot Errors

Errors happen! That's why `cci` provides tools to extract error details so that they can be reported and triaged.

4.9.1 Report Error Logs

The `cci error gist` command sends the most recent log file to a GitHub gist so you can quickly and easily share logs with others. For this feature to work you need to make sure that your GitHub service is set up with the proper scopes.

The gist includes:

- The current version of `cci`
- The current Python version
- The path to the Python executable
- `sysname` of the host (such as Darwin)
- The machine name of the host (such as x86_64)
- The most recent log file (cci.log) that CumulusCI has created.

The URL for the gist is displayed in the terminal as output, and a web browser automatically opens a tab to the gist.

4.9.2 View Stack Traces

If you encounter an error and want more information on what caused it, the `cci error info` command displays the stack trace (if present) from the last command executed in CumulusCI.

Note: The stack trace displayed is a *Python* stacktrace. This is helpful for locating where CumulusCI encountered an error in the source code.

4.9.3 See Stack Traces Automatically

If you'd like to investigate bugs in CumulusCI, set the config option `show_stacktraces` to `True` under the `cli` section of `~/.cumulusci/cumulusci.yml`. It turns off suppression of stack traces.

Usage errors (such as incorrect command line arguments, missing files, and so on) don't show exception tracebacks because they are seldom helpful in that case.

For help with troubleshooting errors or stack traces, reach out to the CumulusCI team on the CumulusCI Trailblazer Community Group.

4.9.4 The `--debug` Flag

All CumulusCI commands can be passed the `--debug` flag, so that:

- Any calls to CumulusCI's logger at the debug level are shown.
- Outgoing HTTP requests are logged.
- If an error is present, the corresponding stack trace is shown, and the user is dropped into a post-mortem debugging session.

Note: To exit a debugging session, type the command `quit` or `exit`.

4.9.5 Log Files

CumulusCI creates a log file every time a cci command runs. There are six rotating log files (`cci.log, cci.log1...5`) with `cci.log` being the most recent. Log files are stored under `~/.cumulusci/logs` for Mac and Linux users, and `C:\Users\<Your User>\.cumulusci\logs` for Windows users.

By default, log files document:

- The last command that was entered by the user.
- All output from the command (including debug information).
- If a Python-level exception occurs, the corresponding stack trace.

If you want debug information regarding HTTP calls made during execution, you must explicitly run the command with the `--debug` flag set.

```
$ cci task run <name> --org <org> --debug
$ cci flow run <name> --org <org> --debug
```

CHAPTER

FIVE

CONFIGURE CUMULUSCI

The `cumulusci.yml` file is located in the project root directory. This is where you define project dependencies, configure new tasks and flows, customize standard tasks and flows for your project, and so much more!

5.1 `cumulusci.yml` Structure

A `cumulusci.yml` file contains these top-level sections.

- `project`: Contains information about the project's associated package (if any) and GitHub repository. This section is largely generated by running `cci project init`.

  If you need custom markup that's unique to your project, you can store it in a section called `custom` under `project`.

- `tasks`: Defines the tasks that are available to run in your project. See *Task Configurations* for configuration options in this section.
- `flows`: Defines the flows that are available to run in your project. See *Flow Configurations* for configuration options in this section.
- `sources`: Defines other CumulusCI projects whose tasks and flows you can use in automation. See *Tasks and Flows from a Different Project* for more information.
- `orgs`: Defines the scratch org configurations that are available for your project. See *Scratch Org Configurations* for configuration options in this section.
- `plans`: Contains any custom plans defined to install your project into a customer org. See the *metadeploy_publish* task for more information.

5.2 Task Configurations

Each task configuration under the `tasks` section of your `cumulusci.yml` file defines a task that can be run using the `cci task run` command, or included in a flow step. With a few simple changes to this section, you can configure build automation functionality to suit your project's specific needs.

5.2.1 Override a Task Option

If you repeatedly specify the same value for an option while running a task, you can configure CumulusCI to use that value as a default value.

For example: Let's enforce a 90% code coverage requirement for Apex code in your project. The `run_tests` task, which executes all Apex tests in a target org, can enforce code coverage at a given percentage by passing the `--required_org_code_coverage_percent` option.

```
run_tests:
    options:
        required_org_code_coverage_percent: 90
```

When the `tasks` section of the `cumulusci.yml` file specifies this option, CumulusCI overrides the default option with a value of `90`. Whenever this task is executed, its customized options apply, unless it's further configured for a particular flow step.

Verify the change by looking for a default option value when running `cci task info <name>`.

```
$ cci task info run_tests
run_tests

Description: Runs all apex tests

Class: cumulusci.tasks.apex.testrunner.RunApexTests

Command Syntax

    $ cci task run run_tests

Options
    .
    .
    .
  -o required_org_code_coverage_percent PERCENTAGE
    Optional
    Require at least X percent code coverage across the org following the test run.
    Default: 90
```

5.2.2 Add a Custom Task

To define a new task for your project, add the task name under the `tasks` section of your `cumulusci.yml` file.

For example, let's create a custom task named `deploy_reports` that deploys a set of reports stored in your project's unpackaged metadata located in `unpackaged/config/reports`.

First, look up the Python class associated with the standard task `deploy`. From there we see that the `deploy` task has a `class_path` value of `cumulusci.tasks.salesforce.Deploy`.

Store the task under the `tasks` section of the `cumulusci.yml` file.

```
deploy_reports:
    description: Deploy Reports
    class_path: cumulusci.tasks.salesforce.Deploy
    group: projectName
```

(continues on next page)

(continued from previous page)

```
    options:
        path: unpackaged/config/reports
```

Tip: Be sure to include the value we retrieved for `class_path`. Also, consider adding a common `group` attribute to make it easier to see the tasks specific to your project when running `cci task list`.

Congratulations! You created a new custom task in CumulusCI.

5.2.3 Custom Tasks in Python

You can use Python to implement entirely new functionality. You do so by putting it in the "tasks" subdiretory of your project. For example, you might write a file called `tasks/write_file_custom_task.py`.

```
from pathlib import Path

from cumulusci.core.tasks import BaseTask, CCIOptions
from cumulusci.utils.options import Field


class WriteFileCustomTask(BaseTask):
    class Options(CCIOptions):
        mypath: Path = Field(..., description="A filepath to be used by the task")
        mystring: str = Field("Hello", description="A string to be used by the task")

    parsed_options: Options

    def _run_task(self):
        file = self.parsed_options.mypath
        data = self.parsed_options.mystring
        file.write_text(data)
        self.logger.info(f"Wrote {data} to {file}")
```

You can make it available to the project by adding it under the `tasks` section of the `cumulusci.yml` file. If one does not exist, you can create one.

```
tasks:
    my_new_task:
        description: My custom task to write data to a file
            class_path: tasks.write_file_custom_task.WriteFileCustomTask
            group: projectName
```

This task will be accessible directly to this project. It will also be accessible to any other project that adds this as a source (see *Tasks and Flows from a Different Project*) with `allow_remote_code: True`.

Options for Custom Tasks

Task options are defined by declaring a nested `Options` class. This class must sublass `cumulusci.utils.options.CCIOptions`. These options are validated via the use of Pydantic models which are generated dynamically for each `Options` class. Each option can define its own type via either a standard library type or by utilizing a custom type from `cumulusci.utils.options`. Additionally the `Field()` function is useful for further customizing options. This can be imported from `cumulusci.utils.options` and used when defining individual options. It has the same features as the pydantic function.

The task above (`WriteFileCustomTask`) takes two options: (1) A defaulted string (myString), and (2) A required file path.

Once the options are defined, they can be accessed via the `parsed_options` property of the task.

Important: When the nested `Options` class is defined within your custom task (or is part of a class you inherit from), it restricts modifications to the `options` property of the task, making it read-only. To make any changes, you should instead modify the `parsed_options` property rather than the `options` property.

Some of the most commonly used types are:

- `pathlib.Path`: simply uses the type itself for validation by passing the value to Path(v);
- `FilePath`: like Path, but the path must exist and be a file
- `DirectoryPath`: like Path, but the path must exist and be a directory
- `MappingOption`: Parses pairs of values from a string in format "a:b,c:d"
- `ListOfStringsOption`: Parses a list of comma-separated strings from an argument in the format "abc,def,ghi"
- `JSON`: Parse a JSON string into a structure, including a deeply nested structure

Others can be found in the pydantic docs.

If you are comfortable with Python types, you can do very sophisticated parsing. For example:

```
from typing import Optional

from pydantic import Json

from cumulusci.core.tasks import BaseTask, CCIOptions
from cumulusci.utils.options import Field


class Person(CCIOptions):
    name: str
    children: Optional[dict[str, "Person"]]


People = dict[str, Person]


class ComplexOptionsCustomTask(BaseTask):
    class Options(CCIOptions):
        lineage: Json[People] | People = Field(..., description="Foo")

    parsed_options: Options
```

(continues on next page)

(continued from previous page)

```
    def _run_task(self):
        self.logger.info(f"Got {self.parsed_options.lineage}")
```

Which can parse this commmand line:

```
$ cci task run complex_options --lineage '{"Bob": {"name": "Bob Cat Sr", "children": {
↪"Bob": {"name": "Bob Cat Jr"}}}}'
    Got {'Bob': Person(name='Bob Cat Sr', children={'Bob': Person(name='Bob Cat Jr',␣
↪children=None)})}
```

Or this YAML:

```
complex_options:
    description: Description of the task
    class_path: tasks.complex_options_custom_task.ComplexOptionsCustomTask
    options:
        lineage:
            Bob:
                name: Bob Cat Sr
                children:
                    Bob:
                        name: Bob Cat Jr
```

5.2.4 Use Variables for Task Options

To reference a project configuration value within the `tasks` section of the `cumulusci.yml` file, use the `$project_config` variable.

For example, NPSP uses a variable for the project's namespace by setting a value of `$project_config.project__package__namespace`. This variable is then referenced in the project's custom `deploy_qa_config` task where it's passed as the value for the `namespace_inject` option.

Tip: A double underscore (__) refers to a subsequent level in the `cumulusci.yml` file.

```
deploy_qa_config:
    description: Deploys additional fields used for QA purposes only
    class_path: cumulusci.tasks.salesforce.Deploy
    group: Salesforce Metadata
    options:
        path: unpackaged/config/qa
        namespace_inject: $project_config.project__package__namespace
```

In this instance, CumulusCI replaces the variable with the value under project -> package -> namespace in the `cumulusci.yml` file. Here is the `project` section of NPSP's `cumulusci.yml` file specifying `npsp` as the namespace value.

```
project:
    name: Cumulus
    package:
```

(continues on next page)

(continued from previous page)

```
        name: Cumulus
        name_managed: Nonprofit Success Pack
        namespace: npsp
        api_version: 48.0
        install_class: STG_InstallScript
        uninstall_class: STG_UninstallScript
```

5.3 Flow Configurations

Each flow configuration listed under the `flows` section of your `cumulusci.yml` file defines a flow that can be run using the `cci flow run` command, or included as a step in another flow. With a few simple changes to this section, you can configure sophisticated build automation that execute workflows throughout your development lifecycle.

5.3.1 Add a Custom Flow

To define a new flow for your project, add the flow name under the `flows` section of your `cumulusci.yml` file. Let's define a new `greet_and_sleep` flow:

```
greet_and_sleep:
    group: projectName
    description: Greets the user and then sleeps for 5 seconds.
    steps:
        1:
            task: command
            options:
                command: echo 'Hello there!'
        2:
            task: util_sleep
```

This flow is comprised of two tasks: `command` greets the user by echoing a string, and `util_sleep` then tells CumulusCI to sleep for five seconds.

You can reference how flows are defined in the universal cumulusci.yml file.

5.3.2 Add a Flow Step

To add a step to a flow, first run `cci flow info <name>` to see the existing steps. In the following example we run this for the `dev_org` flow.

```
$ cci flow info dev_org
Description: Set up an org as a development environment for unmanaged metadata
1) flow: dependencies [from current folder]
    1) task: update_dependencies
    2) task: deploy_pre
2) flow: deploy_unmanaged
    0) task: dx_convert_from
    when: project_config.project__source_format == "sfdx" and not org_config.scratch
    1) task: unschedule_apex
    2) task: update_package_xml
```

(continues on next page)

(continued from previous page)

```
    when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3) task: deploy
    when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3.1) task: dx_push
        when: project_config.project__source_format == "sfdx" and org_config.scratch
    4) task: uninstall_packaged_incremental
    when: project_config.project__source_format != "sfdx" or not org_config.scratch
3) flow: config_dev
    1) task: deploy_post
    2) task: update_admin_profile
4) task: snapshot_changes
```

Of this flow's four steps, the first three are themselves flows, and the last is a task.

All *non-negative numbers and decimals* are valid as step numbers in a flow. You can add steps before, between, or after existing flow steps.

The following shows examples of values that you would use for the various scenarios:

- Add a step *before* step 1 by inserting a step number greater than or equal to zero and less than 1 (such as 0, 0.3, or even 0.89334).
- Add a step *between* steps 2 and 3 by inserting a step number greater than 2 or less than 3.
- Add a step *after* all steps in the flow by inserting a step number greater than 4.

You could also customize the `dev_org` flow to output an additional log line as its final step:

```
dev_org:
    steps:
        5:
            task: log
            options:
                line: dev_org flow has completed
```

5.3.3 Skip a Flow Step

To skip a flow step, set the task or flow for that step number to the value of `None`.

For example, to skip the fourth step of the `dev_org` flow, insert this code under the `flows` section of your `cumulusci.yml` file.

```
dev_org:
    steps:
        4:
            task: None
```

Important: The key `task` must be used when skipping a flow step that is a task. The key `flow` must be used when skipping a flow step that corresponds to a flow.

When CumulusCI detects a task or flow with a value of `None`, the task or flow is skipped.

```
2020-10-16 20:30:01: Starting execution
2020-10-16 20:30:01: ============================================================
2020-10-16 20:30:01:
2020-10-16 20:30:01: ************************************************************
2020-10-16 20:30:01: Skipping task: None
2020-10-16 20:30:01: ************************************************************
```

5.3.4 Replace a Flow Step

Replacing a flow step is easy; just note the name of the flow, step number, and task or flow you would like to run on the given step.

For example, to replace the default fourth step of the `dev_org` flow with a custom task that loads data into a dev environment, specify the custom task to run instead.

```
dev_org:
    steps:
        4:
            task: load_data_dev
```

5.3.5 Configure Options on Tasks in Flows

Specify options on specific tasks in a flow with this syntax:

```
<flow_to_modify>:
    steps:
        <step_number>:
            flow: <sub_flow_name>
            options:
                <task>:
                    <option_name>: <value>
```

Replace all objects with <> with the desired values.

For example, let's examine the definition of the `ci_master` flow from the universal `cumulusci.yml` file.

```
ci_master:
    group: Continuous Integration
    description: Deploy the package metadata to the packaging org and prepare for␣
↪managed package version upload.  Intended for use against main branch commits.
    steps:
        1:
            flow: dependencies
            options:
                update_dependencies:
                    resolution_strategy: production
        2:
            flow: deploy_packaging
        3:
            flow: config_packaging
```

This flow specifies that when the subflow `dependencies` runs, the `resolution_strategy` option has a value of `production` for the `update_dependencies` task (which itself executes in the `dependencies` subflow).

5.3.6 when Clauses

Specify a when clause in a flow step to conditionally run that step. A when clause is written in a Pythonic syntax that should evaluate to a boolean (True or False) result.

You can use the project_config object to reference values from the cumulusci.yml file to help with creation of the when clause's condition. You can use the double underscore (__) syntax to indicate values at subsequent levels of the file. For example, you can reference a project's namespace with project_config.project__package__namespace.

You can also reference values on the org_config object in when clauses. For example, it's common to reference org_config.scratch when building automation that needs to behave differently in a scratch org and a persistent org.

when clauses are frequently used in CumulusCI's standard library to conditionally run a step in a flow based on the source code format of the project. Below is the configuration for the standard library flow build_feature_test_package. The update_package_xml task will execute *only if* the project's source code format is not equal to "sfdx".

```
build_feature_test_package:
    group: Release Operations
    description: Create a 2gp managed package version
    steps:
        1:
            task: update_package_xml
            when: project_config.project__source_format != "sfdx"
        2:
            task: create_package_version
            options:
                package_type: Managed
                package_name: $project_config.project__package__name Managed Feature Test
                version_base: latest_github_release
                version_type: minor
                skip_validation: True
```

See *Use Variables for Task Options* for more information.

5.3.7 Tasks and Flows from a Different Project

It's also possible to use tasks and flows from another project with CumulusCI. The other project must be named under the sources section of the project cumulusci.yml file.

For example, when tasks or flows are referenced using the npsp namespace, CumulusCI fetches the source from the NPSP GitHub repository.

```
sources:
    npsp:
        github: https://github.com/SalesforceFoundation/NPSP
```

By default, CumulusCI uses the resolution strategy production, which will fetch the most recent production release, or the default branch if there are no releases. By specifying resolution_strategy, the behavior can be changed to match desired dependency resolution behavior, such as using beta releases or retrieving feature test packages from a commit status. See *dependency-resolution* for more details about resolution strategies.

Note: This feature requires that the referenced repository be readable (for example, it's public, or CumulusCI's GitHub

service is configured with the token of a user who has read access to it).

It's also possible to fetch a specific `tag` or `release`, where `release` is one of `latest`, `previous`, or `latest_beta`. For example:

```
sources:
    eda:
        github: https://github.com/SalesforceFoundation/EDA
        release: latest
    npsp:
        github: https://github.com/SalesforceFoundation/NPSP
        tag: rel/3.163
```

You can also select a specific `commit` or `branch`. We recommend that most projects, however, use a resolution strategy.

When the repo is listed under `sources`, it's possible to run a task from NPSP...

```
$ cci task run npsp:robot
```

Or a flow...

```
$ cci flow run npsp:install_prod
```

Or even create a new flow that uses a flow from NPSP:

```
flows:
    install_npsp:
        steps:
            1:
                flow: npsp:install_prod
            2:
                flow: dev_org
```

This flow uses NPSP's `install_prod` flow to install NPSP as a managed package, and then run this project's own `dev_org` flow.

If the flow uses tasks that are implemented in custom Python code (see *Add a Custom Task*) then you must instruct CumulusCI to allow that code to run:

```
sources:
    eda:
        github: https://github.com/SalesforceFoundation/EDA
        allow_remote_code: True
```

5.4 Scratch Org Configurations

This section defines the scratch org configurations that are available without explicitly running `cci org scratch` to create a new configuration. For more information on using scratch orgs with CumulusCI, see *Manage Scratch Orgs*.

5.4.1 Override Default Values

Note: These overrides pertain only to scratch orgs.

You can override these values for your org.

- `days` (integer): Number of days for the scratch org to persist.
- `namespaced` (boolean): Is the scratch org namespaced or not. Applies only to managed package projects.
- `config_file` (string): Path to the org definition file to use when building the scratch org.

```
orgs:
    scratch:
        <org_name>:
            <key>: <value>
```

Replace all objects with <> with the desired values.

For example, override the default number of days from 7 to 15 in the `dev` org.

```
orgs:
    dev:
        days: 15
```

5.5 Configuration Scopes

CumulusCI merges multiple YAML files that enable configuration at several distinct scopes. All of these files have the same name, `cumulusci.yml`, but live in different locations in the file system.

You can configure files at these scope levels: *Project*, *Local Project* and *Global*. Configurations have an order of override precedence (from highest to lowest):

1. Project
2. Local Project
3. Global

One override only cascades over another when two configurations set a value for the same element on a task or flow.

Take for example, task `T` which takes two options, `opt1` and `opt2`.

You can specify a default value for `opt1` in your project `cumulusci.yml` file and a default value for `opt2` in your global `cumulusci.yml` file, and you'll see the expected result: both values are available in the project. (The default of `opt1` is not exposed to other projects.)

If you change your project `cumulusci.yml` file to also specify a default value for `opt2`, this new default `opt2` value takes precedence over the default `opt2` value specified in your global `cumulusci.yml` file.

5.5.1 Project Configurations

macOS/Linux: `.../path/to/project/cumulusci.yml`

Windows: `...\path\to\project\cumulusci.yml`

This `cumulusci.yml` file lives in the project root directory and applies solely to this project. Changes here are committed back to a remote repository so other team members can benefit from the customizations. Configurations in this file apply solely to this project, and take precedence over any configurations specified in the global `cumulusci.yml` file, but are overridden by configurations in the local project `cumulusci.yml` file.

5.5.2 Local Project Configurations

macOS/Linux: `~/.cumulusci/project_name/cumulusci.yml`

Windows: `%homepath%\.cumulusci\project_name\cumulusci.yml`

Configurations in this `cumulusci.yml` file apply solely to the project with the given <project*name*>*, and take precedence over _all other* configuration scopes. If you want to make customizations to a project, but don't need them to be available to other team members, make those customizations here.

5.5.3 Global Configurations

macOS/Linux: `~/.cumulusci/cumulusci.yml`

Windows: `%homepath%\.cumulusci\cumulusci.yml`

Configuration of *all* CumulusCI projects on your machine. Configurations in this file have a low precedence, and are overridden by *all other* configurations except for those that are in the universal `cumulusci.yml` file.

5.5.4 Universal Configurations

There is one more configuration file that exists: the universal cumulusci.yml file that ships with CumulusCI itself. This file actually holds the lowest precedence of all, as all other scopes override this file's contents. That said, it contains all of the definitions for the tasks, flows, and org configurations that come standard with CumulusCI.

The commands `cci task info` and `cci flow info` display all of the information about a task's or flow's configuration. They display the information in the standard library alongside any customizations defined in your cumulusci.yml file.

5.6 Advanced Configurations

5.6.1 Customizing Metadata Deployment

CumulusCI's `deploy` task offers deep flexibility to customize your deployment process. Review *deploy* for an in-depth guide.

Configure Metadata Deployment

CumulusCI's `deploy` task uses the Metadata API to deploy metadata from the repository to a Salesforce org. `deploy` offers multiple sophisticated capabilities to suit the needs of your project.

Specifying the Deploying Package and Test Run Levels

Use the `path` option to specify the path to the metadata you wish to deploy. The metadata may be in either Metadata API or Salesforce DX source format. SFDX-format source will automatically be converted to Metadata API for deployment. This conversion takes place prior to application of transforms (below) and requires that the path be listed as a package directory in `sfdx-project.json`.

Set the test run level with the `test_level` option. Available values are `NoTestRun`, `RunLocalTests`, `RunAllTestsInOrg`, and `RunSpecifiedTests`. If you use `RunSpecifiedTests`, you must also supply a list of tests with the `specified_tests` option. This option accept a comma-separated value at the command line or a list in your `cumulusci.yml` markup.

Source Transforms

`deploy` allows you to specify *transforms* that run against your metadata before it is delivered to the Salesforce platform. Some of these transforms are built-in, and others you can specify in your `cumulusci.yml` to suit your project's specific needs.

Specifying Transforms

Four of the `deploy` transformations are built-in, and controlled by task options (see below). Others, including the flexible `find_replace` transform, are specified by your customization in `cumulusci.yml`.

You can specify transforms to run everywhere `deploy` is invoked by using customization in your `tasks` section:

```
tasks:
    deploy:
        options:
            transforms:
                - transform: find_replace
                  options:
                      patterns:
                          - find: foo
                            replace: bar
```

You can also add a transform for just a specific flow step:

```
flows:
    my_flow:
        steps:
            1:
                task: deploy
                options:
                    transforms:
                        - transform: find_replace
                          options:
                              patterns:
```

(continues on next page)

(continued from previous page)

```
                                    - find: foo
                                      replace: bar
```

or override the options in an out-of-the-box flow:

```
flows:
    deploy_packaging:
        options:
            deploy:
                transforms:
                    - transform: find_replace
                      options:
                          patterns:
                              - find: foo
                                replace: bar
```

To learn more about customization, see *Configure CumulusCI*. To discover the available transform options, consult the section for each transform in *deploy*.

Namespace Injection

The `deploy` task, like many others in CumulusCI, supports *namespace injection*. This strategy allows you to flexibly inject your package's namespace into (or remove it from) your metadata to suit various deployment contexts. See *Namespace Injection* for more information about namespace injection.

In most cases, you don't need to customize `deploy` options for namespace injection, as the task will automatically infer the appropriate strategy. If you do find an edge case where inference does not suit your needs, you can configure namespace injection using the following options.

- `namespace_inject`: set to specify the namespace to inject against tokens.
- `namespace_strip`: set to specify the namespace prefix to remove.
- `unmanaged`: set to `True` to replace namespace tokens with the empty string.
- `namespaced_org`: set to `True` to replace the special `NAMESPACED_ORG` tokens with the namespace. Note that the packaging org is considered a namespaced-org context.

Meta-XML Cleaning

`deploy` can automatically remove references (`<packageVersions/>` elements) to the versions of other managed packages from `*-meta.xml` files in your metadata. This cleaning is controlled by the `clean_meta_xml` option, which defaults to `True`.

Static Resource Bundling

For products that are in Metadata API source format, `deploy`, can automatically Static Resource content that is stored uncompressed in an alternate directory into the ZIP files expected by the platform. To use this capability, set the option `static_resource_path` to the path where you store unzipped Static Resource content. CumulusCI will ZIP each subdirectory and place the archives in the `staticresources` directory in the deployment bundle. You must include required `-meta.xml` files in the `static_resource_path`.

Feature-Parameter Cleaning for Unlocked Packages

Unlocked Packages do not support Feature Parameter metadata. When CumulusCI is building an Unlocked Package, this transform will automatically remove Feature Parameter metadata from the deployment bundle.

Find-and-Replace Variable Injection

CumulusCI allows you to specify arbitrary injections of data against tokens you define in your metadata. This capability is often used, for example, to inject secure keys or tokens from environment variables.

Configure injection with the `find_replace` transform in two ways - `find` and `xpath`.

- Using the `find` parameter, it modifies the metadata for each occurrence of its value by utilizing the supplied `replace` parameter. For XML files, this adjustment exclusively affects the values within the XML elements, leaving the tags unchanged.
- Using the `xpath` parameter, it exclusively modifies the content of the XML files within the specific element indicated by the provided `xpath`, replacing it with the specified `replace` parameter.

Note: The Find-and-Replace Injection only supports one of the two parameters `find` or `xpath`.

Injection with `find` parameter:

```
task: deploy
options:
    transforms:
        - transform: find_replace
          options:
              patterns:
                  - find: foo
                    replace: bar
```

Injection with `xpath` parameter:

```
task: deploy
options:
    transforms:
        - transform: find_replace
          options:
              patterns:
                  - xpath: /path/to/element
                    replace: bar
```

The `xpath` also supports the use of predicates such as - `xpath: /path[1]/to/element[text()="some_text"]`

To use an environment variable as the source of the value to replace, use the `replace_env` key. Note that it's valid to use multiple runs of `find_replace`; they will be applied in sequence.

```
task: deploy
options:
    transforms:
        - transform: find_replace
          options:
              patterns:
                  - find: AUTH_TOKEN
                    replace_env: SECURE_ACCESS_KEY
                  - xpath: /path/to/AUTH_TOKEN
                    replace_env: SECURE_ACCESS_KEY
```

Find-and-Replace Id Injection

Some metadata components contain ID references. CumulusCI offers a way to insert the ID of an existing record from the target org directly into such components during a deployment. Specify a SOQL query with the `replace_record_id_query` option as follows:

Injection with `find` parameter:

```
task: deploy
options:
    transforms:
        - transform: find_replace
          options:
              patterns:
                  - find: special_string
                    replace_record_id_query: SELECT Id from Account WHERE name='Specific␣
↪Account'
                    api: rest
```

Injection with `xpath` parameter:

```
task: deploy
options:
    transforms:
        - transform: find_replace
          options:
              patterns:
                  - xpath: /path/to/element
                    replace_record_id_query: SELECT Id from Account WHERE name='Specific␣
↪Account'
                    api: rest
```

Available values for `api` include `rest` and `tooling`.

Find-and-Replace Current Username Injection

CumulusCI can replace a given token with the username of the current running user in the target Salesforce org. All that is needed is to specify a value for `find` or `xpath` and set `inject_username: True`:

Injection with `find` parameter:

```
task: deploy
options:
    transforms:
        - transform: find_replace
          options:
              patterns:
                  - find: special_string
                    inject_username: True
```

Injection with `xpath` parameter:

```
task: deploy
options:
    transforms:
        - transform: find_replace
          options:
              patterns:
                  - xpath: /path/to/element
                    inject_username: True
```

Find-and-Replace Org URL Injection

CumulusCI can replace a given token with the org URL of the target Salesforce org. All that is needed is to specify a value for `find` or `xpath` and set `inject_org_url: True`:

Injection with `find` parameter:

```
task: deploy
options:
    transforms:
        - transform: find_replace
          options:
              patterns:
                  - find: special_string
                    inject_org_url: True
```

Injection with `xpath` parameter:

```
task: deploy
options:
    transforms:
        - transform: find_replace
          options:
              patterns:
                  - xpath: /path/to/element
                    inject_org_url: True
```

Stripping Components with a `package.xml` Manifest

This transformation allows you to deploy a subset of a metadata directory based on a `package.xml` manifest by removing unwanted components. It will compare components available in the source folder with a provided `package.xml` file and delete/modify component files which are not found.

```
task: deploy
options:
    transforms:
        - transform: strip_unwanted_components
          options:
              package_xml: PACKAGE_XML_FILE_PATH
```

5.6.2 Reference Task Return Values

Attention: Current task return values are *not* documented, so finding return values set by a specific task (if any) requires you to read the source code for the given task.

It is sometimes useful for return values to be used as input by a subsequent task in the context of a flow. Tasks can set arbitrary return values on themselves while executing. These values can then be referenced by subsequent tasks in a flow.

To reference a return value on a previous task use the following syntax:

```
^^prior_task.return_value
```

To discover what's available for `return_value`, find the source code for an individual task in the CumulusCI repository.

For example, let's examine how CumulusCI defines the standard `upload_beta` task in the universal `cumulusci.yml` file.

```
upload_beta:
    description: Uploads a beta release of the metadata currently in the packaging org
    class_path: cumulusci.tasks.salesforce.PackageUpload
    group: Release Operations
```

To see if anything is being set on `self.return_values`, find the file that defines the class `cumulusci.tasks.salesforce.PackageUpload`. A little digging yields that this class is defined in the file package_upload.py and has a method called `_set_return_values()`. This method sets `self.return_values` to a dictionary with these keys: `version_number`, `version_id`, and `package_id`.

Now look at the standard `release_unlocked_production` flow defined in the universal `cumulusci.yml` file:

```
release_unlocked_production:
    group: Release Operations
    description: Promote the latest beta 2GP unlocked package version and create a new␣
↪release in GitHub
    steps:
        1:
            task: promote_package_version
        2:
```

(continues on next page)

(continued from previous page)

```
            task: github_release
            options:
                version: ^^promote_package_version.version_number
                version_id: ^^promote_package_version.version_id
                dependencies: ^^promote_package_version.dependencies
                package_type: 2GP
                tag_prefix: $project_config.project__git__prefix_release
        3:
            task: github_release_notes
            ignore_failure: True
            options:
                publish: True
                tag: ^^github_release.tag_name
                version_id: ^^promote_package_version.version_id
```

This flow shows how subsequent tasks can reference the return values of a prior task. In this case, the `github_release` task uses the `version_numer` set by the `promote_package_version` task as an option value with the `^^promote_package_version.version_number` syntax. Here, `dependencies` is of type list and it uses the list from `promote_package_version` task as an option value with `^^promote_package_version.dependencies` syntax.

Similarly, the `github_release_notes` task uses the `version_id` set by the `promote_package_version` task as an option value with the `^^promote_package_version.version_number` syntax and uses the `tag` set by `github_release` task as an option value with the `^^github_release.tag_name` syntax.

The below `example_flow` shows how the task options of type list CANNOT be used. Here, `update_dependencies` task does not set the task option `dependencies` as the list of values from the prior tasks. Similarly, task options of type dictionary cannot be set as key value pairs from the prior tasks.

```
example_flow:
    description: You cannot make a list/dict with return values like below
    steps:
        1:
            task: get_latest_version_example
        2:
            task: get_old_version_example
        3:
            task: update_dependencies
            options:
                dependencies:
                    - latest_version_id: ^^get_latest_version_example.version_id
                    - version_id: ^^get_old_version_example.version_id
                packages_only: true
```

5.7 Troubleshoot Configurations

Use `cci task info <name>` and `cci flow info <name>` to see how a given task or flow behaves with current configurations.

For example, the `util_sleep` task has a `seconds` option with a default value of 5 seconds.

```
$ cci task info util_sleep
util_sleep

Description: Sleeps for N seconds

Class: cumulusci.tasks.util.Sleep

Command Syntax

    $ cci task run util_sleep

Options

    -o seconds SECONDS
    Required
    The number of seconds to sleep
    Default: 5
```

To change the default value to 30 seconds for all projects on your machine, add the desired value in your *global* `cumulusci.yml` file.

```
tasks:
    util_sleep:
        options:
            seconds: 30
```

Now `cci task info util_sleep` shows a default of 30 seconds.

```
$ cci task info util_sleep
util_sleep

Description: Sleeps for N seconds

Class: cumulusci.tasks.util.Sleep

Command Syntax

    $ cci task run util_sleep

Options

    -o seconds SECONDS
    Required
    The number of seconds to sleep
    Default: 30
```

Displaying the active configuration for a given task or flow can help with cross-correlating which configuration scope affects a specific scenario.

Tip: The `cci task info` and `cci flow info` commands show information about how a task or flow is *currently* configured. The information output by these commands change as you make further customizations to your project's `cumulusci.yml` file.

CHAPTER

SIX

MANAGE SCRATCH ORGS

Scratch orgs are temporary Salesforce orgs that can be quickly set up "from scratch," and which last for no more than 30 days. There are several reasons why scratch orgs are encouraged for development and testing over sandboxes or Developer Edition orgs. Scratch orgs:

- Provide a repeatable starting point without the challenge of managing persistent orgs' state over time.
- Are scalable and ensure that individual, customized environments are available to everyone in the development lifecycle.
- Facilitate a fully source-driven development process built around best practices.

CumulusCI offers tools for working with all types of Salesforce orgs, but provides the most value when working with scratch orgs. CumulusCI automation helps realize the promise of scratch orgs as low cost, repeatable, source-driven environments for every phase of the product lifecycle.

This section focuses on managing scratch orgs in a CumulusCI project. To learn about managing persistent orgs, such as sandboxes, production orgs, and packaging orgs, visit the *Connect Persistent Orgs* section.

6.1 What Is an Org in CumulusCI?

An org in CumulusCI's keychain starts out as a named configuration, tailored for a specific purpose within the lifecycle of the project (such as development, QA, beta testing, and so on). CumulusCI creates and uses scratch orgs based on these configurations on demand. In fact, a scratch org is only generated the first time you use the scratch org. When it's expired or been deleted, a new one can be created again with the same configuration.

CumulusCI offers tools that make it easy to discover predefined org configurations, create scratch orgs based on those configurations, and define new orgs and new configurations.

6.2 Set Up the Salesforce CLI

Scratch orgs in CumulusCI allow teams to be confident that the orgs they develop and test in are as close to their production environments as possible. We recommend working with scratch orgs created by Salesforce DX.

See the *Set Up SFDX* section for instructions.

6.3 Predefined Orgs

CumulusCI comes with predefined org configurations. Every project's keychain starts with these configurations ready and available to be turned into a live scratch org.

Org	Role	Definition File	Lifespan
`dev`	Development workflows	`orgs/dev.json`	7 days
`qa`	Testing workflows	`orgs/dev.json`	7 days
`feature`	Continuous integration	`orgs/dev.json`	1 day
`beta`	Continuous integrationHands-on testing	`orgs/beta.json`	1 day
`release`	Continuous integrationHands-on testingProduct demos	`orgs/release.json`	1 day

To see the predefined orgs in your project:

```
$ cci org list
```

If your project has customized org configurations, your listing can include more configurations than shown in the previous table, and your project's versions of the standard configurations can be different.

6.4 Create a Scratch Org

To create a scratch org from a configuration, use it as the target of a command, task, or flow. CumulusCI automatically initializes orgs when they're first used.

You can create a scratch org from the `dev` configuration and review information about the created org with:

```
$ cci org info dev
```

When the org is created, it's associated with the name `dev` in the CumulusCI keychain and can be used with other commands until it expires. When an org expires or is removed, its associated configuration is left in place, and can be recreated whenever needed.

It's possible to create new orgs in the keychain that inherit their configuration from a built-in org.

Here we create a new org that uses the same configuration as the built-in org `dev` and has the alias `myDevOrg`:

```
$ cci org scratch dev myDevOrg
```

Verify that there is now an org with the name of <org_name> that is associated with the `dev` configuration by running `cci org list`.

You can have as many named orgs as you wish, or none at all. Many CumulusCI users work only with built-in orgs.

6.4.1 Scratch Org Limits

Each scratch org you create is counted against limits in your Dev Hub. Scratch orgs count against an *active* scratch org limit, which controls how many orgs can exist at the same time, and a *daily* scratch org limit, which controls how many total orgs can be created per day.

Scratch org limits are based on your Dev Hub's edition and your Salesforce contract. To review limits and consumption, run the command:

```
$ sfdx force:limits:api:display -u <username>
```

<username> is your Dev Hub username. The limit names are `ActiveScratchOrgs` and `DailyScratchOrgs`.

6.5 List Orgs

When inside a project repository, run `cci org list` to see all the orgs you have configured or connected.

6.6 Set a Default Org

When you run a task or flow that performs work on an org, specify the org with the `--org` option.

```
$ cci flow run dev_org --org dev
```

To run many commands against the same org, set a default.

```
$ cci org default dev
$ cci flow run dev_org
```

Alternately, set a default org when creating a new named configuration by passing the `--default` flag.

```
$ cci org scratch dev <org_name> --default
```

To remove the existing default org:

```
$ cci org default dev --unset
```

6.7 Open Orgs in the Browser

Run `cci org browser` <org_name> to log into any org in the keychain in a new browser tab.

6.8 Delete Scratch Orgs

If an org defined in the keychain has created a scratch org, you can delete the scratch org but leave the configuration in the keychain to reuse it later.

```
$ cci org scratch_delete <org_name>
```

Using `scratch_delete` doesn't remove the org <org_name> from your org list. This default behavior lets you easily recreate scratch orgs from a stored, standardized configuration.

To permanently remove an org from the org list, and also delete the associated scratch org:

```
$ cci org remove <org_name>
```

It's not necessary to explicitly remove or delete expired orgs. CumulusCI recreates an expired org the first time you attempt to use it. To clean up expired orgs from the keychain:

```
$ cci org prune
```

6.9 Configure Predefined Orgs

Projects can customize the set of configurations available out of the box, and add further predefined orgs to meet project-specific needs.

An org configuration has a name, such as `dev` or `qa`, and is defined by options set in the `cumulusci.yml` file as well as in the contents of a specific `.json` scratch org definition file in the `orgs` directory. For orgs like `dev` and `qa` that are predefined for all projects, the configuration is located in the CumulusCI standard library, but can be customized by projects in the `cumulusci.yml` file.

When developing a managed package project, it is often useful to test inside of a namespaced scratch org. Many projects configure an org called `dev_namespaced`, a developer org that has a namespace. This org is defined under the `orgs__scratch` section in the `cumulusci.yml` file.

```
orgs:
    scratch:
        dev_namespaced:
            config_file: orgs/dev.json
            days: 7
            namespaced: True
```

This org uses the same scratch org definition file as the `dev` org, but has a different configuration in the `cumulusci.yml` file, resulting in a different org shape and a different use case. The key facets of the org shape that are defined in the `cumulusci.yml` file are whether or not the org has a namespace, and the length of the org's lifespan.

Org definition files stored in the `orgs` directory are configured as specified in the Salesforce DX Developer Guide.

Many projects never add a new org definition `.json` file, and instead add specific features and settings to the files shipped with CumulusCI. However, new definitions can be added and referenced under the `orgs__scratch` section of the `cumulusci.yml` file to establish org configurations that are completely customized for a project.

6.10 Import an Org from the Salesforce CLI

CumulusCI can import existing orgs from the Salesforce CLI keychain.

```
$ cci org import <sfdx_alias> <cci_alias>
```

For `sfdx_alias`, specify the alias or username of the org in the Salesforce CLI keychain. For `cci_alias`, provide the name to use in CumulusCI's keychain.

Important: CumulusCI cannot automatically refresh orgs imported from Salesforce CLI when they expire.

6.11 Use a Non-Default Dev Hub

By default, CumulusCI creates scratch orgs using the DevHub org configured as the `defaultdevhubusername` in `sfdx`. Switch to a different DevHub org within a project by configuring the `devhub` service.

```
$ cci service connect devhub mydevhub --project
Username: <DevHub username>
devhub is now configured for this project.
```

CHAPTER

SEVEN

CONNECT PERSISTENT ORGS

In addition to creating *scratch orgs* in CumulusCI, you can connect persistent orgs to your project to run tasks and flows on them. This feature supports use cases such as deploying to a Developer Edition org to release a package version, or installing to a sandbox for user acceptance testing.

Attention: A different setup is required to connect to orgs in the context of an automated build. See *continuous integration* for more information.

7.1 The `org connect` Command

To connect to a persistent org:

```
$ cci org connect <org_name>
```

This command automatically opens a browser window pointed to a Salesforce login page. The provided `<org_name>` is the alias that CumulusCI will assign to the persistent org.

Note: Connecting an org via `cci org connect` does *not* expose that org to the Salesforce CLI.

If your org has a custom domain, use the `--login-url` option along with the corresponding login url.

```
cci org connect <org_name> --login-url https://example.my.domain.salesforce.com
```

7.1.1 Production and Developer Edition Orgs

No options are needed for these org types. Just run the same command you normally would to connect to a persistent org.

```
$ cci org connect <org_name>
```

7.1.2 Sandboxes

For sandboxes, pass the `--sandbox` flag along with the org name.

```
$ cci org connect <org_name> --sandbox
```

Note: The `--sandbox` flag can also be used for connecting a scratch org created externally to CumulusCI.

7.2 Verify Your Connected Orgs

Run `cci org list` to see your org listed under the "Connected Org" table. This example output shows a single persistent org connected to CumulusCI with the name `devhub`.

```
$ cci org list

Scratch Orgs
| Name        | Default | Days | Expired | Config      | Domain |
|-------------|---------|------|---------|-------------|--------|
| dev         |         | 7    | X       | dev         |        |
| feature     |         | 1    | X       | feature     |        |
| prerelease  |         | 1    | X       | prerelease  |        |
| qa          |         | 7    | X       | qa          |        |
| release     |         | 1    | X       | release     |        |

Connected Orgs
| Name   | Default | Username                  | Expires    |
|--------|---------|---------------------------|------------|
| devhub |         | j.holt@mydomain.devhub    | Persistent |
```

Verify a successful connection to the org by logging in.

```
$ cci org browser <org_name>
```

7.3 Global Orgs

By default, `cci org connect` stores the OAuth credentials for connected orgs in a *project-specific* keychain. Using a project-specific keychain means that an org connected in Project A's directory isn't available when you're working in Project B's directory.

Connect an org and make it available to *all* CumulusCI projects on your computer by passing the `--global-org` flag.

```
$ cci org connect <org_name> --global-org
```

7.4 Use a Custom Connected App

CumulusCI uses a preconfigured Connected App to authenticate to Salesforce orgs that use OAuth2. In most cases this preconfigured app is all you need to authenticate into orgs. To control the Connected App for specific security or compliance requirements (such as adding a private key to sign a certificate connected with the configuration, or enforcing restrictions on user activity), create your own Connected App and configure CumulusCI to use it when connecting to orgs.

To create a custom Connected App, run the `connected_app` task, and then manually edit its configuration to suit your requirements.

Important: Make sure to create the Connected App in a production org!

This command will create a Connected App in the Dev Hub org connected to `sfdx`, and connect it to CumulusCI as a `connected_app` service named `custom`.

```
$ cci task run connected_app --label custom --connect true
```

After running this task, `cci service list` should show that the `custom` connected_app service exists.

```
$ cci service list
                                                Services

Default   Type                Name          Description
-------------------------------------------------------------------------------
✓          connected_app      built-in      A Connected App is required to connect to and␣
↪run commands
                                            against persistent orgs. See https://cumulusci.
↪readthedocs.io/
                                            en/latest/connected-orgs.html#use-a-custom-
↪connected-app for
                                            more info.
           connected_app      custom        A Connected App is required to connect to and␣
↪run commands
                                            against persistent orgs. See https://cumulusci.
↪readthedocs.io/
                                            en/latest/connected-orgs.html#use-a-custom-
↪connected-app for
                                            more info.
```

Use the `--connected_app` option to connect an org using the custom connected app:

```
$ cci org connect <org_name> --connected_app custom
```

To edit the Connected App's OAuth scopes:

1. In Lightning Experience, go to Setup –> Apps –> Apps Manager.
2. Click the arrow on the far right side of the row that pertains to the newly created Connected App.
3. Click "Edit."
4. Add or remove OAuth scopes as desired. CumulusCI requires the `api`, `full`, and `refresh_token` scopes.

For a full list of options, run the *connected_app* task reference documentation.

CHAPTER

EIGHT

DEVELOP A PROJECT

A general overview on how to develop a Salesforce project with CumulusCI.

8.1 Set Up a Dev Org

The `dev_org` flow creates an org to develop on by moving all metadata (managed and unmanaged) into the org, and configuring it to be ready for development.

Tip: Run `cci flow info dev_org` for a full list of the `dev_org` flow steps.

To run the `dev_org` flow against the project's *default org*:

```
$ cci flow run dev_org
```

To run the `dev_org` flow against a specific org, use the `--org` option. The following runs the `dev_org` flow against the org named `dev`.

```
$ cci flow run dev_org --org dev
```

Open the new `dev` org to begin development.

```
$ cci org browser dev
```

8.2 List Changes

To see what components have changed in a target org use the *list_changes* task:

```
$ cci task run list_changes --org dev
```

Wizard Note

This functionality relies on Salesforce's source tracking feature, which is currently available only in Scratch Orgs, Developer Sandboxes, and Developer Pro Sandboxes.

For more information, see *List and Retrieve Options*.

8.3 Retrieve Changes

The *retrieve_changes* task supports both Salesforce DX and Metadata API-format source code. It utilizes the SourceMember `sObject` to detect what has changed in an org, and also gives you discretion regarding which components are retrieved when compared to the *dx_pull* task.

To retrieve *all* changes in an org:

```
$ cci task run retrieve_changes --org dev
```

For information on retrieving specific subsets of changes, see *List and Retrieve Options*.

8.3.1 `--path`

Manual tracking of component versions offers the possibility of retrieving one set of changes into directory A, and retrieving a different set of changes into directory B. By default, changes are retrieved into the `src` directory when using Metadata API source format, or the default package directory (`force-app`) when using Salesforce DX source format.

To retrieve metadata into a *different* location use the `--path` option:

```
$ cci task run retrieve_changes --org dev --path your/unique/path
```

8.4 List and Retrieve Options

When developing in an org, the changes you're most interested in are sometimes mixed with other changes that aren't relevant to what you're doing.

For example, changing metadata like Custom Objects and Custom Fields often results in changes to Page Layouts and Profiles that you don't wish to review or retrieve.

It's a common workflow in CumulusCI to use the `list_changes` task, combined with the options featured in this subsection, to narrow the scope of changes in the org to the exact elements you desire to retrieve in your project. When the correct set of metadata is listed, run the `retrieve_changes` task to bring those changes into the repository.

8.4.1 `--include` & `--exclude`

When retrieving metadata from an org, CumulusCI represents each component name as the combination of its type (such as a `Profile`, `CustomObject`, or `ApexClass`) and its API name: `MemberType: MemberName`. An `ApexClass` named `MyTestClass` would be represented as `ApexClass: MyTestClass`.

The `--include` and `--exclude` options lets you pass multiple regular expressions to match against the names of changed components. This metadata is either included or excluded depending on which option the regular expression is passed. Multiple regular expressions can be passed in a comma-separated list.

The following lists all modified metadata that ends in "Test" and "Data" in the default org.

```
$ cci task run list_changes --include "Test$,Data$"
```

Since the metadata string that CumulusCI processes also includes the `MemberType`, use exclusions and inclusions that filter whole types of metadata.

The following will list all changes *except for* those with a type of `Profile`.

```
$ cci task run list_changes --exclude "^Profile: "
```

8.4.2 --types

To list or retrieve changed metadata of the same type, use the `--types` option along with the metadata type to retrieve.

The following retrieves all changed `ApexClass` and `ApexComponent` entities in the default org.

```
$ cci task run retrieve_changes --types ApexClass,ApexComponent
```

8.5 Push Changes

Developers often use an editor or IDE like Visual Studio Code to modify code and metadata stored in the repository. After making changes in an editor, push these changes from your project's local repository to the target org.

If your project uses the Salesforce DX source format, use the *dx_push* task.

```
$ cci task run dx_push
```

If your project uses the Metadata API source format, use the *deploy* task:

```
$ cci task run deploy
```

The `deploy` task has *many* options for handling a number of different scenarios. For a comprehensive list of options, see the *deploy* task reference.

8.6 Run Apex Tests

CumulusCI can execute Apex tests in an org with the `run_tests` task, and optionally report on test outcomes and code coverage. Failed tests can also be retried automatically.

```
$ cci task run run_tests --org <org_name>
```

The `run_tests` task has *many* options for running tests. For a comprehensive list of options and examples, see the *run_tests* task reference.

8.7 Set Up a QA Org

The `qa_org` flow sets up org environments where quality engineers test features quickly and easily. `qa_org` runs the specialized `config_qa` flow after deploying the project's unmanaged metadata to the org.

The following runs the `qa_org` flow against the `qa` org.

```
$ cci flow run qa_org --org qa
```

8.7.1 Create QA Configurations

Out of the box, and even in some active projects, the `config_dev` and `config_qa` flows are the same. Many teams have a requirement for additional configurations to be deployed when performing QA, but not when developing a new feature.

At Salesforce.org, our product teams often modify the `config_qa` flow to deploy configurations that pertain to large optional features in a package. These configurations are subsequently tested by the product's Robot Framework test suites.

To retrieve your own QA configurations, spin up a new org:

```
$ cci flow run qa_org
```

Make the necessary changes, and run:

```
$ cci task run retrieve_qa_config
```

This task defaults to retrieving this metadata under `unpackaged/config/qa`.

Tip: The configuration metadata can also be stored in a different location by using the `--path` option.

To delete the org…

```
$ cci org remove qa
```

Then re-create it…

```
$ cci flow run qa_org --org qa
```

Then run the `deploy_qa_config` to deploy the previously retrieved configurations to the org.

```
$ cci task run deploy_qa_config --org qa
```

To require that the `qa_org` flow always runs this task, add a `deploy_qa_config` task step under the `flows__config_qa` section of the `cumulusci.yml` file.

```
config_qa:
    steps:
        3:
            task: deploy_qa_config
```

Now `config_qa` (which is included in the `qa_org` flow) executes the `deploy_qa_config` task as the third step in the flow.

8.8 Manage Dependencies

CumulusCI is built to automate the complexities of dependency management for projects that extend, customize, or compose other projects. CumulusCI currently handles these main types of dependencies for projects.

- **GitHub Repository**: Dynamically resolve a product release, and its own dependencies, from a CumulusCI project on GitHub.
- **Packages**: Require a specific version of a managed package or unlocked package.
- **Unmanaged Metadata**: Require the deployment of unmanaged metadata.

Dependencies are listed in the `project__dependencies` section of `cumulusci.yml`

```
project:
    dependencies:
```

The `update_dependencies` task handles deploying dependencies to a target org, and is included in all flows designed to deploy or install to an org, such as `dev_org`, `qa_org`, `install_prod`, and others.

To run the `update_dependencies` task manually:

```
$ cci task run update_dependencies
```

8.8.1 GitHub Repository Dependencies

GitHub repository dependencies create a dynamic dependency between the current project and another CumulusCI project on GitHub. This is an example of listing Salesforce.org's EDA product as a dependency.

```
project:
    dependencies:
        - github: https://github.com/SalesforceFoundation/EDA
```

When `update_dependencies` runs, these steps are taken against the referenced repository.

- Look for the `cumulusci.yml` file and parse if found.
- Determine if the project has subfolders under `unpackaged/pre`. If found, deploy them first, in alphabetical order.
- Determine if the project specifies any dependencies in the `cumulusci.yml` file. If found, recursively resolve those dependencies and any dependencies belonging to them.
- Determine whether to install the project as as a managed package or unmanaged metadata:

  : - If the project has a namespace configured in the `cumulusci.yml` file, treat the project as a managed package unless the `unmanaged` option is set to `True` in the dependency. - If the project has a namespace and is *not* configured as unmanaged, use the GitHub API to locate the latest managed release of the project and install it.
- If the project is an unmanaged dependency, the main source directory is deployed as unmanaged metadata.
- Determine if the project has subfolders under `unpackaged/post`. If found, deploy them next, in alphabetical order. Namespace tokens are replaced with `<namespace>__` if the project is being installed as a managed package, or an empty string otherwise.

Reference Unmanaged Projects

If the referenced repository does not have a namespace configured, or if the dependency specifies the `unmanaged` option as `True`, the repository is treated as unmanaged.

Here is a project with Salesforce.org's EDA package listed as an unmanaged dependency:

```
project:
    dependencies:
        - github: https://github.com/SalesforceFoundation/EDA
          unmanaged: True
```

The EDA repository is configured for a namespace, but the dependency specifies `unmanaged: True`, so EDA deploys as unmanaged metadata.

CumulusCI only supports unmanaged repositories in Metadata API source format at present.

Reference a Specific Tag

To reference a specific version of the product other than the most recent commit on the main branch (for unmanaged projects) or the most recent production release (for managed packages), use the `tag` option to specify a tag from the target repository. This option is useful for testing against specific package versions, pinning a dependency to a version rather than using the latest release, and recreating org environments for debugging.

```
project:
    dependencies:
        - github: https://github.com/SalesforceFoundation/EDA
          tag: rel/1.105
```

The EDA repository's tag `rel/1.105` is used instead of the latest production release of EDA (1.111, for this example).

Skip `unpackaged/*` in Reference Repositories

If the referenced repository has unpackaged metadata under `unpackaged/pre` or `unpackaged/post`, use the `skip` option to skip deploying that metadata with the dependency.

```
project:
    dependencies:
        - github: https://github.com/SalesforceFoundation/EDA
          skip: unpackaged/post/course_connection_record_types
```

8.8.2 Package Dependencies

Managed package and unlocked package dependencies are rather simple. Under the `project__dependencies` section of the `cumulusci.yml` file, specify the namespace of the target package, and the required version number, or specify the package version id.

```
project:
    dependencies:
        - namespace: npe01
          version: 3.6
        - version_id: 04t000000000001
```

Package dependencies can include any package, whether or not it is built as a CumulusCI project. Dependencies on managed packages may be specified using the namespace and version or the version id. Dependencies on unlocked packages should use the version id.

8.8.3 Package Install Keys (Passwords)

Some packages are protected by an install key, which must be present in order to install the package. CumulusCI dependencies can use the `password_env_name` key to instruct CumulusCI to retrieve the package install key from an environment variable. This key is available on both package version dependencies and on GitHub dependencies:

```
project:
    dependencies:
        - namespace: my_namespace
          version: 3.6
          password_env_name: INSTALL_KEY
        - github: https://github.com/MyOrg/MyRepo
          password_env_name: MY_REPO_KEY
```

8.8.4 Unmanaged Metadata Dependencies

Specify unmanaged metadata to be deployed by specifying a `zip_url` or a `github` URL, and, optionally, `subfolder`, `namespace_inject`, `namespace_strip`, and `unmanaged` under the `project__dependencies` section of the cumulusci.yml file.

```
project:
    dependencies:
        - zip_url: https://SOME_HOST/metadata.zip
        - github: https://github.com/SalesforceFoundation/EDA
          subfolder: unpackaged/post/course_connection_record_types
          ref: 0cabfe
```

When the `update_dependencies` task runs, it downloads the zip file or GitHub subdirectory and deploys it via the Metadata API. The zip file must contain valid metadata for use with a deploy, including a `package.xml` file in the root.

Unmanaged metadata dependencies from GitHub may optionally specify the `ref` to download. If they do not, unmanaged GitHub dependencies are resolved like other GitHub references. See *Controlling GitHub Dependency Resolution* for more details on resolution of dynamic dependencies.

Note: In versions of CumulusCI prior to 3.33.0, unmanaged GitHub dependencies always deployed the most recent commit on the default branch.

Specify a Subfolder

Use the `subfolder` option to specify a subfolder of the zip file or GitHub repository to use for the deployment.

Tip: This option is handy when referring to metadata stored in a GitHub repository.

When `update_dependencies` runs, it still downloads the zip from `zip_url`, but then builds a new zip containing only the content of `subfolder`, starting inside `subfolder` as the zip's root.

Inject Namespace Prefixes

CumulusCI has support for tokenizing references to a package's namespace prefix in code. When tokenized, all occurrences of the namespace prefix, are replaced with `%%%NAMESPACE%%%` inside of files and `___NAMESPACE___` in file names. The `namespace_inject` option instructs CumulusCI to replace these tokens with the specified namespace before deploying the unpackaged dependency.

For more on this topic see *Namespace Injection*.

8.8.5 Pinning GitHub Dependencies

By default, CumulusCI resolves dynamic GitHub dependencies to the latest available releases. In some cases, this may be undesirable. You can use dependency pinning to control how dependencies are resolved, including transitive dependencies referenced by your own direct dependencies.

Use the `project__dependency_pins` section of your `cumulusci.yml` to establish pins. Each pin includes the keys `github`, which must match the URL of the repo you wish to pin, and a `tag` to which you wish to pin the dependency. Here's an example that pins NPSP and its transitive dependencies to specific versions:

```
project:
    dependencies:
        - github: https://github.com/SalesforceFoundation/NPSP
    dependency_pins:
        - github: https://github.com/SalesforceFoundation/NPSP
          tag: rel/3.219
        - github: https://github.com/SalesforceFoundation/Contacts_and_Organizations
          tag: rel/3.19
        - github: https://github.com/SalesforceFoundation/Households
          tag: rel/3.16
        - github: https://github.com/SalesforceFoundation/Recurring_Donations
          tag: rel/3.22
        - github: https://github.com/SalesforceFoundation/Relationships
          tag: rel/3.12
        - github: https://github.com/SalesforceFoundation/Affiliations
          tag: rel/3.10
```

Pins affect resolution of managed package versions and any unmanaged dependencies included in the target repositories.

If CumulusCI encounters a conflict with an existing tag or other specifier while attempting to pin dependencies, like this:

```
project:
    dependencies:
```

(continues on next page)

(continued from previous page)

```
        - github: https://github.com/SalesforceFoundation/NPSP
          tag: rel/3.220
    dependency_pins:
        - github: https://github.com/SalesforceFoundation/NPSP
          tag: rel/3.219
```

it will stop and require you to resolve the conflict by removing either the pin or the dependency specification.

We recommend using pins only when referencing external products whose development process or release schedule you do not control, such as NPSP and EDA. In most cases, it's preferable for dependencies within a product suite to remain unpinned to support ongoing development.

8.8.6 Controlling GitHub Dependency Resolution

CumulusCI converts dynamic dependencies specified via GitHub repositories into specific package versions and commit references by applying one or more *resolvers*. You can customize the resolvers that CumulusCI applies to control when it will use beta managed packages or second-generation feature test packages, or to intervene more deeply in the dependency resolution process.

CumulusCI organizes resolvers into *resolution strategies*, which are named, ordered lists of resolvers to apply. When CumulusCI applies a resolution strategy to a dependency, it applies each resolver from top to bottom until a resolver succeeds in resolving the dependency.

Four resolution strategies are provided in the CumulusCI standard library:

- `latest_release`, which will attempt to resolve to the latest managed release of a managed package project.
- `include_beta`, which will attempt to resolve to the latest beta, if any, or managed release of a managed package project.
- `commit_status`, which will resolve to second-generation package betas created on feature branches, if any, or the main branch, before falling back to managed package releases. This strategy is used only in the `qa_org_2gp` and `ci_feature_2gp` flows.
- `unlocked`, which will resolve to unlocked package betas created on feature branches, if any, or the main branch. This strategy does *not* fall back to managed package releases, and is used in the `qa_org_unlocked` flow. The complete list of steps taken by each resolution strategy is given below.

Each flow that resolves dependencies selects a resolution strategy that meets its needs. Two aliases, `production`, and `preproduction`, are defined for this purpose, because in many cases a development flow like `dev_org` or `install_beta` will want to utilize a *different* resolution strategy than a production flow like `ci_master` or `install_prod`.

By default, both `production` and `preproduction` use the `latest_release` resolution strategy. To opt to have development flows use beta versions of managed package dependencies, you can switch the `preproduction` alias to point to the `include_beta` resolution strategy:

```
project:
    dependency_resolutions:
        preproduction: include_beta
        production: latest_release
```

After this change, flows like `dev_org` will install beta releases of dependencies, if present.

Resolution Strategy Details

The standard resolution strategies execute the following steps to resolve a dependency:

latest_release:

This resolution strategy is suitable for any build for products that wish to consume production releases of their dependencies during development and testing. It is also suitable for production flows (such as `install_prod` or a MetaDeploy installer flow) for all products.

- If a `tag` is present, use the commit for that tag, and any package version found there. (Resolver: `tag`)
- Identify the most recent production package release via the GitHub Releases section. If located, use that package and commit. (Resolver: `latest_release`)
- Use the most recent commit on the repository's main branch as an unmanaged dependency. (Resolver: `unmanaged`)

include_beta:

This resolution strategy is suitable for any pre-production build for products that wish to consume beta releases of their dependencies during development and testing.

- If a `tag` is present, use the commit for that tag, and any package version found there. (Resolver: `tag`)
- Identify the most recent beta package release via the GitHub Releases section. If located, use that package and commit. (Resolver: `latest_beta`)
- Identify the most recent production package release via the GitHub Releases section. If located, use that package and commit. (Resolver: `latest_release`)
- Use the most recent commit on the repository's main branch as an unmanaged dependency. (Resolver: `unmanaged`)

commit_status:

This resolution strategy is suitable for feature builds on products that utilize a release branch model and build second-generation package betas (using the `build_feature_test_package` flow) on each commit.

- If a `tag` is present, use the commit for that tag, and any package version found there. (Resolver: `tag`)
- If the current branch is a release branch (`feature/NNN`, where `feature/` is the feature branch prefix and `NNN` is any integer) or a child branch of a release branch, locate a branch with the same name in the dependency repository. If a commit status contains a beta package Id for any of the first five commits on that branch, use that commit and package. (Resolver: `commit_status_exact_branch`)
- If the current branch is a release branch (`feature/NNN`, where `feature/` is the feature branch prefix and `NNN` is any integer) or a child branch of a release branch, locate a matching release branch (`feature/NNN`) in the dependency repository. If a commit status contains a beta package Id for any of the first five commits on that branch, use that commit and package. (Resolver: `commit_status_release_branch`)
- If the current branch is a release branch (`feature/NNN`, where `feature/` is the feature branch prefix and `NNN` is any integer) or a child branch of a release branch, locate a branch for either of the two previous releases (e.g., `feature/230` in this repository would search `feature/229` and `feature/228`) in the dependency repository. If a commit status contains a beta package Id for any of the first five commits on that branch, use that commit and package. (Resolver: `commit_status_previous_release_branch`)
- If a commit status contains a beta package Id for any of the first five commits on the default branch, use that commit and package. (Resolver: `commit_status_default_branch`)
- Identify the most recent beta package release via the GitHub Releases section. If located, use that package and commit. (Resolver: `latest_beta`)

- Identify the most recent production package release via the GitHub Releases section. If located, use that package and commit. (Resolver: `latest_release`)
- Use the most recent commit on the repository's main branch as an unmanaged dependency. (Resolver: `unmanaged`)

unlocked:

This resolution strategy is suitable for feature builds on products that utilize a release branch model and build unlocked package betas (using the `build_unlocked_test_package` flow) on each commit. It is also suitable for use cases where a persistent org and Unlocked Package versions are used for ongoing QA.

- If the current branch is a release branch (`feature/NNN`, where `feature/` is the feature branch prefix and `NNN` is any integer) or a child branch of a release branch, locate a branch with the same name in the dependency repository. If a commit status contains a beta package Id for any of the first five commits on that branch, use that commit and package. (Resolver: `unlocked_exact_branch`)
- If the current branch is a release branch (`feature/NNN`, where `feature/` is the feature branch prefix and `NNN` is any integer) or a child branch of a release branch, locate a matching release branch (`feature/NNN`) in the dependency repository. If a commit status contains a beta package Id for any of the first five commits on that branch, use that commit and package. (Resolver: `unlocked_release_branch`)
- If the current branch is a release branch (`feature/NNN`, where `feature/` is the feature branch prefix and `NNN` is any integer) or a child branch of a release branch, locate a branch for either of the two previous releases (e.g., `feature/230` in this repository would search `feature/229` and `feature/228`) in the dependency repository. If a commit status contains a beta package Id for any of the first five commits on that branch, use that commit and package. (Resolver: `unlocked_previous_release_branch`)
- If a commit status contains a beta package Id for any of the first five commits on the default branch, use that commit and package. (Resolver: `unlocked_default_branch`)

Customizing Resolution Strategies

Projects that require deep control of how dependencies are resolved can create custom resolution strategies.

To add a resolution strategy, add a list of the resolvers desired to the section `project__dependency_resolutions__resolution_strategies` in `cumulusci.yml`. For example:

```
dependency_resolutions:
    production: releases_only
    resolution_strategies:
        releases_only:
            - latest_release
```

would create a new resolution strategy called `releases_only` that *only* can resolve to a production release. (Dependencies without a production release would cause a failure). It also assigns the alias `production` to point to `releases_only`, meaning that standard flows like `install_prod` would use this resolution strategy.

Customizing resolution strategies is an advanced topic. The out-of-the-box resolution strategies provided with CumulusCI will cover the needs of most projects. However, this capability is available for projects that need it.

8.8.7 Automatic Cleaning of `meta.xml` Files on Deploy

To let CumulusCI fully manage the project's dependencies, the `deploy` task (and other tasks based on `cumulusci.tasks.salesforce.Deploy`, or subclasses of it) automatically removes the `<packageVersion>` element and its children from all `meta.xml` files in the deployed metadata. Removing these elements does not affect the files on the filesystem.

This feature supports CumulusCI's automatic dependency resolution by avoiding a need for projects to manually update XML files to reflect current dependency package versions.

Note: If the metadata being deployed references namespaced metadata that does not exist in the currently installed package, the deployment throws an error as expected.

Note: The automatic cleaning of `meta.xml` files can be disabled by setting the `clean_meta_xml` option to `False`.

Developers can also use the `meta_xml_dependencies` task to update the `meta.xml` files locally using the versions from CumulusCI's calculated project dependencies.

8.9 Use Tasks and Flows from a Different Project

Dependency handling is used in a very specific context: to install dependency packages or metadata bundles in a `dependencies` flow that is a component of some other flow.

CumulusCI also makes it possible to use automation (tasks and flows) from another CumulusCI project. This feature supports many use cases, including:

- Applying configuration from a dependency project, rather than just installing the package.
- Running Robot Framework tests that are defined in a dependency.

For more information, see *Tasks and Flows from a Different Project*.

CHAPTER

NINE

AUTOMATE DATA OPERATIONS

CumulusCI offers a suite of tasks to help you to manage data as part of your project automation. Within your repository, you can define one or several *datasets*, collections of data you use for specific purposes. CumulusCI tasks support extracting defined datasets from scratch orgs or persistent orgs, storing those snapshots within the repository, and automating the load of datasets into orgs. Data operations are executed via the Bulk and REST APIs.

CumulusCI has both high level tasks for working with Sample Datasets and low-level tasks for generic Extract, Transform and Load of any data.

9.1 Sample Data

Note: Sample Data features are still under active development and may change based on user feedback.

CumulusCI has easy to use tasks for working with the primary sample datasets used for projects. There is a 'default' dataset which would be used in scratch org configuration flows, as well as other datasets specific to the needs of specific scratch org configurations.

For example, the 'dev' dataset is for 'dev' orgs, and it is used instead of the default dataset if it exists.

You can create a dataset by extracting data from an existing org or by authoring a *Snowfakery recipe*. Extracting from an existing org is easy for use-cases where the data already exists or can be readily created in an org. Snowfakery is better for cases where either a) you would like to dynamically generate data, or b) you would rather edit static data in a text editor.

A Snowfakery dataset can consist of a single file with a name like `datasets/<datasetname>/<datasetname>.recipe.yml` . For example, `datasets/default/default.recipe.yml` or `datasets/qa/qa.recipe.yml`. The rest of what you need to know about Snowfakery is in the section *Generate Fake Data*.

9.1.1 Extracting and Loading Sample Datasets

In the simplest case, you can extract all data from an org using the task `capture_sample_data` like this:

```
$ cci task run capture_sample_data --org orgname
```

That will extract the data from the Salesforce org named `orgname` into the dataset named `default`.

You can then load it into any target org (e.g. `org2`) like this:

```
$ cci task run load_sample_data --org org2
```

A main benefit of sample datasets is that they are always loaded automatically into scratch orgs by the scratch org setup flows like `dev_org` and `qa_org`.

The exact subset of data captured depends on heuristics that may change over time, so do not depend on this task in a highly automated situation. It is designed to be used interactively, and you can control its behavior with an Extract Declaration file.

The extract process generates a mapping YAML file which will be used for subsequent loads. It has the name `datasets/<datasetname>/<datasetname>.mapping.yml`. It is possible to edit this file, but this may not be the best choice. Changes to the file can be overwritten when you capture data a second time. Rather than editing the file, it is preferable to create a Loading Rules file and then re-create the mapping file by capturing sample data again.

A Loading Rules file is a file named `datasets/<datasetname>/<datasetname>.load.yml` which can specify instructions like which API to use and in which order to load objects. This file is in the Loading Rules format. If you create such a file and then re-capture sample data, the mapping file will be updated to match.

9.1.2 Multiple Sample Datasets

If you want different datasets for different scratch org types (e.g. QA orgs verus Dev orgs) then you can change the data loaded by those types by making datasets specific to each one. This data will load instead of the default dataset.

```
$ cci task run capture_sample_data --dataset dev --org org1
```

```
$ cci task run capture_sample_data --dataset qa --org org2
```

This would create two datasets in `datasets/dev` and `datasets/qa` which would be loaded instead of `datasets/default`. You can create as many datasets as you want.

You can download just a subset of the objects or fields in an org with an Extract Declaration file.

9.2 Low level datasets

A dataset consists of:

- a *definition file*, written in YAML, which specifies the sObjects and fields contained in the dataset and the order in which they are loaded or extracted from an org.
- a *storage location*, which may take the form of a SQL database (typically, a SQLite file stored within the repository, although external databases are supported) or a SQL script file.

Datasets are stored in the `datasets/` folder within a repository by default. Projects created with a recent version of CumulusCI ship with this directory in place.

If `load_dataset` is called without any path options, it will automatically use a dataset that matches the org shape, if one exists. For example, a `dev` org will automatically use a dataset that exists at `datasets/dev/`. Within that folder, two files must exist, also matching the org shape name: `dev.mapping.yml` and `dev.dataset.sql`, in this example. If the directory or files do not exist and no paths options were specified, the task will look for `datasets/mapping.yml` and `datasets/dataset.sql` by default. When the `default_dataset_only` option is `True`, this overrides any path options and default files and looks *only* for a dataset directory that matches the org shape name. The `default_dataset_only` option defaults to `False`.

In addition, `load_dataset` is included in `config_dev`, `config_qa`, and `config_managed`, so it is automatically called when running most org setup flows. In this context, it runs with `default_dataset_only` set to `True`, to avoid double loading for backwards compatibility with customer flows that are already customized to call `load_dataset`.

9.3 The Lifecycle of a Dataset

A dataset starts with a definition: which objects, and which fields, are to be captured, persisted, and loaded into orgs? (The details of definition file format are covered below).

With a definition available, the dataset may be captured from an org into the repository. A captured dataset may be stored under version control and incorporated into project automation, loaded as part of flows during org builds or at need. As the project's needs evolve, datasets may be re-captured from orgs and versioned alongside the project metadata.

Projects may define one or many datasets. Datasets can contain an arbitrary amount of data.

9.4 Defining Datasets

A dataset is defined in YAML as a series of steps. Each step registers a specific sObject as part of the dataset, and defines the relevant fields on that sObject as well as its relationships to other sObjects that are included in the data set.

Note: This section discusses how to define a dataset and the format of the definition file. In many cases, it's easier to use the `generate_dataset_mapping` task than to create this definition by hand. See below for more details.

A simple dataset definition looks like this:

```
Accounts:
    sf_object: Account
    fields:
        - Name
        - Description
        - RecordTypeId
    lookups:
        ParentId:
            table: Account
            after: Accounts
Contacts:
    sf_object: Contact
    fields:
        - FirstName
        - LastName
        - Email
    lookups:
        AccountId:
            table: Account
```

This example defines two steps: `Accounts` and `Contacts`. (The names of steps are arbitrary). Each step governs the extraction or load of records in the sObject denoted in its `sf_object` property.

Relationships are defined in the `lookups` section. Each key within `lookups` is the API name of the relationship field. Beneath, the `table` key defines the stored table to which this relationship refers.

CumulusCI loads steps in order. However, sObjects earlier in the sequence of steps may include lookups to sObjects loaded later, or to themselves. For these cases, the `after` key may be included in a lookup definition, with a value set to the name of the step after which the referenced record is expected to be available. CumulusCI will defer populating the lookup field until the referenced step has been completed. In the example above, an `after` definition is used to support the `ParentId` self-lookup on `Account`.

9.4.1 API Selection

By default, CumulusCI will determine the data volume of the specified object and select an API for you: for under 2,000 records, the REST Collections API is used; for more, the Bulk API is used. The Bulk API is also used for delete operations where the hard delete operation is requested, as this is available only in the Bulk API. Smart API selection helps increase speed for low- and moderate-volume data loads.

To prefer a specific API, set the `api` key within any mapping step; allowed values are `"rest"`, `"bulk"`, and `"smart"`, the default.

CumulusCI defaults to using the Bulk API in Parallel mode. If required to avoid row locks, specify the key `bulk_mode: Serial` in each step requiring the use of serial mode.

For all API modes, you can specify a batch size using the `batch_size` key. Allowed values are between 1 and 200 for the REST API and 1 and 10,000 for the Bulk API.

Note that the semantics of batch sizes differ somewhat between the REST API and the Bulk API. In the REST API, the batch size is the size of upload batches and also the actual size of individual transactions. In the Bulk API, the batch size is the maximum record count in a Bulk API upload batch, which is subject to its own limits, including restrictions on total processing time. Bulk API batches are automatically chunked further into transactions by the platform, and the transaction size cannot be controlled.

9.4.2 Upserts

The definition of "upsert" is an operation which creates new records and updates existing records depending on a field (the update key) which determines whether the input row and the existing row are "the same".

You can do ID-based, idLookup-based and external ID-based upserts and updates by specifying additional settings in a mapping step.

```
Insert Accounts:
    sf_object: Account
    action: upsert
    update_key: Extid__c
    fields:
        - Name
        - Extid__c
```

Whenever `update_key` is supplied, the action must be `upsert` and vice versa.

9.4.3 Database Mapping

CumulusCI's definition format includes considerable flexibility for use cases where datasets are stored in SQL databases whose structure is not identical to the Salesforce database. Salesforce objects may be assigned to arbitrary database tables, and Salesforce field names mapped to arbitrary columns.

For new mappings, it's recommended to allow CumulusCI to use sensible defaults by specifying only the Salesforce entities. Legacy datasets are likely to include explicit database mappings, which would look like this for the same data model as above:

```
Accounts:
    sf_object: Account
    table: Account
    fields:
        Name: Name
```

(continues on next page)

(continued from previous page)

```
        Description: Description
        RecordTypeId: RecordTypeId
    lookups:
        ParentId:
            table: Account
            after: Accounts
Contacts:
    sf_object: Contact
    table: Contact
    fields:
        FirstName: FirstName
        LastName: LastName
        Email: Email
    lookups:
        AccountId:
            table: Account
```

Note that in this version, fields are specified as a colon-separated mapping, not a list. Each pair in the field map is structured as `Salesforce API Name: Database Column Name`. Additionally, each object has a `table` key to specify the underlying database table.

New mappings that do not connect to an external SQL database (that is, mappings which simply extract and load data between Salesforce orgs) should not need to use this feature, and new mappings that are generated by CumulusCI use the simpler version shown above. Existing mappings may be converted to this streamlined style in most cases by loading the existing dataset, modifying the mapping file, and then extracting a fresh copy of the data. Note however that datasets which make use of older and deprecated CumulusCI features, such as the `record_type` key, may need to continue using explicit database mapping.

9.4.4 Record Types

CumulusCI supports automatic mapping of Record Types between orgs, keyed upon the Developer Name. To take advantage of this support, simply include the `RecordTypeId` field in any step. CumulusCI will transparently extract Record Type information during dataset capture and map Record Types by Developer Name into target orgs during loads.

Older dataset definitions may also use a `record_type` key:

```
Accounts:
    sf_object: Account
    fields:
        - Name
    record_type: Organization
```

This feature limits extraction to records possessing that specific Record Type, and assigns the same Record Type upon load.

It's recommended that new datasets use Record Type mapping by including the `RecordTypeId` field. Using `record_type` will result in CumulusCI issuing a warning.

9.4.5 Relative Dates

CumulusCI supports maintaining *relative dates*, helping to keep the dataset relevant by ensuring that date and date-time fields are updated when loaded.

Relative dates are enabled by defining an *anchor date*, which is specified in each mapping step with the `anchor_date` key, whose value is a date in the format `2020-07-01`.

When you specify a relative date, CumulusCI modifies all date and date-time fields on the object such that when loaded, they have the same relationship to today as they did to the anchor date. Hence, given a stored date of 2020-07-10 and an anchor date of 2020-07-01, if you perform a load on 2020-09-10, the date field will be rendered as 2020-09-19 -nine days ahead of today's date, as it was nine days ahead of the anchor date.

Relative dates are also adjusted upon extract so that they remain stable. Extracting the same data mentioned above would result in CumulusCI adjusting the date back to 2020-07-10 for storage, keeping it relative to the anchor date.

Relative dating is applied to all date and date-time fields on any mapping step that contains the `anchor_date` clause. If orgs are configured to permit setting audit fields upon record creation and the appropriate user permission is enabled, CumulusCI can apply relative dating to audit fields, such as `CreatedDate`. For more about how to automate that setup, review the `create_bulk_data_permission_set` task below.

For example, this mapping step:

```
Contacts:
    sf_object: Contact
    fields:
        - FirstName
        - LastName
        - Birthdate
    anchor_date: 1990-07-01
```

would adjust the `Birthdate` field on both load and extract around the anchor date of July 1, 1990. Note that date and datetime fields not mapped, as well as fields on other steps, are unaffected.

9.4.6 Person Accounts

CumulusCI supports extracting and loading person account data. In your dataset definition, map person account fields like `LastName`, `PersonBirthdate`, or `CustomContactField__pc` to **Account** steps (i.e. where `sf_object` equals **Account**).

```
Account:
    sf_object: Account
    table: Account
    fields:
        ## Business Account Fields
        - Name
        - AccountNumber
        - BillingStreet
        - BillingCity

        ## Person Account Fields
        - FirstName
        - LastName
        - PersonEmail
        - CustomContactField__pc
```

(continues on next page)

(continued from previous page)

```
        ## Optional (though recommended) Record Type
        - RecordTypeId
```

Record Types

It's recommended, though not required, to extract Account Record Types to support datasets with person accounts so there is consistency in the Account record types loaded. If Account `RecordTypeId` is not extracted, the default business account Record Type and default person account Record Type will be applied to business and person account records respectively.

Extract

During dataset extraction, if the org has person accounts enabled, the `IsPersonAccount` field is extracted for **Account** and **Contact** records so CumulusCI can properly load these records later. Additionally, `Account.Name` is not createable for person account **Account** records, so `Account.Name` is not extracted for person account **Account** records.

Load

Before loading, CumulusCI checks if the dataset contains any person account records (i.e. any **Account** or **Contact** records with `IsPersonAccount` as `true`). If the dataset does contain any person account records, CumulusCI validates the org has person accounts enabled.

You can enable person accounts for scratch orgs by including the PersonAccounts feature in your scratch org definition.

9.4.7 Advanced Features

CumulusCI supports two additional keys within each step

The `filters` key encompasses filters applied to the SQL data store when loading data. Use of `filters` can support use cases where only a subset of stored data should be loaded. :

```
filters:
    - "SQL string"
```

Note that `filters` uses SQL syntax, not SOQL. Filters do not perform filtration or data subsetting upon extraction; they only impact loading. This is an advanced feature.

The `static` key allows individual fields to be populated with a fixed, static value:

```
static:
    CustomCheckbox__c: True
    CustomDateField__c: 2019-01-01
```

The `soql_filter` key lets you specify a WHERE clause that should be used when extracting data from your Salesforce org:

```
Account:
    sf_object: Account
    table: Account
    fields:
```

(continues on next page)

(continued from previous page)

```
        - Name
        - Industry
        - Type
    soql_filter: "Industry = 'Higher Education' OR Type = 'Higher Education'"
```

Note that trying to load data that is extracted using `soql_filter` may cause "invalid cross reference id" errors if related object records are filtered on extract. Use this feature only if you fully understand how *CumulusCI load data task* resolves references to related records when loading data to a Salesforce org.

Primary Keys

CumulusCI offers two modes of managing Salesforce Ids and primary keys within the stored database.

If the `fields` list for an sObject contains a mapping:

```
Id: sf_id
```

CumulusCI will extract the Salesforce Id for each record and use that Id as the primary key in the stored database.

If no such mapping is provided, CumulusCI will remove the Salesforce Id from extracted data and replace it with an autoincrementing integer primary key.

Use of integer primary keys may help yield more readable text diffs when storing data in SQL script format. However, it comes at some performance penalty when extracting data. It's recommended that most mappings do not map the Id field and allow CumulusCI to utilize the automatic primary key.

Handling Namespaces

All CumulusCI bulk data tasks support automatic namespace injection or removal. In other words, the same mapping file will work for namespaced and unnamespaced orgs, as well as orgs with the package installed managed or unmanaged. If a mapping element has no namespace prefix and adding the project's namespace prefix is required to match a name in the org, CumulusCI will add one. Similarly, if removing a namespace is necessary, CumulusCI will do so.

In the extremely rare circumstance that an org contains the same mapped schema element in both namespaced and non-namespaced form, CumulusCI does not perform namespace injection or removal for that element.

Namespace injection can be deactivated by setting the `inject_namespaces` option to `False`.

The `generate_dataset_mapping` generates mapping files with no namespace and this is the most common pattern in CumulusCI projects.

Namespace Handing with Multiple Mapping Files

It's also possible, and common in older managed package products, to use multiple mapping files to achieve loading the same data set in both namespaced and non-namespaced contexts. This is no longer recommended practice.

A mapping file that is converted to use explicit namespacing might look like this:

Original version:

```
Destinations:
    sf_object: Destination__c
    fields:
        Name: Name
```

(continues on next page)

(continued from previous page)

```
        Target__c: Target__c
    lookups:
        Supplier__c:
            table: Supplier__c
```

Namespaced version:

```
Destinations:
    sf_object: MyNS__Destination__c
    table: Destination__c
    fields:
        MyNS__Name: Name
        MyNS__Target__c: Target__c
    lookups:
        MyNS__Supplier__c:
            key_field: Supplier__c
            table: Supplier__c
```

Note that each of the definition elements that refer to *local* storage remains un-namespaced, while those elements referring to the Salesforce schema acquire the namespace prefix.

For each lookup, an additional `key_field` declaration is required, whose value is the original storage location in local storage for that field's data. In most cases, this is simply the version of the field name in the original definition file.

Adapting an originally-namespaced definition to load into a non-namespaced org follows the same pattern, but in reverse.

Note that mappings which use the flat list style of field specification must use mapping style to convert between namespaced and non-namespaced deployment.

It's recommended that all new mappings use flat list field specifications and allow CumulusCI to manage namespace injection. This capability typically results in significant simplication in automation.

Optional Data Elements

Some projects need to build datasets that include optional data elements - fields and objects that are loaded into some of the project's orgs, but not others. This can cover both optional managed packages and features that are included in some, but not all, orgs. For example, a managed package A that does not require another managed package B but is designed to work with it may wish to include data for managed package B in its data sets, but load that data if and only if B is installed. Likewise, a package might wish to include data supporting a particular org feature, but not load that data in an org where the feature is turned off (and its associated fields and objects are for that reason unavailable).

To support this use case, the `load_dataset` and `extract_dataset` tasks offer a `drop_missing_schema` option. When enabled, this option results in CumulusCI ignoring any mapped fields, sObjects, or lookups that correspond to schema that is not present in the org.

Projects that require this type of conditional behavior can build their datasets in an org that contains managed package B, capture it, and then load it safely in orgs that both do and do not contain B. However, it's important to always capture from an org with B present, or B data will not be preserved in the dataset.

9.5 Custom Settings

Datasets don't support Custom Settings. However, a separate task is supplied to deploy Custom Settings (both list and hierarchy) into an org: *load_custom_settings*. The data for this task is defined in a YAML text file

Each top-level YAML key should be the API name of a Custom Setting. List Custom Settings should contain a nested map of names to values. Hierarchy Custom settings should contain a list, each of which contains a `data` key and a `location` key. The `location` key may contain either `profile: <profile name>`, `user: name: <username>`, `user: email: <email>`, or `org`.

Example:

```
List__c:
    Test:
        MyField__c: 1
    Test 2:
        MyField__c: 2
Hierarchy__c:
    - location: org
      data:
          MyField__c: 1
    - location:
          user:
              name: test@example.com
      data:
          MyField__c: 2
```

CumulusCI will automatically resolve the `location` specified for Hierarchy Custom Settings to a `SetupOwnerId`. Any Custom Settings existing in the target org with the specified name (List) or setup owner (Hierarchy) will be updated with the given data.

9.6 Dataset Tasks

9.6.1 create_bulk_data_permission_set

Create and assign a Permission Set that enables key features used in Bulk Data tasks (Hard Delete and Set Audit Fields) for the current user. The Permission Set will be called `CumulusCI Bulk Data`.

Note that prior to running this task you must ensure that your org is configured to allow the use of Set Audit Fields. You can do so by manually updating the required setting in the User Interface section of Saleforce Setup, or by updating your scratch org configuration to include :

```
"securitySettings": {
    "enableAuditFieldsInactiveOwner": true
}
```

For more information about the Set Audit Fields feature, review this Knowledge article.

After this task runs, you'll be able to run the `delete_data` task with the `hardDelete` option, and you'll be able to map audit fields like `CreatedDate`.

9.6.2 extract_dataset

Extract the data for a dataset from an org and persist it to disk.

Options

- mapping: the path to the YAML definition file for this dataset.
- sql_path: the path to a SQL script storage location for this dataset.
- database_url: the URL for the database storage location for this dataset.

mapping and either sql_path or database_url must be supplied.

Example: :

```
cci task run extract_dataset -o mapping datasets/qa/mapping.yml -o sql_path datasets/qa/
↪data.sql --org qa
```

9.6.3 load_dataset

Load the data for a dataset into an org. If the storage is a database, persist new Salesforce Ids to storage.

Options

- mapping: the path to the YAML definition file for this dataset.
- sql_path: the path to a SQL script storage location for this dataset.
- database_url: the URL for the database storage location for this dataset.
- start_step: the name of the step to start the load with (skipping all prior steps).
- ignore_row_errors: If True, allow the load to continue even if individual rows fail to load. By default, the load stops if any errors occur.

mapping and either sql_path or database_url must be supplied.

Example: :

```
cci task run load_dataset -o mapping datasets/qa/mapping.yml -o sql_path datasets/qa/
↪data.sql --org qa
```

9.6.4 generate_dataset_mapping

Inspect an org and generate a dataset definition for the schema found there.

This task is intended to streamline the process of creating a dataset definition. To use it, first build an org (scratch or persistent) containing all of the schema needed for the dataset.

Then, execute generate_dataset_mapping. The task inspects the target org and creates a dataset definition encompassing the project's schema, attempting to be minimal in its inclusion outside that schema. Specifically, the definition will include:

- Any custom object without a namespace
- Any custom object with the project's namespace

- Any object with a custom field matching the same namespace criteria
- Any object that's the target of a master-detail relationship, or a custom lookup relationship, from another included object.

On those sObjects, the definition will include

- Any custom field (including those defined by other packages)
- Any required field
- Any relationship field targeting another included object
- The `Id`, `FirstName`, `LastName`, and `Name` fields, if present

Certain fields will always be omitted, including

- Lookups to the User object
- Binary-blob (base64) fields
- Compound fields
- Non-createable fields

The resulting definition file is intended to be a viable starting point for a project's dataset. However, some additional editing is typically required to ensure the definition fully suits the project's use case. In particular, any fields required on standard objects that aren't automatically included must be added manually.

Reference Cycles

Dataset definition files must execute in a sequence, one sObject after another. However, Salesforce schemas often include *reference cycles*: situations in which Object A refers to Object B, which also refers to Object A, or in which Object A refers to itself.

CumulusCI will detect these reference cycles during mapping generation and ask the user for assistance resolving them into a linear sequence of load and extract operations. In most cases, selecting the schema's most core object (often a standard object like Account) will successfully resolve reference cycles. CumulusCI will automatically tag affected relationship fields with `after` directives to ensure they're populated after their target records become available.

Options

- `path`: Location to write the mapping file. Default: datasets/mapping.yml
- `ignore`: Object API names, or fields in Object.Field format, to ignore
- `namespace_prefix`: The namespace prefix to treat as belonging to the project, if any

Example: :

```
cci task run generate_dataset_mapping --org qa -o namespace_prefix my_ns
```

9.6.5 load_custom_settings

Load custom settings stored in YAML into an org.

Options

- `settings_path`: Location of the YAML settings file.

9.6.6 delete_data

You can also delete records using CumulusCI. You can either delete every record of a particular object, certain records based on a `where` clause or every record of multiple objects. Because `where` clauses seldom make logical sense when applied to multiple objects, you cannot use a `where` clause when specifying multiple objects.

Details are available with `cci org info delete_data` and [in the task reference] (delete-data).

Examples

```
cci task run delete_data -o objects Opportunity,Contact,Account --org qa

cci task run delete_data -o objects Opportunity -o where "StageName = 'Active' "

cci task run delete_data -o objects Account -o ignore_row_errors True

cci task run delete_data -o objects Account -o hardDelete True
```

9.6.7 update_data

To update records using CumulusCI, provide:

- a command line or task configuration describing what to update
- a recipe in a subset of Snowfakery syntax that says how to update it

On the command line, you can run an update like this:

```
$ cci task run update_data --recipe datasets/update.recipe.yml --object Account
```

This command downloads every Account in the org and applies the fields from the specified update recipe file.

You can filter the rows that you're updating like this:

```
$ cci task run update_data --recipe datasets/update.recipe.yml --object Account --where
"name like 'AAA%'"
```

The recipe for an update can be as simple as this:

```
object: Account
fields:
    NumberOfEmployees: 10000
```

You can use all of the power of `snowfakery` to add fake data:

```
object: Account
fields:
    NumberOfEmployees: 10_000
    BillingStreet:
        fake: Streetname
```

Using Snowfakery formulas, you can also refer to specific input fields like this:

```
object: Account
fields:
    Description: ${{input.Name}} is our favorite customer in ${{input.BillingCity}}
```

To tell CumulusCI to extract those fields and make them use the `fields` option:

```
$ cci task run update_data --recipe datasets/update.recipe.yml --object Account --Fields
Name,BillingCity
```

You can learn more about Snowfakery syntax in the next section.

9.7 Generate Fake Data

It is possible to use CumulusCI to generate arbitrary amounts of synthetic data using the `snowfakery` task. That task is built on the Snowfakery language. CumulusCI ships with Snowfakery embedded, so you do not need to install it.

To start, you will need a Snowfakery recipe. You can learn about writing them in the Snowfakery docs.

Once you have it, you can fill an org with data like this:

```
$ cci task run snowfakery --recipe datasets/some_snowfakery_recipe.yml
```

If you would like to execute the recipe multiple times to generate more data, you do so like this:

```
$ cci task run generate_and_load_from_yaml --run-until-recipe-repeated 400
```

Which will repeat the recipe 400 times.

There are two other ways to control how many times the recipe is repeated: `--run-until-records-loaded` and `--run-until-records-in-org`.

9.7.1 Generated Record Counts

Consider this example:

```
$ cci task run snowfakery --run-until-records-loaded 1000:Account
```

This would say to run the recipe until the task has loaded 1000 new Accounts. In the process, it might also load Contacts, Opportunities, custom objects oor whatever else is in the recipe. But it finishes when it has loaded 400 Accounts.

The counting works like this:

- Snowfakery always executes a *complete* recipe. It never stops halfway through. If your recipe creates more records than you need, you might overshoot. Usually the amount of overshoot is just a few records, but it depends on the details of your recipe.
- At the end of executing a recipe, it checks whether it has created enough of the object type mentioned by the `--run-until-records-loaded` parameter.
- If so, it finishes. If not, it runs the recipe again.

So if your recipe creates 10 Accounts, 5 Contacts and 15 Opportunities, then when you run the command above it will run the recipe 100 times (100*10=1000) which will generate 1000 Accounts, 500 Contacts and 1500 Opportunities.

`--run-until-records-in-org}` works similarly, but it determines how many times to run the recipe based on how many records are in the org at the start. For example, if the org already has 300 Accounts in it then:

```
$ cci task run snowfakery --run-until-records-in-org 1000:Account
```

Would be equivalent to `--run-until-records-loaded 700:Account` because one needs to add 700 Accounts to the 300 resdent ones to get to 1000.

9.7.2 Controlling the Loading Process

CumulusCI's data loader has many knobs and switches that you might want to adjust during your load. It supports a ".load.yml" file format which allows you to manipulate these load settings. The simplest way to use this file format is to make a file in the same directory as your recipe with a filename that is derived from the recipe's by replacing everything after the first "." with ".load.yml". For example, if your recipe is called "babka.recipe.yml" then your load file would be "babka.load.yml".

9.7.3 Batch Sizes

You can also control batch sizes with the `-o batch_size BATCHSIZE` parameter. This is not the Salesforce bulk API batch size. No matter what batch size you select, CumulusCI will properly split your data into batches for the bulk API.

You need to understand the loading process to understand why you might want to set the `batch_size`.

If you haven't set the `batch_size` then Snowfakery generates all of the records for your load job at once.

So the first reason why you might want to set the batch_size is because you don't have enough local disk space for the number of records you are generating (across all tables).

This isn't usually a problem though.

The more common problem arises from the fact that Salesforce bulk uploads are always done in batches of records a particular SObject. So in the case above, it would upload 1000 Accounts, then 500 Contacts, then 1500 Opportunities. (remember that our scenario involves a recipe that generates 10 Accounts, 5 Contacts and 15 Opportunities).

Imagine if the numbers were more like 1M, 500K and 1.5M. And further, imagine if your network crashed after 1M Accounts and 499K Contacts were uploaded. You would not have a single "complete set" of 10/5/15. Instead you would have 1M "partial sets".

If, by contrast, you had set your batch size to 100*000, your network might die more around the 250,000 Account mark, but you would have 200,000/20*[1] =10K _complete sets* plus some "extra" Accounts which you might ignore or delete. You can restart your load with a smaller goal (800K Accounts) and finish the job.

Another reason you might choose smaller batch sizes is to minimize the risk of row locking errors when you have triggers enabled. Turning off triggers is generally preferable, and CumulusCI has a task for doing for TDTM trigger handlers, but sometimes you cannot avoid them. Using smaller batch sizes may be preferable to switching to serial mode. If every SObject in a batch uploads less than 10,000 rows then you are defacto in serial mode (because only one "bulk mode batch" at a time is being processed).

In general, bigger batch sizes achieve higher throughput. No batching at all is the fastest.

Smaller batch sizes reduce the risk of something going wrong. You may need to experiment to find the best batch size for your use case.

---

[1] remember that our sets have 20 Accounts each

CHAPTER

TEN

ACCEPTANCE TESTING WITH ROBOT FRAMEWORK

CumulusCI comes with a testing framework called Robot Framework (or just Robot), which is specifically for writing acceptance tests. These are typically end-to-end tests that verify that the high-level requirements of a project have been satisfied. (Think "Add a new student and verify they have been assigned a mentor" or "Create a case plan when the student is not enrolled in a program".) Usually, this involves automating a browser session with Salesforce, but Robot can also be used to test new APIs created by your team.

Later sections of this document will show you how to write tests, call APIs, create custom keywords, and so on. But first there's a bit of manual configuration to do.

10.1 Get Started

The test that comes with CumulusCI opens a browser and performs some automation. For that to work, you need to install Chrome, and a driver for your specific version of Chrome. We don't ship this driver by default because browser versions are continually updating, and different platforms require different drivers.

If you don't already have Chrome on your machine, download and install it in the default location, and then download the appropriate driver from the chromedriver download page. Download the latest stable version that corresponds to your Chrome version, and place it where Robot can find it. This usually means `/usr/local/bin` for Linux and OSX-based systems. (It can go anywhere as long as it's on your PATH.)

For more information, see Getting Started on the chromedriver website.

Fun Fact

You can skip this step and still see Robot in action with CumulusCI. The tests will fail, but you can still see what it's like to run a test, and the output that it produces.

10.1.1 You Get a Test! And You Get a Test!

When you initialize a repository to work with CumulusCI (see Start a new CumulusCI Project), you automatically get a preconfigured `robot` task to run all of your Robot tests at the same time. We also install one example test, `create_contact.robot`, that shows how to write both browser-based and API-based tests. In fact, we've gone ahead and created a complete folder hierarchy for tests, test results, and everything else related to Robot, all starting in a folder named `robot` at the top of your repository.

```
<ProjectName>
├── robot
│   └── <ProjectName>
```

(continues on next page)

(continued from previous page)

```
        ├── doc
        ├── resources
        └── tests
            └── create_contact.robot
```

Note: The `create_contact.robot` file is in plain text, so you can open it with any text editor you have on your machine. One of the features we love about Robot is that the files are not in a proprietary format.

10.2 Run Your First Test

You can run all tests for a project with a simple command line. In case you don't have a default org defined, we'll include instructions on which scratch org to use.

```
$ cci task run robot --org dev
```

If all goes well, the browser pops up, navigates around a bit, and then closes. The output on your screen looks something like this, though you might see additional information about creating the scratch org.

```
$ cci task run robot --org dev
2021-08-04 16:28:32: Getting org info from Salesforce CLI for test-yeqqkbxks2ny@example.
↪com
2021-08-04 16:28:35: Beginning task: Robot
2021-08-04 16:28:35: As user: test-yeqqkbxks2ny@example.com
2021-08-04 16:28:35: In org: 00D0R000000Tz56
2021-08-04 16:28:35:
==============================================================================
Tests
==============================================================================
Tests.Create Contact
==============================================================================
Via API                                                               | PASS |
------------------------------------------------------------------------------
Via UI                                                                | PASS |
------------------------------------------------------------------------------
Tests.Create Contact                                                  | PASS |
2 tests, 2 passed, 0 failed
==============================================================================
Tests                                                                 | PASS |
2 tests, 2 passed, 0 failed
==============================================================================
Output:  /projects/<ProjectName>/robot/<ProjectName>/results/output.xml
Log:     /projects/<ProjectName>/robot/<ProjectName>/results/log.html
Report:  /projects/<ProjectName>/robot/<ProjectName>/results/report.html
```

Notice the three lines at the end that point to an XML file and two HTML files. These paths will be different on your machine and reflect the path to your repository. All Robot results go into the `robot/<ProjectName>/results` folder. These files are overwritten each time you run your Robot tests.

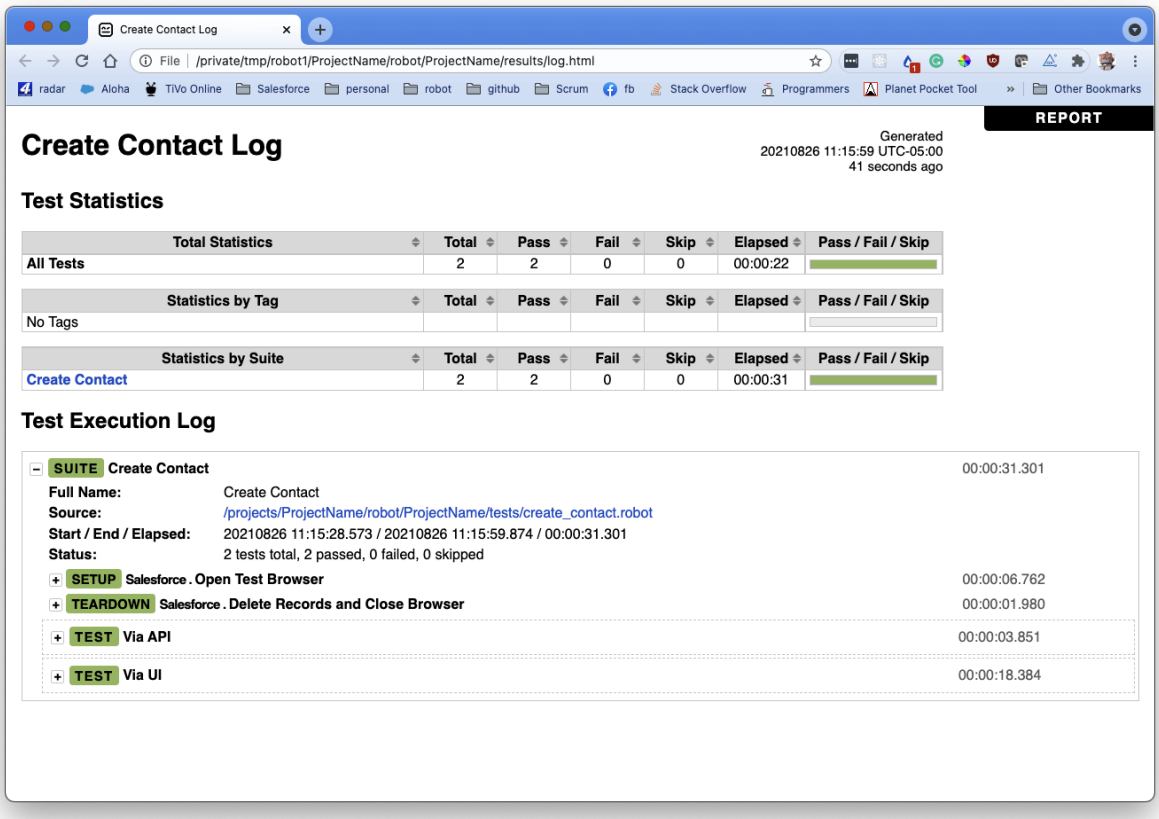
Robot places all of the test results in `output.xml`, and then generates `log.html` and `report.html`, which contain

two different human-readable views of the results. `log.html` is more developer-friendly and contains debugging information. `report.html` is a high-level report of successes and failures.

10.2.1 View Log and Report Files

You can open these files in a browser with the `open` command.

```
$ open robot/<ProjectName>/results/log.html
```



Feel free to open `output.xml` or `report.html` if you're curious. In our experience, `log.html` is the most useful for humans, and it's the one we use when reporting test results.

Want to learn more? The next section goes into more detail about why we love Robot Framework, and how you can write your own tests.

10.3 So Why Robot?

Robot is a keyword-driven acceptance testing framework, which means that users can write test cases in an intuitive, human-readable language made up of high-level, reusable keywords (`Open test browser`, `Delete records and close browser`) rather than in a programming language.

For example, this basic Robot test case file creates a new `Contact` record, and then examines the record to confirm that the fields listed are correct. You can see how straightforward the keyword syntax is. Even someone brand new to test automation can grasp the function of the `Salesforce Insert`, `Salesforce Get`, and `Should be equal` keywords.

```
*** Settings ***
Resource        cumulusci/robotframework/Salesforce.robot
Documentation   A simple Robot test

*** Test Cases ***
Create a Contact using the API

    ## Create a new Contact
    ${contact id}=    Salesforce Insert  Contact
    ...  FirstName=Eleanor
    ...  LastName=Rigby

    ## Get the new Contact and examine it
    &{contact}=       Salesforce Get  Contact  ${contact id}
    Should be equal   ${contact}[FirstName]    Eleanor
    Should be equal   ${contact}[LastName]     Rigby
```

10.4 The Robot Framework Advantage

Acceptance testing touches on multiple aspects of an application such as the data model, custom APIs, performance, and the user experience in the browser. Existing tools like Apex and Jest are good for writing unit tests and low-level integration tests. However, it can be difficult to understand the intent of a test, and the features being tested, when the test itself involves multiple lines of code detailing where to fetch data from, and how, and other such implementation details.

Robot addresses these challenges with a few strategies, helping you write high-level acceptance tests for every aspect of an application, often in a single test suite.

- Human-readable, domain-specific test cases: Robot lets you create a language tailored to the domain of testing Salesforce applications (a domain-specific language, or DSL). The DSL consists of reusable keywords that present a complex set of instructions in a human-readable language. The result? Test cases that all project stakeholders can easily understand, such as a product manager, scrum master, documentation teams, and so on–not just the test authors. In the previous example, `Salesforce Insert`, `Salesforce Get` and `Should be equal` are all keywords.
- Keyword libraries: Robot organizes keywords into libraries, which provide a simple, effective method to organize and share keywords between tests and projects. CumulusCI comes with a comprehensive standard library of Robot keywords created specifically to anticipate the needs of Salesforce testers. In the previous example, when you define `Salesforce.robot` as a resource, it automatically pulls in dozens of Salesforce-specific keywords.
- Streamlined test cases: Keywords allow implementation details to be handled by the test but not explicitly itemized in the test. In the previous example, a new `Contact` record is created with the `Salesforce Insert` keyword, but we don't see all the steps required to make an API call to create the record, such as getting an

access token, creating an API payload, making the API call, and parsing the results. We see only two keywords that communicate with Salesforce via an API: one to create the `Contact` record, and another to retrieve the new record to confirm it has the correct first and last names.

10.4.1 Robot-specific Tasks

CumulusCI integrates with Robot via custom tasks, such as:

- `robot`: Runs one or more Robot tests. This task is the most common.
- `robot_libdoc`: Runs the libdoc command, which creates an HTML file defining all the keywords in a library or resource file.
- `robot_testdoc`: Runs the testdoc command, which creates an HTML file documenting all the tests in a test suite.

Like with any CumulusCI task, you can get documentation and a list of arguments with the `cci task info` command. For example, `cci task info robot` displays documentation for the `robot` task.

10.4.2 Custom Keywords

CumulusCI provides a set of keywords unique to both Salesforce and CumulusCI for acceptance testing. These keywords can run other tasks, interact with Salesforce applications, call Salesforce APIs, and so on. For a list of all custom keywords provided by CumulusCI, see Keywords.html.

Tip: In addition to the keywords that come with CumulusCI, you can write project-specific keywords that are either based on existing keywords, or implemented in Python.

10.5 Write a Sample Robot Test Case

Now that you have a general understanding of why Robot is ideal for acceptance testing with CumulusCI, let's construct a test case file that creates a new `Contact` record.

1. Run `cci project init`, which creates the `create_contact.robot` test case file that comes standard whenever you initialize a project with CumulusCI.
2. In the `robot/<ProjectName>/tests` folder, save this code in a new file named `new_contact_record.robot`.

```
*** Settings ***
Resource        cumulusci/robotframework/Salesforce.robot
Documentation   A simple Robot test

*** Test Cases ***
Create a Contact using the API

    ## Create a new Contact
    ${contact id}=    Salesforce Insert  Contact
    ...  FirstName=Eleanor
    ...  LastName=Rigby
```

(continues on next page)

(continued from previous page)

```
## Get the new Contact and examine it
&{contact}=      Salesforce Get  Contact  ${contact id}
Should be equal  ${contact}[FirstName]    Eleanor
Should be equal  ${contact}[LastName]     Rigby
```

You can tell that both `create_contact.robot` and `new_contact_record.robot` are test case files because each one has a `.robot` extension and contains a `Test Cases` section. The `new_contact_record.robot` test case file is a simplified version of `create_contact.robot`. We feature it in this documentation for simpler code samples.

10.5.1 Syntax

Here's a quick primer on the syntax in the `new_contact_record.robot` test case file.

Symbol	Name	Description and Usage
***	Section Heading	A line that begins with one or more asterisks is a section heading. Byconvention, we use three asterisks on both sides of a heading to designatea section heading. Section headings include `Settings`, `Test Cases`, `Keywords`, `Variables`, `Comments`, and `Tasks`.
#	Hash	Designates code comments.
${}	Variable	Curly braces surrounding a name designate a variable. The lead $character refers to a single value. Variable names are case-insensitive. Spaces and underscores are allowedand are treated the same.
=	Assignment	Equals sign is optional yet convenient for showing that a variable isassigned a value. Before the equals sign, up to one space is allowed but *not* required. After the equals sign, two spaces are required, but moreare allowed to format test cases into readable columns.
...	Ellipses	Ellipses designate the continuation of a single-line row of code splitover multiple lines for easier readability.
	Space	Two or more spaces separate arguments from the keywords, and argumentsfrom each other. Multiple spaces can be used to align data and to aid inreadability.

For more details on Robot syntax, visit the official Robot syntax documentation.

Tip: For users interested in linting their Robot Framework code, we recommend checking out +robotframework-robocop. It's a static code analysis tool for Robot Framework that can help maintain good code quality.

10.5.2 Settings

The `Settings` section of the `.robot` file sets up the entire test suite. Configurations established under `Settings` affect all test cases, such as:

- `Suite Setup` and `Suite Teardown`, which support processes before the test begins and cleanup after the test finishes.
- `Documentation`, which describes the purpose of the test suite.
- `Tags`, which lets a user associate individual test cases with a label.
- `Resource`, which imports keywords from external files.

For example, these are the settings stored in the `new_contact_record.robot` file.

```
*** Settings ***
Resource        cumulusci/robotframework/Salesforce.robot
Documentation   A simple Robot test
```

The `cumulusci/robotframework/Salesforce.robot` resource file comes with CumulusCI and automatically inherits useful configuration and keywords for Salesforce testing. The `Salesforce.robot` resource file is the primary method of importing all keywords and variables provided by CumulusCI, so it's best practice for the file to be the first item imported as a `Resource` under `Settings`. It also imports the CumulusCI Library, the Salesforce Library, the third-party SeleniumLibrary for browser testing via Selenium, and these most commonly used Robot libraries.

- Collections
- OperatingSystem
- String
- XML

CumulusCI also comes bundled with these third-party keyword libraries, which must be explicitly imported by any test suite that needs them.

- RequestsLibrary for testing REST APIs. To use `RequestsLibrary`, explicitly import it under the `Settings` section of your Robot test.
- All other libraries listed in the Standard tab of the Robot libraries documentation.

10.5.3 Test Cases

In the `Test Cases` section of the `.robot` file, each test case gets its own code block; the test case name is the first line of code, with no indentation. The body of the test case is all the indented text underneath.

For example, here is the `Test Cases` section of the `new_contact_record.robot` test case file. It has a single test case named `Create a Contact using the API`.

```
*** Test Cases ***
Create a Contact using the API

    ## Create a new Contact
    ${contact id}=    Salesforce Insert  Contact
    ...  FirstName=Eleanor
    ...  LastName=Rigby

    ## Get the new Contact and examine it
    &{contact}=       Salesforce Get  Contact  ${contact id}
    Should be equal   ${contact}[FirstName]    Eleanor
    Should be equal   ${contact}[LastName]     Rigby
```

Notice these keywords used in the test case.

- `Salesforce Insert` creates a new `Contact` record with the arguments it's given for the `FirstName` and `LastName` fields.
- `Salesforce Get` retrieves the requested `Contact` record based on its ID.
- `Should Be Equal` compares the arguments to the values of the `FirstName` and `LastName` fields of the newly created `Contact` record.

Tip: Keywords in the test cases are separated from arguments by two or more spaces.

10.6 Suite Setup and Teardown

Most real-world tests require setup before the test begins (such as opening a browser or creating test data), and cleanup after the test finishes (such as closing the browser or deleting test data). Robot supports setup and teardown at both the suite level (such as opening the browser before the first test, *and* closing the browser after the last test) and the test level (such as opening and closing the browser at the start *and* the end of the test).

If you run the `new_contact_record.robot` test case file several times, you add a new `Contact` record to your scratch org each time it runs. If you have a test that requires a specific number of `Contact` records, the test can fail the second time you run it. To maintain the required record count, you can add a teardown that deletes any `Contact` records created by running the test.

Let's modify the `new_contact_record.robot` test case file with a `Suite Teardown` that deletes the `Contact` records created by any tests in the suite.

```
*** Settings ***
Resource        cumulusci/robotframework/Salesforce.robot
Documentation   A simple Robot test
Suite Teardown  Delete session records

*** Test Cases ***
Create a Contact using the API

    ## Create a new Contact
    ${contact id}=    Salesforce Insert  Contact
    ...  FirstName=Eleanor
    ...  LastName=Rigby

    ## Get the new Contact and examine it
    &{contact}=       Salesforce Get  Contact  ${contact id}
    Should be equal  ${contact}[FirstName]    Eleanor
    Should be equal  ${contact}[LastName]     Rigby
```

Note: The `Salesforce Insert` keyword keeps track of the record IDs created. The `Delete session records` keyword deletes those records.

To run this test from the command line:

```
$ cci task run robot --suites robot/<ProjectName>/tests/new_contact_record.robot
```

10.7 Generate Fake Data with Faker

The `get fake data` keyword comes with the Faker library that's installed with CumulusCI, and saves you from hard-coding test data for Robot tests. `Get fake data` does much more than just return random strings; it generates strings in an appropriate format. You can ask it for a name, address, date, phone number, credit card number, and so on, and get back properly formatted data.

For example, let's modify the `new_contact_record.robot` test case file to generate a fake name. Because the new `Contact` name is randomly generated in this updated example, you can't hard-code an assertion on the name of the created `Contact` to verify the name. Instead, for illustrative purposes, this test logs the `Contact` name in the test's `log.html` file.

```
*** Settings ***
Resource        cumulusci/robotframework/Salesforce.robot
Documentation   A simple Robot test
Suite Teardown  Delete session records

*** Test Cases ***
Create a Contact with a generated name
   [Teardown]       Delete session records

   ## Generate a name to use for Contact
   ${first name}=   Get fake data  first_name
   ${last name}=    Get fake data  last_name

   ## Create a new Contact
   ${contact id}=   Salesforce Insert  Contact
   ...  FirstName=${first name}
   ...  LastName=${last name}

   ## Get the new Contact and add name to the log
   &{contact}=      Salesforce Get  Contact  ${contact id}
   Log  Contact name: ${contact}[Name]
```

To run this test from the command line:

```
$ cci task run robot --suites robot/<ProjectName>/tests/new_contact_record.robot
```

10.8 Create Custom Keywords

We mentioned earlier that Robot makes use of a domain-specific language. By creating a collection of reusable custom keywords, we can create this DSL for testing Salesforce apps.

Let's create a new Robot test that includes a custom keyword called `Create a test Contact`, which creates a `Contact` record and then saves the data for this record in a test variable. Save this code in a file named `custom_keyword.robot` in the `tests` folder of your project's repository.

```
*** Settings ***
Resource        cumulusci/robotframework/Salesforce.robot
Suite Teardown  Delete session records

*** Test Cases ***
```

(continues on next page)

(continued from previous page)

```
Example of using a custom keyword in a setup step
   [Setup]      Create a test Contact

   ## Get the new Contact that's stored in a test variable,
   ## and add the name to the log
   Log  New Contact: ${new contact}[Name]

*** Keywords ***
Create a test Contact
   [Documentation]  Create a temporary Contact and return it

   ## Generate a name to use for Contact
   ${first name}=   Get fake data  first_name
   ${last name}=    Get fake data  last_name

   ## Create a new Contact
   ${contact id}=   Salesforce Insert  Contact
   ...  FirstName=${first name}
   ...  LastName=${last name}

   ## Fetch the Contact object to be returned and save
   ## it to a test variable
   &{new contact}=    Salesforce Get    Contact    ${contact_id}
   Set test variable  &{new contact}
```

Because the `Contact` record was created inside the `Create a test Contact` keyword, the `&{new contact}` variable is not going to be visible to any test case or keyword that calls the `Create a test Contact` keyword. It's only when we use the built-in keyword Set test variable that the newly created `&{new contact}` variable becomes visible in the `Example of using a custom keyword in a setup step` test case.

You can create settings for individual test cases and keywords by specifying the setting names in square brackets. These settings are often used to override setting defaults in the Setting section. In the previous example:

- `[Setup]` is a setting for the `Example of using a custom keyword in a setup step` test case.
- `[Documentation]` is a setting for the `Create a test Contact` keyword.

For details, see the Settings in the Test Case section in the official Robot Framework documentation.

To run this test from the command line:

```
$ cci task run robot --suites robot/<ProjectName>/tests/custom_keyword.robot
```

10.9 Create a Resource File

Now that you know how to create a reusable custom keyword in a test case file, you can build a library of custom keywords to be shared project-wide with a resource file.

A resource file is similar to a test case file, except it can't contain test cases. Typically, a resource file stores settings that are used by every test in the project, such as defining project-specific variables, or importing project-specific keyword libraries and resource files.

Let's create a resource file that stores the `Create a test Contact` custom keyword, which is currently in the `custom_keyword.robot` test case file defined in *Create Custom Keywords*. There aren't any requirements for nam-

ing resource files. However, most teams have standardized creating a resource file named after the project, such as `NPSP.robot` for NPSP.

For this example, we'll stick to this convention and create a file named after your project. Save this code in a file named `robot/<ProjectName>/resources/<ProjectName>.robot`.

```
*** Settings ***
Resource        cumulusci/robotframework/Salesforce.robot

Create a test Contact
   [Documentation]  Create a temporary Contact and return it

   ## Generate a name to use for Contact
   ${first name}=   Get fake data  first_name
   ${last name}=    Get fake data  last_name

   ## Create a new Contact
   ${contact id}=   Salesforce Insert  Contact
   ...  FirstName=${first name}
   ...  LastName=${last name}

   ## Fetch the Contact object to be returned and save
   ## it to a test variable
   &{new contact}=    Salesforce Get   Contact    ${contact_id}
   Set test variable  &{new contact}
```

Note: Along with moving the `Keywords` section in the `custom_keyword.robot` test case file to this file, you must also import `Salesforce.robot` as a `Resource` because that's where the Faker library is defined.

Next, let's modify the `custom_keyword.robot` test case file. Remove the `Keywords` section, and then under `Settings`, add as many `Resource` statements as needed to import keywords from their specific `.robot` resource files.

```
*** Settings ***
Resource        cumulusci/robotframework/Salesforce.robot
Resource        <ProjectName>/resources/<ProjectName>.robot

Suite Teardown  Delete session records

*** Test Cases ***
Example of using a custom keyword in a setup step
   [Setup]      Create a test Contact

   ## Get the new Contact that's stored in a test variable,
   ## and add the name to the log
   Log  New Contact: ${new contact}[Name]
```

Note: Keywords defined in resource files are accessible to all tests in a suite that imports the resource files.

10.10 Create a Simple Browser Test

Now that you know how to create records using the API, you can use those records in a browser test.

Let's create a Robot test that uses `Suite Setup` to call the `Open test browser` keyword. Save this code in a file named `ui.robot` in the `tests` folder of your project's repository.

```
*** Settings ***
Resource        cumulusci/robotframework/Salesforce.robot

Suite Setup     Open test browser
Suite Teardown  Delete records and close browser

*** Test Cases ***
Take screenshot of landing page
   Wait until page contains  Most Recently Used
   Capture page screenshot
```

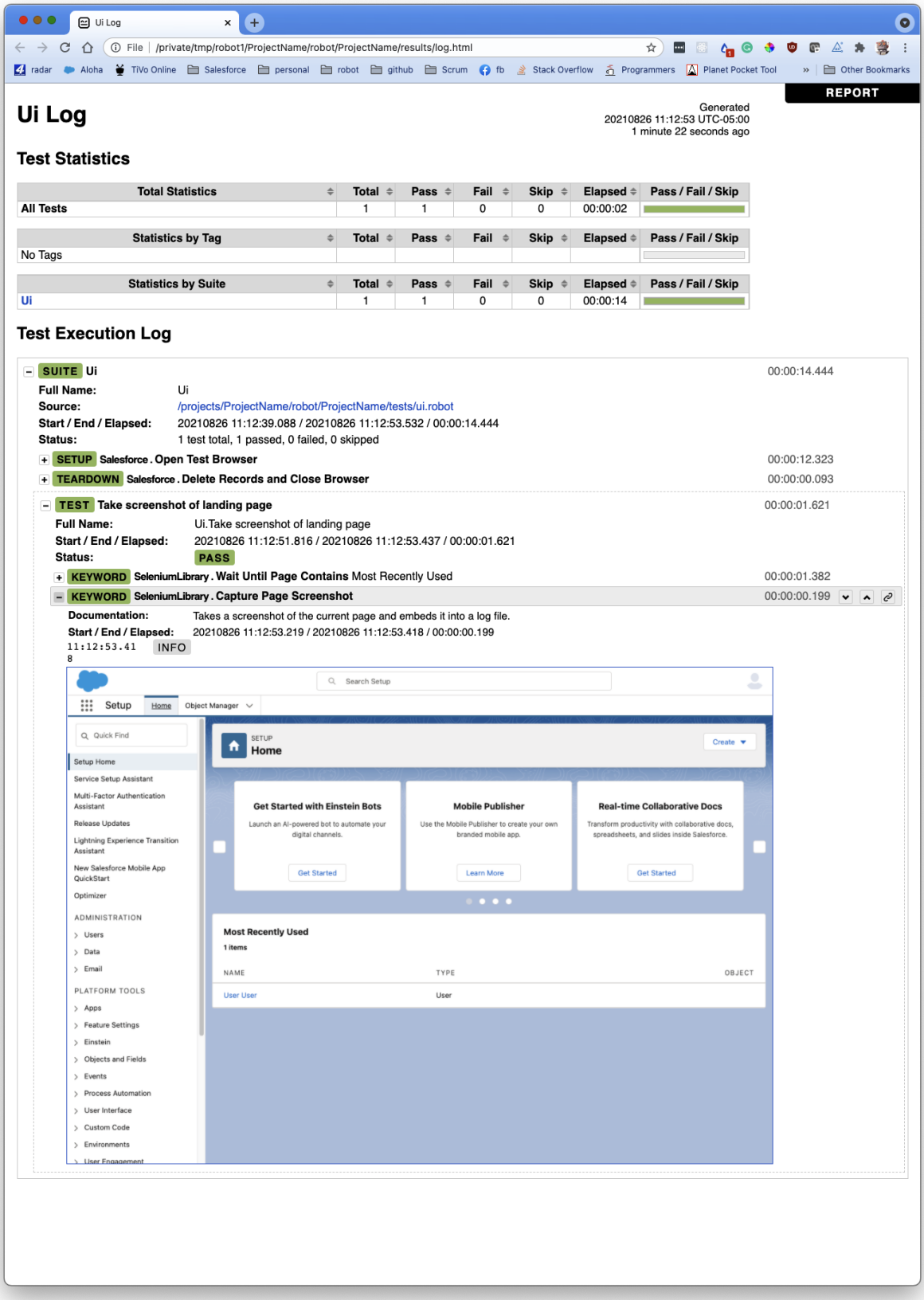
Because this test case file calls `Open test browser`, a browser window appears while the test runs. The test case takes a screenshot, which can be a useful tool when debugging tests (a tool used sparingly because screenshots can take up a lot of disk space). `Suite Teardown` then calls the `Delete records and close browser` keyword to complete the test.

Note: `Open test browser` doesn't always wait long enough for Salesforce to render. That's why the `Wait until page contains` keyword comes in handy. It waits until the "Most Recently Used" section of the web page appears, which is a good indication that the site has loaded.

To run this test from the command line:

```
$ cci task run robot --suites robot/<ProjectName>/tests/ui.robot
```

In addition to the usual output files (`log.html`, `report.html`, `output.xml`), this test also creates a screenshot in the `results` folder. If you open `log.html`, you can see whether each step of the test case passed or failed. Toggle the + tab of the `Take screenshot of landing page` test header to examine the results of the test. Then toggle the + tab of the `Capture page screenshot` keyword to examine the screenshot taken of the landing page.


Ui Log
File /private/tmp/robot1/ProjectName/robot/ProjectName/results/log.html
radar Aloha TiVo Online Salesforce personal robot github Scrum fb Stack Overflow Programmers Planet Pocket Tool Other Bookmarks
REPORT
Ui Log
Generated
20210826 11:12:53 UTC-05:00
1 minute 22 seconds ago
Test Statistics
Total Statistics | Total | Pass | Fail | Skip | Elapsed | Pass / Fail / Skip
All Tests | 1 | 1 | 0 | 0 | 00:00:02
Statistics by Tag | Total | Pass | Fail | Skip | Elapsed | Pass / Fail / Skip
No Tags
Statistics by Suite | Total | Pass | Fail | Skip | Elapsed | Pass / Fail / Skip
Ui | 1 | 1 | 0 | 0 | 00:00:14
Test Execution Log
SUITE Ui 00:00:14.444
Full Name: Ui
Source: /projects/ProjectName/robot/ProjectName/tests/ui.robot
Start / End / Elapsed: 20210826 11:12:39.088 / 20210826 11:12:53.532 / 00:00:14.444
Status: 1 test total, 1 passed, 0 failed, 0 skipped
SETUP Salesforce . Open Test Browser 00:00:12.323
TEARDOWN Salesforce . Delete Records and Close Browser 00:00:00.093
TEST Take screenshot of landing page 00:00:01.621
Full Name: Ui.Take screenshot of landing page
Start / End / Elapsed: 20210826 11:12:51.816 / 20210826 11:12:53.437 / 00:00:01.621
Status: PASS
KEYWORD SeleniumLibrary . Wait Until Page Contains Most Recently Used 00:00:01.382
KEYWORD SeleniumLibrary . Capture Page Screenshot 00:00:00.199
Documentation: Takes a screenshot of the current page and embeds it into a log file.
Start / End / Elapsed: 20210826 11:12:53.219 / 20210826 11:12:53.418 / 00:00:00.199
11:12:53.418 INFO
Search Setup
Setup Home Object Manager
Quick Find
Setup Home
Service Setup Assistant
Multi-Factor Authentication Assistant
Release Updates
Lightning Experience Transition Assistant
New Salesforce Mobile App QuickStart
Optimizer
ADMINISTRATION
Users
Data
Email
PLATFORM TOOLS
Apps
Feature Settings
Einstein
Objects and Fields
Events
Process Automation
User Interface
Custom Code
Environments
SETUP
Home
Create
Get Started with Einstein Bots
Launch an AI-powered bot to automate your digital channels.
Get Started
Mobile Publisher
Use the Mobile Publisher to create your own branded mobile app.
Learn More
Real-time Collaborative Docs
Transform productivity with collaborative docs, spreadsheets, and slides inside Salesforce.
Get Started
Most Recently Used
1 items
NAME TYPE OBJECT
User User User

10.10.1 Open the Browser

The Selenium library comes with a keyword for opening the browser. However, CumulusCI comes with its own keyword, Open Test Browser, which not only opens the browser but also takes care of the details of logging into the org. This keyword uses a variable named `${BROWSER}`, which can be set from the command line or in the `cumulusci.yml` file to specify which browser to use.

Specify variables in the `cumulusci.yml` file or in the `vars` option under `robot` in the `tasks` section. For example, `${BROWSER}` defaults to `chrome` in Robot, but it can be set to `firefox`.

```
tasks:
   robot:
      options:
      vars:
         - BROWSER:firefox
```

To set the browser to `firefox` from the command line *for a single test run*:

```
$ cci task run robot --vars BROWSER:firefox
```

10.10.2 Supported Browsers

The `robot` task supports both Chrome and Firefox browsers, and the headless variations of these browsers, `headlesschrome` and `headlessfirefox`. With the headless version, browser tests run without opening a browser window. The tests still use a browser, but you can't see it while the test runs. This variation is most useful when you run a test on a continuous integration server like MetaCI, where a physical display isn't connected to the server.

To specify the headless version of a browser, prepend `headless` to the browser name. For example, the command line option to specify headless Chrome is `--var BROWSER:headlesschrome`.

Tip: When you run a test in headless mode, you can still capture screenshots of the browser window. The `Capture Page Screenshot` keyword is indispensable for debugging tests that failed in headless mode.

10.11 Combine API Keywords and Browser Tests

In Robot, API and browser keywords can be used together to build more elaborate acceptance tests.

Let's build on the original `new_contact_record.robot` test to integrate the previous configurations covered so far. Replace the entirety of the `new_contact_record.robot` test case file in the `tests` folder of your project's repository with this code.

```
*** Settings ***
Resource        cumulusci/robotframework/Salesforce.robot
Documentation   A simple Robot test

Suite Setup     Open test browser
Suite Teardown  Delete records and close browser

*** Test Cases ***
Take screenshot of list of Contacts
   [Setup]  Create a test Contact
```

(continues on next page)

(continued from previous page)

```
    Go to object home  Contact
    Capture page screenshot

*** Keywords ***
Create a test Contact
    [Documentation]  Create a temporary Contact and return the ID
    [Return]         ${contact id}

    ## Generate a name to use for Contact
    ${first name}=   Get fake data  first_name
    ${last name}=    Get fake data  last_name

    ## Create a new Contact
    ${contact id}=   Salesforce Insert  Contact
    ...  FirstName=${first name}
    ...  LastName=${last name}
```

The `new_contact_record.robot` test case file not only creates a `Contact` record, it also opens the browser to see that the `Contact` record appears in a list of `Contacts`, takes a screenshot of the list, then deletes all new records created during the test run, and closes the browser.

To run this test from the command line:

```
$ cci task run robot --suites robot/<ProjectName>/tests/new_contact_record.robot --org␣
↪dev
```

10.12 Run an Entire Test Suite

At this point, the `robot` folder in your project repository should look like this.

```
<ProjectName>
├── robot
│   └── <ProjectName>
│       ├── doc
│       ├── resources
│       │   └── <ProjectName>.robot
│       ├── results
│       │   ├── log.html
│       │   ├── output.xml
│       │   ├── report.html
│       │   ├── selenium-screenshot-1.png
│       │   └── selenium-screenshot-2.png
│       └── tests
│           ├── create_contact.robot
│           ├── custom_keyword.robot
│           ├── new_contact_record.robot
│           └── ui.robot
```

While a single `.robot` file is considered to be a test suite, Robot also considers folders to be suites. You can pass a folder to Robot to run all tests stored in that folder. So if you've saved the `new_contact_record.robot`, `custom_keyword.robot`, and `ui.robot` test case files in the `tests` folder, you can run all of the tests in the command line.

```
$ cci task run robot --suites robot/<ProjectName>/tests --org dev
```

In the output, you can see that all of the test case files in the `tests` folder have been run, including the `create_contact.robot` test case file that comes with CumulusCI.

```
$ cci task run robot --suites robot/<ProjectName>/tests --org dev
2021-08-24 16:45:36: Getting org info from Salesforce CLI for test-4g5sxdzt9sj3@example.
↪com
2021-08-24 16:45:39: Beginning task: Robot
2021-08-24 16:45:39: As user: test-4g5sxdzt9sj3@example.com
2021-08-24 16:45:39: In org: 00D56000000KC1g
2021-08-24 16:45:39:
==============================================================================
Tests
==============================================================================
Tests.Create Contact
==============================================================================
Via API                                                               | PASS |
------------------------------------------------------------------------------
Via UI                                                                | PASS |
------------------------------------------------------------------------------
Tests.Create Contact                                                  | PASS |
2 tests, 2 passed, 0 failed
==============================================================================
Tests.Custom Keyword
==============================================================================
Example of using a custom keyword in a setup step                     | PASS |
------------------------------------------------------------------------------
Tests.Custom Keyword                                                  | PASS |
1 test, 1 passed, 0 failed
==============================================================================
Tests.New Contact Record :: A simple Robot test
==============================================================================
Take screenshot of list of Contacts                                   | PASS |
------------------------------------------------------------------------------
Tests.New Contact Record :: A simple Robot test                       | PASS |
1 test, 1 passed, 0 failed
==============================================================================
Tests.Ui :: A simple Robot test
==============================================================================
Take screenshot of list of Contacts                                   | PASS |
------------------------------------------------------------------------------
Tests.Ui :: A simple Robot test                                       | PASS |
1 test, 1 passed, 0 failed
==============================================================================
Tests                                                                 | PASS |
5 tests, 5 passed, 0 failed
==============================================================================
Output:  /projects/<ProjectName>/robot/<ProjectName>/results/output.xml
Log:     /projects/<ProjectName>/robot/<ProjectName>/results/log.html
Report:  /projects/<ProjectName>/robot/<ProjectName>/results/report.html
```

Tip: Test suite folders can also contain nested folders of tests, which makes it easy to organize tests into functional

groups. For example, you can store all API tests in a `tests/api` folder, and store all UI tests in a `tests/ui` folder.

Because running everything in the `tests` folder is such common practice, it's the default behavior for the `robot` task.

To run an entire suite of tests with the `robot` task:

```
$ cci task run robot --org dev
```

10.13 Learn More About Robot Framework

To learn more about Robot, visit the Robot Framework User Guide. We also have these resources in the CumulusCI documentation.

10.13.1 Managing Locators

The keywords that come with CumulusCI are based on the open source keyword library SeleniumLibrary. This library supports multiple ways to reference an element: by XPath, by CSS selector, by id, by text, and so on. SeleniumLibrary calls these *location strategies*.

You can specify a strategy by providing a prefix to the locator. For example:

- `id:123` specifies an element with an id of 123
- `xpath://div[text()='Hello, world']` lets you specify an element by an xpath expression
- `css:div.slds-spinner` defines an object by its css path

Tip: You can find the full list of supported locator strategies in the section titled Explicit locator strategy in the SeleniumLibrary documentation.

In this section, we'll show how to create a project-specific locator strategy by storing locators in a dictionary and then associating them with a custom prefix.

Storing locators in a dictionary

The first step toward creating custom locator strategies with the locator manager is to define your project's locators in a dictionary. If you have just a handful of locators you can define them directly in a keyword library. You can also save them in a separate file.

If you need to be able to run tests against a prerelease org you might want to store your locators in two files: one for the current release and one for the prerelease. You can then import the appropriate version at runtime.

Note: In order to keep the examples short we're only going to focus on supporting one release at a time in this documentation.

The locator dictionary can include nested dictionaries, so you can organize locators into logical groups. Each leaf node can be any locator string supported by SeleniumLibrary. Notice that these locator strings can include locators of different types.

For example, consider the following set of locators which we might find in a library of keywords for dealing with the calendar tab:

```
locators = {
    "sidebar": {
        "options button": "css:a[role='button'][title='Calendar Options']",
        "new button": "css:a[role='menuitem'][title='New Calendar']",
    },
    "modal": {
        "window": "xpath://div[@role='dialog'][.//h2[.='Create Calendar']]",
        "next button": "css:a.wzButtonSaveAndNext",
    }
}
```

We've organized the locators into two logical groupings: one related to elements on the sidebar, and one related to elements of a modal window. Notice also that three of the locators are CSS selectors and one is an XPath.

Tip: Dictionaries can be nested as deeply as you want, but it's rarely necessary to have locators more than a couple of levels deep.

Registering the locator dictionary

SeleniumLibrary provides a way to register custom location strategies via the Add Location Strategy keyword. While it's possible to write your own strategies using keywords, the locator manager makes it easy to associate a locator prefix with a dictionary of locators.

This registration is done via the `register_locators` method of the locator manager, and should be done in the `__init__` method of a keyword library.

For example, here is what it might look like for a library that contains keywords for the calendar tab.

```
from robot.libraries.BuiltIn import BuiltIn
from cumulusci.robotframework import locator_manager

locators = {...}  ## see previous example

class CalendarLibrary:
    ROBOT_LIBRARY_SCOPE = "GLOBAL"

    def __init__(self):
        locator_manager.register_locators("calendar", locators)
```

When this library is imported into a test case file, the prefix "calendar" is registered with SeleniumLibrary as a custom locator strategy.

Using custom locators

Once the dictionary has been defined and has been registered with a prefix, the locators work very similarly to any other locator. If the dictionaries are nested, you can separate the levels with a period (ie: dot notation).

For example, with our example locators the options button locator can be used like this:

```
Click element   calendar:sidebar.options button
```

The following table shows how the locator is parsed:

`calendar:` locator prefix

`sidebar` first level of the dictionary (eg: `locators['sidebar']`)

`.` a level separator

`options button` the next level of a nested dictionary (eg: `locators['sidebar']['options_button']`)

Parameterized Locators

Sometimes the only difference between multiple elements on a page is the text displayed in that element. For example, the html markup for a save, edit, and cancel button may be identical except for the word "Save", "Edit", or "Cancel".

While you can create a separate locator for each button, it's better to use a single parameterized locator for multiple buttons, which gives you more flexibility.

Notice in our calendar locators we have one locator for a `menuitem` with the title of 'New Calendar':

```
locators = {
    ...
    "new_button": "css:a[role='menuitem'][title='New Calendar']",
    ...
}
```

For a calendar menu with multiple menuitems, you can use a unique locator for each, or a single parameterized locator so that you only need to maintain one locator.

To create a locator with one or more parameters, replace a portion of the locator with [{}]{.title-ref} like this:

```
locators = {
    ...
    "menu_item": "css:a[role='menuitem'][title='{}']",
    ...
}
```

When you use the locator, you can pass one or more parameters by specfying a comma separated list of values after a colon. For example:

```
Click element  calendar:sidebar.menu_item:New Calendar
```

The [{}]{.title-ref} placeholders are replaced with the parameter values, in order. For example, the title in the above example becomes [New Calendar]{.title-ref}.

Note: If your locator has more than one parameter (ie: more than one instance of [{}]{.title-ref} within the locator definition), parameters will be replaced in the order in which they are supplied. The first parameter after the [:]{.title-ref} and before a comma will be used in place of the first [{}]{.title-ref}, the next parameter will be used in place of the next [{}]{.title-ref}, and so on.

10.13.2 Robot Advanced Topics

In the previous section we gave a broad overview of how Robot Framework is integrated with CumulsCI. In this section we'll take a deeper dive into some advanced topics.

Running CumulusCI Tasks

CumulusCI provides two keywords for running a task from within a robot test case: Run Task and Run Task Class.

Run Task can be used to run any CumulusCI tasks configured for the project. Tasks run can be any of CumulusCI's standard tasks as well as project-specific custom tasks from the project's `cumulusci.yml` file. `Run Task` accepts a single argument, the task name, along with any arguments required by the task.

Run Task Class works in a similar fashion, but the task can be specified as a python class rather than a task name. For example, you can use this keyword to run logic from CumulusCI tasks which have not been configured in the project's cumulusci.yml file. This is most useful in cases where a test needs to use task logic for logic unique to the test and thus not worth making into a named task for the project.

Performance Testing

The Salesforce keyword library somes with several keywords to aid in performance testing.

Setting the elapsed time

Normally, the full execution time of a test is recorded in the robot framework log. This includes the time spent in both test setup and teardown. Sometimes it is preferable to report only the time spent in the test case itself.

The Set Test Elapsed Time keyword allows you to record a computed elapsed time. For example, when performance testing a Salesforce batch process, you have the option to store the Salesforce-measured elapsed time of the batch process instead of the time measured in the CumulusCI client process.

The Set Test Elapsed Time keyword takes a single optional argument, either a number of seconds or a Robot time string.

When using this keyword, the tag `cci_metric_elapsed_time` will automatically be added to the test case.

When the test is run via MetaCI, the computed time will be retrieve and stored inside MetaCI instead of the total elapsed time as measured by Robot Framework.

Start and End Performance Time

A time can be recorded for any group of keywords by calling Start Performance Timer and Stop Performance Timer. The latter will automatically call the Set Test Elapsed Time keyword.

The Start Performance Timer keyword starts a timer. The Stop Performance Timer keyword stops the timer and stores the result with Set Test Elapsed Time.

Setting Test Metrics

The Set Test Metric keyword retrieves any metric for performance monitoring, such as number of queries, rows processed, CPU usage, and more.

The keyword takes a metric name, which can be any string, and a value, which can be any number.

Using this keyword will automatically add the tag `cci_metric` to the test case and `${cci_metric_<metric_name>}` to the test's variables. These permit downstream processing in tools like CCI and MetaCI.

Note: `cci_metric` is not included in Robot's html statistical roll-ups.

```
Set Test Metric    Max_CPU_Percent    30
```

Performance test metrics are output in the CCI logs, log.html and output.xml. MetaCI captures them but does not currently have a user interface for displaying them.

Elapsed Time for Last Record

The Elapsed Time For Last Record keyword queries Salesforce for its recorded log of a job.

For example, to query an Apex bulk job:

```
${time_in_seconds} =    Elapsed Time For Last Record
...             obj_name=AsyncApexJob
...             where=ApexClass.Name='BlahBlah'
...             start_field=CreatedDate
...             end_field=CompletedDate
...             order_by=CompletedDate
```

Browser Testing

Testing salesforce from within a browser presents some unique challenges. This section covers some Salesforce-specific features of our keyword libraries.

Waiting for Lightning UI

A common challenge when writing end-to-end UI tests is waiting for asynchronous actions to complete before proceeding to run the next interaction. The Salesforce Library is aware of the Lightning UI and can handle waiting automatically. After each click, the Salesforce Library waits for any pending requests to the server to complete. (Manually waiting using "sleep", or waiting for a particular element to appear, can still be necessary after specific interactions, and when interacting with pages that don't use the Lightning UI.)

API Keywords

In addition to browser interactions, the Salesforce Library also provides keywords for interacting with the Salesforce REST API. Here are the keywords we provide which talk directly to Salesforce via an API rather than through the UI:

- Salesforce Collection Insert: Creates a collection of objects based on a template.
- Salesforce Collection Update: Updates a collection of objects.
- Salesforce Delete: Deletes a record using its type and ID.
- Salesforce Get: Gets a dictionary of a record from its ID.
- Salesforce Insert: Inserts a record using its type and field values. Returns the ID.
- Salesforce Query: Runs a simple query using the `object type` and `<field_name=value>` syntax. Returns a list of matching record dictionaries.
- Salesforce Update: Updates a record using its type, ID, and `<field_name=value>` syntax.
- SOQL Query: Runs a SOQL query and returns a REST API result dictionary.

Using Page Objects ——————

The PageObjects library provides support for page objects, Robot Framework-style. Even though Robot is a keyword-driven framework, it's also possible to dynamically load in keywords unique to a page or an object on the page.

With the `PageObjects` library, you can define classes that represent page objects. Each class provides keywords that are unique to a page or a component. These classes can be imported on demand only for tests that use these pages or components.

The `pageobject` Decorator

Page objects are normal Python classes that use the `pageobject` decorator provided by CumulusCI. Unlike traditional Robot Framework keyword libraries, you can easily define and use keywords in multiple classes within a single file.

To create a page object class, start by inheriting from one of the provided base classes. You need to use the `pageobject` decorator to designate the class as a page object, and to describe the type of page (Listing, Detail, etc) and the associated salesfore object. From within a test, these page objects are referenced using both the type and object name (eg: `Go to page Listing CustomObject__c`).

The following example illustrates how to create a `Listing` page object for `CustomObject__c`.

```
from cumulusci.robotframework.pageobjects import ListingPage, pageobject

@pageobject(page_type='Listing', object_name='CustomObject__c')
class CustomObjectListingPage(ListingPage):
    ...
```

Using object aliases

Within a test, if you want to refer to the page object with a more human-readable name such as `Custom Object` rather than `CustomObject__c` you can do so by setting `object_name` to `Custom Object` and then defining `_object_name` in the class, as in the following example.

```
from cumulusci.robotframework.pageobjects import ListingPage, pageobject

@pageobject(page_type = 'Listing', object_name = 'My Object')
class CustomObjectListingPage(ListingPage):
    _object_name = 'MyObject__c'
    ...
```

By using an alias, you can reference the page object with either the alias or the actual object name. For example, if `object_name` is set as described above, the following two uses of `Go to page` are identical:

```
Go to page  Listing  My Object
Go to page  Listing  MyObject__c
```

Page Object Base Classes

CumulusCI provides the following base classes, which should be used for all classes that use the `pageobject` decorator. You can import these base classes from `cumulusci.robotframework.pageobjects`.

- cumulusci.robotframework.pageobjects.BasePage A generic base class used by the other pageobject classes. Use the `BasePage` class for creating custom page objects when none of the other base classes make sense.

  : - The `BasePage` adds the `Log current page object` keyword to every page object. This keyword is most useful when debugging tests. It will add information about the currently loaded page object to the log file generated when the test runs.

- `cumulusci.robotframework.pageobjects.DetailPage`: A class for a page object that represents a detail page.
- `cumulusci.robotframework.pageobjects.HomePage`: A class for a page object that represents a home page.
- `cumulusci.robotframework.pageobjects.ListingPage`: A class for a page object that represents a listing page.
- `cumulusci.robotframework.pageobject.NewModal`: A class for a page object that represents the "new object" modal.
- `cumulusci.robotframework.pageobject.ObjectManagerPage`: A class for interacting with the object manager.

Common page object attributes

When using the decorator and inheriting from one of the page object base classes, your class inherits the following attributes and properties.

- `self._object_name`: The name of the object related to the class. If the class does not define this property, it is set to the value provided as the `object_name` parameter to the `pageobject` decorator. Note: do not add the namespace prefix in the decorator. This attribute automatically adds the prefix from the `cumulusci.yml` file when necessary.
- `self.object_name`: A property that combines the `_object_name` attribute with the namespace returned by the `get namespace prefix` keyword from the CumulusCI library. If there is no namespace, this returns the value of the `_object_name` attribute.
- `self.builtin`: A reference to the Robot Framework `BuiltIn` library that you can use to directly call built-in keywords. You can call any built-in keyword by converting the name to all lowercase, and replacing all spaces with underscores (such as `self.builtin.log` and `self.builtin.get_variable_value`).
- `self.cumulusci`: A reference to the CumulusCI keyword library. You can call any keyword in this library by converting the name to all lowercase, and replacing all spaces with underscores (such as `self.cumulusci.get_org_info`).
- `self.salesforce`: A reference to the Salesforce keyword library. You can call any keyword in this library by converting the name to all lowercase, and replacing all spaces with underscores (such as `self.salesforce.wait_until_loading_is_complete`).
- `self.selenium`: A reference to SeleniumLibrary. You can call any keyword in this library by converting the name to all lowercase, and replacing all spaces with underscores (such as `self.selenim.wait_until_page_contains_element`).

Example Page Object

This example shows the definition of a page object for the listing page of custom object `MyObject__c` wherein a new custom keyword, `Click on the row with name`, is added.

```
from cumulusci.robotframework.pageobjects import pageobject, ListingPage

@pageobject(page_type="Listing", object_name="MyObject__c")
class MyObjectListingPage(ListingPage):

    def click_on_the_row_with_name(self, name):
        self.selenium.click_link('xpath://a[@title="{}"]'.format(name))
        self.salesforce.wait_until_loading_is_complete()
```

Importing the Page Object Library Into a Test

The `PageObjects` library is not only a keyword library, but also the mechanism to import files that contain page object classes. You can import these files by providing the paths to one or more Python files that implement page objects. You can also import `PageObjects` without passing any files to it to take advantage of general purpose page objects.

For example, consider a case where you create two files that each have one or more page object definitions: `PageObjects.py` and `MorePageObjects.py`, both located in the `robot/MyProject/resources` folder. You can import these page objects from these files into a test suite.

```
*** Settings ***
Library         cumulusci.robotframework.PageObjects
...   robot/MyProject/resources/PageObjects.py
...   robot/MyProject/resources/MorePageObjects.py
```

Using Page Objects

As mentioned in the previous section, you must first import the `PageObjects` library and any custom page object files you wish to use.

Next, either explicitly load the keywords for a page object, or reference a page object with one of the generic *page object keywords* provided by the `PageObjects` library.

To explicitly load the keywords for a page object, use the `Load Page Object` keyword provided by the `PageObjects` library. If successful, the `PageObjects` library will automatically import the keywords.

For example, call the `Go To Page` keyword followed by a page object reference. If the keyword (or page object reference?) navigates you to the proper page, its keywords will automatically be loaded.

Page Object Keywords

The `PageObjects` library provides these keywords.

- Current Page Should Be
- Get Page Object
- Go To Page Object
- Load Page Object
- Log Page Object Keywords
- Wait For Modal
- Wait For Page Object

Current Page Should Be

Example: `Current Page Should Be Listing Contact`

This keyword attempts to validate that the given page object represents the current page. Each page object may use its own method for making the determination, but the built-in page objects all compare the page location to an expected pattern (such as `.../lightning/o/...`). If the assertion passes, the keywords for that page object automatically load.

This keyword is useful if you get to a page via a button or some other form of navigation because it lets you assert that you are on the page you think you should be on, and load the keywords for that page, with a single statement.

Get Page Object

Example: `Get page object Listing Contact`

This keyword is most often used to get the reference to a keyword from another keyword. It is similar in function to robot's built-in Get Library Instance keyword. It is rarely used in a test.

Go To Page

Example: `Go to page Listing Contact`

This keyword attempts to go to the listing page for the Contact object, and then load the keywords for that page.

Log Page Object Keywords

Example: `Log Page Object Keywords`

This keyword is primarily used as a debugging tool. When called, it will log each of the keywords for the current page object.

Load Page Object

Example: `Load page object Listing Contact`

This keyword loads the page object for the given `page_type` and `object_name`. It is useful when you want to use keywords from a page object without first navigating to that page (for example, when you are already on the page and don't want to navigate away).

Wait for Modal

Example: `Wait for modal New Contact`

This keyword can be used to wait for a modal, such as the one that pops up when creating a new object. The keyword returns once a modal appears, and has a title of `New <object_name>` (such as "New Contact").

Wait for Page Object

Example: `Wait for page object Popup ActivityManager`

Page objects don't have to represent entire pages. You can use the `Wait for page object` keyword to wait for a page object representing a single element on a page, such as a popup window.

Generic Page Objects

You don't need to create a page object in order to take advantage of page object keywords. If you use one of the page object keywords for a page that does not have its own page object, the `PageObjects` library attempts to find a generic page.

For example, if you use `Current page should be Home Event` and there is no page object by that name, a generic `Home` page object will be loaded, and its object name will be set to `Event`.

Or let's say your project has created a custom object named `Island__c`. You don't have a home page, but the object does have a standard listing page. Without creating any page objects, this test works by using generic implementations of the `Home` and `Listing` page objects:

```
*** Test Cases ***
Example test which uses generic page objects
    ## Go to the custom object home page, which should
    ## redirect to the listing page
    Go To Page  Home  Island__c

    ## Verify that the redirect happened
    Current Page Should Be  Listing  Island__c
```

CumulusCI provides these generic page objects.

Detail

Example: `Go to page Detail Contact ${contact id}`

Detail pages refer to pages with a URL that matches the pattern `<host>/lightning/r/<object name>/<object id>/view`.

Home

Example: `Go to page Home Contact`

Home pages refer to pages with a URL that matches the pattern "<host>/lightning/o/<object name>/home"

Listing

Example: `Go to page Listing Contact`

Listing pages refer to pages with a URL that matches the pattern "<host>b/lightning/o/<object name>/list"

New

Example: `Wait for modal New Contact`

The New page object refers to the modal that pops up when creating a new object.

Of course, the real power comes when you create your own page object class that implements keywords that can be used with your custom objects.

Configuring the robot_libdoc Task

If you define a robot resource file named `MyProject.resource` and place it in the `resources` folder, you can add this configuration to the `cumulusci.yml` file to enable the `robot_libdoc` task to generate documentation.

```
tasks:
    robot_libdoc:
        description: Generates HTML documentation for the MyProject Robot Framework␣
↪Keywords
        options:
            path: robot/MyProject/resources/MyProject.resource
            output: robot/MyProject/doc/MyProject_Library.html
```

Normally this task will generate HTML output. If the output file ends with ".csv", a csv file will be generated instead.

To generate documentation for more than one keyword file or library, give a comma-separated list of files for the `path` option, or define `path` as a list under `tasks__robot_libdoc` in the `cumulusci.yml` file.

For example, generate documentation for `MyLibrary.py` and `MyLibrary.resource`.

```
tasks:
    robot_libdoc:
        description: Generates HTML documentation for the MyProject Robot Framework␣
↪Keywords
        options:
            path:
                - robot/MyProject/resources/MyProject.resource
                - robot/MyProject/resources/MyProject.py
            output: robot/MyProject/doc/MyProject_Library.html
```

You can also use basic filesystem wildcards.

For example, to document all Robot files in `robot/MyProject/resources`, configure the `path` option under `tasks__robot_libdoc` in the `cumulusci.yml` file.

```
tasks:
    robot_libdoc:
        description: Generates HTML documentation for the MyProject Robot Framework␣
↪Keywords
        options:
            path: robot/MyProject/resources/*.resource
            output: robot/MyProject/doc/MyProject_Library.html
```

Using Keywords and Tests from a Different Project

Much like you can *use tasks and flows from a different project*, you can also use keywords and tests from other projects. The keywords are brought into your repository the same way as with tasks and flows, via the `sources` configuration option in the `cumulusci.yml` file. However, keywords and tests require extra configuration before they can be used.

Note: This feature isn't for general purpose sharing of keywords between multiple projects. It was designed specifically for the case where a product is being built on top of another project and needs access to product-specific keywords.

Using Keywords

In order to use the resources from another project, you must first configure the `robot` task to use one of the sources that have been defined for the project. To do this, add a `sources` option under the `robot` task, and add to it the name of an imported source.

For exmple, if your project is built on top of NPSP, and you want to use keywords from the NPSP project, first add the NPSP repository as a source in the project's `cumulusci.yml` file:

```
sources:
    npsp:
        github: https://github.com/SalesforceFoundation/NPSP
        release: latest_beta
```

Then add `npsp` under the `sources` option for the robot task. This is because the project as a whole can use tasks or flows from multiple projects, but `robot` only needs keywords from a single project.

```
tasks:
    robot:
        options:
            sources:
                - npsp
```

When the `robot` task runs, it adds the directory that contains the code for the other repository to `PYTHONPATH`, which Robot uses when resolving references to libraries and keyword files.

Once this configuration has been saved, you can import the resources as if you were in the NPSP repository.

For example, in a project which has been configured to use NPSP as a source, the `NPSP.robot` file can be imported into a test suite.

```
*** Settings ***
Resource   robot/Cumulus/resources/NPSP.robot
```

Note: Even with proper configuration, some keywords or keyword libraries might not be usable. Be careful to avoid using files that have the exact same name in multiple repositories.

Running Tests

Running a test from another project requires prefixing the path to the test with the source name. The path needs to be relative to the root of the other repo.

For example, starting from the previous example, to run the `create_organization.robot` test suite from NPSP:

```
$ cci task run robot --suites npsp:robot/Cumulus/tests/browser/contacts_accounts/create_
↪organization.robot
```

10.13.3 Robot Tutorial

This tutorial will step you through writing your first test, then enhancing that test with a custom keyword implemented as a page object. It is not a comprehensive tutorial on using Robot Framework. For Robot Framework documentation see the Robot Framework User Guide

It is assumed you've worked through the CumulusCI *Get Started* section at least up to the point where you've called `cci project init`. It is also assumed that you've read the *Acceptance Testing with Robot Framework* section of this document, which gives an overview of CumulusCI and Robot Framework integration.

Part 1: Folder Structure

We recommend that all Robot tests, keywords, data, and log and report files live under a folder named `robot`, at the root of your repository. If you worked through the *Get Started* section, the following folders will have been created under `MyProject/robot/MyProject`:

- `doc` - a place to put documentation for your tests
- `resources` - a place to put Robot libraries and keyword files that are unique to your project
- `results` - a place for Robot to write its log and report files
- `tests` - a place for all of your tests.

Part 2: Creating a custom object

For this tutorial we're going to use a Custom Object named `MyObject` (e.g. `MyObject__c`). In addition, we need a Custom Tab that is associated with that object.

If you want to run the tests and keywords in this tutorial verbatim, you will need to go to Setup and create the following:

1. A Custom Object with the name `MyObject`.
2. A Custom Tab associated with this object.

Part 3: Creating and running your first Robot test

The first thing we want to do is create a test that verifies we can get to the listing page of the Custom Object. This will let us know that everything is configured properly.

Open up your favorite editor and create a file named `MyObject.robot` in the folder `robot/MyProject/tests`. Copy and paste the following into this file, and then save it.

```
*** Settings ***
Resource  cumulusci/robotframework/Salesforce.robot
Library   cumulusci.robotframework.PageObjects

Suite Setup     Open test browser
Suite Teardown  Delete records and close browser

*** Test Cases ***
Test the MyObject listing page
    Go to page  Listing  MyObject__c
    Current page should be  Listing  MyObject__c
```

Note: The above code uses `Go to page` and `Current page should be`, which accept a page type (`Listing`) and object name (`MyObject__c`). Even though we have yet to create that page object, the keywords will work by using a generic implementation. Later, once we've created the page object, the test will start using our implementation.

To run just this test, run the following command at the prompt:

```
$ cci task run robot -o suites robot/MyProject/tests/MyObject.robot --org dev
```

If everything is set up correctly, you should see the output that looks similar to this:

```
$ cci task run robot -o suites robot/MyProject/tests/MyObject.robot --org dev
2019-05-21 17:29:25: Getting scratch org info from Salesforce DX
2019-05-21 17:29:29: Beginning task: Robot
2019-05-21 17:29:29:        As user: test-wftmq9afc3ud@example.com
2019-05-21 17:29:29:         In org: 00Df0000003cuDx
2019-05-21 17:29:29:
==============================================================================
MyObject
==============================================================================
Test the MyObject listing page                                        | PASS |
------------------------------------------------------------------------------
MyObject                                                              | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  /Users/boakley/dev/MyProject/robot/MyProject/results/output.xml
Log:     /Users/boakley/dev/MyProject/robot/MyProject/results/log.html
Report:  /Users/boakley/dev/MyProject/robot/MyProject/results/report.html
```

Part 4: Creating a page object

Most projects are going to need to write custom keywords that are unique to that project. For example, NPSP has a keyword for filling in a batch gift entry form, EDA has a keyword with some custom logic for validating and affiliated contact, and so on.

The best way to create and organize these keywords is to place them in page object libraries. These libraries contain normal Python classes and methods which have been decorated with the `pageobjects` decorator provided by CumulusCI. By using page objects, you can write keywords that are unique to a given page, making them easier to find and easier to manage.

Defining the class

CumulusCI provides the base classes that are a good starting point for your page object (see *Page Object Base Classes*). In this case we're writing a keyword that works on the listing page, so we want our class to inherit from the `ListingPage` class.

Note: Our class also needs to use the `pageobject` decorator, so we must import that along with the `ListingPage` class.

To get started, create a new file named `MyObjectPages.py` in the folder `robot/MyProject/resources`. At the top of the new keyword file, add the following import statement:

```
from cumulusci.robotframework.pageobjects import pageobject, ListingPage
```

Next we can create the class definition by adding the following two lines:

```
@pageobject(page_type="Listing", object_name="MyObject__c")
class MyObjectListingPage(ListingPage):
```

The first line registers this class as a page object for a listing page for the object `MyObject__c`. The second line begins the class definition.

Creating the keyword

At this point, all we need to do to create the keyword is to create a method on this object. The method name should be all lowercase, with underscores instead of spaces. When called from a Robot test, the case is ignored and all spaces are converted to underscores.

In this case we want to create a method named `click_on_the_row_with_name`. All we want it to do is to find a link with the given name, click on the link, and then wait for the new page to load. To make the code more bulletproof, it will use a keyword from `SeleniumLibrary` to wait until the page contains the link before clicking on it. While probably not strictly necessary on this page, waiting for elements before interacting with them is a good habit to get into.

Add the following under the class definition:

```
def click_on_the_row_with_name(self, name):
    xpath='xpath://a[@title="{}"]'.format(name)
    self.selenium.wait_until_page_contains_element(xpath)
    self.selenium.click_link(xpath)
    self.salesforce.wait_until_loading_is_complete()
```

Notice that the above code is able to use the built-in properties `self.selenium` and `self.salesforce` to directly call keywords in the `SeleniumLibrary` and `Salesforce` keyword libraries.

Putting it all together

After adding all of the above code, our file should now look like this:

```
from cumulusci.robotframework.pageobjects import pageobject, ListingPage


@pageobject(page_type="Listing", object_name="MyObject__c")
class MyObjectListingPage(ListingPage):
    def click_on_the_row_with_name(self, name):
        xpath='xpath://a[@title="{}"]'.format(name)
        self.selenium.wait_until_page_contains_element(xpath)
        self.selenium.click_link(xpath)
        self.salesforce.wait_until_loading_is_complete()
```

We now need to import this page object into our tests. In the first iteration of the test, we imported `cumulusci.robotframework.PageObjects`, which provided our test with keywords such as `Go to page` and `Current page should be`. In addition to being the source of these keywords, it is also the way to import page object files into a test case.

To import a file with one or more page objects you need to supply the path to the page object file as an argument when importing `PageObjects`. The easiest way is to use Robot's continuation characters `...` on a separate line.

Modify the import statements at the top of `MyObject.robot` to look like the following:

```
*** Settings ***
Resource  cumulusci/robotframework/Salesforce.robot
Library   cumulusci.robotframework.PageObjects
...  robot/MyProject/resources/MyObjectPages.py
```

This will import the page object definitions into the test case, but the keywords won't be available until the page object is loaded. Page objects are loaded automatically when you call `Go to page`, or you can explicitly load them with `Load page object`. In both cases, the first argument is the page type (eg: [Listing]{.title-ref}, [Home]{.title-ref}, etc) and the second argument is the object name (eg: `MyObject__c`).

Our test is already using `Go to page`, so our keyword should already be available to us once we've gone to that page.

Part 5: Adding test data

We want to be able to test that when we click on one of our custom objects on the listing page that it will take us to the detail page for that object. To do that, our test needs some test data. While that can be very complicated in a real-world scenario, for simple tests we can use the Salesforce API to create test data when the suite first starts up.

To create the data when the suite starts, we can add a `Suite Setup` in the settings section of the test. This takes as an argument the name of a keyword. In our case we're going to create a custom keyword right in the test to add some test data for us.

It is not necessary to do it in a setup. It could be a step in an individual test case, for example. However, putting it in the `Suite Setup` guarantees it will run before any tests in the same file are run.

Open up `MyObject.robot` and add the following just before `*** Test Cases ***`:

```
*** Keywords ***
Create test data
    [Documentation]
    ...  Creates a MyObject record named "Leeroy Jenkins"
```

(continues on next page)

(continued from previous page)

```
    ...  if one doesn't exist

    ## Check to see if the record is already in the database,
    ## and return if it already exists
    ${status}  ${result}=  Run keyword and ignore error  Salesforce get  MyObject__c␣
↪Name=Leeroy Jenkins
    Return from keyword if  '${status}'=='PASS'

    ## The record didn't exist, so create it
    Log  creating MyObject object with name 'Leeroy Jenkins'  DEBUG
    Salesforce Insert  MyObject__c  Name=Leeroy Jenkins
```

We also need to modify our `Suite Setup` to call this keyword in addition to calling the `Open Test Browser` keyword. Since `Suite Setup` only accepts a single keyword, we can use the built-in keyword `Run keywords` to run more than one keyword in the setup.

Change the suite setup to look like the following, again using Robot's continuation characters to spread the code across multiple rows for readability.

Note: It is critical that you use all caps for `AND`, as that's the way Robot knows where one keyword ends and the next begins.

```
Suite Setup     Run keywords
...  Create test data
...  AND  Open test browser
```

Notice that our `Suite Teardown` calls `Delete records and close browser`. The `records` in that keyword refers to any data records created by `Salesforce Insert`. This makes it possible to both create and later clean up temporary data used for a test.

It is important to note that the suite teardown isn't guaranteed to run if you forcibly kill a running Robot test. For that reason, we added a step in `Create test data` to check for an existing record before adding it. If a previous test was interrupted and the record already exists, there's no reason to create a new record.

Part 6: Using the new keyword

We are now ready to modify our test to use our new keyword, since we now have some test data in our database, and the keyword definition in our page object file.

Once again, edit `MyObject.robot` to add the following two statements at the end of our test:

```
Click on the row with name  Leeroy Jenkins
Current page should be  Detail  MyObject__c
```

The complete test should now look like this:

```
*** Settings ***
Resource  cumulusci/robotframework/Salesforce.robot
Library   cumulusci.robotframework.PageObjects
...  robot/MyProject/resources/MyObjectPages.py

Suite Setup     Run keywords
```

(continues on next page)

(continued from previous page)

```
...  Create test data
...  AND  Open test browser
Suite Teardown  Delete records and close browser

*** Keywords ***
Create test data
    [Documentation]  Creates a MyObject record named "Leeroy Jenkins" if one doesn't␣
↪exist

    ## Check to see if the record is already in the database,
    ## and do nothing if it already exists
    ${status}  ${result}=  Run keyword and ignore error  Salesforce get  MyObject__c ␣
↪Name=Leeroy Jenkins
    Return from keyword if  '${status}'=='PASS'

    ## The record didn't exist, so create it
    Log  creating MyObject object with name 'Leeroy Jenkins'  DEBUG
    Salesforce Insert  MyObject__c  Name=Leeroy Jenkins

*** Test Cases ***
Test the MyObject listing page
    Go to page  Listing  MyObject__c
    Current page should be  Listing  MyObject__c

    Click on the row with name  Leeroy Jenkins
    Current page should be  Detail  MyObject__c
```

With everything in place, we should be able to run the test using the same command as before:

```
$ cci task run robot -o suites robot/MyProject/tests/MyObject.robot --org dev
2019-05-21 22:02:27: Getting scratch org info from Salesforce DX
2019-05-21 22:02:31: Beginning task: Robot
2019-05-21 22:02:31:        As user: test-wftmq9afc3ud@example.com
2019-05-21 22:02:31:         In org: 00Df0000003cuDx
2019-05-21 22:02:31:
==============================================================================
MyObject
==============================================================================
Test the MyObject listing page                                        | PASS |
------------------------------------------------------------------------------
MyObject                                                              | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  /Users/boakley/dev/MyProject/robot/MyProject/results/output.xml
Log:     /Users/boakley/dev/MyProject/robot/MyProject/results/log.html
Report:  /Users/boakley/dev/MyProject/robot/MyProject/results/report.html
```

10.13.4 Robot Debugger

CumulusCI includes a rudimentary Robot debugger which can be enabled by setting the `robot_debug` option of the `robot` task to `true`. When the debugger is enabled you can use the Breakpoint keyword from the Salesforce Library keyword library to pause execution.

Once the `Breakpoint` keyword is encountered you will be given a prompt from which you can interactively issue commands.

For the following examples we'll be using this simple test:

```
*** Settings ***
Resource  cumulusci/robotframework/Salesforce.robot

Suite Setup     Open test browser
Suite Teardown  Close all browsers

*** Test Cases ***
Example test case
    log  this is step one
    Breakpoint
    log  this is step two
    log  this is step three
```

Enabling the debugger

To enable the debugger you must set the `robot_debug` option to `true` for the `robot` task. **You should never do this in the project's cumulusci.yml file.** Doing so could cause tests to block when run on a CI server such as MetaCI.

Instead, you should set it from the command line when running tests locally.

For example, assuming you have the example test in a file named `example.robot`, you can enable the debugger by running the `robot` task like this:

```
$ cci task run robot --robot_debug true --suites example.robot
```

Setting breakpoints

The Salesforce keyword library includes a keyword named [Breakpoint]{.title-ref}. Normally it does nothing. However, once the debugger is enabled it will cause the test to pause. You will then be presented with a prompt where you can interactively enter commands.

```
$ cci task run robot --robot_debug true --suites example.robot
2019-10-01 15:29:01: Getting scratch org info from Salesforce DX
2019-10-01 15:29:05: Beginning task: Robot
2019-10-01 15:29:05:        As user: test-dp7to8ww6fec@example.com
2019-10-01 15:29:05:         In org: 00D0R000000ERx6
2019-10-01 15:29:05:
==============================================================================
Example
==============================================================================
Example test case                                                             .
```

(continues on next page)

(continued from previous page)

```
:::
::: Welcome to rdb, the Robot Framework debugger
:::

Type help or ? to list commands.

> Example.Example test case
-> <Keyword: cumulusci.robotframework.Salesforce.Breakpoint>
rdb>
```

Note: the `Breakpoint` keyword has no effect on a test if the `robot_debug` option is not set to `true`. While we don't encourage you to leave this keyword in your test cases, it's safe to do so as long as you only ever set the `robot_debug` option when running tests locally.

Getting Help

Whenever you see the debugger prompt `rdb>`, you can request help by typing `help` or `?` and pressing return. You will be given a list of available commands. To get help with a specific command you can type `help` followed by the command you want help on.

```
rdb> help

Documented commands (type help <topic>):
========================================
continue  locate_elements  quit            shell  vars
help      pdb              reset_elements  step   where

rdb> help vars
Print the value of all known variables
rdb>
```

Examining Variables

There are two ways you can examine the current value of a Robot variable. The simplest method is to enter the name of a variable at the prompt and press return. The debugger will show you the value of that single variable:

```
rdb> ${BROWSER}
chrome
```

To see a list of all variables and their values, enter the command `vars`.

```
rdb> vars
┌──────────────────────────────┬──────────────────────────────────────────────────────────────────────────────┐
│ Variable                     │ Value                                                                        │
├──────────────────────────────┼──────────────────────────────────────────────────────────────────────────────┤
│ ${/}                         │ /                                                                            │
├──────────────────────────────┼──────────────────────────────────────────────────────────────────────────────┤
│ ${:}                         │ :                                                                            │
├──────────────────────────────┼──────────────────────────────────────────────────────────────────────────────┤
│ ${BROWSER}                   │ chrome                                                                       │
```

(continues on next page)

(continued from previous page)

```
├──────────────────────────────────────────────────────────────────────────────
... <more output> ...
```

Executing Robot keywords

You can execute Robot keywords at the prompt by entering the command `shell` (or the shortcut `!`) followed by the keyword and arguments just as you would in a test. The following example runs the SeleniumLibrary keyword Get Location:

```
rdb> shell get location
status: PASS
result: https://ability-enterprise-4887-dev-ed.lightning.force.com/lightning/setup/
↪SetupOneHome/home
```

Notice that the `shell` command will run the keyword and then report the status of the keyword and display the return value.

Note: just like in a test, you must separate arguments from keywords by two or more spaces.

Setting Robot variables

To capture the output of a keyword into a variable, you do it the same way you would do it in a test: use a variable name, two or more spaces, then the keyword:

```
rdb> ! ${loc}  get location
status: PASS
${loc} was set to https://ability-enterprise-4887-dev-ed.lightning.force.com/lightning/
↪setup/SetupOneHome/home
rdb> ${loc}
https://ability-enterprise-4887-dev-ed.lightning.force.com/lightning/setup/SetupOneHome/
↪home
```

In addition to setting variables from the results of keywords, you can also set variables with the `shell` command using the built-in keywords Set Test Variable, Set Suite Variable, or Set Global Variable.

```
rdb> ! set test variable  ${message}  hello, world
status: PASS
result: None
rdb> ${message}
hello, world
```

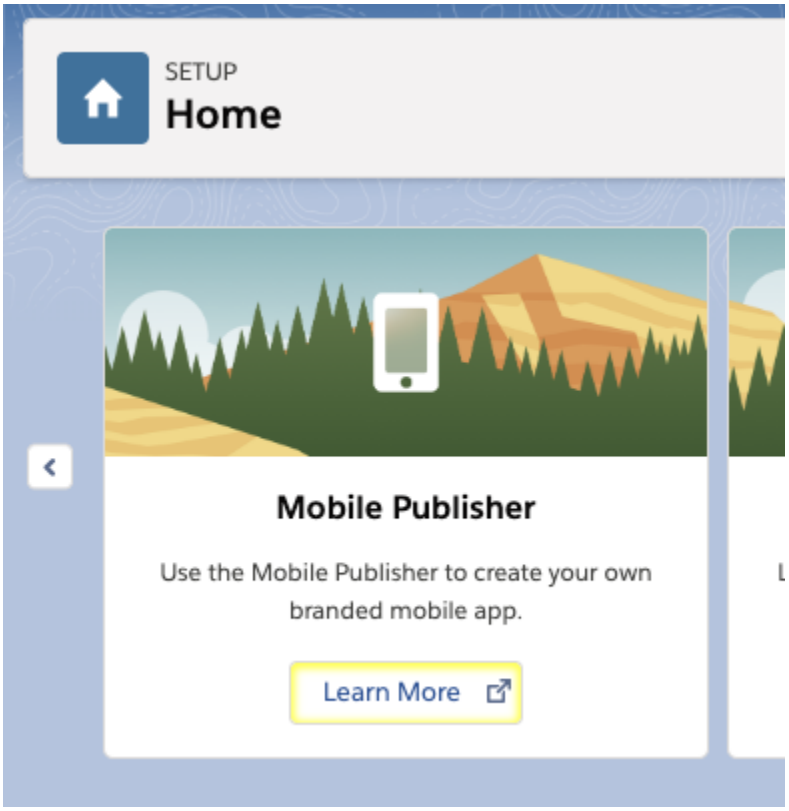
Locating elements on the web page

One of the most powerful features of the debugger is the ability to locate elements on the screen. This makes it easy to experiment with xpaths or other types of locators.

In the following example, we want to find all items on the page that contain the title "Learn More":

```
rdb> locate_elements  //button[@title='Learn More']
Found 1 matches
```

The elements will be highlighted with a yellow border:



To remove the highlighting you can use the debugger command `reset_elements`

Step through the test

The debugger allows you to step through a test one keyword at a time. From the rdb prompt, enter the command `step` to continue to the next step in the test.

```
rdb> step
.

> Example.Example test case
-> <Keyword: BuiltIn.Log  this is step two>
```

The last two lines help to give context. It is showing that you are currently right before the keyword `BuiltIn.Log this is step 2`. To get a full stack you can issue the command `where`

```
rdb> where
0: -> Example
1:   -> Example.Example test case
2:     -> BuiltIn.Log
```

Continuing or quitting the test

To let the test continue to the end, or to the next `Breakpoint` keyword, issue the command `continue`. To stop execution gracefully (ie: allow all test and suite teardowns to run), issue the command `quit`.

10.13.5 Playwright Technology Preview

Since its inception, CumulusCI has relied on Selenium to provide the foundation of our browser automation keywords.

In 2020, Microsoft introduced a new browser automation tool named Playwright. Playwright is a ground-up reinvention of a browser automation library that aims to address several shortcomings of Selenium. For example, Playwright has built-in support for waiting for elements to appear, for working with the shadow DOM, video capture of a testing session, and so on.

In 2021 the Robot Framework project introduced the Browser library which adds keywords that use the Playwright API.

Starting with CumulusCI version 3.59.0, we are providing experimental support for Playwright and the Browser library in CumulusCI.

In CumulusCI 3.60, we've reorganized our keywords so that a test can import the API and performance keywords without importing Selenium keywords. To use Playwright-based keywords, import the resource file SalesforcePlaywright.robot, which imports the non-Selenium keywords along with the keywords in the SalesforcePlaywright library.

Installation

We have not yet bundled Playwright and the Browser library with CumulusCI. However, we have provided a script to make it easy to install or uninstall Playwright and the Browser library while we continue to work on fully supporting it.

Step 1: Install Node.js

Playwright is based on Node.js. If you don't already have Node.js installed, you can find a Node.js installer for your platform on the Node.js downloads page.

Important: You must have Node.js installed before continuing with these instructions.

Step 2: Run the Playwright installation command

Installing the browser library requires a couple of manual steps, which we've automated in a single script. This script does three things:

- it verifies that Node.js has been installed
- it downloads and installs the Browser keyword library
- it downloads and installs the Node.js modules and drivers for playwright.

Note: The installation of Playwright contains drivers for all supported browsers, so there's no need to manually install drivers such as ChromeDriver. It works right out of the box!

Before you run the script, make sure your working directory is at the root of your repository. You can then run the script with the following command:

```
$ cci robot install_playwright
```

Tip: You can use the `--dry_run` (or `-n`) option to see what the command will do without actually installing.

Running an example test

As mentioned earlier, this is an experimental release of Playwright integration, so any CumulusCI keywords that rely on Selenium won't work. However, the following example shows how easy it can be to write Playwright-based tests with off-the-shelf keywords provided by the Browser library

To initialize Playwright support in a test suite, import the `SalesforcePlaywright.robot` resource file as shown in the following example. It imports the Browser library and defines the keywords `Open Test Browser` and `Delete records and close browser`.

```
*** Settings ***
Resource      cumulusci/robotframework/SalesforcePlaywright.robot

Suite Setup       Open test browser
Suite Teardown    Delete records and close browser

*** Test Cases ***

Go to user profile
    Click    button:has-text("View profile")
    Click    .profile-card-name .profile-link-label

    Wait until network is idle
    Take screenshot

Go to contacts home
    Click            button:has-text("App Launcher")
    Fill text        input[placeholder='Search apps and items...']  Contacts
    Click            one-app-launcher-menu-item:has-text("Contacts")

    Wait until network is idle
    Take screenshot
```

To run the test, save the above code in a `.robot file` (e.g. `example.robot`) and then run it with the standard robot task:

```
$ cci task run robot --suites example.robot
```

Things to Notice

This example test is unable to use any of the existing Selenium-based keywords, except for two. We've created a new library based on Playwright and the Browser library with two keywords that are similar to existing keywords: Open Test Browser and Delete Records and Close Browser

This test also uses the Browser keyword Wait until network is idle before taking a screenshot. This is a keyword that waits for there to be at least one instance of 500ms of no network traffic on the page after it starts to load. This seems to be more reliable and easier to use method than waiting for a page-specific element to appear.

This test has no explicit waits for the buttons and links that it clicks on. The underlying Playwright engine automatically waits for elements, so there should almost never be a need for keywords such as `Wait until page contains element` or `Wait until element is enabled`.

Finally, notice how easy it is to interact with both the app menu and the user profile. Playwright locators are often much easier to write than Selenium locators, which translates to tests and keywords that don't have to be tweaked when the page markup changes.

Writing keywords in JavaScript

At its core, Playwright is built on Node.js, which makes it possible to write keywords in JavaScript. This is enabled by providing the path to a JavaScript library when the Browser library is imported. CumulusCI comes with a small bootstrap file which will import a Node module from your project, which itself can import any number of other modules.

Tip: For more information about writing keywords in Javascript see Extending Browser library with a JavaScript module in the Browser library documentation.

Creating a keyword file in JavaScript

When you import `SalesforcePlaywright.robot` into a test, it will import the Browser library and pass it the name of a small bootstrap JavaScript file. This bootstrap file will import the file `robot/<ProjectName>/javascript/index.js`, exposing the exported functions as keywords.

Example

Who doesn't love a good "Hello, world" example? Start by creating the `javascript` folder in your `robot` folder. For example, if your project is named Food-Bank, you should create a folder named `robot/Food-Bank/javascript`.

In that folder, create a file named `keywords.js` with the following content:

```
// keywords.js
function hello_javascript() {
    return "Hello, JavaScript!";
}

exports.__esModule = true;
module.exports = { hello_javascript };
```

Note: The Browser documentation for writing keywords in javascript uses async functions in all of the examples, but async functions are not strictly required.

Next, create a file named `index.js` in the same folder. This file is where you can import the `keywords.js` file and any other files you need to import. For our simple example `index.js` should look like the following example.

```
// index.js
exports.__esModule = true;
module.exports = {
    ...require("keywords.js"),
};
```

Getting CumulusCI test context

CumumlusCI comes with a node module named `cumulusci` which provides access to information about the repository and org.

Let's extend our earlier example to include a keyword for getting the instance URL of the org under test.

The first step is to import the `cumulusci` module. This module returns data that looks like the following:

```
{
    "project_config": {
        "repo_name": "Food-Bank",
        "repo_root": "/projects/Food-Bank"
    },
    "org": {
        "name": "dev",
        "instance_url": "https://something-clever.my.salesforce.com",
        "org_id": "00D5C..."
    }
}
```

We can directly access these member attributes with a keyword. Edit the `keywords.js` file to look like the following example.

```
var cci = require("cumulusci");

function hello_javascript() {
    return "Hello, Javascript!";
}

function get_instance_url() {
    return cci.org.instance_url;
}

exports.__esModule = true;
module.exports = {
    hello_javascript,
    get_instance_url,
};
```

Summary

This is just a preview of things to come. The CumulusCI team will be spending more time evaluating Playwright, with an eye toward making it a viable and more robust replacement for Selenium.

Resources

- Browser Library Home Page
- Browser Library Keyword Documentation
- Playwright Home Page

CHAPTER

ELEVEN

CONTINUOUS INTEGRATION

The "CI" in CumulusCI stands for "continuous integration". Continuous integration is the practice of automatically running a project's tests for any change before merging that change to the `main` branch in the repository. Continuous integration also configures the repository so that changes are merged only if the tests have passed. This practice keeps the `main` branch in an error-free state where it can be released any time.

Teams can create bespoke automation for CumulusCI tailored to their project's needs. Once created, the automation is available to all project participants, from developers and quality engineers, to documentation writers and product managers. CumulusCI takes this reuse of automation one step further by letting it run in the context of CI systems like GitHub Actions, CircleCI, or Azure Pipelines. This consistent reuse of automation from local environments to cloud-based CI systems gives teams the ability to develop, test, and deploy their projects with confidence.

11.1 CumulusCI Flow

CumulusCI Flow is the process by which Salesforce metadata is developed, tested, and deployed to our customers. It is similar to GitHub Flow, with a few tweaks and additions.

To learn which CumulusCI flows are best designed for creating scratch orgs, running CI builds, managing the development process, and more, see *CumulusCI Flow*.

11.2 CumulusCI in GitHub Actions

GitHub Actions specify custom workflows that run directly in your GitHub repository. These workflows perform a variety of tasks, such as running test suites, performing linting checks on code, and creating code coverage reports. CumulusCI can also execute flows in GitHub Actions, making it possible to run scratch org builds and execute Apex and Robot Framework tests that leverage the custom automation defined in `cumulusci.yml`.

We offer a comprehensive framework for using *CumulusCI in GitHub Actions*.

11.3 Other CI Systems and Servers

CumulusCI runs on top of virtually any containerized CI platform. Running CumulusCI in these contexts requires configuring environment variables to provide access to orgs and services *in a headless context*.

11.4 Testing with Second-Generation Packaging

CumulusCI makes it easy to harness the power of second-generation managed packages to implement an advanced, comprehensive testing process for both first- and second-generation managed package products. This is described in *Testing with Second-Generation Packaging*.

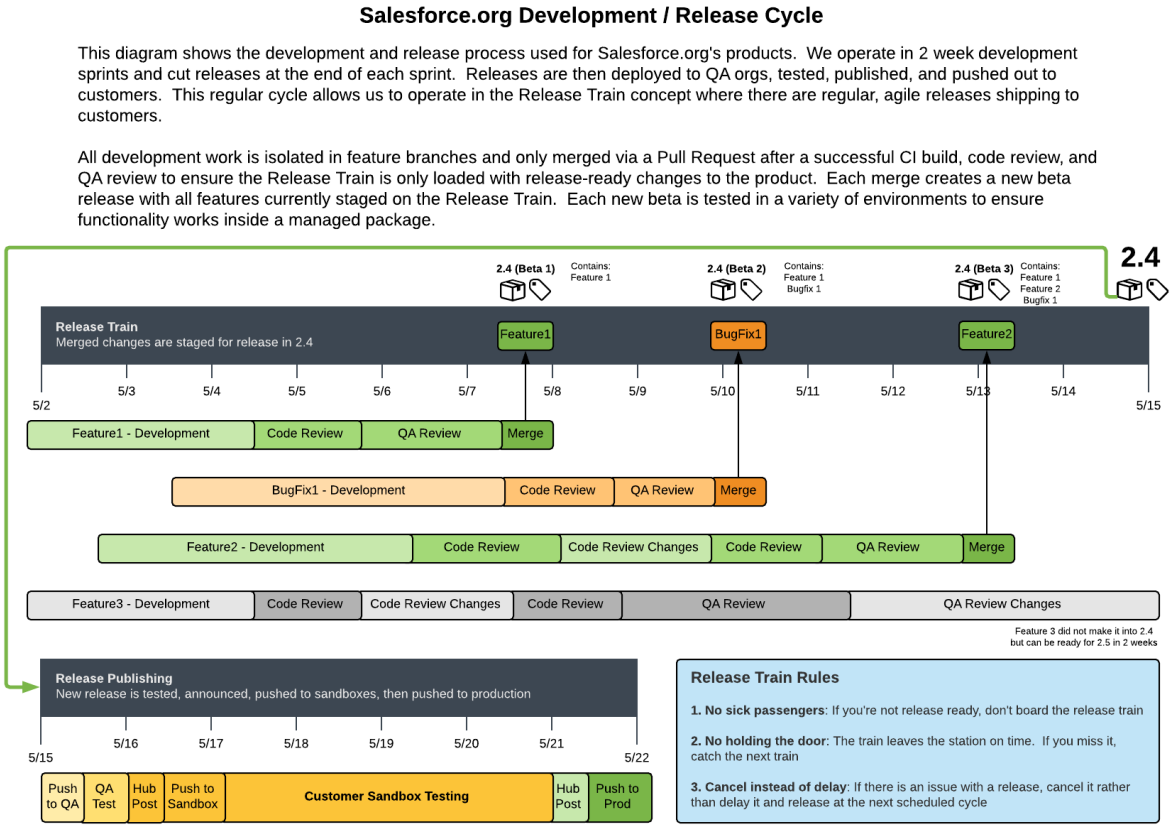
11.5 Further Reading

11.5.1 CumulusCI Flow

CumulusCI Flow is the process that Salesforce.org uses to develop, test, and release our products. This process encompasses both a development and testing philosophy as well as a specific GitHub branching structure. There are several key reasons we like using CumulusCI Flow:

- Everything is done in scratch orgs to eliminate "state drift" that occurs over time in persistent orgs. The only persistent org in this process is the packaging org or production org.
- Changes to branches that are being actively developed are thoroughly tested on each commit.
- For managed package projects, a new beta version of the package is created and tested for each commit on a project's `main` branch.
- Auto-merging functionality keeps branches up-to-date with the `main` branch.

CumulusCI Flow is implemented in the standard library flows provided by CumulusCI, but the approach to working with a GitHub repository does not require the use of CumulusCI.



Project Considerations

CumulusCI Flow was designed for use with Salesforce development projects, which inject some unique considerations into finding the right branching model:

- You cannot re-cut a Salesforce first-generation managed package release with the same version as a prior release. As a result, Git Tags are the best representation of our releases in a repository since they are a read only reference to the exact code we put into a given release.
- Releasing managed packages has some overhead involved including manual checks by release managers to ensure nothing gets permanently locked into the package in a release. As a result, true continuous delivery isn't an option. Whether you're on a team that wants to deliver quickly (e.g. a two week sprint cycle) or at team that makes several larger releases a year, CumulusCI offers functionality to help cut releases for all products with any changes.

Main Builds

The main goal of the CumulusCI Flow is to always have the main branch ready to release. This way, we can merge a fix and cut an emergency release at any time in the development process.

To test that we can package main, we upload a beta release on every commit to main and then test that beta release in a variety of Salesforce org environments concurrently. A passing build is proof we can package main at any point in time.

When the upload of the beta release is completed, the main branch is *auto-merged* into all open feature branches. New betas are published on GitHub as a GitHub Release, along with automatically generated release notes drawn from the content of the Pull Requests merged since the last production release.

CumulusCI and Main Builds

There are three main flows that facilitate main builds:

- `ci_master`: Deploys the main branch and all dependencies into the packaging org including incrementally deleting any metadata removed since the previous deployment. The end result is to prepare the packaging org to upload a new version.
- `release_beta`: Uploads a beta release of the code staged in the packaging org, creates a GitHub Tag and Release, generates release notes, adds them to the release, and merges `main` into all feature branches.
- `ci_beta`: Installs the beta and all dependencies into a fresh scratch org and runs the Apex tests.

CumulusCI and Tag Naming

CumulusCI Flow uses two different tag prefixes for differentiating between beta and production releases of a managed package. The default prefix values for tags are `beta/` and `release/`.

- Example beta release tag: `beta/1.2-Beta_3`
- Example production release tag: `release/1.2`

By differentiating beta and release tags, we allow tooling to query for the latest beta and the latest production release of each repository.

To change the default prefix values see *Branch Configuration*.

Feature Branches

Like GitHub Flow, CumulusCI Flow uses a simple main/feature branch model. The main branch is the only permanent branch in the repository. All development work (features and bug fixes) is done in feature branches prefixed with `feature/`. All commits on all feature branches are tested concurrently via continuous integration, such as MetaCI or another solution.

Once a developer is done with a feature branch, they create a Pull Request to merge their branch into the main branch. The Pull Review serves as the container for the following:

- **Code Review**: We use GitHub's built in review functionality for Pull Requests to conduct line by line code reviews
- **Release Notes**: We use the Pull Request body to create release notes content relevant to the PR. This content is automatically parsed by CumulusCI's release notes generation task to automatically build cumulative release notes on each release.

- **QA**: The goal of the Pull Request is to serve as a gate blocking a change from going into main until it's ready to release. As a result, we do QA on the feature before merging the Pull Request.

When a Pull Request is approved and passing build, it is merged using the Merge button in GitHub's web interface. We use GitHub Protected Branches to enforce both code reviews and passing builds before a Pull Request can be merged to main.

Once the Pull Request is merged, the feature branch is deleted.

Feature Branch Flows

CumulusCI facilitates working with feature branches (mainly) through two default flows:

- `dev_org`: Used to deploy the unmanaged code and all dependencies from the feature branch into a Salesforce org to create a usable development environment.
- `ci_feature`: Deploys the unmanaged code and all dependencies into a Salesforce org (typically a fresh scratch org) and run the Apex tests. This flow is run by a CI app on new commits to any feature branch.

Release Branches

Some teams deliver large releases several times a year. For this type of release cadence, Salesforce.org uses a special type of branch referred to as a release branch. Release branches are simply a feature branch named with a number. These long-lived branches are created off of the `main` branch, serve as the target branch for all features associated with that release and are eventually merged back to the `main` branch when a release occurs. To be able to clearly track what work is associated with a specific release, release branches must fulfill these criteria:

- They are the parent branches of *all* feature work associated with a release. That is, all feature branches associated with a release are child branches of the target release branch.
- Release branches use a strict naming format: `feature/release_num` where `release_num` is a valid integer.

Using the `feature/` branch prefix for the release branch names allow those branches to stay in sync with the `main` branch. Like any other feature branch, they participate in CumulusCI's parent-to-child merge operations. The release number immediately after the `feature/` prefix allows CumulusCI to detect and merge changes from one release branch to other future release branches. See *Release to (Future) Release Merges* for more information.

An example of a release branch with two items of work associated with it could look like this:

- `feature/001`
- `feature/001__feature1`
- `feature/001__feature2`

Branch Configuration

The name of the main (default) branch, as well as the branch prefixes are configurable in your projects `cumulusci.yml` file. The following shows the default values that CumulusCI comes with:

```
project:
    git:
        default_branch: main
        prefix_feature: feature/
        prefix_beta: beta/
        prefix_release: release/
```

These values can be changed to match naming conventions used by your own project.

Auto Merging

CumulusCI Flow helps to keep large diffs and merge conflicts from being the norm. CumulusCI's auto-merge functionality helps teams:

- Keep feature branches up-to-date with the `main` branch (main to feature merges)
- Manage long-lived feature branches for larger features worked on by multiple developers (parent to child merges)
- Manage large releases that occur several times a year (release to future release merges).

Main to Feature Merges

One of the bigger differences between CumulusCI Flow and GitHub Flow or git-flow is that CumulusCI Flow automates the merging of commits to a project's `main` branch into all open feature branches. This auto-merge does a lot for us:

- Ensures feature branches are in sync with the main branch.
- Re-tests each feature branch with any changes to main since the merge generates a new commit.
- Eliminates merge conflicts when merging a Pull Request to main.

To understand the benefit of auto-merging to feature branches, consider the following scenario: A developer starts work on a feature branch, puts in a few weeks on it, and then has to leave unexpectedly for a few months. While they are on leave, their feature branch gets automatically updated with any new commits on main and rebuilt. A few weeks into their leave, a new commit on main gets merged to their feature branch and breaks the build. When the developer returns after their leave, they can look at the build history to find which commit from main broke their feature branch.

Without auto-merging, the developer would return, merge main into their feature branch, and then have to sift through all the commits to main during their leave to figure out which one broke their feature branch. More testing and build history is always a good thing in addition to the other benefits we gain from auto-merging.

CumulusCI facilitates the auto-merge to feature branches via the `github_automerge_main` task, which is included by default in the `release_beta` flow. The `release_beta` flow is run, in CumulusCI Flow, on new commits to the `main` branch.

Parent to Child Merges

There is sometimes a need for multiple developers to collaborate on different parts of a single larger feature. To enable this collaboration CumulusCI expands the concept of auto-merging main-to-feature branches to also handle the concept of Parent and Child Feature Branches.

Parent/Child feature branches are created using a simple naming format:

- **Parent**: feature/parent-branch-name
- **Child**: feature/parent-branch-name__child-branch-name

A child branch extends the parent's name with two additional underscores (__) and an additional description.

Auto-merging from parent to child branches works as follows:

- Child branches never receive the auto-merge from `main`
- Parent branches do receive the merge from `main` which kicks off a Feature Test build. (This assumes the parent branch is not itself a child.)

- At the end of a successful Feature Test build on a Parent branch, the parent branch is auto-merged into all child branches

This allows us to support multiple developers working on a single large feature while keeping that feature isolated from main until we're ready to release it. The parent branch is the branch representing the overall feature. Each developer can create child branches for individual components of the larger feature. Their child branch still gets CI builds like all feature branches. When they are ready to merge from their child branch to the parent branch, they create a Pull Request which gets code reviewed by other developers working on the parent feature branch and finally merged to the parent branch.

CumulusCI facilitates parent to child auto-merges via the *github_automerge_feature* task, which is included by default in the *ci_feature* flow. If a parent feature branch passes the build, it is automatically merged into all child branches.

The parent to child merge functionality works across *multiple levels* of branching. The effects of automerging remains the same, with children only receiving merges from their parents only (e.g. no merges from grandparents) This allows us to have branching structures such as:

- `main`
- `feature/large-feature`
- `feature/large-feature__section1`
- `feature/large-feature__section1__work-item1`
- `feature/large-feature__section1__work-item2`
- `feature/large-feature__section2`
- `feature/large-feature__section2__work-item1`

In this scenario, a commit to the `main` branch triggers the `github_automerge_main` task to run and will automerge that commit into `feature/large-feature`. This triggers a build to run against `feature/large-feature`, and assuming the build passes, runs the `github_automerge_feature` task. This task detects two child branches of `feature/large-feature`: `feature/large_feature__section1` and `feature/large-feature__section2`. The task automerges the commit from the parent, into the child branches, and builds begin to run against those branches. If the build for `feature/large-feature__section1` fails, it doest not trigger `github_automerge_feature` to merge to its child branches. This means that despite `feature/large-feature__section1` having two child branches, they would not receive automerges until the parent branch tests successfully.

Release to (Future) Release Merges

Because release branches are so long-lived, and so much work goes into them, their diffs can get quite large. This means headaches are inevitable the day after a major release, and you need to pull down all of the changes from the new release into the next release branch (which has likely been in development for months already). To alleviate this pain point, CumulusCI can ensure that all release branches propagate commits they receive to other existing release branches that correspond to future releases.

Consider the following branches in a GitHub repository:

- `main` - Source of Truth for Production
- `feature/002` - The next major production release
- `feature/002__feature1` - A single feature associated with release `002`
- `feature/002__large_feature` - A large feature associated with release `002`
- `feature/002__large_feature__child1` - First chunk of work for the large feature
- `feature/002__large_feature__child2` - Second chunk of work for the large feature

- feature/003 - The release that comes after 002
- feature/003__feature1 - A single feature associated with release 003

In this scenario, CumulusCI ensures that when feature/002 receives a commit, that that commit is also merged into feature/003. This kicks off tests in our CI system and ensures that functionality going into feature/002 doesn't break work being done for future releases. Once those tests pass, the commit on feature/003 is merged to feature/003__feature1 because they adhere to the parent/child naming convention described above. Commits **never** propagate in the opposite direction. (A commit to feature/002 would never be merged to feature/001 if it was an existing branch in the GitHub repository).

Propagating commits to future release branches is turned off by default. If you would like to enable this feature for your GitHub repository, you can set the update_future_releases option on the github_automerge_feature task in your cumulusci.yml file:

```
tasks:
    github_automerge_feature:
        options:
            update_future_releases: True
```

Orphan Branches

If you have both a parent and a child branch, and the parent is deleted, this creates an orphaned branch. Orphaned branches do not receive any auto-merges from any branches. You can rename an orphaned branch to include the feature/ prefix and contain no double underscores ('__') to begin receiving merges from the main branch again.

If we have a parent and child branch: feature/myFeature and feature/myFeature__child, and feature/myFeature (the parent) is deleted, then feature/myFeature__child would be considered an orphan. Renaming feature/myFeature__child to feature/child will allow the orphan to begin receiving automerges from the main branch.

CumulusCI Flow vs. GitHub Flow

Since CumulusCI Flow is largely an extension of GitHub Flow, the differences are mostly additional processes in CumulusCI Flow that help make it more effective for large-scale Salesforce projects:

- Feature branches must be prefixed feature/ or they don't get built or receive auto-merges. This allows developers to have experimental branches that don't get built or merged.
- CumulusCI Flow is focused on an agile release process that works well with the technical constraints of Salesforce packaging..
- CumulusCI Flow requires the beta and release tag naming convention so tooling can use the GitHub API to determine the latest beta and the latest production release.
- CumulusCI Flow utilizes parent/child branch relationships and performs auto-merging of commits between branches, where as GitHub flow does not.

CumulusCI Flow vs git-flow

When our team first started figuring out our development/release process, we started where most people do in looking at git-flow. Unlike both CumulusCI Flow and GitHub Flow, git-flow uses multiple permanent branches to separate development work from releases. We decided to go with a main/feature branching model instead of git-flow for a few reasons:

- We only cut and release new releases. We never patch old releases which makes the complexity of git-flow less necessary.
- git-flow is not natively supported in git or GitHub. Using git-flow effectively usually requires extending your git tooling to enforce structure and merging rules for a more complex branching model.
- The main reason for git-flow is to be able to integrate your features together. We get this, along with many other benefits, already from auto-merging main to feature branches.
- Feature branches provide better isolation necessary for a rapid, agile release cycle by keeping all features not ready for release out of the release. Doing testing in the development branch means you've already integrated your features together. If one feature is bad, it is harder to unwind that feature from the development branch than if it were still isolated in its feature branch, tested there, and only merged when truly ready. Plus, with the auto-merge of main, we get the same integration as a development branch.
- In short, auto-merging and parent/child feature branches in CumulusCI Flow provide us everything we would want from git-flow in a simpler branching model.

11.5.2 Run CumulusCI from GitHub Actions

CumulusCI can be used to run continuous integration builds with GitHub Actions. In order to follow along, you should already have a repository that is hosted on GitHub and configured as a CumulusCI project. In other words, we're assuming your project already has a `cumulusci.yml` and that you are successfully running CumulusCI flows locally.

There is also a template repository that is setup to run *CumulusCI Flow* with our GitHub Actions framework. This repository can be used as a starting point for implementing your own project or as a reference for the following material.

Note: GitHub Actions are free for open source (public) repositories. Check with GitHub about pricing for private repositories.

Actions Framework

Our GitHub Actions framework lets you choose how much customization you wish to perform versus how much out-of-the-box functionality you'd like to use. Applications that use our predefined workflows can get started by simply copying and pasting an example file into the repository. Applications that prefer deep customization or do not use *CumulusCI Flow* can easily compose our building-block Actions to suit their specific needs.

Reusable Workflows

At the highest tier of abstraction, your application can consume reusable workflows from `cumulus-actions/standard-workflows`. These workflows offer easy onboarding and don't require any customization of the YAML to create a complete pipeline for a 1GP or 2GP project. The standard workflows include definitions of dependencies between jobs, ensuring that (for example) a Beta Test job runs after an Upload Beta job.

We recommend cloning the `cumulus-actions/standard-workflows` repository and pushing it into your org. (Do not fork the repository; GitHub Actions currently does not consume reusable workflows from a fork). This repository is a central control console for all of your applications using the shared workflows, making it easy to introduce (for example) new feature test-level jobs across all repos without individually updating those repos' configurations. We reserve the right to change `cumulus-actions/standard-workflows` at any time without notice and without tagging a new version to support this control-console use case; this is the reason we encourage cloning the repository to support your own standard workflow usage.

We provide the following standard workflows:

Packaging Workflows

- `beta-1gp`, expected to run on the main branch, executes Release Beta (1GP) followed by Beta Test.
- `production-1gp`, expected to run on the main branch, executes Release Production (1GP) followed by Release Test.
- `beta-2gp`, expected to run on the main branch, executes Release Beta (2GP) followed by Beta Test.
- `production-2gp`, expected to run on the main branch, executes Release Production (2GP) followed by Release Test.

Feature Testing Workflows

Because there is wide variance in the types of tests run in feature-level testing, we provide a number of Feature Test workflows that combine different jobs. All of these workflows are expected to run on `feature/**` branches. Note that references to 2GP below are part of the *Testing with Second-Generation Packaging* process, and can be used on both first- and second-generation package projects.

- `feature` runs unmanaged feature-test builds: Feature Test.
- `feature-with-robot` adds a Robot Framework test job to `feature`.
- `feature-2gp` runs unmanaged and 2GP feature-test package builds: Feature Test, Build Feature Test Package, Feature Test - 2GP.
- `feature-2gp-with-robot` adds a 2GP Robot Framework test job to `feature-2gp`.
- `feature-namespaced` runs unmanaged and namespaced-org tests: Feature Test and Feature Test - Namespaced.

To consume a standard workflow, follow these steps.

1. Clone the `cumulus-actions/standard-workflows` repo. Don't use a fork!
2. Select one or more reusable workflows you want to consume.
3. For each workflow, create a file in your repository under `.github/workflows`. Design your own triggers, or copy from the examples below.
4. Commit the file and merge a Pull Request to your main branch.

Example 1: Feature Test workflow

This workflow runs on feature branches before merging.

```yaml
name: Feature Test
on:
    push:
        branches:
            # Customize as needed for your project's branching structure.
            - feature/**
            - main
jobs:
    feature-test:
        name: "Feature Test"
        # Replace this URL with your forked repo.
        # Use the feature test workflow that matches your project needs.
        uses: cumulus-actions/standard-workflows/.github/workflows/feature-2gp.yml@main
        secrets:
            dev-hub-auth-url: "${{ secrets.DEV_HUB_AUTH_URL }}"
```

Example 2: Upload Beta workflow

This workflow runs on every main-branch commit to upload a beta package release.

```yaml
name: Upload Beta Release
on:
    push:
        branches:
            - "main"
jobs:
    upload-beta:
        # Replace this URL with your forked repo.
        # Use the `beta-2gp` workflow if this is a 2GP project.
        uses: SFDO-Community/standard-workflows/.github/workflows/beta-1gp.yml@main
        secrets:
            # If this is a 2GP project, omit the packaging org auth URL.
            packaging-org-auth-url: "${{ secrets.PACKAGING_ORG_AUTH_URL }}"
            dev-hub-auth-url: "${{ secrets.DEV_HUB_AUTH_URL }}"
```

Example 3: Upload Release workflow

```yaml
name: Upload Production Release
on:
    workflow_dispatch:
jobs:
    upload-production-release:
        name: "Upload Production Release"
        # Replace this URL with your forked repo.
        # Use the `beta-2gp` workflow if this is a 2GP project.
        uses: SFDO-Community/standard-workflows/.github/workflows/production-1gp.yml@main
        secrets:
            # If this is a 2GP project, omit the packaging org auth URL.
            packaging-org-auth-url: "${{ secrets.PACKAGING_ORG_AUTH_URL }}"
            dev-hub-auth-url: "${{ secrets.DEV_HUB_AUTH_URL }}"
```

You can choose whether or not to pin to a version tag in your cloned `standard-workflows` repo. In many use cases, it makes the most sense *not* to do so, as we do here by referencing `@main`. This allows you to use your `standard-workflows` repo as a central control console to instantly update workflow definitions across repos. For

example, when prerelease scratch orgs are made available in advance of each major Salesforce release, you can add a `feature-test-prerelease` job to your standard feature workflows - enabling it for all of your projects without individually updating their repos.

Job-Based Actions

If you prefer to construct your own workflows, you can consume our job-based Actions as building blocks. Each job-based Action will set up the CumulusCI and SFDX CLIs, authorize a Dev Hub and/or packaging org, check out the project, and run one or more Cumulus flows and tasks.

- `cumulus-actions/release-beta-1gp` runs the flows `ci_master` and `release_beta`.
- `cumulus-actions/release-beta-2gp` runs the flow `release_beta_2gp`.
- `cumulus-actions/beta-test` runs the flow `ci_beta`.
- `cumulus-actions/release-production-1gp` runs the flows `ci_master` and `release_production`.
- `cumulus-actions/release-production-2gp` runs the flow `release_production_2gp`.
- `cumulus-actions/release-test` runs the flow `ci_release`.
- `cumulus-actions/build-unlocked-test-package` runs the flow `build_unlocked_test_package` and populates the package id onto a commit status.
- `cumulus-actions/build-feature-test-package` runs the flow `build_feature_test_package` and populates the package id onto a commit status.
- `cumulus-actions/feature-test` runs the flow `ci_feature`.
- `cumulus-actions/feature-test-2gp` runs the flow `ci_feature_2gp`.
- `cumulus-actions/feature-test-namespaced` runs the flow `ci_feature`.
- `cumulus-actions/feature-test-robot` runs the flow `qa_org`, followed by the `robot` task.
- `cumulus-actions/feature-test-robot-2gp` runs the flow `qa_org_2gp`, followed by the `robot` task.

For examples of how to apply these actions, review the standard workflows in `cumulus-actions/standard-workflows`. Each Action accepts version parameters (see Version Pinning). Each Action that creates a scratch org accepts the `dev-hub-auth-url`parameter (required) and the`org-name`parameter (defaulted). Each Action that interacts with a packaging org (1GP only) accepts the`packaging-org-auth-url` parameter.

Scratch Org Base Actions

All job-based Actions that create scratch orgs consume the `cumulus-actions/run-flow-scratch` or `cumulus-actions/run-robot-flow-scratch` scratch org base Actions. You can use these base Actions to create your own job-based Actions for your custom Cumulus flows.

`cumulus-actions/run-flow-scratch` checks out your code, creates a scratch org on a given configuration (`org-name`), runs a given flow (`flow-name`) on that org, and then disposes the org. Optionally, it extracts information from the job log that matches a regex (`commit-status-regex`) and stores that information as the description on a new commit status (`commit-status-name`), prefixed by `commit-status-description-prefix`. This optional feature is used to support jobs that create per-commit package versions. The action also accepts version-pinning options (see Tool Version Pinning).

`cumulus-actions/run-robot-flow-scratch` accepts the parameters `dev-hub-auth-url`, `org-name`, and `setup-flow`. The latter two options are required. The Action will check out your code, create an org from the configuration `org-name`, and execute `setup-flow` against it before running Robot Framework tests. It stores Robot results as an artifact. The action also accepts version-pinning options (see Tool Version Pinning).

Primitive Actions

Finally, we also provide a suite of primitive Actions that underlie the base scratch org Actions and all persistent-org Actions. You can use these primitive Actions to construct your own SDLC automation, even if it's very different from CumulusCI Flow. Compose primitive Actions to create workflow automation that delivers to persistent orgs, authenticates using the JWT flow, or sits within your existing automation structures.

`cumulus-actions/default-package-versions` accepts as input user-supplied `cumulusci-version` and `sfdx-version` values. If these values are non-empty, they are returned; otherwise, default values are returned. The outputs have the same names as the inputs.

`cumulus-actions/setup-cumulus` installs CumulusCI and SFDX. It accepts optional `cumulusci-version` and `sfdx-version` inputs and uses `cumulus-actions/default-package-versions@main` to populate them if not supplied (see below under Tool Version Pinning for discussion). The action uses the built-in caching functionality of GitHub Actions to cache installed NPM and Python packages, using your supplied versions as the cache key.

`cumulus-actions/authorize-org` accepts an `auth-url` and `org-name`, and ingests that org authorization into the SFDX and CumulusCI keychains. If the optional `dev-hub` input is set to `true`, it assigns this org as the default Dev Hub.

`cumulus-actions/run-flow` executes a given flow (`flow-name`) against a given org (`org-name`). The org must be available in the keychain, or be a scratch org configuration, and CumulusCI and SFDX must already be set up.

`cumulus-actions/run-task` executes a given task (`task-name`) against a given org (`org-name`). The org must be available in the keychain, or be a scratch org configuration, and CumulusCI and SFDX must already be set up.

Tool Version Pinning

All non-primitive Actions that run Cumulus operations accept two optional parameters, `cumulusci-version` and `sfdx-version`. If these inputs are supplied, the Actions will ensure that the specified versions of the tools are installed.

If the inputs are not populated, the Action will source a default version for each tool from `cumulus-actions/default-package-versions@main`. Note that we do not pin a tag on `default-package-versions`: the default CumulusCI and SFDX versions may be changed without publishing a new Action tag, provided that the overall behavior of the Action is not altered. We use this flexibility to ensure that we pin stable versions, and reserve the right to roll back the default in case of unexpected regressions.

The Cumulus Suite Actions **require CumulusCI 3.61.1 or greater** for any operation that references a packaging org.

Environment Setup and Org Authorization

All Actions that interact with persistent orgs (such as a packaging org or Dev Hub) authorize those orgs using SFDX Auth URLs. These URLs are obtained via by first authorizing an org to the CLI:

```
sfdx auth:web:login -a packaging
```

and then retrieving the auth URL from the JSON output of the command

```
sfdx force:org:display --json --verbose
```

under the key `sfdxAuthUrl` under `result`.

If you have `jq` installed, you can do `sfdx force:org:display -u packaging-gh --json --verbose | jq -r .result.sfdxAuthUrl`.

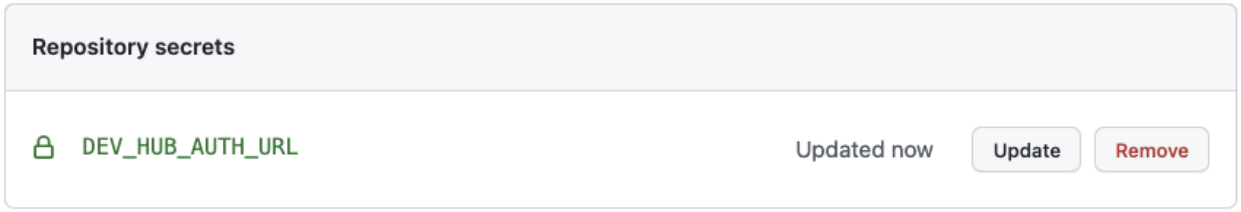
First-generation package projects will have two auth-URL secrets, for the packaging org and for the Dev Hub. Second-generation and Unlocked package projects will have at least one auth-URL secret, for the Dev Hub, and may have two to provide distinct access levels (see below).

Auth URLs should always be stored in GitHub Secrets. There are a number of different ways to organize secret storage. While we offer recommended best practices here, you should always follow the security guidance of your own teams.

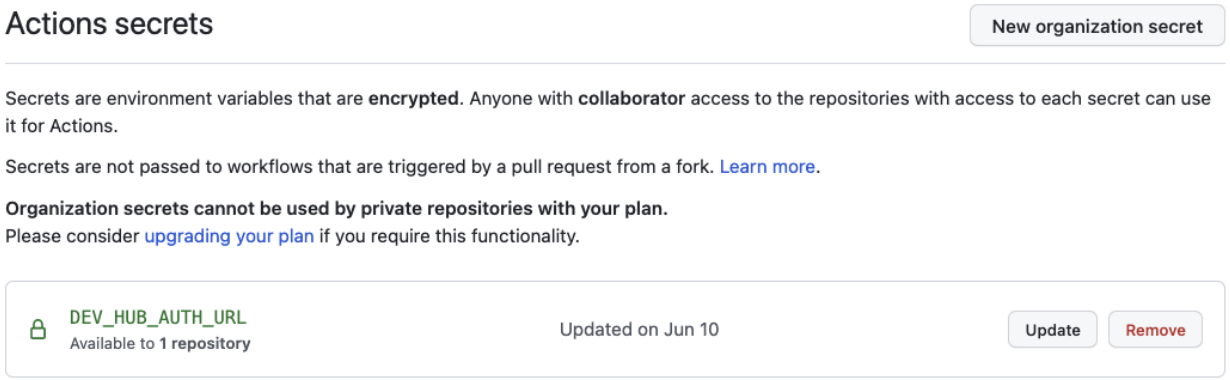
Dev Hub Auth URLs for Scratch Org Builds

Every Action that creates a scratch org requires a `dev-hub-auth-url` parameter containing an Auth URL. We recommend storing this URL in a GitHub Secret called `DEV_HUB_AUTH_URL`, a convention used in our reusable workflows.

If you're working on a single repository, you can store your Dev Hub auth URL in a repository secret. Locate the Settings->Secrets->Actions section of your GitHub repo, and click "New repository secret". Add your auth URL and save the secret.



If you're working in a GitHub organization with multiple repositories, and you want to share the same user access to the Dev Hub across repositories, you can store your auth URL as an organization secret. Locate the Organization->Settings->Secrets->Actions section, and click "New organization secret". Add your auth URL and save the secret.



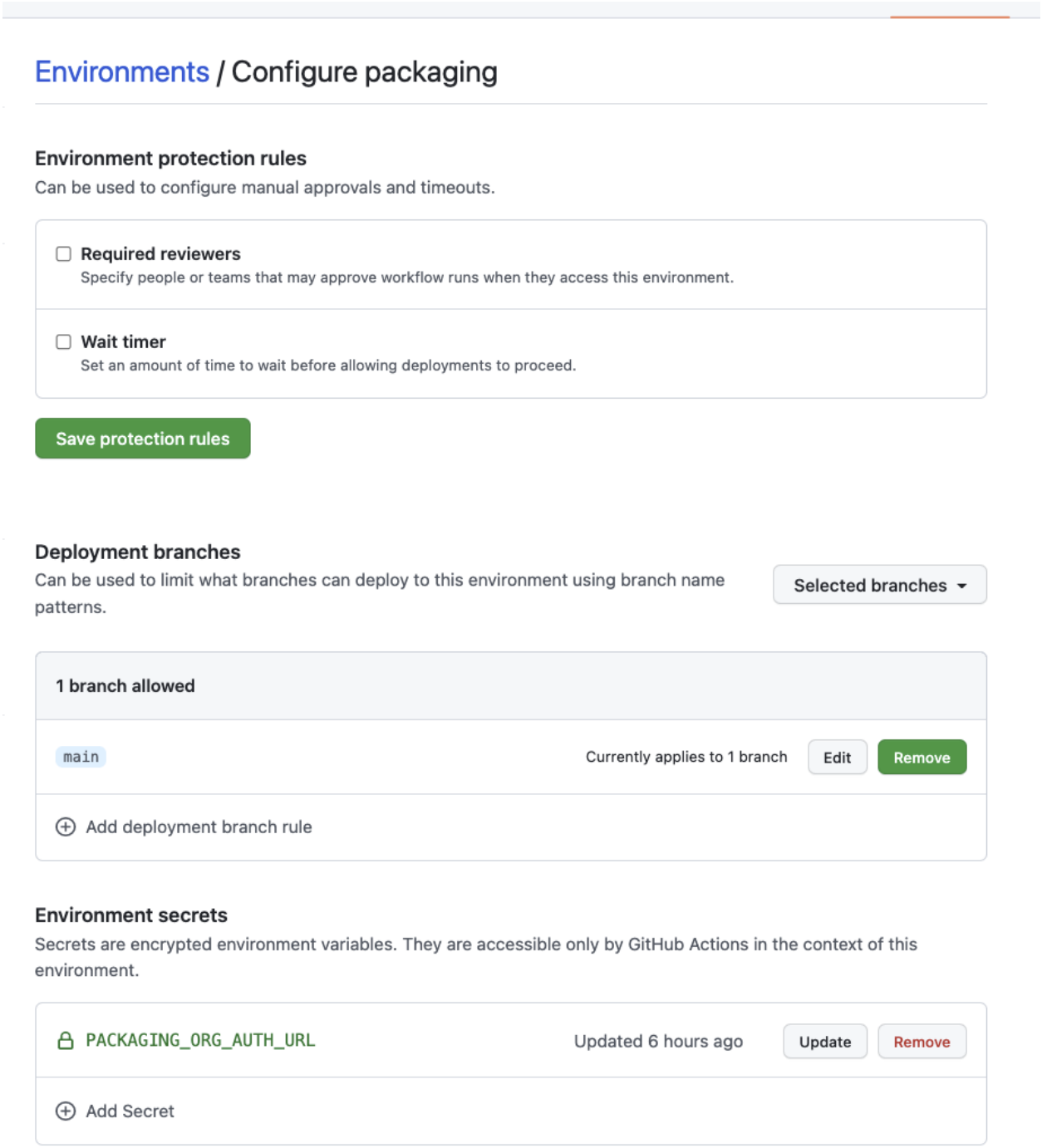
You can select which repositories to expose the secret to. Using an organization secret makes rotating this credential easy.

It's important to use a credential with minimal privileges, as using this secret in feature-level builds that are executed before code review may open routes for unscrupulous users to access the credential. We recommend permissioning a Dev Hub user with the Limited Access - Free License to avoid exposing any other information or functionality to that user.

Packaging Org and Dev Hub Auth URLs for Package Uploads

Packaging org credentials, and Dev Hub credentials that have permissions to upload and promote package versions, are high risk. It's critical to ensure that these credentials are protected and cannot be exfiltrated by malicious changes to your workflow definitions.

We recommend creating a GitHub Environment called `packaging` and adding the auth URL as an environment secret there. This means that the secret will *not* be available to Actions that run against environments other than packaging.



Our standard workflows for creating package versions are pre-configured to use an environment named `packaging`. If you build your own workflows, or use an environment name other than `packaging`, include the declaration

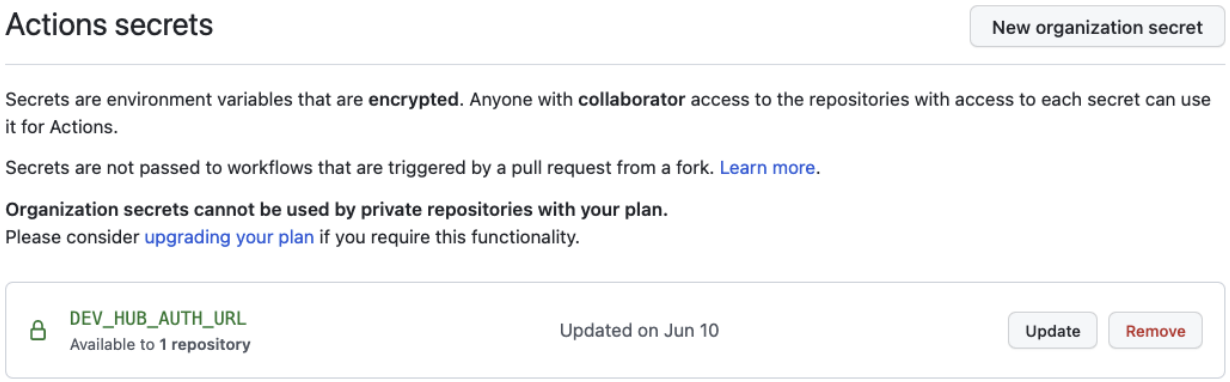
```
environment: packaging # or other name
```

in your workflows that need to access this secret.

We recommend adding the following configurations to avoid disclosure of the packaging org secret:

1. Configure package-upload builds to run only on `main` (using workflow triggers).
2. Configure the `packaging` environment to be available only to builds on `main`.
3. Configure Code Owners to identify one or more trusted staff members as owners of your `.github` directory.
4. Configure Branch Protection for `main` to require code review and Code Owners review prior to merging Pull Requests. This ensures that rogue Actions cannot disclose this sensitive credential without being reviewed by your trusted staff members.

If you are building a second-generation package, you can still use this pattern to protect the higher-privileged Dev Hub credentials required to upload and promote a package. Set up a second Dev Hub user with the appropriate permissions, and store the Dev Hub auth URL for that user in a `packaging` environment.



Environments also allow you to secure your workflow for promoting a production release.

You can choose to set up required reviewers on your `packaging` environment. Your approvers will then need to sign off on all beta and production releases.

If you'd like beta builds to run automatically on commits to `main` *without* individual review, while still requiring review on production releases, you can set up two environments: `packaging` and `packaging-prod`. Configure required reviewers on `packaging-prod`, but not on `packaging`, and set the auth URL secret for your packaging org on both environments.

If you use a trigger, such as modification to a specific file protected by Code Owners, to initiate production package builds rather than a manually-initiated workflow, you may not need to use required reviewers on your `packaging` environment at all.

Concurrency Protection

All Actions other than package uploads are safe to run in parallel, because they run against independent scratch orgs. First-generation package uploads, however, are generally serialized.

In GitHub Actions, concurrency protection takes place at the level of the workflow, not the individual Action. We recommend following the pattern shown in `cumulus-actions/standard-workflows` by ensuring that package-upload Actions are run in a job with the key `concurrency` set to `packaging`. This ensures that GitHub Actions serializes execution of jobs against the packaging org. Note that this concurrency setup is independent of your decisions around environment setup.

Deploying to a Production Org

We've chosen not to provide standard workflows or actions for deploying to a production org. If you're operating in an org development model, you're likely to have specific requirements about how production deployments take place.

In many cases, you'll want to use the `deploy_unmanaged` flow or the `deploy` task. `deploy_unmanaged` deploys metadata, but also unschedules Scheduled Apex and uninstalls previously-deployed components that have been removed from the source repository. If you do not want incremental component removal or Apex unscheduling, use the `deploy` task.

You may also wish to compose custom flows that perform other configuration, deployments, or setup operations on your production org.

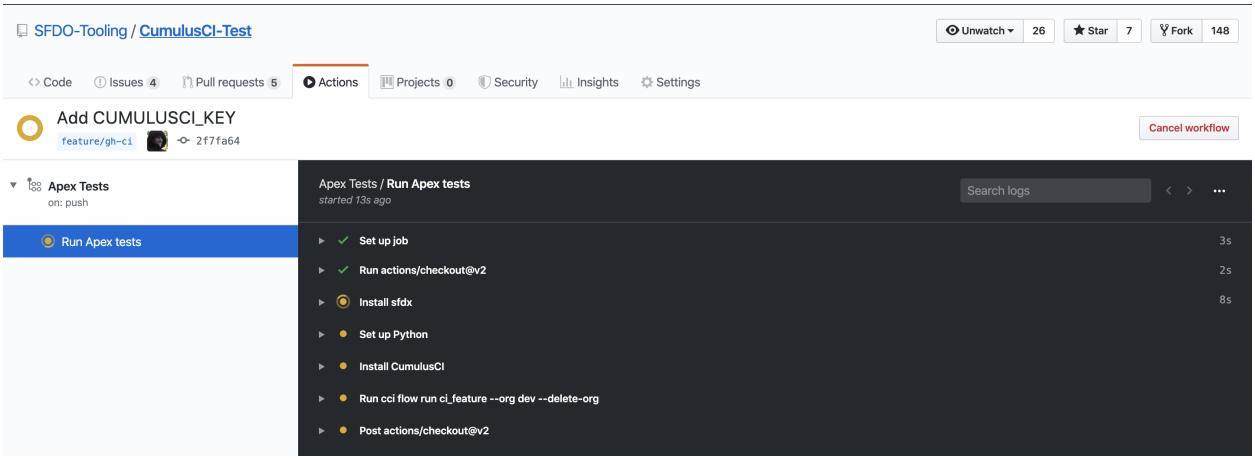
Our `release-production-1gp` Action is an example of running Cumulus automation against a production org (in that case, the packaging org). We recommend using this Action as a model for your own deployment Actions that target Production or long-lived sandbox orgs.

Building from Scratch

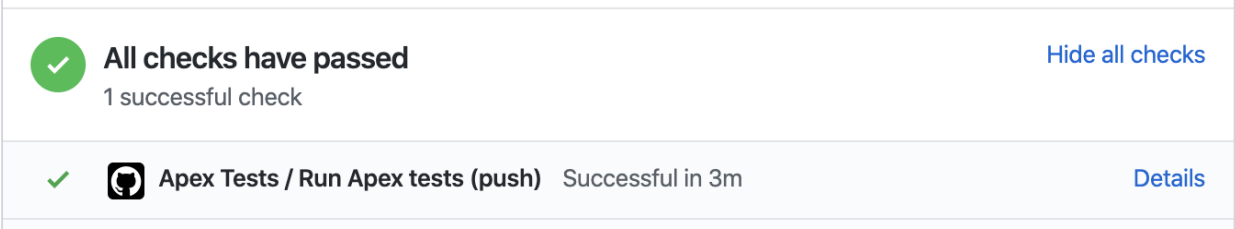
If you prefer to build your own GitHub Actions workflows for scratch, or need to extend the capabilities in our workflow framework, you can learn more about *how CumulusCI consumes services and authenticated orgs* and how to *Register Environment Services*.

If you need to use a different type of authentication, such as the JWT Flow, instead of SFDX auth URLs, you'll need to compose your own workflows from our supplied primitives.

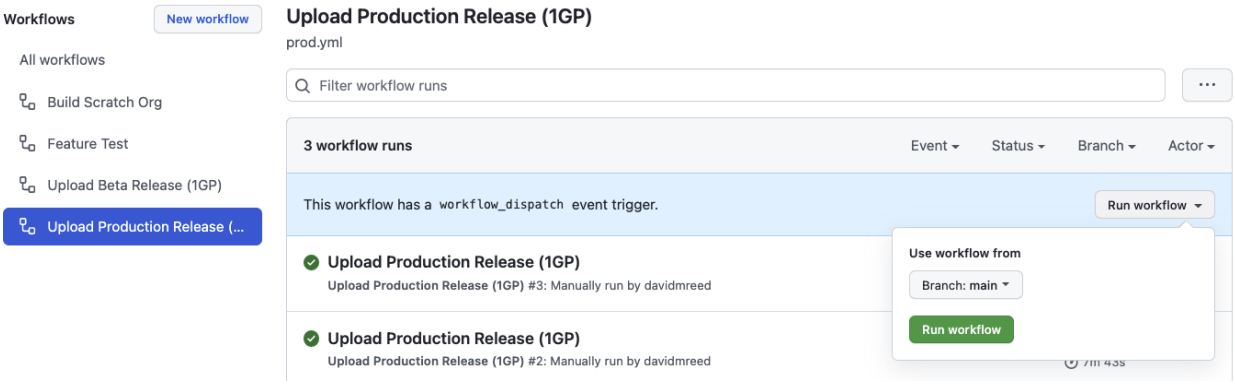
Running and Reviewing Actions Workflows



If you open a pull request for a branch that triggers a run of your Actions workflow, you will find a section at the bottom of the pull request that shows the results of the checks that were performed by the workflow:



Some Actions are manually triggered, including our `release-production-1gp` and `release-production-2gp` Actions, which upload a production release of a package. To run an Action manually, navigate to the Actions tab, then select the workflow by name from the list. Click the **Run workflow** dropdown, choose the appropriate branch, and click the **Run workflow** button to start the run.



You can set any Action as a required status on a *protected branch*, such as your `main` branch. This means that changes can only be merged to that branch if these Actions complete successfully.

See GitHub's documentation for instructions to configure protected branches and enable required status checks.

References

- GitHub Actions Documentation

11.5.3 Testing with Second-Generation Packaging

CumulusCI makes it easy to harness the power of second-generation managed packages to implement an advanced, comprehensive testing process for *both* first- and second-generation managed package products.

This process yields multiple benefits:

- You can test managed packages *as* managed packages, but before merging code.
- You gain the ability to perform end-to-end testing across applications that span multiple packages earlier in the development lifecycle.
- For existing 1GP products, it also allows for the creation of a full-scale 2GP testing and development framework *before* migrating products from 1GP to 2GP. Migration, when generally available, will be much easier because products are already being tested as 2GPs.

Salesforce.org is actively using this process for feature-level testing and end-to-end testing of dozens of existing first-generation packages, while preparing for the migration into second-generation packaging. This process is also applicable for testing products that started as second-generation packages.

Building 2GP Beta Packages in Continuous Integration

Any managed package product - first or second generation - can use CumulusCI automation to build and test 2GP beta packages. The out-of-the-box flow `build_feature_test_package` can be run on any commit. This flow builds a 2GP beta package using an alternate package name (which defaults to `<project name> Managed Feature Test`, reflecting its intended role in supporting feature-branch testing) but with the same namespace as the main package.

The 2GP test package is also built using the `Skip Validation` option, which defers validation of the package until install time. Skipping validation ensures that feature test packages build extremely quickly, and also avoids locking in dependency versions - making it easy to achieve complex end-to-end testing workflows, as described in *End-to-End Testing with Second-Generation Packages*.

CumulusCI stores data about feature test packages in GitHub commit-status messages. When the `build_feature_test_package` flow completes successfully, the `04t` id of the created package version is stored in the "Build Feature Test Package" commit status on GitHub. Testing and 2GP build flows can acquire the package version from this store.

2GP Tests for Feature Branches

The `ci_feature_2gp` flow parallels `ci_feature`, which is used for unmanaged feature testing in continuous integration, but uses a 2GP feature test package instead of deploying the project unmanaged.

When executed on a specific commit, the flow acquires a 2GP feature test package id from the "Build Feature Test Package" commit status on that commit. It installs that package, then executes Apex unit tests.

Note: The `ci_feature_2gp` flow is intended for use after the `build_feature_test_package` flow. On MetaCI, this is implemented by using a Commit Status trigger to run `ci_feature_2gp`; on other CI systems, a `ci_feature_2gp` build may be made dependent on a `build_feature_test_package` build.

Running 2GP tests in CI can replace the use of namespaced scratch orgs for most automated testing objectives. 2GP testing orgs provide a more accurate representation of how namespaces are applied and how metadata will behave once packaged, making it possible to catch packaging-related issues *before* code is merged to the main branch or deployed to a 1GP packaging org.

Note: Component coverage for first- and second-generation packages is very similar, but some projects may use components with differing behaviors. Consult the Metadata Coverage Report with any questions.

Manual QA can be executed on feature branches via the flow `qa_org_2gp`, which operates just like `ci_feature_2gp` but which also executes `config_qa` to prepare an org for manual testing. Similarly, Robot tests may be executed against 2GP orgs by running `qa_org_2gp` instead of `qa_org` before invoking `robot`.

End-to-End Testing with Second-Generation Packages

The `qa_org_2gp` flow allows for performing manual and automated end-to-end tests of multi-package products sooner in the development lifecycle then was previously possible. Take the following example:

- Product B has a dependency on Product A.
- Product B is developing a new feature that is dependent on a new feature being developed for Product A.

Without the ability to test with 2GP packages, end-to-end testing on Product A and B's linked features could only occur once both products have moved significantly forward in the development lifecycle:

- Both A and B merge their feature work into their main branch in a source control system.

- New feature metadata is uploaded to the packaging org, if the products are 1GPs.
- New beta versions for both Product A and B are created
- In many cases, a production release for Product A must also be created to satisfy B's dependency, if the packages are 1GPs.

Once all of the steps above have occurred, end-to-end testing with new managed package versions can take place. However, if *any* errors are found at this point the entire process has to start over again, and first-generation packages may have already incurred component lock-in. With 2GP testing, this is no longer the case.

Instead, a tester may execute the `qa_org_2gp` flow from a feature branch in the repository of Product B. The following will occur:

1. CumulusCI resolves dependencies as they are defined Product B's `cumulusci.yml` file, using the `commit_status` resolution strategy. CumulusCI matches the current branch and release against branches in the upstream dependencies to locate the most relevant 2GP packages for this testing process. See *Controlling GitHub Dependency Resolution* for more details.
2. CumulusCI installs suitable 2GP feature test packages for Product A and any other dependencies, if found, or falls back to 1GP packages if not found.
3. CumulusCI installs a Project B 2GP feature test package, sourced from a GitHub commit status on the current commit. (The commit must have been pushed, and `build_feature_test_package` must have run successfully).
4. Finally, CumulusCI executes the `config_qa` flow to prepare the org for use in testing.

This allows for full end-to-end testing of features that have inter-package-dependencies prior to the merging of code to any long-lived branches (e.g. a release branch or `main`). Because CumulusCI defaults to building packages using `Skip Validation`, any suitable 2GP feature test package installed for Project A may satisfy the dependency, making it possible to test feature development without committing to package version numbers or specific dependency versions.

The process, backed by second-generation packaging, maximizes the utility of feature-level testing processes for both first- and second-generation packages, while helping prepare first-generation packages to migrate to 2GP once migration becomes generally available.

11.5.4 Run CumulusCI Headlessly

CumulusCI can be used to run continuous integration builds in your CI system.

This section outlines how to setup services and orgs that can be defined in a particular environment such that they will be recognized by CumulusCI.

If you're working in GitHub Actions, you can take advantage of our *GitHub Actions framework* to streamline your setup process. You'll need the information in this document only if you wish to use an alternate authorization strategy or configure additional services.

Register Environment Services

It is often the case that services you use for local development will differ from the services that you want to use in your build system. For example, developers will setup a GitHub service locally that is associated with their GitHub User, while an integration build may want to run as an integration user when interacting with GitHub. By providing environment variables with specific prefixes, CumulusCI can detect and register those services for use when running tasks and flows.

Name Environment Services

Environment variables that define CumulusCI services adhere to the following format:

```
CUMULUSCI_SERVICE_<service_type>[__service_name]
```

All services should start with the prefix `CUMULUSCI_SERVICE_` followed immediately by the `service_type` (for a full list of available services run `cci service list`). Additionally, you have the option to provide a unique name for your service by adding a double underscore (`__`) followed by the name you wish to use. If a name is specified it is prepended with "env-" to help establish that this service is coming from the environment. If a name is not specified, a defualt name of `env` is used for that service.

Here are some examples of environment variable names along with their corresponding service types and names:

- `CUMULUSCI_SERVICE_github` –> A `github` service that will have the default name of `env`
- `CUMULUSCI_SERVICE_github__integration-user` –> A `github` service that will have the name `env-integration-user`
- `CUMULUSCI_SERVICE_github_enterprise__enter-user` –> A `github_enterprise` service that will have the name `env-enter-user`
- `CUMULUSCI_SERVICE_connected_app` –> A `connected_app` service with the default name of `env`
- `CUMULUSCI_SERVICE_connected_app__sandbox` –> A `connected_app` service with the name `env-sandbox`

By always prepending `env` to the names of services specified by environment variables, it is easy to see which services are currently set by environment variables and which are not.

Environment Service Config Values

The value of the environment variables (i.e. everything that comes after the = character) are provided in the form of a JSON string. The following shows an example that defines a github service via an environment variable:

```
CUMULUSCI_SERVICE_github='{"username": "jdoe", "email": "jane.doe@some.biz", "token": "
↪<personal_access_token>"}'
```

These values provide CumulusCI with the required attributes for a particular service. The easiest way to find what attributes are needed for a particular service is to look for your service under the services tag in the CumulusCI standard library and provide values for all "attributes" listed under the desired service. You can also use `cci service info` to get the values from a service you've configured locally.

For example, if you're looking to register a `connected_app` service, then the attributes: `callback_url`, `client_id`, and `client_secret` would need to be provided in the following format:

```
"{"callback_url": "<callback_url>", "client_id": "<client_id>", "client_secret": "
↪<client_secret>"}"
```

> The values `<callback_url>`, `<client_id>`, and `<client_secret>` should all be replaced with actual values.

GitHub Enterprise Server considerations

The use of GitHub environment variables is supported when connecting to a repository a GitHub Enterprise Server.

The `GitHubEnterprise` client is selected automatically based on the repository configuration in the `cumulusci.yml` file. When the `repo_url` is NOT GitHub.com an attempt will be made to connect to the base domain in `repo_url`. The GitHub environemnt variables are then used to authorize.

If your use case requires dependencies on a different server, for example, upstream dependencies reside on GitHub.com but your project repository is stored on a GitHub Enterprise Server do not use *`github-token`*, instead configure services for GitHub.com and your GitHub Enterprise Server similarly to this example:

```
CUMULUSCI_SERVICE_github='{"username": "jdoe", "email": "jane.doe@some.biz", "token": "
↪<personal_access_token>"}'
CUMULUSCI_SERVICE_github_enterprise`='{"username": "jdoe", "email": "jane.doe@some.biz",
↪"server_domain":"git.some.biz", "token": "<enterprise_personal_access_token>"}'
```

When connecting to repositories on GitHub.com or GitHub Enterprise Server, the service configuration for authorization is selected automatically by matching the `server_domain` field to the repository domain. When a repository domain is `github.com` the `github` service will be used.

Register Persistent Orgs

SFDX Auth URL Authorization

CumulusCI can import persistent org authorization from the SFDX CLI. This capability allows you to store the SFDX auth URL for an org in your CI system's secret storage, import it into the SFDX keychain, and then use it directly from CumulusCI.

To do so, follow these steps.

1. Retrieve your auth URL.
2. Authorize the org using `sfdx auth:sfdxurl:store`.
3. Run `cci org import <org name> <org name>`.

JWT Flow Authorization

Using JWT authorization provides you with the greatest control over access to your persistent org. This approach uses a Connected App and digital certificate, which you control and can rotate as desired.

First, you need a Connected App that is configured with a certificate in the "Use digital signatures" setting in its OAuth settings. You can follow the Salesforce DX Developer Guide to get this set up:

- Create a private key and self-signed certificate
- Create a Connected App
- In addition to the OAuth scopes mentioned in that article, CumulusCI also requires the `full` scope.

Once you have a Connected App created, you can configure CumulusCI to use this Connected App to login to a persistent org by setting the following environment variables.

- `CUMULUSCI_ORG_orgName`
- `SFDX_CLIENT_ID`
- `SFDX_HUB_KEY`

See the below entries for the values to use with each.

Important: Setting the above environment variables negates the need to use the `cci org connect` command. You can simply run a `cci` command and pass the `--org orgName` option, where `orgName` corresponds to the name used in the `CUMULUSCI_ORG_*` environment variable.

In the context of GitHub Actions, all of these environment variables would be declared under the `env` section of a workflow. Below is an example of what this would look like:

```
env:
    CUMULUSCI_ORG_sandbox:
        {
            "username": "just.in@salesforce.org",
            "instance_url": "https://sfdo--sbxname.my.salesforce.com",
        }
    SFDX_CLIENT_ID: { { $secrets.client_id } }
    SFDX_HUB_KEY: { { $secrets.server_key } }
```

The above assumes that you have `client_id` and `server_key` setup in your GitHub encrypted secrets

Note that the value of the `server_key` environment variable is the content of the file `server.key`, which was created as part of your Connected App setup. The key is in PEM format.

CUMULUSCI_ORG_orgName

The name of this environment variable defines what name to use for the value of the `--org` option. For example, a value of `CUMULUSCI_ORG_mySandbox` would mean you use `--org mySandbox` to use this org in a `cci` command.

Set this variable equal to the following json string:

```
{
    "username": "USERNAME",
    "instance_url": "INSTANCE_URL"
}
```

- `USERNAME` - The username of the user who will login to the target org.
- `INSTANCE_URL` - The instance URL for the org. Should begin with the `https://` schema.

You can see an example of setting this environment variable in a GitHub actions workflow in our demo repository.

Wizard Note

If the target org's instance URL is instanceless (i.e. does not contain a segment like cs46 identifying the instance), then for sandboxes it is also necessary to set `SFDX_AUDIENCE_URL` to `https://test.salesforce.com`". This instructs CumulusCI to set the correct `aud` value in the JWT (which is normally determined from the instance URL).

SFDX_CLIENT_ID

Set this to your Connected App's client id. This, combined with the `SFDX_HUB_KEY` variable instructs CumulusCI to authenticate to the org using the JWT Bearer Flow instead of the Web Server Flow.

SFDX_HUB_KEY

Set this to the private key associated with your Connected App (this is the contents of your `server.key` file). This combined with the `SFDX_CLIENT_ID` variable instructs CumulusCI to authenticate to the org using the JWT Bearer Flow instead of the Web Server Flow.

Multiple Services of the Same Type

In rare cases a build may need to utilize multiple services of the same type. To set a specific service as the default for subsequent tasks/flows run the `cci service default <service_type> <name>` command. You can run this command again to set a new default service to be used for the given service type.

CHAPTER

TWELVE

RELEASE MANAGED AND UNLOCKED PACKAGES

CumulusCI makes it easy to build first- and second-generation managed packages, as well as unlocked packages. While the overall workflows for all types of packages are quite similar, there are key differences between them. The chapters below explore all three packaging options.

New projects should, absent any other considerations, generally choose to use second-generation managed packages or unlocked packages. Learn more about second-generation packaging and its advantages in the Salesforce DX Developer Guide.

12.1 Release a First-Generation Managed Package

This section outlines how to release first-generation (1GP) Salesforce managed package projects. Salesforce.org's Release Engineering team practices *CumulusCI Flow*, which incorporates all of these steps.

12.1.1 Prerequisites

This section assumes:

- *CumulusCI is installed* on your computer.
- A Salesforce managed package project has been *configured* for use with CumulusCI.
- A packaging org *is connected* to CumulusCI under the name of `packaging`.

To verify this setup and display information about the connected packaging org:

```
$ cci org info packaging
```

Note: The packaging org can be listed under an alias. For a complete list of orgs connected to CumulusCI, run `cci org list`.

If your project has been configured for use with CumulusCI, `cci org info` lists the project's namespace under `package__namespace` in the output.

Create a Managed Package Project

If you haven't created a managed package project, follow these steps:

- Create a Developer Edition Org. (Sign up for one here.)
- Create a managed package.
- Assign a namespace.
- Configure the namespace in CumulusCI.

12.1.2 Deploy to a Packaging Org

CumulusCI deploys metadata to a `packaging` org with the `ci_master` flow.

> **Warning:** The `ci_master` flow runs the *uninstall_packaged_incremental* task, which deletes any metadata from the package in the target org that's not in the repository.

```
$ cci flow run ci_master --org packaging
```

The `ci_master` flow executes these tasks in the target org.

- Updates any project dependencies
- Deploys any unpackaged metadata located in the `pre` directory
- Deploys packaged metadata
- Deploys destructive changes to remove metadata in the target org that is no longer in the local repository
- Runs the `config_packaging` flow, which by default consists only of the *update_admin_profile* task.

> **Tip:** To list each step in the `ci_master` flow, run `cci flow info ci_master`.

CumulusCI separates uploading metadata to the packaging org and releasing a beta version of the package into the `ci_master` and `release_beta` flows, respectively. This separation offers discretion to run additional checks against the org, if necessary, between deploy and release steps.

12.1.3 Create a Beta Version

The `release_beta` flow groups the common tasks that must be executed for the release of a new beta version of a project.

```
$ cci flow run release_beta --org packaging
```

This flow *always* runs against the project's `packaging` org, where it:

- Uploads a new beta version of the managed package.
- Creates a new GitHub release tag for the new beta version. Extension packages that also use CumulusCI require this release tag to find the latest version when this repository is listed as a dependency.
- *Generates Release Notes*.

- Syncs feature branches with the `main` branch, which automatically integrates the latest changes from `main`. For more information see *Auto Merging*.

Important: This flow assumes that the package contents were already deployed using the `ci_master` flow. It does *not* include a step to deploy them.

To create a new beta version for your project without the bells and whistles, use the `upload_beta` task:

```
$ cci task run upload_beta --org packaging --name package_version
```

12.1.4 Test a Beta Version

The `ci_beta` flow installs the latest beta version of the project in a scratch org, and runs Apex tests against it.

```
$ cci flow run ci_beta --org beta
```

This flow is intended to be run whenever a beta release is created.

12.1.5 Upload and Test a Final Version

To upload a production release of your managed package project:

```
$ cci flow run release_production --org packaging
```

Similar to `release_beta`, this task uploads a new production version of your package, creates a release tag in GitHub, and aggregates release notes for the new version.

Important: This flow assumes that the package contents have previously been deployed using the `ci_master` flow.

To upload the new production version without creating the GitHub tag and generating release notes:

```
$ cci task run upload_production --name v1.2.1
```

To test the new package version:

```
$ cci flow run ci_release --org release
```

The `ci_release` flow installs the latest production release version, and runs the Apex tests from the managed package on a scratch org.

12.2 Release a Second-Generation Managed Package

This section outlines how to release second-generation (2GP) Salesforce managed package projects. Salesforce.org's Release Engineering team practices *CumulusCI Flow*, which incorporates all of these steps.

12.2.1 Prerequisites

This section assumes:

- *CumulusCI is installed* on your computer.
- A Salesforce managed package project has been *configured* for use with CumulusCI.
- Your Dev Hub has the required features enabled: Enable DevHub Features in Your Org and Enable Unlocked and Second-Generation Managed Packaging.
- A namespace org has been created and linked to the active Dev Hub.

12.2.2 Create a Beta Version

CumulusCI uses the `dependencies` section of your `cumulusci.yml` file to define your 2GP project's dependencies. CumulusCI uses GitHub releases to identify the ancestor id and new version number for the beta package version. By default, the new beta version will increment the minor version number from the most recent GitHub release.

Because Salesforce requires package version Ids (`04txxxxxxxxxxx`) for 2GP package dependencies, dependencies with 1GP releases created *before CumulusCI 3.34.0* must be installed in an org to make those Ids available. If your project has such dependencies, start by running

```
$ cci flow run dependencies --org dev
```

If you are using CumulusCI 3.43.0 or later, your project uses dependencies specified as a `version_id`, 2GP dependencies, or dependencies whose releases were created by CumulusCI 3.34.0 or later, you do not need to execute this step. Current versions of CumulusCI automatically store and consume the package version Id in GitHub releases.

When you're ready, and your org is prepared, to upload a package version, run the command

```
$ cci flow run release_2gp_beta --org dev
```

Important: The org supplied to `release_2gp_beta` has two purposes. One is to look up the Ids of dependency packages (see above). The other is to provide the configuration for the *build org* used to upload the 2GP package version. CumulusCI will use the scratch org definition file used to create the specified org (`dev` here) to create the build org, which defines the features and settings available during package upload.

You may wish to define a separate scratch org configuration just for package uploads to ensure only your required features are present.

The `release_2gp_beta` flow executes these tasks:

- Uploads a new beta version of the managed package.
- Creates a new GitHub release tag for the new beta version. Extension packages that also use CumulusCI require this release tag to find the latest version when this repository is listed as a dependency.
- *Generates Release Notes*.
- Syncs feature branches with the `main` branch, which automatically integrates the latest changes from `main`. For more information see *Auto Merging*.

Tip: To list each step in the `release_2gp_beta` flow, run `cci flow info release_2gp_beta`.

Customizing Package Uploads

2GP package uploads are performed by the `create_package_version` task. If the built-in configuration used by `release_2gp_beta` does not suit the needs of your project - for example, if you want to increment version numbers differently, or build a package with the Skip Validation option - you can customize the options for that task in `release_2gp_beta` or invoke the task directly.

To learn more about the available options, run

```
$ cci task info create_package_version
```

Handling Unpackaged Metadata

CumulusCI projects can include *unpackaged metadata* in directories like `unpackaged/pre` and `unpackaged/post`. These directories are deployed when CumulusCI creates a scratch org, and are installed in the packaging org when CumulusCI creates 1GP package versions.However, second-generation packaging does not have a packaging org, and does not allow interactive access to the build org.

CumulusCI offers two modes of handling unpackaged metadata owned by dependencies when building a second-generation package.

The default behavior is to ignore unpackaged metadata. If unpackaged metadata is intended to satisfy install-time dependencies of packages, this requires that those dependencies be met in other ways, such as by configuring the scratch org definition. For examples of how to satisfy the install-time dependencies for NPSP and EDA without using unpackaged metadata, see *Extending NPSP and EDA with Second-Generation Packaging*.

The other option is to have CumulusCI automatically create unlocked packages containing unpackaged metadata from dependency projects. For example, if your project depended on the repository `Food-Bank`, which contained the unpackaged metadata directories

- `unpackaged/pre/record_types`
- `unpackaged/pre/setup`

CumulusCI would automatically, while uploading a version of your package, upload unlocked package versions containing the current content of those unpackaged directories.

The unlocked package route is generally suitable for testing only, where it may be convenient when working with complex legacy projects that include lots of unpackaged metadata. However, it's generally *not* suitable for use when building production packages, because your packages would have to be distributed along with those unlocked packages. For this reason, this behavior is off by default. If you would like to use it, configure your `cumulusci.yml` to set the option `create_unlocked_dependency_packages` on the `create_package_version` task.

12.2.3 Test a Beta Version

The `ci_beta` flow installs the latest beta version of the project in a scratch org, and runs Apex tests against it.

```
$ cci flow run ci_beta --org beta
```

This flow is intended to be run whenever a beta release is created.

12.2.4 Promote a Production Version

To be installed in a production org, an 2GP package version must be promoted to mark it as released.

To promote a production release of your managed package project:

```
$ cci flow run release_2gp_production --org packaging
```

Unlike first-generation packages, promoting a second-generation package doesn't upload a new version. Instead, it promotes the most recent beta version (found in the project's GitHub releases) to production status. Then, CumulusCI creates a new, production GitHub release, and aggregates release notes for that release.

You can also promote a package using its `04t` package Id, without using the GitHub release operations:

```
$ cci task run promote_package_version --version_id 04t000000000000 --promote_
↪dependencies True
```

Alternatively, you can use the `sfdx force:package:version:promote` command to promote a 2GP package. Note that using this command will also not perform any release operations in GitHub.

Promote Dependencies

If additional unlocked packages were created to hold unpackaged dependencies, they must be promoted as well. To promote dependencies automatically use `--promote_dependencies True` with the `promote_package_version` task, or customize the `release_2gp_production` flow to include that option.

```
$ cci task run promote_package_version --version_id 04t000000000000 --promote_
↪dependencies True
```

Test a Production Version ————————-

To test the new package version:

```
$ cci flow run ci_release --org release
```

The `ci_release` flow installs the latest production release version and runs the Apex tests from the managed package on a scratch org.

12.3 Release an Unlocked Package

While CumulusCI was originally created to develop managed packages, it can also be used to develop and release unlocked packages.

12.3.1 Prerequisites

This section assumes:

- *CumulusCI is installed* on your computer.
- A Salesforce managed package project has been *configured* for use with CumulusCI.
- Your Dev Hub has the required features enabled: Enable DevHub Features in Your Org and Enable Unlocked and Second-Generation Managed Packaging.

- If you're building a namespaced unlocked package, a namespace org has been created and linked to the active Dev Hub.

12.3.2 Create a Beta Version

CumulusCI uses the `dependencies` section of your `cumulusci.yml` file to define your 2GP project's dependencies. CumulusCI uses GitHub releases to identify the ancestor Id and new version number for the beta package version. By default, the new beta version will increment the minor version number from the most recent GitHub release.

Because Salesforce requires package version Ids (`04txxxxxxxxxxx`) for 2GP package dependencies, some CumulusCI dependencies must be installed in an org to make those Ids available. If your project has dependencies that are not specified as a `version_id`, start by running

```
$ cci flow run dependencies --org dev
```

When you're ready, and your org is prepared, to upload a package version, run the command

```
$ cci flow run release_unlocked_beta --org dev
```

Important: The org supplied to `release_unlocked_beta` has two purposes. One is to look up the Ids of dependency packages (see above). The other is to provide the configuration for the *build org* used to upload the 2GP package version. CumulusCI will use the scratch org definition file used to create the specified org (`dev` here) to create the build org, which defines the features and settings available during package upload.

You may wish to define a separate scratch org configuration (`build`) just for package uploads to ensure only your required features are present.

The `release_unlocked_beta` flow executes these tasks:

- Uploads a new beta version of the unlocked package.
- Creates a new GitHub release tag for the new beta version. Extension packages that also use CumulusCI require this release tag to find the latest version when this repository is listed as a dependency.
- *Generates Release Notes*.
- Syncs feature branches with the `main` branch, which automatically integrates the latest changes from `main`. For more information see *Auto Merging*.

Tip: To list each step in the `release_unlocked_beta` flow, run `cci flow info release_unlocked_beta`.

Customizing Package Uploads

2GP package uploads are performed by the `create_package_version` task. If the built-in configuration used by `release_unlocked_beta` does not suit the needs of your project - for example, if you want to increment version numbers differently, or build an org-dependent package - you can customize the options for that task in `release_unlocked_beta` or invoke the task directly.

To learn more about the available options, run

```
$ cci task info create_package_version
```

CumulusCI can also create org-dependent and skip-validation packages when configured with the appropriate options.

Handling Unpackaged Metadata

CumulusCI projects can include *unpackaged metadata* in directories like `unpackaged/pre` and `unpackaged/post`. These directories are deployed when CumulusCI creates a scratch org, and are installed in the packaging org when CumulusCI creates 1GP package versions.However, second-generation packaging does not have a packaging org, and does not allow interactive access to the build org.

CumulusCI offers two modes of handling unpackaged metadata owned by dependencies when building a second-generation package.

The default behavior is to ignore unpackaged metadata. If unpackaged metadata is intended to satisfy install-time dependencies of packages, this requires that those dependencies be met in other ways, such as by configuring the scratch org definition. For examples of how to satisfy the install-time dependencies for NPSP and EDA without using unpackaged metadata, see *Extend NPSP and EDA with Second-Generation Packaging*.

The other option is to have CumulusCI automatically create unlocked packages containing unpackaged metadata from dependency projects. For example, if your project depended on the repository `Food-Bank`, which contained the unpackaged metadata directories

- `unpackaged/pre/record_types`
- `unpackaged/pre/setup`

CumulusCI would automatically, while uploading a version of your package, upload unlocked package versions containing the current content of those unpackaged directories.

The unlocked package route is generally suitable for testing only, where it may be convenient when working with complex legacy projects that include lots of unpackaged metadata. However, it's generally *not* suitable for use when building production packages, because your packages would have to be distributed along with those unlocked packages. For this reason, this behavior is off by default. If you would like to use it, configure your `cumulusci.yml` to set the option `create_unlocked_dependency_packages` on the `create_package_version` task.

12.3.3 Test a Beta Version

The `ci_beta` flow installs the latest beta version of the project in a scratch org, and runs Apex tests against it.

```
$ cci flow run ci_beta --org beta
```

This flow is intended to be run whenever a beta release is created.

12.3.4 Promote a Production Version

To be installed in a production org, an 2GP package version must be promoted to mark it as released.

To promote a production release of your managed package project:

```
$ cci flow run release_unlocked_production --org packaging
```

Unlike first-generation packages, promoting a second-generation package doesn't upload a new version. Instead, it promotes the most recent beta version (found in the project's GitHub releases) to production status. Then, CumulusCI creates a new, production GitHub release, and aggregates release notes for that release.

You can also promote a package using its `04t` package Id, without using the GitHub release operations:

```
$ cci task run promote_package_version --version_id 04t000000000000 --promote_
↪dependencies True
```

Alternatively, you can use the `sfdx force:package:version:promote` command to promote a 2GP package.

Promote Dependencies

If additional unlocked packages were created to hold unpackaged dependencies, they must be promoted as well. To promote dependencies automatically use `--promote_dependencies True` with the `promote_package_version` task, or customize the `release_unlocked_production` flow to include that option.

```
$ cci task run promote_package_version --version_id 04t000000000000 --promote_
↪dependencies True
```

Test a Production Version ————————-

To test the new package version:

```
$ cci flow run ci_release --org release
```

The `ci_release` flow installs the latest production release version and runs the Apex tests from the managed package on a scratch org.

12.4 Extend NPSP and EDA with Second-Generation Packaging

Building packages that extend (depend on) NPSP and EDA using second-generation packaging involves some unique complications. Both NPSP and EDA have install-time dependencies on Record Type org features: an Account Record Type must exist in order for the packages to install.

When building first-generation packages, both NPSP and EDA serve this need by installing unpackaged Record Types stored in their `unpackaged/pre` folders. However, second-generation packaging doesn't allow interactive access to its build orgs, where package versions are created.

We recommend that projects which extend EDA or NPSP with a second-generation package use scratch org definition files to satisfy these Record Type dependencies. You can do this by creating a new org definition file in `orgs`, based on your existing org definitions, and adding to it an `objectSettings` section. We'll call this file `orgs/build.json`.

For NPSP, use:

```
"settings": {
    /* Your project's settings are here */
},
"objectSettings": {
    "account": {
        "defaultRecordType": "default"
    }
}
```

and for EDA, use:

```
"settings": {
    /* Your project's settings are here */
},
"objectSettings": {
    "account": {
        "defaultRecordType": "Administrative"
    }
}
```

This satisfies EDA's requirement for a specific Record Type name.

You'll also add to your `cumulusci.yml`:

```
orgs:
    scratch:
        build:
            config_file: orgs/build.json
```

Then, run your 2GP builds against the org `build`:

```
$ cci flow run dependencies --org build
$ cci flow run release_2gp_beta --org build
```

This will result in the creation of default Record Types in the build org, allowing NPSP to be installed. Meanwhile, your other scratch orgs will continue to use the NPSP or EDA default Record Types, installed by CumulusCI's dependency-management system and reflecting the configuration of the subscriber orgs into which your package will ultimately be installed.

12.5 Generate Release Notes

12.5.1 Using Automatic Release Note Generation

The `github_release_notes` task fetches the text from Pull Requests that were merged between two given tags. The task then searches for specific titles (Critical Changes, Changes, Issues Closed, New Metadata, Installation Info, and so on) in the Pull Request bodies, and aggregates the text together under those titles in the GitHub tag description.

`github_release_notes` is automatically run during CumulusCI's built-in release flows.

To see what the release notes look like without publishing them to GitHub:

```
$ cci task run github_release_notes --tag release/1.2
```

Note: The `--tag` option indicates which release's change notes are aggregated. The previous command aggregates all change notes between the [1.2]{.title-ref} release and the [1.1]{.title-ref} release.

To see where each line in the release notes comes from, use the `--link_pr True` option.

```
$ cci task run github_release_notes --tag release/1.2 --link_pr True
```

To publish the release notes to a release tag in GitHub, use the `--publish True` option:

```
$ cci task run github_release_notes --tag release/1.2 --publish True
```

To use additional headings, add new ones (as parsers) under the `project__git__release_notes` section of the `cumulusci.yml` file.

```
release_notes:
    parsers:
        7: class_path: cumulusci.tasks.release_notes.parser.GithubLinesParser
```

Note: The new parser is listed with the number 7 because the first six are the default parsers that come with CumulusCI.

12.5.2 Using Static Release Notes

In some cases, you may wish to use static text as your release notes instead of aggregating them from Pull Request content. You can do that by making customizations in your `cumulusci.yml`.

Customize the `github_release` task to include your static content:

```
tasks:
    github_release:
        options:
            release_content: |
                # This is my top-level heading

                Here is some body content.
```

Note that you may use Markdown, provided you indent the content as shown and utilize the `|` indicator, which marks a YAML block.

Then, turn off the `github_release_notes` task, where you don't want it to run:

```
flows:
    release_beta:
        steps:
            3:
                task: None
```

You can repeat this configuration for any or all of the flows `release_beta`, `release_production`, `release_2gp_beta`, `release_2gp_production`, `release_unlocked_beta`, and `release_unlocked_production`. You may choose to customize only the production flows if you want to use auto-generated release notes for your beta releases.

12.6 Manage Push Upgrades

If your packaging org (for first-generation packages) or Dev Hub (for second-generation packages) is enabled to use push upgrades, CumulusCI can schedule push upgrades with the `push_sandbox` and `push_all` tasks.

> **Warning:** `push_all` schedules push upgrades to *all* customers' production and sandbox orgs. Please confirm that this action is desired before executing the task.

```
$ cci task run push_all --version <version> --org packaging
```

Replace `<version>` with the version of the managed package to be pushed.

By default, push upgrades are scheduled to run immediately.

To schedule the push upgrades to occur at a specific time, use the `--start_time` option with a time value in UTC.

```
$ cci task run push_all --version <version> --start_time 2020-10-19T10:00 --org packaging
```

There are additional tasks related to push upgrades in the CumulusCI standard library.

- *push_failure_report*: Produces a `csv` report of the failed and otherwise anomalous push jobs.
- *push_list*: Schedules a push upgrade of a package version to all orgs listed in a specified file.

- *push_qa*: Schedules a push upgrade of a package version to all orgs listed in `push/orgs_qa.txt`.
- *push_sandbox*: Schedules a push upgrade of a package version to all subscribers' sandboxes.
- *push_trial*: Schedules a push upgrade of a package version to Trialforce Template orgs listed in `push/orgs_trial.txt`.

CHAPTER

THIRTEEN

MANAGE UNPACKAGED CONFIGURATION

Not everything that's part of an application can be part of a package.

CumulusCI implements the Product Delivery Model by offering support for complex applications – applications that may include multiple managed packages as well as unpackaged metadata, and setup automation that configures org settings or makes precise changes to existing configuration.

The tools used to implement that support are *unpackaged metadata* and *Metadata ETL*.

Unpackaged metadata refers to metadata that is not delivered as part of a package, and can include both support metadata delivered to users as well as metadata that operationally configures orgs used by the product.

Metadata ETL is a suite of tasks that supports surgically altering existing metadata in an org. It's a powerful technique that alters the unpackaged configuration in an org without risking damage to existing customizations by overwriting them with incoming metadata. Metadata ETL is relevant for delivering applications to customers safely, and is often a superior alternative to unpackaged metadata.

To learn more, see `metadata-etl`.

13.1 Roles of Unpackaged Metadata

13.1.1 `unpackaged/pre`: Prepare an Org

Some projects require that unpackaged metadata be deployed to finish the customization of an org *before* the package's own code and metadata are deployed.

For example, the Nonprofit Success Pack (NPSP) must deploy unpackaged Record Types prior to installing its own packages. `unpackaged/pre` is the location designed for such metadata, which is stored in subdirectories such as `unpackaged/pre/first`.

CumulusCI's standard flows that build orgs, such as `dev_org` and `install_prod`, always deploy metadata bundles found in `unpackaged/pre` before proceeding to the deployment of the application. It's also easy to include `unpackaged/pre` metadata in customer-facing installers run via MetaDeploy.

The `deploy_pre` task, which is part of the `dependencies` flow, is responsible for deploying these bundles.

Important: Do not include metadata in `unpackaged/pre` unless it is intended to be delivered to *all* installations of the product.

13.1.2 unpackaged/post: Configuration After Package Install

Projects often include metadata that is genuinely part of the application, but cannot be delivered as part of a managed package for operational reasons. This metadata must be deployed *after* the package's own code and metadata are deployed first and the org is configured.

For example, a product can't deliver `TopicsForObjects` metadata as part of a managed package because that type of metadata isn't packageable. `unpackaged/post` is the home for this kind of metadata, which is stored in subdirectories such as `unpackaged/post/first`.

Note: To learn more about which components are packageable, see the Metadata Coverage Report.

CumulusCI's standard flows that build orgs, such as `dev_org` and `install_prod`, always deploy metadata bundles found in `unpackaged/post`, making it a full-fledged part of the application. It's also easy to include `unpackaged/post` metadata in customer-facing installers run via MetaDeploy.

The `deploy_post` task, which is part of the `config_dev`, `config_qa`, and `config_managed` flows, is responsible for deploying these bundles.

Important: Do not include metadata in `unpackaged/post` unless it is intended to be delivered to *all* environments (both managed installations and unmanaged deployments). It's also critical for managed package projects that this metadata include namespace tokens (see *namespace injection*).

13.1.3 unpackaged/config: Tailor an Org

Projects can come with more than one supported configuration in their CumulusCI automation. For example, projects often support distinct, tailored `dev_org`, `qa_org`, and `install_prod` flows, each of which performs a unique setup for their specific use case.

Unpackaged metadata stored in `unpackaged/config` is a tool to support operational needs that tailor orgs to different configurations. For instance, a testing-oriented scratch org may need to deploy a customized set of Page Layouts to help testers easily visualize data under test. Such page layouts are stored in `unpackaged/config/qa`.

13.2 Unpackaged Metadata Folder Structure

All unpackaged metadata is stored in the `unpackaged` directory tree, which contains these top-level directories.

- `unpackaged/pre`
- `unpackaged/post`
- `unpackaged/config`

These trees contain metadata bundles in Metadata API or Salesforce DX format. CumulusCI automatically converts Salesforce DX-format unpackaged bundles to Metadata API format before deploying them.

13.3 Namespace Injection

Projects that build managed packages often construct their unpackaged metadata to be deployable in multiple contexts, such as:

- Unmanaged deployments, such as developer orgs.
- Unmanaged namespaced scratch orgs.
- Managed contexts, such as a beta test org or a demo org created with `install_prod`.

For example, metadata located in `unpackaged/post` is deployed after the application code in both unmanaged and managed contexts. If that metadata contains references to the application components, it must be deployable when that metadata is namespaced (in a managed context or namespaced scratch org) *and* when it is not (in an unmanaged context).

CumulusCI uses a strategy called *namespace injection* to support this use case. Namespace injection is very powerful, and requires care from the project team to ensure that metadata remains deployable in all contexts.

Important: Projects that are building an org implementation or a non-namespaced package do not have a namespace, or a distinction between managed and unmanaged contexts. These projects typically don't need to use namespace injection.

Metadata files where a namespace is conditionally applied to components for insertion into different contexts must replace the namespace with a *token*, which CumulusCI replaces with the appropriate value or with an empty string as appropriate to the context.

- `%%%NAMESPACE%%%` is replaced with the package's namespace in any context with a namespace (such as a namespaced org or managed org). Otherwise, it remains blank.
- `%%%NAMESPACED_ORG%%%` is replaced with the package's namespace in a namespaced org *only*, not in a managed installation. Otherwise, it remains blank.

Note: This token supports use cases where components in one unpackaged metadata bundle refer to components in another, and the dependency bundle acquires a namespace by being deployed into a namespaced org.

- `%%%NAMESPACE_OR_C%%%` is replaced with the package's namespace in any context with a namespace (such as a namespaced org or managed org). Otherwise, it is replaced with `c`, the generic namespace used in Lightning components.
- `%%%NAMESPACED_ORG_OR_C%%%` is replaced with the package's namespace in a namespaced org *only*, not in a managed installation. Otherwise, it is replaced with `c`, the generic namespace used in Lightning components.
- `%%%NAMESPACE_DOT%%%` is replaced with the package's namespace in any context with a namespace (such as a namespaced org or managed org) followed by a period (`.`) rather than two underscores.

Note: This token is used to construct references to packaged Record Types and Apex classes.

An example case for namespace injection can be found in Salesforce.org's Nonprofit Success Pack (NPSP) managed package. A portion of metadata from NPSP is stored in a subdirectory under `unpackaged/post`, meaning it's deployed after the application metadata. This metadata updates a Compact Layout on the `Account` object, and references packaged metadata from the application as well as from other managed packages. To deploy this as a managed context, this metadata requires the use of namespace tokens to represent the `npsp` namespace, letting CumulusCI automatically adapt the metadata to deploy into managed and unmanaged contexts.

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    <compactLayouts>
        <fullName>NPSP_Household_Account</fullName>
        <fields>Name</fields>
        <fields>npo02__TotalOppAmount__c</fields>
        <fields>%%%NAMESPACE%%%Number_of_Household_Members__c</fields>
        <label>NPSP Household Account</label>
    </compactLayouts>
</CustomObject>
```

Note that only the reference to the NPSP field `Number_of_Household_Members__c` is tokenized. (When installed as part of the managed package, this field appears as `npsp__Number_of_Household_Members__c`.) References to NPSP's own managed package dependency, `npo02`, are not tokenized because this metadata is always namespaced when installed.

If this metadata isn't tokenized, it fails to deploy into an org containing NPSP as a beta or released managed package (because in that context the field `Number_of_Household_Members__c` is namespaced as `npsp__Number_of_Household_Members__c`, and must be referred to as such).

Note: The resolution of component references in namespaced scratch orgs and in managed installations of the same metadata are not identical. Metadata that is tokenized and deploys cleanly in a namespaced scratch org can still fail in a managed context.

13.3.1 Configuration

Most CumulusCI tasks can intelligently determine whether or not to inject the namespace based on the target org. For example, if tokenized metadata is being deployed into an org that contains the project installed as a managed package, CumulusCI knows to inject the namespace; otherwise, it replaces namespace tokens with an empty string for an unmanaged installation.

You can also specify explicit configuration for namespace injection in circumstances where CumulusCI's automatic functionality does not meet your needs, such as when deploying tokenized metadata from another project. If the metadata you are deploying has been tokenized, and you want to deploy metadata with a namespace, use the `namespace_inject: <namespace>` option to inject the namespace.

```
project:
    dependencies:
        - zip_url: https://github.com/SalesforceFoundation/EDA/archive/master.zip
          subfolder: EDA-master/dev_config/src/admin_config
          namespace_inject: hed
```

The metadata in the zip contains the string tokens `%%%NAMESPACE%%%` and `___NAMESPACE___` which is replaced with `hed__` before the metadata is deployed.

To deploy tokenized metadata without any namespace references, specify both `namespace_inject: <namespace>` and `unmanaged: True`. In this example, we do just this for the EDA dependency.

```
project:
    dependencies:
        - zip_url: https://github.com/SalesforceFoundation/EDA/archive/master.zip
          subfolder: EDA-master/dev_config/src/admin_config
```

(continues on next page)

(continued from previous page)

```
        namespace_inject: hed
        unmanaged: True
```

The namespace tokens are replaced with an empty string instead of the namespace, effectively stripping the tokens from the files and filenames.

13.4 Retrieve Unpackaged Metadata

CumulusCI provides tasks to *retrieve changes* to unpackaged metadata, just as with packaged metadata.

When working with unpackaged metadata, it's important to maintain awareness of key considerations related to retrieving metadata that is not part of the main application.

- Take care to separate your development between the different bundles you wish to retrieve. For example, if you have changes to make in the application as well as in unpackaged metadata, complete the application changes first, retrieve them, and then make the unpackaged changes and retrieve those. If you conflate changes to components that live in separate elements of your project, it's difficult to untangle them.
- Whenever possible, build your unpackaged metadata in an org that contains a beta or released managed package. By doing so, the metadata contains namespaces when extracted, which CumulusCI easily replaces with tokens when retrieving metadata. It's difficult to manually tokenize metadata that's retrieved from an unmanaged org without namespaces.

After building changes to unpackaged metadata in a managed org, retrieve it using the `retrieve_changes` task with the additional `namespace_tokenize` option, and use the `path` option to direct the retrieved metadata to your desired unpackaged directory.

In the following example, we run the `retrieve_changes` task to retrieve metadata changes into the `unpackaged/config/qa` subdirectory, and replace references to the namespace `npsp` with the appropriate token.

```
$ cci task run retrieve_changes --path unpackaged/config/qa --namespace_tokenize npsp
```

Projects that use unpackaged metadata extensively define retrieve tasks to streamline this process.

For example, here is a custom task that retrieves changes to specific directory where metadata for QA configuration is kept.

```
retrieve_qa_config:
    description: Retrieves changes to QA configuration metadata
    class_path: cumulusci.tasks.salesforce.sourcetracking.RetrieveChanges
    options:
        path: unpackaged/config/qa
        namespace_tokenize: $project_config.project__package__namespace
```

The `retrieve_changes` task retrieves unpackaged metadata in a managed org, but in this case you must manually insert namespace tokens to deploy metadata in a managed or namespaced context.

13.5 Customize Config Flows

Projects often customize new tasks that deploy `unpackaged/config` bundles, and harness these tasks in flows.

Projects that use `unpackaged/config/qa` often define a `deploy_qa_config` task.

```
deploy_qa_config:
    description: Deploys additional fields used for QA purposes only
    class_path: cumulusci.tasks.salesforce.Deploy
    options:
        path: unpackaged/config/qa
```

This task is then added to relevant flows, such as `config_qa`.

```
config_qa:
    steps:
        3:
            task: deploy_qa_config
```

In most cases, CumulusCI intelligently determines whether or not to inject the namespace. It's rarely necessary to explicitly configure an injection mode. If you need to do so, use the `unmanaged` option:

```
config_regression:
    steps:
        3:
            task: deploy_qa_config
            options:
                unmanaged: False
```

For more details on customizing tasks and flows, see the *Configure CumulusCI* section.

CHAPTER

FOURTEEN

REFERENCE

14.1 Cheat Sheet

CumulusCI offers a great deal of functionality out of the box. This cheat sheet is intended to provide a very brief summary of the most important commands to start working in scratch orgs using CumulusCI, using the basic flows and tasks supplied with the tool.

14.1.1 Naming and Manipulating Orgs

CumulusCI supplies a collection of named org configurations by default. To see what org configurations are available, run `cci org list`. You can provide those names to any of the commands in this guide. Common examples include `dev`, `qa`, `beta`, and `release`. Org names are associated with a scratch org definition file stored in the project's `orgs` directory. The definition file determines how the scratch org is set up.

It's not necessary to name your own orgs, but you may choose to do so if, for example, you'd like to maintain multiple orgs of the same type.

Name a new scratch org

```
$ cci org scratch <configuration_name> <org_name>
```

This creates a new named org that inherits its setup from the configuration name provided.

Get information about a scratch org

```
$ cci org info <org_name>
```

This includes information like the org's domain, username, and password

Open a scratch org in your web browser

```
$ cci org browser <org_name>
```

Set a default scratch org

```
$ cci org default <org_name>
```

This asks CumulusCI to run all flows and tasks against the named org unless otherwise specified. You don't have to specify a default org. You can always direct CumulusCI to use a specific org with the `--org` option when you run a flow or a task.

Delete a scratch org, but leave the org name

```
$ cci org scratch_delete <org_name>
```

Run this command to delete a scratch org so that you can rebuild it, while using the same name.

Remove an org name

```
$ cci org remove <org_name>
```

Note that you will not be able to remove built-in org names, but you can remove names you created with `cci org scratch`.

Connect to a persistent org (sandbox, Developer Edition)

```
$ cci org connect <org_name>
```

Use the `--sandbox` option if this is a sandbox, or any org that uses the `test.salesforce.com` login endpoint.

14.1.2 Building Orgs

Every CumulusCI project includes one or more flows that build an org for a specific purpose or workflow. These flows may be customized for the project, or may be unique to the project. Below are a collection of the standard org building flows that you should expect to find in any CumulusCI project.

Note: This section relies on concepts introduced in the *Key Concepts* section of the documentation.

Note: Each flow should be run against a named org configuration using the `--org` option, or allowed to run against a configured default org.

Flows for Building Orgs

qa_org

This flow builds an unmanaged org that is designed for QA use. Should be used with an org whose configuration is `qa`.

dev_org

This flow builds an unmanaged org that is designed for development use. Should be used with an org whose configuration is `dev` or `dev_namespaced`

install_beta

This flow builds a managed org with the latest beta release installed. Should be used with an org whose configuration is `beta`

install_prod

This flow builds a managed org with the latest release installed. Should be used with an org whose configuration is `release`

push_upgrade_org

This flow builds a managed org that starts with the latest release installed and available for all Profiles. It then upgrades the package dependencies and the package itself to their latest betas, installing upgrades for System Administrators only, and runs `config_qa` to set up the org for testing. This simulates a subscriber push upgrade for non-System Administrator users.

Should be used with an org whose configuration is `release`.

Your project may provide additional org-building flows. Consult the project's automation documentation for more details.

> **Caution:** We do not recommend running an org-building flow against the same scratch org multiple times. While this may work in some situations, in many cases it will fail and/or leave the org in an inconsistent state. If you need to rebuild an org, delete it first. If you need to redeploy updated code into an org, see below.

14.1.3 Common Tasks

Note: Note that each task should be run against a named org configuration using the `--org` option. If not specified, the task will run against a configured default org.

Deploy updated code into an org

```
$ cci flow run deploy_unmanaged
```

Execute Apex unit tests in an org

```
$ cci task run run_tests
```

Execute Robot browser tests

```
$ cci task run robot
```

Review changes to metadata in an org

```
$ cci task run list_changes
```

Retrieve changes to local repository

```
$ cci task run retrieve_changes
```

14.2 Tasks Reference

14.2.1 activate_flow

Description: Activates Flows identified by a given list of Developer Names

Class: cumulusci.tasks.salesforce.activate_flow.ActivateFlow

Command Syntax

```
$ cci task run activate_flow
```

Options

`--developer_names DEVELOPERNAMES`

List of DeveloperNames to query in SOQL

Required

`--status STATUS`

Flag to check whether to activate or deactivate the flow

Default: True

14.2.2 deactivate_flow

Description: deactivates Flows identified by a given list of Developer Names

Class: cumulusci.tasks.salesforce.activate_flow.ActivateFlow

Command Syntax

```
$ cci task run deactivate_flow
```

Options

`--developer_names DEVELOPERNAMES`

List of DeveloperNames to query in SOQL

Required

`--status STATUS`

Flag to check whether to activate or deactivate the flow

Optional

14.2.3 add_page_layout_related_lists

Description: Adds specified Related List to one or more Page Layouts.

Class: cumulusci.tasks.metadata_etl.AddRelatedLists

Command Syntax

```
$ cci task run add_page_layout_related_lists
```

Options

`--related_list RELATEDLIST`

Name of the Related List to include

Required

`--fields FIELDS`

Array of field API names to include in the related list

Optional

`--exclude_buttons EXCLUDEBUTTONS`

Array of button names to suppress from the related list

Optional

`--custom_buttons CUSTOMBUTTONS`

Array of button names to add to the related list

Optional

`--api_names APINAMES`

List of API names of entities to affect

Optional

`--managed MANAGED`

If False, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Default: $project_config.project__package__namespace

`--api_version APIVERSION`

Metadata API version to use, if not project__package__api_version.

Optional

14.2.4 add_page_layout_fields

Description: Adds specified Fields or Visualforce Pages to a Page Layout.

Class: cumulusci.tasks.metadata_etl.layouts.AddFieldsToPageLayout

Inserts the listed fields or Visualforce pages into page layouts specified by API name.

If the targeted item already exists, the layout metadata is not modified.

You may supply a single position option, or multiple options for both pages and fields. The first option to to be matched will be used.

Task option details:

- fields:
  - api_name: [field API name]
  - required: Boolean (default False)
  - read_only: Boolean (default False, not compatible with required)
  - position: (Optional: A list of single or multiple position options.)
    - relative: [before | after | top | bottom]
    - field: [api_name] (Use with relative: before, after)
    - section: [index] (Use with relative: top, bottom)
    - column: [first | last] (Use with relative: top, bottom)
- pages:
  - api_name: [Visualforce Page API name]
  - height: int (Optional. Default: 200)
  - show_label: Boolean (Optional. Default: False)
  - show_scrollbars: Boolean (Optional. Default: False)
  - width: 0-100% (Optional. Default: 100%)

- position: (Optional: A list of single or multiple position options.)
  - relative: [before | after | top | bottom]
  - field: [api_name] (Use with relative: before, after)
  - section: [index] (Use with relative: top, bottom)
  - column: [first | last] (Use with relative: top, bottom)

Example Usage

```
task: add_page_layout_fields
options:
    api_names: "Contact-Contact Layout"
    fields:
      - api_name: Giving_Level__c
        position:
          - relative: bottom
            section: 0
            column: first
      - api_name: Previous_Giving_Level__c
        position:
          - relative: bottom
            section: 0
            column: last
ui_options:
    name: Add custom giving fields to Contact Layout
```

Command Syntax

```
$ cci task run add_page_layout_fields
```

Options

`--fields FIELDS`

List of fields. See task info for structure.

Optional

`--pages PAGES`

List of Visualforce Pages. See task info for structure.

Optional

`--api_names APINAMES`

List of API names of entities to affect

Optional

`--managed MANAGED`

If False, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Optional

`--api_version APIVERSION`

Metadata API version to use, if not project__package__api_version.

Optional

14.2.5 add_profile_ip_ranges

Description: Adds (or optionally replaces) IP Login Ranges to the specified Profiles.

Class: cumulusci.tasks.metadata_etl.permissions.AddIPRanges

Command Syntax

```
$ cci task run add_profile_ip_ranges
```

Options

`--ranges RANGES`

A list of IP ranges, specified as dicts with the keys 'description' (optional) and either 'start_address' and 'end_address' or 'network' (in CIDR notation).

Required

`--replace REPLACE`

If True, replace all existing ranges. Otherwise, just add ranges. Default is False.

Optional

14.2.6 add_standard_value_set_entries

Description: Adds specified picklist entries to a Standard Value Set.

Class: cumulusci.tasks.metadata_etl.AddValueSetEntries

Example Usage

```
task: add_standard_value_set_entries
options:
    api_names: CaseOrigin
    entries:
        - fullName: New Account
          label: New Account
        - fullName: Questionable Contact
          label: Questionable Contact
    ui_options:
        name: Add values to Case Origin picklist
```

Command Syntax

```
$ cci task run add_standard_value_set_entries
```

Options

`--api_names APINAMES`

List of API names of StandardValueSets to affect, such as 'OpportunityStage', 'AccountType', 'CaseStatus', 'LeadStatus'

Required

`--entries ENTRIES`

Array of standardValues to insert. Each standardValue should contain the keys 'fullName', the API name of the entry, and 'label', the user-facing label. OpportunityStage entries require the additional keys 'closed', 'won', 'forecastCategory', and 'probability'; CaseStatus entries require 'closed'; LeadStatus entries require 'converted'.

Required

`--managed MANAGED`

If False, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Default: $project_config.project__package__namespace

`--api_version APIVERSION`

Metadata API version to use, if not project__package__api_version.

Optional

14.2.7 add_picklist_entries

Description: Adds specified picklist entries to a custom picklist field.

Class: cumulusci.tasks.metadata_etl.picklists.AddPicklistEntries

Command Syntax

```
$ cci task run add_picklist_entries
```

Options

`--picklists PICKLISTS`

List of picklists to affect, in Object__c.Field__c form.

Required

`--entries ENTRIES`

Array of picklist values to insert. Each value should contain the keys 'fullName', the API name of the entry, and 'label', the user-facing label. Optionally, specify *default: True* on exactly one entry to make that value the default. Any existing values will not be affected other than setting the default (labels of existing entries are not changed). To order values, include the 'add_before' key. This will insert the new value before the existing value with the given API name, or at the end of the list if not present.

Required

`--record_types RECORDTYPES`

List of Record Type developer names for which the new values should be available. If any of the entries have *default: True*, they are also made default for these Record Types. Any Record Types not present in the target org will be ignored, and * is a wildcard. Default behavior is to do nothing.

Optional

`--api_names APINAMES`

List of API names of entities to affect

Optional

`--managed MANAGED`

If False, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Default: $project_config.project__package__namespace

`--api_version APIVERSION`

Metadata API version to use, if not project__package__api_version.

Optional

14.2.8 add_fields_to_field_set

Description: Adds specified fields to a given field set.

Class: cumulusci.tasks.metadata_etl.field_sets.AddFieldsToFieldSet

Command Syntax

```
$ cci task run add_fields_to_field_set
```

Options

`--field_set FIELDSET`

Name of field set to affect, in Object__c.FieldSetName form.

Required

`--fields FIELDS`

Array of field API names to add to the field set. Can include related fields using AccountId.Name or Lookup__r.CustomField__c style syntax.

Required

`--api_names APINAMES`

List of API names of entities to affect

Optional

`--managed MANAGED`

If False, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Optional

`--api_version APIVERSION`

Metadata API version to use, if not project__package__api_version.

Optional

14.2.9 add_permission_set_perms

Description: Adds specified Apex class access and Field-Level Security to a Permission Set.

Class: cumulusci.tasks.metadata_etl.AddPermissionSetPermissions

Command Syntax

```
$ cci task run add_permission_set_perms
```

Options

`--field_permissions FIELDPERMISSIONS`

Array of fieldPermissions objects to upsert into permission_set. Each fieldPermission requires the following attributes: 'field': API Name of the field including namespace; 'readable': boolean if field can be read; 'editable': boolean if field can be edited

Optional

`--class_accesses CLASSACCESSES`

Array of classAccesses objects to upsert into permission_set. Each classAccess requires the following attributes: 'apexClass': Name of Apex Class. If namespaced, make sure to use the form "namespace__ApexClass"; 'enabled': boolean if the Apex Class can be accessed.

Optional

`--api_names APINAMES`

List of API names of entities to affect

Optional

`--managed MANAGED`

If False, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Default: $project_config.project__package__namespace

`--api_version APIVERSION`

Metadata API version to use, if not project__package__api_version.

Optional

14.2.10 add_record_action_list_item

Description: Adds the specified 'Record' context Lightning button/action to the provided page layout.

Class: cumulusci.tasks.metadata_etl.layouts.AddRecordPlatformActionListItem

Inserts the targeted lightning button/action into specified layout's PlatformActionList with a 'Record' actionListContext. - If the targeted lightning button/action already exists,

the layout metadata is not modified.

- **If there is no 'Record' context PlatformActionList,**
  we will generate one and add the specified action

Task definition example:

dev_inject_apply_quick_action_into_account_layout: group: "Demo config and storytelling" description: Adds an Apply Quick Action button to the beggining of the button list on the Experiences Account Layout. class_path: tasks.layouts.InsertRecordPlatformActionListItem options:

api_names: "Account-%%%NAMESPACE%%%Experiences Account Layout" action_name: "Account.Apply" action_type: QuickAction place_first: True

Reference Documentation: https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/meta_layouts.htm#PlatformActionList

Command Syntax

```
$ cci task run add_record_action_list_item
```

Options

`--action_type ACTIONTYPE`

platformActionListItems.actionType like 'QuickAction' or 'CustomButton'

Required

`--action_name ACTIONNAME`

platformActionListItems.actionName. The API name for the action to be added.

Required

`--place_first PLACEFIRST`

When 'True' the specified Record platformActionListItem will be inserted before any existing on the layout. Default is 'False'

Optional

`--api_names APINAMES`

List of API names of entities to affect

Optional

`--managed MANAGED`

If False, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Optional

`--api_version APIVERSION`

Metadata API version to use, if not project__package__api_version.

Optional

14.2.11 add_remote_site_settings

Description: Adds the specified RemoteSiteSettings records to an org.

Class: cumulusci.tasks.metadata_etl.remote_site_settings.AddRemoteSiteSettings

Example Usage

```
task: add_remote_site_settings
options:
    records:
        - full_name: ExampleRemoteSiteSetting
          description: Descriptions can be included optionally
          url: https://test.salesforce.com
          is_active: True
```

Command Syntax

```
$ cci task run add_remote_site_settings
```

Options

`--records RECORDS`

> Array of RemoteSiteSetting records to insert. Each RemoteSiteSetting requires the keys: 'full_name', 'is_active', and 'url'. 'description' is optional, and defaults to an empty string. 'disable_security_protocol' is optional, and defaults to False.

Required

14.2.12 assign_compact_layout

Description: Assigns the Compact Layout specified in the 'value' option to the Custom Objects in 'api_names' option.

Class: cumulusci.tasks.metadata_etl.UpdateMetadataFirstChildTextTask

Metadata ETL task to update a single child element's text within metadata XML.

If the child doesn't exist, the child is created and appended to the Metadata. Furthermore, the `value` option is namespaced injected if the task is properly configured.

Example: Assign a Custom Object's Compact Layout

Researching CustomObject in the Metadata API documentation or even retrieving the CustomObject's Metadata for inspection, we see the `compactLayoutAssignment` Field. We want to assign a specific Compact Layout for our Custom Object, so we write the following CumulusCI task in our project's `cumulusci.yml`.

```
tasks:
    assign_compact_layout:
        class_path: cumulusci.tasks.metadata_etl.UpdateMetadataFirstChildTextTask
        options:
            managed: False
```

(continues on next page)

(continued from previous page)

```
            namespace_inject: $project_config.project__package__namespace
            entity: CustomObject
            api_names: OurCustomObject__c
            tag: compactLayoutAssignment
            value: "%%%NAMESPACE%%%DifferentCompactLayout"
            # We include a namespace token so it's easy to use this task in a managed␣
↪context.
```

Suppose the original CustomObject metadata XML looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    ...
    <label>Our Custom Object</label>
    <compactLayoutAssignment>OriginalCompactLayout</compactLayoutAssignment>
    ...
</CustomObject>
```

After running `cci task run assign_compact_layout`, the CustomObject metadata XML is deployed as:

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    ...
    <label>Our Custom Object</label>
    <compactLayoutAssignment>DifferentCompactLayout</compactLayoutAssignment>
    ...
</CustomObject>
```

Command Syntax

```
$ cci task run assign_compact_layout
```

Options

`--metadata_type METADATATYPE`

Metadata Type

Default: CustomObject

`--tag TAG`

Targeted tag. The text of the first instance of this tag within the metadata entity will be updated.

Default: compactLayoutAssignment

`--value VALUE`

Desired value to set for the targeted tag's text. This value is namespace-injected.

Required

`--api_names APINAMES`

List of API names of entities to affect

Optional

`--managed MANAGED`

If False, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Default: $project_config.project__package__namespace

`--api_version APIVERSION`

Metadata API version to use, if not project__package__api_version.

Optional

14.2.13 assign_permission_sets

Description: Assigns specified Permission Sets to the current user, if not already assigned.

Class: cumulusci.tasks.salesforce.users.permsets.AssignPermissionSets

Assigns Permission Sets whose Names are in `api_names` to either the default org user or the user whose Alias is `user_alias`. This task skips assigning Permission Sets that are already assigned.

Command Syntax

```
$ cci task run assign_permission_sets
```

Options

`--api_names APINAMES`

API Names of desired Permission Sets, separated by commas.

Required

`--user_alias USERALIAS`

Target user aliases, separated by commas. Defaults to the current running user.

Optional

14.2.14 assign_permission_set_groups

Description: Assigns specified Permission Set Groups to the current user, if not already assigned.

Class: cumulusci.tasks.salesforce.users.permsets.AssignPermissionSetGroups

Assigns Permission Set Groups whose Developer Names are in `api_names` to either the default org user or the user whose Alias is `user_alias`. This task skips assigning Permission Set Groups that are already assigned.

Command Syntax

```
$ cci task run assign_permission_set_groups
```

Options

`--api_names APINAMES`

API Developer Names of desired Permission Set Groups, separated by commas.

Required

`--user_alias USERALIAS`

Alias of target user (if not the current running user, the default).

Optional

14.2.15 assign_permission_set_licenses

Description: Assigns specified Permission Set Licenses to the current user, if not already assigned.

Class: cumulusci.tasks.salesforce.users.permsets.AssignPermissionSetLicenses

Assigns Permission Set Licenses whose Developer Names are in `api_names` to either the default org user or the user whose Alias is `user_alias`. This task skips assigning Permission Set Licenses that are already assigned.

Permission Set Licenses are usually associated with a Permission Set, and assigning the Permission Set usually assigns the associated Permission Set License automatically. However, in non-namespaced developer scratch orgs, assigning the associated Permission Set may not automatically assign the Permission Set License, and this task will ensure the Permission Set Licenses are assigned.

Command Syntax

```
$ cci task run assign_permission_set_licenses
```

Options

`--api_names APINAMES`

API Developer Names of desired Permission Set Licenses, separated by commas.

Required

`--user_alias USERALIAS`

Alias of target user (if not the current running user, the default).

Optional

14.2.16 batch_apex_wait

Description: Waits on a batch apex or queueable apex job to finish.

Class: cumulusci.tasks.apex.batch.BatchApexWait

Command Syntax

```
$ cci task run batch_apex_wait
```

Options

`--class_name CLASSNAME`

Name of the Apex class to wait for.

Required

`--poll_interval POLLINTERVAL`

Seconds to wait before polling for batch or queueable job completion. Defaults to 10 seconds.

Optional

14.2.17 check_my_domain_active

Description: Runs as a preflight check to determine whether My Domain is active.

Class: cumulusci.tasks.preflight.settings.CheckMyDomainActive

Command Syntax

```
$ cci task run check_my_domain_active
```

14.2.18 check_sobjects_available

Description: Runs as a preflight check to determine whether specific sObjects are available.

Class: cumulusci.tasks.preflight.sobjects.CheckSObjectsAvailable

As a MetaDeploy preflight check, validates that an sObject is present in the schema.

The task can be used as a preflight check thus:

```
3:
    task: insert_sobject_records
    checks:
        - when: "'ContentNote' not in tasks.check_sobjects_available()"
          action: error
          message: "Enhanced Notes are not turned on."
```

Command Syntax

```
$ cci task run check_sobjects_available
```

14.2.19 check_sobject_permissions

Description: Runs as a preflight check to determine whether specific sObjects are permissioned as desired (options are required).

Class: cumulusci.tasks.preflight.sobjects.CheckSObjectPerms

As a MetaDeploy preflight check, validates that an sObject's permissions are in the expected state.

For example, specify:

```
check_sobject_permissions:
    options:
        Account:
            createable: True
            updateable: False
        Contact:
            createable: False
```

to validate that the Account object is createable but not updateable, and the Contact object is not createable. The output is True if all sObjects and permissions are present and matching the specification.

Given the above configuration, the task can be used as a preflight check in a MetaDeploy plan:

```
3:
    task: insert_sobject_records
    checks:
        - when: "not tasks.check_sobject_permissions()"
          action: error
          message: "sObject permissions are not configured correctly."
```

Command Syntax

```
$ cci task run check_sobject_permissions
```

Options

`--permissions PERMISSIONS`

The object permissions to check. Each key should be an sObject API name, whose value is a map of describe keys, such as *queryable* and *createable*, to their desired values (True or False). The output is True if all sObjects and permissions are present and matching the specification. See the task documentation for examples.

Required

14.2.20 check_advanced_currency_management

Description: Runs as a preflight check to determine whether Advanced Currency Management is active (True result means the feature is active).

Class: cumulusci.tasks.preflight.sobjects.CheckSObjectPerms

As a MetaDeploy preflight check, validates that an sObject's permissions are in the expected state.

For example, specify:

```
check_sobject_permissions:
    options:
        Account:
            createable: True
            updateable: False
        Contact:
            createable: False
```

to validate that the Account object is createable but not updateable, and the Contact object is not createable. The output is True if all sObjects and permissions are present and matching the specification.

Given the above configuration, the task can be used as a preflight check in a MetaDeploy plan:

```
3:
    task: insert_sobject_records
    checks:
        - when: "not tasks.check_sobject_permissions()"
          action: error
          message: "sObject permissions are not configured correctly."
```

Command Syntax

```
$ cci task run check_advanced_currency_management
```

Options

`--permissions PERMISSIONS`

The object permissions to check. Each key should be an sObject API name, whose value is a map of describe keys, such as *queryable* and *createable*, to their desired values (True or False). The output is True if all sObjects and permissions are present and matching the specification. See the task documentation for examples.

Default: {'DatedConversionRate': {'createable': True}}

14.2.21 check_org_wide_defaults

Description: Runs as a preflight check to validate Organization-Wide Defaults.

Class: cumulusci.tasks.preflight.sobjects.CheckSObjectOWDs

As a MetaDeploy preflight check, validates that an sObject's Org-Wide Defaults are in the expected state.

For example, specify:

```
check_org_wide_defaults:
    options:
        org_wide_defaults:
            - api_name: Account
              internal_sharing_model: Private
              external_sharing_model: Private
            - api_name: Contact
              internal_sharing_model: Private
```

to validate that the Account object has Private internal and external OWDs, and Contact a Private internal model. The output is True if all sObjects and permissions are present and matching the specification.

Given the above configuration, the task can be used as a preflight check in a MetaDeploy plan:

```
3:
    task: insert_sobject_records
    checks:
        - when: "not tasks.check_org_wide_defaults()"
          action: error
          message: "Org-Wide Defaults are not configured correctly."
```

Command Syntax

```
$ cci task run check_org_wide_defaults
```

Options

`--org_wide_defaults ORGWIDEDEFAULTS`

The Organization-Wide Defaults to check, organized as a list with each element containing the keys api_name, internal_sharing_model, and external_sharing_model. NOTE: you must have External Sharing Model turned on in Sharing Settings to use the latter feature. Checking External Sharing Model when it is turned off will fail the preflight.

Required

14.2.22 check_org_settings_value

Description: Runs as a preflight check to validate organization settings.

Class: cumulusci.tasks.preflight.settings.CheckSettingsValue

Command Syntax

```
$ cci task run check_org_settings_value
```

Options

`--settings_type SETTINGSTYPE`

The API name of the Settings entity to be checked, such as ChatterSettings.

Required

`--settings_field SETTINGSFIELD`

The API name of the field on the Settings entity to check.

Required

`--value VALUE`

The value to check for

Required

`--treat_missing_as_failure TREATMISSINGASFAILURE`

If True, treat a missing Settings entity as a preflight failure, instead of raising an exception. Defaults to False.

Optional

14.2.23 check_chatter_enabled

Description: Runs as a preflight check to validate Chatter is enabled.

Class: cumulusci.tasks.preflight.settings.CheckSettingsValue

Command Syntax

```
$ cci task run check_chatter_enabled
```

Options

`--settings_type SETTINGSTYPE`

The API name of the Settings entity to be checked, such as ChatterSettings.

Default: ChatterSettings

`--settings_field SETTINGSFIELD`

The API name of the field on the Settings entity to check.

Default: IsChatterEnabled

`--value VALUE`

The value to check for

Default: True

`--treat_missing_as_failure TREATMISSINGASFAILURE`

If True, treat a missing Settings entity as a preflight failure, instead of raising an exception. Defaults to False.

Optional

14.2.24 check_enhanced_notes_enabled

Description: Preflight check to validate that Enhanced Notes are enabled.

Class: cumulusci.tasks.preflight.settings.CheckSettingsValue

Command Syntax

```
$ cci task run check_enhanced_notes_enabled
```

Options

`--settings_type SETTINGSTYPE`

The API name of the Settings entity to be checked, such as ChatterSettings.

Default: EnhancedNotesSettings

`--settings_field SETTINGSFIELD`

The API name of the field on the Settings entity to check.

Default: IsEnhancedNotesEnabled

`--value VALUE`

The value to check for

Default: True

`--treat_missing_as_failure TREATMISSINGASFAILURE`

If True, treat a missing Settings entity as a preflight failure, instead of raising an exception. Defaults to False.

Optional

14.2.25 retrieve_tasks

Description: Retrieves the tasks under the particular category or group

Class: cumulusci.tasks.preflight.retrieve_tasks.RetrieveTasks

Command Syntax

```
$ cci task run retrieve_tasks
```

Options

`--group_name GROUPNAME`

Name of the category or Group

Required

14.2.26 custom_settings_value_wait

Description: Waits for a specific field value on the specified custom settings object and field

Class: cumulusci.tasks.salesforce.custom_settings_wait.CustomSettingValueWait

Command Syntax

```
$ cci task run custom_settings_value_wait
```

Options

`--object OBJECT`

Name of the Hierarchical Custom Settings object to query. Can include the %%%NAMESPACE%%% token.

Required

`--field FIELD`

Name of the field on the Custom Settings to query. Can include the %%%NAMESPACE%%% token.

Required

`--value VALUE`

Value of the field to wait for (String, Integer or Boolean).

Required

`--managed MANAGED`

If True, will insert the project's namespace prefix. Defaults to False or no namespace.

Optional

`--namespaced NAMESPACED`

If True, the %%%NAMESPACE%%% token will get replaced with the namespace prefix for the object and field.Defaults to False.

Optional

`--poll_interval POLLINTERVAL`

Seconds to wait before polling for batch job completion. Defaults to 10 seconds.

Optional

14.2.27 command

Description: Run an arbitrary command

Class: cumulusci.tasks.command.Command

Example Command-line Usage: `cci task run command -o command "echo 'Hello command task!'"`

Example Task to Run Command:

```
hello_world:
    description: Says hello world
    class_path: cumulusci.tasks.command.Command
    options:
        command: echo 'Hello World!'
```

Command Syntax

```
$ cci task run command
```

Options

`--command COMMAND`

The command to execute

Required

`--pass_env PASSENV`

If False, the current environment variables will not be passed to the child process. Defaults to True

Required

`--dir DIR`

If provided, the directory where the command should be run from.

Optional

`--env ENV`

Environment variables to set for command. Must be flat dict, either as python dict from YAML or as JSON string.

Optional

`--interactive INTERACTIVE`

If True, the command will use stderr, stdout, and stdin of the main process.Defaults to False.

Optional

14.2.28 composite_request

Description: Execute a series of REST API requests in a single call

Class: cumulusci.tasks.salesforce.composite.CompositeApi

This task is a wrapper for Composite REST API calls. Given a list of JSON files (one request body per file), POST each and process the returned composite result. Files are processed in the order given by the `data_files` option.

In addition, this task will process the request body and replace namespace (`%%%NAMESPACE%%%`) and user ID (`%%%USERID%%%`) tokens. To avoid username collisions, use the `randomize_username` option to replace the top-level domains in any `Username` field with a random string.

When the top-level `allOrNone` property for the request is set to true a SalesforceException is raised if an error is returned for any subrequest, otherwise partial successes will not raise an exception.

Example Task Definition

```
tasks:
    example_composite_request:
        class_path: cumulusci.tasks.salesforce.composite.CompositeApi
        options:
           data_files:
               - "datasets/composite/users.json"
               - "datasets/composite/setup_objects.json"
```

Command Syntax

```
$ cci task run composite_request
```

Options

`--data_files DATAFILES`

A list of paths, where each path is a JSON file containing a composite request body.

Required

`--managed MANAGED`

If True, replaces namespace tokens with the namespace prefix.

Optional

`--namespaced NAMESPACED`

If True, replaces namespace tokens with the namespace prefix.

Optional

`--randomize_username RANDOMIZEUSERNAME`

If True, randomize the TLD for any 'Username' fields.

Optional

14.2.29 create_community

Description: Creates a Community in the target org using the Connect API

Class: cumulusci.tasks.salesforce.CreateCommunity

Create a Salesforce Community via the Connect API.

Specify the *template* "VF Template" for Visualforce Tabs community, or the name for a specific desired template

Command Syntax

```
$ cci task run create_community
```

Options

`--template TEMPLATE`

Name of the template for the community.

Required

`--name NAME`

Name of the community.

Required

`--description DESCRIPTION`

Description of the community.

Optional

`--url_path_prefix URLPATHPREFIX`

URL prefix for the community.

Optional

`--retries RETRIES`

Number of times to retry community creation request

Optional

`--timeout TIMEOUT`

Time to wait, in seconds, for the community to be created

Optional

`--skip_existing SKIPEXISTING`

If True, an existing community with the same name will not raise an exception.

Optional

14.2.30 connected_app

Description: Creates the Connected App needed to use persistent orgs in the CumulusCI keychain

Class: cumulusci.tasks.connectedapp.CreateConnectedApp

Command Syntax

```
$ cci task run connected_app
```

Options

`--label LABEL`

The label for the connected app. Must contain only alphanumeric and underscores

Default: CumulusCI

`--email EMAIL`

The email address to associate with the connected app. Defaults to email address from the github service if configured.

Optional

`--username USERNAME`

Create the connected app in a different org. Defaults to the defaultdevhubusername configured in sfdx.

Optional

`--connect CONNECT`

If True, the created connected app will be stored as the CumulusCI connected_app service in the keychain.

Default: True

`--overwrite OVERWRITE`

If True, any existing connected_app service in the CumulusCI keychain will be overwritten. Has no effect if the connect option is False.

Optional

14.2.31 create_network_member_groups

Description: Creates NetworkMemberGroup records which grant access to an Experience Site (Community) for specified Profiles or Permission Sets

Class: cumulusci.tasks.salesforce.network_member_group.CreateNetworkMemberGroups

Command Syntax

```
$ cci task run create_network_member_groups
```

Options

`--network_name NETWORKNAME`

Name of Network to add NetworkMemberGroup children records.

Required

`--profile_names PROFILENAMES`

List of Profile Names to add as NetworkMemberGroups for this Network.

Optional

`--permission_set_names PERMISSIONSETNAMES`

List of PermissionSet Names to add as NetworkMemberGroups for this Network.

Optional

14.2.32 insert_record

Description: Inserts a record of any sObject using the REST API

Class: cumulusci.tasks.salesforce.insert_record.InsertRecord

For example:

cci task run insert_record –org dev -o object PermissionSet -o values Name:HardDelete,PermissionsBulkApiHardDelete:true

Command Syntax

```
$ cci task run insert_record
```

Options

`--object OBJECT`

An sObject type to insert

Required

`--values VALUES`

Field names and values in the format 'aa:bb,cc:dd', or a YAML dict in cumulusci.yml.

Required

`--tooling TOOLING`

If True, use the Tooling API instead of REST API.

Optional

14.2.33 create_package

Description: Creates a package in the target org with the default package name for the project

Class: cumulusci.tasks.salesforce.CreatePackage

Command Syntax

```
$ cci task run create_package
```

Options

`--package PACKAGE`

The name of the package to create. Defaults to project__package__name

Required

`--api_version APIVERSION`

The api version to use when creating the package. Defaults to project__package__api_version

Required

14.2.34 create_package_version

Description: Uploads a 2nd-generation package (2GP) version

Class: cumulusci.tasks.create_package_version.CreatePackageVersion

Facilitates the upload of 2GP (second-generation packaging) package versions using CumulusCI.

The target org is used both for looking up dependency package IDs and configuring the build org during the package upload. Ensure the specified org is a scratch org with the correct configuration for these purposes.

Command Syntax

```
$ cci task run create_package_version
```

Options

`--package_type PACKAGETYPE`

Package type (Unlocked or Managed)

Required

`--package_name PACKAGENAME`

Name of package

Optional

`--namespace NAMESPACE`

Package namespace

Optional

`--version_name VERSIONNAME`

Version name

Optional

`--version_base VERSIONBASE`

The version number to use as a base before incrementing. Optional; defaults to the highest existing version number of this package. Can be set to `latest_github_release` to use the version of the most recent release published to GitHub.

Optional

`--version_type VERSIONTYPE`

The part of the version number to increment. Options are major, minor, patch, build. Defaults to build

Optional

`--skip_validation SKIPVALIDATION`

If true, skip validation of the package version. Default: false. Skipping validation creates packages more quickly, but they cannot be promoted for release.

Optional

`--org_dependent ORGDEPENDENT`

If true, create an org-dependent unlocked package. Default: false.

Optional

`--post_install_script POSTINSTALLSCRIPT`

Post-install script (for managed packages)

Optional

`--uninstall_script UNINSTALLSCRIPT`

Uninstall script (for managed packages)

Optional

`--install_key INSTALLKEY`

Install key for package. Default is no install key.

Optional

`--force_upload FORCEUPLOAD`

If true, force creating a new package version even if one with the same contents already exists

Optional

`--static_resource_path STATICRESOURCEPATH`

The path where decompressed static resources are stored. Any subdirectories found will be zipped and added to the staticresources directory of the build.

Optional

`--ancestor_id ANCESTORID`

The 04t Id to use for the ancestor of this package. Optional; defaults to no ancestor specified. Can be set to `latest_github_release` to use the most recent production version published to GitHub.

Optional

`--resolution_strategy RESOLUTIONSTRATEGY`

The name of a sequence of resolution_strategy (from project__dependency_resolutions) to apply to dynamic dependencies. Defaults to 'production'.

Optional

`--create_unlocked_dependency_packages CREATEUNLOCKEDDEPENDENCYPACKAGES`

If True, create unlocked packages for unpackaged metadata in this project and dependencies. Defaults to False.

Optional

14.2.35 create_managed_src

Description: Modifies the src directory for managed deployment. Strips //cumulusci-managed from all Apex code

Class: cumulusci.tasks.metadata.managed_src.CreateManagedSrc

Apex classes which use the @deprecated annotation can comment it out using //cumulusci-managed so that it can be deployed as part of unmanaged metadata, where this annotation is not allowed. This task is for use when deploying to a packaging org to remove the comment so that the annotation takes effect.

Command Syntax

```
$ cci task run create_managed_src
```

Options

`--path PATH`

The path containing metadata to process for managed deployment

Default: src

`--revert_path REVERTPATH`

The path to copy the original metadata to for the revert call

Default: src.orig

14.2.36 create_permission_set

Description: Creates a Permission Set with specified User Permissions and assigns it to the running user.

Class: cumulusci.tasks.salesforce.create_permission_sets.CreatePermissionSet

Command Syntax

```
$ cci task run create_permission_set
```

Options

`--api_name APINAME`

API name of generated Permission Set

Required

`--user_permissions USERPERMISSIONS`

List of User Permissions to include in the Permission Set.

Required

`--label LABEL`

Label of generated Permission Set

Optional

14.2.37 create_bulk_data_permission_set

Description: Creates a Permission Set with the Hard Delete and Set Audit Fields user permissions. NOTE: the org setting to allow Set Audit Fields must be turned on.

Class: cumulusci.tasks.salesforce.create_permission_sets.CreatePermissionSet

Command Syntax

```
$ cci task run create_bulk_data_permission_set
```

Options

`--api_name APINAME`

API name of generated Permission Set

Default: CumulusCI_Bulk_Data

`--user_permissions USERPERMISSIONS`

List of User Permissions to include in the Permission Set.

Default: ['PermissionsBulkApiHardDelete', 'PermissionsCreateAuditFields']

`--label LABEL`

Label of generated Permission Set

Default: CumulusCI Bulk Data

14.2.38 create_unmanaged_ee_src

Description: Modifies the src directory for unmanaged deployment to an EE org

Class: cumulusci.tasks.metadata.ee_src.CreateUnmanagedEESrc

Command Syntax

```
$ cci task run create_unmanaged_ee_src
```

Options

`--path PATH`

The path containing metadata to process for managed deployment

Default: src

`--revert_path REVERTPATH`

The path to copy the original metadata to for the revert call

Default: src.orig

14.2.39 create_blank_profile

Description: Creates a blank profile, or a profile with no permissions

Class: cumulusci.tasks.salesforce.profiles.CreateBlankProfile

Command Syntax

```
$ cci task run create_blank_profile
```

Options

`--name NAME`

The name of the the new profile

Required

`--license LICENSE`

The name of the salesforce license to use in the profile, defaults to 'Salesforce'

Default: Salesforce

`--license_id LICENSEID`

The ID of the salesforce license to use in the profile.

Optional

`--description DESCRIPTION`

The description of the the new profile

Optional

14.2.40 retrieve_profile

Description: Given a list of profiles, the task retrieves all complete profiles along with their associated dependencies for all permissionable entities - ApexClass, ApexPage, CustomApplications, CustomObjects, CustomPermissions, CustomTabs, ExternalDataSources and Flows

Class: cumulusci.tasks.salesforce.retrieve_profile.RetrieveProfile

Command Syntax

```
$ cci task run retrieve_profile
```

Options

`--profiles PROFILES`

List of profile API names that you want to retrieve

Required

`--path PATH`

Target folder path. By default, it uses force-app/main/default

Optional

`--strict_mode STRICTMODE`

When set to False, enables leniency by ignoring missing profiles when provided with a list of profiles. When set to True, enforces strict validation, causing a failure if any profile is not present in the list. Default is True

Optional

14.2.41 delete_data

Description: Query existing data for a specific sObject and perform a Bulk API delete of all matching records.

Class: cumulusci.tasks.bulkdata.DeleteData

Command Syntax

```
$ cci task run delete_data
```

Options

`--objects OBJECTS`

A list of objects to delete records from in order of deletion. If passed via command line, use a comma separated string

Required

`--where WHERE`

A SOQL where-clause (without the keyword WHERE). Only available when 'objects' is length 1.

Optional

`--hardDelete HARDDELETE`

If True, perform a hard delete, bypassing the Recycle Bin. Note that this requires the Bulk API Hard Delete permission. Default: False

Optional

`--ignore_row_errors IGNOREROWERRORS`

If True, allow the operation to continue even if individual rows fail to delete.

Optional

`--inject_namespaces INJECTNAMESPACES`

If True, the package namespace prefix will be automatically added to (or removed from) objects and fields based on the name used in the org. Defaults to True.

Optional

`--api API`

The desired Salesforce API to use, which may be 'rest', 'bulk', or 'smart' to auto-select based on record volume. The default is 'smart'.

Optional

14.2.42 update_data

Description: Update records of an sObject matching a where-clause.

Class: cumulusci.tasks.bulkdata.update_data.UpdateData

Command Syntax

```
$ cci task run update_data
```

Options

`--object OBJECT`

An SObject

Required

`--recipe RECIPE`

Snowfakery recipe to be executed on each row

Required

`--where WHERE`

A SOQL where-clause (without the keyword WHERE).

Optional

`--api API`

The desired Salesforce API to use, which may be 'rest', 'bulk', or 'smart' to auto-select based on record volume. The default is 'smart'.

Optional

`--fields FIELDS`

Fields to download as input to the Snowfakery recipe

Optional

`--recipe_options RECIPEOPTIONS`

Pass values to override options in the format VAR1:foo,VAR2:bar

Example: –recipe_options weight:10,color:purple

Optional

`--ignore_row_errors IGNOREROWERRORS`

If True, allow the operation to continue even if individual rows fail to delete.

Optional

14.2.43 deploy

Description: Deploys the src directory of the repository to the org

Class: cumulusci.tasks.salesforce.Deploy

Command Syntax

```
$ cci task run deploy
```

Options

`--path PATH`

The path to the metadata source to be deployed

Default: src

`--unmanaged UNMANAGED`

If True, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Optional

`--namespace_strip NAMESPACESTRIP`

If set, all namespace prefixes for the namespace specified are stripped from files and filenames

Optional

`--check_only CHECKONLY`

If True, performs a test deployment (validation) of components without saving the components in the target org

Optional

`--collision_check COLLISIONCHECK`

If True, performs a collision check with metadata already present in the target org

Optional

`--test_level TESTLEVEL`

Specifies which tests are run as part of a deployment. Valid values: NoTestRun, RunLocalTests, RunAllTestsInOrg, RunSpecifiedTests.

Optional

`--specified_tests SPECIFIEDTESTS`

Comma-separated list of test classes to run upon deployment. Applies only with test_level set to RunSpecifiedTests.

Optional

`--static_resource_path STATICRESOURCEPATH`

The path where decompressed static resources are stored. Any subdirectories found will be zipped and added to the staticresources directory of the build.

Optional

`--namespaced_org NAMESPACEDORG`

If True, the tokens %%%NAMESPACED_ORG%%% and ___NAMESPACED_ORG___ will get replaced with the namespace. The default is false causing those tokens to get stripped and replaced with an empty string. Set this if deploying to a namespaced scratch org or packaging org.

Optional

`--clean_meta_xml CLEANMETAXML`

Defaults to True which strips the <packageVersions/> element from all meta.xml files. The packageVersion element gets added automatically by the target org and is set to whatever version is installed in the org. To disable this, set this option to False

Optional

`--transforms TRANSFORMS`

Apply source transforms before deploying. See the CumulusCI documentation for details on how to specify transforms.

Optional

`--rest_deploy RESTDEPLOY`

If True, deploy metadata using REST API

Optional

14.2.44 deploy_marketing_cloud_package

Description: Deploys a package zip file to a Marketing Cloud Tenant via the Marketing Cloud Package Manager API.

Class: cumulusci.tasks.marketing_cloud.deploy.MarketingCloudDeployTask

Command Syntax

```
$ cci task run deploy_marketing_cloud_package
```

Options

`--package_zip_file PACKAGEZIPFILE`

Path to the package zipfile that will be deployed.

Required

`--custom_inputs CUSTOMINPUTS`

Specify custom inputs to the deployment task. Takes a mapping from input key to input value (e.g. 'companyName:Acme,companyWebsite:https://www.salesforce.org:8080').

Optional

`--name NAME`

The name to give to this particular deploy call. Defaults to a universally unique identifier.

Optional

`--endpoint ENDPOINT`

Override the default endpoint for the Marketing Cloud package manager API (optional)

Optional

14.2.45 marketing_cloud_create_subscriber_attribute

Description: Creates a Subscriber Attribute via the Marketing Cloud SOAP API.

Class: cumulusci.tasks.marketing_cloud.api.CreateSubscriberAttribute

Command Syntax

```
$ cci task run marketing_cloud_create_subscriber_attribute
```

Options

`--attribute_name ATTRIBUTENAME`

The name of the Subscriber Attribute to deploy via the Marketing Cloud API.

Required

14.2.46 marketing_cloud_create_user

Description: Creates a new User via the Marketing Cloud SOAP API.

Class: cumulusci.tasks.marketing_cloud.api.CreateUser

Command Syntax

```
$ cci task run marketing_cloud_create_user
```

Options

`--parent_bu_mid PARENTBUMID`

Specify the MID for Parent BU.

Required

`--default_bu_mid DEFAULTBUMID`

Set MID for BU to use as default (can be same as the parent).

Required

`--user_email USEREMAIL`

Set the User's email.

Required

`--user_password USERPASSWORD`

Set the User's password.

Required

`--user_username USERUSERNAME`

Set the User's username. Not the same as their name.

Required

`--external_key EXTERNALKEY`

Set the User's external key.

Optional

`--user_name USERNAME`

Set the User's name. Not the same as their username.

Optional

`--role_id ROLEID`

Assign a Role to the new User, specified as an ID. IDs for system defined roles located here: https://developer.salesforce.com/docs/atlas.en-us.noversion.mc-apis.meta/mc-apis/setting_user_permissions_via_the_web_services_api.htm

Optional

`--activate_if_existing ACTIVATEIFEXISTING`

Activate the user if it already exists in an inactive state. Default: False

Optional

14.2.47 marketing_cloud_get_user_info

Description: Return user info retrieved from the /userinfo endpoint of the Marketing Cloud REST API.

Class: cumulusci.tasks.marketing_cloud.get_user_info.GetUserInfoTask

Command Syntax

```
$ cci task run marketing_cloud_get_user_info
```

14.2.48 marketing_cloud_update_user_role

Description: Assigns a Role to an existing User via the Marketing Cloud SOAP API.

Class: cumulusci.tasks.marketing_cloud.api.UpdateUserRole

Command Syntax

```
$ cci task run marketing_cloud_update_user_role
```

Options

`--account_mid ACCOUNTMID`

Specify the Account MID.

Required

`--user_email USEREMAIL`

Specify the User's email.

Required

`--user_password USERPASSWORD`

Specify the User's password.

Required

`--role_id ROLEID`

Assign a Role to the User, specified as an ID. IDs for system defined roles located here: https://developer.salesforce.com/docs/atlas.en-us.noversion.mc-apis.meta/mc-apis/setting_user_permissions_via_the_web_services_api.htm

Required

`--external_key EXTERNALKEY`

Specify the User's external key.

Optional

`--user_name USERNAME`

Specify the User's name. Not the same as their username.

Optional

14.2.49 deploy_omni_studio_site_settings

Description: Deploys remote site settings needed for OmniStudio.

Class: cumulusci.tasks.vlocity.vlocity.OmniStudioDeployRemoteSiteSettings

Command Syntax

```
$ cci task run deploy_omni_studio_site_settings
```

Options

`--namespace NAMESPACE`

The namespace to inject into RemoteSiteSettings.url values. Defaults to 'omnistudio'.

Optional

14.2.50 deploy_pre

Description: Deploys all metadata bundles under unpackaged/pre/

Class: cumulusci.tasks.salesforce.DeployBundles

Command Syntax

```
$ cci task run deploy_pre
```

Options

`--path PATH`

The path to the parent directory containing the metadata bundles directories

Default: unpackaged/pre

`--unmanaged UNMANAGED`

If True, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Optional

`--namespace_strip NAMESPACESTRIP`

If set, all namespace prefixes for the namespace specified are stripped from files and filenames

Optional

`--check_only CHECKONLY`

If True, performs a test deployment (validation) of components without saving the components in the target org

Optional

`--collision_check COLLISIONCHECK`

If True, performs a collision check with metadata already present in the target org

Optional

`--test_level TESTLEVEL`

Specifies which tests are run as part of a deployment. Valid values: NoTestRun, RunLocalTests, RunAllTestsInOrg, RunSpecifiedTests.

Optional

`--specified_tests SPECIFIEDTESTS`

Comma-separated list of test classes to run upon deployment. Applies only with test_level set to RunSpecifiedTests.

Optional

`--static_resource_path STATICRESOURCEPATH`

The path where decompressed static resources are stored. Any subdirectories found will be zipped and added to the staticresources directory of the build.

Optional

`--namespaced_org NAMESPACEDORG`

If True, the tokens %%%NAMESPACED_ORG%%% and ___NAMESPACED_ORG___ will get replaced with the namespace. The default is false causing those tokens to get stripped and replaced with an empty string. Set this if deploying to a namespaced scratch org or packaging org.

Optional

`--clean_meta_xml CLEANMETAXML`

Defaults to True which strips the <packageVersions/> element from all meta.xml files. The packageVersion element gets added automatically by the target org and is set to whatever version is installed in the org. To disable this, set this option to False

Optional

`--transforms TRANSFORMS`

Apply source transforms before deploying. See the CumulusCI documentation for details on how to specify transforms.

Optional

`--rest_deploy RESTDEPLOY`

If True, deploy metadata using REST API

Optional

14.2.51 deploy_post

Description: Deploys all metadata bundles under unpackaged/post/

Class: cumulusci.tasks.salesforce.DeployBundles

Command Syntax

```
$ cci task run deploy_post
```

Options

`--path PATH`

The path to the parent directory containing the metadata bundles directories

Default: unpackaged/post

`--unmanaged UNMANAGED`

If True, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Optional

`--namespace_strip NAMESPACESTRIP`

If set, all namespace prefixes for the namespace specified are stripped from files and filenames

Optional

`--check_only CHECKONLY`

If True, performs a test deployment (validation) of components without saving the components in the target org

Optional

`--collision_check COLLISIONCHECK`

If True, performs a collision check with metadata already present in the target org

Optional

`--test_level TESTLEVEL`

Specifies which tests are run as part of a deployment. Valid values: NoTestRun, RunLocalTests, RunAllTestsInOrg, RunSpecifiedTests.

Optional

`--specified_tests SPECIFIEDTESTS`

Comma-separated list of test classes to run upon deployment. Applies only with test_level set to RunSpecifiedTests.

Optional

`--static_resource_path STATICRESOURCEPATH`

The path where decompressed static resources are stored. Any subdirectories found will be zipped and added to the staticresources directory of the build.

Optional

`--namespaced_org NAMESPACEDORG`

If True, the tokens %%%NAMESPACED_ORG%%% and ___NAMESPACED_ORG___ will get replaced with the namespace. The default is false causing those tokens to get stripped and replaced with an empty string. Set this if deploying to a namespaced scratch org or packaging org.

Optional

`--clean_meta_xml CLEANMETAXML`

Defaults to True which strips the <packageVersions/> element from all meta.xml files. The packageVersion element gets added automatically by the target org and is set to whatever version is installed in the org. To disable this, set this option to False

Optional

`--transforms TRANSFORMS`

Apply source transforms before deploying. See the CumulusCI documentation for details on how to specify transforms.

Optional

`--rest_deploy RESTDEPLOY`

If True, deploy metadata using REST API

Optional

14.2.52 deploy_qa_config

Description: Deploys configuration for QA.

Class: cumulusci.tasks.salesforce.Deploy

Command Syntax

```
$ cci task run deploy_qa_config
```

Options

`--path PATH`

The path to the metadata source to be deployed

Default: unpackaged/config/qa

`--unmanaged UNMANAGED`

If True, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Optional

`--namespace_strip NAMESPACESTRIP`

If set, all namespace prefixes for the namespace specified are stripped from files and filenames

Optional

`--check_only CHECKONLY`

If True, performs a test deployment (validation) of components without saving the components in the target org

Optional

`--collision_check COLLISIONCHECK`

If True, performs a collision check with metadata already present in the target org

Optional

`--test_level TESTLEVEL`

Specifies which tests are run as part of a deployment. Valid values: NoTestRun, RunLocalTests, RunAllTestsInOrg, RunSpecifiedTests.

Optional

`--specified_tests SPECIFIEDTESTS`

Comma-separated list of test classes to run upon deployment. Applies only with test_level set to RunSpecifiedTests.

Optional

`--static_resource_path STATICRESOURCEPATH`

The path where decompressed static resources are stored. Any subdirectories found will be zipped and added to the staticresources directory of the build.

Optional

`--namespaced_org NAMESPACEDORG`

If True, the tokens %%%NAMESPACED_ORG%%% and ___NAMESPACED_ORG___ will get replaced with the namespace. The default is false causing those tokens to get stripped and replaced with an empty string. Set this if deploying to a namespaced scratch org or packaging org.

Optional

`--clean_meta_xml CLEANMETAXML`

Defaults to True which strips the <packageVersions/> element from all meta.xml files. The packageVersion element gets added automatically by the target org and is set to whatever version is installed in the org. To disable this, set this option to False

Optional

`--transforms TRANSFORMS`

Apply source transforms before deploying. See the CumulusCI documentation for details on how to specify transforms.

Optional

`--rest_deploy RESTDEPLOY`

If True, deploy metadata using REST API

Optional

14.2.53 dx

Description: Execute an arbitrary Salesforce DX command against an org. Use the 'command' option to specify the command, such as 'force:package:install'

Class: cumulusci.tasks.sfdx.SFDXOrgTask

Command Syntax

```
$ cci task run dx
```

Options

`--command COMMAND`

The full command to run with the sfdx cli.

Required

`--extra EXTRA`

Append additional options to the command

Optional

14.2.54 dx_convert_to

Description: Converts src directory metadata format into sfdx format under force-app

Class: cumulusci.tasks.sfdx.SFDXBaseTask

Command Syntax

```
$ cci task run dx_convert_to
```

Options

`--command COMMAND`

The full command to run with the sfdx cli.

Default: force:mdapi:convert -r src

`--extra EXTRA`

Append additional options to the command

Optional

14.2.55 dx_convert_from

Description: Converts force-app directory in sfdx format into metadata format under src

Class: cumulusci.tasks.dx_convert_from.DxConvertFrom

Command Syntax

```
$ cci task run dx_convert_from
```

Options

`--src_dir SRCDIR`

The path to the src directory where converted contents will be stored. Defaults to src/

Default: src

`--extra EXTRA`

Append additional options to the command

Optional

14.2.56 dx_pull

Description: Uses sfdx to pull from a scratch org into the force-app directory

Class: cumulusci.tasks.sfdx.SFDXOrgTask

Command Syntax

```
$ cci task run dx_pull
```

Options

`--command COMMAND`

The full command to run with the sfdx cli.

Default: force:source:pull

`--extra EXTRA`

Append additional options to the command

Optional

14.2.57 dx_push

Description: Uses sfdx to push the force-app directory metadata into a scratch org

Class: cumulusci.tasks.sfdx.SFDXOrgTask

Command Syntax

```
$ cci task run dx_push
```

Options

`--command COMMAND`

The full command to run with the sfdx cli.

Default: force:source:push

`--extra EXTRA`

Append additional options to the command

Optional

14.2.58 enable_einstein_prediction

Description: Enable an Einstein Prediction Builder prediction.

Class: cumulusci.tasks.salesforce.enable_prediction.EnablePrediction

This task updates the state of Einstein Prediction Builder predictions from 'Draft' to 'Enabled' by posting to the Tooling API.

cci task run enable_prediction –org dev -o api_names Example_Prediction_v0

Command Syntax

```
$ cci task run enable_einstein_prediction
```

Options

`--api_names APINAMES`

List of API names of the MLPredictionDefinitions.

Required

`--managed MANAGED`

If False, changes namespace_inject to replace tokens with a blank string

Optional

`--namespaced_org NAMESPACEDORG`

If False, changes namespace_inject to replace namespaced-org tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Optional

14.2.59 ensure_record_types

Description: Ensure that a default Record Type is extant on the given standard sObject (custom objects are not supported). If Record Types are already present, do nothing.

Class: cumulusci.tasks.salesforce.EnsureRecordTypes

Command Syntax

```
$ cci task run ensure_record_types
```

Options

`--record_type_developer_name RECORDTYPEDEVELOPERNAME`

The Developer Name of the Record Type (unique). Must contain only alphanumeric characters and underscores.

Default: Default

`--record_type_label RECORDTYPELABEL`

The Label of the Record Type.

Default: Default

`--sobject SOBJECT`

The sObject on which to deploy the Record Type and optional Business Process.

Required

`--record_type_description RECORDTYPEDESCRIPTION`

The Description of the Record Type. Only uses the first 255 characters.

Optional

`--force_create FORCECREATE`

If true, the Record Type will be created even if a default Record Type already exists on this sObject. Defaults to False.

Optional

14.2.60 execute_anon

Description: Execute anonymous apex via the tooling api.

Class: cumulusci.tasks.apex.anon.AnonymousApexTask

Use the *apex* option to run a string of anonymous Apex. Use the *path* option to run anonymous Apex from a file. Or use both to concatenate the string to the file contents.

Command Syntax

```
$ cci task run execute_anon
```

Options

`--path PATH`

The path to an Apex file to run.

Optional

`--apex APEX`

A string of Apex to run (after the file, if specified).

Optional

`--managed MANAGED`

If True, will insert the project's namespace prefix. Defaults to False or no namespace.

Optional

`--namespaced NAMESPACED`

If True, the tokens %%%NAMESPACED_RT%%% and %%%namespaced%%% will get replaced with the namespace prefix for Record Types.

Optional

`--param1 PARAM1`

Parameter to pass to the Apex. Use as %%%PARAM_1%%% in the Apex code. Defaults to an empty value.

Optional

`--param2 PARAM2`

Parameter to pass to the Apex. Use as %%%PARAM_2%%% in the Apex code. Defaults to an empty value.

Optional

14.2.61 generate_data_dictionary

Description: Create a data dictionary for the project in CSV format.

Class: cumulusci.tasks.datadictionary.GenerateDataDictionary

Generate a data dictionary for the project by walking all GitHub releases. The data dictionary is output as two CSV files. One, in *object_path*, includes

- Object Label
- Object API Name
- Object Description
- Version Introduced

with one row per packaged object.

The other, in *field_path*, includes

- Object Label
- Object API Name
- Field Label
- Field API Name
- Field Type
- Valid Picklist Values
- Help Text
- Field Description
- Version Introduced
- Version Picklist Values Last Changed
- Version Help Text Last Changed

Both MDAPI and SFDX format releases are supported.

Command Syntax

```
$ cci task run generate_data_dictionary
```

Options

`--object_path OBJECTPATH`

Path to a CSV file to contain an sObject-level data dictionary.

Optional

`--field_path FIELDPATH`

Path to a CSV file to contain an field-level data dictionary.

Optional

`--include_dependencies INCLUDEDEPENDENCIES`

Process all of the GitHub dependencies of this project and include their schema in the data dictionary.

Optional

`--additional_dependencies ADDITIONALDEPENDENCIES`

Include schema from additional GitHub repositories that are not explicit dependencies of this project to build a unified data dictionary. Specify as a list of dicts as in project__dependencies in cumulusci.yml. Note: only repository dependencies are supported.

Optional

`--include_prerelease INCLUDEPRERELEASE`

Treat the current branch as containing prerelease schema, and included it as Prerelease in the data dictionary. NOTE: this option cannot be used with *additional_dependencies* or *include_dependencies*.

Optional

`--include_protected_schema INCLUDEPROTECTEDSCHEMA`

Include Custom Objects, Custom Settings, and Custom Metadata Types that are marked as Protected. Defaults to False.

Optional

14.2.62 generate_and_load_from_yaml

Description: None

Class: cumulusci.tasks.bulkdata.generate_and_load_data_from_yaml.GenerateAndLoadDataFromYaml

Command Syntax

```
$ cci task run generate_and_load_from_yaml
```

Options

`--data_generation_task DATAGENERATIONTASK`

Fully qualified class path of a task to generate the data. Look at cumulusci.tasks.bulkdata.tests.dummy_data_factory to learn how to write them.

Required

`--generator_yaml GENERATORYAML`

A Snowfakery recipe file to use

Required

`--num_records NUMRECORDS`

Target number of records. You will get at least this many records, but may get more. The recipe will always execute to completion, so if it creates 3 records per execution and you ask for 5, you will get 6.

Optional

`--num_records_tablename NUMRECORDSTABLENAME`

A string representing which table determines when the recipe execution is done.

Optional

`--batch_size BATCHSIZE`

How many records to create and load at a time.

Optional

`--data_generation_options DATAGENERATIONOPTIONS`

Options to pass to the data generator.

Optional

`--vars VARS`

Pass values to override options in the format VAR1:foo,VAR2:bar

Optional

`--replace_database REPLACEDATABASE`

Confirmation that it is okay to delete the data in database_url

Optional

`--working_directory WORKINGDIRECTORY`

Default path for temporary / working files

Optional

`--database_url DATABASEURL`

A path to put a copy of the sqlite database (for debugging)

Optional

`--mapping MAPPING`

A mapping YAML file to use

Optional

`--start_step STARTSTEP`

If specified, skip steps before this one in the mapping

Optional

`--sql_path SQLPATH`

If specified, a database will be created from an SQL script at the provided path

Optional

`--ignore_row_errors IGNOREROWERRORS`

If True, allow the load to continue even if individual rows fail to load.

Optional

`--reset_oids RESETOIDS`

If True (the default), and the _sf_ids tables exist, reset them before continuing.

Optional

`--bulk_mode BULKMODE`

Set to Serial to force serial mode on all jobs. Parallel is the default.

Optional

`--inject_namespaces INJECTNAMESPACES`

If True, the package namespace prefix will be automatically added to (or removed from) objects and fields based on the name used in the org. Defaults to True.

Optional

`--drop_missing_schema DROPMISSINGSCHEMA`

Set to True to skip any missing objects or fields instead of stopping with an error.

Optional

`--set_recently_viewed SETRECENTLYVIEWED`

By default, the first 1000 records inserted via the Bulk API will be set as recently viewed. If fewer than 1000 records are inserted, existing objects of the same type being inserted will also be set as recently viewed.

Optional

`--org_shape_match_only ORGSHAPEMATCHONLY`

When True, all path options are ignored and only a dataset matching the org shape name will be loaded. Defaults to False.

Optional

`--enable_rollback ENABLEROLLBACK`

When True, performs rollback operation incase of error. Defaults to False

Optional

`--plugin_options PLUGINOPTIONS`

Pass values to override plugin options in the format VAR1:foo,VAR2:bar

Optional

`--generate_mapping_file GENERATEMAPPINGFILE`

A path to put a mapping file inferred from the generator_yaml

Optional

`--continuation_file CONTINUATIONFILE`

YAML file generated by Snowfakery representing next steps for data generation

Optional

`--generate_continuation_file GENERATECONTINUATIONFILE`

Path for Snowfakery to put its next continuation file

Optional

`--loading_rules LOADINGRULES`

Path to .load.yml file containing rules to use to load the file. Defaults to <recipename>.load.yml . Multiple files can be comma separated.

Optional

14.2.63 get_installed_packages

Description: Retrieves a list of the currently installed managed package namespaces and their versions

Class: cumulusci.tasks.preflight.packages.GetInstalledPackages

Command Syntax

```
$ cci task run get_installed_packages
```

14.2.64 get_available_licenses

Description: Retrieves a list of the currently available license definition keys

Class: cumulusci.tasks.preflight.licenses.GetAvailableLicenses

Command Syntax

```
$ cci task run get_available_licenses
```

14.2.65 get_available_permission_set_licenses

Description: Retrieves a list of the currently available Permission Set License definition keys

Class: cumulusci.tasks.preflight.licenses.GetAvailablePermissionSetLicenses

Command Syntax

```
$ cci task run get_available_permission_set_licenses
```

14.2.66 get_assigned_permission_sets

Description: Retrieves a list of the developer names of any permission sets assigned to the running user.

Class: cumulusci.tasks.preflight.permsets.GetPermissionSetAssignments

Command Syntax

```
$ cci task run get_assigned_permission_sets
```

14.2.67 get_assigned_permission_set_licenses

Description: Retrieves a list of the developer names of any Permission Set Licenses assigned to the running user.

Class: cumulusci.tasks.preflight.licenses.GetPermissionLicenseSetAssignments

Command Syntax

```
$ cci task run get_assigned_permission_set_licenses
```

14.2.68 get_available_permission_sets

Description: Retrieves a list of the currently available Permission Sets

Class: cumulusci.tasks.preflight.licenses.GetAvailablePermissionSets

Command Syntax

```
$ cci task run get_available_permission_sets
```

14.2.69 get_existing_record_types

Description: Retrieves all Record Types in the org as a dict, with sObject names as keys and lists of Developer Names as values.

Class: cumulusci.tasks.preflight.recordtypes.CheckSObjectRecordTypes

Command Syntax

```
$ cci task run get_existing_record_types
```

14.2.70 get_existing_sites

Description: Retrieves a list of any existing Experience Cloud site names in the org.

Class: cumulusci.tasks.salesforce.ListCommunities

Lists Communities for the current org via the Connect API.

Command Syntax

```
$ cci task run get_existing_sites
```

14.2.71 github_parent_pr_notes

Description: Merges the description of a child pull request to the respective parent's pull request (if one exists).

Class: cumulusci.tasks.release_notes.task.ParentPullRequestNotes

Aggregate change notes from child pull request(s) to a corresponding parent pull request.

When given the branch_name option, this task will: (1) check if the base branch of the corresponding pull request starts with the feature branch prefix and if so (2) attempt to query for a pull request corresponding to this parent feature branch. (3) if a pull request isn't found, the task exits and no actions are taken.

If the build_notes_label is present on the pull request, then all notes from the child pull request are aggregated into the parent pull request. if the build_notes_label is not detected on the parent pull request then a link to the child pull request is placed under the "Unaggregated Pull Requests" header.

If you have a pull request on branch feature/myFeature that you would like to rebuild notes for use the branch_name and force options:

cci task run github_parent_pr_notes –branch-name feature/myFeature –force True

Command Syntax

```
$ cci task run github_parent_pr_notes
```

Options

`--branch_name BRANCHNAME`

Name of branch to check for parent status, and if so, reaggregate change notes from child branches.

Required

`--build_notes_label BUILDNOTESLABEL`

Name of the label that indicates that change notes on parent pull requests should be reaggregated when a child branch pull request is created.

Required

`--force FORCE`

force rebuilding of change notes from child branches in the given branch.

Optional

14.2.72 github_clone_tag

Description: Clones a github tag under a new name.

Class: cumulusci.tasks.github.CloneTag

Command Syntax

```
$ cci task run github_clone_tag
```

Options

`--src_tag SRCTAG`

The source tag to clone. Ex: beta/1.0-Beta_2

Required

`--tag TAG`

The new tag to create by cloning the src tag. Ex: release/1.0

Required

14.2.73 github_automerge_main

Description: Merges the latest commit on the main branch into all open feature branches

Class: cumulusci.tasks.github.MergeBranch

Merges the most recent commit on the current branch into other branches depending on the value of source_branch.

If source_branch is a branch that does not start with the specified branch_prefix, then the commit will be merged to all branches that begin with branch_prefix and are not themselves child branches (i.e. branches don't contain '__' in their name).

If source_branch begins with branch_prefix, then the commit is merged to all child branches of source_branch.

Command Syntax

```
$ cci task run github_automerge_main
```

Options

`--commit COMMIT`

The commit to merge into feature branches. Defaults to the current head commit.

Optional

`--source_branch SOURCEBRANCH`

The source branch to merge from. Defaults to project__git__default_branch.

Optional

`--branch_prefix BRANCHPREFIX`

A list of prefixes of branches that should receive the merge. Defaults to project__git__prefix_feature

Optional

`--skip_future_releases SKIPFUTURERELEASES`

If true, then exclude branches that start with the branch prefix if they are not for the lowest release number. Defaults to True.

Optional

`--update_future_releases UPDATEFUTURERELEASES`

If true, then include release branches that are not the lowest release number even if they are not child branches. Defaults to False.

Optional

`--create_pull_request_on_conflict CREATEPULLREQUESTONCONFLICT`

If true, then create a pull request when a merge conflict arises. Defaults to True.

Optional

14.2.74 github_automerge_feature

Description: Merges the latest commit on a source branch to all child branches.

Class: cumulusci.tasks.github.MergeBranch

Merges the most recent commit on the current branch into other branches depending on the value of source_branch.

If source_branch is a branch that does not start with the specified branch_prefix, then the commit will be merged to all branches that begin with branch_prefix and are not themselves child branches (i.e. branches don't contain '__' in their name).

If source_branch begins with branch_prefix, then the commit is merged to all child branches of source_branch.

Command Syntax

```
$ cci task run github_automerge_feature
```

Options

`--commit COMMIT`

The commit to merge into feature branches. Defaults to the current head commit.

Optional

`--source_branch SOURCEBRANCH`

The source branch to merge from. Defaults to project__git__default_branch.

Default: $project_config.repo_branch

`--branch_prefix BRANCHPREFIX`

A list of prefixes of branches that should receive the merge. Defaults to project__git__prefix_feature

Optional

`--skip_future_releases SKIPFUTURERELEASES`

If true, then exclude branches that start with the branch prefix if they are not for the lowest release number. Defaults to True.

Optional

`--update_future_releases UPDATEFUTURERELEASES`

If true, then include release branches that are not the lowest release number even if they are not child branches. Defaults to False.

Optional

`--create_pull_request_on_conflict CREATEPULLREQUESTONCONFLICT`

If true, then create a pull request when a merge conflict arises. Defaults to True.

Optional

14.2.75 github_copy_subtree

Description: Copies one or more subtrees from the project repository for a given release to a target repository, with the option to include release notes.

Class: cumulusci.tasks.github.publish.PublishSubtree

Command Syntax

```
$ cci task run github_copy_subtree
```

Options

`--repo_url REPOURL`

The url to the public repo

Required

`--branch BRANCH`

The branch to update in the target repo

Required

`--version VERSION`

(Deprecated >= 3.42.0) Only the values of 'latest' and 'latest_beta' are acceptable. Required if 'ref' or 'tag_name' is not set. This will override tag_name if it is provided.

Optional

`--tag_name TAGNAME`

The name of the tag that should be associated with this release. Values of 'latest' and 'latest_beta' are also allowed. Required if 'ref' or 'version' is not set.

Optional

`--ref REF`

The git reference to publish. Takes precedence over 'version' and 'tag_name'. Required if 'tag_name' is not set.

Optional

`--include INCLUDE`

A list of paths from repo root to include. Directories must end with a trailing slash.

Optional

`--renames RENAMES`

A list of paths to rename in the target repo, given as *local: target:* pairs.

Optional

`--create_release CREATERELEASE`

If True, create a release in the public repo. Defaults to False

Optional

`--release_body RELEASEBODY`

If True, the entire release body will be published to the public repo. Defaults to False

Optional

`--dry_run DRYRUN`

If True, skip creating Github data. Defaults to False

Optional

14.2.76 github_package_data

Description: Look up 2gp package dependencies for a version id recorded in a commit status.

Class: cumulusci.tasks.github.commit_status.GetPackageDataFromCommitStatus

Command Syntax

```
$ cci task run github_package_data
```

Options

`--context CONTEXT`

Name of the commit status context

Required

`--version_id VERSIONID`

Package version id

Optional

14.2.77 github_pull_requests

Description: Lists open pull requests in project Github repository

Class: cumulusci.tasks.github.PullRequests

Command Syntax

```
$ cci task run github_pull_requests
```

14.2.78 github_release

Description: Creates a Github release for a given managed package version number

Class: cumulusci.tasks.github.CreateRelease

Command Syntax

```
$ cci task run github_release
```

Options

`--version VERSION`

The managed package version number. Ex: 1.2

Required

`--package_type PACKAGETYPE`

The package type of the project (either 1GP or 2GP)

Required

`--tag_prefix TAGPREFIX`

The prefix to use for the release tag created in github.

Required

`--version_id VERSIONID`

The SubscriberPackageVersionId (04t) associated with this release.

Optional

`--message MESSAGE`

The message to attach to the created git tag

Optional

`--release_content RELEASECONTENT`

The content to include as the release body. Note: github_release_notes will overwrite this content, if used.

Optional

`--dependencies DEPENDENCIES`

List of dependencies to record in the tag message.

Optional

`--commit COMMIT`

Override the commit used to create the release. Defaults to the current local HEAD commit

Optional

`--resolution_strategy RESOLUTIONSTRATEGY`

The name of a sequence of resolution_strategy (from project__dependency_resolutions) to apply to dynamic dependencies. Defaults to 'production'.

Optional

14.2.79 gather_release_notes

Description: Generates release notes by getting the latest release of each repository

Class: cumulusci.tasks.release_notes.task.AllGithubReleaseNotes

Command Syntax

```
$ cci task run gather_release_notes
```

Options

`--repos REPOS`

The list of owner, repo key pairs for which to generate release notes. Ex: 'owner': Salesforce-Foundation 'repo': 'NPSP'

Required

14.2.80 github_release_notes

Description: Generates release notes by parsing pull request bodies of merged pull requests between two tags

Class: cumulusci.tasks.release_notes.task.GithubReleaseNotes

Command Syntax

```
$ cci task run github_release_notes
```

Options

`--tag TAG`

The tag to generate release notes for. Ex: release/1.2

Required

`--last_tag LASTTAG`

Override the last release tag. This is useful to generate release notes if you skipped one or more releases.

Optional

`--link_pr LINKPR`

If True, insert link to source pull request at end of each line.

Optional

`--publish PUBLISH`

Publish to GitHub release if True (default=False)

Optional

`--include_empty INCLUDEEMPTY`

If True, include links to PRs that have no release notes (default=False)

Optional

`--version_id VERSIONID`

The package version id used by the InstallLinksParser to add install urls

Optional

`--trial_info TRIALINFO`

If True, Includes trialforce template text for this product.

Optional

`--sandbox_date SANDBOXDATE`

The date of the sandbox release in ISO format (Will default to None)

Optional

`--production_date PRODUCTIONDATE`

The date of the production release in ISO format (Will default to None)

Optional

14.2.81 github_release_report

Description: Parses GitHub release notes to report various info

Class: cumulusci.tasks.github.ReleaseReport

Command Syntax

```
$ cci task run github_release_report
```

Options

`--date_start DATESTART`

Filter out releases created before this date (YYYY-MM-DD)

Optional

`--date_end DATEEND`

Filter out releases created after this date (YYYY-MM-DD)

Optional

`--include_beta INCLUDEBETA`

Include beta releases in report [default=False]

Optional

`--print PRINT`

Print info to screen as JSON [default=False]

Optional

14.2.82 install_managed

Description: Install the latest managed production release

Class: cumulusci.tasks.salesforce.InstallPackageVersion

Command Syntax

```
$ cci task run install_managed
```

Options

`--namespace NAMESPACE`

The namespace of the package to install. Defaults to project__package__namespace

Required

`--version VERSION`

The version of the package to install. "latest" and "latest_beta" can be used to trigger lookup via Github Releases on the repository.

Default: latest

`--name NAME`

The name of the package to install. Defaults to project__package__name_managed

Optional

`--version_number VERSIONNUMBER`

If installing a package using an 04t version Id, display this version number to the user and in logs. Has no effect otherwise.

Optional

`--activateRSS ACTIVATERSS`

Deprecated. Use activate_remote_site_settings instead.

Optional

`--retries RETRIES`

Number of retries (default=5)

Optional

`--retry_interval RETRYINTERVAL`

Number of seconds to wait before the next retry (default=5),

Optional

`--retry_interval_add RETRYINTERVALADD`

Number of seconds to add before each retry (default=30),

Optional

`--interactive INTERACTIVE`

If True, stop after resolving the package version and output the package Id that will be installed. Defaults to False.

Optional

`--base_package_url_format BASEPACKAGEURLFORMAT`

If *interactive* is set to True, display package Ids using a format string ({} will be replaced with the package Id).

Optional

`--security_type SECURITYTYPE`

Which Profiles to install packages for (FULL = all profiles, NONE = admins only, PUSH = no profiles, CUSTOM = custom profiles). Defaults to FULL.

Optional

`--name_conflict_resolution NAMECONFLICTRESOLUTION`

Specify how to resolve name conflicts when installing an Unlocked Package. Available values are Block and RenameMetadata. Defaults to Block.

Optional

`--activate_remote_site_settings ACTIVATEREMOTESITESETTINGS`

Activate Remote Site Settings when installing a package. Defaults to True.

Optional

`--password PASSWORD`

The installation key for the managed package.

Optional

`--apex_compile_type APEXCOMPILETYPE`

For Unlocked Packages only, whether to compile Apex in the package only (*package*) or in the whole org (*all*). *all* is the default behavior.

Optional

`--upgrade_type UPGRADETYPE`

For Unlocked Package upgrades only, whether to deprecate removed components (*deprecate-only*), delete them (*delete-only*), or delete and deprecate based on safety (*mixed*). *mixed* is the default behavior.

Optional

14.2.83 install_managed_beta

Description: Installs the latest managed beta release

Class: cumulusci.tasks.salesforce.InstallPackageVersion

Command Syntax

```
$ cci task run install_managed_beta
```

Options

`--namespace NAMESPACE`

The namespace of the package to install. Defaults to project__package__namespace

Required

`--version VERSION`

The version of the package to install. "latest" and "latest_beta" can be used to trigger lookup via Github Releases on the repository.

Default: latest_beta

`--name NAME`

The name of the package to install. Defaults to project__package__name_managed

Optional

`--version_number VERSIONNUMBER`

If installing a package using an 04t version Id, display this version number to the user and in logs. Has no effect otherwise.

Optional

`--activateRSS ACTIVATERSS`

Deprecated. Use activate_remote_site_settings instead.

Optional

`--retries RETRIES`

Number of retries (default=5)

Optional

`--retry_interval RETRYINTERVAL`

Number of seconds to wait before the next retry (default=5),

Optional

`--retry_interval_add RETRYINTERVALADD`

Number of seconds to add before each retry (default=30),

Optional

`--interactive INTERACTIVE`

If True, stop after resolving the package version and output the package Id that will be installed. Defaults to False.

Optional

`--base_package_url_format BASEPACKAGEURLFORMAT`

If *interactive* is set to True, display package Ids using a format string ({} will be replaced with the package Id).

Optional

`--security_type SECURITYTYPE`

Which Profiles to install packages for (FULL = all profiles, NONE = admins only, PUSH = no profiles, CUSTOM = custom profiles). Defaults to FULL.

Optional

`--name_conflict_resolution NAMECONFLICTRESOLUTION`

Specify how to resolve name conflicts when installing an Unlocked Package. Available values are Block and RenameMetadata. Defaults to Block.

Optional

`--activate_remote_site_settings ACTIVATEREMOTESITESETTINGS`

Activate Remote Site Settings when installing a package. Defaults to True.

Optional

`--password PASSWORD`

The installation key for the managed package.

Optional

`--apex_compile_type APEXCOMPILETYPE`

For Unlocked Packages only, whether to compile Apex in the package only (*package*) or in the whole org (*all*). *all* is the default behavior.

Optional

`--upgrade_type UPGRADETYPE`

For Unlocked Package upgrades only, whether to deprecate removed components (*deprecate-only*), delete them (*delete-only*), or delete and deprecate based on safety (*mixed*). *mixed* is the default behavior.

Optional

14.2.84 list_communities

Description: Lists Communities for the current org using the Connect API.

Class: cumulusci.tasks.salesforce.ListCommunities

Lists Communities for the current org via the Connect API.

Command Syntax

```
$ cci task run list_communities
```

14.2.85 list_community_templates

Description: Prints the Community Templates available to the current org

Class: cumulusci.tasks.salesforce.ListCommunityTemplates

Lists Salesforce Community templates available for the current org via the Connect API.

Command Syntax

```
$ cci task run list_community_templates
```

14.2.86 list_metadata_types

Description: Prints the metadata types in a project

Class: cumulusci.tasks.util.ListMetadataTypes

Command Syntax

```
$ cci task run list_metadata_types
```

Options

`--package_xml PACKAGEXML`

The project package.xml file. Defaults to <project_root>/src/package.xml

Optional

14.2.87 list_nonsource_trackable_components

Description: List the components of non source trackable Metadata types.

Class: cumulusci.tasks.salesforce.nonsourcetracking.ListComponents

Command Syntax

```
$ cci task run list_nonsource_trackable_components
```

Options

`--api_version APIVERSION`

Override the API version used to list metadatatypes

Optional

`--metadata_types METADATATYPES`

A comma-separated list of metadata types.

Optional

14.2.88 list_nonsource_trackable_metadatatypes

Description: Returns non source trackable metadata types supported by org

Class: cumulusci.tasks.salesforce.nonsourcetracking.ListNonSourceTrackable

Command Syntax

```
$ cci task run list_nonsource_trackable_metadatatypes
```

Options

`--api_version APIVERSION`

Override the API version used to list metadatatypes

Optional

14.2.89 meta_xml_apiversion

Description: Set the API version in `*meta.xml` files

Class: cumulusci.tasks.metaxml.UpdateApi

Command Syntax

```
$ cci task run meta_xml_apiversion
```

Options

`--version VERSION`

API version number e.g. 37.0

Required

`--dir DIR`

Base directory to search for `*-meta.xml` files

Optional

14.2.90 meta_xml_dependencies

Description: Set the version for dependent packages

Class: cumulusci.tasks.metaxml.UpdateDependencies

Command Syntax

```
$ cci task run meta_xml_dependencies
```

Options

`--dir DIR`

Base directory to search for `*-meta.xml` files

Optional

14.2.91 metadeploy_publish

Description: Publish a release to the MetaDeploy web installer

Class: cumulusci.tasks.metadeploy.Publish

Command Syntax

```
$ cci task run metadeploy_publish
```

Options

`--tag TAG`

Name of the git tag to publish

Optional

`--commit COMMIT`

Commit hash to publish

Optional

`--plan PLAN`

Name of the plan(s) to publish. This refers to the *plans* section of cumulusci.yml. By default, all plans will be published.

Optional

`--dry_run DRYRUN`

If True, print steps without publishing.

Optional

`--publish PUBLISH`

If True, set is_listed to True on the version. Default: False

Optional

`--labels_path LABELSPATH`

Path to a folder containing translations.

Optional

14.2.92 org_settings

Description: Apply org settings from a scratch org definition file or dict

Class: cumulusci.tasks.salesforce.org_settings.DeployOrgSettings

Command Syntax

```
$ cci task run org_settings
```

Options

`--definition_file DEFINITIONFILE`

sfdx scratch org definition file

Optional

`--settings SETTINGS`

A dict of settings to apply

Optional

`--object_settings OBJECTSETTINGS`

A dict of objectSettings to apply

Optional

`--api_version APIVERSION`

API version used to deploy the settings

Optional

14.2.93 promote_package_version

Description: Promote a 2gp package so that it can be installed in a production org

Class: cumulusci.tasks.salesforce.promote_package_version.PromotePackageVersion

Promote a Second Generation package (managed or unlocked).
Lists any 1GP dependencies that are detected, as well as any dependency packages that have not been promoted. Once promoted, the 2GP package can be installed into production orgs.

Command Syntax

```
$ cci task run promote_package_version
```

Options

`--version_id VERSIONID`

The SubscriberPackageVersion (04t) Id for the target package.

Optional

`--promote_dependencies PROMOTEDEPENDENCIES`

Automatically promote any unpromoted versions of dependency 2GP packages that are detected.

Optional

`--install_key INSTALLKEY`

Install key for package. Default is no install key.

Optional

14.2.94 publish_community

Description: Publishes a Community in the target org using the Connect API

Class: cumulusci.tasks.salesforce.PublishCommunity

Publish a Salesforce Community via the Connect API. Warning: This does not work with the Community Template 'VF Template' due to an existing bug in the API.

Command Syntax

```
$ cci task run publish_community
```

Options

`--name NAME`

The name of the Community to publish.

Optional

`--community_id COMMUNITYID`

The id of the Community to publish.

Optional

14.2.95 push_all

Description: Schedules a push upgrade of a package version to all subscribers

Class: cumulusci.tasks.push.tasks.SchedulePushOrgQuery

Command Syntax

```
$ cci task run push_all
```

Options

`--version VERSION`

The managed package version to push

Required

`--subscriber_where SUBSCRIBERWHERE`

A SOQL style WHERE clause for filtering PackageSubscriber objects. Ex: OrgType = 'Sandbox'

Optional

`--min_version MINVERSION`

If set, no subscriber with a version lower than min_version will be selected for push

Optional

`--metadata_package_id METADATAPACKAGEID`

The MetadataPackageId (ID prefix *033*) to push.

Optional

`--namespace NAMESPACE`

The managed package namespace to push. Defaults to project__package__namespace.

Optional

`--start_time STARTTIME`

Set the start time (ISO-8601) to queue a future push. (Ex: 2021-01-01T06:00Z or 2021-01-01T06:00-08:00) Times with no timezone will be interpreted as UTC.

Optional

`--dry_run DRYRUN`

If True, log how many orgs were selected but skip creating a PackagePushRequest. Defaults to False

Optional

14.2.96 push_list

Description: Schedules a push upgrade of a package version to all orgs listed in the specified file

Class: cumulusci.tasks.push.tasks.SchedulePushOrgList

Command Syntax

```
$ cci task run push_list
```

Options

`--csv CSV`

The path to a CSV file to read.

Optional

`--csv_field_name CSVFIELDNAME`

The CSV field name that contains organization IDs. Defaults to 'OrganizationID'

Optional

`--orgs ORGS`

The path to a file containing one OrgID per line.

Optional

`--version VERSION`

The managed package version to push

Optional

`--version_id VERSIONID`

The MetadataPackageVersionId (ID prefix *04t*) to push

Optional

`--metadata_package_id METADATAPACKAGEID`

The MetadataPackageId (ID prefix *033*) to push.

Optional

`--namespace NAMESPACE`

The managed package namespace to push. Defaults to project__package__namespace.

Optional

`--start_time STARTTIME`

Set the start time (ISO-8601) to queue a future push. (Ex: 2021-01-01T06:00Z or 2021-01-01T06:00-08:00) Times with no timezone will be interpreted as UTC.

Optional

`--batch_size BATCHSIZE`

Break pull requests into batches of this many orgs. Defaults to 200.

Optional

14.2.97 push_qa

Description: Schedules a push upgrade of a package version to all orgs listed in push/orgs_qa.txt

Class: cumulusci.tasks.push.tasks.SchedulePushOrgList

Command Syntax

```
$ cci task run push_qa
```

Options

`--csv CSV`

The path to a CSV file to read.

Optional

`--csv_field_name CSVFIELDNAME`

The CSV field name that contains organization IDs. Defaults to 'OrganizationID'

Optional

`--orgs ORGS`

The path to a file containing one OrgID per line.

Default: push/orgs_qa.txt

`--version VERSION`

The managed package version to push

Optional

`--version_id VERSIONID`

The MetadataPackageVersionId (ID prefix *04t*) to push

Optional

`--metadata_package_id METADATAPACKAGEID`

The MetadataPackageId (ID prefix *033*) to push.

Optional

`--namespace NAMESPACE`

The managed package namespace to push. Defaults to project__package__namespace.

Optional

`--start_time STARTTIME`

Set the start time (ISO-8601) to queue a future push. (Ex: 2021-01-01T06:00Z or 2021-01-01T06:00-08:00) Times with no timezone will be interpreted as UTC.

Optional

`--batch_size BATCHSIZE`

Break pull requests into batches of this many orgs. Defaults to 200.

Optional

14.2.98 push_sandbox

Description: Schedules a push upgrade of a package version to sandbox orgs

Class: cumulusci.tasks.push.tasks.SchedulePushOrgQuery

Command Syntax

```
$ cci task run push_sandbox
```

Options

`--version VERSION`

The managed package version to push

Required

`--subscriber_where SUBSCRIBERWHERE`

A SOQL style WHERE clause for filtering PackageSubscriber objects. Ex: OrgType = 'Sandbox'

Default: OrgType = 'Sandbox'

`--min_version MINVERSION`

If set, no subscriber with a version lower than min_version will be selected for push

Optional

`--metadata_package_id METADATAPACKAGEID`

The MetadataPackageId (ID prefix *033*) to push.

Optional

`--namespace NAMESPACE`

The managed package namespace to push. Defaults to project__package__namespace.

Optional

`--start_time STARTTIME`

Set the start time (ISO-8601) to queue a future push. (Ex: 2021-01-01T06:00Z or 2021-01-01T06:00-08:00) Times with no timezone will be interpreted as UTC.

Optional

`--dry_run DRYRUN`

If True, log how many orgs were selected but skip creating a PackagePushRequest. Defaults to False

Optional

14.2.99 push_trial

Description: Schedules a push upgrade of a package version to Trialforce Template orgs listed in push/orgs_trial.txt

Class: cumulusci.tasks.push.tasks.SchedulePushOrgList

Command Syntax

```
$ cci task run push_trial
```

Options

`--csv CSV`

The path to a CSV file to read.

Optional

`--csv_field_name CSVFIELDNAME`

The CSV field name that contains organization IDs. Defaults to 'OrganizationID'

Optional

`--orgs ORGS`

The path to a file containing one OrgID per line.

Default: push/orgs_trial.txt

`--version VERSION`

The managed package version to push

Optional

`--version_id VERSIONID`

The MetadataPackageVersionId (ID prefix *04t*) to push

Optional

`--metadata_package_id METADATAPACKAGEID`

The MetadataPackageId (ID prefix *033*) to push.

Optional

`--namespace NAMESPACE`

The managed package namespace to push. Defaults to project__package__namespace.

Optional

`--start_time STARTTIME`

Set the start time (ISO-8601) to queue a future push. (Ex: 2021-01-01T06:00Z or 2021-01-01T06:00-08:00) Times with no timezone will be interpreted as UTC.

Optional

`--batch_size BATCHSIZE`

Break pull requests into batches of this many orgs. Defaults to 200.

Optional

14.2.100 push_failure_report

Description: Produce a CSV report of the failed and otherwise anomalous push jobs.

Class: cumulusci.tasks.push.pushfails.ReportPushFailures

Command Syntax

```
$ cci task run push_failure_report
```

Options

`--request_id REQUESTID`

PackagePushRequest ID for the request you need to report on.

Required

`--result_file RESULTFILE`

Path to write a CSV file with the results. Defaults to 'push_fails.csv'.

Optional

`--ignore_errors IGNOREERRORS`

List of ErrorTitle and ErrorType values to omit from the report

Default: ['Salesforce Subscription Expired', 'Package Uninstalled']

14.2.101 query

Description: Queries the connected org

Class: cumulusci.tasks.salesforce.SOQLQuery

Command Syntax

```
$ cci task run query
```

Options

`--object OBJECT`

The object to query

Required

`--query QUERY`

A valid bulk SOQL query for the object

Required

`--result_file RESULTFILE`

The name of the csv file to write the results to

Required

14.2.102 retrieve_packaged

Description: Retrieves the packaged metadata from the org

Class: cumulusci.tasks.salesforce.RetrievePackaged

Command Syntax

```
$ cci task run retrieve_packaged
```

Options

`--path PATH`

The path to write the retrieved metadata

Default: packaged

`--package PACKAGE`

The package name to retrieve. Defaults to project__package__name

Required

`--namespace_strip NAMESPACESTRIP`

If set, all namespace prefixes for the namespace specified are stripped from files and filenames

Optional

`--namespace_tokenize NAMESPACETOKENIZE`

If set, all namespace prefixes for the namespace specified are replaced with tokens for use with namespace_inject

Optional

`--namespaced_org NAMESPACEDORG`

If True, the tokens %%%NAMESPACED_ORG%%% and ___NAMESPACED_ORG___ will get replaced with the namespace. The default is false causing those tokens to get stripped and replaced with an empty string. Set this if deploying to a namespaced scratch org or packaging org.

Optional

`--api_version APIVERSION`

Override the default api version for the retrieve. Defaults to project__package__api_version

Optional

14.2.103 describe_metadatatypes

Description: Retrieves the metadata types supported by the org based on the api version

Class: cumulusci.tasks.salesforce.DescribeMetadataTypes

Command Syntax

```
$ cci task run describe_metadatatypes
```

Options

`--api_version APIVERSION`

Override the API version used to list metadatatypes

Optional

14.2.104 retrieve_src

Description: Retrieves the packaged metadata into the src directory

Class: cumulusci.tasks.salesforce.RetrievePackaged

Command Syntax

```
$ cci task run retrieve_src
```

Options

`--path PATH`

The path to write the retrieved metadata

Default: src

`--package PACKAGE`

The package name to retrieve. Defaults to project__package__name

Required

`--namespace_strip NAMESPACESTRIP`

If set, all namespace prefixes for the namespace specified are stripped from files and filenames

Optional

`--namespace_tokenize NAMESPACETOKENIZE`

If set, all namespace prefixes for the namespace specified are replaced with tokens for use with namespace_inject

Optional

`--namespaced_org NAMESPACEDORG`

If True, the tokens %%%NAMESPACED_ORG%%% and ___NAMESPACED_ORG___ will get replaced with the namespace. The default is false causing those tokens to get stripped and replaced with an empty string. Set this if deploying to a namespaced scratch org or packaging org.

Optional

`--api_version APIVERSION`

Override the default api version for the retrieve. Defaults to project__package__api_version

Optional

14.2.105 retrieve_unpackaged

Description: Retrieve the contents of a package.xml file.

Class: cumulusci.tasks.salesforce.RetrieveUnpackaged

Command Syntax

```
$ cci task run retrieve_unpackaged
```

Options

`--path PATH`

The path to write the retrieved metadata

Required

`--package_xml PACKAGEXML`

The path to a package.xml manifest to use for the retrieve.

Required

`--namespace_strip NAMESPACESTRIP`

If set, all namespace prefixes for the namespace specified are stripped from files and filenames

Optional

`--namespace_tokenize NAMESPACETOKENIZE`

If set, all namespace prefixes for the namespace specified are replaced with tokens for use with namespace_inject

Optional

`--namespaced_org NAMESPACEDORG`

If True, the tokens %%%NAMESPACED_ORG%%% and ___NAMESPACED_ORG___ will get replaced with the namespace. The default is false causing those tokens to get stripped and replaced with an empty string. Set this if deploying to a namespaced scratch org or packaging org.

Optional

`--api_version APIVERSION`

Override the default api version for the retrieve. Defaults to project__package__api_version

Optional

14.2.106 list_changes

Description: List the changes from a scratch org

Class: cumulusci.tasks.salesforce.sourcetracking.ListChanges

Command Syntax

```
$ cci task run list_changes
```

Options

`--include INCLUDE`

A comma-separated list of strings. Components will be included if one of these strings is part of either the metadata type or name. Example: `-o include CustomField,Admin` matches both `CustomField: Favorite_Color__c` and `Profile: Admin`

Optional

`--types TYPES`

A comma-separated list of metadata types to include.

Optional

`--exclude EXCLUDE`

Exclude changed components matching this string.

Optional

`--snapshot SNAPSHOT`

If True, all matching items will be set to be ignored at their current revision number. This will exclude them from the results unless a new edit is made.

Optional

14.2.107 retrieve_changes

Description: Retrieve changed components from a scratch org

Class: cumulusci.tasks.salesforce.sourcetracking.RetrieveChanges

Command Syntax

```
$ cci task run retrieve_changes
```

Options

`--include INCLUDE`

A comma-separated list of strings. Components will be included if one of these strings is part of either the metadata type or name. Example: `-o include CustomField,Admin` matches both `CustomField: Favorite_Color__c` and `Profile: Admin`

Optional

`--types TYPES`

A comma-separated list of metadata types to include.

Optional

`--exclude EXCLUDE`

Exclude changed components matching this string.

Optional

`--snapshot SNAPSHOT`

If True, all matching items will be set to be ignored at their current revision number. This will exclude them from the results unless a new edit is made.

Optional

`--path PATH`

The path to write the retrieved metadata

Optional

`--api_version APIVERSION`

Override the default api version for the retrieve. Defaults to project__package__api_version

Optional

`--retrieve_complete_profile RETRIEVECOMPLETEPROFILE`

If set to True, will use RetrieveProfile to retrieve the complete profile. Default is set to False

Optional

`--namespace_tokenize NAMESPACETOKENIZE`

If set, all namespace prefixes for the namespace specified are replaced with tokens for use with namespace_inject

Optional

14.2.108 retrieve_nonsource_trackable

Description: Retrieves the non source trackable components filtered

Class: cumulusci.tasks.salesforce.nonsourcetracking.RetrieveComponents

Command Syntax

```
$ cci task run retrieve_nonsource_trackable
```

Options

`--api_version APIVERSION`

Override the API version used to list metadatatypes

Optional

`--metadata_types METADATATYPES`

A comma-separated list of metadata types.

Optional

`--path PATH`

The path to write the retrieved metadata

Optional

`--include INCLUDE`

Components will be included if one of these namesis part of either the metadata type or name. Example: `-o include CustomField,Admin` matches both `CustomField: Favorite_Color__c` and `Profile: Admin`

Optional

`--exclude EXCLUDE`

Exclude components matching this name.

Optional

`--namespace_tokenize NAMESPACETOKENIZE`

If set, all namespace prefixes for the namespace specified are replaced with tokens for use with namespace_inject

Optional

14.2.109 retrieve_qa_config

Description: Retrieves the current changes in the scratch org into unpackaged/config/qa

Class: cumulusci.tasks.salesforce.sourcetracking.RetrieveChanges

Command Syntax

```
$ cci task run retrieve_qa_config
```

Options

`--include INCLUDE`

A comma-separated list of strings. Components will be included if one of these strings is part of either the metadata type or name. Example: `-o include CustomField,Admin` matches both `CustomField: Favorite_Color__c` and `Profile: Admin`

Optional

`--types TYPES`

A comma-separated list of metadata types to include.

Optional

`--exclude EXCLUDE`

Exclude changed components matching this string.

Optional

`--snapshot SNAPSHOT`

If True, all matching items will be set to be ignored at their current revision number. This will exclude them from the results unless a new edit is made.

Optional

`--path PATH`

The path to write the retrieved metadata

Default: unpackaged/config/qa

`--api_version APIVERSION`

Override the default api version for the retrieve. Defaults to project__package__api_version

Optional

`--retrieve_complete_profile RETRIEVECOMPLETEPROFILE`

If set to True, will use RetrieveProfile to retrieve the complete profile. Default is set to False

Optional

`--namespace_tokenize NAMESPACETOKENIZE`

If set, all namespace prefixes for the namespace specified are replaced with tokens for use with namespace_inject

Default: $project_config.project__package__namespace

14.2.110 set_field_help_text

Description: Sets specified fields' Help Text values.

Class: cumulusci.tasks.metadata_etl.help_text.SetFieldHelpText

Command Syntax

```
$ cci task run set_field_help_text
```

Options

`--fields FIELDS`

List of object fields to affect, in Object__c.Field__c form.

Required

`--overwrite OVERWRITE`

If set to True, overwrite any differing Help Text found on the field. By default, Help Text is set only if it is blank.

Optional

`--api_names APINAMES`

List of API names of entities to affect

Optional

`--managed MANAGED`

If False, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Default: $project_config.project__package__namespace

`--api_version APIVERSION`

Metadata API version to use, if not project__package__api_version.

Optional

14.2.111 snapshot_changes

Description: Tell SFDX source tracking to ignore previous changes in a scratch org

Class: cumulusci.tasks.salesforce.sourcetracking.SnapshotChanges

Command Syntax

```
$ cci task run snapshot_changes
```

14.2.112 snowfakery

Description: Generate and load data from a Snowfakery recipe

Class: cumulusci.tasks.bulkdata.snowfakery.Snowfakery

Do a data load with Snowfakery.

All options are optional.

The most commonly supplied options are *recipe* and one of the three *run_until_...* options.

Command Syntax

```
$ cci task run snowfakery
```

Options

`--recipe RECIPE`

Path to a Snowfakery recipe file determining what data to generate and load.

Default: datasets/recipe.yml

`--run_until_records_in_org RUNUNTILRECORDSINORG`

<sobject>:<count>

Run the recipe repeatedly until the count of <sobject> in the org matches the given <count>.

For example, *–run_until_records_in_org Account:50_000* means:

Count the Account records in the org. Let's say the number is 20,000. Thus, we must run the recipe over and over again until we generate 30,000 new Account records. If the recipe also generates e.g.Contacts, Opportunities or whatever else, it generates the appropriate number of them to match.

Underscores are allowed but optional in big numbers: 2000000 is the same as 2_000_000.

Optional

`--run_until_records_loaded RUNUNTILRECORDSLOADED`

<sobject>:<count>

Run the recipe repeatedly until the number of records of <sobject> uploaded in this task execution matches <count>.

For example, *–run_until_records_loaded Account:50_000* means:

Run the recipe over and over again until we generate 50_000 new Account records. If the recipe also generates e.g. Contacts, Opportunities or whatever else, it generates the appropriate number of them to match.

Optional

`--run_until_recipe_repeated RUNUNTILRECIPEREPEATED`

Run the recipe <count> times,
no matter what data is already in the org.

For example, *–run_until_recipe_repeated 50_000* means run the recipe 50_000 times.

Optional

`--working_directory WORKINGDIRECTORY`

Path for temporary / working files

Optional

`--loading_rules LOADINGRULES`

Path to .load.yml file containing rules to use to load the file. Defaults to *<recipename>.load.yml*. Multiple files can be comma separated.

Optional

`--recipe_options RECIPEOPTIONS`

Pass values to override options in the format VAR1:foo,VAR2:bar

Example: –recipe_options weight:10,color:purple

Optional

`--bulk_mode BULKMODE`

Set to Serial to serialize everything: data generation, data loading, data ingestion through bulk API. Parallel is the default.

Optional

`--drop_missing_schema DROPMISSINGSCHEMA`

Set to True to skip any missing objects or fields instead of stopping with an error.

Optional

`--num_processes NUMPROCESSES`

Number of data generating processes. Defaults to matching the number of CPUs.

Optional

`--ignore_row_errors IGNOREROWERRORS`

Boolean: should we continue loading even after running into row errors? Defaults to False.

Optional

14.2.113 revert_managed_src

Description: Reverts the changes from create_managed_src

Class: cumulusci.tasks.metadata.managed_src.RevertManagedSrc

Command Syntax

```
$ cci task run revert_managed_src
```

Options

`--path PATH`

The path containing metadata to process for managed deployment

Default: src

`--revert_path REVERTPATH`

The path to copy the original metadata to for the revert call

Default: src.orig

14.2.114 revert_unmanaged_ee_src

Description: Reverts the changes from create_unmanaged_ee_src

Class: cumulusci.tasks.metadata.ee_src.RevertUnmanagedEESrc

Command Syntax

```
$ cci task run revert_unmanaged_ee_src
```

Options

`--path PATH`

The path containing metadata to process for managed deployment

Default: src

`--revert_path REVERTPATH`

The path to copy the original metadata to for the revert call

Default: src.orig

14.2.115 robot

Description: Runs a Robot Framework test from a .robot file

Class: cumulusci.tasks.robotframework.Robot

Runs Robot test cases using a browser, if necessary and stores its results in a directory. The path to the directory can be retrieved from the `robot_outputdir` return variable. Command Syntax ————————————————————

```
$ cci task run robot
```

Options

`--suites SUITES`

Paths to test case files/directories to be executed similarly as when running the robot command on the command line. Defaults to "tests" to run all tests in the tests directory

Default: tests

`--test TEST`

Run only tests matching name patterns. Can be comma separated and use robot wildcards like *

Optional

`--include INCLUDE`

Includes tests with a given tag pattern

Optional

`--exclude EXCLUDE`

Excludes tests with a given tag pattern. Excluded tests will not appear in the logs and reports.

Optional

`--skip SKIP`

Do not run tests with the given tag pattern. Similar to 'exclude', but skipped tests will appear in the logs and reports with the status of SKIP.

Optional

`--vars VARS`

Pass values to override variables in the format VAR1:foo,VAR2:bar

Optional

`--xunit XUNIT`

Set an XUnit format output file for test results

Optional

`--sources SOURCES`

List of sources defined in cumulusci.yml that are required by the robot task.

Optional

`--options OPTIONS`

A dictionary of options to robot.run method. In simple cases this can be specified on the comand line using name:value,name:value syntax. More complex cases can be specified in cumulusci.yml using YAML dictionary syntax.

Optional

`--name NAME`

Sets the name of the top level test suite

Optional

`--pdb PDB`

If true, run the Python debugger when tests fail.

Optional

`--verbose VERBOSE`

If true, log each keyword as it runs.

Optional

`--robot_debug ROBOTDEBUG`

If true, enable the *breakpoint* keyword to enable the robot debugger

Optional

`--ordering ORDERING`

Path to a file which defines the order in which parallel tests are run. This maps directly to the pabot option of the same name. It is ignored unless the processes argument is set to 2 or greater.

Optional

`--processes PROCESSES`

experimental Number of processes to use for running tests in parallel. If this value is set to a number larger than 1 the tests will run using the open source tool pabot rather than robotframework. For example, -o parallel 2 will run half of the tests in one process and half in another. If not provided, all tests will run in a single process using the standard robot test runner. See https://pabot.org/ for more information on pabot.

Optional

`--testlevelsplit TESTLEVELSPLIT`

If true, split parallel execution at the test level rather than the suite level. This option is ignored unless the processes option is set to 2 or greater. Note: this option requires a boolean value even though the pabot option of the same name does not.

Optional

14.2.116 robot_libdoc

Description: Generates documentation for project keyword files

Class: cumulusci.tasks.robotframework.RobotLibDoc

Command Syntax

```
$ cci task run robot_libdoc
```

Options

`--path PATH`

The path to one or more keyword libraries to be documented. The path can be single a python file, a .robot file, a python module (eg: cumulusci.robotframework.Salesforce) or a comma separated list of any of those. Glob patterns are supported for filenames (eg: `robot/SAL/doc/*PageObject.py`). The order of the files will be preserved in the generated documentation. The result of pattern expansion will be sorted

Required

`--output OUTPUT`

The output file where the documentation will be written. Normally an HTML file will be generated. If the filename ends with '.csv' then a csv file will be generated instead.

Default: Keywords.html

`--title TITLE`

A string to use as the title of the generated output

Default: $project_config.project__package__name

`--preview PREVIEW`

If True, automatically open a window to view the generated data when the task is successful

Optional

14.2.117 robot_testdoc

Description: Generates html documentation of your Robot test suite and writes to tests/test_suite.

Class: cumulusci.tasks.robotframework.RobotTestDoc

Command Syntax

```
$ cci task run robot_testdoc
```

Options

`--path PATH`

The path containing .robot test files

Default: tests

`--output OUTPUT`

The output html file where the documentation will be written

Default: tests/test_suites.html

14.2.118 run_tests

Description: Runs all apex tests

Class: cumulusci.tasks.apex.testrunner.RunApexTests

Command Syntax

```
$ cci task run run_tests
```

Options

`--test_name_match TESTNAMEMATCH`

Pattern to find Apex test classes to run ("%" is wildcard). Defaults to project__test__name_match from project config. Comma-separated list for multiple patterns.

Optional

`--test_name_exclude TESTNAMEEXCLUDE`

Query to find Apex test classes to exclude ("%" is wildcard). Defaults to project__test__name_exclude from project config. Comma-separated list for multiple patterns.

Optional

`--namespace NAMESPACE`

Salesforce project namespace. Defaults to project__package__namespace

Optional

`--managed MANAGED`

If True, search for tests in the namespace only. Defaults to False

Optional

`--poll_interval POLLINTERVAL`

Seconds to wait between polling for Apex test results.

Optional

`--junit_output JUNITOUTPUT`

File name for JUnit output. Defaults to test_results.xml

Optional

`--json_output JSONOUTPUT`

File name for json output. Defaults to test_results.json

Optional

`--retry_failures RETRYFAILURES`

A list of regular expression patterns to match against test failures. If failures match, the failing tests are retried in serial mode.

Optional

`--retry_always RETRYALWAYS`

By default, all failures must match retry_failures to perform a retry. Set retry_always to True to retry all failed tests if any failure matches.

Optional

`--required_org_code_coverage_percent PERCENTAGE`

Require at least X percent code coverage across the org following the test run.

Optional

`--required_per_class_code_coverage_percent REQUIREDPERCLASSCODECOVERAGEPERCENT`

Require at least X percent code coverage for every class in the org.

Optional

`--verbose VERBOSE`

By default, only failures get detailed output. Set verbose to True to see all passed test methods.

Optional

`--test_suite_names TESTSUITENAMES`

Accepts a comma-separated list of test suite names. Only runs test classes that are part of the test suites specified.

Optional

14.2.119 set_duplicate_rule_status

Description: Sets the active status of Duplicate Rules.

Class: cumulusci.tasks.metadata_etl.SetDuplicateRuleStatus

Command Syntax

```
$ cci task run set_duplicate_rule_status
```

Options

`--active ACTIVE`

Boolean value, set the Duplicate Rule to either active or inactive

Required

`--api_names APINAMES`

List of API names of entities to affect

Optional

`--managed MANAGED`

If False, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Default: $project_config.project__package__namespace

`--api_version APIVERSION`

Metadata API version to use, if not project__package__api_version.

Optional

14.2.120 set_object_settings

Description: Enable and disable object level settings on standard and custom objects

Class: cumulusci.tasks.metadata_etl.SetObjectSettings

Command Syntax

```
$ cci task run set_object_settings
```

Options

`--enable ENABLE`

Array of object settings to enable. Uses the setting name. Available values: Activities, BulkApi, Feeds, History, Licensing, Reports, Search, Sharing, StreamingApi

Optional

`--disable DISABLE`

Array of object settings to disable. Uses the setting name. Available values: Activities, BulkApi, Feeds, History, Licensing, Reports, Search, Sharing, StreamingApi

Optional

`--api_names APINAMES`

List of API names of entities to affect

Optional

`--managed MANAGED`

If False, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Optional

`--api_version APIVERSION`

Metadata API version to use, if not project__package__api_version.

Optional

14.2.121 set_organization_wide_defaults

Description: Sets the Organization-Wide Defaults for specific sObjects, and waits for sharing recalculation to complete.

Class: cumulusci.tasks.metadata_etl.SetOrgWideDefaults

Command Syntax

```
$ cci task run set_organization_wide_defaults
```

Options

`--org_wide_defaults ORGWIDEDEFAULTS`

The target Organization-Wide Defaults, organized as a list with each element containing the keys api_name, internal_sharing_model, and external_sharing_model. This option must be specified in YAML configuration and will not work if passed via the CLI. NOTE: you must have External Sharing Model turned on in Sharing Settings to use the latter feature.

Required

`--timeout TIMEOUT`

The max amount of time to wait in seconds

Optional

`--api_names APINAMES`

List of API names of entities to affect

Optional

`--managed MANAGED`

If False, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Default: $project_config.project__package__namespace

`--api_version APIVERSION`

Metadata API version to use, if not project__package__api_version.

Optional

14.2.122 strip_unwanted_components

Description: Removes components from src folder which are not mentioned in given package.xml file

Class: cumulusci.tasks.metadata.package.RemoveUnwantedComponents

Command Syntax

```
$ cci task run strip_unwanted_components
```

Options

`--path PATH`

The path to a folder of metadata to strip the components

Default: src

`--package_xml PACKAGEXML`

The path to package xml file to refer

Default: src/package.xml

14.2.123 uninstall_managed

Description: Uninstalls the managed version of the package

Class: cumulusci.tasks.salesforce.UninstallPackage

Command Syntax

```
$ cci task run uninstall_managed
```

Options

`--namespace NAMESPACE`

The namespace of the package to uninstall. Defaults to project__package__namespace

Required

`--purge_on_delete PURGEONDELETE`

Sets the purgeOnDelete option for the deployment. Defaults to True

Required

14.2.124 uninstall_packaged

Description: Uninstalls all deleteable metadata in the package in the target org

Class: cumulusci.tasks.salesforce.UninstallPackaged

Command Syntax

```
$ cci task run uninstall_packaged
```

Options

`--package PACKAGE`

> The package name to uninstall. All metadata from the package will be retrieved and a custom destructiveChanges.xml package will be constructed and deployed to delete all deleteable metadata from the package. Defaults to project__package__name

Required

`--purge_on_delete PURGEONDELETE`

> Sets the purgeOnDelete option for the deployment. Defaults to True

Required

`--dry_run DRYRUN`

> Perform a dry run of the operation without actually deleting any components, and display the components that would be deleted.

Optional

14.2.125 uninstall_packaged_incremental

Description: Deletes any metadata from the package in the target org not in the local workspace

Class: cumulusci.tasks.salesforce.UninstallPackagedIncremental

Command Syntax

```
$ cci task run uninstall_packaged_incremental
```

Options

`--path PATH`

> The local path to compare to the retrieved packaged metadata from the org. Defaults to src.

Required

`--package PACKAGE`

> The package name to uninstall. All metadata from the package will be retrieved and a custom destructiveChanges.xml package will be constructed and deployed to delete all deleteable metadata from the package. Defaults to project__package__name.

Required

`--purge_on_delete PURGEONDELETE`

Sets the purgeOnDelete option for the deployment. Defaults to True.

Required

`--ignore IGNORE`

Components to ignore in the org and not try to delete. Mapping of component type to a list of member names.

Optional

`--ignore_types IGNORETYPES`

List of component types to ignore in the org and not try to delete. Defaults to ['RecordType', 'CustomObjectTranslation'].

Optional

`--dry_run DRYRUN`

Perform a dry run of the operation without actually deleting any components, and display the components that would be deleted.

Optional

14.2.126 uninstall_src

Description: Uninstalls all metadata in the local src directory

Class: cumulusci.tasks.salesforce.UninstallLocal

Command Syntax

```
$ cci task run uninstall_src
```

Options

`--path PATH`

The path to the metadata source to be deployed

Default: src

`--unmanaged UNMANAGED`

If True, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Optional

`--namespace_strip NAMESPACESTRIP`

If set, all namespace prefixes for the namespace specified are stripped from files and filenames

Optional

`--check_only CHECKONLY`

If True, performs a test deployment (validation) of components without saving the components in the target org

Optional

`--collision_check COLLISIONCHECK`

If True, performs a collision check with metadata already present in the target org

Optional

`--test_level TESTLEVEL`

Specifies which tests are run as part of a deployment. Valid values: NoTestRun, RunLocalTests, RunAllTestsInOrg, RunSpecifiedTests.

Optional

`--specified_tests SPECIFIEDTESTS`

Comma-separated list of test classes to run upon deployment. Applies only with test_level set to RunSpecifiedTests.

Optional

`--static_resource_path STATICRESOURCEPATH`

The path where decompressed static resources are stored. Any subdirectories found will be zipped and added to the staticresources directory of the build.

Optional

`--namespaced_org NAMESPACEDORG`

If True, the tokens %%%NAMESPACED_ORG%%% and ___NAMESPACED_ORG___ will get replaced with the namespace. The default is false causing those tokens to get stripped and replaced with an empty string. Set this if deploying to a namespaced scratch org or packaging org.

Optional

`--clean_meta_xml CLEANMETAXML`

Defaults to True which strips the <packageVersions/> element from all meta.xml files. The packageVersion element gets added automatically by the target org and is set to whatever version is installed in the org. To disable this, set this option to False

Optional

`--transforms TRANSFORMS`

Apply source transforms before deploying. See the CumulusCI documentation for details on how to specify transforms.

Optional

`--rest_deploy RESTDEPLOY`

If True, deploy metadata using REST API

Optional

`--purge_on_delete PURGEONDELETE`

Sets the purgeOnDelete option for the deployment. Defaults to True

Optional

`--dry_run DRYRUN`

Perform a dry run of the operation without actually deleting any components, and display the components that would be deleted.

Optional

14.2.127 uninstall_pre

Description: Uninstalls the unpackaged/pre bundles

Class: cumulusci.tasks.salesforce.UninstallLocalBundles

Command Syntax

```
$ cci task run uninstall_pre
```

Options

`--path PATH`

The path to the metadata source to be deployed

Default: unpackaged/pre

`--unmanaged UNMANAGED`

If True, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Optional

`--namespace_strip NAMESPACESTRIP`

If set, all namespace prefixes for the namespace specified are stripped from files and filenames

Optional

`--check_only CHECKONLY`

If True, performs a test deployment (validation) of components without saving the components in the target org

Optional

`--collision_check COLLISIONCHECK`

If True, performs a collision check with metadata already present in the target org

Optional

`--test_level TESTLEVEL`

Specifies which tests are run as part of a deployment. Valid values: NoTestRun, RunLocalTests, RunAllTestsInOrg, RunSpecifiedTests.

Optional

`--specified_tests SPECIFIEDTESTS`

Comma-separated list of test classes to run upon deployment. Applies only with test_level set to RunSpecifiedTests.

Optional

`--static_resource_path STATICRESOURCEPATH`

The path where decompressed static resources are stored. Any subdirectories found will be zipped and added to the staticresources directory of the build.

Optional

`--namespaced_org NAMESPACEDORG`

If True, the tokens %%%NAMESPACED_ORG%%% and ___NAMESPACED_ORG___ will get replaced with the namespace. The default is false causing those tokens to get stripped and replaced with an empty string. Set this if deploying to a namespaced scratch org or packaging org.

Optional

`--clean_meta_xml CLEANMETAXML`

Defaults to True which strips the <packageVersions/> element from all meta.xml files. The packageVersion element gets added automatically by the target org and is set to whatever version is installed in the org. To disable this, set this option to False

Optional

`--transforms TRANSFORMS`

Apply source transforms before deploying. See the CumulusCI documentation for details on how to specify transforms.

Optional

`--rest_deploy RESTDEPLOY`

If True, deploy metadata using REST API

Optional

`--purge_on_delete PURGEONDELETE`

Sets the purgeOnDelete option for the deployment. Defaults to True

Optional

`--dry_run DRYRUN`

Perform a dry run of the operation without actually deleting any components, and display the components that would be deleted.

Optional

14.2.128 uninstall_post

Description: Uninstalls the unpackaged/post bundles

Class: cumulusci.tasks.salesforce.UninstallLocalNamespacedBundles

Command Syntax

```
$ cci task run uninstall_post
```

Options

`--path PATH`

The path to a directory containing the metadata bundles (subdirectories) to uninstall

Default: unpackaged/post

`--filename_token FILENAMETOKEN`

The path to the parent directory containing the metadata bundles directories

Default: ___NAMESPACE___

`--purge_on_delete PURGEONDELETE`

Sets the purgeOnDelete option for the deployment. Defaults to True

Required

`--managed MANAGED`

If True, will insert the actual namespace prefix. Defaults to False or no namespace

Optional

`--namespace NAMESPACE`

The namespace to replace the token with if in managed mode. Defaults to project__package__namespace

Optional

14.2.129 unschedule_apex

Description: Unschedule all scheduled apex jobs (CronTriggers).

Class: cumulusci.tasks.apex.anon.AnonymousApexTask

Use the *apex* option to run a string of anonymous Apex. Use the *path* option to run anonymous Apex from a file. Or use both to concatenate the string to the file contents.

Command Syntax

```
$ cci task run unschedule_apex
```

Options

`--path PATH`

The path to an Apex file to run.

Optional

`--apex APEX`

A string of Apex to run (after the file, if specified).

Default: for (CronTrigger t : [SELECT Id FROM CronTrigger]) { System.abortJob(t.Id); }

`--managed MANAGED`

If True, will insert the project's namespace prefix. Defaults to False or no namespace.

Optional

`--namespaced NAMESPACED`

If True, the tokens %%%NAMESPACED_RT%%% and %%%namespaced%%% will get replaced with the namespace prefix for Record Types.

Optional

`--param1 PARAM1`

Parameter to pass to the Apex. Use as %%%PARAM_1%%% in the Apex code. Defaults to an empty value.

Optional

`--param2 PARAM2`

Parameter to pass to the Apex. Use as %%%PARAM_2%%% in the Apex code. Defaults to an empty value.

Optional

14.2.130 update_admin_profile

Description: Retrieves, edits, and redeploys the Admin.profile with full FLS perms for all objects/fields

Class: cumulusci.tasks.salesforce.ProfileGrantAllAccess

Command Syntax

```
$ cci task run update_admin_profile
```

Options

`--package_xml PACKAGEXML`

Override the default package.xml file for retrieving the Admin.profile and all objects and classes that need to be included by providing a path to your custom package.xml

Optional

`--record_types RECORDTYPES`

A list of dictionaries containing the required key *record_type* with a value specifying the record type in format <object>.<developer_name>. Record type names can use the token strings {managed} and {namespaced_org} for namespace prefix injection as needed. By default, all listed record types will be set to visible and not default. Use the additional keys *visible*, *default*, and *person_account_default* set to true/false to override. Page Layout Support: If you are using the Page Layouts feature, you can specify the *page_layout* key with the layout name to use for the record type. If not specified, the default page layout will be used. NOTE: Setting record_types is only supported in cumulusci.yml, command line override is not supported.

Optional

`--managed MANAGED`

If True, uses the namespace prefix where appropriate. Use if running against an org with the managed package installed. Defaults to False

Optional

`--namespaced_org NAMESPACEDORG`

If True, attempts to prefix all unmanaged metadata references with the namespace prefix for deployment to the packaging org or a namespaced scratch org. Defaults to False

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix. Defaults to project__package__namespace

Optional

`--profile_name PROFILENAME`

Name of the Profile to target for updates (deprecated; use api_names to target multiple profiles).

Optional

`--include_packaged_objects INCLUDEPACKAGEDOBJECTS`

Automatically include objects from all installed managed packages. Defaults to True in projects that require CumulusCI 3.9.0 and greater that don't use a custom package.xml, otherwise False.

Optional

`--api_names APINAMES`

List of API names of Profiles to affect

Optional

14.2.131 update_dependencies

Description: Installs all dependencies in project__dependencies into the target org

Class: cumulusci.tasks.salesforce.UpdateDependencies

Command Syntax

```
$ cci task run update_dependencies
```

Options

`--dependencies DEPENDENCIES`

List of dependencies to update. Defaults to project__dependencies. Each dependency is a dict with either 'github' set to a github repository URL or 'namespace' set to a Salesforce package namespace. GitHub dependencies may include 'tag' to install a particular git ref. Package dependencies may include 'version' to install a particular version.

Optional

`--ignore_dependencies IGNOREDEPENDENCIES`

List of dependencies to be ignored, including if they are present as transitive dependencies. Dependencies can be specified using the 'github' or 'namespace' keys (all other keys are not used). Note that this can cause installations to fail if required prerequisites are not available.

Optional

`--purge_on_delete PURGEONDELETE`

Sets the purgeOnDelete option for the deployment. Defaults to True

Optional

`--include_beta INCLUDEBETA`

Install the most recent release, even if beta. Defaults to False. This option is only supported for scratch orgs, to avoid installing a package that can't be upgraded in persistent orgs.

Optional

`--allow_newer ALLOWNEWER`

Deprecated. This option has no effect.

Optional

`--prefer_2gp_from_release_branch PREFER2GPFROMRELEASEBRANCH`

If True and this build is on a release branch (feature/NNN, where NNN is an integer), or a child branch of a release branch, resolve GitHub managed package dependencies to 2GP builds present on a matching release branch on the dependency.

Optional

`--resolution_strategy RESOLUTIONSTRATEGY`

The name of a sequence of resolution_strategy (from project__dependency_resolutions) to apply to dynamic dependencies.

Optional

`--packages_only PACKAGESONLY`

Install only packaged dependencies. Ignore all unmanaged metadata. Defaults to False.

Optional

`--interactive INTERACTIVE`

If True, stop after identifying all dependencies and output the package Ids that will be installed. Defaults to False.

Optional

`--base_package_url_format BASEPACKAGEURLFORMAT`

If *interactive* is set to True, display package Ids using a format string ({} will be replaced with the package Id).

Optional

`--security_type SECURITYTYPE`

Which Profiles to install packages for (FULL = all profiles, NONE = admins only, PUSH = no profiles, CUSTOM = custom profiles). Defaults to FULL.

Optional

`--name_conflict_resolution NAMECONFLICTRESOLUTION`

Specify how to resolve name conflicts when installing an Unlocked Package. Available values are Block and RenameMetadata. Defaults to Block.

Optional

`--activate_remote_site_settings ACTIVATEREMOTESITESETTINGS`

Activate Remote Site Settings when installing a package. Defaults to True.

Optional

`--apex_compile_type APEXCOMPILETYPE`

For Unlocked Packages only, whether to compile Apex in the package only (*package*) or in the whole org (*all*). *all* is the default behavior.

Optional

`--upgrade_type UPGRADETYPE`

For Unlocked Package upgrades only, whether to deprecate removed components (*deprecate-only*), delete them (*delete-only*), or delete and deprecate based on safety (*mixed*). *mixed* is the default behavior.

Optional

14.2.132 update_metadata_first_child_text

Description: Updates the text of the first child of Metadata with matching tag. Adds a child for tag if it does not exist.

Class: cumulusci.tasks.metadata_etl.UpdateMetadataFirstChildTextTask

Metadata ETL task to update a single child element's text within metadata XML.

If the child doesn't exist, the child is created and appended to the Metadata. Furthermore, the `value` option is namespaced injected if the task is properly configured.

Example: Assign a Custom Object's Compact Layout

Researching CustomObject in the Metadata API documentation or even retrieving the CustomObject's Metadata for inspection, we see the `compactLayoutAssignment` Field. We want to assign a specific Compact Layout for our Custom Object, so we write the following CumulusCI task in our project's `cumulusci.yml`.

```
tasks:
    assign_compact_layout:
        class_path: cumulusci.tasks.metadata_etl.UpdateMetadataFirstChildTextTask
        options:
            managed: False
            namespace_inject: $project_config.project__package__namespace
            entity: CustomObject
            api_names: OurCustomObject__c
            tag: compactLayoutAssignment
            value: "%%%NAMESPACE%%%DifferentCompactLayout"
            # We include a namespace token so it's easy to use this task in a managed␣
↪context.
```

Suppose the original CustomObject metadata XML looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    ...
    <label>Our Custom Object</label>
    <compactLayoutAssignment>OriginalCompactLayout</compactLayoutAssignment>
    ...
</CustomObject>
```

After running `cci task run assign_compact_layout`, the CustomObject metadata XML is deployed as:

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    ...
    <label>Our Custom Object</label>
    <compactLayoutAssignment>DifferentCompactLayout</compactLayoutAssignment>
    ...
</CustomObject>
```

Command Syntax

```
$ cci task run update_metadata_first_child_text
```

Options

`--metadata_type METADATATYPE`

Metadata Type

Required

`--tag TAG`

Targeted tag. The text of the first instance of this tag within the metadata entity will be updated.

Required

`--value VALUE`

Desired value to set for the targeted tag's text. This value is namespace-injected.

Required

`--api_names APINAMES`

List of API names of entities to affect

Optional

`--managed MANAGED`

If False, changes namespace_inject to replace tokens with a blank string

Optional

`--namespace_inject NAMESPACEINJECT`

If set, the namespace tokens in files and filenames are replaced with the namespace's prefix

Default: $project_config.project__package__namespace

`--api_version APIVERSION`

Metadata API version to use, if not project__package__api_version.

Optional

14.2.133 update_package_xml

Description: Updates src/package.xml with metadata in src/

Class: cumulusci.tasks.metadata.package.UpdatePackageXml

Command Syntax

```
$ cci task run update_package_xml
```

Options

`--path PATH`

The path to a folder of metadata to build the package.xml from

Default: src

`--output OUTPUT`

The output file, defaults to <path>/package.xml

Optional

`--package_name PACKAGENAME`

If set, overrides the package name inserted into the <fullName> element

Optional

`--managed MANAGED`

If True, generate a package.xml for deployment to the managed package packaging org

Optional

`--delete DELETE`

If True, generate a package.xml for use as a destructiveChanges.xml file for deleting metadata

Optional

`--install_class INSTALLCLASS`

Specify post install class file to be used. Defaults to what is set in project config

Optional

`--uninstall_class UNINSTALLCLASS`

Specify post uninstall class file to be used. Defaults to what is set in project config

Optional

14.2.134 upload_beta

Description: Uploads a beta release of the metadata currently in the packaging org

Class: cumulusci.tasks.salesforce.PackageUpload

Command Syntax

```
$ cci task run upload_beta
```

Options

`--name NAME`

The name of the package version.

Required

`--production PRODUCTION`

If True, uploads a production release. Defaults to uploading a beta

Optional

`--description DESCRIPTION`

A description of the package and what this version contains.

Optional

`--password PASSWORD`

An optional password for sharing the package privately with anyone who has the password. Don't enter a password if you want to make the package available to anyone on AppExchange and share your package publicly.

Optional

`--post_install_url POSTINSTALLURL`

The fully-qualified URL of the post-installation instructions. Instructions are shown as a link after installation and are available from the package detail view.

Optional

`--release_notes_url RELEASENOTESURL`

The fully-qualified URL of the package release notes. Release notes are shown as a link during the installation process and are available from the package detail view after installation.

Optional

`--namespace NAMESPACE`

The namespace of the package. Defaults to project__package__namespace

Optional

`--resolution_strategy RESOLUTIONSTRATEGY`

The name of a sequence of resolution_strategy (from project__dependency_resolutions) to apply to dynamic dependencies. Defaults to 'production'.

Optional

`--major_version MAJORVERSION`

The desired major version number for the uploaded package. Defaults to latest major version.

Optional

`--minor_version MINORVERSION`

The desired minor version number for the uploaded package. Defaults to next available minor version for the current major version.

Optional

14.2.135 upload_production

Description: Uploads a production release of the metadata currently in the packaging org

Class: cumulusci.tasks.salesforce.PackageUpload

Command Syntax

```
$ cci task run upload_production
```

Options

`--name NAME`

The name of the package version.

Default: Release

`--production PRODUCTION`

If True, uploads a production release. Defaults to uploading a beta

Default: True

`--description DESCRIPTION`

A description of the package and what this version contains.

Optional

`--password PASSWORD`

An optional password for sharing the package privately with anyone who has the password. Don't enter a password if you want to make the package available to anyone on AppExchange and share your package publicly.

Optional

`--post_install_url POSTINSTALLURL`

The fully-qualified URL of the post-installation instructions. Instructions are shown as a link after installation and are available from the package detail view.

Optional

`--release_notes_url RELEASENOTESURL`

The fully-qualified URL of the package release notes. Release notes are shown as a link during the installation process and are available from the package detail view after installation.

Optional

`--namespace NAMESPACE`

The namespace of the package. Defaults to project__package__namespace

Optional

`--resolution_strategy RESOLUTIONSTRATEGY`

The name of a sequence of resolution_strategy (from project__dependency_resolutions) to apply to dynamic dependencies. Defaults to 'production'.

Optional

`--major_version MAJORVERSION`

The desired major version number for the uploaded package. Defaults to latest major version.

Optional

`--minor_version MINORVERSION`

The desired minor version number for the uploaded package. Defaults to next available minor version for the current major version.

Optional

14.2.136 upload_user_profile_photo

Description: Uploads a profile photo for a specified or default User.

Class: cumulusci.tasks.salesforce.users.photos.UploadProfilePhoto

Uploads a profile photo for a specified or default User.

Examples

Upload a profile photo for the default user.

```
tasks:
    upload_profile_photo_default:
        group: Internal storytelling data
        class_path: cumulusci.tasks.salesforce.users.UploadProfilePhoto
        description: Uploads a profile photo for the default user.
        options:
            photo: storytelling/photos/default.png
```

Upload a profile photo for a user whose Alias equals `grace` or `walker`, is active, and created today.

```
tasks:
    upload_profile_photo_grace:
        group: Internal storytelling data
        class_path: cumulusci.tasks.salesforce.users.UploadProfilePhoto
        description: Uploads a profile photo for Grace.
        options:
            photo: storytelling/photos/grace.png
            where: (Alias = 'grace' OR Alias = 'walker') AND IsActive = true AND␣
↪CreatedDate = TODAY
```

Command Syntax

```
$ cci task run upload_user_profile_photo
```

Options

`--photo PHOTO`

Path to user's profile photo.

Required

`--where WHERE`

WHERE clause used querying for which User to upload the profile photo for.

- No need to prefix with `WHERE`
- The SOQL query must return one and only one User record.
- If no "where" is supplied, uploads the photo for the org's default User.

Optional

14.2.137 util_sleep

Description: Sleeps for N seconds

Class: cumulusci.tasks.util.Sleep

Command Syntax

```
$ cci task run util_sleep
```

Options

`--seconds SECONDS`

The number of seconds to sleep

Default: 5

14.2.138 log

Description: Log a line at the info level.

Class: cumulusci.tasks.util.LogLine

Command Syntax

```
$ cci task run log
```

Options

`--level LEVEL`

The logger level to use

Default: info

`--line LINE`

A formatstring like line to log

Required

`--format_vars FORMATVARS`

A Dict of format vars

Optional

14.2.139 generate_dataset_mapping

Description: Create a mapping for extracting data from an org.

Class: cumulusci.tasks.bulkdata.GenerateMapping

Generate a mapping file for use with the *extract_dataset* and *load_dataset* tasks. This task will examine the schema in the specified org and attempt to infer a mapping suitable for extracting data in packaged and custom objects as well as customized standard objects.

Mappings must be serializable, and hence must resolve reference cycles - situations where Object A refers to B, and B also refers to A. Mapping generation will stop and request user input to resolve such cycles by identifying the correct load order. If you would rather the mapping generator break such a cycle randomly, set the *break_cycles* option to *auto*.

Alternately, specify the *ignore* option with the name of one of the lookup fields to suppress it and break the cycle. *ignore* can be specified as a list in *cumulusci.yml* or as a comma-separated string at the command line.

In most cases, the mapping generated will need minor tweaking by the user. Note that the mapping omits features that are not currently well supported by the *extract_dataset* and *load_dataset* tasks, such as references to the *User* object.

Command Syntax

```
$ cci task run generate_dataset_mapping
```

Options

`--path PATH`

Location to write the mapping file

Default: datasets/mapping.yml

`--namespace_prefix NAMESPACEPREFIX`

The namespace prefix to use

Default: $project_config.project__package__namespace

`--ignore IGNORE`

Object API names, or fields in Object.Field format, to ignore

Optional

`--break_cycles BREAKCYCLES`

If the generator is unsure of the order to load, what to do? Set to *ask* (the default) to allow the user to choose or *auto* to pick randomly.

Optional

`--include INCLUDE`

Object names to include even if they might not otherwise be included.

Optional

`--strip_namespace STRIPNAMESPACE`

If True, CumulusCI removes the project's namespace where found in fields and objects to support automatic namespace injection. On by default.

Optional

14.2.140 extract_dataset

Description: Extract a sample dataset using the bulk API.

Class: cumulusci.tasks.bulkdata.ExtractData

Command Syntax

```
$ cci task run extract_dataset
```

Options

`--mapping MAPPING`

The path to a yaml file containing mappings of the database fields to Salesforce object fields

Default: datasets/mapping.yml

`--database_url DATABASEURL`

A DATABASE_URL where the query output should be written

Optional

`--sql_path SQLPATH`

If set, an SQL script will be generated at the path provided This is useful for keeping data in the repository and allowing diffs.

Default: datasets/sample.sql

`--inject_namespaces INJECTNAMESPACES`

If True, the package namespace prefix will be automatically added to (or removed from) objects and fields based on the name used in the org. Defaults to True.

Optional

`--drop_missing_schema DROPMISSINGSCHEMA`

Set to True to skip any missing objects or fields instead of stopping with an error.

Optional

14.2.141 load_dataset

Description: Load a SQL dataset using the bulk API.

Class: cumulusci.tasks.bulkdata.load.LoadData

Command Syntax

```
$ cci task run load_dataset
```

Options

`--database_url DATABASEURL`

The database url to a database containing the test data to load

Optional

`--mapping MAPPING`

The path to a yaml file containing mappings of the database fields to Salesforce object fields

Optional

`--start_step STARTSTEP`

If specified, skip steps before this one in the mapping

Optional

`--sql_path SQLPATH`

If specified, a database will be created from an SQL script at the provided path

Optional

`--ignore_row_errors IGNOREROWERRORS`

If True, allow the load to continue even if individual rows fail to load.

Optional

`--reset_oids RESETOIDS`

If True (the default), and the _sf_ids tables exist, reset them before continuing.

Optional

`--bulk_mode BULKMODE`

Set to Serial to force serial mode on all jobs. Parallel is the default.

Optional

`--inject_namespaces INJECTNAMESPACES`

If True, the package namespace prefix will be automatically added to (or removed from) objects and fields based on the name used in the org. Defaults to True.

Optional

`--drop_missing_schema DROPMISSINGSCHEMA`

Set to True to skip any missing objects or fields instead of stopping with an error.

Optional

`--set_recently_viewed SETRECENTLYVIEWED`

By default, the first 1000 records inserted via the Bulk API will be set as recently viewed. If fewer than 1000 records are inserted, existing objects of the same type being inserted will also be set as recently viewed.

Optional

`--org_shape_match_only ORGSHAPEMATCHONLY`

When True, all path options are ignored and only a dataset matching the org shape name will be loaded. Defaults to False.

Optional

`--enable_rollback ENABLEROLLBACK`

When True, performs rollback operation incase of error. Defaults to False

Optional

14.2.142 load_sample_data

Description: Load a saved sample dataset (experimental)

Class: cumulusci.tasks.sample_data.load_sample_data.LoadSampleData

Command Syntax

```
$ cci task run load_sample_data
```

Options

`--dataset DATASET`

The name of the dataset. If none is provided, it will use the scratch org config name (e.g. 'dev', 'qa') or fall back to 'default'.

Optional

`--ignore_row_errors IGNOREROWERRORS`

If True, allow the load to continue even if individual rows fail to load.

Optional

`--drop_missing_schema DROPMISSINGSCHEMA`

Set to True to skip any missing objects or fields instead of stopping with an error.

Optional

`--set_recently_viewed SETRECENTLYVIEWED`

By default, the first 1000 records inserted via the Bulk API will be set as recently viewed. If fewer than 1000 records are inserted, existing objects of the same type being inserted will also be set as recently viewed.

Optional

14.2.143 capture_sample_data

Description: Load a saved sample dataset (experimental)

Class: cumulusci.tasks.sample_data.capture_sample_data.CaptureSampleData

Command Syntax

```
$ cci task run capture_sample_data
```

Options

`--dataset DATASET`

The name of the dataset. If none is provided, it will use 'default'. Names that match scratch org config names (such as 'dev', 'qa') will be loaded into those orgs by the CumulusCI default org setup flows.

Optional

`--extraction_definition EXTRACTIONDEFINITION`

A file describing what to be extracted. Defaults to *datasets/{datasetname}/{datasetname}.extract.yml* if it exists.

Optional

`--loading_rules LOADINGRULES`

Path to .load.yml file containing rules to use when loading the mapping. Defaults to`datasets/{datasetname}/{datasetname}.load.yml` if it exists. Multiple files can be comma separated.

Optional

14.2.144 load_custom_settings

Description: Load Custom Settings specified in a YAML file to the target org

Class: cumulusci.tasks.salesforce.LoadCustomSettings

Command Syntax

```
$ cci task run load_custom_settings
```

Options

`--settings_path SETTINGSPATH`

The path to a YAML settings file

Required

14.2.145 remove_metadata_xml_elements

Description: Remove specified XML elements from one or more metadata files

Class: cumulusci.tasks.metadata.modify.RemoveElementsXPath

Command Syntax

```
$ cci task run remove_metadata_xml_elements
```

Options

`--xpath XPATH`

An XPath specification of elements to remove. Supports the re: regexp function namespace. As in re:match(text(), '.*__c')Use ns: to refer to the Salesforce namespace for metadata elements.for example: ./ns:Layout/ns:relatedLists (one-level) or //ns:relatedLists (recursive)Many advanced examples are available here: https://github.com/SalesforceFoundation/NPSP/blob/26b585409720e2004f5b7785a56e57498796619f/cumulusci.yml#L342

Optional

`--path PATH`

A path to the files to change. Supports wildcards including ** for directory recursion. More info on the details: https://www.poftut.com/python-glob-function-to-match-path-directory-file-names-with-examples/ https://www.tutorialspoint.com/How-to-use-Glob-function-to-find-files-recursively-in-Python

Optional

`--elements ELEMENTS`

A list of dictionaries containing path and xpath keys. Multiple dictionaries can be passed in the list to run multiple removal queries in the same task. This parameter is intended for usages invoked as part of a cumulusci.yml .

Optional

`--chdir CHDIR`

Change the current directory before running the replace

Optional

14.2.146 disable_tdtm_trigger_handlers

Description: Disable specified TDTM trigger handlers

Class: cumulusci.tasks.salesforce.trigger_handlers.SetTDTMHandlerStatus

Command Syntax

```
$ cci task run disable_tdtm_trigger_handlers
```

Options

`--handlers HANDLERS`

List of Trigger Handlers (by Class, Object, or 'Class:Object') to affect (defaults to all handlers).

Optional

`--namespace NAMESPACE`

The namespace of the Trigger Handler object ('eda' or 'npsp'). The task will apply the namespace if needed.

Optional

`--active ACTIVE`

True or False to activate or deactivate trigger handlers.

Optional

`--restore_file RESTOREFILE`

Path to the state file to store or restore the current trigger handler state. Set to False to discard trigger state information. By default the state is cached in an org-specific directory for later restore.

Optional

`--restore RESTORE`

If True, restore the state of Trigger Handlers to that stored in the (specified or default) restore file.

Optional

14.2.147 restore_tdtm_trigger_handlers

Description: Restore status of TDTM trigger handlers

Class: cumulusci.tasks.salesforce.trigger_handlers.SetTDTMHandlerStatus

Command Syntax

```
$ cci task run restore_tdtm_trigger_handlers
```

Options

`--handlers HANDLERS`

List of Trigger Handlers (by Class, Object, or 'Class:Object') to affect (defaults to all handlers).

Optional

`--namespace NAMESPACE`

The namespace of the Trigger Handler object ('eda' or 'npsp'). The task will apply the namespace if needed.

Optional

`--active ACTIVE`

True or False to activate or deactivate trigger handlers.

Optional

`--restore_file RESTOREFILE`

Path to the state file to store or restore the current trigger handler state. Set to False to discard trigger state information. By default the state is cached in an org-specific directory for later restore.

Optional

`--restore RESTORE`

If True, restore the state of Trigger Handlers to that stored in the (specified or default) restore file.

Default: True

14.2.148 vlocity_pack_export

Description: Executes the *vlocity packExport* command against an org

Class: cumulusci.tasks.vlocity.vlocity.VlocityRetrieveTask

Command Syntax

```
$ cci task run vlocity_pack_export
```

Options

`--job_file JOBFILE`

Filepath to the jobfile

Required

`--extra EXTRA`

Any extra arguments to pass to the vlocity CLI

Optional

14.2.149 vlocity_pack_deploy

Description: Executes the *vlocity packDeploy* command against an org

Class: cumulusci.tasks.vlocity.vlocity.VlocityDeployTask

Command Syntax

```
$ cci task run vlocity_pack_deploy
```

Options

`--job_file JOBFILE`

Filepath to the jobfile

Required

`--extra EXTRA`

Any extra arguments to pass to the vlocity CLI

Optional

`--npm_auth_key_env NPMAUTHKEYENV`

Environment variable storing an auth token for the Vlocity NPM Repository (npmAuthKey). If defined, appended to the job file.

Optional

14.3 Flow Reference

CumulusCI's suite of standard flows are grouped into various categories depending on their intended purpose.

14.3.1 Org Setup

These are the primary flows for doing full setup of an org. They typically include a flow from the Dependency Management group, a flow from either the Deployment or Install / Uninstall group, and a flow from the Post-Install Configuration group.

dev_org

Description: Set up an org as a development environment for unmanaged metadata

Flow Steps

```
1) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
2) flow: deploy_unmanaged
    0) task: dx_convert_from
       when: project_config.project__source_format == "sfdx" and not org_config.scratch
    1) task: unschedule_apex
    2) task: update_package_xml
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3) task: deploy
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3.1) task: deploy
         when: project_config.project__source_format == "sfdx" and org_config.scratch
    4) task: uninstall_packaged_incremental
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    5) task: snapshot_changes
3) flow: config_dev
    1) task: deploy_post
    2) task: update_admin_profile
    90) task: load_sample_data
4) task: snapshot_changes
```

dev_org_beta_deps

Description: This flow is deprecated. Please use dev_org instead.

Flow Steps

```
1) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
2) flow: deploy_unmanaged
    0) task: dx_convert_from
       when: project_config.project__source_format == "sfdx" and not org_config.scratch
    1) task: unschedule_apex
```

(continues on next page)

(continued from previous page)

```
    2) task: update_package_xml
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3) task: deploy
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3.1) task: deploy
         when: project_config.project__source_format == "sfdx" and org_config.scratch
    4) task: uninstall_packaged_incremental
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    5) task: snapshot_changes
3) flow: config_dev
    1) task: deploy_post
    2) task: update_admin_profile
    90) task: load_sample_data
```

dev_org_namespaced

Description: Set up a namespaced scratch org as a development environment for unmanaged metadata

Flow Steps

```
1) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
2) flow: deploy_unmanaged
    0) task: dx_convert_from
       when: project_config.project__source_format == "sfdx" and not org_config.scratch
    1) task: unschedule_apex
    2) task: update_package_xml
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3) task: deploy
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3.1) task: deploy
         when: project_config.project__source_format == "sfdx" and org_config.scratch
    4) task: uninstall_packaged_incremental
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    5) task: snapshot_changes
3) flow: config_dev
    1) task: deploy_post
    2) task: update_admin_profile
    90) task: load_sample_data
4) task: snapshot_changes
```

install_beta

Description: Install and configure the latest beta version

Flow Steps

```
1) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
2) task: install_managed_beta
3) flow: config_managed
    1) task: deploy_post
    2) task: update_admin_profile
    90) task: load_sample_data
4) task: snapshot_changes
```

install_prod

Description: Install and configure the latest production version

Flow Steps

```
1) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
2) task: install_managed
3) flow: config_managed
    1) task: deploy_post
    2) task: update_admin_profile
    90) task: load_sample_data
4) task: snapshot_changes
```

qa_org

Description: Set up an org as a QA environment for unmanaged metadata

Flow Steps

```
1) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
2) flow: deploy_unmanaged
    0) task: dx_convert_from
       when: project_config.project__source_format == "sfdx" and not org_config.scratch
    1) task: unschedule_apex
    2) task: update_package_xml
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3) task: deploy
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3.1) task: deploy
         when: project_config.project__source_format == "sfdx" and org_config.scratch
    4) task: uninstall_packaged_incremental
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
```

(continues on next page)

(continued from previous page)

```
    5) task: snapshot_changes
3) flow: config_qa
    1) task: deploy_post
    2) task: update_admin_profile
    90) task: load_sample_data
4) task: snapshot_changes
```

qa_org_2gp

Description: Set up an org as a QA environment using a second-generation package

Flow Steps

```
1) flow: install_2gp_commit
    1) task: github_package_data
    2) flow: dependencies
        1) task: update_dependencies
        2) task: deploy_pre
    3) task: install_managed
2) flow: config_qa
    1) task: deploy_post
    2) task: update_admin_profile
    90) task: load_sample_data
3) task: snapshot_changes
```

qa_org_unlocked

Description: Set up an org as a QA environment using an unlocked package

Flow Steps

```
1) flow: install_unlocked_commit
    1) task: github_package_data
    2) flow: dependencies
        1) task: update_dependencies
        2) task: deploy_pre
    3) task: install_managed
2) flow: config_qa
    1) task: deploy_post
    2) task: update_admin_profile
    90) task: load_sample_data
3) task: snapshot_changes
```

regression_org

Description: Simulates an org that has been upgraded from the latest release of to the current beta and its dependencies, but deploys any unmanaged metadata from the current beta.

Flow Steps

```
1) flow: install_regression
    1) flow: dependencies
        1) task: update_dependencies
        2) task: deploy_pre
    2) task: install_managed
    3) task: install_managed_beta
2) flow: config_regression
    1) flow: config_managed
        1) task: deploy_post
        2) task: update_admin_profile
        90) task: load_sample_data
3) task: snapshot_changes
```

14.3.2 Dependency Management

These flows deploy dependencies (base packages and unmanaged metadata) to a target org environment.

beta_dependencies

Description: This flow is deprecated. Please use the *dependencies* flow and set the *include_beta* option on the first task, *update_dependencies*. Deploy the latest (beta) version of dependencies to prepare the org environment for the package metadata

Flow Steps

```
1) task: update_dependencies
2) task: deploy_pre
```

dependencies

Description: Deploy dependencies to prepare the org environment for the package metadata

Flow Steps

```
1) task: update_dependencies
2) task: deploy_pre
```

14.3.3 Deployment

These flows deploy the main package metadata to a target org environment.

deploy_packaging

Description: Process and deploy the package metadata to the packaging org

Flow Steps

```
0) task: dx_convert_from
   when: project_config.project__source_format == "sfdx"
1) task: unschedule_apex
2) task: create_managed_src
3) task: update_package_xml
4) task: deploy
5) task: revert_managed_src
6) task: uninstall_packaged_incremental
```

deploy_unmanaged

Description: Deploy the unmanaged metadata from the package

Flow Steps

```
0) task: dx_convert_from
   when: project_config.project__source_format == "sfdx" and not org_config.scratch
1) task: unschedule_apex
2) task: update_package_xml
   when: project_config.project__source_format != "sfdx" or not org_config.scratch
3) task: deploy
   when: project_config.project__source_format != "sfdx" or not org_config.scratch
3.1) task: deploy
     when: project_config.project__source_format == "sfdx" and org_config.scratch
4) task: uninstall_packaged_incremental
   when: project_config.project__source_format != "sfdx" or not org_config.scratch
5) task: snapshot_changes
```

deploy_unmanaged_ee

Description: Deploy the unmanaged metadata from the package to an Enterprise Edition org

Flow Steps

```
0) task: dx_convert_from
   when: project_config.project__source_format == "sfdx"
1) task: unschedule_apex
2) task: update_package_xml
3) task: create_unmanaged_ee_src
4) task: deploy
5) task: revert_unmanaged_ee_src
6) task: uninstall_packaged_incremental
```

unmanaged_ee

Description: Deploy the unmanaged package metadata and all dependencies to the target EE org

Flow Steps

```
1) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
2) flow: deploy_unmanaged_ee
    0) task: dx_convert_from
       when: project_config.project__source_format == "sfdx"
    1) task: unschedule_apex
    2) task: update_package_xml
    3) task: create_unmanaged_ee_src
    4) task: deploy
    5) task: revert_unmanaged_ee_src
    6) task: uninstall_packaged_incremental
```

14.3.4 Install / Uninstall

These flows handle package installation and uninstallation in particular scenarios.

install_2gp_commit

Description: Install the 2GP package for the current commit

Flow Steps

```
1) task: github_package_data
2) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
3) task: install_managed
```

install_prod_no_config

Description: Install but do not configure the latest production version

Flow Steps

```
1) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
2) task: install_managed
3) task: deploy_post
```

install_regression

Description: Install the latest beta dependencies and upgrade to the latest beta version from the most recent production version

Flow Steps

```
1) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
2) task: install_managed
3) task: install_managed_beta
```

install_unlocked_commit

Description: Install the unlocked package for the current commit

Flow Steps

```
1) task: github_package_data
2) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
3) task: install_managed
```

uninstall_managed

Description: Uninstall the installed managed version of the package. Run this before install_beta or install_prod if a version is already installed in the target org.

Flow Steps

```
1) task: uninstall_post
2) task: uninstall_managed
```

14.3.5 Post-Install Configuration

These flows perform configuration after the main package has been installed or deployed.

config_apextest

Description: Configure an org to run apex tests after package metadata is deployed

Flow Steps

```
1) task: deploy_post
2) task: update_admin_profile
```

config_dev

Description: Configure an org for use as a dev org after package metadata is deployed

Flow Steps

```
1) task: deploy_post
2) task: update_admin_profile
90) task: load_sample_data
```

config_managed

Description: Configure an org for use after the managed package has been installed.

Flow Steps

```
1) task: deploy_post
2) task: update_admin_profile
90) task: load_sample_data
```

config_packaging

Description: Configure packaging org for upload after package metadata is deployed

Flow Steps

```
1) task: update_admin_profile
```

config_qa

Description: Configure an org for use as a QA org after package metadata is deployed

Flow Steps

```
1) task: deploy_post
2) task: update_admin_profile
90) task: load_sample_data
```

config_regression

Description: Configure an org for QA regression after the package is installed

Flow Steps

```
1) flow: config_managed
    1) task: deploy_post
    2) task: update_admin_profile
    90) task: load_sample_data
```

14.3.6 Continuous Integration

These flows are designed to be run automatically by a continuous integration (CI) system in response to new commits. They typically set up an org and run Apex tests.

ci_beta

Description: Install the latest beta version and runs apex tests from the managed package

Flow Steps

```
1) flow: install_beta
    1) flow: dependencies
        1) task: update_dependencies
        2) task: deploy_pre
    2) task: install_managed_beta
    3) flow: config_managed
        1) task: deploy_post
        2) task: update_admin_profile
        90) task: load_sample_data
    4) task: snapshot_changes
2) task: run_tests
```

ci_feature

Description: Prepare an unmanaged metadata test org and run Apex tests. Intended for use against feature branch commits.

Flow Steps

```
0.5) task: github_parent_pr_notes
1) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
2) flow: deploy_unmanaged
    0) task: dx_convert_from
       when: project_config.project__source_format == "sfdx" and not org_config.scratch
    1) task: unschedule_apex
    2) task: update_package_xml
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3) task: deploy
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3.1) task: deploy
         when: project_config.project__source_format == "sfdx" and org_config.scratch
    4) task: uninstall_packaged_incremental
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    5) task: snapshot_changes
3) flow: config_apextest
    1) task: deploy_post
    2) task: update_admin_profile
4) task: run_tests
5) task: github_automerge_feature
```

(continues on next page)

(continued from previous page)

```
   when: project_config.repo_branch and project_config.repo_branch.startswith(project_
↪config.project__git__prefix_feature)
```

ci_feature_2gp

Description: Install as a managed 2gp package and run Apex tests. Intended for use after build_feature_test_package.

Flow Steps

```
1) flow: install_2gp_commit
    1) task: github_package_data
    2) flow: dependencies
        1) task: update_dependencies
        2) task: deploy_pre
    3) task: install_managed
2) flow: config_apextest
    1) task: deploy_post
    2) task: update_admin_profile
3) task: run_tests
```

ci_feature_beta_deps

Description: This flow is deprecated. Please use ci_feature instead.

Flow Steps

```
0.5) task: github_parent_pr_notes
1) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
2) flow: deploy_unmanaged
    0) task: dx_convert_from
       when: project_config.project__source_format == "sfdx" and not org_config.scratch
    1) task: unschedule_apex
    2) task: update_package_xml
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3) task: deploy
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    3.1) task: deploy
         when: project_config.project__source_format == "sfdx" and org_config.scratch
    4) task: uninstall_packaged_incremental
       when: project_config.project__source_format != "sfdx" or not org_config.scratch
    5) task: snapshot_changes
3) flow: config_apextest
    1) task: deploy_post
    2) task: update_admin_profile
4) task: run_tests
5) task: github_automerge_feature
   when: project_config.repo_branch and project_config.repo_branch.startswith(project_
↪config.project__git__prefix_feature)
```

ci_master

Description: Deploy the package metadata to the packaging org and prepare for managed package version upload. Intended for use against main branch commits.

Flow Steps

```
1) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
2) flow: deploy_packaging
    0) task: dx_convert_from
       when: project_config.project__source_format == "sfdx"
    1) task: unschedule_apex
    2) task: create_managed_src
    3) task: update_package_xml
    4) task: deploy
    5) task: revert_managed_src
    6) task: uninstall_packaged_incremental
3) flow: config_packaging
    1) task: update_admin_profile
```

ci_release

Description: Install a production release version and runs tests from the managed package

Flow Steps

```
1) flow: install_prod
    1) flow: dependencies
        1) task: update_dependencies
        2) task: deploy_pre
    2) task: install_managed
    3) flow: config_managed
        1) task: deploy_post
        2) task: update_admin_profile
        90) task: load_sample_data
    4) task: snapshot_changes
2) task: run_tests
```

14.3.7 Release Operations

These flows are used to release new package versions.

build_feature_test_package

Description: Create a 2gp managed package version

Flow Steps

```
1) task: update_package_xml
   when: project_config.project__source_format != "sfdx"
2) task: create_package_version
```

build_unlocked_test_package

Description: Create an Unlocked package version

Flow Steps

```
1) task: update_package_xml
   when: project_config.project__source_format != "sfdx"
2) task: create_package_version
3) task: promote_package_version
```

release_2gp_beta

Description: Upload and release a beta 2gp managed package version

Flow Steps

```
1) task: create_package_version
2) task: github_release
3) task: github_release_notes
4) task: github_automerge_main
```

release_2gp_production

Description: Promote the latest beta 2gp managed package version and create a new release in GitHub

Flow Steps

```
1) task: promote_package_version
2) task: github_release
3) task: github_release_notes
```

release_beta

Description: Upload and release a beta version of the metadata currently in packaging

Flow Steps

```
1) task: upload_beta
2) task: github_release
3) task: github_release_notes
4) task: github_automerge_main
```

release_production

Description: Upload and release a production version of the metadata currently in packaging

Flow Steps

```
1) task: upload_production
2) task: github_release
3) task: github_release_notes
```

release_unlocked_beta

Description: Upload and release a beta 2gp unlocked package version

Flow Steps

```
1) task: create_package_version
2) task: github_release
3) task: github_release_notes
4) task: github_automerge_main
```

release_unlocked_production

Description: Promote the latest beta 2GP unlocked package version and create a new release in GitHub

Flow Steps

```
1) task: promote_package_version
2) task: github_release
3) task: github_release_notes
```

14.3.8 Other

This is a catch-all group for any flows without a designated "group" attribute in `cumulusci.yml`.

push_upgrade_org

Description:

Flow Steps

```
1) flow: dependencies
    1) task: update_dependencies
    2) task: deploy_pre
2) task: install_managed
3) task: update_dependencies
4) task: install_managed_beta
5) flow: config_qa
    1) task: deploy_post
    2) task: update_admin_profile
    90) task: load_sample_data
```

CHAPTER

FIFTEEN

ABOUT CUMULUSCI

15.1 History

15.1.1 v3.85.0 (2024-03-13)

15.1.2 What's Changed

Changes

- Fix Invalid Lookups of namespaced fields by @aditya-balachander in #3759
- Fix Id Not Specified on Update Call by @aditya-balachander in #3761
- Fix dependency parsing errors caused by `collision_check` option by @jstvz in #3760

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.84.3...v3.85.0

15.1.3 v3.84.3 (2024-02-24)

15.1.4 What's Changed

Changes

- Fix: Type Error while loading Contacts when PersonAccount is enabled by @jkasturi-sf in #3756

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.84.2...v3.84.3

15.1.5 v3.84.2 (2024-02-20)

15.1.6 What's Changed

Changes

- Polymorphic Lookup Data Extraction by @aditya-balachander in #3741
- ADR: Migrate from SFDX to SF CLI by @jstvz in #3630
- Fix: Mismatch in the order of fields between the step and query during generation of the json request by @aditya-balachander in #3750

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.84.1...v3.84.2

15.1.7 v3.84.1 (2024-02-01)

15.1.8 What's Changed

Changes

- Fix `is_person_type` `AttributeError` in record type handling by @aditya-balachander in #3739

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.84.0...v3.84.1

15.1.9 v3.84.0 (2024-01-29)

15.1.10 What's Changed

Changes

- Incorrect generation of record type mapping table name in a namespaced org context by @lakshmi2506 in #3713
- Retrieve pre-flight checks by @lakshmi2506 in #3723
- Error during upsert on identical DeveloperName for RecordType of same sObject by @aditya-balachander in #3702
- Rollback Operation for Inserts and Upserts by @aditya-balachander in #3718
- Profile Retrieval in Metecho by @aditya-balachander in #3711
- Identifies and retrieves non-source-trackable Metadata type components by @lakshmi2506 in #3727
- fix: re-enable telemetry, make CCI usage identifiable by @mshanemc in #3728
- Add permsets to return_values for task get_assigned_permission_sets by @lakshmi2506 in #3736
- ListComponents modified to SOAP call by @lakshmi2506 in #3735

15.1.11 New Contributors

- @mshanemc made their first contribution in #3728

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.83.0...v3.84.0

15.1.12 v3.83.0 (2023-12-08)

15.1.13 What's Changed

Changes

- Allowed namespace injection without managed by @jain-naman-sf in #3677
- Update pyright configuration for 1.1.339 by @jstvz in #3715
- Replace distutils with shutil for 3.12 by @jstvz in #3714

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.82.0...v3.83.0

15.1.14 v3.82.0 (2023-12-01)

15.1.15 What's Changed

Critical Changes

- Remove `robot_lint` task and dependencies by @jstvz in #3697

Changes

- Show an error message when no `origin` remote is present by @lakshmi2506 in #3679
- Improve task `return_values` documentation by @lakshmi2506 in #3689
- Task to retrieve a complete Profile from an org by @aditya-balachander in #3672
- Support Python 3.12 by @jstvz in #3691
- Make task `options` read only when using Pydantic option validation by @aditya-balachander in #3695

Issues Fixed

- Fix Github url parse error for some scenarios by @mgrandhi in #3683

15.1.16 New Contributors

- @mgrandhi made their first contribution in #3683

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.81.0...v3.82.0

15.1.17 v3.81.0 (2023-11-03)

15.1.18 What's Changed

Changes

- Add option to specify release window in `cci org scratch` by @jain-naman-sf in #3653
- Add describe_metadatatypes task by @lakshmi2506 in #3669
- Fix JSON output for `cci org info` by @jain-naman-sf in #3675
- Fix `TypeError` when service sensitive attribute is `None` by @jain-naman-sf in #3674
- metadeploy_publish presents clear errors when plans are not available by @lakshmi2506 in #3684
- Add clear error message when `create_package_version` run against persistent orgs by @lakshmi2506 in #3676
- Add collision-check option to deploy task by @jain-naman-sf in #3667
- Handling exception when the Tooling API returns a test result with a null method name by @lakshmi2506 in #3681

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.80.0...v3.81.0

15.1.19 v3.80.0 (2023-09-29)

15.1.20 What's Changed

Changes

- Allow setting Major and Minor Version in upload_production task by @jain-naman-sf in #3651
- Add better error handling for empty or invalid org and service env vars by @prescod in #3365
- Query with install_key in promote_package_version (via @zenibako) by @davidmreed in #3654
- Trim whitespaces during service data entry by @aditya-balachander in #3661
- Display namespace in output of `cci org info` by @jain-naman-sf in #3662
- Extend `deploy` task to support REST API deployment by @aditya-balachander in #3650
- Add `ApexTestSuite` support in the `run_tests` task by @mjawadtp in #3660
- Implement variable substitution for nested structures in cumulusci.yml by @aditya-balachander in #3665
- Add XPath-based 'find_replace' functionality to `deploy` by @aditya-balachander in #3655
- Improve message description handling for scratch org cli `not_found` error by @lakshmi2506 in #3659
- Add `deactivate_flows` task by @lakshmi2506 in #3664

Issues Fixed

- Add guard for empty body to github_release_report by @jstvz in #3645

15.1.21 New Contributors

- @zenibako made their first contribution in #3654
- @jain-naman-sf made their first contribution in #3651
- @aditya-balachander made their first contribution in #3661
- @mjawadtp made their first contribution in #3660
- @lakshmi2506 made their first contribution in #3659

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.79.0...v3.80.0

15.1.22 v3.79.0 (2023-09-07)

15.1.23 What's Changed

Changes

- Added workarounds to support Vlocity local compilation in CI by @jstvz in #3642

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.78.0...v3.79.0

15.1.24 v3.78.0 (2023-08-10)

15.1.25 What's Changed

Changes

- Updated Permission Set preflights to handle Permission Set Groups by @jkasturi-sf in #3623
- Added the `create_pull_request_on_conflict` option to automerge tasks by [@Benjamin-French](https://github.com/ BenjaminFrench) in #3632

Issues Fixed

- Fixed issues with uploading some Unlocked Packages by @jstvz in #3636
- Corrected behavior of the `skip_future_releases` option by @jstvz in #3624

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.77.0...v3.78.0

15.1.26 v3.77.0 (2023-07-07)

15.1.27 What's Changed

Changes

- Add preflight check for PSL assignments by @davidmreed in #3616
- Use Marketing Cloud's validate endpoint to update deployment packages by @davidmreed and @TheBitShepherd in #3598

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.76.0...v3.77.0

15.1.28 v3.76.0.dev0 (2023-05-25)

15.1.29 What's Changed

Changes

- Use Marketing Cloud's validate endpoint to update deployment packages by @davidmreed and @TheBitShepherd in #3598

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.76.0...v3.76.0.dev0

15.1.30 v3.76.0 (2023-05-25)

15.1.31 What's Changed

Changes

- Added opt-in Pydantic-based task option parsing. by @prescod in #1618
- Add deploy transform to inject org URLs by @davidmreed in #3596

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.75.1...v3.76.0

15.1.32 v3.75.1 (2023-04-14)

15.1.33 What's Changed

Issues Fixed

- Ensure that top-level 'tasks' directory is read for custom tasks by @prescod in #3582

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.75.0...v3.75.1

15.1.34 v3.75.0 (2023-04-13)

15.1.35 What's Changed

Changes

- Fix bug that prevented "api" declaration in `extract_declaration` from being respected during `capture_sample_dataset`. by @prescod in #3568
- Update windows install steps by @TheBitShepherd in #3571
- CCI projects can now run tasks from their "sources" even when they are custom tasks coded in Python. By @prescod in #3345
- The `capture_sample_data` task now looks for a `datasetname.load.yml` file to influence how it generates its .mapping file for subquent loads. By @prescod in #3569

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.74.0...v3.75.0

15.1.36 v3.74.0 (2023-03-06)

15.1.37 What's Changed

Changes

- Allow use of static release notes on GitHub by @davidmreed in #3550
- `cci service update` by @TheBitShepherd in #3481
- Add TimelineObjectDefinition metadata type by @jstvz in #3551

Issues Fixed

- Fix an issue with `capture_sample_data` and record type capturing. by @prescod in #3552

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.73.0...v3.74.0

15.1.38 v3.73.0 (2023-02-16)

No changes (other than dependency updates) since v3.72.1.dev2 . Please see the changes below.

15.1.39 v3.72.1.dev2 (2023-02-10)

15.1.40 What's Changed

Changes

- Include `modelVersion` in MCPM Payload by @TheBitShepherd in #3523
- Fix MCPM deploy again by @TheBitShepherd in #3526
- Mask invalid oauth token by @jstvz in #3529
- Include nearly everything in the wheel by @jstvz in #3531

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.72.1...v3.72.1.dev2

15.1.41 v3.72.1.dev1 (2023-02-08)

15.1.42 What's Changed

Changes

- Include `modelVersion` in MCPM Payload by @TheBitShepherd in #3523
- Fix MCPM deploy again by @TheBitShepherd in #3526

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.72.1...v3.72.1.dev1

15.1.43 v3.72.1.dev0 (2023-02-08)

15.1.44 What's Changed

Changes

- Change `deploy_marketing_cloud_package` task to include `modelVersion` in MCPM Payload by @TheBitShepherd in #3523

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.72.1...v3.72.1.dev0

15.1.45 v3.72.1 (2023-02-07)

15.1.46 What's Changed

Issues Fixed

- Fixed an issue preventing installation of CumulusCI's sdist in #3520

15.1.47 v3.72.0 (2023-02-03)

15.1.48 What's Changed

Changes

- Improve documentation of tasks w.r.t required/option/defauled fields. by @prescod in #3447

Issues Fixed

- Ensure that products have a slug before publishing translations by @prescod in #3483
- Remove deprecated Playwright keywords by @jstvz in #3503
- Stricter check on what counts as a dataset by @prescod in #3502

Full Changelog: https://github.com/SFDO-Tooling/CumulusCI/compare/v3.71.0...v3.72.0

15.1.49 3.71.0 (2022-12-20)

Critical Changes

- A few weeks ago we soft-launched a `load_data` feature that matched the dataset loaded to the `org_shape`. If you had discovered and used the `load_data` option called `org_shape_match_only`, you should switch to the new task because eventually we will remove the option and only use the new `load_sample_data` task. (#3455)

Changes

- CumulusCI has two new tasks for working with sample data: `load_sample_data` and `capture_sample_data`. (#3455)
- CumulusCI now offers the ability to inject arbitrary IDs into a project's metadata components. See the Find-and-Replace Id Injection source transform for more information. (#3460)
- CumulusCI now allows for injecting the current running user's username into a project's metadata components. See the Find-and-Replace Id Injection source transform for more information. (#3460)

15.1.50 3.70.0 (2022-11-29)

- The `retrieve_changes` and `list_changes` tasks now properly exclude metadata types that `SFDX` is unable to process. These include: `AuraDefinition`, `ExperienceResource`, and `LightningComponentResource` by @jstvz in https://github.com/SFDO-Tooling/CumulusCI/pull/3443
- The `deploy_remote_site_settings` task has been updated to load the correct Remote Site Settings records for OmniStudio by @jofsky in https://github.com/SFDO-Tooling/CumulusCI/pull/3444

15.1.51 3.69.0 (2022-11-14)

Changes

- Added support for debugging and breakpoints in Playwright-based Robot Framework by @boakley in https://github.com/SFDO-Tooling/CumulusCI/pull/3408
- Added option for deploy task to remove selected components from the bundle by @jkasturi-sf in https://github.com/SFDO-Tooling/CumulusCI/pull/3421
- Added documentation for using OmniStudio in Cumulus flows by @bethbrains in https://github.com/SFDO-Tooling/CumulusCI/pull/3427
- Update creation of OmniStudio Remote Site Settings for new Salesforce domain patterns (e.g. .*scratch.my.salesforce, *.demo.my.salesforce, etc.) by @jofsky in https://github.com/SFDO-Tooling/CumulusCI/pull/3424
- Added –debug flag to the deploy_marketing_cloud_package task by @TheBitShepherd in https://github.com/SFDO-Tooling/CumulusCI/pull/3430

15.1.52 3.68.1 (2022-11-01)

- Revert a change in parsing config files which was incompatible with some configs

15.1.53 3.68.0 (2022-10-31)

Critical Changes

- Upgraded Robot Framework from version 4.1.3 to version 6. As with most Robot Framework updates, there are a host of new features and a few deprecated features and backwards incompatibilities. For a complete list consult the Robot Framework release notes in https://github.com/SFDO-Tooling/CumulusCI/pull/3417

Changes

- Add support for InstallKey, sourceOrg, release keys on 2GP version create by @davidmreed in https://github.com/SFDO-Tooling/CumulusCI/pull/3403
- We now support the `Territory2`, `Territory2Model`, `Territory2Type`, and `Territory2Rule` MetaData types. by @TheBitShepherd in https://github.com/SFDO-Tooling/CumulusCI/pull/3406
- Add `push_upgrade_org` flow to test in simulated push upgrade environment by @davidmreed in https://github.com/SFDO-Tooling/CumulusCI/pull/3286
- Fixed bug preventing the `deploy_omni_studio_site_settings` task from deploying to scratch orgs with the new `.scratch.` domain names by @jofsky in https://github.com/SFDO-Tooling/CumulusCI/pull/3411
- All new keychain details are serialized using JSON, not pickle by @bethbrains in https://github.com/SFDO-Tooling/CumulusCI/pull/3390

- Snowfakery (upgraded to 3.4) can now generate Event or Meeting Schedules similar to Calendar Apps
- Snowfakery now has a Salesforce.ContentFile feature for generating Content Versions

Issues Fixed

- Bumped key_size from 2048 to 4096 on advice from a security audit by @boakley in https://github.com/SFDO-Tooling/CumulusCI/pull/3389
- Fixed an issue where unknown deploy statuses could cause polling to hang in the deploy_marketing_cloud_package task by @TheBitShepherd in https://github.com/SFDO-Tooling/CumulusCI/pull/3394
- Fixed an injection vulnerability related to unquoted csv writers by @TheBitShepherd in https://github.com/SFDO-Tooling/CumulusCI/pull/3404

15.1.54 3.67.1 (2022-10-14)

Issues Fixed

- We added some javascript files that were missing from the 3.67.0 release

15.1.55 3.67.0 (2022-10-13)

Critical Changes

- CumulusCI now supports GitHub Enterprise. (#3256)
- CumulusCI will no longer be supporting installations via `brew`. See our docs for the officially supported install method via pipx.(#3382)

Changes

- When using Playwright instead of Selenium, we now have limited support for writing keywords in JavaScript. For more information, see the topic "Writing keywords in JavaScript" in the documentation under "Playwright Technology Preview". (#3378)
- Auto-load dataset matching org shape name(#3384)
- Updated the dependency resolvers used for commit-status builds (2GP and Unlocked Package parallel testing) to allow resolution to "fall back" from release branches like `feature/240` to the repository's main branch.(#3386)
- There is a new Playwright-based keyword `Wait until salesforce is ready`. This keyword is automatically called by the Open Test Browser keyword in the SalesforcePlaywright library. In addition to waiting for the page to be rendered, it will auto attempt to detect a classic page on initial render, and automatically switch to a lightning page if one is detected.(#3387)
- CumulusCI now supports the `CustomIndex` metadata type.(#3391)
- CumulusCI now supports performing source transformations during the `deploy` task, including find-and-replace of user-defined tokens. (#3383)
- Includes Snowfakery 3.3 with new datetime features and find_record optimization.

Issues Fixed

- Fixed a bug when using blank dates in upserts and updates.(#3361)
- Fixed an issue where committing changes to a repository above a certain size caused timeouts to occur. (#3379)
- CumulusCI now uses the `defusedxml` library to more securely parse xml.(#3375)

15.1.56 3.66.0 (2022-09-29)

Changes

- New Metadata ETL task `set_object_settings` to enable and disable settings on standard and custom objects (thanks @jlantz) (#3364)
- For Robot tests on the Firefox profile, we now accept all cookies to prevent lightning pages from failing to render due to cross-domain cookies. (#3366)
- Improved the method that playwright-based tests use to wait for the browser to initialize (#3366)
- The following Robot Framework keywords now automatically generate screenshots on failure: `Click related list button`, `Click related item popup link`, `Close modal`, `Current app should be`, `Field value should be`, `Get field value`, `Populate field`, `Populate lookup field`, and `Wait until modal is closed`. (#3370)
- CumulusCI now supports the following Metadata type: `AIApplicationConfig` (#3368)
- CumulusCI now supports the following types from Net Zero Cloud's Tooling API: `BldgEnrgyIntensityCnfg`, `StnryAssetEnvSrcCnfg`, `VehicleAssetEmssnSrcCnfg`. (#3372)

Issues Fixed

- A failure when calling `capture page screenshot` no longer causes a Robot test to fail (#3366)
- Fixed an issue where the `cci --version` command failed to output the current version (#3367)
- A bug that prevented keywords decorated with `@capture_screenshot_on_failure` from returning a value has been fixed. (#3370)
- Fixed a bug where requests were not being logged while running tasks or flows with the `--debug` flag. (#3371)

15.1.57 3.65.0 (2022-09-16)

Changes

- Improve logging and add error handling for Vlocity tasks by @prescod in (#3357)

15.1.58 3.64.0 (2022-08-18)

Changes

- Support for upcoming Salesforce 240 Release in Robot by @boakley (#3321)

Issues Fixed

- Fix crash with unusual Unicode characters in metadata on Windows running `update_package_xml` by @prescod (#3323)
- Fix Scratch org creation in latest sfdx by @prescod (#3334)

15.1.59 3.63.0 (2022-08-04)

Changes

- CumulusCI can now validate MetaDeploy plans in cumulusci.yml by @jstvz in (#3298)
- Sensitive service info is now masked in the `cci service info` command - by @bethbrains in (#3295)
- Updated documentation for GitHub Actions framework by @davidmreed in (#3302)
- New MetaData ETL Task for deploying remote site settings - `add_remote_site_settings` by @Br4nd0R in (#3303)
- New task for deploying remote site settings that are required by OmniStudio - `deploy_omni_studio_site_settings` by @Br4nd0R in (#3303)
- Added new robot keywords available when using the SalesforcePlaywright library; get current record id, wait until loading is complete and go to record home. These keywords work in a similar fashion to the existing keywords of the same name that are based on Selenium - by @boakley in (#3310)

Issues Fixed

- Fixed two issues which prevented the `set_recently_viewed` feature from working by @prescod in (#3305)
- The GithubIssuesParser class will now fall back if issues are disabled by @davidmreed in (#3293)

15.1.60 3.62.0 (2022-07-21)

Changes

- CumulusCI now supports pinning dependencies on other projects by @davidmreed in (#3270)
- Add extract declarations concept to CumulusCI by @prescod in (#3285)
- Make `delete_data` task freezable by @jstvz in (#3294)
- Omnistudio vlocity build tool wrapper tasks `vlocity_pack_export` and `vlocity_pack_deploy` to support datapack export and deployments by @Br4nd0R in (#3296)

15.1.61 3.61.1 (2022-07-10)

Issues Fixed

- Fix `metadeploy_publish` error caused by supported_orgs case-sensitivity (#3277)

15.1.62 3.61.0 (2022-07-09)

Changes

- `metadeploy_publish` consumes legal targets for supported orgs (#3239)
- Add update_profile support for record type layouts by (#3243)
- Add `custom` key to cumulusci.yml schema (#3238)
- Make OrgSettings also deploy objectSettings, if present (#3235)
- Import persistent orgs from the SFDX keychain (#3253)
- Speed up Snowfakery handling of small batch sizes (#3188)

Issues Fixed

- Fix JWT audience for sandboxes (#3240)
- Fixed a bug where scratch org attributes configured via cumulusci.yml were not taking effect. (#3209)
- Correct FeatureParameterString to FeatureParameterDate (#3254)
- No postinstall script for unlocked package builds (#3247)

15.1.63 3.60.0 (2022-06-09)

Changes

- We added a new dependency resolution strategy, `unlocked`. Like our existing dependency strategies using parallel 2GP package builds, this strategy installs parallel unlocked package builds. (#3223)
- We added the flows `build_unlocked_test_package`, `install_unlocked_commit`, and `qa_org_unlocked` to support use of parallel unlocked packages. (#3223)
- We added a property, `is_survey_advanced_features_enabled`, to the org_config entity available in when clauses. (#3149)
- Robot framework API and performance keywords have been moved into separate libraries. They are still automatically imported when using Salesforce.robot so it is not necessary to modify any existing test files. (#3216)
- There are new robot framework keyword libraries with limited support for `Playwright <https://playwright.dev/>_` via the `robotframework-browser library <https://robotframework-browser.org/>_`. To use, import the resource file `cumulusci/robotframework/SalesforcePlaywright.robot` instead of `cumulusci/robotframework/Salesforce.robot`. Note, Playwright is not yet supported by MetaCI. (#3216)
- `cumulusci.robotframework.BaseLibrary` can be used when creating new libraries in order to get some useful properties (`self.salesforce`, `self.salesforce_api`, `self.cumulusci`, etc) (#3216)
- Snowfakery 3.2 is included, with the following changes
- Snowfakery can now do random_reference to nicknames.
- We removed some limitations on what Snowfakery can do with objects referenced through random_reference, especially using a.b.c.d syntax.
- We added a unique option for random_reference to allow for 1-1 relationships across random references and no-duplicates many-to-many join relationships like Account->CampaignMember->Campaign

15.1.64 3.59.1 (2022-05-31)

Issues Closed

- Fixed a regression relating to lookup of variables like `$project_config.repo_branch`. Identified as a problem in the `github_automerge_main` task, but potentially there might be other tasks that were broken and are fixed by this branch. (#3220)

15.1.65 3.59.0 (2022-05-26)

Changes

- Setting `repo_url` in your project config is no longer required to run `metadeploy_publish`. (#3200)
- Adds new commands `cci robot install_playwright` and `cci robot uninstall_playwright` for installing and uninstalling experimental support for the Robot Framework `Browser` library, which is based on Playwright. For more information see the section "Playwright Technology Preview" in the public documentation. (#3194)
- Programs using CumulusCI as a library may generate warnings when getting values from config objects. A new method `config.lookup` is the preferred way to look up these values. (#3026)

15.1.66 3.58.0 (2022-05-16)

Changes

- CumulusCI Robot Framework support has been tested on prerelease orgs (#3187)
- Upgraded the Snowfakery dependency to version 3.1.0. When embedded in CumulusCI, Snowfakery can now do SObject Upserts. (#3195)

Issues Closed

- Fix bug when setting the 'options' option in the robot task from the command line (#3173)

15.1.67 3.57.0 (2022-04-28)

Changes

- Support `useralias` in `Open Test Browser` for connected orgs (#3169)

Issues Closed

- More reliable way to find latest chronological release in GitHub (#3164)
- Add IP Ranges Metadata ETL task (#3146)

15.1.68 3.56.0 (2022-04-07)

Changes

- Added support for Lightning radiobuttons in the Robot keyword input form data. (#3142)

Issues Closed

- Fixed an issue where CumulusCI was incorrectly merging task option overrides. (#3145)

15.1.69 3.55.0 (2022-03-24)

Changes

- CumulusCI now provides an option to verify server certificates using root CA certs loaded from the system, instead of the default set of CA certs bundled with the Python requests library. This is considered experimental, so must be opted in by setting the `CUMULUSCI_SYSTEM_CERTS` environment variable to `True` (#3114)
- Improvements to the custom locator strategy used by the Robot Framework keyword `input form data` to find form fields and lightning components based on label text. This locator strategy (`label:`) is now up to 10x faster at finding non-lightning form components, and is able to find elements with more complicated labels (eg: labels with text inside of nested spans). (#3117)
- Fixed a bug that prevented the use of aliases for Robot Framework page objects.(#3120)
- The Robot Framework keyword `input form data` now does a better job of working with non-lightning checkboxes. (#3122)

Issues Closed

- Fixed an error that could occur when loading a project that uses very old flow syntax with tasks instead of steps. (#3118)
- Fixed a bug when using `sources` to access another project from a github branch, where the branch context was not available to branch-based dependency resolvers. (#3123)
- The `drop_missing_schema` option was not respected by the `snowfakery` task. Now it is. (#3125)

15.1.70 3.54.0 (2022-03-10)

Changes

- The `install_managed` and `update_dependencies` tasks:
  - Now support `apex_compile_type` and `upgrade_type` options for performing Unlocked Package installations. (#3105)
  - Accept the options `interactive` and `base-package-url-format`, which pause the automation and wait for the user to confirm the list of packages to be installed. (#3093)
- The `update_package_xml` task now supports new Slack metadata types (#3103)
- We added the ability to do Upserts during Data Loading (documentation). (#3099)
- Robot Framework tests can now pass a locator when calling `Go To Page` for one of our bulit-in page objects (Listing, Detail, Home), and the keyword will wait until that locator is visible. This is mostly useful for custom pages where you need to wait for an iframe or custom component to be available. (#3098)
- We improved our caching mechanisms to better handle projects whose cross-project sources themselves have cross-project sources. (#3092)

Issues Closed

- Fixed a bug running the `snowfakery` task in parallel mode with orgs connected using a Connected App. (#3109)

15.1.71 3.53.0 (2022-02-24)

Critical Changes

- CumulusCI now ships with Snowfakery 3.0. See the Snowfakery release notes. Snowfakery 3.0 ships with opt-in features that will become the default in June 2022; please test your recipes now with `snowfakery_version: 3` to ensure compatibility.

Changes

- CumulusCI now has an `update_data` task that allows updates to existing datasets, in coordination with Snowfakery 3.0.
- Orgs connected using `cci org connect` now keep track of which `connected_app` service was used to connect them, and will use the same `connected_app` service when getting a fresh access token, even if it isn't currently the default `connected_app` service. This makes it easier to use CumulusCI with multiple `connected_app` services. (Orgs that were connected before this update will continue to use the default `connected_app` service.)
- When connecting a custom `connected_app` service, a default `login_url` can be specified (to use a login URL other than https://login.salesforce.com when connecting orgs using this `connected_app` service).
- The `deploy_marketing_cloud_package` task now utilizes the new API endpoint for deployments.
- CumulusCI commands are more resilient in the face of corrupted or impossible to decode org config files.
- We added documentation about managing Robot Framework locators
- CumulusCI will now warn Windows users if long pathname support is not set up correctly.

Issues Closed

- Fixed a bug in the JSON format output of cci service list –json

15.1.72 3.52.0 (2022-02-03)

Changes

- Flow steps can now be replaced with the same syntax for all step types! (Current step types are: 'task' or 'flow'). See the [replacing a flow step](https://cumulusci.readthedocs.io/en/latest/config.html?highlight=override#replace-a-flow-step) docs for more details. CumulusCI is still compatible with the old syntax which required setting the current step type to `None` when replacing with a step of a differing type. (#3043)
- Whenever possible, while running the `robot_libdoc` task libdoc generates relative pathnames when creating CSV output (#3058)
- Added a new option `--preview` to the `robot_libdoc` task. When set to `true` it automatically opens a browser window to the generated documentation. (#3057)
- The robot keyword `Locate element by label` has been removed from the Salesforce.py library. This wasn't designed to be a keyword but was accidentally exported as one. If you want to find an input or textarea element by its label you can use a locator of the form `label:<text>` (eg: `label:First Name`) (#3048)
- Updated to Snowfakery 2.5.0
- If you have Python code that is importing CumulusCI's config classes, some of them have been reorganized into modules with a snake_case name. The old CamelCase imports should still work, but the new names are preferred for consistency with other parts of the codebase.

Issues Closed

- Fixed an issue where overriding flow steps was not working as documented. (#3043)

- We fixed a bug where the `Run Task` and `Run Task Class` robot keywords throw an error like `AttributeError: module 'robot.api.logger' has no attribute 'log'` in rare cases (#3053)
- Fixed a bug in the *snowfakery* task which caused Unique IDs to not always be actually unique. (#3059)

15.1.73 3.51.1 (2022-01-25)

Issues Closed

- We fixed an issue that could cause `create_package_version` to fail when the `version_base` option is set to `latest_github_release` and the latest GitHub release is a 1GP package version.

15.1.74 3.51.0 (2022-01-20)

Changes

- The `cci flow doc` command now only includes CumulusCI's standard flows unless the `--project` option is specified. (#3033)
- The `run_tests` task now has a `required_per_class_code_coverage_percent` which ensures that every class in your project meets the code coverage level specified. (#3027)
- Marketing Cloud tasks:
  - Added the `marketing_cloud_get_user_info` task to retrieve user information from the Marketing Cloud REST API `userinfo` endpoint. (#3039)
  - The `marketing_cloud_create_user` task now creates an unlocked user with a notification email address so that it is possible for the user to log in. This task also has a new option, `activate_if_existing`, which can be set to `true` to ensure that if the user already exists in an inactive state, it will be activated. (#3040)

Issues Closed

- Fixed an issue preventing step-level preflight checks from working correctly in MetaDeploy when run in a cross-project flow. (#3034)
- The `github_parent_pr_notes` task now handles child pull requests with an empty body. (#3038)
- The `metadeploy_publish` task now displays a clear error message if you supply the wrong API URL for MetaDeploy. (#3034)
- The `cci service default --project` command presents a better error message when called outside of a project directory. (#3037)
- Fixed a bug where the `cci flow doc` command would break when using cross-project flows. (#3033)

15.1.75 3.50.0 (2022-01-06)

Changes

- Robot keywords have been updated to support the Spring 22' release. (#3021)
- Using channel declarations in `load.yml`, users can now do synthetic data loads across multiple user accounts at once. This is faster for some very large orgs. (#3016)

Issues Closed

- Added a missing image to the "Windows Install Steps" portion of the docs. (#3013)

- Fixed a bug that prevented `Get webelements` from returning an empty list if the locator was a custom locator created via the `register_locators` function of `cumulusci.robotframework.locator_manager`. (#3004)
- Fixed a bug that prevented the locator `SF:object.button:Assign Reviewers` from working properly. (#3002)
- Fixed an issue where the `--json` flag was not outputting properly formatted `JSON` with the `cci task list` command. (#3011)

15.1.76 3.49.0 (2021-12-09)

Critical Changes

- **Python versions 3.6 and 3.7 are no longer supported.** Please ensure you have Python version 3.8, 3.9, or 3.10. (#2959)
- The `dx_convert_from` task now uses a custom Python task class instead of `cumulusci.tasks.sfdx.SFDXBaseTask`. In most cases this will have no visible impact, but if you have customized this task config to change the command option, make sure you have also explicitly set its `class_path` to `cumulusci.tasks.sfdx.SFDXBaseTask`. (#2981)

Changes

- You can now get the CumulusCI version with `cci --version`. The output is the same as the `cci version` command. (#2974)
- A new feature has been added to the `robot_libdoc` task. You can now use `-f csv` to generate a CSV file with all of the keywords. (#2985)
- New keywords `Select Rows` and `Unselect Rows` have been added to the robot Listing page object (#2995)
- Queries passed to the `SOQL Query` keyword can now span multiple lines. (#3006)

Issues Closed

- Fixed an issue where deleted components could still deploy into persistent orgs. (#2981)

15.1.77 3.48.2 (2021-11-16)

- Fixed a regression which broke `cci project init` in CumulusCI 3.48.0 and 3.48.1. (#2986)

15.1.78 3.48.1 (2021-11-12)

Issues Closed

- Fixed a packaging issue which caused an error when installing on systems without a C compiler.

15.1.79 3.48.0 (2021-11-11)

Critical Changes

- CumulusCI will be dropping support for Python 3.6 and 3.7 within the next few releases. Please ensure you're running Python 3.8 or above.

Changes

- We added a new command, `cci plan info`. Similar to `cci task info`, this command displays detailed information about a MetaDeploy `plan`, and includes a `--messages` option to display user-facing text. (#2946)
- Improved logging to show reduced timestamps, and provide some syntax highlighting of output. (#2941)
- Snowfakery 2.2 is now included with CumulusCI. New features include unique IDs and Numeric Counters. (#2962)

Issues Closed

- We added an improved error message when `metadeploy_publish` is passed a lightweight tag. (#2955)
- Adjusted a check for .lightning. domains in the `cci org connect` command. (#2970)
- We fixed an issue where stacktraces for some Apex test failures were truncated. (#2961)

15.1.80 3.47.0 (2021-10-28)

Changes

- Added a `cci plan list` command for displaying a list of MetaDeploy plans (#2940)
- Task options can now be marked as "sensitive". These task options that will be obfuscated when displayed at the beginning of each task in a flow. (#2939)
- Improved error message when *uninstall_packaged_incremental* is run in an SFDX-format project without converting to Metadata API format first (#2929)
- Improved error messaging for multiple scenarios where tasks are improperly configured in `cumulusci.yml`. (#2923)
- We added a new task *create_blank_profile* that can be used to create a new profile from scratch without any permissions enabled. (This new task requires a Winter 22 Org or API 53.0) (#2908)
- We've added a user-friendly error message when installing a package using `security_type` "PUSH" with a 04t Package Version ID. (#2935)

15.1.81 3.46.0 (2021-10-14)

Critical Changes

- Backwards incompatibility: the robot task option `debug` has been renamed to `robot_debug`. (#2909)

Changes

- CumulusCI now has a schema published here. This is primarily intended to be use for enabling linting in VS Code, but could be used for any schema-aware editor or any validation purpose. (#2902)
- We added a new task *create_blank_profile* that can be used to create a new profile from scratch without any permissions enabled. (#2908)

- Manually creating a Personal Access Token and pasting it into the CLI is no longer required to connect a GitHub service. Instead, CumulusCI now supports GitHub's device authentication flow, allowing you to authenticate via browser using a temporary device code. (#2911)
- `cci service info` for a github service now displays expiration dates for GitHub personal access tokens, if set. (#2912)
- Improved error messaging for multiple scenarios where tasks are improperly configured in `cumulusci.yml`. (#2923)

Issues Fixed

- Fixed a bug where connecting a GitHub service with `cci service connect` was failing silently. (#2888)
- Fixed an issue where MetaDeploy steps using the old `filename_token` and `namespace_token` options could not be used. (#2914)

15.1.82 3.45.0 (2021-09-30)

Changes

- Updated the `deploy_marketing_cloud_package` task for compatibility with the October 2021 release of Marketing Cloud. (#2899)
- The `--max-lines` option on the `cci error info` command has been removed. (#2895)

Issues fixed

- Removed the unused `--skip` option for the `cci flow run` command. (#2884)
- Flow descriptions no longer generate a warning. (#2885)
- We changed how the output from some commands and tasks are displayed in the CLI. (#2887)
- Fixed a bug in freezing the `load_dataset` task options for MetaDeploy. (#2900)
- The marketing cloud `deploy` task now properly exits when a result status of `FATAL_ERROR` is returned. (#2897)
- We fixed a regression in the `push_list` task that affected 2GP push upgrades (#2898)

15.1.83 3.44.1 (2021-09-17)

Issues Fixed

- We fixed a regression that resulted in upload failures for 2GP packages that extend 1GP packages (closes ##2880).

15.1.84 3.44.0 (2021-09-16)

Changes

- CumulusCI uses package version Ids from 1GP releases wherever available, reducing the need to install 1GP packages in an org to build 2GP dependencies. (#2832)
- We added `metadata_package_id` and `version_id` options to allow passing `MetadataPackage` (prefix 033) and `MetadataPackageVersion` (prefix 04t) IDs to the push upgrade tasks. (#2837)
- `cci flow info` will now output all flow options defined. (#2845)
- We improved error messages for incorrect data mapping files (#2831).
- The `snowfakery` task supports specifying the `loading_rules` option (#2861)

- The `snowfakery` task supports recipe options being supplied to recipes using the `recipe_options` option (#2861).

Issues Fixed

- We fixed some errors in the documentation (#2854)
- We fixed an issue causing CumulusCI to fail to install releases that contain an Unlocked Package without a namespace. (#2851)
- We added handling for issues that occur when running the `generate_dataset_mapping` task for very large orgs (#2860).
- We fixed a regression in using cross-project sources in MetaDeploy installers (#2875).

15.1.85 3.43.0 (2021-09-02)

Critical Changes

- We now support all package installation options for the `update_dependencies` and `install_managed` tasks, including `activate_remote_site_settings`, `security_type`, `name_conflict_resolution`, and `password` (password not available for `update_dependencies`). (#2811)

  We also fixed a minor inconsistency in defaulting the `activate_remote_site_settings` (or formerly `activateRSS`) option. Projects that define custom tasks based on the `InstallPackageVersion` class should ensure they explicitly set the `activate_remote_site_settings` option, or accept the new default of True.

  MetaDeploy install plans now *do not* freeze defaulted package install options. If your install plans are dependent on specific install options, we recommend explicitly specifying them. Install plans without explicit options will use the defaults at the time of execution.

Changes

- The `sources` feature, which allows CumulusCI projects to consume automation from other projects, now supports specifying a `resolution_strategy`, just like dependencies. Sources can now resolve to the same GitHub refs as corresponding dependencies, including branch matching. The default behavior is to use the `production` resolution strategy. (#2807)
- Added several new tasks for configuring Marketing Cloud: `marketing_cloud_create_subscriber_attribute`, `marketing_cloud_create_user`, and `marketing_cloud_update_user_role`. (#2838)
- In the mapping file for the `load_dataset` task, the `batch_size` can now be specified for Bulk API steps in addition to REST API steps. (#2813)
- The `snowfakery` task now supports the `ignore_row_errors` option to continue loading even if there are row errors. (#2819)
- We made significant updates to the documentation for Robot Framework. (#2834, ##2847)
- We improved option validation for the `add_page_layout_fields` task. (#2828)

Issues Fixed

- Fixed handling of timezones when the `start_time` option is specified for the push tasks. (#2814)
- Fixed the `deploy_marketing_cloud_package` task to handle changes to the Marketing Cloud API. (#2816)
- Fixed an issue where MetaDeploy install steps that used 04t package version Ids, including 2GP installations, were frozen with incorrect titles. (#2817)
- Fixed an issue causing 2GP commit-status builds to fail when the local Git repository has a detached HEAD (#2818)

- Fixed a bug in the `dry_run` option for the `metadeploy_publish` task where explicitly setting the option to `False` did not disable the dry run. (#2836)
- Improved the error message shown by the `load_dataset` task if a table is missing from the dataset. (#2813)
- Improved the warning message shown when CumulusCI can't encrypt org and service config files. (#2839)

Internal Changes

- CumulusCI has improved infrastructure for its own integration tests. (#2783)
- Filing a CumulusCI issue on GitHub now presents a form to enter details. (#2829)
- Added a linter to ensure consistent formatting of YAML files within the CumulusCI codebase. (#2844)

15.1.86 3.42.0 (2021-08-19)

Critical Changes

- The `github_release` task now requires the `tag_prefix` option to be passed, because for 2nd-generation packages we can't tell from the version number whether it is a beta or not. We've updated the standard release flows to set the `tag_prefix` appropriately, but if you have custom flows using this task you will need to update them. (#2792)
- In order to run the `github_copy_subtree` task for a specific package version, you must now use the `tag_name` option instead of the `version` option. Using the `version` option set to `latest` or `latest_beta` is deprecated; it's preferred to pass these values in the `tag_name` option instead. (#2792)

Changes

- The `uninstall_packaged_incremental` task now defaults to ignoring non-deletable `CustomObjectTranslation` metadata. If your project customizes the `ignore_types` option on `uninstall_packaged_incremental`, we recommend you add `CustomObjectTranslation` to this option. (#2790)

Issues Fixed

- Fixed an issue where bulk job results were being miscounted. (Thanks @sfdcale!) (#2789)
- Fixed an issue where GitHub tags for a 2GP package would always include the "release" prefix (even for Beta package versions). (#2792)

15.1.87 3.41.0 (2021-08-05)

Changes

- We added a new Metadata ETL task, *add_page_layout_fields*, that allows adding fields to existing layouts. (#2766)
- We added a task to enable an Einstein prediction: *enable_einstein_prediction* (thanks, @erikperkins!) (#2778)
- We added standard flows for releasing unlocked packages: *release_unlocked_beta*] and *release_unlocked_production* (#2768)
- We added documentation for using CumulusCI to build managed 2GP packages, unlocked packages, and extending NPSP and EDA with 2GP packages. (#2768)
- Contributions to CCI now require verification by isort, which ensures consistency in the order that imports are used. (#2770)
- CumulusCI now supports deploying unmanaged dependencies in SFDX source format. (#2735)

- The *create_package_version*] task now handles dependencies that use a `zip_url`. (#2735)
- Updates to Github Actions configuration documentation. Thanks @Julian88Tex (#2773)
- CumulusCI now automatically recognizes services and orgs configured via environment variables. See the docs for more details. (#2676 and ##2776)
- We've updated the Push Upgrade tasks (*push_list*, *push_sandbox*, etc) task option `start_time` to accept ISO-8601 formatted datetimes. (#2769)
- You can now specify "sandbox": true on a `CUMULUSCI_ORG_*` variable in headless environments to indicate that the org you want to connect to is a sandbox. when connecting sandbox orgs in a headless environment. (#2753)

Issues Closed

- Fixed an issue where scratch orgs failed to be deleted in CI environments. (#2676)
- Fixed an issue where deleting an org failed to mark the org as deleted on CumulusCI's keychain. (#2676)
- Fixed an issue where CumulusCI would fail on Linux distributions that were incompatible with the `keyring` package. (#2676)
- We fixed an issue causing the *release_2gp_production* flow to fail with a dependency parsing error. (#2767)
- Fixed a couple issues with connecting CumulusCI to sandboxes using enhanced domains. (#2753 and ##2765)
- Fixed a bug where the github_release task was not marking the "This is a pre-release" checkbox for beta releases. (#2788)

15.1.88 3.40.1 (2021-07-22)

Issues Closed

- Fixed an issue where a missing dependency was causing the homebrew installer formula to break.

15.1.89 3.40.0 (2021-07-22)

Critical Changes

- The `create_package_version` task no longer creates Unlocked Packages from the `unpackaged/pre` and `unpackaged/post` directories of dependencies, or local `unpackaged/pre` directories by default. This behavior is now opt-in via the `create_unlocked_dependency_packages` option, which defaults to False. Projects using the old default behavior must explicitly set this option. We believe the new behavior is a more sane default for most 2GP projects. (#2741)

Changes

- The `add_standard_value_set_entries` task now supports value sets for `LeadStatus`. (#2695, with thanks to @naicigam)
- We updated the default API version to 52.0. (#2740)

Issues Closed

- Fixed an issue where the the built-in connected app was not accessible when running CumulusCI in a headless environment. (#2737)
- The `create_package_version` task now supports `objectSettings` in the org definition file. (#2741)
- We fixed issues in working with files containing Unicode characters on some Windows systems when using source-tracking commands. (#2739)
- Fixed a bug where the `anon_apex` task had option text that was missing spaces. (#2736)

15.1.90 3.39.1 (2021-07-08)

Changes:

- Fix a bug with the integration of CumulusCI and the new SOQLQuery Feature

15.1.91 3.39.0 (2021-07-08)

Changes:

- A new `snowfakery` task with better usability and multi-processor support. Look at the CumulusCI docs to learn the new syntax: https://cumulusci.readthedocs.io/en/stable/data.html#generate-fake-data (#2705)
- CumulusCI now uses Snowfakery 2.0, with various new features, especially the ability to query into orgs. More information: https://github.com/SFDO-Tooling/Snowfakery/releases/tag/2.0 (#2705)
- We improved our Robot documentation so that it's possible to link to keyword documentation instead of having to download it locally (#2696)
- CumulusCI uses a new port (7788) for the built-in connected app to lessen the chances that the port is in use. (#2698)
- CumulusCI now checks if the port associated with a callback URL/redirect URI is in use during OAuth2 flows, and if so, raises a more friendly error. (#2698)
- The `generate_data_dictionary` task now includes Custom Settings, Custom Metadata Types, and Platform Events. (#2712)
- The `generate_data_dictionary` task now excludes any schema with visibility set to Protected. This behavior can be turned off (including protected schema) with the `include_protected_schema` option. (#2712)
- The `generate_data_dictionary` task now parses object and field metadata anywhere in a Salesforce DX release other than in the `unpackaged/` directory tree. (#2712)
- Builds that install feature-test 2GP packages now present a cleaner error message when the current commit is not found on GitHub. (#2713)
- SFDX and CumulusCI both support noancestors as a Scratch org config option but CumulusCI generated a warning if users tried to specify the option in cumulusci.yml. (#2721)

Issues closed:

- Fixed issue where CumulusCI did not correctly convert a package version specified as a number in YAML to a string. This now raises a warning. (#2692)
- Fixed a bug where OAuth errors were not reported in detail. (#2694)
- Fixed an issue where CumulusCI did not grant permissions to Custom Tabs when running `update_admin_profile` without a custom `package.xml`. Projects that use a custom `package.xml` with `update_admin_profile` should update their manifest to include a `CustomTab` wildcard for the same outcome. (#2699)
- Fixed an issue where the `dx`, `dx_push`, and `dx_pull` tasks did not refresh the org's access token. (#2703)
- Fixed issues in the `generate_data_dictionary` task that resulted in failures when processing fields with blank Help Text or processing standard fields. (#2706)
- Fixed an issue preventing `generate_data_dictionary` from working with four-digit (1.0.0.0) 2GP version numbers. (#2712)
- Fixed an issue causing `release_2gp_beta` to fail to create a GitHub release with a dependency-parsing error. (#2720)

15.1.92 3.38.0 (2021-06-24)

Changes:

- The built-in connected app that CumulusCI uses by default is now visible in the output of the `cci service list` command. This makes it possible to switch back and forth between this connected app and another one as the current default when multiple connected_app services are configured. The built-in connected_app service has the name `built-in` and cannot be renamed or removed. (#2664)
- The `generate_data_dictionary` task includes a new option, `include_prerelease`. If set to `True`, CumulusCI will include unreleased schema in the data dictionary from the current branch on GitHub, with the version listed as "Prerelease". (#2671)
- Added a new task, `gather_release_notes`, which generates an HTML file with release notes from multiple repositories. (#2633)
- The `deploy_marketing_cloud_package` task includes a new option, `custom_inputs`, which can be used to specify values to fill in for inputs in a Marketing Cloud package. (#2683)
- Mappings for the `extract_dataset` task can now specify a `soql_filter` to restrict which records are extracted. Thanks @sfdcale (#2663)
- Robot Framework: The `Scroll Element Into View` keyword in the Salesforce library now scrolls the center of the element into view rather than the top. (#2689)

Issues closed:

- Fixed a bug where CumulusCI could not parse the repository owner and name from an ssh git remote URL if it used an ssh alias instead of `github.com`. (#2684)
- Fixed a bug where `cci service info <service_type>` would display `None` as the name for the default service if no name was provided. (#2664)
- Fixed a missing dependency on the `contextvars` Python package in Python 3.6.

15.1.93 3.37.0 (2021-06-10)

Changes

- The `install_managed` task now supports 2GP releases (#2655).
- We changed the behavior of the `release_2gp_beta` flow to always upload a package version, even if metadata has not changed (#2651).
- We now support sourcing install keys for packages from environment variables via the `password_env_name` dependency key (#2622).

Robot Framework

- We upgraded SeleniumLibrary to 5.x (#2660).
- We added a new keyword "select window" to Salesforce library, to replace the keyword of the same name which was renamed in SeleniumLibrary 5.x to 'switch window'. We will be removing this keyword in a future release; tests should use 'switch window' instead.

Issues Closed

- We corrected some JavaScript issues that were occurring with Chrome 91. (#2652)
- We fixed a bug impacting the `generate_data_dictionary` task when used with dependencies (#2653).
- We fixed an issue causing `sfdx` commands that had options with spaces to fail to execute on Windows (#2656).

- We fixed an issue causing the creation of incorrect 2GP beta tags (#2651).

15.1.94 3.36.0 (2021-05-27)

Changes

- Added the option `tag_prefix` to the `github_release` task. This option can be set to specify what prefix you would like to use when CumulusCI creates a release tag for you in GitHub. (#2642)
- The `deploy_marketing_cloud_package` task has been updated to match changes to the Marketing Cloud Package Manager API. It also now raises an exception if the deployment failed. (#2632)

Robot Framework

- Improved the output of the `robot_libdoc` task. (#2627)

Data generation with Snowfakery:

- Updated to Snowfakery 1.12 (#2538)

Issues Closed

- Fixed an issue where flow reference documentation was rendering with an error. (#2646)
- CumulusCI will now remove orgs when the `--delete-org` option is passed to `cci flow run`, even if an error occurs while running the flow. (#2644)
- Fixed a bug where beta tags created via the `release_2gp_beta` flow were not receiving the proper tag prefix. (#2642)
- Fixed namespace injection for filenames with a `___NAMESPACE___` token in sfdx format. (#2631) (Thanks @bethbrains)
- Fixed a bug in `cci org connect` where the `--sandbox` flag was directing users to login at `login.salesforce.com` instead of `test.salesforce.com`. (#2630)
- Fixed a regression where the `skip` key for a dependency could no longer be specified as a single string instead of a list. (#2629)
- Fixed a regression in freezing the `deploy_pre`/`deploy_post` tasks for MetaDeploy install plans. (#2626)
- Fixed bugs in the `deploy_marketing_cloud_package` task's payload construction. (#2620, ##2621)

15.1.95 3.35.0 (2021-05-13)

Critical Changes

- Upgraded Robot Framework to 4.x. For information about new features and some backward incompatibilities see the Robot Framework 4.0 release notes. (#2603)
- The `update_dependencies` task now guarantees to resolve unpackaged metadata directories (subdirectories of `unpackaged/pre` and `unpackaged/post`) in alphabetical order, matching the behavior of `deploy_pre` and `deploy_post`. `unpackaged/pre/bar` will deploy prior to `unpackaged/pre/foo`. The previous behavior was undefined, which caused rare problems. This change is critical only for projects that have deployment-order dependencies between unpackaged directories located in upstream dependencies and rely on the current undefined load order. (#2588)

Changes

- The CumulusCI documentation has a new section: Testing with Second-Generation Packaging (#2597)

- CumulusCI has two new service types: `oauth2_client` & `marketing_cloud`. These are considered experimental. (#2602)
- The `marketing_cloud` service allows users to connect to a Marketing Cloud tenant via OAuth so that tasks that work with Marketing Cloud can make API calls on the user's behalf. (#2602)
- The `oauth2_client` service takes information for an individual OAuth2 client which can then be used in place of the default client. This currently applies only to the `marketing_cloud` service. To setup a Marketing Cloud service with a specific OAuth2 client use: `cci service connect marketing-cloud <name-of-service> --oauth_client <name-of-oauth-client>`. (#2602)
- CumulusCI has a new task: `deploy_marketing_cloud_package`. This task allows a user to pass the path to a .zip file to a Marketing Cloud package (downloaded from the Marketing Cloud Package Manager) and deploy the package via a `marketing_cloud` service (see above). Note that successfully deploying a package using this task may require permissions that are not generally available. (#2602)
- The `install_managed` and `install_managed_beta` tasks now take no action if the specified package is already installed in the target org. (#2590)
- The `cci org list` command can now output in `JSON` format by passing it the `--json` flag. (#2593)

Issues Closed

- Fixed an issue parsing `cumulusci.yml` files that contained Unicode characters on Windows. (#2617)
- Fixed an issue in the `github_copy_subtree` task where CumulusCI would silently generate incorrect or truncated commits when a directory was passed to the `include` task option. (#2601)
- The `deploy_pre` and `deploy_post` tasks avoid warnings by freezing installer steps that match current expectations. (#2589)

15.1.96 3.34.1 (2021-04-30)

Issues Closed

- Fixed a regression in the `load_dataset` task where some sObjects could not be loaded without explicitly turning off the new `set_recently_viewed` option.

15.1.97 3.34.0 (2021-04-29)

Critical Changes:

- If you have custom flows that utilize the `github_release` task, they will need to be updated to include the `package_type` option (which is required). (#2546)

Changes:

- The `github_release` task now has a `package_type` option which is included in the information written to GitHub release tags. The following standard library "release" flows have been updated with hardcoded values (either `1GP` or `2GP`) for this option:

  : - `release_beta` (1GP) - `release_production` (1GP) - `release_2gp_beta` (2GP) - `release_2gp_production` (2GP)

  (#2546)

- The `update_dependencies` task now supports a `packages_only` option, which suppresses the installation of unpackaged metadata dependencies. This option is intended to support building update-only or idempotent installers. (#2587)

- The `github_automerge_main` task has a new option, `skip_future_releases`, which can be set to `False` to disable the default behavior of skipping branches that are numeric (and thus considered release branches) but not the lowest number. (#2582)
- Added an new option `set_recently_viewed` to the `load_dataset` task that sets newly inserted data as recently viewed. This changes the default behavior. By default (if you do not specify the option), the first 1000 records inserted via the Bulk API will be set as recently viewed. If fewer than 1000 records are inserted, existing objects of the same type being inserted will also be set as recently viewed. (#2578)
- The `update_dependencies` task can now consume 2GP releases in upstream repositories, provided they're stored in release tags as generated by CumulusCI. (#2557)
- The `extract_dataset` and `load_datast` tasks now support adding or removing a namespace from a mapping file to match the target org for sObjects and not just fields. (#2532)
- The `create_package_version` task can now increment package version numbers when the package is not in a released state. (#2547)
- Includes Snowfakery 1.10 with upgrades to its Fake data functions.

Issues Closed

- Fixed an error in the `github_automerge_main` task when using a branch prefix that doesn't contain a slash. (#2582)
- Fixed logic in the `push_sandbox` and `push_all` tasks which was selecting the wrong package versions. (#2577)
- Improved logging of errors from sfdx while converting sfdx format metadata to deploy via the Metadata API, so that they are not lost when CumulusCI is embedded in another system like MetaCI or Metecho. (#2574)

15.1.98 3.33.1 (2021-04-20)

Changes:

- The `create_package_version` task now accepts an `--ancestor-id` option to specify the 04t Id of the package version that should be considered the ancestor of a new managed package version. The option can also be set to `latest_github_release` to look up the 04t Id of the project's most recent release on GitHub. (#2540)

Issues closed:

- Fixed a regression where the `release_beta` flow would throw an error if the project has unmanaged github dependencies. (#2566)
- Fixed a regression where the `cci service connect` command could no longer connect a service without giving it a name. Now a default name will be assigned. (#2568)
- Fixed a regression when resolving unpackaged dependencies from GitHub releases. (#2571)
- Fixed a regression with creating a scratch org if the devhub service was configured but not set as the default. (#2570)
- Improved the formatting of `cumulusci.yml` validation warnings. (#2567)

15.1.99 3.33.0 (2021-04-19)

Critical Changes:

- CumulusCI's dependency management modules have been rewritten. This grants new capabilities and removes some existing features. (#2456)
  - All package installations now perform retries if the package is not yet available.
  - Package installations are also retried on common row locking errors.
  - You can now obtain fine-grained control over how your projects resolve dependencies. It's much easier to control where your application uses beta managed packages and second-generation packages to satisfy dependencies.
  - You can now execute 2GP builds that use 2GPs from upstream feature branches matching your current branch, not just release branches.
  - The `update_dependencies` task no longer supports uninstalling managed packages in a persistent org as part of the dependency installation process.
  - The `update_dependencies` task no longer supports the `allow_newer` option, which is always True.
  - The install order of `update_dependencies` changes slightly where multiple levels of upstream dependency have `unpackaged/pre` metadata. Where previously one package's `unpackaged/pre` might be installed prior to its own upstream dependency, `unpackaged/pre` will now always be installed immediately prior to the repo's package.
  - Projects using unmanaged dependencies that reference GitHub subfolders will see a change in resolution behavior. Previously, a dependency specified like this:

    ```
    dependencies:
        - github: https://github.com/SalesforceFoundation/NPSP
          subfolder: unpackaged/config/trial
    ```

    would always deploy from the latest commit on the default branch. Now, this dependency will be resolved to a GitHub commit just like a dependency without a subfolder, selecting the latest beta or production release as determined by the chosen resolution strategy.
  - The `project__dependencies` section in `cumulusci.yml` no longer supports nested dependencies specified like this:

    ```
    dependencies:
        - namespace: "test"
          version: "1.0"
          dependencies:
            - namespace: "parent"
              version: "2.2"
    ```

    All dependencies should be listed in install order.

Changes:

- CumulusCI now supports named services! This means you can configure multiple services of the same *type* under different names. If you run `cci service list` you will note that your existing global services will have the name `global`, and any project-specific services will have the name `project_name`. (#2499)
  - You must now specify both a service type and a service name when connecting a new service using `cci service connect`.

  - CumulusCI has a new command: `cci service default`. This command sets the default service for a given type.
  - CumulusCI has a new command: `cci service rename`. This command renames a given service.
  - CumulusCI has a new command: `cci service remove`. This command removes a given service.
- A validator now checks `cumulusci.yml` and shows warnings about values that are not expected. (#1624)
- Added a friendly error message when a GitHub repository cannot be found when set as a dependency or cross-project source. (#2535)
- Task option command line arguments can now be specified with either an underscore or a dash: e.g. `clean_meta_xml` can be specified as either `--clean_meta_xml` or `--clean-meta-xml` or `-o clean-meta-xml` (#2504)
- Adjustments to existing tasks:
  - The `update_package_xml` task now supports additional metadata types. (#2549)
  - The `push_sandbox` and `push_all` tasks now use the Bulk API to query for subscriber orgs. (#2338)
  - The `push_sandbox` and `push_all` tasks now default to including all orgs whose status is not Inactive, rather than only orgs with a status of Active. This means that sandboxes, scratch orgs, and Developer Edition orgs are included. (#2338)
  - The `user_alias` option for the `assign_permission_sets`, `assign_permission_set_groups`, and `assign_permission_set_licenses` tasks now accepts a list of user aliases, and can now create permission assignments for multiple users with a single task invocation. (#2483)
  - The `command` task now sets the `return_values` to a dictionary that contains the return code of the command that was run. (#2453)
- Data generation with Snowfakery:
  - Updated to Snowfakery 1.9 (#2538)
- Robot Framework:
  - The `run task` keyword now includes all task output in the robot log instead of printing it to stdout. (#2453)
  - Documented the use of the options/options section of CumulusCI for the `robot` task. (#2536)
- Changes for CumulusCI developers:
  - Tasks now get access to the `--debug-mode` option and can output debugging information conditional on it. (#2481)
- `cci org connect` can now connect to orgs running in an internal build environment with a different port. (#2501, with thanks to @force2b)

Issues Closed:

- The `load_custom_settings` task now resolves a relative `settings_path` correctly when used in a cross-project flow. (#2523)
- Fixed the `min_version` option for the `push_sandbox` and `push_all` tasks to include the `min_version` and not only versions greater than it. (#2338)

15.1.100 3.32.1 (2021-04-01)

April Fool's! This is the real new release, because there was a packaging problem with 3.32.0.

15.1.101 3.32.0 (2021-04-01)

Changes:

- A new task, `create_network_member_groups`, creates NetworkMemberGroup records to grant specified Profiles or Permissions Sets access to an Experience Cloud site (community). (#2460, thanks @ClayTomerlin)
- A new preflight check task, `get_existing_sites`, returns a list of existing Experience Cloud site names in the org. (#2493)
- It is now possible to create a flow which runs the same sub-flow multiple times, as long as they don't create a self-referential cycle. (#2494)
- Improvements to support for releasing 2nd-generation (2GP) packages:
  - The `github_release` task now includes the package version's 04t id in the message of the tag that is created. (#2485)
  - The `promote_package_version` task now defaults to promoting the package version corresponding to the most recent beta tag in the repository, if `version_id` is not specified explicitly. (#2485)
  - Added a new flow, `release_2gp_beta`, which creates a beta package version of a 2GP managed package and a corresponding tag and release in GitHub. (#2509)
  - Added a new flow, `release_2gp_production`, which promotes a 2gp managed package version to released status and creates a corresponding tag and release in GitHub. (#2510)
- Data generation with Snowfakery:
  - Updated to Snowfakery 1.8.1 (#2516)
  - Snowfakery can now use "load files" to provide hints about how objects should be loaded.
  - Values for the `bulk_mode`, `api`, and `action` parameters in mapping files are now case insensitive.
- Robot Framework:
  - Added a new keyword, `Input Form Data`, for populating form fields of many different types. This keyword is considered experimental but is intended to eventually replace `Populate Form`. (#2496)
  - Added a new keyword, `Locate Element by Label`, for finding form inputs using their label. (#2496)
  - Added a custom locator strategy called `label` which uses `Locate Element By Label` (e.g. `label:First Name`). (#2496)
  - Added two new options to the robot task: `ordering` and `testlevelsplit`. These only have an effect when combined with the `processes` option to run tests in parallel.

Issues Closed:

- The `cci org import` command now shows a clearer error message if you try to import an org that is not a locally created scratch org. (#2482)

15.1.102 3.31.0 (2021-03-18)

Changes:

- It is now possible to pass the `--noancestors` flag to sfdx when creating a scratch org by setting `noancestors: True` in the scratch org config in `cumulusci.yml`. Thanks @lionelarmanet (#2452)
- The `robot_outputdir` return value from the `robot` task is now an absolute path. (#2442)
- New tasks:
  - `get_available_permission_sets`: retrieves the list of available permission sets from an org. (#2455)
  - `promote_2gp_package`: will promote a `Package2Version` to the "IsReleased" state, making it available for installation in production orgs. (#2454)

Snowfakery 1.7:

- Adds support for Salesforce Person Accounts.

Issues Closed:

- `cci project init` no longer overwrites existing files. If files already exist, it displays a warning and outputs the rendered file template. (#1325)

15.1.103 3.30.0 (2021-03-04)

Critical changes:

- We are planning to remove functionality in CumulusCI's dependency management in a future release.
  - The `update_dependencies` task will no longer support uninstalling managed packages in a persistent org as part of the dependency installation process.
  - The `allow_newer` option on `update_dependencies` will be removed and always be True.
  - The `project__dependencies` section in `cumulusci.yml` will no longer support nested dependencies specified like this :

```
dependencies:
  - namespace: "test"
    version: "1.0"
    dependencies:
      - namespace: "parent"
        version: "2.2"
```

    All dependencies should be listed in install order.

    We recommend reformatting nested dependencies and discontinuing use of `allow_newer` and package uninstalls now to prepare for these future changes.

Changes:

- We released a new suite of documentation for CumulusCI.
- CumulusCI now caches org describe data in a local database to provide significant performance gains, especially in `generate_dataset_mapping`.
- The `cci org browser` command now has a `--path` option to open a specific page and a `--url-only` option to output the login URL without spawning a browser.
- We improved messaging about errors while loading `cumulusci.yml`.

- CumulusCI now uses Snowfakery 1.6 (see its release notes).

15.1.104 3.29.0 (2021-02-18)

Changes:

- The message shown at the end of running a flow now includes the org name. ##2390, thanks @Julian88Tex
- Added new preflight check tasks:
  - `get_existing_record_types` checks for existing Record Types. ##2371, thanks @ClayTomerlin
  - `get_assigned_permission_sets` checks the current user's Permission Set Assignments. ##2386
- The `generate_package_xml` task now supports the Muting Permission Set metadata type. ##2382
- The `uninstall_packaged_incremental` and `uninstall_packaged` tasks now support a `dry_run` option, which outputs the destructiveChanges package manifest to the log instead of executing it. ##2393
- Robot Framework:
  - The `Run Task` keyword now uses the correct project config when running a task from a different source project. ##2391
  - The SalesforceLibrary has a new keyword, `Scroll Element Into View`, which is more reliable on Firefox than the keyword of the same name in SeleniumLibrary. ##2391

Issues closed:

- Fixed very slow `cci org connect` on Safari. ##2373
- Added a workaround for decode errors that sometimes happen while processing cci logs on Windows. ##2392
- If there's an error while doing JWT authentication to an org, we now log the full response from the server. ##2384
- Robot Framework: Improved stability of the `ObjectManagerPageObject`. ##2391

15.1.105 3.28.0 (2021-02-04)

Changes:

- Added a new task, `composite_request`, for calling the Composite REST Resource. ##2341
- The `create_package_version` task has a new option, `version_base`, which can be used to increment the package version from a different base version instead of from the highest existing version of the 2gp package. The `build_feature_test_package` flow now uses this option to create a package version with the minor version incremented from the most recent 1gp release published to github. ##2357
- The `create_package_version` task now supports setting a post-install script and uninstall script when creating a managed package version, by setting the `post_install_script` and `uninstall_script` options. By default, these options will use the values of `install_class` and `uninstall_class` from the `package` section of `cumulusci.yml`. ##2366
- Updated to Snowfakery 1.5.
- Robot Framework:
  - The `Click related list button` keyword has been modified to be more liberal in the types of DOM elements it will click on. Prior to this change it only clicked on anchor elements, but now also works for related list buttons that use an actual button element. ##2356
  - The `Click modal button` keyword now attempts to find the given button anywhere on the modal rather than only inside a `force-form-footer` element. ##2356

Issues closed:

- Robot Framework:
  - Custom locators can now be used with keywords that expect no element to be found (such as `Page should not contain`). This previously resulted in an error. ##2346
  - Fixed an error when setting the `tagstatexclude` option for the `robot` task. ##2365
- Fixed a possible error when running CumulusCI flows embedded in a multi-threaded context. ##2347

15.1.106 3.27.0 (2021-01-21)

Changes:

- Snowfakery 1.4 which includes min, max, round functions. PR ##2335
- The `ensure_record_types` task has a new option, `force_create`, which will create the Record Type even if other Record Types already exist on the object. (Thanks to @bethbrains) PR ##2323
- Allow num_records and num_records_tablename to be omitted when using the task generate_and_load_from_yaml. PR ##2322
- Added a new Metadata ETL task, add_fields_to_field_set which allows adding fields to existing field sets. (Thanks to @bethbrains) PR ##2334
- org_settings now accepts a dict option called settings in addition to (or instead of) the existing definition_file option. (Thanks to @bethbrains) PR ##2337
- New Robot Keywords for Performance Testing: ##2291
  - Set Test Elapsed Time: This keyword captures a computed rather than measured elapsed time for performance tests.
  - Start Perf Time, End Perf Time: start a timer and then store the result.
  - Set Test Metric: store any test metric, not just elapsed time.
- CumulusCI now reports how long it took for flows to run. ##2249

Issues Closed:

- Fixed an error that could occur while cleaning cache directories.
- Fixed potential bugs in the Push Upgrade tasks.
- CumulusCI displays more user friendly error message when encountering parsing errors in cumulusci.yml. ##2311
- We fixed an issue causing the extract_dataset task to fail in some circumstances when both an anchor date and Record Types were used. ##2300
- Handle a possible gack while collecting info about installed packages ##2299

15.1.107 3.26.0 (2021-01-08)

Changes:

- CumulusCI now reports how long it took for flows to run.
- Flows `ci_feature` and `ci_feature_beta_deps` now only run the `github_automerge_feature` task if the branch begins with the configured feature branch prefix.
- Running the `deploy` task with the `path` option set to a path that doesn't exist will log a warning instead of raising an error.
- When the `ci_feature_2gp` and `qa_org_2gp` flows install dependencies, the latest beta version will be used when available.
- CumulusCI can now resolve dependencies using second-generation packages (2GPs) for upstream projects. When a *ci_feature_2gp* or *qa_org_2gp* flow runs on a release branch (starting with `prefix/NNN`, where `prefix` is the feature branch prefix and `NNN` is an integer), CumulusCI will look for a matching release branch in each upstream dependency and use a 2GP package build on that release branch, if present, falling back to a 1GP beta release if not present.

Issues Closed:

- Fixed the `org_settings` task to handle nested structures in org settings.
- Fixed a bug where cci task run could fail without a helpful error if run outside of a cci project folder.
- Fixed an issue that caused CumulusCI to generate invalid `package.xml` entries for Metadata API-format projects that include `__mocks__` or `__tests__` LWC directories.
- Fixed the `update_dependencies` task to handle automatic injection of namespace prefixes when deploying an unpackaged dependency. The fix for the same issue in CumulusCI 3.25.0 was incomplete.
- Fixed an issue where an unquoted `anchor_date` in bulk data mapping failed validation.
- CumulusCI now handles an error that can occur while collecting info about installed packages
- Fixed an issue causing the `extract_dataset` task to fail in some circumstances when both an anchor date and Record Types were used.
- Fixed an issue where the deprecated syntax for record types was not working.

15.1.108 3.25.0 (2020-12-10)

Changes:

- New tasks:
  - `assign_permission_set_groups` assigns Permission Set Groups to a user if not already assigned.
  - `assign_permission_set_licenses` assigns Permission Set Licenses to a user if not already assigned.
- New preflight checks for use with MetaDeploy install plans:
  - `check_enhanced_notes_enabled` checks if Enhanced Notes are enabled
  - `check_my_domain_active` checks if My Domain is active
- The `github_copy_subtree` task has a new option, `renames`, which allows mapping between local and target path names when publishing to support renaming a file or directory from the source repository in the target repository.
- The `ensure_record_types` task has a new option, `record_type_description`, which can be used to set the description of the new record type if it is created.

- Robot Framework:
  - New keyword `Field value should be`
  - New keyword `Modal should show edit error for fields` to check form field error notifications in Spring '21
  - Adjusted `Get field value` and `Select dropdown value` fields to work in Spring '21
- Command line improvements:
  - The various `cci org` commands now accept an org name with the `--org` option, for better consistency with other commands. Specifying an org name without `--org` also still works.
  - Running `cci org default` without specifying an org name will now display the current default org.
- Org configs now have properties `org_config.is_multiple_currencies_enabled` and `org_config.is_advanced_currency_management_enabled` which can be used to check if these features are enabled.
- The `MergeBranchOld` task, which was previously deprecated, has now been removed.

Issues closed:

- Fixed the `update_dependencies` task to handle automatic injection of namespace prefixes when deploying an unpackaged dependency.
- Fixed the `query` task, which was completely broken.
- Fixed the `connected_app` task to pass the correct username to sfdx. Thanks @atrancadoris
- Fixed the display of task options with an underscore in `cci task info` output.
- Fixed a confusing warning when creating record types using Snowfakery. (#2093)
- Improved handling of errors while deleting a scratch org.

15.1.109 3.24.1 (2020-12-01)

Issues Closed:

- Fixed a regression that prevented running unmanaged flows on persistent orgs, due to the use of the `include_beta` option while installing dependencies, which is not allowed for persistent orgs. We changed the `update_dependencies` task to ignore the option and log a warning when running against a persistent org, instead of erroring.

15.1.110 3.24.0 (2020-11-30)

Critical Changes:

- The flows `dev_org`, `dev_org_namespaced`, `qa_org`, `ci_feature`, and `install_beta` now run the `update_dependencies` task with the `include_beta` option enabled, so dependencies will be installed using the most recent beta release instead of the most recent final release. The `beta_dependencies` flow is no longer used and is considered deprecated.
- The flows `ci_feature_beta_deps` and `dev_org_beta_deps` are now deprecated and should be replaced by their default equivalents above.
- The `ci_feature_2gp` flow has been changed to use `config_apextest` instead of `config_managed` to avoid configuration steps that are unnecessary for running Apex tests. This means that in order for `ci_feature_2gp` to work, `config_apextest` must be set up to work in both managed and unmanaged contexts.

- When connecting GitHub using `cci service connect github`, we now prompt for a personal access token instead of a password. (GitHub has removed support for accessing the API using a password as of November 2020.) If you already had a token stored in the `password` field, it will be transparently migrated to `token`. If you were specifying `--password` on the command line when running this command, you need to switch to `--token` instead.
- Removed the old `cumulusci.tasks.command.SalesforceBrowserTest` task class which has not been used for some time.

Changes:

- Added a standard `qa_org_2gp` flow, which can be used to set up a QA org using a 2nd-generation package version that was previously created using the `build_feature_test_package` flow. This flow makes use of the `config_qa` flow, which means that `config_qa` must be set up to work in both managed and unmanaged contexts. This flow is considered experimental and may change at any time.
- The `batch_apex_wait` task can now wait for Queueable Apex jobs in addition to batch Apex.
- The `custom_settings_value_wait` task now waits if the expected Custom Settings record does not yet exist, and does case insensitive comparison of field names.
- Preflight checks:
  - Added a task, `check_sobject_permissions`, to validate sObject permissions.
  - Added a task, `check_advanced_currency_management`, to determine whether or not Advanced Currency Management is active.
- Robot Framework:
  - In the Robot Framework Salesforce resource, the `Open Test Browser` keyword now accepts an optional `useralias` argument which can be used to open a browser as a different user. The user must already have been created or authenticated using the Salesforce CLI.
- Updated to Snowfakery 1.3.

Issues Closed:

- Improved error handling of REST API responses to confirm they are JSON.
- Fixed error handling in the `load_dataset` task in Windows.
- Fixed a bug where pressing `Ctrl+C` while running `cci org connect` in Windows did not exit. (#2027)
- Fixed a bug where deploying an LWC component bundle using the `deploy` task did not include files in subfolders.
- Fixed the `deploy` task so that deploying an empty metadata directory does not error.
- Fixed a bug where the `namespace_inject` option was not included when freezing deploy steps for MetaDeploy, causing namespace injection to not work when running the plan in MetaDeploy.
- Fixed a bug where running the `robot` task as a cross-project task could not load Robot Framework libraries from the other project.

15.1.111 3.23.0 (2020-11-12)

Changes:

- CumulusCI now accepts a normalized task option syntax in the form of: `--opt-name value`. This can be used in place of the old task option syntax: `-o opt-name value`.
- Tasks which perform namespace injection can now automatically determine whether they are running in the context of a managed installation or a namespaced scratch org. This means that in many cases it is no longer necessary to explicitly specify options like `managed/unmanaged/namespaced/namespaced_org/namespace_inject`, or to use a separate flow for namespaced scratch orgs.
- The `deploy_unmanaged` flow now deploys sfdx-formatted metadata using the Metadata API rather than the sfdx `force:source:push` command. This avoids an issue where sfdx could show an error about the pushed components conflicting with other changes that already happened in the org. It also improves consistency between how metadata is deployed to a scratch org and how it is deployed to a packaging org.
- Removed the `namespaced_org` option for the `update_dependencies` task, which was not functional.
- We added support for including SOQL where-clauses `Salesforce Query` Robot keyword via the `where` keyword argument.
- The `create_package_version` task can accept a `static_resource_path` option.
- The FindReplace task now has a `replace_env` option which, if true, will interpret the `replace` option as the name of an environment variable whose value should be used for the replacement.
- We added a new command, `cci project doc`, which will document project-specific tasks to a reStructuredText file.

Issues closed:

- An error that occurred when building a second-generation package using a cross-project task has been fixed.
- The `github_package_data` task will now work for projects using API versions prior to 44.0.
- Fixed a bug where namespace injection of the `%%%NAMESPACED_ORG%%%` token with the `namespaced_org` option enabled did not actually add the namespace prefix unless the `managed` option was also enabled.
- We fixed an issue that resulted in the `batch_size` option in a data mapping file being ignored.

15.1.112 3.22.0 (2020-10-29)

Changes:

- We added support for using Robot keywords from other projects that are included as `sources`.
  - The `suites` option of the robot task can now take a list of suite paths. Paths can include a prefix representing a remote repository as defined by the `sources` configuration option (eg: `-o suites npsp:robot/Cumulus/tests/api`)
  - The robot task has a new `sources` option to work in conjunction with the global `sources` option to allow the use of keywords and tests from other repositories.
  - When running the `robot` task, the folder containing downloaded repositories via the `sources` option are added to `PYTHONPATH` so that robot tests can find library and resource files in those repositories
- Bulk Data tasks now support adding or removing a namespace from a mapping file to match the target org.
- We improved how we parse Boolean values in Bulk Data tasks and in command line options. True can be represented as "yes", "y", "true", "on", or "1", with any capitalization, and False as "no", "n", "false", "off", "0". None as a synonym for False is deprecated.

- We added support for including managed package release details in automatically generated release notes.
- We added a task, `assign_permission_sets`, to assign Permission Sets to a user.
- We updated the default API version for new projects to 50.0.
- The `build_feature_test_package` flow now creates a 2GP package version with the "skip validation" option turned on.
- `github_automerge_main` now only merges to the lowest numbered release branch when multiple are detected.

Issues closed:

- We fixed an issue with relative imports within parallel Robot test runs by adding the repo root to PYTHONPATH.
- We fixed an issue with generating `package.xml` manifests for directories that contain reports in folders that aren't owned by the project.
- We now handle an exception that may occur while creating merge conflict PRs during parent-child automerges.

15.1.113 3.21.1 (2020-10-19)

Issues closed: - Added a workaround for a slow query error while looking up installed packages in Winter '21 orgs.

15.1.114 3.21.0 (2020-10-15)

Changes:

- The `update_admin_profile` task now accepts the `api_names` option to target extra Profiles, even when using a custom `package.xml`.
- The `github_automerge_main` task can now be used on source branches other than the default branch to merge them into branches starting with the `branch_prefix` option, as long as the source branch does not also start with `branch_prefix`.
- Added preflight check tasks to validate org settings (`check_org_settings_value`) and to check that Chatter is enabled (`check_chatter_enabled`). These are intended for use with MetaDeploy install plans.
- Updated to Snowfakery 1.2.

Issues closed:

- Fixed an issue in the `load_dataset` task which left out non-Person-Account Contacts if the dataset was extracted using the REST API.

15.1.115 3.20.1 (2020-10-05)

Issues closed:

- Fixed a bug introduced in CumulusCI 3.20.0 in which the `upload_beta` and `upload_production` tasks could hit a connection error if uploading the package took over 10 minutes.
- We corrected edge cases in how we processed Boolean options for the `custom_settings_wait`, `exec_anon`, and `uninstall_post` tasks. (Thanks to @davidjray)

15.1.116 3.20.0 (2020-09-30)

Critical Changes:

- We've removed the standard flow: `retrieve_scratch`. The recommended way for retrieving source-tracked changes is to use the `retrieve_changes` task.
- Changes to automatic merging:
  - The `github_master_to_feature` task has been renamed to `github_automerge_main`. It still merges changes from the default branch to feature branches. In the case of an orphaned feature branch (a branch with a name like `feature/parent__child` where `feature/parent` does not exist as its own branch), the `github_automerge_main` branch will no longer merge to the orphaned branch.
  - The `github_parent_to_children` task has been renamed to `github_automerge_feature`. It still merges changes from feature branches to their children (e.g. `feature/parent` would be merged to `feature/parent__child`). It is now possible to use multiple double-underscores to create more deeply nested children, and the task will only merge to the next level (e.g. `feature/parent` would merge to `feature/parent__child` which would merge to `feature/parent__child__grandchild`).
  - The `children_only` option for these tasks has been removed. The strategy for picking which branches to target for merging is now determined by the `source_branch`.

Tasks, Flows, and Automation:

- `cci flow list` now displays flows in different groups that are organized by functional area. (This is similar to how `cci task list` currently works).
- The `insert_record` task can now be used against the Tooling API. We clarified that this task can accept a dict of values if configured in `cumulusci.yml`.
- Added support for newer metadata types to the `update_package_xml` task.
- Previously, large data loads and extracts would use enormous amounts of memory. Now they should use roughly constant amounts of memory.
- Adjusted tasks: `install_managed` and `update_dependencies` can now install packages from just a version id (04t).
- Added support for creating 2GP packages (experimental)
  - New task: `github_package_data` gets a package version id from a GitHub commit status. It is intended primarily for use as part of the `ci_feature_2gp` flow. Implementation details can be found in the features section of the documentation.
  - New task: `create_package_version` - Builds a 2gp package (managed or unlocked) via a Dev Hub org. Includes some automated handling of dependencies:
  - New Flow: `build_feature_test_package` - Runs the `create_package_version task`, and in the context of MetaCI it will set a commit status with the package version id.
  - New Flow: `ci_feature_2gp` - Retrieves the package version from the commit status set by `build_feature_test_package`, installs dependencies and the package itself in a scratch org, and runs Apex tests. (There is another new task, `github_package_data`, which is used by this flow.)

User Experience:

- Improved error messaging when encountering errors during bulk data mapping validation.

Issues Closed:

- Fixed a very rare bug that caused CumulusCI to fail to retrieve installed packages from an org when running package-related tasks or evaluating `when` conditional expressions.

- Fixed `UnicodeDecodeError` while opening config files on Windows.
- Fixed a bug in `cumulusci.core.sfdx.sfdx` when capture_output is False

15.1.117 3.19.1 (2020-09-18)

Issues closed:

- Fixed an issue (#2032) where REST API data loads incorrectly handled Boolean values.

15.1.118 3.19.0 (2020-09-17)

Changes:

- Tasks and automation:
  - CumulusCI now supports using the REST Collections API in data load, extract, and delete operations. By default, CumulusCI will select an API for you based on data volume (<2000 records uses the REST API, >=2000 uses Bulk); a desired API can be configured via the mapping file.
  - Removed the namespace_tokenize option from tasks that deploy metadata, and removed the namespace_inject option from tasks that retrieve metadata, because it's unclear when they would be useful.
  - The task create_permission_set allows for creating and assigning a Permission Set that enables specific User Permissions. (Note: other types of permissions are not yet supported).
  - The task create_bulk_data_permission_set creates a Permission Set with the Hard Delete and Set Audit Fields permissions for use with data load operations. The org permission to allow Set Audit Fields must be turned on.
  - CumulusCI's load_dataset and extract_dataset tasks now support relative dates. To take advantage of relative dates, include the anchor_date key (with a date in YYYY-MM-DD format) in each mapping step you wish to relativize. On extract, dates will be modified to be the same interval from the anchor date as they are from the current date; on load, dates will be modified to be the same interval from today's date as they are from their anchor. Both date and date-time fields are supported.
- Other:
  - The oid_as_pk key is no longer supported in bulk data mappings. (This key was already deprecated). Select object Id mode by including the Id field in mappings.

Issues closed:

- Fixed an issue (#2001) that caused CumulusCI to extract invalid data sets when using after: steps with autoincrement primary keys.
- Fixed an issue where the retrieve_changes task did not actually retrieve folders.
- Fixed a bug in the metadeploy-publish task where labels starting with "Install " were not extracted for localization.
- Fixed a bug that prevented using JWT auth with sandboxes if the sandbox's instance_url did not include an instance name.
- Fixed a bug where `cci project init` generated an invalid mapping for bulk data tasks.

15.1.119 3.18.0 (2020-09-03)

Changes:

- Tasks and automation:
  - CumulusCI now tries 10 times (instead of 5) to install managed package versions, which helps ameliorate timeouts when new versions are released.
  - We added support for CSV files to the `push_list` task.
  - We added a `ref` option to `github_copy_subtree` to allow publishing a git reference (commit hash, branch, or tag).
  - Changed the `disable_tdtm_trigger_handlers` (SetTDTMHandlerStatus) task so that trigger handler state is remembered in the cache directory instead of `REPO_ROOT`.
- User experience:
  - The `cci error info` command now defaults to showing the entire traceback when it is more than 30 lines.
- Robot Framework:
  - The following robot keywords have been updated to work with Winter '21:
    - `Load related list`
    - `Click related list button`
    - `Click related item link`
    - `Click related item popup link`
    - `Go to object home`
    - `Go to object list`
    - `Go to record home`
    - `Populate lookup field`
  - The keyword `Load related list` has been rewritten to be slightly more efficient. It also has a new parameter `tries` which can be used if the target is more than 1000 pixels below the bottom of the window.

Issues Closed:

- Fixed a bug where `cci error gist` could throw a UnicodeDecodeError on Windows (fixes ##1977)
- Fixed a bug where `cci org list` could throw a TypeError when run outside a project directory (fixes ##1998)
- The `metadeploy_publish` task can now update translations for language codes with more than 2 letters.
- Fixed a bug where the `extract_dataset` task could fail with a UnicodeEnodeError on Windows.
- `update_dependencies` deduplicates its package install list, making it possible to handle situations where the same beta package is reached by two dependency paths.

15.1.120 3.17.0 (2020-08-20)

Changes:

- Tasks and automation:
  - We added the `upload_user_profile_photo` and `upload_default_user_profile_photo` tasks, which allow for setting Users' profile photos from images stored in the repository. (Thanks to @spelak-salesforce).
  - We added the property `is_person_accounts_enabled` to the `org_config` object, which is available in `when` clauses. (Thanks to @spelak-salesforce).
- Policies and inclusive language:
  - We added information about Salesforce's Open Source Community Code of Conduct and Security policies.
  - We updated documentation to more consistently refer to the "main" branch, reflecting CumulusCI's support for per-project specification of main branches other than `master`.
- User experience:
  - We modified how we handle situations where the default org is not valid for better user experience.
  - We catch a common mistake in entering command-line options (`-org` instead of `--org`, as well as incorrectly-formatted flow options) and show a clearer error.
  - We now capture and display the `InstanceName` of orgs in `cci org list`'s output.
- Robot Framework:
  - We now cleanly fall back to the latest available API version for Robot locators if the newest API version does not have an available locator file. This change helps support Robot testing on the latest prerelease editions of Salesforce.
  - We included some updates to locators for API version 50.0.
- Other:
  - We added a new environment variable, `SFDX_SIGNUP_INSTANCE`, and an `instance` key in org definitions, to specify a preferred instance routing. NOTE: this functionality requires Dev Hub permissions that are not Generally Available.

Issues closed:

- Fixed a bug which prevented package install links from getting added to release notes.
- Fixed a bug (#1914) which caused errors when customizing some Flow steps with decimal step numbers.
- Fixed a bug making it difficult to troubleshoot issues with Snowfakery and CumulusCI on Windows.
- Fixed a bug in `update_admin_profile` that resulted in errors when attempting to manage Record Types across multiple objects.

15.1.121 3.16.0 (2020-08-06)

Changes:

- Project initialization:
  - When starting a new CumulusCI project, the `cci project init` command now uses the current git branch as the project's default branch.
  - API version 49.0 is now set as the default for new projects.
- Bulk data tasks:
  - Added a task called `delete_data` for deleting all data from specified objects. This was previously available but required manually adding it to `cumulusci.yml`
  - The `load_dataset`, `extract_dataset`, and `delete_data` tasks now support automatic namespace injection. When object and field names are specified without namespaces, but the target org only has them with a namespace prefix attached, CumulusCI automatically adds the namespace prefix. This makes it easier for projects to use a single mapping file for unmanaged orgs, namespaced scratch org, and managed orgs.

  This behavior is on by default, but may be disabled by setting the `inject_namespaces` option to False. This feature is believed to be backwards-compatible; however, projects that subclass built-in data loading classes, or which use data loading tasks in very unusual ways, might be impacted.
  - The `load_dataset` and `extract_dataset` tasks have a new option, `drop_missing_schema`. When enabled, this option causes CumulusCI to silently ignore elements in a dataset or mapping that are not present in the target org. This option is useful when building datasets that support additional, optional managed packages or features, which may or may not be installed.
  - The `extract_dataset` and `load_dataset` tasks now support Person Accounts. These will be handled automatically as long as both Account and Contact are in the mapping file. Additional fields should be added to the Account mapping rather than Contact. Thanks @spelak-salesforce
  - The `generate_dataset_mapping` task generates mappings in line with the latest revisions of load/extract functionality: fields are specified as a list, the `table` key is omitted, and namespaces are stripped.
  - The `generate_dataset_mapping` has improved logic for resolving reference cycles between objects. If one of the lookup fields is nillable, the object with that field will be listed first in the generated mapping file.
  - The `generate_and_load_from_yaml` task has a new option, `working_directory`, which can be used to keep temporary files for debugging. The `debug_dir` option has been removed.
- Robot Framework:
  - The `robot` task has a new option, `processes`. If the value is > 1, tests will be run in parallel in the given number of processes, using pabot. Note: It's still up to the test author to make sure the tests won't conflict with each other when running in parallel. This feature is considered experimental.
  - Added an `ObjectManager` page object for interacting with the Object Manager in Setup. Thanks to @rjan-janam
  - RequestsLibrary is now included as a way to test REST APIs from Robot Framework.
- Metadata ETL:
  - Added a new task, `set_field_help_text`, which can be used to update Help Text values on existing fields.
  - Added a new task, `update_metadata_first_child_text`, which can be used to update a single value in existing metadata. Thanks @spelak-salesforce

  - Added a new task, `assign_compact_layout`, which can update a compact layout assignment in existing object metadata. Thanks @spelak-salesforce

- Added a new task, `github_copy_subtree`, to allow publishing selected files or folders to another repository after a release. This allows publishing a subset of your project's code from a private repository to a public one, for example.
- The `create_community` task has a new option, `skip_existing`. When True, the task will not error if a community with the specified name already exists.
- The `release_beta` and `release_production` flows now generate a section in the release notes on GitHub including package install links.
- Task options can now use `$project_config` substitutions in any position, not just at the start of the value.

Issues closed:

- Fixed a bug where changes to global orgs would be saved as project-specific orgs.
- Fixed a bug where `cumulusci.yml` could fail to parse if certain options were specified in `cci project init` (#1780)
- The `install_managed` task now recognizes an additional error message that indicates a package version has not yet finished propagating, and performs retries appropriately.
- Fixed a bug in the logic to prevent installing beta packages in non-scratch orgs.
- Fixed a bug where the `list_changes`, `retrieve_changes`, and `snapshot_changes` tasks could error while trying to reset sfdx source tracking.
- Fixed a bug where the `push_failure_report` task could be missing some failed orgs if there were more than 200 errors.
- Fixed a bug where the `github_release_notes` task could list a change note under a wrong subheading from a different section.
- Fixed freezing of command tasks for MetaDeploy.

Internal changes (these should not affect you unless you're interacting with CumulusCI at the Python level):

- Standardized naming of different levels of configuration:
  - `BaseGlobalConfig` is now `UniversalConfig`.
  - `BaseGlobalConfig.config_global_local_path` is now `UniversalConfig.config_global_path`
  - `BaseGlobalConfig.config_global_path` is now `UniversalConfig.config_universal_path`
  - `BaseProjectConfig.global_config_obj` is now `universal_config_obj`
  - `BaseProjectConfig.config_global` is now `config_universal`
  - `BaseProjectConfig.config_global_local` is now `config_global`
  - `EncryptedFileProjectKeychain.config_local_dir` is now `global_config_dir`
  - `BaseCumulusCI.global_config_class` is now `universal_config_class`
  - `BaseCumulusCI.global_config` is now `universal_config`
- Added `UniversalConfig.cumulusci_config_dir` as a central way to get the path for storing configuration. `UniversalConfig.config_local_dir` was removed.
- OrgConfigs now keep track of which keychain they were loaded from, and have a new `save` method which is the preferred API for persisting updates to the config.

15.1.122 3.15.0 (2020-07-09)

Changes:

- The `run_tests` task now defaults to only logging tests that failed. Set the `verbose` option to True to see all results including tests that passed.
- The `update_dependencies` task now supports an `ignore_dependencies` option, which prevents CumulusCI from processing a specific dependency (whether direct or transitive). This feature may be useful in installers for packages that extend other packages if the installer is not meant to include the base package.
- Improvements to the mapping file for the `extract_dataset` and `load_dataset` tasks:
  - Fields can now be specified as a simple list of Salesforce API names, instead of a mapping. CumulusCI will infer the database column names.
  - Mappings may omit the `table` key and CumulusCI will use the object name.
  - The tasks will check and show an error if mappings do not use a consistent object Id mode.
  - Mappings can now include junction objects with no additional fields.
- The `generate_dataset_mapping` task now has an `include` option to specify additional objects to include in the mapping if they aren't found by the default heuristics.
- Added additional tasks intended for use as preflight checks for MetaDeploy install plans:
  - `check_sobjects_enabled` returns a set of available SObject names.
  - `check_org_wide_defaults` returns a boolean indicating whether Organization-Wide Defaults match the specified values.
- The `update_package_xml` task now supports the MessageChannel metadata type.
- Adjusted the default rules for the `robot_lint` task.
- CumulusCI can be configured to always show Python stack traces in the case of an error by setting the `show_stacktraces` option to True in the `cli` section of `~/.cumulusci/cumulusci.yml`.
- The prompt provided by `cci org shell` now has access to the Tooling API through the keyword `tooling`.
- When using the JWT OAuth2 flow, CumulusCI can be configured to use alternate Salesforce login URLs by setting the SF_PROD_LOGIN_URL and SF_SANDBOX_LOGIN_URL environment variables.

Issues closed:

- Fixed a UnicodeDecodeError that could happen while using the `extract_dataset` task on Windows. (#1838)
- Fixed support for the CustomHelpMenuSection metadata type in the `update_package_xml` task. (#1832)
- Deleting a scratch org now clears its domain from showing in `cci org list`.
- If you try to use `cci org connect` with a login URL containing `lightning.force.com`, CumulusCI will explain that you should use the `.my.salesforce.com` domain instead.
- Fixed an issue with deriving the Lightning domain from the instance URL for some orgs.

15.1.123 3.14.0 (2020-06-18)

Changes:

- Added a generic `dx` task which makes it easy to run Salesforce CLI commands against orgs in CumulusCI's keychain. Use the `command` option to specify the sfdx command.
- Tasks which do namespace injection now support the `%%%NAMESPACE_DOT%%%` injection token, which can be used to support references to packaged Apex classes and Record Types. The token is replaced with `ns.` rather than `ns__` (for namespace `ns`).
- Updated to Robot Framework 3.2.1. Robot Framework has a new parser with a few backwards incompatible changes. For details see the release notes.
- The `run_tests` task now gracefully handles the `required_org_code_coverage_percent` option as a string or an integer.
- CumulusCI now logs a success message when a flow finishes running.

Issues closed:

- Fixed a regression introduced in CumulusCI 3.13.0 where connections to a scratch org could fail with a Read-Timeout or other connection error if more than 10 minutes elapsed since a prior task that interacted with the org. This is similar to the fix from 3.13.2, but for scratch orgs.
- Show a clearer error message if dependencies are configured in an unrecognized format.

15.1.124 3.13.2 (2020-06-10)

Issues closed:

- Fixed a regression introduced in CumulusCI 3.13.0 where connections to Salesforce could fail with a Read-Timeout or other connection error if more than 10 minutes elapsed since a prior task that interacted with the org.

15.1.125 3.13.1 (2020-06-09)

Issues closed:

- Fixed a bug with "after:" steps in the *load_dataset* task.
- Fixed a bug with record types in the *extract_dataset* task.

15.1.126 3.13.0 (2020-06-04)

Changes:

- A new Metadata ETL task, `add_picklist_entries`, safely adds picklist values to an existing custom field.
- Added the `cci org prune` command to automatically remove all expired scratch orgs from the CumulusCI keychain.
- Improvements to the `cci org shell` command:
  - Better inline help
  - New `query` and `describe` functions
- Scratch org creation will now wait up to 120 minutes for the org to be created to avoid timeouts with more complex org shapes.

- The `generate_data_dictionary` task now has more features for complex projects. By default, the task will walk through all project dependencies and include them in the generated data dictionaries. Other non-dependency projects can be included with the `additional_dependencies` option. The output format has been extensively improved.
- The `run_tests` task supports a new option, `required_org_code_coverage_percent`. If set, the task will fail if aggregate code coverage in the org is less than the configured value. Code coverage verification is available only in unmanaged builds.
- The `install_managed` and `update_dependencies` tasks now accept a `security_type` option to specify whether the package should be installed for all users or for admins only.
- `when` expressions can now use the `has_minimum_package_version` method to check if a package is installed with a sufficient version. For example: `when: org_config.has_minimum_package_version("namespace", "1.0")`
- Robot Framework:
  - Added a new keyword in the modal page objects, `Select dropdown value`. This keyword will be available whenever you use the `Wait for modal` keyword to pull in a modal page object.

Issues closed:

- Limited the variables available in global scope for the `cci shell` command.
- Tasks based on `BaseSalesforceApiTask` which use the Bulk API now default to using the project's API version rather than 40.0.
- Bulk data tasks:
  - The `extract_dataset` task no longer converts to snake_case when picking a name for lookup columns.
  - Improved error message when trying to use the `load_dataset` command with an incorrect record type.
  - Fixed a bug with the `generate_mapping_file` option.

15.1.127 3.12.2 (2020-05-07)

Changes:

- Added a task, `set_duplicate_rule_status`, which allows selective activation and deactivation of Duplicate Rules.
- The `create_community` task now retries multiple times if there's an error.
- The `generate_data_dictionary` task now supports multi-select picklist fields and will indicate the related object for lookup fields.
- The `update_package_xml` task now supports the `NavigationMenu` metadata type.

Issues closed:

- In the Salesforce library for Robot Framework, fixed locators for the actions ribbon and app launcher button in Summer '20.
- Fixed the `load_dataset` task so that steps which don't explicitly specify a `bulk_mode` will inherit the option specified at the task level.
- Fixed error handling if an exception occurs within one of the `cci error` commands.
- Fixed error handling if the Metadata API returns a response that is marked as done but also contains an `errorMessage`.

15.1.128 3.12.1 (2020-04-27)

Fixed a problem building the Homebrew formula for installing CumulusCI 3.12.0.

15.1.129 3.12.0 (2020-04-27)

Changes:

- We've removed the prompt that users see when trying to use a scratch org that has expired, and now automatically recreate the scratch org.
- The `load_dataset` task now automatically avoids creating Bulk API batches larger than the 10 million character limit.
- Robot Framework:
  - When opening an org in the browser, the Salesforce library now attempts to detect if the org was created using the Classic UI and automatically switch to Lightning Experience.
  - The Salesforce library now has preliminary support for Summer '20 preview orgs.
- CumulusCI now directs `simple-salesforce` to return results as normal Python dicts instead of OrderedDicts. This should have minimal impact since normal dicts are ordered in the versions of Python that CumulusCI supports, but we mention it for the sake of completeness.

Issues closed:

- Fixed an issue where non-ASCII output caused an error when trying to write to the CumulusCI log in Windows. (#1619)

15.1.130 3.11.0 (2020-04-17)

Changes:

- CumulusCI now includes Snowfakery, a tool for generating fake data. It can be used to generate and load data into an org via the new `generate_and_load_from_yaml` task.
- Added two new preflight check tasks for use in MetaDeploy: `get_available_licenses` and `get_available_permission_set_licenses`. These tasks make available lists of the License Definition Keys for the org's licenses or PSLs.
- The `get_installed_packages` task now logs its result.
- Robot Framework: Added two new keywords (`Get Fake Data` and `Set Faker Locale`) and a global robot variable (`${faker}`) which can be used to generate fake data using the Faker library.

Issues closed:

- Fixed an error when loading a dependency whose `cumulusci.yml` contains non-breaking spaces.
- Fixed a PermissionError when running multiple concurrent CumulusCI commands in Windows. (#1477)
- Show a more helpful error message if a keychain entry can't be loaded due to a change in the encryption key.
- Fixed the `org_settings` task to use the API version of the org rather than the API version of the package.
- In the Salesforce Robot Framework library, the `Open App Launcher` keyword now tries to detect and recover from an occasional situation where the app launcher fails to load.

15.1.131 3.10.0 (2020-04-02)

Changes:

- Added `custom_settings_value_wait` task to wait for a custom setting to have a particular value.
- The `metadeploy_publish` task now has a `labels_path` option which specifies a folder to store translations. After publishing a plan, labels_en.json will be updated with the untranslated labels from the plan. Before publishing a plan, labels from other languages will be published to MetaDeploy.

Issues closed:

- Fixed an issue where running subprocesses could hang if too much output was buffered.

15.1.132 3.9.1 (2020-03-25)

Issues closed:

- The `batch_apex_wait` task will now detect aborted and failed jobs instead of waiting indefinitely.
- Fixed reporting of errors from Robot Framework when it exits with a return code > 250.
- Fixed an ImportError that could happen when importing the new metadata ETL tasks.
- Fixed bugs in how the `set_organization_wide_defaults` and `update_admin_profile` tasks operated in namespaced scratch orgs.
- Show a more helpful error message if CumulusCI can't find a project's repository or release on GitHub. (#1281)
- Fixed the message shown for skipped steps in `cci flow info`.
- Fixed a regression which accidentally removed support for the `bulk_mode` option in bulk data mappings.

15.1.133 3.9.0 (2020-03-16)

Critical changes:

- The `update_admin_profile` task can now add field-level permissions for all packaged objects. This behavior is the default for projects with `minimum_cumulusci_version` >= 3.9.0 that are not using the `package_xml` option. Other projects can opt into it using the `include_packaged_objects` option.

  The Python class used for this task has been renamed to `ProfileGrantAllAccess` and refactored to use the Metadata ETL framework. This is a breaking change for custom tasks that subclassed `UpdateAdminProfile` or `UpdateProfile`.
- Refactored how CumulusCI uses the Bulk API to load, extract, and delete data sets. These changes should have no functional impact, but projects that subclass CumulusCI's bulk data tasks should carefully review the changes.

Changes:

- New projects created using `cci project init` will now get set up with scratch org settings to:
  - Use the Enhanced Profile Editor
  - Allow logging in as another user
  - *Not* force relogin after Login-As
- If `cumulusci.yml` contains non-breaking spaces in indentation, they will be automatically converted to normal spaces.
- Bulk data tasks:

  - Added improved validation that mapping files are in the expected format.
  - When using the `ignore_row_errors` option, warnings will be suppressed after the 10th row with errors.

Issues closed:

- The `github_release` task now validates the `commit` option to make sure it is in the right format.
- If there is an error from `sfdx` while using the `retrieve_changes` task, it will now be logged.

15.1.134 3.8.0 (2020-02-28)

Changes:

- The `batch_apex_wait` task can now wait for chained batch jobs, i.e. when one job starts another job of the same class.
- The metadata ETL tasks that were added in cumulusci 3.7.0 have been refactored to use a new library, `cumulusci.utils.xml.metadata_tree`, which streamlines building Salesforce Metadata XML in Python. If you got an early start writing custom tasks using the metadata ETL task framework, you may need to adjust them to work with this library instead of lxml.

Issues closed:

- Adjusted the `run_tests` task to avoid an error due to not being able to access the symbol table for managed Apex classes in Spring '20. Due to this limitation, CumulusCI now will not attempt to retry class-level concurrency failures when running Apex unit tests in a managed package. Such failures will be logged but will not cause a build failure.
- Corrected a bug in storing preflight check results for MetaDeploy when multiple tasks have the same path.

15.1.135 3.7.0 (2020-02-20)

Changes:

- Added a framework for building tasks that extract, transform, and load metadata from a Salesforce org. The initial set of tasks include:
  - `add_standard_value_set_entries` to add entries to a StandardValueSet.
  - `add_page_layout_related_lists` to add Related Lists to a Page Layout.
  - `add_permission_set_perms` to add field permissions and Apex class accesses to a Permission Set.
  - `set_organization_wide_defaults` to set the Organization-Wide Defaults for one or more objects and wait for the operation to complete.
- Added a new task `insert_record` to insert a single sObject record via the REST API.
- The `update_admin_profile` task now accepts a `profile_name` option, which defaults to Admin. This allows the task to be used to update other Profiles. (The task class has been renamed to UpdateProfile, but can still be used with the old name.)
- Updated to use Metadata API version 48.0 as the default for new projects.
- Robot Framework: Improved documentation for the API keywords in the Salesforce keyword library.

Issues closed:

- Fixed the `cci error info` command. It was failing to load the log from the previous command.
- Fixed a bug where some error messages were not displayed correctly.

- Adjusted the Salesforce Robot Framework keyword library for better stability in Chrome 80.
- Fixed a bug where using SFDXOrgTask to run an sfdx command on a non-scratch org would break with "Must pass a username and/or OAuth options when creating an AuthInfo instance."
- Fixed a bug where an error while extracting the repository of a cross-project source could leave behind an incomplete copy of the codebase which would then be used by future commands.

15.1.136 3.6.0 (2020-02-06)

Changes:

- `cci task info` now has Command Syntax section and improved formatting of option information.
- CumulusCI now displays a more helpful error message when it detects a network connection issue. (#1460)
- We've added the option `ignore_types` to the *uninstall_packaged_incremental* task to allow all components of the specified metadata type to be ignored without having to explicitly list each one.
- The *`FindReplace`* task now accepts a list of strings for the `file_pattern` option.
- If the *`DeleteData`* task fails to delete some rows, this is now reported as an error.
- Robot Framework: Added a new variable `\${SELENIUM_SPEED` that is used to control the speed at which selenium runs when the `Open Test Browser` keyword is called.

Issues Closed:

- Fixed an issue where existing scratch orgs could sometimes not be used in Windows.
- Fixed a regression where [flow info]{.title-ref} and [task info]{.title-ref} commands could show an error [AttributeError: 'NoneType' object has no attribute 'get_service']{.title-ref} when trying to load tasks or flows from a cross-project source. (#1529)
- Fixed an issue where certain HTTP errors while running the bulk data tasks were not reported.

15.1.137 3.5.4 (2020-01-30)

Changes:

- There is a new top level [cci error]{.title-ref} command for interacting with errors in CumulusCI
- [cci gist]{.title-ref} is now [cci error gist]{.title-ref}
- [cci error info]{.title-ref} displays the last 30 lines of a stacktrace from the previous [cci]{.title-ref} command run (if present).
- Changed the prompt users receive when encountering errors in [cci]{.title-ref}.

Issues Closed:

- Robot Framework: Reverted a change to the [select_record_type]{.title-ref} keyword in the Salesforce library to work in both Winter '20 and Spring '20

15.1.138 3.5.3 (2020-01-23)

- Added new features for running Python code (in a file or string) without bringing up an interactive shell. You can now use [–python]{.title-ref} and [–script]{.title-ref} arguments for the [cci shell]{.title-ref} and [cci org shell]{.title-ref} commands.
- Added support for up to two optional parameters in Apex anonymous via token substitution.
- The [EnsureRecordTypes]{.title-ref} class is now exposed as *ensure_record_types* and correctly supports the Case, Lead, and Solution sObjects (in addition to other standard objects).
- Fixed a bug where the github_parent_pr_notes was attempting to post comments on issues related to child pull request change notes.
- Fixed various Robot keyword issues that have been reported for Spring '20.

15.1.139 3.5.2 (2020-01-21)

Issues closed:

- Fixed an issue where errors running the [cci gist]{.title-ref} command prompt the user to use the [cci gist]{.title-ref} command.
- Removed reference to [os.uname()]{.title-ref} so that [cci gist]{.title-ref} works on Windows.
- Fixed an issue where the *dx_pull* task causes an infinite loop to occur on Windows.

15.1.140 3.5.1 (2020-01-15)

Issues closed:

- Fixed an issue that was preventing newlines in output.
- Don't show the prompt to create a gist if the user aborts the process.
- Avoid errors that can happen when trying to store the CumulusCI encryption key in the system keychain using Python's keyring library, which can fail on some systems such as CI systems:
  - We fixed a regression that caused CumulusCI to try to load the keychain even for commands where it's not used.
  - We fixed a bug that caused CumulusCI to try to load the keychain key even when using an unencrypted keychain such as the EnvironmentProjectKeychain.
- Adjusted some keywords in the Salesforce library for Robot Framework to handle changes in the Spring '20 release.

15.1.141 3.5.0 (2020-01-15)

Changes:

- The `load_dataset` task now accepts a `bulk_mode` option which can be set to `Serial` to load batches serially instead of in parallel.
- CumulusCI now stores the logs from the last five executions under `~/.cumulusci/logs`
- CumulusCI has a new top-level command: `cci gist`. This command creates a secret GitHub gist which includes: The user's current CumulusCI version, current Python version, path to python binary, sysname (e.g. Darwin), machine (e.g. x86_64), and the most recent CumulusCI logfile (`~/.cumulusci/logs/cci.log`).

The command outputs a link to the created gist and opens a browser tab with the new GitHub gist. This can be helpful for sharing information regarding errors and issues encountered when working with cci. This feature uses a users GitHub access token for creation of gists. If your access token does not have the 'gist (Create gists)' scope this command will result in a 404 error. For more info see: https://cumulusci.readthedocs.io/en/latest/features.html#reporting-error-logs

- Changed `UpdateAdminProfile` so that it only deploys the modified Admin profile. While it is necessary to retrieve profiles along their associated metadata objects, we don't need to do that for deployments.
- Added options to the *deploy* task: `check_only`, `test_level`, and `specified_tests`. Run `cci task info deploy` for details. (#1066)

15.1.142 3.4.0 (2020-01-09)

Changes:

- Added `activate_flow` task which can be used to activate Flows and Process Builder processes.
- Added two tasks, `disable_tdtm_trigger_handlers` and `restore_tdtm_trigger_handlers`, which can be used to disable trigger handlers for the table-driven trigger management feature of NPSP and EDA.
- In the `load_dataset` task, added a way to avoid resetting the Salesforce Id mapping tables by setting the `reset_oids` option to False. This can be useful when running the task multiple times with the same org.
- Added support for a few new metadata types from API versions 47 and 48 in the `update_package_xml` task.
- Added a way for Robot Framework libraries to register custom locators for use by the selenium library.

Issues closed:

- Fixed a bug with freezing the `load_data` task for MetaDeploy where it would use an invalid option for `database_url`.
- Fixed a bug in the `github_release_notes` task when processing a pull request with no description. (#1444)
- Fixed inaccurate instructions shown at the end of `cci project init`.

15.1.143 3.3.0 (2019-12-27)

Breaking changes:

- Removed tasks which are no longer in use: `mrbelvedere_publish`, `generate_apex_docs`, and `commit_apex_docs`.

Changes:

- Updated Robot Framework Salesforce library to support the Spring '20 release.
- Added `remove_metadata_xml_elements` task which can be used to remove specified XML elements from metadata files.
- Updated references to the NPSP repository to use its new name instead of Cumulus.

Issues closed:

- Fixed the error message shown when a task config has a bad `class_path`.
- Fixed a warning when running the command task in Python 3.8.
- When the CumulusCI Robot Framework library calls Salesforce APIs, it will now automatically retry when it is safe to do so. It will also avoid reusing old connections that might have been closed.
- Fixed the `-o debug True` option for the `robot` task.

15.1.144 3.2.0 (2019-12-11)

Breaking changes:

- We upgraded the SeleniumLibrary for Robot Framework from version 3.3.1 to version 4.1.0. This includes the removal of some deprecated keywords. See the SeleniumLibrary releases for links to detailed release notes.

Changes:

- The `Persistent Orgs` table shown by `cci org list` has been renamed to `Connected Orgs` since scratch orgs will be shown here if they were connected using `cci org connect` instead of created via the Salesforce CLI. This table now shows the org's expiration date, if known.
- Improvements to the `retrieve_changes` task:
  - The task now retrieves only the components that actually changed, not all components listed in `package.xml` in the target directory.
  - Changes can now be retrieved into folders in DX source format. The target directory defaults to `src` if the project is using `mdapi` format or the default entry in `packageDirectories` in `sfdx-project.json` if the project is using `sfdx` format. (Namespace tokenization is not supported in DX format, since there isn't currently a way to deploy DX format source including namespace tokens.)
- Added a task, `load_custom_settings`, to upload Custom Settings defined in YAML into a target org. See https://cumulusci.readthedocs.io/en/latest/bulk_data.html#custom-settings for more info.

Issues closed:

- Fixed an issue with how the package upload task logs Apex test failures to make sure they show up in MetaCI.
- Fixed `KeyError: createdDate` error when trying to get scratch org info.
- A rare issue where CumulusCI could fail to load the symbol table for a failed Apex test class is now caught and reported.
- CumulusCI now displays the underlying error if it encounters a problem with storing its encryption key in the system keychain.

15.1.145 3.1.2 (2019-11-20)

Breaking changes:

- We changed the default path for the mapping file created by the `generate_dataset_mapping` task to `datasets/mapping.yml` so that it matches the defaults for `extract_dataset` and `load_dataset`
- We changed the `extract_dataset` and `load_dataset` tasks to default to storing data in an SQL file, `datasets/sample.sql`, instead of a binary SQLite database file.

Changes:

- `run_tests` can now detect and optionally retry two classes of concurrency issues with Apex unit tests. `run_tests` should always report an accurate total of test methods run, in parallel or serial mode.
- Added the task `generate_data_dictionary`. This task indexes the fields and objects created in each GitHub release for the project and generates a data dictionary in CSV format.
- Added a `devhub` service. This can be used to switch a project to a non-default sfdx Dev Hub using `cci service connect devhub --project`
- Added a predefined `qa` scratch org. It uses the same scratch org definition file as the `dev` org, but makes it easier to spin up a second org for QA purposes without needing to first create it using `cci org scratch`.

- The `database_url` option for the `extract_dataset` and `load_dataset` tasks is no longer required. Either `database_url` or `sql_path` must be specified. If both are specified, the `sql_path` will be ignored.
- Developers can now directly execute CumulusCI from the Python command line using `python -m cumulusci` or `python cumulusci/__main__.py`

Issues closed:

- A problem with how `run_tests` performed Apex test retries when `retry_always` is set to True has been corrected.

15.1.146 3.1.1 (2019-11-13)

New features:

- After connecting an org with `cci org connect`, the browser now shows the message "Congratulations! Your authentication succeeded." instead of "OK"
- External GitHub sources can now specify `release: latest`, `release: latest_beta`, or `release: previous` instead of a commit, branch, or tag.
- The `execute_anon` task has been revised to detect when a gack occurred during execution.

Issues closed:

- When importing a scratch org from sfdx using `cci org import`, the org's `days` is now set correctly from the org's actual expiration date. (#1101)
- The package API version from `cumulusci.yml` is now validated to make sure it's in the "XX.0" format expected by the API. (#1134)
- Fixed an error deploying new setting objects using the `org_settings` task in Winter '20.
- Fixed a bug in processing preflight check tasks for MetaDeploy.
- Fixed path handling in the `update_admin_profile` task when run in a cross-project flow.

15.1.147 3.1.0 (2019-11-01)

Breaking changes:

- The `metadeploy_publish` task now requires setting `-o publish True` in order to automatically set the Version's is_listed flag to True. (This is backwards incompatible in order to provide a safer default.)

New features:

- Python 3.8 is now officially supported.
- Flows can now include tasks or flows from a different project. See Using Tasks and Flows from a Different Project for details.
- In the `metadeploy_publish` task it is now possible to specify a commit hash with `-o commit [sha]`, instead of a tag. This is useful while MetaDeploy plans are in development.
- Bulk data:
  - Added support for mapping Record Types between orgs (by Developer Name) during bulk data extract and load.
  - Added support for Record Type mapping in the `generate_dataset_mapping` task.
  - Added new documentation for bulk data tasks.

- Robot Framework:
  - The sample `create_contact.robot` test that is created when initializing a new project with `cci project init` now makes use of page objects.
  - The page objects library has two new keywords, `wait for modal` and `wait for page object`, which wait for a new page object to appear.
  - `cumulusci.robotframework.utils` now has a decorator named `capture_screenshot_on_error` which can be used to automatically capture a screenshot when a keyword fails.
  - Prior to this change, `Go to page Detail Contact` required you to use a keyword argument for the object id (eg: `Go to page Detail Contact object_id=${object_id}`). You can now specify the object id as a positional parameter (eg: `Go to page Detail Contact ${object_id}`).
- `OrgConfig` objects now have a `latest_api_version` property which can be used to check what Salesforce API version is available.

Issues closed:

- Updated the scratch org definition files generated by `cci project init` to the new recommended format for org settings. Thanks to @umeditor for the fix.
- The `create_unmanaged_ee_src` task (part of the `unmanaged_ee` flow) has been revised to remove the Protected setting on Custom Objects, to ensure that projects using this setting can be deployed to an Enterprise Edition org.
- The Salesforce REST API client used by many tasks will now automatically retry requests on certain connection and HTTP errors.
- Fixed an issue where posts to the Metadata API could reuse an existing connection and get a connection reset error if Salesforce had closed the connection.
- Disabled use of PyOpenSSL by the Python requests library, since it is no longer needed in the versions of Python we support.

15.1.148 3.0.2 (2019-10-17)

Issues closed:

- Fixed a bug in deploying email templates and dashboards that was introduced in 3.0.1.
- Removed broken `config_qa` flow from the `cci project init` template.

15.1.149 3.0.1 (2019-10-16)

New features:

- Added support for new metadata types when generating `package.xml` from a directory of metadata using the `update_package_xml` task.
- The `ci_feature` flow now supports generating change notes for a parent feature branch's pull request from the notes on child pull requests. The parent pull request description will be overwritten with the new notes after a child branch is merged to the parent if the parent pull request has a special label, `Build Change Notes`.
- When running Apex tests with the `run_tasks` task, if there is a single remaining class being run, its name will be logged.
- Apex test failures that happen while uploading a package are now logged.
- In the `robot_libdoc` task, wildcards can now be used in the `path` option.

- Added an `org_settings` task which can deploy scratch org settings from a scratch org definition file.

Issues closed:

- Added a workaround for an issue where refreshing the access token for a sandbox or scratch org could fail if the user's credentials were new and not fully propagated.

15.1.150 3.0.0 (2019-09-30)

Breaking change:

- CumulusCI 3.0.0 removes support for Python 2 (which will reach end of life at the end of 2019). If you're still running Python 2 you can use an older version of CumulusCI, but we recommend upgrading to Python 3. See our installation instructions for your platform.

15.1.151 2.5.9 (2019-09-26)

New features:

- Added a Domain column to the list of scratch orgs in `cci org list`. (thanks @bethbrains)
- Tasks related to Salesforce Communities (thanks @MatthewBlanski)

  : - New `list_community_templates` task - New `list_communities` task - New `publish_community` task - The `create_community` task can now be used to create a community with no URL prefix, as long as one does not already exist.

- Robot Framework:

  : -

```
    Added keywords for generating a collection of sObjects according to a template:

    :   -   `Generate Test Data`
        -   `Salesforce Collection Insert`
        -   `Salesforce Collection Update`

-

    Changes to Page Objects:

    :   -   More than one page object can be loaded at once.
            Once loaded, the keywords of a page object remain
            visible in the suite. Robot will give priority to
            keywords in the reverse order in which they were
            imported.
        -   There is a new keyword, `Log Current Page Object`,
            which can be useful to see information about the
            most recently loaded page object.
        -   There is a new keyword, `Get Page Object`, which
            will return the robot library for a given page
            object. This can be used in other keywords to access
            keywords from another page object if necessary.
        -   The `Go To Page` keyword will now automatically load
            the page object for the given page.
```

(continues on next page)

(continued from previous page)

```
-   Added a basic debugger for Robot tests. It can be enabled
    using the `-o debug True` option to the robot task.
```

- Added support for deploying new metadata types `ProfilePasswordPolicy` and `ProfileSessionSetting`.

Issues closed:

- Fixed a bug where the `batch_apex_wait` task would sometimes fail to conclude that the batch was complete.
- Fixed a bug in rendering tables in Python 2.

15.1.152 2.5.8 (2019-09-13)

New features:

- `LoadData` now supports the key `action: update` to perform a Bulk API update job
- `LoadData` now supports an `after: <step name>` on a lookup entry to defer updating that lookup until a dependent sObject step is completed.
- `GenerateMapping` now handles self-lookups and reference cycles by generating `after:` markers wherever needed.

Issues closed:

- Patch selenium to convert `executeScript` to `executeAsyncScript`. This is a workaround for the `executeScript` issue in chromedriver 77.
- A small issue in `QueryData` affecting mappings using `oid_as_pk: False` has been fixed.

15.1.153 2.5.7 (2019-09-03)

Breaking changes:

- The `retries`, `retry_interval`, and `retry_interval_add` options have been removed from the `run_tests` task. These were misleading as they did not actually retry failing tests.

New features:

- The `run_tests` task now supports a `retry_failures` parameter. This is a list of regular expressions to match against each unit test failure's message and stack trace; if all failing tests match, the failing tests will be retried serially. Set `retry_always` to True to trigger this behavior when any failure matches.
- There is now a default CumulusCI global connected app that can be used to connect to persistent orgs (assuming you know the credentials) without creating a new connected app. It's still possible to configure a custom connected app using `cci service connect connected_app` if more control over the connected app settings is needed.
- When CumulusCI is being run in a non-interactive context it can now obtain an access token for a persistent org using a JWT instead of a refresh token. This approach is used if the SFDX_CLIENT_ID and SFDX_HUB_KEY environment variables are set. This makes it easier to manage persistent org connections in the context of a hosted service because it's possible to replace the connected app's certificate without needing to obtain new refresh tokens for each org.

Issues closed:

- Fixed a bug where showing the summary of flow steps would break with sub-steps in MetaDeploy.
- Fixed a bug in the caching of preflight task results in MetaDeploy.

15.1.154 2.5.6 (2019-08-15)

New features:

- We've changed how the output of some commands are displayed in tables. For users that prefer simpler style tables we've added a `--plain` option to approximate the previous behavior. To permanently set this option, add this in `~/.cumulusci/cumulusci.yml`:

```
cli:
    plain_output: True
```

- Added additional info to the `cci version` command, including the Python version, an upgrade check, and a warning on Python 2.
- Improved the summary of flow steps that is shown at the start of running a flow.
- The `github_release_notes` task now has an `include_empty` option to include links to pull requests that have no release notes. This is enabled by default when this task is called as part of the `release_beta` flow.
- Robot Framework:
  - Added locators file to support the Winter '20 release of Salesforce.
  - New `robot_lint` task to check for common problems in Robot Framework test suites.
  - The `Open Test Browser` keyword will now log details about the browser.
  - Added a new keyword to the CumulusCI library, `Get Community Info`. It can be used to get information about a Community by name via the Salesforce API.

Issues closed:

- Added workarounds for some intermittent 401 errors when authenticating to the GitHub API as a GitHub App.
- `cci org info` shouldn't show traceback if the org isn't found (#1023)

15.1.155 2.5.5 (2019-07-31)

New features:

- Add the `cci org shell` command, which opens a Python shell pre-populated with a simple_salesforce session on the selected org (as `sf`).
- The `cci flow info` command now shows nested subflows.
- Added the `create_community` task allowing for API-based Community creation.
- Added the `generate_dataset_mapping` task to generate a Bulk Data mapping file for a package.
- CumulusCI can now authenticate for GitHub API calls as either a user or an app. The latter is for use when CumulusCI is used as part of a hosted service.
- The `OrgConfig` object now provides access to the Organization SObject data via the `organization_sobject` attribute.

Issues closed:

- The `install_regression` flow now upgrades to the latest beta from the most recent final release instead of from the previous final release.
- Made sure that an `errorMessage` returned from a metadata API deploy will be reported.
- The `load_dataset` task will now stop with an exception if any records fail during the load operation.

15.1.156 2.5.4 (2019-07-03)

- Updated the default API version for new projects to 46.0
- Fixed a bug in reporting scratch org creation errors encountered while running a flow.
- Fixed the `snapshot_changes` and `list_changes` tasks to avoid breaking when the last revision number of a component is null.

15.1.157 2.5.3 (2019-06-24)

Breaking changes:

- Added two new options to the UpdateDependencies task:
  - `allow_newer`: If the org already has a newer release, use it. Defaults to True.
  - `allow_uninstalls`: Allow uninstalling a beta release or newer final release if needed in order to install the requested version. Defaults to False.

  These defaults are a change from prior behavior since uninstalling packages is not commonly needed when working with scratch orgs, and it is potentially destructive.

New features:

- Added support for defining and evaluating preflight checks for MetaDeploy plans.
- The tasks for bulk data extract and load are now configured by default as `extract_data` and `load_data`.
- Updated the project template created by `cci project init`:
  - Added `.gitignore`, `README.md`, and a template for GitHub pull requests
  - Added an option to store metadata in DX source format
  - Added a sample `mapping.yml` for the bulk data tasks
  - Specify the currently installed CumulusCI version as the project's `minimum_cumulusci_version`
  - Check to make sure the project name only contains supported characters
- The `robot_libdoc` task can now generate documentation for Robot Framework page objects.

Issues fixed:

- Colors in terminal output are now displayed correctly in Windows. (#813)
- `cci` no longer prints tracebacks when a flow or task is not found. Additionally, it will suggest a name if a close enough match can be found. (#960)
- Fixed UnicodeDecodeError when reading output from subprocesses if the console encoding is different from Python's preferred file encoding.
- Fixes related to source tracking:
  - Track the max revision retrieved for each component instead of the overall max revision. This way components can be retrieved in stages into different paths.
  - If `snapshot_changes` doesn't find any changes, wait 5 seconds and try again. There can be a delay after a deployment before source tracking is updated.

15.1.158 2.5.2 (2019-06-10)

Issues fixed:

- When generating package.xml, translate `___NAMESPACE___` tokens in filenames into `%%%NAMESPACE%%%` tokens in package.xml (#1104).
- Avoid extraneous output when `--json` output was requested (#1103).
- Display OS notification when a task or flow completes even if it failed.
- Robot Framework: Added logic to retry the initial page load if it is not loading successfully.
- Internal API change: Errors while processing a response from the Metadata API are now raised as MetadataParseError.

15.1.159 2.5.1 (2019-05-31)

Issues fixed:

- Fixed `cci service connect` when run outside of a directory containing a CumulusCI project.

15.1.160 2.5.0 (2019-05-25)

Breaking changes:

- We reorganized the flows for setting up a package for regression testing for better symmetry with other flows. If you were running the `install_regression` flow before, you now probably want `regression_org`.

  Details: The `install_regression` flow now installs the package *without* configuring it. There is a new `config_regression` flow to configure the package (it defaults to calling `config_managed`) and a `regression_org` flow that includes both `install_regression` and `config_regression`.

New features:

- CumulusCI now has experimental support for deploying projects in DX source format. To enable this, set `source_format: sfdx` in the project section of `cumulusci.yml`. CumulusCI will deploy DX-format projects to scratch orgs using `sfdx force:source:push` and to other orgs using the Metadata API (by converting to metadata source format in a temporary directory).
- Setting a default org in CumulusCI (using `cci org default` or the `--default` flag when creating a scratch org) will now also update the sfdx `defaultusername`. (#868)
- When connecting to GitHub using `cci service connect github`, CumulusCI will now check to make sure the credentials are valid before saving them.
- Robot Framework:
  - Added a framework for creating "page object" classes to contain keywords related to a particular page or component.
  - The `robot` task now takes a `name` option to control the name of the robot suite in output.
  - Updates to the keyword `Open Test Browser`:
    - It allows you to open more than one browser in a single test case. (#1068)
    - It sets the default size for the browser window to 1280x1024.
    - Added a new keyword argument `size` to override the default size.
    - Added a new keyword argument `alias` to let you assign an alias to multiple browser windows.

Issues fixed:

- Robot Framework: Fixed a bug where the `Delete Session Records` keyword would skip deleting some records. (#973)
- If Salesforce returns an error response while refreshing an OAuth token, CumulusCI will now show the response instead of just the HTTP status code.
- Fixed a bug in reporting errors from the Metadata API if the response contains `componentFailures` with no `problem` or `problemType`.

15.1.161 2.4.4 (2019-05-09)

New features:

- Added tasks `list_changes` and `retrieve_changes` which interact with source tracking in scratch orgs to handle retrieving changed metadata as Metadata API format source.
- Added task `EnsureRecordTypes` to generate a Record Type and optional Business Process for a specific sObject and deploy the metadata, if the object does not already have Record Types.
- The `update_admin_profile` task now uses Python string formatting on the `package.xml` file used for retrieve. This allows injection of namespace prefixes using `{managed}` and `{namespaced_org}`.

Issues fixed:

- If CumulusCI gets a connection error while trying to call the Salesforce Metadata API, it will now retry several times before giving up.
- The GitHub release notes parser now recognizes Issues Closed if they are linked in Markdown format.
- Robot Framework: Fixed a locator used by the `Select App Launcher App` keyword to work in Summer '19.
- The `cci project init` command now uses an updated repository URL when extending EDA.

15.1.162 2.4.3 (2019-04-26)

- Allow configuration of the email address assigned to scratch org users, with the order of priority being (1) any `adminEmail` key in the scratch org definition; (2) the `email_address` property on the scratch org configuration in `cumulusci.yml`; (3) the `user.email` configuration property in Git.
- CumulusCI can now handle building static resource bundles (`*.resource`) while deploying using the Metadata API. To use this option, specify the `static_resource_path` option for the deploy task. Any subdirectory in this path will be turned into a resource file and added to the package during deployment. There must be a corresponding `*.resource-meta.xml` file for each static resource bundle.
- Bulk data tasks: Fixed a bug that added extra underscores to field names when processing lookups.
- Robot Framework: The Salesforce library now has the ability to switch between different sets of locators based on the Salesforce version, and thanks to it we've fixed the robot so it can click on modal buttons in the Summer '19 release.
- The `cci project init` command now generates projects with a different preferred structure for Robot Framework tests and resources, with everything inside the `robot` directory. Existing projects with tests in the `tests` directory should continue to work.

15.1.163 2.4.2 (2019-04-22)

- The `purgeOnDelete` flag for the `deploy` task will now automatically be set to false when deploying metadata to production orgs (previously deployment would fail on production orgs if this flag was true).
- The installation documentation now recommends using `pipx` to install CumulusCI on Windows, so that you don't have to set up a virtualenv manually.

15.1.164 2.4.1 (2019-04-09)

Changes:

- Updated the default Salesforce Metadata API version to 45.0
- The scratch org definition files generated by `cci project init` now use `orgPreferenceSettings` instead of the deprecated `orgPreferences`.
- The `metadeploy_publish` task now defaults to describing tasks based on `Deploy` as "metadata" steps instead of "other".

Issues Fixed:

- Fixed a couple problems with generating passwords for new scratch orgs:
  - A project's predefined scratch org configs now default to `set_password: True` (which was already the case for orgs created explicitly using cci org scratch).
  - A scratch org config's `set_password` flag is now retained when recreating an expired org. (Fixes ##670)
- Fixed the logic for finding the most recent GitHub release so that it now only considers tags that start with the project's git `prefix_release`.
- Fixed the `install_prod_no_config` flow. The `deploy_post` task was not injecting namespace tokens correctly.
- Fixed the `connected_app` task to work with version 7 of the sfdx CLI. (Fixes ##1013)
- Robot Framework: Fixed the `Populate Field` keyword to work around intermittent problems clearing existing field values.

15.1.165 2.4.0 (2019-03-18)

Critical changes:

- If you are publishing installation plans to MetaDeploy, there have been some significant changes:
  - Plan options are now read from a new `plans` section of `cumulusci.yml` instead of from task options. This means that a single run of the task can now handle publishing multiple plans, and there is now a generic `metadeploy_publish` task which can be used instead of setting up different tasks for each project.
  - Plan steps are now defined inline in the plan configuration rather than by naming a flow. This makes it easier to configure a plan that is like an existing flow with one or two adjustments.
  - There is now a way to customize MetaDeploy step settings such as `name` and `is_required` on a step-by-step basis, using `ui_options` in the plan config.
  - The task will now find or create a `PlanTemplate` as necessary, matching existing PlanTemplates on the product and plan name. This means the plan config no longer needs to reference a plan template by id, which makes it easier to publish to multiple instances of MetaDeploy.

- The `install_upgrade` flow was renamed to `install_regression` to better reflect the use case it is focused on. There are also a few updates to what it does:
  - It will now install the latest beta release of managed packages instead of the latest final release.
  - It now runs the `config_managed` flow after upgrading the managed package, so that it will work if this flow has references to newly added components.

Changes:

- Added support for deploying Lightning Web Components.
- Fixed the bulk data load task to handle null values in a datetime column.
- The *ci_master* flow now explicitly avoids trying to install beta releases of dependencies (since it's meant for use with non-scratch orgs and we block installing betas there since they can't be upgraded).

15.1.166 2.3.4 (2019-03-06)

- Added a new flow, `install_upgrade`, which can be used for testing package upgrades. It installs and configures the *previous* release of the package, then installs the latest release.
- Fixed an error when using `cci org info --json` (fixes ##1013).

15.1.167 2.3.3 (2019-02-28)

- Fixed a bug where flow options specified on the command line were not passed to tasks correctly.
- `cci service connect` now shows a more helpful error message if you call it with a service name that CumulusCI doesn't know about. Fixes ##752.
- Deleted scratch orgs will no longer show the number of days since they were created in `cci org list`. Thanks to @21aslade for the fix.
- Updates to the MetaDeploy publish task:
  - It is now possible to publish a new plan for an existing version.
  - It is now possible to specify the AllowedList to which a plan is visible.
- Updates to Robot Framework support:
  - Fixed a bug in the `robot` task: it now accepts an option called `test` rather than `tests`, since the latter was ignored by Robot Framework.
  - Fixed some stability problems with the `Populate Field` keyword.
  - The `robot_libdoc` task has been replaced with a new task of the same name that can generate a single HTML document for multiple keyword files by passing a comma-separated list of files to the `path` option.

15.1.168 2.3.2 (2019-02-19)

- Mapping enhancements for bulk `QueryData` and `LoadData` tasks
  - The mapping yaml file no longer requires using `Id: sf_id` as a field mapping. If not provided, `QueryData` and `LoadData` will use local database ids instead of Saleforce OIDs for storing lookup relationships. Previous mappings which specify the `Id: sf_id` mapping will continue to work as before using the Salesforce OID as the mapping value.
  - The mapping yaml file's `lookups:` section now handles defaults to allow simpler lookup mappings. The only key required is now `table`. If the `key_field` is provided it will be used.
- The `sql_path` option on `QueryData` can be used to provide the file path where a SQL script should be written. If this option is used, a sqlite in-memory database is used and discarded. This is useful for storing data sets in a Github repository and allowing diffs of the dataset to be visible when reviewing Pull Requests
  - When using this option, it is best to make sure your mapping yaml file does not provide a field mapping for the `Id` field. This will help avoid merge conflicts if querying data from different orgs such as scratch orgs.
- The [sql_path]{.title-ref} option on `LoadData` can be used to provide the file path where a SQL script file should be read and used to load an in-memory sqlite database for the load operation.

15.1.169 2.3.1 (2019-02-15)

- Fixed a bug that caused the `cci` command to check for a newer version on every run, rather than occasionally. Also we now detect whether CumulusCI was installed using Homebrew and recommend an upgrade command accordingly.
- CumulusCI now automatically generates its own keychain key and stores it in the system keychain (using the Python [keyring]{.title-ref} library). This means that it is no longer necessary to specify a CUMULUSCI_KEY as an environment variable. (However, the environment variable will still be preferred if it is there, and it will be migrated to the system keychain.)
- New task `connected_app` makes it easier to deploy and configure the Connected App needed for CumulusCI's keychain to work with persistent orgs. The connected app is deployed using `sfdx` to an org in the `sfdx` keychain and defaults to the `defaultdevhubusername`.
- The `robot` task gives a more helpful error message if you forget to specify an org.
- Updates to the task for publishing to MetaDeploy:
  - Dependency installation steps are now named using the package name and version.
  - The task options have been revised to match changes in the MetaDeploy API. An optional `plan_template_id` is now accepted. `preflight_message` is now named `preflight_message_additional` and is optional. `post_install_message` is now named `post_install_message_additional` and is optional.

15.1.170 2.3.0 (2019-02-04)

Changes:

- When installing a managed package dependency, pre & post metadata bundles are now fetched from the git commit corresponding to the most recent release of the managed package, instead of master.
- Improvements to the task for publishing a release to MetaDeploy:
  - It can now publish a tag even if it's a different commit than what is currently checked out in the working directory.
  - It now pins managed deployments of metadata bundles to the git commit corresponding to the most recent release of the managed package.

Issues Closed:

- ##962: `cumulusci.utils.findReplace` uses wrong file encoding in Python 3
- ##967: Allow `cci service` commands to be run from outside a project repository

15.1.171 2.3.0b1 (2019-01-28)

Breaking Changes:

- We refactored the code for running flows. The full list of steps to run is now calculated from nested flow configuration when the flow is initialized instead of during runtime. Your existing flows should continue to run as before, but if you're interacting with CumulusCI at the Python API level, you'll need to use the `FlowCoordinator` instead of `BaseFlow`.
- Tasks are now expected to have no side effects when they are instantiated. If tasks need to set up resources, do that in `_init_task` instead of `__init__` or `_init_options` to make sure it doesn't happen until the task is actually being run.

Changes:

- There is now a `dev_org_beta_deps` flow which sets up an org in the same way as `dev_org`, but installs the latest beta versions of managed package dependencies.
- The `github_release` task now records the release dependencies as JSON in the release's tag message.
- Looking up the latest release from GitHub is now done using a single HTTP request rather than listing all releases.
- We added S-Controls to the list of metadata types that the `uninstall_packaged_incremental` task will delete.
- Salesforce Robot Framework library: The `Get Current Record Id` keyword now parses the Id correctly when prefixed with `%2F`, which apparently happens.
- The `push_failure_report` task now avoids an error when querying for info about lots of subscriber orgs.

Issues Closed:

- ##911: Fix UnicodeDecodeError when parsing XML retrieved from the Metadata API.

15.1.172 2.2.6 (2019-01-03)

Changes:

- Added support for more metadata types: Group, SharingSet, SharingCriteriaRule, SharingOwnerRule, and SharingTerritoryRule.
- Release process: We now have tools in place to release cumulusci so that it can be installed using Homebrew or Linuxbrew.

Issues Closed:

- Fixed an issue where tasks using the Salesforce REST API could build a wrong URL with an extra slash after the instance URL.
- Fixed an issue where overriding a flow step to set flow: None did not work.
- Robot Framework: Added an automatic retry to work around an issue with an intermittent ConnectionResetError when connecting to headless Chrome in Python 3.

15.1.173 2.2.5 (2018-12-26)

- The `install_managed` and `install_managed_beta` tasks now take optional `activateRSS` and `password` options. `activateRSS` is set to true by default so that any active Remote Site Settings in the package will remain active when installed.
- When running a task with the `--debug` flag, HTTP requests are now logged.
- Robot Framework:
  - Fix issue where "Get Current Record Id" could accidentally match the object name instead of the record Id.
  - Fix issue where "Load Related List" would fail to scroll down to the list.
  - Fix issue where errors deleting records during test teardown would cause a hidden test failure.

15.1.174 2.2.4 (2018-12-17)

Changes:

- Bulk query task:
  - Fixed an issue with querying data filtered by record type (#904).
  - Fixed an issue where the optimized approach for loading data into PostgreSQL was not used.
  - The task will now prevent you from accidentally overwriting existing data by exiting with an error if the table already exists.
- The `deploy` task now logs the size of the zip payload in bytes.
- Fixed a TypeError in the `commit_apex_docs` task (#901).
- Robot Framework:
  - Add location strategies for locating elements by text and by title.

15.1.175 2.2.3 (2018-12-07)

Changes:

- Improved error messages when scratch org creation failed and when a service is not configured.
- Robot Framework: Limit how long the "Load Related List" keyword will wait.

15.1.176 2.2.2 (2018-11-27)

Changes:

- Improved error handling during scratch org creation:
  - Capture and display stderr output from SFDX (issue ##413).
  - Avoid infinite recursion if username wasn't found in output from SFDX.
- Robot Framework: Increased the timeout for initial loading of the browser.

15.1.177 2.2.1 (2018-11-21)

Oops, an update in CumulusCI 2.2.0 ended up breaking the update_dependencies task! Now fixed.

15.1.178 2.2.0 (2018-11-21)

Changes:

- Tasks can now be placed in groups for the task list! Just specify a `group` when defining the task in YAML.
- By popular request, there is now an `org import` command to import an org from the SFDX keychain to the CumulusCI keychain. It takes two arguments: the SFDX username or alias, and the org name.
- Robot Framework:
  - The `Populate Field` keyword now clears an existing value using keystrokes to make sure that change events are fired.
  - Added a `Get Namespace Prefix` keyword to the CumulusCI library to get the namespace prefix for a package.
  - Fixed a bug that broke opening a browser after using the `Run Task` keyword.
- Documentation updates:
  - The readme now includes a link to the full documentation.
  - The instructions for installing CumulusCI on macOS have been simplified and now recommend using the official Python installer from python.org instead of Homebrew. (Homebrew should still work fine, but is no longer necessary.) We also now suggest creating a virtualenv using venv rather than pyenv since the former is included with Python. It's fine to continue using pyenv if you want.
  - Give more useful links for how to set up SFDX.
  - Updated robot library docs.
- Internal refactoring:
  - Removed dependency on HiYaPyCo for YAML loading, which would not report which file failed to load in the event of a YAML parse error.

  - We now consistently load YAML in the same manner throughout the entire library, which will work with all supported Python versions.
  - Simplified the Python API for setting up a CumulusCI runtime. Begone, YamlGlobalConfig and YamlProjectConfig. Our Python API is not yet documented, but we're working on it. In the meantime, if you were relying on running CCI from within Python, you can now just use BaseGlobalConfig (and its get_project_config member) to bootstrap CCI.
  - BaseProjectConfig has shrugged off some methods that just delegated to the keychain.
  - BaseGlobalConfig has shrugged off some unimplemented methods, and BaseGlobalConfig.get_project_config is now deprecated in favor of using a runtime.
  - Introducing… CumulusCIRuntime! In order to alleviate the complexities of getting CumulusCI tasks/flows running from within a Python application, CumulusCIRuntime encapsulates a lot of the details and wiring between Keychain, GlobalConfig, and ProjectConfig. Usage docs are barely included.
  - CliConfig has been renamed to CliRuntime and now inherits from CumulusCIRuntime. It is still accessible as CliConfig.
  - Upgraded dependencies.
- Contributor improvement: The contributor docs now explain how to install pre-commit hooks to make sure our linters have run before you commit.

Issues Closed:

- ##674: `cci org import <username> <org_name>`
- ##877: CumulusCI should be able to connect to any DX alias and/or understand dx auth files

15.1.179 2.1.2 (2018-10-29)

Oops, we broke a few things! This is a bugfix release to fix a few issues found during the Salesforce.org Open Source Community Sprint last week.

Issues Closed:

- ##858 Dataload bulk query fails to load data into the sqlite db
- ##862 CLI options fail on robot task in 2.1.1
- ##864 Deploying a -meta.xml file with non-ASCII characters breaks in Python 2

15.1.180 2.1.1 (2018-10-23)

Changes:

- Our robotframework library for Salesforce got a number of improvements:
  - New keywords:
    * `Click Header Field Link`: Clicks a link in a record header
    * `Load Related List`: Scrolls to a related list and waits for it to load
    * `Click Related List Button`: Clicks a button in the header of a related list
    * `Click Related Item Link`: Clicks the main link for an item in a related list
    * `Click Related Item Popup Link`: Clicks a link in the popup menu for an item in a related list

  - Updated to `robotframework-seleniumlibrary` 3.2.0 which includes a `Scroll Element Into View` keyword.
  - `Wait Until Loading Is Complete` now waits for the main body of the page to render
  - `Populate Lookup Field` now tries several times in case there's an indexing delay
  - Added a `-o verbose True` option to the robot task which logs each keyword as it runs.
  - We now ignore errors while running the script that waits for XHRs to complete (it can fail if the page reloads before the script finishes).
- Popup notifications upon completion of a flow or task now work on Linux too, if you have the `notify-send` command from libnotify. On Ubuntu, install the `notify-osd` package.

Issues Closed:

- ##827 Bulk data load breaks in Python 2
- ##832 pip install cumulusci gets the wrong version of urllib3

15.1.181 2.1.1b1 (2018-10-17)

- `uninstall_packaged_incremental` task: Added `ignore` option to specify components to skip trying to delete even if they are present in the org but not in the local source.

15.1.182 2.1.0 (2018-10-16)

- Fixed the `cci project init` command, which was failing because it wanted the project to already exist! Fixes ##816. In addition, other commands will now function without an active project or keychain when it possible to do so. (For example, try `cci version` which now works when you're not in a project directory.)
- `update_dependencies` task:

  : - Added support for installing private github repositories as dependencies. Thanks to Anthony Backhouse (@1handclapping) for the patch. Fixes ##793 - Added a `dependencies` option to override the project dependencies.
- `execute_apex` task:

  : - Print more useful error messages when there are Apex exceptions.
- `robot` task:

  : - Our logic for automatically retrying failed selenium commands has been encapsulated into the `cumulusci.robotframework.utils.selenium_retry` decorator which can be applied to a robot library class for increased stability. - There is now an option to pause and enter the Python debugger after a keyword fails. Run with `-o pdb True`. - Revised keywords and locators to support the Winter '19 release of Salesforce and improve stability. - The `Salesforce.robot` file now includes the `OperatingSystem` and `XML` libraries from Robot Framework by default. These libraries are helpful in building integration tests such as modifying and deploying a PageLayout to include a field needed in Suite Setup of an integration test.
- Revised installation instructions for Windows. Thanks Matthew Blanski (@Auchtor).
- Internal change: Use a thread-local variable instead of a global to track the current running task.

15.1.183 2.1.0b1 (2018-10-05)

- It's happening! Hot on the heels of the last release, CumulusCI is making the jump to the modern era by adding **support for Python 3**! (Specifically, Python 3.6 and 3.7.) Don't worry, we'll also continue to support Python 2 for the time being. Because this is a bit more wide-reaching change than normal, we're releasing a beta first. To install the beta you'll need to explicitly request its version: `pip install cumulusci==2.1.0b1`. If you already have CumulusCI, after the update it will continue to run under your Python 2 interpreter. If you want to switch to the Python 3 interpreter (which is not yet required), we recommend deleting your Python virtualenv and starting over with the instructions in the tutorial. If you want to keep your Python 2-based virtualenv around just in case, follow those instructions but name the new virtualenv `cci-py3` instead of `cci`.
- There are also some big changes to the **bulk data** tasks. Did you know CumulusCI has bulk data tasks? They are not configured by default, because we need to finish documenting them. But we'll list the changes in case someone is already relying on them:
  - Fixed connection resets by downloading an entire result file before processing.
  - Improved performance by processing batches in parallel, avoiding the SQLAlchemy ORM, storing inserted Ids in separate tables, and doing lookups using SQL joins rather than a separate query for each row.
  - If you're using a postgres database for local storage, performance gets even better by taking advantage of postgres' `COPY` command to load CSV directly.
  - Added a `hardDelete` option for bulk deletes.
  - Added a `start_step` option for bulk loads which can be used to resume loading after an error.
- The `push_failure_report` task will now by default hide failures that occurred due to the "Package Uninstalled" or "Salesforce Subscription Expired" errors, which are generally benign.
- Fixed the check for newer CumulusCI versions to work around an issue with old `setuptools`.
- Contributor change: We switched CumulusCI's own tests to run using `pytest`.
- Internal change: We switched to the `cryptography` library for handling keychain encryption.

15.1.184 2.0.13 (2018-10-02)

- Happy Spooky October! It's unlucky release 2.0.13, with some scary-cool improvements. Just to show you how ramped up our RelEng team is now, this release had TWENTY THREE pull requests in 12 days! From all four of your friendly SFDO Release Engineering committers. Thanks so much for continuing to use CCI for all your Salesforce automation needs.
- NEW FLOW: ci_beta_dependencies installs the latest beta of project dependencies and run tests. Includes task error when running against non-scratch orgs.
- NEW TASK: ReportPushFailures pulls a list of Package Push Upgrade Request failures after a push attempt, including grouping by important factors.
- Issue a terminal "Bell" sound and attempt to display a macOS notification when a commandline task or flow completes.
- Cleaned up python exception and error handling across the board, so that we can provide you, the user, with only the most relevant information. Try using CCI without setting your CUMULUSCI_KEY and see a simplified error message.
- Fixed the utils for processing namespaces in package zip files to handle non-ASCII characters
- The CONTRIBUTING.rst docs and Makefile have been updated to show how we release updates of CCI.
- Skip beta releases when checking for a newer cumulusci version

- When using the strip_namespace option on deployments, we now log which files had changes made before deploying.
- Going Out: the SFDXDeploy and SFDXJsonPollingTasks have been removed, as they didn't work.
- Going Out: Use the safe_load() method when loading YAML instead of the naive load(). If you relied on executing code in your CCI YAML file parsing, that will no longer work.

15.1.185 2.0.12 (2018-09-20)

- Fixed apexdoc URL
- Fixed *update_admin_profile* to set any existing record type defaults to false before setting new defaults.
- Fixed deployment of -meta.xml files containing non-ASCII characters.
- Updated the robot selector for "Click Modal Button" to work for modals opened by a Quick Action.

15.1.186 2.0.11 (2018-09-14)

- *update_admin_profile* now uses xml parsing instead of string replacement for more targeted editing of Admin.profile to fix issues with deploying record types via dependencies
- Projects can declare a dependency on a minimum version of cumulusci by specifying [minimum_cumulusci_version]{.title-ref} in cumulusci.yml

15.1.187 2.0.10 (2018-09-13)

- *update_admin_profile* task now sets application and tab visibility and supports setting record type visibility and default via the new `record-types` task option
- Restructured exceptions to include two new parent exceptions useful in client implementations:
  - CumulusCIFailure: Used to signify a failure rather than an error, such as test or metadata deployment failures
  - CumulusCIUsageError: Use to signify a usage error such as accessing a task that does not exist
- *execute_anon* task now accepts either [apex]{.title-ref} (string) or [path]{.title-ref} (Apex in a local file) for the Apex to execute. Also, the [managed]{.title-ref} and [namespaced]{.title-ref} options allow injecting namespace prefixes into the Apex to be executed.
- New flow [retrieve_scratch]{.title-ref} can be used to retrieve declarative changes from a scratch org into the src/ directory

15.1.188 2.0.9 (2018-09-10)

- Make robot commands use new lightning URLs
- Remove unused filter_name arg from Go to Record Home robot keyword.
- Fix metadata map for Settings.

15.1.189 2.0.8 (2018-08-21)

- Flows that are executed from within another flow now support task-level control flow.
- We no longer support the undocumented ability for a Flow to provide its own class_path.
- Use the connected app details to set a client name on HTTP requests to Salesforce.

15.1.190 2.0.7 (2018-08-16)

- [cci service show]{.title-ref} has been renamed [cci service info]{.title-ref}!
- Update default API version in the base YAML to v43.0.
- Doc updates in the tutorial, thanks to @justindonnaruma!
- Significant refactor of the cli module, including a bunch of small usability and exception handling changes. See https://github.com/SFDO-Tooling/CumulusCI/pull/708 for details.
- Display the file name for error causing files in more cases.
- Strip packageVersions tags from aura/, components/, and pages/ metadata.
- Update PyYAML dependency.

15.1.191 2.0.6 (2018-08-07)

- In Robot tests that use the standard keyword for interacting with a lookup field, we now wait for all AJAX requests to complete before submitting.
- Add unit tests for large sections of the library.
- We now support Flow, DuplicateRule, and other new Metadata types thanks to @carlosvl.
- Fixed refreshing oauth token when deploying metadata to persistent orgs.

15.1.192 2.0.5 (2018-08-01)

- Fixes ##695: Update InstallPackageZipBuilder to set activateRSS to unblock installs.

15.1.193 2.0.4 (2018-07-30)

- Fixes ##611: Scratch org operations were failing on Windows
- Fixes ##664: Scratch org aliases incorrectly included double quotes in the alias name

15.1.194 2.0.3 (2018-07-27)

- Added support for waiting on Aura HTTP Requests to complete after a browser action is performed in selenium from the Robot Salesforce Library: http://cumulusci.readthedocs.io/en/latest/robotframework.html#waiting-for-lightning-ui
- Github API client will now automatically retry on 502 errors
- Better error messages from parsing errors during package.xml generation which show the file causing the error

15.1.195 2.0.2 (2018-06-06)

- Bugfix: Update InstallPackageZipBuilder to use a recent api version to unblock installs.

15.1.196 2.0.1 (2018-06-06)

- Bugfix: Allow passing a connected app directly to OrgConfig.refresh_oauth_token.

15.1.197 2.0.0 (2018-06-01)

After over 19 months of development as alpha (40 version over 3 months) and beta (98 releases over 16 months) releases and over a year running production builds using CumulusCI, it's time to remove the "beta" label.

This marks the first production release of CumulusCI 2.x!

15.1.198 2.0.0-beta99 (2018-05-31)

- Ensure that github credentials are never shown in the log for github dependencies with unmanaged metadata

15.1.199 2.0.0-beta98 (2018-05-31)

WARNING: This release introduces breaking changes to the syntax for flow definitions and to the default flows. If you customized any of the default flows in your project or have defined custom flows, you will need to modify your cumulusci.yml file to work with this release.

Changes default flows shipped with CumulusCI to a new syntax and structure taking advantage of the ability for flows to call other flows. This allows flows to be modularized in ways that weren't possible when the original set of flows was designed.

- The **tasks:** section in cumulusci.yml for a flow is now renamed to **steps:** A **FlowConfigError** will be raised if an old style flow definition is detected. All existing flow customizations and custom flows need to be changed in the **cumulusci.yml** to avoid raising an exception.
- All default flows have been restructured. Existing customizations of default flows likely need to be changed to adapt to the new structure. In most cases, you will want to move your customizations to some of the new **config_*** or **deploy_*** instead of the main flows.
- **ci_beta_install** has been removed and replaced with **install_beta** and **uninstall_managed install_beta** does not attempt to uninstall an existing version of the package. If you need to uninstall the package first, use the **uninstall_managed** flow before running **install_beta**
- Added new **qa_org** flow to allow different configurations for dev vs QA orgs
- New modularized flows structure allows for easier and more reusable customization:
  - **dependencies** Runs the pre-package deployment dependency tasks **update_dependencies** and **deploy_pre** This flow is called by almost all the main flows.
  - **config_*** flows provide a place to customize the package configuration for different environments. These flows are called by the main flows after the package metadata is deployed or a managed version is installed. Customizations to the config flows automatically apply to the main flows.
    - **config_apextest** Configure org for running apex tests
    - **config_dev** Configure org for dev use

        * **config_managed** Configure org with a managed package version installed
        * **config_packaging** Configure the packaging org
        * **config_qa** Configure org for QA use
    - **deploy_*** flows provide a place to customize how metadata deployments are done. The deploy flows do more than just a simple deployment such as unscheduling scheduled jobs, rebuilding the package.xml, and incrementally deleting any stale metadata in the package from the org.
        * **deploy_unmanaged** Used to do a standard deployment of the unmanaged metadata
        * **deploy_packaging** Used to deploy to packaging. Wraps the **create_managed_src** task around the deploy to inject metadata that can only be deployed to the packaging org
        * **deploy_unmanaged_ee** Used to deploy unmanaged metadata to an Enterprise Edition org using the **create_unmanaged_ee_src** task
- **github** dependencies can now point to a private Github repository. All zip downloads from Github will pass the password (should be a personal access token) from the **github** service configured in the CumulusCI keychain.
- **GithubRelease**, **PushUpgradeRequest**, and **PackageUploadRequest** now track the release data as return values

15.1.200 2.0.0-beta97 (2018-05-31)

- Salesforce Connected App is now a CCI Service! Instead of using [cci org config_connected_app]{.title-ref} you can use the familiar [cci service]{.title-ref} commands.
- Better error handling when running commands without specifying a default org (thanks @topherlandry)
- Fix issue where scratch org password may become outdated
- Improve Robot test runner task to use the already configured CCI environment instead of trying to create a new one.
- Enable Robot testing in Headless Chrome on Heroku.
- Address Python3 print statement issues.
- Add LogLine task class to log statements and variables.
- Add PassOptionAsResult, PassOptionAsReturnValue to pass options around in Flows.
- Further extended the Flow runner subclass API.

15.1.201 2.0.0-beta96 (2018-05-18)

- Fixes for CumulusCI on Windows - CumulusCI 2 now supports Windows environments!
- Support skipping scratch org password creation by specifying [–no-password]{.title-ref} to [cci org scratch]{.title-ref}
- Add additional logging to PackageUpload

15.1.202 2.0.0-beta95 (2018-05-10)

- Add pytz to requirements

15.1.203 2.0.0-beta94 (2018-05-10)

- Support added for nested flows. Specify a flow instead of a task inside another flow in cumulusci.yml
- Add new task github_release_report to report info from GitHub release notes
- Add new flow dev_deploy for minimal deploy (tasks: unschedule_jobs, deploy)
- Enhance BaseFlow to be more easily subclassed/overridden/observed. Preserves task step number and adds several hook methods for subclasses (_pre_task, _post_task, _post_task_exception)
- Refactor github_release_notes task to use github3.py instead of calling the GitHub API directly. Includes these minor changes to functionality:

  : - Cannot create release with this task (use github_create_release instead) - Merge existing release notes even when not publishing
- Fix issue that caused duplicate entries in the dependency tree
- Sort output of os.listdir in all occurrences. Guarantees ordered iteration over files on disk
- Validate CUMULUSCI_KEY value and raise more helpful exceptions if invalid

15.1.204 2.0.0-beta93 (2018-04-20)

- Fix issue in command task for Windows
- Support interactive in command task (thanks Chris Landry!)
- Search more pull requests (100 vs 30) when generating release notes
- Add options to Apex documentation generator task

15.1.205 2.0.0-beta92 (2018-04-04)

- Ignore OWNERS file in package.xml generation
- Pipe stderr in command tasks

15.1.206 2.0.0-beta91 (2018-04-03)

- Fix issue in ZIP functionality for Windows

15.1.207 2.0.0-beta90 (2018-03-26)

- Include missing scratch_def.json template file needed by cci project init

15.1.208 2.0.0-beta89 (2018-03-23)

- Improved cci project init

  : - Prompt for extending a repository with HEDA and NPSP as selectable options - Use jinja2 templates included with cumulusci to create files - Include a default Robot test

- update_package_xml now ignores CODEOWNERS files used by Github
- Fixed an import error for click in cci

15.1.209 2.0.0-beta88 (2018-03-20)

- Fix issue in parsing version from tag name

15.1.210 2.0.0-beta87 (2018-03-15)

- Fix issue in getting latest version

15.1.211 2.0.0-beta86 (2018-03-13)

- Initial Integration with Robot Framework (see here for details: http://cumulusci.readthedocs.io/en/latest/robotframework.html)
- Add support for GlobalValueSetTranslation Metadata Type (thanks Christian Szandor Knapp!)
- Use Tooling API for PackageUploadRequest
- New doc "Why CumulusCI?"
- Add documentation for the skip option on GitHub dependencies

15.1.212 2.0.0-beta85 (2018-02-21)

- Support bigobject index element in .object
- Only run meta.xml file cleaning on classes/* and triggers/* directory
- Add docs on CumulusCI Flow
- Add reference to needing the Push API to run release_beta in tutorial doc

15.1.213 2.0.0-beta84 (2018-02-12)

- Add new Status 'Queued' to PackageUploadRequest check

15.1.214 2.0.0-beta83 (2018-02-08)

- Add a sleep in between successful PackageUploadRequest and querying for MetadataPackageVersion to address issue in Spring '18 packaging orgs.

15.1.215 2.0.0-beta82 (2018-02-02)

- Update salesforce-bulk package to version 2.0.0
- Fix issue in bulk load data task

15.1.216 2.0.0-beta81 (2018-01-18)

- Filter SObjects by record type in bulk data retrieve
- Fix issue in removing XML elements from file

15.1.217 2.0.0-beta80 (2018-01-08)

- The deploy tasks now automatically clean all meta.xml files in the deployed metadata of any namespace references by removing the <packageVersions> element and children. This allows CumulusCI to fully manage the dependencies and avoids the need for new commits to change referenced versions in meta.xml files.

  : - The default functionality can be disabled with the by setting [clean_meta_xml]{.title-ref} to False

- Github dependencies can now point to a specific tag in the repository. The tag is used to determine the version to install for the dependency if the repository has a namespace configured and will be used to determine which unpackaged metadata to deploy.

15.1.218 2.0.0-beta79 (2017-11-30)

- Fixes ##540: Using a custom [prefix_beta]{.title-ref} fails if releases with the same version but different prefix already exist in the repository. Changed to use [tag_name]{.title-ref} instead of [name]{.title-ref} to check if the release already exists in Github.

15.1.219 2.0.0-beta78 (2017-11-22)

Resolving a few issues from beta77:

- A bug in BaseKeychain.create_scratch_org was causing the creation of ScratchOrgConfig's with a days value of None. This caused issues with subsequent calls against the org.
- Fixed output from new logging in namespace injection
- Switch to using org_config.date_created to check if an org has been created
- Fix bug in recreation of an expired scratch org

15.1.220 2.0.0-beta77 (2017-11-22)

- New Salesforce DX tasks: *dx_convert_from*, *dx_convert_to*, *dx_pull*, and *dx_push*
- New flow for creating production releases (use with caution!): *release_production*
- Scratch org configs can now specify [days]{.title-ref} as an option which defaults to 1. The default for a scratch config can be overridden in [cci org scratch]{.title-ref} with the [–days N]{.title-ref} option
- [cci org remove]{.title-ref} will now attempt to first delete a scratch org if one was already created
- [cci org scratch]{.title-ref} will prevent you from overwritting a scratch config that has already created a scratch org (which would create an orphaned scratch org) and direct you to use [cci org remove]{.title-ref} instead.
- [cci org list]{.title-ref} now shows the duration days, elapsed days, and if an org is expired.
- [cci org info]{.title-ref} now shows the expiration date for scratch orgs
- All [cci]{.title-ref} commands that update an org config will now attept to automatically recreate an expired scratch org
- New namespace inject token strings are supported for injecting namespaces into Lightning Component references:
  - **%%%NAMESPACE_OR_C%%%***: Replaced with either 'your_namespace' (unmanaged = False) or 'c' (unmanaged = True)
  - **%%%NAMESPACED_ORG_OR_C%%%***: Replaced with either 'your_namespace' (namespaced_org = True) or 'c' (namespaced_org = False)
- Deleted all tasks and code related to [apextestsdb]{.title-ref} since its functionality is now integrated into MetaCI and no longer used

15.1.221 2.0.0-beta76 (2017-11-14)

- Fix bug in namespace injection
- Add option to print org info as JSON

15.1.222 2.0.0-beta75 (2017-11-07)

- Fix syntax for github dependency with [–extend]{.title-ref} option on [cci project init]{.title-ref}

15.1.223 2.0.0-beta74 (2017-11-07)

- Default to Salesforce API version 41.0

15.1.224 2.0.0-beta73 (2017-11-07)

- Fix bug in creating the [dev_namespaced]{.title-ref} scratch org config from [cci project init]{.title-ref}

15.1.225 2.0.0-beta72 (2017-11-06)

- Fix bug in setting namespace from [cci project init]{.title-ref}

15.1.226 2.0.0-beta71 (2017-11-06)

- Update docs, including tutorial for Windows (thanks Dave Boyce!)
- Add missing "purge on delete" option for BaseUninstallMetadata
- Fix crash when decoding certain strings from the Metadata API response
- Add support for featureParameter* metadata types (thanks Christian Szandor Knapp!)

15.1.227 2.0.0-beta70 (2017-10-30)

- Fix issue in zip file processing that was introduced in v2.0.0b69

15.1.228 2.0.0-beta69 (2017-10-27)

- cumulusci.core has been made compatible with Python 3!
- [cci project init]{.title-ref} has been upgraded
  - Better prompt driven user experience with explanations of each prompt
  - [–extend <repo_url>]{.title-ref} option to set up a recursive dependency on another CumulusCI project's Github repository
  - Creates [sfdx-project.json]{.title-ref} if it doesn't already exist
  - Creates and populates the [orgs/]{.title-ref} directory if it does not already exist. The directory is populated with starter scratch org shape files for the 4 main scratch org configs in CumulusCI: [beta.json]{.title-ref}, [dev.json]{.title-ref}, [feature.json]{.title-ref}, [release.json]{.title-ref}
- Fix issue with namespace injection
- [push_*]{.title-ref} tasks now accept [now]{.title-ref} for the [start_time]{.title-ref} option which will start the push upgrade now (technically 5 seconds from now but that's better than 5 minutes).

15.1.229 2.0.0-beta68 (2017-10-20)

- Configure [namespace_inject]{.title-ref} for [deploy_post_managed]{.title-ref}

15.1.230 2.0.0-beta67 (2017-10-20)

- Fix bug where auto-created scratch orgs weren't getting the [scratch]{.title-ref} attribute set properly on their [ScratchOrgConfig]{.title-ref} instance.

15.1.231 2.0.0-beta66 (2017-10-20)

- Configure [namespace_inject]{.title-ref} for *deploy_post*
- Fix the [–debug]{.title-ref} flag on [cci task run]{.title-ref} and [cci flow run]{.title-ref} to allow debugging of exceptions which are caught by the CLI such as MetadataApiError, MetadataComponentError, etc.

15.1.232 2.0.0-beta65 (2017-10-18)

Breaking Changes

- If you created custom tasks off of [DeployNamespaced]{.title-ref} or [DeployNamespacedBundles]{.title-ref}, you will need to switch to using [Deploy]{.title-ref} and [DeployBundles]{.title-ref}. The recommended configuration for such custom tasks is represented below. In flows that need to inject the actual namespace prefix, override the [unmanaged]{.title-ref} option .. :

```
custom_deploy_task:
    class_path: cumulusci.tasks.salesforce.Deploy
    options:
        path: your/custom/metadata
        namespace_inject: $project_config.project__package__namespace
        unmanaged: False
```

Enhancements

- The [cci]{.title-ref} CLI will now check for new versions and print output at the top of the log if a new version is available
- The [cci]{.title-ref} keychain now automatically creates orgs for all named scratch org configs in the project. The orgs are created with the same name as the config. Out of the box, CumulusCI comes with 4 org configs: [dev]{.title-ref}, [feature]{.title-ref}, [beta]{.title-ref}, and [release]{.title-ref}. You can add additional org configs per project using the [orgs]{.title-ref} -> [scratch]{.title-ref} section of the project's [cumulusci.yml]{.title-ref}. With this change, [cci org list]{.title-ref} will always show at least 4 orgs for any project. If an org already exists in the keychain, it is not touched and no scratch org config is auto-created for that config. The goal is to eliminate the need to call [cci org scratch]{.title-ref} in most cases and make it easier for new users to get up and running with scratch orgs and CumulusCI.
- [cci org remove <org_name>]{.title-ref} is now available to remove orgs from the keychain
- Scratch orgs created by CumulusCI are now aliased using the naming format [ProjectName__org_name]{.title-ref} so you can easily run sfdx commands against scratch orgs created by CumulusCI
- [cci org list]{.title-ref} now shows more information including [scratch]{.title-ref}, [config_name]{.title-ref}, and [username]{.title-ref}. NOTE: config_name will only be populated for newly created scratch configs. You can use [cci org scratch]{.title-ref} to recreate the config in the keychain.
- The new flow *dev_org_namespaced* provides a base flow for deploying unmanaged metadata into a namespaced org such as a namespaced scratch org
- All tasks which previously supported [namespace_inject]{.title-ref} now support a new option, [namespaced_org]{.title-ref}. This option is designed to handle use cases of namespaced orgs such as a namespaced scratch org. In namespaced orgs, all unmanaged metadata gets the namespace prefix even if it is not included in the package. You can now use the [namespaced_org]{.title-ref} option along with the file content token

[%%%NAMESPACED_ORG%%%]{.title-ref} and the file name token [___NAMESPACED_ORG___]{.title-ref} to inject the namespace when deploying to a namespaced org. [namespaced_org]{.title-ref} defaults to False to be backwards compatible with previous functionality.

- New task *push_list* supports easily pushing a list of OrgIds via the Push API from the CLI: [cci task run push_list -o file <file_path> -o version 1.2 –org packaging]{.title-ref}

15.1.233 2.0.0-beta64 (2017-09-29)

- Show proper exit status for failed tests in heroku_ci.sh
- Handle BrowserTestFailure in CLI
- Fix issue that prevented auto-merging master to parent branch

15.1.234 2.0.0-beta63 (2017-09-26)

- Documentation has been updated!
- CumulusCI now supports auto detection of repository information from CI environments. This release includes an implementation for Heroku CI

15.1.235 2.0.0-beta62 (2017-09-19)

- cci now supports both namespaced and non-namespaced scratch org configurations in the same project. The default behavior changes slightly with this release. Before, if the [sfdx-project.json]{.title-ref} had a namespace configured, all scratch orgs created via [cci org scratch]{.title-ref} would get the namespace. With the new functionality, all orgs would by default not have the namespace. You can configure individual org configs in your project's [cumulusci.yml]{.title-ref} file by setting [namespace: True]{.title-ref} under [orgs -> scratch -> <org_name>]{.title-ref}

15.1.236 2.0.0-beta61 (2017-09-12)

- Fix bug that was causing a forced token refresh with [sfdx force:org:open]{.title-ref} at the start of a flow or task run against a freshly created scratch org.
- Add support for Big Objects with [__b]{.title-ref} suffix in *update_package_xml* and *update-package-xml-managed*
- Fix bug that caused release notes sections to not render if only h2 content found

15.1.237 2.0.0-beta60 (2017-09-06)

- Add support for Platform Events with [__e]{.title-ref} suffix in *update_package_xml* and *update-package-xml-managed*

15.1.238 2.0.0-beta59 (2017-09-06)

- [YamlProjectConfig]{.title-ref} can now accept an [additional_yaml]{.title-ref} keyword argument on initialization. This allows a 5th level of layering to the [cumulusci.yml]{.title-ref} config. This change is not wired up to the CLI yet but is available for application built on top of cumulusci to use.
- [cumulusci.core.flow]{.title-ref} and [cumulusci.core.keychain]{.title-ref} now have 100% test coverage

15.1.239 2.0.0-beta58 (2017-08-29)

- Fix import error in []github-release-notes() task introduced in beta57

15.1.240 2.0.0-beta57 (2017-08-28)

- Task options can now dynamically reference attributes from the project_config using the syntax [$project_config.attr_name]{.title-ref}. For example, [$project_config.repo_branch]{.title-ref} will resolve to the current branch when the task options are initialized.
- New task [github_parent_to_children]{.title-ref} uses new functionality in [MergeBranch]{.title-ref} to support merging from a parent feature branch (ex. [feature/parent]{.title-ref}) into all child branches (ex. [feature/parent__child]{.title-ref}).
- *github-master-to-feature* task will now skip child branches if their corresponding parent branch exists
- *ci_feature* flow now runs [github_parent_to_children]{.title-ref} at the end of the flow
- Github task classes were restructured but the [class_path]{.title-ref} used in [cumulusci.yml]{.title-ref} remains the same
- New test coverage for github tasks

15.1.241 2.0.0-beta56 (2017-08-07)

- Add stderr logging to scratch org info command

15.1.242 2.0.0-beta55 (2017-08-07)

- Fix API version issue in Apex test runner

15.1.243 2.0.0-beta54 (2017-08-04)

- Fix issue in parsing test failure details when org has objects that need to be recompiled.

15.1.244 2.0.0-beta53 (2017-08-04)

- Fix "cci org config_connected_app" for Windows
- Update tutorial for Windows usage
- Reverse pull request order for release notes

15.1.245 2.0.0-beta52 (2017-08-02)

- Release notes parsers now specified in cumulusci.yml

15.1.246 2.0.0-beta51 (2017-08-01)

- New task to commit ApexDoc output
- New test runner uses Tooling API to get limits data

15.1.247 2.0.0-beta50 (2017-07-18)

- Fix handling of boolean command line args

15.1.248 2.0.0-beta49 (2017-07-10)

- New task *batch_apex_wait* allows pausing until an Apex batch job completes. More details at https://github.com/SFDO-Tooling/CumulusCI/pull/372
- SalesforceBrowserTest task now accepts [extra]{.title-ref} argument for specifying extra command line arguments separate from the command itself
- Resolved ##369: Scratch org tokens expiring after upgrade to SFDX beta

15.1.249 2.0.0-beta48 (2017-06-28)

- Upgraded to the Salesforce DX Beta (thanks to @Szandor72 for the contribution!)
  - NOTE: CumulusCI will no longer work with the sfdx pilot release after this version!
  - Replaced call to [force:org:describe]{.title-ref} with [force:org:display]{.title-ref}
  - Changed json response parsing to match beta format
- New SFDX wrapper tasks
  - `SFDXBaseTask`: Use for tasks that don't need org access
  - `SFDXOrgTask`: Use for sfdx tasks that need org access. The task will refresh the cci keychain org's token and pass it to sfdx as the target org for the command
  - `SFDXJsonTask`: Use for building tasks that interact with sfdx via json responses
  - `SFDXJsonPollingTask`: Use for building tasks that wrap sfdx json responses including polling for task completion
  - `SFDXDeploy`: An example of using [SFDXJsonPollingTask]{.title-ref} to wrap [force:mdapi:deploy]{.title-ref}

- Fixed infinite loop if setting scratch org password fails

15.1.250 2.0.0-beta47 (2017-06-26)

- Fix typo in tasks.util

15.1.251 2.0.0-beta46 (2017-06-23)

- Fix bug in implementation of the [–no-prompt]{.title-ref} flag when sentry is configured

15.1.252 2.0.0-beta45 (2017-06-23)

- The new [BaseSalesforceApiTask]{.title-ref} class replaces [BaseSalesforceApiTask]{.title-ref}, [BaseSalesforceBulkApiTask]{.title-ref}, and [BaseSalesforceToolingApiTask]{.title-ref} by combining them into a single task class with access to all 3 API's via [self.sf]{.title-ref}, [self.tooling]{.title-ref}, and [self.bulk]{.title-ref} from inside a task instance.
- Added integration with sentry.io
  - Use [cci service connect sentry]{.title-ref} to enable the sentry service
  - All task execution exceptions will be logged as error events in sentry
  - [cci task run]{.title-ref} and [cci flow run]{.title-ref} will now show you the url to the sentry event if one was registered and prompt to open in a browser.
  - [cci task run]{.title-ref} and [cci flow run]{.title-ref} now accept the [–no-prompt]{.title-ref} option flag for running in non-interactive mode with the sentry service configured. Use this if you want to log build errors in sentry but not have builds fail due to a hanging prompt.
- If a scratch org password has expired, it is now regenerated when calling [cci org info]{.title-ref}
- New task *unschedule_apex* was added to unschedule background jobs and added to the start of the [dev_org]{.title-ref} flow
- *update-meta-xml* task now uses the project's dependencies as the namespace/version to update in the meta.xml files
- The bulkdata mapping now properly supports Record Types
- Fixed a bug with BulkDataQuery where local references weren't getting properly set
- New CumulusCI Branch & Release Overview diagram presention is available at http://developer.salesforce.org/CumulusCI/diagram/process_overview.html Use left/right arrow buttons on your keyboard to navigate through the presentation.
- CumulusCI is now being built by Heroku CI using the config in [app.json]{.title-ref}

15.1.253 2.0.0-beta44 (2017-06-09)

- Fix issue in *update_dependencies* when a github dependency depends on another github dependency

15.1.254 2.0.0-beta43 (2017-06-09)

- Fix issue in [mrbelvedere_publish]{.title-ref} where the new zip_url dependencies weren't being skipped

15.1.255 2.0.0-beta42 (2017-06-09)

- Move github dependency resolution logic into project_config.get_static_dependencies() for reuse in tasks other than UpdateDependencies
- Fixed the mrbelvedere_publish task when using github references
- Improved output from parsing github dependencies
- Fix issue in [BulkDataQuery]{.title-ref} character encoding when value contains utf8 special characters

15.1.256 2.0.0-beta41 (2017-06-07)

- The [dependencies]{.title-ref} section in cumulusci.yml now supports the [skip]{.title-ref} option for Github dependencies which can be used to skip specific subfolders under [unpackaged/]{.title-ref} in the target repository
- New task class BulkDataQuery reverses the BulkDataLoad and uses the mapping to build SOQL queries to capture the data in the mapping from the target org. The data is written to a database that can then be used by BulkDataLoad to load into a different org.
- The Delete util task now uses the glob library so it can support paths with wildcards like src/*
- New tasks *meta-xml-api* and *meta_xml_dependencies* handle updating [*-meta.xml]{.title-ref} files with api versions or underlying package versions.

15.1.257 2.0.0-beta40 (2017-06-03)

- More enhancements to *update_dependencies* including the ability to handle namespace injection, namespace stripping, and unmanaged versions of managed repositories. See the new doc at http://cumulusci.readthedocs.io/en/latest/dependencies.html

15.1.258 2.0.0-beta39 (2017-06-02)

- Fix new bug in *update_dependencies* which caused failure when running against an org that already has a required package installed

15.1.259 2.0.0-beta38 (2017-06-01)

- *update_dependencies* now properly handles references to a github repository that itself contains dependencies in its cumulusci.yml file
- *update_dependencies* now handles deploying unmanaged metadata from subfolders under unpackaged/pre of a referenced Github repository
- The [dependencies]{.title-ref} section of [cumulusci.yml]{.title-ref} now supports installing from a zip of metadata hosted at a url if you provide a [zip_url]{.title-ref} and optionally a [subfolder]{.title-ref}

15.1.260 2.0.0-beta37 (2017-06-01)

- *update_dependencies* now supports dynamically referencing other Github repositories configured with a cumulusci.yml file. The referenced repository's cumulusci.yml is parsed and the dependencies are included. Also, the Github API is used to find the latest release of the referenced repo if the cumulusci.yml has a namespace configured. Welcome to dynamic package dependency management ;)
- [cci task run]{.title-ref} now supports the option flags [–debug-before]{.title-ref} and [–debug-after]{.title-ref}
- Fix for JUnit output rendering in run_tests

15.1.261 2.0.0-beta36 (2017-05-19)

- Flows can now accept arguments in the CLI to override task options
  - [cci flow run install_beta -o install_managed_beta__version "1.0 (Beta 123)"]{.title-ref}
- Flows can now accept arguments to in the CLI to skip tasks
  - [cci flow run ci_feature –skip run_tests_debug –skip deploy_post]{.title-ref}
- Anonymous apex failures will now throw an exception and fail the build in *execute_anon*
- Fixes ##322: local variable 'message' referenced before assignment

15.1.262 2.0.0-beta35 (2017-05-19)

- New task *execute_anon* is available to run anonymous apex and takes the extra task option [apex]{.title-ref}

15.1.263 2.0.0-beta34 (2017-05-16)

- Fixes ##317: ERROR: Invalid version specified

15.1.264 2.0.0-beta33 (2017-05-11)

- cci org connect and cci org scratch now accept the –default option flag to set the newly connected org as the default org for the repo
- cci org scratch now accepts a new option, –devhub <username>, which allows you to specify an alternate devhub username to use when creating the scratch org
- The SalesforceBrowserTest class now throws a BrowserTestFailure if the command returns an exit status of 1
- Scratch org creation no longer throws an exception if it fails to set a random password on the newly created org

- Push API task enhancements:
  - Push org lists (text files with one org ID per line) can now have comments and blank lines. The first word on the line is assumed to be the org ID and anything after that is ignored.
  - Fixes ##294
  - Fixes ##306
  - Fixes ##208

15.1.265 2.0.0-beta32 (2017-05-04)

- Scratch orgs now get an auto-generated password which is available via [cci org info]{.title-ref}
- Added metadata mapping for StandardValueSets to fix ##310
- Throw nicer exceptions when scratch org interaction fails

15.1.266 2.0.0-beta31 (2017-04-12)

- Use UTC for all Salesforce API date/time fields
- Fix issue with listing metadata types
- Add generic polling method to BaseTask

15.1.267 2.0.0-beta30 (2017-04-04)

- New task list_metadata_types
- [push upgrades] Fix push request status Cancelled –> Canceled
- [push upgrades] Fix datetime namespace issues
- [pyinstaller] Import project-level modules with run-time hook

15.1.268 2.0.0-beta29 (2017-04-04)

- Report push status if start time is less than 1 minute in the future

15.1.269 2.0.0-beta28 (2017-03-30)

- Fix bug in Push API batch retry logic introduced in beta25

15.1.270 2.0.0-beta27 (2017-03-29)

- Skip org in push if statusCode is UKNOWN_EXCEPTION

15.1.271 2.0.0-beta26 (2017-03-29)

- Fixes ##278: Push upgrade raises exception for DUPLICATE_VALUE statusCode

15.1.272 2.0.0-beta25 (2017-03-28)

- Fixes ##277: Push API tasks now correctly handle errors in individual orgs in a batch when scheduling a push job

15.1.273 2.0.0-beta24 (2017-03-27)

- Fixes ##231: Handle unicode in package.xml generation
- Fixes ##239: Replace fix for windows path issues from beta23 with a better implementation
- Fixes ##275: Properly pass purge_on_delete option value in uninstall_packaged_incremental

15.1.274 2.0.0-beta23 (2017-03-22)

- Fixes ##239: Add local path to import path when looking up classes. This should fix an error that appeared only in Windows

15.1.275 2.0.0-beta22 (2017-03-20)

- []github-release-notes() now supports the [link_pr]{.title-ref} option to add links to the pull request where each line of content came from
- Fixes ##266: *update_dependencies* now supports the [purge_on_delete]{.title-ref} option to allow running against production orgs
- Fixes ##267: package.xml generation now skips RecordType when rendering in delete mode

15.1.276 2.0.0-beta21 (2017-03-17)

- Fix parsing of OrgId from the access token using the new sfdx CLI

15.1.277 2.0.0-beta20 (2017-03-17)

- Switch to using the [sfdx]{.title-ref} CLI for interacting with scratch orgs. If you use [cci]{.title-ref} with scratch orgs, this release will no longer work with the [heroku force:*]{.title-ref} commands from the prior Salesforce DX release.
- Upgrades to release notes generator
  - Content is now grouped by subheading under each heading
  - Better error message is thrown if a lightweight tag is found when an annotated tag is needed

15.1.278 2.0.0-beta19 (2017-03-15)

- Fixes ##261: cci org info should refresh token first

15.1.279 2.0.0-beta18 (2017-03-14)

- Skip deleting Scontrols in incremental delete
- Escape package name when generating package.xml

15.1.280 2.0.0-beta17 (2017-03-14)

- OrgConfig and subclasses now support self.username to get the username
- Flows no longer have access to task instance attributes for subsequent task options. Instead, custom task classes should set their task return_values member.
- Improve printing of org info when running tasks from a flow by only printing once at the start of flow. All tasks have an optional self.flow attribute now that contains the flow instance if the task is being run from a flow.
- BaseTask now includes methods for handling retry logic. Implemented in the InstallPackageVersion and RunApexTests
- New task *retrieve_unpackaged* can be used to retrieve metadata from a package.xml manifest
- Fixes ##240 - CumulusCI should now properly handle escaping special characters in xml where appropriate
- Fixes ##245 - Show config values in task info
- Fixes ##251 - ApiRetrieveUnpackaged _clean_package_xml() can't handle metadata with spaces in names
- Fixes ##255 - ApiListMetadata does not list certain metadata types with default folder value

15.1.281 2.0.0-beta16 (2017-02-17)

- Allow batch size to be configured for push jobs with the [batch_size]{.title-ref} job

15.1.282 2.0.0-beta15 (2017-02-15)

- Bug fix release for bug in *update_admin_profile* from the beta 14 release changes to the ApiRetrieveUnpackaged class

15.1.283 2.0.0-beta14 (2017-02-15)

- The new [RetrieveReportsAndDashboards]{.title-ref} task class that can retrieve all reports and dashboards from a specified list of folders
- Documentation improvements contributed by @tet3
- Include userinfo in the OrgConfig, and print username and org id at the beginning of every task run. Contribution by @cdcarter
- [project_local_dir]{.title-ref} (e.g., [~/.cumulusci/NPSP-Extension-Template/]{.title-ref}, home of the encrypted keychain and local override config) now rely on the project name configured in cumulusci.yml instead of the existence of a git remote named origin. Contribution by @cdcarter

15.1.284 2.0.0-beta13 (2017-02-09)

- New services registration support added by community contribution from @cdcarter
  - Services and their schemas can now be defined in the cumulusci.yml file. See https://github.com/SFDO-Tooling/CumulusCI/issues/224 for more details until docs are fully updated
  - [cci services list]{.title-ref}
  - [cci services show github]{.title-ref}
  - [cci services connect github]{.title-ref}
- Improved error handling for metadata deployment failures:
  - Metadata deployments now throw more specific errors when appropriate: MetadataComponentFailure, ApexTestFailure, or MetadataApiError
  - Output for each component failure on a deploy now includes more information such as the column number of the error
- *release_beta* now ignores errors in the []github-release-notes() process by default

15.1.285 2.0.0-beta12 (2017-02-02)

- Throw better exceptions if there are failures creating or deleting scratch orgs

15.1.286 2.0.0-beta11 (2017-02-01)

- Fixes and new functionality for *update-package-xml-managed* task.
  - Added support for project -> package -> name_managed in the cumulusci.yml file to specify a different package name to use when deploying to the packaging org.
  - Fixed bug with install_class and uninstall_class handling

15.1.287 2.0.0-beta10 (2017-01-20)

- Completed removed CumulusCI 1 code from the repository and egg. The egg should be 17MB smaller now.
- Removed [cumulusci.tasks.ant.AntTask]{.title-ref}. Please replace any usage with [cumulusci.tasks.command.Command]{.title-ref} or [cumulusci.tasks.command.SalesforceCommand]{.title-ref}
- Removed the [update_meta_xml]{.title-ref} task for now since it was the only task relying on Ant. A new and much better Python based implementation will be coming soon.

15.1.288 2.0.0-beta9 (2017-01-20)

- A few upgrades to the Command task:
  - No longer strip left side whitespace from output to preserve indentation
  - New method [_process_output]{.title-ref} can be overridden to change how output lines are processed
  - New method [_handle_returncode]{.title-ref} can be overridden to change how exit status is handled

15.1.289 2.0.0-beta8 (2017-01-19)

- Added new task classes util.DownloadZip, command.SalesforceCommand, and command.SalesforceBrowserTestCommand that can be mapped in individual projects to configure browser tests or other commands run against a Salesforce org. The commands are automatically passed a refreshed [SF_ACCESS_TOKEN]{.title-ref} and [SF_INSTANCE_URL]{.title-ref} environment variables.
- Added new CLI commands [cci project connect_saucelabs]{.title-ref} and [cci project show_saucelabs]{.title-ref}
- Added *ci-install-beta* flow that uninstalls the previous managed version then installs the latest beta without running apex tests
- Added new method cumulusci.utils.download_extract_zip to download and extract a zip including re-rooting the zip to a subfolder.
- All Salesforce tasks now delete any tempdirs they create to prevent wasting disk space

15.1.290 2.0.0-beta7 (2017-01-17)

- [run_tests_debug]{.title-ref} now ignores all non-test methods including any method decorated with @testSetup

15.1.291 2.0.0-beta6 (2017-01-17)

- Return full info when a component failure occurs on a Metadata API deployment. Previously only the problem was shown without context like file name and line number making it difficult to figure out what caused the failure.
- [run_tests_debug]{.title-ref} now ignores the @testSetup method when parsing debug logs. Previously it would throw an error if tests used @testSetup

15.1.292 2.0.0-beta5 (2017-01-16)

- Fixes for the *unmanaged_ee* flow to fix a bug where avialableFields elements were not properly being stripped from fieldsSets in .object files
- Fixes for *github-master-to-feature* where merge conflicts would throw exception rather than creating a pull request as expected

15.1.293 2.0.0-beta4 (2017-01-13)

- Add *update_admin_profile* to all flows that deploy or install to a Salesforce org. Note that this adjusted the task numbers in some flows so you should double check your project specific flow customizations.

15.1.294 2.0.0-beta3 (2017-01-13)

- Remove `deploy-post-managed` task from the default *ci_master* flow. Deploying the unpackaged/post content to the packaging org risks the spider accidentally including some of it in the package. Projects that want to run `deploy-post-managed` against the packaging org can extend *ci_master* in their cumulusci.yml file to add it.

15.1.295 2.0.0-beta2 (2017-01-12)

- Fix a bug in project_config.get_latest_version() with tags that don't match either the beta or release prefix.

15.1.296 2.0.0-beta1 (2017-01-12)

- Move into the master branch!
- Changed primary CLI command to [cci]{.title-ref} and left [cumulusci2]{.title-ref} available for legacy support
- Changed all docs to use [cci]{.title-ref} command in examples
- Peg push api tasks to api version 38.0 rather than project api version
- Added 2 new flows: *install_beta* and *install_prod* which install the latest managed version of the package with all dependencies but without running tests
- *release_beta* flow now runs *`github-master-to-feature`* at the end of the flow

15.1.297 2.0.0-alpha42 (2017-01-10)

- Metadata API calls now progressively wait longer between each status check to handle calls with long Pending times. Each check also now outputs a line saying how long it will sleep before the next check.

15.1.298 2.0.0-alpha41 (2017-01-06)

- Fix bug in *uninstall_packaged_incremental* where the task would error out if no metadata was found to delete

15.1.299 2.0.0-alpha40 (2017-01-06)

- *uninstall_packaged_incremental* task now skips the deploy step if now metadata was found to be deleted

15.1.300 2.0.0-alpha39 (2017-01-06)

- Two new task classes exist for loading and deleting data via Bulk API. Note that there are no default task mappings for these classes as the mappings should be project specific. Define your own mappings in your project's cumulusci.yml file to use them.
  - **cumulusci.tasks.bulkdata.LoadData**: Loads relational data from a sqlite database into Salesforce objects using a yaml file for mapping
  - **cumulusci.tasks.bulkdata.DeleteData**: Deletes all records from specified objects in order of object list
- Added support for customPermissions
- Added new Command task that can be used to call arbitrary commands with configurable environment variables

15.1.301 2.0.0-alpha38 (2016-12-28)

- Scratch orgs now cache the org info locally during flow execution to prevent multiple calls out to the Heroku CLI that are unnecessary
- Scratch org calls now properly capture and print both stdout and stderr in the case of an exception in calls to Heroku CLI
- [run_tests_debug]{.title-ref} now deletes existing TraceFlag objects in addition to DebugLevels
- Fix bug in *push_all* and *push_sandbox*
- Push tasks now use timezone for start_date option

15.1.302 2.0.0-alpha37 (2016-12-20)

- []github-release-notes() now correctly handles the situation where a merge commit's date can be different than the PR's merged_at date in Github by comparing commit sha's

15.1.303 2.0.0-alpha36 (2016-12-20)

- *github_release* now works with an existing tag/ref and sleeps for 3 seconds after creating the tag to allow Github time to catch up

15.1.304 2.0.0-alpha35 (2016-12-20)

- Remove [draft]{.title-ref} option from *github_release* since the Github API doesn't support querying draft releases

15.1.305 2.0.0-alpha34 (2016-12-20)

- Fix bug with *github_release* that was causing validation errors from Github

15.1.306 2.0.0-alpha33 (2016-12-20)

- []github-release-notes() now raises an exception in [publish]{.title-ref} mode if the release doesn't exist instead of attempting to create it. Use *github_release* to create the release first before calling []github-release-notes()
- Fix a bug with dynamic task option lookup in flows

15.1.307 2.0.0-alpha32 (2016-12-19)

- Move logger configuration out of core and into CLI so other implementations can provide their own logger configurations
- Added [retry_interval]{.title-ref} and [retry_interval_add]{.title-ref} options to *install_beta* to introduce a progressive delay between retry attempts when the package is unavailable

15.1.308 2.0.0-alpha30 (2016-12-13)

- **IMPORANT** This release changes the yaml structure for flows. The new structure now looks like this:

```
flows:
    flow_name:
        tasks:
            1:
                task: deploy
            2:
                task: run_tests
```

- See the new flow customization examples in the cookbook for examples of why this change was made and how to use it: http://cumulusci.readthedocs.io/en/latest/cookbook.html#custom-flows-via-yaml

15.1.309 2.0.0-alpha30 (2016-12-12)

- Bug fixes submitted by @ccarter:
  - *uninstall_post* was failing to substitute namespaces
  - new util method [findRename]{.title-ref} to rename files with a token in their name
- Bug fix with Unicode handling in run_tests_debug

15.1.310 2.0.0-alpha29 (2016-12-12)

- Require docutils to supprot rst2ansi

15.1.311 2.0.0-alpha28 (2016-12-12)

- Modified tasks and flows to properly re-raise exceptions

15.1.312 2.0.0-alpha27 (2016-12-12)

- [cci]{.title-ref} should now throw the direct exception rather than making it look like the exception came through click
- [cci task doc]{.title-ref} command outputs RST format documentation of all tasks
- New doc with info on all tasks: http://cumulusci.readthedocs.io/en/latest/tasks.html

15.1.313 2.0.0-alpha26 (2016-12-09)

- Bug fix, missing import of re in core/config.py

15.1.314 2.0.0-alpha25 (2016-12-09)

- Fixed run_tests and run_tests_debug tasks to fail throwing an exception on test failure
- run_tests_debug now stores debug logs in a tempdir
- Have the CLI handle ApexTestException events with a nicer error rather than a full traceback which isn't helpful to determining the apex failure
- BaseMetadataApi will now throw MetadataApiError after a Failed status is set
- BaseFlow now throws the original exception rather than a more generic one that obscures the actual failure

15.1.315 2.0.0-alpha24 (2016-12-09)

- Bug fix release, flow_run in the CLI should accept debug argument and was throwing and error

15.1.316 2.0.0-alpha23 (2016-12-09)

- [cci org browser]{.title-ref} now saves the org back to the keychain. This fixes an issue with scratch orgs where a call to org browser on a scratch org that hasn't been created yet gets created but doesn't persist after the command
- [task run]{.title-ref} and [flow run]{.title-ref} now support the [–debug]{.title-ref} flag which will drop you into the Python interactive debugger (pdb) at the point of the exception.
- Added Cookbook to the docs: http://cumulusci.readthedocs.io/en/latest/cookbook.html
- [flow run]{.title-ref} with the [–delete-org]{.title-ref} option flag and scratch orgs no longer fails the flow if the delete org call fails.
- Fixed the *deploy_post* task which has having errors with namespaced file names
- Fixed *update_admin_profile* to properly update the profile. This involved fixing the utils [findReplace]{.title-ref} and [findReplaceRegex]{.title-ref}.
- Reworked exceptions structure and ensure that tasks throw an exception where approriate.

15.1.317 2.0.0-alpha22 (2016-12-02)

- Fix for bug in deploy_post when using the filename token to merge namespace into a filename

15.1.318 2.0.0-alpha21 (2016-12-01)

- Added support for global and project specific orgs, services, and connected app. The global credentials will be used by default if they exist and individual projects an override them.
  - Orgs still default to creating in the project level but the [–global]{.title-ref} flag can be used in the CLI to create an org
  - [config_connected_app]{.title-ref} command now sets the connected app as global by default. Use the '–project' flag to set as a project override
  - [connect_github]{.title-ref}, [connect_mrbelvedere]{.title-ref}, and [connect_apextestsdb]{.title-ref} commands now set the service as global by default. Use the '–project' flag to set as a project override

15.1.319 2.0.0-alpha20 (2016-11-29)

- Remove pdb from BaseFlow.__call__ (oops)

15.1.320 2.0.0-alpha19 (2016-11-29)

- Fix IOError issue with update_admin_profile when using the egg version
- Changed cci task_run and flow_run commands to no longer swallow unknown exceptions so a useful error message with traceback is shown
- Centralized loggers for BaseConfig, BaseTask, and BaseFlow under cumulusci.core.logger and changed logs to always write to a temp file available as self.log_file on any config, task, or flow subclass.

15.1.321 2.0.0-alpha18 (2016-11-17)

- New task [apextestsdb_upload]{.title-ref} uploads json test data to an instance of ApexTestsDB
- Fixed bug in CLI when running tasks that don't require an org
- Include mappings for Community Template metadata types in package.xml generator

15.1.322 2.0.0-alpha17 (2016-11-15)

- Community contributions by @cdcarter
  - *query* task using the Bulk Data API
  - [–login-url]{.title-ref} option on [cci org connect]{.title-ref}
- Salesforce DX wrapper
  - NOTE: Requires developer preview access to Salesforce DX
  - [cci org scratch <config_name> <org_name>]{.title-ref} creates a wrapper for a scratch org in your keychain
  - Tasks and Flows run against a scratch org will create the scratch org if needed

  - [cci org scratch_delete <org_name>]{.title-ref} deletes a scratch org that was created by running a task or flow
  - [cci flow run]{.title-ref} now supports the [–delete-org]{.title-ref} option to delete a scratch org at the end of the flow
  - [BaseSalesforceDXTask]{.title-ref} wraps the heroku force:* commands. The *dx_push* task is provided as an example.
    * NOTE: Currently the command output is buffered and only outputs when the command completes.
- Integration with mrbelvedere
  - `mrbelvedere-publish` task publishes a beta or release tag to an existing package on mrbelvedere
- Flow changes
  - *ci_feature* now runs tests as part of the flow
  - New flow task configuration [ignore_failure]{.title-ref} can be used to ignore a failure from a particular task in the flow
- CUMULUSCI_KEY is no longer required if using a keychain class with the encrypted attribute set to False such as the EnvironmentProjectKeychain
- Refactored OAuth token refresh to be more centralized and raise a proper exception if there is an issue
- The org keychain now correctly uses the instance url when appropriate
- Calls to runTestsAsynchronous in the Tooling API are now done via POST instead of GET

15.1.323 2.0.0-alpha16 (2016-11-3)

- Fix bug in SOAP calls to MDAPI with newer versions of the requests library
- This version was used to record the demo screencast: https://asciinema.org/a/91555

15.1.324 2.0.0-alpha15 (2016-11-3)

- Fix CLI bug in new exception handling logic

15.1.325 2.0.0-alpha14 (2016-11-3)

- Fix version number
- Fix bug in BaseSalesforceBulkApiTask (thanks @cdcarter)

15.1.326 2.0.0-alpha13 (2016-11-3)

- Nicer log output from tasks and flows using [coloredlogs]{.title-ref}
- Added handling for packed git references in the file .git/packed-refs
- Docs now available at http://cumulusci.readthedocs.io
- Tasks and Flows run through the CLI now show a more simple message if an exception is thrown

15.1.327 2.0.0-alpha12 (2016-11-2)

- Automatic detection of latest production and beta release via Github Releases
  - project_config.get_latest_release() added to query Github Releases to find the latest production or beta release version
  - InstallPackage now accepts the virtual versions 'latest' and 'latest_beta' as well as specific versions for the version option
- New flows:
  - ci_feature: Runs a full deployment of the unmanaged code for testing in a feature org
  - ci_master: Runs a full deployment of the managed version of the code into the packaging org
  - ci_beta: Installs the latest beta and runs all tests
  - ci_release: Installs the latest release and runs all tests
  - release_beta: Uploads a beta release of the metadata in the packaging org, creates a Github Release, and generates release notes
- Removed the hard coded slots in the keychain for github, mrbelvedere, and apextestsdb and replaced with a more generic concept of named keychain services. keychain.get_service('name') retrieves a named service. The CLI commands for setting github, mrbelvedere, and apextestsdb were modified to write the service configs to the new structure.
- Flow tasks can now access previous tasks' attributes in their options definitions. The syntax is ^^task_name.attr1.attr2
- Flow output is now nicer showing the flow configuration and the active configuration for each task before execution
- New tasks
  - update_package_xml_managed: Create a new package.xml from the metadata in src/ with attributes only available when deploying to packaging org
  - run_tests: Runs matching apex tests in parallel and generate a JUnit report
  - run_tests_debug: Runs matching apex tests in parallel, generates JUnit report, captures debug logs, and parses debug logs for limits usage outputing results to test_results.json
  - run_tests_managed: Runs matching apex tests in parallel from the package's namespace and generate a JUnit report

15.1.328 2.0.0-alpha11 (2016-10-31)

- project_config.repo_root is now added to the python syspath, thanks @cdcarter for the contribution
- Tasks for the new Package Upload API
  - upload_beta: Uploads a beta release of the metadata currently in the packaging org
  - upload_production: Uploads a production release of the metadata currently in the packaging org
- Dependency management for managed packages:
  - update_dependencies: Task that ensures the target org has all dependencies installed at the correct version
  - Dependencies are configured using the dependencies: heading in cumulusci.yml under the project: section
- Integrated salesforce-bulk and created BaseSalesforceBulkApiTask for building bulk data tasks

- Added [cci version]{.title-ref} command to print out current package version, thanks @cdcarter for the contribution

15.1.329 2.0.0-alpha10 (2016-10-28)

- More pure Python tasks to replace ant targets:
  - create_ee_src
  - retrieve_packaged
  - retrieve_src
  - revert_ee_src
  - uninstall_packaged_incremental
  - update_admin_profile
- New flow:
  - unmanaged_ee: Deploys unmanaged code to an EE org
- New cumulusci.utils
  - CUMULUSCI_PATH: The absolute path to the root of CumulusCI
  - findReplaceRegex: Recursive regex based search/replace for files
  - zip_subfolder: Accepts a zipfile and path, returns a zipfile with path as root
- Fix bug where repo_name was not being properly handled if it origin ended in .git

15.1.330 2.0.0-alpha9 (2016-10-27)

- Switch to using [plaintable]{.title-ref} for printing text tables in the following CLI commands:
  - cci org list
  - cci task list
  - cci task info
  - cci flow list
- Easier project set up: [cci project init]{.title-ref} now prompts for all project values using the global default values
- More pure Python Metadata API tasks:
  - create_package
  - install_package
  - uninstall_managed
  - uninstall_packaged
  - uninstall_pre
  - uninstall_post
  - uninstall_post_managed
- New tasks to interact with the new PackageUploadRequest object in the Tooling API

  - upload_beta
  - upload_production
- Python task to replace deployUnpackagedPost ant target with support for replacing namespace prefix in filenames and file contents
  - deploy_post
  - deploy_post_managed
- Python tasks to replace createManagedSrc and revertManagedSrc ant targets
  - create_managed_src
  - revert_managed_src

15.1.331 2.0.0-alpha8 (2016-10-26)

- New tasks for push upgrading packages
  - push_all: Pushes a package version to all available subscriber orgs
    - ex: cci task run –org packaging -o version 1.1 push_all
  - push_qa: Pushes a package version to all org ids in the file push/orgs_qa.txt in the repo
    - ex: cci task run –org packaging -o version 1.1 push_qa
  - push_sandbox: Pushes a package version to all available sandbox subscriber orgs
    - ex: cci task run –org packaging -o version 1.1 push_sandbox
  - push_trial: Pushes a package version to all org ids in the file push/orgs_trial.txt in the repo
    - ex: cci task run –org packaging -o version 1.1 push_trial
  - Configurable push tasks in cumulusci.tasks.push.tasks:
    - SchedulePushOrgList: uses a file with one OrgID per line as the target list
    - SchedulePushOrgQuery: queries PackageSubscribers to select orgs for the target list
  - Additional push tasks can be built by subclassing cumulusci.tasks.push.tasks.BaseSalesforcePushTask

15.1.332 2.0.0-alpha7 (2016-10-25)

- New commands for connecting to other services
  - cci project connect_apextestsdb: Stores ApexTestDB auth configuration in the keychain for use by tasks that require ApexTestsDB access
  - cci project connect_github: Stores Github auth configuration in the keychain for use by tasks that require Github access
  - cci project connect_mrbelvedere: Stores mrbelvedere auth configuration in the keychain for use by tasks that require access to mrbelvedere
  - cci project show_apextestsdb: Shows the configured ApexTestsDB auth info
  - cci project show_github: Shows the configured Github auth info
  - cci project show_mrbelvedere: Shows the configured mrbelvedere auth info
- Github Tasks

  - The new BaseGithubTask wraps the github3.py API library to allow writing tasks targetting Github
  - The following new Github tasks are implemented on top of BaseGithubTask:
    * github_clone_tag: Clones one git tag to another via the Github API
    * github_master_to_feature: Merges the HEAD commit on master to all open feature branches via the Github API
    * github_release: Creates a Release via the Github API
    * github_release_notes: Generates release notes by parsing merged Github pull request bodies between two tags
- BaseTask now enforces required task_options raising TaskOptionError if required options are missing
- Restructured the project: heading in cumulusci.yml

15.1.333 2.0.0-alpha6 (2016-10-24)

- Moved the build and ci directories back to the root so 2.0 is backwards compatible with 1.0
- Allow override of keychain class via CUMULUSCI_KEYCHAIN_CLASS env var
- New keychain class cumulusci.core.keychain.EnvironmentProjectKeychain for storing org credentials as json in environment variables
- Tasks now support the salesforce_task option for requiring a Salesforce org
- The new BaseSalesforceToolingApi task wraps simple-salesforce for building tasks that interact with the Tooling API
- cumulusci org default <name>
  - Set a default org for tasks and flows
  - No longer require passing org name in task run and flow run
  - –unset option flag unsets current default
  - cumulusci org list shows a * next to the default org
- BaseAntTask split out into AntTask and SalesforceAntTask
- cumulusci.tasks.metadata.package.UpdatePackageXml:
  - Pure python based package.xml generation controlled by metadata_map.yml for mapping in new types
  - Wired into the update_package_xml task instead of the old ant target
- 130 unit tests and counting, and our test suite now exceeds 1 second!

15.1.334 2.0.0-alpha5 (2016-10-21)

- Update README

15.1.335 2.0.0-alpha4 (2016-10-21)

- Fix imports in tasks/ant.py

15.1.336 2.0.0-alpha3 (2016-10-21)

- Added yaml files to the MANIFEST.in for inclusion in the egg
- Fixed keychain import in cumulusci.yml

15.1.337 2.0.0-alpha2 (2016-10-21)

- Added additional python package requirements to setup.py for automatic installation of dependencies

15.1.338 2.0.0-alpha1 (2016-10-21)

- First release on PyPI.

15.2 Contribute to CumulusCI

Contributions are welcome, and they are greatly appreciated!

15.2.1 Types of Contributions

You can contribute in many ways:

Report Bugs

Report bugs at https://github.com/SFDO-Tooling/CumulusCI/issues.

When reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whomever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whomever wants to implement it.

Write Documentation

CumulusCI could always use more documentation, whether as part of the official CumulusCI docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

15.2.2 Install for Development

Ready to contribute? Here's how to set up CumulusCI for local development.

1. Fork the CumulusCI repo on GitHub.
2. Clone your fork to your local workspace.
3. Create a fresh Python 3 virtual environment and activate it (to keep this isolated from other Python software on your machine). Here is one way:

```
$ python3 -m venv cci_venv
$ source cci_venv/bin/activate
```

4. Install the development requirements:

```
$ make dev-install
```

5. Install `pre-commit` hooks for `black` and `flake8`:

```
$ pre-commit install --install-hooks
```

6. After making changes, run the tests and make sure they all pass:

```
$ pytest
```

7. Your new code should also have meaningful tests. One way to double check that your tests cover everything is to ensure that your new code has test code coverage:

```
$ make coverage
```

8. Push your changes to GitHub and submit a Pull Request. The base branch should be a new feature branch that we create to receive the changes (contact us to create the branch). This allows us to test the changes using our build system before merging to main.

Note: We enable typeguard with pytest so if you add type declarations to your code, those declarations will be treated as runtime assertions in your Python tests.

15.2.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

- Documentation is updated to reflect all changes.
- New classes, functions, etc have docstrings.
- New code has comments.
- Code style and file structure is similar to the rest of the project.
- You have run the `black` code formatter.
- If you are a new contributor, don't forget to add yourself to the `AUTHORS.rst` file in your pull request (either GitHub username, or first/last name).
- You have labeled your pull request:
  - `critical-changes` for breaking changes,
  - `enhancement` for new features,
  - `bug` for when fixing a bug or closing an issue, or
  - `ignore-for-release` for internal changes.

15.2.4 Testing CumulusCI

Org-reliant Automated Tests

Some tests are marked `@pytest.mark.vcr()` which means that they can either call into a real (configured) Salesforce org or use a cached YAML file of the request/response.

By default using pytest will use the cached YAML. If you want to work against a real scratch org, you do so like this:

```
$ pytest --org qa <other arguments and options, such as filename or -k testname>
```

Where "orgname" is a configured org name like "qa", "dev", etc.

To regenerate the VCR file, you can run this command:

```
$ pytest --replace-vcrs --org qa
```

This will configure an org named "qa" and regenerate them.

That will run all VCR-backed tests against the org, including all of the slow integration tests.

Running Integration Tests

Some tests generate so much data that we do not want to store the VCR cassettes in our repo. You can mark tests like that with `@pytest.mark.large_vcr()`. When they are executed, their cassettes will go in a .gitignore'd folder called `large_cassettes`.

Do not commit the files (`large_cassettes/\*.yml`) to the repository.

Some tests generate even more network traffic data and it isn't practical to use VCR at all. Still, we'd like to run them when we run all of the other org-reliant tests with –org. Mark them with `@pytest.mark.needs_org()` and they will run with the VCR tests.

Some tests are so slow that you only want to run them on an opt-in basis. Mark these tests with `@pytest.mark.slow()` and run them with `pytest --run-slow-tests` or `pytest --run-slow-tests --orgname <orgname>`.

Writing Integration Tests

All features should have integration tests which work against real orgs or APIs.

Our test suite makes extensive use of pytest fixtures; the ones below should be used in your tests where appropriate. Search the repo to see examples where they are used in context, or to see their definitions:

- gh_api - get a fake github API
- with temp_db():… - create a temporary SQLite Database
- delete_data_from_org("Account,Contacts") - delete named sobjects from an org
- run_code_without_recording(func) - run a function ONLY when the integration tests are being used against real orgs and DO NOT record the network traffic in a VCR cassette
- sf - a handle to a simple-salesforce client tied to the current org
- mock_http_response(status) - make a mock HTTP Response with a particular status
- runtime - Get the CumulusCI runtime for the current working directory
- project_config - Get the project config for the current working directory
- org_config - Get the project config for the current working directory
- create*task - Get a task _factory* which can be used to construct task instances.
- global_describe - Get a function that will generate the JSON that Salesforce would generate if you do a GET on the /sobjects endpoint

Decorators for tests:

- pytest.mark.slow(): a slow test that should only be executed when requested with –run-slow-tests
- pytest.mark.large_vcr(): a network-based test that generates VCR cassettes too large for version control. Use –org to generate them locally.
- pytest.mark.needs_org(): a test that needs an org (or at least access to the network) but should not attempt to store VCR cassettes. Most tests that need network access do so because they need to talk to an org, but you can also use this decorator to give access to the network to talk to github or any other API.
- pytest.mark.org_shape('qa', 'qa_org'): - switch the current org to an org created with org template "qa" after running flow "qa_org". As with all tests, clean up any changes you make, because this org may be reused by other tests.

A complete list is available with:

    $ pytest –markers

Randomized tests

Tests should be executable in any order. You can run this command a few times to verify if they are:

pytest –random-order

It will output something like this:

Using –random-order-bucket=module Using –random-order-seed=986925

Using those two parameters on the command line, you can replicate a particular run later.

In extremely rare cases where it's not possible to make tests independent, you can enforce an order