

Salesforce DX Developer Guide

Version 46.0, Summer '19





© Copyright 2000–2019 salesforce.com, inc. All rights reserved. Salesforce is a registered trademark of salesforce.com, inc., as are other names and marks. Other marks appearing herein may be trademarks of their respective owners.

CONTENTS

Chapter 1: How Salesforce Developer Experience Changes the Way You Work1
Use a Sample Repo to Get Started
Chapter 2: Salesforce CLI Configuration and Tips
CLI Runtime Configuration Values5Environment Variables6Salesforce DX Usernames and Orgs9Override or Add Definition File Options at the Command Line12CLI Parameter Resolution Order13Support for JSON Responses13Log Messages and Log Levels13CLI Deprecation Policy14
Chapter 3: Project Setup
Sample Repository on GitHub16Create a Salesforce DX Project16Create a Salesforce DX Project from Existing Source18Retrieve Source from an Existing Managed Package18Retrieve Unpackaged Source Defined in a package.xml File19Convert the Metadata Source to Source Format20Link a Namespace to a Dev Hub Org21Salesforce DX Project Configuration22
Chapter 4: Authorization
Authorize an Org Using the Web-Based Flow25Authorize an Org Using the JWT-Based Flow25Authorize a Scratch Org27Create a Private Key and Self-Signed Digital Certificate28Create a Connected App28Use an Existing Access Token Instead of Authorizing30Authorization Information for an Org30Log Out of an Org31
Chapter 5: Metadata Coverage 32
Chapter 6: Scratch Orgs34Scratch Org Definition File37Scratch Org Definition Configuration Values40

Create Scratch Orgs	. 49
Select the Salesforce Release for a Scratch Org	. 51
Salesforce DX Project Structure and Source Format	. 52
Push Source to the Scratch Org	. 57
How to Exclude Source When Syncing or Converting	60
Assign a Permission Set	. 61
Ways to Add Data to Your Scratch Org	62
Example: Export and Import Data Between Orgs	63
Pull Source from the Scratch Org to Your Project	. 64
Track Changes Between the Project and Scratch Org	. 65
Scratch Org Users	. 66
Create a Scratch Org User	. 67
User Definition File for Customizing a Scratch Org User	. 68
Generate or Change a Password for a Scratch Org User	. 69
Manage Scratch Orgs from Dev Hub	. 70
Charter 7. Sandbayaa	71
Chapter 7: Sunaboxes	. / I
Authorize in to Your Production Org	. 72
Create a Sandbox Definition File (Beta)	. 72
Create, Clone, or Delete a Sandbox (Beta)	. 74
Chapter 8: Development	. 77
Develop Against Any Org	. 79
Create Lightning Apps and Aura Components	. 82
Create Lightning Web Components	82
Create an Apex Class	83
Create an Apex Trigger	83
Testing	83
View Apex Debug Logs	. 84
Apex Debugger	. 85
Chapter 9: Build and Pologso Your App	94
	00
Build and Release Your App with Metadata API	. 88
Develop and Test Changes Locally	. 90
Build and Test the Release Artitact	. 91
Test the Release Artitact in a Staging Environment	. 91
Release Your App to Production	. 92
Cancel a Metadata Deployment	. 93
Chapter 10: First-Generation Managed Packages	94
Build and Release Your App with Managed Packages	95
Packaging Checklist	. 95
Deploy the Package Metadata to the Packaging Org	96
Create a Beta Version of Your App	. 97
Install the Package in a Target Org	98

Contents

Create a Managed Package Version of Your App	8
View Information About a Package	9
View All Package Versions in the Org	D
Package IDs	D
Chapter 11: Unlocked Packages (Generally Available) and Second-Generation	
Managed Packages (Beta) 10	1
Second-Generation Packaging	2
What's a Package?	2
Types of Packaging Projects	2
Packaging for ISVs	3
Enterprise Customers	3
Before You Create Second-Generation Packages	3
Know Your Orgs	4
Sample Repository	4
Review Org Setup	5
Workflow for Second-Generation Packages	6
Plan Second-Generation Packages	7
Namespaces	7
Package Types	9
Best Practices for Second-Generation Packages	9
Package IDs	D
Configure Packages	1
Project Configuration File for Packages	1
Keywords	5
Package Installation Key	5
Extract Dependency Information from Unlocked Packages	6
Create a Package	7
Generate the Package	B
Generate a Package Version	9
Package Ancestors)
Release a Second-Generation Package	0
Update a Package Version	1
	1
Insidii d Package	2
Install Packages from a LIPI	2
Insidii Packages Itolii a OKL	С Л
Sample Script for Installing Dackages with Dependencies	+ л
Migrate Depresented Metadeta from Unlesked Dackages	+ 7
Inigrale Depreculeu Meladulu IIOITI Officiaeu Fucaujes	′ 7
	¢.
Chapter 12: Continuous Integration 128	B
Continuous Integration Using CircleCl	9

Contents

Configure Your Environment for CircleCl
Connect CircleCl to Your DevHub
Continuous Integration Using Jenkins
Configure Your Environment for Jenkins
Jenkinsfile Walkthrough
Sample Jenkinsfile
Continuous Integration with Travis Cl
Sample CI Repos for Org Development Model
Sample CI Repos for Package Development Model
Chapter 13: Troubleshoot Salesforce DX
CLI Version Information
Pup CLI Commands on macOS Siorra (Varsion 10.12)
Error: No defaultdevhubusername org found
Error: No defaultdevhubusername org found
Error: No defaultdevhubusername org found142Unable to Work After Failed Org Authorization143Error: Lightning Experience-Enabled Custom Domain Is Unavailable143

CHAPTER 1

How Salesforce Developer Experience Changes the Way You Work

In this chapter ...

- Use a Sample Repo to Get Started
- Create an Application
- Migrate or Import
 Existing Source

Salesforce Developer Experience (DX) is a new way to manage and develop apps on the Lightning Platform across their entire life cycle. It brings together the best of the Lightning Platform to enable source-driven development, team collaboration with governance, and new levels of agility for custom app development on Salesforce.

Highlights of Salesforce DX include:

- Your tools, your way. With Salesforce DX, you use the developer tools you already know.
- The ability to apply best practices to software development. Source code and metadata exist outside of the org and provide more agility to develop Salesforce apps in a team environment. Instead of the org, your version control system is the source of truth.
- A powerful command-line interface (CLI) removes the complexity of working with your Salesforce org for development, continuous integration, and delivery.
- Flexible and configurable scratch orgs that you build for development and automated environments. This new type of org makes it easier to build your apps and packages.
- You can use any IDE or text editor you want with the CLI and externalized source.
- Salesforce Extensions for VS Code to accelerate app development. These tools provide features for working with scratch orgs, Apex, Lightning components, and Visualforce.

Are You Ready to Begin?

Here's the basic order for doing your work using Salesforce DX. These workflows include the most common CLI commands. For all commands, see the *Salesforce CLI Command Reference*.

- Use a Sample Repo to Get Started on page 2
- Create an Application on page 2
- Migrate or Import Existing Source on page 3

SEE ALSO:

Salesforce DX (Salesforce Developer Center Web Site) Salesforce DX = UX for Developers (Salesforce Developer Blog) Salesforce CLI Command Reference

Use a Sample Repo to Get Started

The quickest way to get going with Salesforce DX is to clone the sfdx-simple GitHub repo. Use its configuration files and Salesforce application to try some commonly used Salesforce CLI commands.

1. Open a terminal or command prompt window, and clone the sfdx-simple GitHub sample repo using HTTPS or SSH.

```
git clone https://github.com/forcedotcom/sfdx-simple.git
--or--
git clone git@github.com:forcedotcom/sfdx-simple.git
```

2. Change to the sfdx-simple project directory.

cd sfdx-simple

3. Authorize your Developer Hub (Dev Hub) org, set it as your default, and assign it an alias.

```
sfdx force:auth:web:login --setdefaultdevhubusername --setalias DevHub
```

Enter your Dev Hub org credentials in the browser that opens. After you log in successfully, you can close the browser.

4. Create a scratch org using the config/project-scratch-def.json file, set the username as your default, and assign it an alias.

```
sfdx force:org:create --setdefaultusername -f config/project-scratch-def.json --setalias
my-scratch-org
```

5. Push source and tests, located in the force-app directory, to the scratch org.

sfdx force:source:push

6. Run Apex tests.

sfdx force:apex:test:run --resultformat human

7. Open the scratch org and view the pushed metadata under Most Recently Used.

sfdx force:org:open

SEE ALSO: Sample Repository on GitHub Authorization Create Scratch Orgs Push Source to the Scratch Org Testing

Create an Application

Follow the basic workflow when you are starting from scratch to create and develop an app that runs on the Lightning Platform.

1. Set up your project. on page 15

How Salesforce Developer Experience Changes the Way You Work

- 2. Authorize the Developer Hub org for the project. on page 24
- **3.** Configure your local project. on page 22
- **4.** Create a scratch org. on page 49
- 5. Push the source from your project to the scratch org. on page 57
- **6.** Develop the app. on page 77
- 7. Pull the source to keep your project and scratch org in sync. on page 64
- 8. Run tests. on page 83
- 9. Add, commit, and push changes. Create a pull request.

Deploy your app using one of the following methods:

- Build and release your app with managed packages on page 95
- Build and release your app using the Metadata API on page 88

Migrate or Import Existing Source

Use the Metadata API to retrieve the code, and then convert your source for use in a Salesforce DX project.

- Tip: If your current repo follows the directory structure that is created from a Metadata API retrieve, you can skip the retrieve step and go directly to converting the source.
- **1.** Set up your project. on page 15
- 2. Retrieve your metadata. on page 18
- 3. Convert the metadata formatted source you just retrieved to source format. on page 20
- 4. Authorize the Developer Hub org for the project. on page 24
- 5. Configure your local project. on page 22
- 6. Create a scratch org. on page 49
- 7. Push the source from your project to the scratch org. on page 57
- **8.** Develop the app. on page 77
- 9. Pull the source to sync your project and scratch org. on page 64
- 10. Run tests. on page 83
- **11.** Add, commit, and push changes. Create a pull request.

Deploy your app using one of the following methods:

- Build and release your app with managed packages. on page 95
- Build and release your app using the Metadata API. on page 88

CHAPTER 2 Salesforce CLI Configuration and Tips

In this chapter ...

- CLI Runtime
 Configuration Values
- Environment
 Variables
- Salesforce DX
 Usernames and Orgs
- Override or Add Definition File Options at the Command Line
- CLI Parameter
 Resolution Order
- Support for JSON Responses
- Log Messages and Log Levels
- CLI Deprecation
 Policy

Use the Salesforce command-line interface (CLI) for most Salesforce DX tasks. These tasks include authorizing a Dev Hub org, creating a scratch org, synchronizing source code between your scratch orgs and VCS, and running tests.

You can start using the CLI right after you install it.

The CLI commands are grouped into top-level topics. For example, the force top-level topic is divided into topics that group commands by functionality, such as the force:org commands to manage your orgs.

Run --help at each level to get more information.

```
sfdx --help// lists all top-level topicssfdx force --help// lists all the topics under forcesfdx force:org --help// lists all the commands in the topicforce:org// detailed info about theforce:org:open command// detailed info about the
```

Run this command to view all available commands in the force topic.

sfdx force:doc:commands:list

SEE ALSO:

Salesforce DX Setup Guide

CLI Runtime Configuration Values

You can set CLI runtime configuration values for your current project or for all projects. You can set two kinds of configuration values: global and local. Global values apply to all projects on your computer. Local values apply to a specific project. Local values override global values when commands are run from within a Salesforce DX project directory.

To set a configuration value for the current project:

sfdx force:config:set name=<value>

For local configuration values, you must issue this command from within the Salesforce DX project directory.

To set the value for all your projects:

sfdx force:config:set name=<value> --global

You can issue global commands anywhere or within any project, yet they apply to all the Salesforce CLI commands you run.

You can view the local and global configuration values that you have set. The output lists the local values for the project directory from which you are running the command and all global values.

sfdx force:config:list

To return one or more previously set configuration values, use force:config:get. It is often useful to specify JSON output for this command for easier parsing in a continuous integration (CI) environment. For example, to return the value of defaultusername and defaultdevhubusername:

```
sfdx force:config:get defaultusername defaultdevhubusername --json
```

To unset a configuration value, set it to no value. For example, to unset the instanceUrl configuration value:

sfdx force:config:set instanceUrl=

You can set these CLI configuration values.

Note: Alternately, you can set all CLI configuration values as environment variables. Environment variables override configuration values.

Configuration Value Name	Description	Environment Variable
apiVersion	The API version for a specific project or all projects. Normally, the Salesforce CLI assumes that you're using the same version of the CLI as the Dev Hub	SFDX_API_VERSION Example:
	org. However, let's say you decide to use the	SFDX_API_VERSION=42.0
	pre-release version of the CLI (v43 in Summer '18), but your Dev Hub org is running the current API version (v42 in Spring '18). In this case, you'd want to set this value to match the API version of your Dev Hub org (v42). This examples sets the API version for all projects (globally).	
	<pre>sfdx force:config:set apiVersion=42.0global</pre>	

Configuration Value Name	Description	Environment Variable				
	Be sure not to confuse this CLI configuration value with the sourceApiVersion on page 22 project configuration value, which has a similar name.					
defaultusername The username for an org that all commands run against by default.		SFDX_DEFAULTUSERNAME Example:				
		SFDX_DEFAULTUSERNAME=me@my.org				
defaultdevhubusemane	The username of your Dev Hub org that the force:org:create command defaults to.	SFDX_DEFAULTDEVHUBUSERNAME Example:				
		SFDX_DEFAULTDEVHUBUSERNAME=me@devhub.org				
instanceUrl	The URL of the Salesforce instance that is hosting your org.	SFDX_INSTANCE_URL Example:				
		SFDX_INSTANCE_URL=https://yoda.my.salesforce.com				

SEE ALSO:

Salesforce DX Usernames and Orgs Authorization Use an Existing Access Token Instead of Authorizing

Environment Variables

You can set environment variables to configure some values that Salesforce CLI and Salesforce DX tooling use.

Environment variables override CLI runtime configuration values. (Linux and Mac only) To set an environment variable for only the command you're running:

SFDX_API_VERSION=44.0 sfdx force:org:create -<options>

Salesforce CLI Environment Variables

Environment Variable	Description
FORCE_OPEN_URL	Specifies the web page that opens in your browser when you run force:org:open.Forexample, to open Lightning Experience, set to lightning. Equivalent to thepath parameter of force:org:open.

Environment Variable	Description				
FORCE_SHOW_SPINNER	Set to true to show a spinner animation on the command line when running asynchronous CLI commands. Default is false.				
FORCE_SPINNER_DELAY	Specifies the speed of the spinner in milliseconds. Default is 60.				
SFDX_API_VERSION	The API version for a specific project or all projects. Normally, the Salesforce CLI assumes that you're using the same version of the CLI as your production org. However, let's say you decide to use the pre-release version of the CLI (v43 in Summer '18), but your production org is running the current API version (v42 in Spring '18). In this case, you'd want to set this value to match the API version of your production org (v42).				
SFDX_CODE_COVERAGE_REQUIREMENT	Specifies the code coverage percentages that are displayed in green when you run force:apex:test:run or force:apex:test:report with thecodecoverage parameter. If the code coverage percentage for an Apex test is equal to or higher than this setting, it is displayed in green. If the percent is lower, it is displayed in red. Applies only to human-readable output. Default is 70%.				
SFDX_CONTENT_TYPE	All CLI commands output results in JSON format.				
SFDX_DEFAULTUSERNAME	Specifies the username of your default org so you don't have to use thetargetusername CLI parameter. Overrides the value of the defaultusername runtime configuration value.				
SFDX_DISABLE_AUTOUPDATE or SFDX_AUTOUPDATE_DISABLE (either var works)	Set to true to disable the auto-update feature of the CLI. By default, the CLI periodically checks for and installs updates.				
SFDX_DOMAIN_RETRY	Specifies the time, in seconds, that the CLI waits for the Lightning Experience custom domain to resolve and become available in a newly-created scratch org. The default value is 240 (4 minutes). Set the variable to 0 to bypass the Lightning Experience custom domain check entirely.				
SFDX_JSON_TO_STDOUT	Sends messages when Salesforce CLI commands fail to stdout instead of stderr. Setting this environment variable to true is particularly helpful for scripting use cases. Example: SFDX_JSON_TO_STDOUT=true				
SFDX_LOG_LEVEL	Sets the level of messages that the CLI writes to the log file.				
SFDX_MDAPI_TEMP_DIR	Places the files (in metadata format) in the specified directory when you run some CLI commands, such as force:source: <name>. Retaining these files can be useful for several reasons. You can debug problems that occur during</name>				

Environment Variable	Description				
	command execution. You can use the generated package.xml when running subsequent commands, or as a starting point for creating a manifest that includes all the metadata you care about.				
	SFDX_MDAPI_TEMP_DIR=/users/myName/myDXProject/metadata				
SFDX_NPM_REGISTRY	Sets the URL to a private npm server, where all packages that you publish are private. We support only repositories that don't require authentication.				
	SFDX_NPM_REGISTRY= <full_url></full_url>				
	Example:				
	SFDX_NEM_REGISIRY=http://mypkgs.myclient.com/npm/my_npm_pkg				
	Verdaccio is an example of a lightweight private npm proxy registry.				
SFDX_PRECOMPILE_ENABLE	Set to true to enable Apex pre-compile before the tests are run. This variable works with the force:apex:test:run command. Default is false.				
	() Important: The duration of an Apex test pre-compilation can be inconsistent. As a result, runs of the same Apex tests are sometimes quick and other times they time out. We recommend that you set this variable to true only if your Apex tests (without pre-compile) activate multiple concurrent Apex compilations that consume many system resources.				
SFDX_PROJECT_AUTOUPDATE_DISABLE_FOR_PACKAGE_CREATE	For force:package:create, disables automatic updates to the sfdx-project.json file.				
SFDX_PROJECT_AUTOUPDATE_DISABLE_FOR_PACKAGE_VERSION_CREATE	For force:package:version:create, disables automatic updates to the sfdx-project.json file.				
SFDX_USE_GENERIC_UNIX_KEYCHAIN	(Linux and macOS only) Set to true if you want to use the generic UNIX keychain instead of the Linux libsecret library or macOS keychain. Specify this variable when using the CLI with ssh or "headless" in a CI environment.				

General Environment Variables

Environment Variable	Description
HTTP_PROXY	If you receive an error when you install or update the Salesforce CLI on a computer that's behind a firewall or web proxy, set this environment variable. Use the URL and port of your company proxy, for example:

Environment Variable	Description			
	http://username:pwd@proxy.company.com:8080			
HTTPS_PROXY	If you receive an error when you install or update the Salesforce CLI on a computer that's behind a firewall or web proxy, set this environment variable. Use the URL and port of your company proxy, for example: http://username:pwd@proxy.company.com:8080			
NODE_EXTRA_CA_CERTS	Installs your self-signed certificate. Indicate the fully qualified path to the certificate file name. Then run sfdx update. See NODE_EXTRA_CA_CERTS=file for more details.			
NODE_TLS_REJECT_UNAUTHORIZED	Indicate <i>0</i> to allow Node.js to use the self-signed certificate in the certificate chain.			

SEE ALSO:

Log Messages and Log Levels Support for JSON Responses

Salesforce DX Usernames and Orgs

Many CLI commands connect to an org to complete their task. For example, the force:org:create command, which creates a scratch org, connects to a Dev Hub org. The force:source:push|pull commands synchronize source code between your project and a scratch org. In each case, the CLI command requires a username to determine which org to connect to. Usernames are unique within the entire Salesforce ecosystem and have a one-to-one association with a specific org.

Note: The examples in this topic might refer to CLI commands that you are not yet familiar with. For now, focus on how to specify the usernames, configure default usernames, and use aliases. The CLI commands are described later.

When you create a scratch org, the CLI generates a username. The username looks like an email address, such as test-wvkpnfm5z113@example.com. You do not need a password to connect to or open a scratch org, although you can generate one later with the force:user:password:generate command.

Salesforce recommends that you set a default username for the orgs that you connect to the most during development. The easiest way to do this is when you authorize a Dev Hub org or create a scratch org. Specify the --setdefaultdevhubusername or --setdefaultusername parameter, respectively, from within a project directory. You can also create an alias to give the usernames more readable names. You can use usernames or their aliases interchangeably for all CLI commands that connect to an org.

These examples set the default usernames and aliases when you authorize an org and then when you create a scratch org.

```
sfdx force:auth:web:login --setdefaultdevhubusername --setalias my-hub-org
sfdx force:org:create --definitionfile my-org-def.json --setdefaultusername --setalias
my-scratch-org
```

To verify whether a CLI command requires an org connection, look at its parameter list with the --help parameter. Commands that have the --targetdevhubusername parameter connect to the Dev Hub org. Similarly, commands that have

--targetusername connect to scratch orgs, sandboxes, and so on. This example displays the parameter list and help information about force:org:create.

sfdx force:org:create --help

When you run a CLI command that requires an org connection and you don't specify a username, the command uses the default. To see your default usernames, run force:org:list to display all the orgs you've authorized or created. The default Dev Hub and scratch orgs are marked on the left with (D) and (U), respectively.

Let's run through a few examples to see how this works. This example pushes source code to the scratch org that you've set as the default.

```
sfdx force:source:push
```

To specify an org other than the default, use --targetusername. For example, let's say you created another scratch org with alias my-other-scratch-org. It's not the default but you still want to push source to it.

sfdx force:source:push --targetusername my-other-scratch-org

This example shows how to use the --targetdevhubusername parameter to specify a non-default Dev Hub org when creating a scratch org.

```
sfdx force:org:create --targetdevhubusername jdoe@mydevhub.com --definitionfile
my-org-def.json --setalias yet-another-scratch-org
```

More About Setting Default Usernames

If you've already created a scratch org, you can set the default username with the force:config:set command from your project directory.

```
sfdx force:config:set defaultusername=test-wvkpnfm5z113@example.com
```

The command sets the value locally, so it works only for the current project. To use the default username for all projects on your computer, specify the --global parameter. You can run this command from any directory. Local project defaults override global defaults.

sfdx force:config:set defaultusername=test-wvkpnfm5z113@example.com --global

The process is similar to set a default Dev Hub org, except you use the defaultdevhubusername config value.

sfdx force:config:set defaultdevhubusername=jdoe@mydevhub.com

More About Aliasing

Use the force:alias:set command to set an alias for an org or after you've authorized an org. You can create an alias for any org: Dev Hub, scratch, production, sandbox, and so on. So when you issue a command that requires the org username, using an alias for the org that you can easily remember can speed up things.

sfdx force:alias:set my-scratch-org=test-wvkpnfm5z113@example.com

An alias also makes it easy to set a default username. The previous example of using force:config:set to set defaultusername now becomes much more digestible when you use an alias rather than the username.

sfdx force:config:set defaultusername=my-scratch-org

Set multiple aliases with a single command by separating the name-value pairs with a space.

sfdx force:alias:set org1=<username> org2=<username>

You can associate an alias with only one username at a time. If you set it multiple times, the alias points to the most recent username. For example, if you run the following two commands, the alias my-org is set to test-ymmlqf29req5@your_company.net.

```
sfdx force:alias:set my-org=test-blahdiblah@whoanellie.net
sfdx force:alias:set my-org=test-wvkpnfm5z113@example.com
```

To view all aliases that you've set, use one of the following commands.

```
sfdx force:alias:list
sfdx force:org:list
```

To remove an alias, set it to nothing.

```
sfdx force:alias:set my-org=
```

List All Your Orgs

Use the force:org:list command to display the usernames for the orgs that you've authorized and the active scratch orgs that you've created.

sfdx force:org:list									
===	Orgs								
	ALIAS	USERNAME		ORG ID	CONNEC	red s	STATUS		
	DD-ORG	jdoe@dd-204.c	om	00DOEA	Connect	ted			
(D)	devhuborg	jdoe@mydevhub	.com	00DMAC	Connect	ted			
	ALIAS	SCRATCH ORG NAME	USERNAM	E		ORG	ID	EXPIRATION	DATE
	my-scratch	Your Company	test-wv	km5z113@exam	ple.com	00D.	UAI	2017-06-13	
(U)	scratch208	Your Company	test-wv	km5z1130exam	ple.com	00D.	UAY	2017-06-13	

The top section of the output lists the non-scratch orgs that you've authorized, including Dev Hub orgs, production orgs, and sandboxes. The output displays the usernames that you specified when you authorized the orgs, their aliases, their IDs, and whether the CLI can connect to it. A (D) on the left points to the default Dev Hub org username.

The lower section lists the active scratch orgs that you've created and their usernames, org IDs, and expiration dates. A (U) on the left points to the default scratch org username.

To view more information about scratch orgs, such as the create date, instance URL, and associated Dev Hub org, use the --verbose parameter.

sfdx force:org:list --verbose

Use the --clean parameter to remove non-active scratch orgs from the list. The command prompts you before it does anything.

```
sfdx force:org:list --clean
```

SEE ALSO: Authorization Scratch Org Definition File Create Scratch Orgs Generate or Change a Password for a Scratch Org User Push Source to the Scratch Org

Override or Add Definition File Options at the Command Line

Some CLI commands, such as force:org:create and force:user:create, use a JSON definition file to determine the characteristics of the org or user they create. The definition file contains one or more options. You can override some options by specifying them as name-value pairs at the command line. You can also specify options that aren't in the definition file. This technique allows multiple users or continuous integration jobs to share a base definition file and then customize options when they run the command.

Let's say you use the following JSON definition file to create a scratch org. You name the file project-scratch-def.json.

```
{
  "orgName": "Acme",
  "country": "US",
  "edition": "Enterprise",
  "hasSampleData": "true",
  "features": ["MultiCurrency", "AuthorApex"],
  "orgPreferences": {
     "enabled": ["S1DesktopEnabled", "ChatterEnabled"],
     "disabled": ["IsNameSuffixEnabled"]
  }
}
```

To create an Enterprise Edition scratch org that uses all the options in the file, run this command.

sfdx force:org:create --definitionfile project-scratch-def.json

You can then use the same definition file to create a Developer Edition scratch org that doesn't have sample data by overriding the edition and hasSampleData options.

```
sfdx force:org:create --definitionfile project-scratch-def.json edition=Developer
hasSampleData=false
```

Use commas to separate multiple array values, and enclose them in double quotes. For example, to change the features option:

sfdx force:org:create --definitionfile project-scratch-def.json
features="MultiCurrency, PersonAccounts"

This example shows how to add the adminEmail option, which doesn't exist in the definition file.

sfdx force:org:create --definitionfile project-scratch-def.json adminEmail=john@doe.org

Note: You can't override options whose values are JSON objects, such as orgPreferences.

SEE ALSO:

Create Scratch Orgs Create a Scratch Org User

CLI Parameter Resolution Order

Because you can specify parameters for a given CLI command in several ways, it's important to know the order of parameter resolution. The order of precedence for parameter resolution is:

- 1. Command-line parameters, such as --loglevel, --targetusername, or --targetdevhubusername.
- 2. Parameters listed in a file specified by the command line. An example is a scratch org definition in a file specified by the --definitionfile parameter of force:org:create.
- 3. Environment variables, such as SFDX_LOG_LEVEL.
- 4. Local CLI configuration values, such as defaultusername or defaultdevhubusername. To view the local values, run force:config:list from your project directory.
- 5. Global CLI configuration values. To view the global values, run force:config:list from any directory.

For example, if you set the SFDX_LOG_LEVEL environment variable to INFO but specify --loglevel DEBUG for a command, the log level is DEBUG. This behavior happens because command-line parameters are at the top of the precedence list.

If you specify the --targetusername parameter for a specific CLI command, the CLI command connects to an org with that username. It does not connect to an org using the defaultusername, assuming that you set it previously with the force:config:set command.

Support for JSON Responses

Salesforce CLI commands typically display their output to the console (stdout) in non-structured, human-readable format. Messages written to the log file (stderr) are always in JSON format.

To view the console output in JSON format, specify the --json parameter for a particular CLI command.

sfdx force:org:display --json

Most CLI commands support JSON output. To confirm, run the command with the --help parameter to view the supported parameters.

To get JSON responses to all Salesforce CLI commands without specifying the --json option each time, set the SFDX_CONTENT_TYPE environment variable.

```
export SFDX_CONTENT_TYPE=JSON
```

Log Messages and Log Levels

Salesforce CLI writes all log messages to the USER_HOME_DIR/.sfdx.log file. CLI invocations append log messages to this running log file. Only errors are output to the terminal or command window from which you run the CLI.

Important: The files in the USER_HOME_DIR/.sfdx directory are used internally by Salesforce CLI. Do not remove or edit them.

The default level of log messages is warn. You can set the log level to one of the following, listed in order of least to most information. The level is cumulative: for the debug level. The --loglevel parameter supports parameter values in only lowercase (due to the migration to oclif). To assist you with the transition, we support uppercase parameters in Spring '19 but plan to deprecate support for them in Summer '19.

- error
- warn
- info
- debug
- trace
- fatal

You can change the log level in two ways, depending on what you want to accomplish.

To change the log level for the execution of a single CLI command, use the --loglevel parameter. Changing the log level in this way does not affect subsequent CLI use. This example specifies debug-level log messages when you create a scratch org.

sfdx force:org:create --definitionfile config/project-scratch-def.json --loglevel debug
--setalias my-scratch-org

To globally set the log level for all CLI commands, set the SFDX_LOG_LEVEL environment variable. For example, on UNIX:

export SFDX_LOG_LEVEL=debug

Note: Salesforce CLI gathers diagnostic information about its use and reports it to Salesforce so that the development team can investigate issues. The type of information includes command duration and command invocation counts.

CLI Deprecation Policy

Salesforce deprecates CLI commands and parameters when, for example, the underlying API changes.

The Salesforce CLI deprecation policy is:

- Salesforce can deprecate a command or parameter in any major update of the salesforcedx plug-in.
- Salesforce removes the deprecated command or parameter in the next major release of the salesforcedx plug-in. For example, if Salesforce deprecates a command in version 41, it does not appear in version 42.
- If you use a command or parameter that's been deprecated but not yet removed, you get a warning message in stderr when you specify human-readable output. If you specify JSON output, the warning is presented as a property. The message includes the plug-in version of when the command or parameter will be removed. The command help also includes deprecation information when appropriate.
- When possible, Salesforce provides a functional alternative to the deprecated command or parameter.
- Salesforce announces new and upcoming deprecated commands and parameters in the release notes.

CHAPTER 3 Project Setup

In this chapter ...

- Sample Repository on GitHub
- Create a Salesforce
 DX Project
- Create a Salesforce
 DX Project from
 Existing Source
- Retrieve Source from an Existing Managed Package
- Retrieve Unpackaged Source Defined in a package.xml File
- Convert the Metadata Source to Source Format
- Link a Namespace to a Dev Hub Org
- Salesforce DX Project
 Configuration

Salesforce DX introduces a new project structure for your org's metadata (code and configuration), your org templates, your sample data, and all your team's tests. Store these items in a version control system (VCS) to bring consistency to your team's development processes. Retrieve the contents of your team's repository when you're ready to develop a new feature.

You can use your preferred VCS. Most of our examples use Git.

You have different options to create a Salesforce DX project depending on how you want to begin.

Use the Sample Repository on GitHub on page 16	Explore the features of Salesforce DX using one of our sample repos and your own VCS and toolset.
Create a Salesforce DX Project from Existing Source on page 18	Start with an existing Salesforce app to create a Salesforce DX project.
Create a Salesforce DX Project on page 16	Create an app on the Lightning Platform using a Salesforce DX project.

Sample Repository on GitHub

If you want to check out Salesforce DX features quickly, start with the sfdx-simple GitHub repo. It contains an example of the project configuration file (sfdx-project.json), a simple Salesforce app, and Apex tests.

Cloning this repo creates the directory sfdx-simple. See the repo's Readme for more information.

Assuming that you've already set up Git, use the git clone command to clone the master branch of the repo from the command line.

To use HTTPS:

git clone https://github.com/forcedotcom/sfdx-simple.git

To use SSH:

git clone git@github.com:forcedotcom/sfdx-simple.git

If you don't want to use Git, download a .zip file of the repository's source using Clone, or download on the GitHub website. Unpack the source anywhere on your local file system.



Tip: To check out more complex examples, see the Sample Gallery.

It contains sample apps that show what you can build on the Salesforce platform. They're continuously updated to incorporate the latest features and best practices.

SEE ALSO:

sfdx-simple Sample GitHub Repo dreamhouse-sfdx Sample GitHub Repo

Create a Salesforce DX Project

A Salesforce DX project has a specific structure and a configuration file that identifies the directory as a Salesforce DX project.

You can create a project with minimal (empty) or expanded (standard) scaffolding. The default is standard, which provides extended scaffolding to facilitate moving source to and from your orgs.

- 1. Change to the directory where you want the DX project located.
- 2. Create the DX project.

force:project:create -n MyProject --template standard

If you don't indicate an output directory, the project directory is created in the current location. You can also specify the default package directory to target when syncing source to and from the scratch org. If you don't indicate a default package directory, this command creates a default package directory, force-app.

The default --template value, empty, creates a project with these sample configuration files to get you started.

- .forceignore
- config/project-scratch-def.json
- sfdx-project.json

If you select expanded scaffolding (--template standard), you get a complete directory structure that takes the guesswork out of where to put your source. It also provides these files that are especially helpful when using Salesforce Extensions for VS Code.

- .gitignore: Makes it easier to start using Git for version control.
- .prettierrc and .prettierignore: Make it easier to start using Prettier to format your Aura components.
- .vscode/extensions.json: Causes Visual Studio Code, when launched, to prompt you to install the recommended extensions for your project.
- .vscode/launch.json: Configures Replay Debugger, making it more discoverable and easier to use.
- .vscode/settings.json: By default, this file has one setting, for push or deploy on save, which is set to false. You can change this value or add other settings.



Stample:



Next steps:

- (Optional) Register the namespace with the Dev Hub org.
- Configure the project (sfdx-project.json). If you use a namespace, update this file to include it.

Project Setup

 Create a scratch org definition that produces scratch orgs that mimic the shape of another org you use in development, such as sandbox, packaging, or production. The config directory of your new project contains a sample scratch org definition file (project-scratch-def.json).

SEE ALSO:

Create a Salesforce DX Project from Existing Source Salesforce DX Project Configuration Link a Namespace to a Dev Hub Org Scratch Org Definition File How to Exclude Source When Syncing or Converting

Create a Salesforce DX Project from Existing Source

If you are already a Salesforce developer or ISV, you likely have existing source in a managed package in your packaging org or some application source in your sandbox or production org. Before you begin using Salesforce DX, retrieve the existing source and convert it to the source format.



Tip: If your current repo follows the directory structure that is created from a Metadata API retrieve, you can skip to converting the metadata format after you create a Salesforce DX project.

- 1. Create a Salesforce DX project.
- 2. Create a directory for the metadata retrieve. You can create this directory anywhere.

mkdir mdapipkg

3. Retrieve your metadata source.

Format of Current Source	How to Retrieve Your Source for Conversion
You are a partner who has your source already defined as a managed package in your packaging org.	Retrieve Source from an Existing Managed Package on page 18
You have a package.xml file that defines your unpackaged source.	Retrieve Unpackaged Source Defined in a package.xml File on page 19

SEE ALSO:

Convert the Metadata Source to Source Format Create a Salesforce DX Project

Retrieve Source from an Existing Managed Package

If you're a partner or ISV who already has a managed package in a packaging org, you're in the right place. You can retrieve that package, unzip it to your local project, and then convert it to source format, all from the CLI.

Before you begin, create a Salesforce DX project.

1. In the project, create a folder to store what's retrieved from your org, for example, mdapipkg.

2. Retrieve the metadata.

sfdx force:mdapi:retrieve -s -r ./mdapipkg -u <username> -p <package name>

The username can be a username or alias for the target org (such as a packaging org) from which you're pulling metadata. The -s parameter indicates that you're retrieving a single package. If your package name contains a space, enclose the name in single quotes.

-p 'Test Package'

3. Check the status of the retrieve.

When you run force:mdapi:retrieve, the job ID, target username, and retrieve directory are stored, so you don't have to specify these required parameters to check the status. These stored values are overwritten when you run the force:mdapi:retrieve again.

sfdx force:mdapi:retrieve:report

If you want to check the status of a different retrieve operation, specify the retrieve directory and job ID on the command line, which overrides any stored values.

- 4. Unzip the zip file.
- 5. (Optional) Delete the zip file.

After you finish, convert the metadata to source format.

SEE ALSO:

Create a Salesforce DX Project

Convert the Metadata Source to Source Format

Retrieve Unpackaged Source Defined in a package.xml File

If you already have a package.xml file, you can retrieve it, unzip it in your local project, and convert it to source format. You can do all these tasks from the CLI. The package.xml file defines the source you want to retrieve.

But what if you don't have a package.xml file already created? See Sample package.xml Manifest Files in the Metadata API Developer Guide.

Note: If you already have the source in metadata format, you can skip these steps and go directly to converting it to source format.

- 1. In the project, create a folder to store what's retrieved from your org, for example, mdapipkg.
- 2. Retrieve the metadata.

sfdx force:mdapi:retrieve -r ./mdapipkg -u <username> -k ./package.xml

The username can be the scratch org username or an alias. The -k parameter indicates the path to the package.xml file, which is the unpackaged manifest of components to retrieve.

3. Check the status of the retrieve.

When you run force:mdapi:retrieve, the job ID, target username, and retrieve directory are stored, so you don't have to specify these required parameters to check the status. These stored values are overwritten when you run the force:mdapi:retrieve again.

sfdx force:mdapi:retrieve:report

If you want to check the status of a different retrieve operation, specify the retrieve directory and job ID on the command line, which overrides any stored values.

- 4. Unzip the zip file.
- 5. (Optional) Delete the zip file.

After you retrieve the source and unzip it, you no longer need the zip file, so you can delete it.

After you finish, convert from metadata format to source format.

SEE ALSO:

Convert the Metadata Source to Source Format

Convert the Metadata Source to Source Format

After you retrieve the source from your org, you can complete the configuration of your project and convert the metadata source to source format.

The convert command ignores all files that start with a "dot," such as .DS_Store. To exclude more files from the convert process, add a .forceignore file.

1. Convert metadata format to source format. Let's say you created a directory called mdapi_project when you retrieved the metadata.

sfdx force:mdapi:convert --rootdir mdapi_project --outputdir tmp_convert

The --rootdir parameter is the name of the directory that contains the metadata source.

If you don't indicate an output directory with the --outputdir parameter, the converted source is stored in the default package directory indicated in the sfdx-project.json file. If the output directory is located outside of the project, you can indicate its location using an absolute path.

2. To indicate which package directory is the default, update the sfdx-project.json file.

If there are two or more files with the same file name yet they contain different contents, the output directory contains duplicate files. Duplicate files can occur if you convert the same set of metadata more than once. The mdapi:convert process identifies these files with a .dup file extension. The source:push and source:pull commands ignore duplicate files, so you'll want to resolve them. You have these options:

- Choose which file to keep, then delete the duplicate.
- Merge the files, then delete the other.

Next steps:

- Authorize the Dev Hub org and set it as the default
- Configure the Salesforce DX project
- Create a scratch org

SEE ALSO:

How to Exclude Source When Syncing or Converting Salesforce DX Project Configuration Authorization Create Scratch Orgs

Link a Namespace to a Dev Hub Org

To use a namespace with a scratch org, you must link the Developer Edition org where the namespace is registered to a Dev Hub org.

Complete these tasks before you link a namespace.

- If you don't have an org with a registered namespace, create a Developer Edition org that is separate from the Dev Hub or scratch orgs. If you already have an org with a registered namespace, go to Step 1.
- In the Developer Edition org, create and register the namespace.
 - () Important: Choose namespaces carefully. If you're trying out this feature or need a namespace for testing purposes, choose a disposable namespace. Don't choose a namespace that you want to use in the future for a production org or some other real use case. Once you associate a namespace with an org, you can't change it or reuse it.
- 1. Log in to your Dev Hub org as the System Administrator or as a user with the Salesforce DX Namespace Registry permissions.

1 Tip: Make sure your browser allows pop-ups from your Dev Hub org.

- 2. (Required) If you have not already done so, define and deploy a My Domain name.
 - Tip: Why do you need a My Domain? A My Domain adds a subdomain to your Salesforce org URL so that it's unique. As part of the Namespace Registry linking process, you'll be logging into two distinct orgs simultaneously (your Dev Hub org and your Developer Edition org), and your browser can't reliably distinguish between the two without a My Domain.

You receive an email when your domain name is ready for testing. It can take a few minutes.

- 3. From the App Launcher menu, select Namespace Registries.
- 4. Click Link Namespace.

If you don't see the Link Namespace button, make sure your My Domain is deployed to users.

- **a.** From Setup, enter *My Domain* in the Quick Find box, then select **My Domain**.
- b. Do you see the status as Domain Deployed to Users? If not, click Deploy to Users.Log out of your Dev Hub org, then open it again.
- Log in to the Developer Edition org in which your namespace is registered using the org's System Administrator's credentials.
 You cannot link orgs without a namespace, sandboxes, scratch orgs, patch orgs, and branch orgs to the Namespace Registry.

To view all the namespaces linked to the Namespace Registry, select the **All Namespace Registries** view.

SEE ALSO:

Create a Developer Edition Org Lightning Aura Components Developer Guide: Create a Namespace in Your Org Salesforce DX Setup Guide: Add Salesforce DX Users Salesforce Help: Define Your Domain Name Salesforce Help: My Domain Salesforce Help: Test and Deploy Your New My Domain Subdomain

Salesforce DX Project Configuration

The project configuration file sfdx-project.json indicates that the directory is a Salesforce DX project. The configuration file contains project information and facilitates the authentication of scratch orgs and the creation of second-generation packages. It also tells the CLI where to put files when syncing between the project and scratch org.

We provide sample sfdx-project.json files in the sample repos for creating a project using the CLI or Extensions for VS Code.



Note: Are you planning to create second-generation packages? When you're ready, add packaging-specific configuration options to support package creation. See Configure Packages.

We recommend that you check in this file with your source.

```
{
    "packageDirectories" : [
        { "path": "force-app", "default": true},
        { "path" : "unpackaged" },
        { "path" : "utils" }
    ],
    "namespace": "",
    "sfdcLoginUrl" : "https://login.salesforce.com",
    "sourceApiVersion": "44.0"
}
```

You can manually edit these parameters.

oauthLocalPort (optional)

By default, the OAuth port is 1717. However, change this port if this port is already in use, and you plan to create a connected app in your Dev Hub org to support JWT-based authorization. Also, don't forget to follow the steps in Create a Connected App to change the callback URL.

packageDirectories (required)

Package directories indicate which directories to target when syncing source to and from the scratch org. These directories can contain source from your managed package, unmanaged package, or unpackaged source, for example, ant tool or change set.

Keep these things in mind when working with package directories.

• The location of the package directory is relative to the project. Don't specify an absolute path. The following two examples are equivalent.

```
"path": "helloWorld"
"path" : "./helloWorld"
```

- You can have only one default path (package directory). If you have only one path, we assume it's the default, so you don't have to explicitly set the default parameter. If you have multiple paths, you must indicate which one is the default.
- The CLI uses the default package directory as the target directory when pulling changes in the scratch org to sync the local project. This default path is also used when creating second-generation packages.
- If you do not specify an output directory, the default package directory is also where files are stored during source conversions. Source conversions are both from metadata format to source format, and from source format to metadata format.

plugins (optional)

To use the plug-ins you develop using the Salesforce Plugin Generator with your Salesforce DX project, add a plugins section to the sfdx-project.json file. In this section, add configuration values and settings to change your plug-ins' behavior.

```
"plugins": {
    "yourPluginName": {
        "timeOutValue": "2"
    },
    "yourOtherPluginName": {
        "yourCustomProperty": true
    }
}
```

Store configuration values for only those values that you want to check in to source control for the project. These configuration values affect your whole development team.

namespace (optional)

The global namespace that is used with a package. The namespace must be registered with an org that is associated with your Dev Hub org. This namespace is assigned to scratch orgs created with the org:create command. If you're creating an unlocked package, you have the option to create a package with no namespace.

Important: Register the namespace with Salesforce and then connect the org with the registered namespace to the Dev Hub org.

sfdcLoginUrl (optional)

The login URL that the force:auth commands use. If not specified, the default is login.salesforce.com. Override the default value if you want users to authorize to a specific Salesforce instance. For example, if you want to authorize into a sandbox org, set this parameter to test.salesforce.com.

If you do not specify a default login URL here, or if you run force:auth outside the project, you specify the instance URL when authorizing the org.

sourceApiVersion (optional)

The API version that the source is compatible with. The default is the same version as the Salesforce CLI.

The sourceApiVersion determines the fields retrieved for each metadata type during source:push, source:pull, or source:convert. This field is important if you're using a metadata type that has changed in a recent release. You'd want to specify the version of your metadata source. For example, an icon field was added to the CustomTab for API version 14.0. If you retrieve components for version 13.0 or earlier, you'll see errors when running the source commands because the components do not include the icon field.

Be sure not to confuse this project configuration value with the apiVersion on page 22 CLI runtime configuration value, which has a similar name.

SEE ALSO:

Link a Namespace to a Dev Hub Org Authorization How to Exclude Source When Syncing or Converting Pull Source from the Scratch Org to Your Project Push Source to the Scratch Org

CHAPTER 4 Authorization

In this chapter ...

- Authorize an Org Using the Web-Based Flow
- Authorize an Org Using the JWT-Based Flow
- Create a Private Key and Self-Signed Digital Certificate
- Create a Connected
 App
- Use an Existing Access Token Instead of Authorizing
- Authorization Information for an Org
- Log Out of an Org

The Dev Hub org allows you to create, delete, and manage your Salesforce scratch orgs. After you set up your project on your local machine, you authorize with the Dev Hub org before you can create a scratch org.

You can also authorize other existing orgs, such as sandbox or packaging orgs, to provide more flexibility when using CLI commands. For example, after developing and testing an application using scratch orgs, you can deploy the changes to a centralized sandbox. Or, you can export a subset of data from an existing production org and import it into a scratch org for testing purposes.

You authorize an org only once. To switch between orgs during development, specify your username for the org. Use either the --targetusername (or --targetdevhubusername) CLI command parameter, set a default username, or use an alias.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Dev Hub available in: Developer,Enterprise, Performance, and Unlimited Editions

Scratch orgs available in: **Developer**, **Enterprise**, **Group**, and **Professional** Editions

You have some options when configuring authentication depending on what you're trying to accomplish.

- We provide the OAuth Refresh Token flow, also called web-based flow, through a global out-of-the-box connected app. When you authorize an org from the command line, you enter your credentials and authorize the global connected app through the Salesforce web browser authentication flow.
- For continuous integration or automated environments in which you don't want to manually enter credentials, use the OAuth JSON Web Tokens (JWT) Bearer Token flow, also called JWT-based flow. This authentication flow is ideal for scenarios where you cannot interactively log in to a browser, such as a continuous integration script.
- Important: If your Dev Hub org is configured with high assurance (stepped up) authentication, Salesforce prompts the user to verify identity. This verification process means that you can't use the JWT flow and Salesforce CLI for headless authentication.

SEE ALSO:

Authorize an Org Using the Web-Based Flow Authorize an Org Using the JWT-Based Flow Salesforce DX Usernames and Orgs

Authorize an Org Using the Web-Based Flow

To authorize an org with the web-based flow, all you do is run a CLI command. Enter your credentials in a browser, and you're up and running!

Authorization requires a connected app. We provide a connected app that is used by default. If you need more security or control, such as setting the refresh token timeout or specifying IP ranges, you can optionally create a connected app.

- 1. (Optional) Create a connected app if you require more security and control than offered by the provided connected app. Enable OAuth settings for the new connected app. Make note of the consumer key because you need it later.
- 2. If the org you are authorizing is on a My Domain subdomain, update your project configuration file (sfdx-project.json). Set the sfdcLoginUrl parameter to your My Domain login URL. If you are authorizing a sandbox, set the parameter to https://test.salesforce.com. For example:

```
"sfdcLoginUrl" : "https://test.salesforce.com"
```

"sfdcLoginUrl" : "https://somethingcool.my.salesforce.com"

Alternatively, use the --instanceurl parameter of the force:auth:web:login command, as shown in the next step, to specify the URL.

3. Run the force:auth:web:login CLI command. If you are authorizing a Dev Hub org, use the

--setdefaultdevhubusername parameter if you want the Dev Hub org to be the default for commands that accept the --targetdevhubusername parameter.

sfdx force:auth:web:login --setdefaultdevhubusername --setalias my-hub-org
sfdx force:auth:web:login --setalias my-sandbox

If you are using your own connected app, use the --clientid parameter. For example, if your client identifier (also called the consumer key) is 04580y4051234051 and you are authorizing a Dev Hub org:

```
sfdx force:auth:web:login --clientid 04580y4051234051 --setdefaultdevhubusername
--setalias my-hub-org
```

To specify a login URL other than the default, such as https://test.salesforce.com:

```
sfdx force:auth:web:login --setalias my-hub-org --instanceurl https://test.salesforce.com
```

Important: Use the --setdefaultdevhubusername parameter only when authorizing a Dev Hub org. Do not use it when authorizing to other orgs, such as a sandbox.

- 4. In the browser window that opens, sign in to your org with your credentials.
- 5. Close the browser window, unless you want to explore the org.

```
SEE ALSO:
```

Create a Connected App Salesforce DX Project Configuration

Authorize an Org Using the JWT-Based Flow

Continuous integration (CI) environments are fully automated and don't support the human interactivity of a web-based flow. In these environments, you must use the JSON web tokens (JWT) to authorize an org.

The JWT-based authorization flow requires first generating a digital certificate and creating a connected app. You execute these tasks only once. After that, you can authorize the org in a script that runs in your Cl environment.

For information about using JWT-based authorization with Travis CI, see the Continuous Integration Using Salesforce DX Trailhead module.

() Important: If your Dev Hub org is configured with high assurance (stepped up) authentication, Salesforce prompts the user to verify identity. This verification process means that you can't use the JWT flow and Salesforce CLI for headless authentication.

- 1. If you do not have your own private key and digital certificate, use OpenSSL to create the key and a self-signed certificate. It is assumed in this task that your private key file is named server.key and your digital certificate is named server.crt.
- 2. Create a connected app, and configure it for Salesforce DX.

This task includes uploading the server.crt digital certificate file. Make note of the consumer key when you save the connected app because you need it later.

 If the org you are authorizing is not hosted on https://login.salesforce.com, update your project configuration file (sfdx-project.json).

Set the sfdcLoginUrl parameter to the login URL. Examples of other login URLs are your custom subdomain or https://test.salesforce.com for sandboxes. For example:

"sfdcLoginUrl": "https://test.salesforce.com"

Important: If you specify a My Domain subdomain for the login URL, use the version that ends in my.salesforce.com instead of the URL you see in Lightning Experience (.lightning.force.com). To verify the valid My Domain URL, from Setup, enter My Domain in the Quick Find box, then select My Domain.

Alternatively, you can use the --instanceurl parameter of the force:auth:jwt:grant command to specify the URL. This value overrides the login URL you specified in the sfdx-project.json file. See the next step for an example.

4. Run the force:auth:jwt:grant CLI command.

Specify the client identifier from your connected app (also called the consumer key), the path to the private key file (server.key), and the JWT authentication username. When you authorize a Dev Hub org, set it as the default with the --setdefaultdevhubusername parameter. For example:

sfdx force:auth:jwt:grant --clientid 04580y4051234051 \
inthe file (News (idea (News)))

```
--jwtkeyfile /Users/jdoe/JWT/server.key --username jdoe@acdxgs0hub.org \
--setdefaultdevhubusername --setalias my-hub-org
```

This example shows how to use the --instanceurl parameter to specify an org hosted on https://test.salesforce.com rather than the default https://login.salesforce.com:

```
sfdx force:auth:jwt:grant --clientid 04580y4051234051 \
--jwtkeyfile /Users/jdoe/JWT/server.key --username jdoe@acdxgs0hub.org \
--instanceurl https://test.salesforce.com
```

You can authorize a scratch org using the same client identifier (consumer key) and private key file that you used to authorize its associated Dev Huborg. Set the --instanceurl parameter to https://test.salesforce.com and the --username parameter to the administrator user displayed after you create the scratch org with

Authorize a Scratch Org

If you configured your Dev Hub to support the JWT-based authorization flow, you can use the same digital certificate and private key to authorize an associated scratch org. This method is useful for continuous integration (CI) systems that must authorize scratch orgs after creating them, but don't have access to the scratch org's access token.

```
SEE ALSO:
```

Create a Private Key and Self-Signed Digital Certificate

- Create a Connected App
- Salesforce DX Project Configuration
- Trailhead: Create Your Connected App (Continuous Integration Using Salesforce DX Module)
- Salesforce Help: Set up Two-Factor Authentication
- Create, Clone, or Delete a Sandbox (Beta)

Authorize a Scratch Org

If you configured your Dev Hub to support the JWT-based authorization flow, you can use the same digital certificate and private key to authorize an associated scratch org. This method is useful for continuous integration (CI) systems that must authorize scratch orgs after creating them, but don't have access to the scratch org's access token.

It is assumed in this task that:

- You previously authorized your Dev Hub org using the JWT-based flow.
- The private key file you used when authorizing your Dev Hub org is accessible and located in /Users/jdoe/JWT/server.key.
- You've created a scratch org and have its administration user's username, such as test-wvkpnfm5z113@example.com.
- 1. Copy the consumer key from the connected app that you created in your Dev Hub org.
 - **a.** Log in to your Dev Hub org.
 - b. From Setup, enter App Manager in the Quick Find box to get to the Lightning Experience App Manager.
 - c. Locate the connected app in the apps list, then click $\ensuremath{\,\overline{}}$, and select View.
 - **d.** In the API (Enable OAuth Settings) section, copy the Consumer Key to your clipboard. The consumer key is a long string of numbers, letters, and characters, such as 3MVG9szVa2Rx_sqBb444p50Yj (example shortened for clarity.)
- 2. Run the force:auth:jwt:grant CLI command.The --clientid and --jwtkeyfile parameter values are the same as when you ran the command to authorize a Dev Hub org. Set --username to the scratch org's admin username and set --instanceurl to https://test.salesforce.com.For example:

```
sfdx force:auth:jwt:grant --clientid 3MVG9szVa2Rx_sqBb444p50Yj \
--jwtkeyfile /Users/jdoe/JWT/server.key --username test-wvkpnfm5z113@example.com \
--instanceurl https://test.salesforce.com
```

If you get an error that the user is not approved, it means that the scratch org information has not yet been replicated to https://test.salesforce.com. Wait a short time and try again.

SEE ALSO:

Authorize an Org Using the JWT-Based Flow Connected Apps Create Scratch Orgs

Create a Private Key and Self-Signed Digital Certificate

The JWT-based authorization flow requires a digital certificate and the private key used to sign the certificate. You upload the digital certificate to the custom connected app that is also required for JWT-based authorization. You can use your own private key and certificate issued by a certification authority. Alternatively, you can use OpenSSL to create a key and a self-signed digital certificate.

This process produces two files.

- server.key—The private key. You specify this file when you authorize an org with the force:auth:jwt:grant command.
- server.crt—The digital certification. You upload this file when you create the connected app required by the JWT-based flow.
- 1. If necessary, install OpenSSL on your computer.

To check whether OpenSSL is installed on your computer, run this command.

which openssl

2. In Terminal or a Windows command prompt, create a directory to store the generated files, and change to the directory.

mkdir /Users/jdoe/JWT

cd /Users/jdoe/JWT

3. Generate a private key, and store it in a file called server.key.

openssl genrsa -des3 -passout pass:SomePassword -out server.pass.key 2048

openssl rsa -passin pass:SomePassword -in server.pass.key -out server.key

You can delete the server.pass.key file because you no longer need it.

4. Generate a certificate signing request using the server.key file. Store the certificate signing request in a file called server.csr. Enter information about your company when prompted.

openssl req -new -key server.key -out server.csr

5. Generate a self-signed digital certificate from the server.key and server.csr files. Store the certificate in a file called server.crt.

openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out server.crt

SEE ALSO: OpenSSL: Cryptography and SSL/TLS Tools Create a Connected App Authorize an Org Using the JWT-Based Flow

Create a Connected App

If you use JWT-based authorization, you must create your own connected app in your Dev Hub org. You can also create a connected app for web-based authorization if you require more security than provided with our connected app. For example, you can create a connected app to set the refresh token timeout or specify IP ranges.

You create a connected app using Setup in your Dev Hub org. These steps assume that you are using Lightning Experience.

JWT-based authorization requires a digital certificate, also called a digital signature. You can use your own certificate or create a self-signed certificate using OpenSSL.



Note: The steps marked *JWT only* are required only if you are creating a connected app for JWT-based authorization. They are optional for web-based authorization.

- 1. Log in to your Dev Hub org.
- 2. From Setup, enter App Manager in the Quick Find box to get to the Lightning Experience App Manager.
- 3. In the top-right corner, click New Connected App.
- 4. Update the basic information as needed, such as the connected app name and your email address.
- 5. Select Enable OAuth Settings.
- 6. For the callback URL, enter http://localhost:1717/OauthRedirect.

If port 1717 (the default) is already in use on your local machine, specify an available one instead. Make sure to also update your sfdx-project.json file by setting the oauthLocalPort property to the new port. For example, if you set the callback URL to http://localhost:1919/OauthRedirect:

"oauthLocalPort" : "1919"

- 7. (JWT only) Select Use digital signatures.
- 8. (JWT only) Click Choose File and upload the server.crt file that contains your digital certificate.
- 9. Add these OAuth scopes:
 - Access and manage your data (api)
 - Perform requests on your behalf at any time (refresh_token, offline_access)
 - Provide access to your data via the Web (web)
- 10. Click Save.

() Important: Make note of the consumer key because you need it later when you run a force:auth command.

- **11.** (JWT only) Click **Manage**.
- 12. (JWT only) Click Edit Policies.
- 13. (JWT only) In the OAuth Policies section, select Admin approved users are pre-authorized for Permitted Users, and click OK.
- 14. (JWT only) Click Save.
- **15.** (JWT only) Click **Manage Profiles** and then click **Manage Permission Sets**. Select the profiles and permission sets that are pre-authorized to use this connected app. Create permission sets if necessary.
- SEE ALSO:

Create a Private Key and Self-Signed Digital Certificate *Trailhead*: Create Your Connected App (Continuous Integration Using Salesforce DX Module) Connected Apps Authorization *Salesforce Help*: Set up Two-Factor Authentication

Use an Existing Access Token Instead of Authorizing

When you authorize into an org using the force:auth commands, the Salesforce CLI takes care of generating and refreshing all tokens, such as the access token. But sometimes you want to run a few CLI commands against an existing org without going through the entire authorization process. In this case, you must provide the access token and instance URL of the org.

1. Use force:config:set to set the instanceUrl config value to the Salesforce instance that hosts the existing org to which you want to connect.

```
sfdx force:config:set instanceUrl=https://na35.salesforce.com
```

2. When you run the CLI command, use the org's access token as the value for the --targetusername parameter rather than the org's username.

```
sfdx force:mdapi:deploy --deploydir <md-dir> --targetusername <access-token>
```

The CLI does not store the access token in its internal files. It uses it only for this CLI command run.

Authorization Information for an Org

You can view information for all orgs that you have authorized and the scratch orgs that you have created.

Use this command to view authentication information about an org.

sfdx force:org:display --targetusername <username>

If you have set a default username, you don't have to specify the --targetusername parameter. To display the usernames for all the active orgs that you've authorized or created, use force:org:list.

If you have set an alias for an org, you can specify it with the --targetusername parameter. This example uses the my-scratch alias.

sfdx force:org:display --targetusername my-scratch-org

```
=== Org Description
KEY VALUE
```

<long-string></long-string>
my-scratch-org
SalesforceDevelopmentExperience
joe@mydevhub.org
2017-06-07T00:51:59.000+0000
jdoe@fabdevhub.org
Developer
2017-06-14
00D9A000008cKm
https://page-power-5849-dev-ed.cs46.my.salesforce.com
Your Company
Active
test-apraqvkwhcml@example.com
To get more information, such as the Salesforce DX authentication URL, include the --verbose parameter. However, --verbose displays the Sfdx Auth Url only if you authenticated to the org using force:auth:web:login and not force:auth:jwt:grant.

```
sfdx force:org:display -u my-scratch-org --verbose
=== Org Description
               VALUE
KEY
Access Token
              <long-string>
Alias
               my-scratch-org
             SalesforceDevelopmentExperience
Client Id
Created By
             joe@mydevhub.org
Created Date
              2017-06-07T00:51:59.000+0000
Dev Hub Id
              jdoe@fabdevhub.org
Edition
               Developer
Expiration Date 2017-06-14
Id
              00D9A000008cKm
Instance Url https://page-power-5849-dev-ed.cs46.my.salesforce.com
Org Name Your Company
Sfdx Auth Url force://SalesforceDevelopmentExperience:xxx@xxx.my.salesforce.com
Status
               Active
               test-apraqvkwhcml@example.com
Username
```

Note: To help prevent security breaches, the force:org:display output doesn't include the org's client secret or refresh token. If you need these values, perform an OAuth flow outside of the Salesforce CLI.

SEE ALSO:

OAuth 2.0 Web Server Authentication Flow Salesforce DX Usernames and Orgs

Log Out of an Org

For security purposes, you can use the Salesforce CLI to log out of any org you've previously authorized. This practice prevents other users from accessing your orgs if you don't want them to.

Important: The only way to access an org after you log out of it is with a password. By default, new scratch orgs contain one administrator with no password. Therefore, to access a scratch org again after you log out of it, set a password for at least one user. Otherwise, you lose all access to the scratch org. If you don't want to access the scratch org again, rather than log out of it, we recommend that you delete it with force:org:delete.

To log out of an org, use force:auth:logout. This example uses the alias my-hub-org to log out.

sfdx force:auth:logout --targetusername my-hub-org

To log out of all your orgs, including scratch orgs, use the --all parameter.

```
sfdx force:auth:logout --all
```

To access an org again, other than a scratch org, reauthorize it.

When you log out of an org, it no longer shows up in the force:org:list output. If you log out of a Dev Hub org, the associated scratch orgs show up only if you specify the --all parameter.

CHAPTER 5 Metadata Coverage

Launch the Metadata Coverage report to determine supported metadata for scratch org source tracking purposes. The Metadata Coverage report is the ultimate source of truth for metadata coverage across several channels. These channels include Metadata API, scratch org source tracking, unlocked packages, second-generation managed packages, classic managed packages, and more.

You are no longer required to log in to an org to see the Metadata Coverage report.

For more information, see Metadata Types in the Metadata API Developer Guide.

Hard-Deleted Components in Unlocked Packages

Components of these metadata types are hard-deleted from the target install org when deleted from an unlocked package.

- ApexClass
- ApexComponent
- ApexPage
- ApexTrigger
- AuraDefinitionBundle
- BrandingSet
- CompactLayout
- CustomPermission
- Dashboard
- Document
- EmailServicesFunction
- EmailTemplate
- EmbeddedServiceBranding
- EmbeddedServiceConfig
- EmbeddedServiceLiveAgent
- ExternalServiceRegistration
- FeatureParameterBoolean
- FeatureParameterDate
- FeatureParameterInteger
- FlexiPage
- HomePageLayout
- InstalledPackage

Metadata Coverage

- IntegrationHubSettings
- IntegrationHubSettingsType
- Layout
- LicenseDefinition
- LightningComponentBundle
- LightningExperienceTheme
- ListView
- LiveChatAgentConfig
- LiveChatButton
- LiveChatSensitiveDataRule
- NamedCredential
- NetworkBranding
- MatchingRule
- PermissionSet
- Profile
- PermissionSetGroup
- PermissionSetLicense
- QuickAction
- RemoteSiteSetting
- Report
- StaticResource
- UserLicense
- WaveApplication
- WaveDashboard
- WaveDataflow
- WaveDataset
- WaveLens
- WaveRecipe
- WaveTemplateBundle
- WaveXmd
- WorkflowFlowAction
- WorkflowRule
- WorkflowTask(Task)
- SEE ALSO:

Components Available in Managed Packages (ISVForce Guide) Metadata Types (Metadata API Developer Guide)

CHAPTER 6 Scratch Orgs

In this chapter ...

- Scratch Org Definition File
- Scratch Org Definition Configuration Values
- Create Scratch Orgs
- Select the Salesforce Release for a Scratch Org
- Salesforce DX Project
 Structure and Source
 Format
- Push Source to the Scratch Org
- Assign a Permission Set
- Ways to Add Data to Your Scratch Org
- Pull Source from the Scratch Org to Your Project
- Track Changes Between the Project and Scratch Org
- Scratch Org Users
- Manage Scratch Orgs from Dev Hub

The scratch org is a source-driven and disposable deployment of Salesforce code and metadata. A scratch org is fully configurable, allowing developers to emulate different Salesforce editions with different features and preferences. You can share the scratch org configuration file with other team members, so you all have the same basic org in which to do your development.

Scratch orgs drive developer productivity and collaboration during the development process, and facilitate automated testing and continuous integration. You can use the CLI or IDE to open your scratch org in a browser without logging in. You might spin up a new scratch org when you want to:

- Start a new project.
- Start a new feature branch.
- Test a new feature.
- Start automated testing.
- Perform development tasks directly in an org.
- Start from "scratch" with a fresh new org.
- Note: Partners can create partner edition scratch orgs: Partner Developer, Partner Enterprise, Partner Group, and Partner Professional. This feature is available only if creating scratch orgs from a Dev Hub in a partner business org. See Supported Scratch Org Editions for Partners in the *ISVforce Guide* for details.

Scratch Orgs Created in Government Cloud or Public Cloud

The Dev Hub org instance determines where scratch orgs are created.

- Scratch orgs created from a Dev Hub org in Government Cloud are created in a Government Cloud instance.
- Scratch orgs created from a Dev Hub org in Public Cloud are created on a Public Cloud instance.

Scratch Org Allocations and Considerations

To ensure optimal performance, your Dev Hub org edition determines your scratch org allocations. These allocations determine how many scratch orgs you can create daily, and how many can be active at a

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Dev Hub available in: Developer, Enterprise, Performance, and Unlimited Editions

Scratch orgs available in: **Developer**, **Enterprise**, **Group**, and **Professional** Editions given point. By default, Salesforce deletes scratch orgs and their associated ActiveScratchOrg records from your Dev Hub org when a scratch org expires. A scratch org expires in 7 days unless you set a duration when you create it.

Scratch orgs have these storage limits:

- 200 MB for data
- 50 MB for files

To try out scratch orgs, sign up for a Developer Edition org on Salesforce Developers, then enable Dev Hub.

Edition	Active Scratch Org Allocation	Daily Scratch Org Allocation
Developer Edition or trial	3	6
Enterprise Edition	40	80
Unlimited Edition	100	200
Performance Edition	100	200

Note: If you are a partner or ISV, your scratch org allocations might be different. See the ISV force Guide for details.

List Active and Daily Scratch Orgs

To view how many scratch orgs you have allocated, and how many you have remaining:

<pre>sfdx force:limits:api:display -u <dev alias="" hub="" or="" username=""></dev></pre>				
NAME	REMAINING	MAXIMUM		
ActiveScratchOrgs	25	40		
ConcurrentAsyncGetReportInstances	200	200		
ConcurrentSyncReportRuns	20	20		
DailyApiRequests	14994	15000		
DailyAsyncApexExecutions	250000	250000		
DailyBulkApiRequests	10000	10000		
DailyDurableGenericStreamingApiEvents	10000	10000		
DailyDurableStreamingApiEvents	10000	10000		
DailyGenericStreamingApiEvents	10000	10000		
DailyScratchOrgs	80	80		
DailyStreamingApiEvents	10000	10000		
DailyWorkflowEmails	75	75		
DataStorageMB	1073	1073		
DurableStreamingApiConcurrentClients	20	20		
FileStorageMB	1073	1073		
HourlyAsyncReportRuns	1200	1200		
HourlyDashboardRefreshes	200	200		
HourlyDashboardResults	5000	5000		

HourlyDashboardStatuses	999999999	999999999
HourlyODataCallout	10000	10000
HourlySyncReportRuns	500	500
HourlyTimeBasedWorkflow	50	50
MassEmail	10	10
PermissionSets	1489	1500
SingleEmail	15	15
StreamingApiConcurrentClients	20	20

Scratch Org Definition File

The scratch org definition file is a blueprint for a scratch org. It mimics the shape of an org that you use in the development life cycle, such as sandbox, packaging, or production.

The settings and configuration options associated with a scratch org determine its shape, including:

- Edition—The Salesforce edition of the scratch org, such as Developer, Enterprise, Group, or Professional.
- Add-on features—Functionality that is not included by default in an edition, such as multi-currency.
- Settings—Org and feature settings used to configure Salesforce products, such as Chatter and Communities.

By default, scratch orgs are empty. They don't contain much of the sample metadata that you get when you sign up for an org, such as a Developer Edition org, the traditional way. Some of the things not included in a scratch org are:

- Custom objects, fields, indexes, tabs, and entity definitions
- Sample data
- Sample Chatter feeds
- Dashboards and reports
- Workflows
- Picklists
- Profiles and permission sets
- Apex classes, triggers, and pages

Setting up different scratch org definition files allows you to easily create scratch orgs with different shapes for testing. For example, you can turn Chatter on or off in a scratch org by setting the ChatterEnabled org preference in the definition file. If you want a scratch org with sample data and metadata like you're used to, add this option: hasSampleData.

We recommend that you keep this file in your project and check it in to your version control system. For example, create a team version that you check in for all team members to use. Individual developers could also create their own local version that includes the scratch org definition parameters. Examples of these parameters include email and last name, which identify who is creating the scratch org.

Scratch Org Definition File Name

You indicate the path to the scratch org configuration file when you create a scratch org with the force:org:create CLI command.

- If you're using Salesforce CLI on the command line, you can name this file whatever you like and locate it anywhere the CLI can access.
- If you're using Salesforce Extensions for VS Code, make sure that the scratch org definition file is located in the config folder and its name ends in scratch-def.json.

If you're using a sample repo or creating a Salesforce DX project, the sample scratch org definition files are located in the config directory. You can create different configuration files for different org shapes or testing scenarios. For easy identification, name the file something descriptive, such as devEdition-scratch-def.json or packaging-org-scratch-def.json.

Scratch Org Definition File Options

Here are the options you can specify in the scratch org definition file:

Name	Required?	Default If Not Specified
orgName	No	Company

Name	Required?	Default If Not Specified
country	No	Dev Hub's country. If you want to override this value, enter the two-character, upper-case ISO-3166 country code (Alpha-2 code). You can find a full list of these codes at several sites, such as: https://www.iso.org/obp/ui/#search. This value sets the locale of the scratch org.
username	No	test-unique_identifier@example.com
adminEmail	No	Email address of the Dev Hub user making the scratch org creation request
edition	Yes	None. Valid entries are Developer, Enterprise, Group, or Professional
description	No	None. 2000-character free-form text field. The description is a good way to document the scratch org's purpose. You can view or edit the description in the Dev Hub. From App Launcher, select Scratch Org Info or Active Scratch Orgs , then click the scratch org number.
hasSampleData	No	Valid values are true and false. False is the default, which creates an org without sample data.
language	No	Default language for the country. To override the language set by the Dev Hub locale, see Supported Languages for the codes to use in this field.
features	No	None
orgPreferences (to be deprecated soon)	No	None
release	No	Same Salesforce release as the Dev Hub org. Options are preview or previous. Can use only during Salesforce release transition periods.
settings	No	None
<custom api="" field="" name=""></custom>	No	None. Useful for Dev Ops use cases where you want to track extra information on the ScratchOrgInfo object. First, create the custom field, then reference it in the scratch org definition by its API name.

Sample Scratch Org Definition File

Here's what the scratch org definition JSON file looks like. For more information on features and settings, see Scratch Org Definition Configuration Values.

```
{
   "orgName": "Acme",
   "edition": "Enterprise",
   "features": ["Communities", "ServiceCloud", "Chatbot"],
   "settings": {
        "orgPreferenceSettings": {
        "orgPreferenceSettings": {
        " orgPreferenceSettings": {
```

```
"networksEnabled": true,
    "slDesktopEnabled": true,
    "slEncryptedStoragePref2": false
},
"omniChannelSettings": {
    "enableOmniChannel": true
},
"caseSettings": {
    "systemUserEmail": "support@acme.com"
}
}
```

Some features, such as Communities, can require a combination of a feature and a setting to work correctly for scratch orgs. This code snippet sets both the feature and associated setting.

```
"features": ["Communities"],
    "settings": {
        ""networksEnabled": true
        ...
```

Create a Custom Field for ScratchOrgInfo

You can add more options to the scratch org definition to manage your Dev Ops process. To do so, create a custom field on the ScratchOrgInfo object. (ScratchOrgInfo tracks scratch org creation and deletion.)

Important: If you're making these changes directly in your production org, proceed with the appropriate levels of caution. The ScratchOrgInfo object is not available in sandboxes or scratch orgs.

- 1. In the Dev Hub org, create the custom field.
 - a. From Setup, enter Object Manager in the Quick Find box, then select Object Manager.
 - b. Click Scratch Org Info.
 - c. In Fields & Relationships, click New.
 - d. Define the custom field, then click Save.
- 2. After you create the custom field, you can pass it a value in the scratch org definition file by referencing it with its API name.

Let's say you create two custom fields called workitem and release. Add the custom fields and associated values to the scratch org definition:

```
{
    "orgName": "MyCompany",
    "edition": "Developer",
    "workitem_c": "W-12345678",
    "release_c": "June 2018 pilot",
    "settings": {
        "orgPreferenceSettings": {
            "slDesktopEnabled": true
        }
}
```

3. Create the scratch org.

Scratch Org Definition Configuration Values

The scratch org definition file contains the configuration values that determine the shape of the scratch org.

Supported Editions

The Salesforce edition of the scratch org. Possible values are:

- Developer
- Enterprise
- Group
- Professional

Supported Features

You can enable these add-on features in a scratch org. Features aren't case-sensitive. You can indicate them as all-caps, or how we define them here (for readability purposes). If a feature is followed by <value>, it requires that you specify a value as an incremental allocation or limit (see next section).

Note: Some scratch org features require additional configuration.

- For Group and Professional Edition orgs, the AuthorApex feature is disabled by default. Enabling the AuthorApex feature lets you edit and test your Apex classes.
- To use the Chatbot feature, turn on Enable Einstein Features in the Dev Hub org to accept the Terms of Service.
- Enterprise Territory Management (Territory2) is a scratch org setting. See Metadata Coverage report for implementation details.
- DeferSharingCalc requires additional configuration using the Setup menu in the scratch org. See Defer Sharing Rule Calculations in Salesforce Help.
- ActionPlans
- AddCustomApps:<value>
- AddCustomObjects:<quantity> (Min 1: Max 30)
- AddCustomRelationships:<quantity> (Min 1: Max 30: Multiplier 5)
- AddCustomTabs:<value>
- AddDatacomCrmRecordCredit:<quantity> (Min 1: Max 30)
- AddInsightsQueryLimit:<quantity> (Min 1: Max 30: Multiplier 10)
- AddHistoryFieldsPerEntity:<value>
- AnalyticsAdminPerms
- API
- AuthorApex (see Note)
- CascadeDelete
- Chatbot (see Note)
- ChatterAnswers
- ChatterAnswersUser

Scratch Orgs

- Communities
- ContactsToMultipleAccounts
- ContractApprovals
- CPQ
- CustomerSelfService
- CustomNotificationType
- DatacomDnbAccounts
- DatacomFullClean
- DebugApex
- DefaultWorkflowUser
- DeferSharingCalc (see Note)
- DevelopmentWave
- EinsteinAnalyticsPlus
- EinsteinAssistant
- EinsteinBuilderFree
- Entitlements
- EventLogFile
- ExternalldentityLogin
- ExternalSharing (not available in Group Edition)
- FieldService
- FlowSites
- ForceComPlatform
- HealthCloudUser
- IndustriesManufacturing
- InsightsPlatform
- Interaction
- loT
- JigsawUser
- Knowledge
- LightningSalesConsole
- LightningServiceConsole
- LiveAgent
- LiveMessage
- MarketingUser
- MaxApexCodeSize:<value>
- MaxCustomLabels:<value>
- MobileUser
- MultiCurrency
- OfflineUser
- Pardot

Scratch Orgs

- PersonAccounts
- PlatformCache
- PlatformEncryption
- ProcessBuilder
- ProductsAndSchedules
- RecordTypes
- RetainFieldHistory
- SalesUser
- ServiceCloud
- ServiceUser
- SiteDotCom
- SiteForceContributor
- Sites
- StateAndCountryPicklist
- SurveyCreatorUser
- TerritoryManagement (see Note)
- TimeSheetTemplateSettings
- UiPlugin
- WavePlatform
- Workflow

You can specify multiple feature values in a comma-delimited list in the scratch org definition file.

```
"features": ["MultiCurrency", "AuthorApex"],
```

Scratch Org Feature Allocations and Limits

For some features, you need to specify a quantity you want to provision.

Feature Name	Additional Allocation or Limit	Maximum	Notes
AddCustomApps	Allocation	30	Replaces CustomApps
AddCustomObjects	Allocation	30	
AddCustomRelationships	Allocation	30	Multiplier is 5. Setting the quantity to 5 increases the limit by 25.
AddCustomTabs	Allocation	30	Replaces CustomTabs
AddDatacomCrmRecords	Allocation	30	
AddInsightsQueryLimit	Allocation	30	Multiplier is 10. Setting the quantity to 6 increases the query limit by 60.

Feature Name	Additional Allocation or Limit	Maximum	Notes
AddHistoryFieldsPerEntity	Allocation	25	
MaxApexCodeSize	Limit	To use a value greater than the default value of 10, contact Salesforce Customer Support.	Measured in millions. Setting this limit to 10 is equal to 10 million characters of code.
MaxCustomLabels	Limit	15	Measured in thousands. Setting the limit to 10 enables the scratch org to have 10,000 custom labels.

Example scratch org definition file:

```
{
   "orgName": "Acme",
   "edition": "Enterprise",
   "features": ["AddCustomApps:25", "MaxCustomLabels:10"]
}
```

Scratch Org Settings

In Winter '19 and later, scratch org settings are the format for defining org preferences in the scratch org definition. Because you can use all Metadata API settings, they are the most comprehensive way to configure a scratch org. If a setting is supported in Metadata API, it's supported in scratch orgs. Settings provide you with fine-grained control because you can define values for all fields for a setting, rather than just enabling or disabling it.

Important: In Winter 19, you can specify scratch org settings or org preferences in your scratch org definition file, but not both. We encourage you to convert org preferences to scratch org settings in your scratch org definition. Scratch org settings provide more settings that aren't currently available as org preferences. We plan to deprecate support for org preferences in Winter '20.

For information on Metadata API settings and their supported fields, see Settings in Metadata API Developer Guide.

() Important: Although the Settings are upper camel case in the Metadata API Developer Guide, be sure to indicate them as lower camel case in the scratch org definition.

When converting existing org prefs to settings, the org prefs that start with "Is" have a corresponding setting that starts with "enable". For example, "IsOrdersEnabled" is "enableOrders" that takes a Boolean value of true or false.

```
{
   "orgName": "Acme",
   "edition": "Enterprise",
   "features": ["Communities", "ServiceCloud", "Chatbot"],
   "settings": {
        "orgPreferenceSettings": {
            "networksEnabled": true,
            "slDesktopEnabled": true,
            "slEncryptedStoragePref2": false
        },
        "omniChannelSettings": {
            "enableOmniChannel": true
        }
    }
}
```

```
},
"caseSettings": {
    "systemUserEmail": "support@acme.com"
}
}
```

Here's an example of how to configure SecuritySettings in your scratch org. In this case, to define session timeout, you nest the field values.

```
{
   "orgName": "Acme",
   "edition": "Enterprise",
   "features": [],
   "settings": {
        "orgPreferenceSettings": {
            "slEncryptedStoragePref2": false
        },
        "securitySettings": {
              "sessionSettings": {
                 "sessionSettings": {
                "sessionTimeout": "TwelveHours"
        }
    }
}
```

Here's an example of how to configure the IoT feature in your scratch org. It requires a combination of indicating the IoT feature and IoT scratch org settings.

```
{
   "orgName": "Acme",
   "edition": "Enterprise",
   "features": [],
   "settings": {
        "orgPreferenceSettings": {
            "slEncryptedStoragePref2": false
        },
        "iotSettings": {
            "enableIoT": true, {
            "iotInsights": true
        }
    }
}
```

Metadata Coverage Report

The Metadata Coverage report is now available from the Salesforce Developer site rather than from your org. You can select a release version from within the coverage report to view different release versions. We provide coverage information for Summer 18/v43 onward.

The sample scratch definition is available starting in Winter 19/v44. In this example, to work with the Network metadata type in a Developer Edition scratch org, your scratch org definition must include the Communities feature and the networksEnabled setting.



Supported Org Preferences

Before Winter '19, you indicate scratch org settings using org preferences in the scratch org definition file. Use the scratch org settings format if creating a new scratch org definition file. We plan to deprecate org preferences in Winter '20.

It's now easier than ever to move to scratch org settings. If your scratch org definition file still contains org preferences, you see a warning message that provides the format for settings. Just copy and paste, and you're good to go!

```
WARNING: We're deprecating orgPreferences in Summer '19. You can continue to use them
until they are replaced by settings in Winter '20. But why wait? Here's exactly what you
need
to update in the scratch org definition file.
Replace the orgPreferences section:
{
    "orgPreferences" : {
        "enabled": [
          "S1DesktopEnabled"
        ],
        "disabled": [
          "IsActivitiesRemindersEnabled"
        ]
    }
}
With settings:
```

```
{
    "settings": {
        "orgPreferenceSettings": {
            "slDesktopEnabled": true
        },
        "activitySettings": {
               "enableActivitiesReminder": false
        }
    }
}
```

Important: As of Winter 19, you can specify scratch org settings or org preferences in your scratch org definition file, but not both. We encourage you to convert org preferences to scratch org settings in your scratch org definition. Scratch org settings provide more settings that aren't currently available as org preferences.

Org preferences are settings that a user can configure in the org. For example, preferences control which Chatter, Knowledge, and Opportunities settings are enabled, among many others. These settings are enabled (or disabled) in the orgPreferences section of the configuration file, in JSON format.

```
"orgPreferences": {
    "enabled": ["S1DesktopEnabled", "ChatterEnabled", "IsOpportunityTeamEnabled"],
    "disabled": ["IsOrdersEnabled"]
}
```

Warning: Exercise caution when using DisableParallelApexTesting. Your tests could run noticeably slower. Try these Testing Best Practices in the Apex Developer Guide so that you can run your Apex tests in parallel.

You can set the following org preferences in the configuration file. You indicate org preferences using upper camel case, as indicated in the following lists.

Important: To prepare for the deprecation of org preferences, we strongly suggest you switch to settings. The values for settings are similar yet different from the values in the following lists. Another difference is that you must indicate settings using lower camel case. For information on settings and their supported fields, see Settings in *Metadata API Developer Guide*.

General Settings

- AnalyticsSharingEnable
- AsyncSaveEnabled
- ChatterEnabled
- DisableParallelApexTesting
- EnhancedEmailEnabled
- EventLogWaveIntegEnabled
- LoginForensicsEnabled
- NetworksEnabled
- NotesReservedPref01
- OfflineDraftsEnabled
- PathAssistantsEnabled
- S1DesktopEnabled
- S1EncryptedStoragePref2
- S1OfflinePref
- SelfSetPasswordInApi

Scratch Orgs

- SendThroughGmailPref
- SocialProfilesEnable
- Translation
- VoiceEnabled

Account Settings

- IsAccountTeamsEnabled
- ShowViewHierarchyLink

Activities Settings

- IsActivityRemindersEnabled
- IsDragAndDropSchedulingEnabled
- IsEmailTrackingEnabled
- IsGroupTasksEnabled
- IsMultidayEventsEnabled
- IsRecurringEventsEnabled
- IsRecurringTasksEnabled
- IsSidebarCalendarShortcutEnabled
- IsSimpleTaskCreateUIEnabled
- ShowEventDetailsMultiUserCalendar
- ShowHomePageHoverLinksForEvents
- ShowMyTasksHoverLinks

Contract Settings

- AutoCalculateEndDate
- IsContractHistoryTrackingEnabled
- NotifyOwnersOnContractExpiration

Entitlement Settings

- AssetLookupLimitedToActiveEntitlementsOnAccount
- AssetLookupLimitedToActiveEntitlementsOnContact
- AssetLookupLimitedToSameAccount
- AssetLookupLimitedToSameContact
- IsEntitlementsEnabled
- EntitlementLookupLimitedToActiveStatus
- EntitlementLookupLimitedToSameAccount
- EntitlementLookupLimitedToSameAsset
- EntitlementLookupLimitedToSameContact

Forecasting Settings

IsForecastsEnabled

Ideas Settings

- IsChatterProfileEnabled
- IsIdeaThemesEnabled

- IsIdeasEnabled
- IsIdeasReputationEnabled

Knowledge Settings

- IsCreateEditOnArticlesTabEnabled
- IsExternalMediaContentEnabled
- IsKnowledgeEnabled
- ShowArticleSummariesCustomerPortal
- ShowArticleSummariesInternalApp
- ShowArticleSummariesPartnerPortal
- ShowValidationStatusField

Chat Settings

IsLiveAgentEnabled

Marketing Action Settings

IsMarketingActionEnabled

Name Settings

- IsMiddleNameEnabled
- IsNameSuffixEnabled

Opportunity Settings

• IsOpportunityTeamEnabled

Order Settings

- IsNegativeQuantityEnabled
- IsOrdersEnabled
- IsReductionOrdersEnabled

Personal Journey Settings

• IsExactTargetForSalesforceAppsEnabled

Product Settings

- IsCascadeActivateToRelatedPricesEnabled
- IsQuantityScheduleEnabled
- IsRevenueScheduleEnabled

Quote Settings

IsQuoteEnabled

Search Settings

- DocumentContentSearchEnabled
- OptimizeSearchForCjkEnabled
- RecentlyViewedUsersForBlankLookupEnabled
- SidebarAutoCompleteEnabled
- SidebarDropDownListEnabled
- SidebarLimitToltemslownCheckboxEnabled

- SingleSearchResultShortcutEnabled
- SpellCorrectKnowledgeSearchEnabled

SEE ALSO:

Salesforce Editions Settings (Metadata API Developer Guide)

Create Scratch Orgs

After you create the scratch org definition file, you can easily spin up a scratch org and open it directly from the command line.

Before you create a scratch org:

- Set up your Salesforce DX project
- Authorize the Dev Hub org
- Create the scratch org definition file

You can create scratch orgs for different functions, such as for feature development, for development of packages that contain a namespace, or for user acceptance testing.

Tip: Delete any unneeded or malfunctioning scratch orgs in the Dev Hub org or via the command line so that they don't count against your active scratch org allocations.

1. Create the scratch org.

То	Run This Command
Create a scratch org for development using a scratch org definition file	The scratch org definition defines the org edition, features, org preferences, and some other options.
	<pre>sfdx force:org:create -f project-scratch-def.json</pre>
Specify scratch org definition values on the command line using key=value pairs	<pre>sfdx force:org:create adminEmail=me@email.com edition=Developer \ username=admin_user@orgname.org</pre>
Create a scratch org with an alias	Scratch org usernames are long and unintuitive. Setting an alias each time you create a scratch org is a great way to track the scratch org's function. And it's much easier to remember when issuing subsequent CLI commands.
	sfdx force:org:create -f project-scratch-def.json -a MyScratchOrg
Create a scratch org for user acceptance testing or to test installations of packages	In this case, you don't want to create a scratch org with a namespace. You can use this command to override the namespace value in the scratch org definition file.
	<pre>sfdx force:org:create -f project-scratch-def.jsonnonamespace</pre>

То	Run This Command
Indicate that this scratch org is the default	CLI commands that are run from within the project use the default scratch org, and you don't have to manually enter the username parameter each time.
	sfdx force:org:create -f project-scratch-def.json setdefaultusername
Specify the scratch org's duration, which indicates when the scratch org expires (in days)	The default is 7 days. Valid values are from 1-30.
	<pre>sfdx force:org:create -f config/project-scratch-def.jsondurationdays 30</pre>
	<pre>sfdx force:org:create -f config/project-scratch-def.json -d 3</pre>
Specify the Salesforce release for the scratch org	During the Salesforce release transition, you can specify the release (preview or previous) when creating a scratch org. See Select the Salesforce Release for a Scratch Org.

Indicate the path to the scratch definition file relative to your current directory. For sample repos, this file is located in the config directory.

Stdout displays two important pieces of information: the org ID and the username.

```
Successfully created scratch org: 00D3D000000PE5UAM,
    username: test-b4agup43oxmu@example.com
```

If the create command times out before the scratch org is created (the default wait time is 6 minutes), you see an error. Issue this command to see if it returns the scratch org ID, which confirms the existence of the scratch org:

```
sfdx force:data:soql:query -q "SELECT ID, Name, Status FROM ScratchOrgInfo \
    WHERE CreatedBy.Name = '<your name>' \
    AND CreatedDate = TODAY" -u <Dev Hub org>
```

This example assumes that your name is Jane Doe, and you created an alias for your Dev Hub org called DevHub:

```
sfdx force:data:soql:query -q "SELECT ID, Name, Status FROM ScratchOrgInfo \
    WHERE CreatedBy.Name = 'Jane Doe' AND CreatedDate = TODAY" -u DevHub
```

If that doesn't work, create another scratch org and increase the timeout value using the --wait parameter. Don't forget to delete the malfunctioning scratch org.

2. Open the org.

sfdx force:org:open -u <username/alias>

If you want to open the scratch org in Lightning Experience or open a Visual force page, use the --path parameter.

sfdx force:org:open --path lightning

- 3. Push local project source to your scratch org.
- SEE ALSO:

Project Setup Authorization Scratch Org Definition File Push Source to the Scratch Org

Select the Salesforce Release for a Scratch Org

During the Salesforce release transition, you can specify the release (preview or previous) when creating a scratch org.

What is Salesforce Preview?

During every major Salesforce release, you have the opportunity to get early access to the release in your scratch orgs and sandboxes to test new customizations and features before your production org is upgraded. This is called the Salesforce Preview, and scratch orgs created on the upcoming release are called preview scratch orgs.

Normally, you create scratch orgs that are the same version as the Dev Hub. However, during the major Salesforce release transition that happens three times a year, you can select the Salesforce release version, Preview or Previous, based on the version of your Dev Hub.

To try out new features in an upcoming release, you no longer have to create a trial Dev Hub on the upcoming version to create preview scratch orgs. You can use your existing Dev Hub that includes your existing scratch org active and daily limits.

For example, you can select a version over the next three releases during these release transition dates. Preview start date is when sandbox instances are upgraded. Preview end date is when all instances are on the GA release.

Release Version	Preview Start Date	Preview End Date
Winter '20	September 6, 2019	October 12, 2019
Spring '20	January 3, 2020	February 15, 2020
Summer '20	May 8, 2020	June 13, 2020

Because previous and preview are all relative terms, your Dev Hub org version during the release transition determines their relative significance. Here's what happens when you try to create a scratch org with one of the release values.

Dev Hub Version	Preview	Previous
Dev Hub has upgraded to the latest version	Error (Dev Hub is already on the latest version)	Prior Dev Hub version
Dev Hub is still on the GA version	Version following the Dev Hub version (newly released Salesforce version)	Error (Dev Hub is on the GA version; previous version unavailable)

Note: If you don't specify a release value, the scratch org version is the same version as the Dev Hub org.

Create a Scratch Org for a Specific Release

You can specify the release version in the scratch org definition file or directly on the command line. Any option you issue on the command line overrides what you have defined in your scratch definition file.

- 1. Find out which instance your Dev Hub org is on: https://status.salesforce.com.
- 2. Add the release option to your scratch org definition file.

```
{
   "orgName": "Dreamhouse",
   "edition": "Developer",
   "release": "Preview",
   "settings": {
        "orgPreferenceSettings": {
            "slDesktopEnabled": true,
            "selfSetPasswordInApi": true,
            "slEncryptedStoragePref2": false
        }
    }
}
```

Alternatively, you can specify the release value directly on the command line. Any values you specify on the command line override the values in the scratch org definition.

3. Create the scratch org.

In this example, we're creating a scratch org on the preview release.

```
sfdx force:org:create -f config/project-scratch-def.json -a PreviewOrg -v DevHub
release=Preview
```

If you're creating a previous scratch org and you have upgraded the CLI to the pre-release version, be sure to set the apiVersion to match the scratch org version.

To set it globally for all DX projects:

```
sfdx force:config:set apiVersion=46.0 --global
```

To set it on the command line:

```
SFDX_API_VERSION=46.0 sfdx force:org:create -f config/project-scratch-def.json -a PreviewOrg
    -v DevHub release=Preview
```

What If I Want to Create a Pre-Release Scratch Org?

Pre-release is a very early build of the latest version of Salesforce that's available before Salesforce Preview. It's not built to handle scale and doesn't come with any Salesforce Support service-level agreements (SLAs). For this reason, the only way to create a pre-release scratch org is to sign up for a pre-release trial Dev Hub org (subject to availability).

Salesforce DX Project Structure and Source Format

A Salesforce DX project has a specific project structure and source format. Salesforce DX source uses a different set of files and file extensions from what you're accustomed when using Metadata API.

Source Transformation

It's not uncommon for metadata formatted source to be very large, making it difficult to find what you want. If you work on a team with other developers who update the same metadata at the same time, you have to deal with merging multiple updates to the file. If you're thinking that there has to be a better way, you're right.

Before, all custom objects and object translations were stored in one large metadata file.



We solve this problem by providing a new source shape that breaks down these large source files to make them more digestible and easier to manage with a version control system. It's called source format.

A Salesforce DX project stores custom objects and custom object translations in intuitive subdirectories. Source format makes it much easier to find what you want to change or update. And you can say goodbye to messy merges.



Static Resources

Static resources must reside in the /main/default/staticresources directory. The force:source:push and force:source:pull commands support auto-expanding or compressing archive MIME types within your project. These behaviors

support both the .zip and .jar MIME types. This way, the source files are more easily integrated in your Salesforce DX project and version control system.

If, for example, you upload a static resource archive through the scratch org's Setup UI, force:source:pull expands it into its directory structure within the project. To mimic this process from the file system, add the directory structure to compress directly into the static resources directory root, then create the associated .resource-meta.xml file. If an archive exists as a single file in your project, it's always treated as a single file and not expanded.

This example illustrates how different types of static resources are stored in your local project. You can see an expanded .zip archive called expandedzippedresource and its related .resource-meta.xml file. You also see a couple .jpg files being stored with their MIME type, and a single file being stored with the legacy .resource extension

4 forme-app	
⁴ default	
▶ aura	
▶ classes	
▶ layouts	
▶ objects	
▶ permissionsets	
▶ profiles	
▲ staticresources	
expandedzippedresources	
▶ css	
▶ images	
expandedzippedresources.resource-meta.xml	
📼 icon1.jpg	
icon1.resource-meta.xml	
≣ leaflet.resource	
leaflet.resource-meta.xml	
📼 logo.jpg	
logo.resource-meta.xml	
# main_page.css	
main_page.resource-meta.xml	

File Extensions

When you convert existing metadata format to source format, we create an XML file for each bit. All files that contain XML markup now have an .xml extension. You can then look at your source files using an XML editor. To sync your local projects and scratch orgs, Salesforce DX projects use a particular directory structure for custom objects, custom object translations, Lightning web components, Aura components, and documents.

For example, if you had an object called Case.object, source format provides an XML version called Case.object-meta.xml. If you have an app call DreamHouse.app, we create a file called DreamHouse.app-meta.xml. You get the idea. For Salesforce DX projects, all source format files have a companion file with the "-meta.xml" extension.

Traditionally, static resources are stored on the file system as binary objects with a .resource extension. Source format handles static resources differently by supporting content MIME types. For example, .gif files are stored as a .gif instead of .resource. By storing files with their MIME extensions, you can manage and edit your files using the associated editor on your system.

You can have a combination of existing static resources with their .resource extension, and newly created static resources with their MIME content extensions. Existing static resources with .resource extensions keep that extension, but any new static resources show up in your project with their MIME type extensions. We allow .resource files to support the transition for existing customers. Although you get this additional flexibility, we recommend storing your files with their MIME extensions.

Custom Objects

When you convert from metadata format to source format, your custom objects are placed in the <package</p>

directory/main/default/objects directory. Each object has its own subdirectory that reflects the type of custom object. Some parts of the custom objects are extracted into in these subdirectories:

- businessProcesses
- compactLayouts
- fields
- fieldSets
- listViews
- recordTypes
- sharingReasons
- validationRules
- webLinks

The parts of the custom object that are not extracted are placed in a file.

- For objects, <object>.object-meta.xml
- For fields, <field_name>.field-meta.xml

Custom Object Translations

Custom object translations reside in the *<package directory>/main/default/objectTranslations* directory, each in their own subdirectory named after the custom object translation. Custom object translations and field translations are extracted into their own files within the custom object translation's directory.

- For field names, <field_name>.fieldTranslation-meta.xml
- For object names, <object_name>.objectTranslation-meta.xml

The remaining pieces of the custom object translation are placed in a file called <*objectTranslation*>.objectTranslation-meta.xml.

Aura Components

Aura bundles and components must reside in a directory named aura under the *<package directory>* directory.



Lightning Web Components

Lightning web components must reside in a directory named lwc under the *<package directory*>directory.



Documents

Documents must be inside the directories of their parent document folder. The parent document folder must be in a directory called documents. Each document has a corresponding metadata XML file that you can view with an XML editor.

✓ main ✓ default
⊿ default
▲ aura
✓ documents
▲ MyImages
app_development.document-meta.xml
🖙 app_development.jpg
continuous_integration.document-meta.xml
📼 continuous_integration.jpg
✓ MyPDFs
ISVforce_Guide.document-meta.xml
↓ ISVforce_Guide.pdf
Salesforce_DX_Developer_Guide.document-meta.xml
Salesforce_DX_Developer_Guide.pdf
MySpreadsheets
Sales_by_State.document-meta.xml
💷 Sales_by_State.xlsx
Total_Sales.document-meta.xml
🕼 Total_Sales.xlsx
Mylmages.documentFolder-meta.xml
MyPDFs.documentFolder-meta.xml
MySpreadsheets.documentFolder-meta.xml
(i) README.md
{} sfdx-project.json

Push Source to the Scratch Org

After changing the source, you can sync the changes to your scratch org by pushing the changed source to it.

The first time you push metadata to the org, all source in the folders you indicated as package directories is pushed to the scratch org to complete the initial setup. At this point, we start change-tracking locally on the file system and remotely in the scratch org to determine which metadata has changed. Let's say you pushed an Apex class to a scratch org and then decide to modify the class in the scratch org instead of your local file system. The CLI tracks in which local package directory the class was created, so when you pull it back to your project, it knows where it belongs.

Warning: You can use force:source:push for scratch orgs only. If you're synchronizing source to another org, use the Metadata API.

During development, you change files locally in your file system and change the scratch org directly using the builders and editors that Salesforce supplies. Usually, these changes don't cause a conflict and involve unique files.

The push command doesn't handle merges. Projects and scratch orgs are meant to be used by one developer. Therefore, we don't anticipate file conflicts or the need to merge. However, if the push command detects a conflict, it terminates the operation and displays the conflict information to the terminal. You can rerun the push command and force the changes in your project to the scratch org.

Before running the push command, you can get a list of what's new, changed, and the conflicts between your local file system and the scratch org by using force:status. This way you can choose ahead of time which version you want to keep and manually address the conflict.

Pushing Source to a Scratch Org

To push changed source to your default scratch org:

sfdx force:source:push					
STATE	FULL NAME	TYPE	PROJECT PATH		
Changed Changed	MyWidgetClass MyWidgetClass	ApexClass ApexClass	/classes/MyWidgetClass.cls-meta.xml /classes/MyWidgetClass.cls		

To push changed source to a scratch org that's not the default, you can indicate it by its username or alias:

sfdx force:source:push --targetusername test-b4agup43oxmu@example.com

sfdx force:source:push -u test-b4agup43oxmu@example.com

sfdx force:source:push -u MyGroovyScratchOrg

Tip: You can create an alias for an orguing force:alias:set. Run force:org:list to display the usernames of all the scratch orgs you have created.

Selecting Files to Ignore During Push

It's likely that you have some files that you don't want to sync between the project and scratch org. You can have the push command ignore the files you indicate in .forceignore.

If Push Detects Warnings

If you run force:source:push, and warnings occur, the CLI doesn't push the source. Warnings can occur, for example, if your project source is using an outdated version. If you want to ignore these warnings and push the source to the scratch org, run:

sfdx force:source:push --ignorewarnings

Tip: Although you can successfully push using this option, we recommend addressing the issues in the source files. For example, if you see a warning because a Visualforce page is using an outdated version, consider updating your page to the current version of Visualforce. This way, you can take advantage of new features and performance improvements.

If Push Detects File Conflicts

If you run force:source:push, and conflicts are detected, the CLI doesn't push the source.

STATE FULL NAME TYPE PROJECT PATH

Conflict NewClass	ApexClass	/classes/CoolClass.cls-meta.xml
Conflict NewClass	ApexClass	/classes/CoolClass.cls

Notice that you have a conflict. CoolClass exists in your scratch org but not in the local file system. In this new development paradigm, the local project is the source of truth. Consider if it makes sense to overwrite the conflict in the scratch org.

If conflicts have been detected and you want to override them, here's how you use the power of the force (overwrite) to push the source to a scratch org.

```
sfdx force:source:push --forceoverwrite
```

If Push Detects a Username Reference in the Source

Some metadata types include a username in their source. When you run force:source:push to push this source to a scratch org, the push command replaces the username with the scratch org's administrator username. This behavior ensures that the push succeeds, even if the scratch org does not contain the original username.

For example, let's say that you create a scratch org and use Lightning Experience to create a report folder. You then create a report and save it to the new folder. You run force:source:pull to pull down the source from the scratch org to your project. The *.reportFolder-meta.xml source file for the new ReportFolder is similar to this example; note the <sharedTo> element that contains the username test-ymmlqf5@example.com.

You then create a different scratch org whose administrator's username is test-zuwlxy321@example.com. If you push the ReportFolder's source file to the new scratch org, force:source:push replaces the test-ymmlqf5@example.com username with test-zuwlxy321@example.com.

This behavior applies only to force:source:push and scratch orgs. If you use force:mdapi:deploy to deploy metadata to a regular production org, for example, the deploy uses the username referenced in the source.

Next steps:

- Verify that the source was uploaded successfully to the scratch org, open the org in a browser.
- Add some sample test data.

How to Exclude Source When Syncing or Converting

When syncing metadata between your local file system and a scratch org, you often have source files you want to exclude. Similarly, you often want to exclude certain files when converting source to Salesforce DX project format. In both cases, you can exclude individual files or all files in a specific directory with a .forceignore file.

SEE ALSO: How to Exclude Source When Syncing or Converting Track Changes Between the Project and Scratch Org Assign a Permission Set Ways to Add Data to Your Scratch Org Pull Source from the Scratch Org to Your Project

How to Exclude Source When Syncing or Converting

When syncing metadata between your local file system and a scratch org, you often have source files you want to exclude. Similarly, you often want to exclude certain files when converting source to Salesforce DX project format. In both cases, you can exclude individual files or all files in a specific directory with a .forceignore file.

Use your favorite text editor to create a .forceignore file to specify the files or directories you want to exclude.

The .forceignore file excludes files when running the source commands: force:source:convert, force:source:push, force:source:pull, and force:source:status.

Other Files That the Source Commands Ignore

The source commands ignore these files even if they aren't included in your .forceignore file.

- Any source file or directory that begins with a "dot", such as .DS_Store or .sfdx
- Any file that ends in .dup
- package2-descriptor.json
- package2-manifest.json

Exclude Remote Changes Not Yet Synced with Your Local Source

Sometimes, you make a change directly in a scratch org but you don't want to pull that change into your local DX project. To exclude remote metadata changes, use the format <api name>.<metadata type> in .forceignore.

If you have a permission set named "dreamhouse," add dreamhouse.permissionset to .forceignore.

Exclude the Same Metadata for Source Tracking Commands

To exclude the same metadata in your local DX project (file system) and in a scratch org, create two entries to cover source:status, source:push, and source:pull. For example, if you have a custom profile called Marketing Profile, include these two entries in .forceignore:

- force-app/main/default/profiles/Marketing Profile.profile-meta.xml (ignores it during source:push Or source:status)
- Marketing Profile.profile (ignores it during source:pull or source:status)

Metadata with Special Characters

If a metadata name has special characters (such as forward slashes, backslashes, or quotation marks), we encode the file name on the local file system for all operating systems. For example, if you pull a custom profile called Custom: Marketing Profile, the colon is encoded in the resulting file name.

Custom%3A Marketing Profile.profile-meta.xml

If you reference a file name with special characters in .forceignore, use the encoded file name.

Where to Put .forceignore

Be sure the paths that you specify in .forceignore are relative to the directory containing the .forceignore file. For the .forceignore file to work its magic, you must put it in the proper location, depending on which command you are running.

- Add the .forceignore file to the root of your project for the source tracking commands: force:source:push, force:source:pull, force:source:status, and force:source:convert.
- Add the file to the Metadata retrieve directory (with package.xml) for force:mdapi:convert.

Sample Syntax

The .forceignore file has similar functionality to .gitignore. Here are some options for indicating which source to exclude. In this example, all paths are relative to the project root directory.

```
# Specify a relative path to a directory from the project root
helloWorld/main/default/classes
# Specify a wildcard directory - any directory named "classes" is excluded
**classes
# Specify file extensions
**.cls
**.pdf
# Specify a specific file
helloWorld/main/default/HelloWorld.cls
```

Assign a Permission Set

After creating your scratch org and pushing the source, you must sometimes give your users access to your application, especially if your app contains custom objects.

- 1. If needed, create the permission set in the scratch org.
 - a. Open the scratch org in your browser.

```
sfdx force:org:open -u <scratch org username/alias>
```

- **b.** From Setup, enter *Perm* in the Quick Find box, then select **Permission Sets**.
- c. Click New.
- d. Enter a descriptive label for the permission set, then click Save.
- e. Under Apps, click Assigned Apps > Edit.

- f. Under Available Apps, select your app, then click **Add** to move it to Enabled Apps.
- g. Click Save.
- 2. Pull the permission set from the scratch org to your project.

sfdx force:source:pull -u <scratch org username/alias>

3. Assign the permission set to one or more users of the org that contains the app:

```
sfdx force:user:permset:assign --permsetname <permset_name> --targetusername
<username/alias>
```

The target username must have permission to assign a permission set. Use the --onbehalfof parameter to assign a permission set to non-administrator users.

```
sfdx force:user:permset:assign --permsetname <permset_name> --targetusername <admin-user>
    --onbehalfof <non-admin-user>
```

Ways to Add Data to Your Scratch Org

Orgs for development need a small set of stock data for testing. Scratch orgs come with the same set of data as the edition on which they are based. For example, Developer Edition orgs typically include 10–15 records for key standard objects, such as Account, Contact, and Lead. These records come in handy when you're testing something like a new trigger, workflow rule, Lightning web component, Aura component, or Visualforce page.

Sometimes, the stock data doesn't meet your development needs. Scratch orgs have many uses, so we provide you the flexibility to add the data you need for your use cases. Apex tests generally create their own data. Therefore, if Apex tests are the only tests you're running in a scratch org, you can probably forget about data for the time being. However, other tests, such as UI, API, or user acceptance tests, do need baseline data. Make sure that you use consistent data sets when you run tests of each type.

The following sections describe the Salesforce CLI commands you can use to populate your scratch orgs. The commands you use depend on your current stage of development.

You can also use the force:data:soql:query CLI command to run a SOQL query against a scratch org. While the command doesn't change the data in an org, it's useful for searching or counting the data. You can also use it with other data manipulation commands.

force:data:tree Commands

The SObject Tree Save API drives the force:data:tree commands for exporting and importing data. The commands use JSON files to describe objects and relationships. The export command requires a SOQL query to select the data in an org that it writes to the JSON files. Rather than loading all records of each type and establishing relationships, the import command loads parents and children already in the hierarchy.

Note: These commands are intended for developers to test with small datasets. The query for export can return a maximum of 2000 records. The files for import can have a maximum of 200 records.

force:data:bulk Commands

Bulk API drives the force:bulk commands for exporting a basic data set from an org and storing that data in source control. You can then update or augment the data directly rather than in the org from where it came. The force:data:bulk commands use

CSV files to import data files into scratch orgs or to delete sets of data that you no longer want hanging around. Use dot notation to establish child-to-parent relationships.

force:data:record Commands

Everyone's process is unique, and you don't always need the same data as your teammates. When you want to create, modify, or delete individual records quickly, use the force:data:record:create|delete|get|update commands. No data files are needed.

Example: Export and Import Data Between Orgs

Let's say you've created the perfect set of data to test your application, and it currently resides in your default scratch org. You finished coding a new feature that you want to test in a new scratch org. You create the scratch org, push your source code, and assign the needed permission sets. Now you want to populate the scratch org with your perfect set of data from the other org. How? Read on!

SEE ALSO:

SObject Tree Request Body (REST API Developer Guide) Create Multiple Records (REST API Developer Guide) Create Nested Records (REST API Developer Guide) Salesforce Object Query Language (SOQL) Sample CSV File (Bulk API Developer Guide) Salesforce CLI Command Reference

Example: Export and Import Data Between Orgs

Let's say you've created the perfect set of data to test your application, and it currently resides in your default scratch org. You finished coding a new feature that you want to test in a new scratch org. You create the scratch org, push your source code, and assign the needed permission sets. Now you want to populate the scratch org with your perfect set of data from the other org. How? Read on!

This use case refers to the Broker and Properties custom objects of the Salesforce DX Github DreamHouse example. It's assumed that, in the first scratch org from which you are exporting data, you've created the two objects by pushing the DreamHouse source. It's also assumed that you've assigned the permission set and populated the objects with the data. In the second scratch org, however, it's assumed that you've created the two objects and assigned the permission set but not yet populated them with data. See the README of the dreamhouse-sfdx GitHub example for instructions on these tasks.

1. Export the data in your default scratch org.

Use the force:data:soql:query command to fine-tune the SELECT query so that it returns the exact set of data you want to export. This command outputs the results to your terminal or command window, but it doesn't change the data in the org. Because the SOQL query is long, the command is broken up with backslashes for easier reading. You can still cut and paste the command into your terminal window and run it.

```
sfdx force:data:soql:query --query \
    "SELECT Id, Name, Title_c, Phone_c, Mobile_Phone_c, \
        Email_c, Picture_c, \
        (SELECT Name, Address_c, City_c, State_c, Zip_c, \
            Price_c, Title_c, Beds_c, Baths_c, Picture_c, \
            Thumbnail_c, Description_c \
        FROM Properties_r) \
        FROM Broker_c"
```

2. When you're satisfied with the SELECT statement, use it to export the data into a set of JSON files.

```
sfdx force:data:tree:export --query \
    "SELECT Id, Name, Title_c, Phone_c, Mobile_Phone_c, \
        Email_c, Picture_c, \
        (SELECT Name, Address_c, City_c, State_c, Zip_c, \
            Price_c, Title_c, Beds_c, Baths_c, Picture_c, \
            Thumbnail_c, Description_c \
        FROM Properties_r) \
        FROM Broker_c" \
        --prefix export-demo --outputdir sfdx-out --plan
```

The export command writes the JSON files to the sfdx-out directory (in the current directory) and prefixes each file name with the string export-demo. The files include a plan definition file, which refers to the other files that contain the data, one for each exported object.

3. Import the data into the new scratch org by specifying the plan definition file.

Use the --plan parameter to specify the full path name of the plan execution file generated by the force:data:tree:export command. Plan execution file names always end in -plan.json.

In the previous example, you must use the --targetusername option because you are importing into a scratch org that is not your default. Use the force:org:list command to view all your scratch orgs along with their usernames and aliases. You can also use force:config:set to set the new scratch org as your default.

4. (Optional) Open the new scratch org and query the imported data using the Salesforce UI and SOQL.

sfdx force:org:open --targetusername test-wvkpnfm5z113@example.com

If you set an alias for the scratch org username, you can pass it to the --targetusername parameter.

sfdx force:org:open --targetusername <alias>

Example: Looking for a more complicated example? The easy-spaces-lwc sample app has a data plan showing how to import Accounts, related Contacts and a 3-level deep custom object chain.

SEE ALSO:

CLI Runtime Configuration Values dreamhouse-sfdx Sample GitHub Repo

Salesforce CLI Command Reference

Pull Source from the Scratch Org to Your Project

After you do an initial push, Salesforce DX tracks the changes between your local file system and your scratch org. If you change your scratch org, you usually want to pull those changes to your local project to keep both in sync.

During development, you change files locally in your file system and change the scratch org using the builders and editors that Salesforce supplies. Usually, these changes don't cause a conflict and involve unique files.

Important: You can use force:source:pull for scratch orgs only. If you're synchronizing source to any other org, use the Metadata API (force:mdapi:retrieve or force:mdapi:deploy).

By default, only changed source is synced back to your project.

The pull command does not handle merges. Projects and scratch orgs are meant to be used by one developer. Therefore, we don't anticipate file conflicts or the need to merge. However, if the pull command detects a conflict, it terminates the operation and displays the conflict information to the terminal. You can rerun the command with the force option if you want to pull changes from your scratch org to the project despite any detected conflicts.

Before you run the pull command, you can get a list of what's new, changed, and any conflicts between your local file system and the scratch org by using force:source:status. This way you can choose ahead of time which files to keep.

To pull changed source from the scratch org to the project:

sfdx force:source:pull

You can indicate either the full scratch org username or an alias. The terminal displays the results of the pull command. This example adds two Apex classes to the scratch org. The classes are then pulled to the project in the default package directory. The pull also indicates which files have changed since the last push and if a conflict exists between a version in your local project and the scratch org.

STATE	FULL NAME	TYPE	PROJECT PATH
Changed	MyWidgetClass	ApexClass	/classes/MyWidgetClass.cls-meta.xml
Changed	MyWidgetClass	ApexClass	/classes/MyWidgetClass.cls
Changed	CoolClass	ApexClass	/classes/CoolClass.cls-meta.xml
Changed	CoolClass	ApexClass	/classes/CoolClass.cls

To pull source to the project if a conflict has been detected:

```
sfdx force:source:pull --forceoverwrite
```

SEE ALSO:

Track Changes Between the Project and Scratch Org

Track Changes Between the Project and Scratch Org

When you start developing, you can change local files in your project directory or remotely in your scratch org. Before you push local changes to the scratch org or pull remote changes to the local Salesforce DX project, it's helpful to see what changes you've made.

1. To view the status of local or remote files:

sfdx force:source:status					
STATE		FULL NAME	TYPE	PROJECT PATH	
Local	Deleted Deleted	MyClass ApexC MyClass ApexC	lass /MyCl lass /MvCl	ass.cls-meta.xml ass.cls	
Local	Add Add	OtherClass	ApexClass	/OtherClass.cls-meta.xml	
Local Remote	Add Deleted	Event MyWidgetClass	QuickAction ApexClass	/Event.quickAction-meta.xml /MyWidgetClass.cls-meta.xml	

Remote	Deleted		MyWidgetClass	ApexClass	/MyWidgetClass.cls
Remote	Changed	(Conflict)	NewClass	ApexClass	/NewClass.cls-meta.xml
Remote	Changed	(Conflict)	NewClass	ApexClass	/NewClass.cls

Scratch Org Users

A scratch org includes one administrator user by default. The admin user is typically adequate for all your testing needs. But sometimes you need other users to test with different profiles and permission sets.

You can create a user by opening the scratch org in your browser and navigating to the Users page in Setup. You can also use the force:user:create CLI command to easily integrate the task into a continuous integration job.

Scratch Org User Limits, Defaults, and Considerations

- You can create a user only for a specific scratch org. If you try to create a user for a non-scratch org, the command fails. Also specify your Developer Hub, either explicitly or by setting it as your default, so that Salesforce can verify that the scratch org is active.
- Your scratch org edition determines the number of available user licenses. Your number of licenses determines the number of users you can create. For example, a Developer Edition org includes a maximum of two Salesforce user licenses. Therefore, in addition to the default administrator user, you can create one standard user.
- The new user's username must be unique across all Salesforce orgs and in the form of an email address. The username is active only within the bounds of the associated scratch org.
- You can't delete a user. The user is deactivated when you delete the scratch org with which the user is associated. Deactivating a user frees up the user license. But you can't reuse usernames, even if the associated user has been deactivated.
- The simplest way to create a user is to let the force:user:create command assign default or generated characteristics to the new user. If you want to customize your new user, create a definition file and specify it with the --definitionfile (-f) parameter. In the file, you can include all the User sObject fields and a set of Salesforce DX-specific options, described in User Definition File for Customizing a Scratch Org User on page 68. You can also specify these options on the command line.
- If you do not customize your new user, the force:user:create command creates a user with the following default characteristics.
 - The username is the existing administrator's username prepended with a timestamp. For example, if the administrator username is test-wvkpnfm5z113@example.com, the new username is something like 1505759162830_test-wvkpnfm5z113@example.com.
 - The user's profile is Standard User.
 - The values of the required fields of the User sObject are the corresponding values of the administrator user. For example, if the
 administrator's locale (specifically the LocaleSidKey field of User sObject) is en_US, the new user's locale is also en_US.

Create a Scratch Org User

Sometimes you need other users to test with different profiles and permission sets.

User Definition File for Customizing a Scratch Org User

To customize a new user, rather than use the default and generated values, create a definition file.
Generate or Change a Password for a Scratch Org User

By default, new scratch orgs contain one administrator user with no password. You can optionally set a password when you create a new user. Use the CLI to generate or change a password for any scratch org user. Once set, you can't unset a password, you can only change it.

```
SEE ALSO:
User sObject API Reference
```

Create a Scratch Org User

Sometimes you need other users to test with different profiles and permission sets.

Use the force:user:create command to create a user. Specify the --setalias parameter to assign a simple name to the user that you can reference in later CLI commands. When the command completes, it outputs the new username and user ID.

sfdx force:user:create --setalias qa-user

```
Successfully created user "test-b4agup43oxmu@example.com" with ID [0059A00000U0psQAC] for org 00D9A0000000SXKUA2.
```

```
You can see more details about this user by running "sfdx force:user:display -u test-b4agup43oxmu@example.com".
```

Users are associated with a specific scratch org and Developer Hub. Specify the scratch org or Developer Hub username or alias at the command line if they aren't already set by default in your environment. If you try to create a user for a non-scratch org, the force:user:create command fails.

```
sfdx force:user:create --setalias qa-user --targetusername my-scratchorg
--targetdevhubusername my-dev-hub
```

The force:user:create command uses default and generated values for the new user, such as the user's username, profile, and locale. You can customize the new user by creating a definition file and specifying it with the --definitionfile parameter.

sfdx force:user:create --setalias qa-user --definitionfile config/user-def.json

View the list of users associated with a scratch org with the force:user:list command. The (A) on the left identifies the administrator user that was created at the same time that the scratch org was created.

sfdx force:user:list

	ALIAS	USERNAME	PROFILE NAME	USER ID
(A)	admin-user	test-b4agup43oxmu@example.com	System Administrator	005xx000001SvBPAA0
	ci-user	wonder@example.com	Standard User	005xx000001SvBaAAK

Display details about a user with the force:user:display command.

sfdx force:user:display --targetusername ci-user

```
=== User Description
KEY VALUE
```

Access Token <long-string>

Alias	ci-user (
Id	005xx000001SvBaAAK
Instance Url	https://innovation-ability-4888-dev-ed.cs46.my.salesforce.com
Login Url	https://innovation-ability-4888-dev-ed.cs46.my.salesforce.com
Org Id	00D9A000000SXKUA2
Profile Name	Standard User
Username	test-b4agup43oxmu@example.com

User Definition File for Customizing a Scratch Org User

To customize a new user, rather than use the default and generated values, create a definition file.

The user definition file uses JSON format and can include any Salesforce User sObject field and these Salesforce DX-specific options.

Salesforce DX Option	Description	Default If Not Specified
permsets	An array of permission sets assigned to the user. Separate multiple values with commas, and enclose in square brackets.	None
	You must have previously pushed the permission sets to the scratch org with force:source:push.	
generatePassword	Boolean. Specifies whether to generate a random password for the user.	False
	If set to true, force:user:create displays the generated password after it completes. You can also view the password using force:user:describe.	
profileName	Name of a profile to associate with the user. Similar to the ProfileId field of the User sObject except that you specify the name of the profile and not its ID. Convenient when you know only the name of the profile.	Standard User

The user definition file options are case-insensitive. However, we recommend that you use lower camel case for the Salesforce DX-specific options and upper camel case for the User sObject fields. This format is consistent with other Salesforce DX definition files.

This user definition file includes some User sObject fields and three Salesforce DX options (profileName, permsets, and generatePassword).

```
{
    "Username": "testerl@sfdx.org",
    "LastName": "Hobbs",
    "Email": "testerl@sfdx.org",
    "Alias": "testerl",
    "TimeZoneSidKey": "America/Denver",
    "LocaleSidKey": "en_US",
```

```
"EmailEncodingKey": "UTF-8",
"LanguageLocaleKey": "en_US",
"profileName": "Standard Platform User",
"permsets": ["Dreamhouse", "Cloudhouse"],
"generatePassword": true
}
```

In the example, the username tester1@sfdx.org must be unique across the entire Salesforce ecosystem; otherwise, the force:user:create command fails. The alias in the Alias option is different from the alias you specify with the --setalias parameter of force:user:create. You use the Alias option alias only with the Salesforce UI. The --setalias alias is local to the computer from which you run the CLI, and you can use it with other CLI commands.

You indicate the path to the user definition file with the --definitionfile parameter of the force:user:create CLI command. You can name this file whatever you like and store it anywhere the CLI can access.

sfdx force:user:create --setalias qa-user --definitionfile config/user-def.json

You can override an option in the user definition file by specifying it as a name-value pair at the command line when you run force:user:create. This example overrides the username, list of permission sets, and whether to generate a password.

sfdx force:user:create --setalias qa-user --definitionfile config/user-def.json
permsets="Dreamy,Cloudy" Username=tester345@sfdx.org generatePassword=false

You can also add options at the command line that are not in the user definition file. This example adds the City option.

sfdx force:user:create --setalias qa-user --definitionfile config/user-def.json City=Oakland

SEE ALSO:

User sObject API Reference

Generate or Change a Password for a Scratch Org User

By default, new scratch orgs contain one administrator user with no password. You can optionally set a password when you create a new user. Use the CLI to generate or change a password for any scratch org user. Once set, you can't unset a password, you can only change it.

1. Generate a password for a scratch org user with this command:

sfdx force:user:password:generate --targetusername <username>

You can run this command for scratch org users only. The command outputs the generated password.

The target username must be an administrator user. The --onbehalfof parameter lets you assign permsets to multiple users at once, including admin users, or to users who don't have permissions to do it themselves. Specify multiple users by separating them with commas; enclose them in quotes if you include spaces. The command still requires an administrator user which you specify with the --targetusername parameter. For example, let's say the administrator user has alias admin-user and you want to generate a password for users with aliases ci-user and qa-user:

sfdx force:user:password:generate --targetusername admin-user --onbehalfof ci-user,qa-user

2. View the generated password and other user details:

```
sfdx force:user:display --targetusername ci-user
```

=== User Description KEY VALUE	
Access Token Alias	<pre></pre>
Instance Url	https://innovation-ability-4888-dev-ed.cs46.my.salesforce.com
Login Url	https://innovation-ability-4888-dev-ed.cs46.my.salesforce.com
Org Id	00D9A0000000SXKUA2
Profile Name	Standard User
Username	test-b4agup43oxmu@example.com

- 3. Log in to the scratch org with the new password:
 - a. From the force:user:display output, copy the value of Instance URL and paste it into your browser. In our example, the instance URL is https://site-fun-3277.cs46.my.salesforce.com.
 - **b.** If you've already opened the scratch org with the force:org:open command, you're automatically logged in again. To try out the new password, log out and enter the username and password listed in the output of the force:user:display command.
 - c. Click Log In to Sandbox.
 - Note: If you change a scratch org user's password using the Salesforce UI, the new password doesn't show up in the force:user:display output.

Manage Scratch Orgs from Dev Hub

You can view and delete your scratch orgs and their associated requests from the Dev Hub.

In Dev Hub, ActiveScratchOrgs represent the scratch orgs that are currently in use. ScratchOrgInfos represent the requests that were used to create scratch orgs and provide historical context.

- 1. Log in to Dev Hub org as the System Administrator or as a user with the Salesforce DX permissions.
- 2. From the App Launcher, select Active Scratch Org to see a list of all active scratch orgs.

To view more details about a scratch org, click the link in the Number column.

3. To delete an active scratch org from the Active Scratch Org list view, choose **Delete** from the dropdown.

Deleting an active scratch org does not delete the request (ScratchOrgInfo) that created it, but it does free up a scratch org so that it doesn't count against your allocations.

4. To view the requests that created the scratch orgs, select Scratch Org Info from the App Launcher.

To view more details about a request, click the link in the Number column. The details of a scratch org request include whether it's active, expired, or deleted.

To delete the request that was used to create a scratch org, choose Delete from the dropdown.
 Deleting the request (ScratchOrgInfo) also deletes the active scratch org.

SEE ALSO:

Add Salesforce DX Users (Salesforce DX Setup Guide)

CHAPTER 7 Sandboxes

In this chapter ...

- Authorize in to Your Production Org
- Create a Sandbox Definition File (Beta)
- Create, Clone, or Delete a Sandbox (Beta)

Sandboxes are copies of your Salesforce org that you can use for development, testing, and training, without compromising the data and applications in your production org.

Salesforce offers sandboxes and a set of deployment tools, so you can:

- Isolate customization and development work from your production environment until you're ready to deploy changes.
- Test changes against copies of your production data and users.
- Provide a training environment.
- Coordinate individual changes into one deployment to production.

USER PERMISSIONS

To view a sandbox:

• View Setup and Configuration

To create, refresh, activate, and delete a sandbox:

Manage Sandbox

Traditionally, you or your Admin has created and managed your sandboxes through the Setup UI. But we realize that many developers want the ability to create and manage their developer and testing environments programmatically, and to automate their CI processes. Salesforce CLI enables you to do both.

Where Do Sandboxes Fit in the Application Development Lifecycle?

The development model you use determines in which stages you use sandboxes. For more information on our development models and where sandboxes fit, see Determine Which Application Lifecycle Model Is Right for You (Trailhead).

Authorize in to Your Production Org

Right now, to run the sandbox operations commands, we support only the JWT-based flow for authorizing to your production org. This flow requires a production org with sandbox licenses instead of a Dev Hub. However, it's okay if your production org is also a Dev Hub org.

() Important: If you authenticated to your production org already using the web-based flow, you must authenticate again using the JWT-based flow to use the sandbox operations CLI commands.

The examples in Authorize an Org Using the JWT-Based Flow are geared toward scratch orgs. Follow these tips to successfully authorize to your production org.

- Be sure to use https://login.salesforce.com for sfdcLoginUrl in sfdx-project.json file. Alternatively, you can use force:auth:jwt:grant --instanceurl to specify the URL directly on the command line. This value overrides the login URL you specified in the sfdx-project.json file.
- Specify the username for your production org when running the force:auth:jwt:grant command. No need to specify a Dev Hub or indicate a default Dev Hub.
- The JWT authorization flow requires that you create a connected app. When you create the connected app, log in to your production org, not a Dev Hub org.

Create a Sandbox Definition File (Beta)

Before you can create a sandbox using Salesforce CLI, you must create a sandbox definition file. The sandbox definition file is a blueprint for the sandbox. You can create different definition files for each sandbox type that you use in the development process.

Note: As a beta feature, Salesforce CLI for sandbox operations is a preview and isn't part of the "Services" under your master subscription agreement with Salesforce. Use this feature at your sole discretion, and make your purchase decisions only on the basis of generally available products and features. Salesforce doesn't guarantee general availability of this feature within any particular time frame or at all, and we can discontinue it at any time. This feature is for evaluation purposes only, not for production use. It's offered as is and isn't supported, and Salesforce has no liability for any harm or damage arising out of or in connection with it. All restrictions, Salesforce reservation of rights, obligations concerning the Services, and terms for related Non-Salesforce Applications and Content apply equally to your use of this feature. You can provide feedback and suggestions for Salesforce CLI for sandbox operations in the Salesforce DX Trailblazer Community.

Option	Required?	Description
apexClassId	No	A reference to the ID of an Apex class that runs after each copy of the sandbox. Allows you to perform business logic on the sandbox to prepare it for use.
autoActivate	No	If true, you can activate a sandbox refresh immediately.
copyArchivedActivities	No	Full sandboxes only. This field is visible if your organization has purchased an option to copy archived activities for sandbox. To obtain this option, contact Salesforce Customer Support.

Sandbox Configuration Values

Option	Required?	Description
copyChatter	No	If true, archived Chatter data is copied to the sandbox.
description	No	A description of the sandbox (1000 or fewer characters), which helps you distinguish it from other sandboxes.
historyDays	No	Full sandboxes only. Represents the number of days of object history to be copied in the sandbox. Valid values: - 1, which means all available days 0 (default) 10 20 30 60 90 120 120 150 180
licenseType	Yes	Valid values are Developer, Developer_Pro, Partial, and Full.
sandboxName	Yes	A unique alphanumeric string (10 or fewer characters) to identify the sandbox. You can't reuse a name while a sandbox is in the process of being deleted.
sourceSandbox	Yes (for sandbox cloning)	Name of the sandbox being cloned.
templateld	Yes (for Partial sandboxes)	Optional for Full sandboxes. Not available for Developer and Developer Pro sandboxes.
		A reference to the sandbox template as identified by the 15-character ID beginning with 1ps in the URL when viewing a sandbox template in a browser. A sandbox template lets you select which objects to copy in a sandbox.

Sample Sandbox Definition File

Although you can place the sandbox definition file anywhere, we recommend keeping it in your Salesforce DX project in the config directory. When naming the file, we suggest providing a descriptive name that ends in sandbox-def.json, for example, developer-sandbox-def.json.

```
{
    "sandboxName": "dev1",
    "licenseType": "Developer"
}
```

SEE ALSO:

Tooling API: SandboxInfo

Create, Clone, or Delete a Sandbox (Beta)

Create a sandbox to use for development, testing, or training. Clone a sandbox to copy its data and metadata to another sandbox.

Note: As a beta feature, Salesforce CLI for sandbox operations is a preview and isn't part of the "Services" under your master subscription agreement with Salesforce. Use this feature at your sole discretion, and make your purchase decisions only on the basis of generally available products and features. Salesforce doesn't guarantee general availability of this feature within any particular time frame or at all, and we can discontinue it at any time. This feature is for evaluation purposes only, not for production use. It's offered as is and isn't supported, and Salesforce has no liability for any harm or damage arising out of or in connection with it. All restrictions, Salesforce reservation of rights, obligations concerning the Services, and terms for related Non-Salesforce Applications and Content apply equally to your use of this feature. You can provide feedback and suggestions for Salesforce CLI for sandbox operations in the Salesforce DX Trailblazer Community.

Before you create or clone a sandbox:

- Create a Salesforce DX project with a manifest file.
- Authorize to a production org with available sandbox licenses.
- Create the sandbox definition file.

Why We Recommend Using Aliases

When you create or clone a sandbox, the usernames generated in the sandbox are based on the usernames present in the production org or sandbox. The username looks like an email address, such as username@company.com.dev1. If the resulting username is not unique, we prepend some characters and digits to the username. The modified username looks something like **00x7Vq**username@company.com.dev1.

As you can imagine, remembering these usernames can be challenging, especially if you have several sandboxes you're managing. Aliasing is a powerful way to manage and track your orgs, and we consider it a best practice. So when you issue a command that requires the username, using an alias that you can remember can speed up things.

If you didn't set an alias when you created the sandbox, you can set one later.

```
sfdx force:alias:set MyDevSandbox=username@company.com.dev1
```

Create a Sandbox

When you create a sandbox, Salesforce copies the metadata and optionally data from your production org to a sandbox org.

```
sfdx force:org:create --type sandbox --targetusername prodOrg --definitionfile
config/dev-sandbox-def.json -a MyDevSandbox -s -w 30
```

The -s flag indicates that this sandbox is your default org for all CLI commands. If you're working with several orgs and you don't want this one to be the default, exclude this flag.

To override the values defined in the sandbox definition file, specify key=value pairs on the command line.

```
sfdx force:org:create -t sandbox sandboxName=FullSbx licenseType=Full -u prodOrg -a
MyFullSandbox -w 30
```

Tip: Because the sandbox is processed in a queue, the sandbox creation process can take longer than the default wait time of 6 minutes. We recommend setting a larger value for --wait, for example, 30 minutes.

How long the creation process takes depends on the size and complexity of your production org. You see status messages posted to output:

```
Sandbox request dev1(0GXQ000000CftJOWS) is Pending (0% completed). Sleeping 30 seconds.
Will wait 30 minutes more before timing out.
Sandbox request dev1(0GXQ000000CftJOWS) is Processing (0% completed). Sleeping 30 seconds.
Will wait 29 minutes 30 seconds more before timing out.
```

Once the wait period is over, you can run the force:org:status command to check the status of the sandbox creation process. If the sandbox is created within the wait time, the CLI automatically authenticates in to the sandbox. And the sandbox appears in the output of the force:org:listcommand. Team members can authenticate to the sandbox by running the auth:web:login command and providing their usernames and passwords.

sfdx force:auth:web:login -r https://test.salesforce.com

Clone a Sandbox

You can create a sandbox by cloning an existing sandbox rather than using your production org as your source. You can save time by customizing a sandbox with a set of data and metadata and then replicating it.

Sandbox cloning simplifies having multiple concurrent streams of work in your application life cycle. You can set up a sandbox for each type of work, such as development, testing, and staging. Your colleagues can easily clone individual sandboxes instead of sharing one sandbox and stepping on each other's toes.

```
sfdx force:org:clone -t sandbox -f config/dev-sandbox-def.json -u prodOrg -a MyDevSandbox
  -s -w 30
```

To override the configuration values defined in the sandbox definition file, specify key=value pairs on the command line.

```
sfdx force:org:clone -t sandboxName=DevSbx1 sourceSandboxName=Sbx2Clone -u prodOrg
-a MyDevSandbox -w 30
```

Tip: Because the sandbox is processed in a queue, the sandbox cloning process can take longer than the default wait time of 6 minutes. We recommend setting a larger value for --wait, for example, 30 minutes.

Once the wait period is over, you can run the force:org:status command to check the status of the sandbox cloning process. If the sandbox is cloned within the wait time, the CLI automatically authenticates in to the sandbox. And the sandbox appears in the

output of the force:org:list command. Team members can authenticate to the sandbox by running the auth:web:login command and providing their usernames and passwords.

```
sfdx force:auth:web:login -r https://test.salesforce.com
```

Check the Sandbox Status

Creating or cloning a sandbox can take several minutes. Once the command times out, you can run the force:org:status command to report on creation or cloning status. When the sandbox is ready, this command authenticates to the sandbox.

If the org:create or org:clone command times out, the alias isn't set. However, you can set it using the org:status command:

sfdx force:org:status -n DevSbx1 -a MyDevSandbox -u prodOrg

Open a Sandbox

Once the sandbox is ready, you can open it by specifying its username or alias. However, you don't have to provide its password because the CLI manages the authentication details for you.

sfdx force:org:open -u MyDevSandbox

Delete a Sandbox

You can delete a sandbox using the CLI if it was authenticated when running org:create, org:clone, or org:status. Other sandboxes that you authenticated using force:auth:web:login or force:auth:jwt:grant also appear on the org list, but must be deleted using the sandbox detail page in your production org.

sfdx force:org:delete -u MyDevSandbox

Next:

- Retrieve metadata from your sandbox to your local DX project.
- Develop directly in your sandbox, then retrieve the changes to your local DX project.
- Deploy local changes to a sandbox.

SEE ALSO:

Salesforce Help: Deploy Enhancements from Sandboxes Salesforce Help: Create, Clone, or Refresh a Sandbox Using Setup UI Authorize an Org Using the JWT-Based Flow

CHAPTER 8

In this chapter ...

- Develop Against Any Org
- Create Lightning Apps and Aura Components
- Create Lightning Web Components
- Create an Apex • Class
- Create an Apex • Trigger
- Testing
- View Apex Debug Logs
- Apex Debugger •

Development

After you import some test data, you've completed the process of setting up your project. Now, you're ready to start the development process.

Create Source Files from the CLI

To add source files from the CLI, make sure that you're working in an appropriate directory. For example, if your package directory is called force-app, create Apex classes in

force-app/main/default/classes. You can organize your source as you want underneath each package directory except for documents, custom objects, and custom object translations.

As of API version 45.0, you can build Lightning components using two programming models: Lightning Web Components and Aura Components. To organize your components' source files, your Aura components must be in the aura directory. Your Lightning web components must be in the lwc directory.

Execute one of these commands.

- apex:class:create
- apex:trigger:create
- lightning:app:create •
- lightning:component:create
- lightning:event:create
- lightning:interface:create
- lightning:test:create
- visualforce:component:create
- visualforce:page:create

Consider using these two powerful optional flags:

Option	Description
-d,outputdir	The directory for saving the created files. If you don't indicate a directory, your source is added to the current folder.
-t,template	Template used for the file creation.

Tip: If you want to know more information about a command, run it with the --help option. For example, sfdx apex:class:create --help.

Edit Source Files

Use your favorite code editor to edit Apex classes, Visualforce pages and components, Lightning web components, and Aura components in your project. You can also make edits in your default scratch org and then use force:source:pull to pull those changes down to your project. For Lightning pages (FlexiPage files) that are already in your scratch org, use the shortcut to open Lightning App Builder in a scratch org from your default browser. Lightning Pages are stored in the flexipages directory.

To edit a FlexiPage in your default browser—for example, to edit the Property_Record_Page source—execute this command.

sfdx force:source:open -f Property_Record_Page.flexipage-meta.xml

If you want to generate a URL that loads the .flexipage-meta.xml file in Lightning App Builder but does not launch your browser, use the --urlonly flag.

sfdx force:source:open -f Property_Record_Page.flexipage-meta.xml -r

SEE ALSO:

Salesforce CLI Command Reference

Develop Against Any Org

Regardless of the development model you're using, you eventually test and validate your changes in a non-source-tracked org. For those of you who don't use scratch orgs, we provide a similar experience for developing and unit testing in other environments, such as sandboxes.

You can use Salesforce CLI to retrieve and deploy metadata to non-source-tracked orgs with the same ease of pushing and pulling source to and from scratch orgs. And best of all, no extra conversion steps are required! After you retrieve the metadata, you don't have to convert it to source format. When you're ready to deploy it back to the org, you don't have to convert it to metadata format. If you're new to Salesforce CLI, Salesforce DX Project Structure and Source File Format explains the difference between source format and metadata format.

Using force:source:retrieve, you can retrieve the metadata you need in source format to your local file system (DX project). When your changes are ready for testing or production, you can use force:source:deploy to deploy your local files directly to a non-source-tracked org.

So, how do these source commands differ from the scratch org commands, source:push and source:pull? Because the changes aren't tracked, you retrieve or deploy all the specified metadata instead of only what's changed. The source you retrieve or deploy overwrites what's you have locally or in your org, respectively.

Not sure what metadata types are supported or which metadata types support wild cards in package.xml? See Metadata Types in the *Metadata API Developer Guide*.

Before You Begin

Before you begin, don't forget to:

- Create a Salesforce DX project.
- Authorize your non-source-tracked org. If connecting to a sandbox, edit your sfdx-project.json file to set sfdcLoginUrl to https://test.salesforce.com before you authorize the org. Don't forget to create aliases for your non-source-tracked orgs.

Metadata Names That Require Encoding on the Command Line

When retrieving or deploying metadata using the --metadata option, commas in metadata names require encoding to work properly.

```
Don't: sfdx force:source:deploy -m "Profile:Standard User,Layout:Page,Console"
```

```
Do: sfdx force:source:deploy -m "Profile:Standard User,Layout:Page%2CConsole"
```

Retrieve Source from a Non-Source-Tracked Org

Use the force:source:retrieve command to retrieve source from orgs that don't have source tracking, such as a sandbox or your production org. If you already have the source code and metadata in a VCS, you might be able to skip this step. If you're starting anew, you retrieve the metadata associated with the feature, project, or customization you're working on.

Note: The source:retrieve command works differently from source:pull for scratch orgs. This command doesn't notify you if there's a conflict. Instead, the source you retrieve overwrites the corresponding source files in your local project. To retrieve metadata that's in the metadata format, use force:mdapi:retrieve.

You can retrieve metadata in source format using one of these methods:

Specify a package.xml that lists the components to retrieve.

- Specify a comma-separated list of metadata component names.
- Specify a comma-separated list of source file paths to retrieve. You can use the source path option when source exists locally, for example, after you've done an initial retrieve.
- Specify a comma-separated list of package names.

If the comma-separated list you're supplying contains spaces, enclose the entire comma-separated list in one set of double quotes.

To Retrieve:	Command Example
All metadata components listed in a manifest	<pre>sfdx force:source:retrieve -x path/to/package.xml</pre>
Source files in a directory	<pre>sfdx force:source:retrieve -p path/to/source</pre>
A specific Apex class and the objects whose source is in a directory	<pre>sfdx force:source:retrieve -p path/to/apex/classes/MyClass.cls,path/to/source/objects</pre>
Source files in a comma-separated list that contains spaces	<pre>sfdx force:source:retrieve -p "path/to/objects/MyCustomObject/fields/MyField.field-meta.xml, path/to/apex/classes"</pre>
All Apex classes	sfdx force:source:retrieve -m ApexClass
A specific Apex class	<pre>sfdx force:source:retrieve -m ApexClass:MyApexClass</pre>
A layout name that contains a comma (Layout: Page, Console)	<pre>sfdx force:source:retrieve -m "Layout:Page%2C Console"</pre>
All the metadata related to a specific package or packages	sfdx force:source:retrieve -n DreamHouse

You can specify only one scoping parameter when retrieving metadata: --metadata, --sourcepath, or --manifest. If you indicate --packagenames, you can include one additional scoping parameter.

sfdx force:source:retrieve -n DreamHouse, -x manifest/package.xml

Deploy Source to a Non-Source-Tracked Org

Use the force:source:deploy command to deploy source to orgs that don't have source tracking, such as a sandbox or production org.

Note: The source:deploy command works differently from source:push for scratch orgs. The source you deploy overwrites the corresponding metadata in your org, similar to running source:push with the --force option. To deploy metadata that's in the metadata format, use force:mdapi:deploy.

You can deploy metadata in source format using these methods:

- Specify a package.xml that lists the components to deploy
- Specify a comma-separated list of metadata component names
- Specify a comma-separated list of source file paths to deploy

If the comma-separated list you're supplying contains spaces, enclose the entire comma-separated list in one set of double quotes.

To Deploy:	Command Example	
All components listed in a manifest	<pre>sfdx force:source:deploy -x path/to/package.xml</pre>	
Source files in a directory	<pre>sfdx force:source:deploy -p path/to/source</pre>	
A specific Apex class and the objects whose source is in a directory	<pre>sfdx force:source:deploy -p path/to/apex/classes/MyClass.cls,path/to/source/objects</pre>	
Source files in a comma-separated list that contains spaces	<pre>sfdx force:source:deploy -p "path/to/objects/MyCustomObject/fields/MyField.field-meta.xml, path/to/apex/classes"</pre>	
All Apex classes	sfdx force:source:deploy -m ApexClass	
A specific Apex class	<pre>sfdx force:source:deploy -m ApexClass:MyApexClass</pre>	
All custom objects and Apex classes	<pre>sfdx force:source:deploy -m CustomObject,ApexClass</pre>	
All Apex classes and a profile that has a space in its name	sfdx force:source:deploy -m "ApexClass, Profile:Content Experience Profile"	
A recently validated set of components without running Apex tests (often referred to as a quick deploy)	sfdx force:source:deploy -q VALIDATEDDEPLOYREQUESTID You can run this option after you have run tests, passed code coverage requirements, and performed a check-only deployment using the -c checkonly option.	
Even if the deployment contains warnings	sfdx force:source:deploy -g	
Regardless of whether the deployment contains errors (not recommended if deploying to a production org)	sfdx force:source:deploy -o	

Delete Non-Tracked Source

Use the force:source:delete command to delete components from orgs that don't have source tracking, such as sandboxes.

Note: Run this command from within a Salesforce DX project. To remove deleted items from scratch orgs, which have change tracking, use force:source:push.

You can delete metadata by specifying the path to the source or by listing individual metadata components, if the source already exists locally in a DX project. If the comma-separated list you're supplying contains spaces, enclose the entire comma-separated list in one set of double quotes.

To Delete:	Command Example
Source files in a directory	<pre>sfdx force:source:delete -p path/to/source</pre>
A specific component, such as a FlexiPage	sfdx force:source:delete -m FlexiPage:Broker_Record_Page

Do You Want to Retain the Generated Metadata?

Normally, when you run some CLI commands, a temporary directory with all the metadata is created then deleted upon successful completion of the command. However, retaining these files can be useful for several reasons. You can debug problems that occur during command execution. You can use the generated package.xml when running subsequent commands, or as a starting point for creating a manifest that includes all the metadata you care about.

To retain all the metadata in a specified directory path when you run these commands, set the SFDX_MDAPI_TEMP_DIR environment variable:

- force:source:deploy
- force:source:retrieve
- force:source:delete
- force:source:push
- force:source:pull
- force:source:convert
- force:org:create (if your scratch org definition contains scratch org settings, not org preferences)

Example:

SFDX_MDAPI_TEMP_DIR=/users/myName/myDXProject/metadata

Create Lightning Apps and Aura Components

To create Lightning apps and Aura components from the CLI, you must have an aura directory in your Salesforce DX project.

- 1. In <app dir>/main/default, create the aura directory.
- 2. Change to the aura directory.
- 3. In the aura directory, create a Lightning app or an Aura component.

sfdx force:lightning:app:create -n myAuraapp

sfdx force:lightning:component:create --type aura -n myAuraComponent

SEE ALSO:

Create Lightning Web Components

Create Lightning Web Components

To create a Lightning web component from the CLI, you must have an lwc directory in your Salesforce DX project.

- 1. In <app dir>/main/default, create the lwc directory.
- 2. Change to the lwc directory.

3. In the lwc directory, create the Lightning web component.

```
sfdx force:lightning:component:create --type lwc -n myLightningWebComponent
```

SEE ALSO:

Create Lightning Apps and Aura Components

Create an Apex Class

You can create Apex classes from the CLI.

- 1. If the classes directory doesn't exist in <app dir>/main/default, create it.
- 2. In the classes directory, create the class.

```
sfdx force:apex:class:create -n myclass
```

Create an Apex Trigger

Use Apex triggers to perform custom actions before or after a change to a Salesforce record, such as an insertion, update, or deletion. You can create Apex triggers from the CLI.

- 1. If the triggers directory doesn't exist in <app-dir>/main/default, create it.
- 2. Generate skeleton trigger files by executing force:apex:trigger:create.
 - Use the -s parameter to specify the sObject associated with this trigger, such as Account.
 - Use the -e parameter to specify the triggering events, such as before delete or after upsert.
 - Specify that the command generate its output into the triggers directory with the -d parameter.

```
sfdx force:apex:trigger:create -n mytrigger -s Account -e 'before insert, after upsert'
-d <app-dir</a>/main/default/triggers
```

The command generates two files.

- mytrigger.trigger-meta.xml—metadata format
- mytrigger.trigger—Apex source file
- 3. Update the generated Apex and metadata format file with your code.

SEE ALSO:

Triggers (Apex Developer Guide) Apex Triggers (Trailhead Module)

Testing

When you're ready to test changes to your Salesforce app source code, you can run Apex tests from the Salesforce DX CLI. Apex tests are run in your scratch org.

You can also execute the CLI command for running Apex tests (force:apex:test:run) from within third-party continuous integration tools, such as Jenkins.

To run Apex tests from the command line:

sfdx force:apex:test:run

This command runs all Apex tests in the scratch org asynchronously and then outputs a job ID. Pass the ID to the force:apex:test:report command to view the results. The results include the outcome of individual tests, how long each test ran, and the overall pass and fail rate.

sfdx force:apex:test:report --testrunid 7074C00000988ax

Use the --synchronous parameter to run tests from a single class synchronously. The command waits to display the test results until all tests have completed.

sfdx force:apex:test:run --synchronous --classnames TestA

Use parameters to list the test classes or suites to run, specify the output format, view code coverage results, and more. For example, the following command runs the TestA and TestB test classes, provides results in Test Anything Protocol (TAP) format, and requests code coverage results.

sfdx force:apex:test:run --classnames TestA,TestB --resultformat tap --codecoverage

Use the --tests parameter to run specific test methods using the standard notation Class.method. If you are testing a managed package, use namespace.Class.method. If you specify a test class without a method, the command runs all methods in the class. This example shows how to run two methods in the TestA class and all methods in the TestB class.

sfdx force:apex:test:run --tests TestA.excitingMethod,TestA.boringMethod,TestB

Here's the same example but with a namespace.

sfdx force:apex:test:run --tests ns.TestA.excitingMethod,ns.TestA.boringMethod,ns.TestB

You can specify either --tests or --classnames with force:apex:test:run but not both.

SEE ALSO:

Test Anything Protocol (TAP) Salesforce CLI Command Reference

View Apex Debug Logs

Apex debug logs can record database operations, system processes, and errors that occur when executing a transaction or running unit tests in your scratch org. You can use the Salesforce CLI to view the debug logs.

- 1. Open your scratch org by running force:org:open. If you have not set a default username, specify the scratch org's username or alias with the -u parameter.
- 2. Under the quick access menu (Lightning Experience) or your name (Salesforce Classic), click **Developer Console**. Opening the Developer Console starts a listener that is required by the force:apex:log commands.
- 3. If necessary, run Apex tests to generate some debug logs.

sfdx force:apex:test:run

4. Get a list of the debug logs.

```
sfdx force:apex:log:list

APPLICATION DURATION (MS) ID LOCATION SIZE (B) LOG USER OPERATION REQUEST

START TIME STATUS

Unknown 1143 07L9Axx SystemLog 23900 User User ApexTestHandler Api

2017-09-05x Success
```

5. View a debug log by passing its ID to the force:apex:log:get command.

```
sfdx force:apex:log:get --logid 07L9A00000aBYGUA2
38.0
AHEX_COTE,FINEST;AHEX_PROFILING,INFO;CALLOUT,INFO;DE,INFO;SYSTEM,DEUG;VALIDATION,INFO;VISIALFORE,INFO;WAVE,INFO;WORKFLOW,INFO
15:58:57.3
(3717091)|USER_INFO|[EXTERNAL]|0059A00000TwPM|test-ktjauhgzinnp@example.com|Pacific
Standard Time|GMT-07:00
15:58:57.3 (3888677)|EXECUTION_STARTED
15:58:57.3
(3924515)|CODE_UNIT_STARTED|[EXTERNAL]|01p9A00000FmMN|RejectDuplicateFavoriteTest.acceptNonDuplicate()
15:58:57.3 (5372873)|HEAP_ALLOCATE|[72]|Bytes:3
...
```

```
SEE ALSO:
```

Debug Log (Apex Developer Guide)

Apex Debugger

If you use Visual Studio Code (VSC) to develop Lightning Platform applications, you can use the Apex Debugger extension to debug your Apex code. Set breakpoints in your Apex classes and step through their execution to inspect your code in real time to find bugs.

You must have at least one available Apex Debugger session in your Dev Hub org.

- Trial and Developer Edition orgs do not include any Apex Debugger sessions.
- Performance Edition and Unlimited Edition orgs include one Apex Debugger session.
- To purchase Apex Debugger sessions for Enterprise Edition orgs, or to purchase more sessions for orgs that already have allocated sessions, contact Salesforce.

Enable the Apex Debugger in your scratch orgs by adding the DebugApex feature to your scratch org definition file:

```
"features": "DebugApex"
```

SEE ALSO:

Scratch Org Definition File Apex Debugger for Visual Studio Code

CHAPTER 9 Build and Release Your App

In this chapter ...

Build and Release
Your App with
Metadata API

When you finish writing your code, the next step is to deploy it. We offer different deployment options based on your role and needs as a customer, system integrator, or independent software vendor (ISV) partner.

To learn about the benefits of the different development models, review these Trailhead modules:

- Org Development Model
- Package Development Model
- Quick Start: Unlocked Packages
- Unlocked Packages for Customers

Based on your adoption readiness, review this table for your recommended options:

Ready To Create Packages to Deliver Apps

Customers and Non-ISV Partners	ISV Partners
Unlocked package	First-Generation (1GP) managed package
An unlocked package is for customers who want to organize metadata into a package and deploy the metadata (via packages) to different orgs. Unlocked packages were previously called developer-controlled packages. Note: An unlocked package offers a super-set of the capabilities of an unmanaged package. Therefore, unmanaged packages aren't listed in this table.	If you are an ISV that develops apps and lists them on AppExchange, Salesforce recommends managed packages. Second-Generation (2GP) managed packages are in beta, and many ISVs are testing them in their internal development stages. However, we recommend that you continue using 1GP managed packages for customer distribution until 2GP is generally available. 2GP doesn't yet include certain key parity features with managed packages, such as push upgrades, LMA, ability to list on AppExchange, and patch versions.
For more information, see Unlocked Packages (Generally Available) and Second-Generation Managed Packages (Beta).	For more information on 1GP managed packages, see First-Generation Managed Packages. For more information on 2GP managed packages, see Unlocked Packages (Generally Available) and Second-Generation Managed Packages (Beta)
Change sets, or org development via Salesforce CLI	N/A

Not Ready for Package Development

If you or your team isn't ready for package development, you can continue to use change sets, or try to the org development model, where you deploy changes using Salesforce CLI. For more information, see Build and Release Your App with Metadata API.

Build and Release Your App with Metadata API

Develop and test your app in your sandboxes. Use Salesforce CLI or Salesforce Extensions for VS Code to retrieve and deploy your source. This development work flow is called the org development model.

Develop and Test in a Sandbox Using the Org Development Model

With the org development model, you develop and test your changes in a sandbox using Salesforce CLI, then use Metadata API to deploy to production. Similar to change sets, the release artifact is a set of changed metadata to update in the production org. If you want to know more about this development model, see the Org Development Model module in Trailhead.

Development and Release Environments

- 1. Develop and test: Each team member has their own Developer sandbox to create their assigned customization. Developer sandboxes contain no production data.
- 2. Build release: Team members each migrate their customizations from their respective developer sandboxes to a shared Developer Pro sandbox for integration. Developer Pro sandboxes don't contain production data, but you can seed them with testing data.
- 3. Test release: For user-acceptance testing, the team uses a Partial sandbox to create a complete replica of production.
- 4. **Release:** After the release is in production, the team can use the Full sandbox to train users without the risk of altering production data. A Full sandbox includes a copy of production data.



What Tools Do I Need?

ΤοοΙ	Description	
Salesforce DX project	The Salesforce DX project contains the metadata and source files that comprise your changes. A DX project has a specific project structure and source format.	
	In addition to source files, the project contains a configuration file, sfdx-project.json. This file contains project information and enables you to leverage Salesforce DX tools for many of your development tasks.	
Deployment artifact	After testing the changes, you create the deployment artifact, a $z \pm p$ file that contains changed files to deploy. Deploy the release artifact to the full (staging) sandbox first, and then finally to production. You can think of the deployment artifact as the inbound change set. The changes don't take effect until they are deployed.	
Source control system	All changes are merged and stored in a source control system, which contains the Salesforce DX project.	
Salesforce CLI	You can use Salesforce CLI for every phase of the org development life cycle. It improves productivity by providing a single interface for all your development, testing, and automation use cases.	
Salesforce Extensions for VS Code	Salesforce Extensions for VS Code is built on top of Salesforce CLI and Visual Studio Code. Together, they are an integrated development environment for custom development on Lightning Platform. You can run Salesforce CLI commands directly from the command palette or terminal.	
Change management mechanisms	It's still important to capture your changes externally using formal change-tracking tools, such as a change list, a deployment run list, and other project management tools.	

Considerations for Deploying Apex Code

To deploy Apex to production, unit tests of your Apex code must meet coverage requirements. Code coverage indicates how many executable lines of code in your classes and triggers are covered by your test methods. Write test methods to test your triggers and classes, and then run those tests to generate code coverage information.

If you don't specify a test level when initiating a deployment, the default test execution behavior depends on the contents of your deployment package.

- If your deployment package contains Apex classes or triggers, when you deploy to production, all tests are executed, except tests that originate from a managed package.
- If your package doesn't contain Apex code, no tests are run by default.

You can run tests for a deployment of non-Apex components. You can override the default test execution behavior by setting the test level in your deployment options. Test levels are enforced regardless of the types of components present in your deployment package.

We recommend that you run all local tests in your development environment, such as a sandbox, before deploying to production. Running tests in your development environment reduces the number of tests required in a production deployment.

Develop and Test Changes Locally

Develop changes in source format, deploying to and retrieving from your Developer sandbox.

Build and Test the Release Artifact

After your team has finished its development tasks, transition to the build release phase to integrate your changes in a Developer Pro sandbox. Then build the release artifact.

Test the Release Artifact in a Staging Environment

Stage the changes and run regression tests in a Full sandbox.

Release Your App to Production

Now that all your tests have passed in the Full sandbox, you're ready to deploy to production.

Cancel a Metadata Deployment

You can cancel a metadata deployment from the CLI and specify a wait time for the command to complete.

SEE ALSO:

Metadata API Developer Guide Salesforce CLI Command Reference

Develop and Test Changes Locally

Develop changes in source format, deploying to and retrieving from your Developer sandbox.

These steps provide the high-level work flow.

- **1.** Create the source control repository.
- 2. Create a DX project.
- 3. Add the DX project files to your source control repository.
- 4. Authorize the Developer sandbox.
- 5. Perform development tasks in your developer sandbox.
- 6. Retrieve the changes from the developer sandbox. If you have a few changes, you can indicate a comma-separated list of metadata component names. If you have many changes, you can use a manifest (package.xml).

sfdx force:source:retrieve --manifest path/to/package.xml

7. Commit the changes to the source control repository.

Next: Deploy all changes the team has made to the Developer Pro sandbox, then test those changes.

SEE ALSO:

Metadata API Developer Guide Salesforce CLI Command Reference

Build and Test the Release Artifact

After your team has finished its development tasks, transition to the build release phase to integrate your changes in a Developer Pro sandbox. Then build the release artifact.

In the early testing phases, you use the source commands to deploy and retrieve metadata to and from your Developer sandboxes. The source:deploy command is not transactional and attempts to deploy all components. If any change in your project has errors but other changes are valid, the command deploys all changes that are valid and compile. Those changes that can't compile are not deployed, but the overall command completes successfully. Therefore, when building and testing your release artifact, use the mdapi:deploy command.

Here are the high-level steps in the work flow to create the release artifact.

- 1. Pull the changes from the repo so your local project contains all the changes your team has made.
- 2. Authorize the Developer Pro sandbox.
- 3. Build the release artifact so you can deploy the changes to the sandbox.
 - a. Convert the source to metadata format.

sfdx force:source:convert --rootdir force-app --outputdir tmp convert

This command also creates a manifest in the output directory. The manifest indicates what changes to include in the build artifact.

- **b.** Create a .zip file of the contents in the tmp convert directory.
- 4. Delete the output directory, for example, tmp convert.
- 5. Run the deploy command that mimics what you'll deploy to production, for example:

```
sfdx force:mdapi:deploy --zipfile winter19.zip --targetusername dev-pro-sandbox \
--testlevel RunSpecifiedTests --runtests TestMyCode
```

- 6. Open the sandbox.
- 7. Perform testing.
- **8.** If the testing passes, continue to the test release phase where you deploy the release artifact to the partial sandbox. Then perform user-acceptance testing.

After the testing passes, move to the release phase and perform regression tests in the Full sandbox.

Test the Release Artifact in a Staging Environment

Stage the changes and run regression tests in a Full sandbox.

After you have made all your changes based on the integration testing, the next step is to stage the changes in a Full sandbox. The process of deploying changes to the Full sandbox is similar to the process you used to deploy changes to your Developer Pro sandbox. This phase includes regression testing and mimics how you release the changes to production.

These steps provide the high-level work flow.

- 1. Authorize the Full sandbox.
- 2. (Optional) If you made any changes based on your testing in the Developer Pro sandbox, create a new release artifact (.zip). If not, use the existing release artifact.

3. To validate the deployment without saving the components in the target org, run all local (regression) tests. A validation enables you to verify the results of tests that would be executed during a deployment, but doesn't commit any changes.

```
sfdx force:mdapi:deploy --checkonly --zipfile mdapi_output/winter19.zip --targetusername
full-sandbox --testlevel RunLocalTests
```

4. Test the actual production deployment steps in the staging sandbox. Set up the same quick deploy that you plan to execute against the production org.

```
sfdx force:mdapi:deploy --checkonly --zipfile mdapi_output/winter19.zip --targetusername
full-sandbox --testlevel RunSpecifiedTests TestLanguageCourseTrigger
```

This command returns a job ID that you reference in the quick deploy.

5. Next, test the quick deploy using the job ID returned in the previous step.

```
sfdx force:mdapi:deploy --targetusername full-sandbox --validateddeployrequestid jobID
```

After you run the quick deploy, you have 10 days to perform the deployment to production.

Release Your App to Production

Now that all your tests have passed in the Full sandbox, you're ready to deploy to production.

You have two options to deploy your metadata to production using Salesforce CLI. For information on all command options, view the command --help.

- Deploy your metadata directly to your production org using the source:deploy command.
- Convert your local DX project files to metadata format, then use the mdapi:deploy command.
- 1. In your deployment run list, complete any pre-deployment tasks.
- 2. Authorize your production org.
- 3. If you plan to use the mdapi:deploy command, convert your files from source format to metadata format.

sfdx force:source:convert

4. Set up the quick deploy.

Salesforce CLI Command	Command-Line Example			
<pre>sfdx force:source:deploy</pre>	<pre>sfdx force:source:deploycheckonly \sourcepath force-apptargetusername production-org \testlevel RunLocalTests</pre>			
sfdx force:mdapi:deploy	<pre>sfdx force:mdapi:deploycheckonly \zipfile winter19.ziptargetusername production-org \testlevel RunLocalTests</pre>			

This command returns a job ID that you reference in the quick deploy.

- 5. After the tests are run, verify that all the Apex tests have passed. Be sure that the tests cover at least 75% of the code being deployed.
- **6.** Run the quick deploy:

Salesforce CLI Command	Command-Line Example
<pre>sfdx force:source:deploy</pre>	<pre>sfdx force:source:deploy \targetusername production-org \validateddeployrequestid jobID</pre>
<pre>sfdx force:mdapi:deploy</pre>	<pre>sfdx force:mdapi:deploy \targetusername production-org \validateddeployrequestid jobID</pre>

7. Open the production org, then perform any post-deployment tasks listed in the deployment run list.

SEE ALSO:

Metadata API Developer Guide Salesforce CLI Command Reference

Cancel a Metadata Deployment

You can cancel a metadata deployment from the CLI and specify a wait time for the command to complete.

To cancel your most recent deployment, run force:mdapi:deploy:cancel. You can cancel earlier deployments by adding the -i (JOBID) parameter to specify the deployment that you want to cancel.

```
$ sfdx force:mdapi:deploy:cancel -i <jobid>
```

The default wait time for the cancel command to complete and display its results in the terminal window is 33 minutes. If the command isn't completed by the end of the wait period, the CLI returns control of the terminal window to you. You can adjust the wait time as needed by specifying the number of minutes in the -w (WAIT) parameter, as shown in the following example:

\$ sfdx force:mdapi:deploy:cancel -w 20

Curious about the status of a canceled deployment? Run a deployment report.

\$ sfdx force:mdapi:deploy:report

SEE ALSO:

Metadata API Developer Guide Salesforce CLI Command Reference

CHAPTER 10 First-Generation Managed Packages

In this chapter ...

- Build and Release Your App with Managed Packages
- View Information About a Package

If you're an ISV, you want to build a managed package. A managed package is a bundle of components that make up an application or piece of functionality. A managed package is a great way to release an app for sale and to support licensing your features. You can protect intellectual property because the source code of many components is not available through the package. You can also roll out upgrades to the package.

When you're working with your production org, you create a .zip file of metadata components and deploy them through Metadata API. The .zip file contains:

- A package manifest (package.xml) that lists what to retrieve or deploy
- One or more XML components organized into folders

If you don't have the packaged source already in the source format, you can retrieve it from the org and convert it using the CLI.

Build and Release Your App with Managed Packages

If you developed and tested your app, you're well on your way to releasing it. Luckily, when it's time to build and release an app as a managed package, you've got options. You can package an app you developed from scratch. If you're experimenting, you can also build the sample app from Salesforce and emulate the release process.

Working with a package is an iterative process. You typically retrieve, convert, and deploy source multiple times as you create scratch orgs, test, and update the package components.

Chances are, you already have a namespace and package defined in your packaging org. If not, run this command to open the packaging org in your browser.

sfdx force:org:open --targetusername me@my.org --path lightning/setup/Package/home

In the Salesforce UI, you can define a namespace and a package. Each packaging org can have a single managed package and one namespace.

Be sure to link the namespace to your Dev Hub org.

Packaging Checklist

Ready to deploy your packaging metadata and start creating a package? Take a few minutes to verify that you covered the items in this checklist, and you're good to go.

Deploy the Package Metadata to the Packaging Org

Before you deploy the package metadata into your packaging org, you convert from source format to metadata format.

Create a Beta Version of Your App

Test your app in a scratch org, or share the app for evaluation by creating a beta version.

Install the Package in a Target Org

After you create a package with the CLI, install the package in a target org. You can install the package in any org you can authenticate, including a scratch org.

Create a Managed Package Version of Your App

After your testing is done, your app is almost ready to be published in your enterprise or on AppExchange. Generate a new managed package version in your Dev Hub org.

SEE ALSO:

ISVforce Guide Link a Namespace to a Dev Hub Org Retrieve Source from an Existing Managed Package

Packaging Checklist

Ready to deploy your packaging metadata and start creating a package? Take a few minutes to verify that you covered the items in this checklist, and you're good to go.

- 1. Link the namespace of each package you want to work with to the Dev Hub org.
- 2. Copy the metadata of the package from your version control system to a local project.
- **3.** Update the config files, if needed.

For example, to work with managed packages, sfdx-project.json must include the namespace.

"namespace": "acme_example",

4. (Optional) Create an alias for each org you want to work with.

If you haven't yet created an alias for each org, consider doing that now. Using aliases is an easy way to switch between orgs when you're working in the CLI.

- 5. Authenticate the Dev Hub org.
- 6. Create a scratch org.

A scratch org is different than a sandbox org. You specify the org shape using project-scratch.json. To create a scratch org and set it as the defaultusername org, run this command from the project directory.

sfdx force:org:create -s -f config/project-scratch-def.json

- **7.** Push source to the scratch org.
- 8. Update source in the scratch org as needed.
- 9. Pull the source from the scratch org if you used declarative tools to make changes there.

With these steps complete, you're ready to deploy your package metadata to the packaging org.

SEE ALSO:

Sample Repository on GitHub Authorization Create Scratch Orgs Push Source to the Scratch Org

Deploy the Package Metadata to the Packaging Org

Before you deploy the package metadata into your packaging org, you convert from source format to metadata format.

It's likely that you have some files that you don't want to convert to metadata format. Create a .forceignore file to indicate which files to ignore.

1. Convert from source format to the metadata format.

sfdx force:source:convert --outputdir mdapi_output_dir --packagename managed_pkg_name Create the output directory in the root of your project, not in the package directory. If the output directory doesn't exist, it's created. Be sure to include the --packagename so that the converted metadata is added to the managed package in your packaging org.

- **2.** Review the contents of the output directory.
 - ls -lR mdapi_output_dir
- 3. Authenticate the packaging org, if needed. This example specifies the org with an alias called MyPackagingOrgAlias, which helps you refer to the org more easily in subsequent commands. sfdx force:auth:web:login --setalias MyPackagingOrgAlias You can also authenticate with an OAuth client ID: sfdx force:auth:web:login --clientid oauth client id
- 4. Deploy the package metadata back to the packaging org. sfdx force:mdapi:deploy --deploydir mdapi_output_dir --targetusername me@example.com

The --targetusername is the username. Instead of the username, you can use -u MyPackagingOrgAlias to refer to your previously defined org alias. You can use other options, like --wait to specify the number of minutes to wait. Use the --zipfile parameter to provide the path to a zip file that contains your metadata. Don't run tests at the same time as you deploy the metadata. You can run tests during the package upload process.

A message displays the job ID for the deployment.

5. Check the status of the deployment.

When you run force:mdapi:deploy, the job ID and target username are stored, so you don't have to specify these required parameters to check the status. These stored values are overwritten when you run force:mdapi:deploy again.

sfdx force:mdapi:deploy:report

If you want to check the status of a different deploy operation, specify the job ID on the command line, which overrides the stored job ID.

SEE ALSO:

Salesforce CLI Command Reference How to Exclude Source When Syncing or Converting

Create a Beta Version of Your App

Test your app in a scratch org, or share the app for evaluation by creating a beta version.

If you specified the package name when you converted source to metadata format, both the changed and new components are automatically added to the package. Including the package name in that stage of the process lets you take full advantage of end-to-end automation.

If, for some reason, you don't want to include new components, you have two choices. You can omit the package name when you convert source or remove components from the package in the Salesforce UI before you create the package version.

Create the beta version of a managed package by running the commands against your packaging org, not the Dev Hub org.

1. Ensure that you've authorized the packaging org.

```
sfdx force:auth:web:login --targetusername me@example.com
```

2. Create the beta version of the package.

```
sfdx force:package1:version:create --packageid package_id --name package_version_name
```

You can get the package ID on the package detail page in the packaging org. If you want to protect the package with an installation key, add it now or when you create the released version of your package. The --installationkey supplied from the CLI is equivalent to the Password field that you see when working with packages through the Salesforce user interface. When you include a value for --installationkey, you or a subscriber must supply the key before you can install the package in a target org.

You're now ready to create a scratch org and install the package there for testing. By default, the create command generates a beta version of your managed package.

Later, when you're ready to create the Managed - Released version of your package, include the -m (--managedreleased true) parameter.

Note: After you create a managed-released version of your package, many properties of the components added to the package are no longer editable. Refer to the *ISVforce Guide* to understand the differences between beta and managed-released versions of your package.

SEE ALSO: Salesforce CLI Command Reference ISVforce Guide Link a Namespace to a Dev Hub Org

Install the Package in a Target Org

After you create a package with the CLI, install the package in a target org. You can install the package in any org you can authenticate, including a scratch org.

If you want to create a scratch org and set it as the defaultusername org, run this command from the project directory.

```
sfdx force:org:create -s -f config/project-scratch-def.json
```

To locate the ID of the package version to install, run force:package1:version:list.

METADATAPACKAGEVERSIONID	METADATAPACKAGEID	NAME	VERSION	RELEASESTATE	BUILDNUMBER
04txx000000690AAA	033xx00000007coAAA	r00	1.0.0	Released	1
04txx00000069tAAA	033xx00000007coAAA	r01	1.1.0	Released	1
04txx00000069uAAA	033xx00000007coAAA	r02	1.2.0	Released	1
04txx00000069yAAA	033xx00000007coAAA	r03	1.3.0	Released	1
04txx000000069zAAA	033xx00000007coAAA	r04	1.4.0	Released	1

You can then copy the package version ID you want to install. For example, the ID 04txx000000069zAAA is for version 1.4.0.

1. Install the package. You supply the package alias or version ID, which starts with 04t, in the required --package parameter.

sfdx force:package:install --package 04txx00000069zAAA

If you've set a default target org, the package is installed there. You can specify a different target org with the --targetusername parameter. If the package is protected by an installation key, supply the key with the --installationkey parameter.

To uninstall a package, open the target org and choose Setup. On the Installed Packages page, locate the package and choose Uninstall.

SEE ALSO: ISVforce Guide Salesforce CLI Command Reference

Create a Managed Package Version of Your App

After your testing is done, your app is almost ready to be published in your enterprise or on AppExchange. Generate a new managed package version in your Dev Hub org.

Ensure that you've authorized the packaging org and can view the existing package versions.

sfdx force:auth:web:login --instanceurl https://test.salesforce.com --setdefaultusername
org_alias

View the existing package versions for a specific package to get the ID for the version you want to install.

sfdx force:package1:version:list --packageid 033...

To view details for all packages in the packaging org, run the command with no parameters.

More than one beta package can use the same version number. However, you can use each version number for only one *managed* package version. You can specify major or minor version numbers.

You can also include URLs for a post-installation script and release notes. Before you create a managed package, make sure that you've configured your developer settings, including the namespace prefix.



Note: After you create a managed package version, you can't change some attributes of Salesforce components used in the package. The *ISVforce Guide* has information on editable components.

1. Create the managed package. Include the --managedreleased parameter.

```
sfdx force:package1:version:create --packageid 033xx00000007oi --name "Spring 17"
--description "Spring 17 Release" --version 3.2 --managedreleased
```

You can use other options, like --wait to specify the number of minutes to wait.

To protect the package with an installation key, include a value for --installationkey. Then, you or a subscriber must supply the key before you can install the package in a target org.

After the managed package version is created, you can retrieve the new package version ID using force:package1:version:list.

SEE ALSO:

Salesforce CLI Command Reference ISV force Guide Link a Namespace to a Dev Hub Org

View Information About a Package

View the details about a specific package version, including its metadata package ID, package name, release state, and build number.

 From the project directory, run this command, supplying a package version ID. force:package1:version:display -i 04txx000000069yAAA The output is similar to this example.

METADATAPACKAGEVERSIONID	METADATAPACKAGEID	NAME	VERSION	RELEASESTATE	BUILDNUMBER
04txx000000069yAAA	033xx00000007coAAA	r03	1.3.0	Released	1
04txx000000069yAAA	033xx00000011coAAA	r03		Released	1

View All Package Versions in the Org

View the details about all package versions in the org.

Package IDs

When you work with packages using the CLI, the package IDs refer either to a unique package or a unique package version.

SEE ALSO:

Salesforce CLI Command Reference

View All Package Versions in the Org

View the details about all package versions in the org.

- 1. From the project directory, run the list command.
 - force:package1:version:list

The output is similar to this example. When you view the package versions, the list shows a single package for multiple package versions.

METADATAPACKAGEVERSIONID	METADATAPACKAGEID	NAME	VERSION	RELEASESTATE	BUILDNUMBER
04txx000000690AAA	033xx00000007coAAA	r00	1.0.0	Released	1
04txx00000069tAAA	033xx00000007coAAA	r01	1.1.0	Released	1
04txx00000069uAAA	033xx00000007coAAA	r02	1.2.0	Released	1
04txx00000069yAAA	033xx00000007coAAA	r03	1.3.0	Released	1
04txx000000069zAAA	033xx00000007coAAA	r04	1.4.0	Released	1

SEE ALSO:

Salesforce CLI Command Reference

Package IDs

When you work with packages using the CLI, the package IDs refer either to a unique package or a unique package version.

The relationship of package version to package is one-to-many.

ID Example	Description	Used Where
033xx00000007oi	Metadata Package ID	Generated when you create a package. A single package can have one or more associated package version IDs. The package ID remains the same, whether it has a corresponding beta or released package version.
04tA00000081MX	Metadata Package Version ID	Generated when you create a package version.

CHAPTER 11

Unlocked Packages (Generally Available) and Second-Generation Managed Packages (Beta)

In this chapter ...

- Second-Generation Packaging
- What's a Package?
- Types of Packaging Projects
- Before You Create Second-Generation Packages
- Workflow for Second-Generation Packages
- Plan Second-Generation Packages
- Configure Packages
- Create a Package
- Install a Package
- Migrate Deprecated Metadata from Unlocked Packages
- Uninstall a Package

Unlocked packages give customers and system integrators a means to organize their metadata into a package and then deploy the metadata (via packages) to different orgs. Second-generation (2GP) managed packages are for Independent Software Vendors (ISVs) and partners who want to test this beta version in preparation for 2GP managed packages when they are generally available.

Note: As a beta feature, Second-Generation Managed Packages is a preview and isn't part of the "Services" under your master subscription agreement with Salesforce. Use this feature at your sole discretion, and make your purchase decisions only on the basis of generally available products and features. Salesforce doesn't guarantee general availability of this feature within any particular time frame or at all, and we can discontinue it at any time. This feature is for evaluation purposes only, not for production use. It's offered as is and isn't supported, and Salesforce has no liability for any harm or damage arising out of or in connection with it. All restrictions, Salesforce Applications and Content apply equally to your use of this feature. You can provide feedback and suggestions for second-generation managed packages in the Packaging 2 Beta group in the Trailblazer Community.

Have you turned on the beta in your Dev Hub org? For information on enabling the beta, see Enable Second-Generation Packaging in the *Salesforce DX Setup Guide*.

Second-Generation Packaging

Second-generation packaging (2GP) allows customers and system integrators to create packages in a source-driven development environment. You can create and deploy packages in your own Salesforce org or develop and distribute packages to your customers.

Use packaging to take advantage of these helpful features.

- The Salesforce CLI provides commands for the entire application life cycle so you can work efficiently with your packages, scratch orgs, and development processes.
- Options for enterprise customers to organize and deploy metadata to production orgs.
- Multiple packages per namespace, so you can better organize your source and easily share Apex code. You can use public Apex classes across packages rather than global Apex classes.
- Feature branch development and testing.
- Completely API-driven functionality.
- Packages that are built directly from the source.
- Ease of development and distribution of dependent packages.

SEE ALSO:

Salesforce DX (Salesforce Developer Center Web Site) Salesforce CLI Command Reference

What's a Package?

If you're new to packaging, you can think about a package as a container that you fill with metadata. You use packages to move the metadata from one location to another. Each second-generation package has a distinct life cycle.

You add metadata to a package and, when you're ready to "send" or release the package, you take a snapshot of it. We call the snapshot a package version. The package version can be installed in a scratch org, sandbox org, DE org, or production org. Installing the package is similar to deploying metadata.

As you add, remove, or change the package metadata, you can create another snapshot, called a package versions. Because each package version has a version number, you can install a new package version into the same org through a package upgrade.

The cycle of package development can be repeated any number of times. You can change metadata, create a package version, test the package version, and finally deploy or install the package to a production org. This distinct app development lifecycle lets you control exactly what, when and how your metadata is rolled out. In the installed org, you can inspect which metadata came from which package and the set of all metadata associated with a specific package version.

Types of Packaging Projects

Salesforce offers several methods for building and releasing apps. In the package development model, the new and improved source of truth is your version control system. You use Salesforce DX projects to organize your source into package directories. Your end goal is to create packages using those directories that are versionable, easy to maintain, update, install, and upgrade.

The type of projects you define are up to you. It all depends on whether you're an ISV or an enterprise customer, and your business goals.
Packaging for ISVs

Salesforce ISVs approach packaging projects with laser focus on the steps required to build and deliver a successful commercial app. ISVs are interested in sound design patterns that support solving common problems as they develop their packages. ISVs also have considerations such as protecting intellectual property and rolling out upgrades to their installed base.

Enterprise Customers

If you're an enterprise customer, you may be wondering: can I use packaging in my Salesforce org? Isn't packaging a partner-oriented feature? We have good news for you. With unlocked packages, developers can more easily develop and deploy their apps and functionality.

Packaging for ISVs

Salesforce ISVs approach packaging projects with laser focus on the steps required to build and deliver a successful commercial app. ISVs are interested in sound design patterns that support solving common problems as they develop their packages. ISVs also have considerations such as protecting intellectual property and rolling out upgrades to their installed base.

Enterprise Customers

If you're an enterprise customer, you may be wondering: can I use packaging in my Salesforce org? Isn't packaging a partner-oriented feature? We have good news for you. With unlocked packages, developers can more easily develop and deploy their apps and functionality.

To understand the power of unlocked packages, let's first discuss how packaging works in the traditional change set development model. For most production orgs, metadata is traditionally contained in two buckets: a set of managed packages installed from AppExchange or unpackaged metadata. At the same time, enterprises often invest in Salesforce customizations to support business processes and extend the power of the Salesforce platform. In the change set development model, all metadata that belongs to the custom app or extension is contained in your Salesforce org. However, it's not isolated or organized in a way that makes it easy to upgrade and maintain.

In the package development model, unlocked packages let you harness and extend your investments in the Salesforce platform in two ways. You can organize the unpackaged metadata in your production org into well-defined packages. These packages are versionable and easy to maintain, update, install, and upgrade. In your production org, you can inspect which metadata came from which package version and the set of all metadata associated with the package version. When your "happy soup" of metadata is organized into packages, updates and customizations are much easier to manage across your development team.

Before You Create Second-Generation Packages

When you use second-generation (2GP) packaging, be sure that you've set up things properly.

Is the beta in your Dev Hub org enabled? For information on enabling the beta, see Enable Second-Generation Packaging in the Salesforce DX Setup Guide.

🕜 Note: Second-generation packaging is available with these licenses: Salesforce or Salesforce Limited Access - Free (partners only).

Users who work with 2GP packages need the correct permission set in the Dev Hub org. A user who doesn't have the System Administrator profile needs the Create and Update Second-Generation Packages permission. For more information, see Add Salesforce DX Users in the Salesforce DX Setup Guide.

The maximum number of 2GP packages that you can create from a Dev Hub per day is the same as your daily scratch org allocation. Scratch orgs and packages are counted independently, so creating a 2GP package does not count against your daily scratch org limit. To view your scratch org limits, use the CLI:

sfdx force:limits:api:display -u <Dev Hub username or alias>

For more information on scratch org limits, see Scratch Orgs.

Know Your Orgs

Some of the orgs that you use with second-generation packaging have a unique purpose.

Sample Repository

To work with a sample repository, get the Salesforce DX DreamHouse app in GitHub and check out the Packaging 2 branch.

Review Org Setup

After you verify or configure the org details, you're ready to go.

Know Your Orgs

Some of the orgs that you use with second-generation packaging have a unique purpose.

Choose Your Dev Hub Org

Use the Dev Hub org for these purposes.

- As owner of all second-generation packages
- To link your namespaces so that they're known to your scratch orgs
- To authorize and run your force:package commands
- Important: You can create second-generation packages in any Dev Hub org where Packaging 2 is enabled. You can easily move metadata from one Dev Hub org to another. But every time you create a second-generation package using the Salesforce CLI, the Dev Hub org "owns" the package and package ownership can't be transferred from one Dev Hub org to another. When you're ready to define and create a package for production use, be sure to create it in your production or "master" Dev Hub org and not in a test Dev Hub org.

Other Orgs

When you work with packages, you also use these orgs:

• The primary purpose of the namespace org is to provide a package namespace. If you want to use the namespace strictly for testing, choose a disposable namespace.

Note: After you create a namespace org and specify the namespace in it, open the Dev Hub org and link the namespace org to the Dev Hub org.

- You can create scratch orgs on the fly to use while testing your packages.
- The target or installation org is where you install the package. This org is sometimes called a "subscriber" org.

SEE ALSO:

Link a Namespace to a Dev Hub Org

Sample Repository

To work with a sample repository, get the Salesforce DX DreamHouse app in GitHub and check out the Packaging 2 branch.

You can clone the sample repository from GitHub.

```
git clone https://github.com/dreamhouseapp/dreamhouse-sfdx
cd dreamhouse-sfdx
git checkout pkg2-beta
```

Learn more about using other Salesforce DX features in the README file in the master branch of the dreamhouse-sfdx repository. The pkg2-beta branch contains sample files and resources you can use.

Review Org Setup

After you verify or configure the org details, you're ready to go.

First, if you plan to use namespaces with your packages, complete these tasks.

- Note: If you're creating a managed second-generation package, a namespace is required. If you're creating an unlocked package, a namespace is optional.
- 1. Log in to the Dev Hub org.
- 2. Enable and deploy My Domain in the Dev Hub org.

If you don't enable and deploy My Domain, you can't link your namespace to the Dev Hub org.

3. Create a Developer Edition (DE) org to use as your namespace org for testing.

Unlike first-generation packaging, there's no need to create a package or packaging org for second-generation packaging. You can create multiple test packages for a namespace by using the CLI and authenticating to the Dev Hub org.

4. Log in to the namespace org, and specify a namespace.

Important: Use a disposable namespace, because you can use it only for testing. Use your real namespace later when you're ready to adopt packaging for your production work.

Most customers use a single namespace org but you can use more than one if you want more granular control over how you organize metadata. There can be a short delay between the time when you deploy My Domain and when the Link Namespace button appears in the Dev Hub org.

5. In the Dev Hub org, link each namespace you want to work with.

For all types of packages, complete these tasks from the CLI.

- 1. Create a project and specify your configuration options in sfdx-project.json.
- 2. Create a scratch org where you can develop, install, and test versions of your packages.
- **3.** (Recommended) Specify an alias for each org you plan to work with. An alias lets you give your orgs an easy to remember name so you can move between orgs easily while you work with packages.

SEE ALSO:

Salesforce DX Setup Guide Salesforce CLI Command Reference Link a Namespace to a Dev Hub Org Scratch Orgs

Workflow for Second-Generation Packages

After developing an app in a scratch org and testing it, you can create and install a second-generation package directly from the Salesforce command line.



Note: Be sure to run the package CLI commands from the directory that contains the sfdx-project.json file.

The basic workflow includes these steps. See specific topics for details about each step.

1. Authorize the Dev Hub org, and create a scratch org.

When you perform this step, include the -d option. You can then omit the Dev Hub username when running subsequent Salesforce CLI commands.



- 2. Verify that all package components are in the project directory where you want to create a package.
- **3.** From the Salesforce DX project directory, create the package.

```
sfdx force:package:create --name "Expense Manager" --path force-app \
--packagetype Unlocked
```

Wait a few minutes before you proceed to the next step.

4. Configure the package in the sfdx-project.json file. The CLI automatically updates the project file to include the package directory and creates an alias based on the package name.

```
{
   "packageDirectories": [
      {
         "path": "force-app",
         "default": true,
         "package": "Expense Manager",
         "versionName": "ver 0.1",
         "versionNumber": "0.1.0.NEXT"
      }
   ],
   "namespace": "",
   "sfdcLoginUrl": "https://login.salesforce.com",
   "sourceApiVersion": "43.0",
   "packageAliases": {
      "Expense Manager": "OHoxxx"
   }
}
```

Notice the placeholder values for versionName and versionNumber. You can update these values, or indicate base packages that this package depends on. You can specify the features and org preferences required for the metadata of your package in the project file, or you can opt to specify this information in an external .json file, such as the scratch org definition.

5. Create a package version. This example assumes the package metadata is in the force-app directory.

```
sfdx force:package:version:create --package "Expense Manager" --directory force-app \
--installationkey test1234 --definitionfile config/project-scratch-def.json --wait 10
```

You can indicate the .json file on the command line or define it in the sfdx-project.json. Wait a few minutes before you proceed to the next step.

6. Install and test the package version in your scratch org.

```
sfdx force:package:install --package "Expense Manager" -u MyScratchOrgAlias \
--wait 10 --publishwait 10
```

7. After the package is installed, open the scratch org to view the package.

```
sfdx force:org:open -u MyScratchOrgAlias
```

Plan Second-Generation Packages

Investing time to plan your package helps you to develop, build, and deploy it successfully. Good planning ensures that your package is reusable and easily upgradable.

Namespaces

A namespace is a one to 15-character alphanumeric identifier that distinguishes your package and its contents from packages of other developers. A namespace is assigned to a package at the time that it's created, and can't be changed.

Package Types

Here are the package types for second-generation packaging.

Best Practices for Second-Generation Packages

We suggest that you follow these best practices when working with second-generation packages.

Package IDs

When running Salesforce CLI commands, you can identify packages and package versions by their aliases or package IDs. A package ID is a unique identifier for packages and package versions.

Namespaces

A namespace is a one to 15-character alphanumeric identifier that distinguishes your package and its contents from packages of other developers. A namespace is assigned to a package at the time that it's created, and can't be changed.

Important: When creating a namespace, use something that's useful and informative to users. However, don't name a namespace after a person (for example, by using a person's name, nickname, or private information).

When you work with 2GP namespaces, keep these considerations in mind.

- For a managed package, a namespace is required.
- You can develop more than one package with the same namespace but you can associate each package with only a single namespace. If you're an ISV, we recommend that you use the same namespace for all your packages.
- If you work with more than one namespace, you set up one project for each namespace.

When you specify a package namespace, every component added to a package has the namespace prefixed to the component API name. Let's say you have a custom object called Insurance_Agent with the API name, Insurance_Agent_c. If you add this component to a package associated with the Acme namespace, the API name becomes Acme__Insurance_Agent_c.

Namespaces for Unlocked Packages

You can choose to create unlocked packages with or without a specific namespace.

If you're an enterprise customer who's new to packaging, you're probably adopting packages in several stages. In that case, a namespace prefix such as Acme _____ can help you identify what's packaged and what's still unpackaged metadata in your production orgs.

Creating a no-namespace package gives developers more control over how to organize and distribute parts of an application. This flexibility can be useful during the initial package adoption phase.

When you build new functionality with no dependencies on unpackaged org metadata, evaluate your development plans to decide whether to use a namespace.

Existing, unpackaged metadata can be migrated only to an unlocked package with no namespace. Therefore, to be able to migrate existing metadata, create no-namespace, unlocked packages.

Ø

Note: You can't install a no-namespace, unlocked package into any org with a namespace (for example, a scratch org with a namespace or a first-generation packaging org).

Namespace-Based Visibility for Apex Classes in Second-Generation (2GP) Packages

The @namespaceAccessible annotation marks public or protected Apex in a package as available to other packages with the same namespace. Unless explicitly annotated, Apex classes, methods, interfaces, and properties defined in a 2GP package aren't accessible to other packages with which they share a namespace. There is no impact on Apex that isn't packaged.

Note: Apex that is declared global is always available across all namespaces, without the need of this annotation.

Restrictions on package-level Apex visibility are enforced when Apex is invoked from another Apex class or trigger. However, when Apex is invoked as a Visualforce or Lightning controller, annotation used across packages is not enforced; it will be enforced in a future release.

You can add or remove the <code>@namespaceAccessible</code> annotation at any time, even on managed and released Apex code. Make sure that you don't have dependent packages relying on the functionality of the annotation before adding or removing it.

Note: When adding or removing @namespaceAccessible Apex from a package, consider the impact to customers with installed versions of other packages that reference this package's annotation. Before pushing a package upgrade, ensure that no customer is running a package version that would fail to fully compile when the upgrade is pushed.

This example shows an Apex class marked with the <code>@namespaceAccessible</code> annotation. The class is accessible to other packages within the same namespace. The first constructor is also visible within the namespace, but the second constructor isn't.

```
// A namespace-visible Apex class
@namespaceAccessible
public class MyClass {
    private Boolean bypassFLS;
    // A namespace-visible constructor that only allows secure use
    @namespaceAccessible
    public MyClass() {
        bypassFLS = false;
    }
    // A package private constructor that allows use in trusted contexts,
    // but only internal to the package
    public MyClass (Boolean bypassFLS) {
        this.bypassFLS = bypassFLS;
    }
    @namespaceAccessible
```

Unlocked Packages (Generally Available) and Second-Generation Managed Packages (Beta)

```
protected Boolean getBypassFLS() {
    return bypassFLS;
}
```

SEE ALSO: Configure Packages

Package Types

Here are the package types for second-generation packaging.

Managed Packages (Beta)

A managed second-generation package is similar to a first-generation managed package. The managed package type is often used by Salesforce partners (ISVs) to develop, distribute, and sell applications to customers. A managed package is fully upgradeable.

Unlocked Packages

Some enterprises want admins to have the flexibility to make changes directly in a production org in response to emergent issues that come up. If you need this flexibility, you can create "unlocked" packages. However, your development team still controls the package. A package upgrade overwrites the changes made directly in the production org.

() Important: Because developers create these packages, a new package version overwrites any changes made directly in a production org. Admins must communicate changes back to the development team who can update the package source.

If you create an unlocked package, you can change or delete any component in the installed package from the installed org. For example, you can:

- Change the data type of a custom field from number to text.
- Change the markup on a Visualforce page.
- Modify a workflow rule.
- Remove permissions in a permission set.
- Change the description of a custom object.
- Delete a custom field.
- Delete a task associated with a workflow rule.

Some components are "hard-deleted." When you delete the component from the package and install the new package version, components are deleted from the install org.

Best Practices for Second-Generation Packages

We suggest that you follow these best practices when working with second-generation packages.

- We strongly recommend that you work with only one Dev Hub, unless you have a strong use case for concurrent use of multiple Dev Hubs.
- A specific Dev Hub org owns every second-generation package that you create. If the Dev Hub org associated with a package expires or is deleted, its packages no longer work.

• Take care when you decide how to utilize namespaces. For most customers, we recommend that you work with a single namespace to avoid unnecessary complexity in how you manage components. If you're test-driving second-generation packages, use a test namespace. Use real namespaces only when you're ready to embark on a development path headed for release in a production org.

Note: You can't install a no-namespace, unlocked package into any org with a namespace (for example, a scratch org with a namespace or a first-generation packaging org).

- Link your namespace with only one Dev Hub org. It's possible to link a namespace with more than one Dev Hub org. However, some packaging features such as keywords aren't designed to work with a namespace linked in this way.
- Include the --tag option when you use the package:version:create and package:version:update commands. This option helps you keep your version-control system tags in sync with specific package versions.
- Add the package version name and description to your sfdx-project.json file to store info about specific package versions. You can also update the name or description of an existing package version using the package:version:update command.
- Use user-friendly aliases for packaging IDs, and start using those aliases in your Salesforce DX project file and when running CLI packaging commands. The sfdx-project.json includes a section for creating package aliases. The CLI maps the OHo or O4t IDs to the package names or aliases when creating packages or package versions, and automatically updates the Salesforce DX project file.

Package IDs

When running Salesforce CLI commands, you can identify packages and package versions by their aliases or package IDs. A package ID is a unique identifier for packages and package versions.

When you create a package or package version, the CLI creates an alias in the sfdx-project.json file that references the unique package ID.

Note: As a shortcut, the documentation sometimes refers to an ID by its three-character prefix. For example, a package version ID always starts with 04t.

Each package has two IDs. You interact with one ID when you update the package. The other ID is what you and your customers use when you install the package. At the command line, you also see IDs for things like package members (a component in a package) and requests (like a package version create request).

Here are the most commonly used IDs.

ID Example	Short ID Name	Description
04t6A0000004eytQAA	Subscriber Package Version ID	Use this ID to install a package version. Returned by force:package:version:create.
0Hoxx0000000CqCAI	Package ID	Use this ID on the command line to create a package version. Or enter it into the sfdx-project.json file and use the directory name. Generated by force:package:create.
05ixx0000000DZAAY	Package Version ID	Returned by force:package:version:create. Use this ID to specify ancestry among package versions and for promoting a

ID Example	Short ID Name	Description
		<pre>package version preleased using force:package:version:promote.</pre>
08cxx00000000BEAAY	Version Creation Request ID	ID for a specific request to create a package version such as force:package:version:create:get

Configure Packages

You include an entry in the Salesforce DX project configuration file for each package to specify its alias, version details, dependencies, features, and org preferences. From the command line, you can also set or change options, such as specify an installation key, update the package name, or add a description.

Project Configuration File for Packages

The project configuration file is a blueprint for your project and for the outline of a package. The settings in the file determine the package attributes and package contents.

Keywords

A keyword is a variable that you can use to specify a package version number.

Package Installation Key

To ensure the security of the metadata in your package, you must specify an installation key when creating a package version. Package creators provide the key to authorized subscribers so they can install the package. Package users provide the key during installation, whether installing the package from the CLI or from a browser. An installation key is required as the first step during installation, ensuring that no package information, like the name and components, is disclosed until the correct installation key is supplied. The installer UI and API changes are effective in the Winter '18 release. Changes to the Salesforce CLI will be announced in a future release.

Extract Dependency Information from Unlocked Packages

For an installed unlocked package, you can now run a simple SOQL query to extract its dependency information. You can also create a script to automate the installation of unlocked packages with dependencies.

Project Configuration File for Packages

The project configuration file is a blueprint for your project and for the outline of a package. The settings in the file determine the package attributes and package contents.

Here are the parameters you can specify in the project definition file.

Name	Required?	Default if Not Specified
path	Yes	If you don't specify a path, the Salesforce CLI uses a placeholder when you create a package.
default	No	true If you have specified more than one path, include this parameter for the default path to indicate which is the default package directory.

Unlocked Packages (Generally Available) and Second-Generation Managed Packages (Beta)

Name	Required?	Default if Not Specified		
package	Yes	The package name you specified when creating the package.		
versionName	No	If not specified, the CLI uses versionNumber as the version name.		
versionDescription	No	None.		
versionNumber	Yes	None. Version numbers are formatted as major.minor.patch.t For example, 1.2.1.8. A subscriber can upgrade a managed package only if the upg has an ancestor that was previously installed in the subscriber When you create a managed package version, you specify th ancestorID or ancestorVersion.		
dependencies	No	None. Specify the dependencies on other packages. To specify dependencies for 2GP within the same Dev Hub, use either the package version alias or a combination of the package name and the version number.		
		<pre>"dependencies": [{ "package": "MyPackageName@0.1.0.1" }]</pre>		
		<pre>"dependencies": [{ "package": "MyPackageName", "versionNumber": "0.1.0.LATEST" }]</pre>		
		To specify dependencies for 2GP outside of the Dev Hub and for first-generation packages within or outside of the current Dev Hub, use:		
		<pre>"dependencies": [{ "package": "OtherOrgPackage@1.2.0" }]</pre>		
		Note: You can use the LATEST keyword for the version number to set the dependency.		
		To denote dependencies with package IDs instead of package aliases, use:		
		• The OHO ID if you specify the package ID along with the version number		
		• The 04t ID if you specify only the package ID		

Name	Required?	Default if Not Specified		
		If the package has more than one dependency, provide a comma-separated list of packages in the order of installation. For example, if a package depends on the package Expense Manager - Util, which in turn depends on the package External Apex Library, the package dependencies are:		
		<pre>"dependencies": [{ "package": "Expense Manager - Util", "versionNumber": "4.7.0.LATEST" }, { "package": "External Apex Library - 1.0.0.4" }] For information on extracting dependency information from unlocked packages, see Extract Dependency Information from Unlocked Packages</pre>		
ancestorID ancestorVersion	No	None. Use the ancestor that's the immediate parent of the version that you're creating. You can specify either ancestorID or ancestorVersion. The package version ID to supply starts with "05i". You can also specify the ancestor version using the format major.minor.patch.build.		
definitionFile	No	<pre>Reference an external .json file to specify the features and org preferences required for the metadata of your package, such as the scratch org definition. "definitionFile": "config/project-scratch-def.json",</pre>		
features	No	None. You can use all features supported by Salesforce DX. We recommend that you specify features in an external .json file, such as the scratch org definition.		
orgPreferences	No	None. You can use all Salesforce org preferences supported by Salesforce DX, such as Chatter and Communities. We recommend that you specify org preferences in an external .json file, such as the scratch org definition.		
packageAliases	Yes	The Salesforce CLI updates this file with the aliases when you create a package or package version. You can also manually update this section for existing packages or package versions. You can use the alias instead of the cryptic package ID when running CLI force:package commands.		

The Salesforce DX project definition file is a JSON file is located in the root directory of your project. Here's what the parameters in packageDirectories look like.

```
{
   "namespace": "",
   "sfdcLoginUrl": "https://login.salesforce.com",
   "sourceApiVersion": "43.0",
   "packageDirectories": [
      {
         "path": "util",
         "default": true,
         "package": "Expense Manager - Util",
         "versionName": "Spring '18",
         "versionDescription": "Welcome to Spring 2018 Release of Expense Manager Util
Package",
         "versionNumber": "4.7.0.NEXT",
         "definitionFile": "config/scratch-org-def.json"
      },
      {
         "path": "exp-core",
         "default": false,
         "package": "Expense Manager",
         "versionName": "v 3.2",
         "versionDescription": "Spring 2018 Release",
         "versionNumber": "3.2.0.NEXT",
         "definitionFile": "config/scratch-org-def.json",
         "dependencies": [
            {
               "package": "Expense Manager - Util",
               "versionNumber": "4.7.0.LATEST"
            },
            {
               "package" : "External Apex Library - 1.0.0.4"
            }
         ]
      }
   ],
   "packageAliases": {
      "Expense Manager - Util": "OHoB0000004CFpKAM",
      "External Apex Library - 1.0.0.4": "04tB0000000IB1EIAW",
      "Expense Manager": "OHoB0000004CFuKAM"}
}
```

What If I Don't Want My Salesforce DX Project Automatically Updated?

In some circumstances, automatic updates to the sfdx-project.json file aren't desirable. When more control is required, use these environment variables to suppress automatic updates to the project file.

For This Command	Set This Environment Variable to True	
force:package:create	SFDX_PROJECT_AUTOUPDATE_DISABLE_FOR_PACKAGE_CREATE	
force:package:version:create	SFDX_PROJECT_AUTOUPDATE_DISABLE_FOR_PACKAGE_VERSION_CREATE	

Keywords

A keyword is a variable that you can use to specify a package version number.

You can use two different keywords to automatically increment the value of the package build numbers and set the package dependency to the latest version.

Use the NEXT keyword to increment the build number to the next available for the package.

```
"versionNumber": "1.2.0.NEXT"
```

Use the LATEST keyword in the version number to assign the latest version of the package dependency when you create a package version.

```
"dependencies": [
    {
        "package": "MyPackageName",
        "versionNumber": "0.1.0.LATEST"
    }
]
```

Package Installation Key

To ensure the security of the metadata in your package, you must specify an installation key when creating a package version. Package creators provide the key to authorized subscribers so they can install the package. Package users provide the key during installation, whether installing the package from the CLI or from a browser. An installation key is required as the first step during installation, ensuring that no package information, like the name and components, is disclosed until the correct installation key is supplied. The installer UI and API changes are effective in the Winter '18 release. Changes to the Salesforce CLI will be announced in a future release.

To set the installation key, add the --installationkey parameter to the command when you create the package version. This command creates a package and protects it with the Open Sesame installation key.

```
sfdx force:package:version:create --package "Expense Manager" --directory common \
--tag 'Release 1.0.0' --installationkey "Open Sesame"
```

You must supply the installation key when you install the package version in the target org.

sfdx force:package:install --package "Expense Manager" --installationkey "Open Sesame"

Change the Installation Key for an Existing Package Version

You can change the installation key for an existing package version with the force:package:version:update command.

```
sfdx force:package:version:update --package "Expense Manager" --installationkey "Open
Sesame"
```

Create a Package Version Without an Installation Key

If you don't require security measures to protect your package metadata, you can create a package version without an installation key.

```
sfdx force:package:version:create --package "Expense Manager" --directory common \
--tag 'Release 1.0.0' --installationkeybypass
```

Extract Dependency Information from Unlocked Packages

For an installed unlocked package, you can now run a simple SOQL query to extract its dependency information. You can also create a script to automate the installation of unlocked packages with dependencies.

The SubscriberPackageVersion Tooling API object now provides dependency information. Using a SOQL query on SubscriberPackageVersion, you can identify the packages on which your unlocked package has a dependency. You can get the (04t) IDs and the correct install order for those packages.

{

💿 Example: Package B has a dependency on package A. Package D depends on packages B and C. Here's a sample sfdx-project.json that you would have specified while creating a package version. Package D dependencies are noted

as packages A, B, and C.

```
"packageDirectories": [
    {
        "path": "pkg-a-workspace",
        "package": "pkgA",
        "versionName": "ver 4.9",
        "versionNumber": "4.9.0.NEXT",
        "default": true
    },
    {
        "path": "pkg-b-workspace",
        "package": "pkgB",
        "versionName": "ver 3.17",
        "versionNumber": "3.17.0.NEXT",
        "default": false,
        "dependencies": [
            {
                "package": "pkgA",
                "versionNumber": "3.3.0.LATEST"
            }
        ]
    },
    {
        "path": "pkg-c-workspace",
        "package": "pkgC",
        "versionName": "ver 2.1",
        "versionNumber": "2.1.0.NEXT",
        "default": false
    },
    {
        "path": "pkg-d-workspace",
        "package": "pkgD",
        "versionName": "ver 1.1",
        "versionNumber": "1.1.0.NEXT",
        "default": false,
        "dependencies": [
            {
                "package": "pkgA",
                "versionNumber": "3.3.0.LATEST"
            },
            {
                "package": "pkgB",
```

```
"versionNumber": "3.12.0.LATEST"
                },
                {
                    "package": "pkgC",
                    "versionNumber": "2.1.0.LATEST"
                }
            ]
        }
   ],
    "namespace": "",
    "sfdcLoginUrl": "https://login.salesforce.com",
    "sourceApiVersion": "44.0",
    "packageAliases": {
        "pkgA": "0HoB00000080q6KAE",
        "pkgB": "0HoB00000080qBKAU",
        "pkgC": "0HoB00000080qGKAU",
        "pkgD": "0HoB00000080qGKAQ"
    }
}
```

Before installing pkgD (with ID=04txx00000082hAAA), use this SOQL query to determine its dependencies. The username is typically the target subscriber org where the unlocked package is to be installed.

```
sfdx force:data:soql:query -u {USERNAME} -t
-q "SELECT Dependencies FROM SubscriberPackageVersion
WHERE Id='04txx00000082hAAA'" --json
```

You see this output when you run the query, with the (04t) IDs for pkgA, pkgB, and pkgC in that order.

```
"Dependencies":{"Ids":[
    {"subscriberPackageVersionId":"04txx00000080vAAA"},
    {"subscriberPackageVersionId":"04txx00000082XAAQ"},
    {"subscriberPackageVersionId":"04txx0000000AiGAAU"}]}
```

SEE ALSO:

Sample Script for Installing Packages with Dependencies

Create a Package

A package is a top-level container that holds important details about the app or package: the package name, description, and associated namespace.

You supply the package details in the package descriptor section of your sfdx-project.json project configuration file. You can associate multiple second-generation packages with a single Dev Hub org. Unlike first-generation managed packages, a second-generation package has no packaging org.

Each package can have many versions. Because you can create multiple second-generation packages with a single org, you can use the Salesforce CLI to see a list of all packages associated with your Dev Hub org.

It's helpful to review the list before you create a package so you can see previous versions and confirm package details like the status or ID.

Authorize your Dev Hub org, and run the force:package:list command.

sfdx force:package:list						
NAMESPACE PREFIX	NAME	ID	DESCRIPTION	PACKAGE TYPE	PACKAGE ALIAS	
acme	exp-mgr exp-mgr-util	0Но 0Но	Exp Manager Exp Manager Utils	Unlocked Unlocked	exp-mgr exp-mgr-util	

Generate the Package

When you're ready to test or share your package, use the force:package:create command to create a package. The package can be a base package or an extension package that depends on an existing package.

Generate a Package Version

A package version is a fixed snapshot of the package contents and related metadata. The package version lets you manage changes and track what's different each time you release or deploy a specific set of changes.

Package Ancestors

The package ancestor attribute specifies the version branch to associate with a new package version.

Release a Second-Generation Package

During the development cycle, you can iterate a package version until it's ready for release. You can complete the release process from the command line using the Salesforce CLI.

Update a Package Version

You can update most properties of a package version from the command line. For example, you can change the package name or description. One important exception is that you can't change the release status.

View Package Details

The Dev Hub org can own multiple second-generation packages.

Generate the Package

When you're ready to test or share your package, use the force:package:create command to create a package. The package can be a base package or an extension package that depends on an existing package.

You specify the package namespace in the sfdx-project.json file, along with other package properties.

To create the package, change to the project directory. The name becomes the package alias, which is automatically added to the project file.

```
sfdx force:package:create --name "My Awesome App" --description "My Package" \
--packagetype Unlocked
```

The output is similar to this example.

```
Successfully created a second-generation package. 0HoB0000004CXHJA2
=== Ids
NAME VALUE
Package Id 0HoB0000004CXHJA2
```

Update the Package

To update the name or description of an existing package, use this command.

```
sfdx force:package:update --package "Expense Manager" --name "New Name" \
--description "New Description"
```

Note: You can't change the package namespace after you create the package.

Generate a Package Version

A package version is a fixed snapshot of the package contents and related metadata. The package version lets you manage changes and track what's different each time you release or deploy a specific set of changes.

Before you create a package version, first specify package details, such as the package ID, ancestors, dependencies, and major and minor version numbers, in the sfdx-project.json file. Verify that the metadata you want to change or add in the new package version is located in the package's main directory.

How Many Package Versions Can I Create Per Day?

Run this command to see how many package versions you can create per day and how many you have remaining.

sfdx force:limits:api:display

Look for the PackageVersionCreates entry.

NAME	REMAINING	MAXIMUM
PackageVersionCreates	50	50

Create a Package Version

Create the package version with this command. Specify the package alias or ID (0Ho). You can also include a scratch definition file that contains a list of features and org preferences that the metadata of the package version depends on.

```
sfdx force:package:version:create --package "Expense Manager" \
--definitionfile config/project-scratch-def.json --wait 10
```

Note: When creating a package version, specify a --wait time to run the command in non-asynchronous mode. If the package version is created within that time, the sfdx-project.json file is automatically updated. If not, you must manually edit the project file.

It can be a long-running process to create a package version, depending on the package size and other variables. You can easily view the status and monitor progress.

```
sfdx force:package:version:create:report --packagecreaterequestid 08cxx0000000YDAAY
```

The output shows details about the request.

```
=== Package Version Create Request
NAME VALUE
```

```
Version Create Request Id
```

08cB0000004CBxIAM

Unlocked Packages (Generally Available) and Second-Generation Managed Packages (Beta)

```
StatusInProgressPackage IdOHoB000004C9hKAEPackage Version Id05iB000000CaaNIASSubscriber Package Version Id04tB00000NOimIAGTagTagBranch2018-05-08 09:48Installation URLHttps://login.salesforce.com/packaging/installPackage.apexp?p0=04tB00000NOimIAG
```

You can find the request ID (08c) in the initial output of force:package:version:create.

Depending on the size of the package and other variables, the create request can take several minutes. When you have more than one pending request to create package versions, you can view a list of all requests with this command.

sfdx force:package:version:create:list --createdlastdays 0

Details for each request display as shown here (IDs and labels truncated).

```
=== Package Version Create Requests [3]
ID STATUS PACKAGE2 ID PKG2 VERSION ID SUB PKG2 VER ID TAG BRANCH CREATED DATE ===
08c... Error 0Ho...
08c... Success 0Ho... 05i... 04t...
2017-06-22 12:07
08c... Success 0Ho... 05i... 04t...
```

Package Ancestors

The package ancestor attribute specifies the version branch to associate with a new package version.

If you're familiar with first-generation managed packages, you probably know that they use linear package versions. Each package has a single parent in a single branch. For second-generation (2GP) managed packages, you have more flexibility to implement a tree structure of inheritance.

🗹 Note: Package inheritance can be specified only for 2GP managed packages, not for unlocked packages.

Specify the ancestor attribute in the sfdx-project.json file. Use the ancestor that's the immediate parent of the version you're creating. You can specify either the ancestorID or with the ancestorVersion. The package version ID you supply starts with "05i".

"ancestorId" :"05iB0000004CIeIAM",

Note: When you create a scratch org, any ancestors defined for a package version that you include in the sfdx-project.json file are automatically added to the scratch org. You can exclude the ancestors by using the --noancestors option when you create a scratch org with force:org:create.

Release a Second-Generation Package

During the development cycle, you can iterate a package version until it's ready for release. You can complete the release process from the command line using the Salesforce CLI.

With this command, you promote your beta package versions to released. You can run this command only once for each package version number, and you can't undo the change to the package status.

A version number uses the format major.minor.patch.build. When you promote a package version, you can't promote the same package again unless you increment the minor or major number. For example, if you created and promoted package 1.0.0.2, you can create

packages 1.0.0.3, 1.0.0.4, and so on using the force:package:version:create command. However, you can't promote more packages with the 1.0.0 scheme. To promote another package, create a new package with an incremented major or minor version number.

To promote packages, you must enable the **Promote a package version to released** setting for the user profile. We recommend that you create a permission set enabling the **Promote a package version to released** system permission, and then assign the permission set to the appropriate user profiles.

For managed packages, after you promote a package version, you can't change some component attributes of its metadata in subsequent package versions. This restriction doesn't apply to unlocked packages, where you can make changes to subsequent package versions.

When you're ready to release, use force:package:version:promote.

sfdx force:package:version:promote --package "Expense Manager"

If the command is successful, a confirmation message appears.

Successfully updated the package version. ID: 05ixx000000DZAAY.

After the update succeeds, view the package details.

sfdx force:package:version:report --package "Expense Manager-1.0.0.5"

Confirm that the value of the Released property is true.

=== Package Version	
NAME	VALUE
Name	ver 1.0
Alias	Expense Manager-1.0.0.5
Package Version Id	05iB000000CaahIAC
Package Id	0HoB000000CabmKAC
Subscriber Package Version Id	04tB000000NPbBIAW
Version	1.0.0.5
Description	update version
Branch	
Tag	
Released	true
Created Date	2018-05-08 09:48
Installation URL	
https://login.salesforce.com/pa	ackaging/installPackage.apexp?p0=04tB0000000NPbBIAW

Update a Package Version

You can update most properties of a package version from the command line. For example, you can change the package name or description. One important exception is that you can't change the release status.

View Package Details

The Dev Hub org can own multiple second-generation packages.

To display a list of all second-generation packages in the Dev Hub org, use this command.

```
sfdx force:package:list --targetdevhubusername jdev@example.com
```

You can view the namespace, package name, ID, and other details in the output.

NAMESPACE PREFIX	NAME	ID	DESCRIPTION	PACKAGE TYPE	PACKAGE ALIAS
acme	exp-mgr exp-mgr-util	ОНо ОНо	Exp Manager Exp Manager Utils	Unlocked Unlocked	exp-mgr exp-mgr-util

To list all second-generation package versions in the Dev Hub org, use this command.

```
sfdx force:package:version:list --createdlastdays 0 --verbose \
--orderby PatchVersion
```

Include optional parameters to filter the list results based on the modification date, creation date, and to order by specific fields or package IDs. To limit the details, use --concise. To show expanded details, use --verbose.

Install a Package

Install second-generation packages using the CLI or the Salesforce browser. You can install package versions in a scratch org, sandbox org, DE org, or production org, depending on the package type.

Install Packages with the CLI

If you're working with the Salesforce CLI, you can use the force:package:install command to install packages in a scratch org or target subscriber org.

Install Packages from a URL

Install second-generation packages from the CLI or from a browser, similar to how you install classic managed packages.

Upgrade a Package Version

Are you introducing metadata changes to an existing package? You can use the CLI to upgrade one package version to another.

Sample Script for Installing Packages with Dependencies

Use this sample script as a basis to create your own script to install packages with dependencies. This script contains a query that finds dependent packages and installs them in the correct dependency order.

Install Packages with the CLI

If you're working with the Salesforce CLI, you can use the force:package:install command to install packages in a scratch org or target subscriber org.

Before you install a package to a scratch org, run this command to list all the packages and locate the ID or package alias.

```
sfdx force:package:version:list
```

Identify the version you want to install. Enter this command, supplying the package alias or package ID (starts with 04t).

```
sfdx force:package:install --package "Expense Manager@1.2.0-12" --targetusername
jdoe@example.com
```

If you've already set the scratch org with a default username, enter just the package version ID.

```
sfdx force:package:install --package "Expense Manager@1.2.0-12"
```



Note: If you've defined an alias (with the -a parameter), you can specify the alias instead of the username for --targetusername.

The CLI displays status messages regarding the installation.

```
Waiting for the subscriber package version install request to get processed. Status =
InProgress Successfully installed the subscriber package version: 04txx000000FIuAAM.
```

Control Package Installation Timeouts

When you issue a force:package:install command, it takes a few minutes for a package version to become available in the target org and for installation to complete. To allow sufficient time for a successful install, use these parameters that represent mutually exclusive timers.

--publishwait defines the maximum number of minutes that the command waits for the Package Version to be available in the target org. The default is 0. If the package is not available in the target org in this time frame, the install is terminated.

Setting --publishwait is useful when you create a new package version and then immediately try to install it to target orgs.



Vote: If --publishwait is set to 0, the package installation immediately fails, unless the package version is already available in the target org.

--wait defines the maximum number of minutes that the command waits for the installation to complete after the package is available. The default is 0. When the --wait interval ends, the install command completes, but the installation continues until it either fails or succeeds. You can poll the status of the installation using sfdx force:package:install:report.

Note: The --wait timer takes effect after the time specified by --publishwait has elapsed. If the --publishwait interval times out before the package is available in the target org, the --wait interval never starts.

For example, consider a package called Expense Manager that takes five minutes to become available on the target org, and 11 minutes to install. The following command has publishwait set to three minutes and wait set to 10 minutes. Because Expense Manager requires more time than the set publishwait interval, the installation is aborted at the end of the three minute publishwait interval.

```
sfdx force:package:install --package "Expense Manager@1.2.0-12" --publishwait 3 --wait 10
```

The following command has publishwait set to six minutes and wait set to 10 minutes. If not already available, Expense Manager takes five minutes to become available on the target org. The clock then starts ticking for the 10 minute wait time. At the end of 10 minutes, the command completes because the wait time interval has elapsed, although the installation is not yet complete. At this point, package:install:report indicates that the installation is in progress. After one more minute, the installation completes and package:install:report indicates a successful installation.

sfdx force:package:install --package "Expense Manager@1.2.0-12" --publishwait 6 --wait 10

Install Packages from a URL

Install second-generation packages from the CLI or from a browser, similar to how you install classic managed packages.

If you create packages from the CLI, you can derive an installation URL for the package by adding the subscriber package ID to your Dev Hub URL. You can use this URL to test different deployment or installation scenarios.

For example, if the package version has the subscriber package ID, 04tB00000009oZ3JBI, add the ID as the value of apvld.

https://my-domain.lightning.force.com/packagingSetupUI/ipLanding.app?apvId=04tB00000009oZ3JBI

Anyone with the URL and a valid login to a Salesforce org can install the package.

To install the package:

- 1. In a browser, go to the installation URL.
- 2. Enter your username and password for the Salesforce org in which you want to install the package, and then click Login.
- 3. If the package is protected by an installation key, enter the installation key you received from the publisher.
- 4. For a default installation, click Install.

A message describes the progress. You receive a confirmation message when the installation is complete.

For more information about installing packages and custom installation options, see the ISV force Guide and Salesforce Help.

Upgrade a Package Version

Are you introducing metadata changes to an existing package? You can use the CLI to upgrade one package version to another.

When you perform a package upgrade, here's what to expect for metadata changes.

- Metadata introduced in the new version is installed as part of the upgrade.
- If an upgraded component has the same API name as a component already in the target org, the component is overwritten with the changes.
- If a component in the upgrade was previously deleted from the target org, the component is re-created during the upgrade.
- Metadata that was removed in the new package version is also removed from the target org as part of the upgrade. Removed
 metadata is metadata not included in the current package version install, but present in the previous package version installed in
 the target org. If metadata is removed before the upgrade occurs, the upgrade proceeds normally. Some examples where metadata
 is deprecated and not deleted are:
 - User-entered data in custom objects and fields are deprecated and not deleted. Admins can export such data if necessary.
 - An object such as an Apex class is deprecated and not deleted if it is referenced in a Lightning component that is part of the package.
- In API version 45.0 and later (salesforcedx plug-in for Salesforce CLI version 45.0.9 or later), you can specify what happens to removed metadata during package upgrade. Use the force:package:install command's -t | --upgradetype parameter, specifying one of these values:
 - DeprecateOnly specifies that all removed components must be marked deprecated. The removed metadata exists in the target org after package upgrade, but is shown in the UI as deprecated from the package. This option is useful when migrating metadata from one package to another.
 - Mixed (the default) specifies that some removed components are deleted, and other components are marked deprecated. For more information on hard-deleted components, see Metadata Coverage.

Sample Script for Installing Packages with Dependencies

Use this sample script as a basis to create your own script to install packages with dependencies. This script contains a query that finds dependent packages and installs them in the correct dependency order.

```
Sample Script
🕜 Note: Be sure to replace the package version ID and scratch org user name with your own specific details.
#!/bin/bash
# The execution of this script stops if a command or pipeline has an error.
# For example, failure to install a dependent package will cause the script
# to stop execution.
set -e
# Specify a package version id (starts with 04t)
# If you know the package alias but not the id, use force:package:version:list to find it.
PACKAGE=04tB000000NmnHIAS
# Specify the user name of the subscriber org.
USER NAME=test-bvdfz3m9tqdf@example.com
# Specify the timeout in minutes for package installation.
WAIT TIME=15
echo "Retrieving dependencies for package Id: "$PACKAGE
# Execute soql query to retrieve package dependencies in json format.
RESULT JSON=`sfdx force:data:soql:query -u $USER NAME -t -q "SELECT Dependencies FROM
SubscriberPackageVersion WHERE Id='$PACKAGE'" -- json`
# Parse the json string using python to test whether the result json contains a list of
ids or not.
DEPENDENCIES=`echo $RESULT JSON | python -c 'import sys, json; print
json.load(sys.stdin)["result"]["records"][0]["Dependencies"]'`
# If the parsed dependencies is None, the package has no dependencies. Otherwise, parse
the result into a list of ids.
# Then loop through the ids to install each of the dependent packages.
```

Unlocked Packages (Generally Available) and Second-Generation Managed Packages (Beta)

```
if [[ "$DEPENDENCIES" != 'None' ]]; then
    DEPENDENCIES=`echo $RESULT JSON | python -c '
import sys, json
ids = json.load(sys.stdin)["result"]["records"][0]["Dependencies"]["ids"]
dependencies = []
for id in ids:
   dependencies.append(id["subscriberPackageVersionId"])
print " ".join(dependencies)
• `
   echo "The package you are installing depends on these packages (in correct dependency
order): "$DEPENDENCIES
   for id in $DEPENDENCIES
   do
       echo "Installing dependent package: "$id
       sfdx force:package:install --package $id -u $USER NAME -w $WAIT TIME --publishwait
10
   done
else
   echo "The package has no dependencies"
fi
# After processing the dependencies, proceed to install the specified package.
echo "Installing package: "$PACKAGE
sfdx force:package:install --package $PACKAGE -u $USER NAME -w $WAIT TIME --publishwait
10
exit 0;
```

Migrate Deprecated Metadata from Unlocked Packages

You can deprecate metadata in an unlocked package, move that metadata to a new package, and then install the new package in your production org.

As you start creating more unlocked packages, you might need to refactor your package and move metadata from one unlocked package to another unlocked package. Before the Winter '18 release, you had to use the UI to manually remove the installed metadata from the original package because deprecate-upon-upgrade metadata was still associated with the package. You couldn't install a new package containing that metadata without the manual step.

To move production metadata from package A to package B, follow these steps.

- 1. Identify the metadata to be moved from package A to package B.
- 2. Remove the metadata from package A, create a version, and release the package.
- 3. Add the metadata to package B, create a version, and release the package.
- 4. In your production org, upgrade package A.
- 5. In your production org, install package B.

Your metadata is now a part of package B in your production org.

Uninstall a Package

You can uninstall a package from a subscriber org using the CLI or from Salesforce Setup. When you uninstall second-generation packages, all components in the package are deleted from the org.

To use the CLI to uninstall a second-generation package from the target org, authorize the Dev Hub org and run this command.

sfdx force:package:uninstall --package "Expense Manager"

You can also uninstall a package from the web browser. Open the Salesforce org where you installed the package.

sfdx force:org:open -u me@my.org

Then uninstall the package in the same way you uninstall first-generation managed packages.

- 1. From Setup, enter Installed Packages in the Quick Find box, then select Installed Packages.
- 2. Click Uninstall next to the package that you want to remove.
- 3. Select Yes, I want to uninstall and click Uninstall.
- 4. After an uninstall, Salesforce automatically creates an export file containing the package data, associated notes, and any attachments. When the uninstall is complete, Salesforce sends an email containing a link to the export file to the user performing the uninstall. The export file and related notes and attachments are listed below the list of installed packages. We recommend storing the file elsewhere because it's available for only two days after the uninstall completes, then it's deleted from the server.

Tip: If you reinstall the package later and want to reimport the package data, see Importing Package Data.

SEE ALSO:

Salesforce CLI Command Reference

CHAPTER 12 Continuous Integration

In this chapter ...

- Continuous
 Integration Using
 CircleCl
- Continuous Integration Using Jenkins
- Continuous
 Integration with
 Travis Cl
- Sample CI Repos for Org Development Model
- Sample CI Repos for Package Development Model

Continuous integration (CI) is a software development practice in which developers regularly integrate their code changes into a source code repository. To ensure that the new code does not introduce bugs, automated builds and tests run before or after developers check in their changes.

Many third-party CI tools are available for you to choose from. Salesforce DX easily integrates into these tools so that you can set up continuous integration for your Salesforce applications.

SEE ALSO:

Trailhead: Build an Automated CI/CD Pipeline with GitLab

Continuous Integration Using CircleCI

CircleCl is a commonly used integration tool that integrates with your existing version control system to push incremental updates to the environments you specify. CircleCl can be used as a cloud-based or on-premise tool. These instructions demonstrate how to use GitHub, CircleCl, and your Dev Hub org for continuous integration.

Configure Your Environment for CircleCl Before integrating your existing CircleCl framework, configure your Dev Hub org and CircleCl project. Connect CircleCl to Your DevHub

Authorize CircleCl to push content to your Dev Hub via a connected app.

SEE ALSO:

CircleCl The sfdx-circleci Github Repo

Configure Your Environment for CircleCI

Before integrating your existing CircleCl framework, configure your Dev Hub org and CircleCl project.

- 1. Set up your GitHub repository with CircleCl. You can follow the sign-up steps on the CircleCl website to access your code on GitHub.
- 2. Install the Salesforce CLI, if you haven't already.
- 3. Follow Authorize an Org Using the JWT-Based Flow for your Dev Hub org, if you haven't already.
- 4. Encrypt your server key.
 - a. First, generate a key and initialization vector (iv) to encrypt your server.key file locally. CircleCl uses the key and vi to decrypt your server key in the build environment.

Run the following command in the directory containing your server.key file. For the *<passphrase>* value, enter a word of your own choosing to create a unique key.

```
openssl enc -aes-256-cbc -k <passphrase> -P -md shal -nosalt
```

The key and iv value display in the output.

key=***24B2 iv =***DA58

- **b.** Note the key and iv values, you need them later.
- c. Encrypt the server.key file using the newly generated key and iv values. Run the following command in the directory containing your server.key file, replacing <key> and <iv> with the values from the previous step.

```
openssl enc -nosalt -aes-256-cbc -in server.key -out server.key.enc -base64 -K <key> -iv <iv>
```

Note: Use the key and iv values only once, and don't use them to encrypt more than the server.key. While you can reuse this pair to encrypt other things, it is considered a security violation to do so.

You generate a new key and iv value every time you run the command in step a. In other words, you can't regenerate the same pair. If you lose these values you must generate new ones and encrypt again.

Next, you'll store the key, iv, and contents of server.key.enc as protected environment variables in the CircleCl UI. These values are considered secret, so take the appropriate precautions to protect them.

Connect CircleCl to Your DevHub

Authorize CircleCl to push content to your Dev Hub via a connected app.

- 1. Make sure that you have the Salesforce CLI installed. Check by running sfdx force --help and confirm that you see the command output. If you don't have it installed, see Install the Salesforce CLI.
- 2. Confirm you can perform a JWT-based authorization from the directory containing your server.key file. Run the following command from the directory containing your server.key (replace <your_consumer_key> and <your_username> values where indicated).

```
sfdx force:auth:jwt:grant --clientid <your_consumer_key> --jwtkeyfile server.key
--username <your username> --setdefaultdevhubusername
```

- 3. Fork the sfdx-circleci repo into your GitHub account using the Fork link at the top of the page.
- 4. Create a local directory for this project and clone your forked repo locally into the new directory. Replace <git_username> with your own GitHub username.

```
git clone https://github.com/<git_username>/sfdx-circleci.git
```

- 5. Retrieve the generated consumer key from your JWT-Based Authorization connected app. From Setup, in the Quick Find box, enter *App*, and then select **App Manager**. Select **View** in the row-menu next to the connected app.
- 6. In the CircleCl UI, you should see a project named sfdx-circleci. In the project settings, store the consumer key in a CircleCl environment variable named HUB_CONSUMER_KEY. For more information, see the CircleCl documentation Setting an Environment Variable in a Project.
- 7. Store the username that you use to access your Dev Hub in a CircleCl environment variable named HUB_SFDX_USER using the CircleCl UI.
- 8. Store the key and iv values from Encrypt Your Server Key in CircleCl environment variables named DECRYPTION_KEYand DECRYPTION_IV, respectively. When you finish setting the environment variables, your project screen should look like the following image.

\bigcirc	~	Э	Updates 💙	Support ~
■	Settings » » sfdx-c	ircleci		View sfdx-circleci »
÷	PROJECT SETTINGS Overview	Environment Variables		
ı.	Org Settings BUILD SETTINGS	Environment Variables for /sfd:	x-circleci Import Variat	les Add Variable
٥	Build Environment Adjust Parallelism	Add environment variables to the job. You can add s placing them in the repository.	ensitive data (e.g. API keys) here, rather than
	Environment Variables	Name	Value	Remove
-	Advanced Settings	DECRYPTION_IV	xxxxDA58	×
	TEST COMMANDS	DECRYPTION_KEY	xxxx24B2	×
Q [°]	Dependency Commands	HUB_CONSUMER_KEY	xxxxue7f	×
	Test Commands	HUB_SFDX_USER	xxxx.com	×

Note: In the directory containing your server.key file, use the command rm server.key to remove the server.key. Never store keys or certificates in a public place.

You're ready to go! Now when you commit and push a change, your change kicks off a CircleCI build.

Contributing to the Repository

If you find any issues or opportunities for improving this repository, fix them! Feel free to contribute to this project, fork this repository, and then change the content. Once you've made your changes, share them back with the community by sending a pull request. See How to send pull requests for more information about contributing to GitHub projects.

Reporting Issues

If you find any issues with this demo that you can't fix, feel free to report them in the issues section of this repository.

Continuous Integration Using Jenkins

Jenkins is an open-source, extensible automation server for implementing continuous integration and continuous delivery. You can easily integrate Salesforce DX into the Jenkins framework to automate testing of Salesforce applications against scratch orgs.

To integrate Jenkins, we assume:

- You are familiar with how Jenkins works. You can configure and use Jenkins in many ways. We focus on integrating Salesforce DX into Jenkins multibranch pipelines.
- The computer on which the Jenkins server is running has access to your version control system and to the repository that contains your Salesforce application.

Configure Your Environment for Jenkins

Before integrating Salesforce DX into your existing Jenkins framework, configure your Jenkins environment.

Jenkinsfile Walkthrough

The sample Jenkinsfile shows how to integrate Salesforce DX into a Jenkins job. The sample uses Jenkins multibranch pipelines. Every Jenkins setup is different. This walkthrough describes one of the ways to automate testing of your Salesforce applications. The walkthrough highlights the Salesforce DX CLI commands to create a scratch org, upload your code, and run your tests.

Sample Jenkinsfile

A Jenkinsfile is a text file that contains the definition of a Jenkins Pipeline. This Jenkinsfile shows how to integrate the Salesforce DX CLI commands to automate testing of your Salesforce applications using scratch orgs.

SEE ALSO:

Jenkins

Pipeline-as-code with Multibranch Workflows in Jenkins

Configure Your Environment for Jenkins

Before integrating Salesforce DX into your existing Jenkins framework, configure your Jenkins environment.

1. In your Dev Hub org, create a connected app as described by the JWT-based authorization flow. This step includes obtaining or creating a private key and digital certificate.

Make note of your consumer key (sometimes called a client ID) when you save the connected app. You need the consumer key to set up your Jenkins environment. Also have available the private key file used to sign the digital certificate.

- 2. On the computer that is running the Jenkins server, do the following.
 - a. Download and install the Salesforce DX CLI.
 - **b.** Store the private key file as a Jenkins Secret File using the Jenkins Admin Credentials interface. Make note of the new entry's ID. You later reference this Credentials entry in your Jenkinsfile.
 - c. Set the following variables in your Jenkins environment.
 - HUB_ORG_DH—The username for the Dev Hub org, such as juliet.capulet@myenvhub.com.
 - SFDC_HOST_DH—The login URL of the Salesforce instance that is hosting the Dev Hub org. The default is https://login.salesforce.com
 - CONNECTED_APP_CONSUMER_KEY_DH—The consumer key that was returned after you created a connected app in your Dev Hub org.
 - JWT_CRED_ID_DH—The credentials ID for the private key file that you stored in the Jenkins Admin Credentials interface.

The names for these environment variables are just suggestions. You can use any name as long as you specify it in the Jenkinsfile.

You can also optionally set the SFDX_AUTOUPDATE_DISABLE variable to true to disable auto-update of the Salesforce CLI. CLI auto-update can interfere with the execution of a Jenkins job.

3. Set up your Salesforce DX project so that you can create a scratch org.

4. (Optional) Install the Custom Tools Plugin into your Jenkins console, and create a custom tool that references the Salesforce CLI. The Jenkins walkthrough assumes that you created a custom tool named toolbelt in the /usr/local/bin directory, which is the directory in which the Salesforce CLI is installed.

```
SEE ALSO:
```

Authorize an Org Using the JWT-Based Flow Salesforce DX Setup Guide Jenkins: Credentials Binding Plugin Project Setup

Jenkinsfile Walkthrough

The sample Jenkinsfile shows how to integrate Salesforce DX into a Jenkins job. The sample uses Jenkins multibranch pipelines. Every Jenkins setup is different. This walkthrough describes one of the ways to automate testing of your Salesforce applications. The walkthrough highlights the Salesforce DX CLI commands to create a scratch org, upload your code, and run your tests.

We assume that you are familiar with the structure of the Jenkinsfile, Jenkins Pipeline DSL, and the Groovy programming language. This walkthrough focuses solely on Salesforce DX information. See the Salesforce DX Command Reference regarding the commands used.

This Salesforce DX workflow most closely corresponds to Jenkinsfile stages.

- Define Variables
- Check Out the Source Code
- Wrap All Stages in a withCredentials Command
- Authorize Your Dev Hub Org and Create a Scratch Org
- Push Source and Assign a Permission Set
- Run Apex Tests
- Delete the Scratch Org

Define Variables

Use the def keyword to define the variables required by the Salesforce DX CLI commands. Assign each variable the corresponding environment variable that you previously set in your Jenkins environment.

```
def HUB_ORG=env.HUB_ORG_DH
def SFDC_HOST = env.SFDC_HOST_DH
def JWT_KEY_CRED_ID = env.JWT_CRED_ID_DH
def CONNECTED_APP_CONSUMER_KEY=env.CONNECTED_APP_CONSUMER_KEY_DH
```

Define the SFDC USERNAME variable, but don't set its value. You do that later.

```
def SFDC_USERNAME
```

Although not required, we assume you've used the Jenkins Global Tool Configuration to create the toolbelt custom tool that points to the CLI installation directory. In your Jenkinsfile, use the tool command to set the value of the toolbelt variable to this custom tool.

```
def toolbelt = tool 'toolbelt'
```

You can now reference the Salesforce CLI executable in the Jenkinsfile using \${toolbelt}/sfdx.

Check Out the Source Code

Before testing your code, get the appropriate version or branch from your version control system (VCS) repository. In this example, we use the checkout scm Jenkins command. We assume that the Jenkins administrator has already configured the environment to access the correct VCS repository and check out the correct branch.

```
stage('checkout source') {
    // when running in multi-branch job, one must issue this command
    checkout scm
}
```

Wrap All Stages in a withCredentials Command

You previously stored the JWT private key file as a Jenkins Secret File using the Credentials interface. Therefore, you must use the withCredentials command in the body of the Jenkinsfile to access the secret file. The withCredentials command lets you name a credential entry, which is then extracted from the credential store and provided to the enclosed code through a variable. When using withCredentials, put all stages within its code block.

This example stores the credential ID for the JWT key file in the variable JWT_KEY_CRED_ID. You defined JWT_KEY_CRED_ID earlier and set it to its corresponding environment variable. The withCredentials command fetches the contents of the secret file from the credential store and places the contents in a temporary location. The location is stored in the variable jwt_key_file. You use the jwt_key_file variable with the force:auth:jwt command to specify the private key securely.

```
withCredentials([file(credentialsId: JWT_KEY_CRED_ID, variable: 'jwt_key_file')]) {
    # all stages will go here
}
```

Authorize Your Dev Hub Org and Create a Scratch Org

The dreamhouse-sfdx example uses one stage to authorize the Dev Hub org and create a scratch org.

```
stage('Create Scratch Org') {
    rc = sh returnStatus: true, script: "${toolbelt}/sfdx force:auth:jwt:grant --clientid
${CONNECTED_APP_CONSUMER_KEY} --username ${HUB_ORG} --jwtkeyfile ${jwt_key_file}
--setdefaultdevhubusername --instanceurl ${SFDC_HOST}"
    if (rc != 0) { error 'hub org authorization failed' }
    // need to pull out assigned username
    rmsg = sh returnStdout: true, script: "${toolbelt}/sfdx force:org:create --definitionfile
    config/project-scratch-def.json --json --setdefaultusername"
    printf rmsg
    def jsonSlurper = new JsonSlurperClassic()
    def robj = jsonSlurper.parseText(rmsg)
    if (robj.status != "ok") { error 'org creation failed: ' + robj.message }
    SFDC_USERNAME=robj.username
    robj = null
}
```

Use the force:auth:jwt:grant CLI command to authorize your Dev Hub org.

You are required to run this step only once, but we suggest you add it to your Jenkinsfile and authorize each time you run the Jenkins job. This way you're always sure that the Jenkins job is not aborted due to lack of authorization. There is typically little harm in authorizing multiple times, although keep in mind that the API call limit for your scratch org's edition still applies.

Use the parameters of the force:auth:jwt:grant command to provide information about the Dev Hub org that you are authorizing. The values for the --clientid, --username, and --instanceurl parameters are the CONNECTED_APP_CONSUMER_KEY, HUB_ORG, and SFDC_HOST environment variables you previously defined, respectively. The value of the --jwtkeyfile parameter is the jwt_key_file variable that you set in the previous section using the withCredentials command. The --setdefaultdevhubusername parameter specifies that this HUB_ORG is the default Dev Hub org for creating scratch orgs.

Use the force:org:create CLI command to create a scratch org. In the example, the CLI command uses the config/project-scratch-def.json file (relative to the project directory) to create the scratch org. The --json parameter specifies that the output be in JSON format. The --setdefaultusername parameter sets the new scratch org as the default.

The Groovy code that parses the JSON output of the force:org:create command extracts the username that was auto-generated as part of the org creation. This username, stored in the SFDC_USERNAME variable, is used with the CLI commands that push source, assign a permission set, and so on.

Push Source and Assign a Permission Set

Let's populate your new scratch org with metadata. This example uses the force:source:push command to upload your source to the org. The source includes all the pieces that make up your Salesforce application: Apex classes and test classes, permission sets, layouts, triggers, custom objects, and so on.

```
stage('Push To Test Org') {
    rc = sh returnStatus: true, script: "${toolbelt}/sfdx force:source:push --targetusername
    ${SFDC_USERNAME}"
    if (rc != 0) {
    error 'push all failed'
    }
    // assign permset
    rc = sh returnStatus: true, script: "${toolbelt}/sfdx force:user:permset:assign
--targetusername ${SFDC_USERNAME} --permsetname DreamHouse"
    if (rc != 0) {
    error 'push all failed'
    }
}
```

Recall the SFDC_USERNAME variable that contains the auto-generated username that was output by the force:org:create command in an earlier stage. The code uses this variable as the argument to the --targetusername parameter to specify the username for the new scratch org.

The force:source:push command pushes all the Salesforce-related files that it finds in your project. Add a .forceignore file to your repository to list the files that you do not want pushed to the org.

After pushing the metadata, the example uses the force:user:permset:assign command to assign a permission set (named DreamHouse) to the SFDC_USERNAME user. The XML file that describes this permission set was uploaded to the org as part of the push.

Run Apex Tests

Now that your source code and test source have been pushed to the scratch org, run the force:apex:test:run command to run Apex tests.

```
stage('Run Apex Test') {
    sh "mkdir -p ${RUN_ARTIFACT_DIR}"
    timeout(time: 120, unit: 'SECONDS') {
        rc = sh returnStatus: true, script: "${toolbelt}/sfdx force:apex:test:run --testlevel
        RunLocalTests --outputdir ${RUN_ARTIFACT_DIR} --resultformat tap --targetusername
        ${SFDC_USERNAME}"
        if (rc != 0) {
        error 'apex test run failed'
        }
      }
}
```

You can specify various parameters to the force:apex:test:run CLI command. In the example:

- The --testlevel RunLocalTests option runs all tests in the scratch org, except tests that originate from installed managed packages. You can also specify RunSpecifiedTests to run only certain Apex tests or suites or RunAllTestsInOrg to run all tests in the org.
- The --outputdir option uses the RUN_ARTIFACT_DIR variable to specify the directory into which the test results are written. Test results are produced in JUnit and JSON formats.
- The --resultformat tap option specifies that the command output is in Test Anything Protocol (TAP) format. The test results
 that are written to a file are still in JUnit and JSON formats.
- The --targetusername option specifies the username for accessing the scratch org (the value in SFDC_USERNAME).

The force:apex:test:run command writes its test results in JUnit format. You can collect the results using industry-standard tools as shown in the following example.

```
stage('collect results') {
    junit keepLongStdio: true, testResults: 'tests/**/*-junit.xml'
}
```

Delete the Scratch Org

Salesforce reserves the right to delete a scratch org a specified number of days after it was created. You can also create a stage in your pipeline that uses force:org:delete to explicitly delete your scratch org when the tests complete. This cleanup ensures better management of your resources.

```
stage('Delete Test Org') {
    timeout(time: 120, unit: 'SECONDS') {
        rc = sh returnStatus: true, script: "${toolbelt}/sfdx force:org:delete
--targetusername ${SFDC_USERNAME} --noprompt"
        if (rc != 0) {
            error 'org deletion request failed'
        }
```

}

SEE ALSO:

Sample Jenkinsfile Pipeline-as-code with Multibranch Workflows in Jenkins TAP: Test Anything Protocol Configure Your Environment for Jenkins Salesforce CLI Command Reference

Sample Jenkinsfile

A Jenkinsfile is a text file that contains the definition of a Jenkins Pipeline. This Jenkinsfile shows how to integrate the Salesforce DX CLI commands to automate testing of your Salesforce applications using scratch orgs.

The Jenkinsfile Walkthrough topic uses this Jenkinsfile as an example.

```
#!groovy
import groovy.json.JsonSlurperClassic
node {
   def BUILD NUMBER=env.BUILD NUMBER
   def RUN ARTIFACT DIR="tests/${BUILD NUMBER}"
   def SFDC USERNAME
   def HUB_ORG=env.HUB_ORG_DH
   def SFDC HOST = env.SFDC HOST DH
   def JWT KEY CRED ID = env.JWT_CRED_ID_DH
   def CONNECTED APP CONSUMER KEY=env.CONNECTED APP CONSUMER KEY DH
   def toolbelt = tool 'toolbelt'
    stage('checkout source') {
       // when running in multi-branch job, one must issue this command
        checkout scm
    }
   withCredentials([file(credentialsId: JWT KEY CRED ID, variable: 'jwt key file')]) {
        stage('Create Scratch Org') {
            rc = sh returnStatus: true, script: "${toolbelt}/sfdx force:auth:jwt:grant
--clientid ${CONNECTED APP CONSUMER KEY} --username ${HUB ORG} --jwtkeyfile ${jwt key file}
--setdefaultdevhubusername --instanceurl ${SFDC HOST}"
            if (rc != 0) { error 'hub org authorization failed' }
            // need to pull out assigned username
            rmsg = sh returnStdout: true, script: "${toolbelt}/sfdx force:org:create
--definitionfile config/project-scratch-def.json --json --setdefaultusername"
            printf rmsg
            def jsonSlurper = new JsonSlurperClassic()
            def robj = jsonSlurper.parseText(rmsq)
```

```
if (robj.status != 0) { error 'org creation failed: ' + robj.message }
            SFDC USERNAME=robj.result.username
            robj = null
        }
        stage('Push To Test Org') {
            rc = sh returnStatus: true, script: "${toolbelt}/sfdx force:source:push
--targetusername ${SFDC USERNAME}"
            if (rc != 0) {
                error 'push failed'
            }
            // assign permset
          rc = sh returnStatus: true, script: "${toolbelt}/sfdx force:user:permset:assign
--targetusername ${SFDC USERNAME} --permsetname DreamHouse"
            if (rc != 0) {
                error 'permset:assign failed'
            }
        }
        stage('Run Apex Test') {
            sh "mkdir -p ${RUN ARTIFACT DIR}"
            timeout(time: 120, unit: 'SECONDS') {
                rc = sh returnStatus: true, script: "${toolbelt}/sfdx force:apex:test:run
--testlevel RunLocalTests --outputdir ${RUN ARTIFACT DIR} --resultformat tap
--targetusername ${SFDC USERNAME}"
                if (rc != 0) {
                    error 'apex test run failed'
                }
            }
        }
        stage('collect results') {
            junit keepLongStdio: true, testResults: 'tests/**/*-junit.xml'
        }
    }
}
```

SEE ALSO:

Jenkinsfile Walkthrough

Continuous Integration with Travis CI

Travis Cl is a cloud-based continuous integration (Cl) service for building and testing software projects hosted on GitHub.

Setting up Salesforce DX with Travis CI is easy. See the sfdx-travisci GitHub sample and the Salesforce DX Trailhead modules to get started.

```
SEE ALSO:
sfdx-travisci Sample GitHub Repo
Travis Cl
```
Sample CI Repos for Org Development Model

Get started quickly with CI by cloning a sample repository from your vendor of choice. Each repo has a sample configuration file and a comprehensive README.md with step-by-step information.

These sample repositories support the org development model. This model uses Salesforce CLI, a source control system, and sandboxes during the application life cycle. To determine if this model is right for you, head over and earn your badge by completing the Org Development Model module.

Vendor	Link to GitHub Repository
AppVeyor	https://github.com/forcedotcom/sfdx-appveyor-org
Bamboo	https://github.com/forcedotcom/sfdx-bamboo-org
Bitbucket	https://github.com/forcedotcom/sfdx-bitbucket-org
CircleCl	https://github.com/forcedotcom/sfdx-circleci-org
GitLab	https://github.com/forcedotcom/sfdx-gitlab-org
Jenkins	https://github.com/forcedotcom/sfdx-jenkins-org
TravisCl	https://github.com/forcedotcom/sfdx-travisci-org

Sample CI Repos for Package Development Model

Get started quickly with CI by cloning a sample repository from your vendor of choice. Each repo has a sample configuration file and a comprehensive README.md with step-by-step information.

These sample repositories support the package development model. This model uses Salesforce CLI, a source control system, scratch orgs for development, and sandboxes for testing and staging. To determine if this model is right for you, head over and earn your badge by completing the Package Development Model module.

Vendor	Link to GitHub Repository
AppVeyor	https://github.com/forcedotcom/sfdx-appveyor-package
Bamboo	https://github.com/forcedotcom/sfdx-bamboo-package
Bitbucket	https://github.com/forcedotcom/sfdx-bitbucket-package
CircleCl	https://github.com/forcedotcom/sfdx-circleci-package
GitLab	https://github.com/forcedotcom/sfdx-gitlab-package
	CI/CD template for Salesforce/Apex apps:
	https://gitlab.com/sfdx/sfdx-cicd-template
Jenkins	https://github.com/forcedotcom/sfdx-jenkins-package

Vendor	Link to GitHub Repository
TravisCl	https://github.com/forcedotcom/sfdx-travisci-package

SEE ALSO:

Trailhead: Build an Automated CI/CD Pipeline with GitLab

CHAPTER 13 Troubleshoot Salesforce DX

In this chapter ...

- CLI Version
 Information
- Run CLI Commands on macOS Sierra (Version 10.12)
- Error: No defaultdevhubusemame org found
- Unable to Work After
 Failed Org
 Authorization
- Error: Lightning Experience-Enabled Custom Domain Is Unavailable

This guide is a work in progress. Log in to the Salesforce Trailblazer Community and let us know if you find a solution that would help other users so that we can incorporate it.

SEE ALSO:

Salesforce Trailblazer Community

CLI Version Information

Use these commands to view version information about Salesforce CLI.

```
sfdx plugins --core// Versions for all installed plugins and sfdx-clisfdx plugins// salesforcedx plugin versionsfdx force --version// Salesforce API version that Salesforce CLI uses
```

Run CLI Commands on macOS Sierra (Version 10.12)

Some users who upgrade to macOS Sierra can't execute CLI commands. This is a general problem and not isolated to Salesforce DX. To resolve the issue, reinstall your Xcode developer tools.

Execute this command in Terminal:

xcode-select --install

If you still can't execute CLI commands, download the **Command Line Tools (macOS sierra) for Xcode 8** package from the Apple Developer website.

SEE ALSO:

Apple Developer Downloads

Stack Overflow: Command Line Tools bash (git) not working - macOS Sierra Final Release Candidate

Error: No defaultdevhubusername org found

Let's say you successfully authorize a Dev Hub org using the --setdefaultdevhubusername parameter. The username associated with the org is your default Dev Hub username. You then successfully create a scratch org without using the --targetdevhubusername parameter.

carge cae vitababername parameter.

But when you try to create a scratch org another time using the same CLI command, you get this error:

Unable to invoke command. name: NoOrgFound message: No defaultdevhubusername org found

What happened?

Answer: You are no longer in the directory where you ran the authorization command. The directory from which you use the --setdefaultdevhubusername parameter matters.

If you run the authorization command from the root of your project directory, the defaultdevhubusername config value is set locally. The value applies only when you run the command from the same project directory. If you change to a different directory and run force:org:create, the local setting of the default Dev Hub org no longer applies and you get an error.

Solve the problem by doing one of the following.

Set defaultdevhubusername globally so that you can run force:org:create from any directory.

sfdx force:config:set defaultdevhubusername=<devhubusername> --global

Run force:org:create from the same project directory where you authorized your Dev Hub org.

• Use the --targetdevhubusername parameter with force:org:create to run it from any directory.

```
sfdx force:org:create --definitionfile <file> --targetdevhubusername <devhubusername>
--setalias my-scratch-org
```

• To check whether you've set configuration values globally or locally, use this command.

sfdx force:config:list

SEE ALSO:

How Salesforce Developer Experience Changes the Way You Work

Unable to Work After Failed Org Authorization

Sometimes you try to authorize a Dev Hub org or a scratch org using the Salesforce CLI or an IDE, but you don't successfully log in to the org. The port remains open for the stray authorization process, and you can't use the CLI or IDE. To proceed, end the process manually.

macOS or Linux

To recover from a failed org authorization on macOS or Linux, use a terminal to kill the process running on port 1717.

1. From a terminal, run:

lsof -i tcp:1717

- 2. In the results, find the ID for the process that's using the port.
- 3. Run:

```
kill -9 <the process ID>
```

Windows

To recover from a failed org authorization on Windows, use the Task Manager to end the Node process.

- 1. Press Ctrl+Alt+Delete, then click Task Manager.
- 2. Select the Process tab.
- 3. Find the process named Node.

🕜 Note: If you're a Node.js developer, you might have several running processes with this name.

4. Select the process that you want to end, and then click End Process.

Error: Lightning Experience-Enabled Custom Domain Is Unavailable

If you create a scratch org with force:org:create, and then immediately try to use it, you sometimes get an error after waiting a few minutes for the command to finish.

For example, if you try to open the new scratch org in a browser with force:org:open, you might get this error:

Waiting to resolve the Lightning Experience-enabled custom domain... ERROR running force:org:open: The Lightning Experience-enabled custom domain is unavailable.

The error occurs because it takes a few minutes for the Lightning Experience-enabled custom domain to internally resolve.

When using the CLI interactively, wait a few more minutes and run the command again. In a CI environment, however, you can avoid the error altogether by changing how long the CLI itself waits.

By default, the CLI waits 240 seconds (4 minutes) for the custom domain to become available. You can configure the CLI to wait longer by setting the SFDX_DOMAIN_RETRY environment variable to the number of seconds you want it to wait. For example, to wait 5 minutes (300 seconds):

export SFDX_DOMAIN_RETRY=300

If you want the CLI to bypass the custom domain check entirely, set SFDX_DOMAIN_RETRY to 0.

CHAPTER 14 Limitations for Salesforce DX

Here are some known issues you could run into while using Salesforce DX.

For the latest known issues, visit the Trailblazer Community's Known Issues page.

Salesforce CLI

Authorization Fails If Using auth:web:login with Client Secret

Description: If you run force:auth:web:login with a client ID and client secret, you can't use Salesforce CLI to issue commands to the scratch org because the authorization file isn't properly created.

Workaround: Use the web-based flow without client ID and client secret, or use the JWT-based flow to authorize to the org. See Authorization in the *Salesforce DX Developer Guide* for instructions on Dev Hub and scratch org authorization methods.

Windows Defender Suspends CLI Installation

Description: When you are installing the Salesforce CLI on Windows, you see a Windows Defender warning. This message is expected because we updated the installer's code signing certificate.

Wind	ows protected your PC	×
Windows starting. R	Defender SmartScreen prevented an unrecognized app from unning this app might put your PC at risk.	
App: Publisher:	sfdx-cli-v6.16.10-5f5ed66abf-x64.exe Salesforce.com, Inc	
	Run anyway Don't run	

Workaround: To ignore this message, click Run anyway.

Can't Import Record Types Using the Salesforce CLI

Description: We don't support Record Type when running the data:tree:import command.

Workaround: None.

Limited Support for Shell Environments on Windows

Description: Salesforce CLI is tested on the Command Prompt (cmd.exe) and Powershell. There are known issues in the Cygwin and Min-GW environments, and with Windows Subsystem for Linux (WSL). These environments might be tested and supported in a future release. For now, use a supported shell instead.

Workaround: None.

The force:apex:test:run Command Doesn't Finish Executing

Description: In certain situations, the force:apex:test:run command doesn't finish executing. Examples of these situations include a compile error in the Apex test or an Apex test triggering a pre-compile when another is in progress.

Workaround: Stop the command execution by typing control-C. If the command is part of a continuous integration (CI) job, try setting the environment variable SFDX PRECOMPILE DISABLE=true.

Dev Hub and Scratch Orgs

Salesforce CLI Sometimes Doesn't Recognize Scratch Orgs with Communities

Description: Sometimes, but not in all cases, the Salesforce CLI doesn't acknowledge the creation of scratch orgs with the Communities feature. You can't open the scratch org using the CLI, even though the scratch org is listed in Dev Hub.

Workaround: You can try this workaround, although it doesn't fix the issue in all cases. Delete the scratch org in Dev Hub, then create a new scratch org using the CLI. Deleting and recreating scratch orgs counts against your daily scratch org limits.

Error Occurs If You Pull a Community and Deploy It

Description: The error occurs because the scratch org doesn't have the required guest license.

Workaround: In your scratch org definition file, if you specify the Communities feature, also specify the Sites feature.

Source Management

ERROR: No Results Found for force:source:status After Deleting a Custom Label

Description: The force:source:status command returns a No Results Found error after you delete a custom label in a scratch org.

Workaround: Option #1: If you have only one or two scratch orgs and you can easily identify the affected scratch org by its generated username, use this workaround. In the *Your DX project/.sfdx/org* directory, delete only the folder of the affected scratch org.

Option #2: If you have several scratch orgs associated with your DX project and you don't know which scratch org's local data to delete, use this workaround. Delete the *Your DX project*/.sfdx/org directory. This directory contains source tracking information for all scratch orgs related to the project. When you run the next source-tracking command for this or another scratch org (source:push, source:pull, or source:status), the CLI reconstructs the source tracking information for that org.

After you delete the directory (after option #1 or option #2), run force:source:status again.

ERROR: Entity of type 'RecordType' named 'Account.PersonAccount' cannot be found

Description: Although you can turn on Person Accounts in your scratch org by adding the feature to your scratch org definition, running source:push or source:pull results in an error,

Workaround: None.

force:source:convert Doesn't Add Post-Install Scripts to package.xml

Description: If you run force:source:convert, package.xml does not include the post install script.

Workaround: To fix this issue, choose one of these methods:

- Manually add the <postInstallClass> element to the package.xml in the metadata directory that force:source:convert produces
- Manually add the element to the package in the release org or org to which you are deploying the package.

Must Manually Enable Feed Tracking in an Object's Metadata File

Description: If you enable feed tracking on a standard or custom object, then run force:source:pull, feed tracking doesn't get enabled.

Workaround: In your Salesforce DX project, manually enable feed tracking on the standard or custom object in its metadata file (-meta.xml) by adding <enableFeeds>true</enableFeeds>.

Unable to Push Lookup Filters to a Scratch Org

Description: When you execute the force:source:push command to push the source of a relationship field that has a lookup filter, you sometimes get the following error:

duplicate value found: <unknown> duplicates value on record with id: <unknown> at line num, col num.

Workaround: None.

Deployment

Compile on Deploy Can Increase Deployment Times in Scratch Orgs

Description: If your deployment times for Apex code are slow, your scratch org might have the CompileOnDeploy preference set to true.

Workaround:To turn it off, set it to false (the default) or delete the setting from the scratch org definition.

```
{
  "orgName": "ekapner Company",
  "edition": "Developer",
  "features": [],
  "settings": {
    "orgPreferenceSettings": {
        "s1DesktopEnabled": true,
        "compileOnDeploy": false
    }
  }
}
```

Managed First-Generation Packages

When You Install a Package in a Scratch Org, No Tests Are Performed

Description: If you include tests as part of your continuous integration process, those tests don't run when you install a package in a scratch org.

Workaround: You can manually execute tests after the package is installed.

New Terminology in CLI for Managed Package Password

Description: When you use the CLI to add an installation key to a package version or to install a key-protected package version, the parameter name of the key is --installationkey. When you view a managed package version in the Salesforce user interface, the same package attribute is called "Password". In the API, the corresponding field name, "password", is unchanged.

Workaround: None.

Managed Second-Generation Packages

Unable to Specify a Patch Version for Managed Packages

Description: The four-part package version number includes a patch segment, defined as major.minor.patch.build. However, you can't create a patch for a second-generation managed package. Package creation fails if you set a patch number in the package descriptor. We plan to provide this functionality for managed packages in the Winter '20 release.

Workaround: Always set the patch segment of the version number, to 0. For example, 1.2.0.1 is valid but 1.2.1.1 is not.

Protected Custom Metadata and Custom Settings are Visible to Developers in a Scratch Org If Installed Packages Share a Namespace

Description: Use caution when you store secrets in your second-generation packages using protected custom metadata or protected custom settings. You can create multiple second-generation packages with the same namespace. However, when you install these packages in a scratch org, these secrets are visible to any of your developers that are working in a scratch org with a shared namespace. In the future, we might add a "package-protected" keyword to prevent access to package secrets in these situations.

Workaround: None.

Unlocked Packages

Protected Custom Metadata and Custom Settings are Visible to Developers in a Scratch Org If Installed Packages Share a Namespace

Description: Use caution when you store secrets in your unlocked packages using protected custom metadata or protected custom settings. You can create multiple unlocked packages with the same namespace. However, when you install these packages in a scratch org, these secrets are visible to any of your developers that are working in a scratch org with a shared namespace. In the future, we might add a "package-protected" keyword to prevent access to package secrets in these situations.

Workaround: None.