Recipes

Date published: 2019-12-17
Date modified: 2025-08-18

CLOUDERA

Legal Notice

Contents

Recipes

A recipe is a script that runs on all nodes of a selected host group at a specific time. You can use recipes to create and run scripts that perform specific tasks on your Cloudera Data Hub, Data Lake, or FreeIPA cluster nodes.

You can use recipes for tasks such as installing additional software or performing advanced cluster configuration. For example, you can use a recipe to put a JAR file on the Hadoop classpath.

Recipes can be uploaded and managed via the Cloudera web interface or CLI and then selected, when needed, for a specific cluster and for a specific host group. If selected, they will be executed on a specific host group at a specified time.

Recipes are stored on the Cloudera Manager server for the lifetime of the master node, and are executed at specific times of your choosing:

- pre-service-deployment (formerly pre-cloudera-manager-start and pre-cluster-manager-start): During a Data Hub, Data Lake, or environment deployment, the script will be executed on every node before the CM server starts, and after node repair or OS upgrade of a cluster.
- post-cloudera-manager-start: During a Cloudera Data Hub or Data Lake deployment, the script will be executed on every node after the Cloudera Manager server starts, but before cluster installation. post-cluster-manager start recipes are also executed after node repair or OS upgrade of a cluster. This option is not available for FreeIPA recipes.
- post-service-deployment (formerly post-cluster-install): The script will be executed on every node after cluster installation on the CM server is finished.
- pre-termination: The script will be executed on every node before cluster termination.

> **Note:** On the master node, recipes are triggered when the Cloudera Manager server starts; on other nodes, recipes are triggered when the Cloudera Manager agent starts.

Writing recipes

Refer to these guidelines when creating your recipes.

When using recipes, consider the following guidelines:

- Running bash and python scripts as recipes is supported. We recommend using scripts with Shebang character sequence, for example:

```
#!/bin/sh
#!/bin/bash
#!/usr/bin/sh
#!/usr/bin/bash
#!/usr/bin/env sh
#!/usr/bin/env bash
#!/bin/sh -x
#!/usr/bin/python
#!/usr/bin/env python
```

- The scripts are executed as root. The recipe output is written to /var/log/recipes on each node on which it was executed.
- Supported parameters can be specified as variables by using mustache kind of templating with "{{{ }}}" syntax. Once specified in a recipe, these variables are dynamically replaced when the recipe is executed, generating the

actual values that you provided as part of cluster creation process. For the list of parameters, refer to Recipe and cluster template parameters. For an example, see Example: Recipe using parameters.

Note: Using variable parameters is not supported for FreeIPA recipes.

For example, if your cluster includes an external LDAP and your recipe includes {{{ldap.connectionURL}}}, as demonstrated in the following example

```
#!/bin/bash -e

main() {
 ping {{{ ldap.connectionURL }}}
}
[[ "$0" == "$BASH_SOURCE" ]] && main "$@"
```

then, when this recipe runs, the {{{ldap.connectionURL}}} is replaced with the actual connection URL specified as part of cluster creation process, as demonstrated in the following example:

```
#!/bin/bash -e

main() {
 ping 192.168.59.103
}
[[ "$0" == "$BASH_SOURCE" ]] && main "$@"
```

- Recipe logs can be found at /var/log/recipes/${RECIPE_TYPE}/${RECIPE_NAME}.log
- The scripts are executed on all nodes of the host groups that you select (such as "master", "worker", "compute").
- In order to be executed, your script must be in a network location which is accessible from the Cloudera Management Console and the virtual network in which your cluster is located.
- Make sure to follow Linux best practices when creating your scripts. For example, don't forget to script "Yes" auto-answers where needed.
- Do not execute yum update –y as it may update other components on the instances (such as salt) – which can create unintended or unstable behavior.

Example Python script

```
#!/usr/bin/python
print("An example of a python script")
import sys
print(sys.version_info)
```

Example bash script for yum proxy settings

```
#!/bin/bash
cat >> /etc/yum.conf
<<ENDOF
proxy=http://10.0.0.133:3128
ENDOF
```

Example recipe including variables

Original recipe:

```
#!/bin/bash -e

function setupAtlasServer() {
```

```
    curl -iv -u {{{ general.userName }}}:{{{ general.password }}} -H "X-Re
quested-By: ambari" -X POST -d '{"RequestInfo":{"command":"RESTART","context
":"Restart all components required ATLAS","operation_level":{"level":"SERVIC
E","cluster_name":"{{{ general.clusterName }}}","service_name":"ATLAS"}},"Re
quests/resource_filters":[{"hosts_predicate":"HostRoles/stale_configs=false&
HostRoles/cluster_name={{{ general.clusterName }}}"}]}' http://$(hostname -f
):8080/api/v1/clusters/{{{ general.clusterName }}}/requests
}

main() {
    setupAtlasServer
}

[[ "$0" == "$BASH_SOURCE" ]] && main "$@"
```

Generated recipe (to illustrate how the variables from the original recipe were replaced during cluster creation):

```
#!/bin/bash -e

function setupAtlasServer() {
    curl -iv -u admin:admin123 -H "X-Requested-By: ambari" -X POST -d '{"R
equestInfo":{"command":"RESTART","context":"Restart all components required
ATLAS","operation_level":{"level":"SERVICE","cluster_name":"super-cluster","
service_name":"ATLAS"}},"Requests/resource_filters":[{"hosts_predicate":"Hos
tRoles/stale_configs=false&HostRoles/cluster_name=super-cluster"}]}' http://
$(hostname -f):8080/api/v1/clusters/super-cluster/requests
}

main() {
    setupAtlasServer
}

[[ "$0" == "$BASH_SOURCE" ]] && main "$@"
```

Recipe and cluster template parameters

The following supported parameters can be specified as variables/dynamic parameters in recipes or cluster templates by using mustache formatting with "{{{ }}}" syntax.

> **Note:** Using variable parameters is not supported for FreeIPA recipes.

The parameter keys listed below follow the following general conventions:

- { } indicates that the parameter key has multiple supported values, which are provided in this documentation. For example {fileSystemType} can be one of the following: s3, adls_gen_2, or wasb.
- [index] indicates that the parameter includes an index value for example sharedService.datalakeComponents.[index] can be "sharedService.datalakeComponents.[0]", "sharedService.datalakeComponents.[1]", and so on. There is no easy way to find out what the index will be, but you may still be able to use these parameters (for example by creating a condition to filter them).

For information on how to set these parameters dynamically in a cluster template, refer to Setting custom properties.

Custom properties

Any custom property specified in the cluster template can be used as a recipe parameter. Refer to Custom properties documentation.

General

The general parameter group includes parameters related to general cluster configuration.

Description	Example key	Example value
Name of stack	general.stackName	teststack
UUID of cluster	general.uuid	9aab7fdb-8940-454b-bc0a-62f04bce6519
Cloudera Manager user name	general.cmUserName	
Cloudera Manager password	general.cmPassword	
Cloudera Manager IP	general.clusterManagerIp	127.0.0.1
Number of nodes	general.nodeCount	5
FQDN of primary gateway instance	general.primaryGatewayInstanceDiscoveryFQDN	ip-10-0-88-28.example.com
Number indicating the Kafka replication factor (3 or 1)	general.kafkaReplicationFactor	1

Blueprint

The blueprint parameter group includes parameters related to cluster template configuration.

Parameter key	Description	Example key	Example value
blueprint.blueprintText	Blueprint text in JSON format	blueprint.blueprintText	
blueprint.version	Version of blueprint	blueprint.version	7.2.8

Cloud storage

The fileSystemConfigs parameter group includes parameters related to cloud storage configuration.

When forming the parameter keys, the {fileSystemType} should be replaced with an actual cloud storage type such as "s3", "adls_gen_2", or "wasb".

Parameter key	Description	Example key	Example value
File system common configurations			
fileSystemConfigs. {fileSystemType}.storageContainer	Name of container in Azure storage account (Cloudbreak + stackId)	fileSystemConfigs.s3.storageContainer	cloudbreak123
fileSystemConfigs. {fileSystemType}.type	Type of filesystem	fileSystemConfigs.s3.type	S3
fileSystemConfigs. {fileSystemType}.locations. [index].configFile	Configuration file used to configure the filesystem	fileSystemConfigs.s3.locations. [0].configFile	hbase-site
fileSystemConfigs. {fileSystemType}.locations. [index].property	Property key of filesystem path in defined config	fileSystemConfigs.s3.locations. [0].property	hbase.rootdir
fileSystemConfigs. {fileSystemType}.locations. [index].value	Value of filesystem path in defined config	fileSystemConfigs.s3.locations. [0].value	s3a://testranger/testrecipe2/apps/ hbase/data
Amazon S3 configurations			
fileSystemConfigs.s3.storageContainer	Generated name (cloudbreak + stack id number)	fileSystemConfigs.s3.storageContainer	cloudbreak7941
fileSystemConfigs.s3.locations. [index].configFile	Hadoop component configuration file	fileSystemConfigs.s3.locations. [0].configFile	zeppelin-site
fileSystemConfigs.s3.locations. [index].property	Component property name	fileSystemConfigs.s3.locations. [0].property	zeppelin.notebook.dir

Parameter key	Description	Example key	Example value
fileSystemConfigs.s3.locations.[index].value	Component property value	fileSystemConfigs.s3.locations.[0].value	s3a://storagename/clustername/zeppelin/notebook
ADLS Gen2 configurations			
fileSystemConfigs.adls_gen_2.accountName	Name of the corresponding Azure storage account	fileSystemConfigs.adls_gen_2.accountName	teststorageaccount
fileSystemConfigs.adls_gen_2.storageContainerName	Name of the container in Azure storage account	fileSystemConfigs.adls_gen_2.storageContainerName	testcontainerName

External database

The rds parameter group includes parameters related to external database configuration.

When forming the parameter keys, the {rdsType} should be replaced with the actual database type such as "cloudera_manager", "beacon", "druid", "hive", "oozie", "ranger", "superset", or some other user-defined type.

Parameter key	Description	Example key	Example value
rds.{rdsType}.connectionURL	JDBC connection URL	rds.hive.connectionURL	Value is specified in the following format: jdbc:postgresql://host:port/database
rds.{rdsType}.connectionDriver	JDBC driver used for connection	rds.hive.connectionDriver	org.postgresql.Driver
rds.{rdsType}.connectionUserName	Username used for the JDBC connection	rds.hive.connectionUserName	testuser
rds.{rdsType}.connectionPassword	Password used for the JDBC connection	rds.hive.connectionPassword	TestPssword123
rds.{rdsType}.databaseName	Target database of the JDBC connection	rds.hive.databaseName	myhivedb
rds.{rdsType}.host	Host of the JDBC connection	rds.hive.host	mydbhost
rds.{rdsType}.hostWithPortWithJdbc	Host of JDBC connection with port and JDBC prefix	rds.hive.hostWithPortWithJdbc	Value is specified in the following format: jdbc:postgresql://host:port
rds.{rdsType}.subprotocol	Sub-protocol from the JDBC URL	rds.hive.subprotocol	postgresql
rds.{rdsType}.connectionString	URL of JDBC the connection. In case of Ranger, this does not contain the port	rds.hive.connectionString	Value is specified in the following format: jdbc:postgresql://host:port/database
rds.{rdsType}.databaseVendor	Database vendor	rds.hive.databaseVendor	POSTGRES
rds.{rdsType}.withoutJDBCPrefix	URL of the JDBC connection without JDBC prefix	rds.hive.withoutJDBCPrefix	Value is specified in the following format: host:port/database

Gateway

The gateway parameter group includes parameters related to Knox gateway configuration.

Parameter key	Description	Example key	Example value
gateway.ssoProvider	Path to the SSO provider	gateway.ssoProvider	/test/sso/api/v1/websso
gateway.signKey	Base64 encoded signing key	gateway.signKey	
gateway.signPub	Signing certificate (x509 format)	gateway.signPub	
gateway.signCert	Public SSH key used for signing (standard public key format)	gateway.signCert	

Shared services

The sharedService parameter group includes parameters related to Data Lake configuration.

Parameter key	Description	Example key	Example value
sharedService.rangerAdminPassword	Admin password of the Ranger component	sharedService.rangerAdminPassword	Admin1234!
sharedService.datalakeCluster	Flag indicating that the cluster is a data lake cluster	sharedService.datalakeCluster	true
sharedService.datalakeClusterManagerIp	Cloudera Manager IP of data lake cluster	sharedService.datalakeClusterManagerIp	127.0.0.1
sharedService.datalakeClusterManagerFqdn	Cloudera Manager FQDN of data lake cluster (or the IP if FQDN cannot be found)	sharedService.datalakeClusterManagerFqdn	ip-10-0-88-28.example.com

Related Information

Example: Recipe with parameters

Example: Recipe with parameters

If you pass the supported parameters in a recipe, their values are dynamically fetched and replaced.

> **Note:** Using variable parameters is not supported for FreeIPA recipes.

Example recipe template (the {{{general.clusterName}}} is included as a template):

```
#!/bin/bash -e

function setupDefaultClusterFolder() {
  mkdir -p /var/log/{{{general.clusterName}}}
}

main() {
    setupDefaultClusterFolder
}

[[ "$0" == "$BASH_SOURCE" ]] && main "$@"
```

Example recipe after {{{general.clusterName}}} is set to my-super-cluster based on the actual cluster name:

```
#!/bin/bash -e

function setupDefaultClusterFolder() {
  mkdir -p /var/log/my-super-cluster
}

main() {
    setupDefaultClusterFolder
}
[[ "$0" == "$BASH_SOURCE" ]] && main "$@"
```

Related Information

Recipe and cluster template parameters

Registering a recipe

In order to use your recipe for clusters, you must first register it with the Cloudera Management Console.

About this task

Required role: EnvironmentCreator can create a shared resource and then assign users to it.

SharedResourceUser or Owner of the shared resource can use the resource.

Before you begin

If you are using Cloudera with a proxy, note that the Cloudera proxy settings do not apply to cluster recipes. If you planning to use the recipes, then you can set the proxy settings manually. You can find the proxy settings in the /etc/cdp/proxy.env file.

Procedure

1. Place your script in a network location accessible from Cloudera Management Console and from the virtual network in which your clusters are located.
2. Log in to the Cloudera web interface.
3. Navigate to Shared ResourcesRecipes and click Register Recipe.
4. Provide the following:

Parameter	Value
Name	Enter a name for your recipe.
Description	(Optional) Enter a description for your recipe.
Execution Type	Select one of the following options: • pre-service-deployment (formerly pre-cluster-manager-start): During a Cloudera Data Hub, Data Lake, or environment deployment, the script will be executed on every node (in the host group where you assigned the recipe) before the Cloudera Manager server starts. • post-cluster-manager-start: During a Cloudera Data Hub or Data Lake deployment, the script will be executed on every node (in the host group where you assigned the recipe) after the Cloudera Manager server starts, but before cluster installation. This option is not available for FreeIPA recipes. • post-service-deployment (formerly post-cluster-install):: The script will be executed on every node (in the host group where you assigned the recipe) after cluster installation on the Cloudera Manager server is finished. • pre-termination: The script will be executed on every node (in the host group where you assigned the recipe) before cluster termination.
Script	Select one of: • File: Point to a file on your machine that contains the recipe. • Text: Paste the script.

5. Click Register.

What to do next

• When you create a Cloudera Data Hub cluster, you can select a previously added recipe on the advanced Cluster Extensions page of the create cluster wizard.
• When you create an environment, you can select a previously added recipe on the Data Access and Data Lake Scaling page of the environment creation wizard, under Advanced Options > Cluster Extensions > Recipes.
• When you create an environment, you can select a previously added FreeIPA recipe on the **Region, Networking, and Security** page of the environment creation wizard, under Advanced OptionsCluster ExtensionsRecipes.
• You can also attach recipes to Cloudera Data Hub or Data Lake clusters when you create an environment/Data Lake or Cloudera Data Hub cluster through the CDP CLI.

Updating a recipe

You can attach or detach recipes to/from existing Cloudera Data Hub clusters. Using this capability, you can update a recipe by removing it from the cluster, replacing the old recipe with a modified recipe of the same type, and attaching the new modified recipe to the cluster.

About this task

Attaching or detaching a recipe will not execute the recipe. The next execution of the recipe will take place based on the type of the recipe. After an upscale, a newly attached recipe runs only on the new hosts.

Required role (one of the following):

- PowerUser on Cloudera tenant
- Owner of the environment
- EnvironmentAdmin
- Owner of the Cloudera Data Hub cluster
- DataHubAdmin

Procedure

1. Create a new recipe (with updated/modified content) of the same type as the old recipe that you want to replace.
2. Click Data Hub Clusters <Cluster Name> Recipes and find the recipe that you want to remove in the list of recipes for the cluster.
3. Click Remove Recipe next to the name of the recipe that you want to remove, then click Yes in the confirmation window.

4. Once you have removed the old recipe, click on the Add Recipe button for the cluster and select the same host group that you previously used for the old recipe. Then select the name of the new recipe that contains the modified content and click Add.
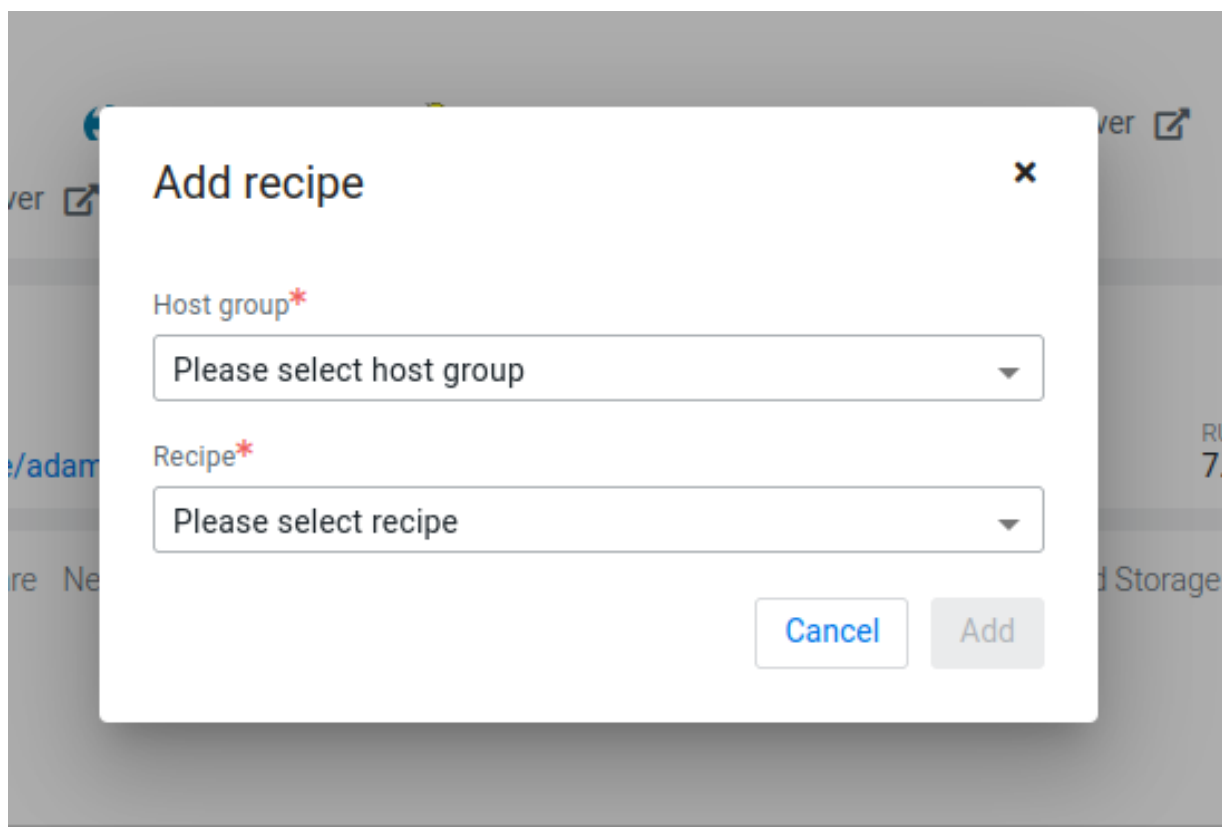
Results

You should see the new recipe appear for the same host group. After this change, the next recipe execution will execute the new script.

Managing recipes from CLI

You can manage recipes from CLI using cdp datahub commands.

Required role: EnvironmentCreator can create a shared resource and then assign users to it.

SharedResourceUser or Owner of the shared resource can use the resource. The Owner of the shared resource can delete it.

> **Note:** Currently, recipes use cdp datahub commands regardless of whether the recipe is intended to run on Cloudera Data Hub, Data Lake, or FreeIPA cluster nodes.

- Register a new recipe: cdp datahub create-recipe --recipe-name <value> --recipe-content <value> --type <value>

  Supported types:

  - PRE_SERVICE_DEPLOYMENT (formerly PRE_CLOUDERA_MANAGER_START)
  - POST_CLOUDERA_MANAGER_START (this option is not available for FreeIPA recipes)
  - POST_SERVICE_DEPLOYMENT (formerly POST_CLUSTER_INSTALL)
  - PRE_TERMINATION
- List all available recipes: cdp datahub list-recipes
- Describe a specific recipe: cdp datahub describe-recipe --recipe-name   <value>

- Delete one or more existing recipes: cdp datahub delete-recipes --recipe-name   <value>