Red Hat CloudForms 4.6

Installing Red Hat CloudForms on OpenShift Container Platform

How to install and configure Red Hat CloudForms on an OpenShift Container Platform environment

Last Updated: 2018-07-05

Red Hat CloudForms 4.6 Installing Red Hat CloudForms on OpenShift Container Platform

How to install and configure Red Hat CloudForms on an OpenShift Container Platform environment

Red Hat CloudForms Documentation Team
cloudforms-docs@redhat.com

Legal Notice

Abstract

This guide provides instructions on how to deploy and configure a Red Hat CloudForms appliance as multiple pods on an OpenShift Container Platform environment. If you have a suggestion for improving this guide or have found an error, please submit a Bugzilla report at http://bugzilla.redhat.com against Red Hat CloudForms Management Engine for the Documentation component. Please provide specific details, such as the section number, guide name, and CloudForms version so we can easily locate the content.

Table of Contents

CHAPTER 1. INSTALLING RED HAT CLOUDFORMS

Red Hat CloudForms can be installed on OpenShift Container Platform in a few steps.

This procedure uses a template to deploy a multi-pod CloudForms appliance with the database stored in a persistent volume on OpenShift Container Platform. It provides a step-by-step setup, including cluster administrative tasks as well as information and commands for the application developer using the deployment.

The ultimate goal of the deployment is to be able to deconstruct the CloudForms appliance into several containers running on a pod or a series of pods.

Running the CloudForms appliance in a series of pods has several advantages. For example, running each worker in a separate pod allows OpenShift Container Platform to manage worker processes and reduce worker memory consumption. OpenShift can also easily scale workers by adding or removing pods, and perform upgrades by using images.

There are two options for installing CloudForms on OpenShift:

- During OpenShift Container Platform 3.7 installation:
  - When you install OpenShift Container Platform 3.7, you have the option to install CloudForms inside OpenShift at the time. This method leverages the Ansible installer to run and deploy the CloudForms template, instead of building the environment manually. See the OpenShift Container Platform 3.7 Release Notes for details.
- Manual install on an existing OpenShift Container Platform environment:
  - Deploy CloudForms pods using the CloudForms template (*.yaml* file). This is the method described in this guide.

After deployment, you can configure the CloudForms environment to use any external authentication configurations supported by CloudForms.

1.1. PREREQUISITES

To successfully deploy a CloudForms appliance on OpenShift Container Platform, you need a functioning **OpenShift Container Platform 3.7** install with the following configured:

- NFS or other compatible volume provider
- A **`cluster-admin`** user
- A regular user (such as an application developer)

IMPORTANT

OpenShift Container Platform 3.7 is required for this installation. Red Hat has not tested this procedure with earlier versions of OpenShift Container Platform.

1.1.1. Cluster Sizing

To avoid deployment failures due to resource starvation, Red Hat recommends the following minimum cluster size for a test environment:

- 1 master node with at least 8 vCPUs and 12GB RAM

- 2 schedulable nodes with at least 4 vCPUs and 8GB RAM
- 25GB storage for CloudForms physical volume use

These recommendations assume CloudForms is the only application running on this cluster. Alternatively, you can provision an infrastructure node to run registry, metrics, router, and logging pods.

Each CloudForms application pod will consume at least 3GB RAM on initial deployment (without providers added). RAM consumption increases depending on the appliance use. For example, after adding providers, expect higher resource consumption.

1.1.2. Limitations

The following limitations exist when deploying this version of CloudForms on OpenShift Container Platform 3.7:

- This configuration cannot run on public OpenShift (OpenShift.io and OpenShift Dedicated environments) because of necessary privileges
- The Embedded Ansible pod must run as a privileged pod
- OpenShift cannot independently scale workers
- A highly available database is not supported in PostgreSQL pods

1.1.3. Templates and Images

The CloudForms deployment uses **`.yaml`** template files to create the appliance, including **`cfme-template.yaml`**, which is the CloudForms template used for the deployment, and **`cfme-pv-example.yaml`** and **`cfme-pv-app-example.yaml`**, two pod volume files.

These templates are available in RPMs from Red Hat-provided image streams. To obtain the templates:

1. Configure image streams as described in OpenShift Container Platform *Installation and Configuration*.
2. After loading the image streams and templates, the templates will be available on your OpenShift system in **`/usr/share/ansible/openshift-ansible/roles/openshift_examples/files/examples/v3.7/cfme-templates`**.

The CloudForms template points to several image files to create the OpenShift pods that comprise the appliance. These image files are obtained from the Red Hat Container Catalog during deployment.

1.2. PREPARING TO DEPLOY CLOUDFORMS

To prepare for deploying the CloudForms appliance to OpenShift Container Platform, create a project, configure security contexts, and create persistent storage.

1. As a regular user, log in to OpenShift:

```
$ oc login -u <user> -p <password>
```

2. Create a project with your desired parameters. The project name (**`<your_project>`** in this example) is mandatory, but **`<description>`** and **`<display_name>`** are optional:

```
$ oc new-project <your_project> \
--description="<description>" \
--display-name="<display_name>"
```

3. As the admin user, configure security context constraints (SCCs) for your OpenShift service accounts:

   a. Add the **cfme-anyuid** service account to the **anyuid** SCC:

   ```
   $ oc adm policy add-scc-to-user anyuid system:serviceaccount:
   <your-project>:cfme-anyuid
   ```

   b. Add the **cfme-orchestrator** service account to the **anyuid** SCC:

   ```
   $ oc adm policy add-scc-to-user anyuid system:serviceaccount:
   <your-project>:cfme-orchestrator
   ```

   c. Add the **cfme-httpd** service account to the **anyuid** SCC:

   ```
   $ oc adm policy add-scc-to-user anyuid system:serviceaccount:
   <your-project>:cfme-httpd
   ```

   d. Add the **cfme-privileged** service account to the **privileged** SCC:

   ```
   $ oc adm policy add-scc-to-user privileged system:serviceaccount:
   <your-project>:cfme-privileged
   ```

4. Verify the SCCs are added correctly to the service accounts and project:

```
$ oc describe scc anyuid | grep Users
Users:     system:serviceaccount:<your-project>:cfme-
anyuid,system:serviceaccount:<your-project>:cfme-
httpd,system:serviceaccount:<your-project>:cfme-orchestrator
$ oc describe scc privileged | grep Users
Users:     system:admin,system:serviceaccount:openshift-infra:build-
controller,system:serviceaccount:management-infra:management-
admin,system:serviceaccount:management-infra:inspector-
admin,system:serviceaccount:logging:aggregated-logging-
fluentd,system:serviceaccount:<your-project>:cfme-privileged
```

NOTE

For more information on SCCs, see the OpenShift documentation.

5. As the admin user, add the **httpd-configmap-generator** service account to the **httpd-scc-sysadmin** SCC so the Httpd Configmap Generator can run.

```
Users:        system:serviceaccount:<your-namespace>:httpd-
configmap-generator
```

6. Add the **view** and **edit** roles to the **cfme-orchestrator** service account:

```
$ oc policy add-role-to-user view system:serviceaccount:<your-
project>:cfme-orchestrator -n <your-project>
$ oc policy add-role-to-user edit system:serviceaccount:<your-
project>:cfme-orchestrator -n <your-project>
```

7. As the admin user, prepare persistent storage for the deployment. (Skip this step if you have already configured persistent storage.)
   A basic CloudForms deployment needs at least two persistent volumes (PVs) to store CloudForms data. As the admin user, create two persistent volumes: one to host the CloudForms PostgreSQL database, and one to host the application data.

   Example NFS-backed volume templates are provided by **`cfme-pv-db-example.yaml`** and **`cfme-pv-server-example.yaml`**, available from the image stream or repository configured in Section 1.1.3, “Templates and Images”.

   NOTE

   For NFS-backed volumes, ensure your NFS server firewall is configured to allow traffic on port 2049 (TCP) from the OpenShift cluster.

   Red Hat recommends setting permissions for the pv-app (privileged pod volume) as 777, uid/gid 0 (owned by root). For more information on configuring persistent storage in OpenShift Container Platform, see the OpenShift Container Platform Installation and Configuration guide.

   a. Configure your NFS server host details within these files, and edit any other settings needed to match your environment.

   b. Create the two persistent volumes:

   ```
   $ oc create -f cfme-pv-db-example.yaml
   $ oc create -f cfme-pv-server-example.yaml
   ```

   c. Process the templates, editing the NFS_HOST parameter (mandatory) and any other parameters:

   ```
   $ oc process cfme-pv-db-example.yaml -p NFS_HOST=nfs.example.com
   | oc create -f -
   ```

   Alternatively, you can create the two persistent volumes and process the templates in a single command:

   ```
   $ oc process cfme-pv-server-example.yaml -p
   NFS_HOST=nfs.example.com | oc create -f -
   ```

NOTE

There are three parameters required to process the template. Only NFS_HOST is required, PV_SIZE and BASE_PATH contain defaults that do not need editing unless desired:

- PV_SIZE - Defaults to the recommended PV size for the App/DB template (5Gi/15Gi respectively)
- BASE_PATH - Defaults to /exports
- NFS_HOST - No Default - Hostname or IP address of the NFS server

d. Verify the persistent volumes were created successfully:

```
$ oc get pv
NAME                CAPACITY   ACCESSMODES   RECLAIMPOLICY
STATUS      CLAIM          STORAGECLASS   REASON    AGE
cfme-app            5Gi        RWO           Retain
Available                                          16s

cfme-db             15Gi       RWO           Retain
Available                                          49s
```

NOTE

Red Hat recommends validating NFS share connectivity from an OpenShift node before attempting a deployment.

8. Increase the maximum number of imported images on ImageStream.
   By default, OpenShift Container Platform can import five tags per image stream, but the CloudForms repositories contain more than five images for deployments.

   You can modify this setting on the master node at **`/etc/origin/master/master-config.yaml`** so OpenShift can import additional images.

   a. Add the following at the end of the **`/etc/origin/master/master-config.yaml`** file:

```
...
imagePolicyConfig:
  maxImagesBulkImportedPerRepository: 100
```

   b. Restart the master service:

```
$ systemctl restart atomic-openshift-master
```

9. On each OpenShift node, persistently enable the **`container_manage_cgroup`** SELinux boolean to allow container processes to make changes to the *cgroup* configuration:

```
# setsebool -P container_manage_cgroup on
```

1.3. DEPLOYING THE CLOUDFORMS APPLIANCE

To deploy the appliance on OpenShift Container Platform, create the CloudForms template and verify it is available in your project.

1. As a regular user, create the CloudForms template:

```
$ oc create -f cfme-template.yaml
template "cloudforms" created
```

2. Verify the template is available with your project:

```
$ oc get templates
NAME         DESCRIPTION
PARAMETERS        OBJECTS
cloudforms   CloudForms appliance with persistent storage   18 (1
blank)      12
```

3. (Optional) Customize the template's deployment parameters. Use the following command to see the available parameters and descriptions:

```
$ oc process --parameters -n <your-project> cloudforms
```

   To customize the deployment configuration parameters, run:

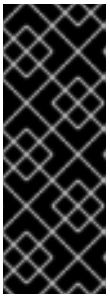
```
$ oc edit dc/<deployconfig_name>
```

4. To deploy CloudForms from template using default settings, run:

```
$ oc new-app --template=cloudforms
```

   Alternatively, to deploy CloudForms from a template using customized settings, add the **-p** option and the desired parameters to the command. For example:

```
$ oc new-app --template=cloudforms -p
DATABASE_VOLUME_CAPACITY=2Gi,POSTGRESQL_MEM_LIMIT=4Gi,APPLICATION_DO
MAIN=hostname
```



IMPORTANT

The **APPLICATION_DOMAIN** parameter specifies the hostname used to reach the CloudForms application, which eventually constructs the route to the CloudForms pod. If you do not specify the **APPLICATION_DOMAIN** parameter, the CloudForms application will not be accessible after the deployment; however, this can be fixed by changing the route. For more information on OpenShift template parameters, see the OpenShift Container Platform Developer Guide.

1.3.1. Deploying the CloudForms Appliance Using an External Database

Before attempting to deploy CloudForms using an external database deployment, ensure the following conditions are satisfied:

- Your OpenShift cluster can access the external PostgreSQL server

- The CloudForms user, password, and role have been created on the external PostgreSQL server
- The intended CloudForms database is created, and ownership has been assigned to the CloudForms user

To deploy the appliance:

1. Import the CloudForms external database template:

```
$ oc create -f templates/cfme-template-ext-db.yaml
```

2. Launch the deployment with the following command. The database server IP address is required, and the other settings must match your remote PostgreSQL server.

```
$ oc new-app --template=cloudforms-ext-db -p DATABASE_IP=<server_ip>
-p DATABASE_USER=<user> -p DATABASE_PASSWORD=<password> -p
DATABASE_NAME=<database_name>
```

1.4. VERIFYING THE CONFIGURATION

Verify the deployment was successful by running the following commands as a regular user under the CloudForms project:

NOTE

The first deployment can take several minutes to complete while OpenShift downloads the necessary images.

1. Confirm the CloudForms pod is bound to the correct security context constraints:
   a. List and obtain the name of the **`cfme-app`** pod:

```
$ oc get pod
NAME                READY     STATUS    RESTARTS   AGE
cloudforms-0        1/1       Running   0          4m
httpd-1-w486v       1/1       Running   0          4m
memcached-1-4xtjc   1/1       Running   0          4m
postgresql-1-n5tm6  1/1       Running   0          4m
```

   b. Export the configuration of the pod:

```
$ oc export pod <cfme_pod_name>
```

   c. Examine the output to verify that **`openshift.io/scc`** has the value **`anyuid`**:

```
...
metadata:
  annotations:
    openshift.io/scc: anyuid
...
```

2. Verify the persistent volumes are attached to the **`postgresql`** and **`cfme-app`** pods:

```
$ oc volume pods --all
pods/postgresql-1-437jg
  pvc/cfme-pgdb-claim (allocated 2GiB) as cfme-pgdb-volume
    mounted at /var/lib/pgsql/data
  secret/default-token-2se06 as default-token-2se06
    mounted at /var/run/secrets/kubernetes.io/serviceaccount
pods/cfme-1-s3bnp
  pvc/cfme (allocated 2GiB) as cfme-app-volume
    mounted at /persistent
  secret/default-token-9q4ge as default-token-9q4ge
    mounted at /var/run/secrets/kubernetes.io/serviceaccount
```

3. Check the readiness of the CloudForms pod:

NOTE

Allow approximately five minutes once pods are in running state for CloudForms to start responding on HTTPS.

```
$ oc describe pods <cfme_pod_name>
...
Conditions:
  Type      Status
  Ready     True
Volumes:
...
```

4. After you have successfully validated your CloudForms deployment, disable automatic image change triggers to prevent unintended upgrades.
By default, on initial deployments the automatic image change trigger is enabled. This could potentially start an unintended upgrade on a deployment if a newer image is found in the ImageStream.

   Disable the automatic image change triggers for CloudForms deployment configurations (DCs) on each project with the following commands:

```
$ oc set triggers dc --manual -l app=cloudforms
deploymentconfig "memcached" updated
deploymentconfig "postgresql" updated

$ oc set triggers dc --from-config --auto -l app=cloudforms
deploymentconfig "memcached" updated
deploymentconfig "postgresql" updated
```

NOTE

The configuration change trigger is kept enabled; to have full control of your deployments, you can alternatively turn it off. See the OpenShift Container Platform Developer Guide for more information on deployment triggers.

1.5. LOGGING INTO CLOUDFORMS

As part of the deployment, a route to the CloudForms appliance is created for HTTPS access. Once the pods have been successfully deployed, you can log into CloudForms.

You can obtain the CloudForms host address from the project in the OpenShift user interface, or by opening a shell on the pod and getting the route information.

1. To open a shell on the pod, run:

```
$ oc rsh <pod_name> bash -l
```

2. Get the route information:

```
$ oc get routes
NAME         HOST/PORT                    PATH
SERVICE      TERMINATION   LABELS
cloudforms   cfme.apps.e2e.example.com  cloudforms:443-tcp
passthrough               app=cloudforms
```

3. Navigate to the reported URL/host on a web browser (in this example, **`cfme.apps.e2e.example.com`**).

4. Enter the default CloudForms credentials (Username: **admin** | Password: **smartvm**) for the initial login.

5. Click **Login**.

CHAPTER 2. CONFIGURING EXTERNAL AUTHENTICATION TO CLOUDFORMS

After installing CloudForms, configure external authentication by updating the **httpd-auth-configs** configuration map on the **httpd** pod to include all necessary configuration files and certificates.

Upon startup, the **httpd** pod overlays its files with the ones specified in the **auth-configuration.conf** file in the configuration map. This is done by the **initialize-httpd-auth** service that runs before **httpd**.

You can automatically generate an updated configuration map by running the **httpd-configmap-generator** tool in its own pod using the steps in Section 2.1, “Configuring External Authentication Automatically” (recommended). Alternatively, you can define the configuration map manually using the commands in Section 2.2, “Defining the Configuration Map Manually”.

2.1. CONFIGURING EXTERNAL AUTHENTICATION AUTOMATICALLY

To automatically generate an **authconfig** map, run the **httpd_configmap_generator** tool with your desired parameters:

```
$ httpd_configmap_generator <command_or_authentication_type>
```

NOTE

Run **httpd_configmap_generator --help** or see Section 5.1, “External Authentication Configuration Map Settings” for configuration map parameters.

2.1.1. Supported Authentication Types

The following authentication types can be configured with the **httpd_configmap_generator** tool to configure external authentication.

For usage, run:

```
$ httpd_configmap_generator <auth-type> --help
```

Table 2.1. Supported Authentication Types

auth-type	Identity Provider/Environment
active-directory	Active Directory domain realm join
ipa	IPA, IPA 2-factor authentication, IPA/AD Trust
ldap	LDAP directories
saml	Keycloak, Red Hat SSO

2.1.2. Updating an `authconfig` Map

With the **update** subcommand, you can add file(s) to the configuration map to specify file ownership and permissions. The **--add-file** option can be specified multiple times (once per file) to add files to a configuration map.

Supported file specifications for the **--add-file** option are:

```
--add-file=file-path
--add-file=source-file-path,target-file-path
--add-file=source-file-path,target-file-path,file-permission
--add-file=file-url,target-file-path,file-permission
```

When entering file specifications, **file-url** is an HTTP URL and **file-permission** can be specified as **mode:owner:group**.

Adding files by specifying paths

The file ownership and permissions are based on the files specified. For example:

```
$ httpd_configmap_generator update \
--input=/tmp/original-auth-configmap.yaml                \
--add-file=/etc/openldap/cacerts/primary-directory-cert.pem  \
--add-file=/etc/openldap/cacerts/seconday-directory-cert.pem \
--output=/tmp/updated-auth-configmap.yaml
```

Adding target files from different source directories

```
$ httpd_configmap_generator update \
--input=/tmp/original-auth-configmap.yaml
\
--add-file=/tmp/uploaded-cert1,/etc/openldap/cacerts/primary-directory-
cert.pem  \
--add-file=/tmp/uploaded-cert2,/etc/openldap/cacerts/seconday-directory-
cert.pem \
--output=/tmp/updated-auth-configmap.yaml
```

The file ownership and permissions are based on the source files specified; in this case the ownership and permissions of the **/tmp/uploaded-cert1** and **/tmp/uploaded-cert2** files will be used.

Adding a target file with user-specified ownership and mode

```
$ httpd_configmap_generator update \
--input=/tmp/original-auth-configmap.yaml                          \
--add-file=/tmp/secondary-keytab,/etc/http2.keytab,600:apache:root \
--output=/tmp/updated-auth-configmap.yaml
```

Adding files by URL

```
$ httpd_configmap_generator update \
--input=/tmp/original-auth-configmap.yaml \
--add-file=http://aab-
keycloak:8080/auth/realms/testrealm/protocol/saml/description,/etc/httpd/s
aml2/idp-metadata.xml,644:root:root \
--output=/tmp/updated-auth-configmap.yaml
```

When downloading a file by URL, a target file path and file ownership/mode must be specified.

2.1.3. Exporting a File from an `authconfig` Map

With the **`export`** subcommand, you can export a file from the configuration map. For example, to extract the **`sssd.conf`** file from the **`authconfig`** map:

```
$ httpd_configmap_generator export \
--input=/tmp/external-ipa.yaml \
--file=/etc/sssd/sssd.conf     \
--output=/tmp/sssd.conf
```

2.1.4. Building the `httpd_configmap_generator` in a Container

The **`httpd_configmap_generator`** is the container for configuring external authentication for the **`httpd`** auth pod. It is based on the auth **`httpd`** container and generates the **`httpd authconfig`** map needed to enable external authentication.

Two templates are required to run the **`httpd-configmap-generator`** application (**`httpd-configmap-generator-htmplate.yaml`** and **`httpd-scc-sysadmin.yaml`**), which are available from the Red Hat Container Catalog.

2.1.4.1. Preparing to Deploy the `httpd-configmap-generator` Application

1. To obtain the latest **`cfme-httpd-configmap-generator`** image from the Red Hat Container Catalog, run:

   ```
   $ oc import-image my-cloudforms46/cfme-httpd-configmap-generator --
   from=registry.access.redhat.com/cloudforms46/cfme-httpd-configmap-
   generator --confirm
   ```

2. The **`httpd-configmap-generator`** service account must be added to the **`httpd-scc-sysadmin`** SCC before the **`httpd-configmap-generator`** can run. To edit the SCC, log in to OpenShift as an admin user:

   ```
   $ oc login -u <user> -p <password>
   ```

3. Create the **`httpd-scc-sysadmin`** SCC:

   ```
   $ oc create -f templates/httpd-scc-sysadmin.yaml
   ```

4. Add the **`httpd-configmap-generator`** service account to the **`httpd-scc-sysadmin`** SCC:

   ```
   $ oc adm policy add-scc-to-user httpd-scc-sysadmin
   system:serviceaccount:<your-namespace>:httpd-configmap-generator
   ```

5. Verify that the **`httpd-configmap-generator`** service account is now included in the **`httpd-scc-sysadmin`** SCC:

   ```
   $ oc describe scc httpd-scc-sysadmin | grep Users
   Users:        system:serviceaccount:<your-namespace>:httpd-
   configmap-generator
   ```

2.1.4.2. Deploying the `httpd-configmap-generator` Application

1. As a regular user, run:

```
$ oc create -f httpd-configmap-generator-template.yaml
```

2. Verify the template is available with your project:

```
$ oc get templates
NAME                          DESCRIPTION
PARAMETERS     OBJECTS
httpd-configmap-generator   Httpd Configmap Generator
6 (all set)    3
```

3. Deploy the **`httpd-configmap-generator`**:

```
$ oc new-app --template=httpd-configmap-generator
```

4. Check the readiness of the **`httpd-configmap-generator`**:

```
$ oc get pods
NAME                                READY     STATUS    RESTARTS
AGE
httpd-configmap-generator-1-txc34   1/1       Running   0
1h
```

2.1.4.3. Getting the Pod Name

To work with the **`httpd-configmap-generator`** script in the **`httpd-configmap-generator`** pod, it is necessary to get the pod name as below:

```
$ CONFIGMAP_GENERATOR_POD=`oc get pods | grep "httpd-configmap-generator"
| cut -f1 -d" "`
```

2.1.4.4. Example: Generating an `authconfig` Map for External Authentication Against IPA

The following example shows how to generate a configuration map for external authentication using IPA.

1. To generate an **`authconfig`** map for external authentication using IPA, run:

```
$ oc rsh $CONFIGMAP_GENERATOR_POD -- bash -c
httpd_configmap_generator ipa \
--host=appliance.example.com        \
--ipa-server=ipaserver.example.com  \
--ipa-domain=example.com            \
--ipa-realm=EXAMPLE.COM             \
--ipa-principal=admin               \
--ipa-password=smartvm1             \
-o /tmp/external-ipa.yaml
```

NOTE

`--host` must be the DNS of the application exposing the **`httpd`** pod, for example ${APPLICATION_DOMAIN}.

2. Copy the new **`authconfig`** map back locally:

```
$ oc cp $CONFIGMAP_GENERATOR_POD:/tmp/external-ipa.yaml ./external-
ipa.yaml
```

3. Apply the new configuration map to the **`httpd`** pod, and then redeploy it to take effect:

```
$ oc replace configmaps httpd-auth-configs --filename ./external-
ipa.yaml
```

To generate a new **`authconfig`** map, redeploy the **`httpd-configmap-generator`** pod first to get a clean environment before running the **`httpd-configmap-generator`** tool.

If additional configuration is needed, you can configure the configuration map manually using the steps in Section 2.2, “Defining the Configuration Map Manually”. See Section 5.1, “External Authentication Configuration Map Settings” for configuration map parameters.

2.1.4.5. Cleaning up

After generating an **`authconfig`** map, the **`httpd-configmap-generator`** pod can be scaled down, or deleted if no longer needed.

To scale down the pod, run:

```
$ oc scale dc httpd-configmap-generator --replicas=0
```

To delete the pod, run:

```
$ oc delete all  -l app=httpd-configmap-generator
$ oc delete pods -l app=httpd-configmap-generator
```

2.2. DEFINING THE CONFIGURATION MAP MANUALLY

The **`authconfig`** map can be defined and customized in the **`httpd`** pod as follows:

```
$ oc edit configmaps httpd-auth-configs
```

Alternatively, you can replace the **`httpd-auth-configs`** file with an externally generated and edited configuration file as follows:

```
$ oc replace configmaps httpd-auth-configs --filename external-auth-
configmap.yaml
```

After editing the configuration map, redeploy the **`httpd`** pod for the new authentication configuration to take effect.

CHAPTER 3. MANAGING RED HAT CLOUDFORMS WITH OPENSHIFT

This section includes common tasks to manage your Red Hat CloudForms deployment from OpenShift.

3.1. CONFIGURING CUSTOM SSL CERTIFICATES FOR CLOUDFORMS

By default, the route that is deployed as part of the template uses edge termination and the certificates that OpenShift is installed with. It is possible to change this in the OpenShift UI with the following steps:

1. Navigate to **Applications** → **Routes**.
2. Click on the route named **httpd**, then select **Actions** → **Edit**.
3. Scroll down to the **Certificates** section. Here you can upload or paste the required certificate files.
4. Click **Save**.

3.2. SCALING CLOUDFORMS APPLIANCES

StatefulSets in OpenShift manage the deployment and scaling of a set of pods (in this case, CloudForms appliances). StatefulSets ensure ordering that applications will come up by providing unique identities for pods.

IMPORTANT

Each new replica (server) consumes a physical volume. Before scaling, ensure you have enough physical volumes available to scale.

The following example shows scaling using StatefulSets:

Example: Scaling to two replicas

```
$ oc scale statefulset cloudforms --replicas=2
statefulset "cloudforms" scaled
$ oc get pods
NAME                 READY     STATUS    RESTARTS   AGE
cloudforms-0           1/1       Running   0          34m
cloudforms-1           1/1       Running   0          5m
memcached-1-mzeer    1/1       Running   0          1h
postgresql-1-dufgp   1/1       Running   0          1h
```

The newly created replicas will join the existing CloudForms region. Each new pod is numbered in the order it is deployed, starting with 0 and increasing sequentially. For example, replicas in a StatefulSet will be numbered *cloudforms-0 cloudforms-1*, and so on.

3.3. CREATING A BACKUP

Create a persistent volume for backups using the PV backup template (**`cfme-pv-backup-example.yaml`**) in case you need to restore to a previous version.

1. Create the persistent volume for the backup:

```
$ oc create -f cfme-pv-backup-example.yaml
```

2. Create the backup persistent volume claim (PVC):

```
$ oc create -f cfme-backup-pvc.yaml
```

3. Verify the persistent volume claim was created:

```
$ oc get pvc
```

4. Back up secrets, such as database encryption keys and credentials.

IMPORTANT

Be careful to back up secrets in a secure location.

```
$ oc get secret -o yaml --export=true > secrets.yaml
$ oc get pvc -o yaml --export=true > pvc.yaml
```

5. Initiate the database backup:

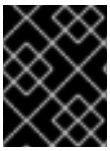
```
$ oc create -f cfme-backup-job.yaml
```

This step creates a container, and connects to the database pod, **`pg_basebackup`**.

3.4. RESTORING TO A BACKUP

You can restore to a database backup created in Section 3.3, “Creating a Backup” using the restore template, **`cfme-restore-job.yaml`**.

The restore job will look for **`cfme-backup`** and **`cfme-postgresql`** PVs by default, and the latest successful backup will be restored by default. If existing data is found on the **`cfme-postgresql`** volume, it will be renamed and left on the volume.



IMPORTANT

You must perform a database restore on an offline environment. All pods must be scaled down to 0, and not running.

1. Shut down all pods:

```
$ oc stop all pods
```

2. To initiate the database restore, create the restore template:

```
$ oc create -f cfme-restore-job.yaml
```

After the restore job is complete, you can scale the pods back up.

3.5. UNINSTALLING RED HAT CLOUDFORMS FROM A PROJECT

If no longer needed, you can uninstall the Red Hat CloudForms pod from your project. Note the following commands do not remove SCC permissions, or the project itself.

IMPORTANT

Use this procedure if only Red Hat CloudForms exists in the project.

1. Inside the project, run the following as a regular user:

```
$ oc delete all --all
```

2. Wait approximately 30 seconds for the command to process, then run:

```
$ oc delete pvc --all
```

CHAPTER 4. TROUBLESHOOTING DEPLOYMENT

Under normal circumstances, the deployment process takes approximately 10 minutes. If the deployment is unsuccessful, examining deployment events and pod logs can help identify any issues.

1. As a regular user, first retry the failed deployment:

```
$ oc get pods
NAME                  READY     STATUS    RESTARTS   AGE
cloudforms-1-deploy   0/1       Error     0          25m
memcached-1-yasfq     1/1       Running   0          24m
postgresql-1-wfv59    1/1       Running   0          24m
```

```
$ oc deploy cloudforms --retry
Retried #1
Use 'oc logs -f dc/cloudforms' to track its progress.
```

2. Allow a few seconds for the failed pod to get re-scheduled, then check events and logs:

```
$ oc describe pods <pod-name>
...
Events:
  FirstSeen LastSeen Count From       SubobjectPath   Type  Reason
Message
  --------- -------- ----- ----       -------------   -------- --
----  -------
15m  15m  1 {kubelet ocp-eval-node-2.e2e.example.com}
spec.containers{cloudforms} Warning  Unhealthy Readiness probe
failed: Get http://10.1.1.5:80/: dial tcp 10.1.1.5:80: getsockopt:
connection refused
```

   Liveness and readiness probe failures, like in the output above, indicate the pod is taking longer than expected to come online. In this case, check the pod logs.

3. As the **cfme-app** container is **systemd** based, use **oc rsh** instead of **oc logs** to obtain journal dumps:

```
$ oc rsh <pod-name> journalctl -x
```

4. Transferring all logs from the **cfme-app** pod to a directory on the host for further examination can be useful for troubleshooting. Transfer the logs with the **oc rsync** command:

```
$ oc rsync <pod-name>:/persistent/container-deploy/log \
/tmp/fail-logs/
receiving incremental file list
log/
log/appliance_initialize_1477272109.log
log/restore_pv_data_1477272010.log
log/sync_pv_data_1477272010.log

sent 72 bytes  received 1881 bytes  1302.00 bytes/sec
total size is 1585  speedup is 0.81
```

CHAPTER 5. APPENDIX

5.1. EXTERNAL AUTHENTICATION CONFIGURATION MAP SETTINGS

See Section 5.2, “Sample External Authentication Configuration” for an example configuration map file.

The configuration map includes the following parameters:

auth-type

The authentication type.
This parameter controls which configuration files **`httpd`** will load upon startup. The default is **`internal`**. Supported values are:

Table 5.1. `auth-type` values

Value	External Authentication Configuration
`internal`	Application Based Authentication - Database, LDAP/LDAPS, Amazon. This is the default.
`external`	IPA, IPA 2-factor authentication, IPA/AD Trust, LDAP (OpenLDAP, RHDS, Active Directory, etc.)
`active-directory`	Active Directory domain realm join
`saml`	SAML based authentication (Keycloak, ADFS, etc.)

IMPORTANT

Enabling external authentication must be done from the CloudForms user interface; see Configuring External Authentication in *Managing Authentication* for details.

auth-kerberos-realms

The Kerberos realms to join.
When configuring external authentication against IPA, Active Directory or LDAP, this parameter defines the Kerberos realm **`httpd`** is configured against, such as *example.com*. When specifying multiple Kerberos realms, they must be separated by spaces. The default is **`undefined`**.

auth-configuration.conf

The external authentication configuration file which declares the list of files to overlay upon startup if **`auth-type`** is other than **`internal`**.
Syntax for the file is as follows:

```
# for comments
file = basename1 target_path1 permission1
file = basename2 target_path2 permission2
```

For the files to overlay on the **`httpd`** pod, one **`file`** directive is needed per file.

basename

The name of the source file in the configuration map.

permission

(optional) By default, files are copied using the pod's default umask, owner and group, so files are created as mode 644 owner root, group root.
`permission` can be specified as follows, reflecting the mode and ownership to set the copied files to:

- mode
- mode:owner
- mode:owner:group
  For example:
- 755
- 640:root
- 644:root:apache
  Binary files can be specified in the configuration map in their base64 encoded format with a basename having a **`.base64`** extension. Such files are then converted back to binary as they are copied to their target path.

  When an **`/etc/sssd/sssd.conf`** file is included in the configuration map, the **`httpd`** pod automatically enables the SSSD service upon startup.

target_path

The path of the file on the pod to overwrite, i.e. **`/etc/sssd/sssd.conf`**.

5.2. SAMPLE EXTERNAL AUTHENTICATION CONFIGURATION

The following is an example of the data section of a SAML auth-config map data section (excluding the content of the files):

```
apiVersion: v1
data:
auth-type: saml
auth-kerberos-realms: example.com
auth-configuration.conf: |
#
# Configuration for SAML authentication
#
file = manageiq-remote-user.conf        /etc/httpd /conf.d/manageiq-
remote-user.conf        644
file = manageiq-external-auth-saml.conf /etc/httpd/conf.d/manageiq-
external-auth-saml.conf 644
file = idp-metadata.xml                 /etc/httpd/saml2/idp-metadata.xml
644
file = sp-key.key                       /etc/httpd/saml2/sp-key.key
600:root:root
file = sp-cert.cert                     /etc/httpd/saml2/sp-cert.cert
644
file = sp-metadata.xml                  /etc/httpd/saml2/sp-metadata.xml
```

```
644
manageiq-remote-user.conf: |
RequestHeader unset X_REMOTE_USER
...
manageiq-external-auth-saml.conf: |
LoadModule auth_mellon_module modules/mod_auth_mellon.so
...
idp-metadata.xml: |
<EntitiesDescriptor ...
...
</EntitiesDescriptor>
sp-key.key: |
-----BEGIN PRIVATE KEY-----
...
-----END PRIVATE KEY-----
sp-cert.cert: |
-----BEGIN CERTIFICATE-----
...
-----END CERTIFICATE-----
sp-metadata.xml: |
<EntityDescriptor ...
...
</EntityDescriptor>
```