Flux Helm Operator Documentation

Flux development team

Feb 21, 2020

CONTENTS

CHAPTER

ONE

REFERENCES

1.1 Helm operator (`helm-operator`)

The Helm operator deals with Helm chart releases. The operator watches for changes of Custom Resources of kind `HelmRelease`. It receives Kubernetes Events and acts accordingly.

1.1.1 Responsibilities

When the Helm Operator sees a `HelmRelease` resource in the cluster, it either installs or upgrades the named Helm release so that the chart is released as specified.

It will also notice when a `HelmRelease` resource is updated, and take action accordingly.

1.1.2 Setup and configuration

`helm-operator` requires setup and offers customization though a multitude of flags.

1.2 `HelmRelease` Custom Resource

Each release of a chart is declared by a `HelmRelease` resource. The schema for these resources is given in the custom resource definition. They look like this:

```
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
metadata:
  name: rabbit
  namespace: default
spec:
  releaseName: rabbitmq
  targetNamespace: mq
  timeout: 300
  resetValues: false
  wait: false
  forceUpgrade: false
  chart:
    repository: https://kubernetes-charts.storage.googleapis.com/
    name: rabbitmq
    version: 3.3.6
  values:
    replicas: 1
```

The `releaseName` will be given to Helm as the release name. If not supplied, it will be generated by affixing the HR namespace and the target namespace to the resource name. In the above example, if `releaseName` were not given, it would be generated as `default-mq-rabbitmq`. Because of the way Helm works, release names must be unique in the cluster.

If you don't supply the `targetNamespace`, the release will be installed in the same namespace as the HelmRelease object.

The `chart` section gives a pointer to the chart; in this case, to a chart in a Helm repo. Since the Helm operator is running in your cluster, and doesn't have access to local configuration, the repository is given as a URL rather than an alias (the URL in the example is what's usually aliased as `stable`). The `name` and `version` specify the chart to release.

The `timeout` sets the timeout value for the Helm install or upgrade. If you don't supply it, it is set to 300.

The `resetValues`, if set to `true`, will reset values on Helm upgrade.

The `wait`, if set to `true`, will instruct the operator to wait for an Helm upgrade to complete before it is marked as successful in the `HelmRelease` resource.

The `forceUpgrade`, if set to `true`, will force Helm upgrade through delete/recreate

The `values` section is where you provide the value overrides for the chart. This is as you would put in a `values.yaml` file, but inlined into the structure of the resource. See below for examples.

Why use URLs to refer to repositories, rather than names? ^

A `HelmRelease` must be able to stand on its own. If we used names in the spec, which were resolved to URLs elsewhere (e.g., in a `repositories.yaml` supplied to the operator), it would be possible to change the meaning of a `HelmRelease` without altering it. This is undesirable because it makes it hard to specify exactly what you want, in the one place; or to read exactly what is being specified, in the one place. In other words, it's better to be explicit.

1.2.1 Using a chart from a Git repo instead of a Helm repo

You can refer to a chart from a *git* repo, rather than a chart repo, with a `chart:` section like this:

```
spec:
  chart:
    git: git@github.com:fluxcd/flux-get-started
    ref: master
    path: charts/ghost
```

In this case, the git repo will be cloned, and the chart will be released from the ref given. If not supplied, the operator uses the value specified by the `--git-default-ref` flag (which defaults to `master`). Commits to the git repo may result in releases, if they update the chart at the path given.

Note that you will usually need to provide an SSH key to grant access to the git repository. The example deployment shows how to mount a secret at the expected location of the key (`/etc/fluxd/ssh/`). If you need more than one SSH key, you'll need to also mount an adapted ssh_config; this is also demonstrated in the example deployment.

Notifying Helm Operator about Git changes

The Helm Operator fetches the upstream of mirrored Git repositories with a 5 minute interval. In some scenarios (think CI/CD), you may not want to wait for this interval to occur.

To help you with this the Helm Operator serves a HTTP API endpoint to instruct it to immediately refresh all Git mirrors.

```
$ kubectl -n flux port-forward deployment/flux-helm-operator 3030:3030 &
$ curl -XPOST http://localhost:3030/api/v1/sync-git
OK
```

> **Note:** the HTTP API has no built-in authentication, this means you either need to port forward before making the request or put something in front of it to serve as a gatekeeper.

1.2.2 Extending the supported Helm repository protocols

By default, the Helm operator is able to pull charts from repositories using HTTP/S. It is however possible to extend the supported protocols by making use of a Helm downloader plugin, this allows you for example to use charts hosted on Amazon S3 or Google Cloud Storage.

> **Note:** the operator only offers support for *downloader plugins*, other plugins will not be recognized nor used.

Plugin folder paths per Helm version:

Install a Helm downloader plugin

The easiest way to install a plugin so that it becomes accessible to the Helm operator to use an init container and one of the available `helm` binaries in the operator's image and a volume mount. For the Helm chart of the operator, see the documentation.

Using an init container

Create a volume entry of type `emptyDir` to the deployment of your Helm operator, this is where the plugins will be stored for the lifetime duration of the pod.

```yaml
spec:
  volumes:
  - name: helm-plugins-cache
    emptyDir: {}
```

Add a new init container that utilizes the same image as the operator's container and makes use of the earlier mentioned volume with correct volume mounts for the Helm version your are making use of. The available `helm2` and `helm3` binaries can then be used to install the plugin.

```yaml
spec:
  initContainers:
  - name: helm-3-downloader-plugin
    image: docker.io/fluxcd/helm-operator:<tag>
    imagePullPolicy: IfNotPresent
    command:
      - 'sh'
      - '-c'
      # Replace '<plugin>' and '<version>' with the respective
      # values of the plugin you want to install
      - 'helm3 plugin install <plugin> --version <version>'
    volumeMounts:
    - name: helm-plugins-cache
      # See: 'plugin folder paths per Helm version'
      mountPath: /root/.cache/helm/plugins
```

(continues on next page)

(continued from previous page)

```
      subPath: v3
    - name: helm-plugins-cache
      # See: 'plugin folder paths per Helm version'
      mountPath: /root/.local/share/helm/plugins
      subPath: v3-config
```

Last, add the same volume mounts to the operator's container so the downloaded plugin becomes available.

```
spec:
  containers:
  - name: flux-helm-operator
    image: docker.io/fluxcd/helm-operator:<tag>
    ...
    volumeMounts:
    - name: helm-plugins-cache
      # See: 'plugin folder paths per Helm version'
      mountPath: /root/.cache/helm/plugins
      subPath: v3
    - name: helm-plugins-cache
      # See: 'plugin folder paths per Helm version'
      mountPath: /root/.local/share/helm/plugins
      subPath: v3-config
```

Using an installed protocol in your `HelmRelease`

Once a Helm downloader plugin has been successfully installed, the newly added protocol can be used in the `.spec.chart.repository` value of a `HelmRelease`.

> **Note:** most downloader plugins expect some form of authentication to be available to be able to download a chart, make sure those are available in the operator's container before attempting to make use of the newly added protocol.

```
spec:
  chart:
    repository: s3://bucket-name/charts
    name: chart-name
    version: 1.0.0
```

1.2.3 Supplying values to the chart

You can supply values to be used with the chart when installing it, in two ways.

`.spec.values`

This is a YAML map as you'd put in a file and supply to Helm with `-f values.yaml`, but inlined into the `HelmRelease` manifest. For example,

```
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
# metadata: ...
spec:
  # chart: ...
```

(continues on next page)

(continued from previous page)

```
  values:
    foo: value1
    bar:
    baz: value2
    oof:
    - item1
    - item2
```

.spec.valuesFrom

This is a list of secrets, config maps (in the same namespace as the `HelmRelease` by default, or in a configured namespace) or external sources (URLs) from which to take values.

The values are merged in the order given, with later values overwriting earlier. These values always have a lower priority than those passed via the `.spec.values` parameter.

This is useful if you want to have defaults such as the `region`, `clustername`, `environment`, a local docker registry URL, etc., or if you simply want to have values not checked into git as plaintext.

Config maps

```
spec:
  # chart: ...
  valuesFrom:
  - configMapKeyRef:
      # Name of the config map
      name: default-values  # mandatory
      # Namespace of the config map
      namespace: my-ns      # optional; defaults to HelmRelease namespace
      # Key in the config map to get the values from
      key: values.yaml      # optional; defaults to values.yaml
      # If set to true successful retrieval of the values file is no
      # longer mandatory
      optional: false       # optional; defaults to false
```

Secrets

```
spec:
  # chart: ...
  valuesFrom:
  - secretKeyRef:
      # Name of the secret
      name: default-values # mandatory
      # Namespace of the secret
      namespace: my-ns      # optional; defaults to HelmRelease namespace
      # Key in the secret to get thre values from
      key: values.yaml      # optional; defaults to values.yaml
      # If set to true successful retrieval of the values file is no
      # longer mandatory
      optional: true        # optional; defaults to false
```

External sources

```
spec:
  # chart: ...
  valuesFrom:
  - externalSourceRef:
      # URL of the values.yaml
      url: https://example.com/static/raw/values.yaml # mandatory
      # If set to true successful retrieval of the values file is no
      # longer mandatory
      optional: true                                  # optional; defaults to␣
↪false
```

Chart files

```
spec:
  # chart: ...
  valuesFrom:
  - chartFileRef:
      # path within the helm chart (from git repo) where environment-prod.yaml is␣
↪located
      path: overrides/environment-prod.yaml # mandatory
      # If set to true successful retrieval of the values file is no
      # longer mandatory
      optional: true                                  # optional; defaults to␣
↪false
```

1.2.4 Rollbacks

From time to time a release made by the Helm operator may fail, it is possible to automate the rollback of a failed release by setting `.spec.rollback.enable` to `true` on the `HelmRelease` resource.

> **Note:** a successful rollback of a Helm chart containing a `StatefulSet` resource is known to be tricky, and one of the main reasons automated rollbacks are not enabled by default for all `HelmReleases`. Verify a manual rollback of your Helm chart does not cause any problems before enabling it.

When enabled, the Helm operator will detect a faulty upgrade and perform a rollback, it will not attempt a new upgrade unless it detects a change in values and/or the chart.

Configuration

```
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
# metadata: ...
spec:
  # Listed values are the defaults.
  rollback:
    # If set, will perform rollbacks for this release.
    enable: false
    # If set, will force resource update through delete/recreate if
    # needed.
    force: false
```

(continues on next page)

(continued from previous page)

```
    # Prevent hooks from running during rollback.
    disableHooks: false
    # Time in seconds to wait for any individual Kubernetes operation.
    timeout: 300
    # If set, will wait until all Pods, PVCs, Services, and minimum
    # number of Pods of a Deployment are in a ready state before
    # marking the release as successful. It will wait for as long
    # as the set timeout.
    wait: false
```

1.2.5 Reinstalling a Helm release

If a Helm release upgrade fails due to incompatible changes like modifying an immutable field (e.g. headless svc to ClusterIP)you can reinstall it using the following command:

```
$ kubectl delete hr/my-release
```

When the Helm Operator receives a delete event from Kubernetes API it will call Tiller and purge the Helm release. On the next Flux sync, the Helm Release object will be created and the Helm Operator will install it.

1.2.6 Authentication

At present, per-resource authentication is not implemented. The `HelmRelease` definition includes a field `chartPullSecret` for attaching a `repositories.yaml` file, but this is ignored for now.

Instead, you need to provide the operator with credentials and keys (see the following *Authentication for Helm repos* section for how to do this).

Authentication for Helm repos

As a workaround, you can mount a `repositories.yaml` file with authentication already configured, into the operator container.

> **Note:** When using a custom `repositories.yaml` the default that ships with the operator is overwritten. This means that for any repository you want to make use of you should manually add an entry to your `repositories.yaml` file.

To prepare a file, add the repo *locally* as you would normally:

```
helm repo add <URL> --username <username> --password <password>
```

You need to doctor this file a little, since it will likely contain absolute paths that will be wrong when mounted inside the container. Copy the file and replace all the `cache` entries with just the filename.

```
cp ~/.helm/repository/repositories.yaml .
sed -i -e 's/^\( *cache: \).*\/\(.*\.yaml\)/\1\2/g' repositories.yaml
```

If you are using OSX and Helm 3 the command will be:

```
cp ~/Library/Preferences/helm/repositories.yaml
```

Now you can create a secret in the same namespace as you're running the Helm operator, from the repositories file:

```
kubectl create secret generic flux-helm-repositories --from-file=./repositories.yaml
```

Lastly, mount that secret into the container. This can be done by setting `helmOperator.configureRepositories.enable` to `true` for the flux Helm release, or as shown in the commented-out sections of the example deployment.

Azure ACR repositories

For Azure ACR repositories, the entry in `repositories.yaml` created by running `az acr helm repo add` is unsufficient for the Helm operator. Instead you will need to create a service principal and use the plain text id and password this gives you. For example:

```
- caFile: ""
  cache: <repository>-index.yaml
  certFile: ""
  keyFile: ""
  name: <repository>
  url: https://<repository>.azurecr.io/helm/v1/repo
  username: <service principal id>
  password: <service principal password>
```

Authentication for Git repos

In general, it's necessary to have an SSH key to clone a git repo. This is sometimes (e.g., on GitHub) called a "deploy key". To use a chart from git, the Helm Operator needs a key with read-only access.

To provide an SSH key, put the key in a secret under the entry `identity`, and mount it into the operator container as shown in the example deployment. The default ssh_config expects an identity file at `/etc/fluxd/ssh/identity`, which is where it'll be if you just uncomment the blocks from the example.

If you're using more than one repository, you may need to provide more than one SSH key. In that case, you can create a secret with an entry for each key, and mount that *as well as* an ssh_config file mentioning each key as an `IdentityFile`.

CHAPTER

TWO

GUIDES

2.1 Upgrading from Helm operator beta (>=0.5.0) to stable (>=1.0.0)

Due to the Flux CD project joining the CNCF Sandbox and the API becoming stable, the Helm operator has undergone changes that necessitate some changes to your `HelmRelease` resources.

The central difference is that the Helm operator now works with resources of the kind `HelmRelease` in the API version `helm.fluxcd.io/v1`, the format of the resource is backwards compatible.

Here are some things to know:

- The new operator will ignore the old custom resources (and the old operator will ignore the new resources).
- Deleting a resource while the corresponding operator is running will result in the Helm release also being deleted
- Deleting a `CustomResourceDefinition` will also delete all custom resources of that kind.
- If both operators are running and both new and old custom resources defining a release, the operators will fight over the release.

The safest way to upgrade is to avoid deletions and fights by stopping the old operator. Replacing it with the new one (e.g., by changing the deployment, or re-releasing the Flux chart with the new version) will have that effect.

Once the old operator is not running, it is safe to deploy the new operator, and start replacing the old resources with new resources. You can keep the old resources around during this process, since the new operator will ignore them.

2.1.1 Updating custom resources

> **Note:** once the new CRD is applied it is no longer possible to list your old and new `HelmRelease` resources with just `kubectl get <hr|helmrelease>`, due to them sharing the same names. It is however still possible to list them by their full name.

```
# Old `HelmRelease` resources
$ kubectl get helmreleases.flux.weave.works
# New `HelmRelease` resources
$ kubectl get helmreleases.helm.fluxcd.io
```

The only difference between the old resource format and the new is the changed API version.

Changing an old resource to a new resource is thus as simple as changing the `apiVersion` field to `helm.fluxcd.io/v1`.

As a full example, this is an old resource:

```
---
apiVersion: flux.weave.works/v1beta1
kind: HelmRelease
metadata:
  name: foobar
  namespace: foo-ns
spec:
 chart:
    git: git@example.com:user/repo
    path: charts/foobar
  values:
    image:
      repository: foobar
      tag: v1
```

The new custom resource would be:

```
---
apiVersion: helm.fluxcd.io/v1       # <- change API version
kind: HelmRelease
metadata:
  name: foobar
  namespace: foo-ns
spec:
 chart:
    git: git@example.com:user/repo
    path: charts/foobar
  values:
    image:
      repository: foobar
      tag: v1
```

2.1.2 Deleting the old resources

Once you have migrated all your `HelmRelease` resources to the new API version and domain. You can remove all of the old resources by removing the old Custom Resource Definition.

```
$ kubectl delete crd helmreleases.flux.weave.works
```

2.2 Upgrading from Helm operator alpha (<=0.4.0) to beta

The Helm operator has undergone changes that necessitate some changes to custom resources, and the deployment of the operator itself.

The central difference is that the Helm operator now works with resources of the kind `HelmRelease` in the API version `flux.weave.works/v1beta1`, which have a different format to the custom resources used by the old Helm operator (`FluxHelmRelease`).

Here are some things to know:

- The new operator will ignore the old custom resources (and the old operator will ignore the new resources).
- Deleting a resource while the corresponding operator is running will result in the Helm release also being deleted
- Deleting a `CustomResourceDefinition` will also delete all custom resources of that kind.

- If both operators are running and both new and old custom resources defining a release, the operators will fight over the release.

The safest way to upgrade is to avoid deletions and fights by stopping the old operator. Replacing it with the new one (e.g., by changing the deployment, or re-releasing the Flux chart with the new version) will have that effect.

Once the old operator is not running, it is safe to deploy the new operator, and start replacing the old resources with new resources. You can keep the old resources around during this process, since the new operator will ignore them.

2.2.1 Upgrading the operator deployment

Using the Flux chart

The chart (from v0.5.0, or from this git repo) provides the correct arguments to the operator; to upgrade, do

```
helm repo update

helm upgrade flux --reuse-values \
--set image.tag=1.8.1 \
--set helmOperator.tag=0.5.1 \
--namespace=flux \
fluxcd/flux --version 0.5.1
```

The chart will leave the old custom resource definition and custom resources in place. You will need to replace the individual resources, as described below.

Using manifests

You will need to adapt any existing manifest that you use to run the Helm operator. The arguments to the operator executable have changed, since it no longer needs the git repo to be specified (and in some cases, just to tidy up):

- the new operator does not use the `--git-url`, `--git-charts-path`, or `--git-branch` arguments, since the git repo and so on are provided in each custom resource.
- the `--queue-worker-count` argument has been removed
- the `--chart-sync-timeout` argument has been removed
- other arguments stay the same

It is entirely valid to run the operator with no arguments, which you may end up with after removing those mentioned above. It will work with the secrets mounted as for the old operator, to start off with, since it expects the SSH key for the git repo to be in the same place.

Once you want to use the new capabilities of the operator – e.g., releasing charts from Helm repos – you will probably need to adapt the manifest further. The *HelmRelease Custom Resource* and *operator* references explain all the details.

2.2.2 Updating custom resources

The main differences between the old resource format and the new are:

- the API version and kind have changed
- you can now specify a chart to release either as a path in a git repo, or a named, versioned chart from a Helm repo

Here is how to change an old resource to a new resource:

- change the `apiVersion` field to `flux.weave.works/v1beta1`
- change the `kind` field to `HelmRelease`
- you can remove the label `chart :` from the labels, if it's still there, just to tidy up (it doesn't matter if it's there or not)
- replace the field `chartGitPath`, with the structure:

```
chart:
  git: <URL to git repo>
  ref: <optional branch name>
  path: <path from top directory of git repo to chart directory>
```

- the `values`, `releaseName`, and `valueFileSecrets` can stay as they are.

Note that you now give the git repo URL and branch and full path in each custom resource, rather than supplying arguments to the Helm operator. (As you've been using the old operator, you'll only have one git repo for all charts – but now you can use different repos for charts!)

As a full example, this is an old resource:

```
---
apiVersion: helm.integrations.flux.weave.works/v1alpha2
kind: FluxHelmRelease
metadata:
  name: foobar
  namespace: foo-ns
spec:
  chartGitPath: foobar
  values:
    image:
      repository: foobar
      tag: v1
```

Say the arguments given to the old Helm operator were

```
args:
  - --git-url=git@example.com:user/repo
  - --git-charts-path=charts
  - --git-branch=master
```

Then the new custom resource would be:

```
---
apiVersion: flux.weave.works/v1beta1 # <- change API version
kind: HelmRelease                    # <- change kind
metadata:
  name: foobar
  namespace: foo-ns
spec:
  chart:
    git: git@example.com:user/repo # <- --git-url from operator args
    path: charts/foobar            # <- join --git-chart-path and chartGitPath
  values:
    image:
      repository: foobar
      tag: v1
```

CHAPTER

THREE

TUTORIALS

3.1 Get started with the Helm operator and Tiller

3.1.1 Install Helm / Tiller

Generate certificates for Tiller and Flux. This will provide a CA, servercerts for Tiller and client certs for Helm / Flux.

> **Note:** When creating the certificate for Tiller the Common Name should match the hostname you are connecting to from the Helm operator.

The following script can be used for that (requires cfssl):

```
# TILLER_HOSTNAME=<service>.<namespace>
export TILLER_HOSTNAME=tiller-deploy.kube-system
export TILLER_SERVER=server
export USER_NAME=flux-helm-operator

mkdir tls
cd ./tls

# Prep the configuration
echo '{"CN":"CA","key":{"algo":"rsa","size":4096}}' | cfssl gencert -initca - |␣
↪cfssljson -bare ca -
echo '{"signing":{"default":{"expiry":"43800h","usages":["signing","key encipherment",
↪"server auth","client auth"]}}}' > ca-config.json

# Create the tiller certificate
echo '{"CN":"'$TILLER_SERVER'","hosts":[""],"key":{"algo":"rsa","size":4096}}' |␣
↪cfssl gencert \
  -config=ca-config.json -ca=ca.pem \
  -ca-key=ca-key.pem \
  -hostname="$TILLER_HOSTNAME" - | cfssljson -bare $TILLER_SERVER

# Create a client certificate
echo '{"CN":"'$USER_NAME'","hosts":[""],"key":{"algo":"rsa","size":4096}}' | cfssl␣
↪gencert \
  -config=ca-config.json -ca=ca.pem -ca-key=ca-key.pem \
  -hostname="$TILLER_HOSTNAME" - | cfssljson -bare $USER_NAME
```

Alternatively, you can follow the Helm documentation for configuring TLS.

Next create the RBAC configuration for Tiller:

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: tiller
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: tiller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: tiller
    namespace: kube-system

---
# Helm client serviceaccount
apiVersion: v1
kind: ServiceAccount
metadata:
  name: helm
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: Role
metadata:
  name: tiller-user
  namespace: kube-system
rules:
- apiGroups:
  - ""
  resources:
  - pods/portforward
  verbs:
  - create
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - list
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: RoleBinding
metadata:
  name: tiller-user-binding
  namespace: kube-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tiller-user
subjects:
- kind: ServiceAccount
```

(continues on next page)

(continued from previous page)

```
  name: helm
  namespace: kube-system
```

Save the above config as `helm-rbac.yaml` and deploy Tiller:

```
kubectl apply -f helm-rbac.yaml

# Deploy helm with mutual TLS enabled.
# --history-max limits the maximum number of revisions Tiller stores;
# leaving it to the default (0) may result in request timeouts after N
# releases, due to the excessive amount of ConfigMaps Tiller will
# attempt to retrieve.
helm init --upgrade --service-account tiller --history-max 10 \
    --override 'spec.template.spec.containers[0].command'='{/tiller,--storage=secret}
↪' \
    --tiller-tls \
    --tiller-tls-cert ./tls/server.pem \
    --tiller-tls-key ./tls/server-key.pem \
    --tiller-tls-verify \
    --tls-ca-cert ./tls/ca.pem
```

To check if Tiller installed succesfully with TLS enabled, try `helm ls`. This should give an error:

```
# Should give an error
$ helm ls
Error: transport is closing
```

When providing the certificates, it should work correctly:

```
helm --tls --tls-verify \
  --tls-ca-cert ./tls/ca.pem \
  --tls-cert ./tls/flux-helm-operator.pem \
  --tls-key ./tls/flux-helm-operator-key.pem \
  --tls-hostname tiller-deploy.kube-system \
  ls
```

3.1.2 Deploy the Helm Operator

First create a new Kubernetes TLS secret for the client certs:

```
kubectl create secret tls helm-client --cert=tls/flux-helm-operator.pem --key=./tls/
↪flux-helm-operator-key.pem
```

Note: this has to be in the same namespace as the flux-helm-operator is deployed in.

Deploy Flux with Helm:

```
helm repo add fluxcd https://fluxcd.github.io/flux

helm upgrade --install \
    --set helmOperator.create=true \
    --set helmOperator.createCRD=true \
    --set git.url=$YOUR_GIT_REPO \
    --set helmOperator.tls.enable=true \
    --set helmOperator.tls.verify=true \
```

(continues on next page)

(continued from previous page)

```
    --set helmOperator.tls.secretName=helm-client \
    --set helmOperator.tls.caContent="$(cat ./tls/ca.pem)" \
    flux \
    fluxcd/flux
```

Note:

- include –tls flags for `helm` as in the `helm ls` example, if talking to a tiller with TLS
- optionally specify target –namespace

3.1.3 Check if it worked

Use `kubectl logs` on the Helm Operator and observe the helm client being created.

3.2 How to bootstrap Helm operator using Kustomize

This guide shows you how to use Kustomize to bootstrap Helm Operator on a Kubernetes cluster.

3.2.1 Prerequisites

You will need to have Kubernetes set up. For a quick local test, you can use `minikube`, `kubeadm` or `kind`. Any other Kubernetes setup will work as well though.

If working on e.g. GKE with RBAC enabled, you will need to add a cluster role binding:

```
kubectl create clusterrolebinding "cluster-admin-$(whoami)" \
    --clusterrole=cluster-admin \
    --user="$(gcloud config get-value core/account)"
```

3.2.2 Prepare Helm Operator installation

Create a directory called `fluxcd` and add the `flux` namespace definition to it:

```
mkdir fluxcd

cat > fluxcd/namespace.yaml <<EOF
apiVersion: v1
kind: Namespace
metadata:
  name: flux
EOF
```

Create the `repositories.yaml` file and add the stable, flagger and podinfo Helm repositories to it:

```
cat > fluxcd/repositories.yaml <<EOF
apiVersion: v1
repositories:
- name: stable
  url: https://kubernetes-charts.storage.googleapis.com
  cache: /var/fluxd/helm/repository/cache/stable-index.yaml
```

(continues on next page)

(continued from previous page)

```
- name: flagger
  url: https://flagger.app
  cache: /var/fluxd/helm/repository/cache/flagger-index.yaml
- name: podinfo
  url: https://stefanprodan.github.io/podinfo
  cache: /var/fluxd/helm/repository/cache/podinfo-index.yaml
EOF
```

Create a kustomization file and use the Helm operator deploy YAMLs as base:

```
cat > fluxcd/kustomization.yaml <<EOF
namespace: flux
resources:
  - namespace.yaml
bases:
 - github.com/fluxcd/helm-operator//deploy
secretGenerator:
  - name: helm-repositories
    files:
      - repositories.yaml
patchesStrategicMerge:
  - patch.yaml
EOF
```

> **Note:** If you want to install a specific Helm operator release, add the version number to the base URL: `github.com/fluxcd/helm-operator//deploy?ref=v1.0.0-rc2`

Create a patch file for Helm operator and mount the repositories secret:

```
cat > fluxcd/patch.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flux-helm-operator
spec:
  template:
    spec:
      volumes:
       - name: repositories-yaml
         secret:
           secretName: helm-repositories
       - name: repositories-cache
         emptyDir: {}
      containers:
        - name: flux-helm-operator
          volumeMounts:
             - name: repositories-yaml
               mountPath: /var/fluxd/helm/repository
             - name: repositories-cache
               mountPath: /var/fluxd/helm/repository/cache
EOF
```

3.2.3 Install Helm Operator with Kustomize

In the next step, deploy Flux to the cluster (you'll need kubectl **1.14** or newer):

```
kubectl apply -k fluxcd
```

Wait for Helm operator to start:

```
kubectl -n flux rollout status deployment/flux-helm-operator
```

3.2.4 Use the `HelmRelease` custom resource

Install podinfo by referring to its Helm repository:

```
cat <<EOF | kubectl apply -f -
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
metadata:
  name: podinfo
  namespace: default
spec:
  releaseName: podinfo
  chart:
    repository: https://stefanprodan.github.io/podinfo
    version: 2.1.0
    name: podinfo
  values:
    replicaCount: 1
EOF
```

Verify that the Helm Operator has installed the release:

```
kubectl get hr

NAME       RELEASE    STATUS     MESSAGE                   AGE
podinfo    podinfo    DEPLOYED   helm install succeeded   1m
```

Delete the release with:

```
kubectl delete hr/podinfo
```

3.2.5 Next steps

Try out fluxcd/helm-operator-get-started to learn more about Helm Operator capabilities.

CHAPTER

FOUR

FREQUENTLY ASKED QUESTIONS

4.1 I'm using SSL between Helm and Tiller. How can I configure the operator to use the certificate?

When installing Flux, you can supply the CA and client-side certificate using the `helmOperator.tls` options, more details *here*.

4.2 I've deleted a HelmRelease file from Git. Why is the Helm release still running on my cluster?

Flux doesn't delete resources by default, you can enable Flux garbage collection feature by passing the command-line flag `--sync-garbage-collection` to `fluxd`.

With Flux garbage collection enabled, Helm operator will receive the delete event and will purge the Helm release.

4.3 I've manually deleted a Helm release. Why is Flux not able to restore it?

If you delete a Helm release with `helm delete my-release`, the release name can't be reused. You need to use the `helm delete --purge` option only then Flux will be able reinstall a release.

4.4 It takes a long time before the operator processes an update to a `HelmRelease` resource. How can I speed up the processing of releases?

The operator watches for changes of Custom Resources of kind `HelmRelease`. It receives Kubernetes Events and queues these to be processed, all resources will also be re-queued every `--charts-sync-interval` (default `3m`) for a dry-run to detect and revert manual changes made to a release.

Depending on how many resources the operator is watching and the complexity of the charts (umbrella charts for example generally take a longer time to process); the default amount of workers may not be sufficient to instantly process a release the moment it enters the queue as the queue works on a FIFO basis, and other releases may have to be processed first.

The solution is to increase the amount of workers processing the releases using the `--workers` flag (default `2`), i.e. by steps of `2` until the releases are processed within a time frame that feels right to you.

If this does not give the desired effect, or if the amount of workers required is incredible high, there are two other tweaks possible:

1. increasing the `--charts-sync-interval`; this causes the queue to be less heavy occupied at the cost of detecting mutations less rapidly
2. using multiple Helm operator instances, i.e. by having one operator per namespace; namespace scoping is possible by configuring the `--allow-namespace` flag

4.5 I have a dedicated Kubernetes cluster per environment and I want to use the same Git repo for all. How can I do that?

Option 1 For each cluster create a directory in your config repo. When installing Flux Helm chart set the Git path using `--set git.path=k8s/cluster-name` and set a unique label for each cluster `--set git.label=cluster-name`.

You can have one or more shared dirs between clusters. Assuming your shared dir is located at `k8s/common` set the Git path as `--set git.path="k8s/common\,k8s/cluster-name"`.

Option 2 For each cluster create a Git branch in your config repo. When installing Flux Helm chart set the Git branch using `--set git.branch=cluster-name` and set a unique label for each cluster `--set git.label=cluster-name`.

4.6 Are there prerelease builds I can run?

There are builds from CI for each merge to master branch. See fluxcd/helm-operator-prerelease.

CHAPTER

FIVE

TROUBLESHOOTING

Also see the issues labeled with `question`, which often explain workarounds.

5.1 Error creating helm client: failed to append certificates from file: /etc/fluxd/helm-ca/ca.crt

Your CA certificate content is not set correctly, check if your ConfigMap contains the correct values. Example:

```
$ kubectl get configmaps flux-helm-tls-ca-config -o yaml
apiVersion: v1
data:
  ca.crt: |
    -----BEGIN CERTIFICATE-----
    ....
    -----END CERTIFICATE-----
kind: ConfigMap
metadata:
  creationTimestamp: 2018-07-04T15:27:25Z
  name: flux-helm-tls-ca-config
  namespace: helm-system
  resourceVersion: "1267257"
  selfLink: /api/v1/namespaces/helm-system/configmaps/flux-helm-tls-ca-config
  uid: c106f866-7f9e-11e8-904a-025000000001
```