Deploying Applications in Kubernetes Using Flux

Pavan Kumar
Dec 6, 2020 · 6 min read

Introduction to Flux

Flux is an Open and extensible continuous delivery solution for Kubernetes. Flux is a GitOps tool for Kubernetes that synchronizes the state of manifests in a Git repository to what is running in a cluster. So what is GitOps? Is it a new tool in the market? GitOps provides a way for developers to manage operational workflow for using Kubernetes using Git. It is all about using a version-controlled system for the deployment of applications in Kubernetes. So Developers can directly push the code into production from the version-controlled system like Git. Moreover, any changes made can be easily tracked and reverted in case of any chaos. There are multiple tools in the market to run GitOps. Today in this article we would be experimenting with a tool called Flux.

Features of Flux:

1. Automated synchronization between a version control repository and a cluster.
2. Any changes made to the repository are instantly reflected in the cluster.
3. Developers can directly push the code into production from the repositories.
4. All the configuration is stored in the version control system and is up to date.
5. Built-in support for kustomize and helm.
6. It can also be integrated with **flagger**.
7. In case of a disaster, the new cluster can be brought up with the same configuration.

flux

What is the entire story all about? (TLDR)

1. We will be using Flux to synchronize the Helm Charts stored in a version control system to our Kubernetes cluster.
2. We will use HelmRelease (CRD) with Flux.

Story Resources

1. **GitHub Link**: https://github.com/pavan-kumar-99/medium-manifests
2. **GitHub Branch**: fluxcd-demo

Installing Flux

Let us now Install fluxcd in our Kubernetes cluster using a helm chart. If you are not familiar with what a helm chart is, refer to this guide. Before we Install fluxcd we will have to Install the HelmRelease CRD (Explained later in the article).

```
helm repo add fluxcd https://charts.fluxcd.io
#Adding the Flux Repo

kubectl apply -f https://raw.githubusercontent.com/fluxcd/helm-operator/master/deploy/crds.yaml
#Installing the HelmRelease CRD

kubectl create namespace flux
#Create the namespace for flux Installation
```

Flux connects to the Git Repository using an ssh key. If the ssh key already exists, A Kubernetes secret can be created from the key. Else configure the key with your GitHub given by fluxcd after installation. Since I already have an existing key pair I would be creating a Kubernetes Secret from the Private Key.

```
kubectl create secret generic flux-git-deploy --from-
file=identity=./id_rsa -n flux --dry-run=client -o yaml | kubectl
apply -f -
#This would create the kubernetes secret for flux to communicate
with GitHub
```

Since we have made the configuration for our flux deployment to communicate with our git repo let us deploy fluxcd and HelmOperator deployment.

```
helm install flux fluxcd/flux --set git.url=git@github.com:pavan-
kumar-99/medium-manifests.git --set git.branch=fluxcd --set
git.secretName="flux-git-deploy" --set git.user=flux-user --set
git.path=helm-releases --namespace flux

#Install fluxcd deployment

helm upgrade -i helm-operator fluxcd/helm-operator --set
git.ssh.secretName=flux-git-deploy --namespace flux

#Install helm-operator deployment

kubectl create ns fluxcd-demo

#Create a namespace to deploy our HelmRelease
```

Helm Operator

The Helm Operator is a Kubernetes Operator, allowing one to declaratively manage Helm chart releases. The desired state of a Helm release is described through a Kubernetes Custom Resource named **HelmRelease.** . Based on the creation, mutation, or removal of a **HelmRelease** resource in the cluster, Helm actions are performed by the operator.

Fluxcd with helm operator

pavan-kumar-99/medium-manifests

This repo contains the files for Kustomization demo GitHub is home to over 50 million developers working together to...

github.com

Here is a sample repo which contains some sample helm charts and a sample HelmRelease file. We would now understand what is written in the HelmRelease file.

```yaml
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
metadata:
  name: fluxcd-demo
  namespace: default
  annotations:
    fluxcd.io/automated: "true"
spec:
  releaseName: fluxcd-demo
  targetNamespace: fluxcd-demo
  chart:
    git: git@github.com:pavan-kumar-99/medium-manifests.git
    path: helm-charts/fluxcd-demo
    ref: fluxcd-demo
```

flux-demo-release.yaml hosted with ♡ by GitHub view raw

1. kind: HelmRelease (Kubernetes CRD).
2. metadata.name: The name of the HelmRelease.
3. metadata.namespace: The namespace in which the HelmRelease is supposed to be deployed in.
4. metadata.annotations: **fluxcd.io/automated**: To enable automation for fluxcd.

5. spec.releaseName: The name of the helm chart release name.
6. spec.targetNamespace: The namespace into which the helm chart has to be installed. (Make sure you create the namespace before the HelmRelease gets Installed)
7. spec.chart.git: The Git Repository URL from which the helm charts has to be installed.
8. spec.chart.path: The path from GitHub Repository.
9. spec.chart.ref: The name of the GitHub branch.

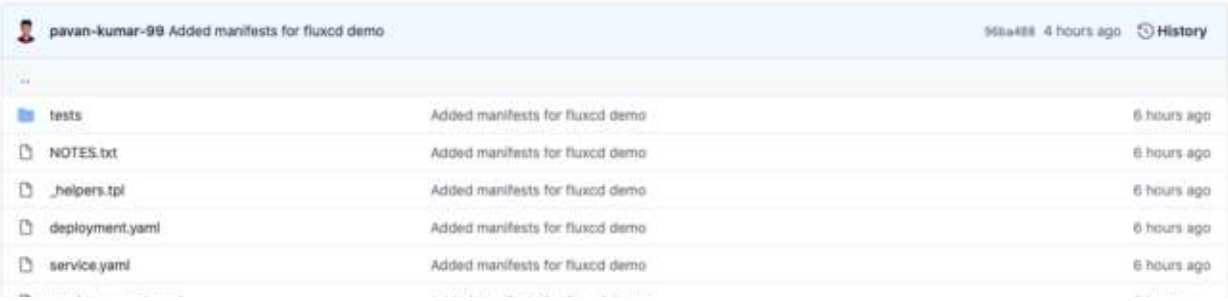
Demo

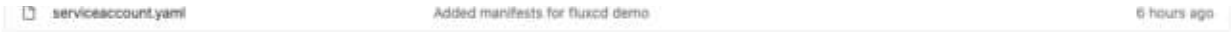
Once the fluxcd and helm operator charts are installed you should see the flux components created in the **flux** namespace.

```
[root@master ~]# kubectl get all -n flux
NAME                                    READY   STATUS    RESTARTS   AGE
pod/flux-65d8fb5d-hvgbb                 1/1     Running   0          3h7m
pod/flux-memcached-869757cb88-rmtdw     1/1     Running   0          7h51m
pod/helm-operator-686dc669cd-w6csj      1/1     Running   0          7h4m

NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)     AGE
service/flux            ClusterIP   10.107.115.197   <none>        3030/TCP    7h51m
service/flux-memcached  ClusterIP   10.106.10.59     <none>        11211/TCP   7h51m
service/helm-operator   ClusterIP   10.111.179.65    <none>        3030/TCP    7h6m

NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/flux            1/1     1            1           7h51m
deployment.apps/flux-memcached  1/1     1            1           7h51m
deployment.apps/helm-operator   1/1     1            1           7h6m

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/flux-65d8fb5d              1         1         1       3h7m
replicaset.apps/flux-memcached-869757cb88  1         1         1       7h51m
replicaset.apps/helm-operator-686dc669cd   1         1         1       7h4m
[root@master ~]#
```

fluxcd components

Now go grab a cup of coffee and wait for 5 minutes. You should now have all your resources created in your cluster defined in the helm chart. These are the resources defined in our helm chart.





Resources in the helm chart

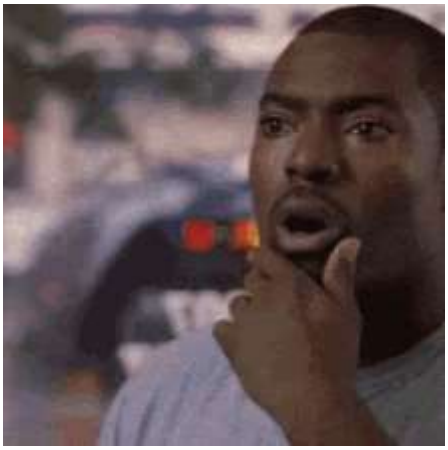
Let us watch the resources in the fluxcd-demo namespace (spec.targetNamespace-> HelmRelease file)

watch -n 5 kubectl get all -n fluxcd-demo

```
[root@master ~]# kubectl get all -n fluxcd-demo
NAME                                          READY   STATUS    RESTARTS   AGE
pod/fluxcd-demo-helm-release-69cb948496-nzsff 1/1     Running   0          3h14m

NAME                              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/fluxcd-demo-helm-release  ClusterIP   10.104.60.226   <none>        80/TCP    3h14m

NAME                                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/fluxcd-demo-helm-release  1/1     1            1           3h14m

NAME                                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/fluxcd-demo-helm-release-69cb948496  1         1         1       3h14m
[root@master ~]#
```

fluxcd-demo namespace

And now you have all the resources defined in the helm chart created in your Kubernetes Cluster.



Woooo !!!

As we know that fluxcd will watch, pick up the changes from git, and will update our cluster, let us update the number of replicas of fluxcd-demo deployment.

```
[root@master ~]# kubectl get deploy -n fluxcd-demo
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
fluxcd-demo-helm-release  1/1     1            1           3h34m
[root@master ~]#
```

Number of replicas = 1

Let us edit our HelmRelease manifests to override the values defined in the values.yaml file of the Helm Chart.

```yaml
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
metadata:
  name: fluxcd-demo
  namespace: default
  annotations:
    fluxcd.io/automated: "true"
spec:
  releaseName: fluxcd-demo
  targetNamespace: fluxcd-demo
  chart:
    git: git@github.com:pavan-kumar-99/medium-manifests.git
    path: helm-charts/fluxcd-demo
    ref: fluxcd-demo
  values:
   replicaCount: 5
```

flux-demo-release.yaml hosted with ♡ by GitHub view raw

I have updated the number of replicas to 5 in GitHub by overriding the replicaCount value in the HelmRelease file.

Now go grab another cup of coffee and wait for 5 minutes.

```
[root@master ~]# kubectl get deploy -n fluxcd-demo
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
fluxcd-demo-helm-release   5/5     5            5           3h49m
[root@master ~]#
```

Number of replicas = 5

We have now successfully deployed our first HelmRelease using fluxcd.

Conclusion

Thanks for reading my article. Here are some of my other articles that may interest you.

Reference

https://docs.fluxcd.io/en/latest/tutorials/get-started/

Flux Helm Operator

The Helm Operator is a Kubernetes operator, allowing one to declaratively manage Helm chart releases. Combined with...

docs.fluxcd.io

Recommended

Introduction to Kustomize

How to use Kustomize to efficiently manage Your Kubernetes manifests.

pavan1999-kumar.medium.com

Create a Kubernetes Cluster using Kind

How to create a Kubernetes cluster in 5 minutes using kind.

pavan1999-kumar.medium.com

Introduction to Istio Service Mesh

What is Istio Service Mesh?

pavan1999-kumar.medium.com

Securing your secrets using vault-k8s in Kubernetes — Part 1

Kubernetes secrets let you store and manage sensitive data such as passwords, ssh keys, TIs certificates, etc. However...

pavan1999-kumar.medium.com

Deploying applications in Kubernetes using Argo CD

Deploying applications in Kubernetes using Argo CD

Deploying applications in Kubernetes using Argo CDpavan1999-kumar.medium.com

Gitops Kubernetes Helm Flux Github