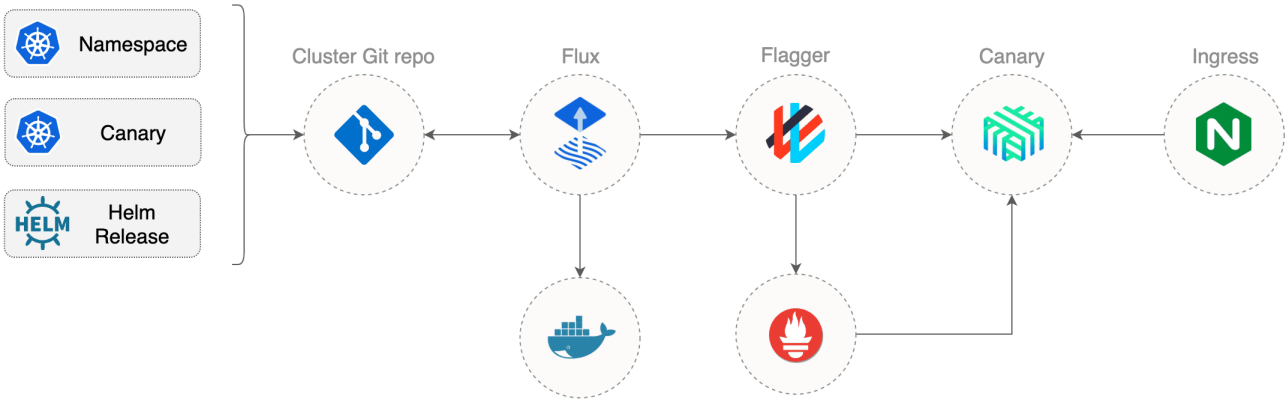
GitOps Helm Workshop



Progressive Delivery for Kubernetes

Welcome to the GitOps Helm Workshop

Get Started →

Flux CD

Flux is a Kubernetes controller that automatically ensures that the state of a cluster matches the config in git. Helm Operator is a Kubernetes CRD controller that manages the Helm release lifecycle.

Linkerd

Linkerd v2 is a lightweight, easy to install and maintain, service mesh for Kubernetes. For HTTP and gRPC apps, Linkerd automatically enables load balancing tracing Prometheus metrics and mTLS with zero configuration

Flagger

Flagger is a Kubernetes operator that automates the promotion of canary deployments using Linkerd routing for traffic shifting, Prometheus metrics for canary analysis and Helm for testing.

Introduction

This guide walks you through setting up a progressive delivery GitOps pipeline on a Kubernetes cluster.

What is GitOps?

GitOps is a way to do Continuous Delivery, it works by using Git as a source of truth for declarative infrastructure and workloads. For Kubernetes this means using `git push` instead of `kubectl create/apply` or `helm install/upgrade` .

> **GitOps vs CiOps**
>
> In a traditional CI/CD pipeline, CD is an implementation extension powered by the continuous integration tooling to promote build artifacts to production. In the GitOps pipeline model, any change to production must be committed in source control (preferable via a pull request) prior to being applied on the cluster. If the entire production state is under version control and described in a single Git repository, when disaster strikes, the whole infrastructure can be quickly restored without rerunning the CI pipelines.
>
> Kubernetes anti-patterns: Let's do GitOps, not CIOps!

In order to apply the GitOps model to Kubernetes you need three things:

- a Git repository with your workloads definitions in YAML format, Helm charts and any other Kubernetes custom resource that defines your cluster desired state
- a container registry where your CI system pushes immutable images (no *latest* tags, use *semantic versioning* or git *commit sha*)
- a Kubernetes controller that does a two-way synchronization:
  - watches for changes in the config repository and applies them to your cluster
  - watches the container registry for new images and updates the workload definitions based on deployment policies

What is Progressive Delivery?

Progressive delivery is an umbrella term for advanced deployment patterns like canaries, feature flags and A/B testing. Progressive delivery techniques are used to reduce the risk of introducing a new software version in production by giving app developers and SRE teams a fine-grained control over the blast radius.

> **Canary release**
>
> A benefit of using canary releases is the ability to do capacity testing of the new version in a production environment with a safe rollback strategy if issues are found. By slowly ramping up the load, you can monitor and capture metrics about how the new version impacts the production environment.
>
> Martin Fowler blog

In this workshop you'll be using Flagger, Linkerd and Prometheus to automate canary releases for Helm charts.

← Home

Prerequisites →

Prerequisites

In order to install the workshop prerequisites you'll need a Kubernetes cluster **1.13** or newer with **Load Balancer** support and **RBAC** enabled. Make sure you have the following tools installed locally:

- kubectl 1.14
- git 2.20

Helm v3

Download the Helm v3 CLI:

```sh
OS=darwin-amd64 && \
mkdir -p $HOME/.helm3/bin && \
curl -sSL "https://get.helm.sh/helm-v3.0.0-beta.3-${OS}.tar.gz" | tar xvz && \
chmod +x ${OS}/helm && mv ${OS}/helm $HOME/.helm3/bin/helmv3
```

Add the helmv3 binary to your path and set Helm home:

```sh
export PATH=$PATH:$HOME/.helm3/bin
export HELM_HOME=$HOME/.helm3
```

Verify the installation with:

```sh
helmv3 version
```

Git

```sh
export GHUSER=stefanprodan
git clone https://github.com/${GHUSER}/gitops-helm-workshop
```

Set your GitHub username and email:

```sh
cd gitops-helm-workshop
git config user.name "${GHUSER}"
git config user.email "your@main.address"
```

Cluster state directory structure:

```
├── cluster
    ├── canaries
    ├── charts
    │   └── podinfo
    ├── namespaces
    └── releases
```

Flux

Add FluxCD repository to Helm repos:

```sh
helmv3 repo add fluxcd https://charts.fluxcd.io
```

Create the fluxcd namespace:

```sh
kubectl create ns fluxcd
```

Install Flux by providing your GitHub repository URL:

```
--namespace fluxcd \
--set registry.pollInterval=1m \
--set git.pollInterval=1m \
--set git.url=git@github.com:${GHUSER}/gitops-helm-workshop
```

Install fluxctl:

```sh
# macOS and linux
curl -sL https://fluxcd.io/install | sh
export PATH=$PATH:$HOME/.fluxcd/bin

# windows
https://github.com/fluxcd/flux/releases
```

Find the Git SSH public key:

```sh
export FLUX_FORWARD_NAMESPACE=fluxcd

fluxctl identity
```

Copy the public key and create a deploy key with write access on your GitHub repository. Go to `Settings > Deploy keys` click on `Add deploy key`, check `Allow write access`, paste the Flux public key and click `Add key`.

Helm Operator

Install the HelmRelease CRD:

```sh
kubectl apply -f https://raw.githubusercontent.com/fluxcd/helm-operator/helm-v3/
```

Install Flux Helm Operator in the `fluxcd` namespace:

```sh
helmv3 upgrade -i helm-operator fluxcd/helm-operator --wait \
--namespace fluxcd \
```

```
--set git.ssh.secretName=flux-git-deploy \
--set git.pollInterval=1m \
--set chartsSyncInterval=1m \
--set configureRepositories.enable=true \
--set configureRepositories.repositories[0].name=stable \
--set configureRepositories.repositories[0].url=https://kubernetes-charts.storag
--set extraEnvs[0].name=HELM_VERSION \
--set extraEnvs[0].value=v3 \
--set image.repository=docker.io/fluxcd/helm-operator-prerelease \
--set image.tag=helm-v3-71bc9d62
```

Linkerd

Download the Linkerd v2 CLI:

```sh
# macOS and linux
curl -sL https://run.linkerd.io/install | sh
export PATH=$PATH:$HOME/.linkerd2/bin

# windows
https://github.com/linkerd/linkerd2/releases
```

Install the Linkerd control plane in the `linkerd` namespace:

```sh
linkerd install | kubectl apply -f -
```

Validate the install with:

```sh
linkerd check
```

Flagger

Add Flagger Helm repository:

```sh
helmv3 repo add flagger https://flagger.app
```

```sh
kubectl apply -f https://raw.githubusercontent.com/weaveworks/flagger/master/art
```

Install Flagger in the `linkerd` namespace:

```sh
helmv3 upgrade -i flagger flagger/flagger --wait \
--namespace linkerd \
--set crd.create=false \
--set metricsServer=http://linkerd-prometheus:9090 \
--set meshProvider=linkerd
```

← Introduction

Helm Releases →

Helm Releases

A chart release is described through a Kubernetes custom resource named **HelmRelease**.

A Helm release can refer a chart from:

- public or private Helm repositories over HTTPS
- public or private Git repositories over SSH

Install NGINX

To expose applications outside of the cluster you'll be using the NGINX ingress controller. The controller will run inside the Linkerd mesh.

Create a namespace with linkerd injection enabled:

```yaml
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    fluxcd.io/ignore: "false"
    linkerd.io/inject: enabled
  name: ingress-nginx
```

Create a Helm release to install the NGINX ingress controller:

```yaml
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
metadata:
  name: nginx-ingress
  namespace: ingress-nginx
  annotations:
    fluxcd.io/ignore: "false"
spec:
  releaseName: nginx-ingress
```

```
    name: nginx-ingress
    version: 1.19.0
  values:
    controller:
      service:
        type: LoadBalancer
```

Apply changes:

```sh
git add -A && \
git commit -m "install ingress" && \
git push origin master && \
fluxctl sync
```

Validate that the Helm operator has installed the release:

```sh
kubectl -n ingress-nginx get hr
```

Find the public IP of the ingress controller:

```sh
kubectl -n ingress-nginx get svc
```

Install podinfo

Podinfo is tiny Go web application. You'll be installing podinfo using a Helm chart stored in the git repository at `cluster/charts/podinfo`.

Create the `prod` namespace with linkerd injection enabled:

```yaml
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    fluxcd.io/ignore: "false"
```

GitOps Helm Workshop

Create a Helm release to install the podinfo chart (replace `GHUSER` with your GitHub username and `LB-PUBLIC-IP` with your ingress IP):

```yaml
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
metadata:
  name: podinfo
  namespace: prod
  annotations:
    fluxcd.io/ignore: "false"
spec:
  releaseName: podinfo
  chart:
    git: git@github.com:GHUSER/gitops-helm-workshop
    ref: master
    path: cluster/charts/podinfo
  values:
    image:
      repository: stefanprodan/podinfo
      tag: 3.0.0
    service:
      enabled: true
      type: ClusterIP
    ingress:
      enabled: true
      annotations:
        kubernetes.io/ingress.class: "nginx"
        nginx.ingress.kubernetes.io/configuration-snippet: |
          proxy_set_header l5d-dst-override $service_name.$namespace.svc.cluster
          proxy_hide_header l5d-remote-ip;
          proxy_hide_header l5d-server-id;
      path: /
      hosts:
        - LB-PUBLIC-IP.nip.io
```

Note that if you are on EKS, the host should be set to the `elb.amazonaws.com` address:

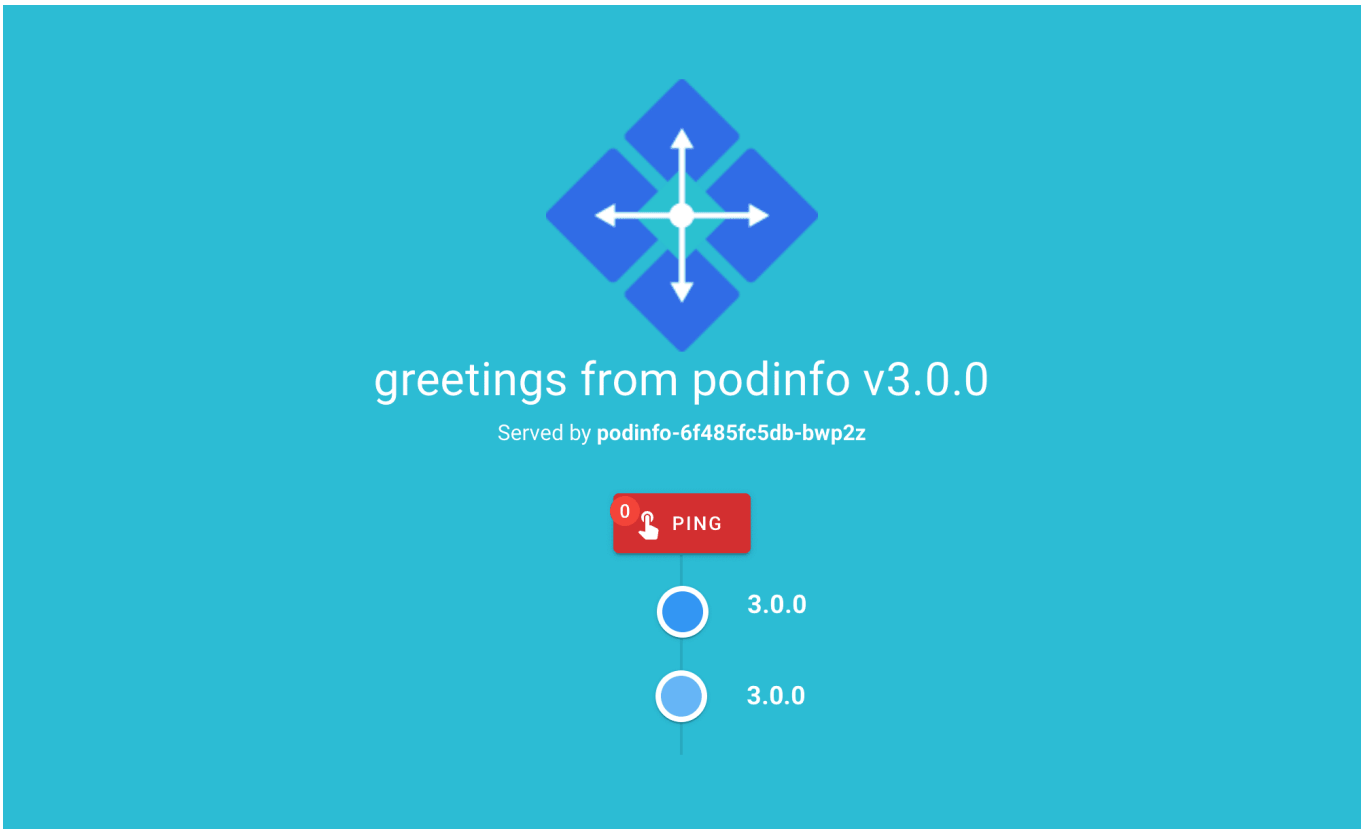
Apply changes:

```sh
git add -A && \
git commit -m "install podinfo" && \
git push origin master && \
fluxctl sync
```

Validate that the Helm operator has installed podinfo:

```sh
kubectl -n prod get hr
```

Open your browser and navigate to `http://LB-PUBLIC-IP.nip.io/` , you should see podinfo v3.0.0 UI.



Automated upgrade

Flux can be used to automate container image updates in your cluster. You can enable the automate image tag updates by annotating Helm release objects. You can also control what tags should be considered for an update by using glob, regex or semantic version expressions.

Edit the podinfo Helm release and enable Flux automated image updates:

```yaml
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
metadata:
  annotations:
    fluxcd.io/automated: "true"
    fluxcd.io/tag.chart-image: semver:~3.0
```

Apply changes:

```sh
git add -A && \
git commit -m "automate podinfo" && \
git push origin master && \
fluxctl sync
```

Validate that the Helm operator has upgraded podinfo:

```sh
kubectl -n prod get hr
```

Pull the changes made by Flux locally:

```sh
git pull origin master
```

Open your browser and navigate to `http://LB-PUBLIC-IP.nip.io/` , you should see podinfo v3.0.5 UI.

Sealed secrets

In order to store secrets safely in a public Git repo you can use the Sealed Secrets controller and encrypt your Kubernetes Secrets into **SealedSecrets**. The sealed secret can be decrypted only by the controller running in your cluster.

Create the Sealed Secrets Helm release:

yaml

```yaml
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
metadata:
  name: sealed-secrets
  namespace: fluxcd
  annotations:
    fluxcd.io/ignore: "false"
spec:
  releaseName: sealed-secrets
  chart:
    repository: https://kubernetes-charts.storage.googleapis.com/
    name: sealed-secrets
    version: 1.4.0
```

```sh
git add -A && \
git commit -m "install sealed-secrets" && \
git push origin master && \
fluxctl sync
```

Install the kubeseal CLI:

```sh
wget https://github.com/bitnami-labs/sealed-secrets/releases/download/v0.8.1/kub
sudo install -m 755 kubeseal-darwin-amd64 /usr/local/bin/kubeseal
```

At startup, the sealed-secrets controller generates a RSA key and logs the public key. Using kubeseal you can save your public key as pub-cert.pem, the public key can be safely stored in Git, and can be used to encrypt secrets without direct access to the Kubernetes cluster:

```sh
kubeseal --fetch-cert \
--controller-namespace=fluxcd \
--controller-name=sealed-secrets \
> pub-cert.pem
```

You can generate a Kubernetes secret locally with kubectl and encrypt it with kubeseal:

```sh
kubectl -n prod create secret generic basic-auth \
--from-literal=user=admin \
--from-literal=password=admin \
--dry-run \
-o json > basic-auth.json

kubeseal --format=yaml --cert=pub-cert.pem < basic-auth.json > basic-auth.yaml
```

This generates a custom resource of type SealedSecret that contains the encrypted credentials.

Flux will apply the sealed secret on your cluster and sealed-secrets controller will then decrypt it into a Kubernetes secret.

```sh
kubectl get secret -n fluxcd sealed-secrets-key -o yaml \
--export > sealed-secrets-key.yaml
```

To restore from backup after a disaster, replace the newly-created secret and restart the controller:

```sh
kubectl replace secret -n fluxcd sealed-secrets-key -f sealed-secrets-key.yaml
kubectl delete pod -n fluxcd -l app=sealed-secrets
```

← Prerequisites

Canary Releases →

Canary Releases

A canary release is described with a Kubernetes custom resource named **Canary**.

Application bootstrap

Edit the podinfo Helm release and disable the image updates and the ClusterIP service:

```yaml
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
metadata:
  name: podinfo
  namespace: prod
  annotations:
    fluxcd.io/automated: "false"
spec:
  releaseName: podinfo
  values:
    image:
      repository: stefanprodan/podinfo
      tag: 3.0.0
    service:
      enabled: false
      type: ClusterIP
```

Apply changes:

```sh
git add -A && \
git commit -m "prep canary" && \
git push origin master && \
fluxctl sync
```

Create a canary release for podinfo:

```
apiVersion: flagger.app/v1alpha3
kind: Canary
metadata:
  name: podinfo
  namespace: prod
  annotations:
    fluxcd.io/ignore: "false"
spec:
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: podinfo
  service:
    port: 9898
  canaryAnalysis:
    interval: 10s
    stepWeight: 5
    threshold: 5
    metrics:
    - name: request-success-rate
      threshold: 99
      interval: 1m
    - name: request-duration
      threshold: 500
      interval: 1m
    webhooks:
      - name: load-test
        url: http://load-tester.prod/
        metadata:
          cmd: "hey -z 2m -q 10 -c 2 http://podinfo:9898/"
```

Apply changes:

sh

```
git add -A && \
git commit -m "add canary" && \
git push origin master && \
fluxctl sync
```

Validate that Flagger has initialized the canary:

Automated canary promotion

Install the load testing service to generate traffic during the canary analysis:

```yaml
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
metadata:
  name: load-tester
  namespace: prod
  annotations:
    fluxcd.io/ignore: "false"
spec:
  releaseName: load-tester
  chart:
    git: https://github.com/weaveworks/flagger
    ref: 0.18.4
    path: charts/loadtester
  values:
    fullnameOverride: load-tester
```

When you deploy a new podinfo version, Flagger gradually shifts traffic to the canary, and at the same time, measures the requests success rate as well as the average response duration. Based on an analysis of these Linkerd provided metrics, a canary deployment is either promoted or rolled back.

Trigger a canary deployment by updating the container image:

```yaml
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
spec:
  releaseName: podinfo
  values:
    image:
      tag: 3.0.1
```

```sh
git add -A && \
git commit -m "update podinfo" && \
git push origin master && \
fluxctl sync
```

When Flagger detects that the deployment revision changed it will start a new rollout. You can monitor the traffic shifting with:

```sh
watch kubectl -n prod get canaries
```

Automated rollback

During the canary analysis you can generate HTTP 500 errors and high latency to test if Flagger pauses and rolls back the faulted version.

Trigger another canary release:

```yaml
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
spec:
  releaseName: podinfo
  values:
    image:
      tag: 3.0.2
```

Apply changes:

```sh
git add -A && \
git commit -m "update podinfo" && \
git push origin master && \
fluxctl sync
```

Exec into the tester pod and generate HTTP 500 errors:

```
kubectl -n prod exec -it $(kubectl -n prod get pods -o name | grep -m1 load-test

$ hey -z 1m -c 5 -q 5 http://podinfo-canary:9898/status/500
```

When the number of failed checks reaches the canary analysis threshold, the traffic is routed back to the primary and the canary is scaled to zero.

Watch Flagger logs with:

```
$ kubectl -n linkerd logs deployment/flagger -f | jq .msg

 Starting canary analysis for podinfo.prod
 Advance podinfo.test canary weight 5
 Advance podinfo.test canary weight 10
 Advance podinfo.test canary weight 15
 Halt podinfo.test advancement success rate 69.17% < 99%
 Halt podinfo.test advancement success rate 61.39% < 99%
 Halt podinfo.test advancement success rate 55.06% < 99%
 Halt podinfo.test advancement request duration 1.20s > 0.5s
 Halt podinfo.test advancement request duration 1.45s > 0.5s
 Rolling back podinfo.prod failed checks threshold reached 5
 Canary failed! Scaling down podinfo.test
```

Monitoring with Linkerd

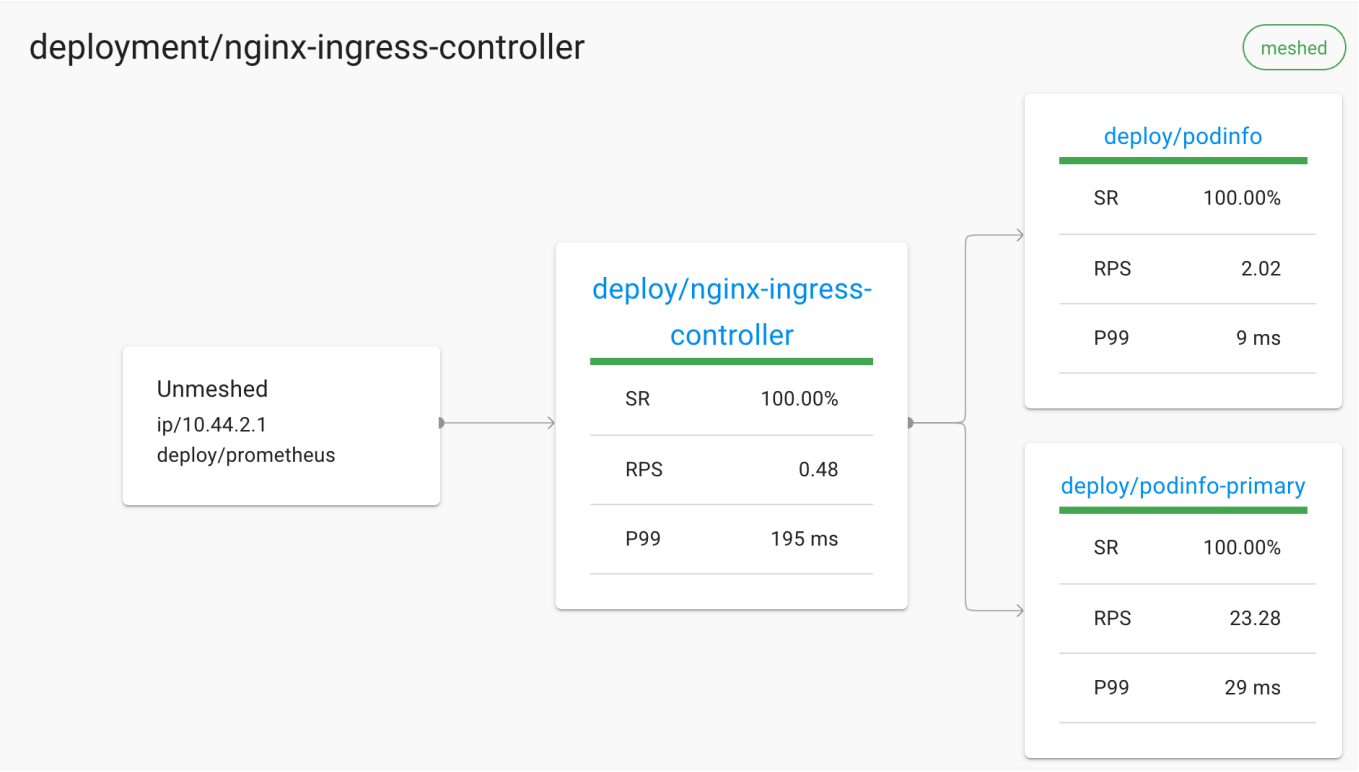
The Linkerd dashboard provides a high level view of what is happening with your services in real time. It can be used to visualize service dependencies, traffic splitting and understand the health of specific service routes.

Open the dashboard by running:

```sh
linkerd dashboard --port=50750
```

During the canary analysis, navigate to:

GitOps Helm Workshop



You can monitor the live traffic for the production namespace from the command line with:

```sh
linkerd -n prod top deploy
```

And you can view all the routes exposed by podinfo with:

```sh
linkerd -n prod routes service/podinfo
```

The above routes have been generated from the podinfo swagger spec and exported as Linkerd service profile.

← Helm Releases Canary Helm Tests →

Helm Tests

Flagger comes with a testing service that can run Helm tests when configured as a webhook.

Create tests

Create a test for the podinfo token API:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: {{ template "podinfo.fullname" . }}-jwt-test-{{ randAlphaNum 5 | lower }
  labels:
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version }}
    app: {{ template "podinfo.name" . }}
  annotations:
    linkerd.io/inject: disabled
    "helm.sh/hook": test-success
spec:
  containers:
    - name: tools
      image: giantswarm/tiny-tools
      command:
        - sh
        - -c
        - |
          TOKEN=$(curl -sd 'test' ${PODINFO_SVC}/token | jq -r .token) &&
          curl -H "Authorization: Bearer ${TOKEN}" ${PODINFO_SVC}/token/validate
      env:
      - name: PODINFO_SVC
        value: {{ template "podinfo.fullname" . }}:{{ .Values.service.externalPo
  restartPolicy: Never
```

`cluster/charts/podinfo/tests`

Deploy the Helm test runner in the `prod` namespace:

yaml
```yaml
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
metadata:
  name: helm-tester
  namespace: prod
  annotations:
    fluxcd.io/ignore: "false"
spec:
  releaseName: helm-tester
  chart:
    git: https://github.com/weaveworks/flagger
    ref: 0.18.4
    path: charts/loadtester
  values:
    fullnameOverride: helm-tester
    serviceAccountName: helm-tester
```

Apply changes:

sh
```sh
git add -A && \
git commit -m "install helm-tester" && \
git push origin master && \
fluxctl sync
```

Run tests

Add the helm test as a pre-rollout webhook:

yaml
```yaml
apiVersion: flagger.app/v1alpha3
kind: Canary
metadata:
  name: podinfo
  namespace: prod
spec:
  canaryAnalysis:
```

```
    webhooks:
      - name: "helm test"
        type: pre-rollout
        url: http://helm-tester.prod/
        timeout: 2m
        metadata:
          type: "helmv3"
          cmd: "test run podinfo --cleanup"
      - name: load-test
        url: http://load-tester.prod/
        metadata:
          cmd: "hey -z 2m -q 10 -c 2 http://podinfo:9898/"
```

Apply changes:

```sh
git add -A && \
git commit -m "update podinfo" && \
git push origin master && \
fluxctl sync
```

When the canary analysis starts, Flagger will call the pre-rollout webhooks before routing traffic to the canary. If the helm test fails, Flagger will retry until the analysis threshold is reached and the canary is rolled back.

← Canary Releases