Kubernetes Canary Deployment #2 Argo Rollouts

We're using k8s-native Argo Rollouts and GitlabCI to perform Canary Kubernetes deployments

Kim Wuestkamp
Jan 13, 2020 · 6 min read ★

https://unsplash.com/photos/V41PulGL1z0

Parts

1. Canary Deployment using GitlabCI + GitOps/Manual Approach
2. (this article)
3. Canary Deployment using Istio
4. Canary Deployment using Jenkins-X Istio Flagger

Canary Deployment

Make sure to read part 1 where we explained shortly what Canary Deployments are. There we also showed how to implement those using vanilla Kubernetes resources.

Argo Rollouts

Argo Rollouts is Kubernetes native. It provides a CRD (Custom Resource Definition) for Kubernetes. This way we can use a new resource `kind: Rollout` which handles Green/Blue and Canary deployments with various options to configure.

> *Argo Rollouts controller, uses the Rollout custom resource to provide additional deployment strategies such as Blue Green and Canary to Kubernetes. The Rollout custom resource provides feature parity with the deployment resource but with additional deployment strategies.*
>
> *Deployments resources offer two strategies to deploy changes:* `RollingUpdate` *and* `Recreate`*. While these strategies can solve a wide number of use cases, large scale production deployments use additional strategies, such as blue-green or canary, that are missing from the Deployment controller. In order to use these strategies in Kubernetes, users are forced to build scripts on top of their deployments. The Argo Rollouts controller provides these strategies as simple declarative, configurable options.*
>
> *https://argoproj.github.io/argo-rollouts*

There is also Argo CI which provides a nice webinterface to use in conjunction with the Rollouts, we might have a look a this in a later article.

Install Argo Rollouts

Server Side

```
kubectl create namespace argo-rollouts
```

```
kubectl apply -n argo-rollouts -f
https://raw.githubusercontent.com/argoproj/argo-
rollouts/stable/manifests/install.yaml
```

In our infrastructure repo (see further below) we included the `install.yaml` already at `i/k8s/argo-rollouts/install.yaml`. This way the GitlabCI job installs it in the cluster.

Client Side (kubectl plugin)

https://argoproj.github.io/argo-rollouts/features/kubectl-plugin

Example App

It's good practice to have one repository for the application code and one for the infrastructure.

Application Repo

Kim Wuestkamp / k8s-deployment-example-app

GitLab.com

gitlab.com

It's a very simple Python+Flask api which returns a JSON response. We build the package using GitlabCI and push the results into Gitlab Registry. In the registry there are two different versions released:

- `wuestkamp/k8s-deployment-example-app:v1`
- `wuestkamp/k8s-deployment-example-app:v2`

The only difference between both versions is that the returned JSON is changed. We use this app for a very simple visualisation of which version we're talking to.

Infrastructure Repo

Kim Wuestkamp / k8s-deployment-example-argo-canary-infrastructure

GitLab.com

gitlab.com

gitlab.com

In this repo we use GitlabCI for the Kubernetes deployment, the `.gitlab-ci.yml` looks like:

```
image: traherom/kustomize-docker

before_script:
    - printenv
    - kubectl version

stages:
  - deploy

deploy test:
    stage: deploy
    before_script:
      - echo $KUBECONFIG
    script:
      - kubectl get all
      - kubectl apply -f i/k8s

    only:
      - master
```

To run this yourself first you need a cluster, you can use Gcloud:

```
gcloud container clusters create canary --num-nodes 3 --zone europe-
west3-b

gcloud compute firewall-rules create incoming-80 --allow tcp:80
```

You need to fork https://gitlab.com/wuestkamp/k8s-deployment-example-canary-infrastructure and create a GitlabCI variable named `KUBECONFIG` (type file) which should contain the `kubectl` access config to your cluster.

Read this on how to get your (Gcloud) cluster credentials.

Infrastructure Yaml

We have a `service.yaml`:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    id: rollout-canary
  name: app
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 5000
  selector:
    id: app
  type: LoadBalancer
```

and a `rollout.yaml` :

```
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: rollout-canary
spec:
  replicas: 10
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      id: rollout-canary
  template:
    metadata:
      labels:
        id: rollout-canary
    spec:
      containers:
      - name: rollouts-demo
        image: registry.gitlab.com/wuestkamp/k8s-deployment-example-
app:v2
        imagePullPolicy: Always
  strategy:
    canary:
      steps:
      - setWeight: 10
      # Rollouts can be manually resumed by running `kubectl argo
rollouts promote ROLLOUT`
      - pause: {}
      - setWeight: 50
      - pause: { duration: 120 } # two minutes
```

The Rollout acts mostly like a Deployment. If we wouldn't configure a strategy (like canary here) it would act as a default rolling-update Deployment.

We defined 2 steps in the yaml for the canary deployment:

1. 10% traffic to canary (wait for manual OK)
2. 50% traffic to canary (wait 2 minutes then continue to 100%)

Initial Deployment

After an initial deployment our resources look like this:

```
NAME                                    READY   STATUS    RESTARTS   AGE
pod/rollout-canary-6fd7bd74c5-2wjj6     1/1     Running   0          3m16s
pod/rollout-canary-6fd7bd74c5-4w9cf     1/1     Running   0          3m16s
pod/rollout-canary-6fd7bd74c5-982qc     1/1     Running   0          3m16s
pod/rollout-canary-6fd7bd74c5-9j4r9     1/1     Running   0          3m16s
pod/rollout-canary-6fd7bd74c5-bwr5x     1/1     Running   0          3m16s
pod/rollout-canary-6fd7bd74c5-dl522     1/1     Running   0          3m16s
pod/rollout-canary-6fd7bd74c5-f7m8l     1/1     Running   0          3m16s
pod/rollout-canary-6fd7bd74c5-gf7xh     1/1     Running   0          3m16s
pod/rollout-canary-6fd7bd74c5-k9pbr     1/1     Running   0          3m16s
pod/rollout-canary-6fd7bd74c5-nmfb9     1/1     Running   0          3m16s

NAME                                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
rollout.argoproj.io/rollout-canary    10        10        10           10
```

And we only get responses from app version 1:

```
➜  ~ while true; do curl -s 35.198.149.232 | grep label; sleep 0.1; done
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
```

Perform a Canary Deployment

Step 1: 10% traffic

To initiate a canary deployment we now simply have to change the image version as we usually do with a deployment:

```
apiVersion: argoproj.io/v1alpha1
kind: Rollout
```

```
metadata:
  name: rollout-canary
spec:
...
  template:
    metadata:
      labels:
        id: rollout-canary
    spec:
      containers:
      - name: rollouts-demo
        image: registry.gitlab.com/wuestkamp/k8s-deployment-example-app:v2
...
```

And we push the change so GitlabCI can deploy which results in:

```
NAME                                    READY   STATUS    RESTARTS   AGE
pod/rollout-canary-6fd7bd74c5-2wjj6     1/1     Running   0          5m44s
pod/rollout-canary-6fd7bd74c5-982qc     1/1     Running   0          5m44s
pod/rollout-canary-6fd7bd74c5-9j4r9     1/1     Running   0          5m44s
pod/rollout-canary-6fd7bd74c5-bwr5x     1/1     Running   0          5m44s
pod/rollout-canary-6fd7bd74c5-dl522     1/1     Running   0          5m44s
pod/rollout-canary-6fd7bd74c5-f7m8l     1/1     Running   0          5m44s
pod/rollout-canary-6fd7bd74c5-gf7xh     1/1     Running   0          5m44s
pod/rollout-canary-6fd7bd74c5-k9pbr     1/1     Running   0          5m44s
pod/rollout-canary-6fd7bd74c5-nmfb9     1/1     Running   0          5m44s
pod/rollout-canary-7c88459764-kwv96     1/1     Running   0          38s

NAME                                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
rollout.argoproj.io/rollout-canary    10        10        1            10
```

Now if we call the service:

```
➜  ~ while true; do curl -s 35.198.149.232 | grep label; sleep 0.1; done
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "VERSION2"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "v1"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "v1"
  "label": "v1"
  "label": "VERSION2"
  "label": "v1"
  "label": "v1"
```

```
  "label": "v1"
  "label": "v1"
```

Nice! We're in the middle of our canary deployment. We can see this by running:

```
kubectl argo rollouts get rollout rollout-canary
```

```
→ ~ kubectl argo rollouts get rollout rollout-canary
Name:            rollout-canary
Namespace:       default
Status:          ॥ Paused
Strategy:        Canary
  Step:          1/4
  SetWeight:     10
  ActualWeight:  10
Images:          registry.gitlab.com/wuestkamp/k8s-deployment-example-app:v1 (stable)
                 registry.gitlab.com/wuestkamp/k8s-deployment-example-app:v2 (canary)
Replicas:
  Desired:       10
  Current:       10
  Updated:       1
  Ready:         10
  Available:     10

NAME                                       KIND        STATUS     AGE    INFO
⟳ rollout-canary                           Rollout     ॥ Paused   5m36s
├──# revision:2
│  └──⧉ rollout-canary-7c88459764          ReplicaSet  ✔ Healthy  30s    canary
│     └──□ rollout-canary-7c88459764-kwv96 Pod         ✔ Running  30s    ready:1/1
└──# revision:1
   └──⧉ rollout-canary-6fd7bd74c5          ReplicaSet  ✔ Healthy  5m36s  stable
      ├──□ rollout-canary-6fd7bd74c5-2wjj6 Pod         ✔ Running  5m36s  ready:1/1
      ├──□ rollout-canary-6fd7bd74c5-982qc Pod         ✔ Running  5m36s  ready:1/1
      ├──□ rollout-canary-6fd7bd74c5-9j4r9 Pod         ✔ Running  5m36s  ready:1/1
      ├──□ rollout-canary-6fd7bd74c5-bwr5x Pod         ✔ Running  5m36s  ready:1/1
      ├──□ rollout-canary-6fd7bd74c5-dl522 Pod         ✔ Running  5m36s  ready:1/1
      ├──□ rollout-canary-6fd7bd74c5-f7m8l Pod         ✔ Running  5m36s  ready:1/1
      ├──□ rollout-canary-6fd7bd74c5-gf7xh Pod         ✔ Running  5m36s  ready:1/1
      ├──□ rollout-canary-6fd7bd74c5-k9pbr Pod         ✔ Running  5m36s  ready:1/1
      └──□ rollout-canary-6fd7bd74c5-nmfb9 Pod         ✔ Running  5m36s  ready:1/1
```

Step 2: 50% traffic:

Now to the next step of redirecting 50% of the traffic, we configured this to require a manual step:

```
kubectl argo rollouts promote rollout-canary # continue to step 2
```

```
NAME                                   READY   STATUS    RESTARTS   AGE
pod/rollout-canary-6fd7bd74c5-2wjj6    1/1     Running   0          8m36s
pod/rollout-canary-6fd7bd74c5-982qc    1/1     Running   0          8m36s
pod/rollout-canary-6fd7bd74c5-dl522    1/1     Running   0          8m36s
```

```
pod/rollout-canary-6fd7bd74c5-gf7xh   1/1     Running   0          8m36s
pod/rollout-canary-6fd7bd74c5-nmfb9   1/1     Running   0          8m36s
pod/rollout-canary-7c88459764-8ts8j   1/1     Running   0          25s
pod/rollout-canary-7c88459764-k9dmb   1/1     Running   0          25s
pod/rollout-canary-7c88459764-kwv96   1/1     Running   0          3m30s
pod/rollout-canary-7c88459764-s7t2n   1/1     Running   0          22s
pod/rollout-canary-7c88459764-tp4tq   1/1     Running   0          25s

NAME                                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
rollout.argoproj.io/rollout-canary    10        10        5            10
```

And our app returns 50% new versions:

```
➜  ~ while true; do curl -s 35.198.149.232 | grep label; sleep 0.1; done
  "label": "VERSION2"
  "label": "v1"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "v1"
  "label": "v1"
  "label": "VERSION2"
  "label": "v1"
  "label": "v1"
  "label": "VERSION2"
  "label": "v1"
  "label": "v1"
  "label": "VERSION2"
  "label": "v1"
  "label": "VERSION2"
  "label": "v1"
```

and the rollout overview:

```
➜  ~ kubectl argo rollouts get rollout rollout-canary
Name:            rollout-canary
Namespace:       default
Status:          ॥ Paused
Strategy:        Canary
  Step:          3/4
  SetWeight:     50
  ActualWeight:  50
Images:          registry.gitlab.com/wuestkamp/k8s-deployment-example-app:v1 (stable)
                 registry.gitlab.com/wuestkamp/k8s-deployment-example-app:v2 (canary)
Replicas:
  Desired:       10
  Current:       10
  Updated:       5
  Ready:         10
  Available:     10

NAME                                KIND        STATUS        AGE    INFO
⟳ rollout-canary                    Rollout     ॥ Paused      8m22s
├──# revision:2
│  └──⧉ rollout-canary-7c88459764   ReplicaSet  ✔ Healthy     3m16s  canary
```

```
        ├──□ rollout-canary-7c88459764-kwv96  Pod         ✔ Running      3m16s  ready:1/1
        ├──□ rollout-canary-7c88459764-8ts8j  Pod         ✔ Running      11s    ready:1/1
        ├──□ rollout-canary-7c88459764-k9dmb  Pod         ✔ Running      11s    ready:1/1
        ├──□ rollout-canary-7c88459764-tp4tq  Pod         ✔ Running      11s    ready:1/1
        └──□ rollout-canary-7c88459764-s7t2n  Pod         ✔ Running      8s     ready:1/1
└──# revision:1
    └──⧉ rollout-canary-6fd7bd74c5            ReplicaSet  ✔ Healthy      8m22s  stable
        ├──□ rollout-canary-6fd7bd74c5-2wjj6  Pod         ✔ Running      8m22s  ready:1/1
        ├──□ rollout-canary-6fd7bd74c5-982qc  Pod         ✔ Running      8m22s  ready:1/1
        ├──□ rollout-canary-6fd7bd74c5-dl522  Pod         ✔ Running      8m22s  ready:1/1
        ├──□ rollout-canary-6fd7bd74c5-gf7xh  Pod         ✔ Running      8m22s  ready:1/1
        ├──□ rollout-canary-6fd7bd74c5-k9pbr  Pod         ○ Terminating  8m22s  ready:0/1
        └──□ rollout-canary-6fd7bd74c5-nmfb9  Pod         ✔ Running      8m22s  ready:1/1
```

Beautiful.

Step 3: 100% traffic:

We configured that after 2 minutes the 50% step finishes automatically and we should be at 100%:

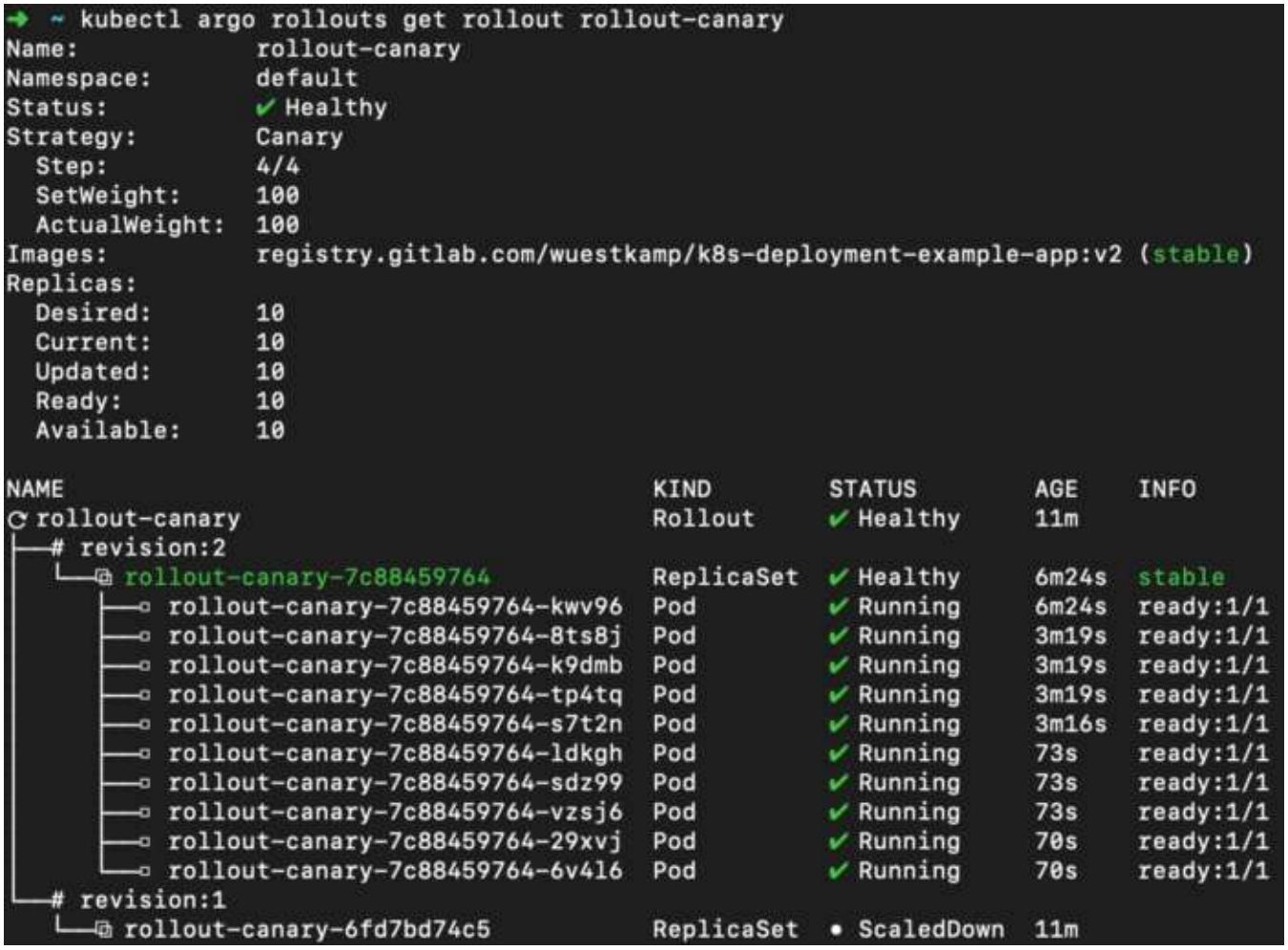
```
NAME                                    READY   STATUS    RESTARTS   AGE
pod/rollout-canary-7c88459764-29xvj     1/1     Running   0          23s
pod/rollout-canary-7c88459764-6v416     1/1     Running   0          23s
pod/rollout-canary-7c88459764-8ts8j     1/1     Running   0          2m32s
pod/rollout-canary-7c88459764-k9dmb     1/1     Running   0          2m32s
pod/rollout-canary-7c88459764-kwv96     1/1     Running   0          5m37s
pod/rollout-canary-7c88459764-ldkgh     1/1     Running   0          26s
pod/rollout-canary-7c88459764-s7t2n     1/1     Running   0          2m29s
pod/rollout-canary-7c88459764-sdz99     1/1     Running   0          26s
pod/rollout-canary-7c88459764-tp4tq     1/1     Running   0          2m32s
pod/rollout-canary-7c88459764-vzsj6     1/1     Running   0          26s

NAME                                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
rollout.argoproj.io/rollout-canary    10        10        10           10
```

and we hit the app:

```
➜  ~ while true; do curl -s 35.198.149.232 | grep label; sleep 0.1; done
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
  "label": "VERSION2"
```

and the rollout overview:



Canary deployment done.

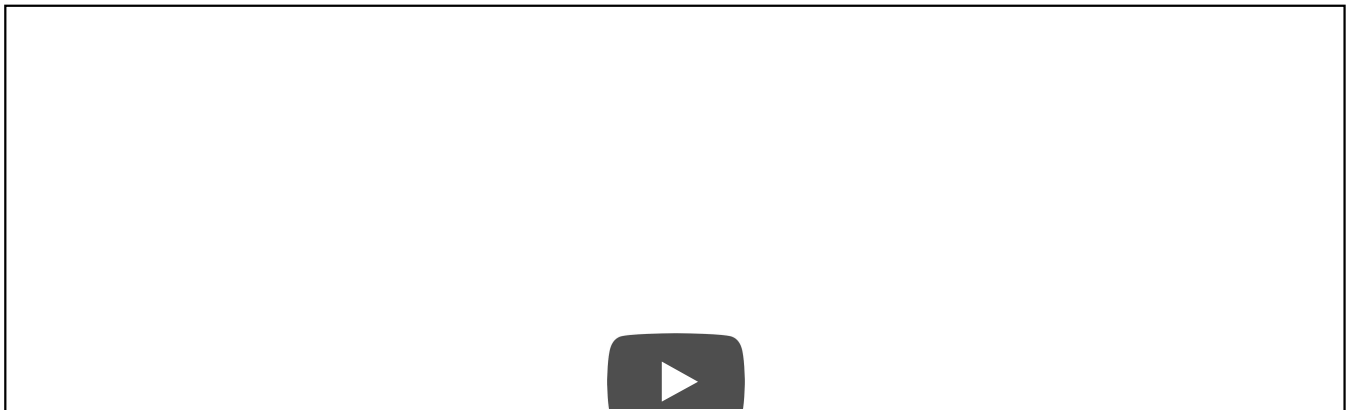
More Argo Rollouts Examples

There are more examples which for example show how to setup preview environments and baseline-canary comparisons:

https://github.com/argoproj/argo-rollouts/tree/master/examples

Video of Argo Rollouts and Argo CI

I can really recommend this video which shows Argo Rollouts and Argo CI working together:



Recap

I really like the idea of using CRDs which handle the creation of extra deployments or replicasets and redirecting traffic etc. This works smooth. Next I would like to test the integration with Argo CI.

Though it seems there is also a big merge of Argo CI and Flux CI coming, so I might wait for this tobe released at first: Argo Flux.

Have you made experience with Argo Rollouts or Argo CI?

Kubernetes DevOps Continuous Integration Continuous Delivery Continuous Deployment