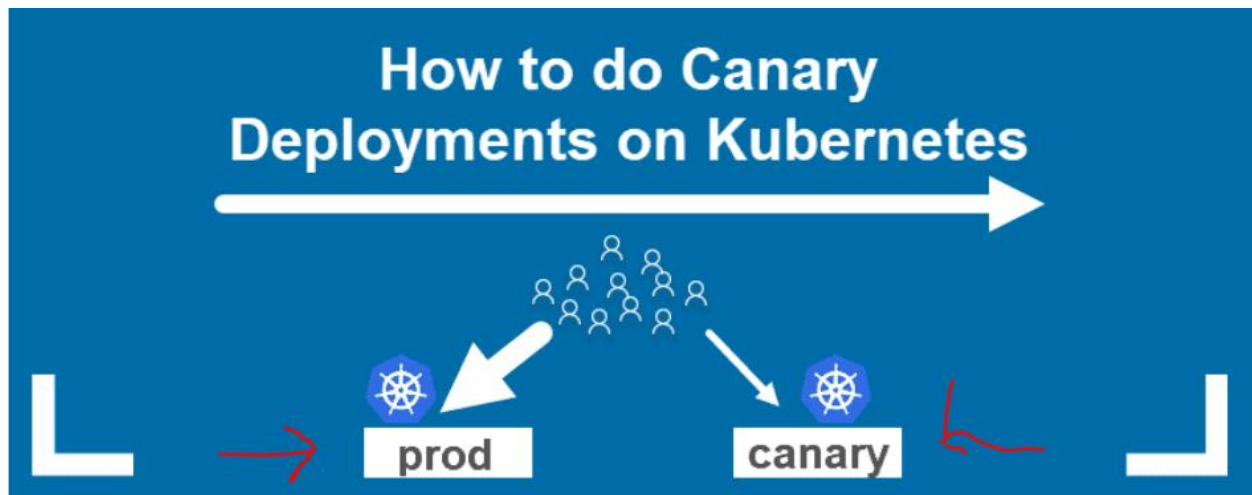


How to do Canary Deployments on Kubernetes

- ✚ Testing out a new feature or upgrade in production is a stressful process.
- ✚ You want to roll out changes frequently but without affecting the user experience.
- ✚ To minimize downtime during this phase, set up canary deployments to streamline the transition.
- ✚ You can use canary deployments on any infrastructure.
- ✚ Accordingly, it is one of the deployment strategies in Kubernetes.



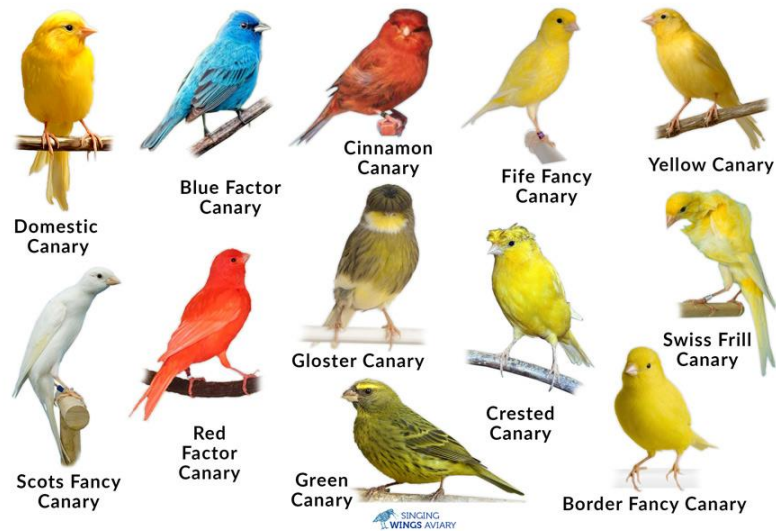
What is a Canary Deployment?

- ✚ A canary deployment is an upgraded version of an existing deployment, with all the required application code and dependencies.
- ✚ It is used to test out new features and upgrades to see how they handle the production environment.
- ✚ When you add the canary deployment to a Kubernetes cluster, it is managed by a service through selectors and labels.
- ✚ The service routes traffic to the pods that have the specified label. This allows you to add or remove deployments easily.
- ✚ The amount of traffic that the canary gets corresponds to the number of pods it spins up.
- ✚ In most cases, you start by routing a smaller percentage of traffic to the canary and increase the number over time.

- ✚ With both deployments set up, you monitor the canary behavior to see whether any issues arise.
- ✚ Once you are happy with the way it is handling requests, you can upgrade all the deployments to the latest version.

Note: Canary deployments got their name from an old British mining practice. Back in the day, miners used canaries to test coal mines' safety before they went in. If the canaries returned unharmed, the miners felt safe to enter. However, if something did happen to the birds, they knew that the mines were filled with toxic gases.

Types of Canaries



Setting up Canary Deployment on Kubernetes

- ✚ We created a simple Kubernetes cluster of Nginx pods with a basic, two-sentence static HTML page.
- ✚ The versions of the deployment vary by the content they display on the web page.
- ✚ The process of setting up your canary deployment will differ according to the application you are running.

Step 1: Pull Docker Image

The first step is to pull or create the image for the containers in your Kubernetes cluster.

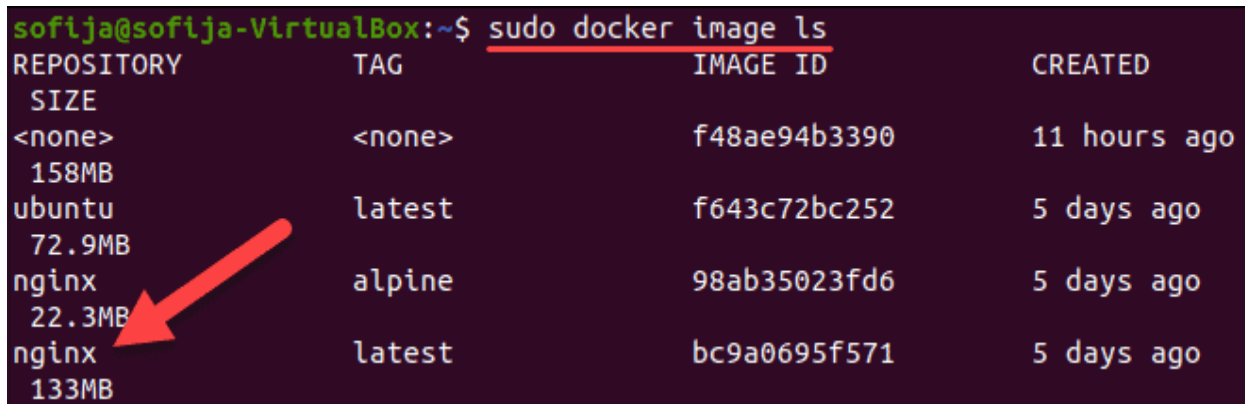
Since we are building Nginx containers in this example, we use the Nginx image available on [Docker Hub](#).

1. Download the image with:

```
# docker pull nginx
```

2. Verify you have it by listing all local images:

```
# docker image ls
```



```
soflja@soflja-VirtualBox:~$ sudo docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED
SIZE			
<none>	<none>	f48ae94b3390	11 hours ago
158MB			
ubuntu	latest	f643c72bc252	5 days ago
72.9MB			
nginx	alpine	98ab35023fd6	5 days ago
22.3MB			
nginx	latest	bc9a0695f571	5 days ago
133MB			

Step 2: Create the Kubernetes Deployment

1. Create the deployment definition using a yaml file.

Use a text editor of your choice and provide a name for the file. We are going to name the file nginx-deployment.yaml and create it with Nano:

```
# nano nginx-deployment.yaml
```

Add the following content to the file:

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
name: nginx
```

```
spec:
```

```
selector:
```

matchLabels:

app: nginx

replicas: 3

template:

metadata:

labels:

app: nginx

version: "1.0"

spec:

containers:

- name: nginx

image: nginx:alpine

resources:

limits:

memory: "128Mi"

cpu: "50m"

ports:

- containerPort: 80

volumeMounts:

- mountPath: /usr/share/nginx/html

name: index.html

volumes:

- name: index.html

hostPath:

path: /Users/sofija/Documents/nginx/v1

- ✚ We created 3 replicas of Nginx pods for the Kubernetes cluster.
- ✚ All the pods have the label version: "1.0".
- ✚ Additionally, they have a host volume containing the index.html mounted to the container.
- ✚ The sample HTML file consisting of:

```
<html>
<h1>Hello World!</h1>
<p>This is version 1</p>
</html>
```

3. Save and exit the file.

4. Create the deployment by running:

```
# kubectl apply -f nginx-deployment.yaml
```

5. Check whether you have successfully deployed the pods with:

```
# kubectl get pods -o wide
```

The output should display three running Nginx pods.

Step 3: Create the Service

The next step is to **create a service definition** for the Kubernetes cluster. The service will route requests to the specified pods.

1. Create a new **yaml** file with:

```
# nano nginx-deployment.service.yaml
```

2. Then, add the following content

```
apiVersion: v1
```

```
kind: Service
```

```
metadata:
```

```
name: nginx-service
```

```
spec:
```

type: LoadBalancer

selector:

app: nginx

version: "1.0"

ports:

- port: 8888

targetPort: 80

The yaml file specifies the type of service – LoadBalancer. It instructs the service to balance workloads between pods **with the labels app: nginx and version: "1.0"**. The pod needs to have both labels to be part of the service.

3. Save and exit the service file.

4. Now, create the service:

```
# kubectl apply -f nginx-deployment.service.yaml
```

Step 4: Check First Version of Cluster

To verify the service is running, open a web browser, and navigate to the IP and port number defined in the service file.

To see the external IP address of the service, use the command:

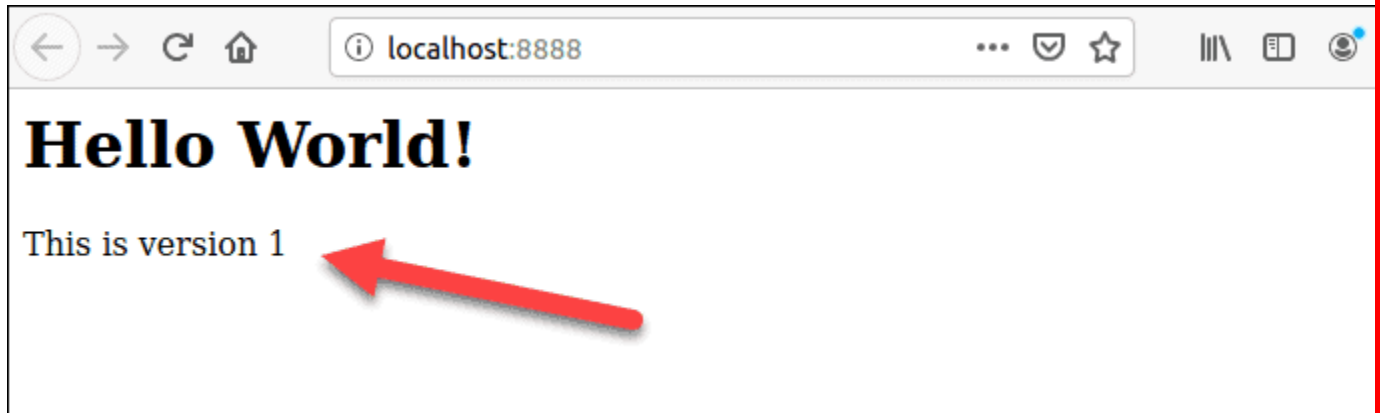
```
# kubectl get service
```

If you are running Kubernetes locally, use localhost as the IP.

Since the sample cluster we created is running locally on port 8888, the URL is:

<http://<IP address>:8888>

The browser should display a Hello World message from version 1.



Step 5: **Create a Canary Deployment**

With version 1 of the application in place, you deploy version 2, the canary deployment.

1. Start by creating the yaml file for the canary deployment. Run the command:

```
# nano nginx-canary-deployment.yaml
```

2. Add the following content to the file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: nginx-canary-deployment
spec:
selector:
matchLabels:
app: nginx
replicas: 3
template:
metadata:
```

labels:

app: nginx

version: "2.0"

spec:

containers:

- name: nginx

image: nginx:alpine

resources:

limits:

memory: "128Mi"

cpu: "50m"

ports:

- containerPort: 80

volumeMounts:

- mountPath: /usr/share/nginx/html

name: index.html

volumes:

- name: index.html

hostPath:

path: /Users/sofija/Documents/nginx/v2

The content of the canary deployment file differs by three important parameters:

- The name in the metadata is nginx-canary-deployment.
- It has the label version: "2.0".
- It is linked to a html file index.html which consists of:

<html>


```
<h1>Hello World!</h1>
<p>This is version 2</p>
</html>
```

Save and exit the file.

3. Create the canary deployment with the command:

```
# kubectl apply -f nginx-canary-deployment.yaml
```

4. Verify you have successfully deployed the three additional pods:

```
# Kubectl get pods -o wide
```

The output should display the Nginx canary deployment pods, along with the original Nginx pods.

Step 6: Run the Canary Deployment

Open a web browser and navigate to the same IP address as in Step 4. You will notice there are no changes to the web page. This is because the service file is configured to load balance only pods with the label version: "1.0".

To test out the updated pods, you need to modify the service file and direct part of the traffic to version: "2.0".

1. To do so, open the yaml file with:

```
# nano nginx-deployment.service.yaml
```

2. Find and remove the line version: "1.0". The file should include the following:

```
apiVersion: v1
kind: Service
metadata:
```

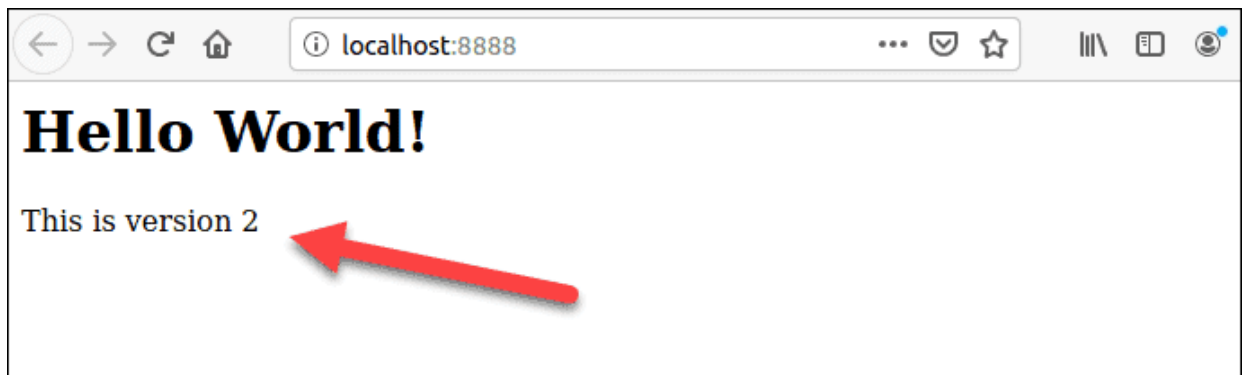
```
name: nginx-service
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
    - port: 8888
      targetPort: 80
```

Save the changes and exit the file.

3. Create the updated service with the command:

```
# kubectl apply -f nginx-deployment.service.yml
```

4. The traffic is now split between version 1 and version 2 pods. If you refresh the web page a few times, you see different results depending on where the service redirects your request.



Note: In the example above, half of the traffic is redirected to the canary deployment. This is because there are three replicas of version 1 and three replicas of version 2. However, you can start off by redirecting a smaller percentage of the requests. Simply configure the canary deployment to have fewer pods. You can gradually increase the number of replicas once you are confident the canary can handle more traffic.

Step 7: Monitor the Canary Behavior

With both deployments up and running, monitor the behavior of the new deployment. Depending on the results, you can roll back the deployment or upgrade to the newer version.

Roll Back Canary Deployment

If you notice the canary is not performing as expected, you can roll back the deployment and delete the upgraded pods with:

```
# kubectl delete deployment.apps/nginx-canary-deployment
```

The service continues load balancing the traffic to the initial (version 1) pods.

Roll Out Upgraded Deployment

If you conclude the canary deployment is performing as expected, you can route all incoming traffic to the upgraded version. There are three ways to do so:

1. Upgrade the first version by modifying the Docker image and building a new deployment. Then, remove the canaries with:

```
# kubectl delete deployment.apps/nginx-canary-deployment
```

2. You can keep the upgraded pods and remove the ones with the version 1 label:

```
# kubectl delete deployment.apps/nginx
```

3. Alternatively, you can even modify the service.yaml file and add the version specifier to the selector label. This instructs the load balancer to only route traffic to version 2 pods.

Important Content

Visit the docker PowerPoint on creating volumes

Important link

<https://www.docker.com/blog/how-to-use-the-official-nginx-docker-image/>