## Overview

There are a multitude of approaches to rolling out new versions of an app. We will explore a few in this lab. The app we will be deploying is a simple hostname app that outputs the hostname and version of the app.

* Create deployments
* Update the application
* Roll back and update
* Perform a Canary deployment

[Overview](#_brow32mvt65p)

[Instructions](#_h7xnyttuehza)

[Task 0: Setup](#_q1eipr6h6290)

[Step 1: Connect to lab](#_avy5sf30savv)

[Step 2: Make new directory](#_30rfxpp431je)

[Task 1: Recreate Updates](#_zb96oq1l8t7a)

[Step 1: Create a Deployment](#_z9ad9pzgs0f1)

[Step 2: Create a service](#_6mpenovnyswm)

[Step 3: Test the app](#_4yavdch7id2i)

[Step 4: Update from the v1 to the v2 image](#_pedbv644iqbw)

[Task 2: Change to rolling update strategy](#_i2e82b4aksib)

[Step 1: Make a rolling deployment](#_efzf5n84ceea)

[Step 2: Roll out v2](#_aco882v3ou5c)

[Task 3: Roll back](#_ys1bqap0uua)

[Task 4: Cleanup](#_u9xvcp7wwuqd)

[Task 5: Canary Deployment](#_i81c4cnc1o7q)

[Step 1: Create the deployment](#_fw3xko6atbzd)

[Step 2: Test v2](#_9rtf5zxpo3sr)

[Step 3: Deploy v2](#_d1p2cq6trv9o)

[Task 6: Cleanup](#_o1d9zpkk4agg)

## 

## Instructions

Read this lab like a book, all text is there for a reason!

"→" denotes an action you must take

<> denotes a placeholder you should replace with a value

Ex: <IP> becomes 127.0.0.1

Use your favorite editor to edit files within the console. I suggest VI, nano, or emacs.

|  |
| --- |
| White boxes with black text denote commands and file contents |

|  |
| --- |
| Black boxes with green text denote example output |

### **Task 0: Setup**

#### **Step 1: Connect to lab**

→ Open up 2 SSH sessions into your lab vm, we will refer to these as ***window 1*** and ***window 2***.

#### **Step 2: Make new directory**

→ In ***window 1***, create a new directory to work in

|  |
| --- |
| mkdir ~/deploy  cd ~/deploy |

### **Task 1: Recreate Updates**

We will start by creating a Deployment with the most basic deployment strategy, recreate. This means that any time we have a new version of our Pod spec in the Deployment (like a new image) then it will first delete all existing Pods before creating pods with the new spec.

#### **Step 1: Create a Deployment**

Note that in our Deployment YAML below, we are specifying the strategy field. You can see the details of what values are allowed with:

→ Examine strategy options

|  |
| --- |
| kubectl explain deployments.spec.strategy |

|  |
| --- |
| KIND: Deployment  VERSION: extensions/v1beta1  RESOURCE: strategy <Object>  DESCRIPTION:  The deployment strategy to use to replace existing pods with new ones.  DeploymentStrategy describes how to replace existing pods with new ones.  FIELDS:  rollingUpdate <Object>  Rolling update config params. Present only if DeploymentStrategyType =  RollingUpdate.  type <string>  Type of deployment. Can be "Recreate" or "RollingUpdate". Default is  RollingUpdate. |

→ Create a file named deployment.yaml

|  |
| --- |
| vi deployment.yaml |

→ Type ":set paste", press **enter**. Then press **i**, paste the following:

|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:  name: app-deployment  labels:  app: http-app  spec:  replicas: 3  strategy:  type: Recreate  selector:  matchLabels:  app: http-app  template:  metadata:  labels:  app: http-app  spec:  containers:  - name: http-app  image: acmeade/http-app:v1  ports:  - containerPort: 80 |

→ Hit **ESC**, then type ':wq' and press **enter** to save the file.

→ Create Deployment

|  |
| --- |
| kubectl apply -f deployment.yaml |

|  |
| --- |
| deployment.apps/app-deployment created |

#### **Step 2: Create a service**

Now we need a service to load balance traffic to our Pods.

→ Create a file named service.yaml

|  |
| --- |
| vi service.yaml |

→ Type ":set paste", press **enter**. Then press **i**, paste the following:

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:  labels:  app: http-app  name: app-service  spec:  ports:  - port: 80  protocol: TCP  targetPort: 80  selector:  app: http-app  sessionAffinity: None  type: LoadBalancer |

→ Hit **ESC**, then type ':wq' and press **enter** to save the file.

→ Create Service

|  |
| --- |
| kubectl apply -f service.yaml |

|  |
| --- |
| service/app-service created |

#### **Step 3: Test the app**

→ Grab the app-service IP from your output

|  |
| --- |
| kubectl get services |

|  |
| --- |
| NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S)  AGE  app-service LoadBalancer 10.11.247.171  **35.225.84.45**  80:32343/TCP  15m |

→ Set the IP variable in ***window 2***

|  |
| --- |
| export IP=<EXTERNAL IP> |

The following script will make a request to the service once every 5 seconds and display the output.

→ In ***window 2***, execute the following:

|  |
| --- |
| while true; do date +%T; curl http://$IP; sleep 5; done |

|  |
| --- |
| 16:58:53  Version: 1  Host: app-deployment-v2-6b95c4f8b6-wz4km |

#### **Step 4: Update from the v1 to the v2 image**

→ In ***window 1***, Open deployment.yaml

|  |
| --- |
| vi deployment.yaml |

→ move with the arrow keys and hover over the **1** on the following line

|  |
| --- |
| image: acmeade/http-app:v1 |

→ type **r** then **2** in order to replace the character

→ type **:wq** to save the file

→ Apply changes

|  |
| --- |
| kubectl apply -f deployment.yaml |

|  |
| --- |
| deployment.apps/app-deployment configured |

**Step 5: Watch the update**

Let's look at the pods as they all get deleted and then new ones are created. We can use the '--watch' flag to have kubectl make the requests continuously for us.

→ List the pods

|  |
| --- |
| kubectl get pods --watch |

|  |
| --- |
| NAME READY STATUS RESTARTS AGE  app-deployment-6bd6dd6755-887gw 1/1 Terminating 0 117s  app-deployment-6bd6dd6755-jkws2 1/1 Terminating 0 117s  app-deployment-6bd6dd6755-r68gx 1/1 Terminating 0 117s |

Once we again have 3 pods running, notice how in ***window 2***, we can see that during the update, requests started failing. This is because between the time the deployment deletes all v1 pods and creates v2 pods, there was nothing to serve our requests!

### **Task 2: Change to rolling update strategy**

#### **Step 1: Make a rolling deployment**

View the options for rolling update. We are going to change our deployment to handle updates in a rolling fashion. We will also set the options so that new versions come up 2 at a time, and old Pods remain until the new Pods are healthy.

|  |
| --- |
| kubectl explain deployments.spec.strategy.rollingUpdate |

→ Delete our deployment file

|  |
| --- |
| rm deployment.yaml |

→ Create the new deployment.yaml

|  |
| --- |
| vi deployment.yaml |

→ Type '**:set paste'**, press **enter**. Then press **i**, paste the following:

|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:  name: app-deployment  labels:  app: http-app  spec:  replicas: 3  strategy:  **type: RollingUpdate**  **rollingUpdate:**  **maxSurge: 2**  **maxUnavailable: 0**  selector:  matchLabels:  app: http-app  template:  metadata:  labels:  app: http-app  spec:  containers:  - name: http-app  image: acmeade/http-app:v1  ports:  - containerPort: 80 |

→ Hit **ESC**, then type '**:wq**' and press **enter** to save the file.

→ Apply changes

|  |
| --- |
| kubectl apply -f deployment.yaml |

|  |
| --- |
| deployment.apps/app-deployment configured |

#### **Step 2: Roll out v2**

→ Open deployment.yaml

|  |
| --- |
| vi deployment.yaml |

→ Move with the arrow keys and hover over the 1 on the following line

|  |
| --- |
| image: acmeade/http-app:v1 |

→ Type **r** then **2** in order to replace the character

→ Type **:wq** to save the file

→ Apply changes

|  |
| --- |
| kubectl apply -f deployment.yaml |

|  |
| --- |
| deployment.apps/app-deployment configured |

→ List replica sets

|  |
| --- |
| kubectl get replicaset -o wide |

|  |
| --- |
| NAME DESIRED CURRENT READY AGE CONTAINERS IMAGES SELECTOR  app-deployment-6bd6dd6755 0 0 0 7m36s http-app acmeade/http-app:v1 app=http-app,pod-template-hash=6bd6dd6755  app-deployment-fdfff784d 3 3 3 5m5s http-app acmeade/http-app:v2 app=http-app,pod-template-hash=fdfff784d |

Notice that the way rolling deployments are performed is by creating 2 replica sets, one that manages the number of Pods of the old version and another for the new version. The new ReplicaSet will scale up while the old ReplicaSet scales down, all while still having a single Deployment object.

→ Notice in ***window 2*,** that our requests did not fail during the update but rather, we received a mix of traffic from v1 and v2 until eventually all requests were going to v2 pods.

#### **Task 3: Roll back**

What do we do if our new version is undesirable? For example, it crashes or is much too slow for the user. We can simply roll back to the previous configuration.

→ Roll back

|  |
| --- |
| kubectl rollout undo deployment app-deployment |

|  |
| --- |
| deployment.extensions/app-deployment rolled back |

→ View replica sets

|  |
| --- |
| kubectl get replicaset -o wide |

|  |
| --- |
| NAME DESIRED CURRENT READY AGE CONTAINERS IMAGES SELECTOR  app-deployment-6bd6dd6755 3 3 3 8m37s http-app acmeade/http-app:v1 app=http-app,pod-template-hash=6bd6dd6755  app-deployment-fdfff784d 0 0 0 6m6s http-app acmeade/http-app:v2 app=http-app,pod-template-hash=fdfff784d |

Notice that the old ReplicaSet scaled back up and the newer one back down. Essentially, a roll back is just a rolling update from v2 -> v1.

### **Task 4: Cleanup**

→ Delete resources

|  |
| --- |
| kubectl delete all --all |

→ In ***window 2***, type **Ctrl-C** to kill the loop

### **Task 5: Canary Deployment**

Canary deployments are similar to rolling deployments except we may want both versions running concurrently for longer so we can test the new version against real user traffic. This is a much more "manual" process that involves creating two Deployment objects with a Service that sends traffic to Pods of both.

#### **Step 1: Create the deployment**

→ In ***window 1***, create the new deployment.yaml

|  |
| --- |
| vi deployment\_v1.yaml |

→ Type '**:set paste'**, press **enter**. Then press **i**, paste the following:

|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:  name: app-deployment-v1  labels:  app: http-app  **version: v1**  spec:  replicas: 3  selector:  matchLabels:  app: http-app  template:  metadata:  labels:  app: http-app  **version: v1**  spec:  containers:  - name: http-app  **image: acmeade/http-app:v1**  ports:  - containerPort: 80 |

→ Hit **ESC**, then type '**:wq**' and press **enter** to save the file.

→ Apply changes

|  |
| --- |
| kubectl apply -f deployment\_v1.yaml |

|  |
| --- |
| deployment.apps/app-deployment-v1 created |

→ Recreate our service

|  |
| --- |
| kubectl apply -f service.yaml |

|  |
| --- |
| service/app-service created |

Okay, now we are back to having 3 pods running v1 of our application with a service acting as a load balancer. The difference is that this time, we have added the *version* label to our Pod spec. This will allow us to differentiate pods running different versions.

#### **Step 2: Test v2**

In order to create a canary deployment running v2, we are going to create a whole new deployment object that manages pods running v2. The differences in the two deployment yamls is in bold.

→ In ***window 1***, create the new deployment\_v2.yaml

|  |
| --- |
| vi deployment\_v2.yaml |

→ Type '**:set paste'**, press **enter**. Then press **i**, paste the following:

|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:  name: app-deployment-v2  labels:  app: http-app  **version: v2**  spec:  **replicas: 1**  selector:  matchLabels:  app: http-app  template:  metadata:  labels:  app: http-app  **version: v2**  spec:  containers:  - name: http-app  **image: acmeade/http-app:v2**  ports:  - containerPort: 80 |

→ Hit **ESC**, then type '**:wq**' and press **enter** to save the file.

→ Apply changes

|  |
| --- |
| kubectl apply -f deployment\_v2.yaml |

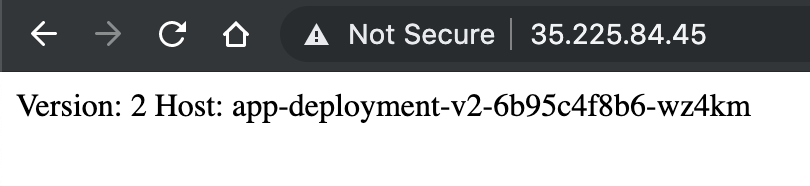
|  |
| --- |
| deployment.apps/app-deployment-v2 created |

→ Grab the app-service IP from your output

|  |
| --- |
| kubectl get services |

|  |
| --- |
| NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S)  AGE  app-service LoadBalancer 10.11.247.171 **35.225.84.45** 80:32343/TCP  15m |

→ In a web browser, go to the <IP>. **Refresh** a few times

Notice our service is balancing traffic across both our v1 and v2 pods. This is because our service is only looking at the *app* label on our pods, not our *version* label. It turns out, that about 25% of our requests go to v2, this is just because we have 3 v1 Pods and 1 v2 Pod.

For testing purposes, lets create a service that only sends requests to v2. This type of configuration is common so that we can test v2 directly while our end-users go through the main service.

→ Create a file named service\_v2.yaml

|  |
| --- |
| vi service\_v2.yaml |

→ Type ":set paste", press **enter**. Then press **i**, paste the following:

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:  labels:  app: http-app  name: app-service-v2  spec:  ports:  - port: 80  protocol: TCP  targetPort: 80  selector:  app: http-app  **version: v2**  sessionAffinity: None  type: LoadBalancer |

→ Hit **ESC**, then type ':wq' and press **enter** to save the file.

→ Create Service

|  |
| --- |
| kubectl apply -f service\_v2.yaml |

|  |
| --- |
| service/app-service-v2 created |

→ Grab it's IP, you may have to try multiple times while it is <pending>

|  |
| --- |
| kubectl get service app-service-v2 |

|  |
| --- |
| NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  app-service-v2 LoadBalancer 10.11.249.9 35.232.214.67 80:31329/TCP 45s |

→ Make some requests

|  |
| --- |
| curl <IP> |

|  |
| --- |
| Version: 2  Host: app-deployment-v2-6b95c4f8b6-wp4bd |

#### **Step 3: Deploy v2**

It turns out that the v2 of our app runs just fine, even with real user traffic. Let's finish the update to v2.

→ Open deployment\_v2.yaml

|  |
| --- |
| vi deployment\_v2.yaml |

→ move with the arrow keys and hover over the 1 on the following line

|  |
| --- |
| replicas: 1 |

→ type **r** then **3** in order to replace the character

→ type **:wq** to save the file

→ Apply changes

|  |
| --- |
| kubectl apply -f deployment\_v2.yaml |

|  |
| --- |
| deployment.apps/app-deployment-v2 configured |

→ Delete v1

|  |
| --- |
| kubectl delete deployment app-deployment-v1 |

→ Make a request to our original service IP in the browser

Continue refreshing and witness that we eventually only ever get v2, we have successfully completed the update of our application from v1 to v2.

### **Task 6: Cleanup**

→ Delete resources

|  |
| --- |
| kubectl delete all --all |