

Kubernetes Lab 1

Basic Kubernetes objects

1	REFERENCES AND CHEAT SHEETS.....	1
2	COMMANDS COVERED	1
3	LAB SETUP	2
4	STARTING A KUBERNETES CLUSTER	2
4.1	STARTING UP WITH MINIKUBE	2
4.2	STARTING WITH RANCHER DESKTOP	3
5	INTRO TO PODS	4
6	INTRO TO REPLICASETS	6
7	INTRO TO DEPLOYMENTS AND SERVICES	9
7.1	PERFORMING UPDATES AND ROLLBACKS WITH DEPLOYMENTS	11
7.2	CREATING DEPLOYMENTS FROM A CUSTOM IMAGE.....	14
7.3	USING SERVICES TO EXPOSE DEPLOYMENTS FOR ACCESS	16
7.4	SCALING POD REPLICAS TO MAINTAIN DESIRED STATE	17
7.5	UPDATING DEPLOYMENTS WITH CUSTOM IMAGES.....	19
7.6	ROLLING BACK DEPLOYMENT VERSIONS WITH ERRORS.....	21
7.7	DECLARATIVE MANIFEST FOR DEPLOYMENT AND SERVICE USING CUSTOM IMAGE.....	23
7.8	PERFORMING UPDATES AND ROLLBACKS WITH DEPLOYMENT	25
7.9	MERGING MANIFEST YAML FILES	26
7.10	CONFIGURING THE IMAGE PULL POLICY	27

1 References and cheat sheets

The official reference for all `kubectl` commands:

<https://kubernetes.io/docs/reference/kubectl/quick-reference/>

<https://spacelift.io/blog/kubernetes-cheat-sheet>

<https://www.bluematador.com/learn/kubectl-cheatsheet>

<https://www.geeksforgeeks.org/kubectl-cheatsheet/>

2 Commands covered

3 Lab setup

You should have basic Kubernetes installed through 2 possible approaches:

Approach a: Minikube (if you were using Docker Desktop)

<https://minikube.sigs.k8s.io/docs/start>

Approach b: Rancher Desktop

<https://rancherdesktop.io/>

The root folders for all the various projects here can be found in the `Lab 1` subfolder in the main `labcode` folder of your downloaded zip for this workshop.

4 Starting a Kubernetes cluster

4.1 Starting up with Minikube

For Minikube installation, when starting it, you will need to specify a driver to allow it to be deployed accordingly as either a VM, a container or bare metal.

<https://minikube.sigs.k8s.io/docs/drivers/>

If you just run `minikube start` on Windows without specifying a driver and without Docker running background, the automatic driver chosen is VirtualBox, which will typically crash since HyperV needs to be enabled to run Docker and VirtualBox requires Hyper V to be disabled.

The preferred approach on Windows would be to start Minikube using either the Hyper-V or Docker driver.

For the remaining lab sessions, we will use the Docker driver, but the Hyper V driver should also work in the event of any issues.

Make sure that you have Docker Desktop running before attempting to start it.

<https://minikube.sigs.k8s.io/docs/drivers/docker/>

In a Powershell terminal, start Minikube's single node Kubernetes cluster with:

```
minikube start --driver=docker
```

Check the status of the cluster with:

```
minikube status
```

If at any point, you get any error message similar to the following:

```
Unable to resolve the current Docker CLI context "default": context
"default": context not found:
```

Type this to set the context correctly to default.

```
docker context use default
```

Other possible approaches to resolve:

<https://stackoverflow.com/questions/77208814/cant-log-into-minikube-message-launching-minikube-has-me-concerned-unable-to>

This issue usually arises if you have multiple installations of Kubernetes (for e.g. using the Docker on Desktop option or Rancher desktop) on a single physical machine, all of which use different contexts when running. In that case, you can check for all the existing contexts matching to these installations with:

```
kubectl config get-contexts
```

and if you wish to use minikube as the Kubernetes cluster for the lab session, then you will need to set:

```
kubectl config set-context minikube
```

Check that the cluster nodes are running with:

```
kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
minikube	Ready	control-plane	63m	v1.30.0

Notice that the Minikube single node cluster is running within a Docker container with a specific configuration and port mapping:

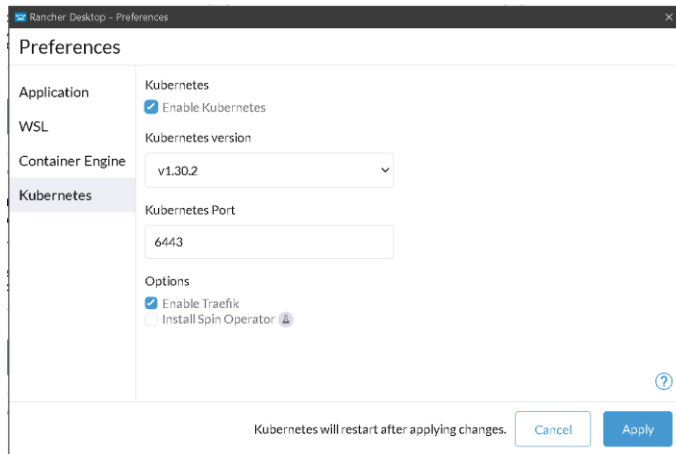
```
docker ps
```

Finally, you can start the built-in dashboard UI that allows you to monitor your various Kubernetes objects (pods, ReplicaSets, Deployments, Services) as an alternative to checking on these objects using the standard `kubectl` commands that we will be working with later.

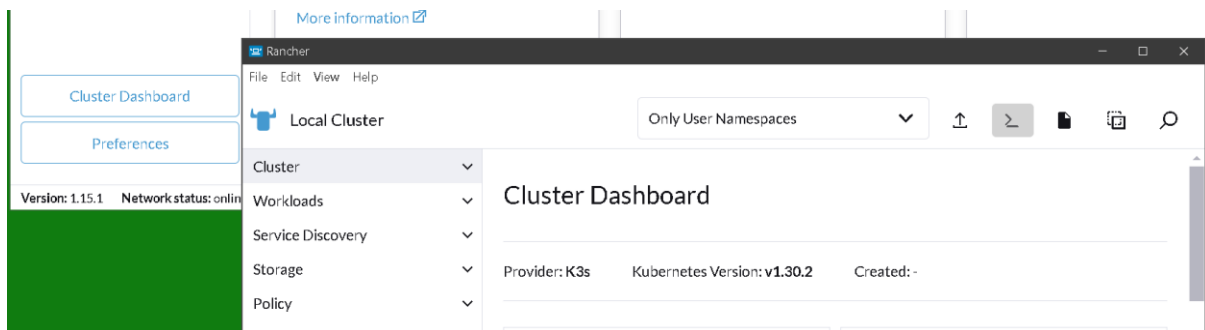
```
minikube dashboard
```

4.2 Starting with Rancher Desktop

Rancher Desktop already comes preinstalled with Kubernetes based on K3s (<https://docs.k3s.io/>), which by default should be already enabled. However, if you have disabled Kubernetes while working on your Docker labs, then remember to enable it again via preferences:



Once it is up and running you should be able to access the cluster dashboard in a separate UI window from the option at the bottom of the main pane:



This will allow you to monitor your various Kubernetes objects (pods, ReplicaSets, Deployments, Services) as an alternative to checking on these objects using the standard `kubectl` commands that we will be working with later.

Check that the cluster nodes are running by typing into a Powershell terminal:

```
kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
machinename	Ready	control-plane,master	63m	v1.30.0

5 Intro to pods

Kubernetes will by default pull images from a DockerHub repo (either official or custom user) to create a container within a single pod. Here, we will create a single pod running a container based on the image from the official DockerHub nginx repo:

https://hub.docker.com/_/nginx

You can execute all these `kubectl` CLI commands in a Powershell terminal in a folder that you have created specifically for these labs:

```
kubectl run nginx --image=nginx
```

Check on the status of all running pods with:

```
kubectl get pods
```

Initially, you will see a showing the status of the pod as ContainerCreating before it eventually transitions to Running.

You can get slightly more detailed info on all running pods with:

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

To obtain more descriptive info on any active running pod, reference the pod name:

```
kubectl describe pod nginx
```

Some of the relevant info from this description include name, IP address, node that the pod is running on, image, state, and the list of events within the pod up to the current point of time (which should indicate relevant activities such as starting the pod, pulling the relevant image from a DockerHub repo and starting a container from this image within the pod).

```
Name:          nginx
Namespace:     default
Node:          minikube/192.168.49.2
IP:            10.244.0.9
Image:         nginx
State:         Running
Ready:         True
.....
Events:
  Type     Reason      Age    From          Message
  ----     -
  Normal    Scheduled   2m27s  default-scheduler  Successfully assigned default/nginx to minikube
  Normal    Pulling     2m28s  kubelet        Pulling image "nginx"
  Normal    Pulled      2m2s   kubelet        Successfully pulled image "nginx" in 25.259s
(25.259s including waiting). Image size: 187603368 bytes.
  Normal    Created     2m2s   kubelet        Created container nginx
  Normal    Started     2m2s   kubelet        Started container nginx
```

You can remove a running pod by referencing the pod name again:

```
kubectl delete pod nginx
```

Check that it has been successfully deleted with:

```
kubectl get pods
```

There are 2 main ways to create objects:

- a) The imperative approach, that involves typing commands `kubectl run` with the specific options and objects that are to be created (demonstrated earlier)
- b) The declarative approach using a manifest configuration YAML file. This is conceptually equivalent to Docker Compose files that allow us to specify a number of services / containers that are started simultaneously and configured in a single file.

An example of a manifest to declaratively create a pod is shown at:

<https://kubernetes.io/docs/concepts/workloads/pods/#using-pods>

Create a configuration manifest file in an empty directory. We will use this to generate the same pod that we just created

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
    tier: frontend
spec:
  containers:
  - name: nginx
    image: nginx
```

Generate a new pod based on the configuration manifest with:

```
kubectl apply -f pod.yaml
```

Check that it is started with:

```
kubectl get pods
```

As before, you can get more info on it with:

```
kubectl describe pod nginx
```

You should be able to see the info provided here is nearly identical to the case when we generated the pods using

You can delete the pod in the same way:

```
kubectl delete pod nginx
```

6 Intro to ReplicaSets

An example of a manifest to declaratively create a ReplicaSet is shown at:

<https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/#example>

Create a configuration manifest file in an empty directory. We will use this to generate a ReplicaSet.

replicaset.yaml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
    app: myapp
spec:
  selector:
    matchLabels:
      env: production
  replicas: 3
  template:
    metadata:
      name: nginx-2
      labels:
        env: production
    spec:
      containers:
        - name: nginx
          image: nginx
```

Create the ReplicaSet with:

```
kubectl apply -f replicaset.yaml
```

NOTE: Sometimes, you will also see the `kubectl create` command being used with a configuration manifest instead of `kubectl apply`. This is fine if the configuration manifest is being used for the first to create the resource. Subsequently, after that you will need to use `kubectl apply` to update configuration changes to that existing resource.

Get info on the generated ReplicaSet and its associated pod replicas:

```
kubectl get replicaset
```

The relevant info here would be the desired number of pod replicas vs the number that are actually ready. It will take some time here for all 3 pod replicas to become ready from the time when the configuration is applied to generate the pods.

When we check on the pods:

```
kubectl get pods
```

You will see each of the pod replicas have a random number sequence appended to the ReplicaSet name in order to uniquely identify each of them and also indicate that they belong to the same ReplicaSet.

Try deleting any one of these pod replicas

```
kubectl delete pod myapp-ReplicaSet-xxxx
```

And check again on the running pods after a short while:

```
kubectl get pods
```

A new pod replica will be generated to replace the deleted one (it has a unique name from the one that was just deleted). Kubernetes will always automatically try to ensure that the number of pod replicas matches that specified in the configuration file.

Get more info on the ReplicaSet

```
kubectl describe replicaset myapp-ReplicaSet
```

Now we will create a configuration manifest for a new pod that uses the same label utilized by the ReplicaSet selector:

```
nginx.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-2
  labels:
    env: production
spec:
  containers:
    - name: nginx
      image: nginx
```

Notice here that the key value pair for the `labels` in this YAML manifest matches the key value pair for the `selector matchLabels` section in the `replicaset.yaml`, which means that the pod generated from the application of this YAML will be managed or targeted by this ReplicaSet.

Before proceeding, double check that there are only 3 replica pods running currently:

```
kubectl get pods
```

Then go ahead and try and create an extra pod using this new manifest file.

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

And if we check again with

```
kubectl get pods
```


we will see that the newly created pod was terminated as soon as it was started as the ReplicaSet would keep the number of replicas of that pod with that specified label to the set number of 3 as specified in the ReplicaSet YAML file.

If we check the event history log of the ReplicaSet

```
kubectl describe replicaset myapp-ReplicaSet
```

You will see the last event was the deletion of that newly created pod

To change the number of replicas to a higher number in order to scale it,

```
kubectl edit replicaset myapp-ReplicaSet
```

This opens up a copy of the original YAML file in the default editor: this copy is stored in memory and any changes made here will immediately be applied to the ReplicaSet. Change the key value pair `replicas : 3` to `replicas : 4` and save.

Now check the number of running pods again with:

```
kubectl get pods
```

Another way to change the number of replicas is via with:

```
kubectl scale replicaset myapp-ReplicaSet --replicas=2
```

Now check the number of running pods again with:

```
kubectl get pods
```

To delete an existing ReplicaSet, you can either explicitly specify the ReplicaSet based on its name:

```
kubectl delete replicaset <ReplicaSet-name> or
```

or use the manifest file that was originally used to create it with:

```
kubectl delete -f <file-name>.yaml
```

7 Intro to Deployments and Services

An example of a manifest to declaratively create a Deployment is shown at:

<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#creating-a-deployment>

Create a manifest configuration file for a Deployment in the same directory that you had used earlier:

```
deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
    tier: frontend
    app: nginx
spec:
  selector:
    matchLabels:
      app: myapp
  replicas: 3
  template:
    metadata:
      name: nginx-2
      labels:
        app: myapp
    spec:
      containers:
        - name: nginx
          image: nginx
```

Create the deployment using this manifest

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

Check for it and the pods with:

```
kubectl get deployments
```

```
kubectl get pods
```

Also note that a ReplicaSet is automatically generated for this deployment, which is how deployments manage scaling and replication for the pods. Get info on this generated ReplicaSet

```
kubectl get replicaset
```

You will notice by default it specifies only one replica.

Check for description:

```
kubectl describe deployment myapp-development
```

To obtain info on all objects available in the cluster at one go (for e.g. pods, deployments, replicaset, services, etc)

```
kubectl get all
```

7.1 Performing updates and rollbacks with deployments

First delete any existing deployments

```
kubectl delete deployment myapp-deployment
```

Update this previously used YAML file to use incorporate 6 replicas

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
    tier: frontend
    app: nginx
spec:
  selector:
    matchLabels:
      app: myapp
  replicas: 6
  template:
    metadata:
      name: nginx-2
      labels:
        app: myapp
    spec:
      containers:
        - name: nginx
          image: nginx
```

Next create the deployment:

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

and then check rollout status:

```
kubectl rollout status deployment.apps/myapp-deployment
```

Now delete, recreate this deployment and run the rollout status to see the effect:

```
kubectl delete deployment myapp-deployment
```

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

```
kubectl rollout status deployment.apps/myapp-deployment
```

You will see the rollout happening for each of the 6 pod replicas one at a time

To check the history:

```
kubectl rollout history deployment.apps/myapp-deployment
```

Repeat the deletion, but when recreating use the `--record` option which will result in the cause to be recorded as part of the rollout history

```
kubectl delete deployment myapp-deployment
```

```
kubectl create -f deployment.yaml --record
```

```
kubectl rollout history deployment.apps/myapp-deployment
```

Get more info on the deployment:

```
kubectl describe deployment myapp-deployment
```

Next edit the deployment and also record it:

```
kubectl edit deployment myapp-deployment --record
```

change to an earlier version of nginx, for e.g.

```
....
  spec:
    containers:
      - image: nginx:1.18.0
.....
```

You can also double check on the official DockerHub nginx repo to search for any valid earlier version tag to use here:

Then save and immediately check again on the rollout status:

```
kubectl rollout status deployment.apps/myapp-deployment
```

Here you will see new pods being created (with the latest version of nginx), and the old pods are being terminated

Check the events for the deployment to verify this has also happened with:

```
kubectl describe deployment myapp-deployment
```

Another way to change the image version for the pod replicas is to directly set via the CLI:

```
kubectl set image deployment myapp-deployment nginx=nginx:1.22.1-perl
--record
```

Once again immediately check again on the rollout status:

```
kubectl rollout status deployment.apps/myapp-deployment
```

Check the history again to see all the key changes to the deployment:

```
kubectl rollout history deployment.apps/myapp-deployment
```

Check the pods:

```
kubectl get pods
```

To revert / rollback the running pods back to the previous revision

```
kubectl rollout undo deployment.apps/myapp-deployment
```

And check again on the revert status:

```
kubectl rollout status deployment.apps/myapp-deployment
```

Double check that it has in fact reverted back to the older nginx image `nginx:1.18.0`

```
kubectl describe deployment myapp-deployment
```

If we check the history again, we notice that the 2nd revision is gone, and it has actually become revision 4 due to the undo operation.

```
kubectl rollout history deployment.apps/myapp-deployment
```

Now we make some more changes:

```
kubectl edit deployment myapp-deployment --record
```

Change the image to some random name that does not exist:

```
.....
  spec:
    containers:
      - image: nginx:weird-value
.....
```

And check again on the revert status:

```
kubectl rollout status deployment.apps/myapp-deployment
```

It will be stuck here since it cannot get this particular invalid image

In a different Powershell terminal, check on it with:

```
kubectl get deployment myapp-deployment
```

```
kubectl get pods
```

It has terminated one of the pods and is trying to start the other 3 pods, which are stuck because of attempting to pull an invalid image

Lets roll back to the previous valid revision:

```
kubectl rollout undo deployment.apps/myapp-deployment
```

Check the status again with:

```
kubectl describe deployment myapp-deployment
```

```
kubectl get pods
```

Finally, when you are done you can delete all the objects and resources associated with this deployment to commence work on the next lab session:

```
kubectl delete deployment myapp-deployment
```

7.2 Creating deployments from a custom image

The root folder for this project is: kub-action

In a Powershell terminal in this root project folder, build a custom image using the Dockerfile:

```
docker build -t kub-first-app .
```

Let's attempt to create a new deployment using this custom image directly with a command:

```
kubectl create deployment first-app --image=kub-first-app
```

and check for it:

```
kubectl get deployments
```

To check the reason why it is failing, type:

```
kubectl get pods
```

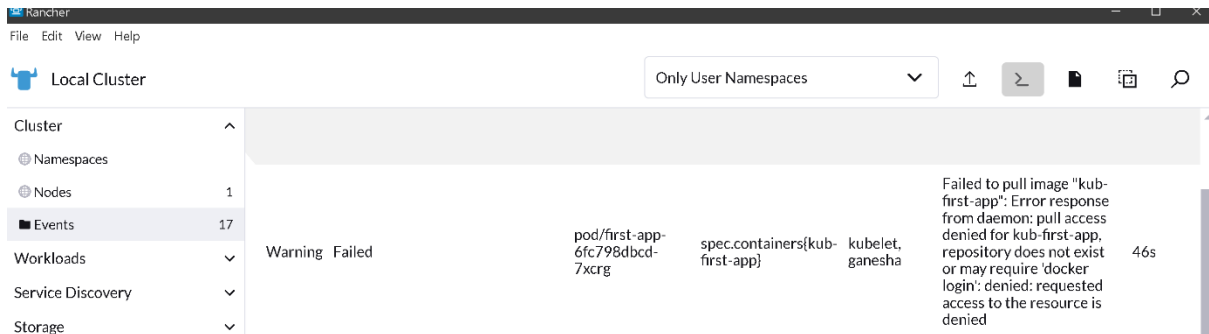
```
kubectl describe pod first-app
```

Note: You can also get all this status info from the dashboard (Minikube / Rancher Desktop) that you started earlier.

On Minikube

Events				
Name	Reason	Message	Source	
first-app-6fc798dbcd-sld42.17eb91f2c09737	BackOff	Back-off pulling image "kub-first-app"	kubelet minikube	
first-app-6fc798dbcd-sld42.17eb91f2c097f94	Failed	Error: ImagePullBackOff	kubelet minikube	
first-app-6fc798dbcd-sld42.17eb91f28c2623	Failed	Failed to pull image "kub-first-app": Error response from daemon: pull access denied for kub-first-app, repository does not exist or may require 'docker login': denied: requested access to the resource	kubelet minikube	

On Rancher



Here the image pull error is because the image only exists on the local registry on our local machine, and not within the container that is running the Kubernetes cluster (minikube or your local machine for the case of Rancher Desktop). Kubernetes default pull policy is to always in the first instance obtain the image specified imperatively or declaratively from the public Docker Hub registry.

<https://refine.dev/blog/kubernetes-image-pull-policy>

Delete the deployment:

```
kubectl delete deployment first-app
```

Now, lets push this new image we built to our DockerHub account

Retag the current image appropriately to enable it to be pushed correctly to this new repo in your DockerHub account:

```
docker tag kub-first-app dockerhubaccount/kub-first-app
```

Then push it:

```
docker push dockerhubaccount/kub-first-app
```

Verify that this new image is in fact in the repo as shown on the DockerHub page for this repo:

Next, we can repeat the previous deployment using this DockerHub image:

```
kubectl create deployment first-app --image=dockerhubaccount/kub-first-app
```

And verify that the deployment and the pods associated with it is up and running.

```
kubectl get deployments
```

```
kubectl get pods
```

SIDE NOTE: Possible options to the default image pull policy of Kubernetes (which is pull from Docker Hub).

Option A: If you are using Minikube, a possible approach is to build your image within the local registry of the Docker container running Minikube as explained in detailed in the steps below:

<https://stackoverflow.com/questions/42564058/how-can-i-use-local-docker-images-with-minikube>

<https://minikube.sigs.k8s.io/docs/handbook/pushing/#1-pushing-directly-to-the-in-cluster-docker-daemon-docker-env>

Option B: Configure a local private registry (external to the local registry of the Docker container that Minikube is in) that runs as a Docker container, and then configure secrets to allow Kubernetes to authenticate and pull from that local private registry:

<https://stackoverflow.com/questions/58654118/pulling-local-repository-docker-image-from-kubernetes>

<https://spacelift.io/blog/kubernetes-imagepullpolicy#running-a-container-from-an-image-in-a-container-registry>

<https://stackoverflow.com/questions/36874880/kubernetes-cannot-pull-local-image>

7.3 Using services to expose deployments for access

Expose the deployment for external access from localhost with one of the 2 possible types of services which enable external access to pods on a node: LoadBalancer or PortNode

```
kubectl expose deployment first-app --type=LoadBalancer --port=8080
```

Check that it started with:

```
kubectl get services
```

If you are using minikube to implement your K8s cluster, you can access the service with:

```
minikube service first-app
```

This will create a URL with a tunneling port which is then opened in a new browser tab that allows you access the Node.js application within the container within the pod. The tunneling port is necessary to link the original specified application container port (8080) to a port that can be directly accessed from a browser tab. This tunneling port is thus available on the loopback address.

If you press Ctrl+C to return to the terminal, the IP address and port specified are no longer valid.

We will see later on how to declaratively specify a service in a manifest configuration, the same way that we have done for pods, ReplicaSets and deployments.

If you are using Rancher Desktop, you should be able to directly access this service through the port 8080 on localhost

<http://localhost:8080/>

This is the original specified application container port (8080) and Rancher automatically maps it to an equivalent port on localhost for case of a Service of type `loadbalancer`.

7.4 Scaling Pod replicas to maintain desired state

Previously we have seen how ReplicaSets will ensure that Pod replicas are kept to the specified number in the spec: either by generating new pods to replace pods that have been deleted or deleting new extra pods that have been generated.

We will replicate the same effect, which will be performed by the ReplicaSet object that is automatically generated with the creation of a new deployment.

There is already a specific command to explicitly crash the app related to an API endpoint path implemented in `app.js`

```
app.get('/error', (req, res) => {  
  process.exit(1);  
});
```

Now if you access this specific endpoint (`/error`) in the open browser tab, it should crash and the webpage will report this accordingly. Alternatively, use a REST client (Postman) to send the HTTP GET request to this endpoint, as you can control how many times the request is sent vs a browser which will automatically periodically send multiple requests to the same endpoint when it detects an error response.

If you immediately check via the commands (as well as in the dashboard).

```
kubectl get deployments
```

```
kubectl get pods
```

You will see that the container in the pod is down. After approximately 15 - 30 seconds, the pod should revert to `CrashLoopBackoff` status and eventually return to running and the app should again be accessible at its root domain: `/`

You can try crashing it again by navigating to this specific endpoint (`/error`) in the open browser tab or using the REST client (Postman) to send this GET Request. You will notice it will eventually restart again with a longer time period between each restart.

The status will be `CrashLoopBackoff` for the commands:

```
kubectl get pods
```

More detailed info is available in the dashboard showing the number of restarts, and the latest event in the event list.

Daemon Sets

Deployments

Jobs

Pods

Replica Sets

Replication Controllers

Stateful Sets

Service

first-app-86f74dd75b-jpf52

default

Aug 1, 2024

2 hours ago

834b2c22-f91d-4c97-8b1f-3c8cd26705f2

Labels

app: first-app pod-template-hash: 86f74dd75b

Resource information

Node

Status

IP

QoS Class

Restarts

Service Account

minikube

Running

10.244.0.11

BestEffort

5

default

Name	Reason	Message	Source	Sub-object	Count	First Seen	Last Seen ↑
<div><div></div><div>first-app-86f74dd75b-jpf52.17e78b5e7b650627</div></div>	BackOff	Back-off restarting failed container kub-first-app in pod first-app-86f74dd75b-jpf52_default(834b2c22-f91d-4c97-8b1f-3c8cd26705f2)	kubelet minikube	spec.containers(kub-first-app)	10	14 minutes ago	8 minutes ago
<div><div></div><div>first-app-86f74dd75b-jpf52.17e783344624d326</div></div>	Started	Started container kub-first-app	kubelet minikube	spec.containers(kub-first-app)	5	2 hours ago	12 minutes ago
<div><div></div><div>first-app-86f74dd75b-jpf52.17e783343d28e26c</div></div>	Created	Created container kub-first-app	kubelet minikube	spec.containers(kub-first-app)	5	2 hours ago	12 minutes ago
<div><div></div><div>first-app-86f74dd75b-jpf52.17e78b57bb1b0abaf</div></div>	Pulled	Successfully pulled image "chirondeveloper/kub-first-app" in 2.381s (2.381s including waiting). Image size: 123931515 bytes.	kubelet minikube	spec.containers(kub-first-app)	1	12 minutes ago	12 minutes ago
<div><div></div><div>first-app-86f74dd75b-jpf52.17e7833131f778a8</div></div>	Pulling	Pulling image "chirondeveloper/kub-first-app"	kubelet minikube	spec.containers(kub-first-app)	5	2 hours ago	12 minutes ago
<div><div></div><div>first-app-86f74dd75b-jpf52.17e78b6e73fe13df</div></div>	Pulled	Successfully pulled image "chirondeveloper/kub-first-app" in 2.512s (2.512s including waiting). Image size: 123931515 bytes.	kubelet minikube	spec.containers(kub-first-app)	1	13 minutes ago	13 minutes ago

Notice that the restart process for the first time when container crashes including the following events: pulling, pulled, created, started and also backoff (to prevent too quickly a restart). Subsequently, after that the restart is primarily just the backoff event.

This info is also available from the CLI:

```
kubectl describe pod first-app
```

We already saw earlier that when we create a new deployment, the autogenerated ReplicaSet will specify exactly one replica for all pods specified in the pod template section.

```
kubectl get replicaset
```

We can however change the required number of replicas imperatively with:

```
kubectl scale deployment/first-app --replicas=3
```

Now if we check the ReplicaSet again, we should see this change:

```
kubectl get replicaset
```

And also the number of pods available:

```
kubectl get pods
```

We should be able to see the 3 pods now also in the dashboard

To verify that load balancing is in effect, we will trigger a crash in a browser tab by sending a HTTP Request again (either through a browser or a REST client) to the `/error` endpoint. Check with:

```
kubectl get pods
```

For every request sent, one pod will go down, so if you sent multiple requests successively quickly, you can take down more than one pod.

And now if you send a HTTP request back to the root domain `/`, we can see that the app is still accessible (i.e. from a different pod) - which indicates the load balancer automatically redirects HTTP traffic to any of the remaining running pods.

You can view the process of the pods going down and coming back live again in the dashboard pane.

Pods									
Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑	
● first-app-86f74cd75b-fbj2j	chirondeveloper/kub-first-app	app: first-app pod-template-hash: 86f74cd75b	minikube	Error	2	-	-	4 minutes ago	⋮
● first-app-86f74cd75b-wfj4t	chirondeveloper/kub-first-app	app: first-app pod-template-hash: 86f74cd75b	minikube	Running	1	-	-	4 minutes ago	⋮
● first-app-86f74cd75b-jpf52	chirondeveloper/kub-first-app	app: first-app pod-template-hash: 86f74cd75b	minikube	Error	7	-	-	2 hours ago	⋮

Pods									
Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑	
● first-app-86f74cd75b-fbj2j	chirondeveloper/kub-first-app	app: first-app pod-template-hash: 86f74cd75b	minikube	Running	3	-	-	4 minutes ago	⋮
● first-app-86f74cd75b-wfj4t	chirondeveloper/kub-first-app	app: first-app pod-template-hash: 86f74cd75b	minikube	Running	1	-	-	4 minutes ago	⋮
● first-app-86f74cd75b-jpf52	chirondeveloper/kub-first-app	app: first-app pod-template-hash: 86f74cd75b	minikube	Running	8	-	-	2 hours ago	⋮

You can scale it back down to 1 replica if you wish with:

```
kubectl scale deployment/first-app --replicas=1
```

You will now see the 2 other pods terminating in the background with:

```
kubectl get pods
```

This is also visible from the dashboard. Eventually, you will be left with exactly one running pod.

7.5 Updating deployments with custom images

Earlier we saw how to update deployments with multiple different image versions (i.e. with different tags). We now repeat the same process here but using the custom image we have been working with.

Update the `app.js` to simulate new changes

```
app.get('/', (req, res) => {  
  res.send(`  
    <h1>Hello from this NodeJS app!</h1>  
    <p>Try sending a request to /error and see what happens</p>  
    <h2>Some new stuff down here</h2>  
  `);  
});
```

Build a new image with a different tag from the one that we had previously used for pushing to our DockerHub account:

```
docker build -t dockerHubAccount/kub-first-app:v2 .
```

Then push this updated image to DockerHub in the usual way:

```
docker push dockerHubAccount/kub-first-app:v2
```

To update the deployment to a new image, we also need to specify which current container and image to update with which future image:
for e.g.

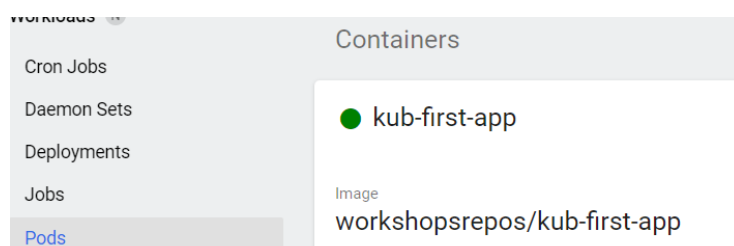
```
kubectl set image deployment/first-app container-name=new-image-to-use
```

To obtain the container name running in a pod, you can use the describe command to get detailed info:

```
kubectl describe pod first-app
```

```
Name:          first-app-7cd8895d96-vjhc2  
Namespace:     default  
...  
...  
Controlled By: ReplicaSet/first-app-7cd8895d96  
Containers:  
  kub-first-app:
```

It is also visible from the dashboard



Typically, the name of the container will be the same as the tag of the image it was generated from, in this case: `kub-first-app`

The command then becomes for our case:

```
kubectl set image deployment/first-app kub-first-app=dockerHubAccount/kub-first-app:v2
```

You can view the status of the ongoing update immediately after applying it with:

```
kubectl rollout status deployment/first-app
```

You should now be able to view the latest changes you have incorporated in your source code by refreshing the webpage at the root domain \

You can also see that this new image was pulled in the events part of the Dashboard.

Events							
Name	Reason	Message	Source	Sub-object	Count	First Seen	Last Seen ↑
first-app-55b687b87d-6qdsh.17e78de551d62bbf	Pulled	Successfully pulled image "chirondeveloper/kub-first-app:v2" in 4.287s (4.287s including waiting). Image size: 123931543 bytes.	kubelet minikube	spec.containers(kub-first-app)	1	3 minutes ago	3 minutes ago
first-app-55b687b87d-6qdsh.17e78de556ded589	Created	Created container kub-first-app	kubelet minikube	spec.containers(kub-first-app)	1	3 minutes ago	3 minutes ago
first-app-55b687b87d-6qdsh.17e78de55e5cea84	Started	Started container kub-first-app	kubelet minikube	spec.containers(kub-first-app)	1	3 minutes ago	3 minutes ago
first-app-55b687b87d-6qdsh.17e78de452455cd6	Pulling	Pulling image "chirondeveloper/kub-first-app:v2"	kubelet minikube	spec.containers(kub-first-app)	1	3 minutes ago	3 minutes ago
first-app-55b687b87d-6qdsh.17e78de428baed00	Scheduled	Successfully assigned default/first-app-55b687b87d-6qdsh to minikube		-	0	3 minutes ago	3 minutes ago

7.6 Rolling back deployment versions with errors

Let's try updating the deployment again but this time with an image that does not exist for e.g.

```
kubectl set image deployment/first-app kub-first-app=dockerhubaccount/kub-first-app:xxxx
```



Then immediately check on the status of the deployment with:

```
kubectl rollout status deployment/first-app
```

After a while some error messages coming out:

```
Waiting for deployment "first-app" rollout to finish: 1 old replicas are pending termination...
error: deployment "first-app" exceeded its progress deadline
```

The dashboard also shows an issue with a new pod trying to start by attempting to pull the non-existent image

Pods						
Name	Images	Labels	Node	Status	Restarts	CF
 first-app-765587fdd4-hncqf	chirondeveloper/kub-first-app:vxxx	app: first-app pod-template-hash: 765587fdd4	minikube	ImagePullBackOff	0	-
 first-app-55b687b87d-6qdsh	chirondeveloper/kub-first-app:v2	app: first-app pod-template-hash: 55b687b87d	minikube	Running	0	-

This old replica is not terminating, because the new pod is not starting up successfully due to issues with pulling a non-existent image. Because of the rolling update strategy that Kubernetes uses, it doesn't shut down the old pod before the new pod is up and running.

We verify again which pods are stuck:

```
kubectl get pods
```

In the events section for this pod in the dashboard, it encounters an error when trying to pull the image, backs off for a short period of time, and reattempts the pull again, and this process continues indefinitely.

And let's now roll back this problematic deployment, which clearly won't complete here.

```
kubectl rollout undo deployment/first-app
```

And check pods again:

```
kubectl get pods
```

The original one pod is running, and the problematic pod is gone. You can also verify in the dashboard pane.

Now if you wanna go back to an even older deployment, so not just to the previous one, we can first of all have a look at the deployment history

```
kubectl rollout history deployment/first-app
```

We can then checkout which particular image was used in any particular one of the deployment revisions in the past:

```
kubectl rollout history deployment/first-app --revision=3
```

This is the one that caused the error to occur due to a non-existent image

```
kubectl rollout history deployment/first-app --revision=4
```

This is the latest update that rolled back to the previous deployment using a working image. It is actually revision #2 now applied to become #4 in the sequence.

To specifically roll back to a deployment configuration far back in the history, for e.g. the first:

```
kubectl rollout history deployment/first-app --revision=1
```

```
kubectl rollout undo deployment/first-app --to-revision=1
```

Now you should see the webpage corresponding to the first initial deployment based on our first initial build (`docker build -t kub-first-app .`) without the extra web page content that we introduced later.

Let's delete the existing service with:

```
kubectl delete service first-app
```

Then delete the deployment:

```
kubectl delete deployment first-app
```

The deployments, ReplicaSets, services and pods should now disappear from the dashboard pane.

7.7 Declarative manifest for deployment and service using custom image

Let's create a new configuration manifest to create a deployment that uses the custom images that we had generated earlier.

```
deployment-custom.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: second-app-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: second-app
      tier: backend
  template:
    metadata:
      labels:
        app: second-app
        tier: backend
    spec:
      containers:
        - name: second-node
          image: dockerhubaccount/kub-first-app:v2
        # - name: ...
        #   image: ...
```

To create the deployment specified in this configuration file:

```
kubectl apply -f deployment-custom.yaml
```

And check that it is up and running with:

```
kubectl get deployments
```

```
kubectl get pods
```

You can also verify in the dashboard pane that the single pod replica associated with this deployment has the labels assigned to it in the configuration file.

Next, create a `service-custom.yaml` file to declaratively create a service that will expose this deployment for access:

```
service-custom.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  name: backend
spec:
  selector:
    app: second-app
  ports:
    - protocol: 'TCP'
      port: 80
      targetPort: 8080
    # - protocol: 'TCP'
    #   port: 443
    #   targetPort: 443
  type: LoadBalancer
```

Create this service in the usual way:

```
kubectl apply -f service-custom.yaml
```

Check that it is available with:

```
kubectl get services
```


Then expose this service with Minikube in the usual way:

```
minikube service backend
```

This will create a URL with a tunneling port which is then opened in a new browser tab that allows you access the Node application within the container within the pod.

7.8 Performing updates and rollbacks with deployment

We have done this before previously, which we repeat here again with this new deployment.

Change this in `deployment-custom.yaml`

```
replicas: 3
```

Then apply it again to update the deployment state:

```
kubectl apply -f deployment-custom.yaml
```

Check that there are now 3 pod replicas running:

```
kubectl get deployments
```

```
kubectl get pods
```

Also check in the Minikube dashboard

Make another change in `deployment-custom.yaml` to go back to an earlier version of the image

```
spec:
  containers:
    - name: second-node
      image: dockerhubaccount/kub-first-app
```

Then apply it again to update the deployment state:

```
kubectl apply -f deployment-custom.yaml
```

Reload and check that the webpage renders the HTML corresponding to the initial version of `app.js`

You can also check the deployment revision history

```
kubectl rollout history deployment/second-app-deployment
```

Now change `deployment-custom.yaml` to revert back again to the latest image revision

```
spec:
  containers:
    - name: second-node
      image: dockerhubaccount/kub-first-app:v2
```

Then apply it again to update the deployment state:

```
kubectl apply -f deployment-custom.yaml
```

Reload and check that the webpage renders the HTML corresponding to the latest version of `app.js`

To delete the running deployment, we could either use an existing command that we used for imperatively declared deployments, i.e.

```
kubectl delete deployment second-app-deployment
```

Or we could use the deployment file to specify deletion of all objects / resources specified in there with:

```
kubectl delete -f deployment-custom.yaml
```

7.9 Merging manifest YAML files

If we wish, we can create a single manifest configuration file that includes all the configuration for all the objects that are relevant for our cluster (deployments, services and any other related workload resources).

Create a `master-deployment.yaml` file that merges the previous 2 separate config files

```
master-deployment.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  name: backend
spec:
  selector:
    app: second-app
  ports:
    - protocol: 'TCP'
      port: 80
      targetPort: 8080
```

```
# - protocol: 'TCP'
#   port: 443
#   targetPort: 443
type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: second-app-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: second-app
      tier: backend
  template:
    metadata:
      labels:
        app: second-app
        tier: backend
    spec:
      containers:
        - name: second-node
          image: dockerhubaccount/kub-first-app:v2
        # - name: ...
        #   image: ...
```

We can bring down the deployment and the service we created earlier with:

```
kubectl delete -f deployment-custom.yaml -f service-custom.yaml
```

Then restart both of them again with this single file:

```
kubectl apply -f master-deployment.yaml
```

As usual open the service on the browser with:

```
minikube service backend
```

7.10 Configuring the image pull policy

Kubernetes has an image pull policy that can cause subtle errors if not understood properly:

<https://spacelift.io/blog/kubernetes-imagepullpolicy#image-pull-policy>

To see this in action, make a change to `app.js` to add in more additional output statements with a variety of HTML headers with random content such as:

`app.js`

```
app.get('/', (req, res) => {
  res.send(`
    <h1>Hello from this NodeJS app!</h1>
    <p>Try sending a request to /error and see what happens</p>
    <p>More new stuff </p>
    <p>And lots more stuff </p>
  `);
});
```

Now build it again with exactly the same image name that we pushed up to our DockerHub account most recently:

```
docker build -t dockerHubAccount/kub-first-app:v2 .
```

and then push it to the DockerHub account repo:

```
docker push dockerHubAccount/kub-first-app:v2 .
```

Now if you attempt to get the latest image that you just pushed by reapplying the configuration manifest:

```
kubectl apply -f master-deployment.yaml
```

The output message will indicate that nothing has changed, and that Kubernetes did not pull down this latest image update from your DockerHub account to recreate the pod. This is because the default image policy (if none is specified) is according to the rules:

If the image tag isn't `:latest` (which in this case is `v2`), the `imagePullPolicy` will be automatically set to `IfNotPresent`. This means Kubernetes will only pull the image when it doesn't already exist on the node. In this case, the image with that specific tag already exists and Kubernetes does not keep track of timestamps of revisions on the image, and so does not know that it has changed from the last time it was applied in the existing deployment.

This is problematic because applying the configuration manifest again will not update our pod with the latest container image unless we explicitly build the image again with a completely new tag (for e.g. `dockerHubAccount/kub-first-app:v3` and then use this new tag in the spec for the container in the configuration manifest).

However, we can also configure the image pull policy, so that we always pull the image from the repository. This ensures that if we ever do a rebuild of that image without changing its tag and push it to DockerHub account, Kubernetes will always pull the image from DockerHub whenever the configuration manifest is applied.

Make this change to `master-deployment.yaml` to configure the policy in the way just described.

`master-deployment.yaml`

```
....  
  spec:  
    containers:  
      - name: second-node  
        image: dockerhubaccount/kub-first-app:v2  
        imagePullPolicy: Always  
.....
```

Now if you reapply the `master-deployment.yaml` file with the new `imagePullPolicy`

```
kubectl apply -f master-deployment.yaml
```

You will see it will stop the previous pod and start a new pod to pull down the latest updated image (even though the image tag did not change).

```
kubectl rollout status deployment/second-app-deployment
```

If this does not happen automatically, you may need to bring down the existing deployment first before restarting it:

```
kubectl delete -f master-deployment.yaml
```

```
kubectl apply -f master-deployment.yaml
```

Check the webpage at the service URL / port and refresh it to see that the latest changes you made to `app.js` are now finally present

Finally, when you are done, clean up all deployments and services:

```
kubectl delete -f master-deployment.yaml
```