**Kubernetes**

* What is Kubernetes?
* Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.
* Kubernetes is a container Orchestration technology used to orchestrate the deployment and management of 100s and 1000s of containers in a clustered environment.
* What are Containers and Orchestration?
* Containers are completely isolated environments, they can have their own process or services their own network interfaces, their own mounts, just like virtual machines, except that they all share the same OS kernel.
* Containers have existed for about 10 years now and some of the different types of containers are LXC, LXD, LXCFS etc. Docker utilizes LXC containers. Setting up these container environments is hard as they are very low level and that is where Docker offers a high-level tool with several powerful functionalities making it really easy for end users like us.
* Container orchestration is the process of automatically deploying, scaling, networking, and managing of containers is known as Container Orchestration.

**Kubernetes Architecture:**

* What is Node?
* A node is a machine physical or virtual on which kubernetes is installed.
* A node is a worker machine, and this is where containers will be launched by kubernetes.
* It was also known as Minions in the past.
* What is cluster?
* A cluster is a set of nodes grouped together.
* This way even if one node fails you have your application still accessible from the other nodes. Moreover, having multiple nodes helps in sharing load as well.
* What is Master Node?
* The master is another node with Kubernetes installed in it and is configured as a Master.
* The master watches over the nodes in the cluster and is responsible for the actual orchestration of containers on the worker nodes.
* What is kubernetes component?

Following are the components.

1. API Server.
2. ETCD service.
3. kubeletservice.
4. Container Runtime
5. Controllers
6. Schedulers.

* **API Server**: The API server acts as the front-end for kubernetes. The users, management devices, Command line interfaces all talk to the API server to interact with the kubernetes cluster.
* **ETCD Service:** ETCD is a distributed reliable key-value store used by kubernetes to store all data used to manage the cluster. Think of it this way, when you have multiple nodes and multiple masters in your cluster, etcdstores all that information on all the nodes in the cluster in a distributed manner. ETCD is responsible for implementing locks within the cluster to ensure there are no conflicts between the Masters.
* **Schedulers:** The scheduler is responsible for distributing work or containers across multiple nodes. It looks for newly created containers and assigns them to Nodes.
* **Controllers:** The controllers are the brain behind orchestration. They are responsible for noticing and responding when nodes, containers or endpoints goes down. The controllers make decisions to bring up new containers in such cases.
* **Container Runtime:** The container runtime is the underlying software that is used to run containers. In our case it happens to be Docker.
* **Kubelet:** Kubelet is the agent that runs on each node in the cluster. The agent is responsible for making sure that the containers are running on the nodes as expected.
* What is aim of kubernetes?

With kubernetes our ultimate aim is to deploy our application in the form of containers on a set of machines that are configured as worker nodes in a cluster. kubernetes does not deploy containers directly on the worker nodes.

* What is POD?
* The containers are encapsulated into a Kubernetes object known as PODs.
* A POD is a single instance of an application.
* A POD is the smallest object, that you can create in kubernetes.

On a single node kubernetes cluster a single instance of your application running in a single docker container encapsulated in a POD.

What if the number of users accessing your application increase and you need to scale your application?

You need to add additional instances of your web application to share the load. Now, were would you spin up additional instances? Do we bring up a new container instance within the same POD? No!

We create a new POD altogether with a new instance of the same application. As you can see we now have two instances of our web application running on two separate PODs on the same kubernetes system or node.

What if the user base FURTHER increases and your current node has no sufficient capacity? Well THEN you can always deploy additional PODs on a new node in the cluster. You will have a new node added to the cluster to expand the cluster’s physical capacity.

SO, what I am trying to illustrate in this slide is that, PODs usually have a one-to-one relationship with containers running your application.

To scale UP you create new PODs and to scale down you delete PODs.

You do not add additional containers to an existing POD to scale your application. Also, if you are wondering how we implement all of this and how we achieve load balancing between containers etc, we will get into all of that in a later lecture.

PODs usually have a one-to-one relationship with the containers, but, are we restricted to having a single container in a single POD? No!

A single POD CAN have multiple containers, except for the fact that they are usually not multiple containers of the same kind.

As we discussed in the previous slide, if our intention was to scale our application, then we would need to create additional PODs. But sometimes you might have a scenario were you have a helper container, that might be doing some kind of supporting task for our web application such as processing a user entered data, processing a file uploaded by the user etc. and you want these helper containers to live along side your application container.

In that case, you CAN have both of these containers part of the same POD, so that when a new application container is created, the helper is also created and when it dies the helper also dies since they are part of the same POD.

The two containers can also communicate with each other directly by referring to each other as ‘localhost’ since they share the same network namespace. Plus they can easily share the same storage space as well.

If you still have doubts in this topic (I would understand if you did because I did the first time I learned these concepts), we could take another shot at understanding PODs from a different angle. Let’s, for a moment, keep kubernetes out of our discussion and talk about simple docker containers. Let’s assume we were developing a process or a script to deploy our application on a docker host. Then we would first simply deploy our application using a simple docker run python-app command and the application runs fine and our users are able to access it. When the load increases we deploy more instances of our application by running the docker run commands many more times. This works fine and we are all happy. Now, sometime in the future our application is further developed, undergoes architectural changes and grows and gets complex. We now have new helper containers that helps our web applications by processing or fetching data from elsewhere. These helper containers maintain a one-to-one relationship with our application container and thus, needs to communicate with the application containers directly and access data from those containers. For this we need to maintain a map of what app and helper containers are connected to each other, we would need to establish network connectivity between these containers ourselves using links and custom networks, we would need to create shareable volumes and share it among the containers and maintain a map of that as well. And most importantly we would need to monitor the state of the application

container and when it dies, manually kill the helper container as well as its no longer required. When a new container is deployed we would need to deploy the new helper container as well.

With PODs, kubernetes does all of this for us automatically. We just need to define what containers a POD consists of and the containers in a POD by default will have access to the same storage, the same network namespace, and same fate as in they will be created together and destroyed together.

Even if our application didn’t happen to be so complex and we could live with a single container, kubernetes still requires you to create PODs. But this is good in the long run as your application is now equipped for architectural changes and scale in the future.

However, multi-pod containers are a rare use-case and we are going to stick to single container per POD in this course.

Kubernetes Concepts - <https://kubernetes.io/docs/concepts/>

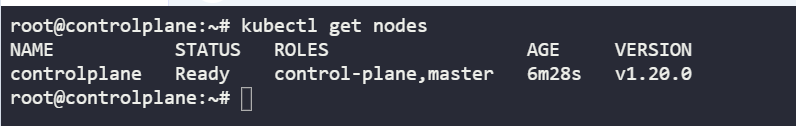
Pod Overview- <https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/>

**Kubernetes Lab practice**

**Nodes**

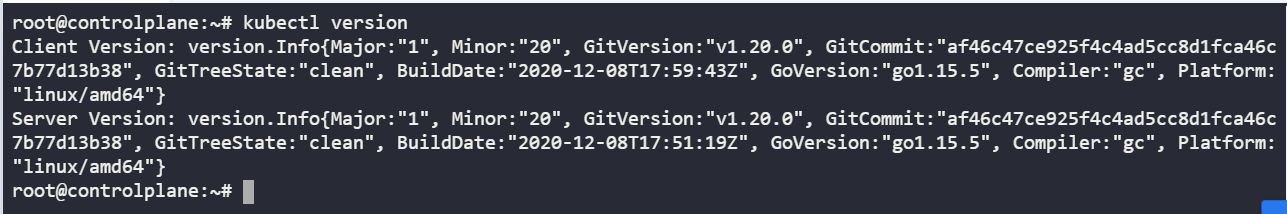
* How many nodes are part of the cluster?

kubectl get nodes



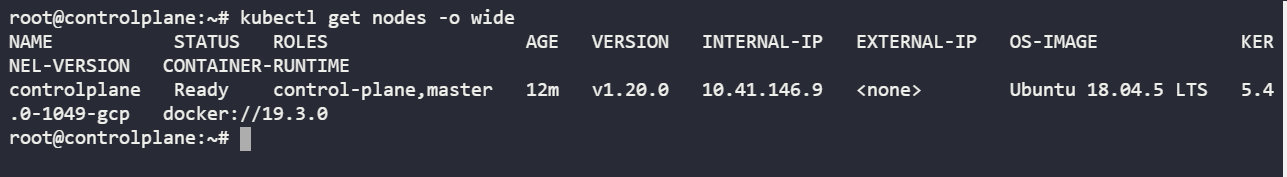
* What is the version of Kubernetes running on the nodes ?

kubectl version



* What is the flavor and version of Operating System on which the Kubernetes nodes are running?

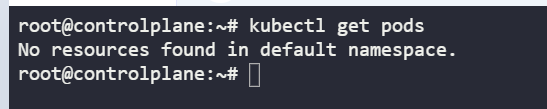
Type kubectl get nodes -o wide and look at the Operating System.



PODS:

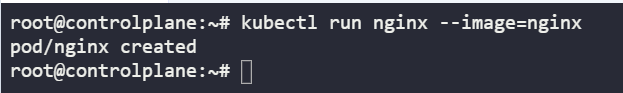
* How many pods exist on the system? In the current(default) namespace.

Run the command kubectl get pods and count the number of pods.



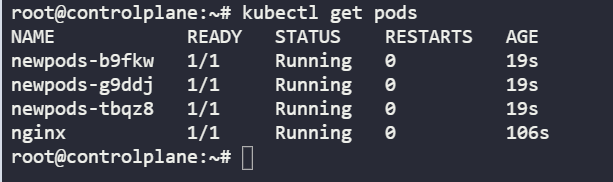
* Create a new pod with the nginx image.

Run the command: kubectl run nginx --image=nginx



* How many pods are created

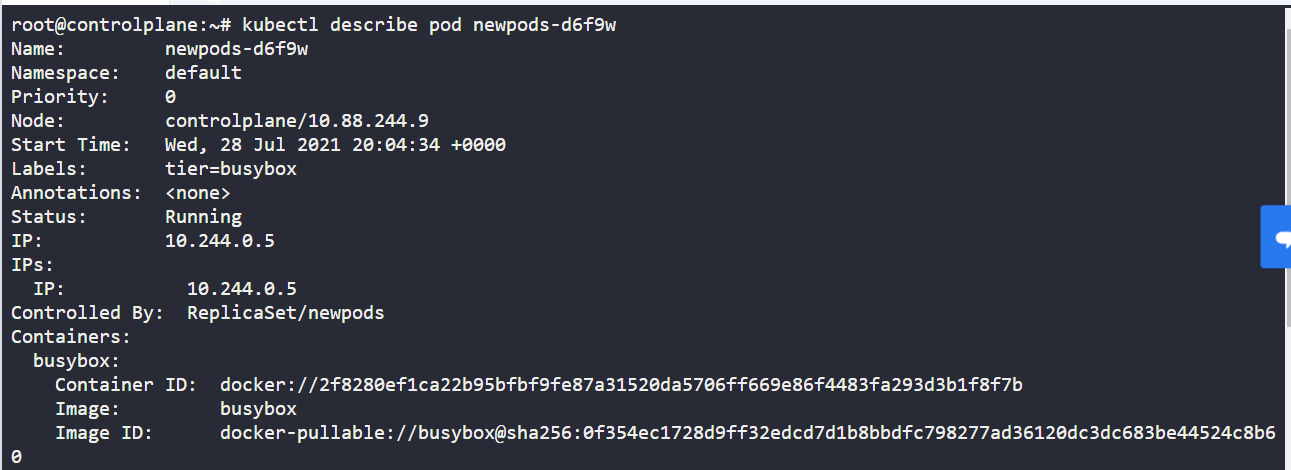
kubectl get pods



* What is the image used to create the new pods? You must look at one of the new pods in detail to figure this out.

kubectl describe pod newpods-<id>| grep -I image

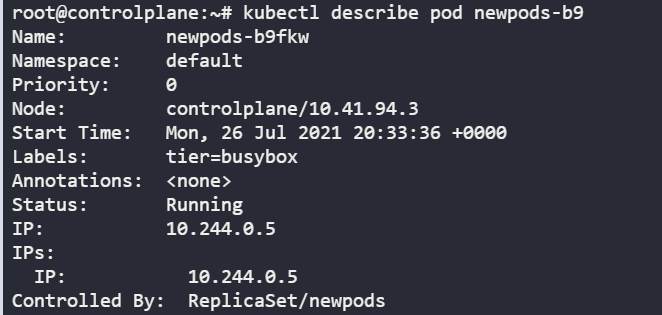
Run the command kubectl describe pod newpods-<id> and look under the containers section.



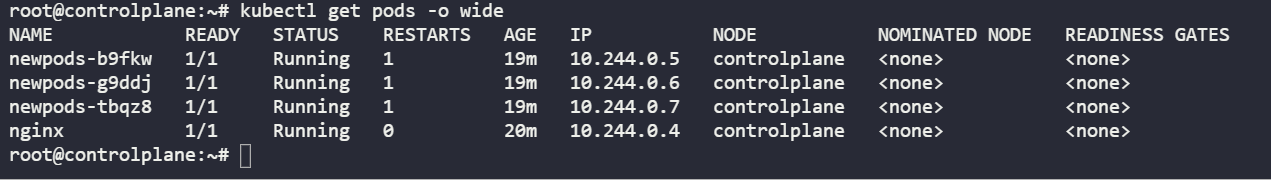
* Which nodes are these pods placed on? You must look at all the pods in detail to figure this out.

Run the command kubectl describe pod newpods-<id> and look at the node field.

Solution : Controlplan



Alternatively run kubectl get pods -o wide and check for the node the pod is placed on.



* How many containers are part of the pod webapp?

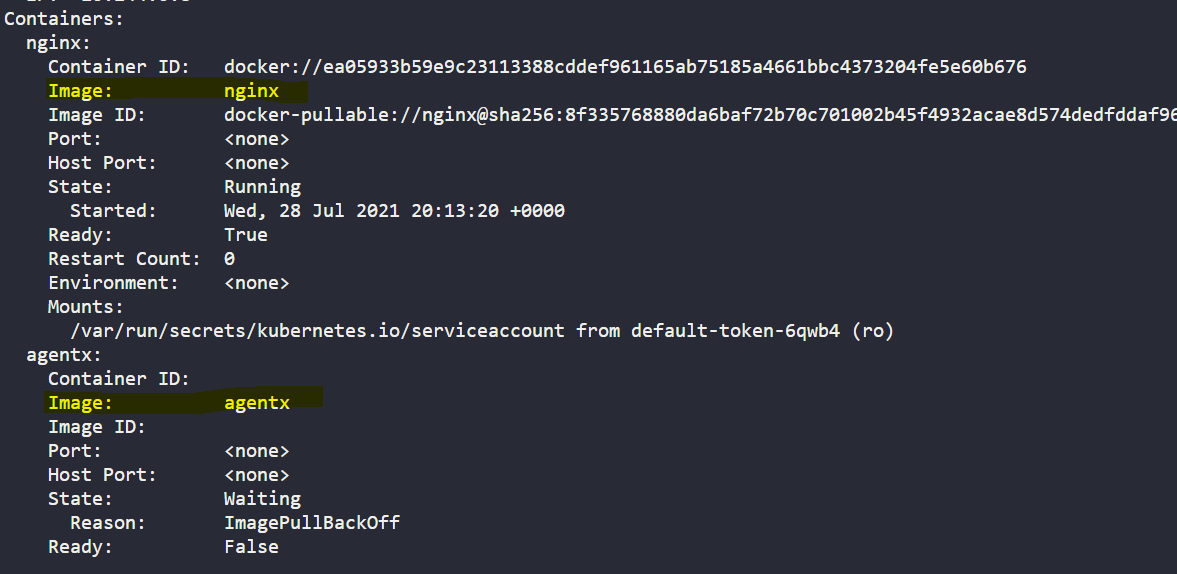
Note: We just created a new POD. Ignore the state of the POD for now.

Two containers are used



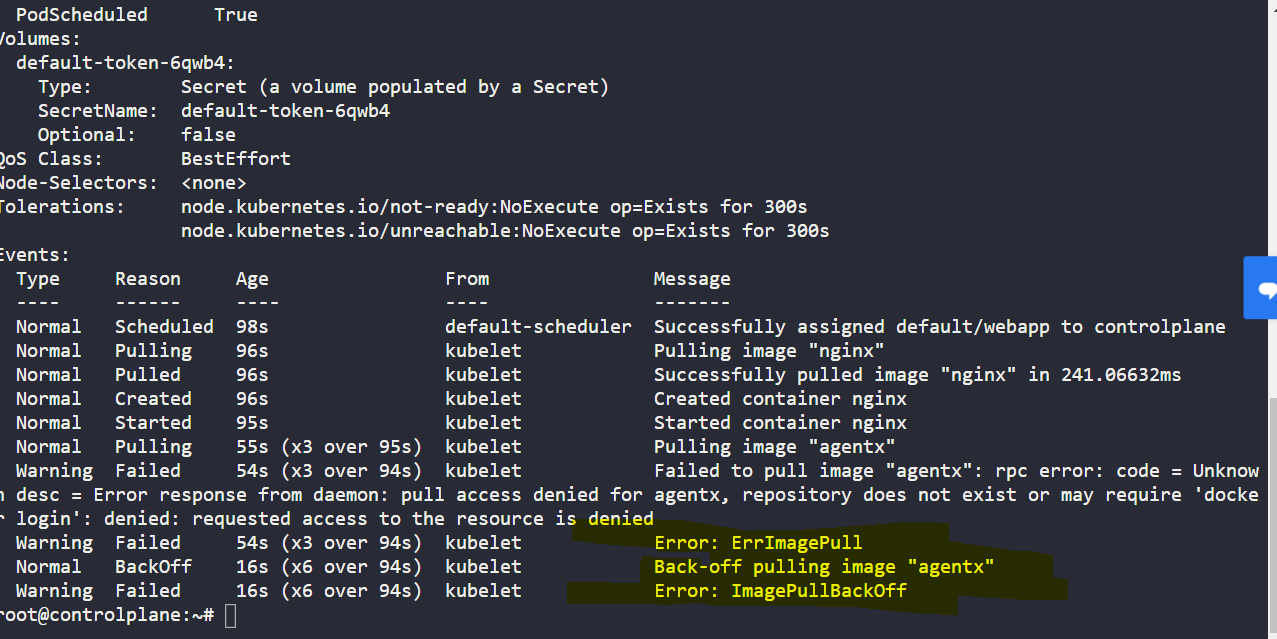
* How many images are used to create the Pod web app

Two image one is nginx and agentx.



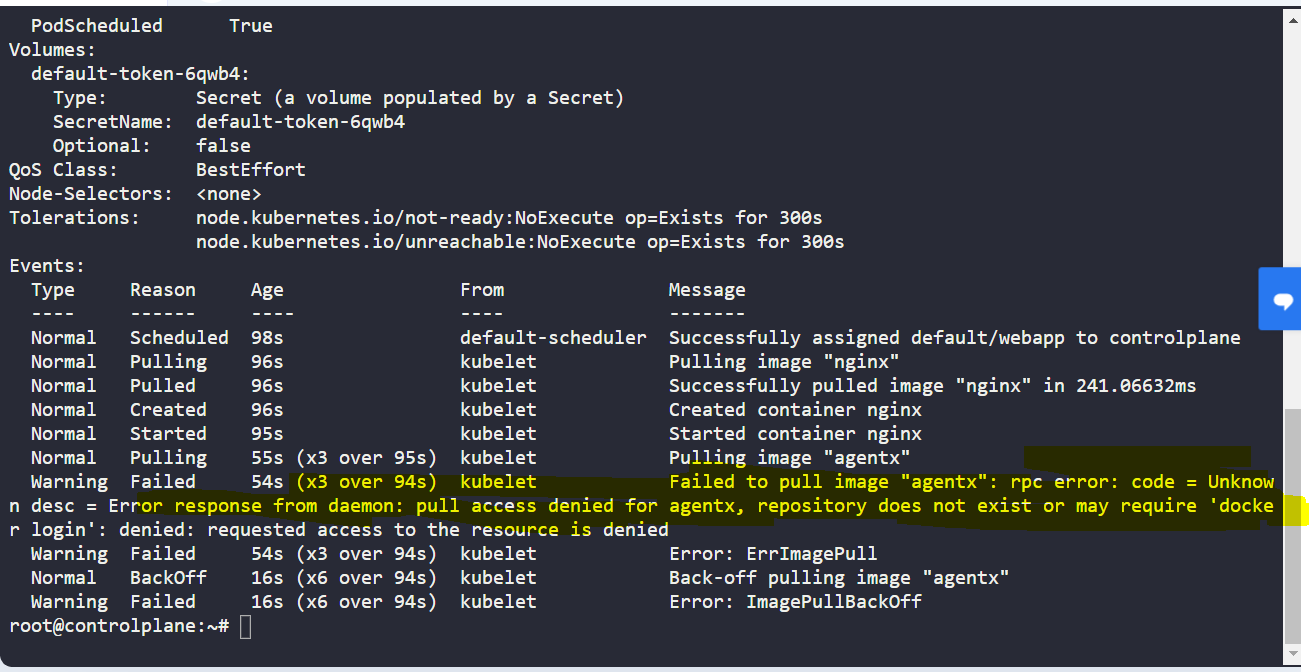
* What is the state of the container agentx in the pod webapp?

Error or waiting

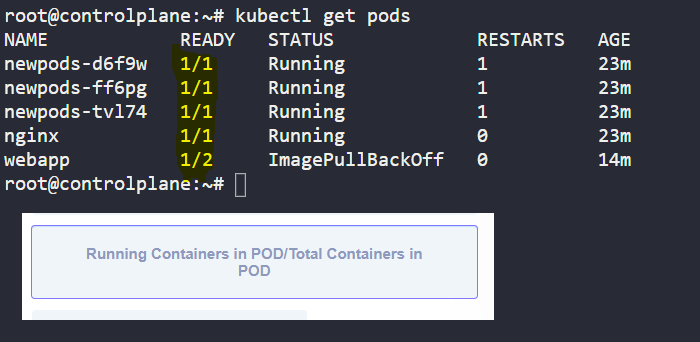


* What is the error for agentx

Dockerfile in the docker hub doesn’t exits.

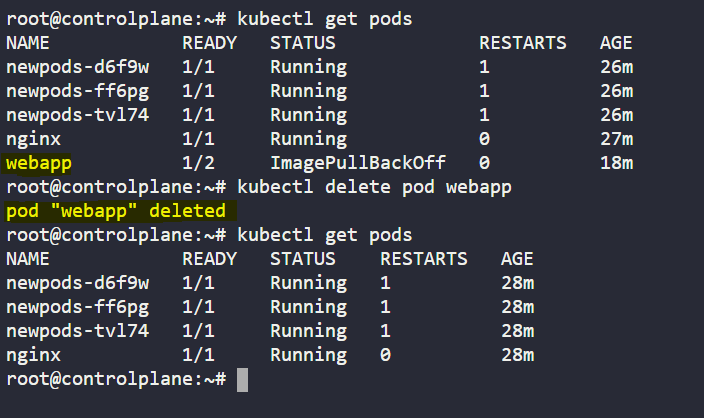


* What does the READY column in the output of the kubectl get pods command indicate?



* Delete the webapp Pod. Once deleted, wait for the pod to fully terminate.

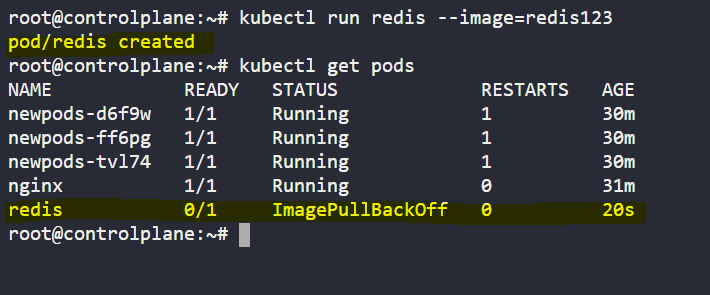
Run the command: kubectl delete pod webapp



* Create a new pod with the name redis and with the image redis123.

Use a pod-definition YAML file. And yes the image name is wrong!

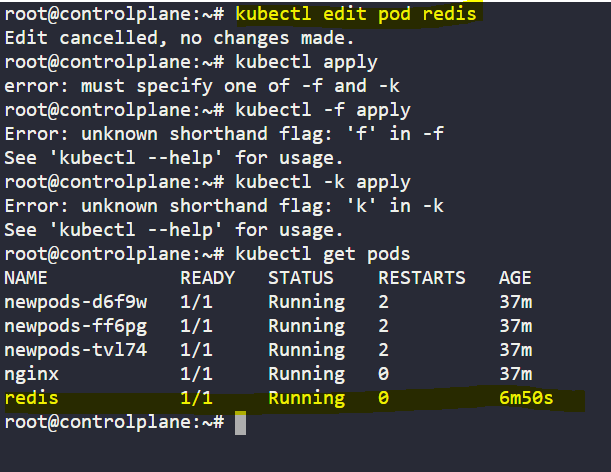
Run the command: kubectl run redis --image=redis123



* Now change the image on this pod to redis.

Once done, the pod should be in a running state.

Update the pod-definition file and use kubectl apply command or use kubectl edit pod redis command.



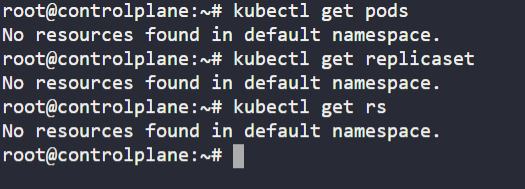
**Replicaset**

* How many ReplicaSets exist on the system?

In the current(default) namespace.

dRun the command: kubectl get replicaset or kubectl get rs and count the number of ReplicaSets

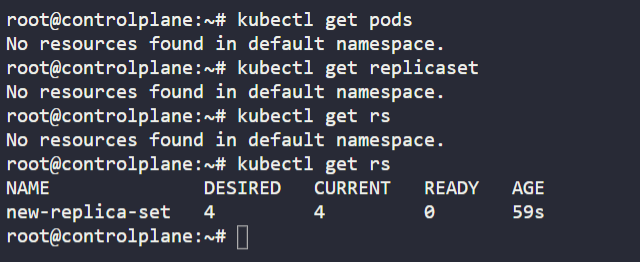
Answer is 0 replicaset



* How about now? How many ReplicaSets do you see?

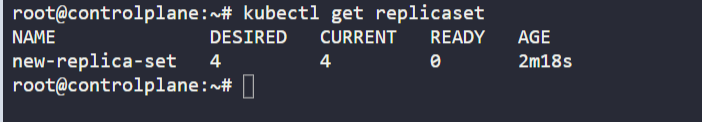
We just made a few changes!

Answer is 1 replicaset



* How many pods are resides in the replicaset

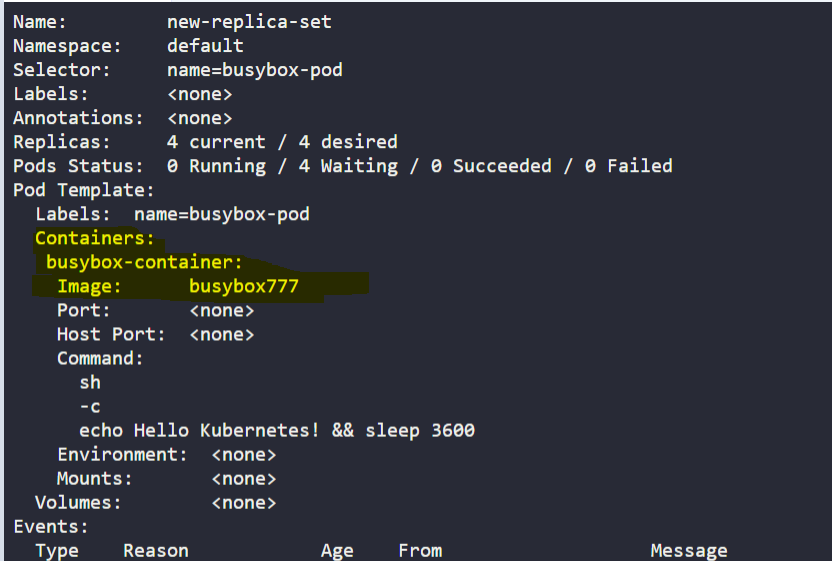
Run the command: kubectl get replicaset and look at the count under the Desired column.



* What is the image used to create the pods in the new-replica-set?

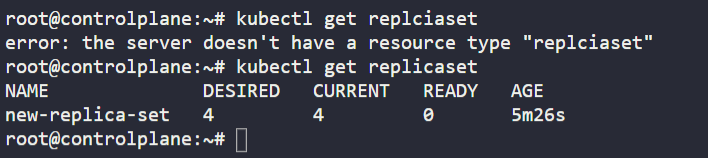
Run the command: kubectl describe replicaset and look under the Containers section.

Answer I s busybox777



* How many PODs are READY in the new-replica-set?

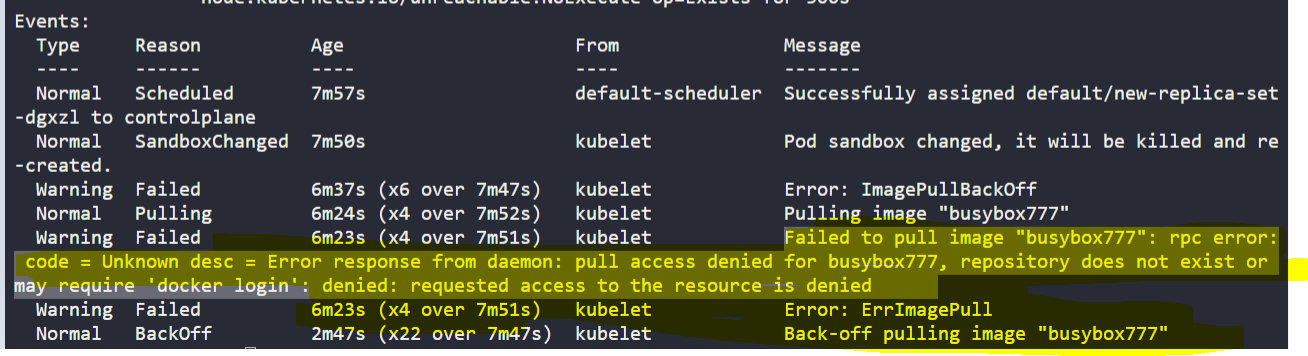
Run the command: kubectl get replicaset and look at the count under the Ready column.



* Why do you think the PODs are not ready?

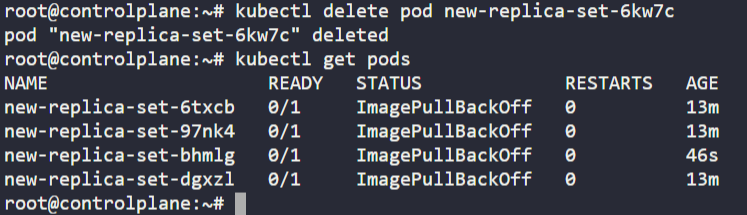
Run the command: kubectl describe pods and look at under the Events section.

* The image BUSYBOX777 doesn’t exits



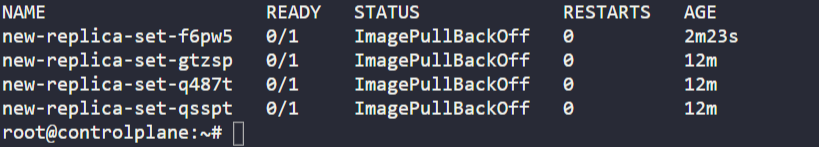
* Delete any one of the 4 PODs.

Run the command: kubectl delete pod <pod-name>



* How many PODs exist now?

Answer 4



* Why are there still 4 PODs, even after you deleted one?

Answer is ReplicaSet ensures that desired number of PODs always run

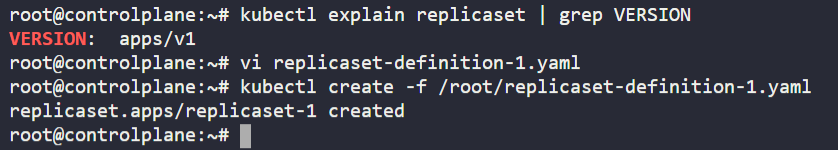
* Create a ReplicaSet using the replicaset-definition-1.yaml file located at /root/.

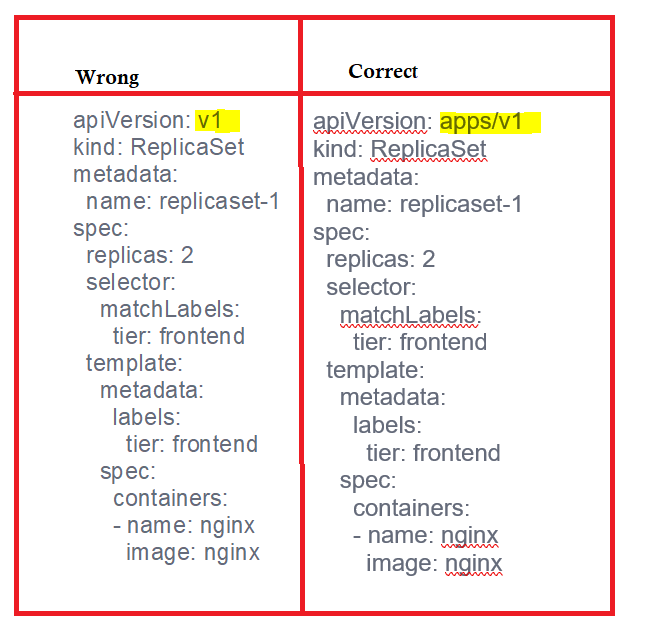
There is an issue with the file, so try to fix it.

The value for apiVersion is incorrect. Correct the apiVersion for ReplicaSet.

Run the command: kubectl explain replicaset | grep VERSION and correct the apiVersion for ReplicaSet. Answer is apps/v1

Then run the command: kubectl create -f /root/replicaset-definition-1.yaml





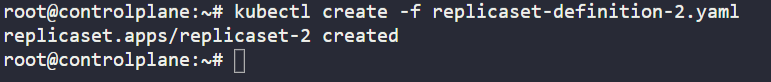
* Fix the issue in the replicaset-definition-2.yaml file and create a ReplicaSet using it.

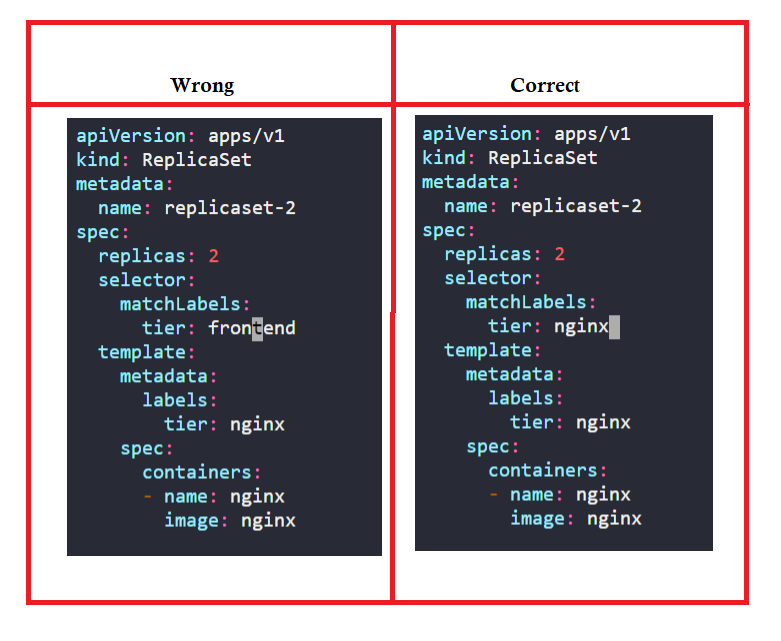
This file is located at /root/.

The values for labels on lines 9 and 13 should match.

Then run: kubectl apply -f /root/replicaset-definition-2.yaml

Or

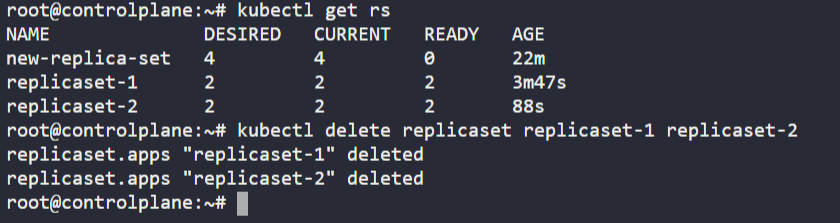


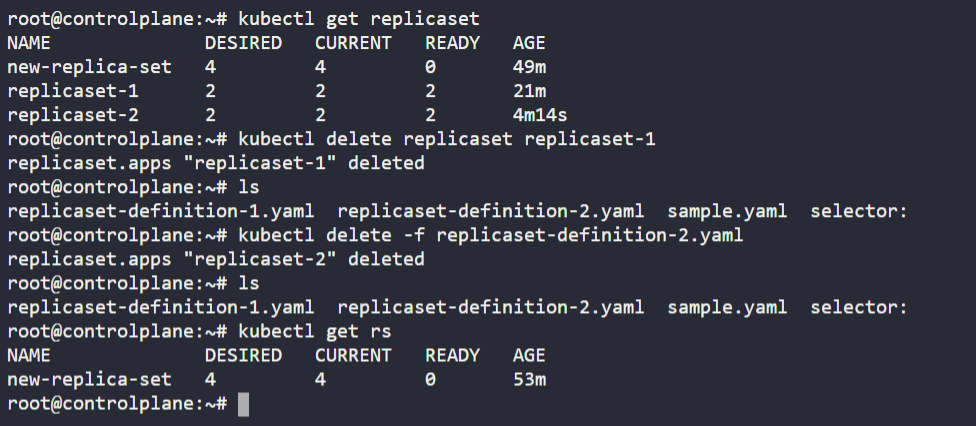


* Delete the two newly created ReplicaSets - replicaset-1 and replicaset-2

Use the command kubectl delete to delete the replicaset.

Run the command: kubectl delete replicaset <replicaset-name> or kubectl delete -f <file-name>.yaml





* Fix the original replica set new-replica-set to use the correct busybox image.

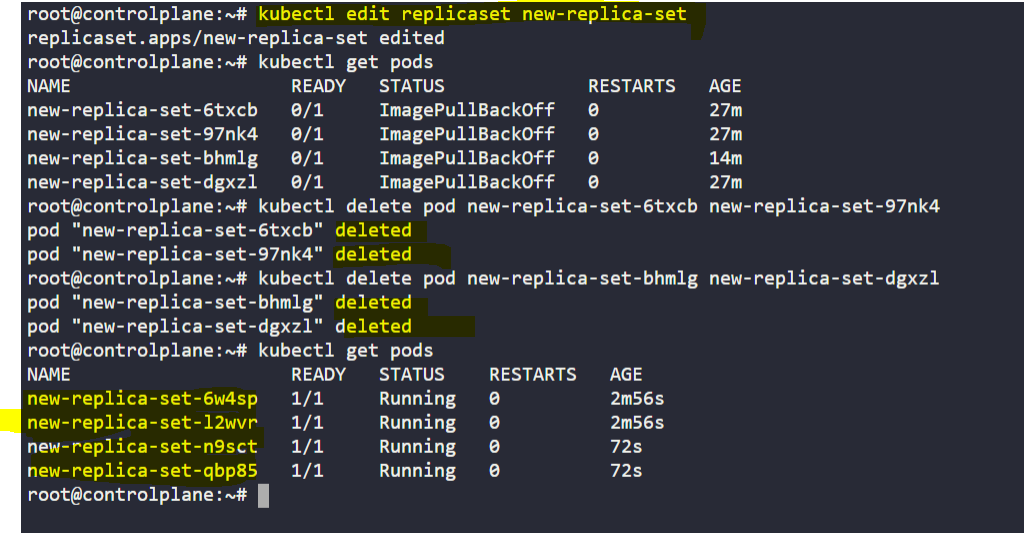
Either delete and recreate the ReplicaSet or Update the existing ReplicaSet and then delete all PODs, so new ones with the correct image will be created.

Use the command kubectl edit to edit the existing running replicaset.

Run the command: kubectl edit replicaset new-replica-set, modify the image name and then save the file.

Answer: Edit the image from BUSYBOX777 TO BUSYBOX

Delete the previous pods to get the new ones with the correct image.  
For this, run the command: kubectl delete po <pod-name>



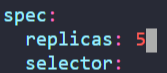
* Scale the ReplicaSet to 5 PODs.

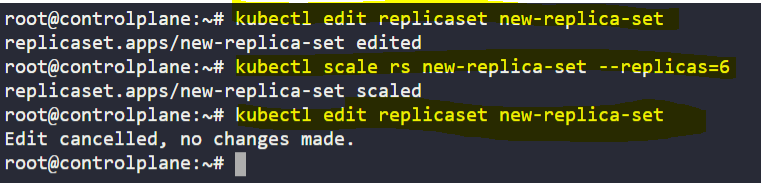
Use kubectl scale command or edit the replicaset using kubectl edit replicaset.

Make use of kubectl edit replicaset new-replica-set or the kubectl scale replicaset command.

Run the command: kubectl edit replicaset new-replica-set, modify the replicas and then save the file

OR run: kubectl scale rs new-replica-set --replicas=5 to scale up to 5 PODs.





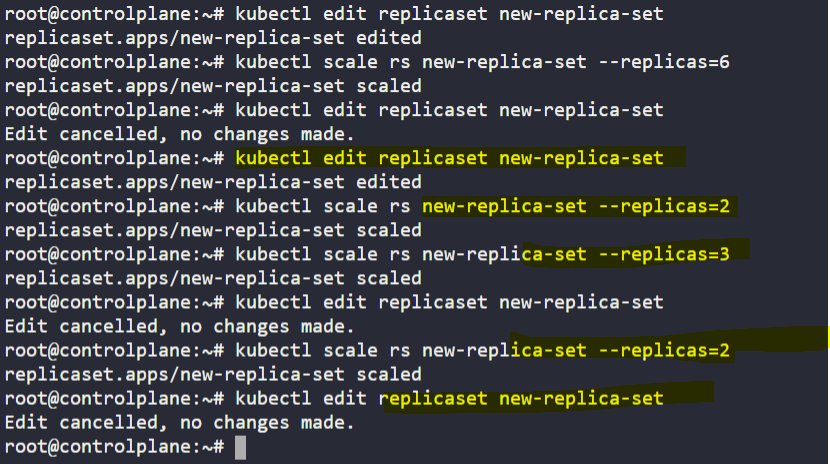
* Now scale the ReplicaSet down to 2 PODs.

Use the kubectl scale command or edit the replicaset using kubectl edit replicaset.

Make use of kubectl edit replicaset new-replica-set or the kubectl scale replicaset command.

Run the command: kubectl edit replicaset new-replica-set, modify the replicas and then save the file

OR run: kubectl scale rs new-replica-set --replicas=2 to scale down to 2 PODs.

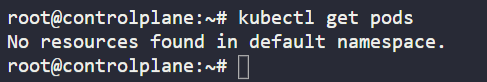


**Deployments**

* How many PODs exist on the system?

In the current(default) namespace.

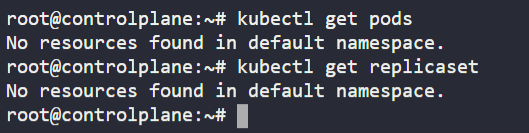
Answer is 0



* How many Replicasets exist on the system?

In the current(default) namespace.

Answer is 0



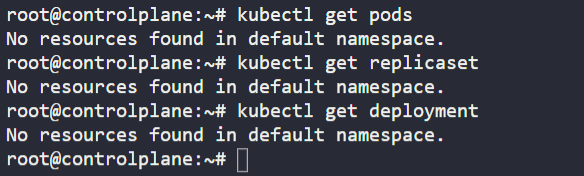
* How many Deployments exist on the system?

In the current(default) namespace.

Run the command: kubectl get deployment

or kubectl get deploy

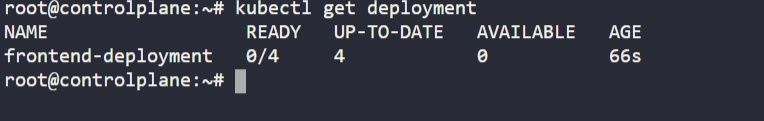
and count the number of Deployments.



* How many Deployments exist on the system now?

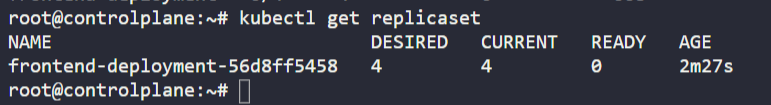
We just created a Deployment! Check again!

Answer is 1



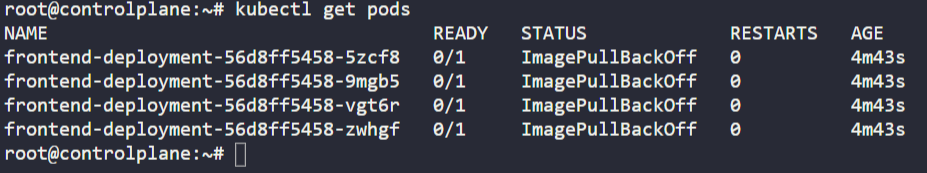
* How many ReplicaSets exist on the system now?

Run the command: kubectl get replicaset or replicasets



* How many PODs exist on the system now?

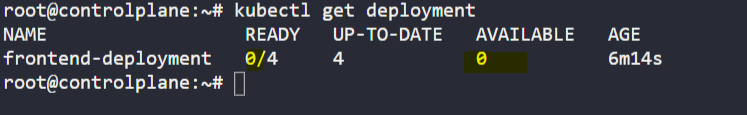
Run the command: kubectl get pods and count the number of PODs.



* Out of all the existing PODs, how many are ready?

Run the command: kubectl get deployment and count the number of PODs.

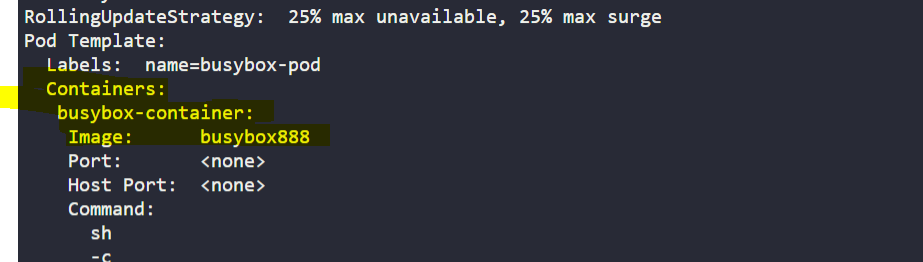
Answer is 0



* What is the image used to create the pods in the new deployment?

Use the command kubectl describe to know the detailed information of resources.

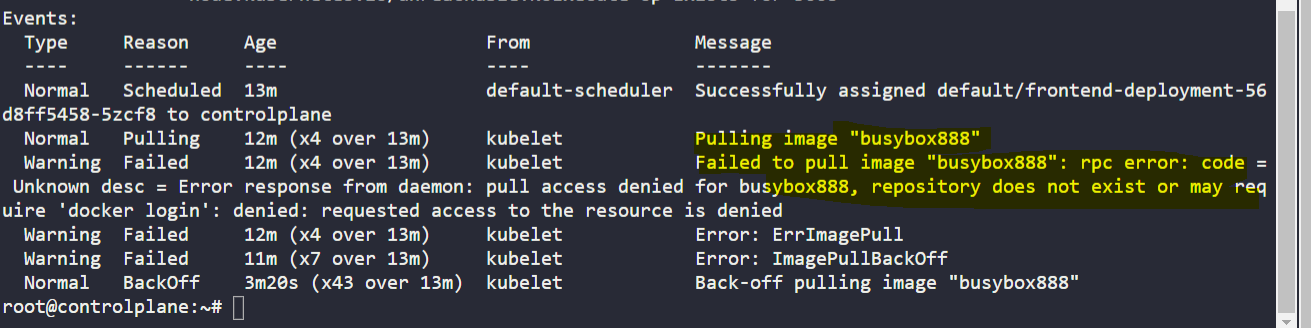
Run the command: kubectl describe deployment <deployment-name> and look under the Containers section.



* Why do you think the deployment is not ready?

Run the command: kubectl describe pods <pod-name> and look under the Events section.

Answer is The image BUSYBOX888 doesn't exist



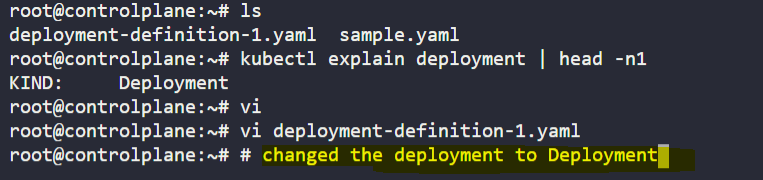
* Create a new Deployment using the deployment-definition-1.yaml file located at /root/.

There is an issue with the file, so try to fix it.

The value for kind is incorrect. It should be Deployment with a capital D.

Run the command: kubectl explain deployment | head -n1 and correct the value of kind.

Answer: Open the vi deployment-definition-1.yaml and change the deployment to Deployment



* Name: httpd-frontend;  
  Replicas: 3;  
  Image: httpd:2.4-alpine
* Name: httpd-frontend
* Replicas: 3
* Image: httpd:2.4-alpine

Use the command kubectl create deployment to create a deployment.

Solution manifest file deployment-definition-httpd.yaml:

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: httpd-frontend

spec:

replicas: 3

selector:

matchLabels:

name: httpd-frontend

template:

metadata:

labels:

name: httpd-frontend

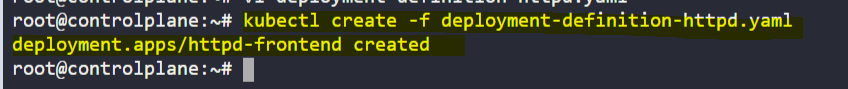
spec:

containers:

- name: httpd-frontend

image: httpd:2.4-alpine

Then run kubectl create -f deployment-definition-httpd.yaml



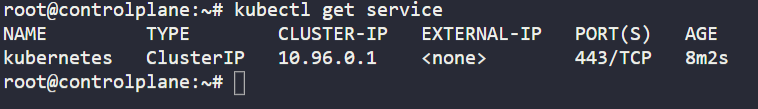
**Services**

* How many Services exist on the system?

in the current(default) namespace

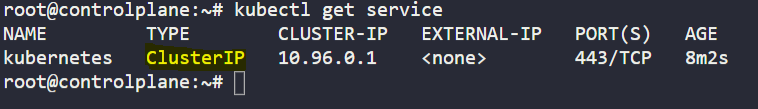
Run the command kubectl get service and count the number of services.

That is a default service created by Kubernetes at launch.



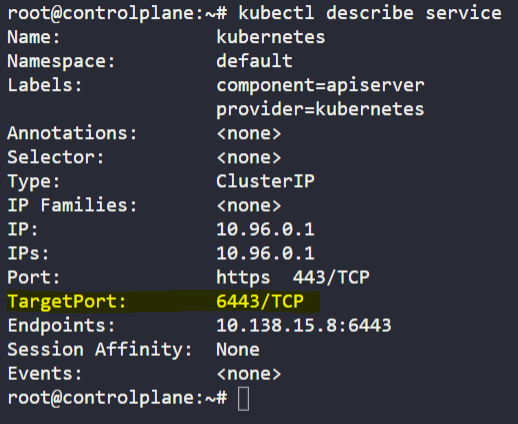
* What is the type of the default kubernetes service?

Answer : ClusterIP

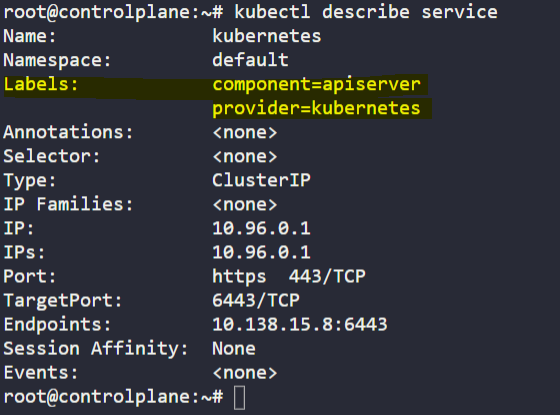


* What is the targetPort configured on the kubernetes service?

Run the command kubectl describe service and look at TargetPort.

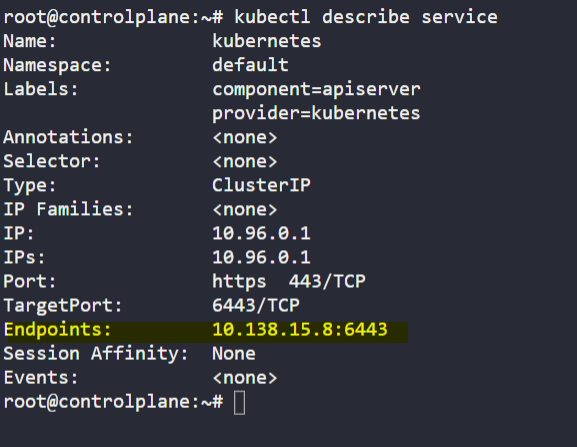


* How many labels are configured on the kubernetes service?
* Run the command kubectl describe service and look at Labels.



* How many Endpoints are attached on the kubernetes service?

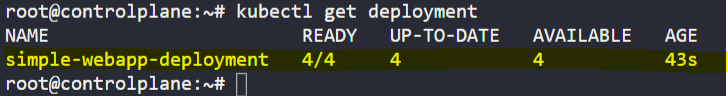
Run the command kubectl describe service and look at Endpoints.



* How many Deployments exist on the system now?

in the current(default) namespace

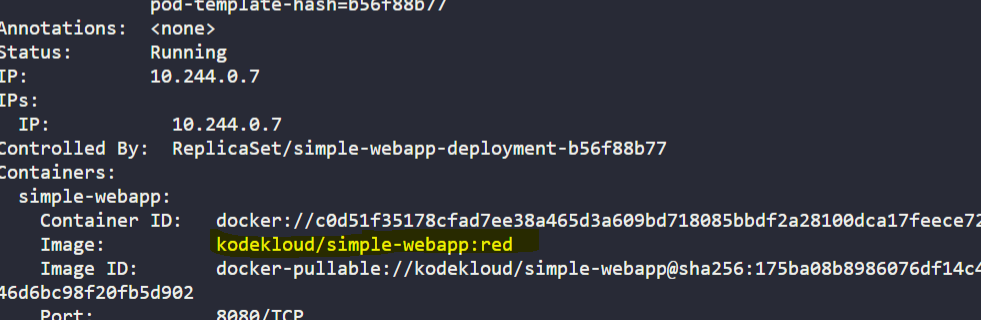
Answer is 1



* What is the image used to create the pods in the deployment?

Answer: kodekloud/simple-webapp:red

Run the command kubectl describe deployment and look under the containers section.



* Are you able to accesss the Web App UI?

Try to access the Web Application UI using the tab simple-webapp-ui above the terminal.

NO

* Create a new service to access the web application using the service-definition-1.yaml file

Name: webapp-service  
Type: NodePort  
targetPort: 8080  
port: 8080  
nodePort: 30080  
selector: simple-webapp

Update the given values in the service definition file and create the service.

Update the /root/service-definition-1.yaml file as follows:

---

apiVersion: v1

kind: Service

metadata:

name: webapp-service

spec:

type: NodePort

ports:

- targetPort: 8080

port: 8080

nodePort: 30080

selector:

name: simple-webapp

Then run: kubectl apply -f /root/service-definition-1.yaml

