Kubernate ultimate aim is to deploy our application in the form of containers on a set of machines that are configured as worker node in cluster

However, kubernate does not deploy containers directly on the worker nodes

Containers are encapsulated into a kubernates object know as pods

A pod is a single instance of an application

A pod is the smallest object that you can create in kubernate

If you want to increase the capacity because current node is not sufficient

Well than you can always deploy additional pods on a new node in the cluster

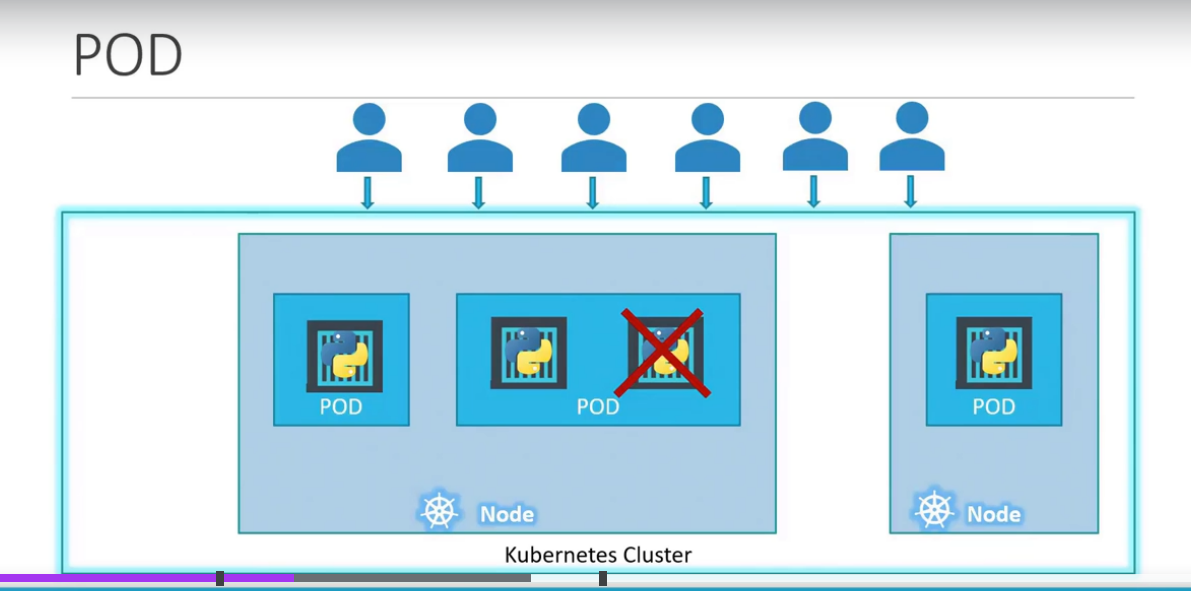
You will have a new node added to the cluster to expand the clusters physical capacity

Pods usually have 1 to 1 relationship with containers running your application to scale up

Scale up – you create new pod

Scale down – you delete existing pods

You don’t add additional containers to an existing pods to scale your application

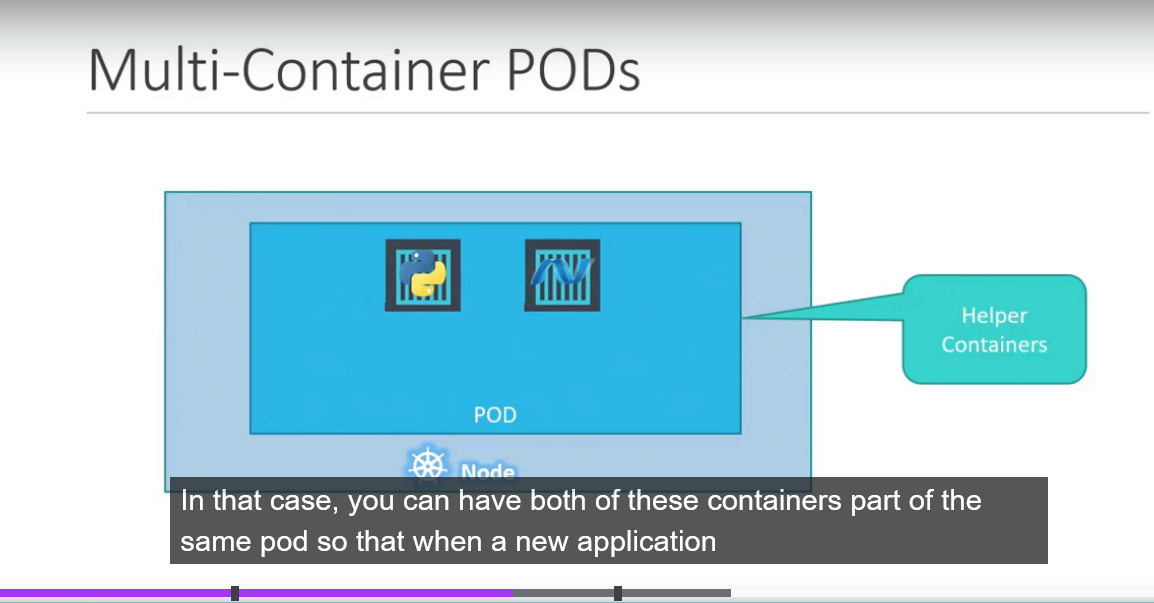


How do you restrict one pod for one container?

You cannot restrict

A single pod can have multiple containers except for the fact that that are usually not multiple containers of the same kind

Sometime you might have a scenario where you have a helper container that might be doing some kind of supporting task for our web application such as processing user entre data processing file uploaded by user



Both containers ca also communicate with each other directly by referring to each other as local host since they share the same network space

Plus they can easily share the same storage space as well

Kubernate definition file always contains 4 top level fields

ApiVersion: this is the version of kubernate api we are using to create the object

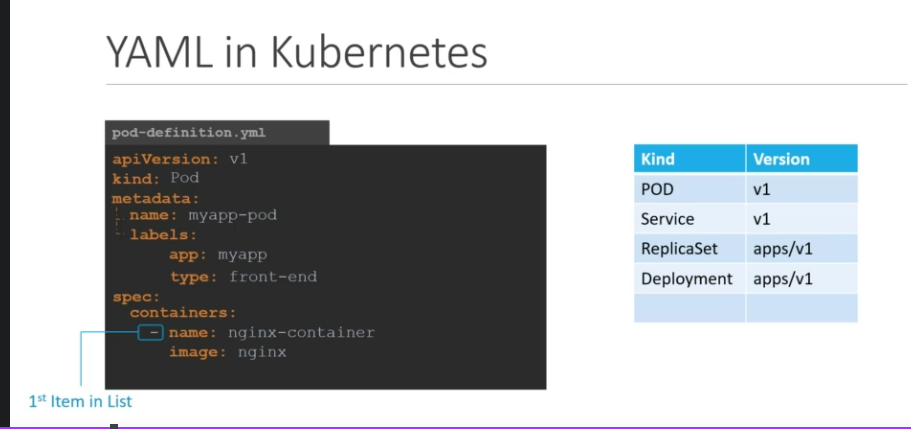
kind: kind refers to the type of object we are trying to create

metadata:

metadata is data about the object like it name labels etc

spec:

Depending on object we are going to create this is where we would provide additional information to kubernate pertaining to the object



Once file has created then run the command

#kubectl create -f pod-defination.yaml

Then kubernate is created pod

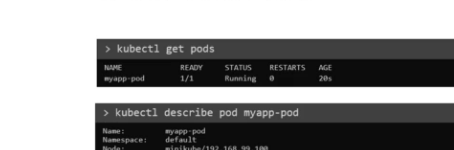
Once create the pod , how do you see it

To See list of pods available

#kubectl get pods

Detailed information about the pod

#kubectl describe pod name of the pod



Once file has been created then run that file

Kubectl apply –f pod.yaml

Lab:

How many pods exist on the system?

#kubectl get pods

2. create a new pods with nginx image ?

#kubectl run nginx –image=nginx

3. which nodes are these pods placed on ?

#kubectl get pods –o wide

3. how many containers are part of the pod in webapp

#kubectl get pods

Name READY

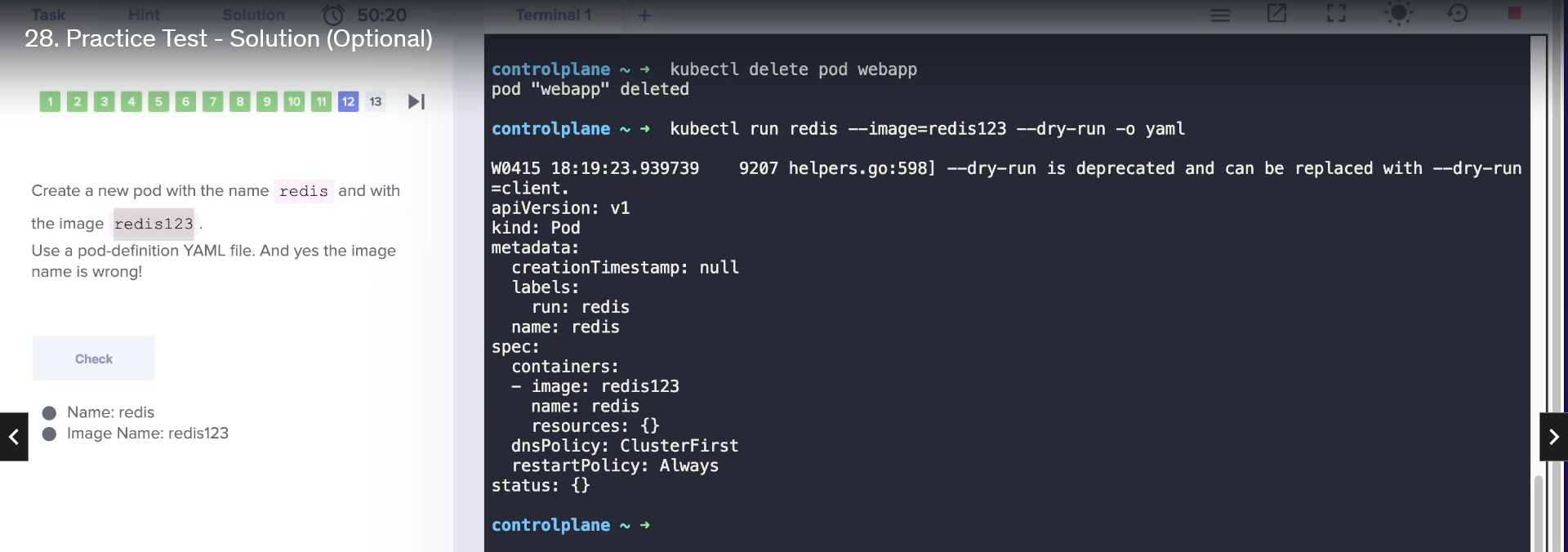
Webapp ½ ( 2 number of containers )(1/2 one container is running on pod out of 2 containers (running container in pods / total container in pods ) )

Or

Kubectl describe pod webapp

4. delete the webapp pod ?

#Kubectl delete pod webapp







REPLICAsETS :

Let's go back to our first scenario

where we had a single pod running our application.

What if for some reason our application crashes

and the pod fails?

Users will no longer be able to access our application.

To prevent users from losing access to our application,

we would like to have more than one instance or pod

running at the same time.

That way, if one fails

we still have our application running on the other one.

The Replication Controller helps us run multiple instances

of a single pod in the Kubernetes cluster,

thus providing high availability.

So does that mean you can't use a Replication Controller

if you plan to have a single pod?

No.

Even if you have a single pod,

the Replication Controller can help

by automatically bringing up a new pod

when the existing one fails.

Thus, the Replication Controller ensures

that the specified number of pods are running at all times

even if it's just one or 100.

Another reason we need Replication Controller

is to create multiple pods to share the load across them.

For example, in this simple scenario

we have a single pod serving a set of users.

When the number of users increase

we deploy additional pod to balance the load

across the two parts.

If the demand further increases

and if we were to run out of resources on the first node

we could deploy additional parts

across the other nodes in the cluster.

As you can see, the Replication Controller

spans across multiple nodes in the cluster.

It helps us balance the load across multiple pods

on different nodes as well as scale our application

when the demand increases.

It's important to note that there are two similar terms.

Replication Controller and Replica Set.

Both have the same purpose, but they're not the same.

Replication Controller is the older technology

that is being replaced by Replica Set.

Replica Set is the new recommended way

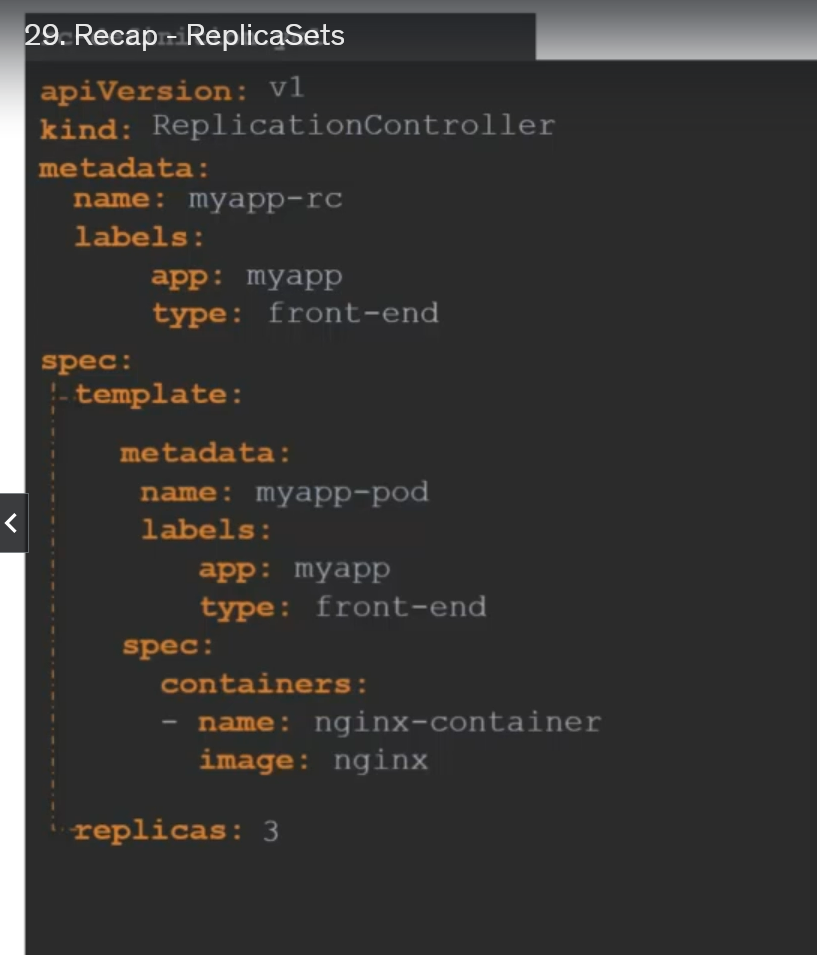
to set up replication.

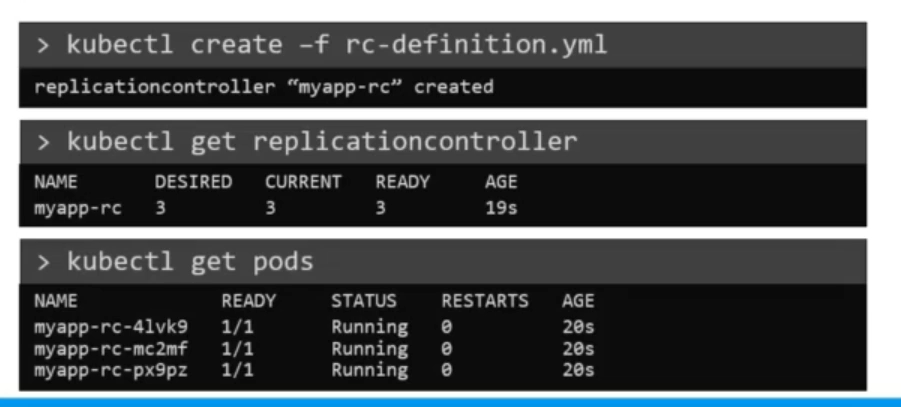
However, whatever we discussed in the previous few slides

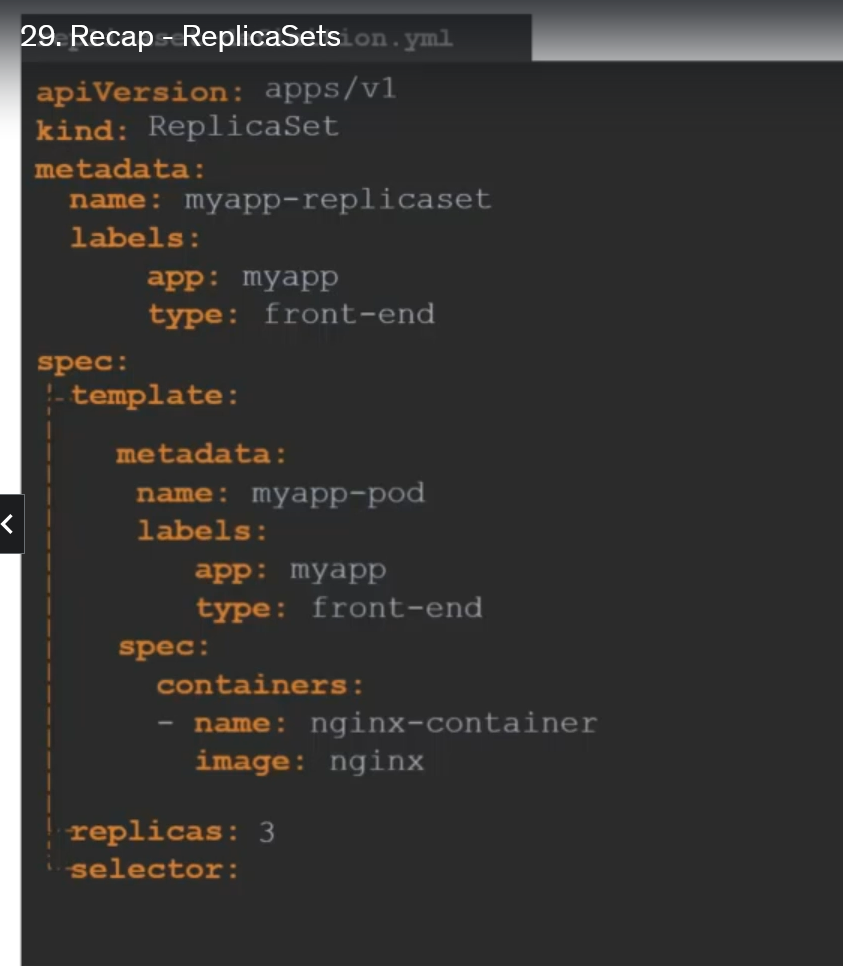
remain applicable to both these technologies.

There are minor differences in the way each works

HOW DO YOU CREATE THE REPLICASET







Replicaset and ReplicationController difference is “selector”

The selector section helps the Replica Set

identify what pods fall under it.

But why would you have to specify what pods fall under it

if you have provided the contents of the pod definition file

itself in the template?

It's because Replica Set can also manage pods

that were not created as part of the Replica Set creation.

Say for example, there were pods created

before the creation of the Replica Set

that match labels specified in the selector,

the Replica Set will also take those pods

into consideration when creating the replicas.

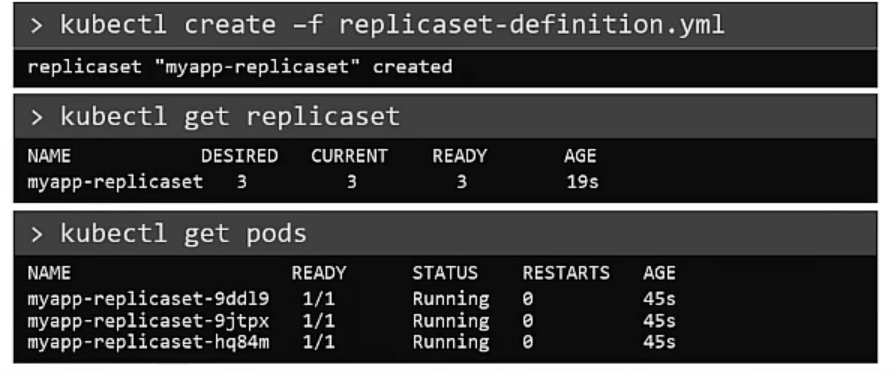
that the selector is one of the major differences

between Replication Controller and Replica Set.

The selector is not a required field

in case of a Replication Controller

but it is still available.



The Replica Set is in fact a process that monitors the pods.

Now, how does the Replica Set know what pods to monitor?

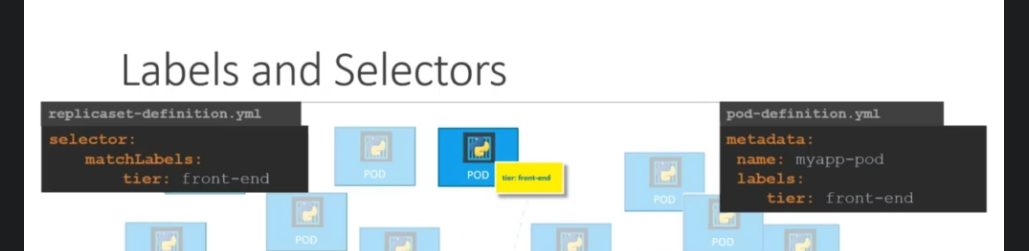
There could be hundreds of other pods in the cluster

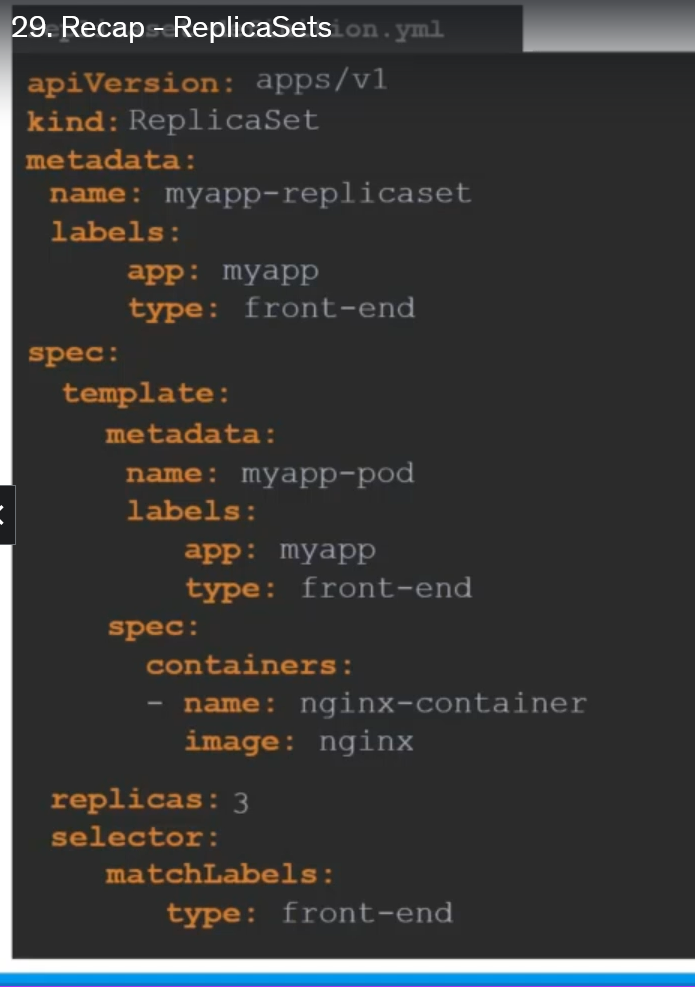
running different applications.

This is where labeling our pods

This is where labeling our pods

during creation comes in handy.





How to scale the rplicasets

1. First update the yaml file for replicas =6

Then

Kubectl replace –f replicaset-defination.yml

1. #kubectl scale - -replicas=6 –f replicaset-defination.yml
2. Kubectl scale - - replicas=6 replicaset myapp-replicaset

Type name

Commands of replicasset :

#kubectl create –f replicaset-defination.yaml

To display all replicas

#kubectl get replicaset

To delete replicaset , also delelts all underlying PODS

#kubectl delete replicaset myapp-replicaset

To increase pods

#kubectl replace –f replicaset-defination.yaml

#kubectl scale –replicas=6 –f replicaset-defination.yaml