SkillAssure

Kubernetes Concepts - ReplicaSet

SkillAssure

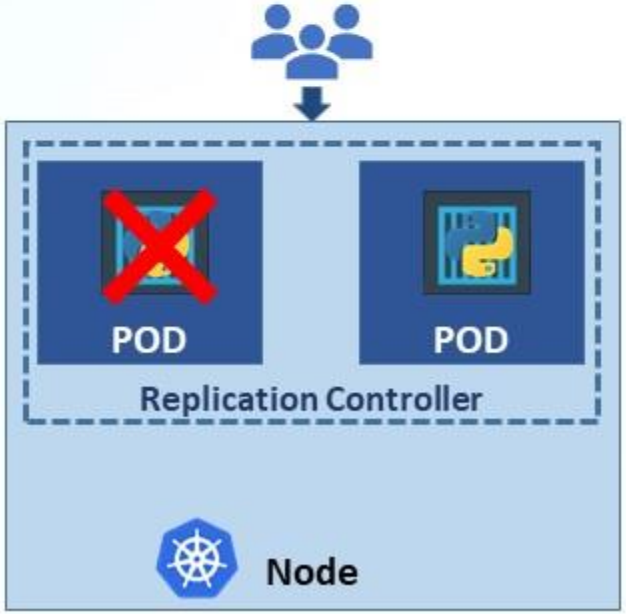
Controllers

- Controllers are the brains behind Kubernetes
- They are the processes that monitor Kubernetes objects and respond accordingly
- We will discuss about the replication controller
- What is a replica and why do we need a replication controller?

SkillAssure

Replication Controller

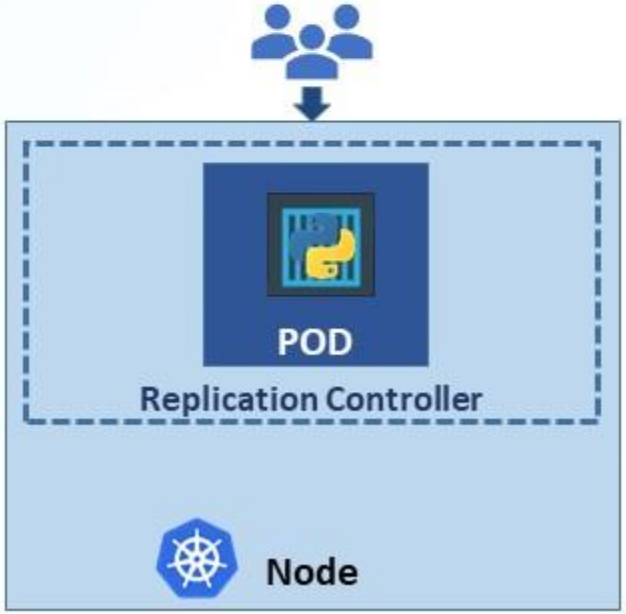
- Here we have a single pod running our application.
- If Pod fails, our application crashes and the users will no longer be able to access our application
- To prevent users from losing access to our application, we need to have more than one Pod running at the same time.
- That way if one failed, we 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



SkillAssure

Replication Controller

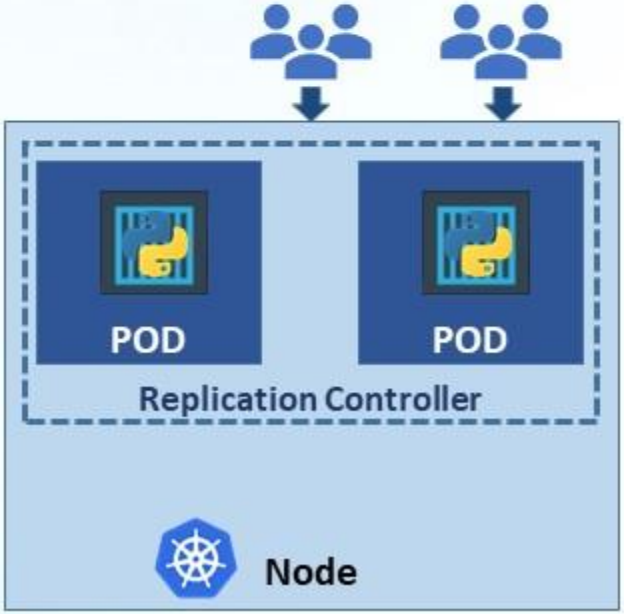
- Can you use a replication controller if you plan to have a single pod?
- **Yes**
- Even if you have a single pod the replication controller can help by automatically bringing up a new pod when the existing one fails.
- The replication controller ensures that the specified number of Pods are always running even if it's just 1 or 100



SkillAssure

Load Balancing & Scaling

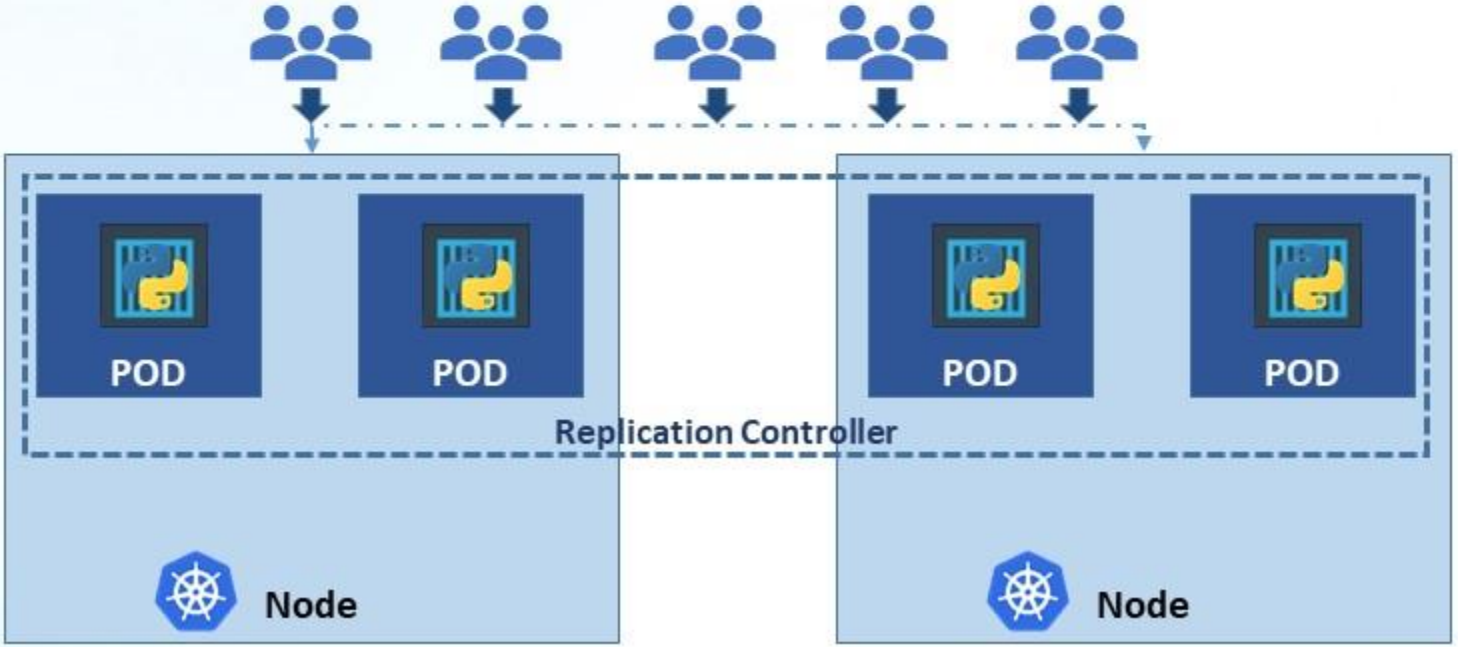
- Replication controller also creates multiple Pods to share the load across them
- Let's say we have a single pod serving a set of users
- When the number of users increase, we deploy additional pod to balance the load across the two pods.
- That way if one failed, we still have our application running on the other one.
- The replication controller helps us run multiple instances of a single Pod in the Kubernetes cluster thus providing high availability



SkillAssure

Load Balancing & Scaling

- If the demand further increases and if we were to run out of resources on the first node
- We could deploy additional pods across the other nodes in the cluster
- The replication controller spans across multiple nodes in the cluster
- It helps us balance the load across multiple Pods on different nodes
- It also scales our application, when the demand increases



SkillAssure

Replication Controller

Replica Set

SkillAssure

Creating Replication Controller

- We will start by creating a file rc-definition.yml
- Kubernetes definition file always contains 4 top level fields.
  - **apiVersion**
  - **Kind**
  - **metadata**
  - **Spec**

rc-definition.yml

```
apiVersion:
kind:
metadata:



spec:
```

SkillAssure

Creating Replication Controller

- We will start by creating a file rc-definition.yml
- Kubernetes definition file always contains 4 top level fields.
  - **apiVersion**
  - **Kind**
  - **metadata**
  - **Spec**

rc-definition.yml

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
   name: myapp-rc
   labels:
      app: myapp
      type: front-end
spec:
```

SkillAssure

Creating Replication Controller

- So far it has been very similar to how we created a pod.
- The next is the most crucial part of the definition file and that is the spec
- The spec section defines what's inside the object we are creating.
- The replication controller creates multiple instances of a Pod
- But what Pod?

rc-definition.yml

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: myapp-rc
  labels:
    app: myapp
    type: front-end
spec:
```

SkillAssure

Creating Replication Controller

- We create a template section under spec to provide a Pod template to be used by the replication controller to create replicas
- How do we define the Pod template?
- It's not that hard because we have already done that in the previous exercise
- We can reuse the contents of pod definition file to populate the template section

rc-definition.yml

```
apiVersion: v1
kind: ReplicationController
metadata:
   name: myapp-rc
   labels:
      app: myapp
      type: front-end
spec:
   template:
```

POD

SkillAssure

Creating Replication Controller

- Open Pod definition file and move all the contents of that file into the template section of the replication controller except for the first few lines which are API version and kind

rc-definition.yml

```
apiVersion: v1
kind: ReplicationController
metadata:
   name: myapp-rc
   labels:
      app: myapp
      type: front-end
spec:
   template:
```

POD

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
   name: myapp-pod
   labels:
      app: myapp
      type: front-end
spec:
   containers:
   - name: nginx-container
     image: nginx
```

SkillAssure

Creating Replication Controller

- Remember whatever we move must be under the template section meaning this should be intended to the right and have more spaces before them. Children of the template section

rc-definition.yml

```
apiVersion: v1
kind: ReplicationController
metadata:
   name: myapp-rc
   labels:
      app: myapp
      type: front-end
spec:
   template:
      metadata:
         name: myapp-pod
         labels:
            app: myapp
            type: front-end
      spec:
         containers:
         - name: nginx-container
           image: nginx
```

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
   name: myapp-pod
   labels:
      app: myapp
      type: front-end
spec:
   containers:
   - name: nginx-container
     image: nginx
```

SkillAssure

Creating Replication Controller

- Now we have two metadata sections and two spec sections, one is for the replication controller and another for the pod

rc-definition.yml

```
apiVersion: v1
kind: ReplicationController
metadata:                          <- Replication Controller
  name: myapp-rc
  labels:
    app: myapp
    type: front-end
spec:                              <- Replication Controller
  template:
    metadata:                      <- POD
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:                          <- POD
      containers:
      - name: nginx-container
        image: nginx
```

- We have nested 2 definition files together the replication controller being the parent and the Pod definition being the child.
- There is something still missing.
- We haven't mentioned how many replicas we need in the replication controller

SkillAssure

Creating Replication Controller

- Add another property to the spec called replicas and input the number of replicas you need under it

rc-definition.yml

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: myapp-rc
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  replicas: 3
```

- Remember that the template and replicas are direct children of spec section
- They must be on the same vertical line

SkillAssure

Creating Replication Controller

- Create replication controller

```
kubectl create –f rc-definition.yml
```

- View the list of created replication controllers

```
kubectl get replicationcontroller
```

- View the list of created Pods

```
kubectl get pods
```

- Note that all of them are starting with the name of the replication controller which is myapp-rc indicating that they are all created automatically by the replication controller

- Change number of replicas

```
kubectl replace -f rc-definition.yml
```

SkillAssure

Creating ReplicaSet

- It is very similar to replication controller. As usual first we have API version kind metadata and spec

replicaset-definition.yml

```
apiVersion: apps/v1
kind:
metadata:



spec:
```

- The API version is apps/v1 which is different from what we had before for application controller which was just v1
- If you get this wrong, you will get an error
- It would say no match for a kind replica set because the specified v1 API version has no support for ReplicaSet

```
error: unable to recognize "replicaset-
definition.yml": no matches for /, Kind=ReplicaSet
```

SkillAssure

Creating ReplicaSet

- The kind would be ReplicaSet

replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  replicas: 3
```

- We add name and labels in the metadata
- The specification section looks very similar to replication controller
- It has a template section where we provide Pod definition as before and we have number of replicas which is set to 3

SkillAssure

Creating ReplicaSet

- However there is one major difference between replication controller and replicaset

replicaset-definition.yml

```yaml
apiVersion: v1
kind: ReplicaSet
metadata:
  name: myapp-rc
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  replicas: 3
  selector:
  matchLabels:
    type: front-end
```

- ReplicaSet requires a selector definition
- The selector section helps the replica set identify what Pods fall under it
- Selector has to be written in the form of matchLabels as shown here.
- The match labels selector simply matches the labels specified under it to the labels on the pod.

SkillAssure

Creating ReplicaSet

- But why do we have to specify this, if we have provided the contents of the pod definition file itself in the template

replicaset-definition.yml

```yaml
apiVersion: v1
kind: ReplicaSet
metadata:
  name: myapp-rc
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  replicas: 3
  selector:
  matchLabels:
    type: front-end
```

- It's because replicaset can also manage pods that were not created as part of the replicaset creation.
- For example the pods created before the creation of the replicaset that match labels specified in the selector.
- The replica set will also take those pods into consideration when creating the replicas.

SkillAssure

Creating ReplicaSet

- Create ReplicaSet

```
kubectl create –f replicaset-definition.yml
```

- View the list of created ReplicaSet

```
kubectl get replicaset
```

- View the list of created Pods

```
kubectl get pods
```

SkillAssure

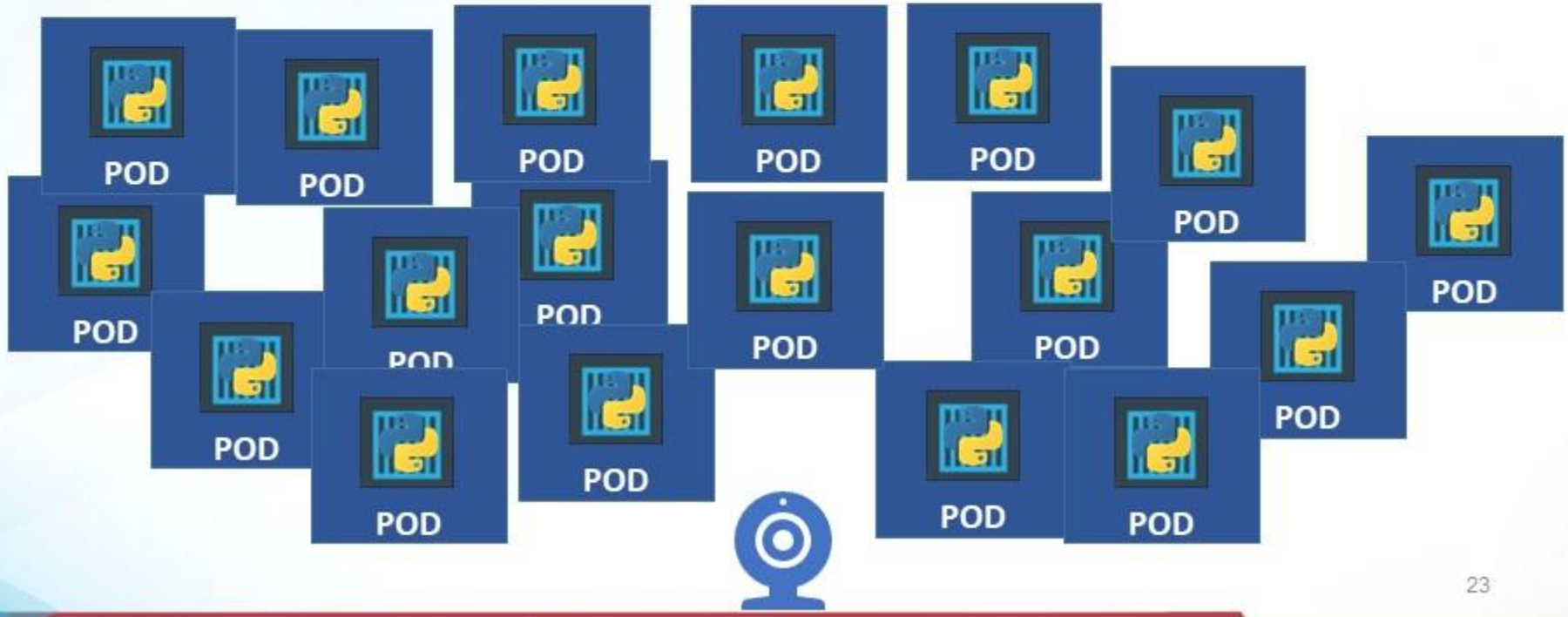
Labels and Selectors

- Why do we label our pods and objects in Kubernetes?
- Let's look at a simple scenario
  - We deployed 3 instances of our front-end web application as 3 Pods
  - Now we create a replicaset to ensure that we have three active Pods at any time.
- We can use replicaset to monitor existing Pods
  - If we have them already created as it is in this example, it will do nothing
  - In case they were not created the replicaset will create them for us
  - If any of the Pod fails, it also deploys new one.

SkillAssure

Labels and Selectors

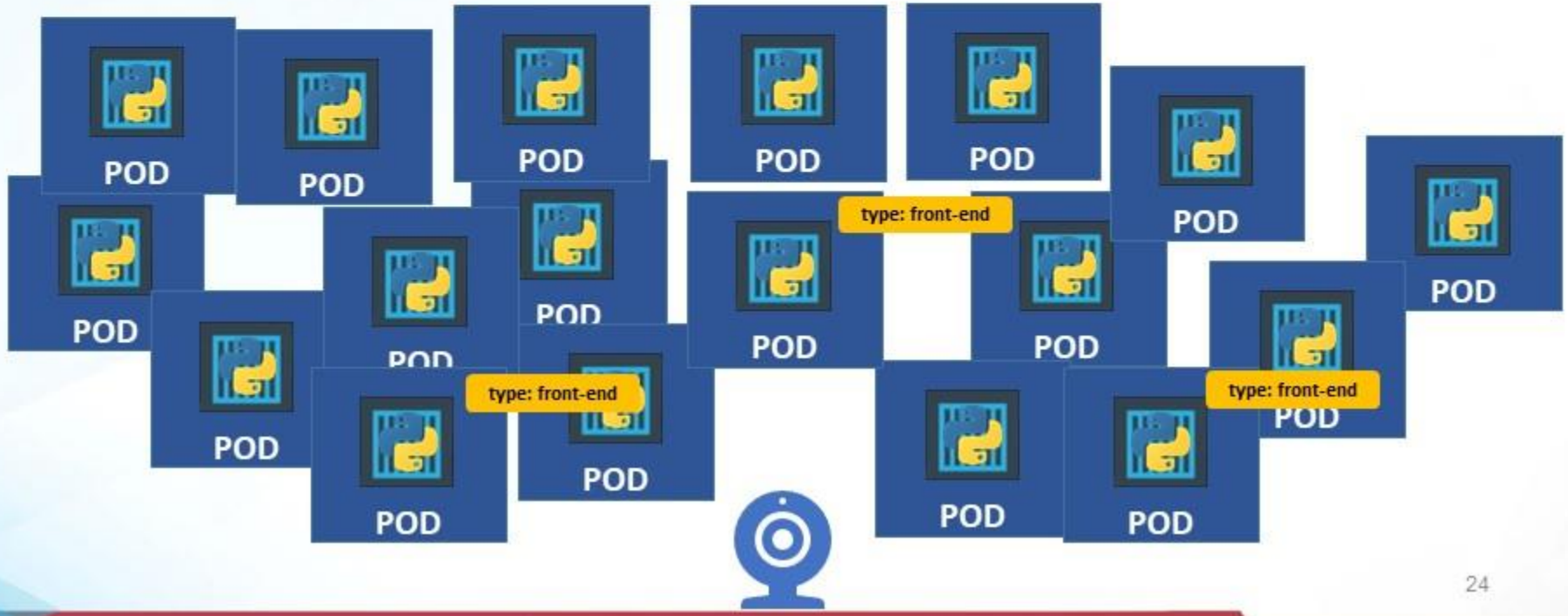
- How does the replica set know what pods to monitor?
- There could be hundreds of other pods in the cluster running different applications.
- This is where labelling our pods during creation comes in handy



SkillAssure

Labels and Selectors

- We could now provide these labels as a filter for replica set
- Under the selector section we use the matchLabels filter and provide the same label that we used while creating the pods.
- This way the replica set knows which pods to monitor.
- The same concept of labels and selectors is used in many other places throughout Kubernetes.



SkillAssure

ReplicaSet Question

Introduction:

Let's say we have three existing pods that were created already and we need to create a replica set to monitor the pods to ensure there are a minimum of three running at all times.

In the replica set specification section, we learnt that there are three sections: template, replicas and the selector. We need three replicas and we have updated our selector based on our discussion in the previous slide.

When the replication controller is created it is not going to deploy a new instance of pods as three of them with matching labels are already created.

Question: In that case do we really need to provide a template section in the replica set specification? Because we are not expecting the replica set to create a new pod

Yes, we do because in case one of the pods fail in the future the replicaset need to create a new one to maintain the desired number of pods and for the replica set to create a new pod, the template definition section is required.

SkillAssure

Scale

- Let's say we started with three replicas
- In the future we decided to scale to six.

replicaset-definition.yml

```yaml
apiVersion: v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  replicas: 3
  selector:
  matchLabels:
    type: front-end
```

How do we update our replica set to scale to six replicas?

SkillAssure

Scale

replicaset-definition.yml

```yaml
apiVersion: v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  replicas: 3
  selector:
  matchLabels:
    type: front-end
```

- **Option 1**
  - Update the number of replicas in the definition file to **6**

```
kubectl replace -f replicaset-definition.yml
```

- **Option 2**
  - Use Scale Command View the list of created ReplicaSet

```
kubectl scale --replicas=6 -f replicaset-definition.yml
```

```
kubectl scale --replicas=6 replicaset myapp-replicaset
```

- **Remember using the option 2 will not result in the number of replicas being updated automatically in the file.**

SkillAssure

ReplicaSet

- Describe ReplicaSet

  `kubectl describe replicaset`

- Delete Pod

  `kubectl delete <Pod_Name>`

- Try deleting a pod to test the replicaset

- Delete ReplicaSet

  `kubectl delete replicaset myapp-replicaset`

- Also deletes all underlying Pods

SkillAssure

Exercise

- Scale up pods to 6 using replicaset
- Scale down pods to 2 using replicaset

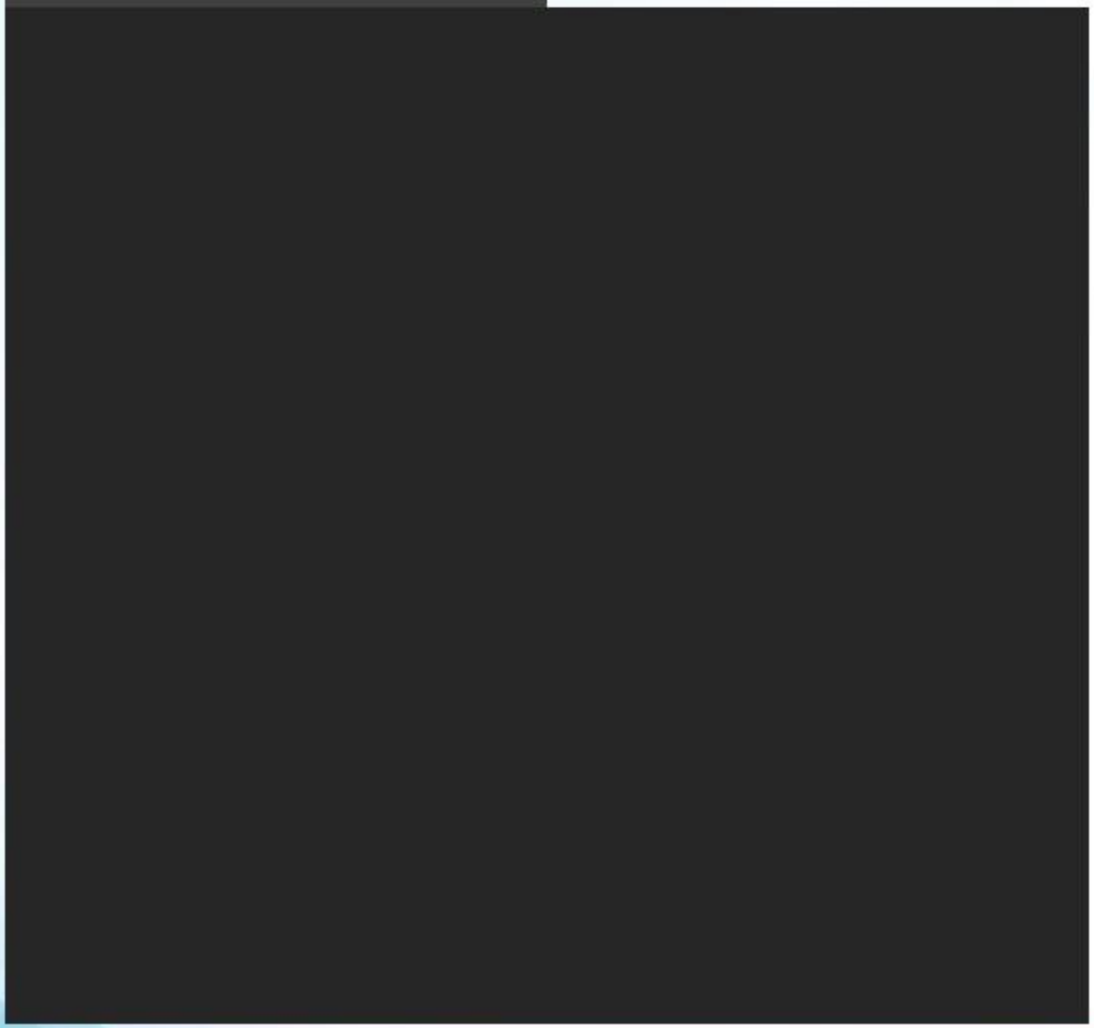
SkillAssure

ReplicaSet with YAML

SkillAssure

Exercise 16

replicaset-definition.yml



Introduction: Let's start simple! Given a blank replicaset-definition.yml file. We are only getting started with it.

Instruction: Add all the root level properties to it

SkillAssure

Exercise 16 - Solution

replicaset-definition.yml

```
apiVersion:
kind:
metadata:
spec:
```

Introduction: Let's start simple! Given a blank replicaset-definition.yml file. We are only getting started with it.

Instruction: Add all the root level properties to it

SkillAssure

Exercise 17

replicaset-definition.yml

```
apiVersion:
kind:
metadata:
spec:
```

Introduction: Let us now add values for Replicaset.

Instruction: Update values for apiversion and kind

SkillAssure

Exercise 17 - Solution

replicaset-definition.yml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
spec:
```

Introduction: Let us now add values for Replicaset.

Instruction: Update values for apiversion and kind

SkillAssure

Exercise 18

replicaset-definition.yml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
spec:
```

Introduction: Let us now add values for metadata.

Instruction: Name the Replicaset – frontend.And add labels app=>mywebsite and tier=> frontend

SkillAssure

Exercise 18 - Solution

replicaset-definition.yml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
   name: frontend
   labels:
      app: mywebsite
      tier: frontend
spec:
```

Introduction: Let us now add values for metadata.

Instruction: Name the Replicaset – frontend.And add labels app=>mywebsite and tier=> frontend

SkillAssure

Exercise 19

replicaset-definition.yml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
   name: frontend
   labels:
      app: mywebsite
      tier: frontend
spec:
```

Introduction: Let us now get to the specification.

Instruction: The spec solution for Replica has 3 fields: replicas, template and selector. Simply add these properties. Do not add values yet.

SkillAssure

Exercise 19 - Solution

replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: mywebsite
    tier: frontend
spec:
  template:
  replicas:
  selector:
```

Introduction: Let us now get to the specification.

Instruction: The spec solution for Replica has 3 fields: replicas, template and selector. Simply add these properties. Do not add values yet.

SkillAssure

Exercise 20

replicaset-definition.yml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
   name: frontend
   labels:
      app: mywebsite
      tier: frontend
spec:
   template:
   replicas:
   selector:
```

Introduction: Let us now get to the specification.

Instruction: Let us update the number of replicas to 4.

SkillAssure

Exercise 20 - Solution

replicaset-definition.yml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
   name: frontend
   labels:
      app: mywebsite
      tier: frontend
spec:
   template:
   replicas: 4
   selector:
```

Introduction: Let us now get to the specification.

Instruction: Let us update the number of replicas to 4.

SkillAssure

Exercise 21

replicaset-definition.yml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
   name: frontend
   labels:
      app: mywebsite
      tier: frontend
spec:
   template:
   replicas: 4
   selector:
```

Introduction: The template section expects a Pod definition. Luckily, we have the one we created in the previous set of exercises.

Instruction: Let us now **copy the contents of the pod-definition.yml** file, except for the apiVersion and kind and place it under the template section. Take extra care on moving the contents to the right so it falls under template

SkillAssure

Exercise 21 - Solution

replicaset-definition.yml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: mywebsite
    tier: frontend
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
    spec:
      containers:
      - name: nginx
        image: nginx
  replicas: 4
  selector:
```

Introduction: The template section expects a Pod definition. Luckily, we have the one we created in the previous set of exercises.

Instruction: Let us now **copy the contents of the pod-definition.yml** file, except for the apiVersion and kind and place it under the template section. Take extra care on moving the contents to the right so it falls under template

SkillAssure

Exercise 22

replicaset-definition.yml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: mywebsite
    tier: frontend
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
    spec:
      containers:
      - name: nginx
        image: nginx
  replicas: 4
  selector:
```

Introduction: Let us now link the pods to the ReplicaSet by updating selectors.

Instruction: Add a property "**matchLabels**" under selector and copy the labels defined in the pod – definition under it

SkillAssure

Exercise 22 - Solution

replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: mywebsite
    tier: frontend
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
    spec:
      containers:
      - name: nginx
        image: nginx
  replicas: 4
  selector:
    matchLabels:
      app: myapp
```

Introduction: Let us now link the pods to the ReplicaSet by updating selectors.

Instruction: Add a property "**matchLabels**" under selector and copy the labels defined in the pod – definition under it


Exercise

SkillAssure

Exercise

- Create replicaset definition files for example voting app
  - Vote
  - Result
  - Worker
  - Redis
  - Postgres

SkillAssure

Thank You