Kubernetes- Deepdive

Section 5 : Kubernetes deep-dive
PODs with YAML
Replication Controllers
    Replication controller | Replica Sets
    Hand-on for Replication controller | Replica sets
        Replication controller
        Replicaset
    Scale & Test cluster
        Scale up and down ReplicaSet
        Test Replication-controller & Replica-sets
        Deleting PODs created through Replicas
Deployment
    Hand-on Deployment
    Updates & Rollback
        Rollout & Versioning
        Deployment Strategy
        Rollout update
        Upgrades
        Rollback
Session 6 : Kubernetes on cloud

Section 5 : Kubernetes deep-dive

PODs with YAML

- **In this section will talk about creating the PODs using YAML based configuration file.**
- **We will be developing some YAML files which is going to create PODs for us in an automated fasion.**
- **Any file which you are going to create should be with .yml extension, For instance.**

```
E0-first-pod.yml
```

- **The kubernetes yml file should always have 4 top level fields (Mandatory).**

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

apiVersion :

- **This is the version of the Kubernetes api we are using to create the objects.**
- **Depending on what we are trying to create we must use the right apiVersion.**
- **For now since we are working on PODs we will set the apiVersion as v1**
- **Other possible values for this field are**

Kind	Version
POD	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

kind :

- **Kind referes the type of object we are trying to create, which in this case happens to be POD.**
- **Other possible values for this filed would be [Service, Replicaset & Deployment]**

metadata :

- **Metadata is going to contain the details about the object [POD] you are going to create, Like its Name, Lables etc about your POD..**
- **Metadata mostly will be declared in form of Dictionary.**
- **Whatever you are going define under this filed is considered as children of Metadata.**

spec :

- **Specification is where you are going to define the details about your container, Like name of your container, Image, port and volume details.**

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

Now your file is created, how to execute it ?

```
kubectl create -f E0-first-pod.yml
```

- **-f used to pass file**

```
kubectl apply -f E0-first-pod.yml
```

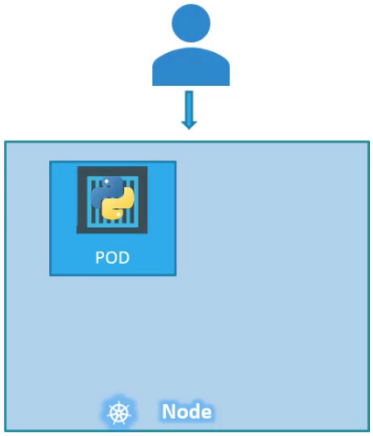
Replication Controllers

- **Controllers are brain behind the Kubernetes.**

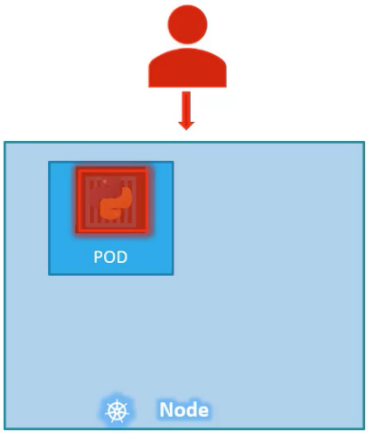
- They are the processes that monitor kubernetes objects and respond accordingly.
- In this section we are going to discuss specifically about Replication controller.

What is Replication controller ?

- Let's go back to our first scenario where we had single pod running our application.

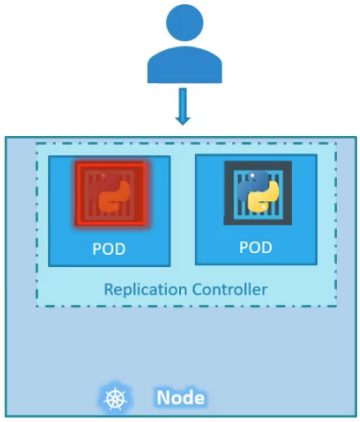


- What if for some reason our application got crashes & the Pod fails.? Users will no longer able to access the application hosted in Pod.

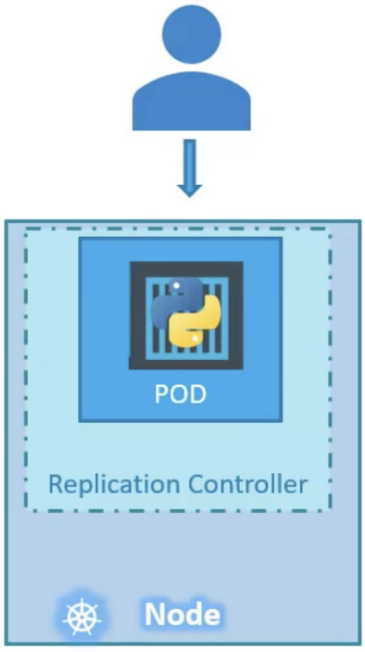


- To prevent users loosing access to application we would like to have more than one instance or Pods running at same time.
- In this way we can achieve the high availability of our application. If one Pod fails application will be still accessible from other Pod.

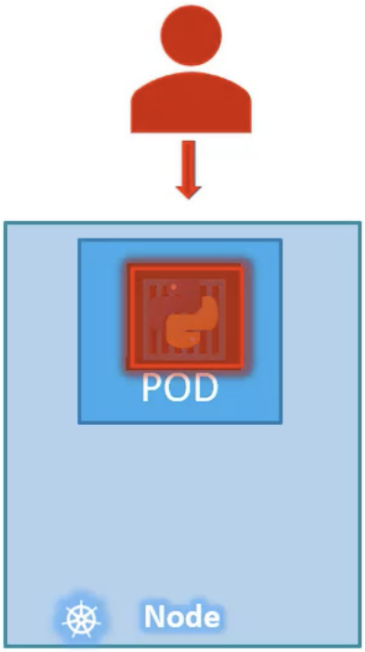
- **Replication controller helps us to run multiple instances of a single Pod as in cluster form to achieve High availability.**



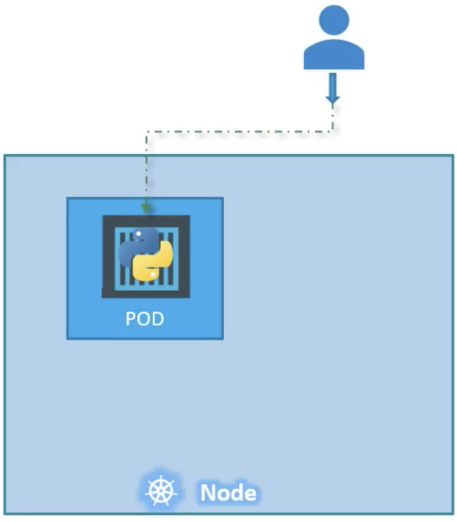
- **But does that mean you can't use replication factor if you plan to have single Pod ?**

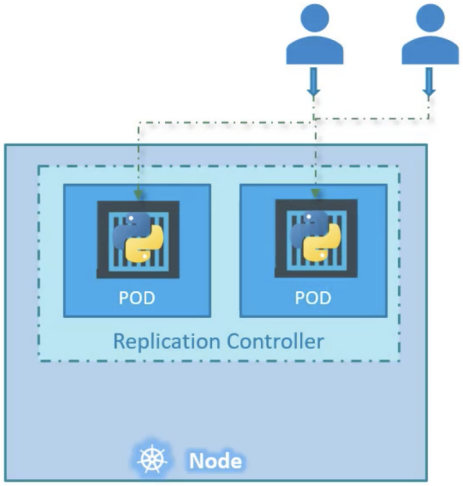


  - **No "Even if you have single POD the replication factor can help by automatically bringing up the new POD when the existing one fails"**

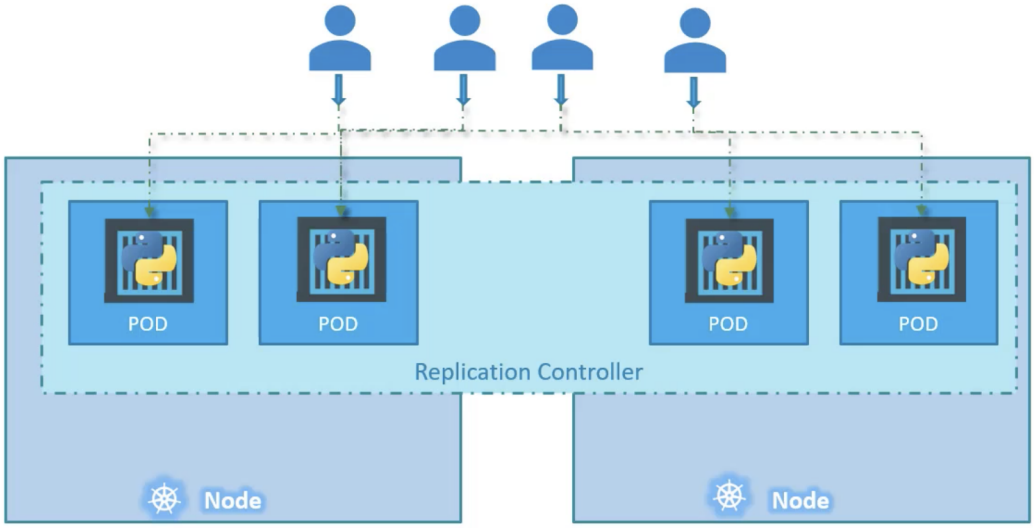


- **Thus the replication controller ensures the specified number of PODs are running all time, Even if it is 1 or 100 PODs.**
- **Another reason to have replication controller is to create multiple PODs to share the Loads across them.**
- **Simple scenario which we discussed earlier, when the number of people accessing your application things should be fine.**
- **When the load increases your replication controller will automatically create required PODs to serve your customers.**





- **If the demand further increases and if we are running out of resources in first node we can deploy additional PODs across the other Nodes in the cluster.**
- **As you can see replication controller spans across multiple nodes in the cluster.**
- **This approach helps us to balance the load across multiple PODs on different nodes and scale our application when the demand increases.**



Replication controller | Replica Sets

- **Its important to understand that there are 2 similar terms Replication controller | Replica Set**
- **Both have the same purpose but they are not same.**

- **Replication controller is the older technology that is been replaced by Replica Set.**
- **Replica set the new recommended way to setup replication.**
- **However whatever we have discussed in previous slides are applicable for both this functionalities.**
- **There are minor differences in the way how it works we shall look at those further.**

Hand-on for Replication controller | Replica sets

Replication controller

- **Lets create an yaml file here for this example**

```
E1-rc-definition.yml
```

- **This file will have the default four fields as-usual [apiVersion, Kind, Metadata & spec]**
- **Under spec we are going to provide POD details under template section.**

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

Execute file

```
kubectl apply -f E1-rcontroller.yml
```

To view the list of created relication controller

```
kubectl get replicationcontroller
```

To get more details about your replication controller

```
kubectl describe replicationcontroller myapp-rc
```

To see the PODs

```
kubectl get pods
```

Replicaset

Let's talk about Replicaset now

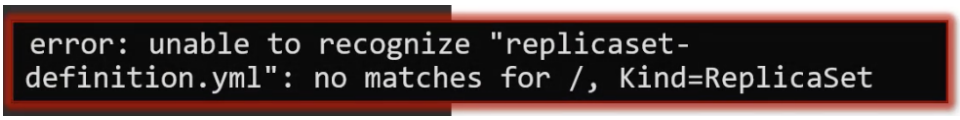
- **The code is going to look similar here but with some changes and additional parameters.**
- **Let's create new file here.**

```
E2-replicaset.yml
```

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
    app: myapp
    type: front-end
spec:
 template:
   metadata:
    name: myapp-prod
    labels:
        app: myapp
        type: front-end
   spec:
```

```
    containers:
      - name: nginx-container
        image: nginx
  replicas: 3
  selector:
     matchLabels:
       type: front-end
```

- **Some changes we did here is..**
- **Change of apiVersion [Ensure if you are making mistake in apiVersion you might likely to get error like this]**



- **Adding additional section called "selector" under replicas.**
- **The "selector" section helps replicasset to identify what pods falls under it.**
- **But why would you have to specify what PODs falls under it ? As you can see we have already provided the contents of the PODs in template section already.**
- **The reason for doing it is because "replicaset can also manage PODs that were not created as part of the replicaset creation"**
- **For Example : Lets assume there were some PODs created before the creation of Replicasets creation.**
- **Now the Replicaset will also consider those PODs while creation if incase the label specified in the selector section matches that independent POD.**
- **The major difference between ReplicationController & ReplicaSet is this "selector section"**

Execute file

```
kubectl create -f E2-replicaset.yml
```

To view the list of created relication controller

```
kubectl get replicaset
```

To get more details about your replicaset

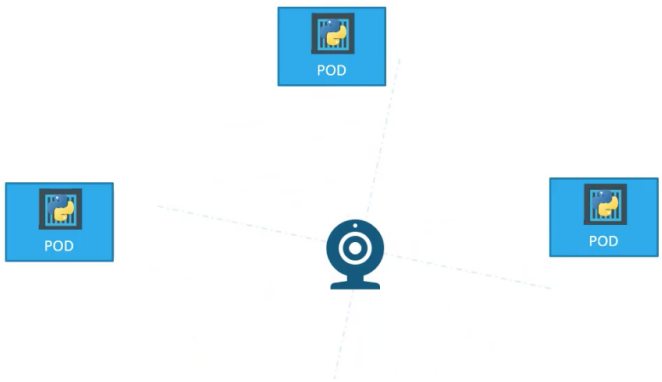
```
kubectl describe replicaset myapp-replicaset
```

To see the PODs

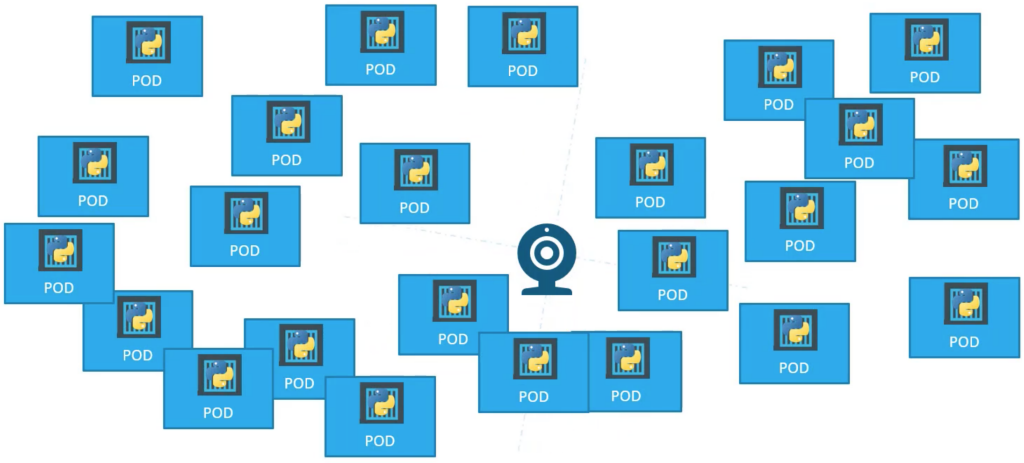
```
kubectl get pods
```

Labels and Selectors

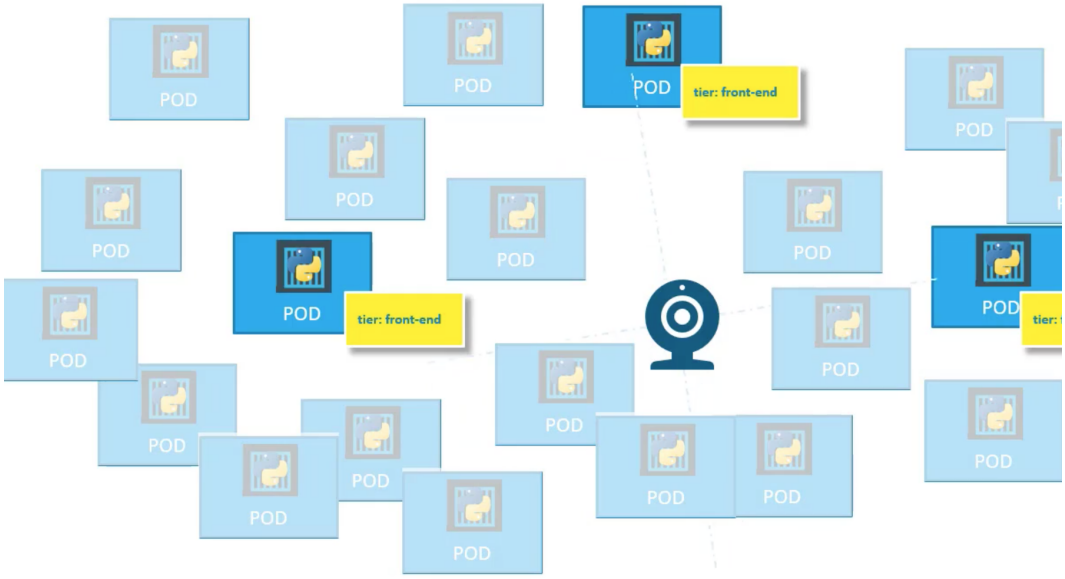
- **Whats the deal with Lables and Selectors ? and why do we label PODs and objects in kubernetes.?**
- **Let's look into a simple scenario.**
  - **Just assume we have 3 PODs with instances configured in it. We would like to create Replication controller or Replicaset to ensure that we have 3 active PODs at any time and all the time.**



  - **As you already know the role of Replication Controller or ReplicaSet is to monitor the PODs and recreate if any of those fails.**
  - **Now how does the replicaset know which PODs to be monitored ? because we might have many PODs running in your cluster.**



- **This is where lables is going to be handy to filter the PODs which need to be monitored by ReplicaSet.**
- **In this way Labels might be handy in many places across Kubernetes cluster.**



Scale & Test cluster

Scale up and down ReplicaSet

- **Let's see how to scale the ReplicaSet.**
- **For now we started with 3 Replicas, what if we want to scale it in future.**
- **Lets assume we need to scale our Replicas from 3 to 6.**
- **There are multiple ways to do this**

Step 1 : Update the code with ReplicaSet

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
    app: myapp
    type: front-end
spec:
 template:
   metadata:
    name: myapp-prod
    labels:
        app: myapp
        type: front-end
   spec:
    containers:
      - name: nginx-container
        image: nginx
 replicas: 6
 selector:
    matchLabels:
      type: front-end
```

Then run the code like below

```
kubectl replace -f E2-replicaset.yml
```

Step 2 : Update the code with scale command

```
kubectl scale --replicas=6 -f E2-replicaset.yml
```

Similarly to scale down the cluster

```
kubectl scale --replicas=2 -f E2-replicaset.yml
```

Step 3 : Changing the replicas in running configuration

- **We can also change this configuration at running config file of replicas.**

- **When we open this file we can see the running configuration of replicaset.**
- **Note that this is not the actual file which we created, This file is created by Kubernetes in memory.**
- **Kubernetes allows us to edit the objects from this file as well.**
- **There might be lot of additional fields available which we have not specified in our YAML file and there are generated by Kubernetes itslef.**

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

Test Replication-controller & Replica-sets

Now lets try deleting the PODs and just see if its getting created.

```
kubectl delete pods myapp-replicaset-<pod-name>
```

Eventually POD should be recreated now.

Deleting PODs created through Replicas

How to delete the machines created through Replicas & Replication-controller.

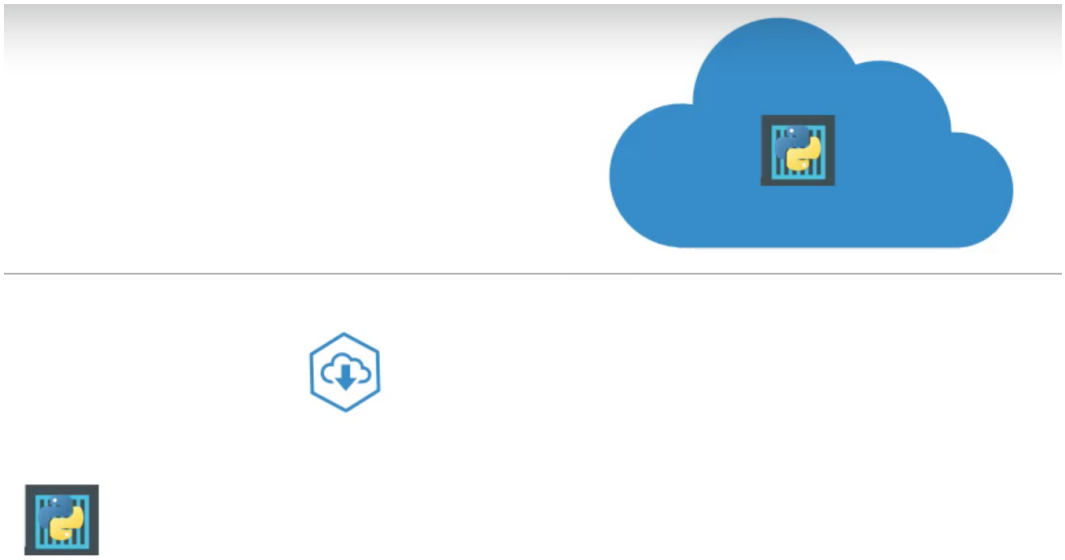
```
kubectl delete replicationcontroller myapp-rc
```

```
kubectl delete replicaset myapp-replicaset
```

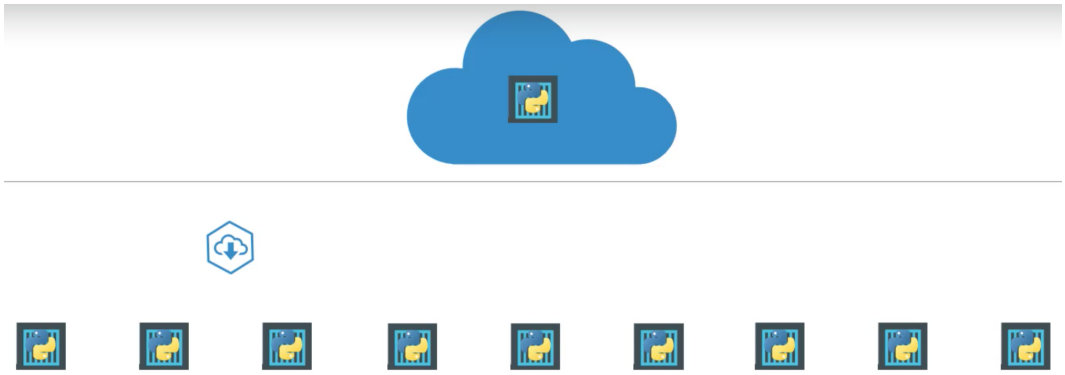
Deployment

Let's talk about concept called Deployment.

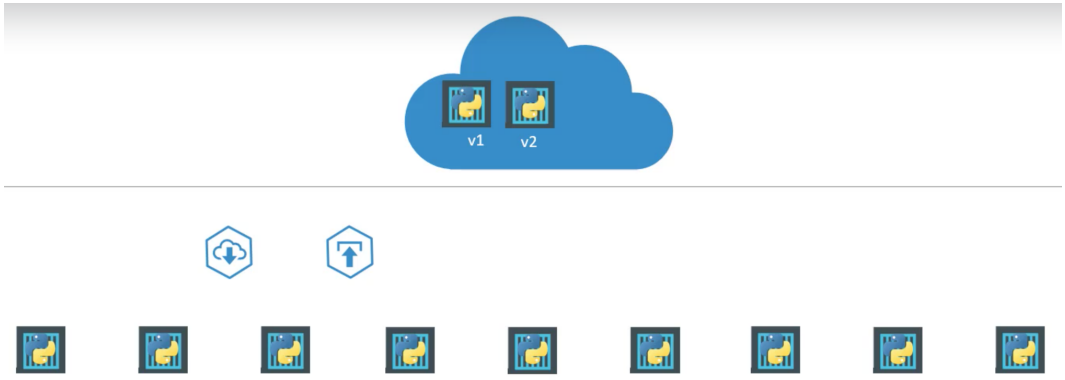
- **Let's first think about how we might want to deploy the applications in Production environment to understand Deployments.**
- **Say for example you have web server which needs to be deployed in Production environment.**



- **For sure you are not going to stop having one server for serving your customers, eventually over the period number of servers might increase.**

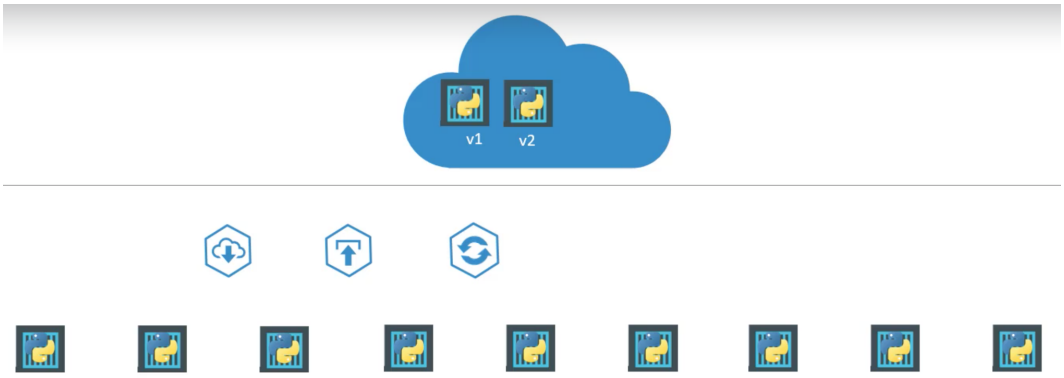


- **Secondly, whenever newer versions of application become available in Docker registry. You may like to upgrade your applications seamlessly.**

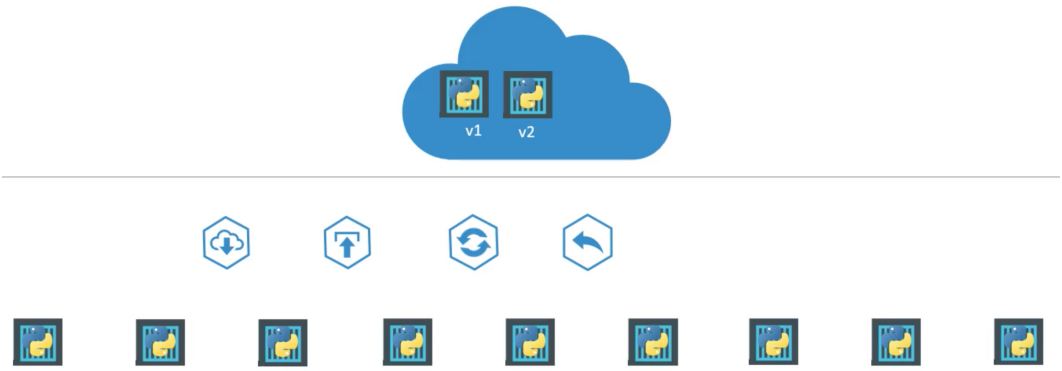


- **However, while you upgrade your instances you do not want them to upgrade all of them once because it may impact users accessing the**

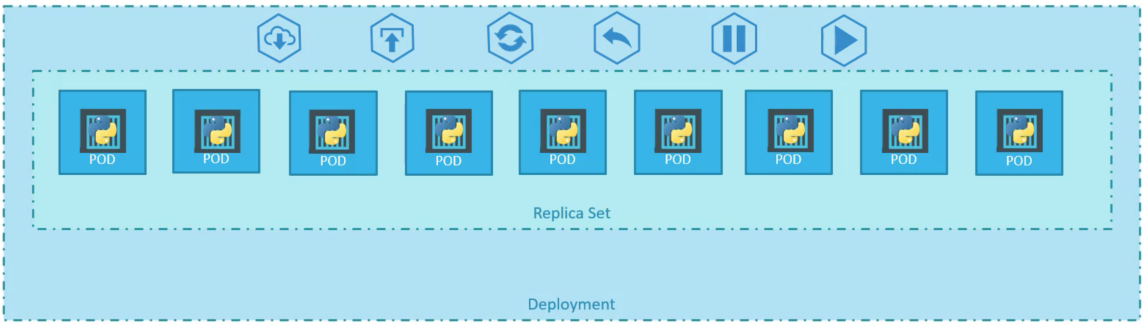
application. Hence you might want to upgrade the instance one after the other which is know as Rolling updates.



- Suppose one of the upgrades which you perform encountered an unexpected error and you are asked to undo the recent upgrades which is known as Rollback changes.



- So far in this training we have discussed about PODs which helps to encapsulate your Instances.
- Multiple such PODs are deployed using Replica Sets.
- Then comes the kubernetes object which comes higher in the hierarchy.



- **The main objective of Deployment is to help you in Making multiple changes in your environment such as**
  - **upgrading the underlying instances application versions seamlessly.**
  - **Rollback if incase of any problems encountered.**

Hand-on Deployment

Let's see how to create deployments.

- **Creating file**

```
E3-deployment.yml
```

- **Changes here are kind & name of object.**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
    app: myapp
    type: front-end
spec:
 template:
   metadata:
    name: myapp-prod
    labels:
        app: myapp
        type: front-end
   spec:
    containers:
      - name: nginx-container
        image: nginx
 replicas: 3
 selector:
    matchLabels:
      type: front-end
```

Execute code

```
kubectl apply -f E3-deployment.yml
```

Verify all the details

```
kubectl get deployment
```

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

```
kubectl get replicaset
```

```
kubectl get pods
```

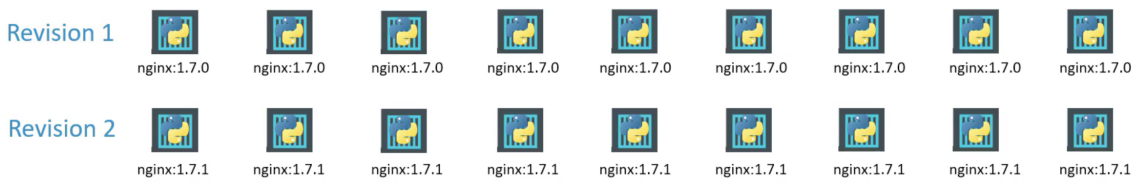
To see all these details in one command

```
kubectl get all
```

Updates & Rollback

Rollout & Versioning

- **When you first create deployment it triggers rollout.**
- **New rollout creates new deployment revision.**
- **In future when application is upgraded a new rollout is triggered and it creates new deployment revision.**
- **This approach actually helps us to keep track of the changes made to our deployments & enables us to rollback to previous version when and where required.**



How to check the rollout and revision status ?

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

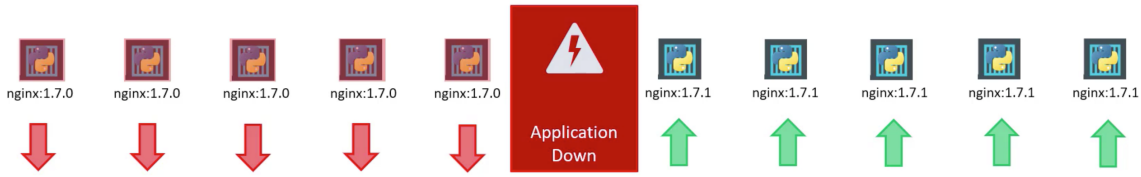
```
kubectl rollout history deployment/myapp-deployment
```

Deployment Strategy

There are two different deployment strategy.

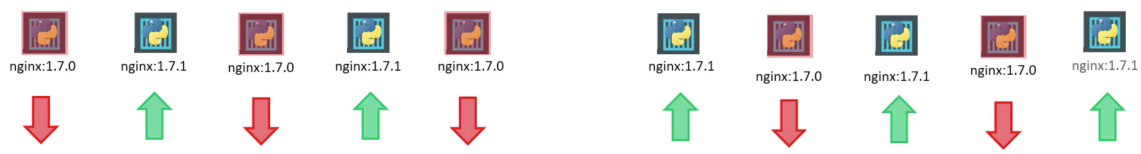
Strategy 1 : Recreate

- **For example lets assume you have 5 web application instance running.**
- **One way to upgrade all these instance to newer version is "destroy all the older one and recreate new instance with newer version"**
- **But the problem with this approach is application might be down and inaccessible for your customers until new instances are coming up.**



Strategy 2 : Rolling update

- **Through this strategy we are going to bring down the instance one by one and and replace it with newer version before we move to the next one.**
- **Through this approach you can achieve zero downtime.**
- **Note : This is the default approach followed by kubernetes even if you are not specifying anything in your YAML configuration file.**



Rollout update

Let's try to rollout the update for existing containers running.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
    app: myapp
    type: front-end
spec:
 template:
   metadata:
    name: myapp-prod
    labels:
        app: myapp
        type: front-end
   spec:
    containers:
      - name: nginx-container
        image: nginx:1.7.1
 replicas: 3
 selector:
    matchLabels:
      type: front-end
```

Apply code

```
kubectl apply -f E4-deploy-update.yml
```

You can also do this upgrade in other way without updating your YAML file.

```
kubectl set image deployment/myapp-deployment nginx=nginx:1.9.1
```

Once done you can check the new revision using command

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

Now you can understand the difference between recreate and rollout.

- **Recreate will zero down all the instance first before create new.**

- **Whereas rollout method is going to do it one by one.**

```
C:\Kubernetes>kubectl describe deployment myapp-deployment
Name:               myapp-deployment
Namespace:          default
CreationTimestamp:  Sat, 03 Mar 2018 17:01:55 +0800
Labels:             app=myapp
                    type=front-end
Annotations:        deployment.kubernetes.io/revision=2
                    kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"apps/v1","kind":"Deployment","me
s\\Google...
                    kubernetes.io/change-cause=kubectl apply --filename=d:\Mumshad Files\Google Drive\Udemy\Kubernet
Selector:           type=front-end
Replicas:           5 desired | 5 updated | 5 total | 5 available | 0 unavailable
StrategyType:       Recreate
MinReadySeconds:    0
Pod Template:
  Labels:  app=myapp
           type=front-end
  Containers:
   nginx-container:
    Image:        nginx:1.7.1
    Port:         <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   myapp-deployment-54c7d6ccc (5/5 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  11m   deployment-controller  Scaled up replica set myapp-deployment-6795844b58 to 5
  Normal  ScalingReplicaSet  1m    deployment-controller  Scaled down replica set myapp-deployment-6795844b58 to 0
  Normal  ScalingReplicaSet  56s   deployment-controller  Scaled up replica set myapp-deployment-54c7d6ccc to 5
```

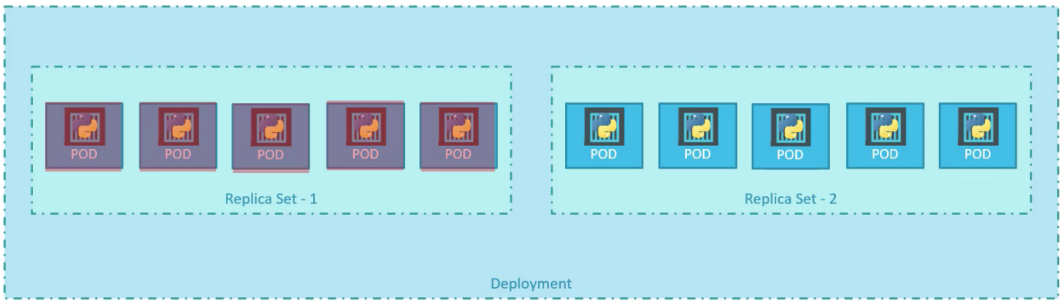
```
C:\Kubernetes>kubectl describe deployment myapp-deployment
Name:               myapp-deployment
Namespace:          default
CreationTimestamp:  Sat, 03 Mar 2018 17:16:53 +0800
Labels:             app=myapp
                    type=front-end
Annotations:        deployment.kubernetes.io/revision=2
                    kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"apps/v1","kind":"Deployment","metadat
Files\\Google...
                    kubernetes.io/change-cause=kubectl apply --filename=d:\Mumshad Files\Google Drive\Udemy\Kubernetes\De
Selector:           type=front-end
Replicas:           5 desired | 5 updated | 6 total | 4 available | 2 unavailable
StrategyType:       RollingUpdate
MinReadySeconds:    0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=myapp
           type=front-end
  Containers:
   nginx-container:
    Image:        nginx
    Port:         <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    ReplicaSetUpdated
OldReplicaSets:  myapp-deployment-67c749c58c (1/1 replicas created)
NewReplicaSet:   myapp-deployment-7d57dbdb8d (5/5 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  1m    deployment-controller  Scaled up replica set myapp-deployment-67c749c58c to 5
  Normal  ScalingReplicaSet  1s    deployment-controller  Scaled up replica set myapp-deployment-7d57dbdb8d to 2
  Normal  ScalingReplicaSet  1s    deployment-controller  Scaled down replica set myapp-deployment-67c749c58c to 4
  Normal  ScalingReplicaSet  1s    deployment-controller  Scaled up replica set myapp-deployment-7d57dbdb8d to 3
  Normal  ScalingReplicaSet  0s    deployment-controller  Scaled down replica set myapp-deployment-67c749c58c to 3
  Normal  ScalingReplicaSet  0s    deployment-controller  Scaled up replica set myapp-deployment-7d57dbdb8d to 4
  Normal  ScalingReplicaSet  0s    deployment-controller  Scaled down replica set myapp-deployment-67c749c58c to 2
  Normal  ScalingReplicaSet  0s    deployment-controller  Scaled up replica set myapp-deployment-7d57dbdb8d to 5
  Normal  ScalingReplicaSet  0s    deployment-controller  Scaled down replica set myapp-deployment-67c749c58c to 1
```

Upgrades

Let's all try to understand behind the woods when you are performing all these steps.

- **When you rollout first deployment your instances will be running under Revision 1**

- **When you upgrad your instances kubernetes will create new Revision with updated instances & will bring down all the instance running in current Revision.**



This can be verified using command

```
kubectl get replicaset
```

Rollback

Now how to rollback from currnt updated revision to older revision if incase you are encountering some problem with your application after upgrade.

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

This command wiill rollback from revision2 to revision 1 again, You can find this from command.

```
kubectl get replicaset
```

Session 6 : Kubernetes on cloud

You can also create kubernetes cluster in cloud.

- In AWS kubernetes cluster is refered as Elastic Kubernetes Service (EKS)
- In Azure kubernetes cluster is refered as Azure Kubernetes Service (AKS)
- In GCP kubernetes cluster is refered as Google Kuberenets Engine (GKE)

Prerequistes to setup kubernets cluster in AWS.

- AWS account
- Knows some AWS basics