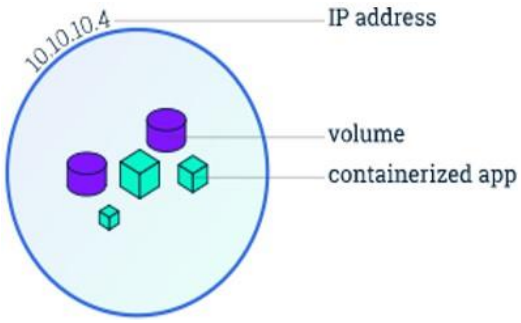
# KUBERNETES WORKLOADS

# WORKLOADS

- A workload is an application running on Kubernetes. Whether workload is a single component or several that work together, on Kubernetes run it inside a set of pods.
- Kubernetes provides several built-in workload resources:
  - Deployment and ReplicaSet
  - StatefulSet
  - StatefulSet
  - Job and CronJob

## PODS:

- A pod is a smallest and simplest **Kubernetes object**.
- Pods represents a single instance of a running process in your cluster.
- Pods contain one or more containers, such as **Docker containers.**
- When a Pod runs multiple containers, the containers are managed as a **single entity** and share the **Pod's resources**.




**Example of a Pod which consists of a container running the image nginx:**

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

### Create a Pod using YAML File:

$kubectl create -f <filename>

$kubectl get pods

$kubectl describe pod <pod-name>

### Connect a Container in a Pod:

$kubectl exec -it <podname> -c <containe_rname> -- sh

$exit

## HOW PODS MANAGE MULTIPLE CONTAINERS:

- Pods are designed to support multiple cooperating processes (as containers) that form a cohesive unit of service.
- The containers in a Pod are automatically co-located and co-scheduled on the same physical or virtual machine in the cluster.
- The containers can share resources and dependencies, communicate with one another, and coordinate when and how they are terminated.

```
apiVersion: v1
kind: Pod
metadata:
   name: two-containers
   labels:
     app: web
spec:
   containers:
     - name: Linux-Centos
       image: centos
       ports:
         - containerPort: 80
     - name: Linux-Ubuntu
       image: ubuntu
       ports:
         - containerPort: 90
```

### Create a Pod using above YAML File:

$kubectl create -f <file-name>

$kubectl get pods

**Connect a First Container:**

$kubectl exec -it <pod-name> -c Linux-Centos bash

$exit

**Connect Second Container:**

$kubectl exec -it <pod-name> -c Linux-Ubuntu bash

$exit

**NOTE:** Restarting a container in a Pod should not be confused with restarting a Pod. A Pod is not a process, but an environment for running container(s). A Pod persists until it is deleted.

## PODS AND CONTROLLERS:

- A controller for the resource handles replication and rollout and automatic healing in case of Pod failure.

  **E.g.:** If a Node fails, a controller notices that Pods on that Node have stopped working and creates a replacement, Pod. The scheduler places the replacement Pod onto a healthy Node.

## POD TEMPLATES:

- Controllers for workload resources create Pods from a pod template and manage those Pods on your behalf.
- PodTemplates are specifications for creating Pods, and are included in workload resources such as Deployments, Jobs, and DaemonSets.
- Each controller for a workload resource uses the PodTemplate inside the workload object to make actual Pods.

**A manifest for a simple Job with a template that starts one container:**

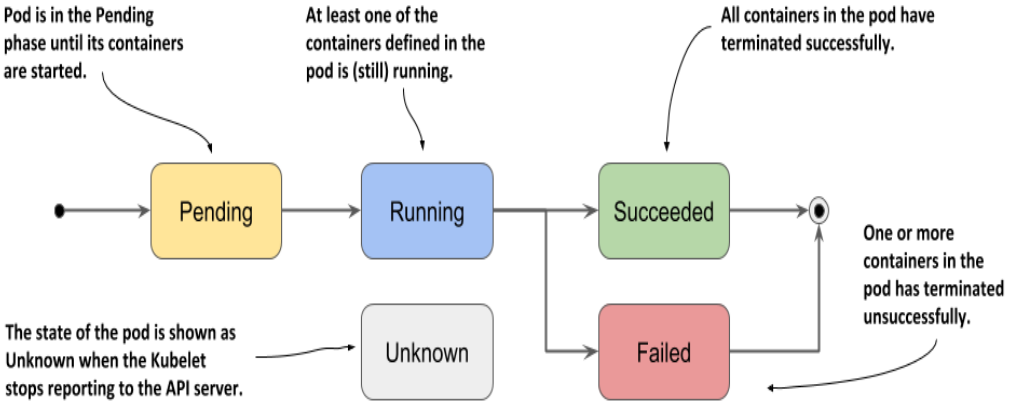
```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: hello
spec:
  template:
    # This is the pod template
    spec:
      containers:
      - name: hello
        image: busybox:1.28
        command: ['sh', '-c', 'echo "Hello, Kubernetes!" && sleep 3600']
      restartPolicy: OnFailure
    # The pod template ends here
```

## POD UPDATE AND REPLACEMENT:

- When the Pod template for a workload resource is changed, the controller creates new Pods based on the updated template instead of updating or patching the existing Pods.
- **Pod update operations like patch, and replace have some limitations:**
  - Most of the metadata about a Pod is immutable. For example, you cannot change the **namespace, name, uid,** or **creationTimestamp** fields; the generation field is unique. It only accepts updates that increment the field's current value.
  - Pod updates may not change fields other than **spec.containers[*].image , spec.initContainers[*].image, spec.activeDeadlineSeconds or spec.tolerations**. For **spec.tolerations**, you can only add new entries.
  - When updating the **spec.activeDeadlineSeconds** field, two types of updates are allowed:
    - setting the unassigned field to a positive number;
    - updating the field from a positive number to a smaller, non-negative number.

## POD LIFECYCLE:

- Pods follow a defined lifecycle, **starting** in the **Pending phase**, moving through **Running** if at least one of its primary containers starts OK, and then through either the **Succeeded** or **Failed** phases depending on whether any container in the Pod terminated in failure.




## CONTAINER RESTART POLICY:

- The **spec** of a Pod has a **restartPolicy** field with possible values **Always**, **OnFailure**, and **Never.** The default value is **Always.**
- **restartPolicy** only refers to restarts of the containers by the kubelet on the same node.
- After containers in a Pod exit, the kubelet restarts them with an exponential back-off delay (10s, 20s, 40s, …), that is capped at five minutes. Once a container has executed for 10 minutes without any problems, the kubelet resets the restart backoff timer for that container.

## TERMINATION OF PODS:

$kubectl delete pod <pod-name>

**NOTE:** By default, all deletes are graceful within 30 seconds.

The kubectl delete command supports the **--grace-period=<seconds>** option which allows you to override the default and specify your own value.

# WORKLOAD RESOURCES

## DEPLOYMENTS:

- A Deployment provides declarative updates for Pods and ReplicaSets.
- Deployments to create new ReplicaSets, or to remove existing Deployments.

## USE CASE:

- **Create a Deployment to rollout a ReplicaSet.** The ReplicaSet creates Pods in the background. Check the status of the rollout to see if it succeeds or not.
- **Declare the new state of the Pods** by updating the PodTemplateSpec of the Deployment. A new ReplicaSet is created and the Deployment manages moving the Pods from the old ReplicaSet to the new one at a controlled rate. Each new ReplicaSet updates the revision of the Deployment.
- **Rollback to an earlier Deployment revision** if the current state of the Deployment is not stable. Each rollback updates the revision of the Deployment.
- **Scale up the Deployment to facilitate more load.**
- **Pause the rollout of a Deployment** to apply multiple fixes to its PodTemplateSpec and then resume it to start a new rollout.
- **Use the status of the Deployment** as an indicator that a rollout has stuck.
- **Clean up older ReplicaSets** that you don't need anymore.

## CREATING A DEPLOYMENT:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

**Create the Deployment:**

$kubectl create -f <file-name>

$kubectl get deployments

$kubectl describe deployment file-name

To Check the replicaset (rs):

$kubectl get rs

## UPDATING A DEPLOYMENT:

**update the nginx Pods to use the nginx:1.16.1 image instead of the nginx:1.14.2 image:**

$kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1

**Alternatively, you can edit the Deployment and change:**

$kubectl edit deployment/nginx-deployment

**To see the rollout status:**

$kubectl rollout status deployment/nginx-deployment

$kubectl get pods

$kubectl describe deployments

## ROLLING BACK A DEPLOYMENT:

Sometimes, you may want to rollback a Deployment; for example, when the Deployment is not stable, such as crash looping. By default, all of the Deployment's rollout history is kept in the system so that you can rollback anytime you want.

$kubectl set image deployment/nginx-deployment nginx=nginx:1.161

$kubectl rollout status deployment/nginx-deployment

**Checking Rollout History of a Deployment:**

$kubectl rollout history deployment/nginx-deployment

**To see the details of each revision:**

$kubectl rollout history deployment/nginx-deployment --revision=2

**Rolling back to a previous revision:**

$kubectl rollout undo deployment/nginx-deployment

(or)

$kubectl rollout undo deployment/nginx-deployment --to-revision=2

**To check the deployments:**

$kubectl get deployment nginx-deployment

$kubectl describe deployment nginx-deployment

## SCALING A DEPLOYMENT:

$kubectl scale deployment/nginx-deployment --replicas=5

Assuming **horizontal Pod autoscaling** is enabled in your cluster, you can set up an autoscaler for your Deployment and choose the minimum and maximum number of Pods you want to run based on the CPU utilization of your existing Pods.

$kubectl autoscale deployment/nginx-deployment --min=5 --max=10 --cpu-percent=60

## PAUSING AND RESUMING A ROLLOUT OF A DEPLOYMENT:

**To Pause the deployment:**

$kubectl rollout pause deployment/nginx-deployment

**Then update the image of the Deployment:**

$kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1

**Notice that no new rollout started:**

$kubectl rollout history deployment/nginx-deployment

$kubectl get rs

**You can make as many updates as you wish, for example, update the resources that will be used:**

$kubectl set resources deployment/nginx-deployment -c=nginx --limits=cpu=200m,memory=512Mi

**Resume the Deployment rollout and observe a new ReplicaSet coming up with all the new updates:**

$kubectl rollout resume deployment/nginx-deployment

**Watch the status of the rollout until it's done.**

$kubectl get rs -w

## DELETE A DEPLOYMENTS:

$kubectl get deployments

$kubectl delete deployment <deployment-name>

$kubectl get rs

$kubectl get deployments

# REPLICASET:

- It is used to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

## Create a Replicaset:

$kubectl create -f file-name

$kubectl get rs

$kubectl describe rs rs-name

## Delete a Replicaset:

$kubectl delete rs <rs-name>

$kubectl get rs