kubernetes
WORKLOADS

❖ WORKLOADS:

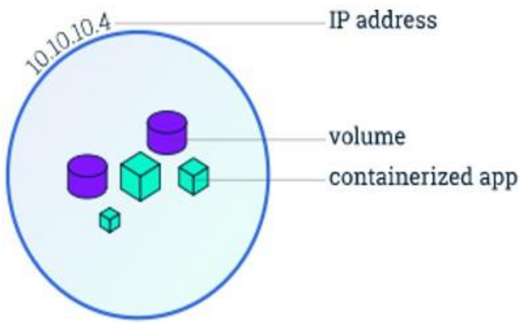
- A workload is an application running on Kubernetes. Whether workload is a single component or several that work together, on Kubernetes run it inside a set of pods.
- Kubernetes provides several built-in APIs for declarative management of your workloads and the components of those workloads.
- Kubernetes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

➢ PODS:

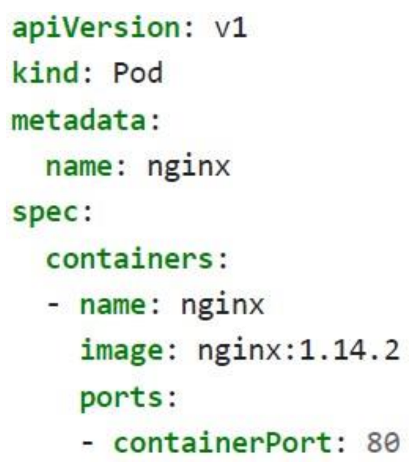
- A pod is a smallest and simplest **Kubernetes object**.
- Pods represents a single instance of a running process in your cluster.
- When a Pod runs multiple containers, the containers are managed as a **single entity** and share the **Pod's resources**.
- Pods in a Kubernetes cluster are used in two main ways:

PODS THAT RUN A SINGLE CONTAINER: The "one-container-per-Pod" model is the most common Kubernetes use case; in this case, you can think of a Pod as a wrapper around a single container; Kubernetes manages Pods rather than managing the containers directly.

PODS THAT RUN MULTIPLE CONTAINERS THAT NEED TO WORK TOGETHER: A Pod can encapsulate an application composed of multiple co-located containers that are tightly coupled and need to share resources. These co-located containers form a single cohesive unit. Grouping multiple co-located and co-managed containers in a single Pod is a relatively advanced use case.



EXAMPLE: 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> -- bash

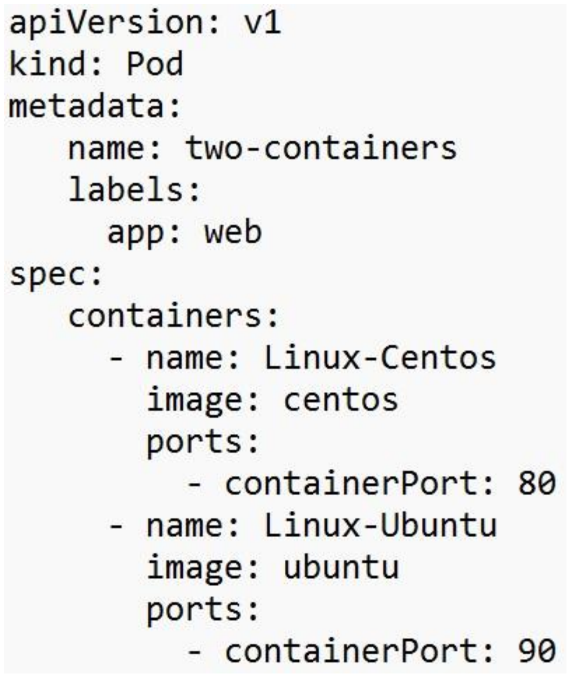
$kubectl exec -it nginx -- bash

$exit

$kubectl get pods

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:

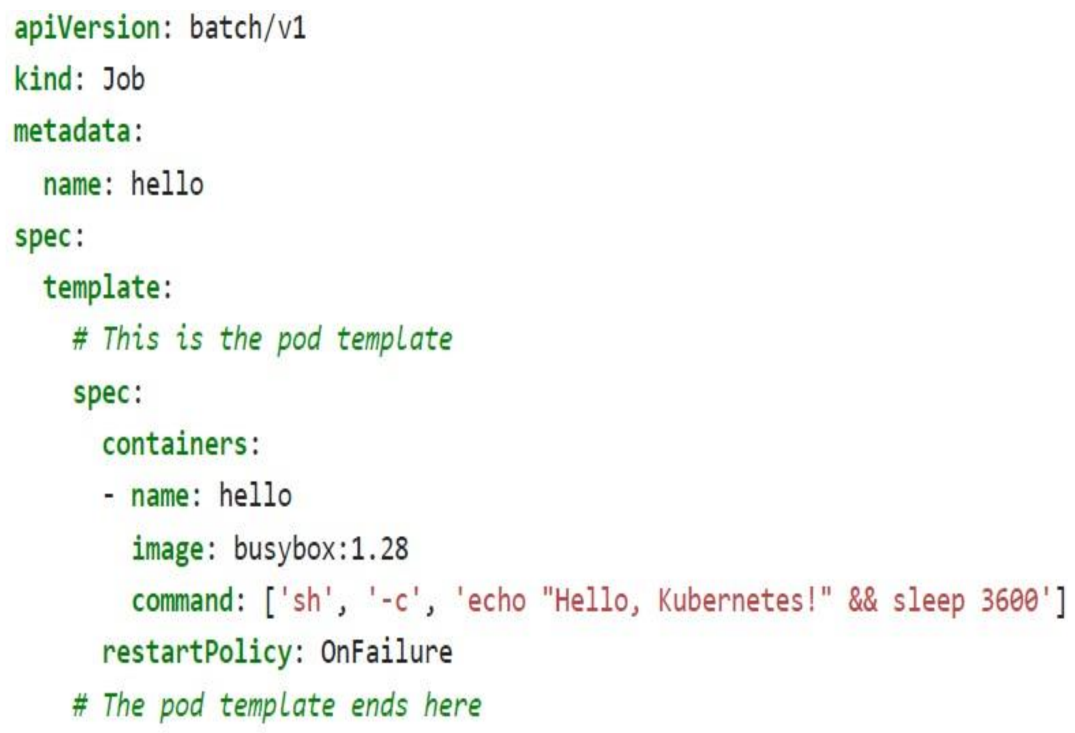
- A controller for the resource handles replication and rollout and automatic healing in case of Pod failure.

  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.

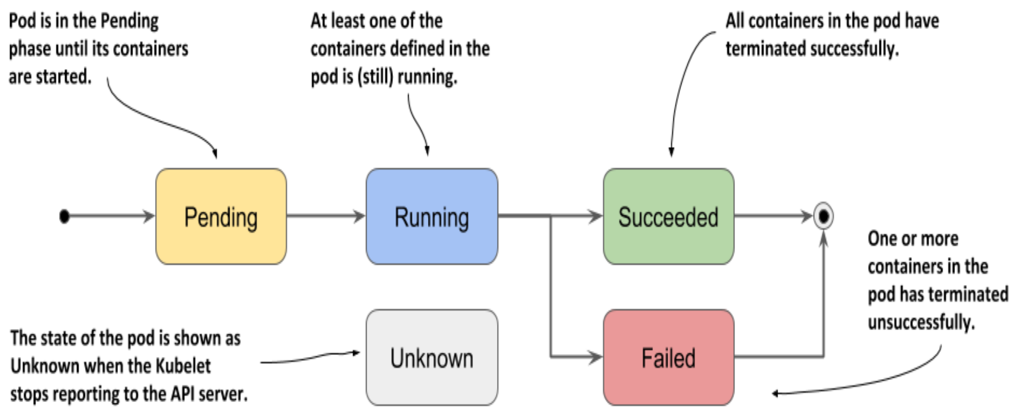
  A manifest for a simple Job with a template that starts one container:



```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 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.


Pod is in the Pending phase until its containers are started.
At least one of the containers defined in the pod is (still) running.
All containers in the pod have terminated successfully.
Pending
Running
Succeeded
One or more containers in the pod has terminated unsuccessfully.
The state of the pod is shown as Unknown when the Kubelet stops reporting to the API server.
Unknown
Failed

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.