UPES
UNIVERSITY OF TOMORROW

Kubernetes

Prepared by:
Ms. Avita Katal
Assistant Professor (SG)
School of Computer Science
UPES, Dehradun

UPES
UNIVERSITY OF TOMORROW

Kubernetes Objects-Part 1

Prepared by:
Ms. Avita Katal
Assistant Professor (SG)
School of Computer Science
UPES, Dehradun

Objectives

1. Define a Kubernetes object and its properties.
2. Describe basic Kubernetes objects and their features.
3. Demonstrate how Kubernetes objects relate to each other.

Key terms review

Object: A software object is a **bundle of data** that has an **identity, a state, and a behavior**. Examples include variables, data structures, and specific functions.

Entity: Entity has an **identity and associated data**. For example, in banking, a customer account is an entity.

Persistent: Persistent means **something will last even if there is a server failure or network attack**.

Kubernetes Objects

Kubernetes objects are **persistent entities**.

Examples include: **Pods, Namespaces, ReplicaSets, Deployments,** and

Pods

Namespaces

ReplicaSets

Deployments

Kubernetes objects consist of two main fields ***- object spec and status.***

- The object spec is provided by the user which dictates an **object's desired state.**
- Status is provided by Kubernetes. This describes the **current state of the object.**
- Kubernetes works towards matching the current state to the desired state.

To work with these objects, use the Kubernetes API directly with the client libraries, and the kubectl command-line interface, or both.

Labels and Selectors

- Labels are **key/value pairs attached to objects**.

  - Intended for identification of objects
  - Does not uniquely identify a single object.
  - Helps to organize and group objects.

- Label selectors are the **core grouping method** in Kubernetes.

  - Identify and group a set of objects.

Namespaces and names

- Namespaces provide a mechanism for **isolating groups of resources within a single cluster.**

- This is useful when teams share a **cluster for cost-saving purposes or for maintaining multiple projects in isolation.**

- Namespaces are **ideal when the number of cluster users is large.**

- Examples of namespaces are **kube-system**, intended for **system users** and the **default namespace** used to hold **users' applications**.

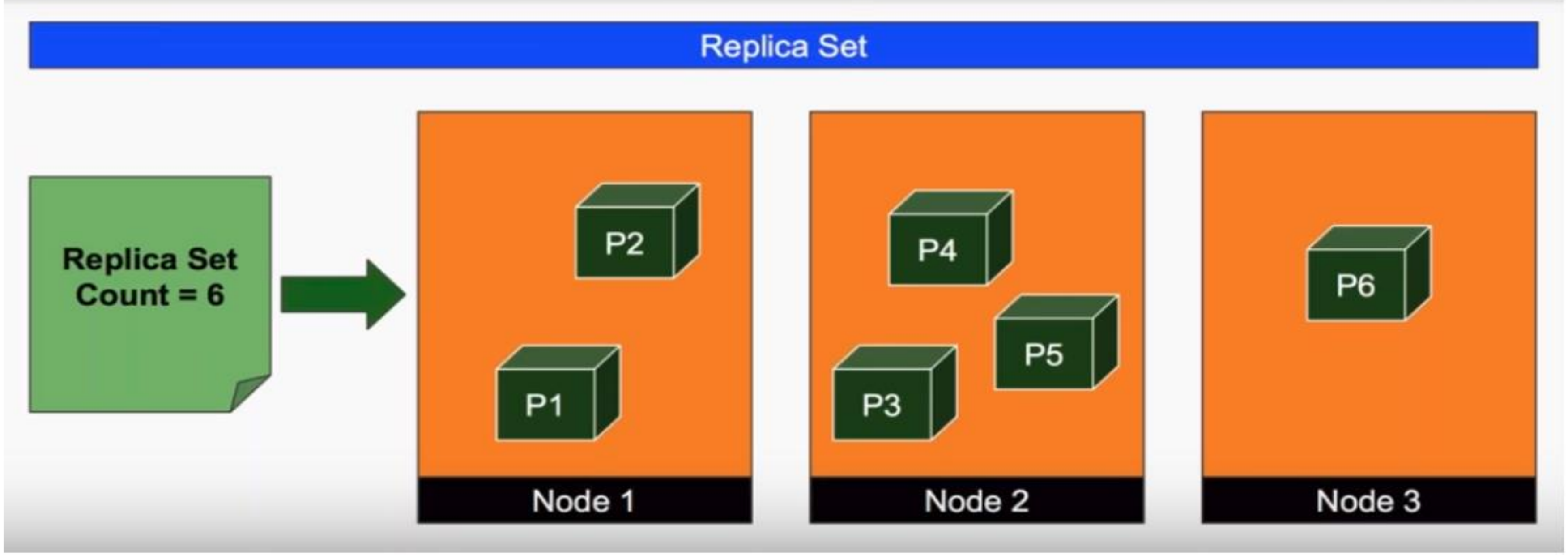
Pods

- A Pod is the **simplest unit** in Kubernetes.
- A Pod represents a **process or a single instance of an application** running in the cluster.
- A Pod usually **wraps one or more containers.**
- Creating **replicas of a Pod serves to scale an application horizontally.**
- YAML files are often used to **define the objects that you want to create**. The YAML files shown defines a simple pod.

```
apiVersion:v1
kind: Pod
metadata:
        name: nginx
spec:
         containers:
             -name: nginx
               image:nginx:1.7.9
                ports:
                 -containerPort: 80
```

ReplicaSet

- A ReplicaSet is a **set of identical running replicas of a Pod** that are **horizontally scaled.**



ReplicaSet

- The **replicas field specifies the number of replicas that should be running at any given time**. Whenever this field is updated, the ***ReplicaSet creates or deletes Pods to meet the desired number of replicas.***

- A **Pod template is included in the ReplicaSet spec** which defines the Pods that should be created by the ReplicaSet.

- Under the selector field, the **labels supplied in the MatchLabels field specify the Pods that can be acquired by the ReplicaSet.**

- Notice that **the label identified in the MatchLabels field is the same as the labels field in the Pod template**. Both are the app: nginx.

- Creating **ReplicaSets directly is not recommended**. Instead, create a **Deployment, which is a higher-level concept that manages ReplicaSets and offers more features and better control**.

```
apiVersion:apps/v1
kind: ReplicaSet
metadata:
        name: nginx-replicaset
         labels:
            app:nginx
  spec:
         replicas: 3
         selector:
         matchLabels:
               app:nginx
    template:
          metadata:
           labels:
               app:nginx
     spec:
         containers:
             -name: nginx
               image:nginx:1.7.9
                ports:
                 -containerPort: 80
```

Deployment

- A Deployment is a **higher-level object that provides updates for both Pods and ReplicaSets.**

- Deployments run **multiple replicas of an application using ReplicaSets** and offer **additional management capabilities** on top of these ReplicaSets.

- Deployments are suitable for **stateless applications**. For stateful applications, **Stateful Sets** are used.

- One key feature provided by Deployments but not by ReplicaSets is **rolling updates**. A rolling update **scales up a new version to the appropriate number of replicas and scales down the old version to zero replicas.**

- The ReplicaSet **ensures that the appropriate number of Pods exist**, while the Deployment **orchestrates the roll out of a new version.**

```
apiVersion:apps/v1
kind: Deployment
metadata:
        name: nginx-replicaset
         labels:
            app:nginx
  spec:
         replicas: 3
         selector:
         matchLabels:
               app:nginx
    template:
          metadata:
           labels:
               app:nginx
     spec:
         containers:
             -name: nginx
               image:nginx:1.7.9
                ports:
                 -containerPort: 80
```

Kubernetes Objects-Part 2

Prepared by:
Ms. Avita Katal
Assistant Professor (SG)
School of Computer Science
UPES, Dehradun

Objectives

1. Describe the purposes, properties, and uses of a Service.
2. Describe the roles and uses for the ClusterIP, NodePort, LoadBalancer, and External Name Services.
3. Describe the roles and uses for Ingress, DaemonSet, StatefulSet, and a Job.

Service

- A Service is a **REST object, like Pods**. Services are a **logical abstraction for a set of Pods** in a cluster.

- They provide **policies for accessing the Pods and cluster and act as a load balancer across the Pods.**

- Each Service is assigned a **unique IP address for accessing applications deployed on Pods** and a Service **eliminates the need for a separate service discovery process.**

- A Service **supports multiple protocols such as TCP, which is the default protocol, UDP**, and others, and supports multiple port definitions.

- The **port number with the same name can vary in each backend Pod**. In addition, a Service can have an optional selector and can **optionally map incoming ports to a targetPort.**

Why service is needed?

- A service is needed **because Pods in a cluster can be destroyed and new Pods can be created at any time.**
- This **volatility leads to discoverability issues because of changing IP addresses.**
- A Service keeps track of Pod changes and **exposes a single IP address or a DNS name** and **utilizes selectors to target a set of Pods**.

Native Kubernetes Applications

For native Kubernetes applications, **API endpoints are updated whenever changes are detected to the Pods in the Service.**

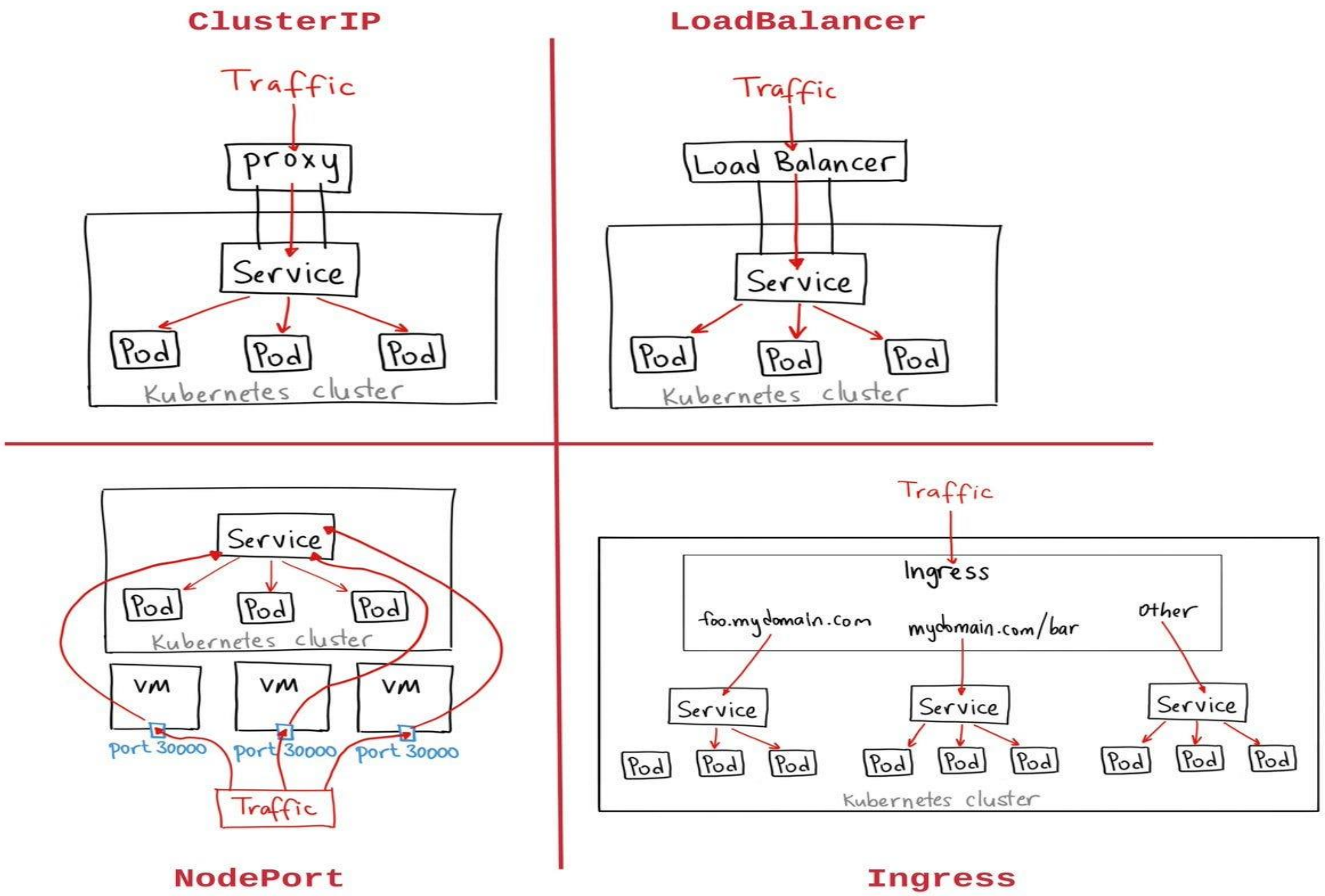
Non-Native Kubernetes Applications

For Non-native applications, Kubernetes uses a **virtual-IP-based bridge or load balancer in between the applications and the backend Pods.**

Types of Services

There are four types of Services:

- **ClusterIP**
- **NodePort**
- **LoadBalancer and**
- **External Name**.



Service: ClusterIP

- ClusterIP is the **default and most common service type**.
- Kubernetes assigns a **cluster-internal IP address** to the **ClusterIP Service** that makes the **Service only reachable within the cluster.**
- A ClusterIP service **cannot make requests to Service from outside the cluster.**
- You can set the ClusterIP address in the **Service definition file**, and the ClusterIP Service **provides Inter-service communication within the cluster.** For example, **communication between the front-end and back-end components of your app.**

Service: NodePort

- An **extension of ClusterIP Service**, a **NodePort Service creates and routes the incoming requests automatically to the ClusterIP Service.**

- A NodePort **exposes the Service on each Node's IP address at a static port.** Note that for security purposes, production use is not recommended.

- Kubernetes exposes a **single Service with no load-balancing requirements for multiple services.**

Service: External Load Balancer

- An **extension of the NodePort Service**, an External Load Balancer, or ELB, **creates NodePort and ClusterIP Services automatically.**

- An ELB **integrates and automatically directs traffic to the NodePort Service.**

- To expose a **Service to the Internet, you need a new ELB with an IP address.** You can use a cloud provider’s ELB to host your cluster.

Service: External Name

- The **External Name Service type maps to a DNS name** and not to any selector and **requires a `spec.externalName` parameter**.

- The External Name Service maps the Service to the contents of the externalName field that returns a CNAME record and its value.

- You can use an **External name to create a Service that represents external storage and enable Pods from different namespaces to talk to each other.**

Maps the Service to the contents of the externalName field (e.g., foo.example.com), using DNS.

When you create a Service, it selects Pods based on the labels defined in the Service manifest. For example:

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

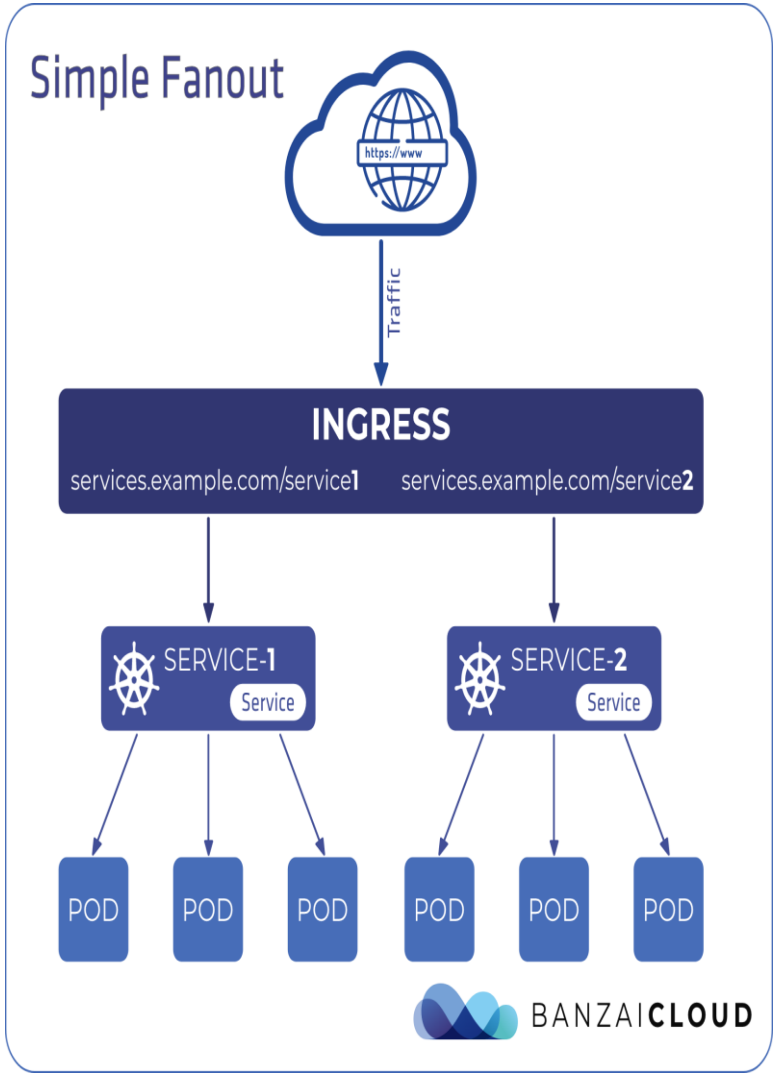
In this example, the Service my-service selects all Pods with the label app=MyApp and routes traffic to them on port 9376. Other parts of your application can now refer to my-service to access those Pods, regardless of the Pods' individual IP addresses.

Ingress

- Ingress is an **API object** that, when **combined with a controller, provides routing rules to manage external users' access to multiple services** in a Kubernetes cluster.

- In production**, Ingress exposes applications to the Internet via port 80 (for HTTP) or port 443 (for HTTPS)**

- While the cluster monitors Ingress, an external Load Balancer is expensive and is managed outside the cluster.



DaemonSet

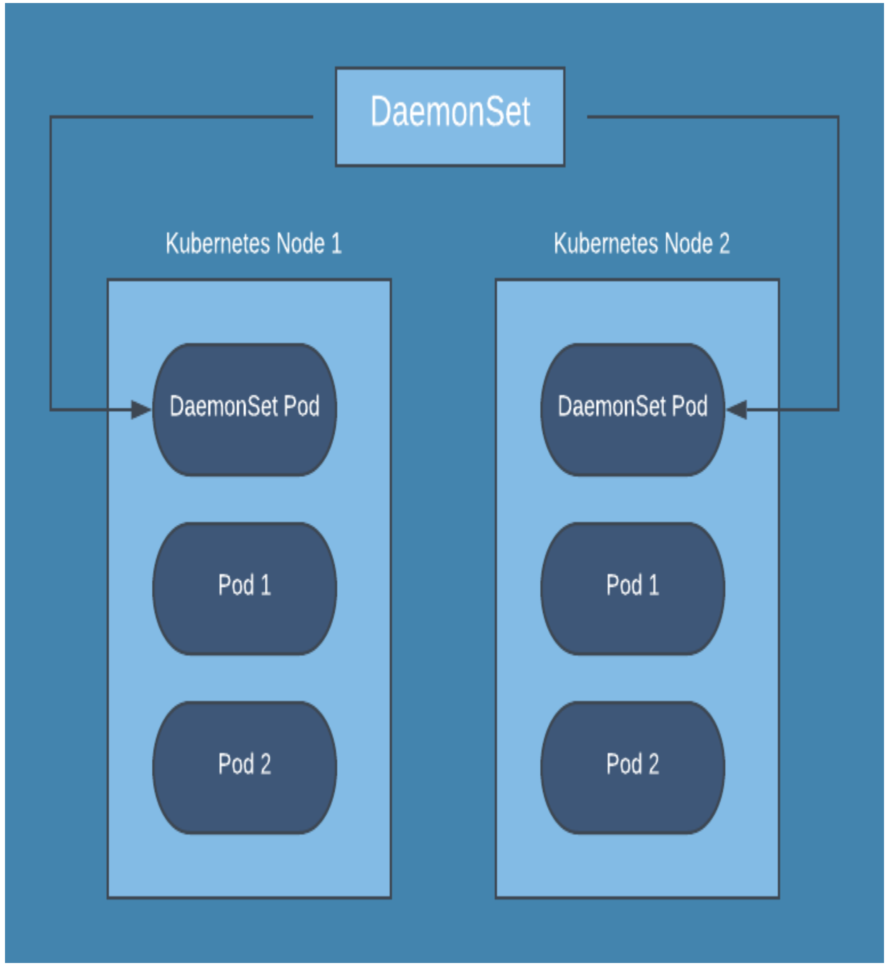
- A DaemonSet is an **object that makes sure that Nodes run a copy of a Pod**.

- As nodes are added to a cluster, Pods are added to the nodes.

- Pods are **garbage collected when removed from a cluster.**

- If you **delete a DaemonSet, all Pods are removed.**

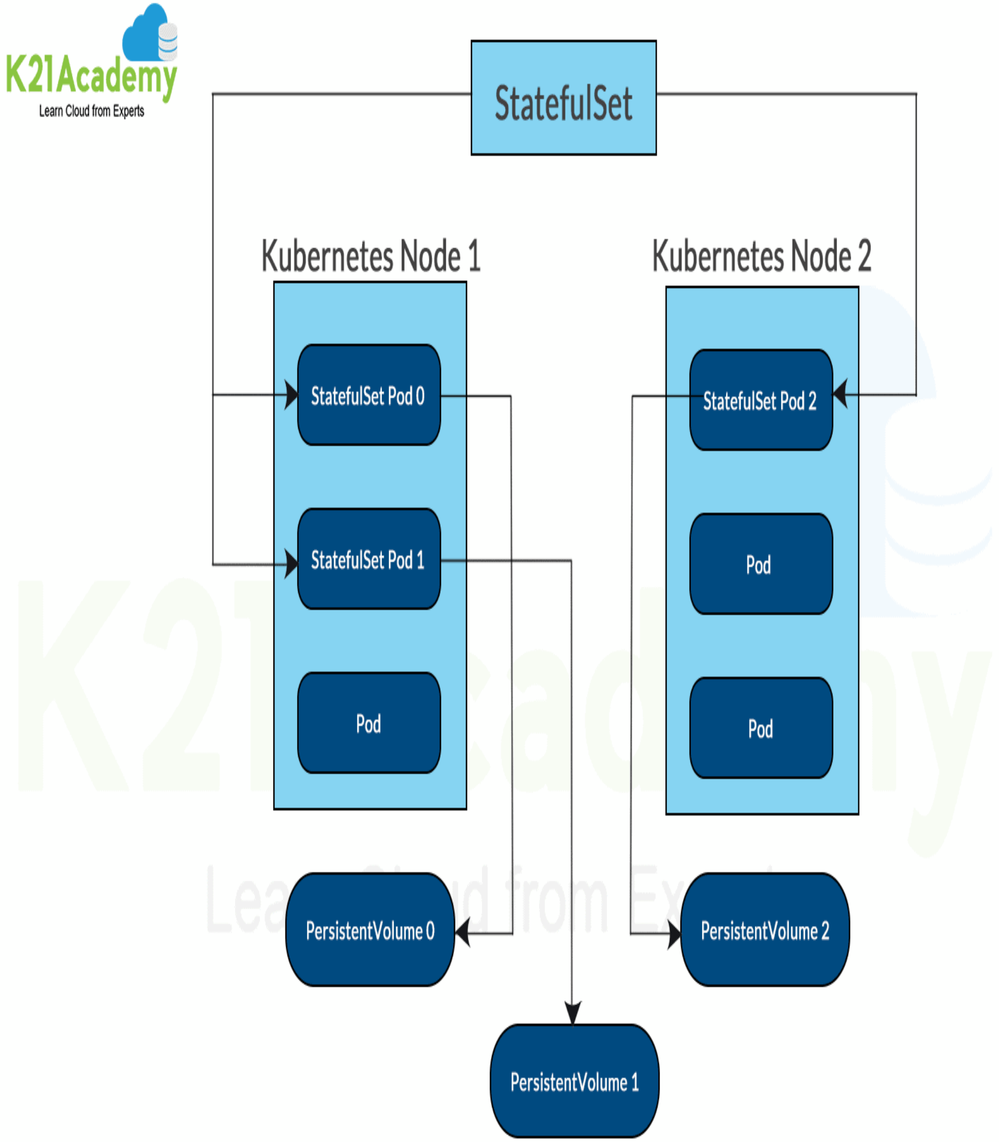
- DaemonSets are ideally **used for storage, logs, and monitoring nodes.**



StatefulSet

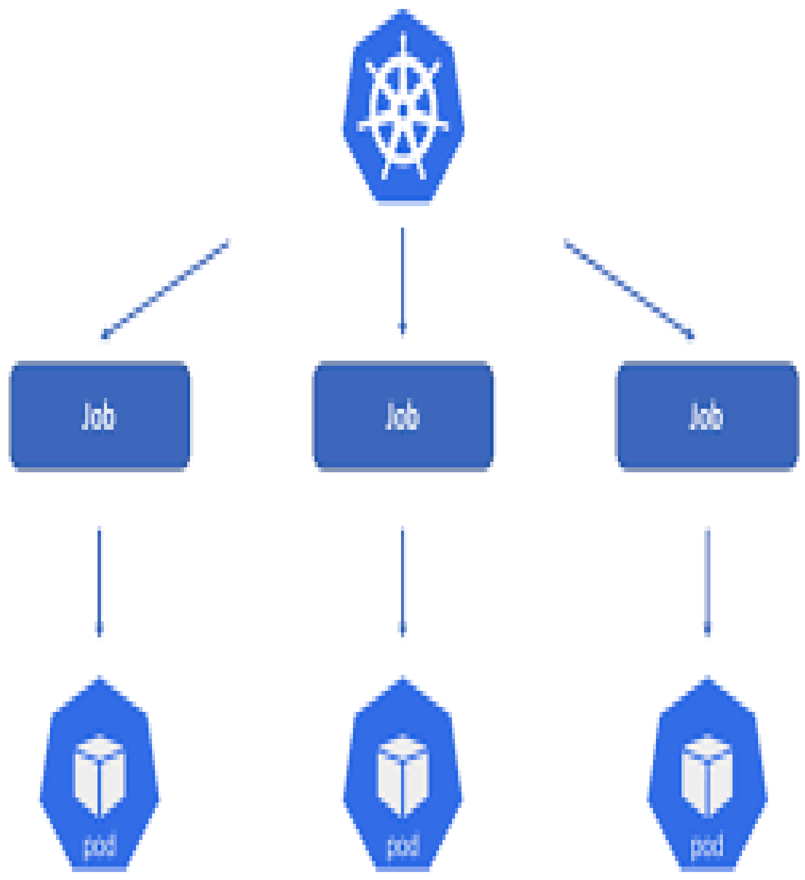
- A StatefulSet is an object that **manages stateful applications, manages deployment and scaling of Pods, and provides guarantees about the ordering and uniqueness of Pods**.

- A StatefulSet maintains a **sticky identity for each Pod request and provides persistent storage volumes for your workloads.**



Job

- A job **creates Pods and tracks the Pod completion process.**
- Jobs are **retried until completed.**
- Deleting a job will **remove the created Pods.**
- Suspending a Job will **delete its active Pods until the job resumes.**
- A job can run **several Pods in parallel**.
- A **CronJob is regularly used to create Jobs** on an iterative schedule.



```
apiVersion: batch/v1
kind: Job
metadata:
  name: my-job
spec:
  parallelism: 3
  completions: 5
  template:
    metadata:
      name: my-pod
    spec:
      containers:
      - name: my-container
        image: my-image:latest
      restartPolicy: Never
```

Here are the key characteristics of a Kubernetes Job:

Completing Tasks: A Job creates one or more Pods and ensures that a specified number of them successfully terminate. Once a specified number of successful completions is reached, the Job is considered complete.

Parallel Execution: You can run multiple Pods in parallel by specifying a parallelism parameter. This means the Job can create and manage several Pods concurrently to speed up the task.

Retries: Jobs support retries. If a Pod fails, the Job controller can create a replacement Pod to retry the task. You can specify the maximum number of retries.

Pod Cleanup: Once a Job completes successfully, it does not clean up the Pods it created. However, you can set a completion deadline, after which the Job and its Pods are terminated.

Task Specification: Jobs are defined using a Pod template, specifying the container image and other parameters required for the task. Kubernetes ensures that the specified number of Pods with the specified template are running.