Kubernetes Pods

A pod is a collection of containers and its storage inside a node of a Kubernetes cluster. It is possible to create a pod with multiple containers inside it. *Pods* are the smallest deployable units of computing that can be created and manage in Kubernetes.

Pod contents are always co-located, co-scheduled and run in a shared host. A Pod models an application-specific ***logical host***. It contains one or more applications that are relatively tightly coupled. In non-cloud contexts, applications executed on the same physical or virtual machine are similar to cloud applications executed on the same logical host. If we frequently deploy single containers, we can replace the word ***pod*** with ***container***.

Pods operate at one level higher than individual containers because it's very common to have a group of containers work together to produce a set of the work process.

Kubernetes
supports more container runtimes than Docker. Docker is the most commonly known term, and it helps to describe Pods using some terminology from Docker.

A Pod's shared context is a set of Linux namespaces, groups, and potentially other facets of isolation that isolate a Docker container. Within a Pod's context, the individual applications may have further sub-isolations.

In Docker, a Pod is similar to a group of Docker containers with shared namespaces and shared filesystem volumes.

Types of Pod

There are two types of Kubernetes Pods, such as:

1. **Single container pod:** The "one-container-per-Pod" model is the most common Kubernetes use case. They can be created with the ***kubctl*** run command, where we have a defined image on the Docker registry, which we will pull while creating a pod. Kubernetes manages Pods rather than managing the containers.
2. **Multi container pod:** These pods are created using ***YAML*** mail with the definition of the containers. A Pod can encapsulate an application composed of multiple co-located containers tightly coupled and need to share resources.

The Pod wraps these containers, storage resources, and a temporary network identity together as a single, cohesive unit of service.

What does a Pod do?

Pods represent the processes running on a cluster. By limiting pods to a single process, Kubernetes can report on each process's health running in the cluster.

- Pods have a unique IP address, which allows them to communicate with each other.
- Pods have persistent storage volumes.
- Pods have configuration information that determines how a container should run.

Although many pods contain a single container, and many will have a few containers that work closely together to execute the desired function.

We can also consider a Pod to be a self-contained, isolated ***logical host*** containing the systemic needs of the application it serves.

A Pod is meant to run a single instance of the application on the cluster. However, it is not directly recommended to create individual Pods. Instead, we create a set of identical Pods, called ***replicas***, to run your application. Such a set of replicated pods are created and managed by a ***controller***. Controllers manage their constituent Pods' lifecycle and can also perform ***horizontal scaling***, changing the number of Pods as necessary.

Although we might occasionally interact with Pods directly to debug, troubleshoot, or inspect them, we recommend using a controller to manage the Pods.

Pods run on ***nodes*** in the cluster. A Pod remains on its node until its process is complete**. The pod is evicted or deleted** from the node due to a lack of resources, or the node fails. If a node fails, Pods on the node are automatically scheduled for deletion.

Pod Lifecycle

Pods are ephemeral. They are not designed to run forever, and when a Pod is terminated, it cannot be brought back. Pods do not disappear until a user or a controller deletes them.

Pods do not repair themselves. For example, if a Pod is scheduled on a node that later fails, it is deleted. Similarly, if a Pod is evicted from a node for any reason, the Pod does not replace itself.

Each Pod has a ***PodStatus*** API object, which is represented by a Pod's status field. Pods publish their phase to the status phase field. The phase of a Pod is a high-level summary of the Pod in its current state.

When we run kubect1, get pod to inspect a Pod running on the cluster, a Pod can be in one of the following possible phases:

- **Pending:** Pod is created and accepted by the cluster, but one or more containers are not running. This phase includes time spent being scheduled on a node and downloading images.
- **Running:**Pod has been bound to a node, and all of the containers have been created. At least one container is running, is in the process of starting, or is restarting.
- **Succeeded:**All containers in the Pod have terminated successfully. Terminated Pods do not restart.
- **Failed:**All containers in the Pod have terminated, and at least one container has terminated in failure. A container "fails" if it exits with a non-zero status.
- **Unknown:**The state of the Pod cannot be determined.

How does a Pod Work?

Pods are created by workload resources called ***controllers***, which manage rollout, replication, and pods' health in the cluster. For example, if a node in the cluster fails, a controller detects that the pods on that node are unresponsive and create replacement pods (s) on other nodes. There are three most common types of controllers, such as:

- **Jobs** for batch-type are temporary and will run a task to completion.
- **Deployments** for applications that are stateless and persistent, such as web servers (HPPT servers).
- **Stateful Sets**for applications that are both stateful and persistent such as databases.

If a pod has multiple containers, they are all scheduled together on the same server in the cluster, whether VM or physical server. All containers in the pod share their resources and dependencies and can coordinate their execution and termination.

For example, pods can contain ***init*** containers that run before the application containers run, setting up the environment for the applications that follow.

Pods are almost always created by controllers, which can automatically manage the pod life cycle, including replacing failed pods, replicating pods when necessary, and evicting the pod from cluster nodes when they are complete no longer needed.

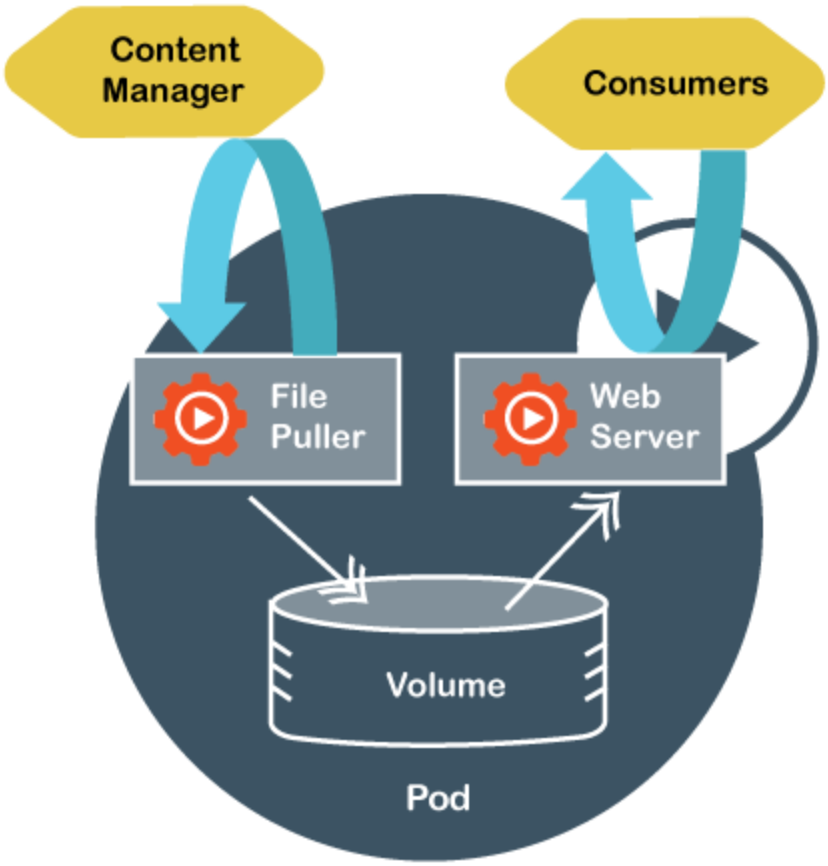
Controllers use information in a pod template to create the pods, and controllers ensure that running pods match the deployment defined in the pod template, for example, by creating replicas to match the number defined in the deployment.

How Pods Manage Multiple Containers

Pods are designed to support multiple cooperating processes (as containers) that form a cohesive service unit. The Pod containers 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.

For example, a container that acts as a web server for files in a shared volume, and a separate ***sidecar*** container that updates those files from a remote source, as shown in the following image:



Some Pods have to init containers as well as app containers. Init containers run and complete before the app containers are started.

Pods natively provide two kinds of shared resources for their constituent containers, such as:

1. **Networking:** Pods are automatically assigned unique IP addresses. Pod containers share the same network namespace, including IP addresses and network ports. Containers in a Pod communicate with each other inside the Pod on the localhost.
2. **Storage:** Pods can specify a set of shared storage volumes that can be shared among the containers.

Benefits of Pod

- When pods contain multiple containers, communications and data sharing between them are simplified. Since all containers in a pod share the same network namespace, they can locate and communicate via the localhost.
- Pods can communicate with each other by using another pod's IP address or referencing a resource that resides in another pod.
- Pods can include containers that run when the pod is started and perform initiation before the application containers run. Additionally, pods simplify scalability, enabling replica pods to be created and automatically shut down based on demand changes.