Kubernetes the Happy Path

# Install Kubernetes

# Kubernetes Overview

Kubernetes is an Orchestrator. It runs applications, can auto scale them, can self-healing.

The Kubernetes application has one or more control planes to manage the operation of Kubernetes. It has one or more Work Nodes which contain the applications to be run.

## Backup

Even though clusters are highly available they can fail and should be backed up so that a recent state of the cluster can be retrieved.

# The kubectl Command

The kubectl command is the main command line utility to query current state and apply new state requests to Kubernetes.

The kubectl command can be used for both the declarative and imperative command model described below.

# Specifying State

## Declarative

The preferred mode to specify state in Kubernetes is the declarative model. Commands to create, modify and remove resources are specified in YAML files. This has the advantage of preserving the current state of the Kubernetes configuration. The YAML files can be preserved in a version control system too so the history of configuration can be examined and state can be rolled back to a previous configuration if needed.

## Imperative

State can also be specified imperatively at the command line. This is more complicated to maintain than the declarative model and is not recommended for production.

# Clusters

Clusters run the Control Plan and Worker Nodes. They receive commands and pass them on to the Control Plane which may pass them on to a Worker Node.

# Control Plane

The control plan manages Kubernetes. It exposes the REST API, implements self-healing, schedules task, does scaling and other tasks.

Control planes communicate with worker nodes to give them commands to start pods, remove pods, scale pods and perform upgrades or rollbacks of Pod versions.

## Running Control Planes

For testing or development one control plane can be run. In production three to five control planes in different availability zones to provide high availability.

An odd number of control planes should be run because if a control plane goes down there will still be most control planes to provide a consensus for running commands. If there are an even number of control planes are run and half of them cannot be reached the remaining control planes cannot reach a consensus for running commands. The control plane consists of the API server, the cluster store and the controller manager and controllers.

## The API server

The API server is the handles all commands for Kubernetes. It handles all requests query and change the state of clusters.

The API server implements a RESTful API. All requests use HTTPS. All requests must be authenticated and checked for authorization.

## The cluster store

The cluster store stores the current state of the cluster. It uses the etcd distributed database. Most clusters run an etcd replica on each control plane for high availability. The etcd database is stateful unlike all other parts of Kubernetes.

The etcd database holds the desired state of the cluster. The observed state is the current state of the cluster. If the observed state does not match the desired state, the control plane issues commands to make the observed state the same as desired state. This can include starting new copies of failed Pods, doing a rollout or rollback of application versions, and scaling up or down the number of Pods.

## The scheduler

The kube-scheduler is the default scheduler and runs as part of the control plane. It is possible to implement a scheduler to replace kube-scheduler.

The scheduler examines nodes to find a node that can run a new pod. If node auto-scaling is enabled, it will create a new node if no eligible node is found.

Factors that need to be considered for scheduling decisions include individual and collective resource requirements, hardware / software / policy constraints, affinity and anti-affinity specifications, taints, data locality, inter-workload interference, and so on.

The scheduler first filters for available nodes as described in the previous paragraph. It then scores the eligible nodes based on active scoring rules. The node with the highest score is the one the new pod is assigned to.

If no eligible node is found the pod is in the unscheduled state and is listed as pending.

## The controller manager

The controller manager is responsible for starting and monitoring all controllers. If a controller fails a new one of the same type will be started.

## Controllers

Control planes run controllers which run as background watch loop to monitor pods and perform actions on them. Users can implement their own controllers to perform tasks not implemented by the default controllers. There are many built in controllers. Some of the common ones are listed below.

Each controller only monitors Pods it creates.

### Deployment

The Deployment controller is the most used controller. It monitors the health of Pods and starts new copies of the Pods for those that have crashed or are non-responsive. It provides auto scaling and no downtime rollouts of Pod upgrades and rollbacks of Pods to a previous version if a newly rolled out application version has a problem.

### Job

The Job controller starts a job running one or more Pods. It monitors the job and terminates when the job finishes. If needed it will start new copies of Pods if any fail.

### CronJob

The CronJob controller works like the Job controller but schedules a job to run at a certain time, date, day of week, day of month. Only one of these values must be set. The CronJob can be started and run repeatedly at the specified time.

The format for the CronJob scheduling is the same as the Linux cron utility.

### StatefulSet

The StatefulSet is similar to the Deployment controller, but each Pod maintains a unique identity for each Pod if a Pod needs to be restarted the new Pod will have the same identity. A StatefulSet provides guarantees about the ordering and uniqueness of Pods.

The StatefulSet controller can be used as part of the solution for maintaining persistent volumes to store data even when Pods fail or are added. This is often used for databases.

StatefulSet controllers can be used when any of the following are needed:

* Stable, unique network identifiers.
* Stable, persistent storage.
* Ordered, graceful deployment and scaling.
* Ordered, automated rolling updates.

### ReplicaSet

A ReplicaSet maintains a stable set of replica Pods always running. It can be used to specify a specific number of identical Pods.

The ReplicaSet identifies the Pods it owns through the *metadata.ownerReferences* field.

## Worker Nodes

Worker Nodes run the Pods. Each node has a Pod called the kubelet which monitors the application Pods and communicates with the control plane.

### The kubelet

The kubelet monitors the worker Pods and communicates with the control plane about various aspects of the Pod state. If a Pod fails it communicates with the control plane which will generally send back a command to start another copy of the Pod.

When instructed by the control plane it will perform rollouts or rollbacks. It will create a new Pod with the new version of the application, then kill the old Pod.

When the number of Pods is scaled up it will create new Pods. When the Pods are scaled down it will kill unneeded Pods.

### Container Runtime

Pods can run containers, VMs, WASM applications and others. Each worker node utilizes one or more runtimes. Runtimes by default use the *containerd* runtime to execute tasks.

### The kube-proxy

The kube-proxy implements cluster networking and load balances traffic to tasks running on the node.

# Kubernetes Parts

## Pods

Pods are the lowest level object in Kubernetes. They run containers such as those generated by containerd, Docker and CRI-O. There are other container runtimes. Some container runtimes are optimized for greater security or performance. It can run any container that supports the Container Runtime Interface. The containerd, Docker and CRI-O generated containers support this interface.

### Immutability

When a Pod is deployed its contents does not change. If a Pod fails a new unique copy of the Pod is started. When the Pod is scaled it starts unique copies of the Pods

If the Pod is upgrade to another version of the application or rolled back a new unique Pod is started with the new version and then the old Pod is killed.

### Pod Contents

Pods generally run one container but can run multiple containers. If they run multiple containers, it should have one main container running the application. The Pod can also contain a sidecar container performing some helper function such as monitoring a source code repository for changes and downloading them to update the source for a web application for instance. Sidecars can support other tasks. Each container can run one application. A Pod can also run an initialization container that runs once when a Pod starts up. Initialization Containers can do tasks like retrieving configuration information from a repository or retrieving the source for a web application.

Pods should only run one application and any supporting sidecar containers, if needed. It can also run an initialization container.