***KUBERNETES NOTES***

Kubernetes (also written as **K8s**)

**What is Kubernetes???**

open-source **container-orchestration** system

Kubernetes follows the **master-slave** architecture.

**Used for:**

**automating deployment, scaling and management** of containerized applications across clusters of host

**Origin:**

designed by Google and is now maintained by the Cloud Native Computing Foundation.

works with a range of container tools, including **Docker.**

Kubernetes defines a set of **building blocks**

which collectively provide mechanisms for deploying, maintaining, and scaling applications.

The components that make up Kubernetes are designed to be **loosely coupled** and extensible to meet a wide variety of different workloads.

This extensibility is provided in large part by the **Kubernetes API**, which is used by internal components as well as extensions and containers running on Kubernetes.

**Components of Kubernetes**

**Kubectl**

Kubectl is the command line interface used to communicate to the kubernetes API server

**Pods:** The basic scheduling unit in Kubernetes is called a "pod". Pod is nothing but a wrapper for container

Each pod in Kubernetes is assigned a unique IP address within the cluster, which allows applications to use ports without the risk of conflict.

Pods can have network configuaration and volumes

**Labels and selectors**: Labels and selectors are the primary grouping mechanism in Kubernetes, and are used to determine the components to which an operation applies.

**Controllers**

Used for managing a set of pods.

**types of controller**

"**Replication Controller**," which handles replication and scaling by running a specified number of copies of a pod across the cluster. It also handles creating replacement pods if the underlying node fails.

"**DaemonSet Controller**" for running exactly one pod on every machine (or some subset of machines),

"**Job Controller**" for running pods that run to completion, e.g. as part of a batch job.

The set of pods that a controller manages is determined by label selectors that are part of the controller’s definition.

**Services**

A Kubernetes service is a set of pods that work together, such as one tier of a multi-tier application.

The set of pods that constitute a service are defined by a **label selector**.

**Service Discovery and routing**

Kubernetes provides service discovery and request routing by **assigning a stable IP address and DNS** name to the service, and load balances traffic in a **round-robin** manner to network connections of that IP address among the pods matching the selector (even as failures cause the pods to move from machine to machine).

By default a service is **exposed inside a cluster** (e.g. back end pods might be grouped into a service, with **requests from the front-end** pods **load-balanced** among them),

but a service can also be exposed outside a cluster (e.g. for clients to reach frontend pods).

The **Kubernetes Master** is the main controlling unit of the cluster that manages its workload and directs communication across the system.

The **Kubernetes control plane** consists of various components, each its own process, that can run both on a single master node or on multiple masters supporting high-availability clusters. The various components of Kubernetes control plane are as follows:

**etcd**

etcd is a persistent, lightweight, distributed, key-value data store developed by CoreOS that reliably stores the configuration data of the cluster, representing the overall state of the cluster at any given point of time. Other components watch for changes to this store to bring themselves into the desired state.

**API server**

The API server is a key component and serves the Kubernetes API using JSON over HTTP, which provides both the internal and external interface to Kubernetes. The API server processes and validates REST requests and updates state of the API objects in etcd, thereby allowing clients to configure workloads and containers across Worker nodes.

**Scheduler**

The scheduler is the pluggable component that selects which node an unscheduled pod (the basic entity managed by the scheduler) should run on based on resource availability. Scheduler tracks resource utilization on each node to ensure that workload is not scheduled in excess of the available resources. For this purpose, the scheduler must know the resource requirements, resource availability and a variety of other user-provided constraints and policy directives such as quality-of-service, affinity/anti-affinity requirements, data locality and so on. In essence, the scheduler’s role is to match resource "supply" to workload "demand".

**Controller manager**

The controller manager is the process in which the core Kubernetes controllers like DaemonSet Controller and Replication Controller run. The controllers communicate with the API server to create, update and delete the resources they manage (pods, service endpoints, etc.).

**Kubernetes node**

The Node, also known as Worker or Minion, is a machine where containers (workloads) are deployed. Every node in the cluster must run a container runtime such as Docker, as well as the below-mentioned components, for communication with master for network configuration of these containers.

**Kubelet**

Kubelet is responsible for the running state of each node, ensuring that all containers on the node are healthy. It takes care of starting, stopping, and maintaining application containers organized into pods as directed by the control plane.

Kubelet monitors the state of a pod and if not in the desired state, the pod will be redeployed to the same node. The node status is relayed every few seconds via heartbeat messages to the master. Once the master detects a node failure, the Replication Controller observes this state change and launches pods on other healthy nodes

**Container**

A container resides inside a Pod. The container is the lowest level of a micro-service which holds the running application, the libraries and their dependencies. Containers can be exposed to the world through an external IP address.

**Kube-proxy**

The Kube-proxy is an implementation of a network proxy and a load balancer, and it supports the service abstraction along with other networking operation. It is responsible for routing traffic to the appropriate container based on IP and port number of the incoming request.

**cAdvisor**

cAdvisor is an agent that monitors and gathers resource usage and performance metrics such as CPU, memory, file and network usage of containers on each node.