* [Kubernetes](https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/), also known as **K8s**.
* It is an **open-source** system for **automating deployment, scaling, and management of containerized applications**.
* **Container orchestration**
* Developed by **google**
* Configuration information is stored in a JSON format but is most often written in YAML. Kubernetes agents convert the YAML to JSON prior to persistence to the database.
* Kubernetes is written in Go Language

# Feature

* High Availability | No Down time
* Scalability | High Performance
* Disaster Recovery | provide backup and restore

# Architecture

A Kubernetes cluster is made of a master node and a set of worker nodes. The cluster is all driven via API calls to controllers, both interior as well as exterior traffic.

Diagram

Description automatically generated

# Master Node

* The Kubernetes master runs various server and manager processes for the cluster. Among the components of the master node are **the kube-apiserver, the kube-scheduler, and the etcd database.**
* As the software has matured, new components have been created to handle dedicated needs, such as the **cloud-controller-manager**; it handles tasks once handled by the kube-controller-manager to interact with other tools, such as Rancher or DigitalOcean for third-party cluster management and reporting.
* There are several add-ons which have become essential to a typical production cluster, such as DNS services..

# Worker Nodes

* All worker nodes run the **kubelet** and **kube-proxy**, as well as the **container engine**, such as Docker or rkt. Other management daemons are deployed to watch these agents or provide services not yet included with Kubernetes.
* The kubelet interacts with the underlying Docker Engine also installed on all the nodes, and makes sure that the containers that need to run are actually running. The kube-proxy is in charge of managing the network connectivity to the containers. It does so through the use of iptables entries. It also has the userspace mode, in which it monitors Services and Endpoints using a random port to proxy traffic and an alpha feature of ipvs.
* Supervisord is a lightweight process monitor used in traditional Linux environments to monitor and notify about other processes. In the cluster, this daemon monitors both the kubelet and docker processes. It will try to restart them if they fail, and log events.
* Kubernetes does not have cluster-wide logging yet. Instead, another CNCF project is used, called Fluentd. When implemented, it provides a unified logging layer for the cluster, which filters, buffers, and routes messages

# 3 Node process

1. Kubelet:

Kubelet interact with both the container and node

Kubelet starts the pod with a container inside

1. Kube Proxy:

Kube Proxy forwards the request

1. Container Runtime

# kube-apiserver

* The kube-apiserver is central to the operation of the Kubernetes cluster.
* All calls, both internal and external traffic, are handled via this agent. All actions are accepted and validated by this agent, and it is the only connection to the etcd database. As a result, it acts as a master process for the entire cluster, and acts as a frontend of the cluster's shared state.

kube-scheduler

* The kube-scheduler uses an algorithm to determine which node will host a Pod of containers. The scheduler will try to view available resources (such as volumes) to bind, and then try and retry to deploy the Pod based on availability and success.
* There are several ways you can affect the algorithm, or a custom scheduler could be used instead. You can also bind a Pod to a particular node, though the Pod may remain in a pending state due to other settings.
* One of the first settings referenced is if the Pod can be deployed within the current quota restrictions. If so, then the taints and tolerations, and labels of the Pods are used along with those of the nodes to determine the proper placement.

etcd Database

* The state of the cluster, networking, and other persistent information is kept in an etcd database, or, more accurately, a b+tree key-value store. Rather than finding and changing an entry, values are always appended to the end. Previous copies of the data are then marked for future removal by a compaction process. It works with curl and other HTTP libraries, and provides reliable watch queries.
* Simultaneous requests to update a value all travel via the kube-apiserver, which then passes along the request to etcd in a series. The first request would update the database. The second request would no longer have the same version number, in which case the kube-apiserver would reply with an error 409 to the requester. There is no logic past that response on the server side, meaning the client needs to expect this and act upon the denial to update.
* There is a master database along with possible followers. They communicate with each other on an ongoing basis to determine which will be master, and determine another in the event of failure. While very fast and potentially durable, there have been some hiccups with new tools, such as kubeadm, and features like whole cluster upgrades

kubelet

* The kubelet agent is the heavy lifter for changes and configuration on worker nodes. It accepts the API calls for Pod specifications (a PodSpec is a JSON or YAML file that describes a pod). It will work to configure the local node until the specification has been met.
* Should a Pod require access to storage, Secrets or ConfigMaps, the kubelet will ensure access or creation. It also sends back status to the kube-apiserver for eventual persistence. ​

Other Agents

* The kube-controller-manager is a core control loop daemon which interacts with the kube-apiserver to determine the state of the cluster. If the state does not match, the manager will contact the necessary controller to match the desired state. There are several controllers in use, such as endpoints, namespace, and replication. The full list has expanded as Kubernetes has matured.
* In alpha since v1.8, the cloud-controller-manager interacts with agents outside of the cloud. It handles tasks once handled by kube-controller-manager. This allows faster changes without altering the core Kubernetes control process. Each kubelet must use the --cloud-provider-external settings passed to the binary.

Pods

* The whole point of Kubernetes is to orchestrate the lifecycle of a container. We do not interact with particular containers. Instead, the smallest unit we can work with is a Pod. Some would say a pod of whales or peas-in-a-pod. Due to shared resources, the design of a Pod typically follows a one-process-per-container architecture.
* Containers in a Pod are started in parallel. As a result, there is no way to determine which container becomes available first inside a pod. To support a single process running in a container, you may need logging, a proxy, or special adapter. These tasks are often handled by other containers in the same pod.
* There is only one IP address per Pod. If there is more than one container, they must share the IP. To communicate with each other, they can either use IPC, or a shared filesystem.
* While Pods are often deployed with one application container in each, a common reason to have multiple containers in a Pod is for logging. You may find the term sidecar for a container dedicated to performing a helper task, like handling logs and responding to requests, as the primary application container may have this ability.

Labels

* Nametags to identify things.

# Pod

* Smallest Unit of K8s
* Abstraction layer over a container
* Runnable unit of work
* Usually, 1 application per pod it can handle multiple containers.
* Each Pod get its own unique IP address not the container, and a new IP is generated on each time of recreation (When Pod die or restart). (Which is inconvenient & service resolved this issue)

Replication controllers

* Create multiple pods

Service

* Used to connect outside replication controller
* Also, a load balancer
* Used to provide communication between Pods or with outer world or browser.
* Provide permanent/static IP address which is attach to each pod.
* Lifecycle of Pod and Service are not connected (If pod dies service and IP address remains)
* Two Types- External Service(open to public) and Internal service(not available to public)

Namespace

* to provide a degree of isolation

Secret

* Used to store non-public information such as token, certificate, password.
* Secret can attach to pod at runtime so that sensitive data can stored securely in the cluster.
* The build-in security mechanism is not enable by default.
* Stored in base 64 encoding rather than plain text.

# Volume

* Provide Physical storage on hard drive or cloud storage.
* Data Persistence

# Ingress

* To provide a specific URL
* To route traffic into cluster

# Config Map

* External Configuration(URL, database or other services) of app

# Deployment

* Blueprint for pods
* Used for stateless apps
* Database won’t be replicate through deployment because it has state.
* Deployment for stateless apps

# Stateful Set

* Used for stateful apps like database

*Ref = https://medium.com/google-cloud/kubernetes-101-pods-nodes-containers-and-clusters-c1509e409e16*