Kubernetes

- production grade container orchestration system. It's similar to docker.
- Open source designed to automate, deploying and scaling and operation of containerized applications.
- Distributed system. Multiple machines are configured to form a cluster.
- Schedule containers to run on different machines
- Can Move containers as machines are added or removed.
- Can use different container run times, container run time agnostic, so can use different container run times. This adapbility is due to its modular design.
- Uses declarative configuration for everything to abstract everything.
- Allows us to quickly deploy containers, wire up networking and scale and expose services to the real world
- Can move containers from failing machines to running machines, thus saving the container. Thus automatically recover from machine failures in the cluster.
- Built in support for machine maintence.
- Multiple clusters can join up and form a federation this is beneficial for reduncancy, as in if one cluster dies, containers will automatically migrate to the other cluster
- Automated deployment and rollback features
- Seamless horizontal scaling
- Secret management
- service discovery and load balance
- Linux and windows container support
- simple log collection
- Stateful application support
- Persistent volume management
- CPU and memory quotas
- Batch job processing
- Role-base access control
- Similar to docker swarm

Deploying kubernetes

- To support linux containers you need Linux nodes running in the cluster. If you want windows containers you also need Windows nodes running in the cluster.

Kubernetes architecture

- Kubernetes introduces its own dialiect to the orchestration space
- Clusters are composed of nodes. The term cluster refers to all the machines collectively and can be thought of one monolithic system. Nodes are machines in the cluster. A node can be a VM or a physical machine. Nodes are categorized as workers or master nodes.
- Worker nodes include software to run containers managed by the kubernetes control plane
- Master nodes run the control plane. The control plane is a set of APIs and software that kubernetes users interact with. These APIs are referred to as master components.
- The control plane schedules containers onto nodes.Scheduling decisions consider required CPU and other factors. Scheduling refers to the decision process of placing containers onto nodes.
- Pods are a term used to describe a group of containers, containers are grouped into pods. Thus a pod has one or more containers. All containers inside a pod run on the same node. Pods are the smallest building block in kubernetes. More complex and useful abstractions built on top of pods.

- Services define networking rules for exposing groups of pods to other pods or to the internet.
- Deployments are used to manage deploying configuration changes to running pods. As well as used for horizontal scaling.

Interacting with Kubernete clusters

The way you retrieve and modify state information in the cluster is by sending requests to the API server. The API server is the mater component that acts as a front end for the cluster.

The first method of interacting directly with kubernetes is through REST API. It is possible but not common to work directly with the API server. However the use case arises when using a programming language with no client library for kubernetes.

The second method of interacting with kubernetes is through client libraries. Which handle authenticating managing individual REST API requests and responses. Kubernetes maintains official client libraries for various languges. And has community created client libraries as well for langauges which are not officially supported

The third way is working with the command line tool: kubectl. We can issue high level commands that are translated into Rest api calls. Works with local and remote clusters.

- Kubectl create – to create resources such as pods, services etc.
- Kubectl delete – delete resources.
- Kubectl get – get a list of resources of a given type such as get pods.
- Kubectl describe – print detailed info about a resource. Such as describe pod server , which gives information about the pod named server.
- Kubectl logs – to print container logs

When issuing commands, we are sending them to the API master node server which acts as the front end to the cluster. Which then manages the nodes running pods.

The fourth way is interacting through the web interface dashboard.

Pods

Pods are the basic building block in kubernetes. Pods contain one or more containers. Pod containers all share a container network. That allows pods to communicate with each other. Regardless of the node the pods are running on. Each pod gets a single IP address. One IP address per pod.

Pod declaration:

- Container image
- Container ports
- Container restart policy
- Resource limits

All the desired properties are written in Manifest files. Where we declare desired properties and can describe all kinds of resources.

Manifests in action

kubctl create – sends manifest Kubernetes API server. The API server does the following for pod manifests:

- select a node with sufficient resources
- schedule pod onto node
- node pulls Pod’s container image
- and then starts Pod’s container.

Why use manifests

- can check into a source control system to track history and rollback when required. Also makes it easier to share work. And easier to work with manifests. Compared to string a sequence of commands. I would guess they are the equivalent to docker files in a way. But for describing any resource, not just pods (being groups of containers)