Kubernetes Principles of Operation

Overview of the major concepts.

Kubernetes from 40K feet

Kubernetes as a Cluster

K8s is like any other cluster, a bunch of machines to host apps. Kubernetes cluster is made of a control plane and worker nodes.

- Control plane
    - Exposes the API
    - Scheduler for assigning work
    - records the state of the cluster and apps in a persistent store
- Worker Node: Where the app runs

Kubernetes as a Orchestrator

1. Start with an app
2. Package it as a container
3. Give it to the cluster (Kubernetes)

    Already control plane nodes implement the cluster intelligence and worker nodes are where user applications run

    1. Design and wirte the application as small independent microservices in your favorites languages
    2. Package each microservice as its own container
    3. Wrap each container in a Kubernetes Pod
    4. Deploy pods to the cluster via higher level controllers such a devployments, demonSets, StatefulSets, Cronjobs, etc

At highter level, kubernetes has several controllers that augment Pods.

How it works

- The cluster
    - One or more **control plane node**
    - a bunch of worker nodes

1. Design and write the app as small independent microservices
2. Package each microservice as its own container
3. Wrap each container in a **Kubernetes Pod**
4. Deploy Pods to the cluster via higher level controllers)Deployments, DaemonSets, StatefulSets, CronJobs)

- Kubernetes manage applications declaratively (a set of config file)

Control Plane and worker nodes

CP and WN are Linux Hosts: bare metal servers or instances in a private or public cloud, ARM or IoT devices.

Control Plane

Kubernetes Control Plane Node runs a collection of system services that make up the control plane of the cluster.

- Called Heads or Head Nodes.

Simple setups run a single control plane node. For prod envs, multiple CP nodes are configured. 3 or 5 is recommended.

Good practice: not run user apps on CP nodes. This frees to concentrate entirely on managing the cluster

API Server

API Server is the Grand Central Station of Kubernetes.

- ALL COMMUNICATION between all components must go through API server

It exposes RESTful API to POST YAML configuration files to over HTTPS, sometimes called **manifests**.

The Manifests describes desired app state: container image, ports to expose, Pod replicas to run, etc.

Requests to the API server are subject to authentication and authorization checks.

Cluster Store

The only **stateful** part of the control plane.

- Persistently stores the entire configuration and state of the cluster
- Based on **etcd** distributed database. 3-5 etcd replicas for high availability
- Default config stores a replica of the cluster store on every control plane node and automatically configures HA.

Controller Manager and Controllers

Controller Manager implements all background controllers that monitor cluster components and **respond to events**

- The CM is a controller of controllers
  - Deployment Controller
  - StatefulSet Controller
  - ReplicaSet Controller
- GOAL OF EACH CONTROLLER: ensure the **observed state** matches **desired state**

1. OBtain desired state
2. Observe current state

3. Determine differences
4. Reconcile differences

Scheduler

Scheduler watches the API server for new work tasks and assigns them to appropiate healthy worker nodes.

When identifying nodes capable of running a task, performs various predicate checks.

- affinity rules
- required network ports availables on the node
- sufficient available resources
- etc

If does not find a suitable node, the task is not scheduled and gets marked as **pending**

Cloud Controller Manager

Control Plane will be running a **cloud controller manager** to facilitate integrations with cloud services.

- Instances
- Load balancers
- storage

Worker Nodes

Worker nodes are where user app runs.

1. Watch the API server for new work assignments
2. Execute work assignments
3. Report back to the control plane via API Server

Kubelet

- Main Kubernetes Agent
- Runs on every worker node
- When you join a node to a cluster, the process installs the kubelet
- Responsible for registering it with the cluster
- Registers CPU, memory, storage into the cluster pool
- Watch the API server for new work tasks
- **can not execute a tasks/ Simply reports back to the control plane**

Container Runtime

- Performs container-related tasks like pulling images, start or stop containers
- CRI (Container Runtime Interface)
- containerd (container-dee) is the container supervisor and runtime logic stripped out from Docker.
- Donated by Docker

Kube-proxy

- Runs on every node
- Responsible for local cluster networking
- Ensures each node gets its own unique IP Address
- Implements local iptables
- handle rounting
- handles load balancing and traffic on the Pod network

Kubernetes DNS

- Kubernetes has an internal DNS service
- Cluster's DNS service has a static IP address that is hard-coded into every Pod on the cluster.
- Service registration is automatic
- Based on CoreDNS project

Packagking Apps for Kubernetes

An application needs:

1. Packaged as a container
2. Wrapped in a Pod
3. Deployed via a declarative manifest file

- You write an app microservice
- You built it into a container image and store in a registry -> containerized
- You define a Kubernetes Pod to run the containerized app
    - a **Pod** is a wrapper that allows a container to run on a Kubernetes Cluster.
    - \ The preferred model is to deploy all Pods via highter-level controllers
        - Most common controller is Deployment.
        - Deployment Offers scalability, self-healing, rolling updates for stateless apps
        - Deployments are definedd in YAML manifest files.
- Once defined in Deployment YAML file, you can use Kubernetes command-line tool to post it to the API server as the desired state of the app.

Declarative model and desired state

The declarative model and the concept of desired state are at the very heart of Kubernetes.
Kubernetes declarative model works like:

1. Declare the **desired state of an app microservice in a manifest file**
2. Post it to the API server
3. Kubernetes stores it in the cluster store as the application's desired state
4. Kubernetes implements the desired state on the cluster
5. A controller makes sure the observed state of the application does not vary from the desired state

- Manifest files are written in a YAML and tell kubernetes what an app sohuld look like -> **desired state**.
    - Which image to use, replicas, network ports, updates, etc

- Post to the API via `kubectl` utility via HTTP POST (usually port 443)
- Kubernetes authenticates and authorizates the request. Inspects the manifest, identifies which controller to send, record the config in the cluster store.
- Schedules the work to worker nodes where kublet co-coordinates the work
- Controllers run as background reconciliation loops that constantly monitor the state of the cluster

Pods

Atomic Units:

- VMware: Virtual Machine
- Docker: Container
- Kubernetes: Pod

1. Kubernetes demands that every container runs inside a Pod.
2. Pods are objects in the Kubernetes API.

Pods & Containers

Pod therm comes from a pod of whales. The simplest: run a single container in every Pod. There are advanced user-cases that run multiple containers in a single Pod. The point **Kubernetes Pod is a construct for running one or more containers.**

Anatomy

- Multiple containers in a Pod: there all share Pod environments
    - network stack
    - volumes
    - IPC namespace
    - shared memory
    - same IP address
    - etc
- If two containers in the same opd neet to talk to each other they can use the Pod's localhost interface.

__Multi container pods are ideal for tightly coupled containers that may need to share memory and storage.__Put each container in a single pots keeps things clean but increment the traffic between pods.

To scale an app do not scale by adding more containers to existing pods. Multi-Container pods are ONLY for situations where complementary containers need to **share resources**.

- atomic ops: Deployment of a Pod is an atomic operation. A single Pod ONLY can be scheduled to a single node. Pod is only ready for service when all its containers are up and running.

Lifecycle

Pods are mortal. If a Pod dies unexpectedly, Kubernetes starts a new one in its place. New pods feels like the old one, but with a new ID and IP address.

You have to desgin the apps kowning this implications.

Immutability

Can't be changed once the are running. To change a config, you replace it with a new one running the new configuration.

Deployments

The Pod deployment is via higher level controllers as **Deployments, DaemonSets, StatefulSets.**

Deployment is a Kubernetes object that wraps around a Pod and adds features such self-healing, scaling, zero-downtime rollouts, and versioned rollbacks. Controllers run as watch loops observing the cluster, matching desired and observed states.

Service Objects and Networking

Rollouts and scaling ops replace old Pods with new ones with new IPs. *Services* provide relaiable networking for a set of Pods. Service is a Kubernetes API object, privdes reliable name and IP and load balancing to the microservices Pods behind it.

like Pods and Deployments, Services are a fully-fledged object in the Kubernetes API. They have a front-end consisting of stable DNS name, IP address, and port. On the backend, they load-balance traffic across a dynamic set of Pods.

Service is a stable network abstraction point that provides TCP and UPD load-balancing across a dynamic set of Pods. So they don;t poseess application intelligence. The cannot provide application-layer host, and path routing.

Ingress understands HTTP and provides host and path-based routing.