Kubernetes Workshop 

Overview

What is Kubernetes?

- container orchestration tool
- developed by Google
- manage containerized applications in different deployment environments (cloud, physical, VMs, etc.)

Why Kubernetes?

- microservices in containers
- 100s - 1000s of containers for an application
  - management can be hard! → container orchestration

What do orchestration tools offer?

- high availability (no downtime)
- scalability (scale up or down depending on load from users)
- disaster recovery (backup, restore)

Kubernetes Architecture

Node

- virtual or physical machine

Cluster

- made up of at least one **master node** (aka Control Plane Node)

    - runs K8s processes to manage cluster (typically in a 🐳 container)
        - 🐳 API Server (api) = entry point to K8s cluster
            - UI (K8s dashboard), API, CLI
        - 🐳 Controller Manager (c-m) = keeps track of whats happening in cluster
        - 🐳 Scheduler (sched) = ensures Pods replacement based on available resources & workload
        - etcd = Kubernetes backing store ([key, value] storage) - etcd snapshots used for backup/restore
    - in production environments usually at least 2 master nodes (for backup in case first master node is down)
- **worker nodes** (typically just called nodes), each node runs a kubelet process (kubelet allows for node communication)
    - container(s) running on each worker node
    - where you applications are running ⇒ majority of the workload, bigger and more resources
- **virtual network** spans all nodes that are part of the cluster, creates one unified machine

Main Kubernetes Components

Pod (pod)

- smallest unit in K8s
- abstraction over a container (running environment/layer over container)
    - only interact with K8s layer
- usually 1 application per Pod
- each Pod gets its own internal IP address
    - used to communicate with each other

- **pods are ephemeral** (can die easily)
  - run out or resources, crash, etc.
  - new IP address on re-creation ⇒ service

Service (svc)

- permanent IP address that can be attached to each pod (each pod gets its own service)
  - lifecycle of pod and service not connected
- external service - service that opens communication from external sources (like web browsers)
  - http://124.89.101.2:8080 → http://[node ip address (not service)]:[port of service]
- internal service for things like databases
  - default type
- also used to connect to cloned pod (no downtime)
  - load balancer - service will forward request to least-busy pod
  - we define a blueprint for pods that specifies how many replicas we want to have (Deployment)

Ingress (ing)

- request goes to ingress which forwards to service (route traffic into cluster)
- for custom URLs
  - https://my-app.com

ConfigMap (cm)

Database URL usually in built application, but instead:

ConfigMap ⇒ external configuration of your application

- connect to pod
- if you change endpoint of service, just adjust ConfigMap (no need to build new image)
- non-confidential data only

Secret (secret)

- like cm but used to store secret data (base64 encoded format)
  - built-in security mechanism not enabled by default
  - should use 3rd party encryption
  - credentials
- reference Secret in Deployment/Pod with environmental vars, properties file, etc.

Volume (vol)

- attach physical storage on hard drive to pod
  - local machine, remote storage (outside of K8s cluster) (cloud storage)
- ensures data is retained if database pod is re-created
  - K8s does not manage data persistence

Deployment (deploy)

- blueprint for “my-app” (user interacted/stateless) pods
- you create Deployments (abstraction of Pods) - in practice you mostly work with Deployments, not Pods
- scale up or scale down how many replicas you need
- cannot replica database (DB) pod since it has a state ⇒ StatefulSet

StatefulSet (sts)

- for STATEFUL apps (MySQL, elastic, mongoDB, etc.)
- synchronizes read/writes to DB
- harder so DB are often hosted outside of K8s cluster

K8s Configuration

all config requests go through API Server (YAML or JSON)

- config requests are declarative
  - is == should (Controller Manager checks if *desired state* == *actual state*)

3 parts to every config file

1. metadata
2. specification (spec)
3. status (automatically generated + added by K8s) - desired vs. actual state
   a. state updated continuously
   b. status data retrieved from etcd

Format of YAML Configuration Files

syntax: strict indentation

store config file with your code (or own git repo)

Minikube and Kubectl

Minikube

- virtual node - one node cluster where master node and worker node processes run on one node (for testing locally)
    - good when we might not have enough resources to run a normal cluster on one machine for local development
- Docker pre-installed
- can run as a container or virtual machine on your machine

Kubectl

- CLI (most powerful of all 3 clients - UI, API, Kubectl)
- interact with cluster
- works for cloud cluster as well (not just Minikube cluster)

Installation

minikube start

minikube is local Kubernetes

https://minikube.sigs.k8s.io/docs/start/

or for Macs:

```
> brew install minikube
```

may need to install Docker (to use as a driver)

also installs kubectl as dependency

Commands/Setup

```
> minikube start --driver docker
```

```
> minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

```
> kubectl get node
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   75m   v1.26.1
```

Demo Project

Overview

We will create a web application that uses mongoDB as a backend, all deployed in a K8s cluster

From: https://www.youtube.com/watch?v=s_o8dwzRlu4

More Kubernetes: https://www.youtube.com/watch?v=X48VuDVv0do&t=1s

Links We Will Use

K8s Docs - https://kubernetes.io/docs/home/

Docker Image for web app - https://hub.docker.com/repository/docker/nanajanashia/k8s-demo-app

K8s Config Files

1. ConfigMap - MongoDB Endpoint
2. Secret - MongoDB User/Password

3. Deployment & Service - MongoDB Application with internal Service (typically group Deployment and Service since Deployments need Services)
4. Deployment & Service - Our own webapp with external Service

mongo-secret.yaml

```
> echo -n mongouser | base64
bW9uZ291c2Vy
```

mongo.yaml

Deployment:

- Under `spec`, `template` = configuration for Pod
- `label` - key/value pairs that are attached to K8s resources (any K8s component can have a label)
  - additional component identifier for users (make meaningful)
  - do not provide uniqueness (Pod replicas will have same label but unique name)
  - required field for Pods, optional else but good idea
  - “app: key” is the standard but can use whatever you want instead of “app”
- `matchLabels` tells us which Pods (with specific `label`) belong to this Deployment

Service:

- `selector` for forwarding requests to specific Pods

When mongoDB starts, auto generates username and password

references like variables and arguments with functions

External Service

add `type` under spec to `webapp.yaml`

Deploy to Minikube Cluster

```
> kubectl get pod
No resources found in default namespace.
```

ConfigMap and Secret must exist before Deployments

```
> kubectl apply -f mongo-config.yaml
```

```
> kubectl get all
```

```
> kubectl get configmap|secret
```

To access webapp on browser:

```
> minikube ip

or

> kubectl get node -o wide
```

but… our network is limited

```
> minikube service list
> minikube service webapp-service
```