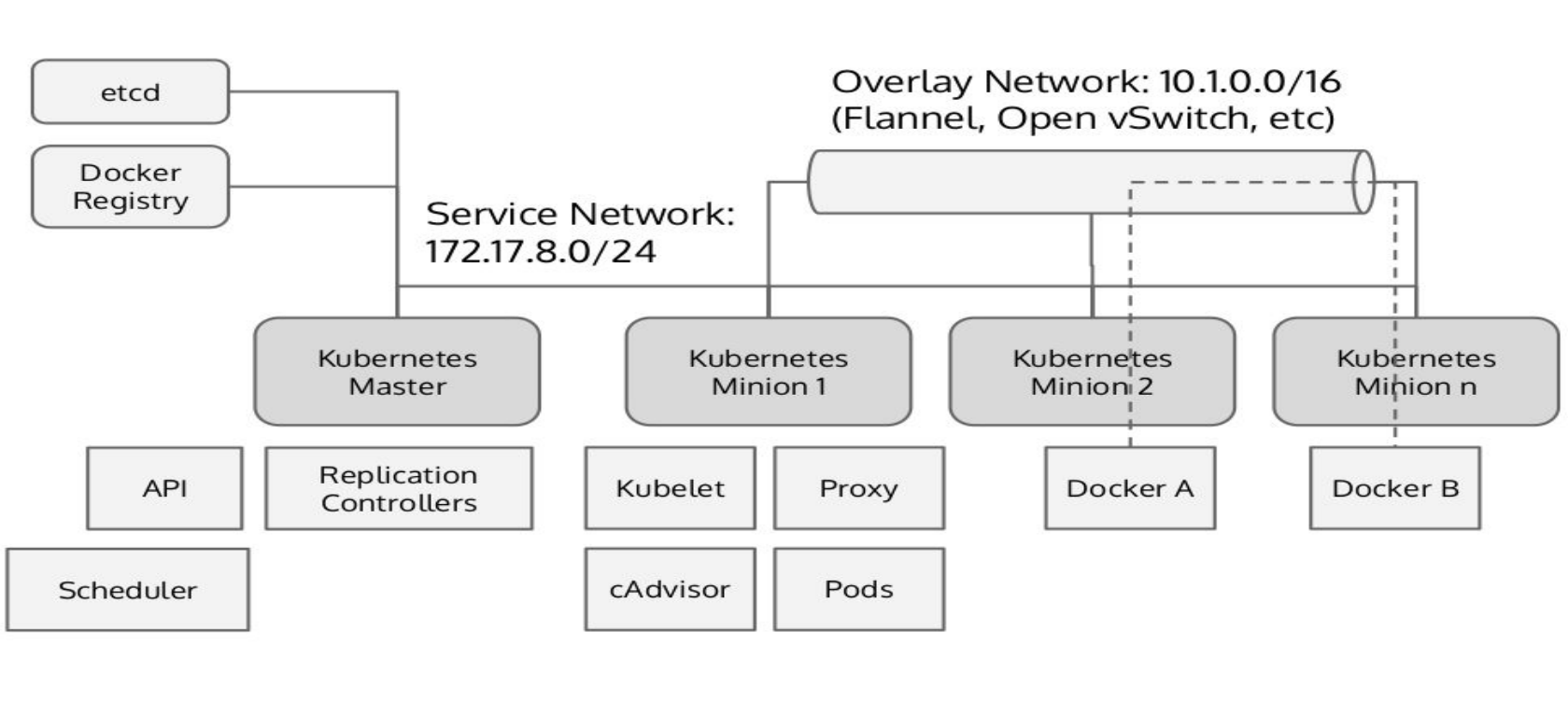
Introduction to Kubernetes

Kubernetes

- Kubernetes is a platform for hosting Docker containers in a clustered environment with multiple Docker hosts
- Provides container grouping, load balancing, auto-healing, scaling features
- Project was started by Google
- Contributors == Google, CodeOS, Redhat, Mesosphere, Microsoft, HP, IBM, VMWare, Pivotal, SaltStack, etc

Kubernetes Architecture



Terminology

- Clusters
- Pods
- Replication controllers/Replica set
- Services
- Labels
- Namespaces
- Configmaps
- Secrets
- Daemonsets
- Deployments

Clusters

A set of resources where pods are deployed managed and scaled

- Network
- Compute
- storage

Pods

Pods are colocated groups of application containers

- Smallest unit in K8S
- Can contain multiple containers
- Only run standalone pods for jobs
- Long running applications should be managed by replication controllers
- Docker images of one pod run on one machine
- Single IP per pod

Replication controllers/Replica set

Replication controllers ensure that a specific number of pods are running on the cluster

- Based on a template which can contain multiple pods
- Can use labels for grouping

Replica sets are the Next Gen replication controller

- Only difference is the ability to select based on set rather than single label

Services

Services deliver cluster wide service discovery and basic load balancing

- Provide persistent name or address for pods
- Can use a single or multiple sets of labels

Labels

Used to organize and select group of objects such as pods based on Key/Value pairs

Namespaces

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces.

- Used to divide cluster resources between multiple uses
  - For example: dev, staging, production etc..
- Not to be confused with labels

Configmaps

Configmaps are a way to decouple data from image content

- Used to keep kubernetes data decoupled from images
- Can be used to store simple information
- But can even contain full configiration or json files

Secrets

objects of type secret are intended to hold sensitive information

- Passwords
- oauth tokens
- Ssh keys

Deployments

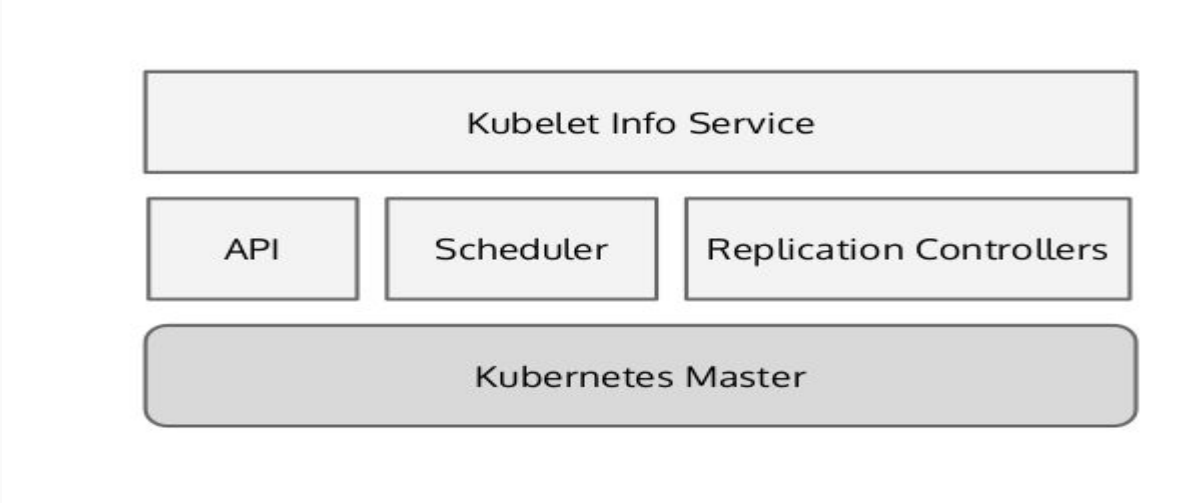
A Deployment provides declarative updates for Pods and Replica Sets

- Create a Deployment to bring up a Replica Set and Pods
- check the status of a Deployment to see if it succeeds or not.
- Later, update that Deployment to recreate the Pods (for example, to use a new image)
- Rollback to an earlier Deployment revision if the current Deployment isn't stable
- Pause and resume a Deployment

Control plane components

- Etcd
- Api server
- Scheduler
- Controller manager

Kubernetes master - Control plane



etcd

Etcd is a consistent value store

- Used in K8s for shared configuration and service discovery
- Focuses on being simple, fast and reliable
- Uses Raft consensus algorithm which allows fault tolerance and HA
- **All persistent cluster state is stored in etcd**

Api server

Is responsible for serving the Kubernetes API and proxying cluster components such as the web UI

- Exposes a REST interface which allows creating pods, services and updating the corresponding objects in etcd
- The only component that talks directly to etcd
- Horizontally scalable by running more api servers

Scheduler

Watches the apiserver for unscheduled pods and schedules them onto healthy nodes based on resource requirements

Controller manager

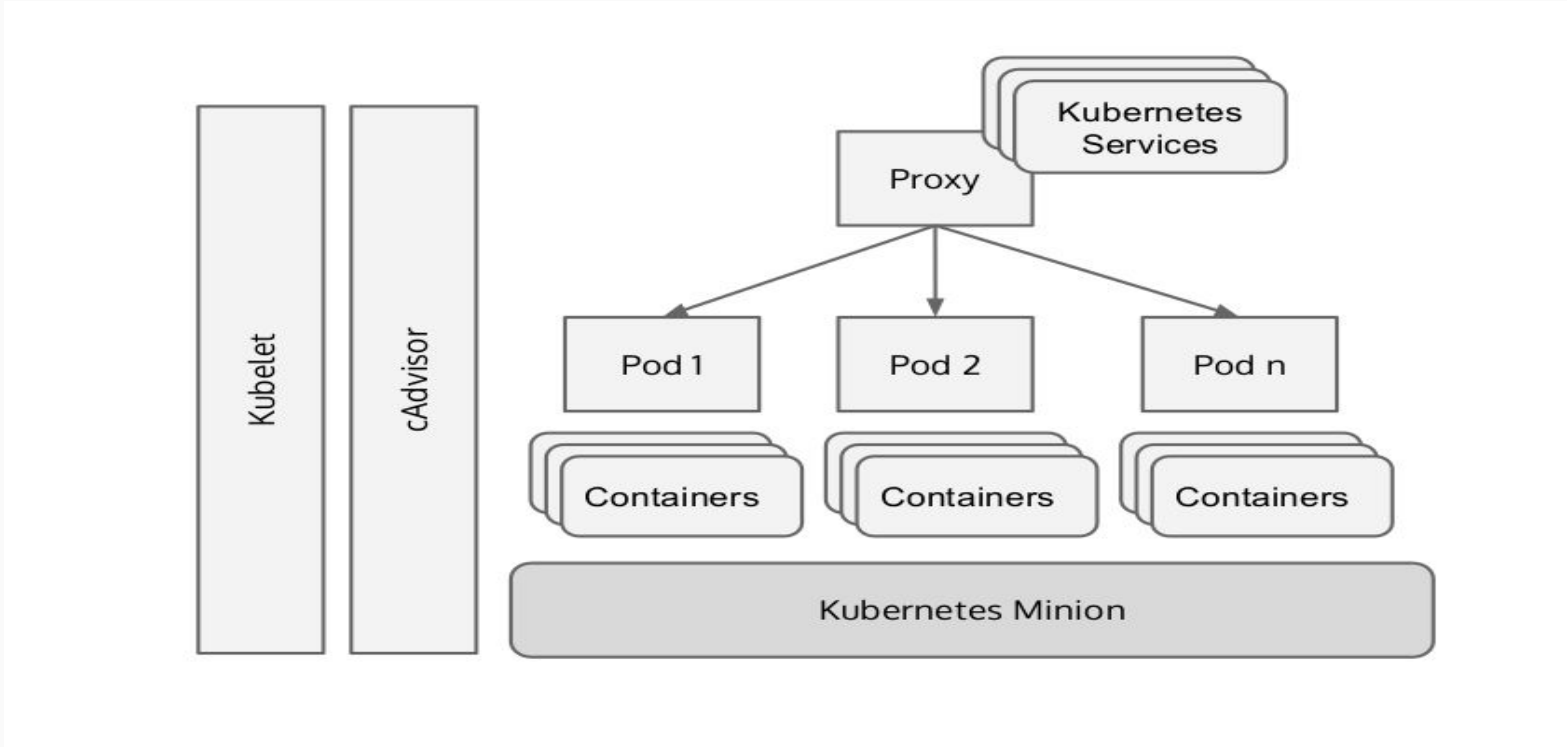
Contains multiple cluster level functions

- Manages service end-points (endpoint controller)
- Node lifecycle managment (node controller) - responds to notice when nodes go down
- Replication controller manager - in charge of maintaining the correct number of pods for every replication controller in the system

Kubernetes node components

- Docker
- Kubelet
- proxy

Kubernetes node (worker node)



Docker

Every node should contain the docker runtime engine

- Handles downloading and running containers
- Controller locally via API by the kubelet

Kubelet

Manages the node and is responsible for

- Node registration
- Pod management
- Resource utilization report for scheduling purposes
- Health status for pods running on node

kube-proxy

Every runs a kube-proxy daemon

- In charge of simple TCP and UDP stream forwarding to services managed in kubernetes (sually using IPtables)
- Manages docker links
- Optionally allows dns management using an addon

Kubernetes resource declaration

- Pods
- Replication controller/Replica set
- Services
- Configmaps
- Secrets
- Deployments

Pod resource declaration

```
{
  "kind": "Pod",
  "apiVersion": "v1",
  "metadata": {
    "name": "my-cool-nginx",
    "labels": {
      "app": "mynginx"
    }
  },
  "spec": {
    "containers": [
      {
        "name": "my-cool-nginx",
        "image": "omrisiri/nginx-hello",
        "ports": [
          {
            "containerPort": 8080,
            "protocol": "TCP"
          }
        ]
      }
    ]
  }
}
```

Save it as single-pod.json

```
$ kubectl create -f ./single-pod.json
```

Replication controller declaration

```
{
  "kind": "ReplicationController",
  "apiVersion": "v1",
  "metadata": {
    "name": "nginx-controller"
  },
  "spec": {
    "replicas": 2,
    "selector": {
      "app": "nginx"
    },
    "template": {
      "metadata": {
        "labels": {
          "app": "nginx"
        }
      },
      "spec": {
        "volumes": null,
        "containers": [
          {
            "name": "nginx",
            "image": "omrisiri/nginx-hello",
            "ports": [
              {
                "containerPort": 8080,
                "protocol": "TCP"
              }
            ],
            "imagePullPolicy": "IfNotPresent"
          }
        ],
        "restartPolicy": "Always",
        "dnsPolicy": "ClusterFirst"
      }
    }
  }
}
```

```
$ kubectl create -f ./single-rc.json
```

Replication set declaration

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 80
```

Service declaration

```
{
    "kind": "Service",
    "apiVersion": "v1",
    "metadata": {
        "name": "test-service"
    },
    "spec": {
        "selector": {
            "app": "nginx"
        },
        "ports": [
            {
                "protocol": "TCP",
                "port": 80,
                "targetPort": 8080
            }
        ]
    }
}
```

```
$ kubectl create -f ./single-service.json
```

Configmap declaration

```
kind: ConfigMap
apiVersion: v1
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: example-config
  namespace: default
data:
  example.property.1: hello
  example.property.2: world
  example.property.file: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
```

$ kubectl create configmap my-config --from-file=configfile.yaml

Secrets declaration

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
data:
  data-1: dmFsdWUtMQ0K
  data-2: dmFsdWUtMg0KDQo=
```

```
$ kubectl create -f secret.yaml
```

Deployment declaration

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

$ kubectl create -f nginx-deployment.yaml

Example time

Now we'll show an example of deploying an nginx application using deployment

Rolling back and removing

The Kubectl Command

The kubectl command is used to manage all kubernetes functions

- Creating new resources
- Nodes
- Pods
- Services
- Replication controller
- Increasing replication size
- Deployments

Kubectl commands - creating resources

```
# Create a pod using the data in pod.json.

kubectl create -f ./pod.json



# Create a pod based on the JSON passed into stdin.

cat pod.json | kubectl create -f -
```

Kubectl commands - Nodes and cluster info

#Get the state of your cluster

$ kubectl cluster-info

#Get all the nodes of your cluster

$ kubectl get nodes -o wide

Kubectl commands - pods

#Get info about the pods of your cluster

$ kubectl get pods -o wide

#Get the IP of a Pod

$ kubectl get pod <NAME_OF_POD> -template={{.status.podIP}}

#Delete a Pod

$ kubectl delete pod NAME

Kubectl commands - replication controller

#Get info about the replication controllers of your cluster

$ kubectl get rc -o wide

Kubectl commands - services

#Get info about the services of your cluster

$ kubectl get services

#Get full config info about a Service

$ kubectl get service <NAME_OF_SERVICE> -o json

#Delete a Service

$ kubectl delete service NAME_OF_THE_SERVICE

Deploying Kubernetes

- Provision cluster nodes
- Configure kubernetes compatible network
- Deploy kubernetes services
- Client tools

Consistent data store - etcd

- Etcd is a distributed, consistent key-value store
  - Secure - optional SSL client cert authentication
  - Fast - Benchmarked at 1000s of writes per second
  - Reliable - properly distributes using the raft consensus algorithm