CLOUD COMPUTING APPLICATIONS

Kubernetes - Intro

Prof. Reza Farivar

Kubernetes

- A platform to orchestrate the deployment, scaling, and management of container-based applications
  - The primary responsibility of Kubernetes is container orchestration.
  - All the containers that execute workloads are scheduled to run on physical or virtual machines
  - Containers must be packed efficiently
  - replace dead, unresponsive, or unhealthy containers
- Kubernetes is not a **Platform as a Service** (**PaaS**). It doesn't dictate many important aspects that are left to you or to other systems built on top of Kubernetes, such as Deis, OpenShift, and Eldario

Local Kubernetes for Development

- minikube is local Kubernetes, focusing on making it easy to learn and develop for Kubernetes
- Docker Desktop comes with a pre-installed Kubernetes local installation
- Other options
  - Kind
  - K3s
  - MicroK8s

Cloud Kubernetes Offerings

- GCP Google Kubernetes Engine GKE
- AWS Elastic Kubernetes Engine EKS
  - Eksktl, created by Weaveworks and endorsed by AWS
- Azure Kubernetes Service AKS
- IBM Cloud Kubernetes Service
- Oracle Container Engine for Kubernetes
- Digital Ocean
- Alibaba
- Rackspace
- Tencent
- …

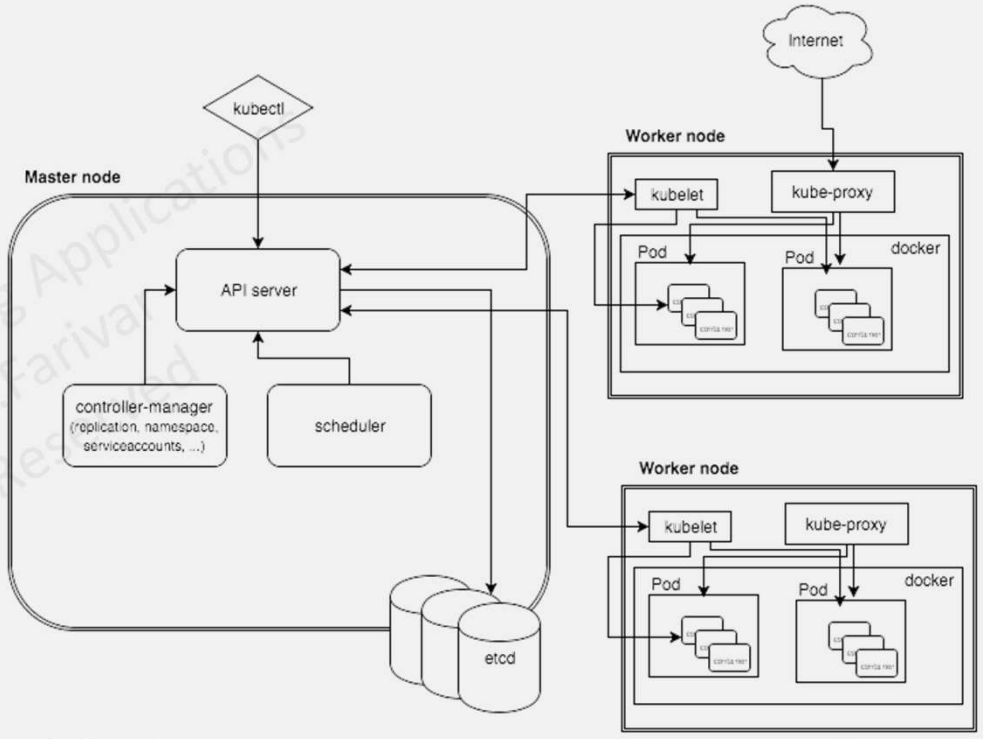
CLOUD COMPUTING APPLICATIONS

Kubernetes - Architecture

Prof. Reza Farivar

Kubernetes Architecture

- Master node
  - API Server
  - Controller
  - Scheduler
  - etcd
- Worker node
  - kubelet
  - Kube-proxy
  - Container Runtime



Clusters and Nodes

- A cluster is a collection of hosts (*nodes*) that provide compute, memory, storage, and networking resources
- Kubernetes runs workloads by placing containers into Pods to run on *Nodes*
  - A node may be a virtual or physical machine, depending on the cluster
- Each node is managed by the control plane and contains the services necessary to run Pods

Control Plane

- The master is the control plane of Kubernetes
  - It consists of several components, such as an API server, a scheduler, and a controller manager
- Kubernetes has a "hub-and-spoke" API pattern
  - All API usage from nodes (or the pods they run) terminates at the API Server
- The master is responsible for the global state of the cluster, cluster-level scheduling of pods, and handling of events
- It is designed to horizontally scale, which means that you can keep adding more instances of it to make your cluster highly available

etcd

- A highly available consistent key-value store
- Used to store the state of the cluster

kubelet

- An agent that runs on each node within the cluster
  - The means by which the managers interact with the nodes
  - Responsible for managing the pods

kube-proxy

- Manages routing of requests and traffic for both the node and the pods

container-runtime

- Container runtime could be Docker, CRI-O, or any other OCI-compliant runtime
- As of Kubernetes 1.10, `containerd` is the default

Namespace

- Kubernetes supports multiple virtual clusters backed by the same physical cluster.
- These virtual clusters are called namespaces.
- Names of resources need to be unique within a namespace, but not across namespaces

Resource Quota

- A resource quota, defined by a ResourceQuota object, provides constraints that limit aggregate resource consumption per namespace
- It can limit the quantity of objects that can be created in a namespace by type, as well as the total amount of compute resources that may be consumed by resources in that namespace.

Workload Model

- A workload is an application running on Kubernetes
- Pods
- ReplicaSets
- Deployment
- StatefulSet
- DaemonSet
- Job
- CronJob
- Service

CLOUD COMPUTING APPLICATIONS

Kubernetes – Pods

Prof. Reza Farivar

Pods

- Pod is a special Kubernetes construct
- A grouping of one or more containers that share some namespaces
  - E.g. Network
  - Containers in a Pod can communicate with each other through `localhost`
- The Pod abstraction allows certain innovative design patterns
  - Still, in practice, most of the times we typically see a single-container pod

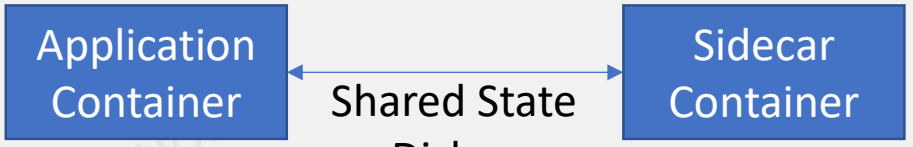
Pod Container Design Patterns

- The Pod design in Kubernetes allows to explore some interesting container-based design patterns
- Containers in a pod share some namespaces, including network namespace
- Single Node patterns
  - Sidecar
  - Ambassador
  - Adapter

"Designing Distributed Systems" by Brendan Burns

Sidecar Design Pattern

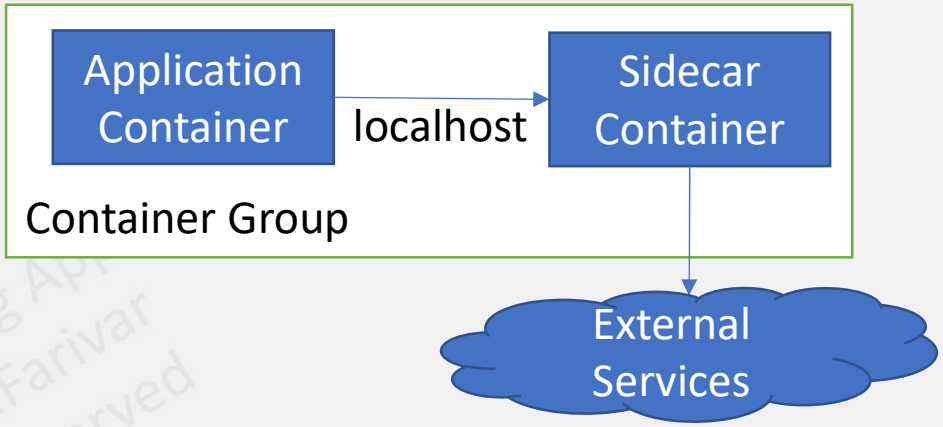
- Made up of two containers
  - Application Container
  - Sidecar container
    - Augment and improve the application container
    - Without the application container's knowledge
- Example Use cases:
  - Adding HTTPS to a legacy service
  - Dynamic Configuration with sidecars



- Disk
- Network

Ambassador Design Pattern

- Ambassador container brokers interactions between the application container and the rest of the world
- Example use cases:
  - Shard a service
  - Service Discovery
  - Experiments or Request Splitting



CLOUD COMPUTING APPLICATIONS

Kubernetes – Higher Order Resource Abstractions

Prof. Reza Farivar

Labels and Label Selectors

- *Labels* are key/value pairs that are attached to objects, such as pods
  - Labels can be used to organize and to select subsets of objects
  - Labels enable users to map their own organizational structures onto system objects in a loosely coupled fashion
- Label selectors are used to select objects based on their labels
  - The client/user can identify a set of objects
  - The label selector is the core grouping primitive in Kubernetes

```
"metadata": {
  "labels": {
    "key1" : "value1",
    "key2" : "value2"
  }
}
```

YAML

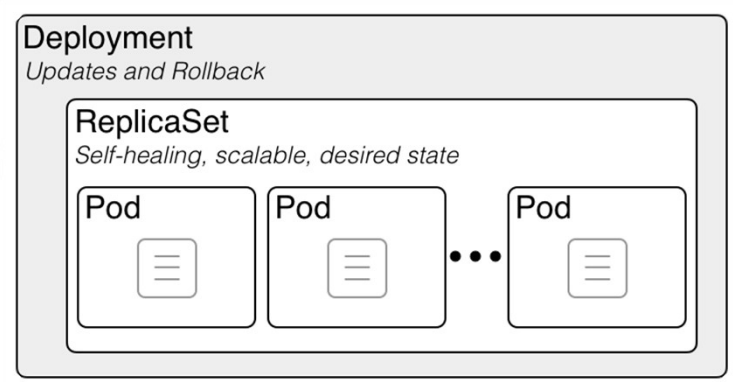
ReplicaSet

- A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time
  - used to guarantee the availability of a specified number of identical Pods
- Fields
  - Selector that specifies how to identify Pods it can acquire
  - Number of replicas indicating how many Pods it should be maintaining
  - A pod template specifying the data of new Pods it should create

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

Deployment

- a Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features
  - You can define Deployments to create new ReplicaSets
- You describe a desired state in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate
- Manages the rollout for a ReplicaSet



StatefulSet

- Manages the deployment and scaling of a set of Pods, AND provides guarantees about the ordering and uniqueness of these Pods
- Unlike a Deployment, a StatefulSet maintains a sticky identity for each of their Pods
- These pods are created from the same spec, but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling.
- Although individual Pods in a StatefulSet are susceptible to failure, the persistent Pod identifiers make it easier to match existing volumes to the new Pods that replace any that have failed
- The storage for a given Pod must either be provisioned by a `PersistentVolume` Provisioner based on the requested storage class, or pre-provisioned by an admin.

Jobs

- A Job creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate
  - As pods successfully complete, the Job tracks the successful completions
  - When a specified number of successful completions is reached, the task (ie, Job) is complete
- A simple case is to create one Job object in order to reliably run one Pod to completion
  - The Job object will start a new Pod if the first Pod fails or is deleted

Jobs

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

CLOUD COMPUTING APPLICATIONS

Kubernetes - Services

Prof. Reza Farivar

Exposing Pods

- Create an nginx Pod, and note that it has a container port specification
- This makes it accessible from any node in your cluster
- You should be able to ssh into any node in your cluster and curl both IPs

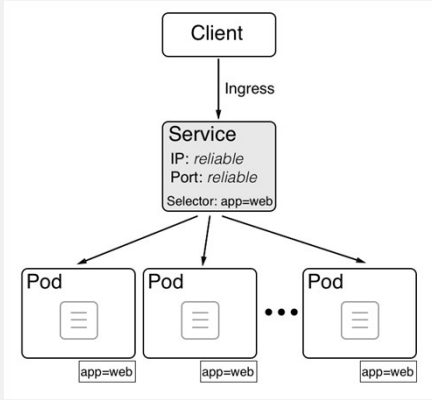
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
        - containerPort: 80
```

Service

- Kattle vs. pets
- In the past, when systems were small, each server had a name
  - You knew exactly what software was running on each machine
- Pods run in a flat, cluster wide address space
  - In theory, you could talk to these pods directly using <IP:port>
- In Kubernetes when a node dies:
  - The pods die with it, and the Deployment will create new ones, with different IPs.
  - This is the problem a Service solves.

Service

1. A Kubernetes Service is an abstraction which defines a logical set of Pods running somewhere in your cluster, that all provide the same functionality.
   - A group of Pods, as determined by a label selector, which all offer a common port name/number that serves a single purpose.
   - A pool of HTTP servers that all have the same content available.
2. A policy by which to access them



- Sometimes this pattern is called a micro-service

Service example in YAML

- For example, suppose you have a set of Pods where each listens on TCP port 9376 and contains a label app=MyApp
- This specification creates a new Service object named "my-service", which targets TCP port 9376 on any Pod with the app=MyApp label.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Service

- When created, each Service is assigned a unique IP address (also called clusterIP).
  - Used by the Service proxies
- This address is tied to the lifespan of the Service, and will not change while the Service is alive.
- Pods can be configured to talk to the Service, and know that communication to the Service will be automatically load-balanced out to some pod that is a member of the Service.

Cloud-Native Service Discovery

- If you're able to use Kubernetes APIs for service discovery in your application, you can query the API server for Endpoints, that get updated whenever the set of Pods in a Service changes.
- For non-native applications, Kubernetes offers ways to place a network port or load balancer in between your application and the backend Pods.

CLOUD COMPUTING APPLICATIONS

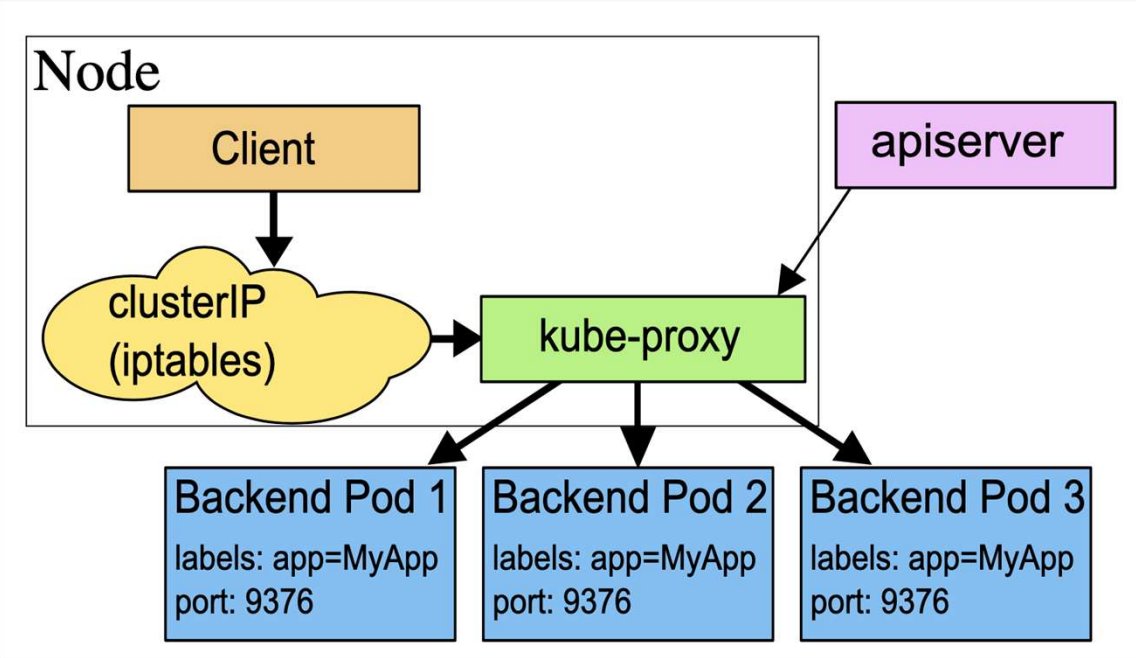
Kubernetes – Networking Model

Prof. Reza Farivar

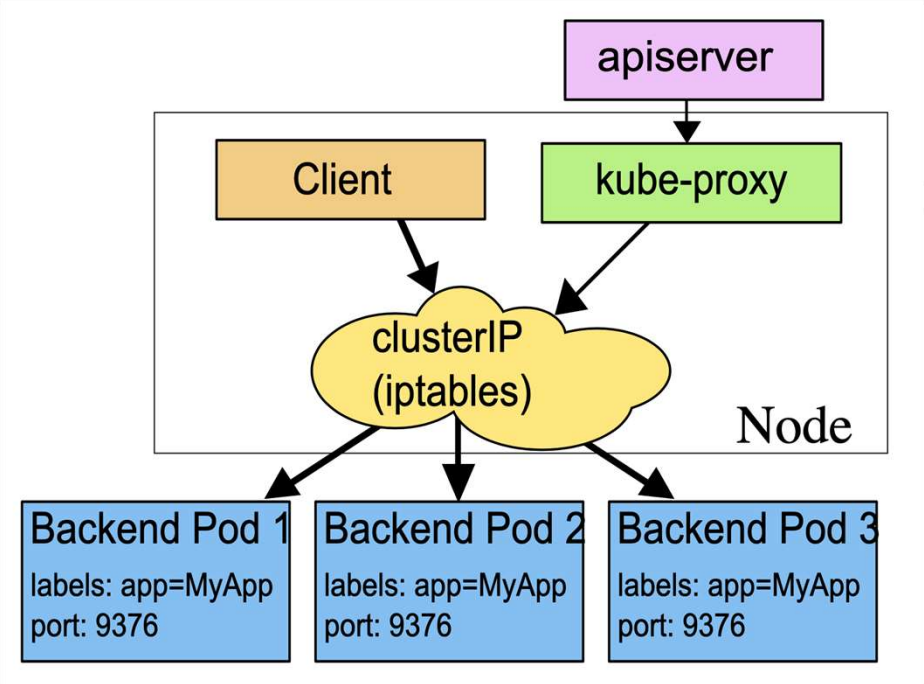
Virtual IPs and service proxies

- Every node in a Kubernetes cluster runs a `kube-proxy`.
- `kube-proxy` watches the Kubernetes control plane for the addition and removal of Service and Endpoint objects
- `kube-proxy` is responsible for implementing a form of virtual IP for Services
  - User space proxy mode
  - iptables proxy mode
  - IPVS proxy mode

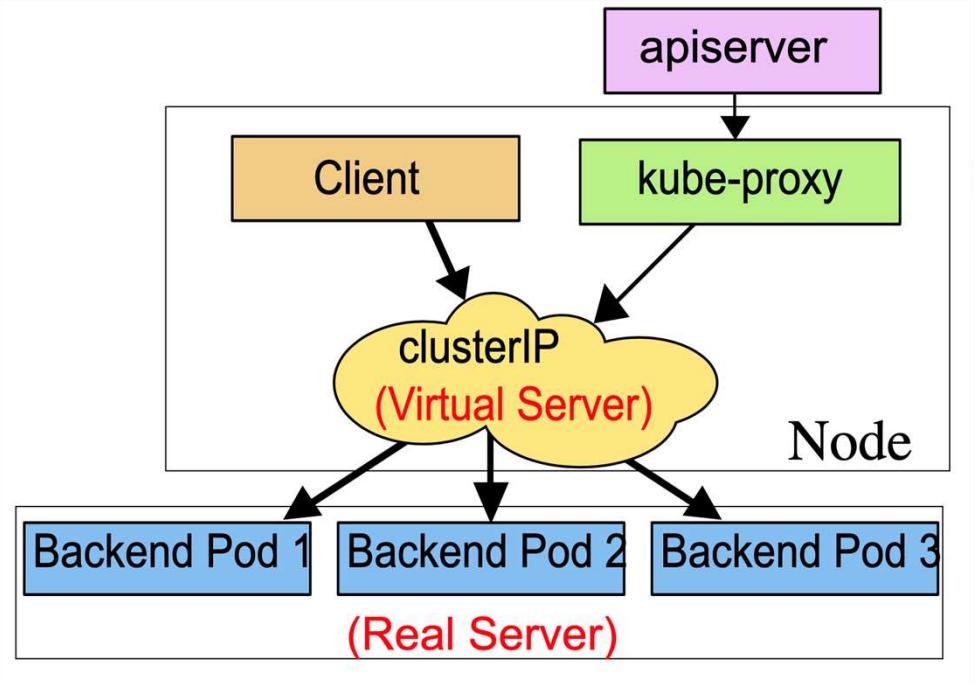
User space proxy mode



iptables proxy mode



IPVS proxy mode



Networking

- Every Pod gets its own IP address. This means you do not need to explicitly create links between Pods and you almost never need to deal with mapping container ports to host ports.
  - Kubernetes assumes that pods can communicate with other pods, regardless of which host they land on
- Why not Round Robin DNS?
  - Some DNS do not respect TTL
  - Apps have to respect TTL
  - Low or zero TTLs on the DNS records could impose a high load on DNS

CLOUD COMPUTING APPLICATIONS

Kubernetes - Service Discovery and Ingress

Prof. Reza Farivar

Two Primary Modes of Discovering Services

- Environment variables
  - When a Pod is run on a Node, the kubelet adds a set of environment variables for each active Service
  - For example, the Service `redis-master` which exposes TCP port 6379 and has been allocated cluster IP address 10.0.0.11, produces the following environment variables
- DNS
  - A cluster-aware DNS server, such as CoreDNS, watches the Kubernetes API for new Services and creates a set of DNS records for each one.

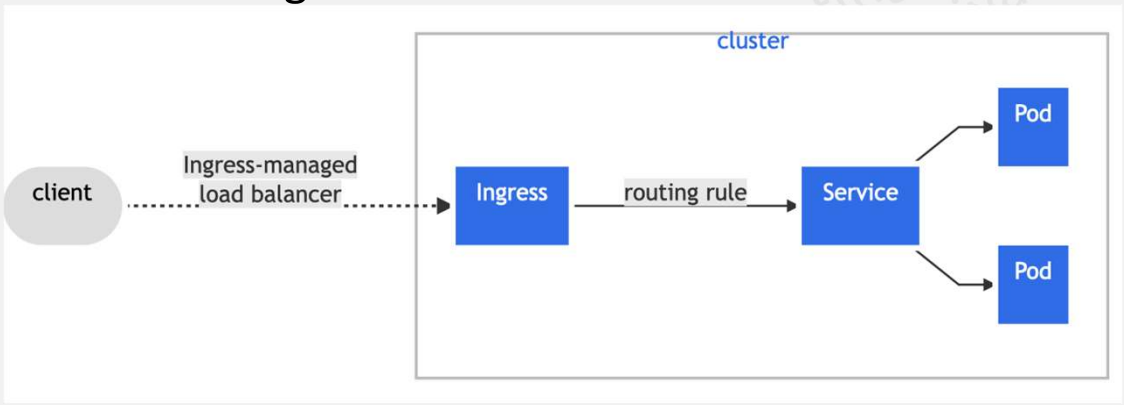
```
REDIS_MASTER_SERVICE_HOST=10.0.0.11
REDIS_MASTER_SERVICE_PORT=6379
REDIS_MASTER_PORT=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP_PROTO=tcp
REDIS_MASTER_PORT_6379_TCP_PORT=6379
REDIS_MASTER_PORT_6379_TCP_ADDR=10.0.0.11
```

Publishing Services

- For some cases (e.g. frontends) you can expose a Service onto an external IP address outside of cluster
  - ClusterIP (Default): Exposes the Service on a cluster-internal IP
    - Service only reachable from within the cluster
  - NodePort: Exposes the Service on each Node's IP at a static port
    - Contact the NodePort Service, from outside the cluster, by requesting <NodeIP>:<NodePort>
    - Each node proxies that port (the same port number on every Node) into your Service
    - A ClusterIP Service automatically created
  - LoadBalancer: Exposes the Service externally using a cloud provider's load balancer
    - NodePort and ClusterIP Services automatically created
  - ExternalName: Maps the Service to the contents of the externalName field (e.g. foo.bar.example.com), by returning a CNAME record with its value
    - No proxying of any kind is set up

Ingress

- An API object that manages external access to the services in a cluster, typically HTTP
  - Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster
- Ingress may provide externally-reachable URLs, load balancing, SSL termination and name-based virtual hosting



Exposing services other than HTTP and HTTPS to the internet typically uses a service of type Service.Type=NodePort or Service.Type=LoadBalancer

Example Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

Context-based Routing

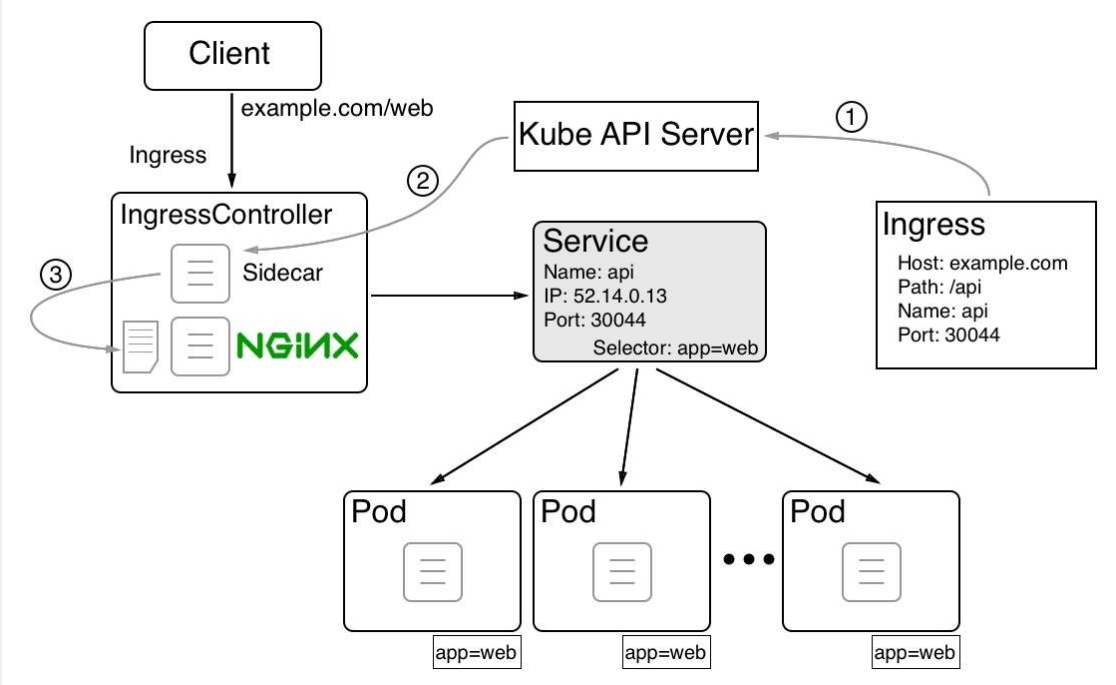


Image Courtesy of "Learn Docker - Fundamentals of Docker 19.x - Second Edition"

CLOUD COMPUTING APPLICATIONS

Kubernetes – Final Thoughts

Prof. Reza Farivar

Comparing Docker Swarm and Kubernetes

Docker Swarm	Kubernetes	Description
Container	Container**	An instance of a container image running on a node. **Note: In a Kubernetes cluster, we cannot run a container directly.
Task	Pod	An instance of a service (Swarm) or ReplicaSet (Kubernetes) running on a node. A task manages a single container while a Pod contains one to many containers that all share the same network namespace.
Service	ReplicaSet	Defines and reconciles the desired state of an application service consisting of multiple instances.
Service	Deployment	A deployment is a ReplicaSet augmented with rolling updates and rollback capabilities.
Routing Mesh	Service	The Swarm Routing Mesh provides L4 routing and load balancing using IPVS. A Kubernetes service is an abstraction that defines a logical set of pods and a policy that can be used to access them. It is a stable endpoint for a set of pods.
Network	Network policy	Swarm **software-defined networks** (**SDNs**) are used to firewall containers. Kubernetes only defines a single flat network. Every pod can reach every other pod and/or node, unless network policies are explicitly defined to constrain inter-pod communication.

Comparison Table Courtesy of "Learn Docker - Fundamentals of Docker 19.x - Second Edition"

Helm

- Helm is a package manager for Kubernetes.
- Writing and maintaining Kubernetes **YAML manifests** for all the required Kubernetes objects can be a time consuming and tedious task
- Helm Charts are simply Kubernetes YAML manifests combined into a single package that can be advertised to your Kubernetes clusters
- Helm deploys charts, which you can think of as a packaged application.
  - Helm is the K8s equivalent of yum or apt.
- It is a collection of all your versioned, pre-configured application resources which can be deployed as one unit.