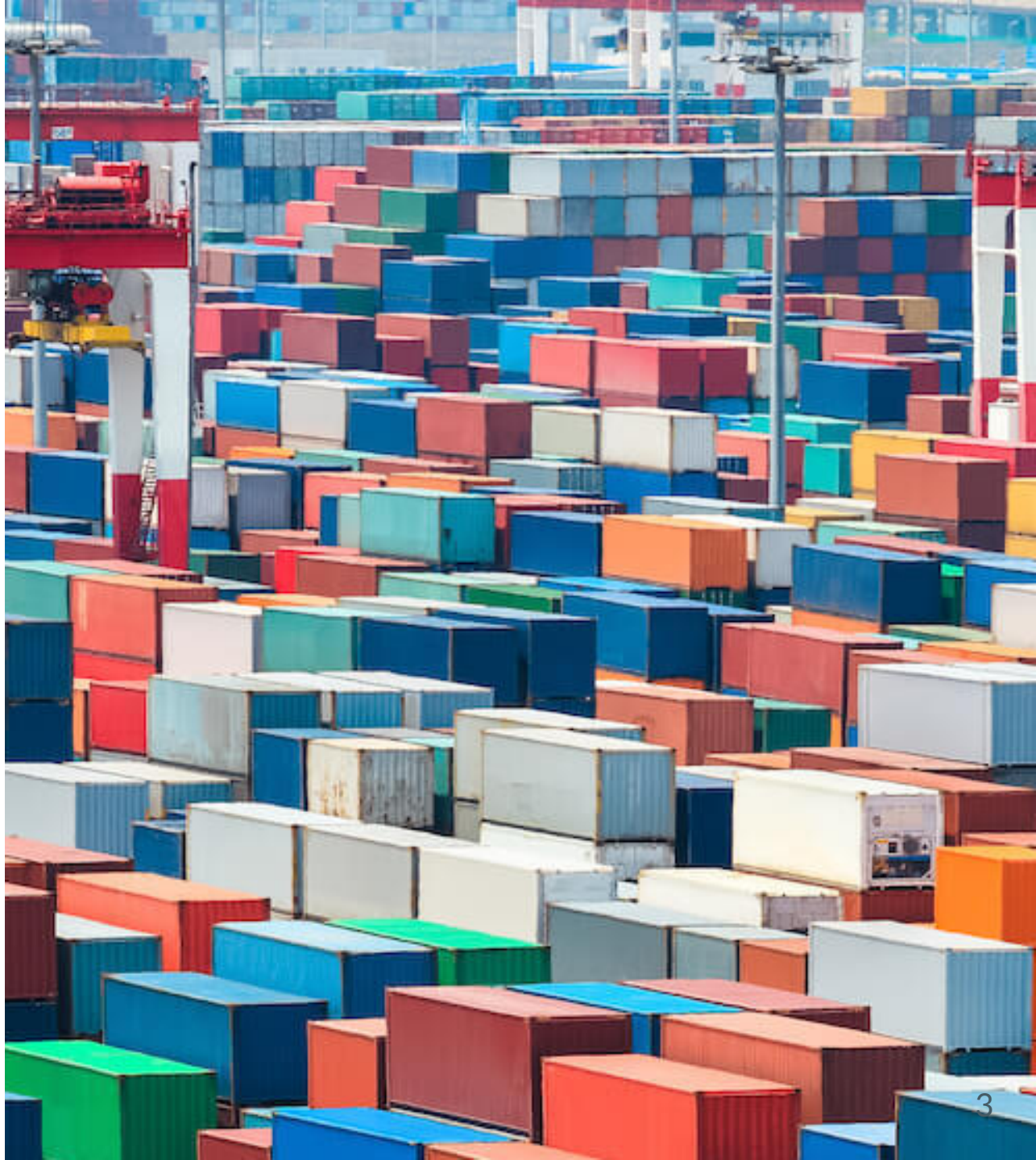
Kubernetes 101

Kubernetes 101

김영우 (Youngwoo Kim)
Data Labs, ICT기술센터, SK Telecom

What is Kubernetes?

- Kubernetes is an open source **container orchestration engine** for automating deployment, scaling, and management of containerized applications. The open source project is hosted by the Cloud Native Computing Foundation



History of Kubernetes

- 2003-2004: Birth of the Borg System
- 2014: Google Introduces Kubernetes
- 2015: The year of Kube v1.0 & CNCF
- 2017: The Year of Enterprise Adoption & Support
- 2018: ……
- 2019: Kubernetes v1.16.2 released (Oct. 2019)

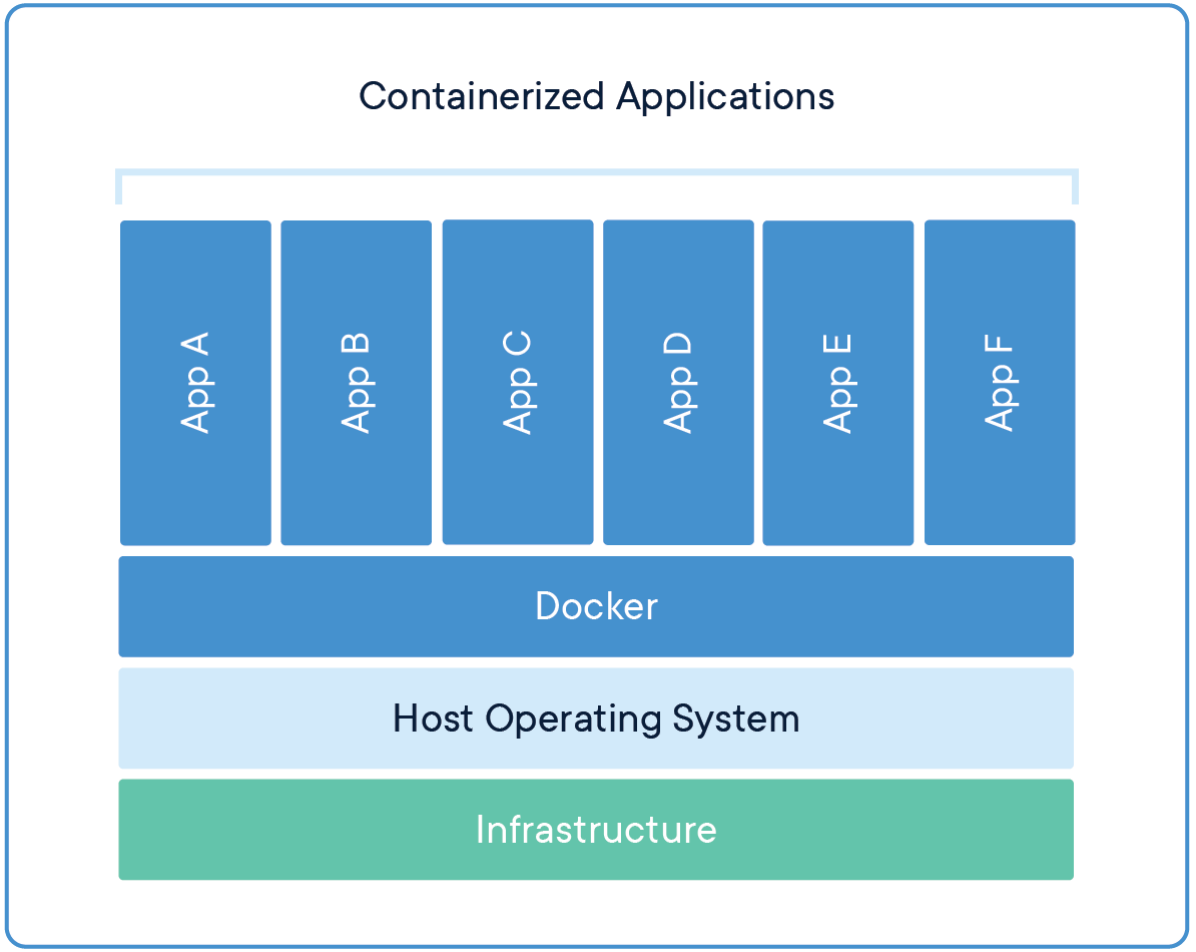
https://blog.risingstack.com/the-history-of-kubernetes/

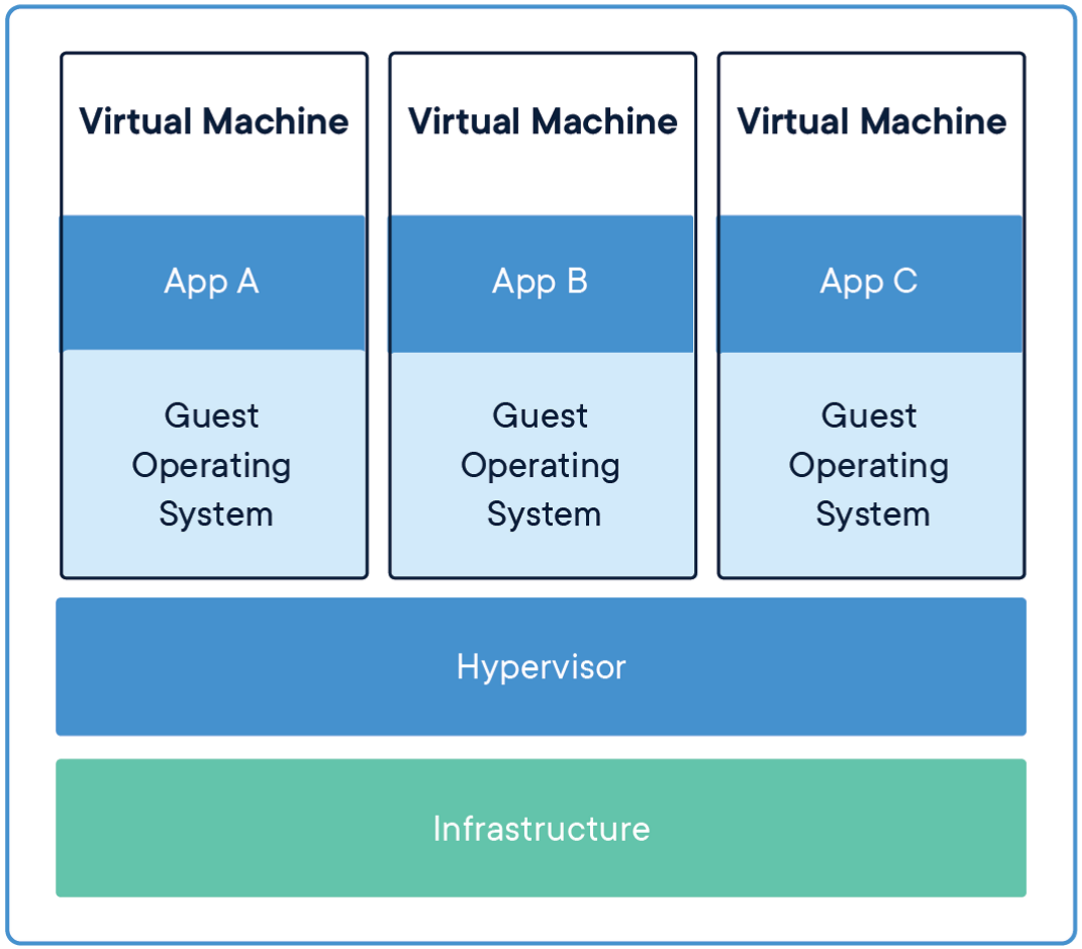
What is Container?

- Containers are a method of virtualization that packages an application's code, configurations, and dependencies into building blocks for consistency, efficiency, productivity, and version control. This page gathers resources about containers, including technical definitions and comparisons.

https://www.aquasec.com/wiki/display/containers/What+is+a+Container

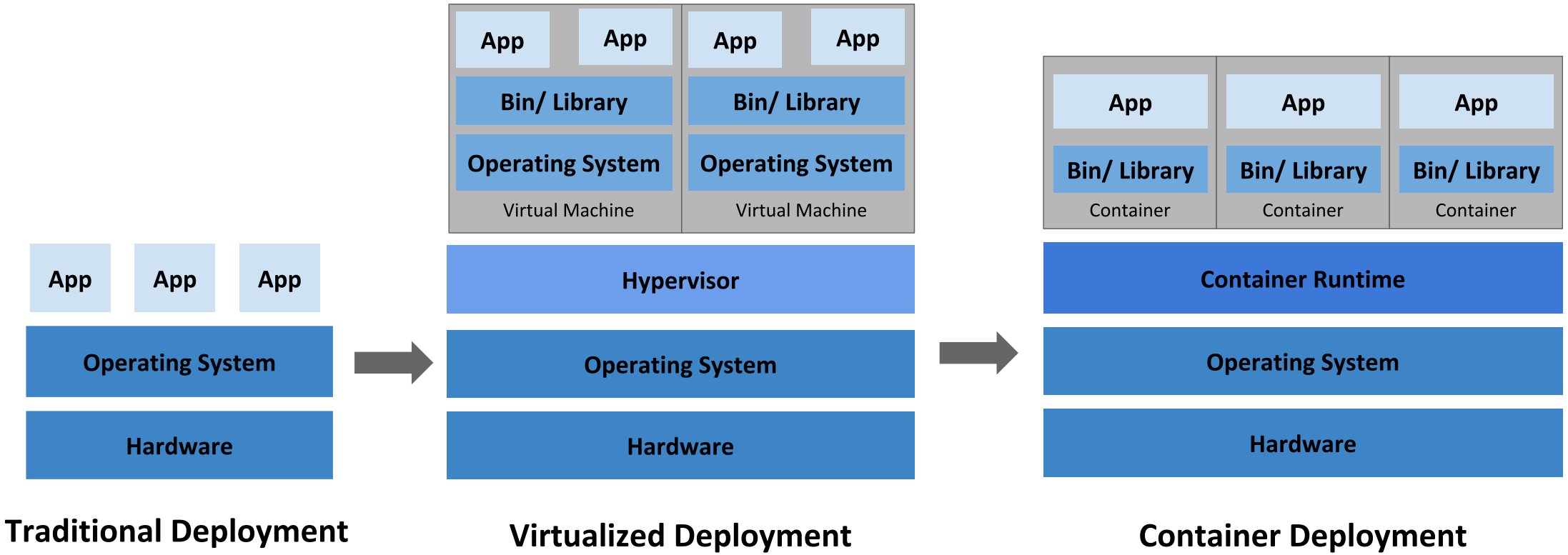
Containers


Containerized Applications
App A
App B
App C
App D
App E
App F
Docker
Host Operating System
Infrastructure



Virtual Machines

What Kubernetes can do



- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Automatic bin packing
- Self-healing
- Secret and configuration management

Kubernetes Components

- A cluster is a set of machines, called nodes, that run containerized applications managed by Kubernetes. A cluster has at least one worker node and at least one master node.
- Master components
  - kube-apiserver, etcd, kube-scheduler, kube-controller-manager, cloud-controller-manager
- Node components
  - kubelet, kube-proxy, Container Runtime
- Addons
  - DNS, Dashboard, Container Resource Monitoring, Cluster-level Logging

Kubernetes API

- Refer to Kubernetes API reference, https://kubernetes.io/docs/reference/

```
$ kubectl api-versions | more
$ kubectl api-resources | more
$ kubectl explain --api-version=apps/v1 replicaset
$ kubectl explain deployment.metadata | more
$ kubectl explain deployment.spec | more
$ kubectl explain deployment.spec.template.spec.containers --recursive | more
```

Kubernetes Objects

- *Kubernetes Objects* are persistent entities in the Kubernetes system.
- A Kubernetes object is a “record of intent”–once you create the object, the Kubernetes system will constantly work to ensure that object exists.
- To work with Kubernetes objects–whether to create, modify, or delete them–you’ll need to use the Kubernetes API. When you use the kubectl command-line interface, for example, the CLI makes the necessary Kubernetes API calls for you.

Object spec and status

- Every Kubernetes object includes two nested object fields that govern the object's configuration: the object *spec* and the object *status*.
  - E.g., A Kubernetes *Deployment* is an object that can represent an application running on your cluster.

- Describing a Kubernetes Object
  - An example .yaml file that shows the required fields and object spec for a Kubernetes Deployment:

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

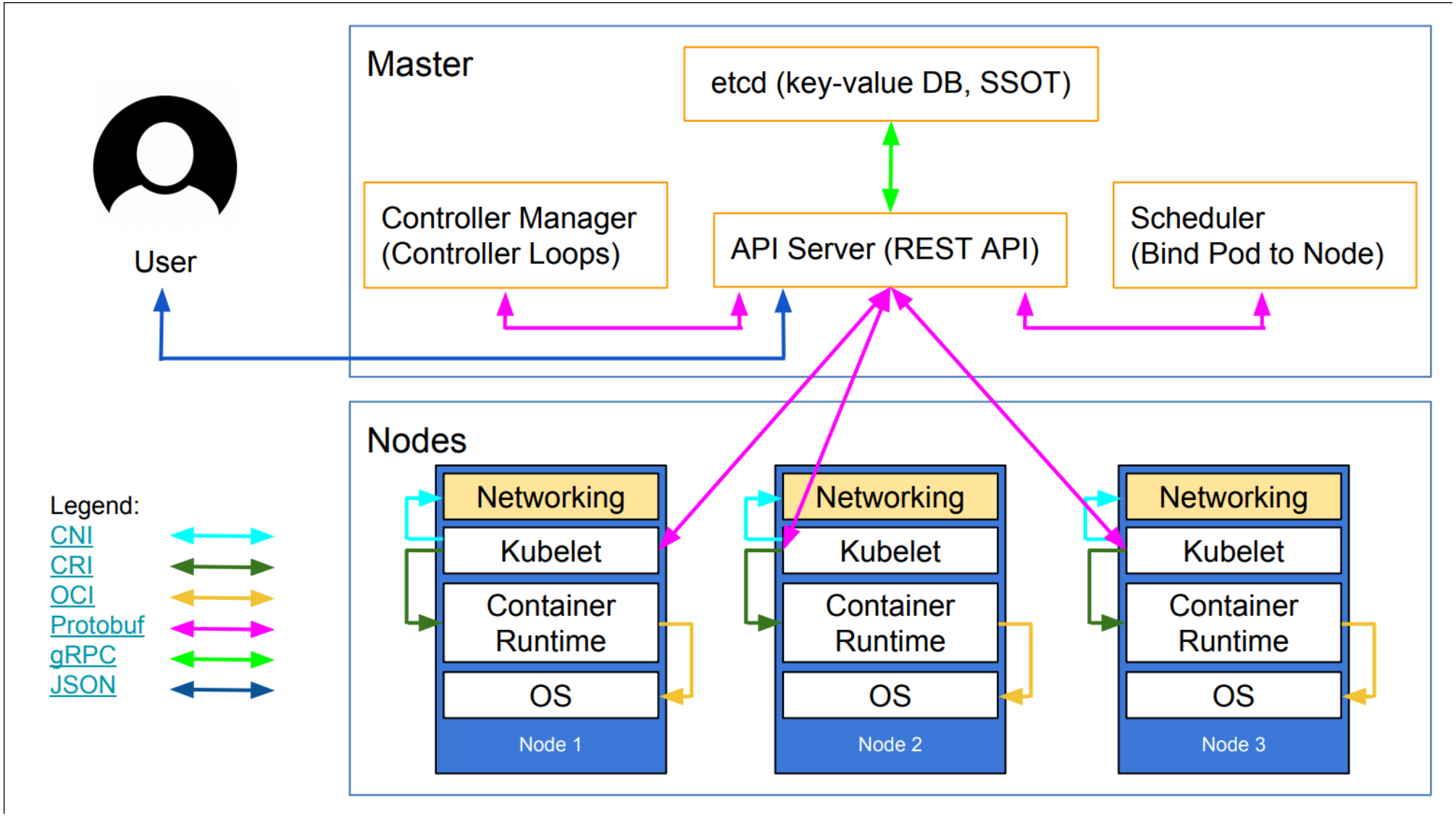
Managing Kubernetes Object

- Kubectl is a command line interface for running commands against Kubernetes clusters.
  - https://kubernetes.io/docs/reference/kubectl/kubectl/

```
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"16", GitVersion:"v1.16.2", ......
Server Version: version.Info{Major:"1", Minor:"16", GitVersion:"v1.16.2", ......
```

Namespaces, Labels and Selectors, Annotations and Field Selectors

Kubernetes Architecture



Getting started with Minikube

- Minukube?
  - Run Kubernetes locally, https://github.com/kubernetes/minikube
- Install Minikube:
  - https://kubernetes.io/docs/tasks/tools/install-minikube/

```
# Install Minikube on MacOS
$ brew install minikube

$ minikube version
minikube version: v1.5.2
commit: 792dbf92a1de583fcee76f8791cff12e0c9440ad
```

- Start Minikube:

```
$ minikube start

$ minikube status
$ minikube addons enable ingress
```

- Start Dashboard:

```
$ minikube dashboard
```

- Stop and delete Minikube instance:

```
$ minikube stop
$ minikube delete
```

Kubernetes Fundamentals

- Pod
- Controllers
  - ReplicaSet
  - ReplicationController
  - Deployments
  - StatefulSets
  - DaemonSet
  - Garbage Collection
  - TTL Controller for Finished Resources
  - Jobs - Run to Completion
  - CronJob

Pod

- A *Pod* is the basic execution unit of a Kubernetes application–the smallest and simplest unit in the Kubernetes object model that you create or deploy.
- A Pod represents a unit of *deployment*: a single instance of an application in Kubernetes, which might consist of either a single container or a small number of containers that are tightly coupled and that share resources.

Controllers

- Kubernetes runs a group of controllers that take care of routine tasks to ensure the desired state of the cluster matches the observed state.
- Basically, each controller is responsible for a particular resource in the Kubernetes world.

Controllers: Deployments

- A Deployment provides declarative updates for Pods and ReplicaSets.

- nginx-deployment.yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Controllers: ReplicaSet

- A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.
- Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features. Therefore, we recommend using Deployments instead of directly using ReplicaSets, unless you require custom update orchestration or don't require updates at all.

Controllers: ReplicationController

- A ReplicationController ensures that a specified number of pod replicas are running at any one time.

> **Note: A Deployment that configures a ReplicaSet is now the recommended way to set up replication.**

Controllers: StatefulSets

- StatefulSet is the workload API object used to manage stateful applications.
- Manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods.
- Using StatefulSets
  - Stable, unique network identifiers.
  - Stable, persistent storage.
  - Ordered, graceful deployment and scaling.
  - Ordered, automated rolling updates.
- Components
  - A Headless Service, named nginx, is used to control the network domain.
  - The StatefulSet, named web, has a Spec that indicates that 3 replicas of the nginx container will be launched in unique Pods.
  - The volumeClaimTemplates will provide stable storage using PersistentVolumes provisioned by a PersistentVolume Provisioner.

- simple-statefulset.yaml:

```
apiVersion: v1
kind: Service
metadata:
name: nginx
labels:
app: nginx
spec:
ports:
- port: 80
name: web
clusterIP: None
selector:
app: nginx
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
name: web
spec:
selector:
matchLabels:
  app: nginx # has to match .spec.template.metadata.labels
serviceName: "nginx"
```

```yaml
replicas: 3 # by default is 1
template:
metadata:
  labels:
    app: nginx # has to match .spec.selector.matchLabels
spec:
  terminationGracePeriodSeconds: 10
  containers:
  - name: nginx
    image: k8s.gcr.io/nginx-slim:0.8
    ports:
    - containerPort: 80
      name: web
    volumeMounts:
    - name: www
      mountPath: /usr/share/nginx/html
volumeClaimTemplates:
- metadata:
  name: www
spec:
  accessModes: [ "ReadWriteOnce" ]
  storageClassName: "my-storage-class"
  resources:
    requests:
      storage: 1Gi
```

Controllers: DaemonSet

- A *DaemonSet* ensures that all (or some) Nodes run a copy of a Pod.
- Some typical uses of a DaemonSet are:
  - running a cluster storage daemon, such as glusterd, ceph, on each node.
  - running a logs collection daemon on every node, such as fluentd or logstash.
  - running a node monitoring daemon on every node, such as Prometheus Node Exporter, Sysdig Agent, collectd, Dynatrace OneAgent, AppDynamics Agent, Datadog agent, New Relic agent, Ganglia gmond or Instana Agent.

Controllers: Jobs

- A Job creates one or more Pods and ensures that a specified number of them successfully terminate.

Controllers: CronJob

- A Cron Job creates Jobs on a time-based schedule.

Controllers: TTL Controller

Controllers: Garbage Collection

Services, Load Balancing, and Networking

Services

- An abstract way to expose an application running on a set of Pods as a network service.
  - A service functions as a proxy to replicated pods and service requests can be load balanced across pods.

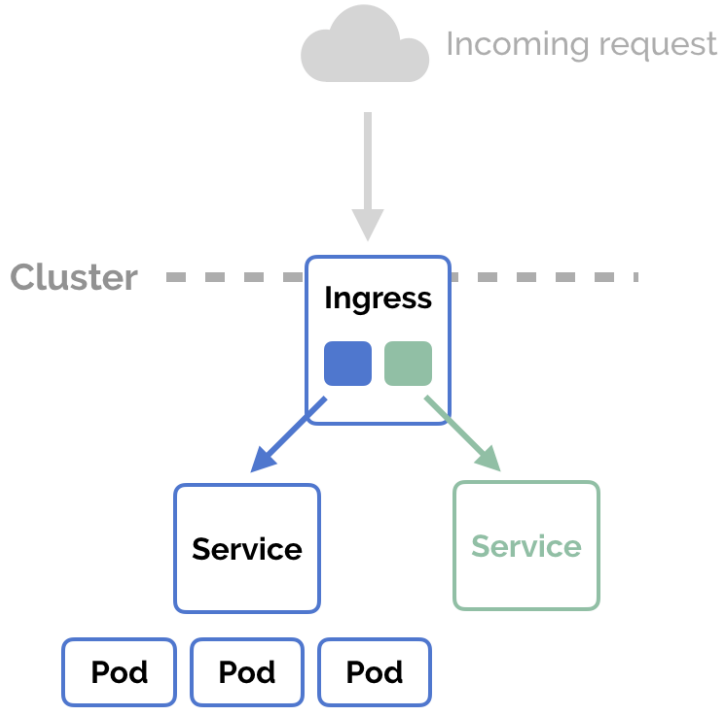
Endpoint Slices

DNS for Services and Pods

Connecting Applications with Services

Ingress

- Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster.



Ingress Controllers

- https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/

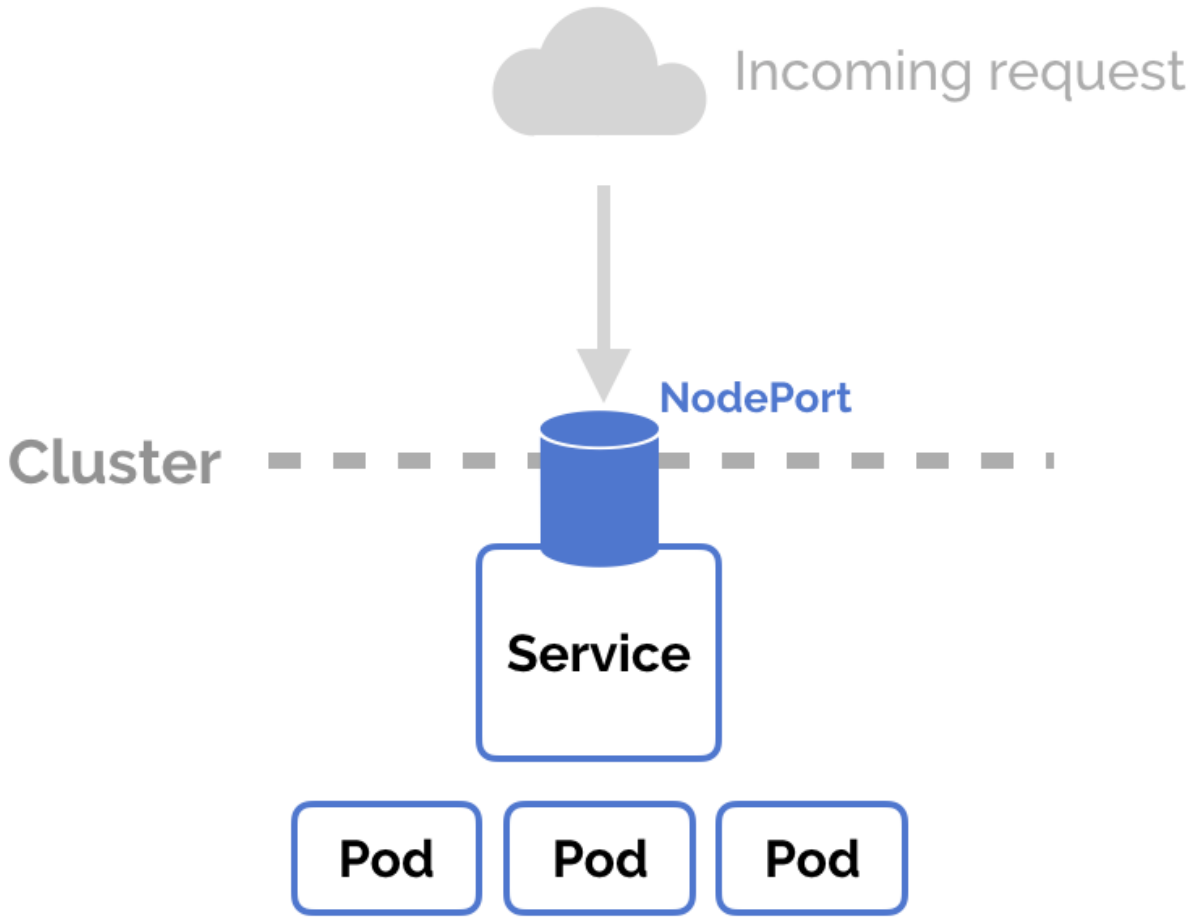
ClusterIP, NodePort, LoadBalancer and Ingress

- They are all different ways to get external traffic into your cluster, and they all do it in different ways.
- https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-ingress-guide-nginx-example.html
- https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0

ClusterIP

- A ClusterIP service is the default Kubernetes service. It gives you a service inside your cluster that other apps inside your cluster can access. There is no external access.
- When would you use ClusterIP:
  - Debugging your services, or connecting to them directly from your laptop for some reason
  - Allowing internal traffic, displaying internal dashboards, etc.

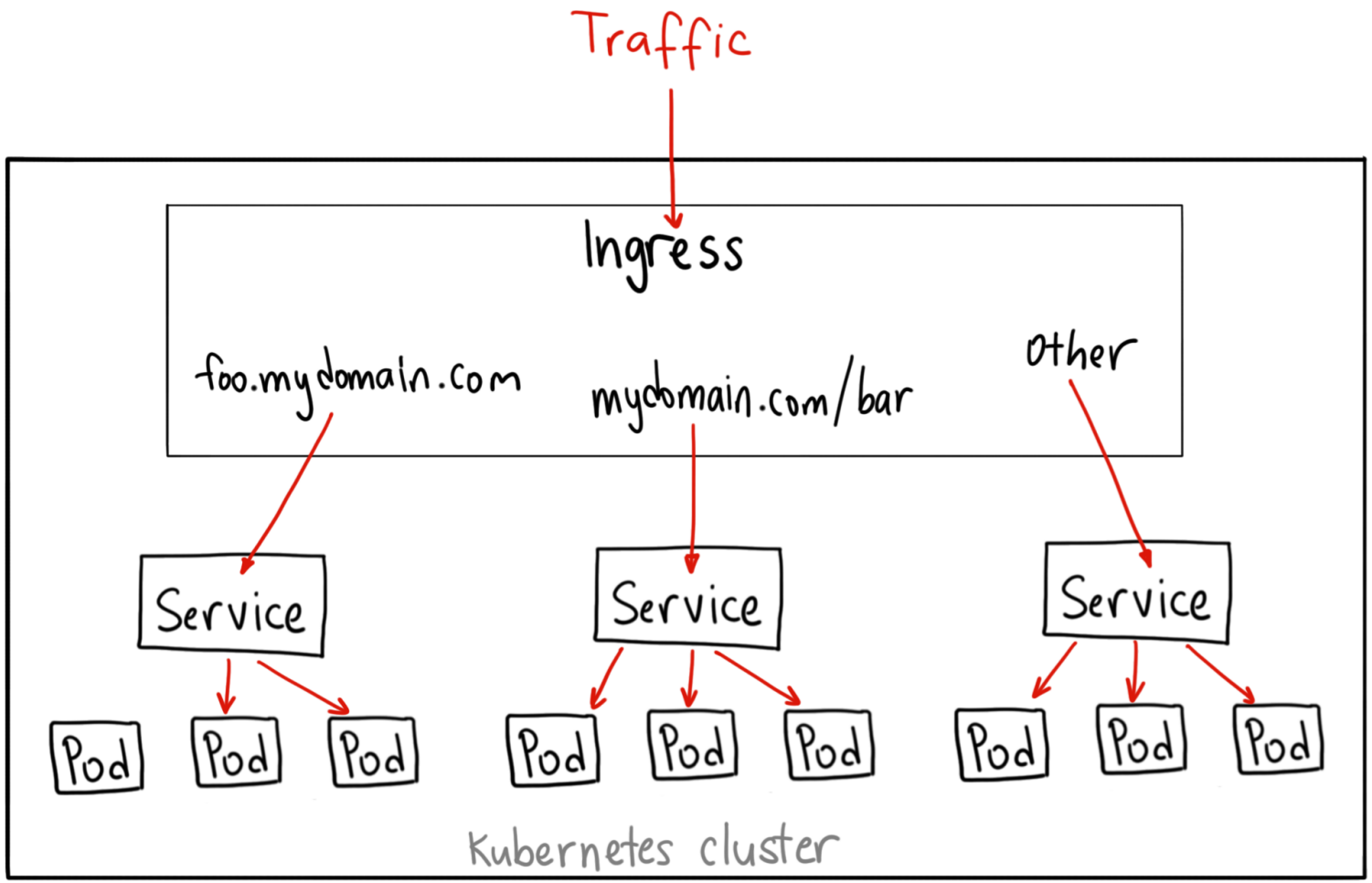
NodePort



- NodePort:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nodeport-service
spec:
  selector:
    app: my-app
  type: NodePort
  ports:
  - name: http
    port: 80
    targetPort: 80
    nodePort: 30036
    protocol: TCP
```

Ingress


Traffic
Ingress
foo.mydomain.com
mydomain.com/bar
Other
Service
Service
Service
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Kubernetes cluster

- Ingress is actually NOT a type of service. Instead, it sits in front of multiple services and act as a “smart router” or entrypoint into your cluster.

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: my-ingress
spec:
  backend:
    serviceName: other
    servicePort: 8080
  rules:
  - host: foo.mydomain.com
    http:
      paths:
      - backend:
          serviceName: foo
          servicePort: 8080
  - host: mydomain.com
    http:
      paths:
      - path: /bar/*
        backend:
          serviceName: bar
          servicePort: 8080
```

Storage

Volumes

- What problem does Kubernetes solve?
  - First, when a Container crashes, kubelet will restart it, but the files will be lost - the Container starts with a clean state.
  - Second, when running Containers together in a Pod it is often necessary to share files between those Containers.
- Types of Volumes
  - awsElasticBlockStore, azureDisk, cephfs, configMap, csi, emptyDir, glusterfs, hostPath, local, nfs...
  - https://kubernetes.io/docs/concepts/storage/volumes/#types-of-volumes

Peristent Volumes

- The PersistentVolume subsystem provides an API for users and administrators that abstracts details of how storage is provided from how it is consumed. To do this, we introduce two new API resources: PersistentVolume and PersistentVolumeClaim.
  - PersistentVolume
  - PersistentVolumeClaim

Storage Classes

- A StorageClass provides a way for administrators to describe the “classes” of storage they offer. Different classes might map to quality-of-service levels, or to backup policies, or to arbitrary policies determined by the cluster administrators.
- Storage classes have a provisioner that determines what volume plugin is used for provisioning PVs.
  - https://kubernetes.io/docs/concepts/storage/storage-classes/#provisioner

Dynamic Volume Provisioning

- Dynamic volume provisioning allows storage volumes to be created on-demand.
- StorageClass `slow`:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
```

- StorageClass: `fast`:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
```

- To select the “fast” storage class, for example, a user would create the following `PersistentVolumeClaim`:

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: claim1
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: fast
  resources:
    requests:
      storage: 30Gi
```

Hands-on Labs

How to run a simple Hello World Node.js app on Kubernetes

- Prerequisites
  - Minikube
  - Node.js

Hello World Node.js application

- Source code of *Hello World* Node.js application (server.js)

```
var http = require('http');

var handleRequest = function(request, response) {
  console.log('Received request for URL: ' + request.url);
  response.writeHead(200);
  response.end('Hello World!');
};
var www = http.createServer(handleRequest);
www.listen(8080);
```

- Validation:

```
$ HOSTNAME=`hostname` node server.js
$ curl http://localhost:8080
```

Dockerize Node.js application

- Build & Push Docker image:

```
$ docker build -t hello-node .

......

Successfully built e43978e8832c
Successfully tagged hello-node:latest

$ docker images | grep hello
hello-node                  latest              e43978e8832c        6 minutes ago       655MB
```

```
$ docker login
Username: youngwookim
Password:
Login Succeeded

$ docker tag hello-node youngwookim/hello-node
$ docker push youngwookim/hello-node
```

Create a Kubernetes Deployment and Service

- Create a Deployment:

```
$ kubectl create -f hello_node-deployment.yaml
deployment.apps/hello-node created

$ kubectl get deployments
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
hello-node   1/1     1            1           9m28s
```

- Create a Service:

```
$ kubectl create -f hello_node-service.yaml
service/hello-node-service created
```

Verify the Node.js app

- Find service url via minikube:

```
$ minikube service hello-node-service --url
http://IP:PORT
```

- Browse http://IP:PORT
- Or run via curl:

```
$ curl $(minikube service hello-node-service --url)
```

Visit https://github.com/kubernetes/examples for more examples...

Cloud Native Landscape

https://landscape.cncf.io/

참고

- https://kubernetes.io
- https://www.ibm.com/cloud/garage/content/course/kubernetes-101/
- https://github.com/contino/kubernetes-101
- https://www.aquasec.com/wiki/display/containers/Kubernetes+Guide
- https://container.training/
- https://dzone.com/articles/the-complete-kubernetes-collection-tutorials-and-tools
- https://github.com/ThijsFeryn/hello-nodejs-minikube
- https://github.com/kubernetes/examples

Image Credits

- https://www.docker.com/resources/what-container
- https://selleo.com/blog/kubernetes-101
- https://container.training/kube-halfday.yml.html

FIN