fands™
A Fast AND Steady Approach

Kubernetes

Presented by
VAISHALI TAPASWI

FANDS INFONET Pvt.Ltd.

www.fandsindia.com
vaishali@fandsindia.com

Ground Rules

- Turn off cell phone. If you cannot, please keep it on silent mode. You can go out and attend your call.
- If you have questions or issues, please let me know immediately.
- Let us be punctual.

fands™

A Fast AND Steady Approach

Why Docker?

Why containerization

The Challenge

Multiplicity of Stacks

Static website
nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB
postgresql + pgv8 + v8

Queue
Redis + redis-sentinel

Analytics DB
hadoop + hive + thrift + Ope

Background workers
thon 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

Web frontend
Ruby + Rails + sass + Unicorn

API endpoint
Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

Do services and apps interact appropriately?

of hardware environmen ts

Development VM

QA server

Customer Data Center

Public Cloud

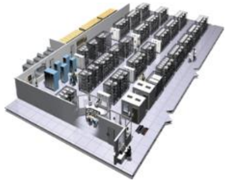
Disaster recovery

Production Servers

Production Cluster

Contributor's laptop

Can I migrate smoothly and quickly?



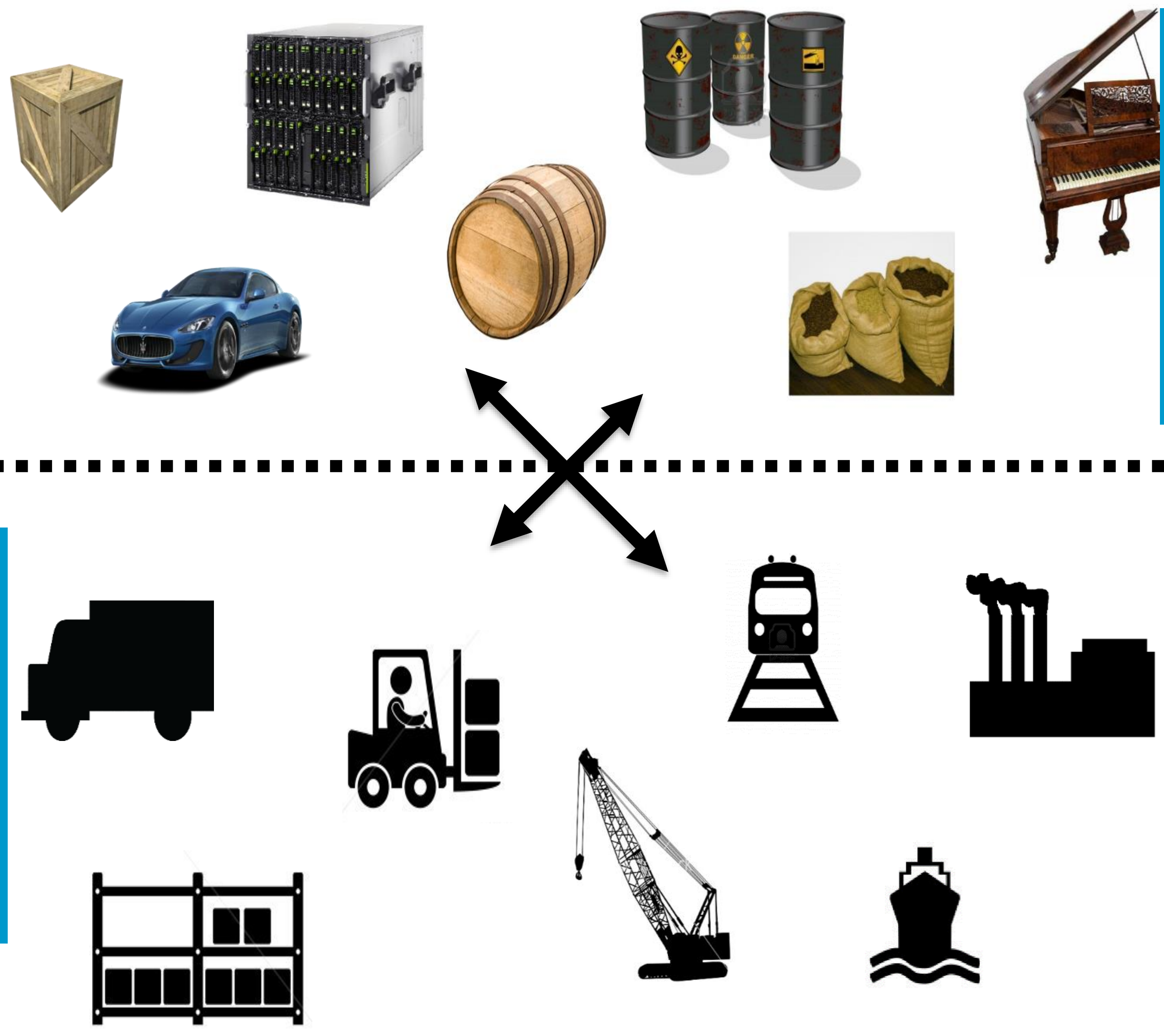
The Matrix From Hell

Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers

Cargo Transport Pre-1960

Multiplicity of Goods

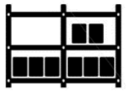
about how goods interact (e.g. coffee beans next to spices)



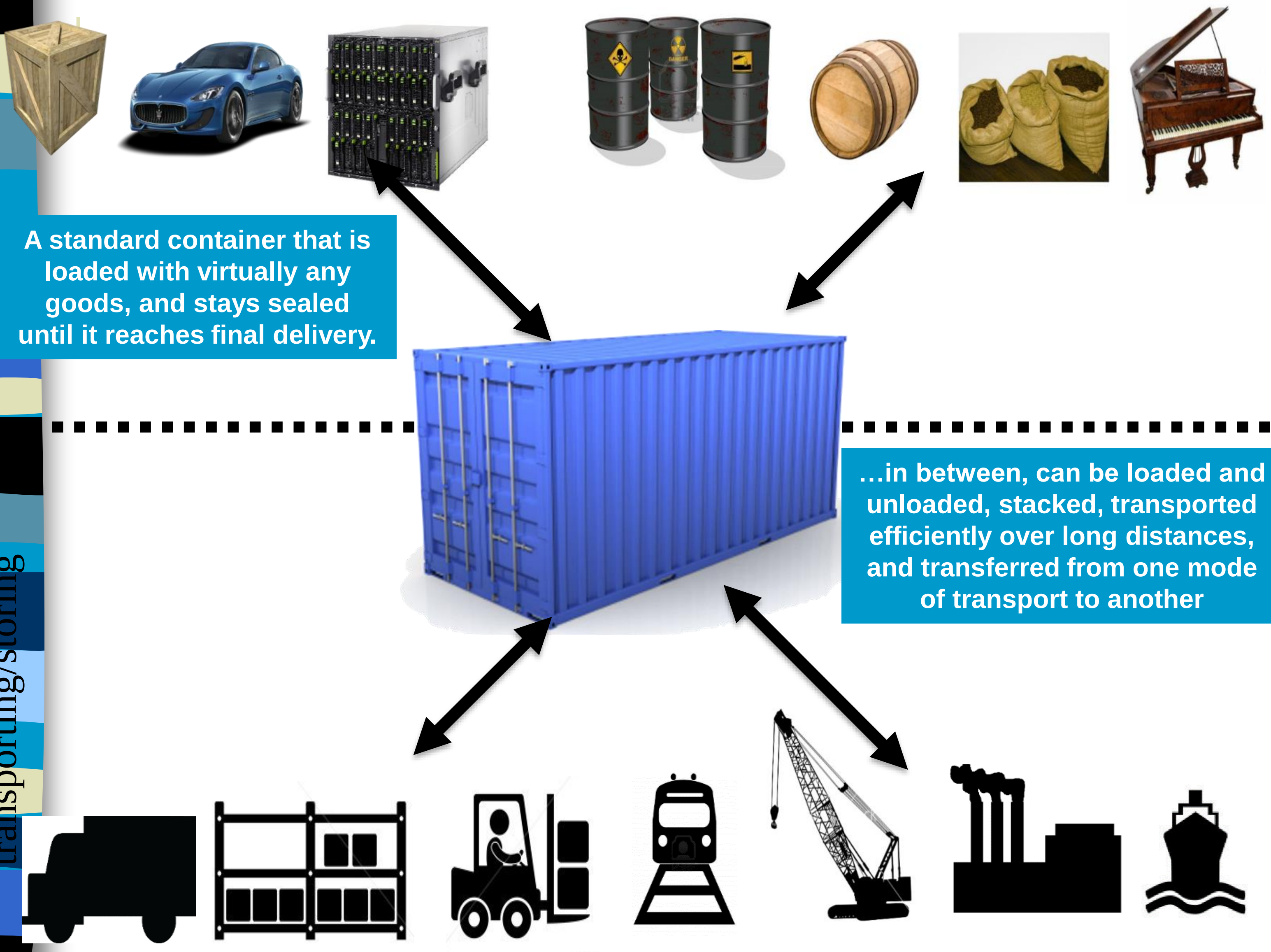
Multiplicity of methods for transporting/storing

quickly and smoothly (e.g. from boat to train to truck)

Matrix Management

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

Solution: Intermodal Shipping Container



Multiplicity of Goods

A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

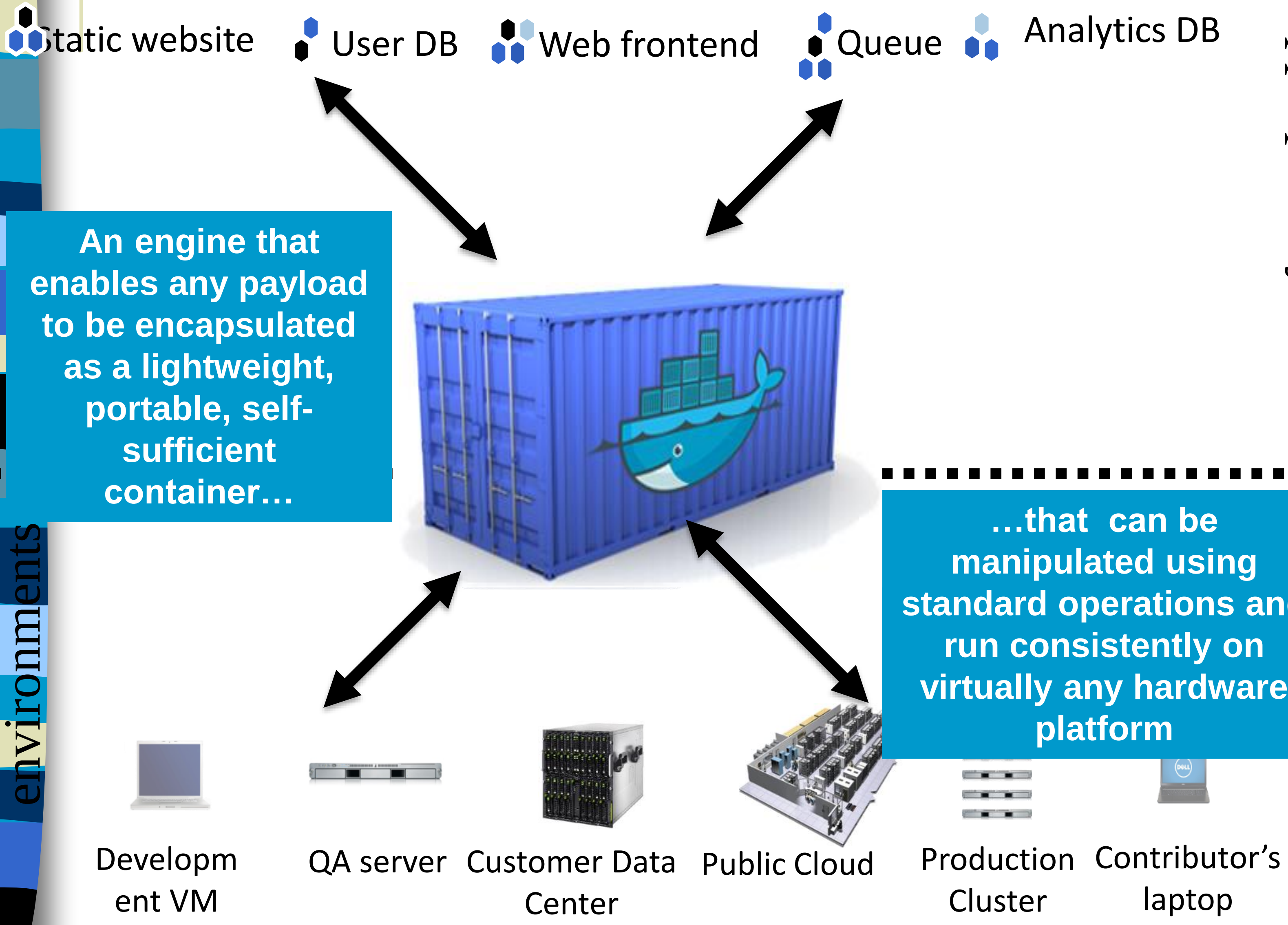
Do I worry about how goods interact (e.g. coffee beans next to spices)

…in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

Multiplicity of methods for transporting/storing

Can I transport quickly and smoothly (e.g. from boat to train to truck)

Docker is a shipping container system for code



An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…

…that can be manipulated using standard operations and run consistently on virtually any hardware platform

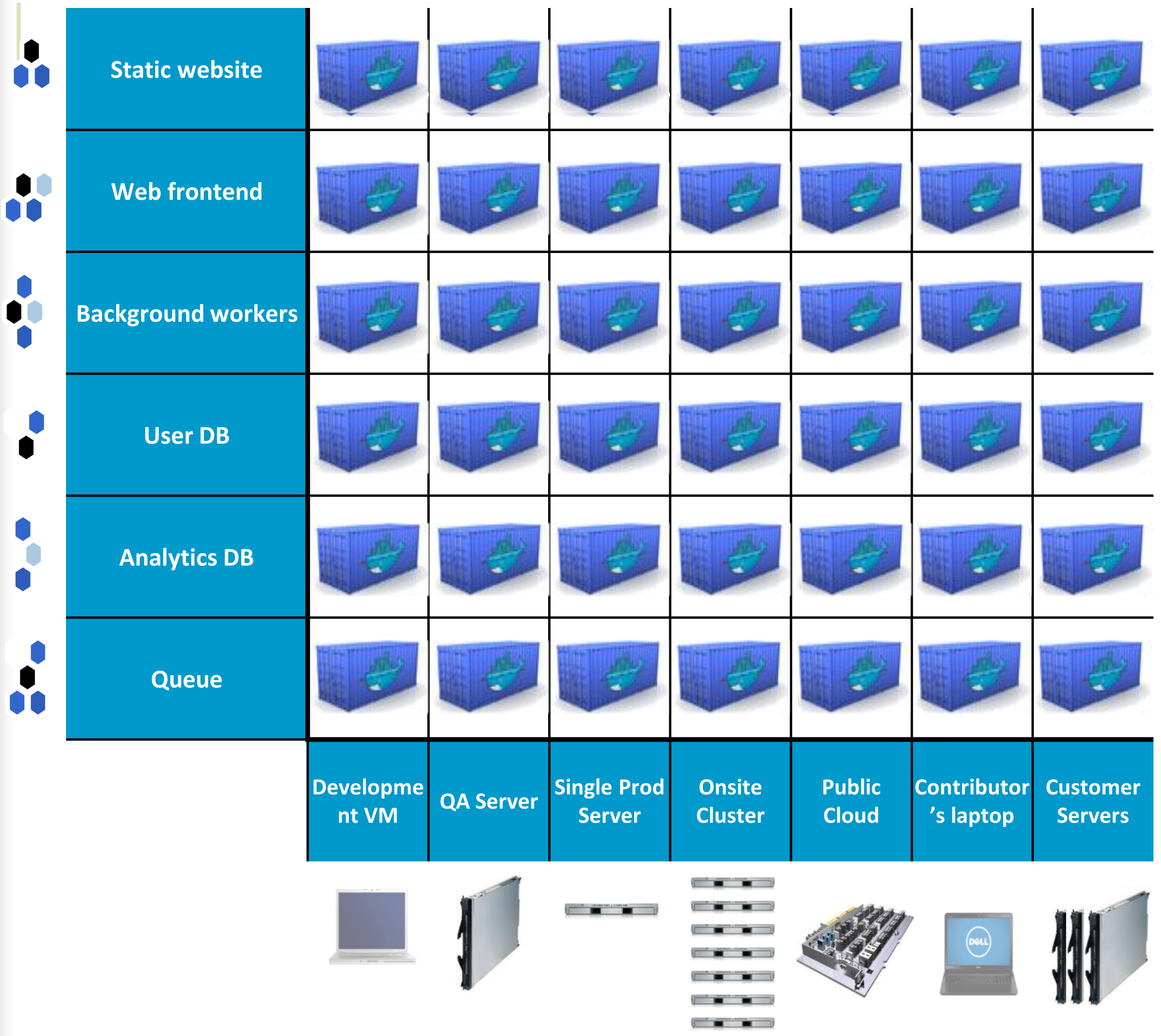
Multiplicity of Stacks

Multiplicity of hardware environments

Do services and apps interact appropriately?

Can I migrate smoothly and quickly

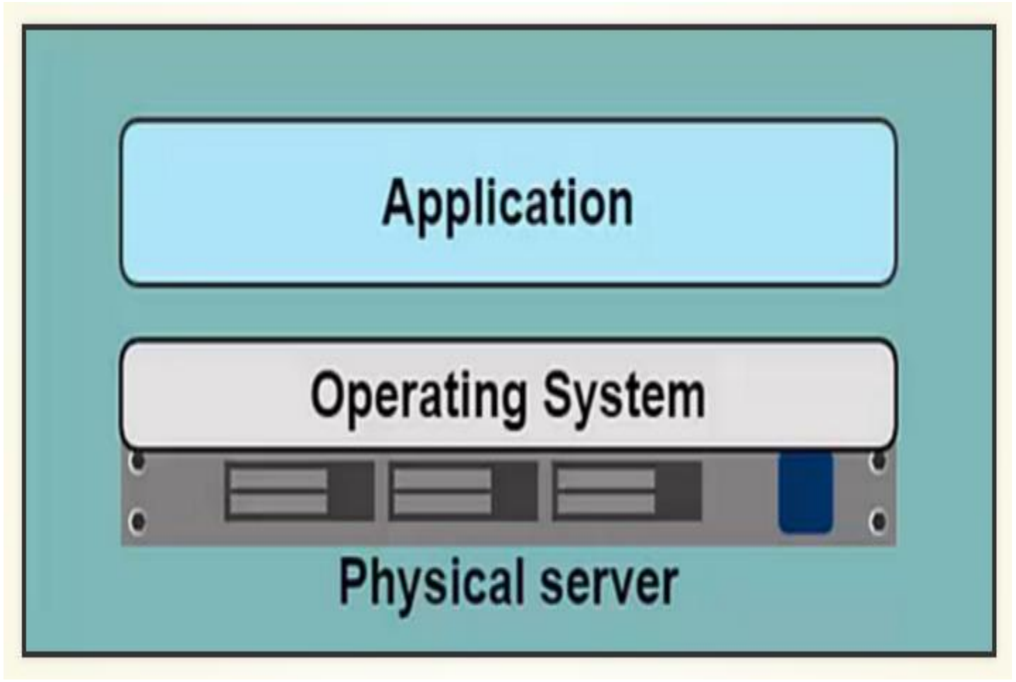
Docker eliminates the matrix management



What is Docker?

- Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux, Mac OS and Windows.
  – Wikipedia

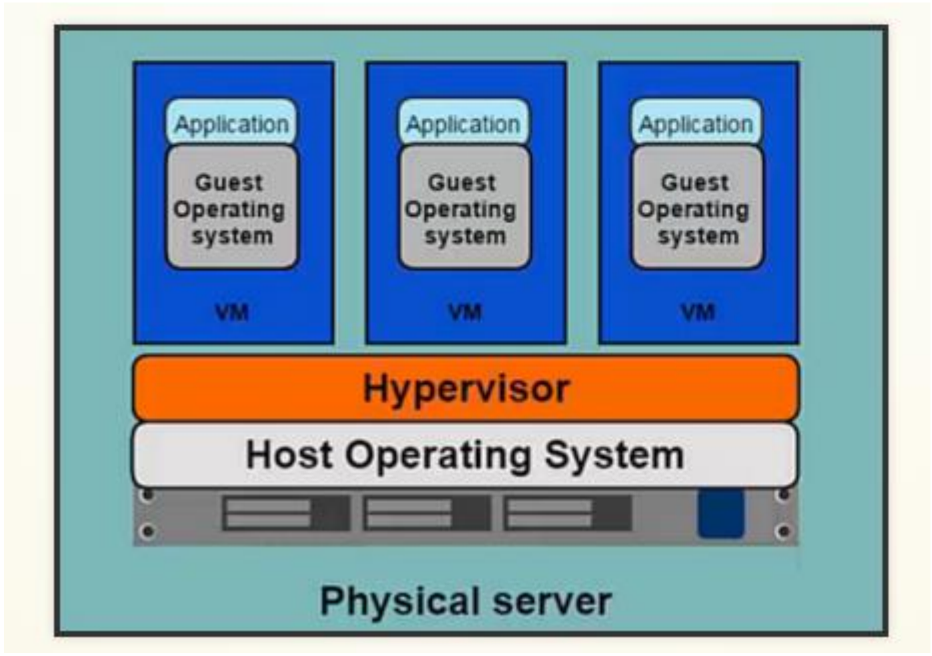
Basic Application Hosting



- Problems
- Slow deployment times
- Huge costs
- Wasted resources
- Difficult to scale or migrate
- Vendor lock in

Hypervisor-based Virtualization

- One physical server can contain multiple applications
- Each application runs in a virtual machine



Pros and Cons

- Better resource pooling
  - one physical machine divided into multiple VM
- Easier to scale
- VM's in the cloud
  - Pay as you go

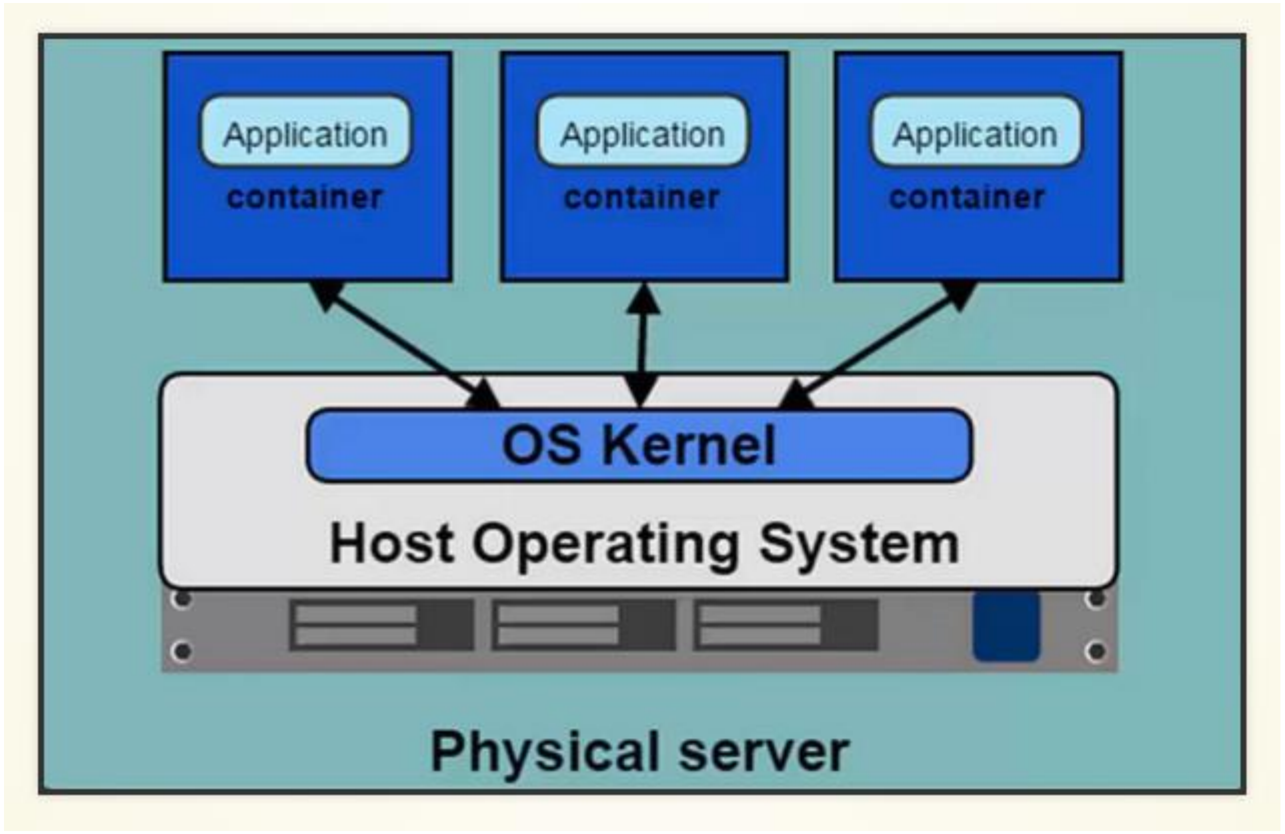
- Each VM stills requires
  - CPU allocation
  - storage
  - RAM
  - An entire guest operation system
- More VM's you run, the more resources you need
- Guest OS means wasted resources

Introducing Containers

Container based virtualization uses the kernel on the host's operating system to run multiple guest instances

- Each guest instance is called a container
- Each container has its own
  - Root filesystem
  - Processes
  - Memory
  - Network ports

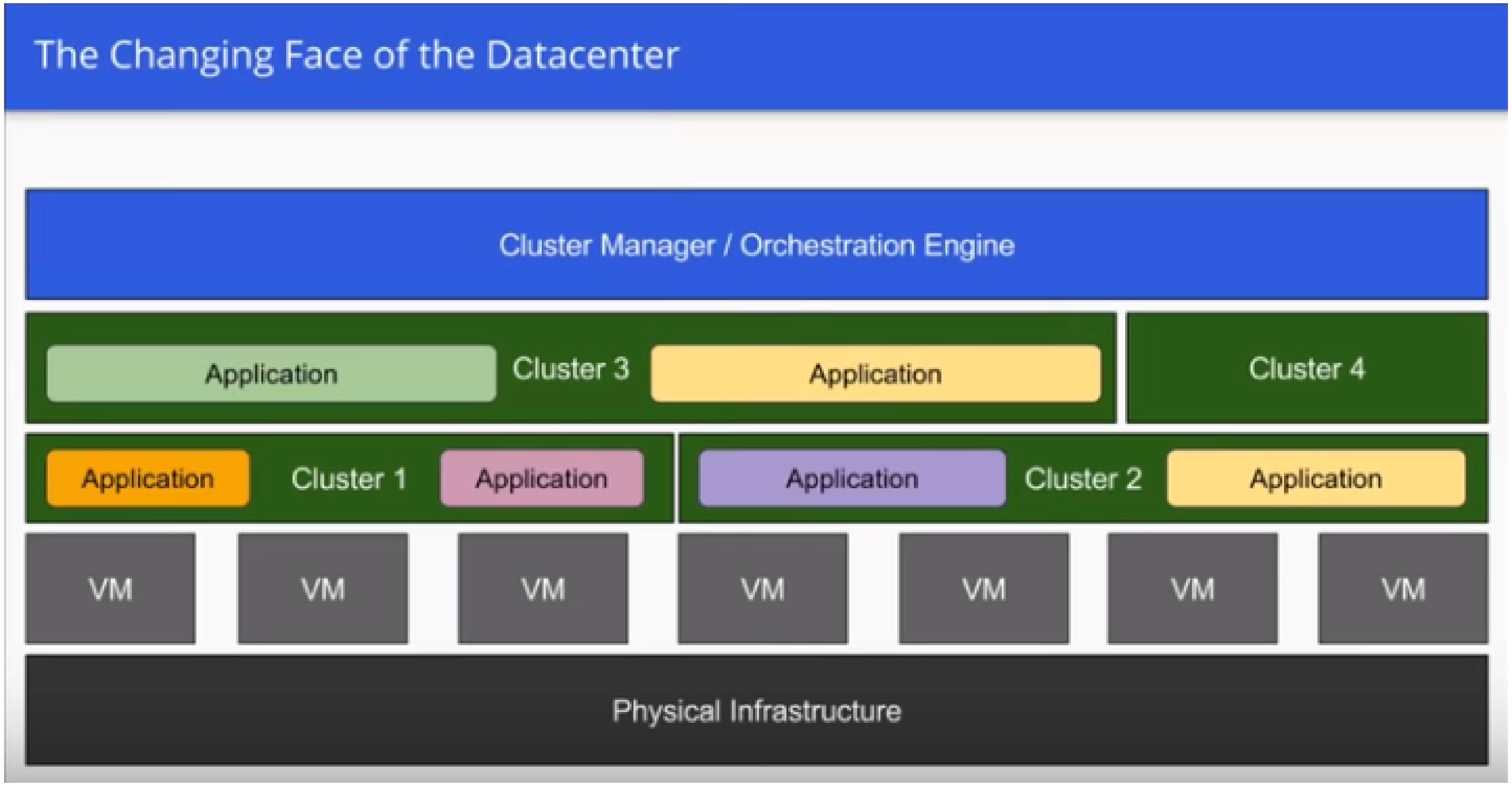
Containers


Application
container
Application
container
Application
container
OS Kernel
Host Operating System
Physical server

Containers Vs VMs

- Containers are more lightweight
- No need to install guest OS
- Less CPU, RAM, storage space required
- More containers per machine than VMs
- Greater portability

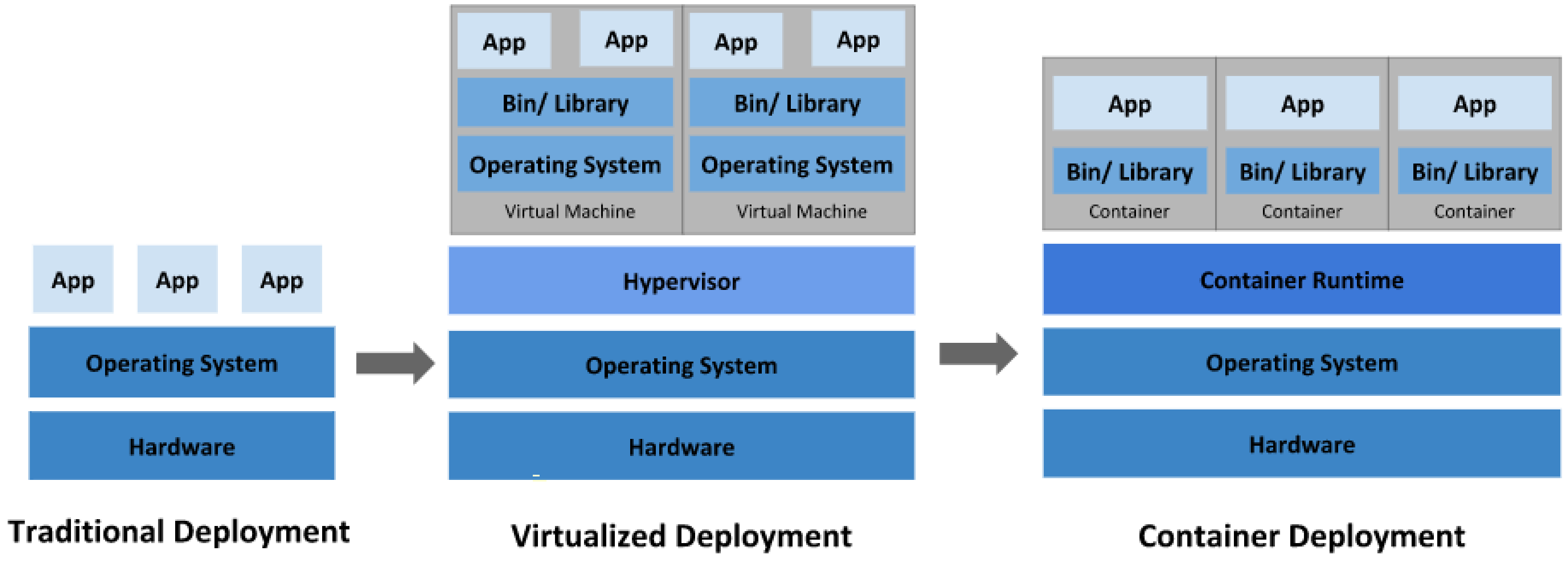
Why Kubernetes?



What Kubernetes?

- Kubernetes is a production-ready, open source platform designed with Google's accumulated experience in container orchestration
- Kubernetes orchestrates the placement (scheduling) and execution of application containers within and across computer clusters

Towards Containerization



Kubernetes

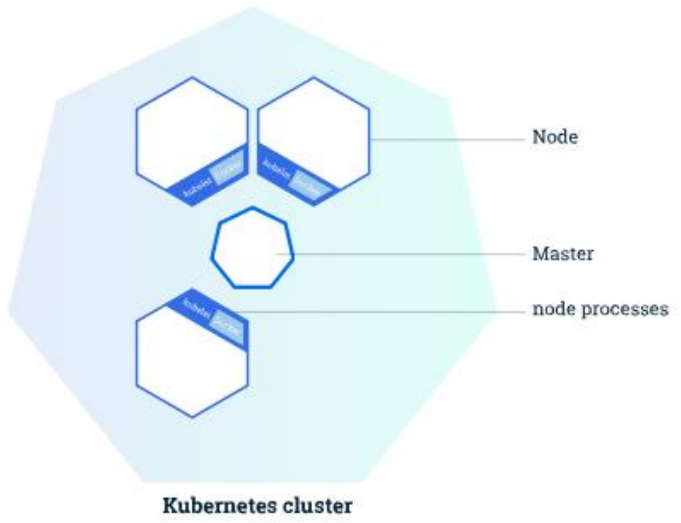
- With modern web services, users expect applications to be available 24/7, and developers expect to deploy new versions of those applications several times a day. Containerization helps package software to serve these goals, enabling applications to be released and updated in an easy and fast way without downtime. Kubernetes helps you make sure those containerized applications run where and when you want, and helps them find the resources and tools they need to work

Kubernetes Clusters

- Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit.
- Kubernetes automates the distribution and scheduling of application containers across a cluster in a more efficient way

Kubernetes Cluster

- A Kubernetes cluster consists of two types of resources:
  - – The Master coordinates the cluster
  - – Nodes are the workers that run applications



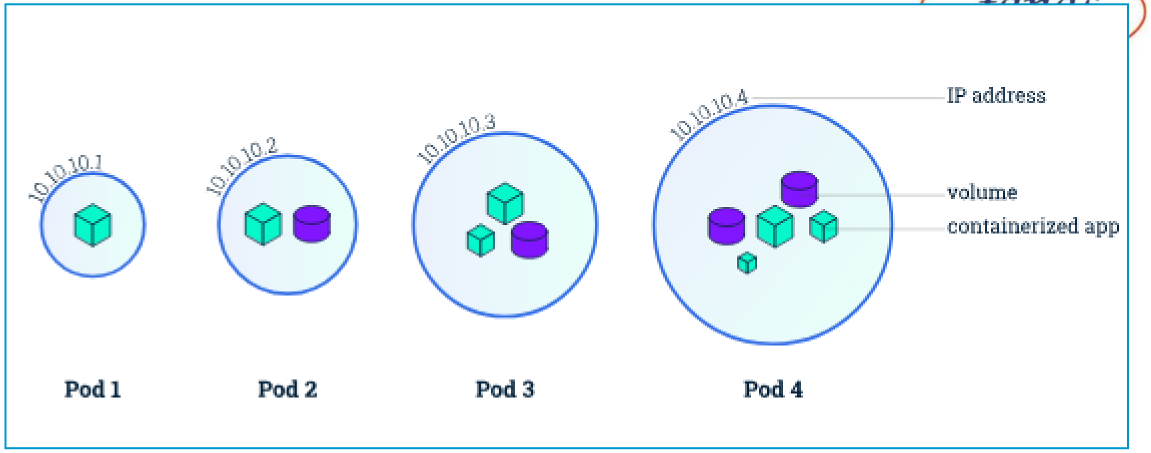
Master and Nodes

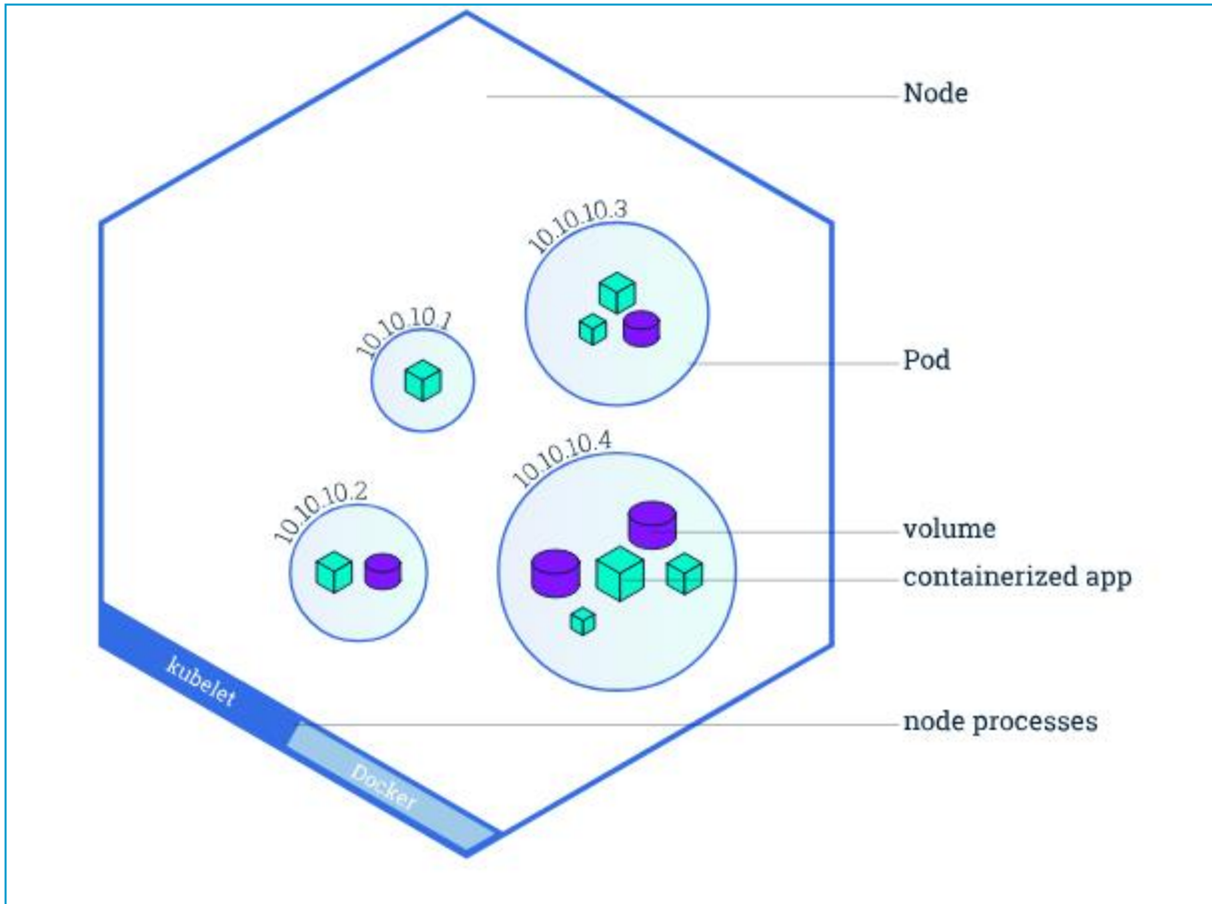
- The Master
  - responsible for managing the cluster.
  - Co-ordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, and rolling out new updates.
- A node
  - is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster.
  - Each node has a Kubelet, which is an agent for managing the node and communicating with the Kubernetes master.
- A node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster.
- The nodes communicate with the master using the Kubernetes API

Kubernetes Pods and Nodes

- A Pod is a Kubernetes abstraction that represents a group of one or more application containers (such as Docker or rkt), and some shared resources for those containers. Those resources include:
  - – Shared storage, as Volumes
  - – Networking, as a unique cluster IP address
  - – Information about how to run each container, such as the container image version or specific ports to use

Pods and Nodes





Kubernetes Pods and Nodes

- A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster.
- Pod always runs on a **Node**.
- Each Node is managed by the Master. A Node can have multiple pods, and the Kubernetes master automatically handles scheduling the pods across the Nodes in the cluster. The Master's automatic scheduling takes into account the available resources on each Node.

Node

- Every Kubernetes Node runs at least:
  – Kubelet, a process responsible for communication between the Kubernetes Master and the Node; it manages the Pods and the containers running on a machine.
  – A container runtime (like Docker, rkt) responsible for pulling the container image from a registry, unpacking the container, and running the application.

Kubernetes Architecture

Cluster up and running

- Check Version
  - minikube version
- Start Cluster
  - minikube start
- Cluster version
  - kubectl version

Cluster up and running

- Cluster details
  - kubectl cluster-info
  - kubectl get nodes

Using kubectl to Create a Deployment

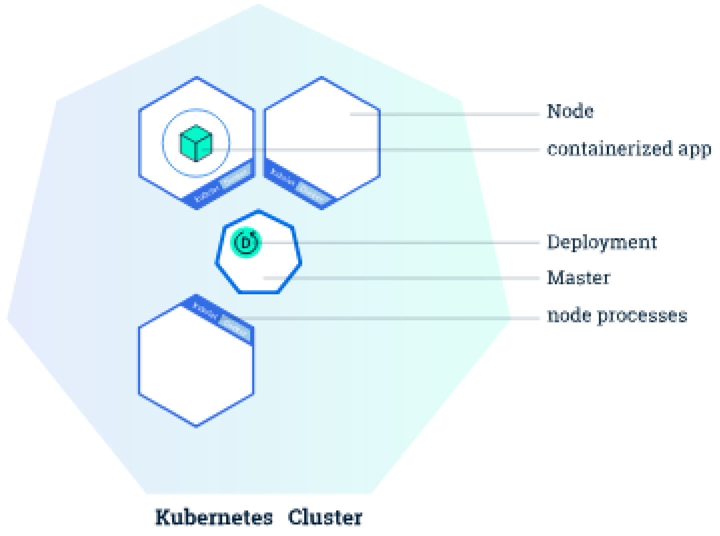
- Kubernetes Deployments
  - Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a **Deployment** configuration. The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes master schedules mentioned application instances onto individual Nodes in the cluster.

Kubernetes Deployments

- Once the application instances are created, a Kubernetes Deployment Controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces it. **This provides a self-healing mechanism to address machine failure or maintenance.**

Deploying App on Kubernetes

- kubectl version
- kubectl get nodes
- kubectl run kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1 --port=8080



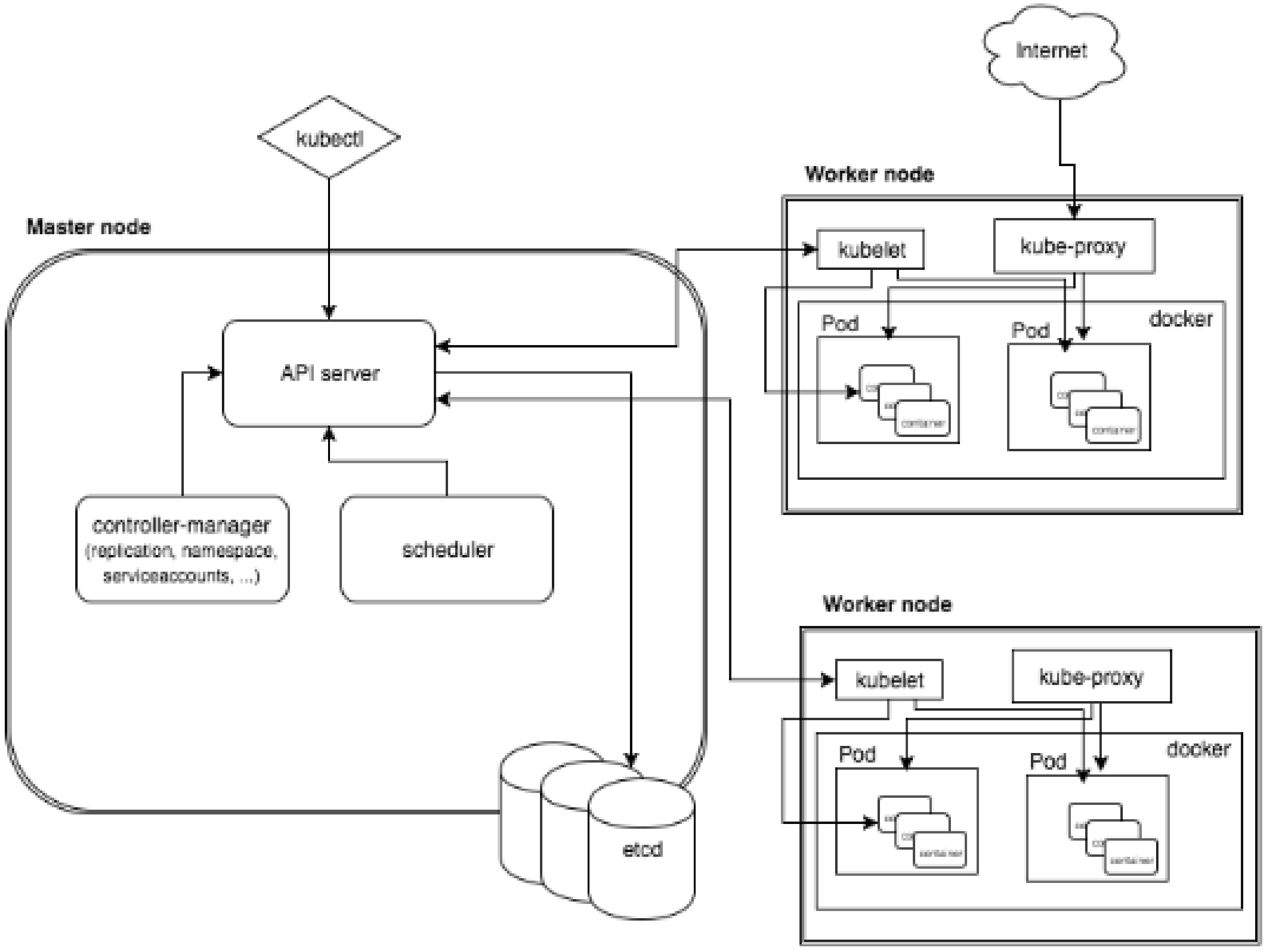
Troubleshooting with kubectl

- kubectl get - list resources
- kubectl describe - show detailed information about a resource
- kubectl logs - print the logs from a container in a pod
- kubectl exec - execute a command on a container in a pod

Kubernetes Features

- Replication of components
- Auto-scaling
- Load balancing
- Rolling updates
- Logging across components
- Monitoring and health checking
- Service discovery
- Authentication

High Level Kubernetes Architecture



Main Components

- Master Node(s)
- Worker Nodes
- Distributed Key-Value Store like etcd

Master Node

- The master node
  - is responsible for the management of Kubernetes cluster
  - is the entry point of all administrative tasks.
  - is the one taking care of orchestrating the worker nodes, where the actual services are running.
  - Can be more than one master nodes in the cluster
  - Only one of them will be the leader

Master Node Components

- Four Components
  - API server
  - scheduler
  - etcd storage
  - controller-manager

API server

- The API server is the entry points for all the REST commands used to control the cluster. It processes the REST requests, validates them, and executes the bound business logic. The result state has to be persisted somewhere, and that brings us to the next component of the master node.

Scheduler

- The deployment of configured pods and services onto the nodes happens thanks to the scheduler component.
- The scheduler has the information regarding resources available on the members of the cluster, as well as the ones required for the configured service to run and hence is able to decide where to deploy a specific service.

fands™

ETCD Storage

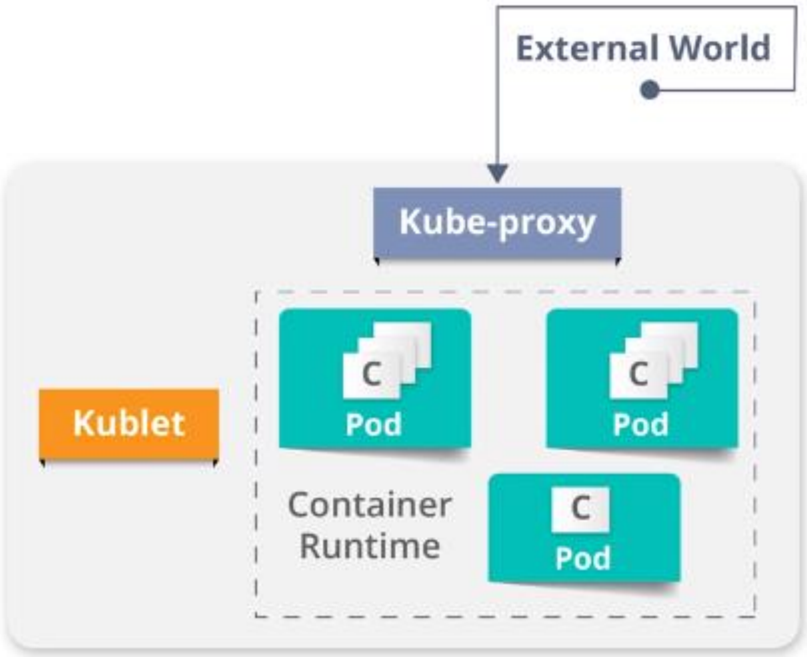
- etcd is a simple, distributed, consistent key-value store. It’s mainly used for shared configuration and service discovery.
- It provides a REST API for CRUD operations as well as an interface to register watchers on specific nodes, which enables a reliable way to notify the rest of the cluster about configuration changes.
- An example of data stored by Kubernetes in etcd is jobs being scheduled, created and deployed, pod/service details and state, namespaces and replication information, etc.

Controller-Manager

- Optionally you can run different kinds of controllers inside the master node. controller-manager is a daemon embedding those.
- A controller uses apiserver to watch the shared state of the cluster and makes corrective changes to the current state to change it to the desired one.
- An example of such a controller is the Replication controller, which takes care of the number of pods in the system. The replication factor is configured by the user, and it's the controller’s responsibility to recreate a failed pod or remove an extra-scheduled one.
- Other examples of controllers are endpoints controller, namespace controller, and serviceaccounts controller

Worker Node

- The pods are run here, so the worker node contains all the necessary services to manage the networking between the containers, communicate with the master node, and assign resources to the containers scheduled.



Worker Node Components

- Three Components
  - – Container Runtime (Docker)
  - – kubelet
  - – kube-proxy

Container Runtime

- Container runtime is used to manage containers lifecycle on worker node. Docker is a platform to use container as container runtime.
- Docker runs on each of the worker nodes, and runs the configured pods. It takes care of downloading the images and starting the containers.

kubelet

- Kubelet gets the configuration of a pod from the apiserver and ensures that the described containers are up and running. This is the worker service that’s responsible for communicating with the master node.
- It also communicates with etcd, to get information about services and write the details about newly created ones

kube-proxy

- kube-proxy acts as a network proxy and a load balancer for a service on a single worker node. It takes care of the network routing for TCP and UDP packets.

Kubernetes Concepts (Re-Visit)

- Making use of Kubernetes requires understanding the different abstractions it uses to represent the state of the system
- Pod - generally refers to one or more containers that should be controlled as a single application. A pod encapsulates application containers, storage resources, a unique network ID and other configuration on how to run the containers.
- Service - pods are volatile, that is Kubernetes does not guarantee a given physical pod will be kept alive. Instead, a service represents a logical set of pods and acts as a gateway, allowing (client) pods to send requests to the service without needing to keep track of which physical pods actually make up the service.

Kubernetes Concepts (Re-Visit)

- Volume - similar to a container volume in Docker, but a Kubernetes volume applies to a whole pod and is mounted on all containers in the pod. Kubernetes guarantees data is preserved across container restarts. The volume will be removed only when the pod gets destroyed. Also, a pod can have multiple volumes (possibly of different types) associated.
- Namespace - a virtual cluster (a single physical cluster can run multiple virtual ones) intended for environments with many users spread across multiple teams or projects, for isolation of concerns. Resources inside a namespace must be unique and cannot access resources in a different namespace. Also, a namespace can be allocated a resource quota to avoid consuming more than its share of the physical cluster’s overall resources.

Kubernetes Concepts (Re-Visit)

- Deployment - describes the desired state of a pod or a replica set, in a yaml file. The deployment controller then gradually updates the environment (for example, creating or deleting replicas) until the current state matches the desired state specified in the deployment file. For example, if the yaml file defines 2 replicas for a pod but only one is currently running, an extra one will get created. Note that replicas managed via a deployment should not be manipulated directly, only via new deployments.

Deployment Vs Service

- A deployment is responsible for keeping a set of pods running.
- A service is responsible for enabling network access to a set of pods.
- We could use a deployment without a service to keep a set of identical pods running in the Kubernetes cluster. The deployment could be scaled up and down and pods could be replicated. Services and Deployments are different, but they work together nicely.

Deployment Vs Service

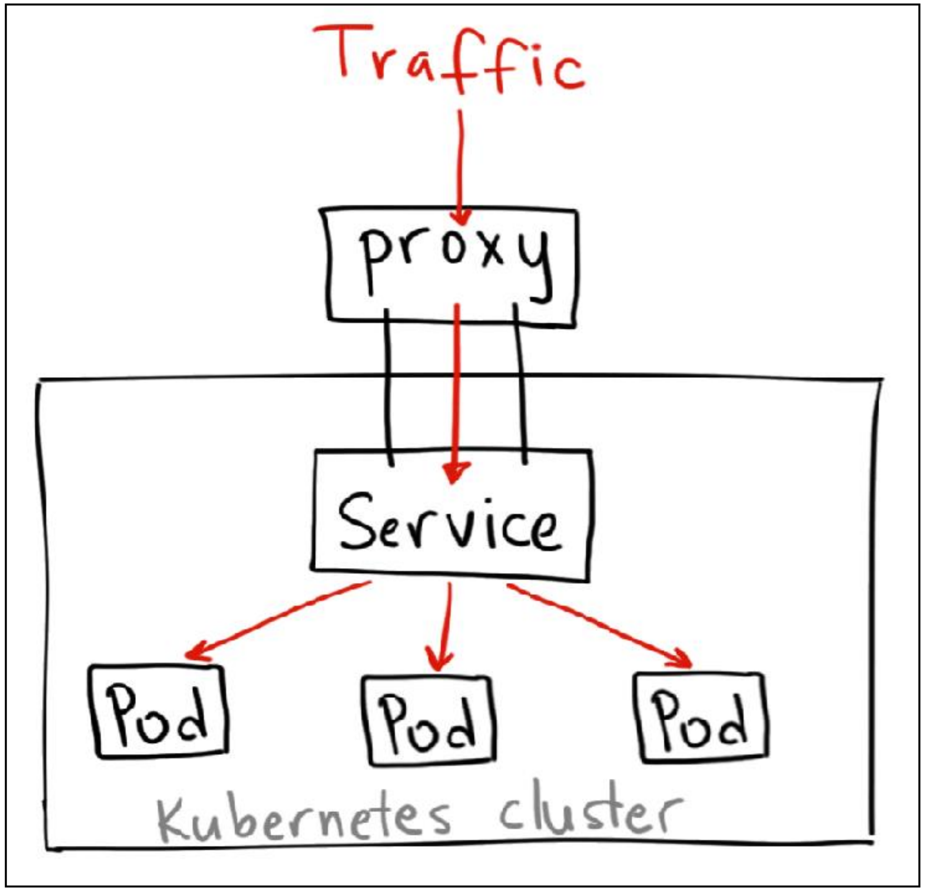
- Each pod could be accessed individually via direct network requests (rather than abstracting them behind a service), but keeping track of this for a lot of pods is difficult.
- We could also use a service without a deployment. We'd need to create each pod individually (rather than "all-at-once" like a deployment). Then our service could route network requests to those pods via selecting them based on their labels.

ClusterIP, NodePort and LoadBalancer

- The type property – service <-> network
- Options
  - ClusterIP : (default) - only accessible from within the Kubernetes cluster or from proxy
  - NodePort : accessible on a static port on each Node in the cluster. This means that the service is accessible outside the cluster without proxy
  - LoadBalancer : The service becomes accessible externally through a cloud provider's load balancer functionality. GCP, AWS, Azure, and OpenStack offer this functionality. The cloud provider will create a load balancer, which then automatically routes requests to your Kubernetes Service
  - 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.

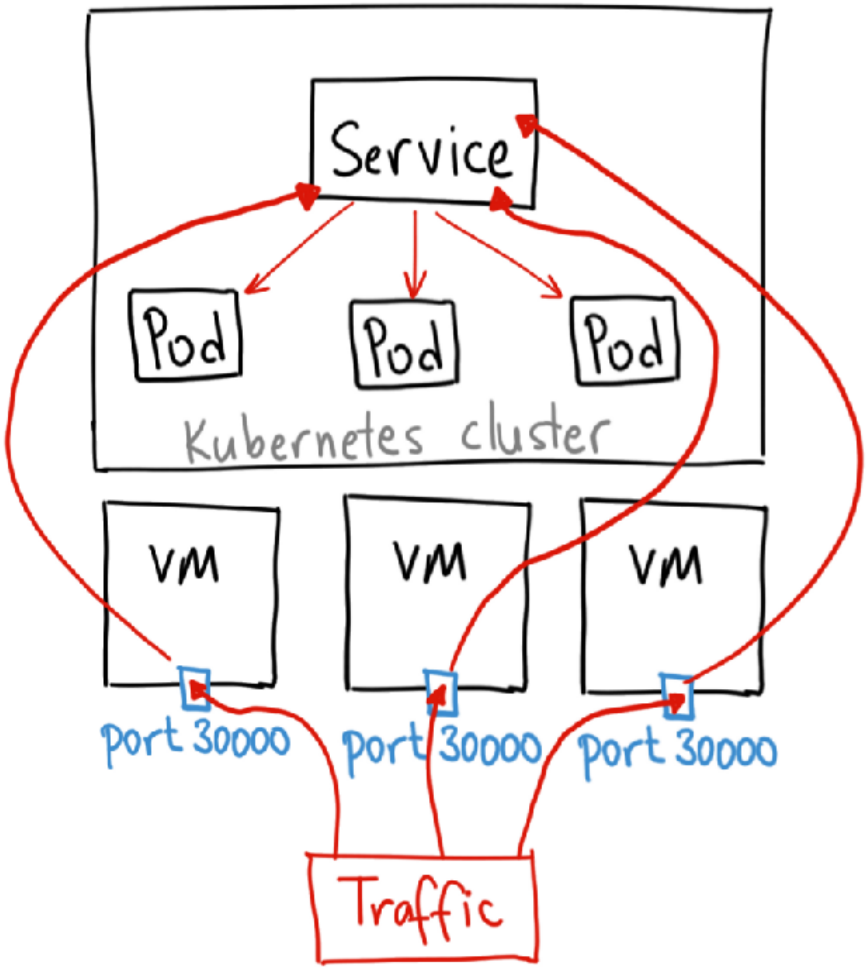
ClusterIP

```
apiVersion: v1
kind: Service
metadata:
  name: my-internal-service
spec:
  selector:
    app: my-app
  type: ClusterIP
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
```

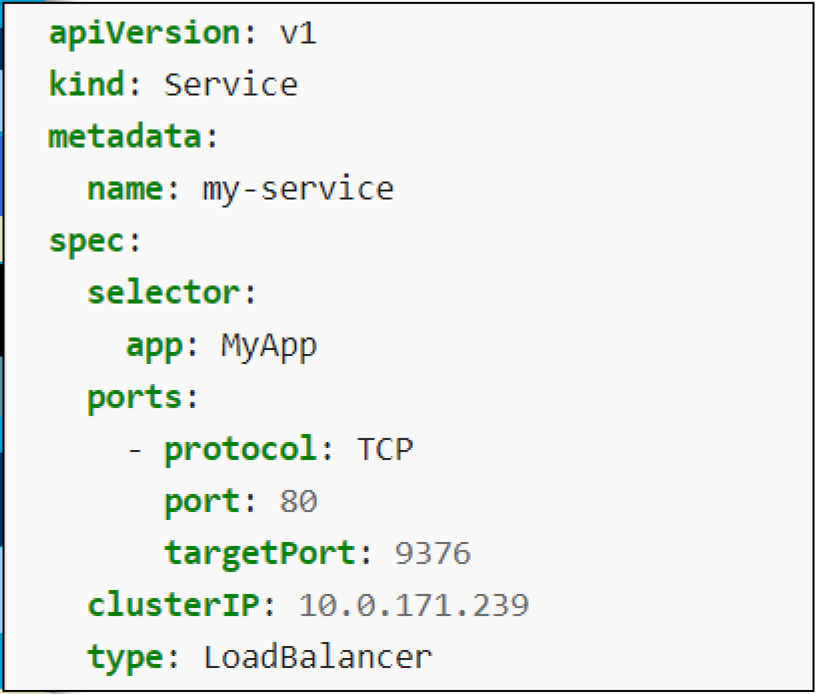


NodePort

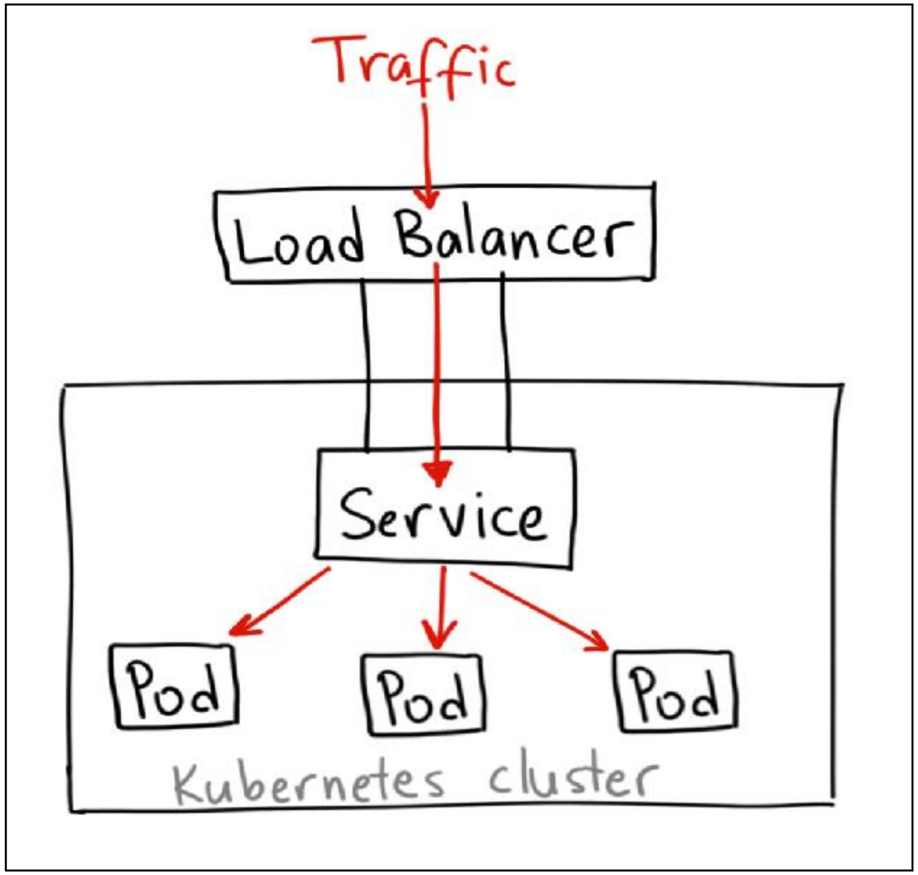
```
apiVersion: v1
kind: Service
metadata:
  name: my-nodeport-service
spec:
  selector:
    app: my-app
  type: NodePort
  ports:
  - name: http
    port: 80
    targetPort: 80
    nodePort: 30036
    protocol: TCP
```



Load Balancer

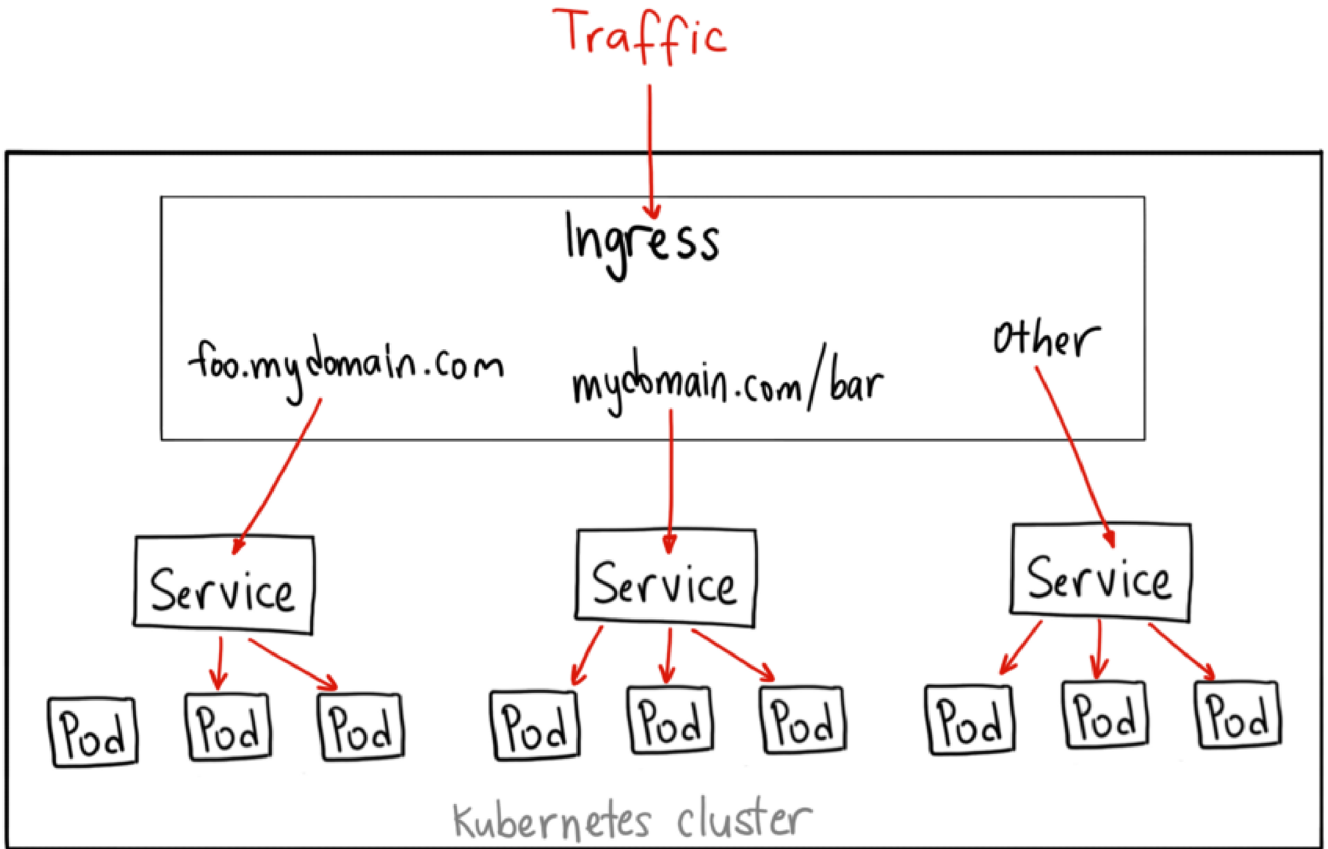


```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
  clusterIP: 10.0.171.239
  type: LoadBalancer
```



Ingress

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

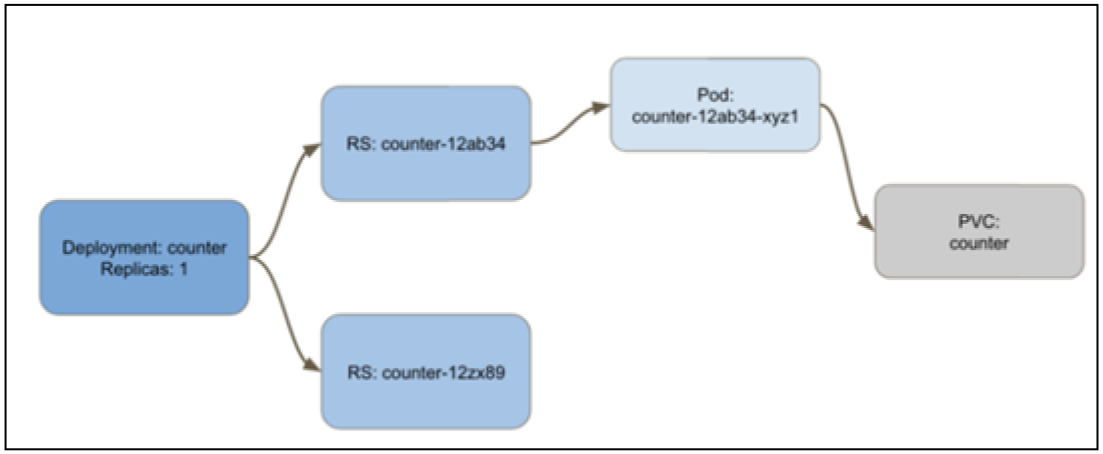


Pod Deployment with

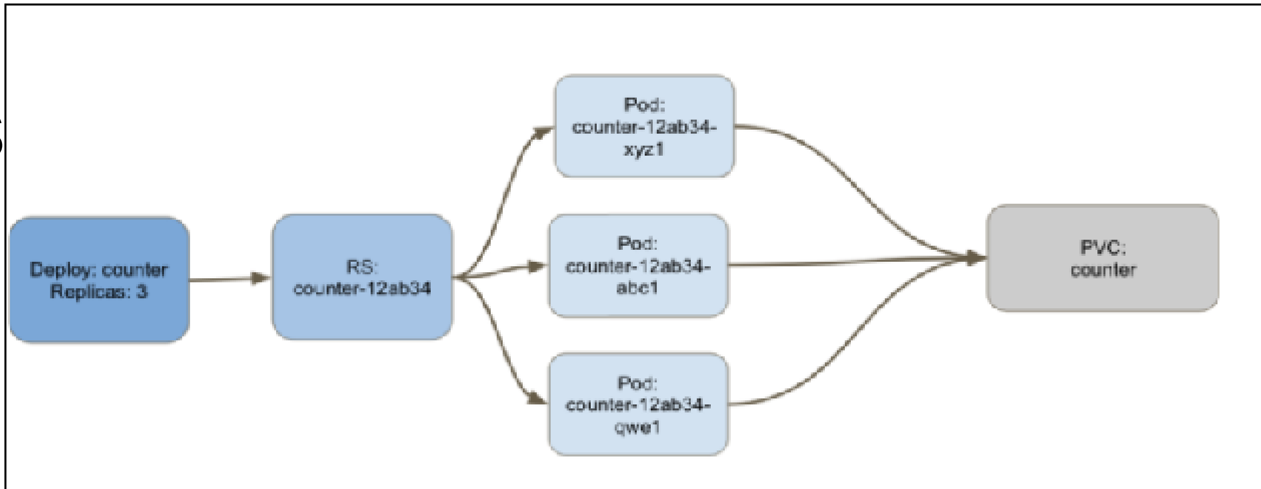
- Kubernetes using different Kubernetes resources. Below are 3 different resources that Kubernetes provides for deploying pods.
  - Deployments (replicaset)
  - StatefulSets
  - DaemonSets

Deployment (Single Pod)

- Single Pod



- Replicas

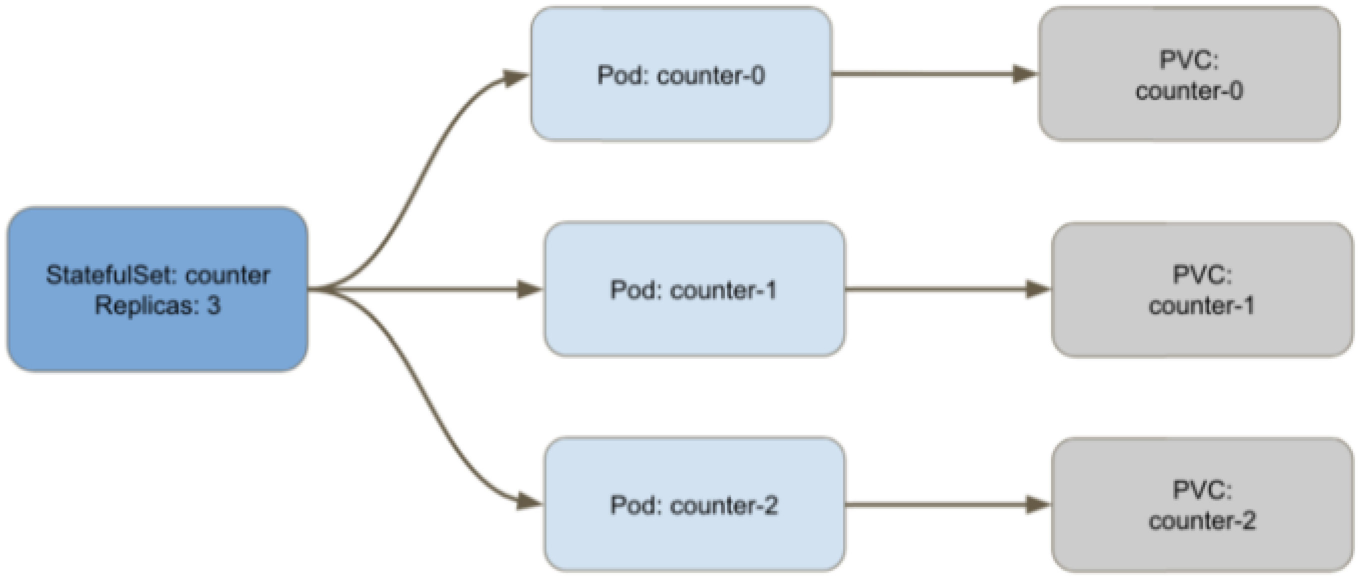


Persistence for Deployments sharing single Volume can cause Data Inconsistency

StatefulSets

- It manages the deployment and scaling of a set of Pods, and provides guarantee about the ordering and uniqueness of these Pods.
- Every replica of a stateful set will have its own state, and each of the pods will be creating its own PVC(Persistent Volume Claim). A statefulset with 3 replicas will create 3 pods, each having its own Volume, so total 3 PVCs.

StatefulSets



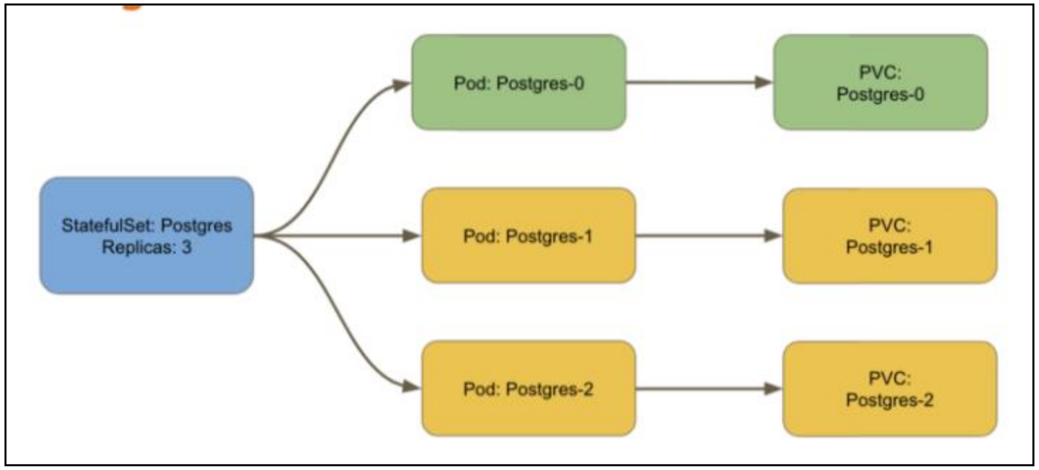
Persistence for StatefulSets each having its own Volume

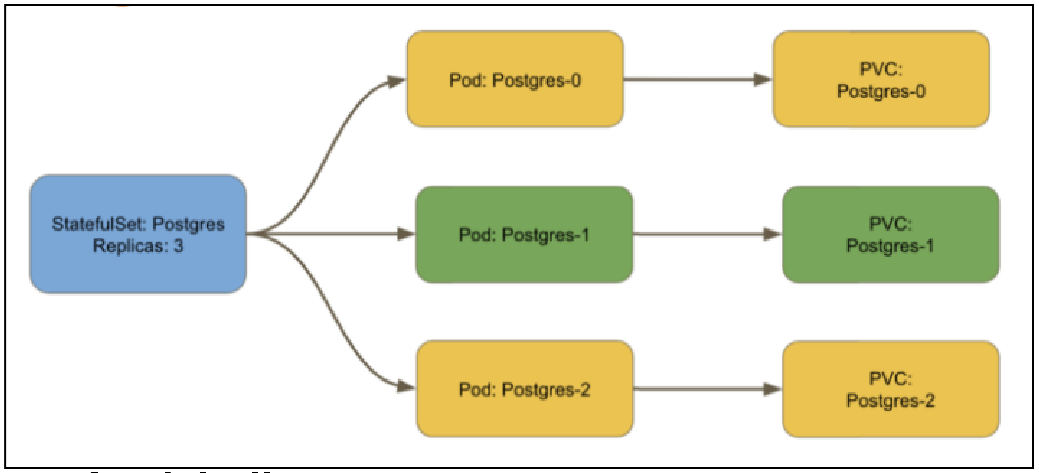
StatefulSets

- StatefulSets don’t create ReplicaSet or anything of that sort, so you cant rollback a StatefulSet to a previous version.
- Supports RollingUpdate
- Really useful for highly available databases / service in production

StatefulSets

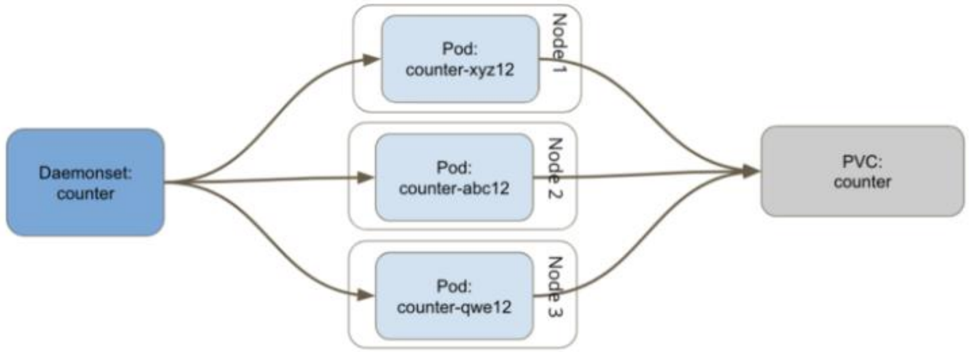
- Starts with first one as Primary Replica
- If the primary goes down, any of the secondary replica will become primary and the StatefulSet controller will create a new replica in account of the one that went down, which will now become a secondary replica.





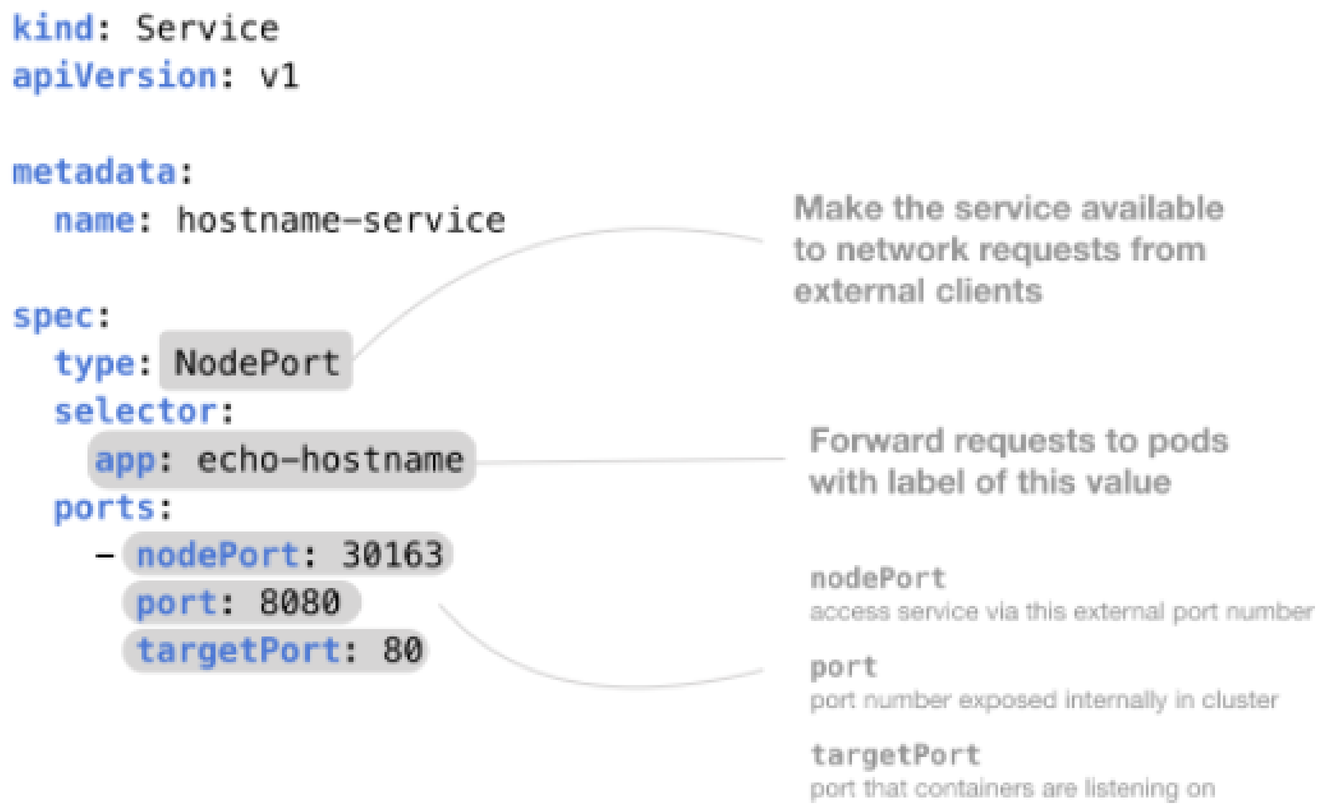
DaemonSet

- A DaemonSet is a controller that ensures that the pod runs on all the nodes of the cluster. If a node is added/removed from a cluster, DaemonSet automatically adds/deletes the pod.
  – Loggers
  – Monitoring



Each Replica running on each

Yml



```
kind: Service
apiVersion: v1

metadata:
  name: hostname-service

spec:
  type: NodePort
  selector:
    app: echo-hostname
  ports:
    - nodePort: 30163
      port: 8080
      targetPort: 80
```

Make the service available to network requests from external clients

Forward requests to pods with label of this value

`nodePort`
access service via this external port number

`port`
port number exposed internally in cluster

`targetPort`
port that containers are listening on

Yml

```
kind: Service
apiVersion: v1
metadata:
  name: hostname-service
spec:
  # Expose the service on a static port on each node
  # so that we can access the service from outside the cluster
  type: NodePort
  # When the node receives a request on the static port (30163)
  # "select pods with the label 'app' set to 'echo-hostname'"
  # and forward the request to one of them
 selector:
    app: echo-hostname
  ports:
    # Three types of ports for a service
    # nodePort - a static port assigned on each the node
    # port - port exposed internally in the cluster
    # targetPort - the container port to send requests to
    - nodePort: 30163
      port: 8080
      targetPort: 80
```

ConfigMaps and Secrets

ConfigMaps

- A ConfigMap is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.
- A ConfigMap allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable.

ConfigMap object

- Lets you store configuration for other objects to use.
- ConfigMap has data and binaryData fields.
  - Accept key-value pairs as their values.
  - Both are optional.
  - The data field can contain UTF-8 byte sequences while the binaryData field can contain binary data as base64-encoded strings.
- The name of a ConfigMap must be a valid DNS subdomain name.
- Starting from v1.19, you can add an immutable field to a ConfigMap definition to create an immutable ConfigMap.

ConfigMap yml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

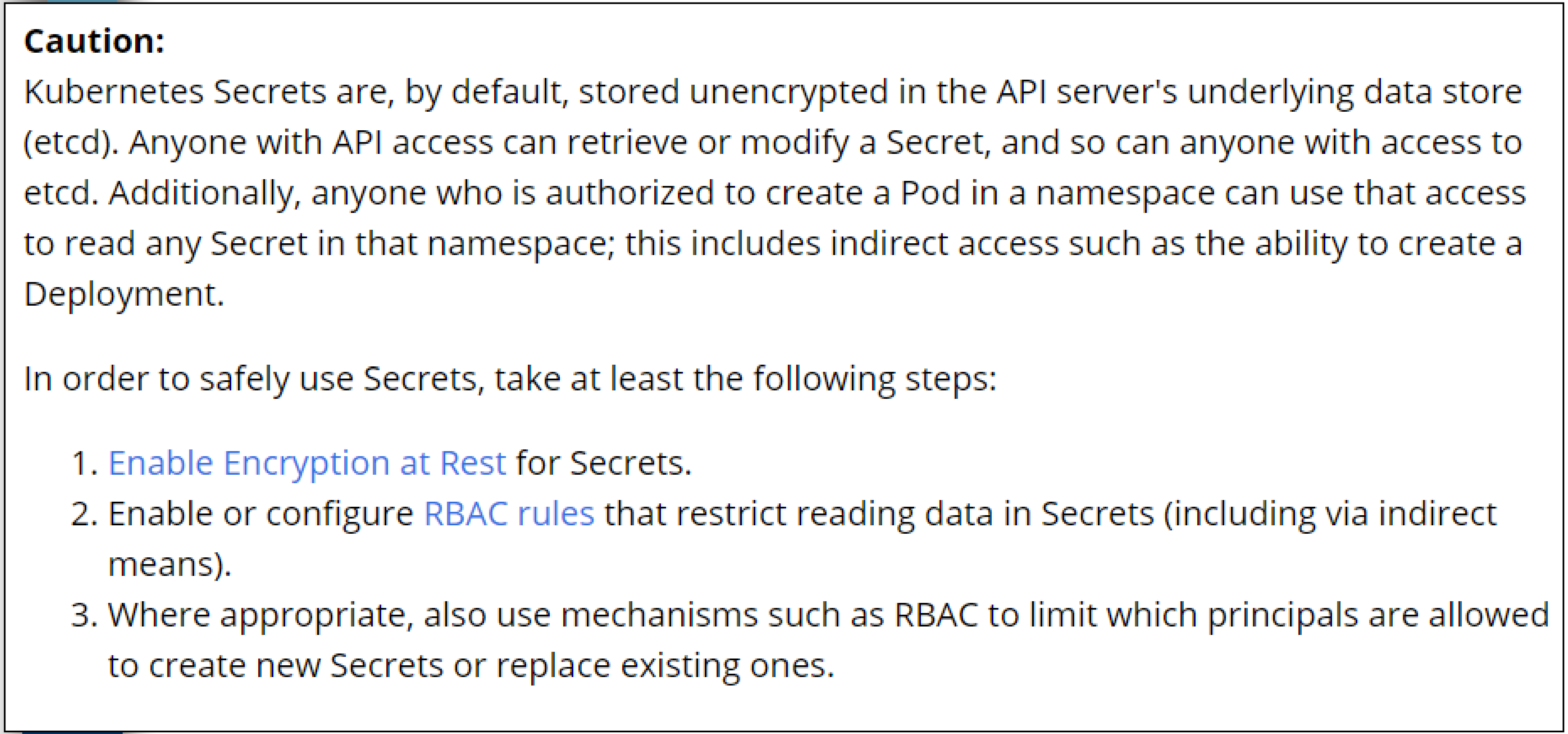
4 Ways to Configure

- Inside a container command and args
- Environment variables for a container
- Add a file in read-only volume, for the application to read
- Write code to run inside the Pod that uses the Kubernetes API to read a ConfigMap

Secrets

- A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a Pod specification or in a container image. Using a Secret means that you don't need to include confidential data in your application code.
- Secrets are similar to ConfigMaps but are specifically intended to hold confidential data.

Caution from K8S



Caution:

Kubernetes Secrets are, by default, stored unencrypted in the API server's underlying data store (etcd). Anyone with API access can retrieve or modify a Secret, and so can anyone with access to etcd. Additionally, anyone who is authorized to create a Pod in a namespace can use that access to read any Secret in that namespace; this includes indirect access such as the ability to create a Deployment.

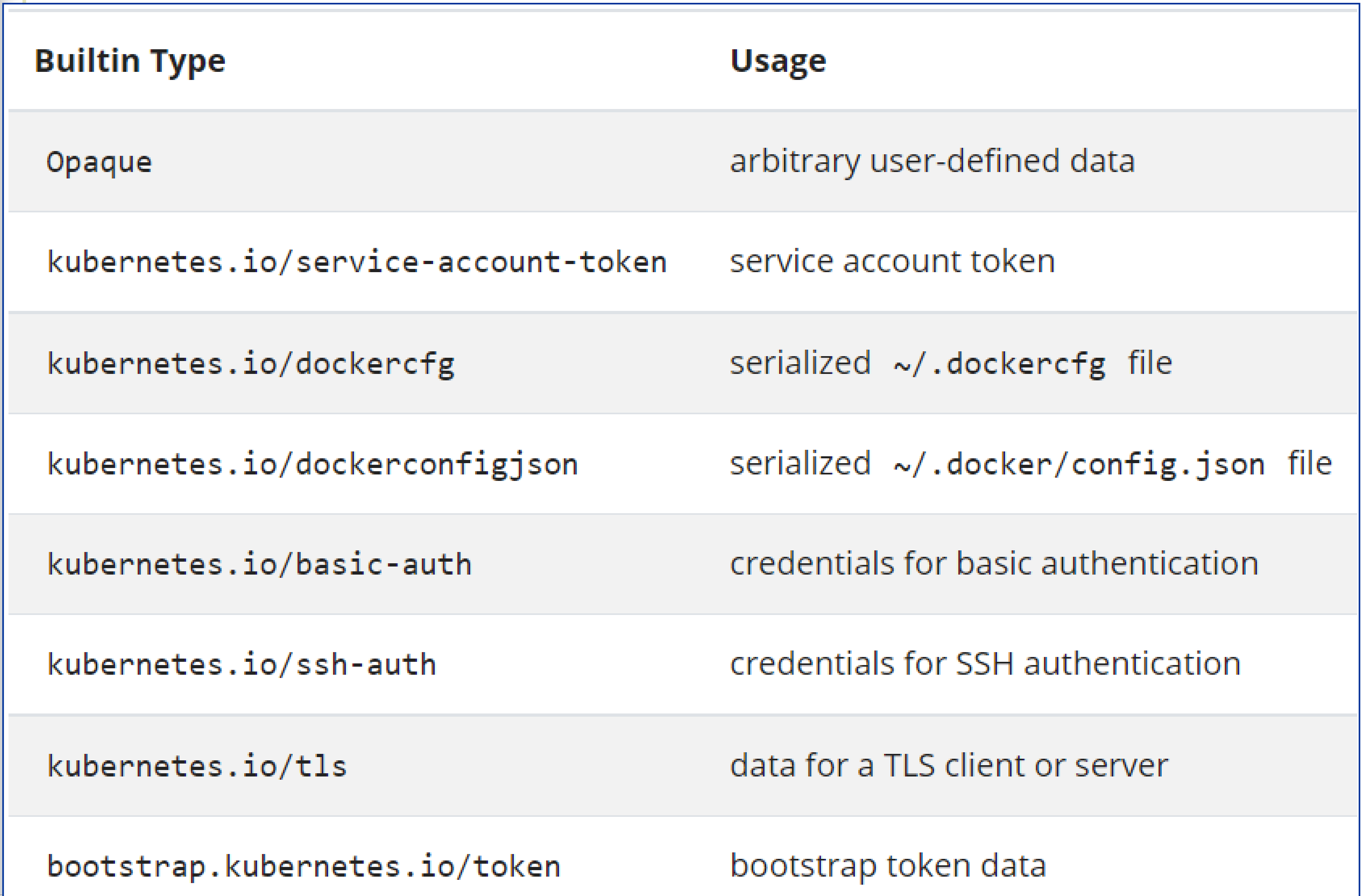
In order to safely use Secrets, take at least the following steps:

1. Enable Encryption at Rest for Secrets.
2. Enable or configure RBAC rules that restrict reading data in Secrets (including via indirect means).
3. Where appropriate, also use mechanisms such as RBAC to limit which principals are allowed to create new Secrets or replace existing ones.

Overview of Secrets

- To use a Secret, a Pod needs to reference the Secret. A Secret can be used with a Pod in three ways:
  – As files in a volume mounted on one or more of its containers.
  – As container environment variable.
  – By the kubelet when pulling images for the Pod.

Types of Secret



Builtin Type	Usage
`Opaque`	arbitrary user-defined data
`kubernetes.io/service-account-token`	service account token
`kubernetes.io/dockercfg`	serialized `~/.dockercfg` file
`kubernetes.io/dockerconfigjson`	serialized `~/.docker/config.json` file
`kubernetes.io/basic-auth`	credentials for basic authentication
`kubernetes.io/ssh-auth`	credentials for SSH authentication
`kubernetes.io/tls`	data for a TLS client or server
`bootstrap.kubernetes.io/token`	bootstrap token data

A Fast AND Steady Approach

Jobs and CronJobs

Jobs and CronJobs

- Job creates one or more pods and ensures that a specified number of the pods terminates when the task (Job) completes.
- Create pods for automation and much more using
  - Jobs
  - CronJobs

Jobs Vs CronJobs

- Kubernetes Jobs are used to create transient pods that perform specific tasks they are assigned to. CronJobs just do do it based on a defined schedule.
- Jobs play an important role in Kubernetes, especially for running batch processes or important ad-hoc operations. Jobs differ from other Kubernetes controllers in that they run tasks until completion, rather than managing the desired state such as in Deployments, ReplicaSets, and StatefulSets.

Cron schedule syntax

```
#       ┌─────────────────── timezone (optional)
#       |          ┌───────────── minute (0 - 59)
#       |          │ ┌───────────── hour (0 - 23)
#       |          │ │ ┌───────────── day of the month (1 - 31)
#       |          │ │ │ ┌───────────── month (1 - 12)
#       |          │ │ │ │ ┌───────────── day of the week (0 - 6) (Sunday to Saturday;
#       |          │ │ │ │ │                                   7 is also Sunday on some systems)
#       |          │ │ │ │ │
#       |          │ │ │ │ │
# CRON_TZ=UTC * * * * *
```

Entry	Description	Equivalent to
@yearly (or @annually)	Run once a year at midnight of 1 January	0 0 1 1 *
@monthly	Run once a month at midnight of the first day of the month	0 0 1 * *
@weekly	Run once a week at midnight on Sunday morning	0 0 * * 0
@daily (or @midnight)	Run once a day at midnight	0 0 * * *
@hourly	Run once an hour at the beginning of the hour	0 * * * *

Rolling Updates

- Support
  - – Promote an application from one environment to another (via container image updates)
  - – Rollback to previous versions
  - – Continuous Integration and Continuous Delivery of applications with zero downtime

A Fast AND Steady Approach

Helm Charts

www.fandsindia.com

What is Helm?

- Helm is a package manager for Kubernetes. (like yum or apt)
- Helm deploys charts, which you can think of as a packaged application. It is a collection of all your versioned, pre-configured application resources which can be deployed as one unit. You can then deploy another version of the chart with a different set of configuration.

What are Helm charts?

- Helm Charts are simply Kubernetes YAML manifests combined into a single package that can be advertised to your Kubernetes clusters. Once packaged, installing a Helm Chart into your cluster is as easy as running a single helm install, which really simplifies the deployment of containerized applications.

Helm Vocabulary (For V2)

- Helm has two parts to it:
  - The client (CLI), which lives on your local workstation.
  - The server (Tiller), which lives on the Kubernetes cluster to execute what’s needed.
- The idea is that you use the CLI to push the resources you need and tiller will make sure that state is in fact the case by creating/updating/deleting resources from the chart.

Helm Components

- Main Components
  - Chart: A package of pre-configured Kubernetes resources.
  - Release: A specific instance of a chart which has been deployed to the cluster using Helm.
  - Repository: A group of published charts which can be made available to others.

fands™
A Fast AND Steady Approach

Kubernetes Cluster Administration and Monitoring

Monitoring the cluster

- Tools which we have seen
  - – Kubelet
  - – API Server
  - – Kubernetes Dashboard
  - – KubeWatch
  - – REST API

Types of health checks

- Two types of health checks
  - Readiness
    - Designed to let Kubernetes know when your app is ready to serve traffic. Kubernetes makes sure the readiness probe passes before allowing a service to send traffic to the pod. If a readiness probe starts to fail, stops sending traffic to the pod until it passes.
  - Liveness
    - Know if your app is alive or dead. If you app is alive, then Kubernetes leaves it alone. If your app is dead, Kubernetes removes the Pod and starts a new one to replace it.

Limit Ranges

- By default, containers run with unbounded compute resources on a cluster. With resource quotas, cluster administrators can restrict resource consumption and creation on a namespace basis. Within a namespace, a Pod or Container can consume as much CPU and memory as defined by the namespace's resource quota. There is a concern that one Pod or Container could monopolize all available resources. A LimitRange is a policy to constrain resource allocations (to Pods or Containers) in a namespace.

LimitRange

- A LimitRange provides constraints that can:
  - Enforce minimum and maximum compute resources usage per Pod or Container in a namespace.
  - Enforce minimum and maximum storage request per PersistentVolumeClaim in a namespace.
  - Enforce a ratio between request and limit for a resource in a namespace.
  - Set default request/limit for compute resources in a namespace and automatically inject them to Containers at runtime.
  - Enabling LimitRange

Pod and LimitRange

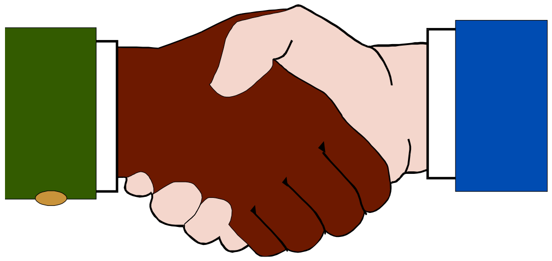
```
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-min-max-demo-lr
spec:
  limits:
  - max:
      cpu: "800m"
    min:
      cpu: "200m"
    type: Container
```

fands™

QUESTION / ANSWERS

www.fandsindia.com

THANKING YOU !



www.fandsindia.com