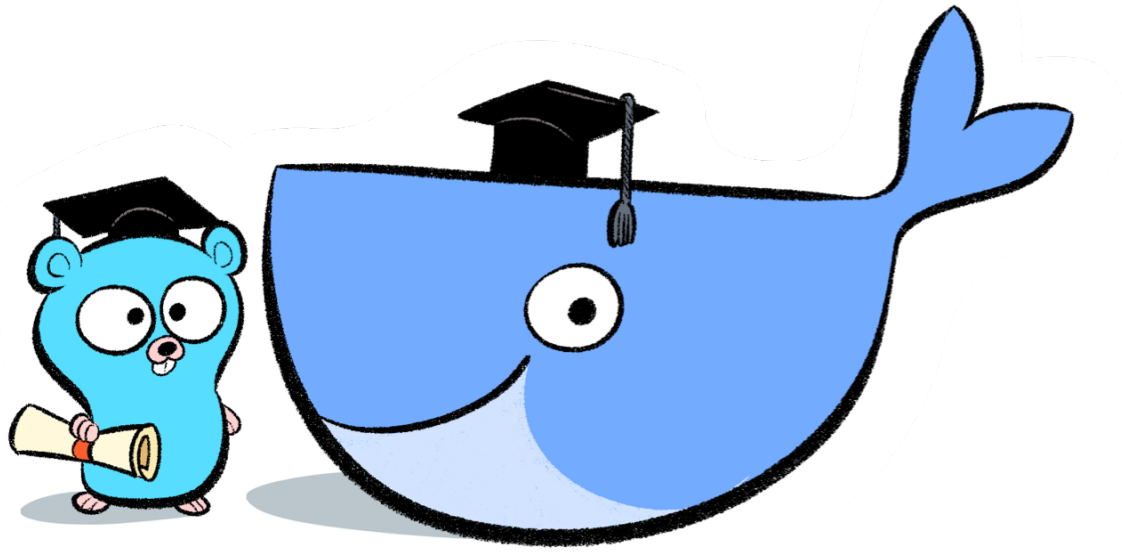


NetApp K8s Workshop

The Challenge

Multiplicity of Stacks

Do services and apps interact appropriately?

User DB

postgresql + pgv8 + v8

Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2

Queue

Redis + redis-sentinel

Analytics DB

hadoop + hive + thrift + OpenJDK

Background workers

Python 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

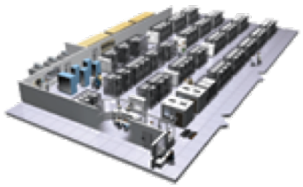
Multiplicity of hardware environments

Can I migrate smoothly and quickly?

Development VM

Public Cloud

Production Cluster



QA server

Disaster recovery

Customer Data Center

Contributor's laptop

Production Servers

Cargo Transport pre 1960

Multiplicity of Goods





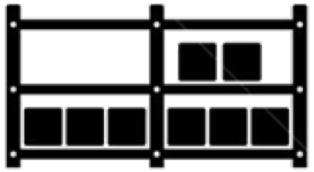


Do I worry about how goods interact (e.g. coffee beans next to spices)

Multipilicity of methods for transporting/storing






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

Intermodal Shipping Container

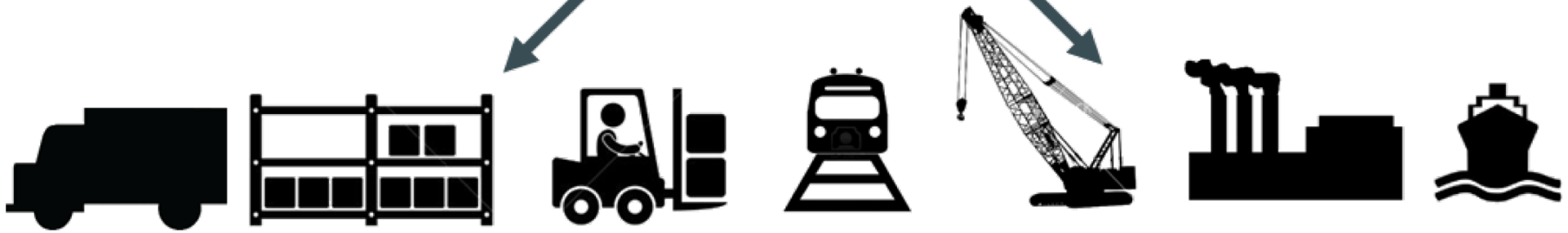
Multiplicity of Goods

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)

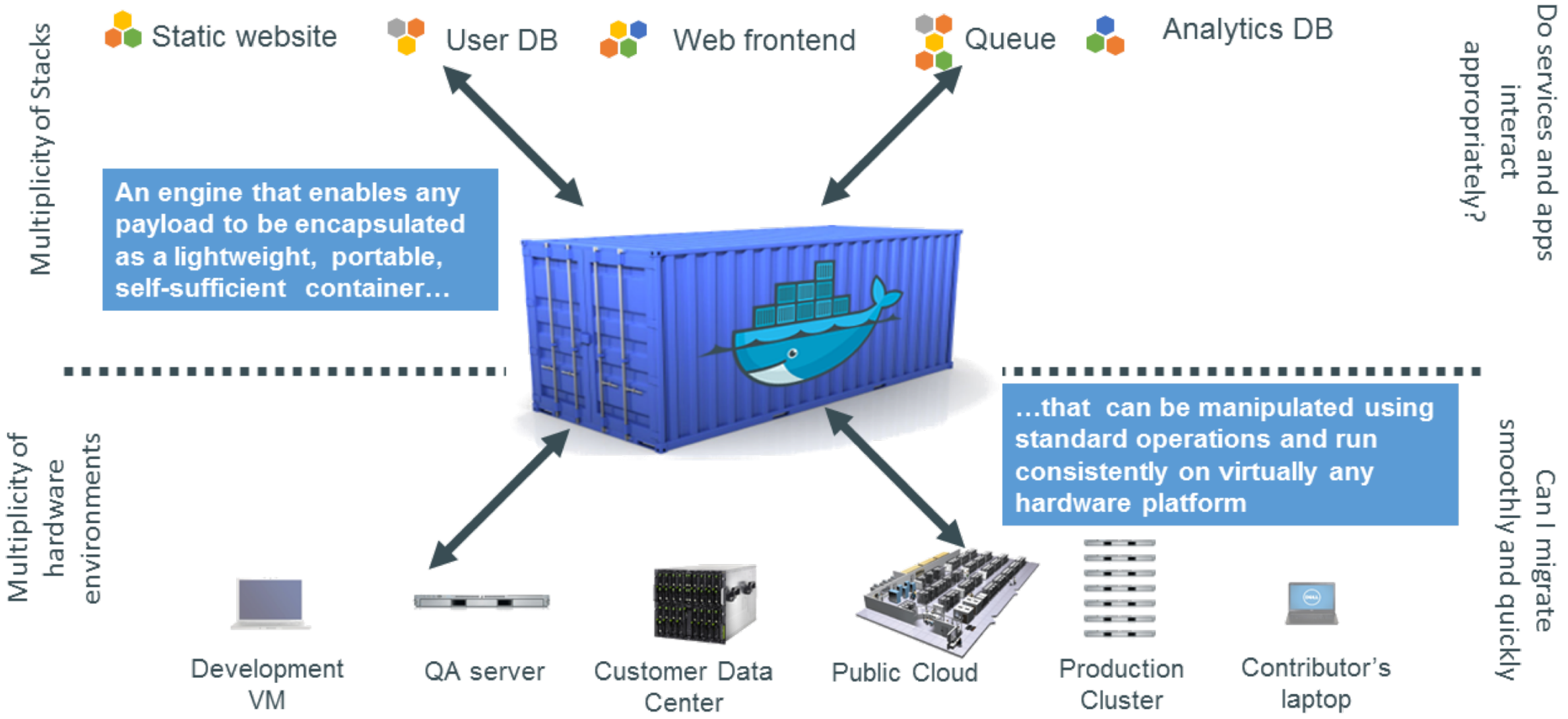
…in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

Multiplicity of methods for transporting/storing



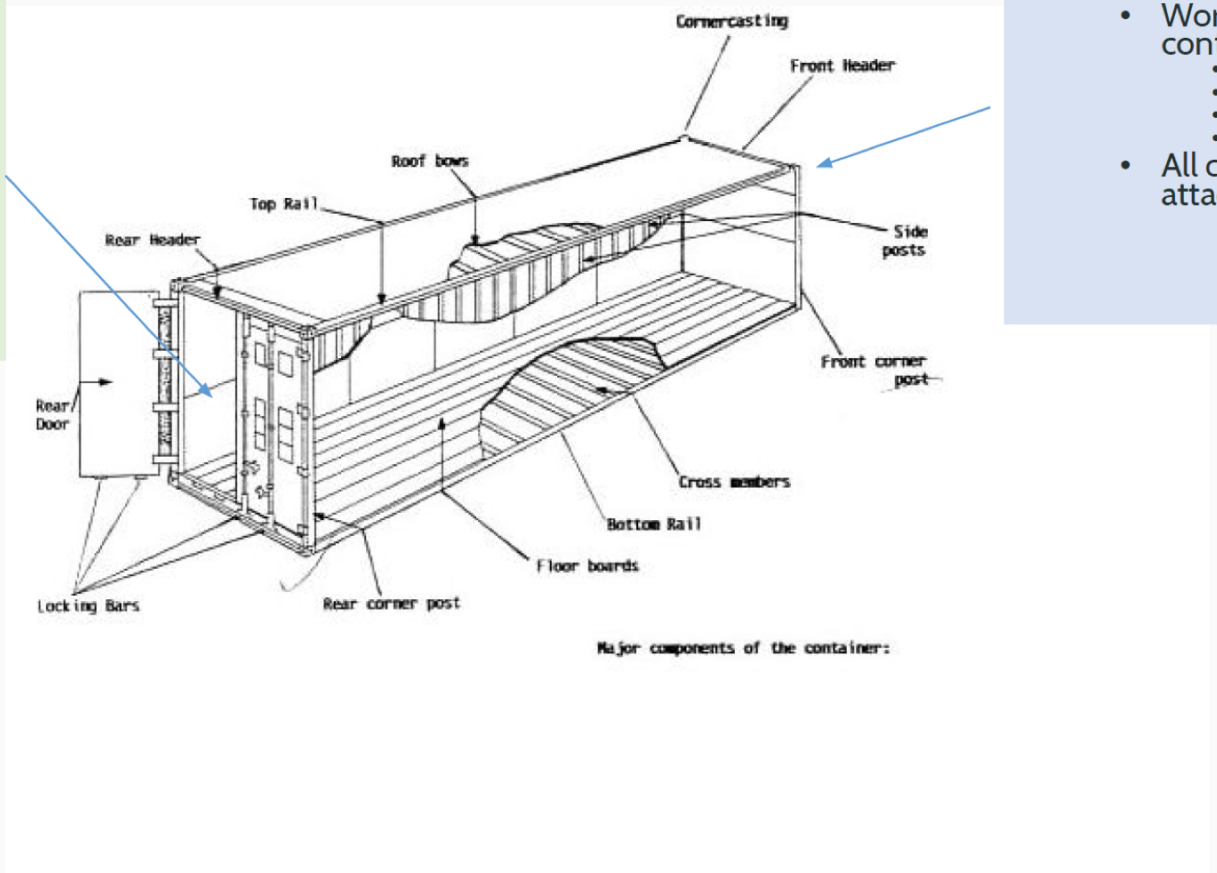
Can I transport quickly and smoothly (e.g. from boat to train to truck)

Shipping Container for Code



Why it works?

- Dan the Developer
  - Worries about what's "inside" the container
    - His code
    - His Libraries
    - His Package Manager
    - His Apps
    - His Data
  - All Linux servers look the same



- Oscar the Ops Guy
  - Worries about what's "outside" the container
    - Logging
    - Remote access
    - Monitoring
    - Network config
  - All containers start, stop, copy, attach, migrate, etc. the same way

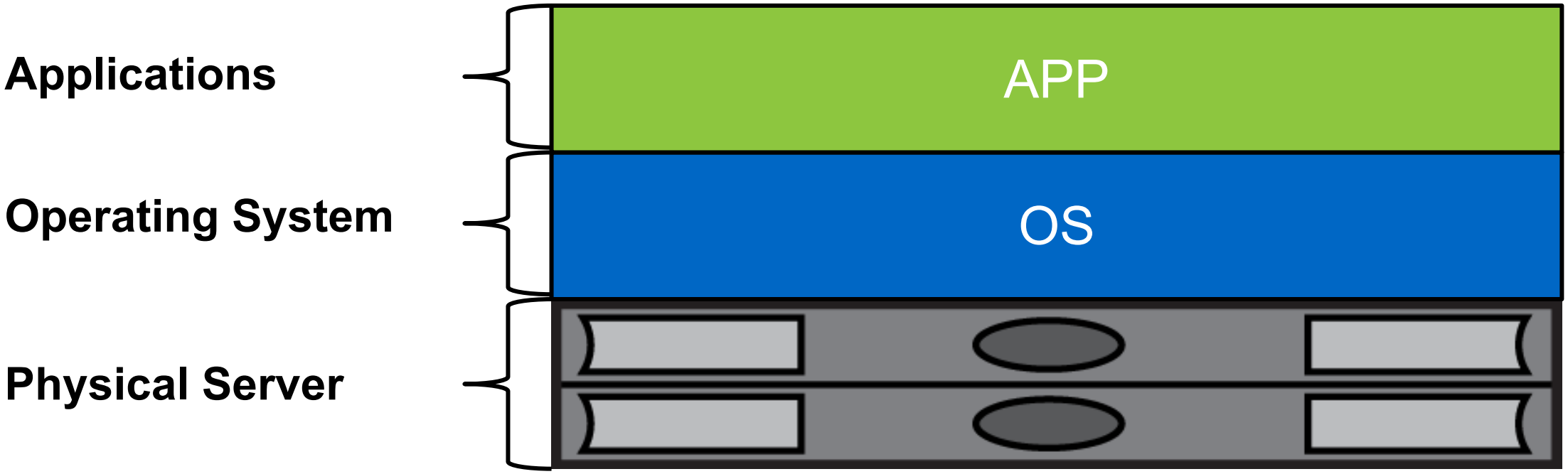
What Docker isn’t?

- Docker is not a hypervisor
- Docker does not emulate anything
- Docker is not new
- A container does not run on Docker
- Containers do not require Docker
- Docker doesn’t really do anything once a container is running

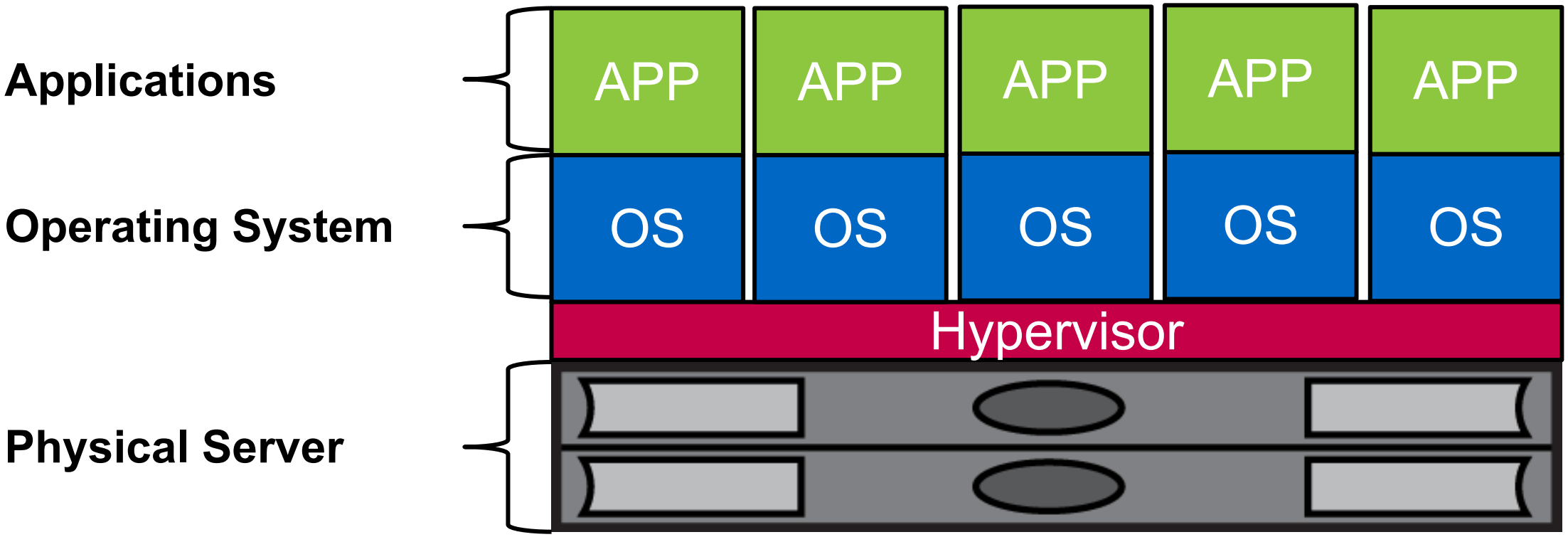
What Docker is?

- An Open-Source framework (written in Go) to manage Namespacing of services
- The boundaries of an active namespace are enforced by the OS
- Docker just helps you launch and manage application in namespaces
- Anyone can write an application that runs as a namespace
- Expect other applications to use namespaces outside the use of Docker

Applications on a Physical server



Applications on a Virtual server



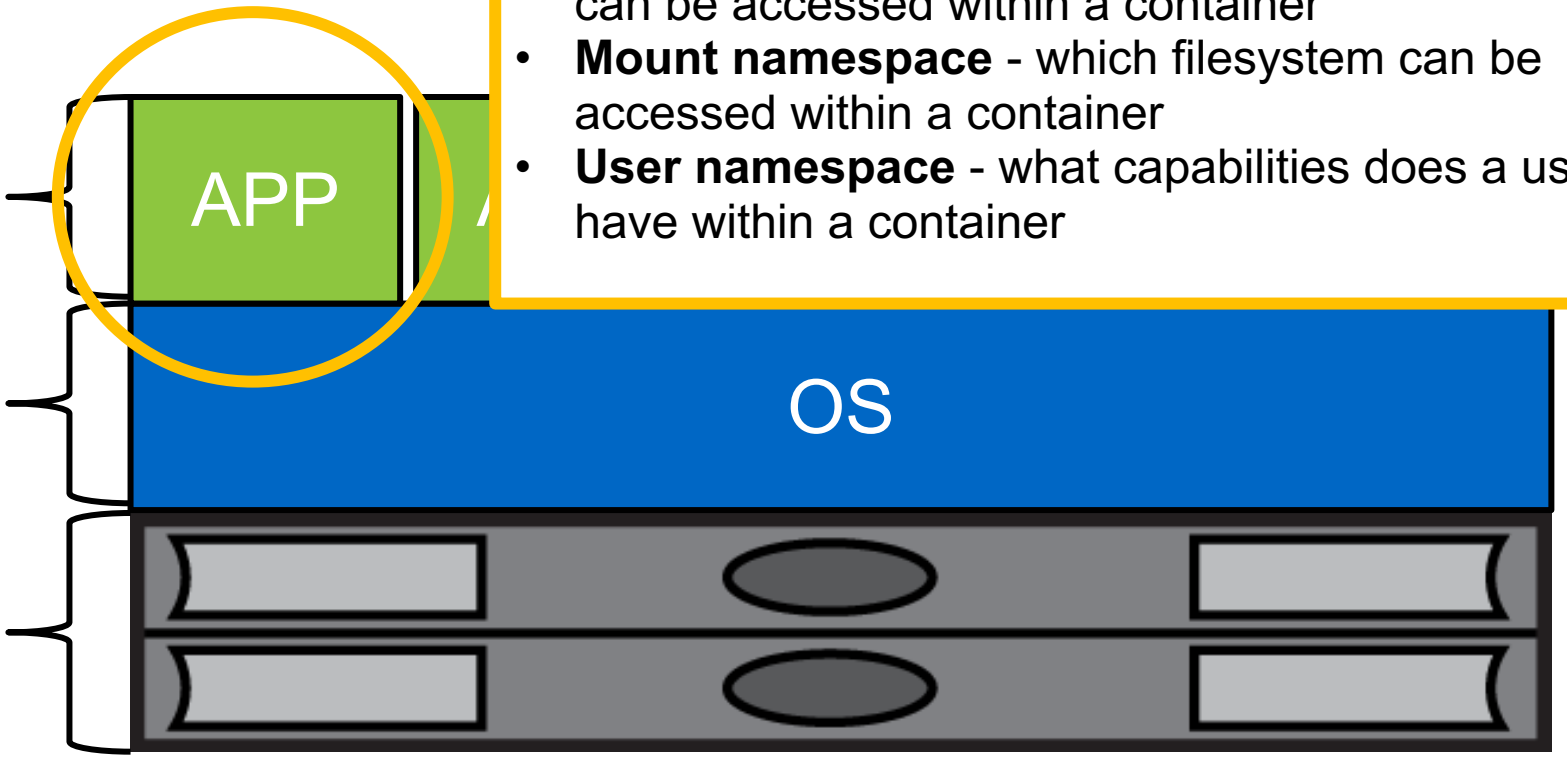
Applications on Namespaces

- **Process namespace** - which application processes are allowed so see each other within a container
- **Network namespace** - what network interfaces can be accessed within a container
- **Mount namespace** - which filesystem can be accessed within a container
- **User namespace** - what capabilities does a user have within a container

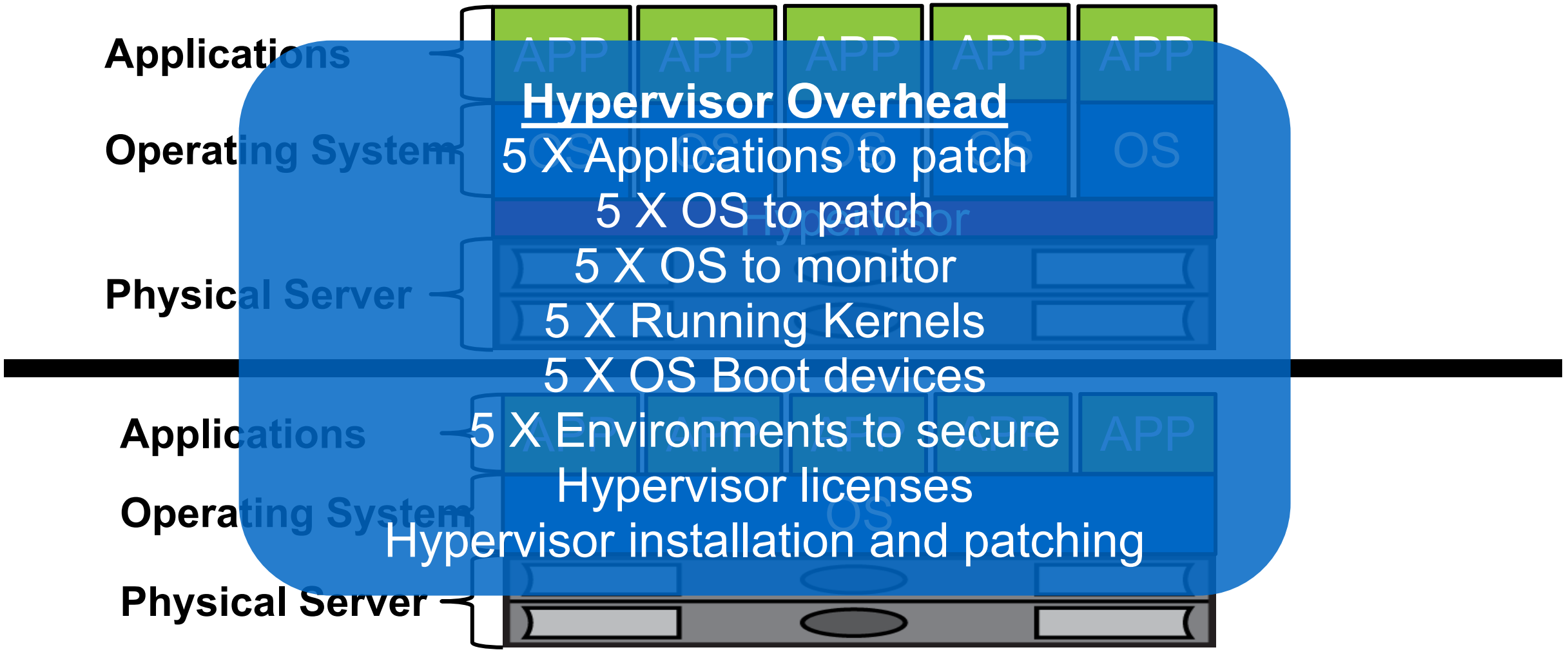
Applications

Operating System

Physical Server



Applications on Namespaces



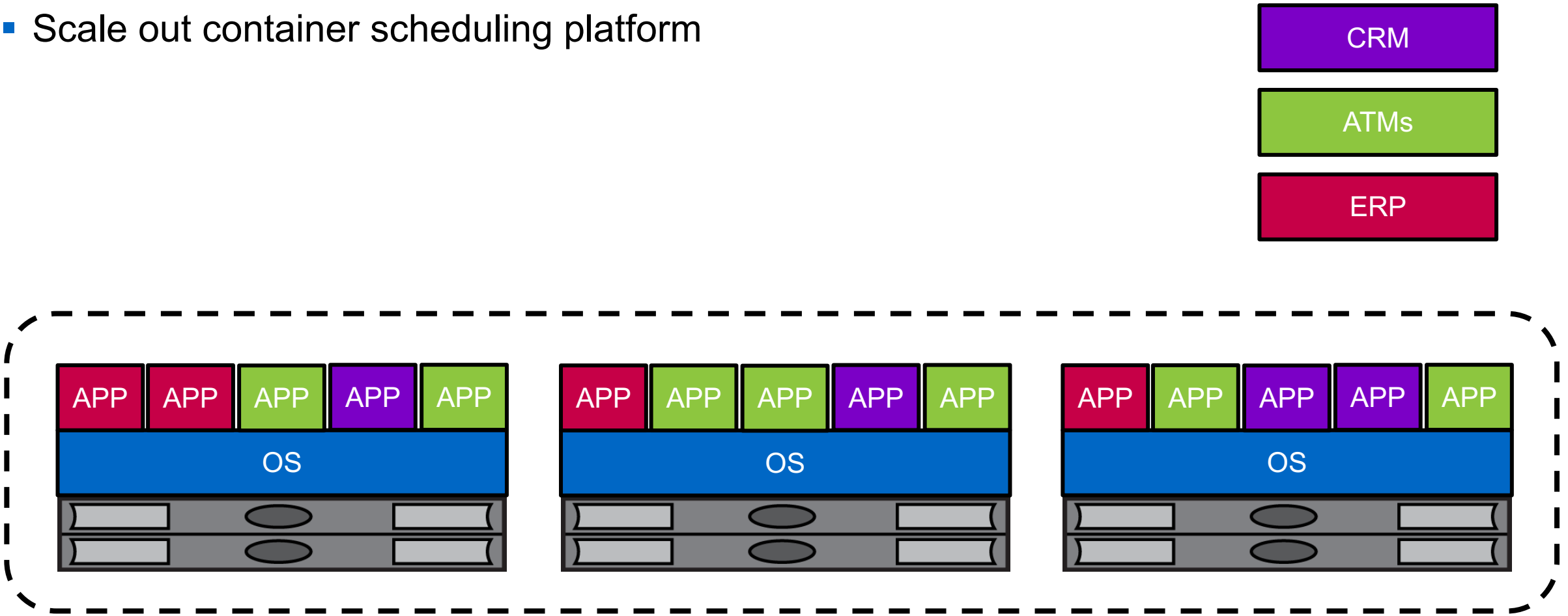
Docker Terminology

- **Docker** - aka Docker Engine - the daemon managing docker images and containers (using namespaces and cgoups). It runs on the Linux based host.
- **Docker Client** - The Binary interacting with the Docker Engine.
- **Docker Image** - A Filesystem (read only) template used to create a container.
- **Docker Container** - A running image providing a service
- **Host** - The server/computer running the Docker Engine
- **Docker Registry** - A Public (Docker Hub) or Private collection of Docker images
- **Docker CE** - Docker Community Edition
- **Docker EE** - Docker Enterprise Edition

What is Kubernetes?

What is Kubernetes?

- Scale out container scheduling platform

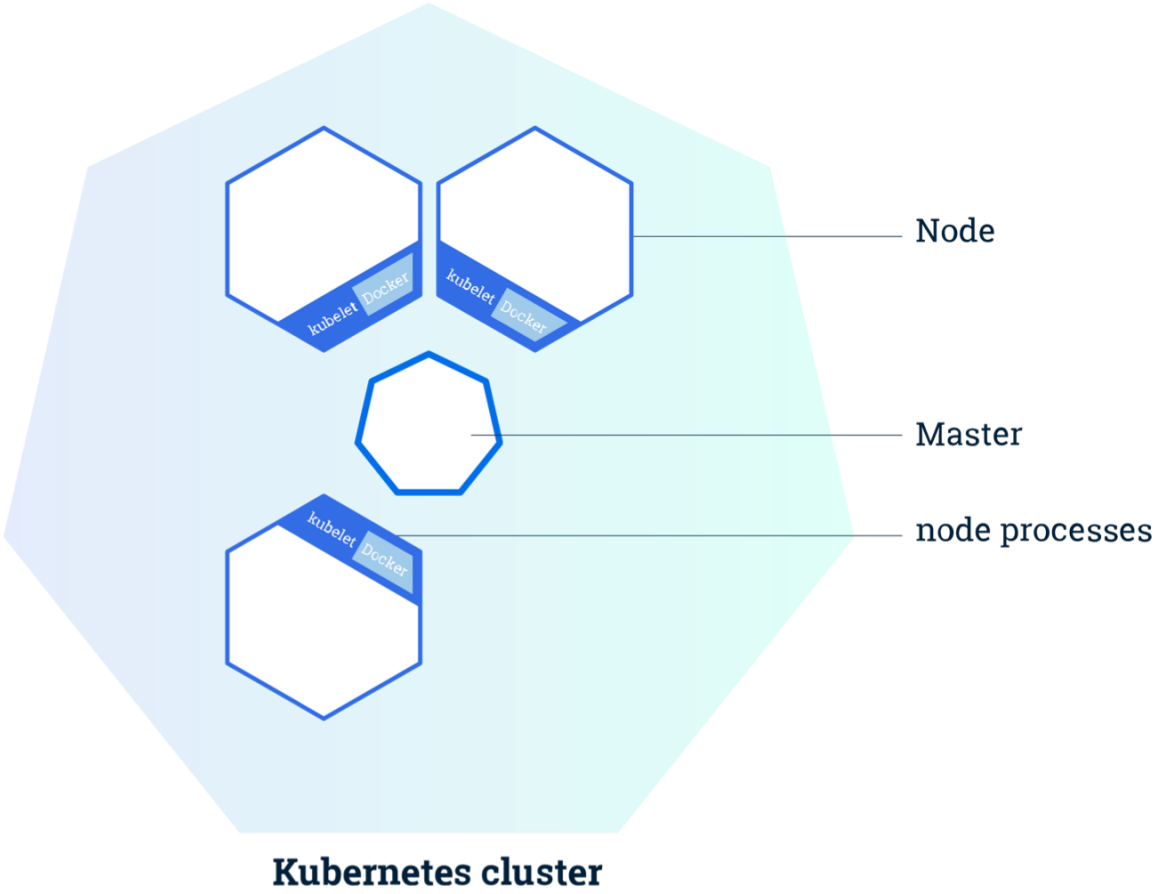


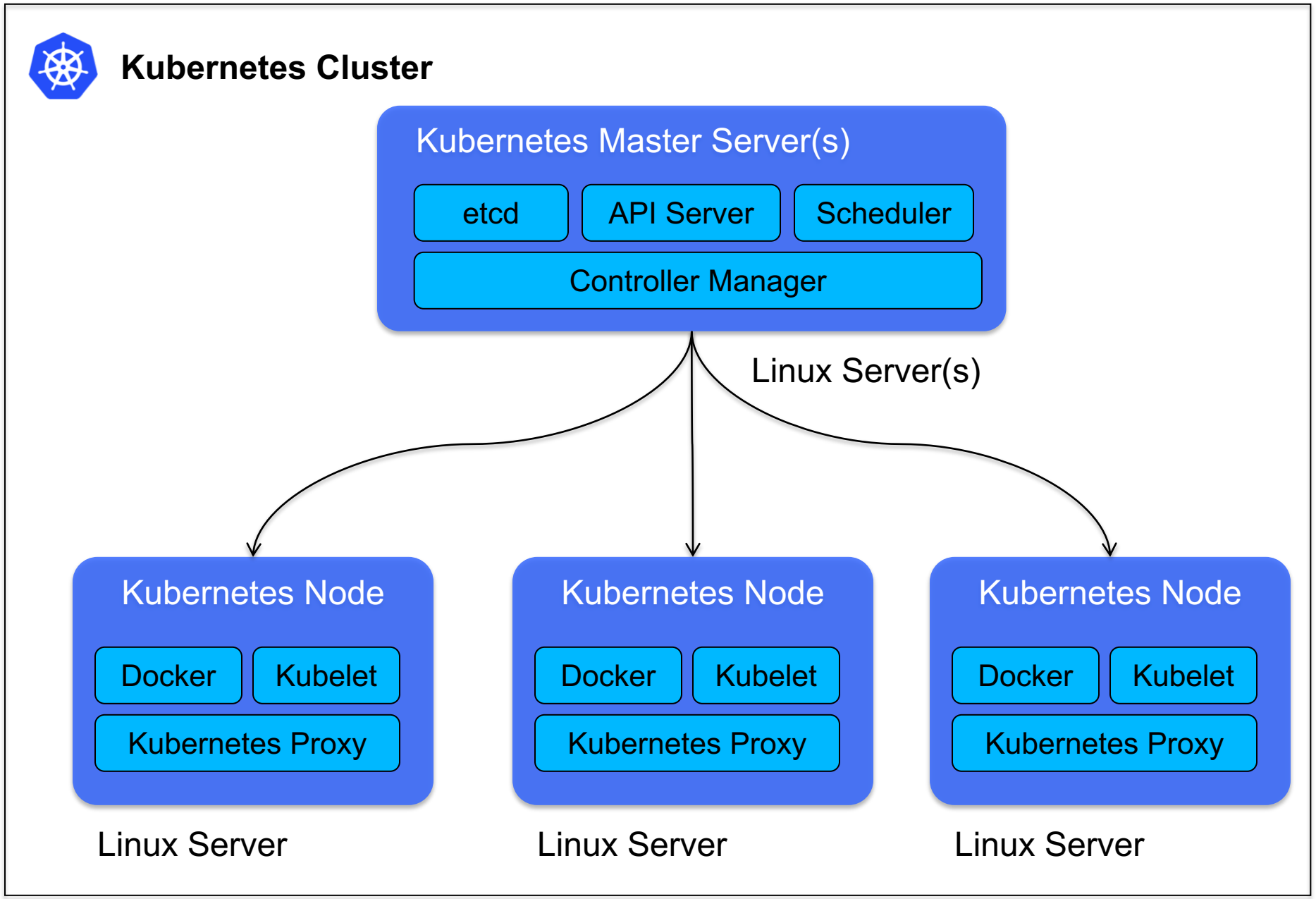
K8s Cluster

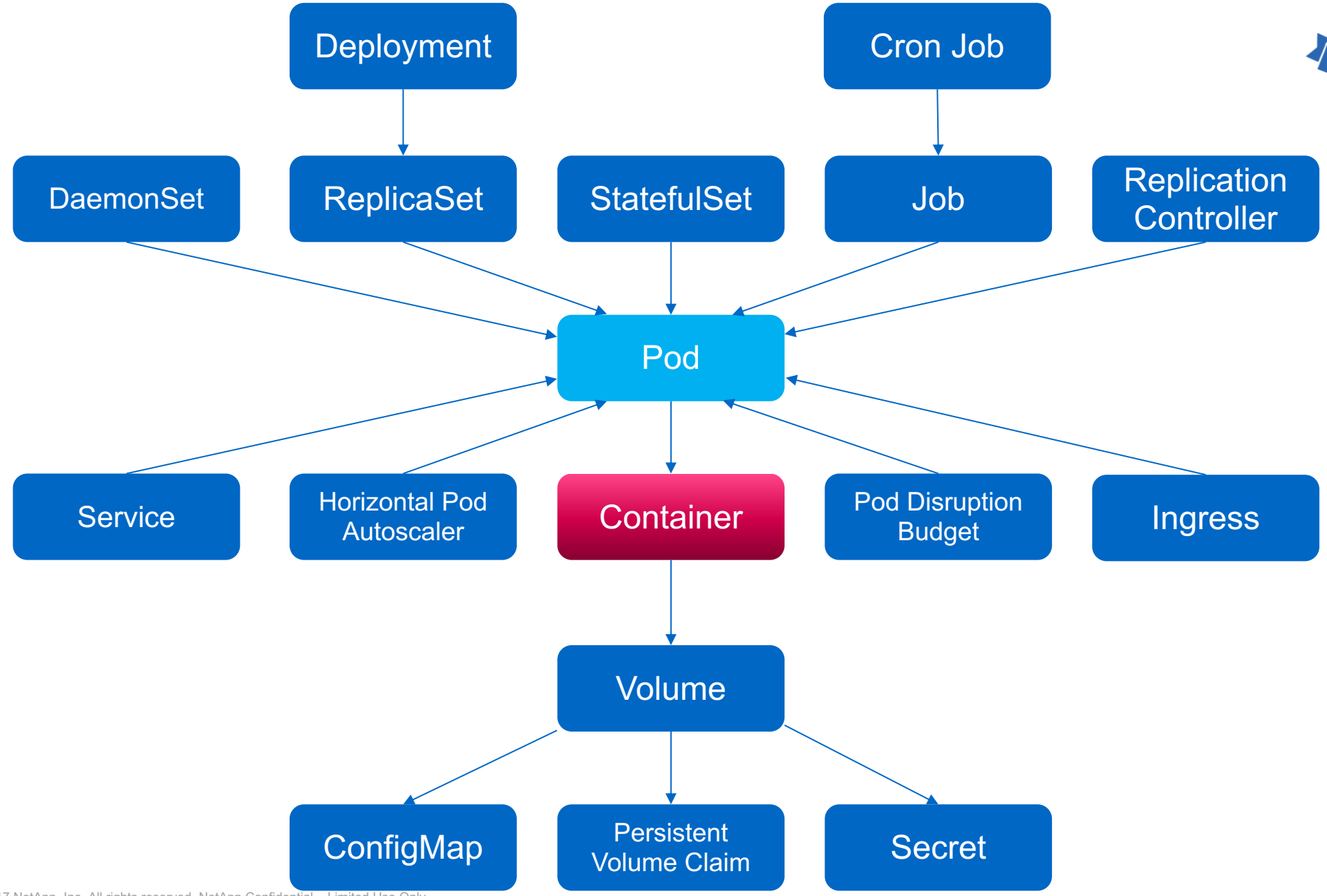
K8s has six main components that need to be running to get a functioning cluster:

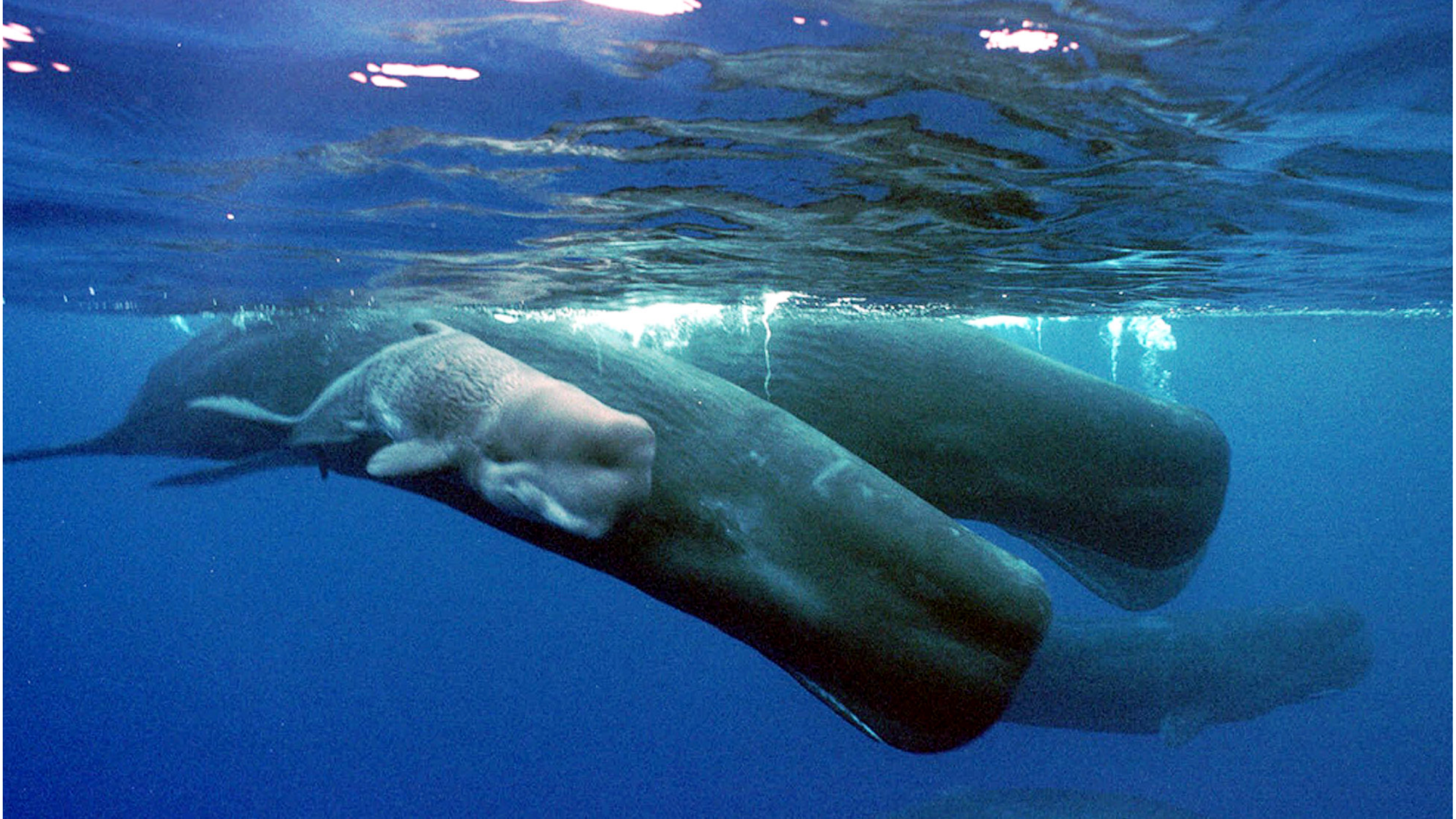
- The API Server
- The Scheduler
- The Controller Manager
- The kubelet
- The proxy (aka kube proxy)
- An etcd cluster

Each of these component can run as a standard Linux process or they can run as Docker containers



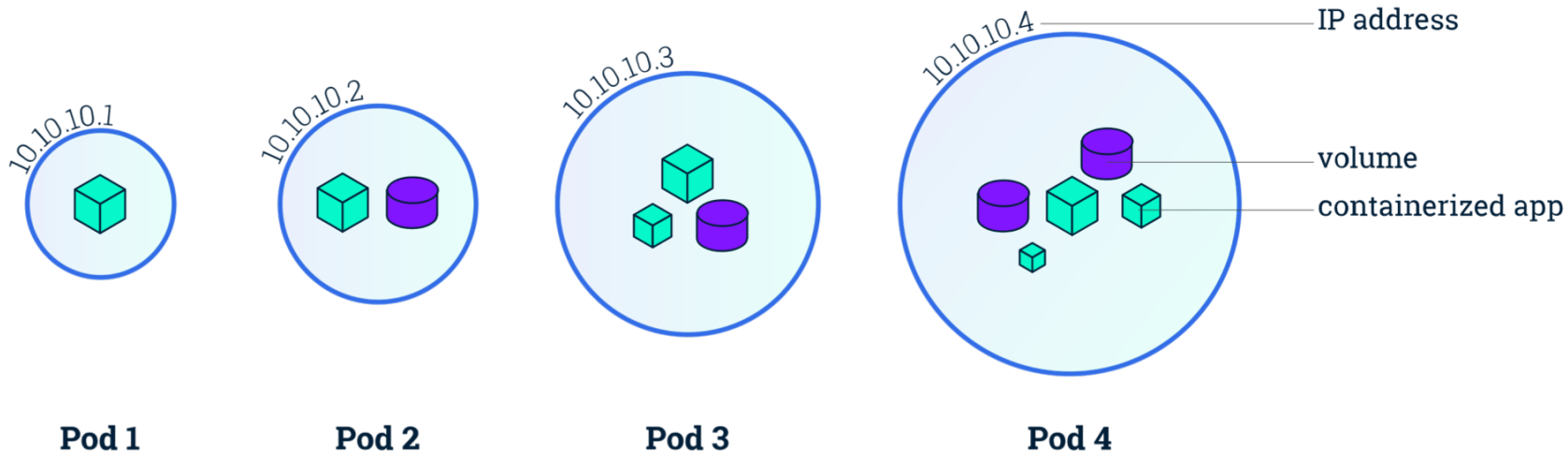






Pods

- A group of one or more application container
- Pods share resources including:
  - Storage as volumes
  - Networking as unique Cluster IP address
  - Information about how to run each container such as images version and ports



Pods - Liveness Probe

- Confirm the Pod in healthy and shouldn't be restarted
- Are defined per container
- Parameters:
  - *httpGet* - GET request against the address (supports tcpSocket or exec as well)
  - *initialDelaySeconds* - Delay after the container is created
  - *failureThreshold* - Number for health check failed before failing
  - *timeoutSeconds* - The Pod must respond to the health check within this time
  - *peroidSeconds* - K8s will call the probe every number of seconds

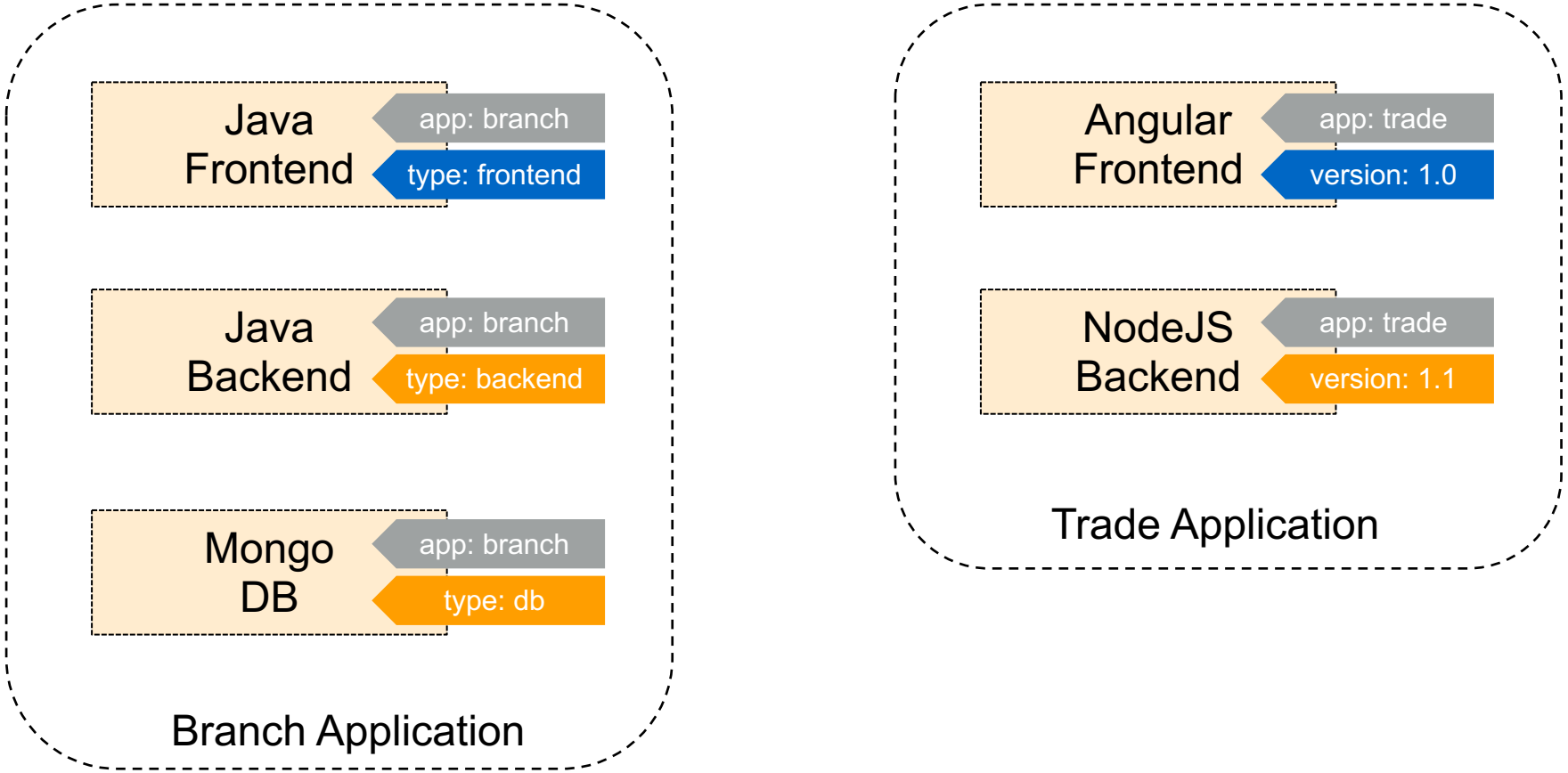
Pods - Readiness Probe

- Confirm the Pod in ready to accept traffic from services
- Are defined per container
- Container will not be restarted, but marked unready for service
- Parameters:
  - *httpGet* - GET request against the address (supports tcpSocket or exec as well)
  - *initialDelaySeconds* - Delay after the container is created
  - *failureThreshold* - Number for health check failed before failing
  - *successThreshold* - Minimum consecutive successes for the probe to be considered successful after having failed
  - *timeoutSeconds* - The Pod must respond to the health check within this time
  - *peroidSeconds* - K8s will call the probe every number of seconds

Pods - Resource Management

- Request the resources required to run a container (not a Pod)
- Most commonly requested resources are CPU and Memory (others are supported)
- *requests*:
  - *cpu* - amount of CPU cores requested (either 1 or 1000m for 1 CPU Core)
  - *memory* - amount of RAM memory requested (In MB, GB)
- Limits:
  - *cpu* - the container can’t get more CPU than this limit (either 1 or 1000m for 1 CPU Core)
  - *memory* - the container can’t get more RAM than this limit (In MB, GB)

Labels



Labels

- Labels are used by the replication controller to keep a number of instances of a specific pod running. That means every pod definition needs to have unique combination of labels used for scheduling.
- Labels are also used heavily by the scheduler. In order to place pods on the hosts that satisfy the pods' requirements, the scheduler uses labels for collocating or spreading pods.
- A label can indicate whether a pod belongs to a logic group of pods (such as application) and only together this group of pods can provide a business value.
- A label can give application identity to a group of pods and containers. In addition to these common use cases, labels can be used to track any kind of meta data describing a pod. It might be difficult to guess what a label might be useful for in the future, so it is better to have enough labels that describe any important aspect of the pod to begin with. For example, having labels to indicate the logical group of an application, the business characteristics

Service

- A *Service* is a way to create a named label selector
- The *Service* will expose all the pods with a specific label selector and a specific port in one of the following ways:
  - *ClusterIP* - the service will be assigned a new virtual IP address called Cluster IP. This is special IP address the system will load-balance across all the pods that are identified by the selector.
  - *NodePort* - in addition to a Cluster IP, the system picks a port (or the user can specify one), and every node in the cluster then forwards traffic from that port to the service
  - *LoadBalancer* - The builds on NodePorts by additionally configuring the cloud to create a new load balancer and direct it at nodes in your cluster

ReplicaSets

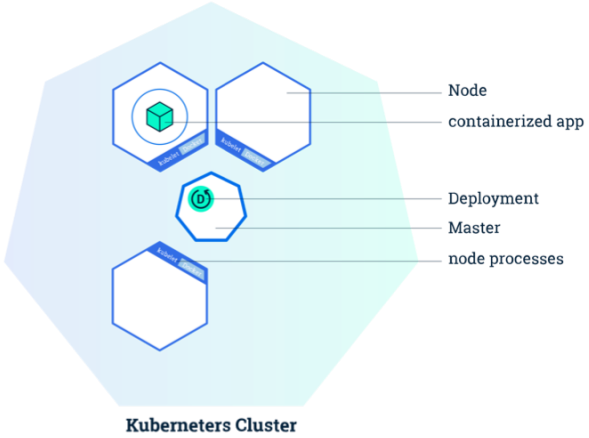
- Maintain multiple replicas of a pod for:
  - Redundancy - failure toleration
  - Scale - Handle more user requests
  - Sharding - Parallel computation
- Runs a Reconciliation Loop that makes sure that the *observed/current* state is always set to the *desired* state
- For example:
  - The *desired* state is that are 3 replicas of a Pod
  - The *observed* state is that are 2 replicas of a Pod
- ReplicaSets are decoupled from Pods. The use Label Selectors to identify the set of Pods they should be managing

DaemonSet

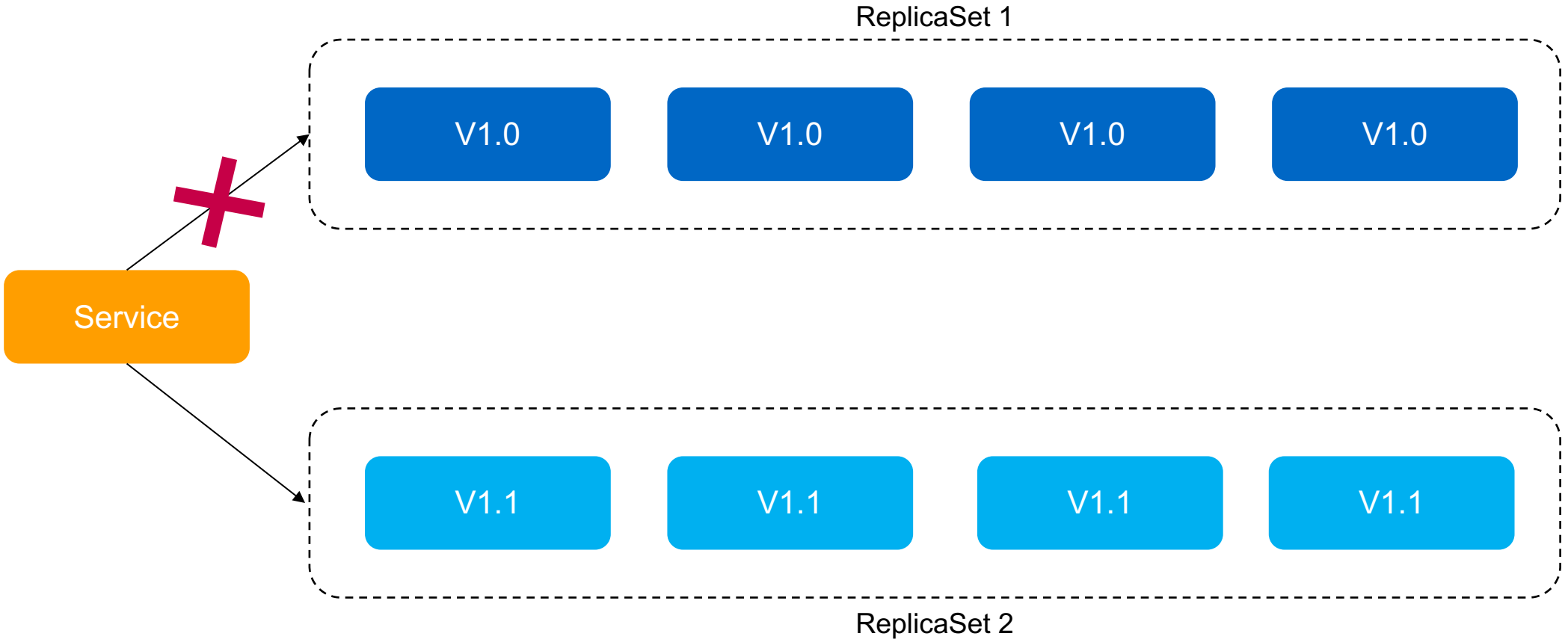
- A DaemonSet ensures a copy of the Pod is running across a set of nodes in a K8s cluster.
- DaemonSets are used to deploy system daemons such as log collectors and monitoring agents, which typically must run on every node
- DaemonSets share typical functionality with ReplicaSets
- By default a DaemonSet will create a copy of a Pod on every node unless a node selector is used
- As with rolling updates of deployments, the rolling update strategy gradually updates members of a DaemonSet until all of the Pods are running the new configuration. There are two parameters that control the rolling update of a DaemonSet:
  - *minReadySeconds* - which determines how long a Pod must be “ready” before the rolling update proceeds to upgrade subsequent Pods
  - *maxUnavailable* - which indicates how many Pods may be simultaneously updated by the rolling update

Deployments

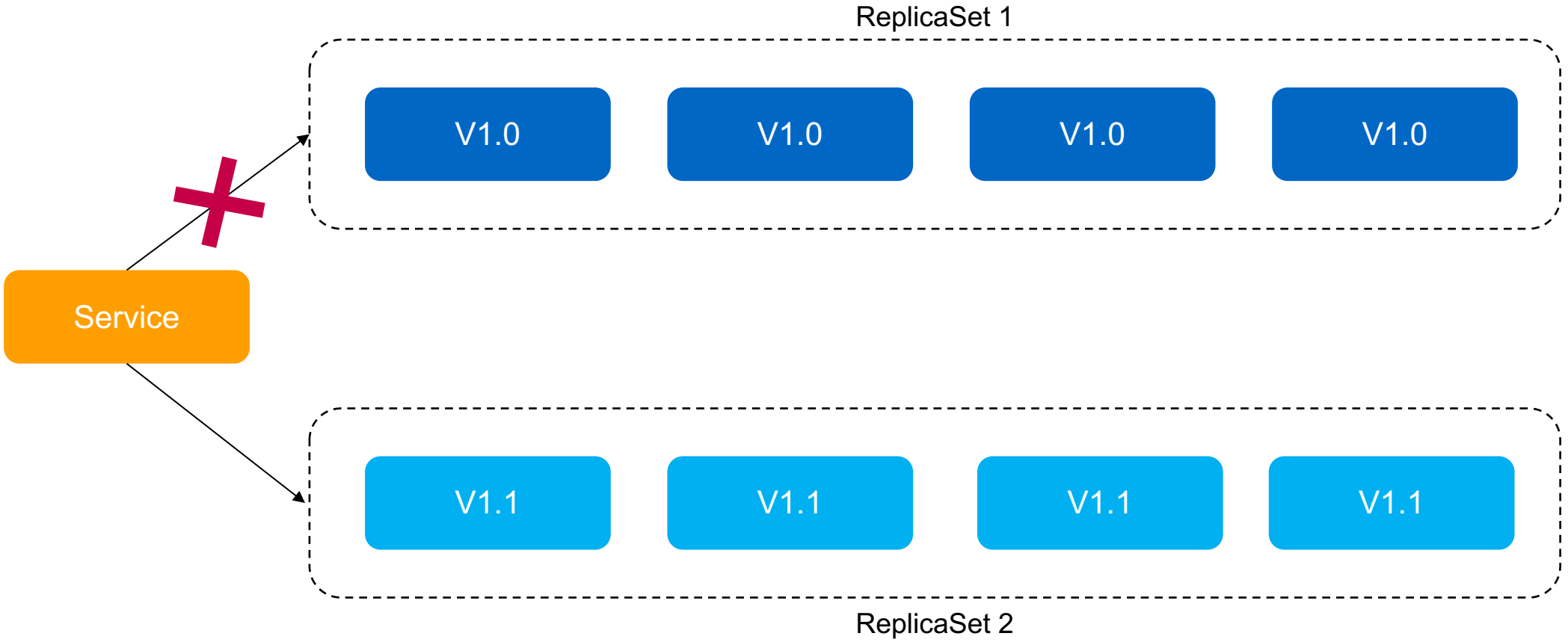
- Help manage your daily or weekly cadence of releasing new versions of your application
- A *Deployment* represents a deployed application
- Enable to easily move from one version of your code to the next
- *Deployments* use Label Selectors to manage *ReplicaSets*
- In *Deployments* you define your rollout *strategy*:
  - *type - RollingUpdate* or *Recreate*
  - *maxUnavilable* - the maximum number of Pods that can be unavailable (can be number or percentage)
  - *maxSurge* - how many extra resources cab be created to achieve a rollout (can be number or percentage)
  - *minReadySeconds - Deployment* must wait after seeing a Pod become healthy before moving on to updating the next pod.
  - *progressDeadlineSeconds* - If any stage in the rollout fails to progress then the *Deployment* is marked as failed.



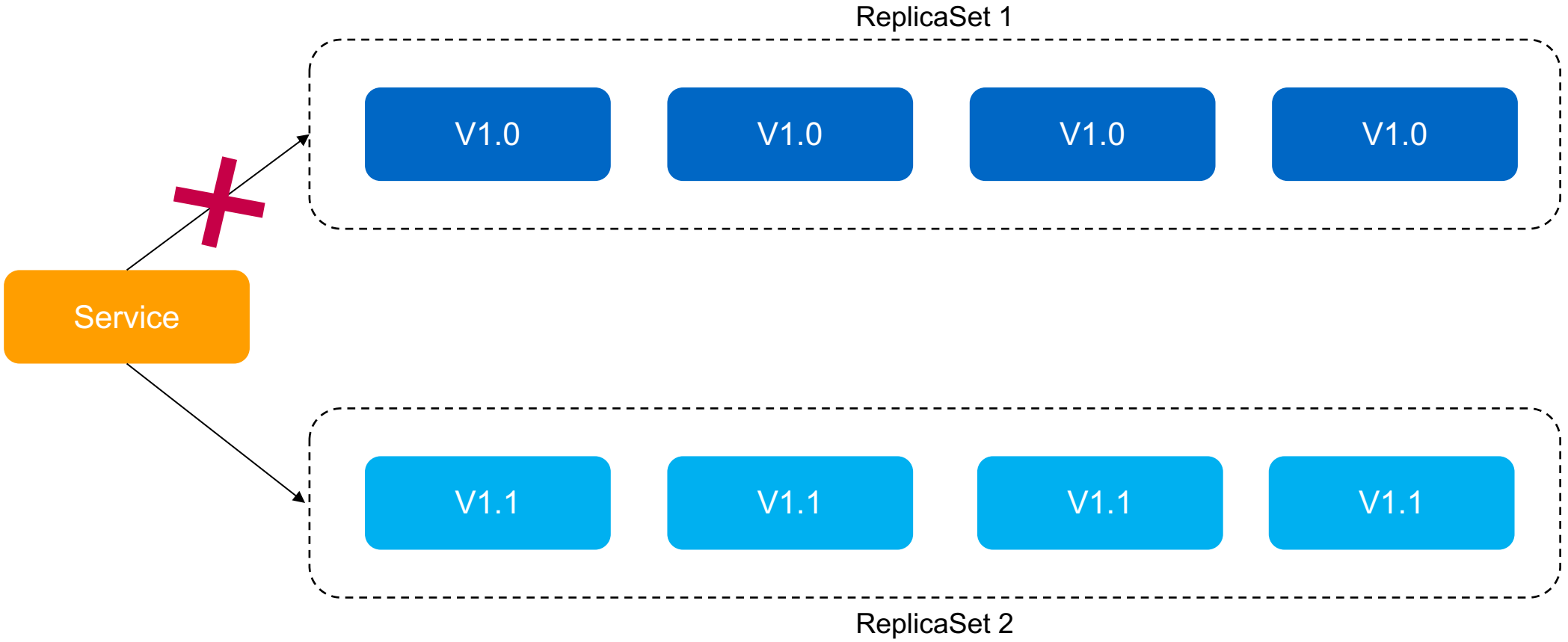
Rolling Deployments



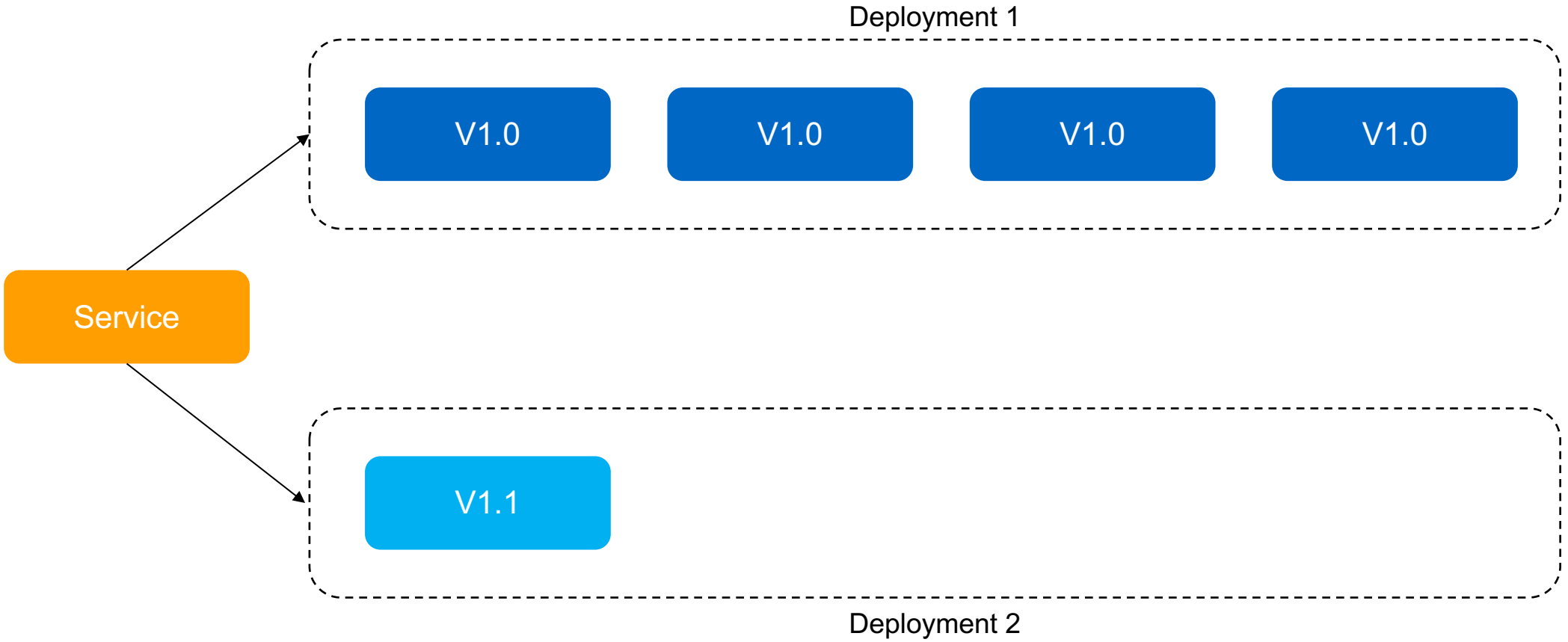
Fixed Deployments



Blue/Green Deployments



Canary Deployments



Persistent Volumes

- Storage which has been introduced to Kubernetes by an administrator
- Configured for backing storage device
  - NFS, iSCSI, Cinder, AWS EBS, GCE, Azure
- Abstracts the physical storage volume into an allocatable unit for applications
- Includes connection information for the storage volume

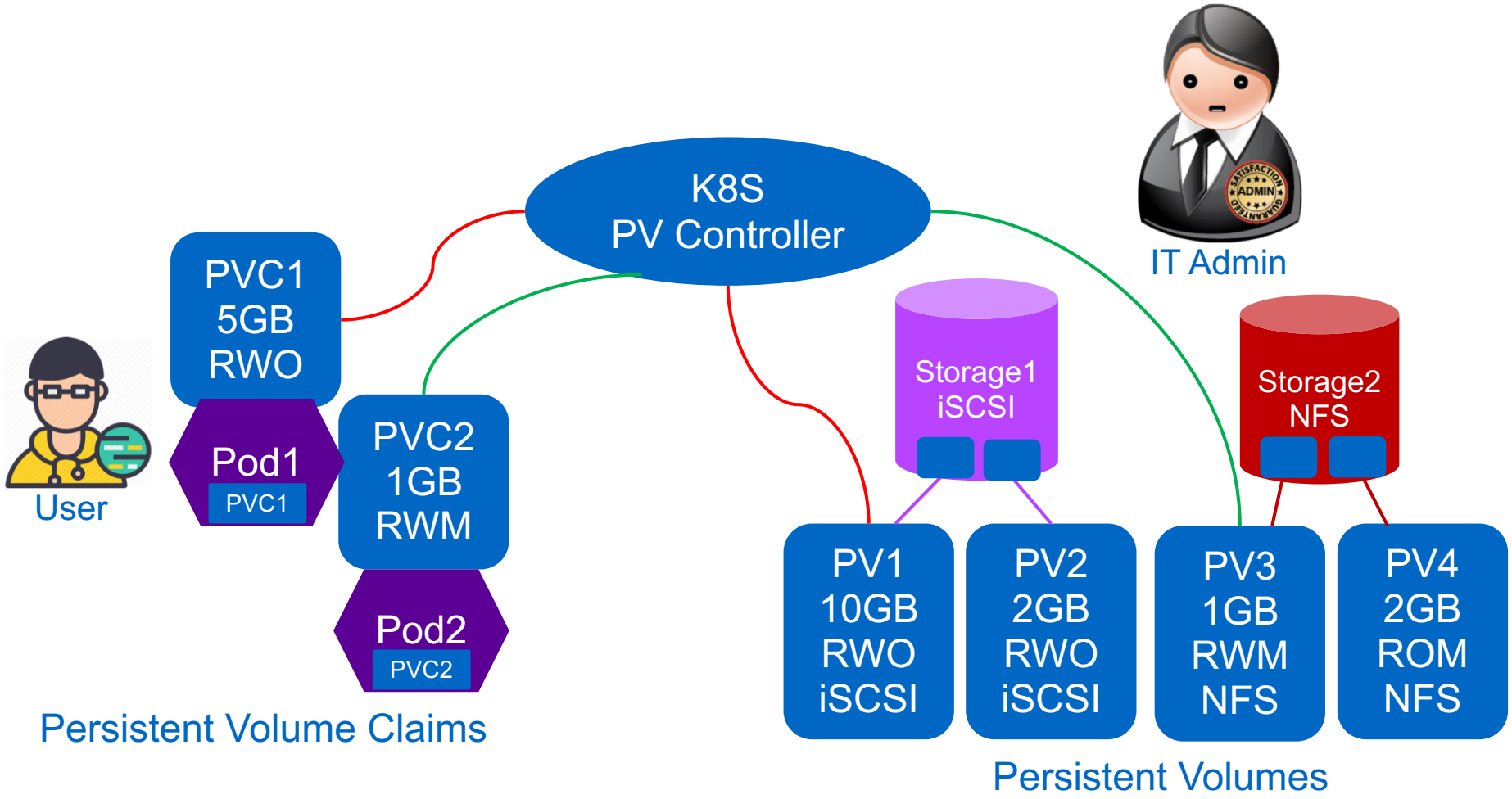
```
apiVersion: v1
  kind: PersistentVolume
  metadata:
    name: pv0003
  spec:
    capacity:
      storage: 5Gi
    accessModes:
      - ReadWriteMany
  storageClassName: bronze
    nfs:
      path: /tmp
      server: 172.17.0.2
```

Persistent Volume Claims

- Created by a user to request storage
- Specifies desired capacity and access mode, along with labels to aid with selection
- Kubernetes assigns a PV to meet the requirements requested in the PVC
  - Does not require an exact match for capacity

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: thepub
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: bronze
```

Persistent Volume Framework



Storage Classes

- Describes a storage offering and associates a provisioner
- Parameters are used to provide additional information to the provisioner
- Parameters are opaque to Kubernetes
- Storage classes can be used by statically provisioned PVs
- Used by PVCs to inform Kubernetes that the PV should belong

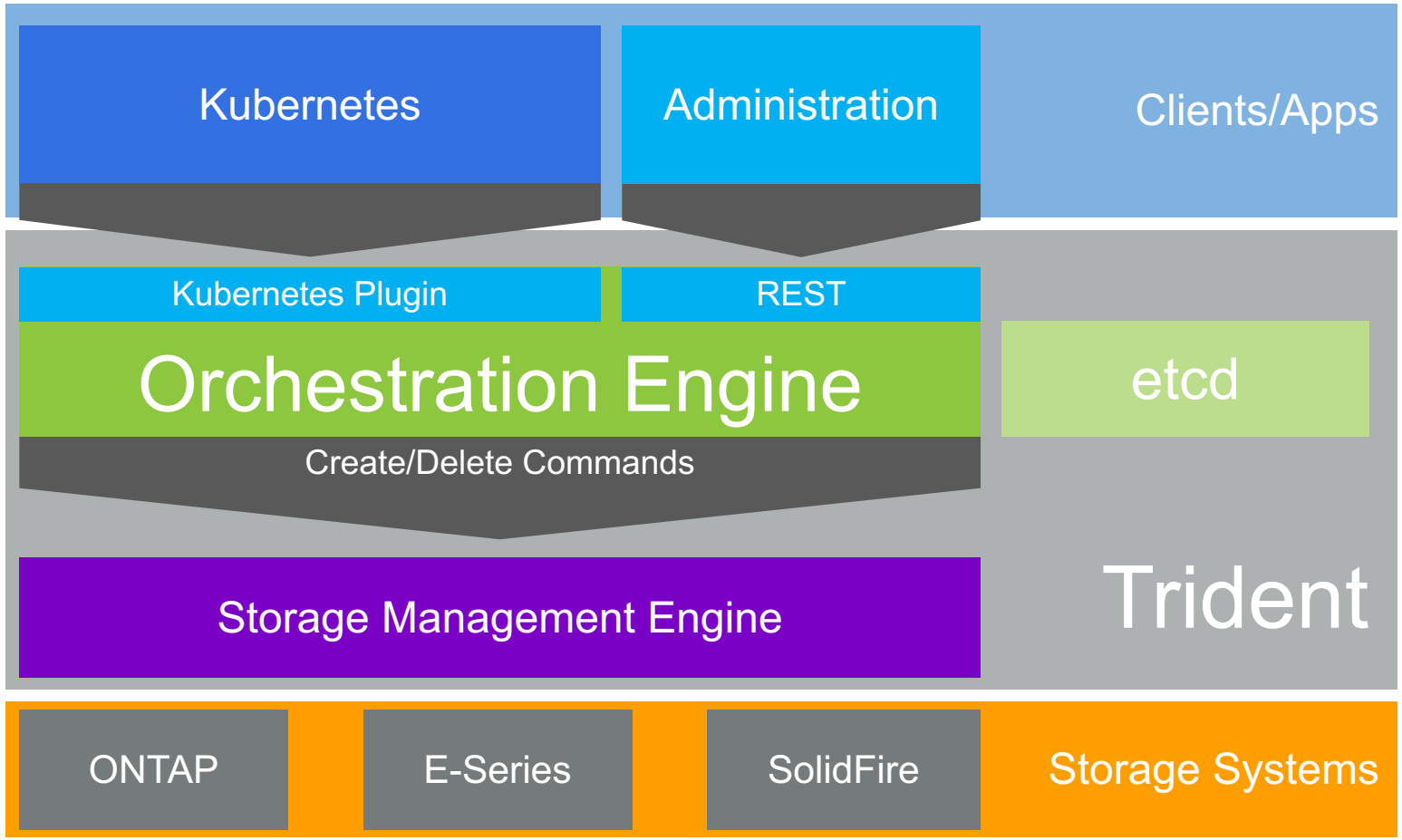
```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: bronze
provisioner: netapp.io/trident
parameters:
  backendType: "ontap-nas"
  mediaType: "hdd"
```

Trident

- A dynamic storage provisioner for Kubernetes
- Supports ONTAP, SolidFire, and E-Series
- Retains the ability to differentiate storage
  - IOPS, compression, disk type, etc. all able to be specified
- Abstracts backends into pools of capabilities
- Maps storage requests to storage pools, each backend can contain one or more storage pools

kubernetes

Trident Architecture

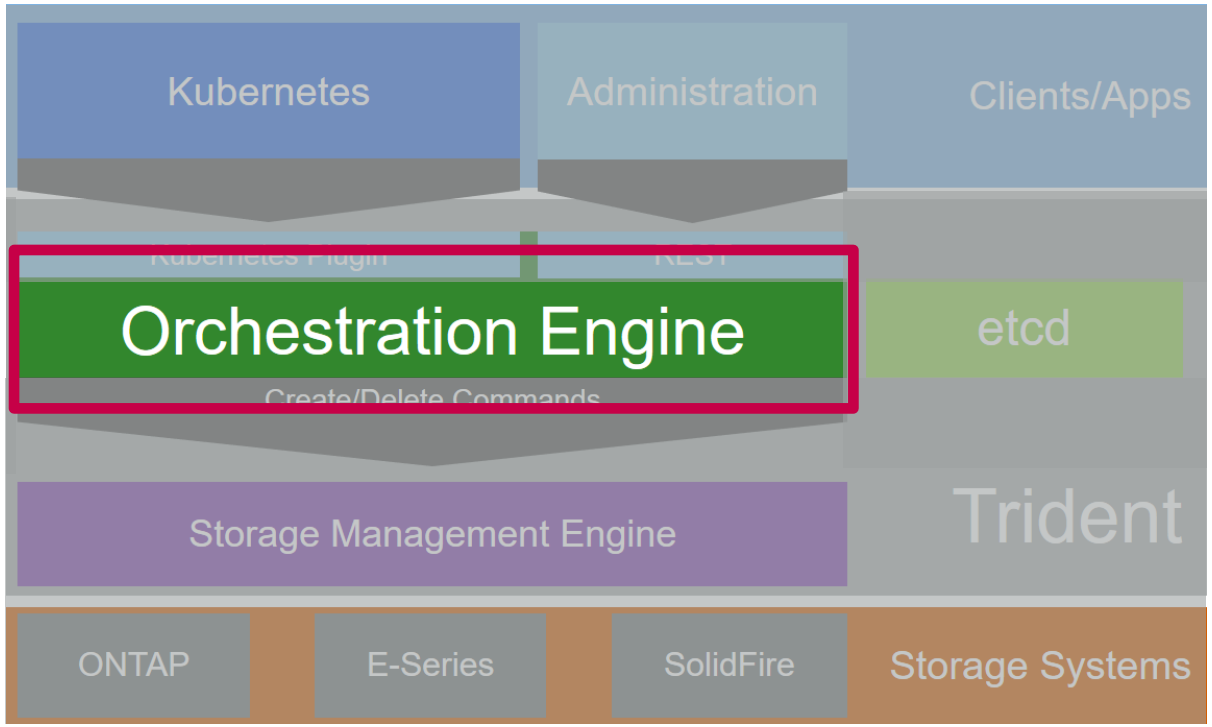


etcd – Persistent Metadata Storage

- Distributed key-value store
- Stores metadata about Trident managed volumes, backends, and storage classes
- Deployed with a persistent volume when used with Trident
- Trident can use external etcd if desired

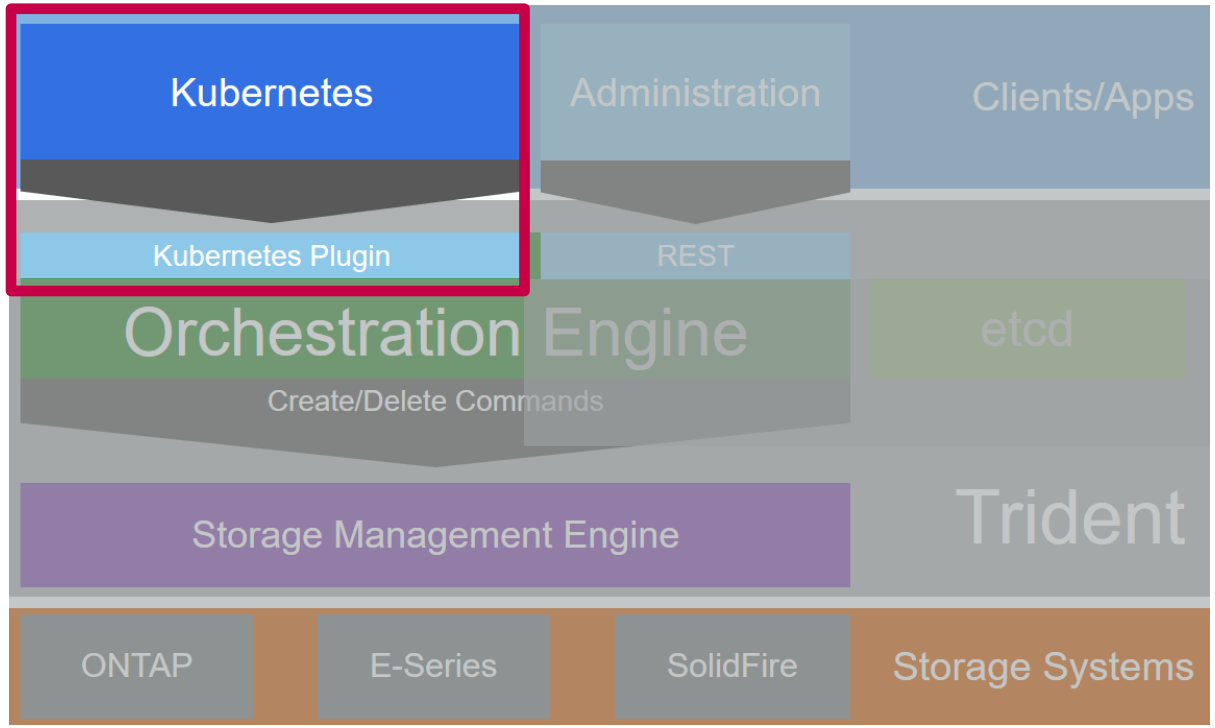
Trident – Management and Decision Engine

- Connects to backends to determine capabilities of storage array
- Receives requests for storage from frontends
  - E.g. Kubernetes, REST
- Evaluates configured backends against desired attributes in the storage class
  - Creates a list of eligible backends
- Selects one of the eligible backends, requests provisioning
  - Will retry on failure until all eligible backends are exhausted



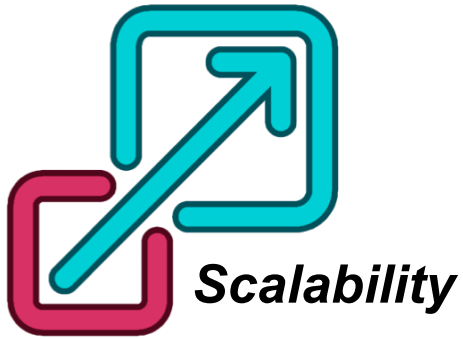
Trident – Kubernetes Frontend

- Handles interaction with Kubernetes
  - Configures watchers for tracking PVCs, PVs, and Storage Classes
- Evaluates whether Trident should take action, e.g. if the default provisioner or explicitly specified
- Forwards create/destroy actions to orchestration engine for execution
- Create/destroy Kubernetes PV, depending on requested action

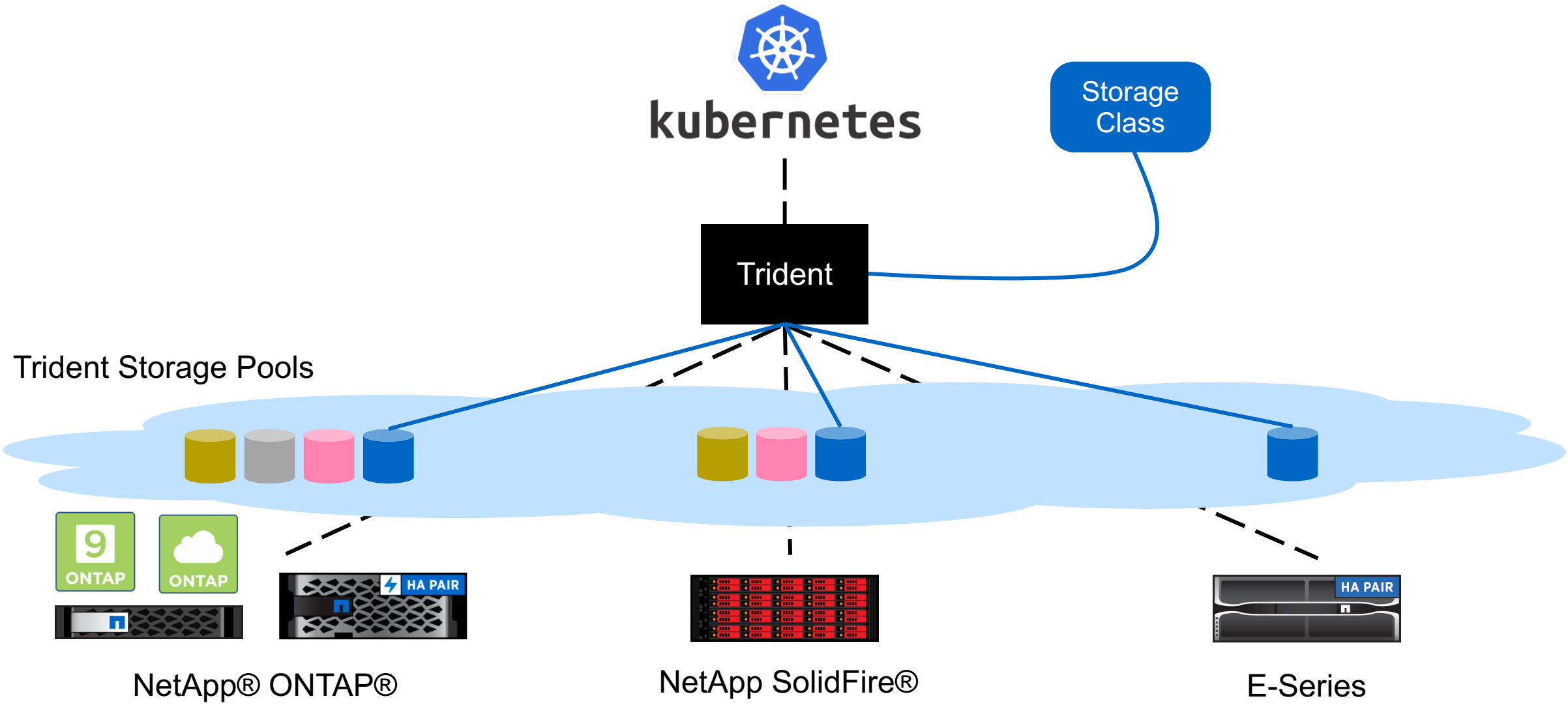


Trident Workflows

- Orchestrating storage across heterogeneous backends
- Provisioning storage
- Deprovisioning storage



Storage Orchestration



Trident Backends

- Supported platforms: NetApp® ONTAP®, SolidFire®, and E-Series
- Backends are managed using JSON config files
- Config files provide:
  - Connectivity information
    - Username / password, IP address(es)
  - Uniqueness information
    - Tenant name, storage prefix
  - Backend specific storage information
    - igroup name, access group IDs, data/LIF/SV IP address
    - QoS policy definitions
    - Pool name pattern

Trident Storage Pools

Platform	Storage Pool
ONTAP	Aggregates or FlexVols
SolidFire	QoS Profiles
E-Series	Volume Groups or Dynamic Disk Pools

Storage Pool Characteristics

Pool Attribute	Valid Values
media	ssd, hybrid, hdd
provisioningType	thin, thick
backendType	ontap-nas, ontap-san, solidfire-san, e-series
snapshots	true, false
IOPS	Positive integer range(s)

Mapping Storage Classes to Storage Pools

- Storage class parameters give Trident hints about the desired characteristics
  - Storage class parameters match storage pool attributes
  - Backend selection can be forced using `requiredStorage`
- Trident uses discovered pool attributes to match against requested parameters

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: bronze
provisioner: netapp.io/trident
parameters:
  media: "hdd"
  provisioningType: "thin"
  snapshots: "true"
```

Example 1: Performance, Multiple Platforms

- Trident matches this storage class to any storage pool that meet the specified criteria

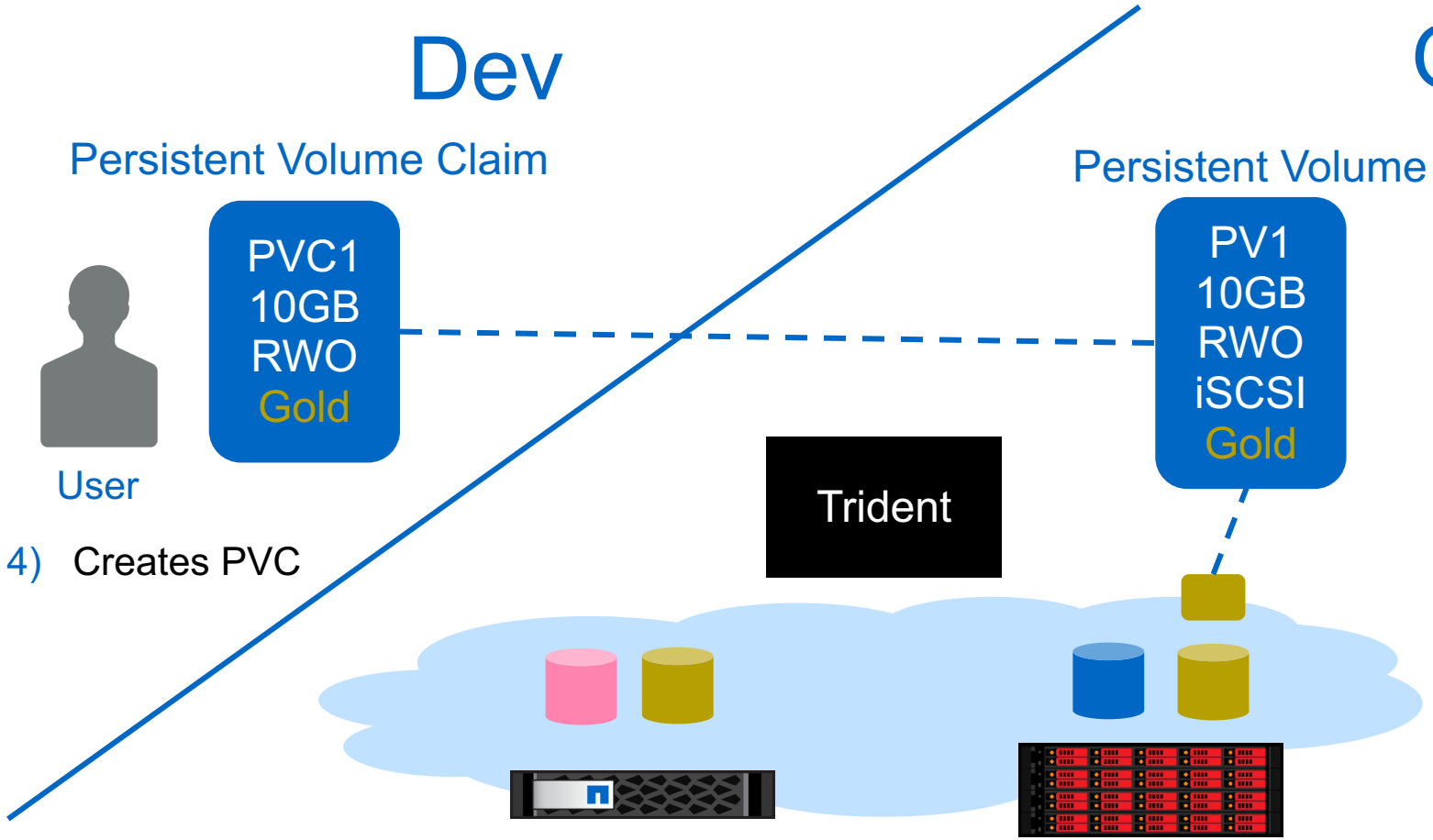
```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: performance
provisioner: netapp.io/trident
parameters:
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"
```

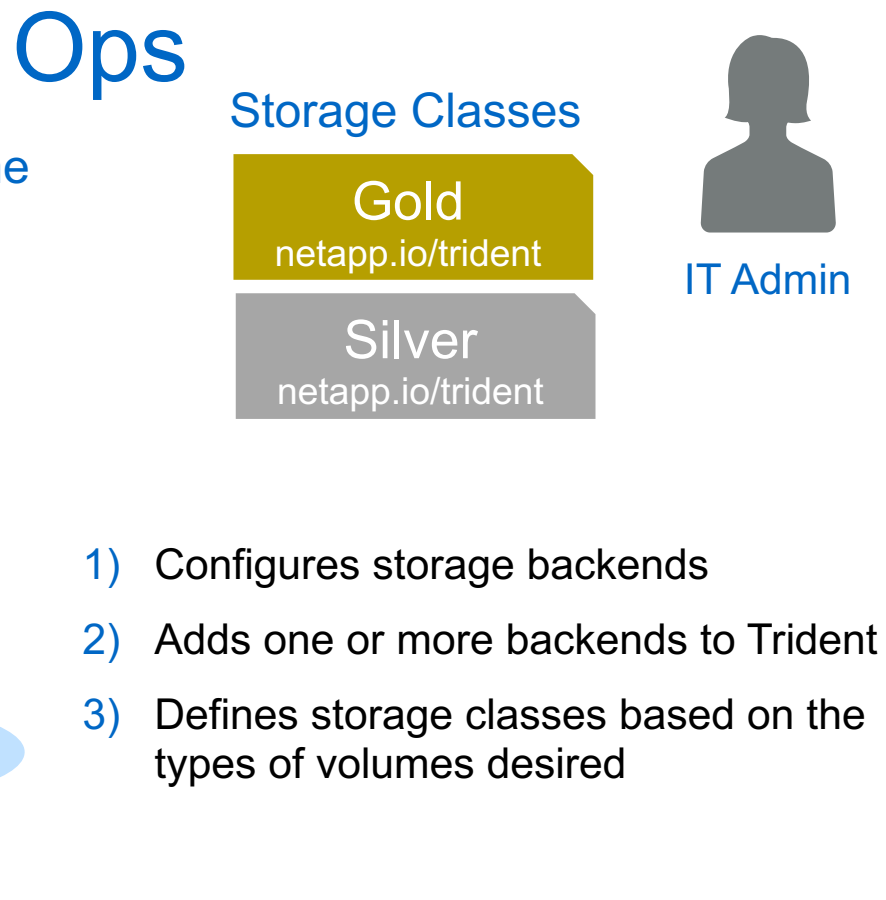
Example 2: Value, Single Platform, Default

- `requiredStorage` parameter will override all others, if specified
- Multiple backends, including SolidFire QoS profiles and ONTAP aggrs can be specified simultaneously

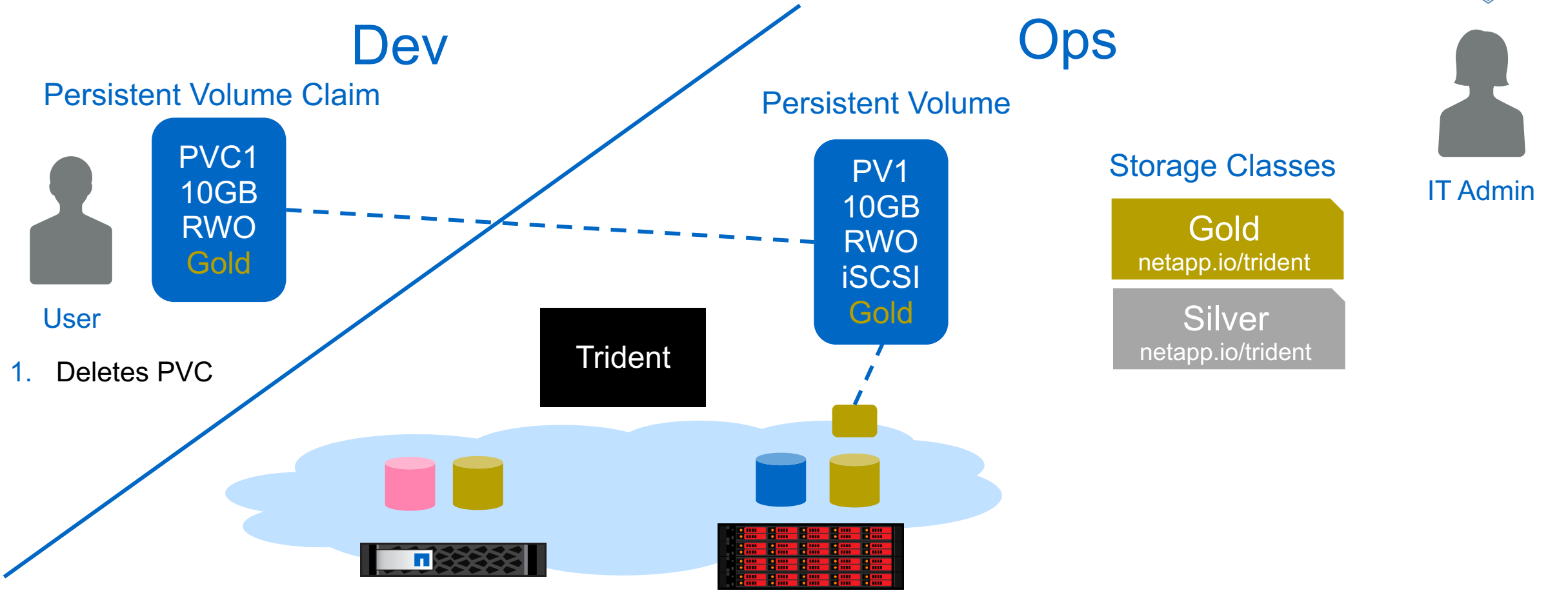
```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: value
  annotations:   storageclass.kubernetes.io/is-default-class: "true"
provisioner: netapp.io/trident
parameters:
  requiredStorage: "ontapnas_10.1.1.1:aggr1,aggr4,aggr10"
```

Storage Provisioning





Storage Deprovisioning



1. Deletes PVC
2. Detects deletion of PVC
3. Finds and validates the matching PV
4. Determines PV's reclaim policy
5. Deletes volume and PV if the reclaim policy is "Delete"

Improved Trident Install and Uninstall

- We heard from a number of users that the Trident install and uninstall processes were too complicated and difficult to use. So, we fixed it!
- Trident 17.07 introduces a greatly simplified install process which manages the creation and management of the Kubernetes resources needed by Trident. In addition to the pods and PV needed for Trident, this includes:
  - A dedicated namespace
  - A service account
  - RBAC roles

Simplified Trident Management using *tridentctl*

```
[root@trident ~]# tridentctl --help
A CLI tool for managing the NetApp Trident external storage provisioner for Kubernetes

Usage:
  tridentctl [command]

Available Commands:
  create      Add a resource to Trident
  delete      Remove one or more resources from Trident
  get         Get one or more resources from Trident
  help        Help about any command
  version     Print the version of Trident

Flags:
  -d, --debug               Debug output
  -h, --help                help for tridentctl
  -n, --namespace string    Namespace of Trident deployment
  -o, --output string       Output format.  One of json|yaml|name|wide|ps (default)
  -s, --server string       Address/port of Trident REST interface

Use "tridentctl [command] --help" for more information about a command.
```

Storage Classes Are Out of Beta

- Both the *StorageClass* and *PersistentVolumeClaim* objects have changed slightly as a result. The beta annotations will continue to work in 1.6 and 1.7, but you should take this opportunity to migrate to the attribute nomenclature now used since the beta annotation will be deprecated in the future.
- To illustrate the change, here is a PVC using the beta annotation nomenclature:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myfavoritepv
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: performance
```

Storage Classes Are Out of Beta

- This worked in 1.4 and 1.5, and continues to work in 1.6 and 1.7. However, as of 1.6, with storage classes coming out of beta and into stable status, there is a new way of specifying the storage class:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myfavoritepv
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: performance
```

- Notice that the “storageClassName” is now a part of the spec definition, but otherwise functions the same.

Dynamically Provisioned PVs Default to “delete” Reclaim Policy as of Kubernetes 1.6

- As the title implies, any PV created dynamically, by any provisioner, will default to the “delete” reclaim policy. This means that once the PV is released by the application, Kubernetes will ask the provisioner to destroy the underlying volume.
- If this is not the behavior you would like, changing the default behavior for volumes created by Trident is very easy, just set the trident.netapp.io/reclaimPolicy attribute on the storage class to the value you want.

Using a Default Storage Class

- Default storage classes, new in Kubernetes 1.6, make it super easy for your users to take advantage of persistence without having to worry about the infrastructure at all. Simply define a storage class and identify it as the default, and any PVC that does not specify a storage class will use it

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard
provisioner: netapp.io/trident
parameters:
  backendType: "ontap-nas"
  media: "hybrid"
  storageclass.kubernetes.io/is-default-class: "true"
```

- To list the storage classes available, and view the default, users can simply list the storage classes using this command:

```
$ kubectl get sc
NAME                    TYPE
archive                 netapp.io/trident
extreme                 netapp.io/trident
performance             netapp.io/trident
standard    (default)   netapp.io/trident
```

Kubernetes Resource Quotas

- Storage resource quotas are confined to a particular namespace, also known as projects in OpenShift.
- You can define them for the entire namespace across all storage classes or for particular storage classes, and use them to limit the number of persistent volumes and/or the capacity of those persistent volumes along those dimensions.

```
# define pvc count limit
cat <<EOF > pvc-count-limit.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: pvc-count-limit
spec:
  hard:
    persistentvolumeclaims: "5"
    value.storageclass.storage.k8s.io/persistentvolumeclaims: "3"
EOF

# create the limit
kubectl create -f ./pvc-count-limit.yaml --namespace=thepub
```

Databases, VM vs. Containers

VM

Guest	• Provision a VM
OS	• Install a supported OS
Prepare Host	• Install required packages
Storage	• Provision Volumes
Install	• Download and install DB
Run	• Configure and run DB
Test	• Test DB Connection

Container

K8s	• kubectl create -f my-db.yaml

Why Database in Containers?

- Easier standard **repeatable** deployments
- Easier **upgrades**
- Increase overall **reliability** and **availability**
- **Automated** and repeatable configuration
- Manage DBs not VMs/hosts
- **Speed**, **efficiency** and better CPU/Memory **utilization**

Why K8s?

- External **storage** management
- Distributing **Secrets**
- Application **health** checks
- **Replicating** application instances
- Horizontal Pod **auto scaling**
- **Load Balancing**
- Rolling updates
- **Monitoring** Resources

StatefulSet

- StatefulSets are a replicatied group of Pods similar to ReplicaSets, but unlike the ReplicaSets, they have certain unique properties:
  - Each replica gets a persistent hostname with a unique index (e.g. database-00, database-01 etc.)
  - Each replica is created in order from lowest to highest index, and create will block until the Pod at the previous index is healthy and available. This also applied to scaling up.
  - When deleted, each replica will be deleted in order from highest to lowest. This also applied to scaling down the number of replicas.
- In StatefulSets you can define your update strategy:
  - *OnDelete* - will only update a pod after it is manually deleted.
  - *RollingUpdate* - will delete and recreate the Pods in a StatefulSet from highest to lowest index.
  - *Partitions* - if a partition is specified, all Pods with an ordinal that is greater than or equal to the partition will be updated when the StatefulSet template is updated.