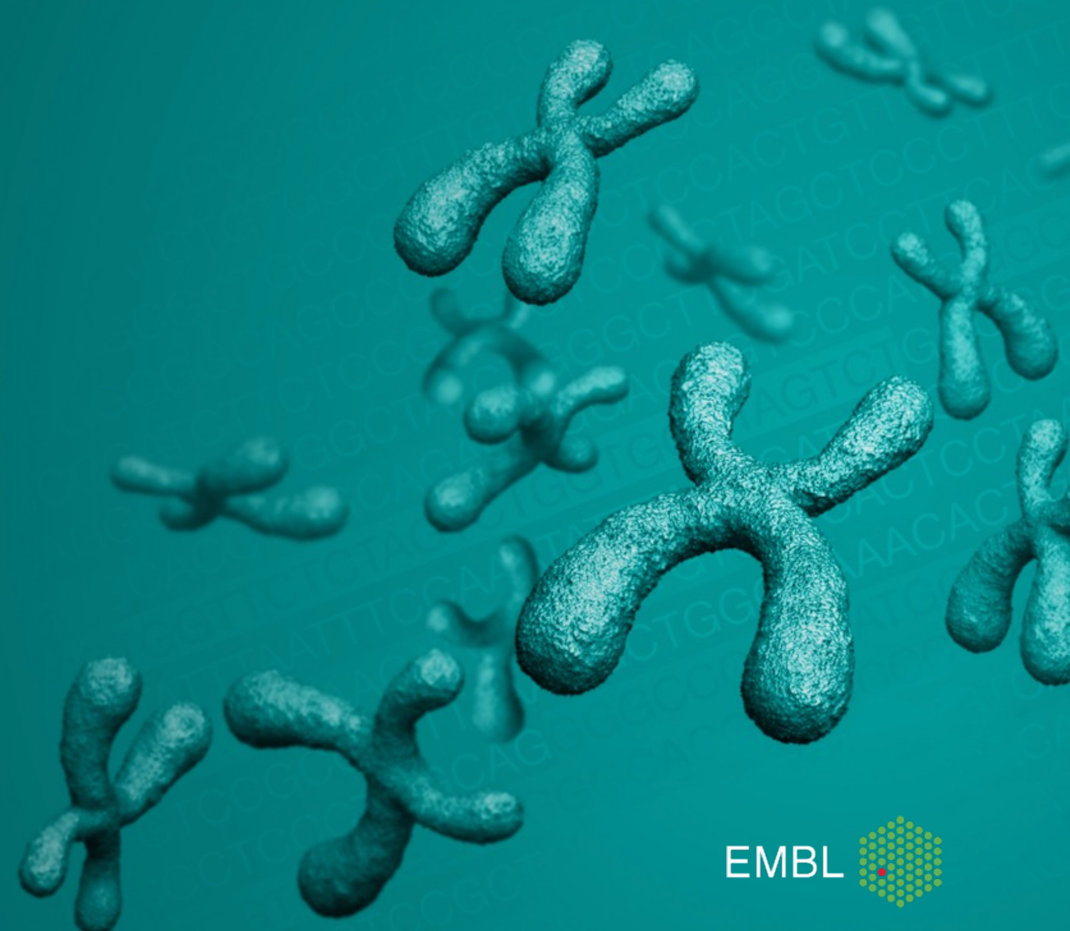
Kubernetes 101

C.D. Tiwari | @cdtiwari

Cloud Bioinformatics Application Architect,

Technology & Science Integration

EMBL-EBI



EMBL

What is Kubernetes?

- **Container Orchestrator**
  - Provision, manage, scale applications
- **Manage infrastructure resources needed by applications**
  - Volumes
  - Networks
  - Secrets
  - And many many many more…
- **Declarative model**
  - Provide the "desired state" and Kubernetes will make it happen
- **What's in a name?**
  - Kubernetes (K8s/Kube): "Helmsman" in ancient Greek

Decouples Infrastructure and Scaling

- **All services** within Kubernetes are natively Load Balanced.
- Can scale up and down dynamically.
- Used both to enable self-healing and seamless upgrading or rollback of applications.

Self Healing

Kubernetes will **ALWAYS** try and steer the cluster to its desired state.

- **Me:** “I want 3 healthy instances of redis to always be running.”
- **Kubernetes:** “Okay, I’ll ensure there are always 3 instances up and running.”
- **Kubernetes:** “Oh look, one has died. I’m going to attempt to spin up a new one.”

Kubernetes Resource Model

- A resource for every purpose

- Config Maps
- Daemon Sets
- **Deployments**
- Events
- Endpoints
- Ingress
- Jobs
- Nodes
- Namespaces
- **Pods**
- **Persistent Volumes**
- **Replica Sets**
- Secrets
- Service Accounts
- Services
- Stateful Sets, and more...

- Kubernetes aims to have the building blocks on which you build a cloud native platform.

- Therefore, the internal resource model **is** the same as the end user resource model.

Key Resources

- Pod: set of co-located containers
  - Smallest unit of deployment
  - Several types of resources to help manage them
  - Replica Sets, Deployments, Stateful Sets, ...

- Services
  - Define how to expose your app as a DNS entry
  - Query based selector to choose which pods apply

Core Concepts

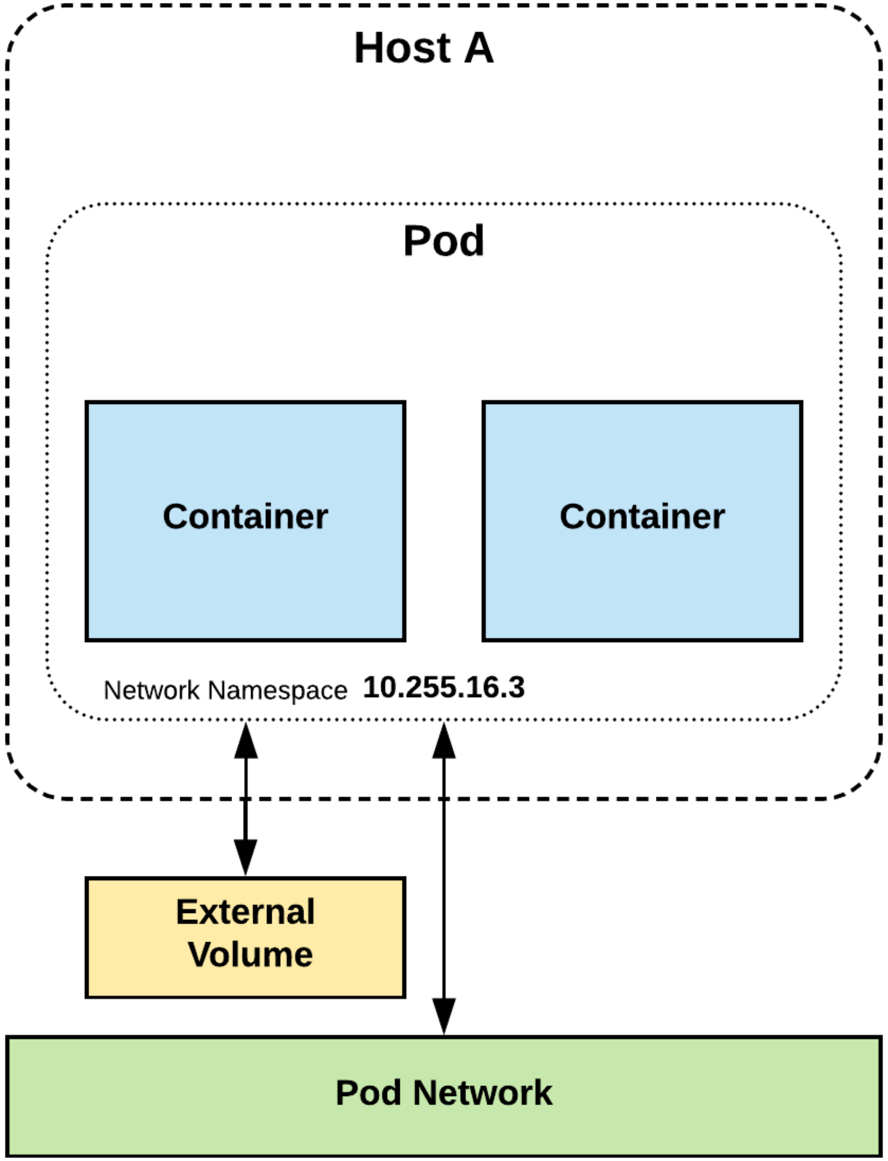
- Kubernetes has several core building blocks that make up the foundation of their higher level components.

Basic K8s Object

- Pods
- Volume
  - Persistent Vol, Persistent Vol Claims
  - StorageClass
- ReplicaSet
- Deployment

Pods

- **Atomic unit** or smallest *"unit of work" of* Kubernetes.
- Pods are **one or MORE containers** that share volumes, a network namespace, and are a part of a **single context.**
- They are also ephemeral
- [Working with pods!!]()



Pod Examples

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: multi-container-example
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
  - name: content
    image: alpine:latest
    command: ["/bin/sh", "-c"]
    args:
      - while true; do
          date >> /html/index.html;
          sleep 5;
        done
    volumeMounts:
    - name: html
      mountPath: /html
  volumes:
  - name: html
    emptyDir: {}
```

Key Pod Container Attributes

Container

```
name: nginx
image: nginx:stable-alpine
ports:
  - containerPort: 80
    name: http
    protocol: TCP
env:
  - name: MYVAR
    value: isAwesome
command: [“/bin/sh”, “-c”]
args: [“echo ${MYVAR}”]
```

- `name` - The name of the container
- `image` - The container image
- `ports` - array of ports to expose. Can be granted a friendly name and protocol may be specified
- `env` - array of environment variables
- `command` - Entrypoint array (equiv to Docker ENTRYPOINT)
- `args` - Arguments to pass to the command (equiv to Docker CMD)

Storage

- Pods by themselves are useful, but many workloads require exchanging data between containers, or persisting some form of data.
- For this we have
  - **Volumes**,
  - **PersistentVolumes**,
  - **PersistentVolumeClaims**, and
  - **StorageClasses**.

Volumes

- Storage that is tied to the **Pod's Lifecycle.**
- A pod can have one or more types of volumes attached to it.
- Can be consumed by any of the containers within the pod.
- Survive Pod restarts; however their durability beyond that is dependent on the Volume Type.
- Types of Volumes

Volumes

- `volumes:` A list of volume objects to be attached to the Pod. Every object within the list must have it’s own unique `name`.
- `volumeMounts:` A container specific list referencing the Pod volumes by `name`, along with their desired `mountPath`.

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-example
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
      ReadOnly: true
  - name: content
    image: alpine:latest
    command: ["/bin/sh", "-c"]
    args:
      - while true; do
          date >> /html/index.html;
          sleep 5;
        done
    volumeMounts:
    - name: html
      mountPath: /html
  volumes:
  - name: html
    emptyDir: {}
```

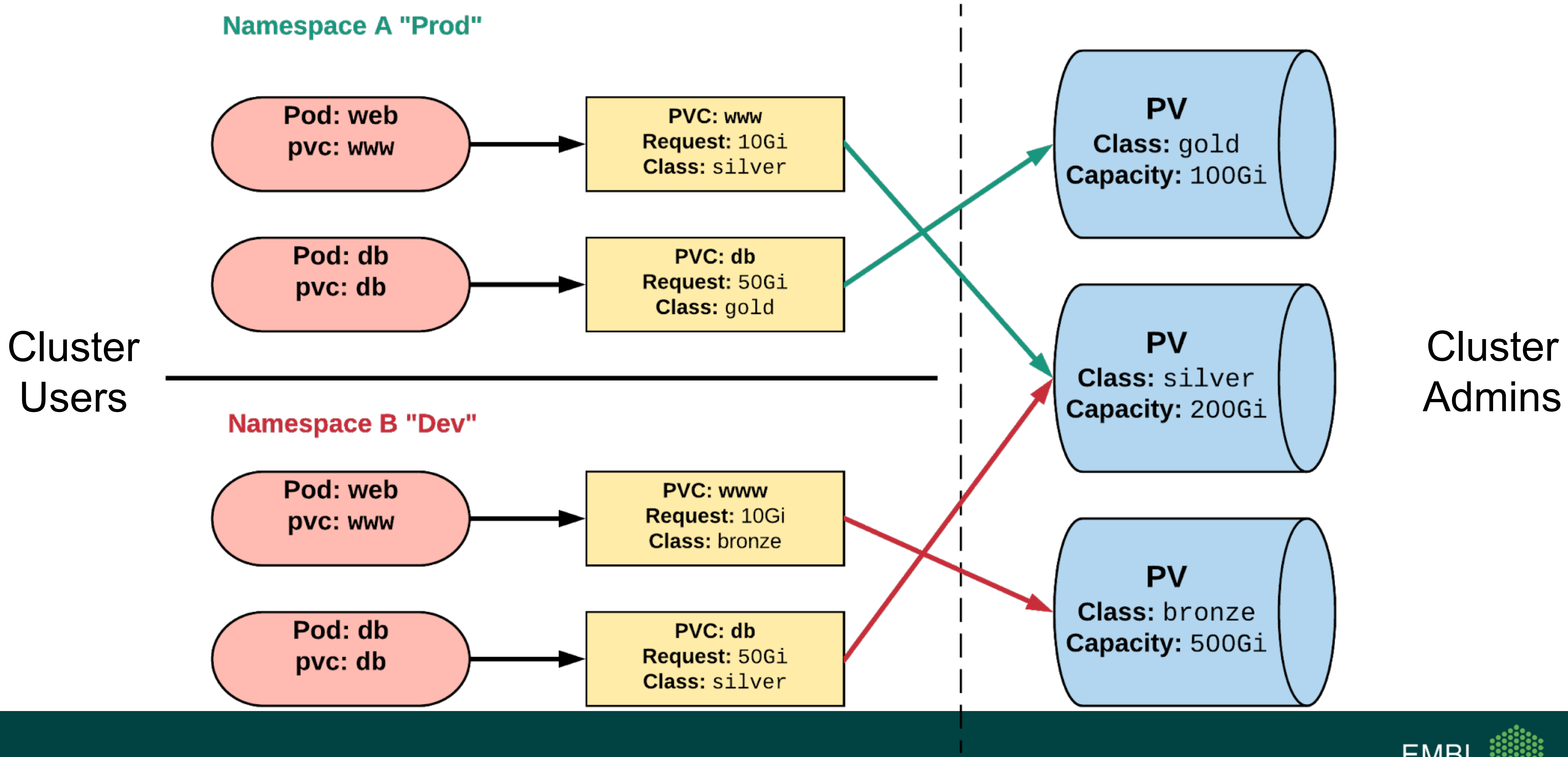
Persistent Volumes

- A **PersistentVolume** (PV) represents a storage resource.
- PVs are a **cluster wide resource** linked to a backing storage provider: NFS, GCEPersistentDisk, RBD etc.
- Generally provisioned by an administrator.
- Their lifecycle is handled independently from a pod
- **CANNOT** be attached to a Pod directly. Relies on a **PersistentVolumeClaim**
- Ref: PersistentVolumes

PersistentVolumeClaims

- A **PersistentVolumeClaim** (PVC) is a **namespaced** request for storage.
- Satisfies a set of requirements instead of mapping to a storage resource directly.
- Ensures that an application's '*claim*' for storage is portable across numerous backends or providers.
- Ref: PersistentVolumeClaims

Persistent Volumes and Claims



Namespace A "Prod"

Pod: web
pvc: www

PVC: www
Request: 10Gi
Class: silver

Pod: db
pvc: db

PVC: db
Request: 50Gi
Class: gold

Cluster Users

Namespace B "Dev"

Pod: web
pvc: www

PVC: www
Request: 10Gi
Class: bronze

Pod: db
pvc: db

PVC: db
Request: 50Gi
Class: silver

PV
Class: gold
Capacity: 100Gi

PV
Class: silver
Capacity: 200Gi

PV
Class: bronze
Capacity: 500Gi

Cluster Admins

PersistentVolume

- `capacity.storage:` The total amount of available storage.
- `volumeMode:` The type of volume, this can be either `Filesystem` or `Block`.
- `accessModes:` A list of the supported methods of accessing the volume. Options include:
  - `ReadWriteOnce`
  - `ReadOnlyMany`
  - `ReadWriteMany`
- `persistentVolumeReclaimPolicy:` The behaviour for PVC's that have been deleted. Options include:
  - `Retain` - manual clean-up
  - `Delete` - storage asset deleted by provider.
- `storageClassName:` Optional name of the storage class that PVC's can reference. If provided, **ONLY** PVC's referencing the name consume use it.
- `mountOptions:` Optional mount options for the PV.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfsserver
spec:
  capacity:
    storage: 50Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Delete
  storageClassName: silver
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /exports
    server: 172.22.0.42
```

PersistentVolumeClaim

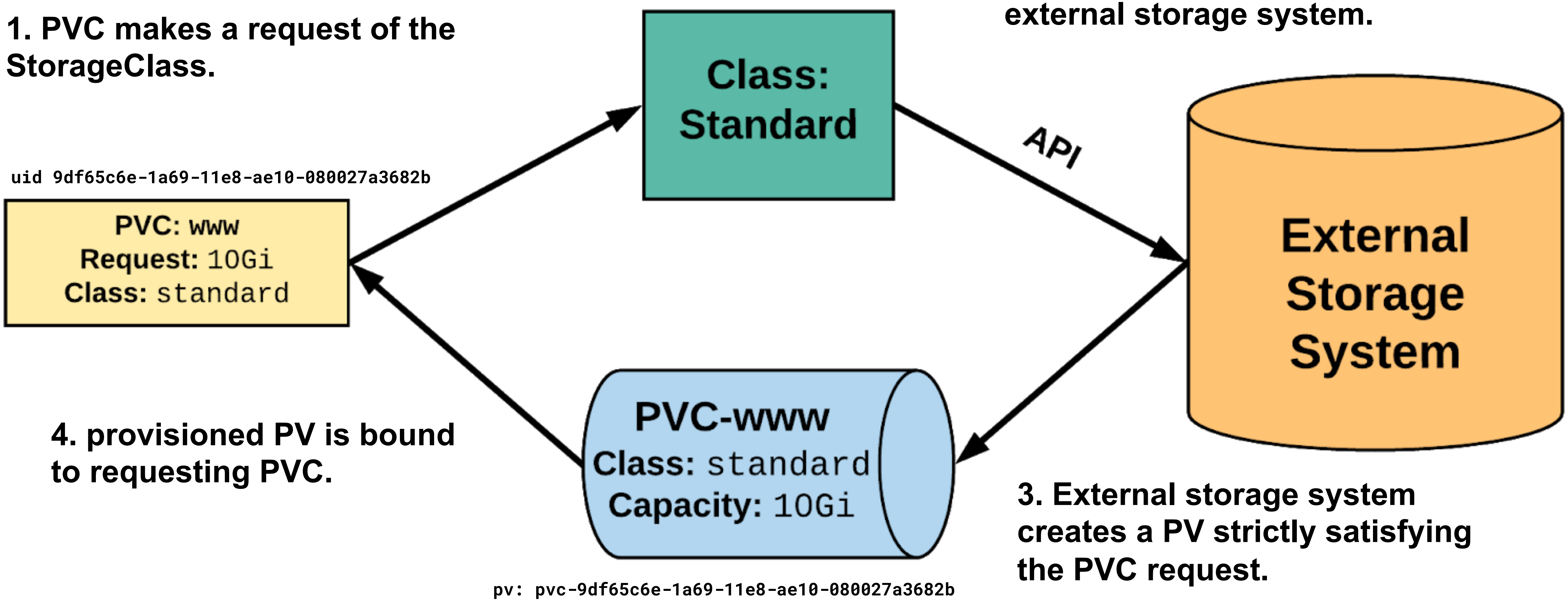
- `accessModes:` The selected method of accessing the storage. This **MUST** be a subset of what is defined on the target PV or Storage Class.
    - `ReadWriteOnce`
    - `ReadOnlyMany`
    - `ReadWriteMany`
- `resources.requests.storage:` The desired amount of storage for the claim
- `storageClassName:` The name of the desired Storage Class

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-sc-example
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: silver
```

Storage Classes

- Storage classes are an abstraction on top of an external storage resource (PV)
- Work hand-in-hand with the external storage system to enable **dynamic provisioning** of storage
- Eliminates the need for the cluster admin to pre-provision a PV
- Ref: Storage Classes

StorageClass



1. PVC makes a request of the StorageClass.

2. StorageClass provisions request through API with external storage system.

3. External storage system creates a PV strictly satisfying the PVC request.

4. provisioned PV is bound to requesting PVC.

uid 9df65c6e-1a69-11e8-ae10-080027a3682b

PVC: www
Request: 10Gi
Class: standard

Class: Standard

API

External Storage System

PVC-www
Class: standard
Capacity: 10Gi

pv: pvc-9df65c6e-1a69-11e8-ae10-080027a3682b

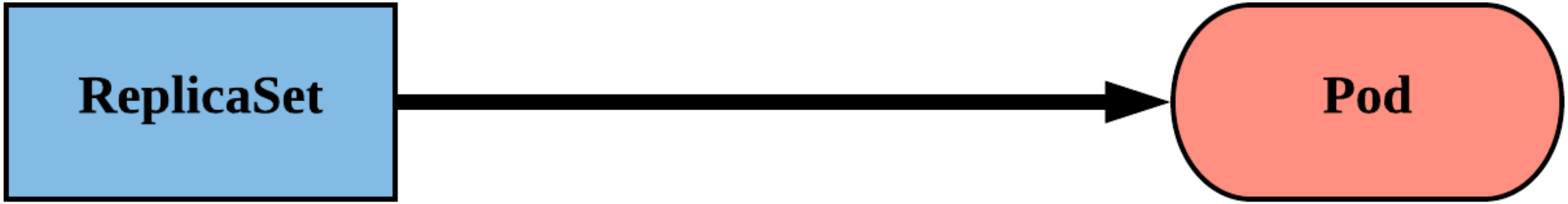
Storage Class

- `provisioner:` Defines the ‘*driver*’ to be used for provisioning of the external storage.
- `parameters:` A hash of the various configuration parameters for the provisioner.
- `reclaimPolicy:` The behaviour for the backing storage when the PVC is deleted.
  - `Retain` - manual clean-up
  - `Delete` - storage asset deleted by provider

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  zones: us-central1-a, us-central1-b
reclaimPolicy: Delete
```

ReplicaSet

- Primary method of managing pod replicas and their lifecycle.
- Includes their scheduling, scaling, and deletion.
- Their job is simple: **Always ensure the desired number of pods are running.**
- Ref: when to use a replicaset?



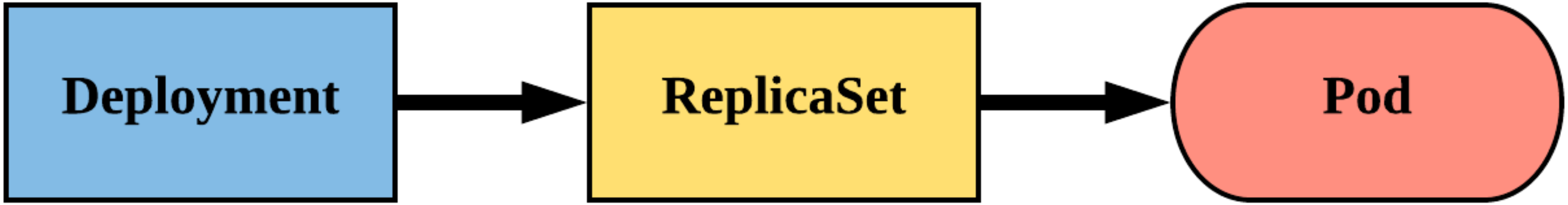
ReplicaSet

- `replicas:` The desired number of instances of the Pod.
- `selector:`The label selector for the **ReplicaSet** will manage **ALL** Pod instances that it targets.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-example
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
      env: prod
  template:
    <pod template>
```

Deployment

- Declarative method of managing Pods via **ReplicaSets.**
- Provide rollback functionality and update control.
- Each iteration creates a unique label that is assigned to both the **ReplicaSet** and subsequent Pods.
- Ref: [Creating a Deployment]()



Deployment

- revisionHistoryLimit: The number of previous iterations of the Deployment to retain.
- strategy: Describes the method of updating the Pods based on the type. Valid options are Recreate or RollingUpdate.
  - Recreate: All existing Pods are killed before the new ones are created.
  - RollingUpdate: Cycles through updating the Pods according to the parameters: maxSurge and maxUnavailable.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-example
spec:
  replicas: 3
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: nginx
      env: prod
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    <pod template>
```

Kubernetes Client

- CLI tool to interact with Kubernetes cluster

- Platform specific binary available to download
  - https://kubernetes.io/docs/tasks/tools/install-kubectl

- The user directly manipulates resources via json/yaml

```
$ kubectl (create|get|apply|delete) -f myResource.yaml
```

Summary

- K8s objects are defined in YAML files;
- **kubectl** CLI is used to create/update/delete k8s objects.
- **Pods** are the smallest deployable units of computing that can be created and managed in Kubernetes.
- **Services** is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service.
- Storage is defined at the administrative level by **storage classes.**
- Storage is accessed in pods by **claiming** a **persistent volume.**
- **ReplicaSet** and **Deployments** provides higher-level management of pods.

Kubernetes Resources

- Main Website - http://kubernetes.io
- K8s Documentation
- Youtube Channel
- Many SIG’s(Special Interest Groups), Zoom

Thanks, Our team.

David Yuan

Steven Newhouse

Tony Wildish

C.D.

Sammy

EMBL