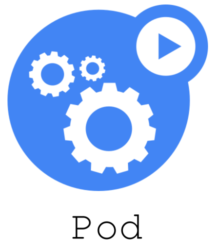
Kubernetes Storage 101

David Zhu, Google
Jan Šafránek, Red Hat

Kubernetes

- ~~Container~~ Pod orchestrator.
    - Pod = one or more containers.
    - Containers are stateless.
        - Cleared on exit.
        - Unless a *persistent volume* is used.

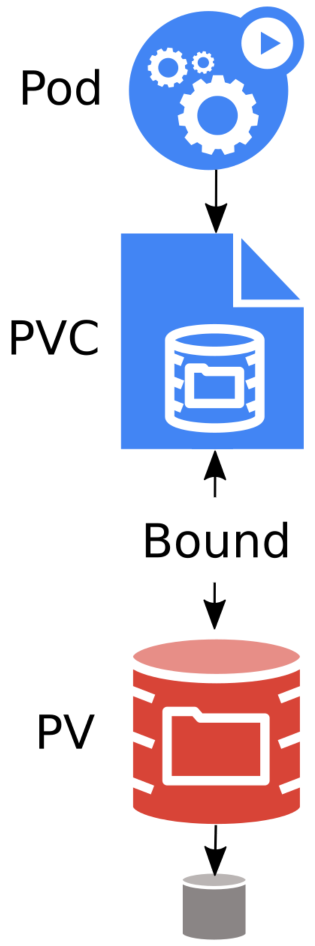
Pod



```
kind: Pod
apiVersion: v1
metadata:
  name: mysql
spec:
  containers:
  - image: mysql:5.6
    name: mysql
    ports:
    - containerPort: 3306
      name: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: opensesame
```

- Database is lost when `mysql` container ends!

Kubernetes Persistent Storage Objects



Pod

- Mounts `PersistentVolumeClaim` into container(s).

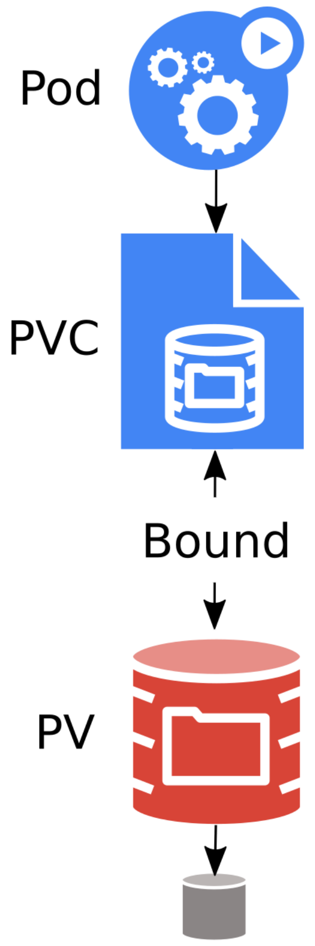
PersistentVolumeClaim (PVC)

- Application request for storage.
- Created by user / devops.
- Binds to single PV.
- Usable in Pods.

PersistentVolume (PV)

- Pointer to physical storage.
- Binds to single PVC.
- Created by admin ("pre-provisioning").
- Created by Kubernetes on demand ("dynamic provisioning").

Kubernetes Persistent Storage Objects Portability



Portable across Kubernetes clusters.

- Pod
- PersistentVolumeClaim (PVC)

Not portable across Kubernetes clusters.

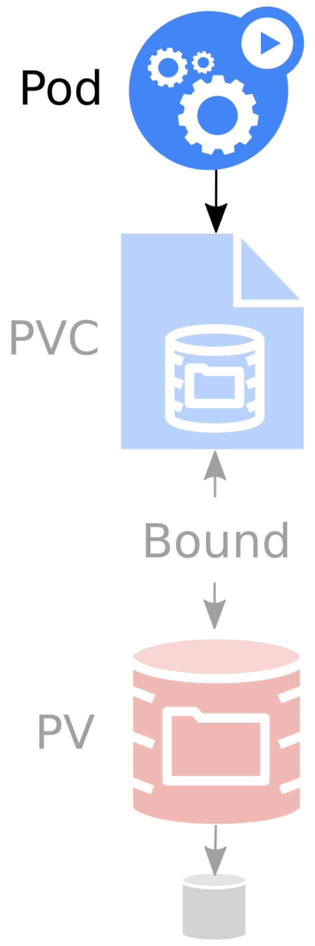
- PersistentVolume (PV)
- StorageClass
- Both contain details about the storage:
  - Volume plugin.
  - IP addresses of storage server(s).
  - Paths.
  - Usernames / passwords.
  - ...

Kubernetes Persistent Storage Objects

`StorageClass`

- Collection of PersistentVolumes with the same characteristics.
  - "Fast", "Cheap", "Replicated", ...
- Parameters for dynamic provisioning.
- Created by admin.
- Subject of quota per namespace.

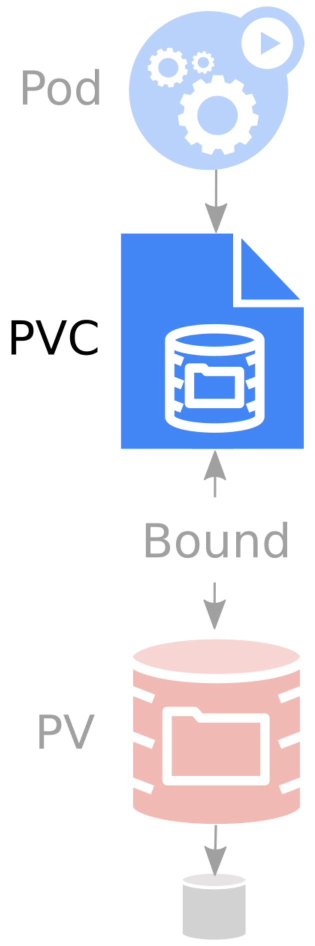
Pod



Mounts `PersistentVolumeClaim` into container(s).

```
kind: Pod
apiVersion: v1
metadata:
  name: mysql
spec:
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: my-mysql-claim
  containers:
  - image: mysql:5.6
    name: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: opensesame
    volumeMounts:
    - name:  data
      mountPath: /var/lib/mysql
```

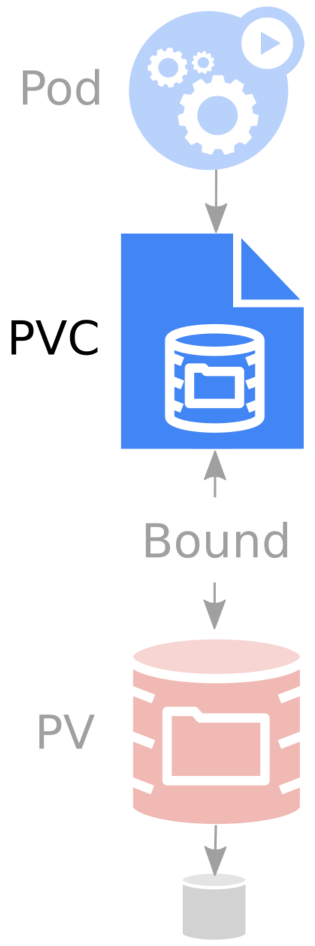
PersistentVolumeClaim



Request for storage.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my-mysql-claim
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteOnce
```

PersistentVolumeClaim

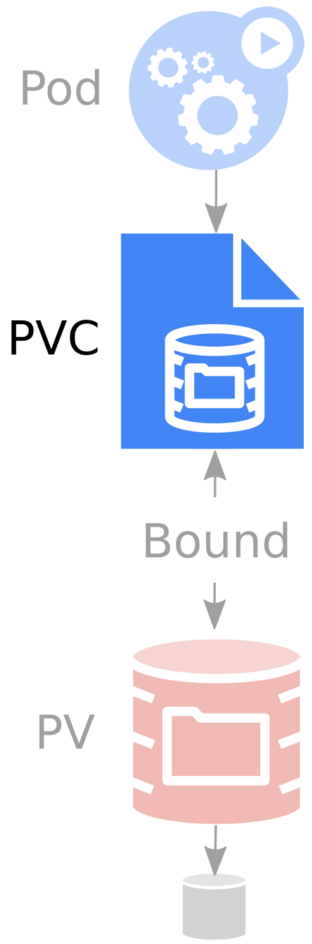


Request for storage.

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my-mysql-claim
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteOnce
```

- *"Give me 1 GiB of storage."*

PersistentVolumeClaim

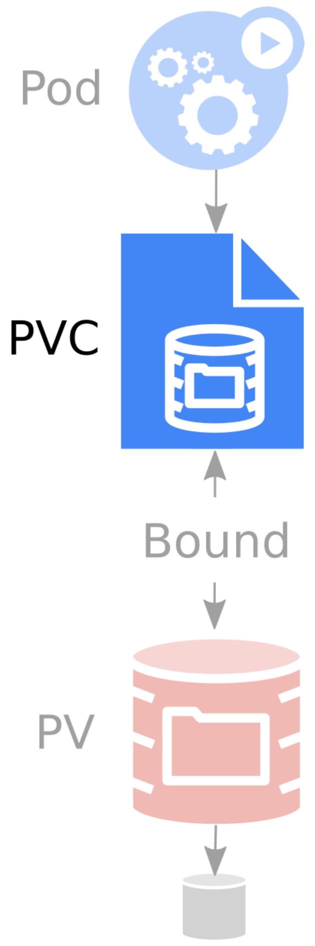


Request for storage.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my-mysql-claim
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteOnce
```

- *"Give me 1 GiB of storage."*
- *"That is mountable to single pod as read/write."*

PersistentVolumeClaim

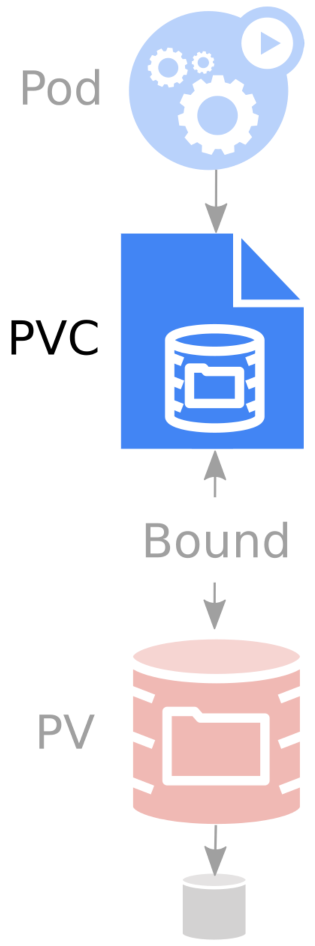


Request for storage.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my-mysql-claim
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteOnce
```

- *"Give me 1 GiB of storage."*
- *"That is mountable to single pod as read/write."*
- *"And I don't really care about the rest."*

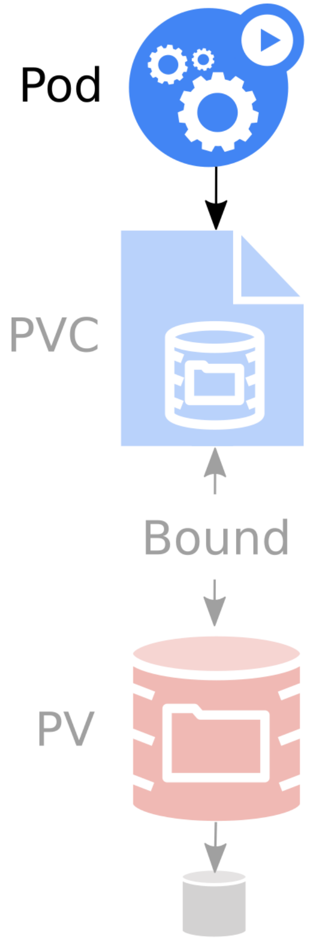
PVC creation



```
$ kubectl create -f claim.yaml
persistentvolumeclaim/my-mysql-claim created

$ kubectl get pvc
NAME             STATUS  VOLUME    CAPACITY ACCESS MODES  STORAGECLASS AGE
my-mysql-claim   Bound   pvc-6428  1Gi      RWO           standard     26s
```

Pod creation



```
$ kubectl create -f pod.yaml
pod/mysql created

$ kubectl get pod
NAME     READY    STATUS     RESTARTS    AGE
mysql    1/1      Running    0           19s
```

PVC debugging

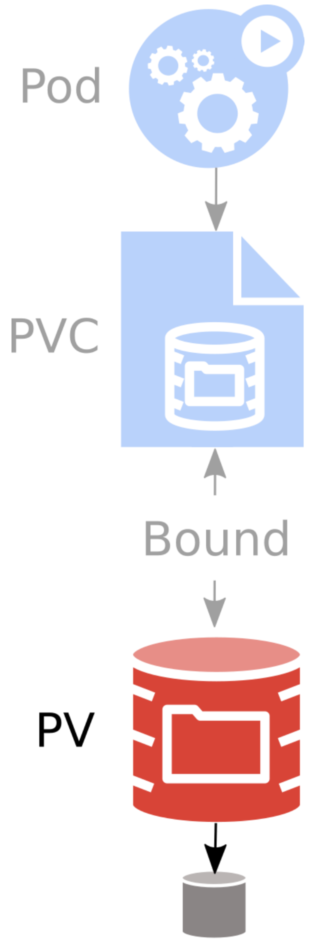
```
$ kubectl get pvc
NAME                 STATUS
my-broken-claim      Pending
```

PVC debugging

```
$ kubectl get pvc
NAME                 STATUS
my-broken-claim      Pending

$ kubectl describe pvc
...
Events:
  Type     Reason              Age              From                         Message
  ----     ------              ----             ----                         -------
  Warning  ProvisioningFailed  8s (x4 over 53s)  persistentvolume-controller  storageclass.storage.k8s
.io "foo" not found
```

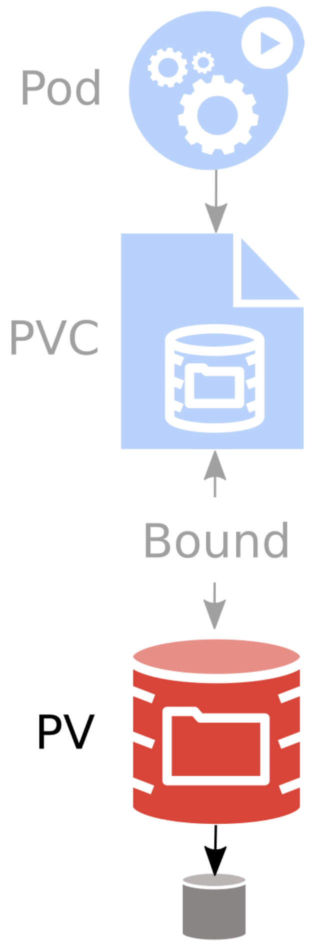
PersistentVolume



```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
    - ReadWriteOnce
    - ReadOnlyMany
  storageClassName: cheap
  persistentVolumeReclaimPolicy: Retain
  nfs:
    server: 192.168.121.1
    path: "/vol/share-1"
```

- Some metadata.

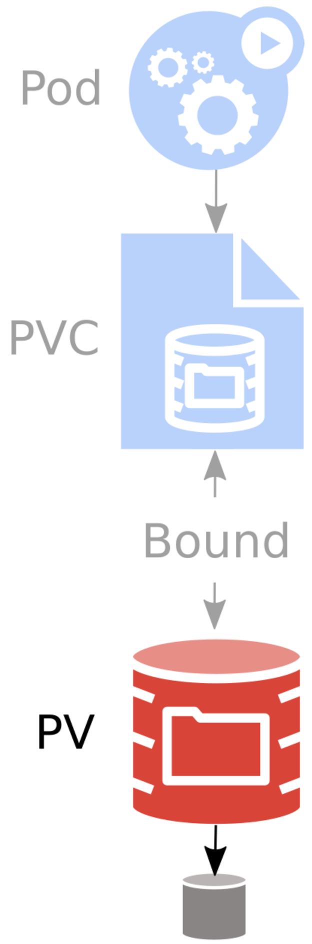
PersistentVolume



```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
    - ReadWriteOnce
    - ReadOnlyMany
  storageClassName: cheap
  persistentVolumeReclaimPolicy: Retain
  nfs:
    server: 192.168.121.1
    path: "/vol/share-1"
```

- Pointer to storage.
  - AWS EBS, Azure DD, Ceph FS & RBD, CSI, FC, Flex, GCE PD, Gluster, iSCSI, NFS, OpenStack Cinder, Photon, Quobyte, StorageOS, vSphere

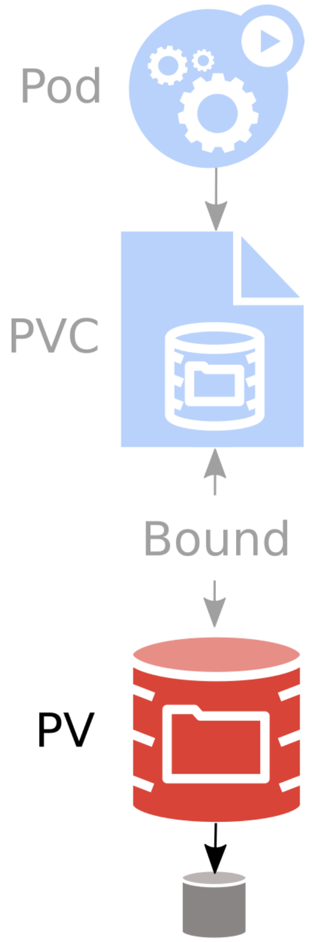
PersistentVolume



```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
    - ReadWriteOnce
    - ReadOnlyMany
  storageClassName: cheap
  persistentVolumeReclaimPolicy: Retain
  nfs:
    server: 192.168.121.1
    path: "/vol/share-1"
```

- Size of the volume.

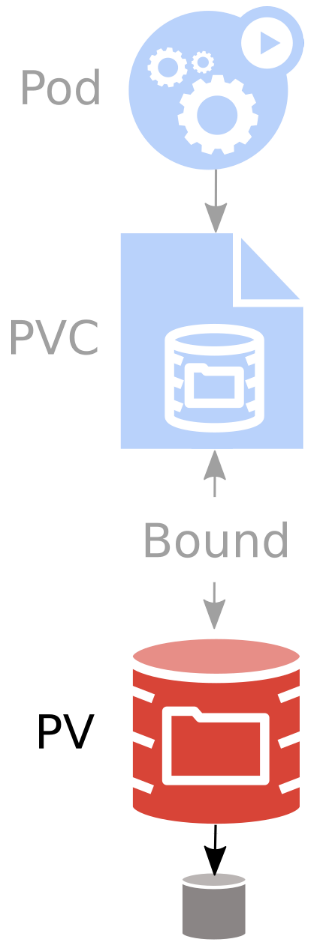
PersistentVolume



```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
    - ReadWriteOnce
    - ReadOnlyMany
  storageClassName: cheap
  persistentVolumeReclaimPolicy: Retain
  nfs:
    server: 192.168.121.1
    path: "/vol/share-1"
```

- Access modes that the volume supports.

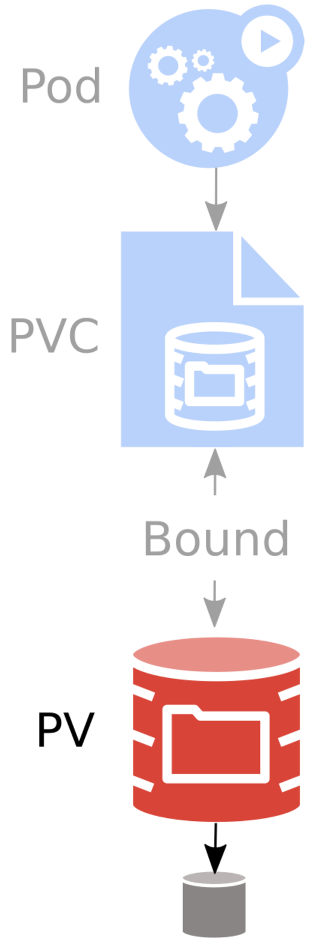
PersistentVolume



```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
    - ReadWriteOnce
    - ReadOnlyMany
  storageClassName: cheap
  persistentVolumeReclaimPolicy: Retain
  nfs:
    server: 192.168.121.1
    path: "/vol/share-1"
```

- `StorageClass` where this volume belongs.

PersistentVolume



```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
    - ReadWriteOnce
    - ReadOnlyMany
  storageClassName: cheap
  persistentVolumeReclaimPolicy: Retain
  nfs:
    server: 192.168.121.1
    path: "/vol/share-1"
```

- What to do when the volume is not needed any longer.
  - `Recycle` (deprecated), `Retain`, `Delete`

StorageClass

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "50"
```

- Collection of PersistentVolumes with the same characteristics.
- Usually admin territory.
- Global, not namespaced.

StorageClass

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "50"
```

- Who dynamically provisions volumes.
  - Name of hardcoded volume plugin.
  - Name of external provisioner.
  - Name of CSI driver.

StorageClass

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "50"
```

- Parameters for dynamic provisioning.
  - Depend on the provisioner.

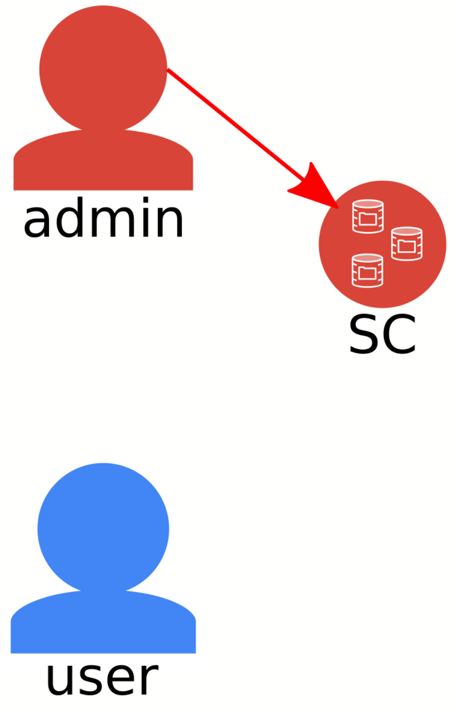
StorageClass

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "50"
```

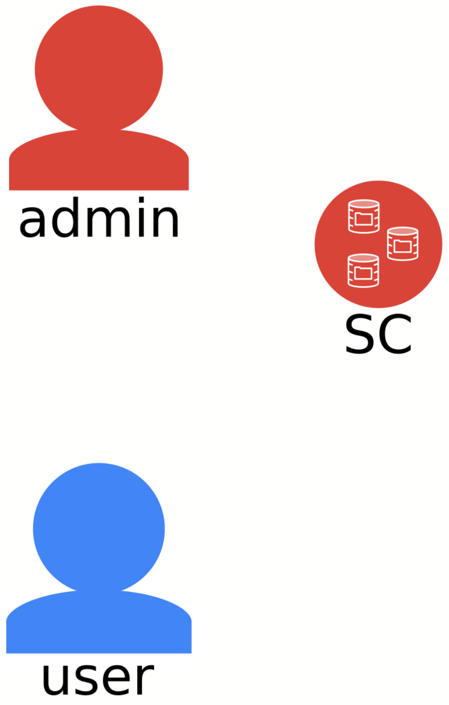
- One `StorageClass` in the cluster can be default.
  - PVC without any `StorageClass` gets the default one.

PersistentVolume Life Cycle

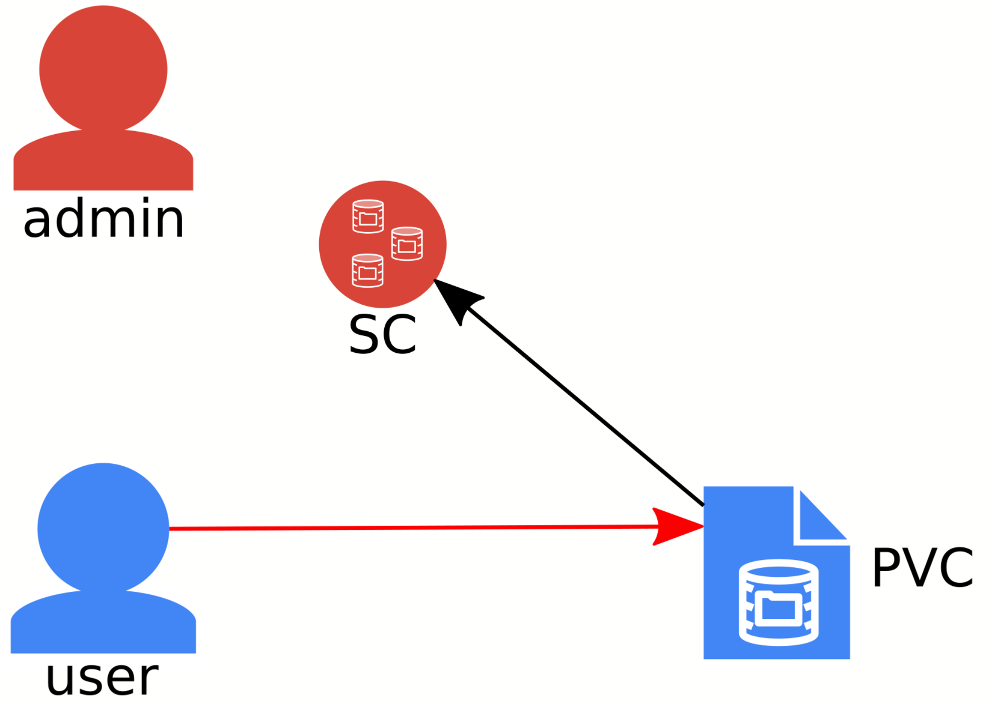
PersistentVolume: Dynamic Provisioning



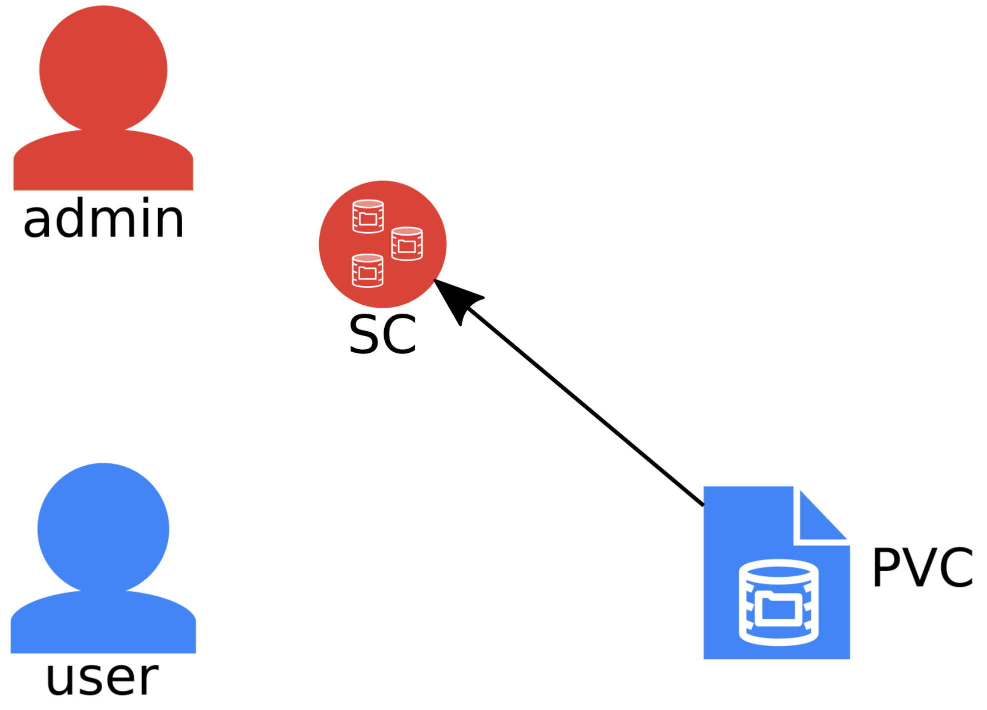
PersistentVolume: Dynamic Provisioning



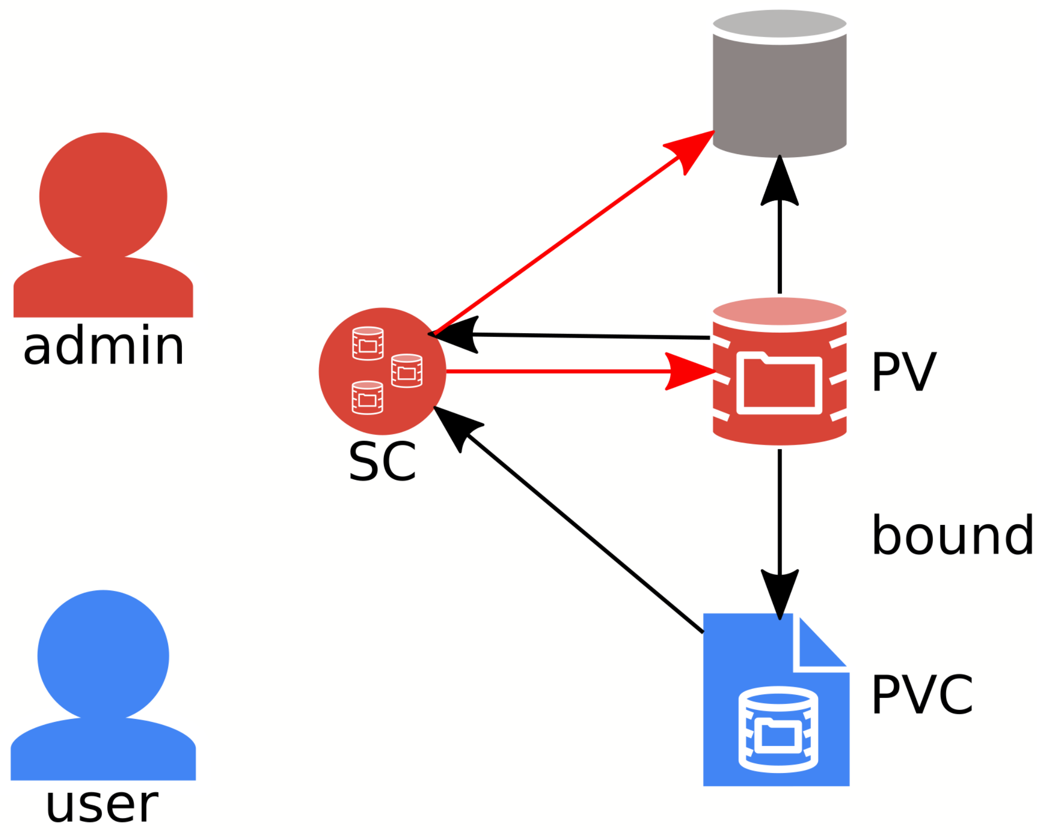
PersistentVolume: Dynamic Provisioning



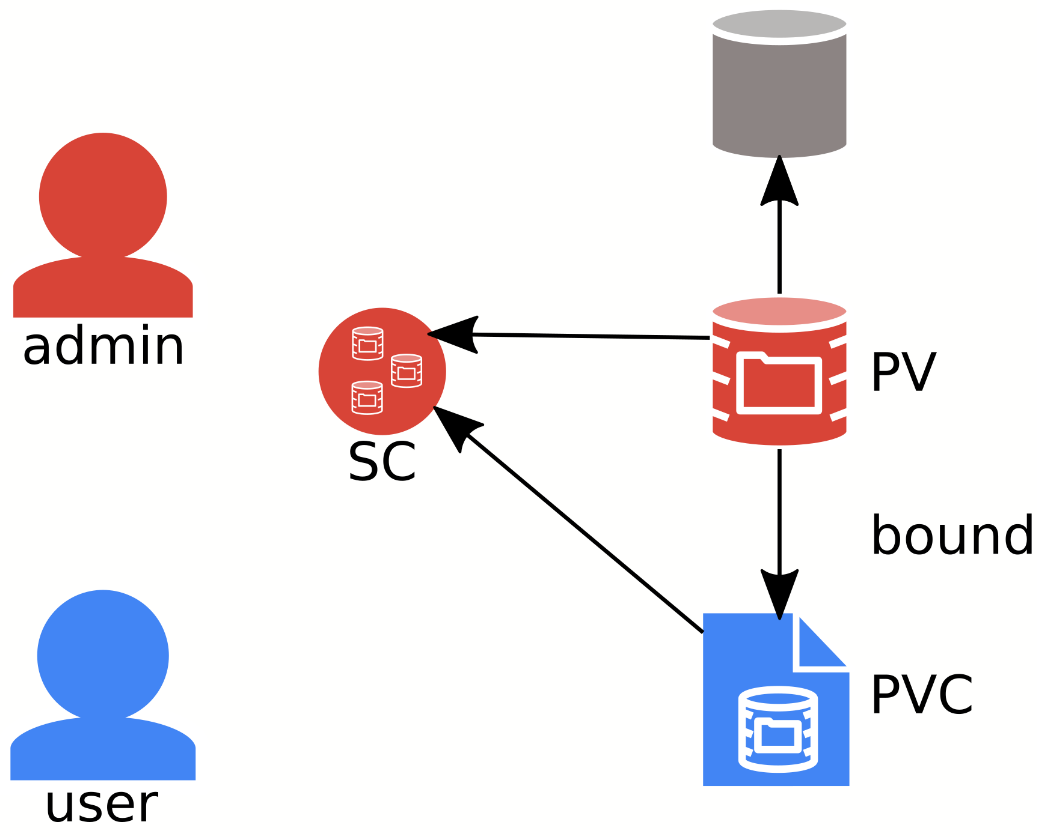
PersistentVolume: Dynamic Provisioning



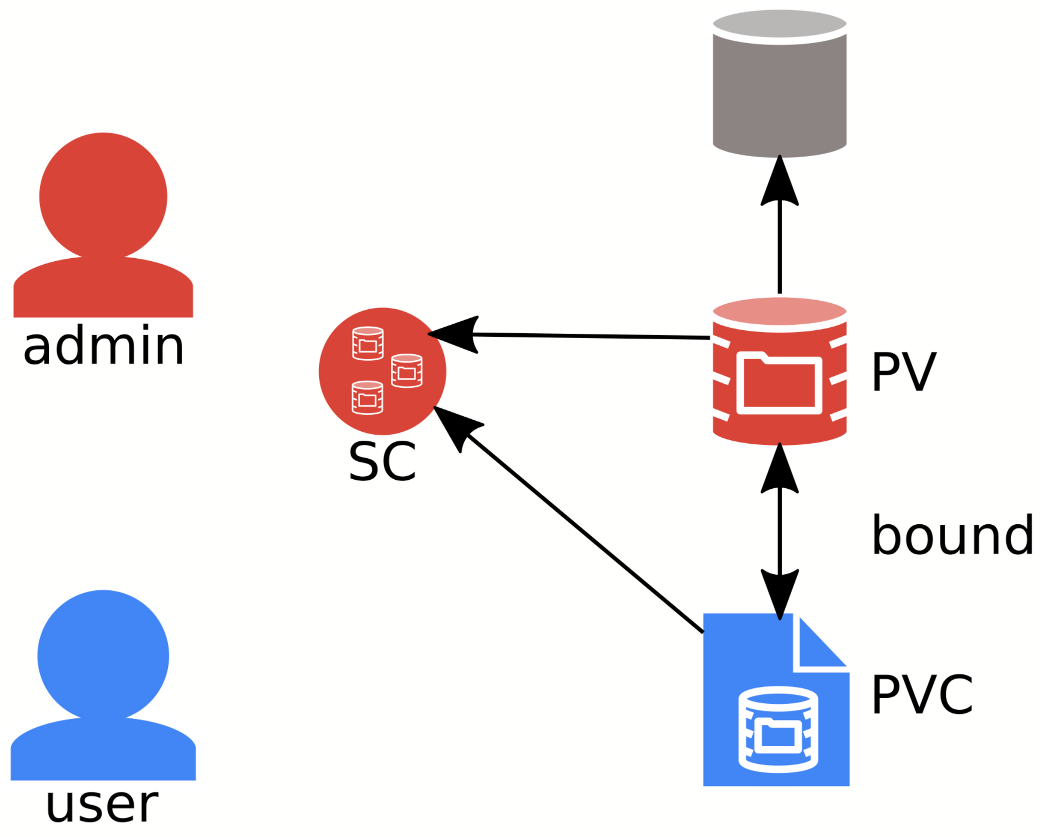
PersistentVolume: Dynamic Provisioning



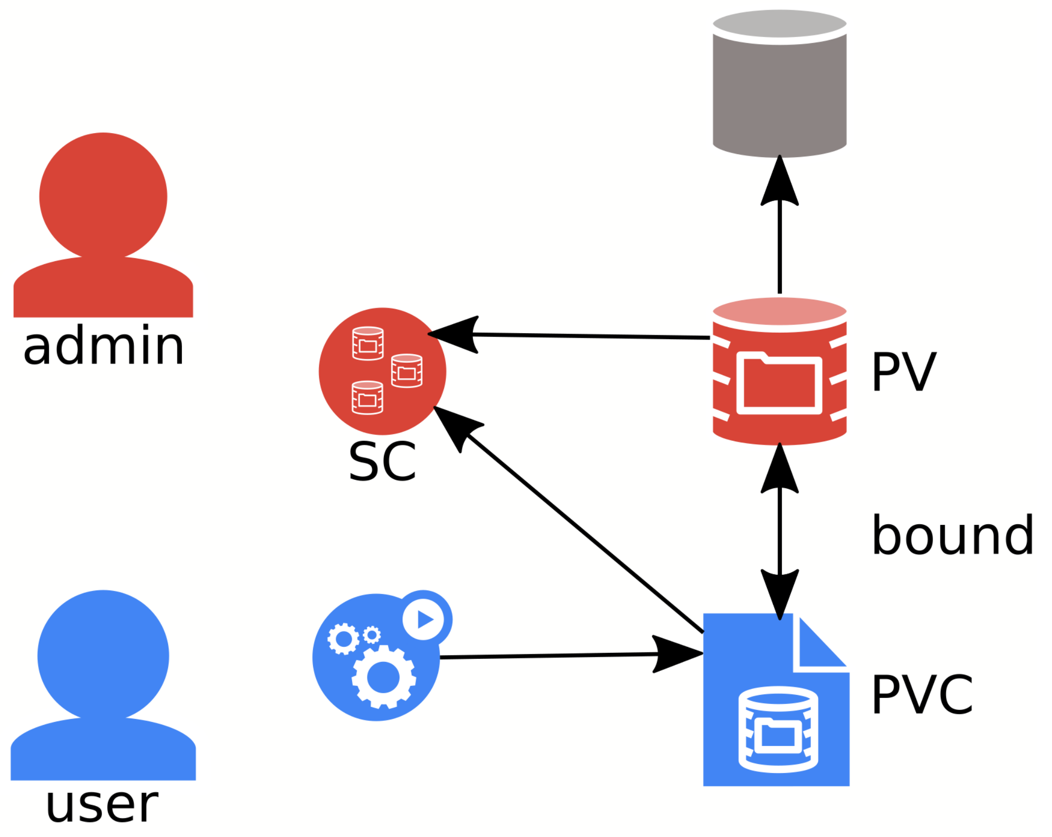
PersistentVolume: Dynamic Provisioning



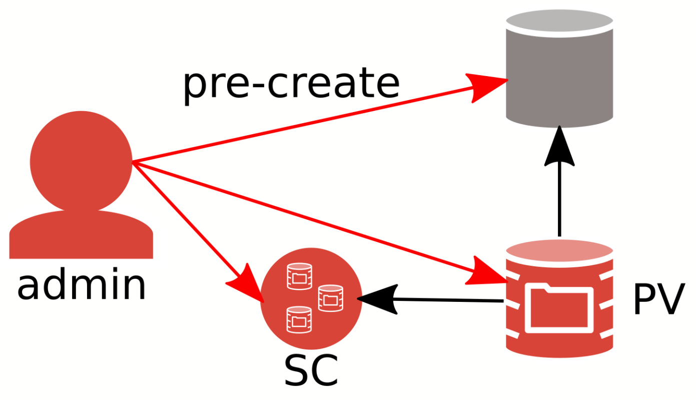
PersistentVolume: Dynamic Provisioning



PersistentVolume: Dynamic Provisioning

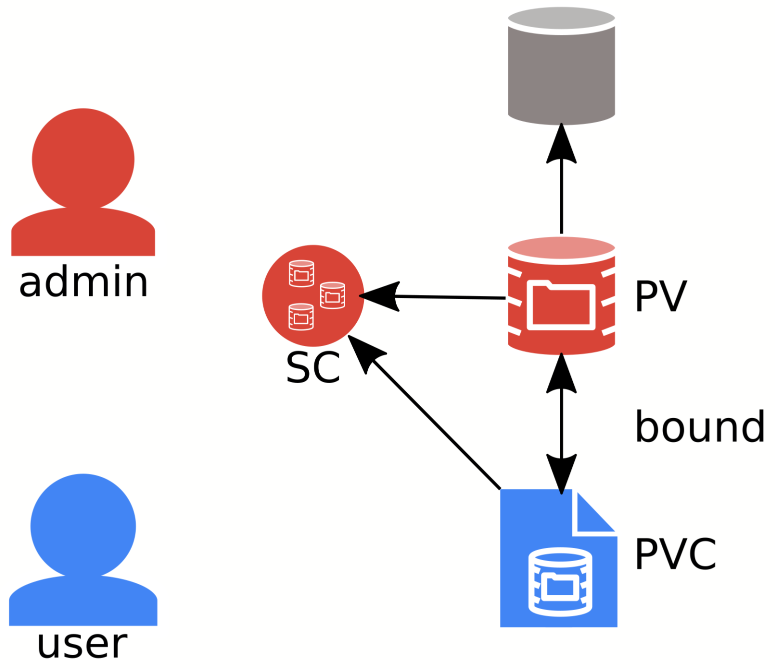


PersistentVolume: Manual Provisioning



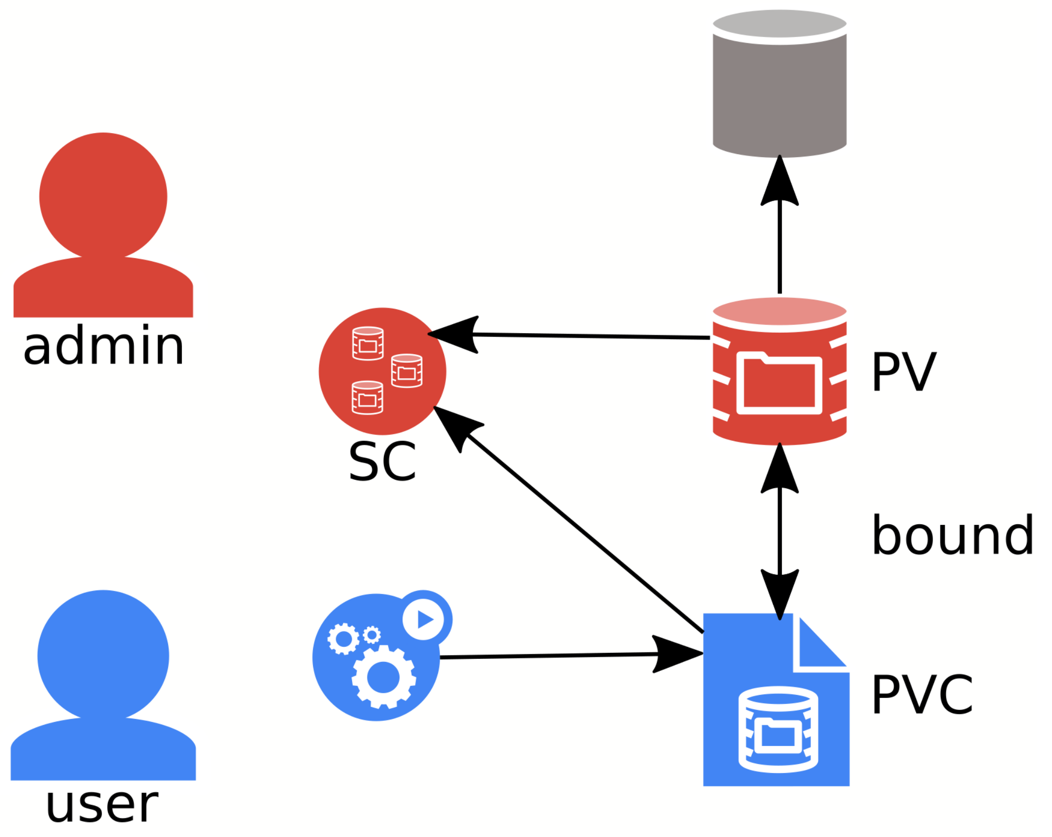
- "Brownfield" use case.
    - Using data of old apps.

PersistentVolume: Manual Provisioning

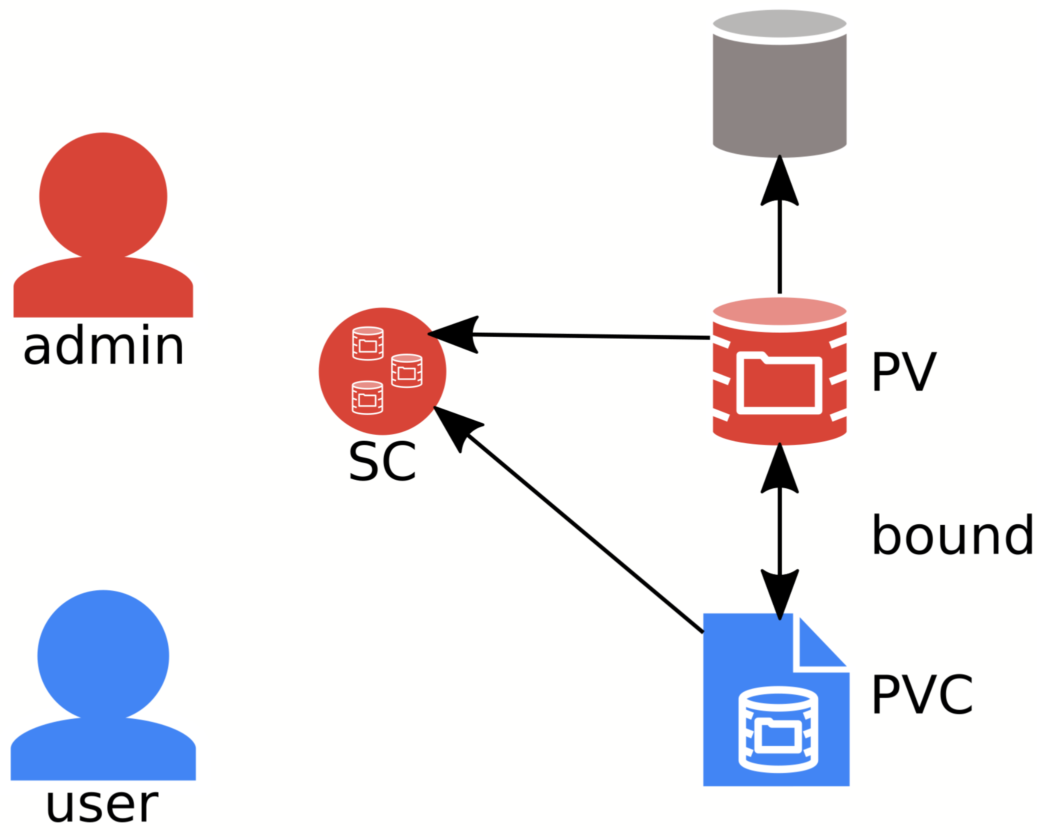


- "Brownfield" use case.
    - Using data of old apps.

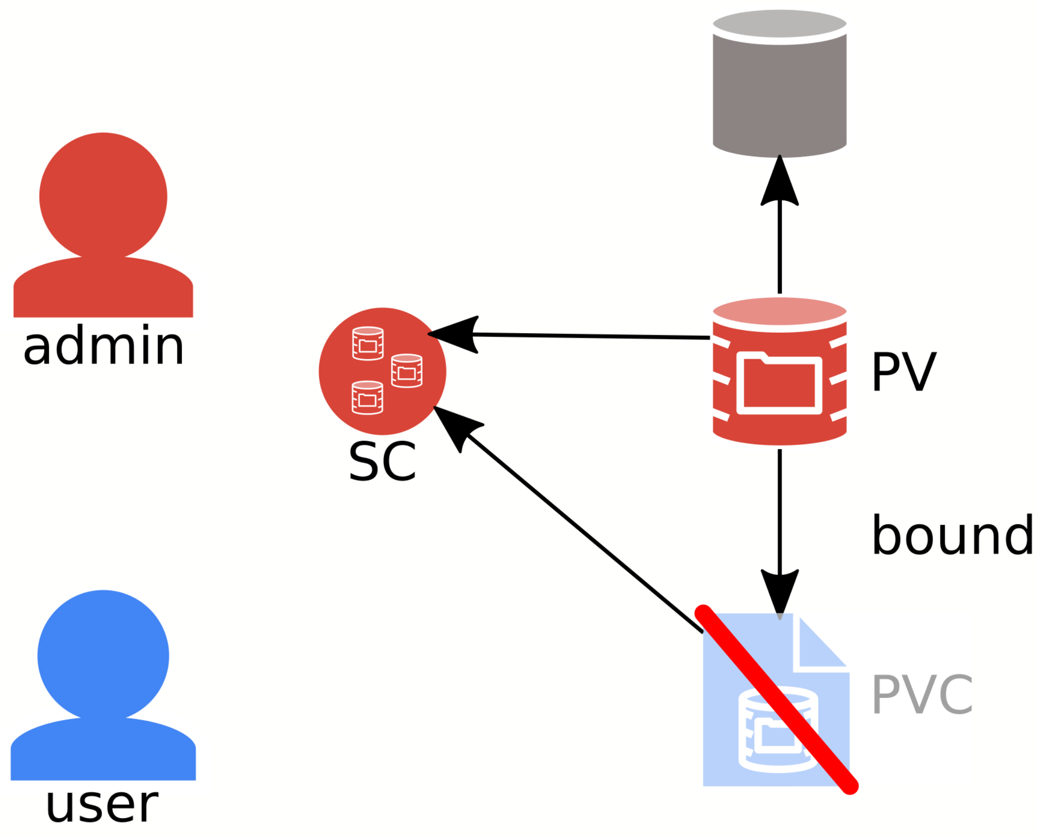
PersistentVolume: Release



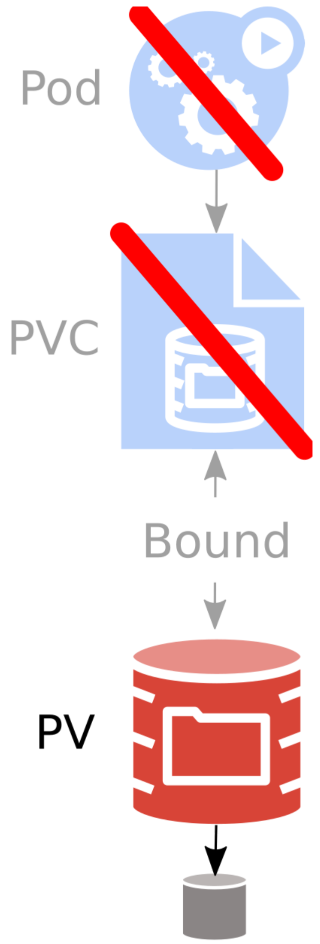
PersistentVolume: Release



PersistentVolume: Release



PersistentVolume: Release

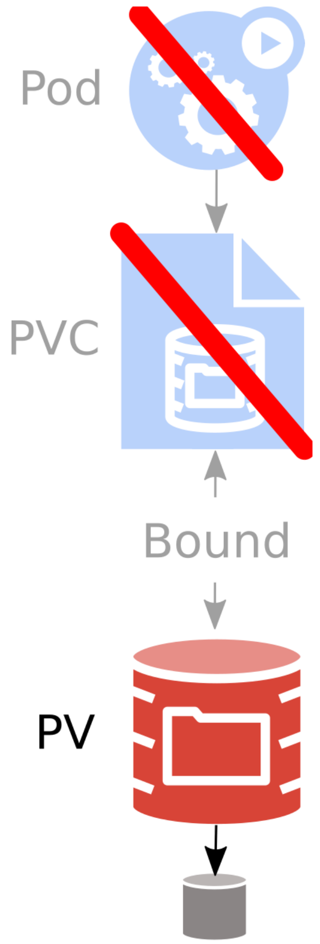


PVC is deleted: `persistentVolumeReclaimPolicy` is executed:

- `Recycle` (deprecated):
  - **All data from the volume are removed** ("`rm -rf *`").
  - PV is `Available` for new PVCs.
- `Delete`:
  - **Volume is deleted in the storage backend.**
  - PV is deleted.
  - Usually for dynamically-provisioned volumes
- `Retain`:
  - PV is kept `Released`.
  - **No PVC can bind to it.**
  - Admin should manually prune `Released` volumes.

In all cases, user can't access the data!

PersistentVolume Life Cycle: Deletion



- Automatic:
  - `persistentVolumeReclaimPolicy = Delete`.
- Manual:
  - PV is not `Bound`.
  - Does not delete volume on storage backend!

Stateful applications

Pods are not for users

- Pod can be deleted.
  - Preemption.
  - Node is drained (for update, ...)
  - Node goes down.

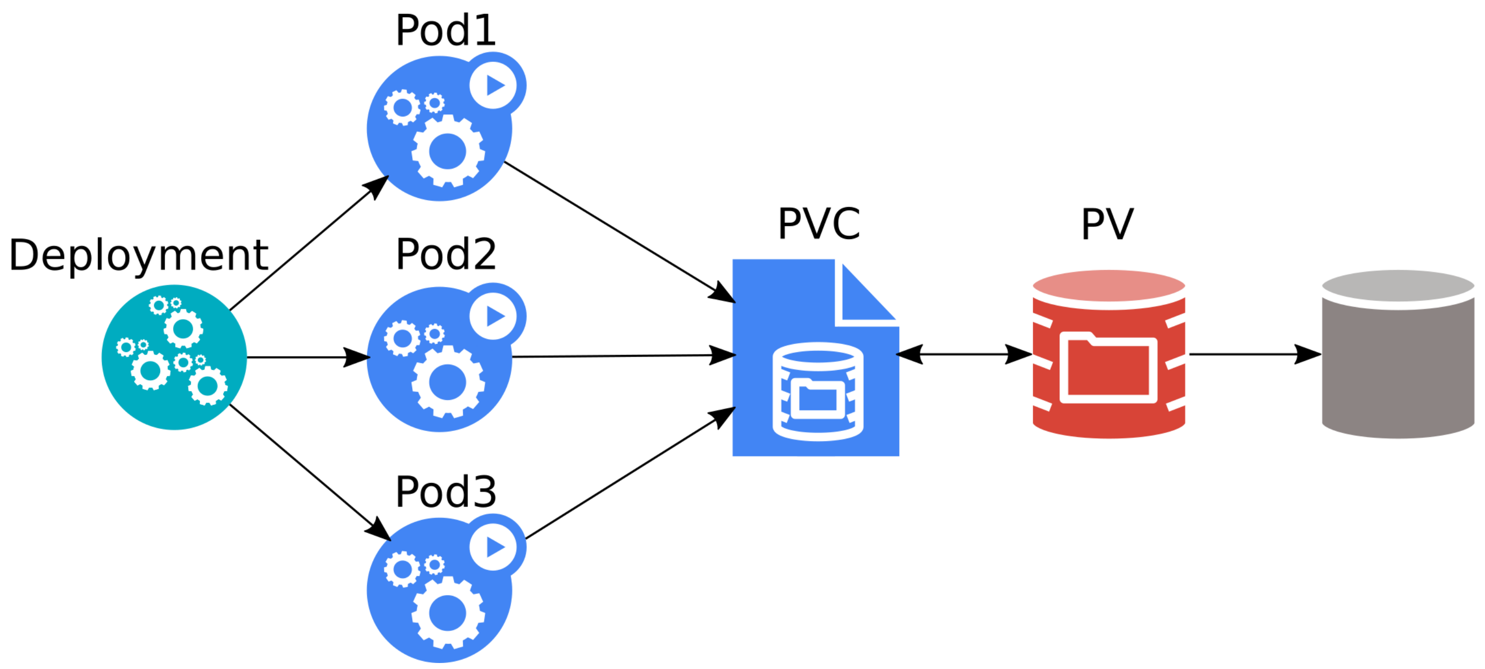
-> Users should not create Pod objects.

Kubernetes high-level objects

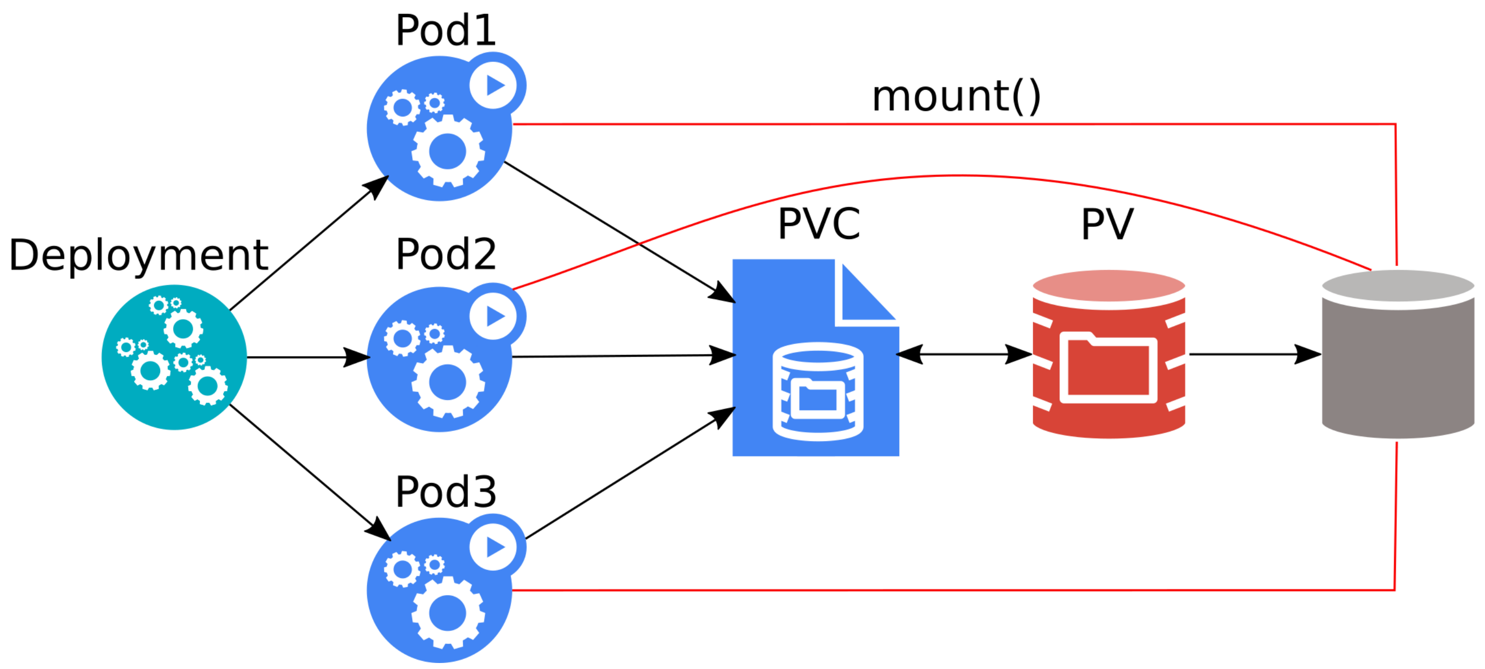
`Deployment`

- Runs X replicas of a single Pod template.
- When a pod is deleted, `Deployment` automatically creates a new one.
- Scalable up & down.
- All pods share the same PVC!

Deployment



Deployment



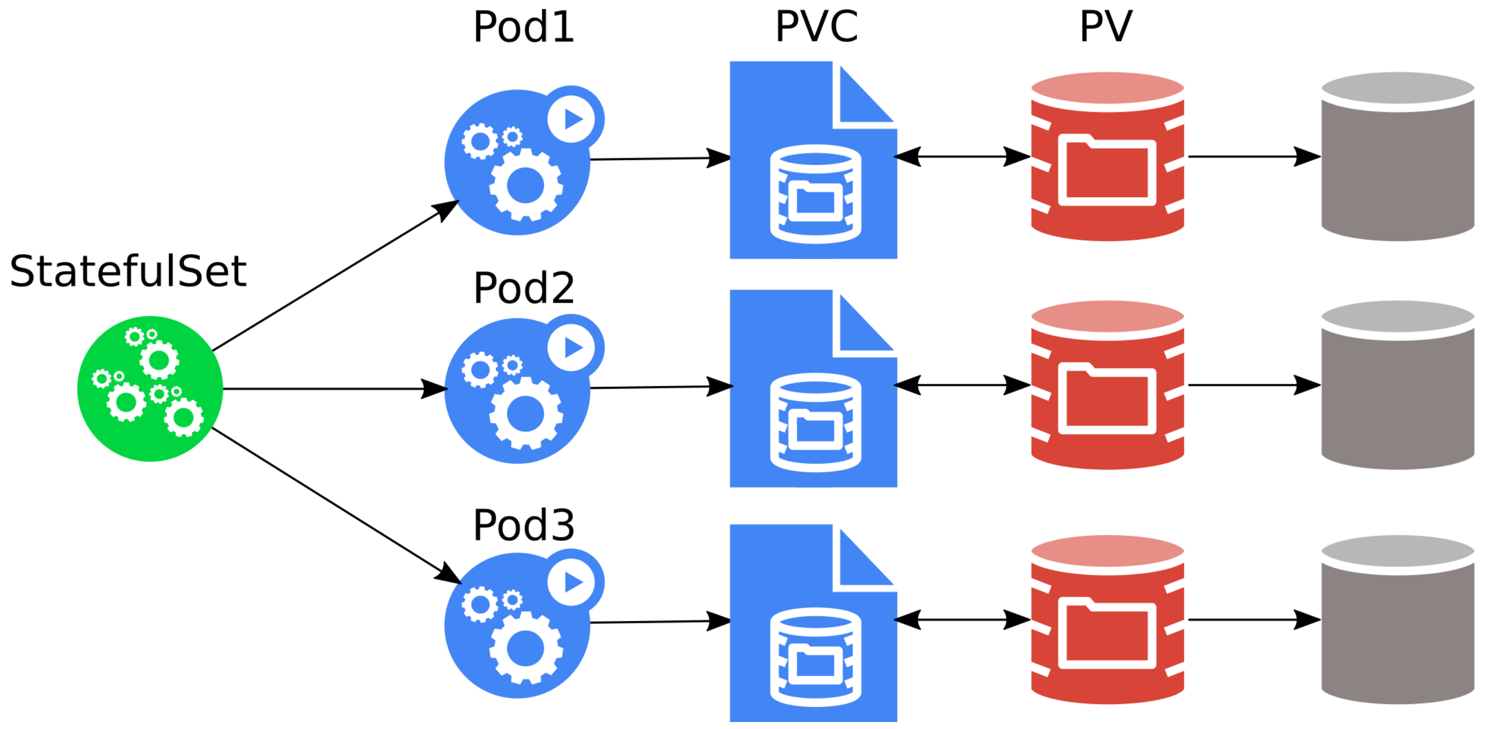
- All three pods can overwrite data of each other!
- Most applications crash / refuse to work.

Kubernetes high-level objects

`StatefulSet`

- Runs X replicas of a single Pod template.
    - Each pod gets its own PVC(s) from a PVC template.
- When a pod is deleted, `StatefulSet` automatically creates a new one.
- Each pod has a stable identity.
- Scalable up & down.

StatefulSet



- The pods must be aware of the other StatefulSet members!
- Usually very complex setup.

Storage features

Topology aware scheduling

- PV can be usable only by subset of nodes.
    - Cloud *regions* / *availability zones*.
    - Bare metal datacenters.
    - …
- Pod must be scheduled:
    - Where the PV is reachable.
    - Where is enough resources to run the pod (CPU, memory, GPU, …)

PV provisioning is delayed until Pod is created for scheduler to pick a node that matches both PV & Pod.

Topology aware scheduling: Delayed binding

PV provisioning is delayed until Pod is created for scheduler to pick a node that matches both PV & Pod.

```
$ kubectl get pvc
NAME              STATUS
my-delayed-claim  Pending

$ kubectl describe pvc
...
Events:
  Type    Reason                 Age   From                          Message
  ----    ------                 ----  ----                          -------
  Normal  WaitForFirstConsumer   9s    persistentvolume-controller   waiting for first consumer to
be created before binding
```

Topology aware scheduling: Delayed binding

PV provisioning is delayed until Pod is created for scheduler to pick a node that matches both PV & Pod.

```
$ kubectl get pvc
NAME              STATUS
my-delayed-claim  Pending

$ kubectl describe pvc
...
Events:
  Type    Reason                Age   From                          Message
  ----    ------                ----  ----                          -------
  Normal  WaitForFirstConsumer  9s    persistentvolume-controller   waiting for first consumer to
be created before binding
```

```
$ kubectl create -f pod.yaml
pod/mysql created
```

Topology aware scheduling: Delayed binding

PV provisioning is delayed until Pod is created for scheduler to pick a node that matches both PV & Pod.

```
$ kubectl get pvc
NAME              STATUS
my-delayed-claim  Pending

$ kubectl describe pvc
...
Events:
  Type    Reason                Age   From                         Message
  ----    ------                ----  ----                         -------
  Normal  WaitForFirstConsumer  9s    persistentvolume-controller  waiting for first consumer to
be created before binding
```

```
$ kubectl create -f pod.yaml
pod/mysql created
```

```
$ kubectl get pvc
NAME              STATUS
my-delayed-claim  Bound
```

Wednesday, Hall 8.0 D2, 15:55: Improving Availability for Stateful Applications in Kubernetes - Michelle Au

Local volumes

- Unused local disks can be used as PVs.
    - Extra speed.
    - Lower reliability.
    - No pod scheduling flexibility.

Raw block

- Pods can get a block device of a PV.
  - For extra speed.
  - For software defined storage.

Resize

- Only expansion is supported.
- Offline.
- Online (alpha).

[Container Storage Interface (CSI)]()

Industry standard that will enable storage vendors (SP) to develop a plugin once and have it work across a number of container orchestration (CO) systems.

- No change from user perspective, Pods & PVCs as usual.

- Extra work for cluster admin.
  - New Kubernetes external components:
    - `external-attacher`
    - `external-provisioner`
    - `node-driver-registrar`
    - `cluster-driver-registrar`
    - `external-resizer`
    - `external-snapshotter`
    - …

Snapshots

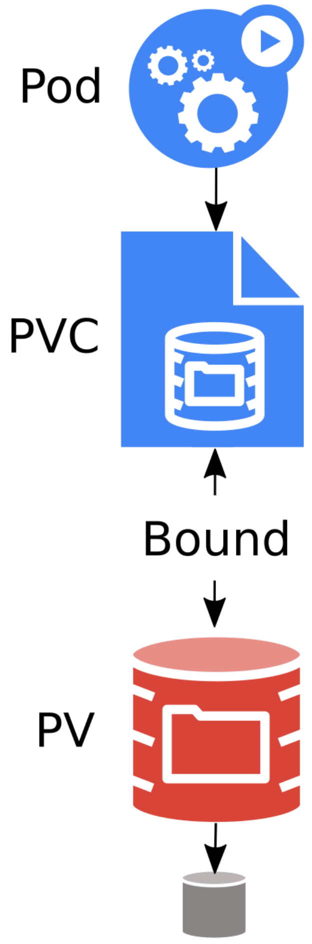
- Alpha.
- Part of CSI.
- Can take a snapshot of PVC.
- PVC can be provisioned from a snapshot.

CSI migration

- Alpha.
- Remove in-tree volume plugins.
- Translate to CSI seamlessly.

Summary

Persistent Storage objects



Pod

- Mounts `PersistentVolumeClaim` into container(s).

PersistentVolumeClaim (PVC)

- Application request for storage.
- Created by user / devops.

PersistentVolume (PV)

- Pointer to physical storage.
- Created by Kubernetes on demand ("dynamic provisioning").

StorageClass

- Collection of PersistentVolumes with the same characteristics.
- Parameters for dynamic provisioning.

It's not all!

Kubecon EU 2019

- David Zhu, Google & Jan Šafránek: Tutorial: Back to Basics: Hands-On Deployment of Stateful Workloads on Kubernetes, Tue 11:05
- Josh Berkus: Benchmarking Cloud Native Storage, Tue 11:55
- Saad Ali: Debunking the Myth: Kubernetes Storage is Hard (keynote), Wed 9:58
- Jared Watts: Data Without Borders - Using Rook Storage Orchestration at a Global Scale, Wed 11:05
- Jared Watts & Bassam Tabbara: Deep Dive: Rook, Wed 11:55
- Iqbal Farabi & Tara Baskara: Benchmarking Cloud Native Databases Performance on Kubernetes, Wed 11:55
- Sheng Yang: Build a Kubernetes Based Cloud Native Storage Solution From Scratch, Wed 12:30
- Federico Lucifredi & Sébastien Han: Rook, Ceph, and ARM: A Caffeinated Tutorial, Wed 16:45
- Michelle Au: Improving Availability for Stateful Applications in Kubernetes, Wed 15:55
- Saad Ali: Intro + Deep Dive: Kubernetes Storage SIG, Thu 11:05

Reach out

Kubernetes SIG Storage

- Bi-weekly meetings
- Slack
- Mailing list

Questions?