Pods-as-Volumes: Effortlessly Integrating Storage Systems and Middleware into Kubernetes

Alberto Faria
INESC TEC & University of Minho

Ricardo Macedo
INESC TEC & University of Minho

João Paulo
INESC TEC & University of Minho

Kubernetes

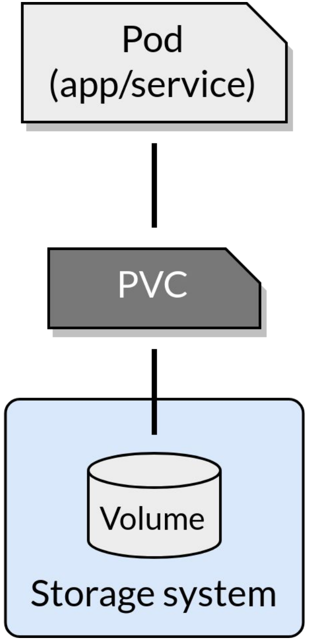
- A **container orchestration** system
- **Automates** many aspects of app deployment
  - **Scheduling** to nodes, **scaling**, handling **failures**, …
- Declarative **objects** specify desired state
  - Kubernetes **autonomously** adjusts actual state
- **Pods** are sets of one or more containers
  - Defined by `Pod` objects
  - Containers of a pod run **together** on a single node

kubernetes

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: container
      image: ubuntu:20.04
      command: [ sleep, infinity ]
```

Kubernetes: Volumes

- Can manage **storage** resources through the ***volume*** abstraction
  - Each volume corresponds to a **file system** or **block device**
- User **creates** a volume by defining a **`PersistentVolumeClaim`** **(PVC)**
  - Describes **desired properties** of the volume (*e.g.*, storage capacity)
- **PVC** objects also (indirectly) specify a ***volume provisioner***
  - Provisioner implements all volume **(de)allocation** logic
  - Interacts with underlying **storage system**
- Pod **containers** can access a volume by **mounting** its PVC
  - Provisioner **coordinates** with storage system to **expose** volume to containers



Kubernetes: Container Storage Interface (CSI)

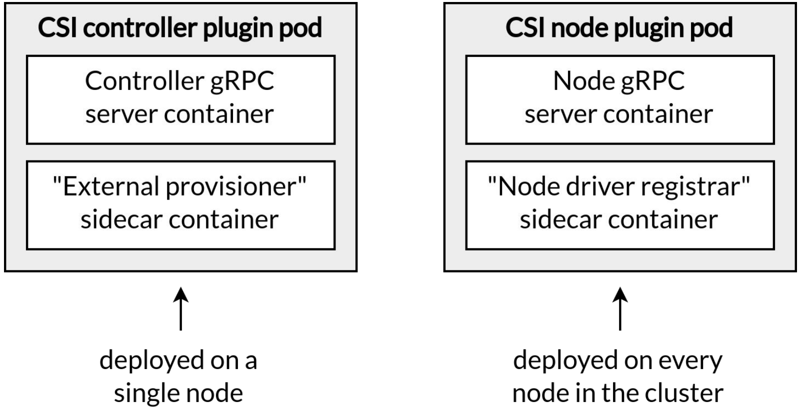
- Kubernetes includes several **built-in provisioners**
  - **CephFS** provisioner, **NFS** provisioner, **iSCSI** provisioner, ...
- Can build provisioners by implementing the **Container Storage Interface (CSI)**
  - **gRPC** interface specification: https://github.com/container-storage-interface/spec
  - Standardizes interaction between **container orchestration** systems and **storage** systems

CONTAINER
STORAGE
INTERFACE

A problem: Implementing CSI takes effort

- Implementing **CSI** fully and correctly can be **difficult** and **time-consuming**
  - Must typically build **2** gRPC servers
  - Correct **error handling** and **cleanup** is tricky

- Must deploy components across **all nodes** of the cluster
  - **CSI controller plugin** pod: gRPC server container, "external provisioner" sidecar container
  - **CSI node plugin** pod: gRPC server container, "node driver registrar" sidecar container



Another problem: Kubernetes storage is inflexible

- **Limited** ability to manage storage **stacks**
- Can only **represent volumes** and attach them **directly to pods**
  - Can't define intermediate **layers** to create more complex **stacks**
- Leads to **lack of modularity** and **reimplementation** of features
- *E.g.*, no **general** mechanism to enable volume **encryption**
  - Each storage system/provisioner must implement **their own**
  - Or each client pod must **embed** encryption logic
- Problem is exacerbated when **combining** several storage **functionalities**
  - *E.g.*, applying **compression** *and* **encryption**

A solution: Pods-as-Volumes (PaV)

- Kubernetes plugin that **simplifies** the implementation of new **volume provisioners**
- Allows specifying **logic** for creating, deleting, exposing volumes as **pod templates**
  - Pods automatically **instantiated** from templates when needed
- Easier **integration** of **storage systems** into Kubernetes
- Enables straightforward creation of **storage middleware** components
- **Open source**: https://github.com/albertofaria/pav

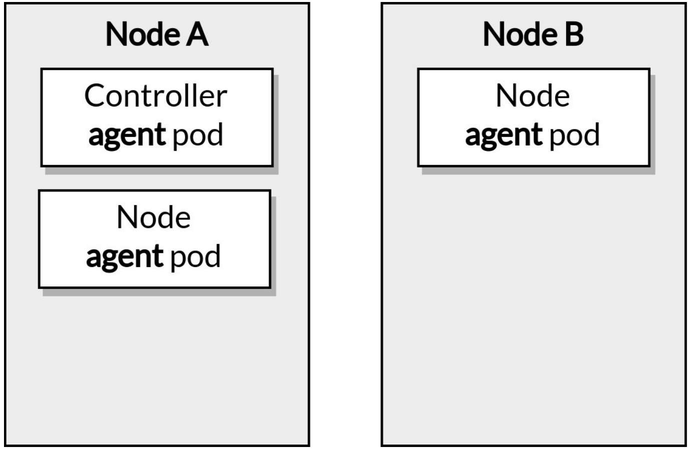
`PavProvisioner` objects

- New object kind: **`PavProvisioner`**
- Each `PavProvisioner` object implements a **volume provisioner**
- Pod templates **embedded** in object definition
- Can use `PavProvisioner` in the **same manner** as built-in provisioners

```yaml
apiVersion: pav.albertofaria.github.io/v1alpha1
kind: PavProvisioner
metadata: …
spec:
  provisioningModes: …
  volumeValidation:
    volumeModes: …
    accessModes: …
    minCapacity: …
    maxCapacity: …
    podTemplate: …
  volumeCreation:
    volumeHandle: …
    capacity: …
    podTemplate: …
  volumeDeletion:
    podTemplate: …
  volumeStaging:
    podTemplate: …
  volumeUnstaging:
    podTemplate: …
```

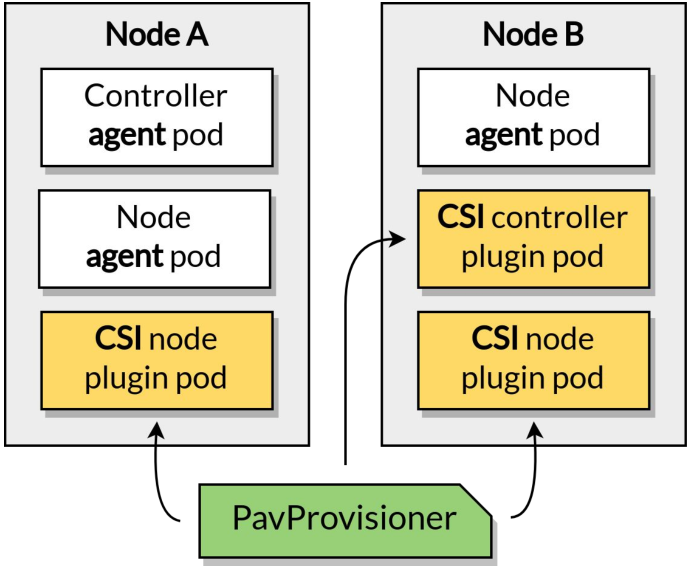
Architecture

- **Two** main components: **agent** and **CSI plugin**
- The **agent** is deployed once per Kubernetes **cluster**
  - **Controller agent**, deployed as a single pod for the entire cluster
  - **Node agent**, deployed on all nodes



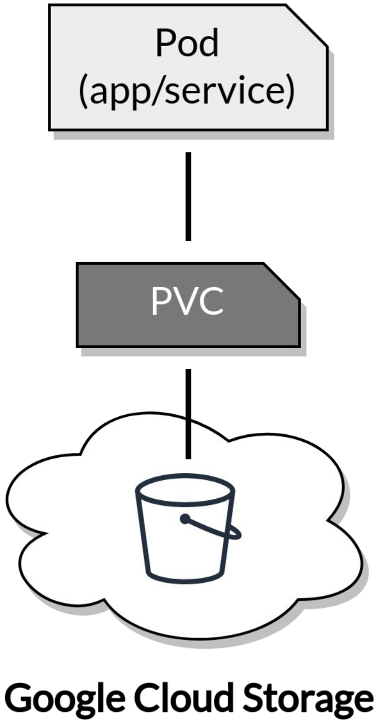
Architecture (cont.)

- For each `PavProvisioner`, a **CSI plugin** instance is deployed
  - **CSI controller plugin**, deployed as a single pod for the entire cluster
  - **CSI node plugin**, deployed on all nodes
- Kubernetes **interacts** with CSI plugins
  - CSI plugins **delegate** most work to the agent



Use cases: Google Cloud Storage integration

- Google's **object storage** cloud service
- Users create **buckets**, which contain **objects**
  - **Libraries** in many languages to access buckets
  - Buckets can also be **mounted** locally as a **file system**
- No **built-in** Kubernetes integration
- We create one with **PaV**
  - Each **volume** is a **bucket**
  - Bucket is **mounted** into client pods as a **file system**



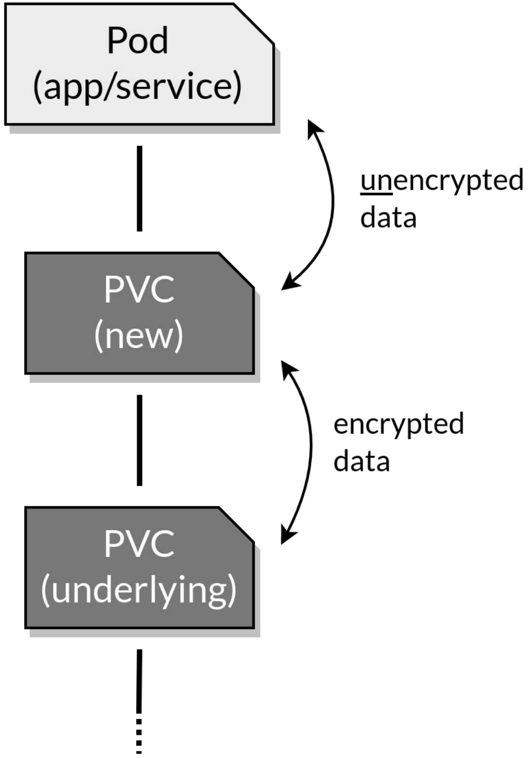
Use cases: Google Cloud Storage integration (cont.)

- **62 lines** of YAML
- **No coding** required
- Relies on existing ***gsutil*** and ***gcsfuse*** tools

```yaml
apiVersion: pav.albertofaria.github.io/v1alpha1
kind: PavProvisioner
metadata: { name: gcs-provisioner }
spec:
  provisioningModes: [ Dynamic, Static ]
  volumeCreation:
    capacity: 1Ei
    podTemplate:
      metadata: { namespace: "{{params.secretNamespace}}" }
      spec: &gsutil-pod-spec
        restartPolicy: Never
        containers:
          - &gsutil-container
            name: gsutil
            image: albertofaria/gsutil:5.2
            command: [ gsutil, -o,
              Credentials:gs_service_key_file=/secret/key,
              -o, "GSUtil:default_project_id={{
              params.projectId }}" ]
            args: [ mb, -b, "on",
              -l, "{{ params.location or 'US' }}",
              "gs://{{ defaultVolumeHandle }}" ]
            volumeMounts:
              - { name: secret, mountPath: /secret }
        volumes:
          - &secret-volume
            name: secret
            secret: { secretName: "{{params.secretName}}" }
  volumeDeletion:
    podTemplate:
      metadata: { namespace: "{{params.secretNamespace}}" }
      spec:
        <<: *gsutil-pod-spec
        containers:
          - <<: *gsutil-container
            args: [ rm, -r, "gs://{{ volumeHandle }}" ]
  volumeStaging:
    podTemplate:
      metadata: { namespace: "{{params.secretNamespace}}" }
      spec:
        restartPolicy: Never
        containers:
          - name: gcsfuse
            image: albertofaria/gcsfuse:0.36.0
            command: [ /bin/bash, -c ]
            args:
              - |
                mkdir /pav/volume &&
                gcsfuse -o=allow_other \
                  --key-file=/secret/key --dir-mode=777 \
                  --temp-dir=/temp --file-mode=666 \
                  --stat-cache-ttl=0 --type-cache-ttl=0 \
                  {{ volumeHandle|tobash }} /pav/volume &&
                touch /pav/ready &&
                sleep infinity
            securityContext: { privileged: true }
            volumeMounts:
              - { name: secret, mountPath: /secret }
              - { name: temp, mountPath: /temp }
        volumes:
          - *secret-volume
          - { name: temp, emptyDir: {} }
```

Use cases: Transparent encryption middleware

- No **general** mechanism for **volume encryption** in Kubernetes
  - Each storage system/provisioner must implement **their own**
  - Or each client pod must **embed** encryption logic
- We create an **encryption middleware** with PaV
  - **Transparent encryption** for block volumes
- User creates **new PVC** referencing **underlying PVC**
  - Client pods mount **new PVC**



Use cases: Transparent encryption middleware (cont.)

- **70 lines** of YAML
- **No coding** required
- Relies on existing ***cryptsetup*** utility
- Pod templates **mount** the **underlying PVC**

```yaml
apiVersion: pav.albertofaria.github.io/v1alpha1
kind: PavProvisioner
metadata: { name: crypt-provisioner }
spec:
  provisioningModes: [ Dynamic ]
  volumeValidation: { volumeModes: [ Block ] }
  volumeCreation:
    podTemplate:
      metadata: { namespace: "{{pvc.metadata.namespace}}" }
      spec: &cryptsetup-pod-spec
        restartPolicy: Never
        containers:
          - &cryptsetup-container
            name: cryptsetup
            image: albertofaria/cryptsetup:2.4.1
            command: [ /bin/bash, -c ]
            args:
              - |
                set -o errexit -o pipefail
                cryptsetup -q luksFormat \
                  /volume /secret/passphrase
                size="$( blockdev --getsize64 /volume )"
                offset="$( cryptsetup luksDump \
                  /volume --dump-json-metadata |
                  jq '.segments."0".offset' | tr -d '"' )"
                echo "$(( size - offset ))" > /pav/capacity
            securityContext: { privileged: true }
            volumeMounts:
              - { name: secret, mountPath: /secret }
            volumeDevices:
              - { name: underlying, mountPath: /volume }
        volumes:
          - name: secret
            secret:
              secretName: "{{ pvc.metadata.annotations[
                'crypt/secretName'] }}"
          - name: underlying
            persistentVolumeClaim:
              claimName: "{{ pvc.metadata.annotations[
                'crypt/underlyingClaimName'] }}"
  volumeDeletion:
    podTemplate:
      metadata: { namespace: "{{pvc.metadata.namespace}}" }
      spec:
        <<: *cryptsetup-pod-spec
        containers:
          - <<: *cryptsetup-container
            args: [ cryptsetup -q erase /volume ]
  volumeStaging:
    podTemplate:
      metadata: { namespace: "{{pvc.metadata.namespace}}" }
      spec:
        <<: *cryptsetup-pod-spec
        containers:
          - <<: *cryptsetup-container
            args:
              - |
                dev={{ volumeHandle|tobash }} &&
                cryptsetup open /volume "$dev" \
                  --key-file /secret/passphrase &&
                cp -p "/dev/mapper/$dev" /pav/volume
  volumeUnstaging:
    podTemplate:
      metadata: { namespace: "{{pvc.metadata.namespace}}" }
      spec:
        <<: *cryptsetup-pod-spec
        containers:
          - <<: *cryptsetup-container
            args: [ "cryptsetup close
              {{ volumeHandle|tobash }} || (( $? == 4 ))" ]
```

Summary

- Implementing new provisioners using **CSI is difficult** and **time-consuming**
- Kubernetes' ability to manage storage **stacks** is **limited**
  - Insufficient **expressiveness** to represent and manage storage **layers**
- We propose **Pods-as-Volumes (PaV)**
  - **Easily** implement new volume **provisioners**
  - Specify **logic** underlying volume lifecycle and behavior as **pod templates**
- Simplifies **integration** of **storage systems** into Kubernetes
- Enables straightforward creation of **composable** storage **middleware** components

PaV is open source!

https://github.com/albertofaria/pav