Deploying a First (Stateless) Application

INTRODUCTION TO KUBERNETES

Frank Heilmann

Platform Architect and Freelance Instructor

More on "kubectl"

- `kubectl` : main command to interact with Kubernetes objects
- Objects are, e.g., `pod` , `service` , etc.

- Typical usage patterns:
  - `kubectl create -f <Manifest.yml>` : create new objects, with `-f` for "filename"
  - `kubectl apply -f <Manifest.yml>` : create new objects & change the state of objects
  - `kubectl get <object>` : overview about objects deployed on Kubernetes
  - `kubectl describe <object>` : detailed information about an object

- Detailed help available via command line option `--help`

More on Manifests

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.25.4
        ports:
        - containerPort: 80
```

- Remember: Manifests are **declarative**
- Typically YAML, but also in JSON format
- Two important sections:
  - **metadata:** essential information about the object or resource
  - **spec:** defines the specifications, or desired state, of the object or resource
- Sections can be quite deep, depending on the resource to be deployed

Stateless Applications

- Stateless apps:
  - General concept
  - Not specific to Kubernetes
  - Do not save an internal state, or context of processed data
- When interrupted, a new replica of the stateless app is recreated and starts operating.

- Examples:
  - The database frontend querying a database backend
  - A search app querying a full text index
  - A data stream app that converts temperature readings from an IoT sensor from °F to °C

Kubernetes Deployments

- "Stateless applications" translate to "Kubernetes Deployments"
- A sample Manifest consists of:
  - `apiVersion` and `kind`
  - `metadata` and `spec`
- `spec` defines number of `replicas`, a `selector`, and a `template`
- More on `selector` later
- `template` describes details for the creation the pods in the Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: <deployment name>
  labels:
    app: <a label for the application>
spec:
  replicas: <number of initial replicas>
  selector:
    matchLabels:
      app: <matches the label above>
  template:
    metadata:
      labels:
        app: <label to be given to each pod>
    spec:
      containers:
      - name: <container name>
        image: <the image to be used>
        ports:
        - containerPort: <ports for networking>
```

Deploying to a Kubernetes Cluster

- `kubectl apply -f <manifest.yml>` for creating pods and applying changes.
- Kubernetes Control Plane will schedule the Deployment on Nodes.
  - Then, Pods created is triggered on the Nodes.
- Pods get a unique, but random (unpredictable) identifier, each Pod is "as good as any other"

Let's practice!

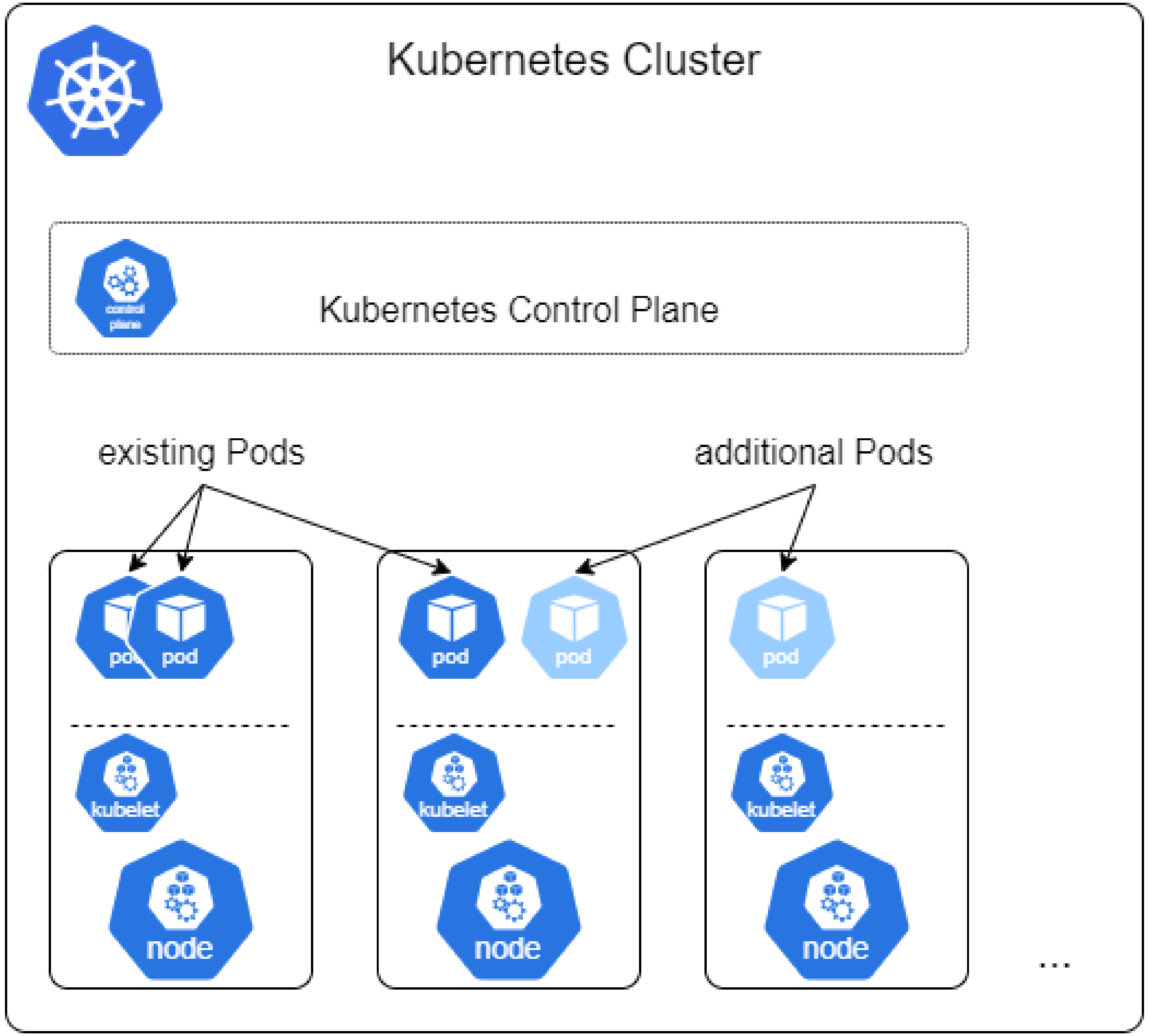
INTRODUCTION TO KUBERNETES

Scaling and Monitoring an Application

INTRODUCTION TO KUBERNETES

Frank Heilmann
Platform Architect and Freelance Instructor

Scaling on Kubernetes



- Scaling is a technique to add (scale up) or remove (scale down) resources:
  - Scale up: react to increasing load
  - Scale down: save resources
- Scaling the number of Pods is easy:
  - Either change the number of `replicas` in the Manifest and re-apply,
  - Or use the command `kubectl scale deployment ...`
  - with `--replicas <number>`

Scalability and Cloud Nativeness

- An application needs to be designed for scalability
- Legacy applications, in particular monoliths, are typically not scalable in the way shown here
- Modern, cloud native applications are designed with the the goal to be easily scalable

Monitoring an Application

- Monitoring: observing applications in real-time
  - Enables reaction to all kind of problems
- Examples of modern monitoring application for Kubernetes:
  - Prometheus, Grafana, or `kubectl`
- Here, we use `kubectl` for basic monitoring tasks

- Typical command: `kubectl get <object to be monitored>`
- **Example 1:** `kubectl get pods` returns all pods
- **Example 2:** `kubectl get services` returns all services

Let's practice!

INTRODUCTION TO KUBERNETES

Deploying, Scaling, and Monitoring a Stateful Application

INTRODUCTION TO KUBERNETES

Frank Heilmann
Platform Architect and Freelance Instructor

Recap Stateless Applications

- Short recap: stateless applications map to "Deployments" in Kubernetes
- Used when each Pod of the applications has exactly the same tasks
- Stateful applications need Pods that belong together in set, but may work on different tasks and different data
- Much of what we have learned about Deployments can be applied to StatefulSets as well

Stateful Applications

- Stateful apps:
  - general concept
  - fit well to Kubernetes
  - save some state
- When interrupted or stopped, a new replica (Pod) can read the saved state and continue operating from this state

- Example:
  - A database backend (e.g. PostgreSQL) delivers data to a frontend using 3 Pods.
  - Each time we update data using any of the Pods, that data needs to be persisted
  - When a Pod terminates, a new one is created and needs to pick up the saved state

Kubernetes StatefulSets

- Stateful applications translate to "Kubernetes StatefulSets"
- A sample manifest consists of the same sections like:
  - `apiVersion` , `kind` , `metadata` , `spec` , `template`
- `replicas` defines the number of Pods in the `StatefulSet`
- More on `selector` later

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: <deployment name>
  labels:
    app: <a label for the application>
spec:
  replicas: <number of initial replicas>
  selector:
    matchLabels:
      app: <matches the label above>
  template:
    metadata:
      labels:
        app: <label to be given to each pod>
    spec:
      containers:
      - name: <container name>
        image: <the image to be used>
        ports:
        - containerPort: <ports for networking>
```

Deploying to a Kubernetes Cluster

- `StatefulSet` is deployed similar to `Deployments` : `kubectl apply -f <manifest.yml>`
- Once deployed, a `StatefulSet` is created different than a `Deployment` :
  - Pods are created one after the other, not all at once like Pods in a Deployment
  - Pods get predictable names like `pod-0` , `pod-1` , `pod-2` . etc.
- This means: in contrast to the Pods of a Deployment, the Pods of a StatefulSet have an identity, and a state
- Hence, different Pods of a `StatefulSet` with different identity can perform different roles in an application

Scaling A StatefulSet

- Like Deployments, StatefulSets can be scaled up or scaled down:
  - Either change the number of `replicas` in the Manifest and re-apply,
  - Or use the command `kubectl scale statefulsets ...`
- When scaling up, new Pods will be created one after another:
  - e.g, `pod-0` , `pod-1` , `pod-2` first `pod-3` , then `pod-4` will be added
- When scaling down, Pods created last will be deleted first:
  - e.g, first `pod-4` , then `pod-3`

Monitoring a StatefulSet

- Like in the case of Deployments, Monitoring enables reactions to all kind of problems, like outages, load spikes, or missing storage
- Here, we use `kubectl` for basic monitoring tasks
- Typical command: same like with Deployments
- **Example 1:** `kubectl get pods` returns all pods in a StatefulSet with their current status
- **Example 2:** `kubectl get services` returns all services that a StatefulSet may use

Let's practice!

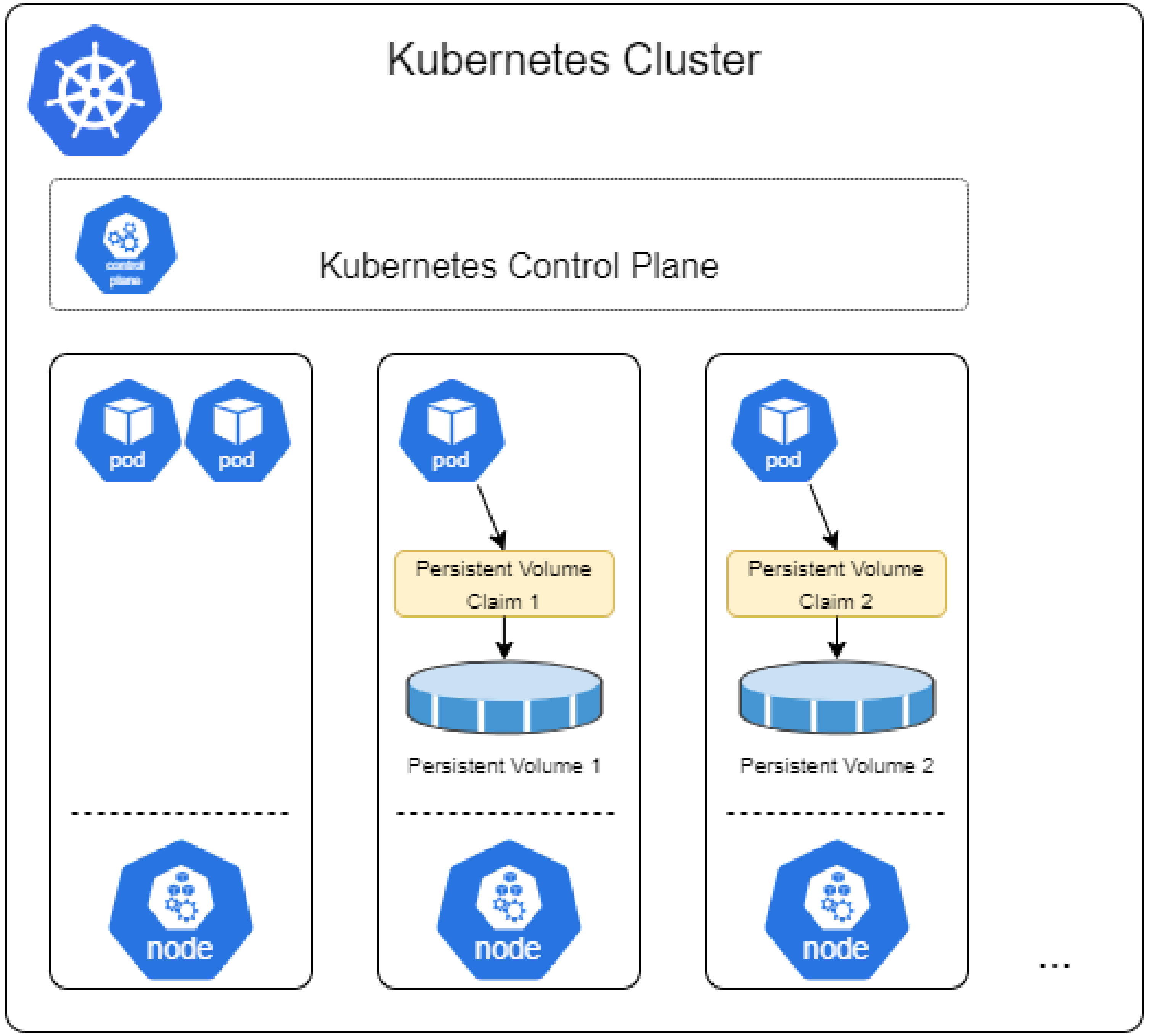
INTRODUCTION TO KUBERNETES

Deploying, Scaling, and Monitoring Kubernetes Storage

INTRODUCTION TO KUBERNETES

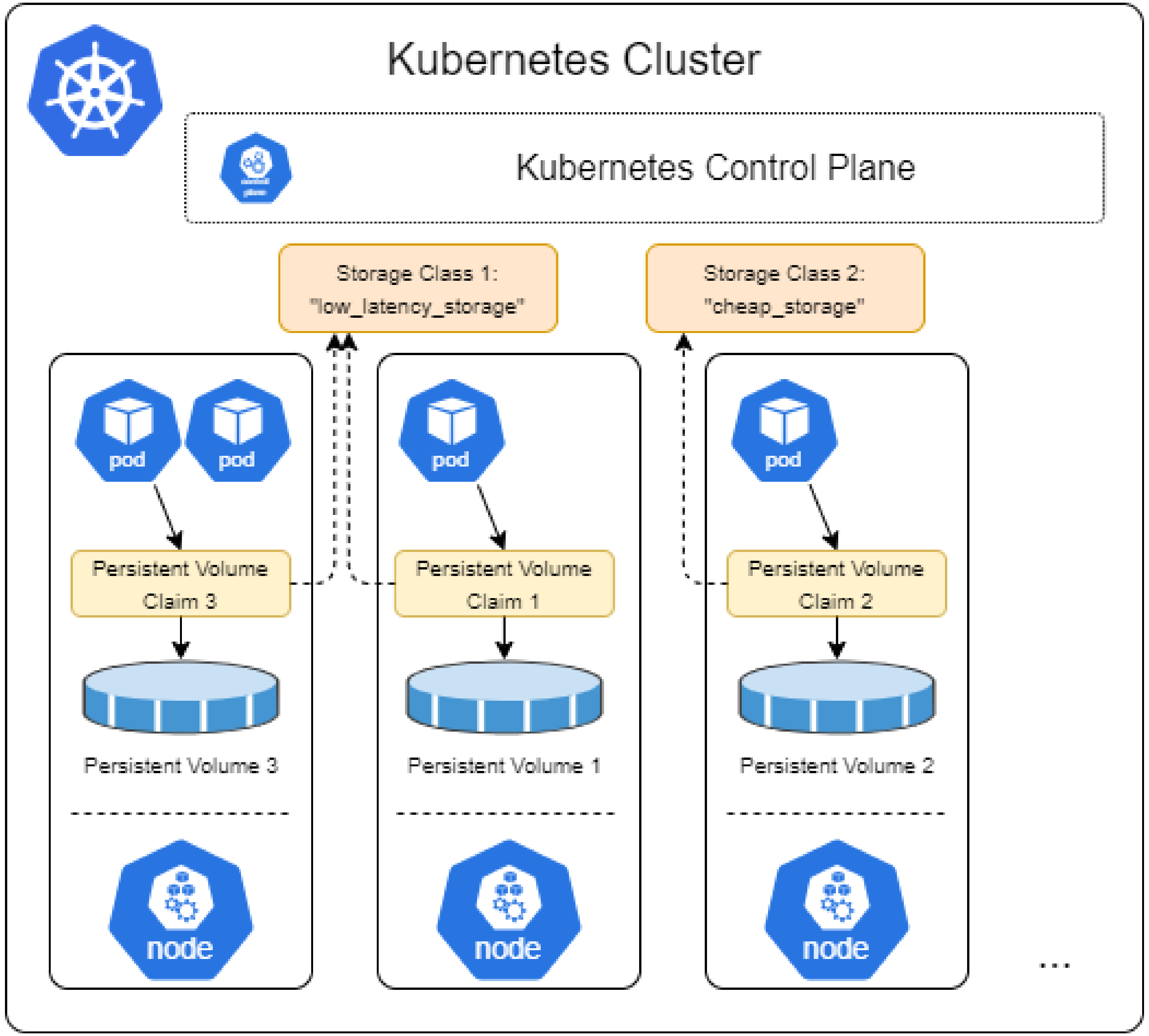
Frank Heilmann
Platform Architect and Freelance Instructor

Persistent Volumes and Persistent Volume Claims



- Fundamental Objects for storage: **Persistent Volumes (PV)**, maintained in parallel to Pods
- PVs are mapped to Pods using **Persistent Volume Claims (PVC)**
- A mapped PV allows data persistence when the Pod is stopped, killed, or restarted
- PVs enable the separation of storage and compute

Storage Classes



- PVs: provisioned either
  - manually by an Kubernetes admin
  - dynamically by regular user
- Dynamic provisioning happens via **Storage Classes (SC)** without human intervention
- **Storage Classes (SC):**
  - defined by Kubernetes admin
  - different types (different latency, e.g., SDD vs HDD, different backup strategies)
- If in doubt, use Storage Classes ;-)

Putting it all together

- There are are only three objects that make storage work:
  - `PersistentVolume`
  - `PersistentVolumeClaim`
  - `StorageClass`
- A Pod with demand for persisted data uses a `PersistentVolumeClaim`
- This PVC has Kubernetes create a `PersistentVolume` for the Pod
- This `PersistentVolume` is mapped to the claiming Pod
- A named `StorageClass` is used, which defines details like latency and backup strategy of the PV
- This `PersistentVolume` survives (together with stored data), even when the `Pod` is terminated

Manifest Snippets

Pod with PersistentVolume

```
apiVersion: v1
kind: Pod
...
spec:
  containers:
  ...
    volumeMounts:
    - name: pv-mydata
      mountPath: /mydata
  volumes:
  - name: pv-mydata
    persistentVolumeClaim:
      claimName: datacamp-pvc
```

PersistentVolumeClaim with StorageClass

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: datacamp-pvc
spec:
  storageClassName: "standard"
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

"kubectl" Commands For Storage

- `kubectl` offers a complete set of commands to create and monitor Kubernetes Storage
- Examples:
  - `kubectl get sc` lists all available Storage Classes
  - `kubectl get pvc` lists all deployed Persistent Volume Claimes
  - `kubectl get pv` lists all deployed Persistent Volumes
  - As usual, `kubectl apply -f <manifest>` can be used to deploy storage resources that are declared in Manifests.

Let's practice!

INTRODUCTION TO KUBERNETES