PERCONA

Databases on Kubernetes with Percona Everest and ArgoCD

Mayank Shah, Percona

Who am I?

- Software Engineer at Percona, working on Percona Everest
- Experience with Kubernetes (operators), Golang, backend systems, DBaaS and SaaS
- Past: Red Hat and ArangoDB

Agenda

Basics of databases on Kubernetes – operators and CRDs

GitOps principles and uses for database management

Percona Everest

Demo

A short history of databases on Kubernetes

Early days – Kubernetes was designed for stateless workloads

Introduction of StatefulSets in 1.5, but running databases was still complex.

PVs and PVCs provided a way to manage storage, but tight coupling with cloud providers.

CSI standardized storage management – dynamic provisioning, snapshots, storage resizing, etc.

Present – Operator pattern, CRDs, declarative systems and GitOps

Imperative v/s Declarative systems

```
kubectl create deployment my-app --image=my-
image
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
```

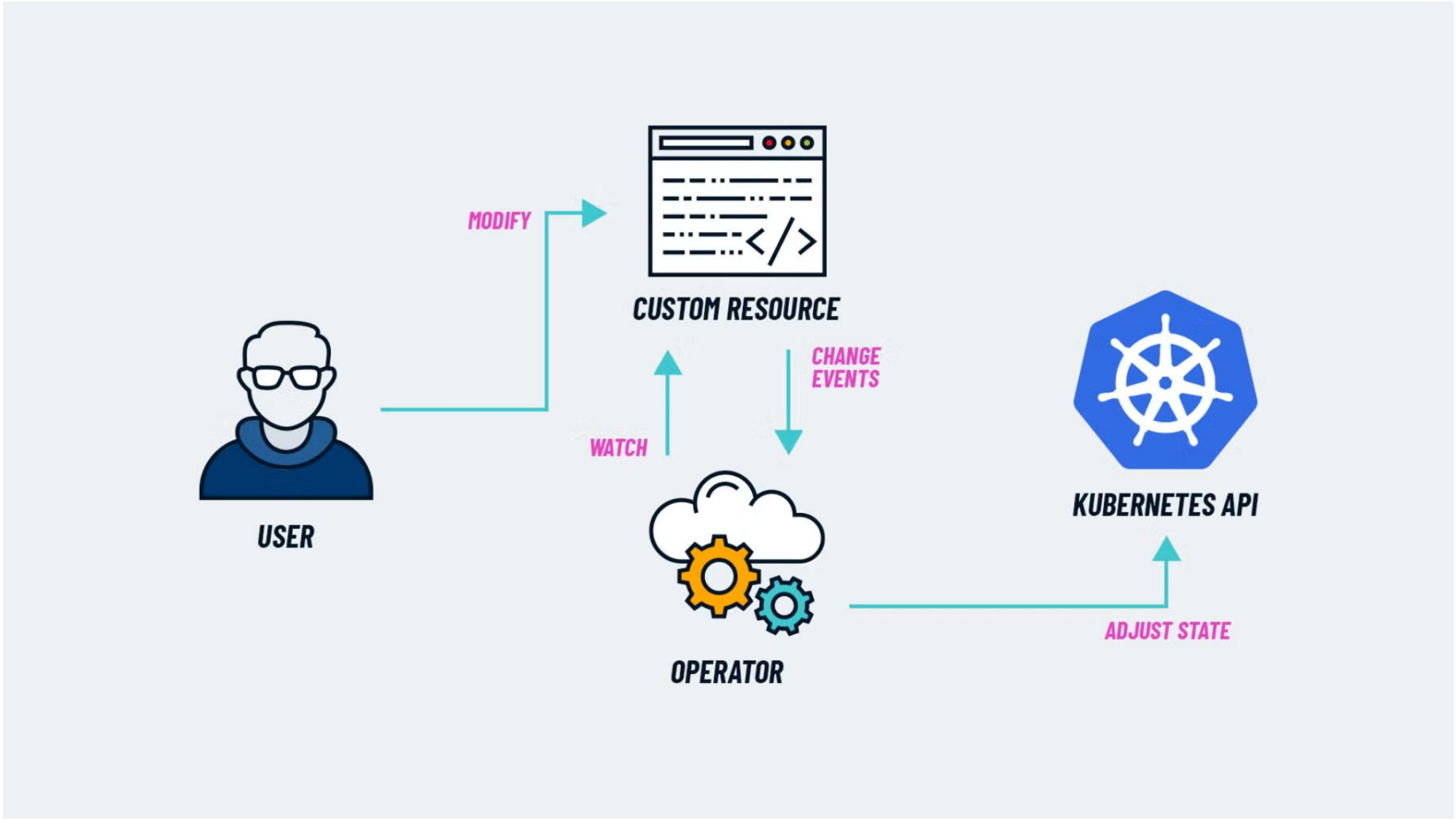
The operator pattern

CRDs:

- Extends Kubernetes beyond the built-in resource types
- Declare the state of some system (e.g, a database cluster)

Controllers:

- A software agent – watches CRDs and actively reconciles the system
- Automates complex management of applications



source - https://www.cncf.io/blog/2022/06/15/kubernetes-operators-what-are-they-some-examples/

Taking it one step further with GitOps

Manage infrastructure and application using Git as the single source of truth.

- Declarative — define desired state in YAML files
- Versioned – all changes are tracked in git
- Automated – Tools like ArgoCD actively reconcile the actual state with the desired (git)

Why GitOps?

Consistency – Ensures the cluster state always matches the Git repository.

Auditability – All changes are tracked via Git commits, providing a clear history.

Automation – Reduces manual intervention and human error.

Collaboration – Enables team collaboration through Git workflows (e.g., pull requests, code reviews).

GitOps for databases

- Database configurations (e.g., schemas, users, backups) are defined as code in Git
- Changes to database deployments are versioned and auditable.
- If a database deployment fails, GitOps tools can automatically revert to the last known good state
- Simplifies managing multiple database instances across clusters
- Secrets and configurations are managed securely through GitOps workflows

What’s missing?

Percona Everest

Provides a unified abstraction on top of database operators

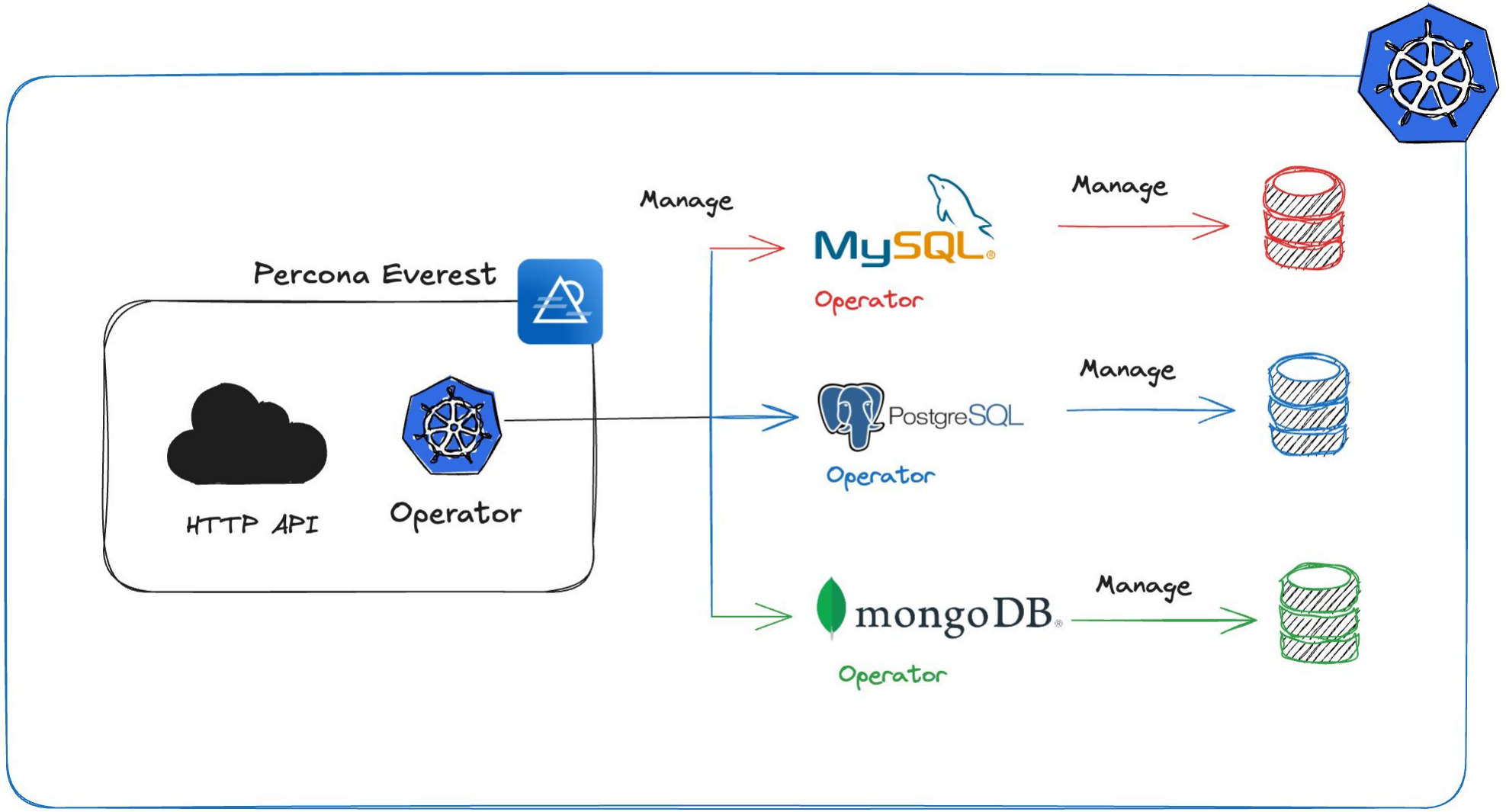
Enables DBaaS like experience (without vendor lock in)

Completely self hosted – you are in control of everything

Fully (and truly) open source

Cloud native

Architecture



PERCONA

Key features

- Private DBaaS, fully self-hosted – no vendor lock in
- Multiple database support (MongoDB, MySQL, Postgres) - harnesses the power of Percona Operators
- Horizontal & Vertical scaling
- Backups & Restores
- Sharding (MongoDB)
- Resource allocation flexibility
- Monitoring - support for integration with Percona Monitoring and Management
- Authentication - native user management capabilities & OIDC integration
- Authorization - support for Role-based access control

Ways to use Percona Everest

Web UI

HTTP API

CRDs (Custom Resource Definitions)

The Setup

- Git repository
- Kubernetes cluster
- Teams commit Percona Everest CRs to a Git repository
- ArgoCD watches the Git repository and applies CRs to the cluster
- Percona Everest will reconcile the CRs (provision database)
- SealedSecrets for managing secrets
- Read-only Everest web UI

Demo

Questions?

Thank You!

Lets connect

https://www.linkedin.com/in/mayankshah1607/
https://x.com/mayankshah__