CNCF Project Focus

Episode #4

PROMETHEUS

Cloud-Native Metrics & Monitoring

Christian Dussol

Dashboards show you what happened. **Metrics** tell you what to do next.

Most clusters have the first. Few have the second.

You can't **optimize** what you can't **measure**.

Your Kubernetes cluster runs 300 microservices.

→ Which ones are over-provisioned?
→ Which ones are about to break?
→ How much are you actually spending?

Without **metrics**, every **decision** is a **guess**.

What you cannot see

Without metrics visibility

❌ No idea which services cost the most
❌ Over-provisioned "just in case"
❌ Manual capacity planning
❌ Budget discussions = guesswork
❌ No team accountability

With metrics-based visibility

✅ Cost attribution per namespace/pod
✅ Automated right-sizing alerts
✅ Team-level showback dashboards
✅ Data-driven capacity planning
✅ Every optimization decision backed by data

What is Prometheus ?

CNCF Graduated project (August 2018)

One of the first CNCF graduated projects (Production-proven at hyperscale)

CNCF's first mature Observability stack foundation.

Core Idea

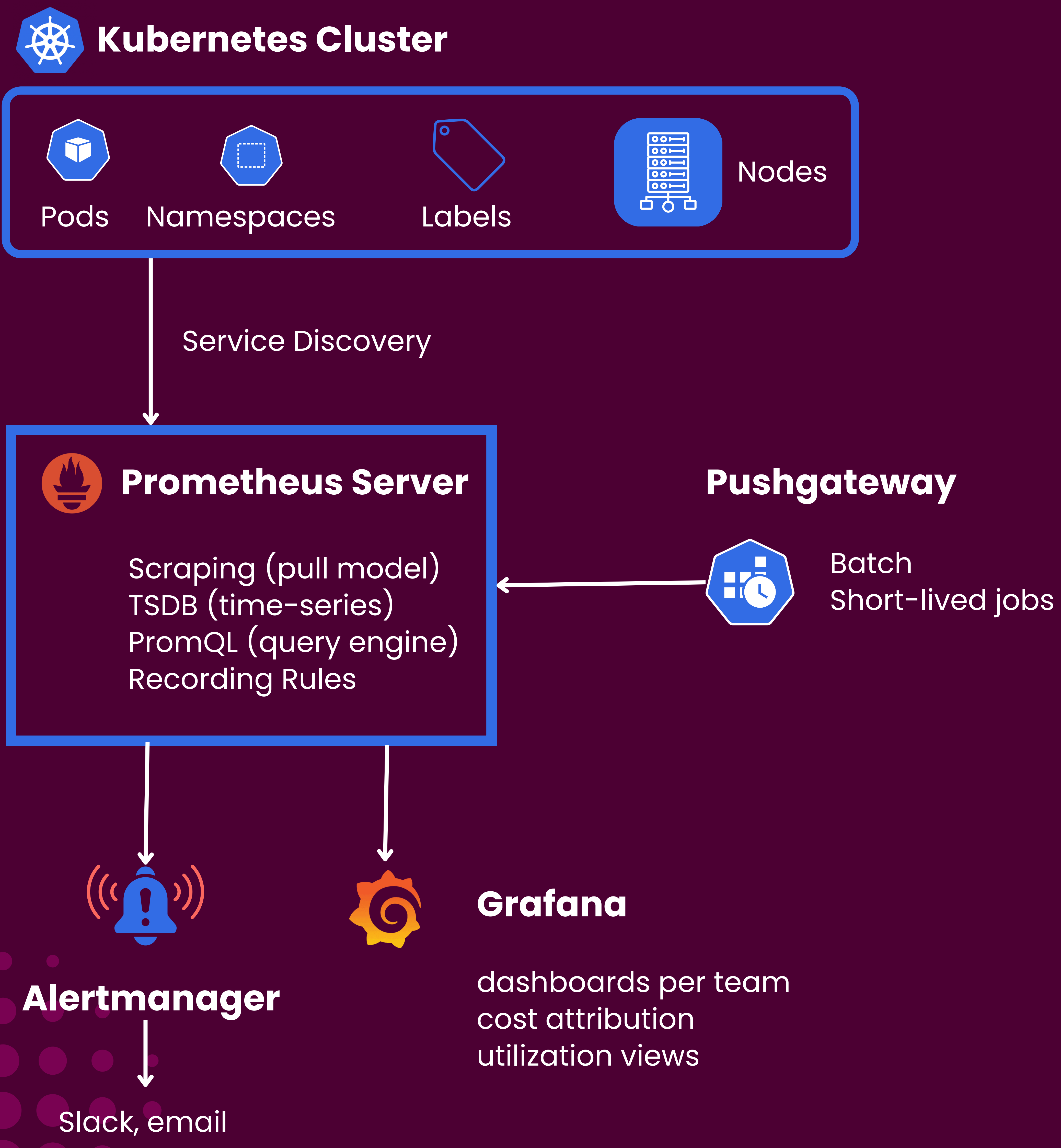
Pull-based metrics collection with powerful query language

🎯 Time-series database
📊 PromQL for querying
🔔 Built-in alerting

Adopted by

NETFLIX Uber reddit GitLab

From Metrics to Decisions



Kubernetes Cluster

Pods

Namespaces

Labels

Nodes

Service Discovery

Prometheus Server

Scraping (pull model)
TSDB (time-series)
PromQL (query engine)
Recording Rules

Pushgateway

Batch
Short-lived jobs

Alertmanager

Slack, email

Grafana

dashboards per team
cost attribution
utilization views

Prometheus for Cost Optimization

4 pillars to control your cloud spend

Resource Cost Attribution

CPU consumption per namespace

```
sum by (namespace)
(rate(container_cpu_usage_seconds_total[5m])
)
```

Utilization Tracking

Memory efficiency per pod

```
avg(container_memory_usage_bytes
/ container_spec_memory_limit_bytes)
```

Waste Detection

Identify over-provisioned workloads

→ Compare actual usage vs requested resources

Budget Alerts

Trigger alerts when thresholds are exceeded

→ Combine PromQL with Alertmanager rules

Example PromQL patterns: full queries in my GitHub repo

Know your tool

Prometheus is built for

✅ Kubernetes metrics (native integration)
✅ Real-time monitoring & alerting
✅ Cost tracking & optimization
✅ SLA/SLO monitoring
✅ Compliance reporting (audit trails)
✅ Multi-tenant cost attribution

Prometheus is NOT designed for

❌ Full distributed tracing → Jaeger / Tempo
❌ Long-term scalable storage → Thanos / Cortex
❌ Log aggregation → Loki
❌ Event-based monitoring → CloudEvents

From the field

Treasury solution (Front, Middle, Back office)

Our observability stack

🔥 **Prometheus**: metrics & alerting
📊 **Grafana**: team dashboards
🔭 **OpenTelemetry**: distributed tracing
🌐 **Istio**: service mesh observability

The journey

Phase 1: Deploy the stack
→ *First time we could see our system*

Phase 2: Reactive monitoring
→ *Alerts fire, investigate, fix*

Phase 3: Proactive

→ *Working with SREs to define alerts that anticipate problems before they impact users*

Deploying tools is step one.

Shifting from reactive to proactive is where it gets hard.

PROMETHEUS + KYVERNO

Observability meets **Policy-as-Code**

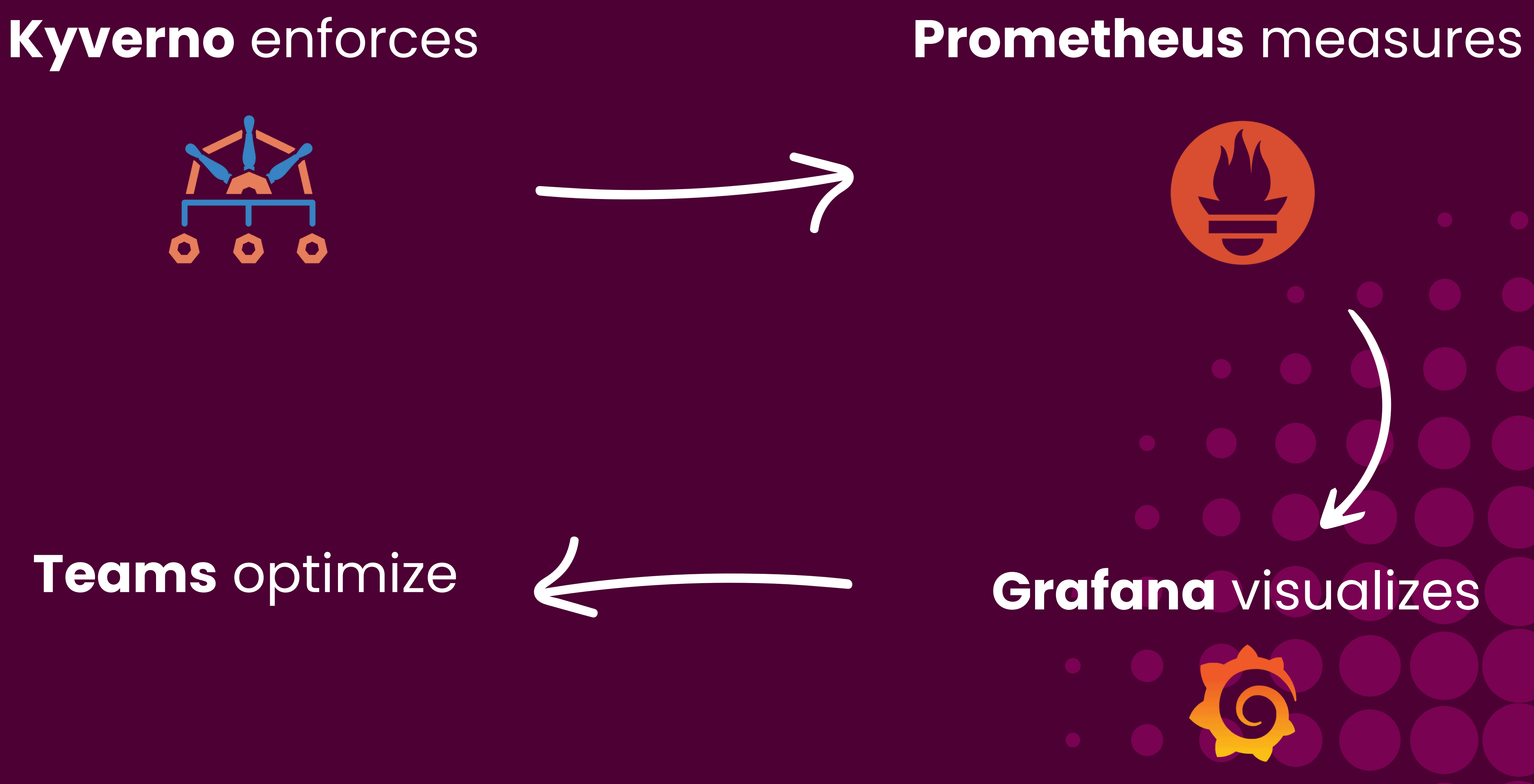
🛡️ Resource Governance

Kyverno enforces CPU/memory limits on every pod
→ Prometheus metrics become accurate and meaningful

💰 Cost Governance

Kyverno mandates team/project labels
→ Prometheus enables cost attribution per team

The Governance loop



Kyverno enforces

Prometheus measures

Grafana visualizes

Teams optimize

Educational GitHub Repository

Explore my learning toolkit

github.com/christian-dussol-cloud-native/prometheus

Complete hands-on tutorial, including:

✅ **Quick start**
Cluster setup, Prometheus & Grafana installation

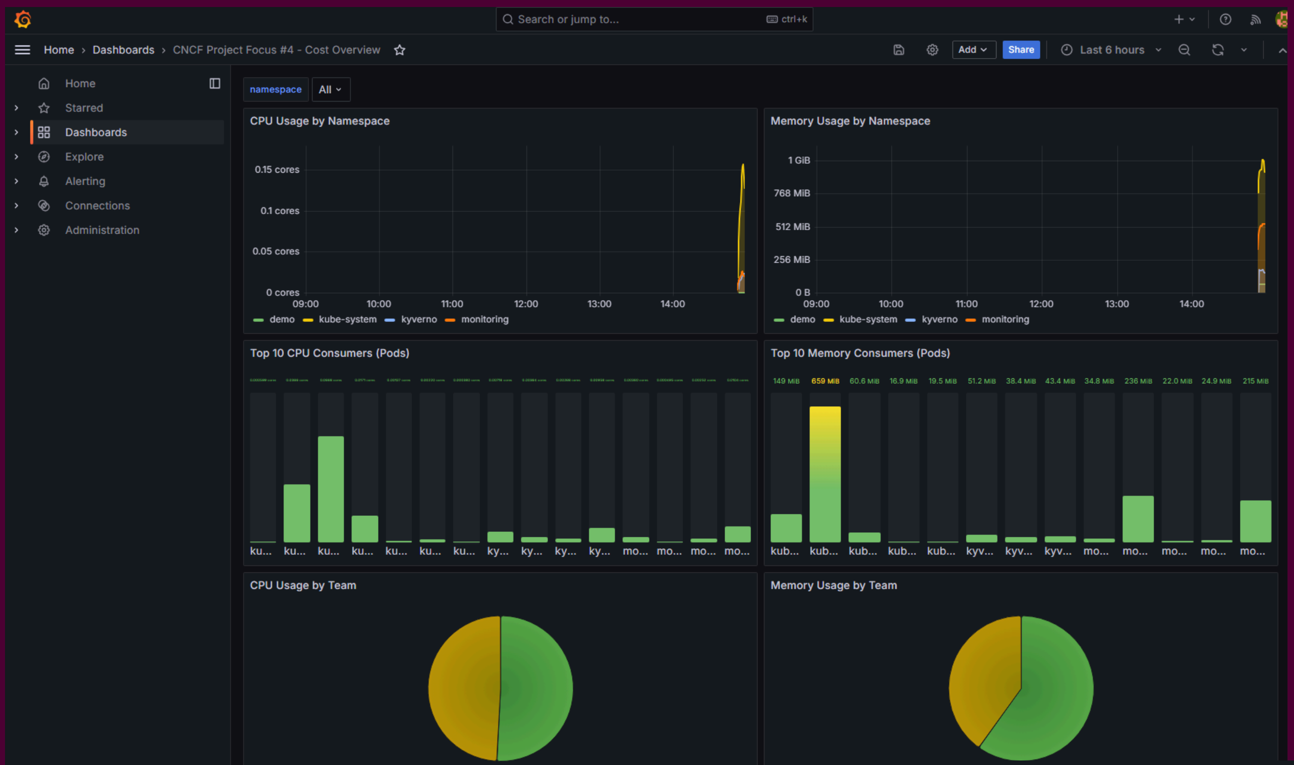
✅ Complete setup scripts (Helm)

✅ PromQL cost queries

✅ Grafana dashboards (importable JSON)

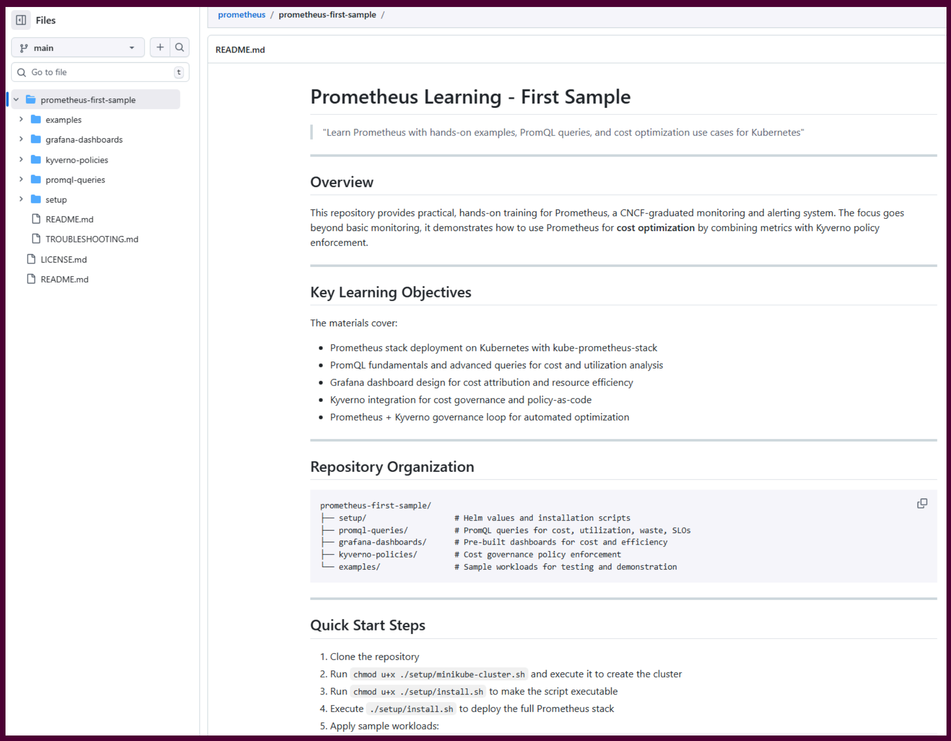
✅ Kyverno policies for cost governance

Educational GitHub Repository



Namespaces with Over-Provisioning Indicators

Time	namespace
2026-02-22 14:58:14.401	monitoring
2026-02-22 14:58:14.401	demo



Prometheus Learning - First Sample

> "Learn Prometheus with hands-on examples, PromQL queries, and cost optimization use cases for Kubernetes"

Overview

This repository provides practical, hands-on training for Prometheus, a CNCF-graduated monitoring and alerting system. The focus goes beyond basic monitoring, it demonstrates how to use Prometheus for **cost optimization** by combining metrics with Kyverno policy enforcement.

Key Learning Objectives

The materials cover:

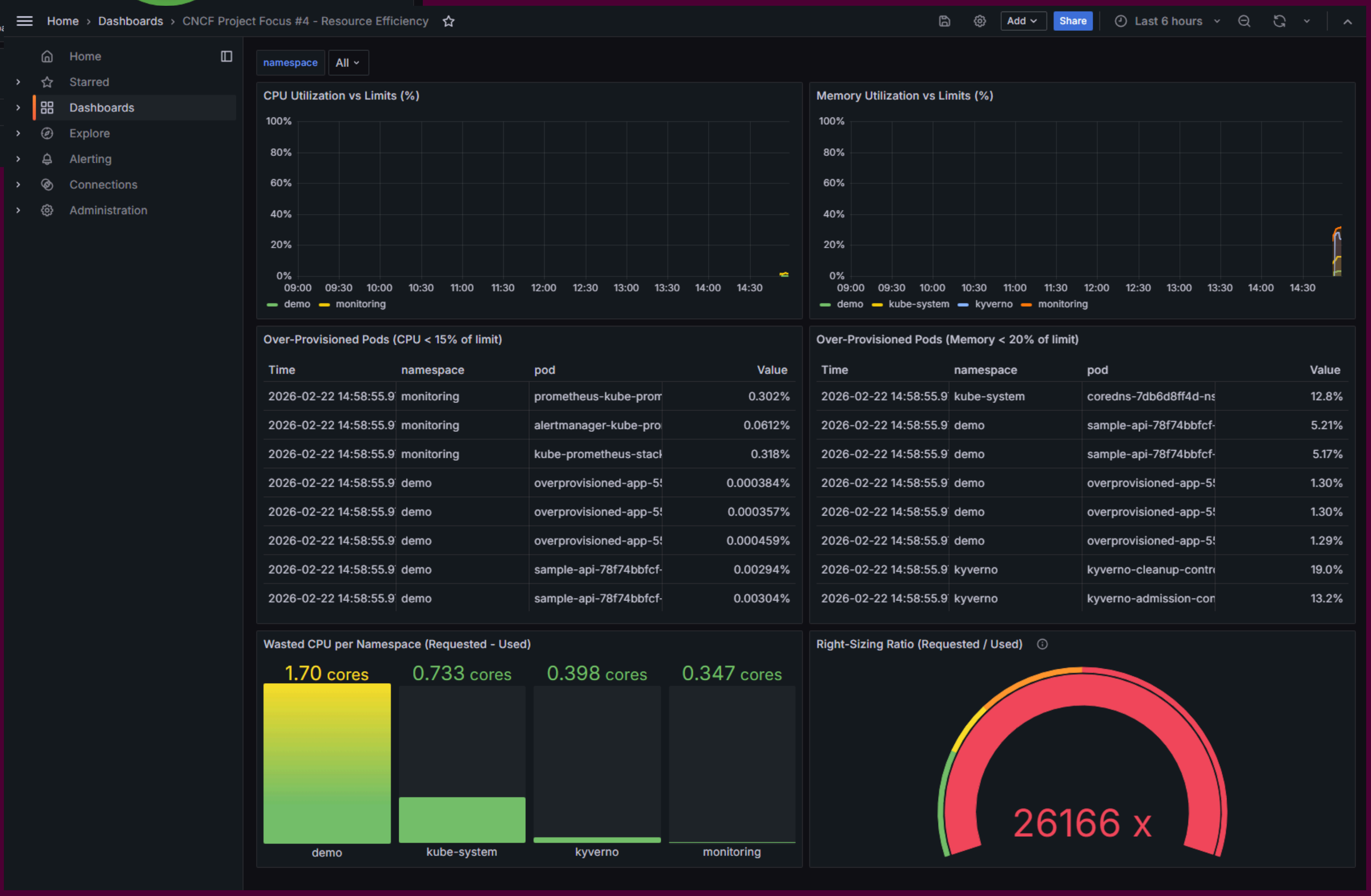
- Prometheus stack deployment on Kubernetes with kube-prometheus-stack
- PromQL fundamentals and advanced queries for cost and utilization analysis
- Grafana dashboard design for cost attribution and resource efficiency
- Kyverno integration for cost governance and policy-as-code
- Prometheus + Kyverno governance loop for automated optimization

Repository Organization

```
prometheus-first-sample/
├── setup/               # Helm values and installation scripts
├── promql-queries/      # PromQL queries for cost, utilization, waste, SLOs
├── grafana-dashboards/  # Pre-built dashboards for cost and efficiency
├── kyverno-policies/    # Cost governance policy enforcement
└── examples/            # Sample workloads for testing and demonstration
```

Quick Start Steps

1. Clone the repository
2. Run `chmod u+x ./setup/minikube-cluster.sh` and execute it to create the cluster
3. Run `chmod u+x ./setup/install.sh` to make the script executable
4. Execute `./setup/install.sh` to deploy the full Prometheus stack
5. Apply sample workloads:

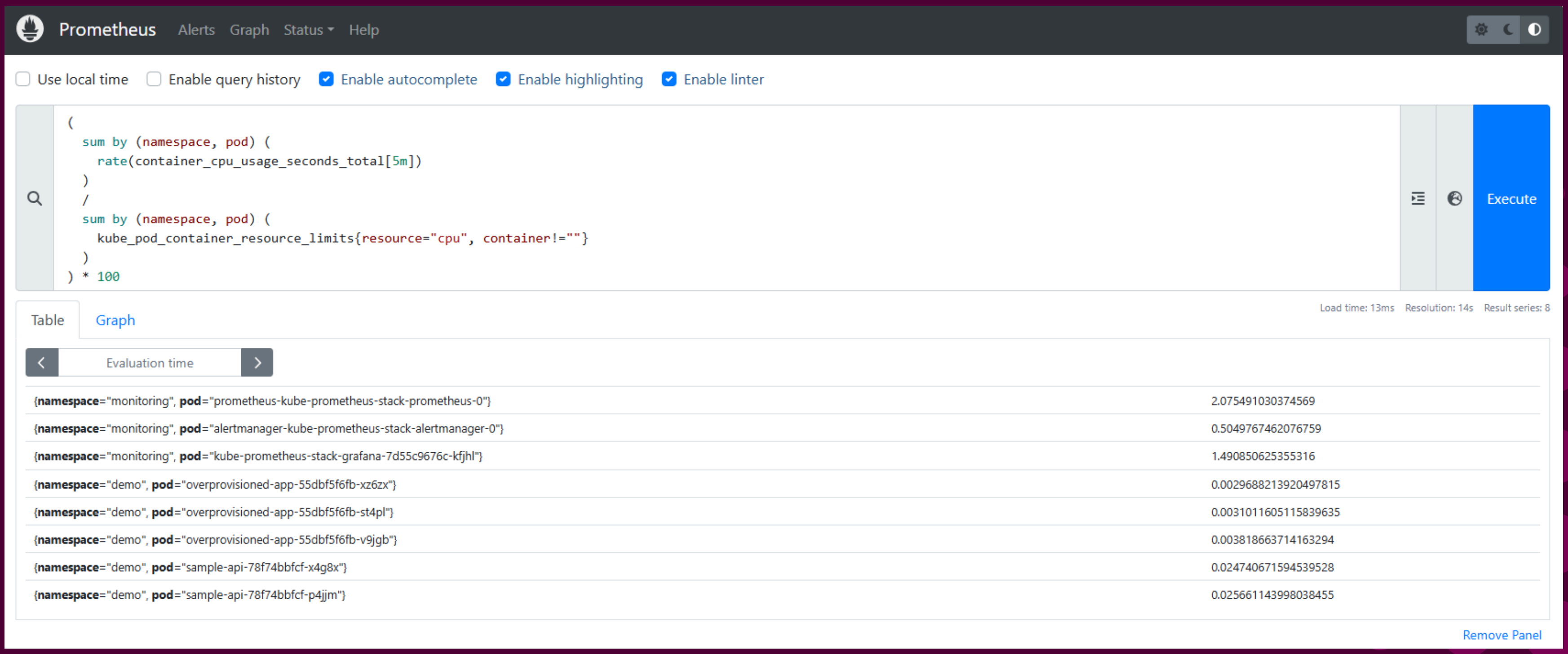


Over-Provisioned Pods (CPU < 15% of limit)

Time	namespace	pod	Value
2026-02-22 14:58:55.9	monitoring	prometheus-kube-prom	0.302%
2026-02-22 14:58:55.9	monitoring	alertmanager-kube-pro	0.0612%
2026-02-22 14:58:55.9	monitoring	kube-prometheus-stack	0.318%
2026-02-22 14:58:55.9	demo	overprovisioned-app-5	0.000384%
2026-02-22 14:58:55.9	demo	overprovisioned-app-5	0.000357%
2026-02-22 14:58:55.9	demo	overprovisioned-app-5	0.000459%
2026-02-22 14:58:55.9	demo	sample-api-78f74bbfcf-	0.00294%
2026-02-22 14:58:55.9	demo	sample-api-78f74bbfcf-	0.00304%

Over-Provisioned Pods (Memory < 20% of limit)

Time	namespace	pod	Value
2026-02-22 14:58:55.9	kube-system	coredns-7db6d8ff4d-ns	12.8%
2026-02-22 14:58:55.9	demo	sample-api-78f74bbfcf-	5.21%
2026-02-22 14:58:55.9	demo	sample-api-78f74bbfcf-	5.17%
2026-02-22 14:58:55.9	demo	overprovisioned-app-5	1.30%
2026-02-22 14:58:55.9	demo	overprovisioned-app-5	1.30%
2026-02-22 14:58:55.9	demo	overprovisioned-app-5	1.29%
2026-02-22 14:58:55.9	kyverno	kyverno-cleanup-contr	19.0%
2026-02-22 14:58:55.9	kyverno	kyverno-admission-con	13.2%



□ Use local time □ Enable query history ☑ Enable autocomplete ☑ Enable highlighting ☑ Enable linter

```
(
  sum by (namespace, pod) (
    rate(container_cpu_usage_seconds_total[5m])
  )
  /
  sum by (namespace, pod) (
    kube_pod_container_resource_limits{resource="cpu", container!=""}
  )
) * 100
```

Load time: 13ms Resolution: 14s Result series: 8

```
{namespace="monitoring", pod="prometheus-kube-prometheus-stack-prometheus-0"}          2.075491030374569
{namespace="monitoring", pod="alertmanager-kube-prometheus-stack-alertmanager-0"}      0.5049767462076759
{namespace="monitoring", pod="kube-prometheus-stack-grafana-7d55c9676c-kfjhl"}         1.490850625355316
{namespace="demo", pod="overprovisioned-app-55dbf5f6fb-xz6zx"}                         0.0029688213920497815
{namespace="demo", pod="overprovisioned-app-55dbf5f6fb-st4pl"}                         0.0031011605115839635
{namespace="demo", pod="overprovisioned-app-55dbf5f6fb-v9jgb"}                         0.003818663714163294
{namespace="demo", pod="sample-api-78f74bbfcf-x4g8x"}                                  0.024740671594539528
{namespace="demo", pod="sample-api-78f74bbfcf-p4jjm"}                                  0.025661143998038455
```