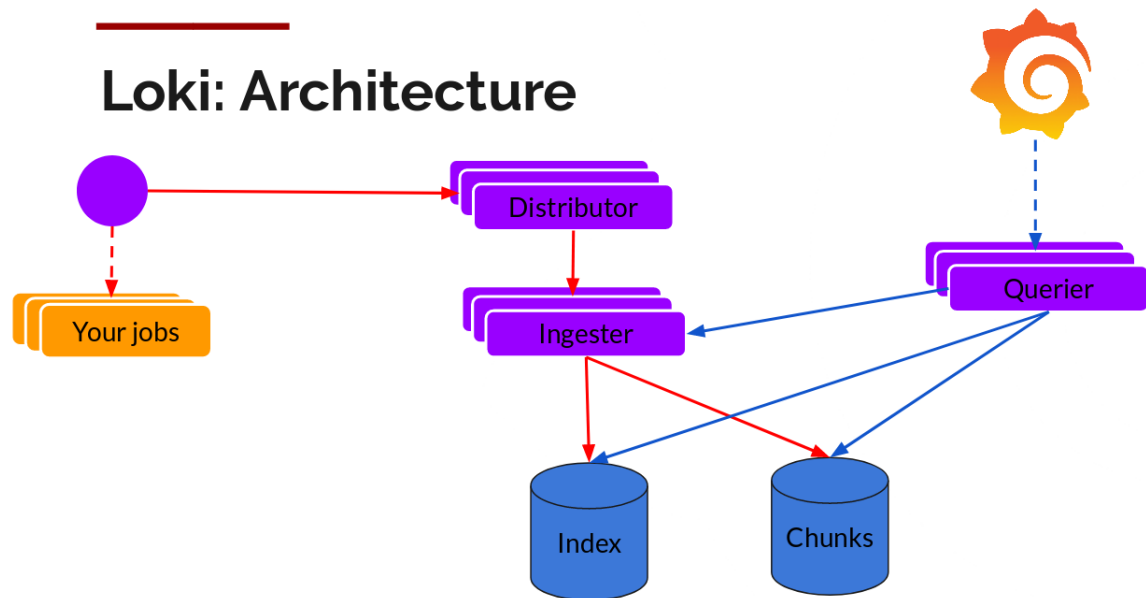


## Module 6: Monitoring and Logging in Kubernetes — Detailed Notes

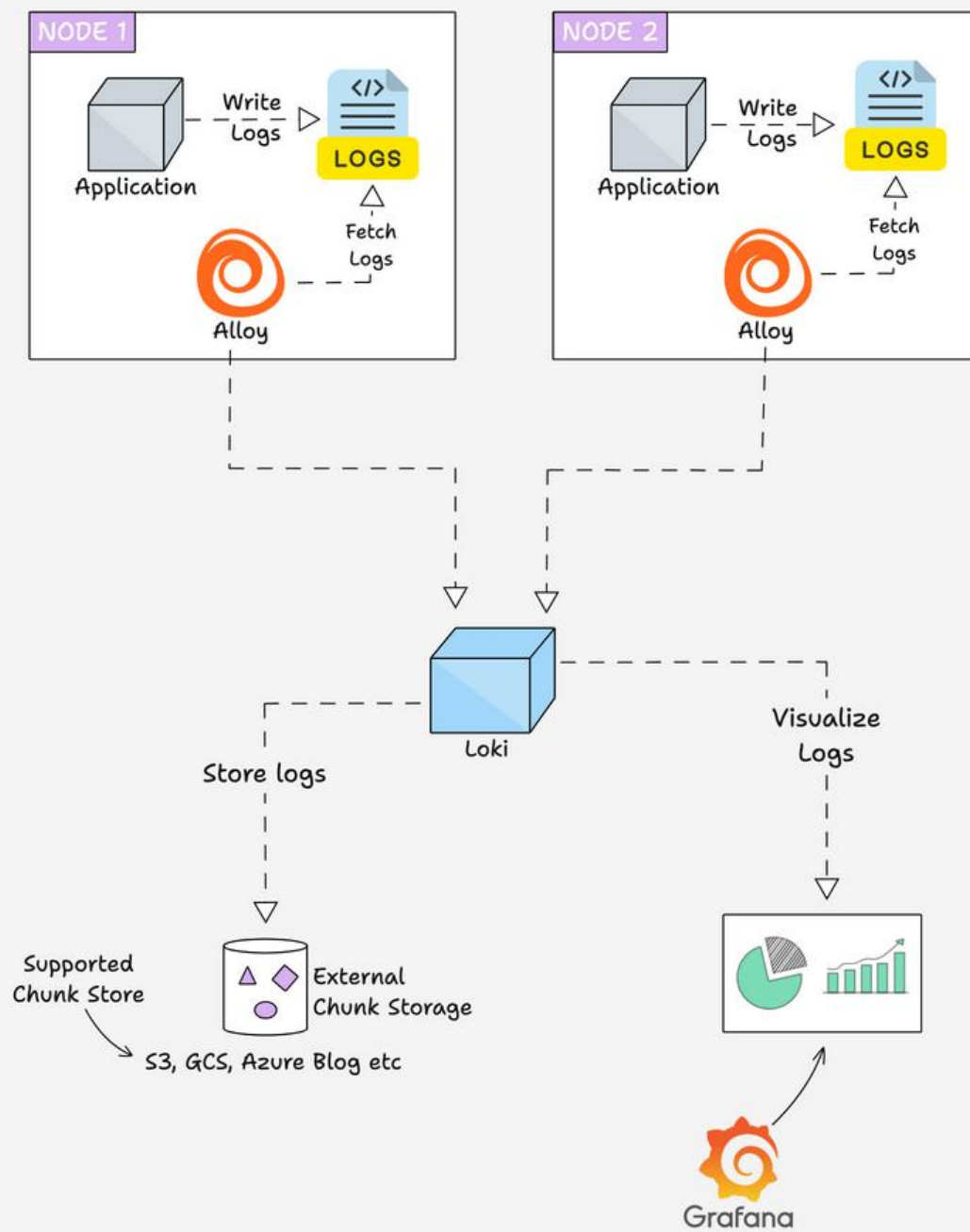
---

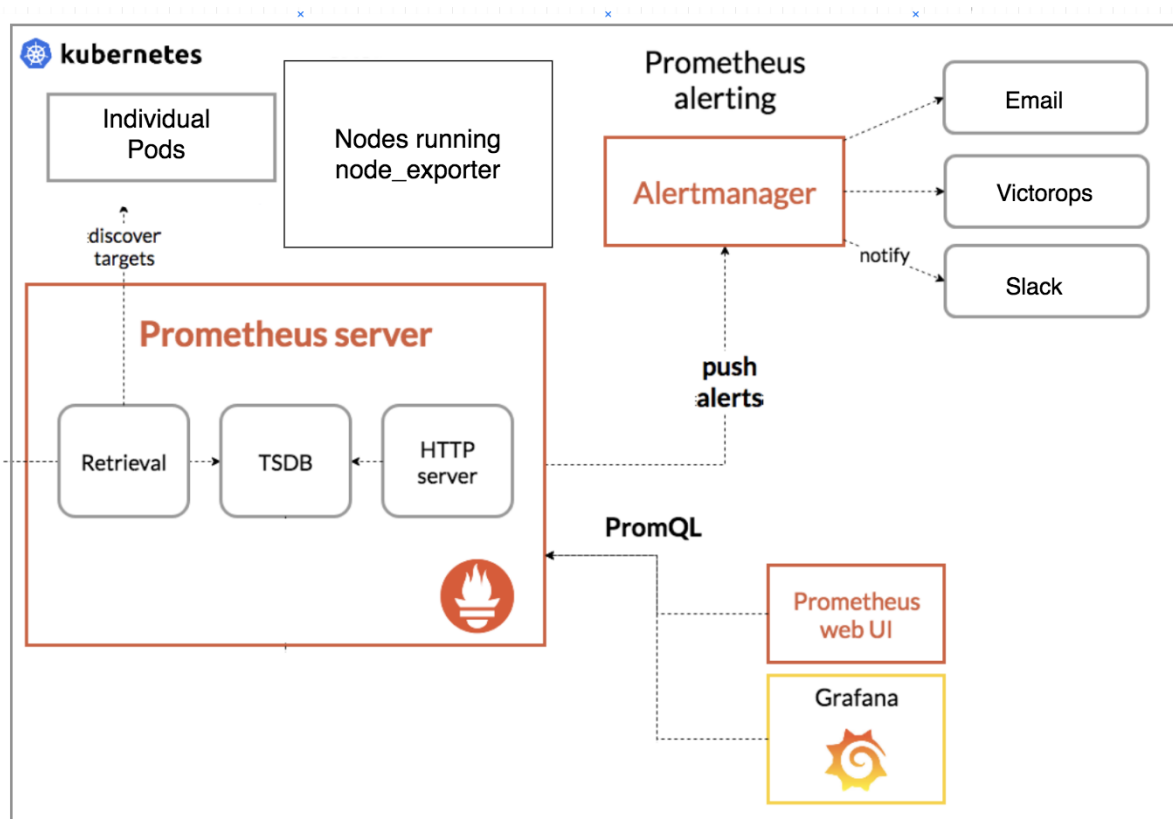
### 1. Prometheus, Grafana, Loki Setup



# LOKI ARCHITECTURE

devopscube.com





## 1.1 Prometheus Overview

Prometheus is a **time-series monitoring system** used for Kubernetes.

### Key Features:

- Pull-based metrics collection
- Multi-dimensional data model
- PromQL query language
- Service discovery
- AlertManager integration

## 1.2 Prometheus Setup (Helm)

Install Prometheus stack:

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

```
helm install k8s-monitoring prometheus-community/kube-prometheus-stack
```

This installs:

- Prometheus
- Alertmanager

- Node Exporters
- Kube-state-metrics
- Grafana

### 1.3 Grafana Overview

Grafana visualizes metrics from:

- Prometheus
- Loki
- ElasticSearch
- CloudWatch
- InfluxDB

#### Grafana Features:

- Dashboards
- Alerts
- Data sources
- Provisioning (automated setup)

Access Grafana:

```
kubectrl port-forward svc/k8s-monitoring-grafana 3000:80 -n monitoring
```

---

### 1.4 Loki (Logging System)

Loki is a **log aggregation system** optimized for Kubernetes.

#### Why Loki?

- Does **not index logs**, only labels → cheaper storage
- Lightweight compared to ELK stack
- Integrates with Promtail + Grafana

### 1.5 Loki Setup using Helm

```
helm repo add grafana https://grafana.github.io/helm-charts
```

```
helm install loki grafana/loki-stack
```

Installs:

- Loki
- Promtail (log shipper)
- Grafana dashboards

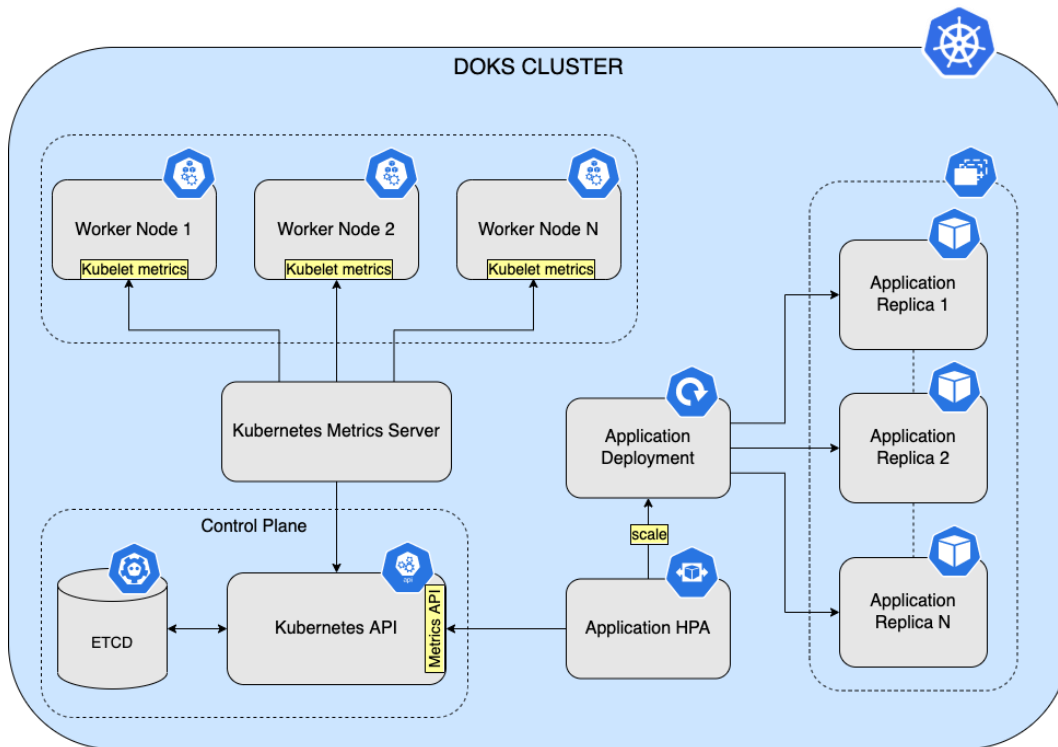
## 1.6 Architecture Flow

Promtail → Loki → Grafana

Node Exporter → Prometheus → Grafana

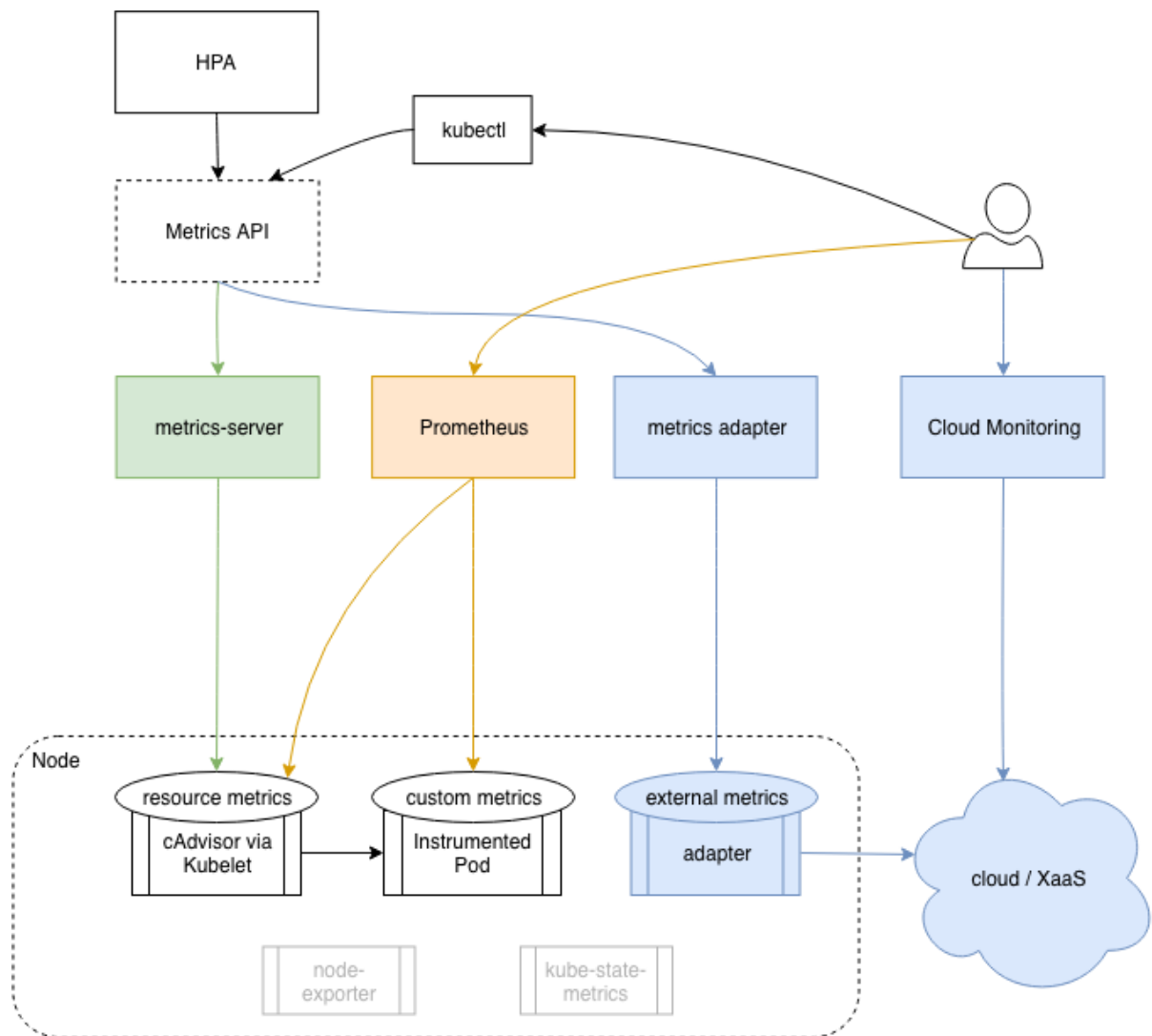
Kube-state-metrics → Prometheus → Grafana

## 2. Kubernetes Metrics Server & Alerts



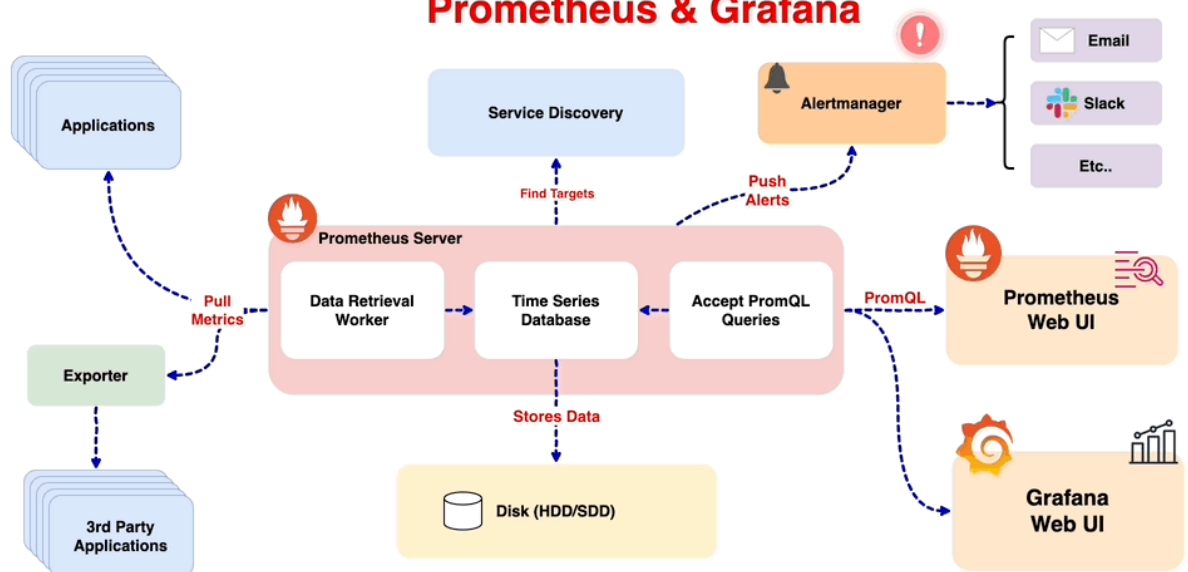
### Overview:

- Deploy Metrics Server to your DOKS cluster.
- Metrics server scrapes kubelet metrics from each worker node, collecting CPU and memory utilization data for each application workloads.
- Metrics server exposes CPU and memory usage metrics via the Kubernetes API server.
- Horizontal Pod Autoscaler fetches CPU and memory usage metrics, via the Kubernetes API server. Then, based on metrics observation and target threshold, decides when to scale up or down your application deployment Pods.



## K8s Monitoring Using Prometheus & Grafana

Anvesh Muppada



---

## 2.1 Metrics Server

Metrics Server provides **resource usage metrics**:

- CPU
- Memory
- Node usage
- Pod usage

These metrics are required for:

- Horizontal Pod Autoscaling (HPA)
- `kubectl top`

Install:

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

Check metrics:

```
kubectl top nodes
```

```
kubectl top pods
```

---

## 2.2 Prometheus Alerts

Prometheus uses **AlertManager** to:

- Send notifications
- Trigger alerts
- Silence, group, inhibit alerts

Alert rules file:

groups:

```
- name: cpu-alerts
```

rules:

```
- alert: HighCPU
```

```
  expr: node_cpu_seconds_total > 80
```

```
  for: 2m
```

labels:

```
  severity: warning
```

annotations:

description: "CPU usage high on node"

Alert Receiver options:

- Email
- Slack
- PagerDuty
- Webhooks
- Teams

---

## 2.3 Using Grafana for Alerts

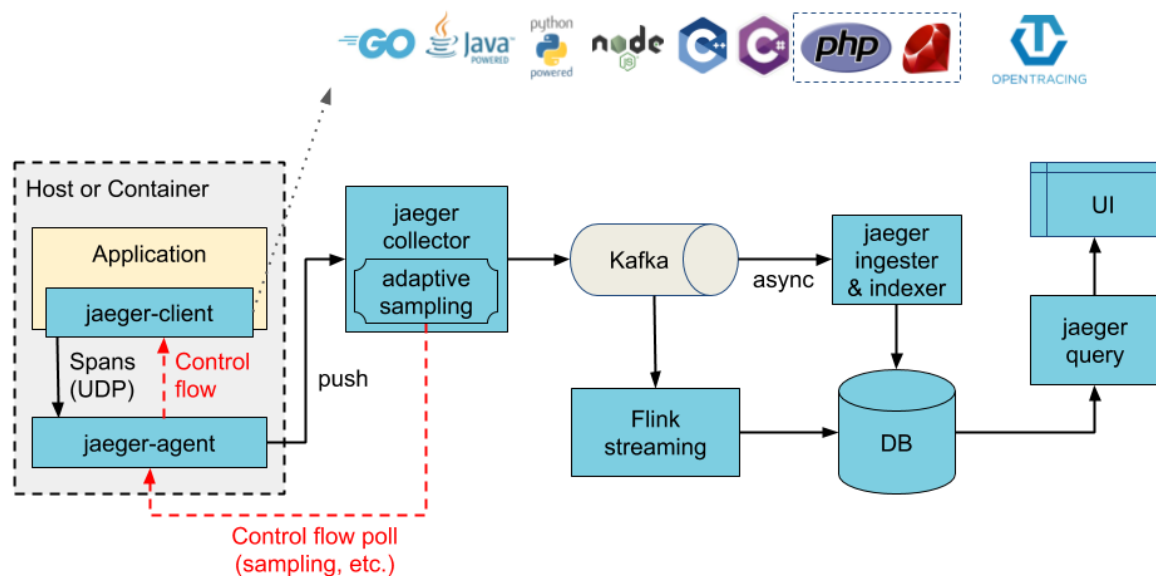
Grafana provides **Unified Alerting**:

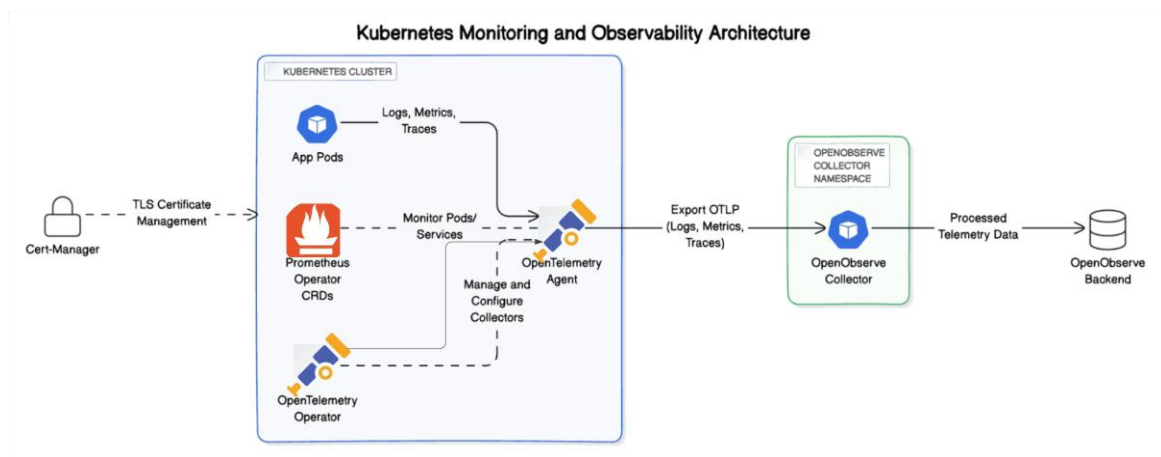
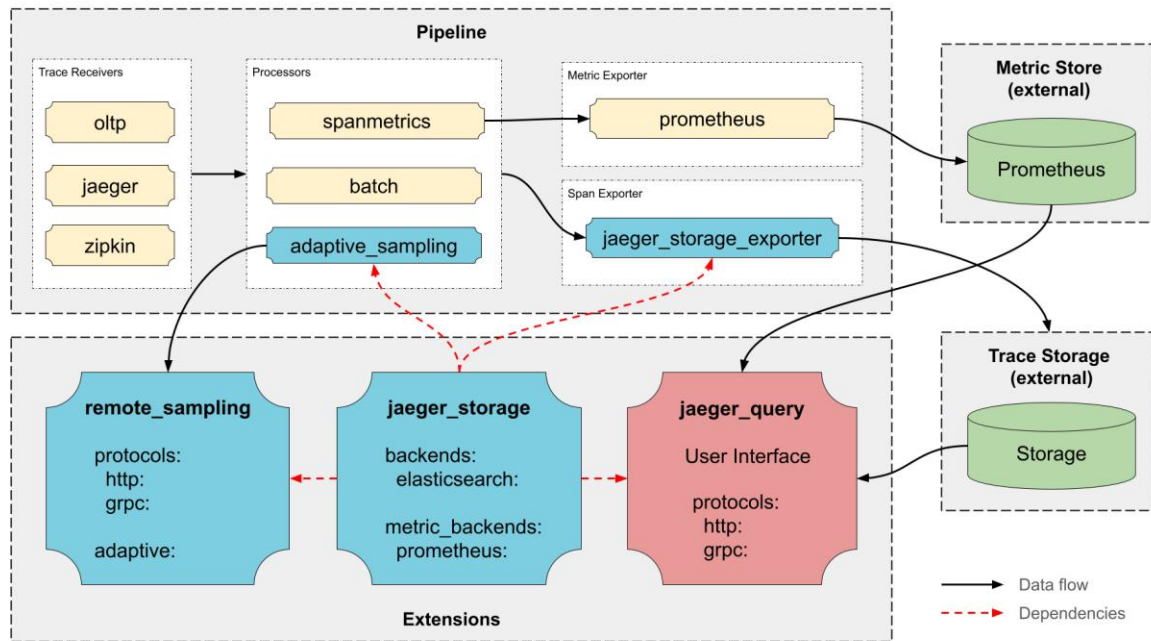
Features:

- Alerts from multiple data sources
- Alert rules in UI
- Slack/MS Teams webhooks

---

## 3. Distributed Tracing (Jaeger / OpenTelemetry)





### 3.1 What is Distributed Tracing?

Distributed tracing is used to analyze:

- Microservice latency
- Request path across services
- Bottlenecks
- Performance issues

Core concepts:

- **Trace** → Entire request lifecycle
- **Span** → Individual unit of work

- **Service graph** → How services interact
- 

### 3.2 Jaeger Overview

Jaeger provides:

- Trace storage
- Query UI
- Service Dependency Graph
- Sampling strategies

Deploy Jaeger in Kubernetes:

kubectl create namespace observability

kubectl apply -f <https://github.com/jaegertracing/jaeger-operator/releases/latest/download/jaeger-operator.yaml>

Create simple Jaeger instance:

apiVersion: jaegertracing.io/v1

kind: Jaeger

metadata:

name: jaeger

spec:

strategy: allInOne

---

### 3.3 OpenTelemetry (OTel)

OpenTelemetry provides **standard formats** for:

- Metrics
- Logs
- Traces

Components:

- Instrumentation SDKs
- OpenTelemetry Collector
- Exporters (Jaeger, Loki, Prometheus, Zipkin)

Collector Example:

receivers:

otlp:

protocols:

http:

grpc:

exporters:

jaeger:

endpoint: jaeger:14250

service:

pipelines:

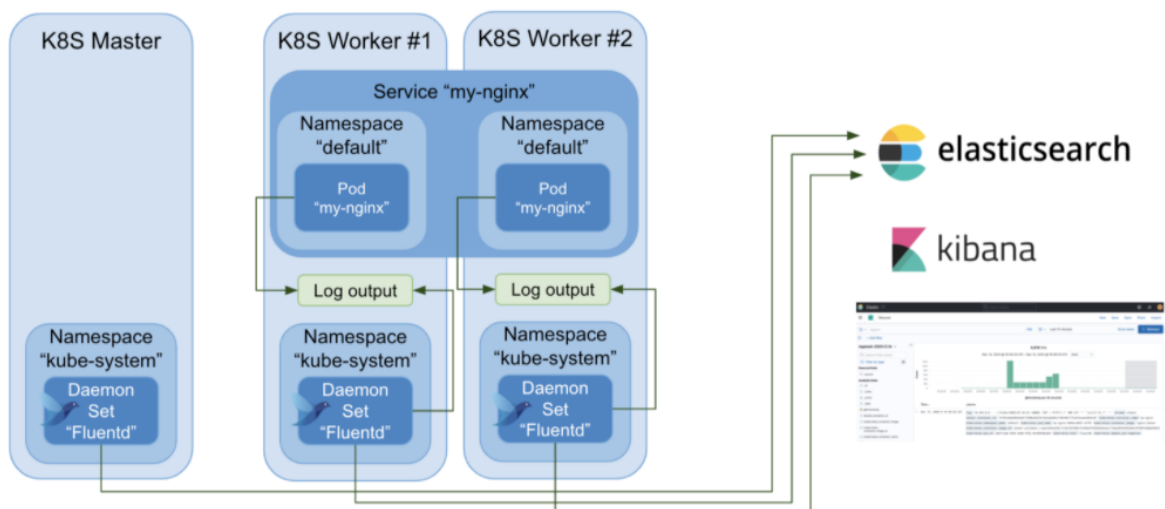
traces:

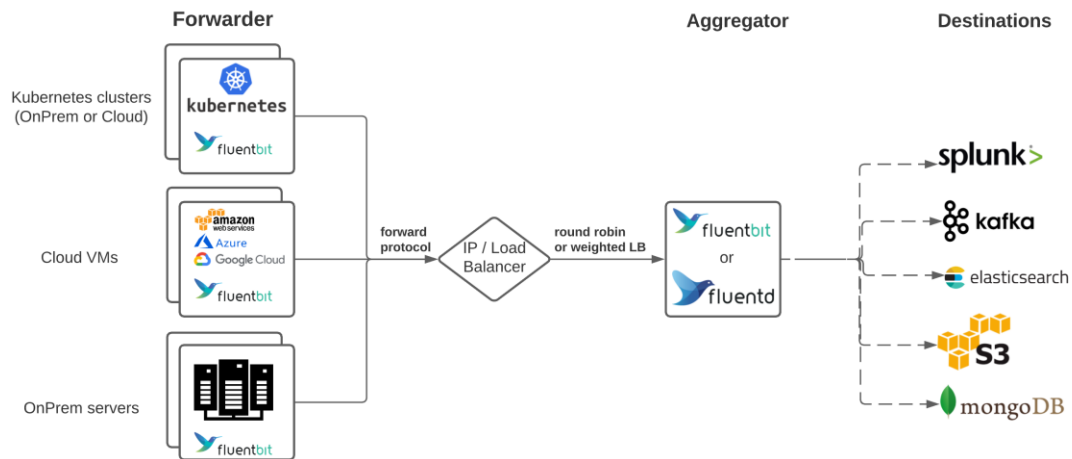
receivers: [otlp]

exporters: [jaeger]

---

#### 4. Log Aggregation with Fluentd / Filebeat





---

#### 4.1 Kubernetes Logging Challenges

- Logs are ephemeral
  - Containers restart frequently
  - Pods move between nodes
  - Developers need centralized access
- 

#### 4.2 Fluentd Overview

Fluentd is a **log collector and forwarder**.

**Features:**

- Parse logs
- Buffering
- Reliability
- Integrates with ElasticSearch, Loki, S3

Fluentd DaemonSet Example:

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: fluentd

spec:

template:

spec:

containers:

- name: fluentd

image: fluent/fluentd-kubernetes-daemonset:v1.16

---

### 4.3 Filebeat Overview

Filebeat is a lightweight **log shipper** from Elastic.

Key advantages:

- Low resource usage
- Auto-harvesting of logs
- Elasticsearch native integration

Installation using Helm:

helm repo add elastic <https://helm.elastic.co>

helm install filebeat elastic/filebeat

---

### 4.4 Logging Architecture Patterns

Logging Method	Description
Sidecar Logging	Log collector runs next to the app
Node Redirection	Container logs → node filesystem → collector
DaemonSet Collectors	Filebeat/Fluentd on each node
Streaming Logs	Application pushes logs to Kafka

Most Kubernetes clusters use:

**DaemonSet logging (Filebeat/Fluentd/Promtail)**

---

## Module 6 Summary

Topic	Summary
Prometheus	Metrics collection, scraping, alerting
Grafana	Dashboards + alerts
Loki	Log aggregation optimized for Kubernetes
Metrics Server	CPU/Memory usage for HPA
Alerts	PromQL + AlertManager
Jaeger	Distributed tracing & service graphs
OpenTelemetry	Unified logs, traces, metrics
Fluentd/Filebeat	Log aggregation and forwarding

---