Monitoring

TEKsystems Global Services

WORKFORCE DEVELOPMENT

Prometheus and SLO Dashboards

- This lab introduces Prometheus as the metrics engine for observability. You will use metrics to reason about reliability and performance.
  - Time series database
  - Pull based collection
  - Metrics focused
  - SLO driven

What Prometheus Is

- Prometheus is an open source system for collecting and storing metrics. It is designed for cloud native and microservice environments.
  - Metrics database
  - Scrapes endpoints
  - Label based model
  - Kubernetes native

Metrics as Telemetry

- Metrics are numeric telemetry emitted continuously over time. They describe system health and behavior.
  - Counts
  - Rates
  - Latencies
  - Resource usage

Prometheus Data Model

- Prometheus stores data as labeled time series. Each metric is identified by a name and a set of labels.
  - Metric name
  - Key value labels
  - Timestamped values
  - High cardinality

Metric Types

- Prometheus supports different metric types. Each type represents a different kind of measurement.
  - Counter
  - Gauge
  - Histogram
  - Summary

Counters

- Counters only increase over time. They are used to count events.
  - Request counts
  - Error counts
  - Always increasing
  - Use rate()

Gauges

- Gauges represent values that can go up and down. They show current state.
  - In progress requests
  - Memory usage
  - CPU usage
  - Queue size

Histograms

- Histograms track distributions of values. They are used for latency and size measurements.
  - Latency buckets
  - Request duration
  - Percentiles
  - histogram_quantile

Time Series

- Every metric produces a time series. PromQL operates over these time series.
  - Timestamped points
  - Continuous data
  - Historical trends
  - Graphable

PromQL Purpose

- PromQL is the query language for Prometheus. It is used to analyze time series data.
  - Filter metrics
  - Aggregate values
  - Calculate rates
  - Build SLOs

Instant vs Range Vectors

- PromQL works with two main data shapes. They represent values at a moment or over a window of time.
  - Instant vector
  - Range vector
  - Single timestamp
  - Time window

Instant Vectors

- Instant vectors represent metric values at a single point in time. They answer the question what is happening right now.
  - Current state
  - Used for gauges
  - Point in time
  - Live view

Range Vectors

- Range vectors represent metric values over a time window. They answer the question what happened recently.
  - Time window
  - Used for counters
  - Historical view
  - Trend analysis

Why rate() Exists

- Counters only go up so raw values are not useful. rate converts counters into meaningful per second signals.
  - Events per second
  - Normalizes counters
  - Used for traffic
  - Used for errors

Aggregation Operators

- PromQL aggregates multiple time series into fewer series. This is how we reason about systems instead of instances.
  - sum
  - avg
  - max
  - min

The sum by Pattern

- sum by groups time series by a label. This is the most important PromQL pattern.
  - Group by label
  - Collapse dimensions
  - Service level view
  - Cluster view

Ratio Queries

- Ratios compare two related metrics. This is how SLOs are calculated.
  - Error rate
  - Availability
  - Success percentage
  - Reliability math

Availability Mental Model

- Availability is the percentage of successful requests. It measures whether users can use the system.
  - Success divided by total
  - User focused
  - Uptime metric
  - SLO foundation

Latency Mental Model

- Latency measures how long requests take. It describes user experience quality.
  - Response time
  - Performance signal
  - Distribution based
  - Not averages

Why Percentiles Matter

- Averages hide real user experience. Percentiles show the slowest users.
  - P50
  - P95
  - P99
  - Tail latency

MCQ

- What type of database is Prometheus?
  - Relational database
  - Document database
  - Time series database
  - Log database

Answer

- What type of database is Prometheus?
  - Relational database
  - Document database
  - **Time series database**
  - Log database

MCQ

- Which metric type should be used to count requests?
  - Gauge
  - Counter
  - Histogram
  - Summary

Answer

- Which metric type should be used to count requests?
  - Gauge
  - **Counter**
  - Histogram
  - Summary

MCQ

- Why is rate() commonly used with counters?
  - To reset counters
  - To convert counters into per-second signals
  - To delete old data
  - To group labels

Answer

- Why is rate() commonly used with counters?
  - To reset counters
  - **To convert counters into per-second signals**
  - To delete old data
  - To group labels

MCQ

- What does sum by (handler) achieve in PromQL?
  - Deletes labels
  - Groups time series by handler
  - Filters errors only
  - Creates new metrics

Answer

- What does sum by (handler) achieve in PromQL?
  - Deletes labels
  - **Groups time series by handler**
  - Filters errors only
  - Creates new metrics

MCQ

- Why are histograms used for latency?
  - They reduce storage cost
  - They store log messages
  - They track distributions of values
  - They replace gauges

Answer

- Why are histograms used for latency?
  - They reduce storage cost
  - They store log messages
  - **They track distributions of values**
  - They replace gauges

MCQ

- Why are percentiles preferred over averages for latency?
  - They are easier to calculate
  - They hide slow requests
  - They show tail latency
  - They reduce cardinality

Answer

- Why are percentiles preferred over averages for latency?
  - They are easier to calculate
  - They hide slow requests
  - **They show tail latency**
  - They reduce cardinality

Kubernetes Monitoring Environment

- In this lab you are working inside a real Kubernetes cluster. The goal is to become comfortable with a full production-style monitoring stack.
  - EKS cluster
  - Multiple nodes
  - Real workloads
  - Production patterns

The Cluster Mental Model

- A Kubernetes cluster is a group of machines running workloads together. Monitoring happens at both the node level and application level.
  - Control plane
  - Worker nodes
  - Pods and services
  - Shared network

Why We Monitor the Cluster

- Clusters are dynamic systems where things constantly change. Without monitoring we cannot reason about reliability or performance.
  - Node health
  - Pod health
  - Resource usage
  - Service behavior

Node Exporter DaemonSet

- Node Exporter runs on every node in the cluster. It exposes system metrics about the host machine.
  - CPU usage
  - Memory usage
  - Disk usage
  - Network usage

What a DaemonSet Means

- A DaemonSet ensures one pod runs on every node. This guarantees full cluster coverage.
  - One per node
  - Automatic scaling
  - Infrastructure visibility
  - Cluster wide data

Grafana Alloy Agent

- Grafana Alloy is an observability agent running inside the cluster. It collects and forwards telemetry data.
  - Scrapes metrics
  - Collects logs
  - Forwards traces
  - Acts as a pipeline

Metrics Flow in the Cluster

- Metrics flow from nodes and applications into Prometheus. Prometheus stores and makes them queryable.
  - Node exporter
  - Service metrics
  - Prometheus scraping
  - Time series storage

Logs Flow in the Cluster

- Logs flow from pods into Loki through agents. This creates centralized logging for the entire cluster.
  - Pod logs
  - Alloy or Promtail
  - Loki storage
  - Searchable logs

Grafana as the Control Center

- Grafana sits on top of all telemetry systems. It is where humans interact with observability.
  - Dashboards
  - Explore mode
  - Alerts
  - Single pane of glass

The LGTM Stack

- The full stack gives complete observability coverage. Each component solves a different problem.
  - Loki for logs
  - Grafana for UI
  - Tempo for traces
  - Prometheus for metrics

Cluster Telemetry Sources

- Telemetry in Kubernetes comes from both infrastructure and applications. Every layer emits signals about its behavior.
  - Nodes
  - Kubernetes components
  - Applications
  - System services

Infrastructure vs Application Signals

- Infrastructure telemetry shows platform health. Application telemetry shows user experience.
  - CPU and memory
  - Pod scheduling
  - Request rates
  - Error rates

Why Agents Exist

- Agents run inside the environment being observed. They remove the need to modify every system manually.
  - Automatic discovery
  - Standard collection
  - Low overhead
  - Scalable

Service Monitors

- Service monitors tell agents what to scrape. They define how metrics are collected from services.
  - Target selection
  - Scrape endpoints
  - Label mapping
  - Dynamic discovery

Cluster Wide Visibility

- With agents and exporters deployed, every node and service is visible. This creates a real time picture of the entire system.
  - No blind spots
  - Unified telemetry
  - Consistent metrics
  - Full coverage

Observability as a Platform

- Observability becomes part of the platform itself. It is no longer an add-on tool.
  - Always on
  - Shared service
  - Built into cluster
  - Operational foundation

Why GitOps for Monitoring

- Monitoring configuration lives in Git. This makes the monitoring stack reproducible and auditable.
  - Version control
  - Change history
  - Rollbacks
  - Consistency

ArgoCD Mental Model

- ArgoCD continuously reconciles desired state. It keeps the cluster aligned with Git.
  - Declarative
  - Self healing
  - Automatic sync
  - Drift detection

From Dev to Production

- The same monitoring patterns apply across environments. Only scale and complexity change.
  - Local clusters
  - Staging
  - Production
  - Same stack

The Goal of This Lab

- This lab is about understanding the environment. Not about mastering tools or commands.
  - System thinking
  - Platform awareness
  - Observability mindset
  - Confidence in the stack