Introduction to Grafana

- Grafana is an open-source platform used to visualize, monitor, and explore data from many sources. It is a core tool for modern observability and incident response workflows.
  - Web-based visualization platform
  - Open source and widely adopted
  - Used by SRE and platform teams
  - Focused on observability use cases

What Grafana Is

- Grafana turns raw data into dashboards made of graphs, charts, and alerts. It helps teams understand system behavior at a glance.
  - Dashboards and panels
  - Time-series focused
  - Query-based visualizations
  - Supports alerting

What Grafana Is Not

- Grafana does not store metrics or logs by itself in most cases. It relies on external data sources to provide information.
  - Not a database
  - Not a metrics collector
  - Not a log storage engine
  - Acts as a visualization layer

Why Grafana Is Used

- Grafana is commonly used to monitor applications and infrastructure in real time. It helps teams detect issues early and understand system behavior.
  - Application monitoring
  - Infrastructure visibility
  - Real-time dashboards
  - Shared team views

Grafana for Observability

- Observability focuses on understanding system health through data. Grafana acts as the visualization layer for observability stacks.
  - Metrics visualization
  - Log exploration
  - Trace analysis
  - Unified dashboards

Common Grafana Use Cases

- Grafana supports many operational and business monitoring scenarios. It is flexible enough for simple and complex environments.
  - Response time tracking
  - Error rate monitoring
  - Capacity planning
  - Incident investigation

Grafana Architecture Basics

- Grafana runs as a web application accessed through a browser. It queries external data sources to render dashboards.
  - Web-based UI
  - Backend query engine
  - External data sources
  - Stateless by design

Running Grafana Locally

- Grafana can be run locally for testing and learning purposes. Docker provides a fast and consistent way to start it.
  - No local install required
  - Runs as a container
  - Accessible via browser
  - Easy to reset

Dockerized Grafana

- When running in Docker, Grafana behaves like any web service. Ports and logs are managed through Docker.
  - Listens on port 3000
  - Runs in background
  - Writes application logs
  - Managed with docker commands

Grafana Web Access

- Grafana exposes a web interface on port 3000 by default. Users interact with dashboards through the browser.
  - http://localhost:3000
  - Browser-based UI
  - No client install needed
  - Supports remote access

Grafana Authentication

- Grafana ships with default admin credentials for first-time use. Authentication can be customized in real deployments.
  - Default admin login
  - Password change prompt
  - User management support
  - SSO integrations available

Grafana Navigation

- Grafana’s UI is organized around dashboards and connections. The left sidebar provides access to core features.
  - Dashboards section
  - Connections menu
  - Alerting features
  - Administration settings

Exploring Safely

- Grafana encourages exploration without breaking data sources. Most actions are non-destructive.
  - Read-only dashboards
  - Preview queries
  - Safe experimentation
  - Undo and revert options

Observability Pillars Overview

- Observability is built on three core data types that explain system behavior. Together they provide context, depth, and confidence during incidents.
  - Metrics
  - Logs
  - Traces
  - Used together, not in isolation

Metrics

- Metrics are numeric measurements collected over time. They are the fastest way to detect abnormal system behavior.
  - CPU, memory, disk usage
  - Request rates and error rates
  - Latency and throughput
  - Efficient and low cost

Logs

- Logs are timestamped records of events and messages. They explain what happened when something goes wrong.
  - Application logs
  - System and service logs
  - Structured or unstructured text
  - High detail, higher volume

Traces

- Traces show how a single request flows through a system. They are essential for debugging distributed architectures.
  - End-to-end request visibility
  - Service-to-service latency
  - Dependency mapping
  - Critical for microservices

Why All Three Matter

- Each pillar answers a different operational question. Using only one creates blind spots.
  - Metrics show something is wrong
  - Logs explain why
  - Traces show where
  - Correlation is key

Monitoring vs Observability

- Monitoring tells you when a known problem occurs.
  Observability helps you understand unknown failures.
  - Monitoring is threshold-based
  - Observability is exploratory
  - Dashboards vs investigations
  - Both are required

MTTD and MTTR

- Good monitoring reduces how long issues go unnoticed. Observability reduces how long it takes to fix them.
  - MTTD: Mean Time To Detect
  - MTTR: Mean Time To Resolve
  - Faster detection saves downtime
  - Context speeds resolution

Why Monitoring Matters

- Anything that moves in production should be monitored. Without monitoring, reliability cannot be measured or improved.
  - Applications
  - Infrastructure
  - Networks
  - Dependencies

SLOs and SLAs

- Service reliability is defined using measurable objectives. Monitoring provides the data needed to track them.
  - SLOs define internal targets
  - SLAs define customer promises
  - Metrics validate compliance
  - No data means no guarantees

Grafana and Observability

- Grafana unifies metrics, logs, and traces into one view. This enables faster understanding during incidents.
  - Single-pane-of-glass
  - Cross-linked data
  - Incident-focused dashboards
  - Operational visibility

MCQ

- Which observability pillar is best for detecting abnormal behavior quickly?
  - Logs
  - Metrics
  - Traces
  - Dashboards

Answer

- Which observability pillar is best for detecting abnormal behavior quickly?
  - Logs
  - **Metrics**
  - Traces
  - Dashboards

MCQ

- What observability pillar helps explain why an error occurred?
  - Metrics
  - Logs
  - Traces
  - Alerts

Answer

- What observability pillar helps explain why an error occurred?
  - Metrics
  - **Logs**
  - Traces
  - Alerts

MCQ

- What data type shows how a request flows across multiple services?
  - Metrics
  - Logs
  - Traces
  - Dashboards

Answer

- What data type shows how a request flows across multiple services?
  - Metrics
  - Logs
  - **Traces**
  - Dashboards

MCQ

- Why is monitoring required to track SLOs and SLAs?
  - It stores application data
  - It provides measurable reliability data
  - It replaces incident response
  - It eliminates downtime

Answer

- Why is monitoring required to track SLOs and SLAs?
  - It stores application data
  - **It provides measurable reliability data**
  - It replaces incident response
  - It eliminates downtime

Telemetry Fundamentals

- Telemetry is the concept of an application emitting data about its own behavior. This data can later be collected and analyzed by external systems.
  - Emitted by the application itself
  - Collected by monitoring tools
  - Independent of vendors
  - Core to observability

Telemetry vs Monitoring Tools

- Telemetry is not a tool, database, or dashboard. It is the raw signal that monitoring systems consume.
  - Application responsibility
  - Tool agnostic
  - Exists before dashboards
  - Enables observability

Why Applications Emit Telemetry

- Applications emit telemetry so their internal state is visible externally. Without telemetry, systems operate as black boxes.
  - External visibility
  - Reduced guesswork
  - Inspectable behavior
  - Operational transparency

Telemetry Data Types

- Telemetry is commonly emitted as metrics, logs, and traces. Each type answers a different operational question.
  - Metrics show trends
  - Logs show events
  - Traces show flow
  - Used together

Metrics as Telemetry

- Metrics are numeric values emitted continuously over time. They measure health, performance, and reliability.
  - Request counts
  - Error rates
  - Latency
  - Resource usage

Logs as Telemetry

- Logs are event records emitted during application execution. They provide detailed context when something goes wrong.
  - Event history
  - High detail
  - Debugging focused
  - Human readable

Traces as Telemetry

- Traces describe how a single request moves through a system. They are essential in distributed architectures.
  - Request lifecycle
  - Service boundaries
  - Latency attribution
  - Dependency insight

Telemetry Improves MTTR and MTTD

- Telemetry allows teams to detect and resolve issues faster. This directly improves system reliability.
  - Lower MTTD
  - Lower MTTR
  - Faster root cause
  - Reduced downtime

Metrics Endpoints

- A metrics endpoint exposes telemetry in a standard format. Monitoring systems collect this data automatically.
  - /metrics endpoint
  - Machine readable
  - Pull based collection
  - Time series data

Telemetry and Reliability Targets

- Telemetry provides the data needed to measure reliability objectives. Without telemetry, SLOs and SLAs cannot be validated.
  - Failed request rates
  - Latency thresholds
  - Availability tracking
  - Reliability evidence

MCQ

- What best describes telemetry?
  - A monitoring dashboard
  - An application emitting data about itself
  - A logging format
  - A metrics database

Answer

- What best describes telemetry?
  - A monitoring dashboard
  - **An application emitting data about itself**
  - A logging format
  - A metrics database

MCQ

- Which telemetry type is primarily used to calculate SLOs?
  - Logs
  - Traces
  - Metrics
  - Alerts

Answer

- Which telemetry type is primarily used to calculate SLOs?
  - Logs
  - Traces
  - **Metrics**
  - Alerts

MCQ

- Why does telemetry reduce MTTR?
  - It replaces incident response
  - It provides context and visibility
  - It prevents failures entirely
  - It scales infrastructure automatically

Answer

- Why does telemetry reduce MTTR?
  - It replaces incident response
  - **It provides context and visibility**
  - It prevents failures entirely
  - It scales infrastructure automatically

MCQ

- What is the purpose of the /metrics endpoint?
  - To expose logs
  - To store metrics
  - To expose telemetry for scraping
  - To visualize dashboards

Answer

- What is the purpose of the /metrics endpoint?
  - To expose logs
  - To store metrics
  - **To expose telemetry for scraping**
  - To visualize dashboards

Next Lab: Centralized Logging with Loki

- In this lab, you will collect and query logs emitted by your API. These logs will be centralized for investigation and troubleshooting.
  - Centralized logging
  - Application visibility
  - Incident investigation
  - Log correlation

Why Centralized Logging

- Logs are most valuable when they are searchable and correlated. Centralized logging removes the need to access individual servers.
  - Single source of truth
  - Searchable history
  - Cross-service visibility
  - Faster debugging

Centralized Logging Overview

- Centralized logging collects logs from many services into one place. This makes searching, correlating, and retaining logs practical at scale.
  - Single log location
  - Searchable history
  - Multi-service visibility
  - Operational at scale

Why docker logs Is Not Enough

- docker logs works for a single container during development. It breaks down quickly in real production environments.
  - One container at a time
  - No historical search
  - No correlation
  - Not team friendly

Logging at Scale Problem

- Modern systems run many containers across many hosts. Logs must be centralized to remain useful.
  - Many containers
  - Frequent restarts
  - Distributed systems
  - Shared responsibility

What Is Loki

- Loki is a log aggregation system designed for observability.
  It is often described as Prometheus for logs.
  - Log aggregation
  - Label-based indexing
  - Grafana integration
  - Cost efficient

Loki Design Philosophy

- Loki indexes metadata instead of full log contents. This makes it simpler and cheaper than traditional logging systems.
  - Labels not full text
  - Low storage cost
  - Fast queries
  - Observability focused

The Logging Stack

- Centralized logging is built from multiple cooperating services. Each component has a clear responsibility.
  - Application
  - Log shipper
  - Log store
  - Visualization layer

Promtail Role

- Promtail collects logs from containers and forwards them to Loki. It runs close to the log source.
  - Reads container logs
  - Adds labels
  - Ships to Loki
  - Lightweight agent

Grafana and Logs

- Grafana provides a UI to explore and query logs stored in Loki. This removes the need for SSH access to servers.
  - Explore view
  - Search and filter
  - Time based queries
  - Team accessible

From CLI to Centralized Logs

- Centralized logging replaces ad hoc CLI commands. Queries become repeatable and shareable.
  - No grep pipelines
  - Saved queries
  - Shared dashboards
  - Audit friendly

When Centralized Logging Matters

- Centralized logging becomes essential as systems grow. It directly improves incident response and reliability.
  - Production incidents
  - Multiple services
  - On-call rotations
  - Post-incident reviews

Logs as Telemetry Signals

- Logs are a form of telemetry emitted by applications. They describe events that occurred during execution.
  - Application emitted
  - Event driven
  - Human readable
  - Operational signal

Why Centralize Logs

- Centralized logs provide a single source of truth. This enables faster investigation and collaboration.
  - Search once
  - Query across services
  - Team access
  - Historical retention

Promtail Log Collection

- Promtail runs close to where logs are generated. It forwards logs to Loki with useful labels attached.
  - Reads container logs
  - Discovers Docker containers
  - Adds metadata
  - Ships to Loki

Labels in Loki

- Labels describe log streams rather than log content. They are critical for efficient querying.
  - Container name
  - Application name
  - Environment
  - Service identity

Why Labels Matter

- Labels allow logs to be filtered before searching text. This keeps Loki fast and cost effective.
  - Pre-filtering
  - Reduced scan scope
  - Predictable queries
  - Scalable design

LogQL Basics

- LogQL is Loki’s query language for logs. It is inspired by PromQL and grep.
  - Label selectors
  - Line filters
  - Regex support
  - Time awareness

Selecting Log Streams

- LogQL queries begin by selecting streams using labels. This narrows the data before filtering text.
  - {container="telemetry-api"}
  - Label based selection
  - Efficient querying
  - Predictable results

Filtering Log Lines

- After selecting streams, LogQL filters log lines. This is similar to grep but queryable.
  - |= contains
  - |~ regex match
  - != exclude
  - Composable filters

From grep to LogQL

- Most grep patterns translate directly to LogQL. This makes LogQL easy to learn.
  - grep ERROR
  - |= "ERROR"
  - grep -v DEBUG
  - != "DEBUG"

Logs During Incidents

- Logs provide context when metrics indicate a problem. They help explain what happened and why.
  - Error messages
  - Warnings
  - Execution flow
  - Root cause clues

Using Grafana Explore for Logs

- Grafana Explore is the primary interface for querying logs in Loki. It allows fast, interactive investigation during incidents.
  - Explore view
  - Time range selection
  - Live tailing
  - Ad hoc analysis

Selecting a Data Source

- Log queries in Grafana require selecting Loki as the data source. This ensures queries are sent to the log backend.
  - Loki data source
  - Explore dropdown
  - Query context
  - Consistent results

Building a LogQL Query

- LogQL queries are built step by step. Labels narrow the scope before filtering content.
  - Select labels first
  - Filter lines second
  - Reduce noise
  - Improve performance

Time Range Matters

- Log queries always operate within a selected time range. Smaller ranges return results faster.
  - Incident windows
  - Reduce scan size
  - Faster feedback
  - Accurate context

Finding Errors Quickly

- Error logs are often the first clue during incidents. LogQL makes error discovery repeatable.
  - Filter ERROR
  - Combine with labels
  - Save queries
  - Share findings

Combining Multiple Filters

- LogQL filters can be chained together. This enables precise searches.
  - AND logic
  - Multiple |= operators
  - Noise reduction
  - Targeted results

Excluding Log Noise

- Not all logs are useful during investigations. Excluding noisy logs improves clarity.
  - Exclude DEBUG
  - Exclude health checks
  - Focus on failures
  - Cleaner output

Saved Log Queries

- Common log queries can be reused. This reduces investigation time.
  - Repeatable queries
  - Team shared knowledge
  - Operational consistency
  - Faster response

Logs and Metrics Together

- Metrics tell you that something is wrong. Logs help explain why it happened.
  - Metrics detect
  - Logs explain
  - Complementary signals
  - Correlated analysis

Logs in Post Incident Reviews

- Logs provide factual timelines after incidents. They support learning and improvement.
  - Event timelines
  - Root cause analysis
  - Evidence based reviews
  - Reliability improvement

MCQ

- What problem does centralized logging primarily solve?
  - Reducing application latency
  - Searching and correlating logs across services
  - Replacing metrics systems
  - Eliminating application errors

Answer

- What problem does centralized logging primarily solve?
  - Reducing application latency
  - **Searching and correlating logs across services**
  - Replacing metrics systems
  - Eliminating application errors

MCQ

- What is Loki optimized to index?
  - Full log message text
  - Application binaries
  - Log metadata and labels
  - Database records

Answer

- What is Loki optimized to index?
  - Full log message text
  - Application binaries
  - **Log metadata and labels**
  - Database records

MCQ

- What is the main role of Promtail?
  - Visualize logs
  - Store logs long term
  - Collect and ship logs to Loki
  - Generate application logs

Answer

- What is the main role of Promtail?
  - Visualize logs
  - Store logs long term
  - **Collect and ship logs to Loki**
  - Generate application logs

MCQ

- What is the first step in a LogQL query?
  - Filter log lines
  - Select log streams using labels
  - Apply regex matching
  - Aggregate results

Answer

- What is the first step in a LogQL query?
  - Filter log lines
  - **Select log streams using labels**
  - Apply regex matching
  - Aggregate results

MCQ

- Why are labels important in Loki?
  - They format log messages
  - They allow efficient pre-filtering
  - They replace log content
  - They store metrics

Answer

- Why are labels important in Loki?
  - They format log messages
  - **They allow efficient pre-filtering**
  - They replace log content
  - They store metrics

Kubernetes Monitoring Environment

- This lab introduces a full monitoring stack running inside Kubernetes. All observability components are deployed together as a unified system.
  - Cloud native environment
  - GitOps managed
  - Production style setup
  - End to end visibility

Why Microservices Need Observability

- Microservice architectures introduce complexity through distribution. Observability tools are required to understand system behavior.
  - Many services
  - Network boundaries
  - Independent deployments
  - Shared infrastructure

The LGTM Stack

- This environment runs the LGTM observability stack. Each component specializes in a different signal.
  - Loki for logs
  - Grafana for visualization
  - Tempo for traces
  - Metrics collection

Logs with Loki

- Loki stores logs emitted by applications running in the cluster. Logs are pushed into Loki and queried through Grafana.
  - Application stdout
  - Centralized storage
  - Label based queries
  - Incident context

Metrics with Prometheus

- Prometheus collects metrics by scraping endpoints. It uses a pull based model well suited for Kubernetes.
  - Pull based collection
  - Time series metrics
  - Service discovery
  - Reliability data

Traces with Tempo

- Tempo stores distributed traces emitted by applications. Traces show how requests move across services.
  - Request flows
  - Latency breakdown
  - Service dependencies
  - Distributed debugging

Alloy as the Collection Agent

- Alloy runs as a cluster wide agent. It collects metrics and forwards telemetry.
  - Metrics scraping
  - Service monitors
  - Unified agent
  - Kubernetes native

Automatic Telemetry Collection

- Applications do not need custom configuration to be observable. The platform handles collection automatically.
  - Logs to Loki
  - Metrics via Alloy
  - Traces to Tempo
  - Minimal app changes

Grafana as the Single View

- Grafana connects all telemetry signals together. Teams investigate issues from one interface.
  - Metrics dashboards
  - Log exploration
  - Trace views
  - Correlation

Lab Focus

- This lab focuses on environment setup rather than query syntax. You will explore a working production style monitoring stack.
  - Infrastructure first
  - Observability platform
  - Microservice ready
  - Hands on exploration

MCQ

- Why do observability tools thrive in microservice environments?
  - Microservices reduce logging needs
  - Distributed systems increase complexity
  - Metrics replace application logic
  - Containers eliminate failures

Answer

- Why do observability tools thrive in microservice environments?
  - Microservices reduce logging needs
  - **Distributed systems increase complexity**
  - Metrics replace application logic
  - Containers eliminate failures

MCQ

- Which LGTM component is responsible for storing logs?
  - Prometheus
  - Tempo
  - Loki
  - Alloy

Answer

- Which LGTM component is responsible for storing logs?
  - Prometheus
  - Tempo
  - **Loki**
  - Alloy

MCQ

- How does Prometheus collect metrics in Kubernetes?
  - Applications push metrics to Prometheus
  - Prometheus scrapes metrics endpoints
  - Metrics are written to logs
  - Grafana polls services directly

Answer

- How does Prometheus collect metrics in Kubernetes?
  - Applications push metrics to Prometheus
  - **Prometheus scrapes metrics endpoints**
  - Metrics are written to logs
  - Grafana polls services directly

MCQ

- What role does Alloy play in the monitoring stack?
  - It visualizes telemetry
  - It stores logs
  - It collects and forwards telemetry
  - It replaces Prometheus

Answer

- What role does Alloy play in the monitoring stack?
  - It visualizes telemetry
  - It stores logs
  - **It collects and forwards telemetry**
  - It replaces Prometheus

MCQ

- Why is Grafana described as a single pane of glass?
  - It replaces all monitoring tools
  - It stores telemetry data
  - It unifies metrics, logs, and traces
  - It runs inside applications

Answer

- Why is Grafana described as a single pane of glass?
  - It replaces all monitoring tools
  - It stores telemetry data
  - **It unifies metrics, logs, and traces**
  - It runs inside applications