Unit-5
Prometheus, Grafana and Selenium

Prometheus

Prometheus is an open-source, powerful system monitoring, Alerting, and performance management tool. Originally, it is developed by SoundCloud, But at this time it is managed by Cloud Native Computing Foundation (CNCF) like kubernates.

Features of Prometheus

Time-series Database

Prometheus stores all data as time-series data or metrics. which means it tracks how values change over time, all data is associated with a timestamp, metric name, and labels.

Multi-dimensional Data Model

Prometheus organizes data using a multi-dimensional system with key-value pairs called labels. This makes it highly flexible for querying.

Pull-Based Data Collection

Prometheus actively pulls metrics from configured endpoints using HTTP, rather than relying on agents to push data.

PromQL: (Prometheus query language)

Prometheus uses its own powerful and expressive query language called PromQL for querying collected metrics.

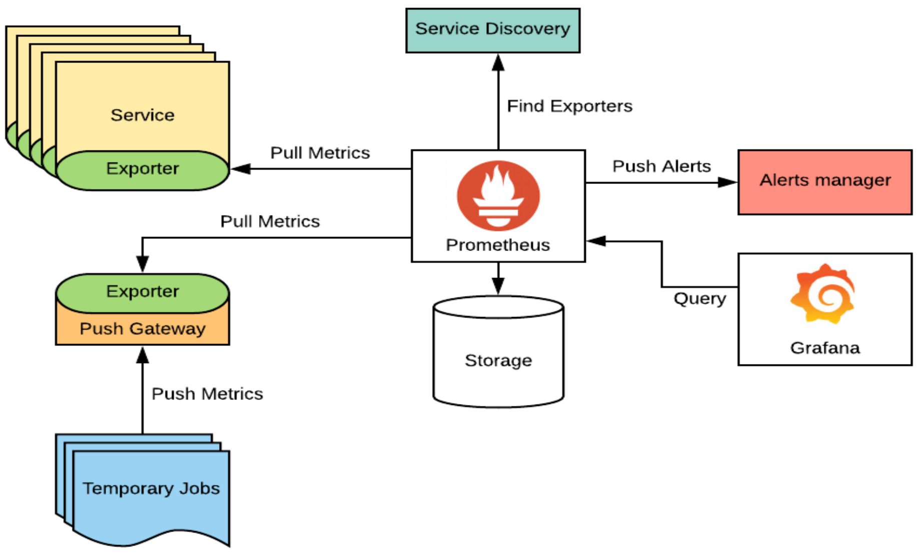
Standalone Architecture

Prometheus operates independently, without dependencies on external storage systems.

Alerting System

Prometheus integrates with alerting tools to notify users about critical conditions.

Prometheus Architecture



Why Learn Prometheus

1. Comprehensive Monitoring and Alerting

Prometheus specializes in collecting and analyzing time-series data, making it invaluable for:

- Monitoring the health and performance of applications, servers, and networks.
- Setting up alerting systems to notify you about potential issues before they impact users.

2. PromQL – A Powerful Query Language

Prometheus Query Language (PromQL) allows for:

- Writing highly flexible and detailed queries to extract metrics.
- Generating precise insights, which can be visualized using tools like Grafana.

3. Open-Source and Cost-Efficient

- Prometheus is completely free and open-source, which means it's accessible to everyone.
- Organizations of all sizes adopt Prometheus to save costs while achieving enterprise-grade monitoring capabilities.

4. Versatility and Custom Metrics

With Prometheus, you can monitor not just infrastructure metrics (like CPU, memory, and disk) but also custom application metrics. Developers can instrument their code to track key metrics like:

- API latency.
- Number of user requests.
- Error rates.

5. Career Growth Opportunities

- Expertise in Prometheus is in high demand across DevOps, SRE, and IT operations roles.
- By learning Prometheus, you not only develop practical monitoring skills but also position yourself as a critical asset in maintaining reliable systems.

6. Integration with Tools Like Grafana

Prometheus works effortlessly with visualization tools like Grafana to create dashboards. This combination makes it easier to:

- Spot trends.
- Share insights with stakeholders.
- Debug production issues visually.

Infrastructure monitoring in Prometheus

Infrastructure monitoring in Prometheus involves collecting, analyzing, and visualizing metrics from servers, virtual machines, containers, and other system resources to ensure optimal performance and identify potential issues in real time.

Why Infrastructure Monitoring Matters:

We know that, System Infrastructure is the backbone of any application. So Monitoring helps us to:

- Track system health (e.g., CPU, memory, disk usage).
- Predict and prevent failures.
- Optimize resource usage.
- Diagnose issues quickly and ensure high availability of services.

Prometheus provides a powerful solution for this by collecting time-series data from infrastructure components and making it available for analysis through querying and alerting.

How Prometheus Monitors Infrastructure

We know that prometheus is a System Monitoring and Alerting tool. So It collects data from system components like servers, containers, and network devices using exporters, which expose metrics in a format Prometheus can scrape via HTTP. Key exporters include Node Exporter for hardware metrics (CPU, memory, disk, network) and cAdvisor for container monitoring.
The collected metrics are stored in Prometheus' time-series database, organized with labels and timestamps for flexible querying through PromQL. After that, Metrics can be visualized using dashboards created in tools like Grafana, that can ensure infrastructure reliability, scalability, and performance and improe to solve real time problem

Alerting and Alert Receiver in Prometheus

Alerting

Alerting in Prometheus enables you to define conditions based on the metrics collected, and when these conditions are met, alerts are triggered. These alerts notify users or systems so that they can take timely actions to resolve potential issues before they escalate.

How Alerting Works in Prometheus:

Alerting Rules

- Alerts are created using alerting rules in Prometheus’ configuration file (prometheus.yml) or separate rule files.
- Each rule specifies a condition, an evaluation interval, and labels for the alert.

Alerting YAML File Structure

```
groups:
  - name: example-alert
    rules:
      - alert: HighCPUUsage
        expr: rate(node_cpu_seconds_total{job="node-exporter"}[5m]) > 0.85
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "High CPU usage detected"
          description: "CPU usage is above 85% for more than 2 minutes."
```

Alert manager

The Alert manager is a companion tool to Prometheus that handles alerts and manages notification delivery to receivers.

Alert Receivers

Alert receivers are the endpoints that Alertmanager sends notifications to. Common types of receivers include:

Email:
Sends alerts as email notifications.

Messaging Services:
Integrates with platforms like Slack, Teams, or PagerDuty for real-time communication.

Webhook:
Sends alerts to custom HTTP endpoints for integration with other systems.

SMS or Phone Calls:
Integrates with third-party tools to deliver alerts via SMS or automated phone calls.

Grafana

Grafana

Grafana is an open-source data analytics and data visualization tool, that is designed for monitoring and managing system metrics data, It also used for creating interactive dashboards, exploring data, and observing trends across multiple sources like, Node, Web-based source and medical machines.

Features of Grafana

- Grafana provide Interactive Dashboard and it allows us to design and customized with chat, graph and other visualization for real-time monitoring of metrics.
- Grafana Provide different types of Data source integration. It means we can connect to a wide variety of data-source like. Prometheus, MySQL, SqlLite, InfluxDB, Elasticsearch etc..
- Grafana provides an alerting system. You can set rule-based alerts on dashboards, and when the specified conditions are met, it will send notifications via SMS, email, Slack, and other channels.
- Grafana provide intuitive query editor where we can write any SQL query like, PromQL, MySQL etc for retrieving and transforming data from sources.
- Grafana Provide Predefine templates and plugins, using this we can Enhance the Functionality of Grafana for data visualization.

Creating Grafana Dashboard

Creating a Grafana dashboard helps us to visualize and moniter the data from various data source like MySQL, SQL-Lite etc.

We have some steps for creating a Grafana Dashboard.

Step-1: First Install Grafna in our System or on Cloud using given Command

- sudo apt-get update
- sudo apt-get install -y grafana
- sudo systemctl start grafana-server
- sudo systemctl enable grafana-server

Step-2: Now Open your Grafana tool on given port using localhost:<port> and login using default user_id and password

- ➢ Username: Admin
- ➢ Password: Admin

Step-3: After that Add data source using given sub-steps (Note: You can add various data source in Grafana).

To add a data source:

- ➢ Navigate to Configuration → Data Sources.
- ➢ Click Add data source and select a database (e.g., Prometheus).
- ➢ Provide connection details (e.g., Prometheus URL: http://localhost:9090).
- ➢ Click Save & Test to ensure the data source is successfully added.

Step-4: After adding data source create a Dashboard using dashboard option

- ➢ Go to Dashboards → Create → New Dashboard.
- ➢ Click Add a new panel to start creating visualizations.

Step-5: After creating a dashboard and panel you have to configure the panel according to data-source and here you can add and configure data visualization type like. Graph, table, heat map etc. and you can also set some adjust setting like. Title, Legend, Time Range, Alert etc..

(Note: You can Add Multiple Panels)

Step-6: After configure the panel you can set a alert system and threshold **(Optional)**.

Step-7: Now Save a Dashboard and share with your team.

Grafana API and Auto Healing

Grafana API

Grafana provides a robust REST API that allows programmatic access to its functionalities, including creating dashboards, managing users, querying data sources, and automating tasks.

Features of Grafana API
1. Dashboard Management
Create, update, delete, and retrieve dashboards via API calls.

2. User & Authentication Management
Manage users, roles, and authentication tokens.

3. Data Source Integration
Add, update, and remove data sources programmatically.

4. Alerts & Notification Channels
Configure alerts and set up notification channels via API.

5. Organizations & Permissions
Organize Grafana users and configure permissions dynamically.

Note: Grafana API uses API tokens for authentication, ensuring secure and controlled access to its endpoints.

Some API Calls Command in Grafana

1. **Create a dashboard**
   ```
   curl -X POST http://localhost:3000/api/dashboards/db -H "Content-Type: application/json" -d @dashboard.json
   ```
2. **Get all dashboards**
   ```
   curl -X GET http://localhost:3000/api/search
   ```
3. **Retrieve metrics from Prometheus**
   ```
   curl -X GET http://localhost:9090/api/v1/query?query=node_cpu_seconds_total
   ```

Auto Healing in Grafana

Grafana Doesn't have any auto healing system or capabilities, it can be integrated with monitoring tools (e.g., Prometheus, Kubernetes) to trigger self-healing mechanisms, and these tools provide a auto-healing facilities to grafana.

Auto-healing Mechanism in Monitoring System

1. Detection

- Grafana monitors infrastructure for failures using real-time metrics.
- Prometheus alerts Grafana when a system metric exceeds defined thresholds.

2. Alerting & Incident Response

- Grafana alerts are sent to Alertmanager or other notification tools.
- Alerts trigger automated workflows to handle incidents.

3. Automated Remediation (Self-Healing)

- Kubernetes can automatically restart failing pods when an issue is detected.
- Infrastructure automation tools like Ansible or Terraform can respond to alerts by adjusting configurations.

Selenium

Selenium

Selenium is an open-source testing tools, that is used to automate web browsers. the primarily use of selenium is automated testing of web applications. However, it's also used for web scraping and browser automation tasks.

Features of Selenium

1. Selenium is used for Cross-Browser Compatibility: e.g. Chrome, Firefox, Edge, safari etc

2. Cross-Platform Support: e.g. Windows, Linux, MacOS.

3. Supports Many Programming Languages for Script: e.g. Java, JavaScript, Python, Ruby, C# etc..

4. Selenium can integrate with Multiple Framework: e.g. Junit, TestNG(java). PyTest(python). NUnit(C#)

5. Selenium can Supports Headless Browser Testing: e.g. CI/CD Pipeline, testing in environment without display.

6. Selenium Support Mobile Testing: e.g. Appium, Selendroid

7. Selenium can integrate with Cloud-Based Test Execution: e.g. Sauce Labs, BrowserStack, LambdaTest

8. Selenium is Open Source(Free) and Strong Community Driven

Testing Backend Integration points

Testing backend integration points is essential to ensure that different system components communicate and function correctly. It helps identify issues related to APIs, databases, third-party services, and internal system dependencies.

Backend Integration Testing
Backend integration testing involves verifying data flow, interactions, and communication between different services, APIs, and databases in a system. It ensures that backend components work together seamlessly without errors.

Backend Integration Testing Points

1. API Testing:

- Ensures that REST, GraphQL, or SOAP APIs function correctly.
- Verifies request/response formats, status codes, headers, and authentication mechanisms.

2. Database Testing:

- Validates data storage, retrieval, updates, transactions, and integrity.
- Checks for performance issues in queries and indexing.

3. Message Queues & Event Systems:

- Tests communication via Kafka, RabbitMQ, AWS SQS to ensure messages are correctly published and consumed.

4. Third-Party Services:

- Ensures integration with payment gateways, authentication providers, analytics tools, etc.

5. Authentication & Security:

- Verifies login, token generation, encryption, and authorization policies.

Types of Backend Integration Tests

1. Unit Tests

- Focus on individual components using mocking or stubbing.
- Example: Testing a function that retrieves data from a database.

2. Functional Tests

- Validate APIs and backend logic against expected outcomes.
- Example: Sending requests and checking JSON responses.

3. Load & Performance Testing

- Tests how backend systems handle high volumes of requests.
- Tools like JMeter, Locust, or k6 measure response times and scalability.

4. End-to-End Tests

- Validate interactions across multiple backend services.
- Example: Checking database updates after an API call.

5. Security Tests

- Ensure APIs and databases are protected from vulnerabilities (SQL injection, unauthorized access).
-

Test-Driven Development

Test-Driven Development (TDD) is a software development methodology where tests are written before the actual implementation of the code. The idea is to create tests that define the expected behaviour of a function or feature and then write the minimal amount of code required to pass those tests.

Test-Driven Development is based on a simple cycle called Red-Green-Refactor:

- **Red**→ Write a failing test for a function that does not exist yet.
- **Green**→ Implement the function minimally to make the test pass.
- **Refactor**→ Optimize the code without changing its functionality.

This iterative approach ensures high code quality, bug prevention, and continuous validation.

Steps that are exist in Test-Driven Development

- Write a test. e.g. Like function for add two number
- Run the test (Failing Stage - RED). It will check add function is exist or not.
- Write minimal code to pass the test (GREEN)
- Run the test again
- Refactor the code
- Repeat for other functions

Programming Languages Tools for Test-Driven Development

- **Python** → pytest, unittest
- **JavaScript** → Jest, Mocha
- **Java** → JUnit
- **C#** → xUnit, NUnit
- **Go** → Testify

Benefits of Test-Driven Development

- **Early Bug Detection→** Writing tests first helps catch problems before they reach production.
- **Improved Code Quality→** Forces developers to write cleaner, more modular code.
- **Faster Debugging→** When a test fails, it's clear which part of the code is broken.
- **Encourages Better Design→** Encourages writing small, testable, and reusable functions.
- **Confidence in Changes→** Code modifications can be done safely without fear of breaking functionality.

REPL Driven Development

REPL-Driven Development (RDD) is an interactive programming approach where developers write, test, and refine code in a Read-Eval-Print Loop (REPL) environment. This method allows for immediate feedback, making it a highly efficient way to develop software.

REPL stands for

- **Read →** Accepts user input (code snippets).
- **Evaluate →** Executes the code in a live runtime.
- **Print →** Displays the result instantly.
- **Loop →** Repeats the process, allowing continuous interaction.

REPL environments exist for many languages, like.

- Python (python shell)
- JavaScript (node REPL)
- Ruby (irb)
- Clojure (clojure REPL)
- Scala (scala interactive shell)

REPL-Driven Development Works and Workflow

REPL-Driven Development focuses on incremental coding, where developers experiment with functions, debug issues, and refine logic in real time before committing changes to files.

Basic Workflow

- **Start the REPL→** Open an interactive shell for the programming language.
- **Write a small function→** Immediately execute to check the output.
- **Iterate on improvements→** Modify the function and test again.
- **Debug in real-time→** Identify mistakes and fix them instantly.
- **Save finalized code→** Store well-tested logic into actual source files.

Benefits of REPL-Driven Development

- **Rapid Prototyping**→ Quickly try ideas and refine them before writing full applications.
- **Immediate Feedback**→ Instantly see errors or outputs, reducing debugging time.
- **Interactive Debugging**→ Modify functions while the program is running.
- **Better Understanding of Language Features**→ Experiment with syntax and libraries in real time.