Observability

What is Observability?

Observability helps us understand the internal state of a system or application in detail. It provides insights into applications, infrastructure, and network details, acting like a "CCTV" that monitors everything and delivers information through tools like Prometheus, Grafana, and others. Observability not only shows the current state of an application but also helps determine why it is in that state and how to fix issues effectively.

Use Cases of Observability:

1. Monitoring disk utilization for a specific node.
2. Checking CPU utilization of a system or application.
3. Measuring memory usage of an application or node.
4. Analyzing HTTP requests to see how many succeeded or failed out of a given set (e.g., 100 requests).

The Three Pillars of Observability:

To understand why certain events occur, we need to explore the three main pillars of observability:

1. Metrics
   Metrics provide a numerical representation of an application's state, showing what is currently happening inside it. For example, metrics can display CPU utilization, memory usage, or the number of requests processed. They also include historical data, allowing us to examine the application's state at a specific time and date. Historical data of events to understand the health of the system.

2. Logs
   Logs are detailed records of system events, helping us understand why a particular state occurred. They provide granular details about what happened during a specific event or error.

3. Traces
   Traces illustrate how requests flow through the application, showing the path taken and pinpointing where issues arise. For example, they can track the flow of a request from the client to the backend services and identify bottlenecks or failures in the process.

Example:

Suppose you want to investigate what happened yesterday at 10:00 AM with a specific request:

- Metrics: You can check metrics to analyze the resource utilization (e.g., CPU, memory) during that time.
- Logs: You can examine the logs to find out why the application was in a particular state and identify the exact issue.
- Traces: You can trace the request flow to see the sequence of interactions and pinpoint where the issue occurred. For instance:
  Client → Load Balancer → Frontend → Backend → Database

Monitoring vs. Observability

- **Monitoring**
  Monitoring involves setting up metrics, alerts, and dashboards to track the performance and health of an application or system. It focuses on predefined metrics and conditions, such as CPU utilization, memory usage, or disk space. Monitoring is primarily reactive, helping you detect and respond to known issues based on pre-configured thresholds.
- **Observability**
  Observability is a broader concept that encompasses everything about an

application. It not only collects metrics but also includes logs, traces, and other data to provide a deep understanding of the system's internal state. Observability is proactive, enabling you to investigate and resolve unknown issues, uncover root causes, and understand the system's behavior in depth.

In summary:

- Monitoring helps you track and respond to known issues.
- Observability helps you understand and diagnose both known and unknown issues comprehensively.

Metrics Use Cases

Metrics provide numerical data that helps monitor and analyze the performance and health of applications, infrastructure, and systems. Here are some key use cases:

1. **CPU Utilization**: Track the CPU usage of a node or an EC2 instance in a cluster.
2. **Memory Usage**: Monitor memory consumption of an AWS virtual machine or instance.
3. **Pod Status**: Check for specific Kubernetes pod statuses, such as "CrashLoopBackOff."
4. **Replica Count**: Observe the number of replicas running at a particular time.
5. **HTTP Requests**: Monitor how many HTTP requests are received by an application.
6. **New User Signups**: Count how many new users signed up during a specific period.
7. **Signup Time**: Record the exact time when a user signed up.

Data Collection Mechanisms

Metrics are collected using two primary mechanisms:

- **Push Mechanism**
  - Systems or applications actively send (push) metrics data to a central monitoring system.
- **Pull Mechanism (Scraping)**
  - The monitoring system periodically retrieves (pulls) metrics data from applications or services. Tools like Prometheus use this mechanism to scrape metrics endpoints.

Monitoring and Alerts

Monitoring systems use metrics to create alerts and notify teams when certain conditions are met.

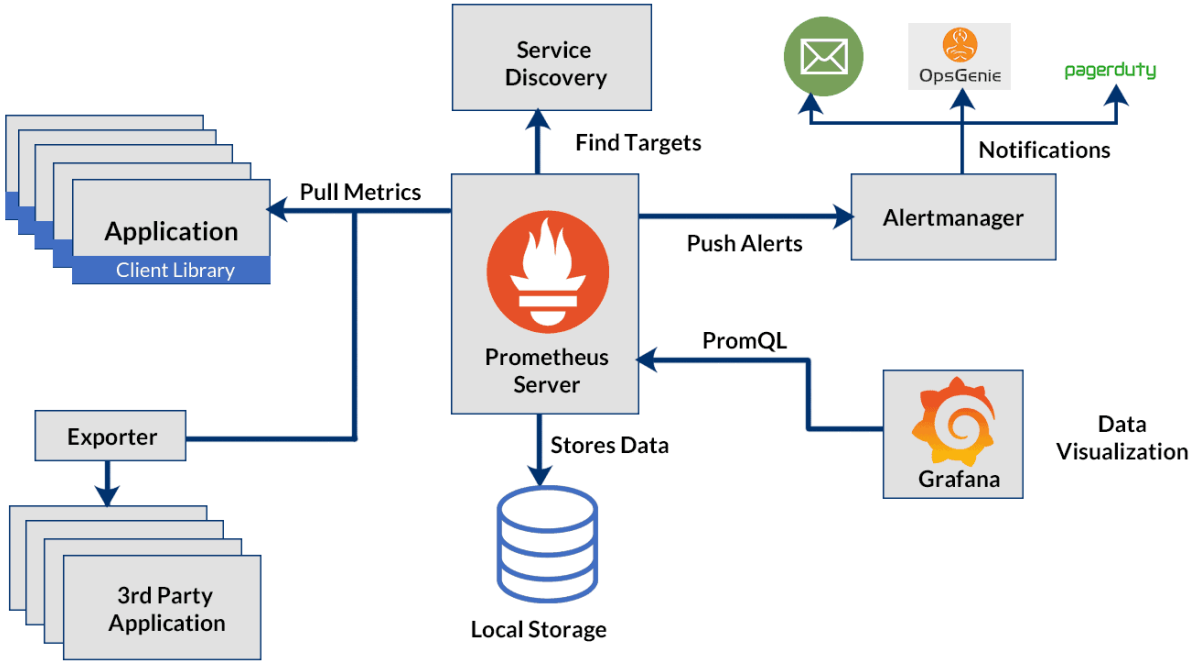
Example Alerts:

- CPU utilization exceeds 70%.
- Disk utilization is above 60%.
- Application experiences latency more than 30 times in a day.

Alert Notifications:
When an alert condition is met, the monitoring system sends notifications to channels like Slack, Gmail, or other messaging tools, allowing teams to respond promptly.

Prometheus

Prometheus is an open-source monitoring and alerting tool designed for collecting and analyzing metrics from various sources.



How Prometheus Works

- **Pull Metrics**
  Prometheus collects metrics by pulling (scraping) data from:
  - **Exporters**: Specialized tools that expose metrics from systems like databases, servers, or hardware.
  - **Endpoints**: Applications or services expose metrics over HTTP endpoints.
  - **Kube State Metrics**: Kubernetes-specific metrics, such as pod states, node conditions, and deployment statuses.

- **Storing and Querying Data**
  The scraped metrics are stored in Prometheus's time-series database. You can query this data using **PromQL**, Prometheus's query language.
- **Dashboard Representation**
  While Prometheus itself has a basic UI, it is often integrated with tools like **Grafana** to create more advanced and visually appealing dashboards. These dashboards help visualize metrics data for analysis.

- **Alerting**
  - Prometheus uses **Alertmanager** to define and manage alerts based on specific conditions (e.g., CPU utilization > 70%).
  - Alertmanager can send notifications to various channels, such as Slack, email, PagerDuty, or others, ensuring timely responses to issues.

Example Workflow:

- Prometheus scrapes metrics from a Kubernetes pod's metrics endpoint.
- The data is visualized in a Grafana dashboard, showing trends like CPU or memory usage.
- An alert is configured in Alertmanager to notify the team via Slack if CPU utilization exceeds 80%.

Conclusion

Observability is essential for managing and optimizing modern, complex systems. By leveraging metrics, logs, and traces, it provides engineers with the tools to gain deeper insights into system performance and behavior.
Thank You for today tomorrow i will cover how to setup prometheus and grafana for cluster.