Prometheus:

- Prometheus is a highly popular open-source monitoring and alerting system originally developed at SoundCloud.
- It is now a standalone project maintained by the Cloud Native Computing Foundation (CNCF).
- It's known for its flexible query language, efficient storage, and powerful dimensional data model.
- Prometheus is designed for reliability, scalability, and ease of use, making it a widely adopted solution for monitoring containerized and cloud-native applications.
- Prometheus collects and stores its metrics as time series data, i.e. metrics information is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels.

Features:

- ✓ **Multi-dimensional Data Model:**
  - Prometheus stores time-series data with labels, allowing flexible querying and aggregation based on various dimensions.
- ✓ **Powerful Query Language:**
  - PromQL (Prometheus Query Language) enables users to query and analyze time-series data efficiently, allowing for complex aggregation, filtering, and mathematical operations.
- ✓ **Scalable and Resilient:**
  - Prometheus is designed to be highly scalable, supporting horizontal scalability through sharding and federation.
  - It can handle large-scale deployments with thousands of nodes and millions of time series.
- ✓ **Dynamic Service Discovery:**
  - Prometheus integrates with various service discovery mechanisms to automatically discover and monitor new instances of services as they are deployed or removed.
- ✓ **Alerting and Alert Manager:**
  - Prometheus supports alerting based on defined rules and thresholds.
  - The Alertmanager component handles alerts by deduplicating, grouping, and routing them to the appropriate channels (e.g., email, Slack).
- ✓ **Pull-based Metrics Collection:**
  - Prometheus follows a pull-based model where it scrapes metrics from configured targets (applications, services, or infrastructure components) at regular intervals.
  - This approach simplifies setup and reduces overhead on monitored targets.
- ✓ **Rich Ecosystem:**
  - Prometheus has a rich ecosystem of exporters, integrations, and libraries, making it easy to monitor various systems, including cloud platforms, databases, message brokers, and more.

- ✓ **Grafana Integration:**
    - Prometheus integrates seamlessly with Grafana, a popular visualization tool, allowing users to create rich dashboards and visualizations based on Prometheus metrics.

Prometheus Working:

- **Scraping:** Prometheus periodically "scrapes" metrics endpoints exposed by services and systems.
- **Storage:** Scraped data is stored locally in a time series database.
- **Querying:** You query and visualize metrics via PromQL in the expression browser or Grafana.
- **Alerting:** Alerting rules are evaluated, triggering notifications as needed.

Metrics:

- Metrics are numerical measurements in layperson terms.
- The term time series refers to the recording of changes over time.
- What users want to measure differs from application to application.
- For a web server, it could be request times; for a database, it could be the number of active connections or active queries, and so on.
- Metrics play an important role in understanding why your application is working in a certain way.
- Let's assume you are running a web application and discover that it is slow. To learn what is happening with your application, you will need some information. For example, when the number of requests is high, the application may become slow. If you have the request count metric, you can determine the cause and increase the number of servers to handle the load.

Metrics Types:

1. Counter – Observes number of events happened (only increased values)
2. Gauge - Observes number of events happened (both increased and decreased values)
3. Summary - How long event should took and how big event was.
    a. count: shows number of time event observed.
    b. sum: shows sum of time taken by that event.
4. Histogram

Components:

The Prometheus ecosystem consists of multiple components, many of which are optional:

- The main Prometheus Server which scrapes and stores time series data.
- Client Libraries for instrumenting application code.
- A push gateway for supporting short-lived jobs.
- An AlertManager to handle alerts.
- Various support tools

Grafana:

- Grafana is an open-source analytics and monitoring platform.
- Grafana is a powerful open-source software that enables you to create beautiful and informative dashboards for visualizing data from various sources.
- It allows you to query, visualize, alert on, and understand your metrics no matter where they are stored.
- Grafana is commonly used for visualizing time series data for infrastructure and application analytics, but it can also be used in other domains like business intelligence.

Features:

- **Data Source Agnostic:**
  - One of the standout features of Grafana is its ability to connect to a wide range of data sources.
  - This includes popular time-series databases like Prometheus, InfluxDB, Graphite, and Elasticsearch, but also relational databases such as MySQL, PostgreSQL, Microsoft SQL Server, and cloud-based data sources like AWS CloudWatch and Google Cloud Monitoring.
  - This flexibility enables users to visualize and analyze metrics and logs from various sources in a unified dashboard.

- **Rich Visualization Options:**
  - Grafana offers a plethora of visualization options to cater to diverse monitoring and analytics needs.
  - Users can create visually appealing and informative graphs, including line charts, bar charts, histograms, heatmaps, scatter plots, and more.

- **Dashboarding:**
  - Dashboards in Grafana serve as centralized hubs for monitoring and analysis.
  - Users can create dashboards composed of multiple panels, each displaying different metrics or visualizations.
  - Panels can be arranged and resized according to preference, and Grafana provides features for easy dashboard navigation, organization, and sharing.

- **Alerting:**
  - Monitoring and alerting are essential components of any observability solution, and Grafana provides robust alerting capabilities.
  - Users can define alert rules based on query results from data sources, specifying conditions such as thresholds, time periods, and aggregation functions.
  - When an alert condition is met, Grafana can trigger notifications via various channels, including email, Slack, PagerDuty, and more.

- **Plugin System:**
  - Grafana's plugin system allows users to extend its functionality beyond built-in features.
  - Users can install community-contributed plugins to add support for additional data sources, visualization types, authentication methods, and more.

- **Community and Ecosystem:**
  - Grafana boasts a large and active community of users, developers, and contributors.
  - The community provides support through forums, mailing lists, and chat channels, helping users troubleshoot issues, share best practices, and exchange knowledge.

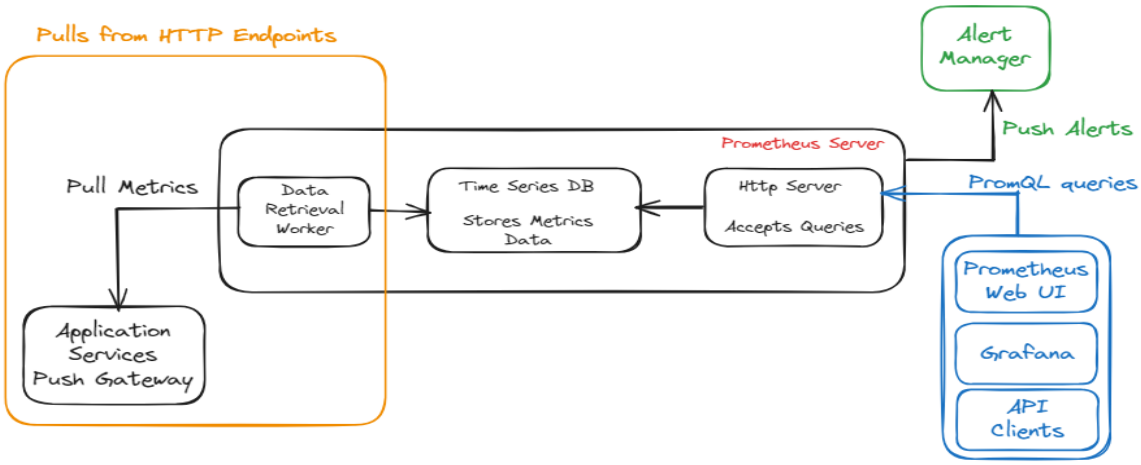

Fig. Prometheus and Grafana Architecture

In prometheus.yml file, you will expose application endpoint in scrape_configs by specifying, job_name, metrics_path, scrape_interval and static_configs

Steps to Run Prometheus and Grafana with Spring Boot:

1. **Using Docker:**
   1. Download docker desktop in local environment and start docker service.
   2. Create Spring Boot application with required dependencies,

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
        <groupId>io.micrometer</groupId>
        <artifactId>micrometer-registry-prometheus</artifactId>
```

```
</dependency>
<dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

3. Add following configurations in application.properties or yml file,
   management.endpoints.web.exposure.include=*
   management.endpoint.health.show-details=always
4. Create Restcontroller class as per your requirements.
5. Run and check the actuator endpoint,
   http://localhost:8080/actuator
6. In end points you will find - http://localhost:8080/actuator/prometheus
7. Create prometheus.yml file in resource folder and define Prometheus related configurations,
   prometheus.yml

```
# my global config
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'
    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.
    static_configs:
    - targets: ['127.0.0.1:9090']

  - job_name: 'spring-actuator'
    metrics_path: '/actuator/prometheus'
```

```
scrape_interval: 5s
static_configs:
- targets: ['192.168.0.206:8080']
  #- targets: ['localhost:8080']
```

8. **Set up Prometheus server,**
   Start up the Prometheus server using by running docker image, Open CMD and enter below commands,
   $ docker run -p 9090:9090 -v /your_path/41_sb_prometheus_grafana/src/main/resources/prometheus.yml prom/Prometheus
9. Go to http://192.168.0.206:9090 or http://localhost:9090 and perform operations.
10. **Setup Grafana Server,**
    Startup Grafana server to provide more beautiful UI by using docker image,
    $ docker run -d --name grafana_server -p 3000:3000 grafana/Grafana
11. Go to http://localhost:3000, Login with username: admin and password:admin
12. Add datasource: select prometheus and provide prometheus url (http://192.168.0.206:9090 or http://localhost:9090) and click Save & Test If success, Datasource will be created.
13. Create Dashboard for Prometheus metrics and explore other things.
14. Perform other operations and explore.

2. **Using Locally Downloaded Prometheus and Grafana Software's:**

Prometheus Setup:

1. Download Prometheus Zip file from official website, https://prometheus.io/download/
2. Extract Zip file, edit prometheus.yml file to expose application endpoint, and save it.
3. Open CMD and run prometheus.exe file or double click on it, the server will start on port 9090 by default.
4. Go to http://localhost:9090 and explore the things.

Grafana Setup:

5. Download Grafana zip file from official website, https://grafana.com/grafana/download
6. Extract zip file, go to bin folder, open CMD and execute grafana-server.exe file or double click on that file.
7. Grafana serve will up and running on port 3000 by default.
8. Go to http://localhost:3000 and explore the things.

With Spring Boot:

9. Follow the same steps of Using Docker from 2-6 and 11-14