Prometheus Advanced Overview

Prometheus is a leading open-source system monitoring and alerting toolkit. It provides powerful features for monitoring your infrastructure and applications, including advanced monitoring capabilities, various exporters for collecting data from external systems, and integrated alerting through the **Alertmanager**. This guide will cover advanced Prometheus topics: advanced monitoring, exporters, and Alertmanager, followed by a step-by-step setup guide.

---

1. Advanced Monitoring with Prometheus

Prometheus offers highly customizable and powerful monitoring capabilities.

a. Time-Series Data Model Prometheus stores data as time-series metrics with labels, which can be dynamically queried. Advanced usage includes:

- **Relabeling**: Modify or create labels before storing data to better categorize or filter it.
- **Federation**: Prometheus can scrape metrics from multiple Prometheus servers to create a global view.
- **Remote Write/Read**: Offload or read back metrics to/from long-term storage like InfluxDB, Thanos, or Cortex.

b. PromQL (Prometheus Query Language) PromQL is used to query metrics in Prometheus. Advanced queries allow for:

- Aggregations (e.g., `sum()`, `avg()`, `max()`)
- Joins (e.g., combining multiple metrics for more detailed insights)
- Rate calculations for time-based queries (e.g., `rate()`, `irate()`). Example:

promql
```
sum(rate(http_requests_total[5m])) by (status_code)
```

This query calculates the request rate for different HTTP status codes over the last 5 minutes.

c. Rules and Recording Rules Prometheus supports the creation of **alerting rules** and **recording rules**.

- **Alerting Rules**: Generate alerts based on metric thresholds.
- **Recording Rules**: Precompute expensive or frequently used queries to save processing time. Example of a recording rule to store average CPU usage:

yaml
```yaml
groups:
  - name: example
    rules:
      - record: job:cpu_usage:avg
        expr: avg(rate(cpu_usage_seconds_total[5m])) by (job)
```

---

2. Prometheus Exporters

Exporters are crucial for collecting metrics from external systems that Prometheus cannot scrape natively. They expose these metrics in a format that Prometheus understands (typically over HTTP).

a. Popular Exporters

- **Node Exporter**: Collects metrics from Linux/Unix systems (e.g., CPU, memory, disk).
- **Blackbox Exporter**: For probing endpoints (HTTP, DNS, TCP, etc.).
- **MySQL/MongoDB Exporter**: For scraping database metrics.
- **CAdvisor**: For collecting container metrics.
- **JMX Exporter**: For monitoring Java applications.

b. Writing Custom Exporters You can write custom exporters using Prometheus libraries for your language of choice (e.g., Go, Python). A custom exporter scrapes metrics from your application and exposes them in the `/metrics` endpoint in Prometheus format.

Example Go code snippet for a custom exporter:

go
```go
package main

import (
    "net/http"
    "github.com/prometheus/client_golang/prometheus"
```

```
    "github.com/prometheus/client_golang/prometheus/promhttp"
)

var requestCount = prometheus.NewCounter(prometheus.CounterOpts{
    Name: "myapp_requests_total",
    Help: "Total number of requests",
})

func main() {
    prometheus.MustRegister(requestCount)
    http.Handle("/metrics", promhttp.Handler())
    http.ListenAndServe(":8080", nil)
}
```

---

3. Alertmanager

Prometheus uses the **Alertmanager** to handle alerts. It manages alert notifications and deduplicates, groups, and routes alerts based on configured policies.

a. Alertmanager Features

- **Routing**: Define routing rules to send alerts to different channels (email, Slack, PagerDuty, etc.).
- **Grouping**: Group similar alerts together to avoid alert flooding.
- **Silencing**: Silence certain alerts during planned maintenance or outages.
- **Inhibition**: Suppress alerts based on the presence of other alerts.

b. Configuring Alertmanager Alertmanager configuration involves defining receivers (e.g., Slack, email) and routing rules. Example of an Alertmanager configuration for email and Slack:

yaml

```
route:
  group_by: ['alertname', 'severity']
  receiver: 'slack'
  routes:
    - match:
```

```
        severity: critical
      receiver: 'email'

receivers:
  - name: 'slack'
    slack_configs:
      - api_url: 'https://hooks.slack.com/services/xxx/yyy/zzz'
        channel: '#alerts'
  - name: 'email'
    email_configs:
      - to: 'alerts@example.com'
        from: 'prometheus@example.com'
        smarthost: 'smtp.example.com:587'
        auth_username: 'user'
        auth_password: 'password'
```

c. Integration with Prometheus In Prometheus, you define alerting rules that trigger alerts based on the collected metrics and notify the Alertmanager. Example alerting rule for CPU usage:

yaml

```
groups:
  - name: example
    rules:
      - alert: HighCPUUsage
        expr: avg(rate(cpu_usage_seconds_total[5m])) by
(instance) > 0.85
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "High CPU usage on instance {{
$labels.instance }}"
          description: "CPU usage is above 85% for more than 2
minutes."
```

Step-by-Step Setup Guide

Prerequisites

- A Linux server (Ubuntu/CentOS) or a Docker environment.
- Basic networking and access to your infrastructure.

Step 1: Install Prometheus

a. Install Prometheus on Linux

1. Download Prometheus:
   bash
   ```
   wget https://github.com/prometheus/prometheus/releases/download/v2.44.0/prometheus-2.44.0.linux-amd64.tar.gz
   tar -xzf prometheus-2.44.0.linux-amd64.tar.gz
   cd prometheus-2.44.0.linux-amd64
   ```

2. Run Prometheus:
   bash
   ```
   ./prometheus --config.file=prometheus.yml
   ```

   Access Prometheus via `http://<your-server-ip>:9090`.

b. Install Prometheus with Docker

bash
```
docker run -d --name=prometheus -p 9090:9090 -v /path/to/prometheus.yml:/etc/prometheus/prometheus.yml prom/prometheus
```

Step 2: Configure Prometheus

1. Modify the `prometheus.yml` file:
   yaml
   ```
   global:
   ```

```
  scrape_interval: 15s

scrape_configs:
  - job_name: 'node_exporter'
    static_configs:
      - targets: ['localhost:9100']
```

2. Add the Prometheus **Alertmanager** configuration:
   yaml
```
alerting:

  alertmanagers:
    - static_configs:
        - targets:
            - 'localhost:9093'

rule_files:

  - 'alert.rules'
```

Step 3: Set Up Exporters

a. Node Exporter for Linux Metrics

1. Download and run Node Exporter:
   bash
```
wget
https://github.com/prometheus/node_exporter/releases/downlo
ad/v1.5.0/node_exporter-1.5.0.linux-amd64.tar.gz

tar -xzf node_exporter-1.5.0.linux-amd64.tar.gz

./node_exporter
```

2. Update Prometheus `prometheus.yml` to scrape Node Exporter:
   yaml
   ```
   scrape_configs:

     - job_name: 'node_exporter'

       static_configs:

         - targets: ['localhost:9100']
   ```

b. Blackbox Exporter Install and configure Blackbox Exporter for HTTP/TCP probing:

Bash

```
   docker run -d --name=blackbox_exporter -p 9115:9115
prom/blackbox-exporter
```

Step 4: Set Up Alertmanager

1. Download and install Alertmanager:
   bash
   ```
   wget
   https://github.com/prometheus/alertmanager/releases/downloa
   d/v0.24.0/alertmanager-0.24.0.linux-amd64.tar.gz

   tar -xzf alertmanager-0.24.0.linux-amd64.tar.gz

   ./alertmanager --config.file=alertmanager.yml
   ```
2. Configure Alertmanager (e.g., Slack, email integration).

Step 5: Create Alerts in Prometheus

1. Define alerting rules in the `alert.rules` file:
   Yaml

```
groups:
  - name: example
    rules:
      - alert: InstanceDown
        expr: up == 0
        for: 5m
        labels:
          severity: page
        annotations:
          summary: "Instance {{ $labels.instance }} down"
```

2. Restart Prometheus to load the new rules:

```
sudo systemctl restart prometheus
```

Conclusion

With advanced monitoring, exporters, and Alertmanager, Prometheus provides a comprehensive monitoring and alerting solution for modern infrastructure. This guide covers the key components and a step-by-step setup to enable these features for your environment.