# Metrics

1. How Prometheus works:

Prometheus is an open-source monitoring and alerting system. Here's a simplified explanation of how it works:

1. Scraping: Prometheus periodically scrapes metrics from configured targets. Each target is a host and port combination where Prometheus expects to find a metrics HTTP endpoint.

 Initially, you configure Prometheus with the addresses of the targets you want to monitor. This configuration is done through a YAML file (prometheus.yml), where you specify the job names and the scrape configs for each job. A scrape config defines the targets to scrape and the frequency of scraping. Each target exposes its metrics at a specific HTTP endpoint, usually /metrics. This endpoint returns a text-based format (like Prometheus exposition format) that Prometheus understands and can ingest. At the configured interval, Prometheus sends HTTP GET requests to the /metrics endpoint of each target. It parses the returned data, storing it in its time-series database for later querying.

1. Storage: Collected metrics are stored in a time-series database. Each time series consists of a metric name and a set of key-value pairs (labels). Metrics are recorded at regular intervals, and each recording is a timestamped value.
2. Querying: Users can query the stored time series data using PromQL, Prometheus's flexible query language. Queries return the latest values for each time series that match the query criteria.
3. Scraping system-level metrics: Prometheus doesn’t collect server metrics without additional components. The Node Exporter is typically used for collecting system-level metrics. The Node Exporter is installed on the target machine like servers, VMs, etc. It runs as a separate process, typically as a system service or daemon. The Node Exporter uses various sources to collect system metrics like from /proc filesystem (on Linux) for CPU, memory, disk, and network stats and sysfs for hardware and system information. By default, Node Exporter exposes metrics on port 9100.
4. Alerting: Prometheus can trigger alerts based on the evaluation of alerting rules. These rules define conditions that, when met, generate alerts. Alerts can be sent to various integrations, including email, Slack, PagerDuty, and others via Alertmanager.
5. To create custom Prometheus alerts and alerting rules for Kubernetes monitoring, you define alert rules in YAML format. These rules specify the conditions that trigger an alert and the labels and annotations attached to it. Here's an example of an alert rule that triggers when the CPU usage of any node exceeds 80% for more than 5 minutes:

groups:

- name: node-alerts

rules:

- alert: HighCPUUsage

expr: 100 - (avg by (instance) (irate(node\_cpu\_seconds\_total{mode="idle"}[5m])) \* 100) > 80

for: 5m

labels:

severity: warning

annotations:

summary: High CPU usage on {{ $labels.instance }}

description: The CPU usage of {{ $labels.instance }} is above 80% for more than 5 minutes.

To apply this alert rule to your Kubernetes cluster, you replace the default ConfigMap with your custom one that contains the alert rule file. Then, you reload Prometheus with the new alert rules by deleting the existing Prometheus pod, which triggers a new pod creation with the updated rules. Finally, you can test your alert rule by artificially increasing the CPU usage of a node.

1. To properly show the usage trend of an application metric that is a counter in Grafana, you typically want to use the rate function. Here's an example query:

rate(http\_requests\_total[5m])

This would show the rate of HTTP requests per second over 5-minute intervals, which is often more useful for visualizing trends than the raw counter value.