Prometheus Deep Dive

Course Navigation

- **Introduction to Prometheus** Section 1
- **Prometheus Data** Section 2
- **Visualization** Section 3
- **Collecting Metrics** Section 4
- **Alerting** Section 5
- **Advanced Concepts** Section 6
- **Conclusion**

Prometheus Deep Dive

Prometheus – An open-source monitoring solution

You can find Prometheus documentation at **prometheus.io/docs**
We will cover:

- The Basics
- Installation and Configuration
- Prometheus Data
- Visualization
- Collecting Metrics
- Alerting
- Advanced Concepts
- Security
- Client Libraries

Prometheus is an open-source **monitoring** and **alerting** tool. It collects data about applications and systems and allows you to **visualize** the data and issue **alerts** based on the data.

Monitoring is an important component of DevOps automation. To manage a robust and complex infrastructure, you need to be able to quickly and easily understand what is happening in your systems!

High-Level Use Cases

- **Metric collection** - Collect important metrics about your systems and applications in one place.
- **Visualization** - Build dashboards that provide an overview of the health of your systems.
- **Alerting** - Receive an email when something is broken.

Prometheus Background

- **Language** - Prometheus is primarily written in **Go**. Some components are written in **Java**, **Python**, and **Ruby**.

- **License** - Prometheus uses the open-source **Apache 2.0** license.

- **History** - Prometheus development was started by **Matt T. Proud** and **Julius Volz**. It was initially sponsored by **SoundCloud**. Today, it is a fully open-source project maintained by many individuals and organizations.

- **Website** - More information and full documentation can be found at **prometheus.io**.

Prometheus Architecture

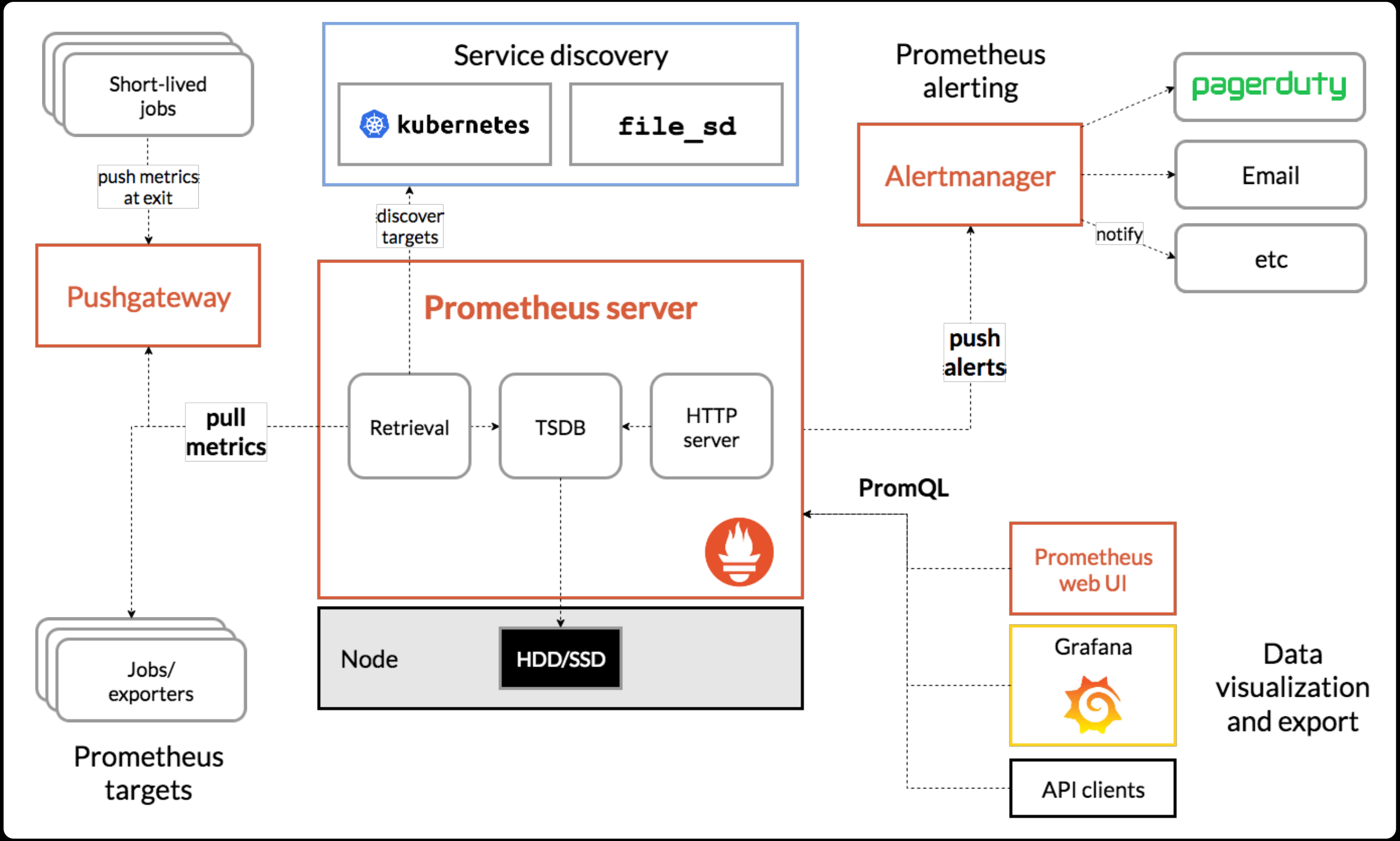
Prometheus consists of a variety of components.

The two most basic components of a Prometheus system are:

- **Prometheus server** - A central server that gathers metrics and makes them available.
- **Exporters** - Agents that expose data about systems and applications for collection by the Prometheus server.

Prometheus Pull Model

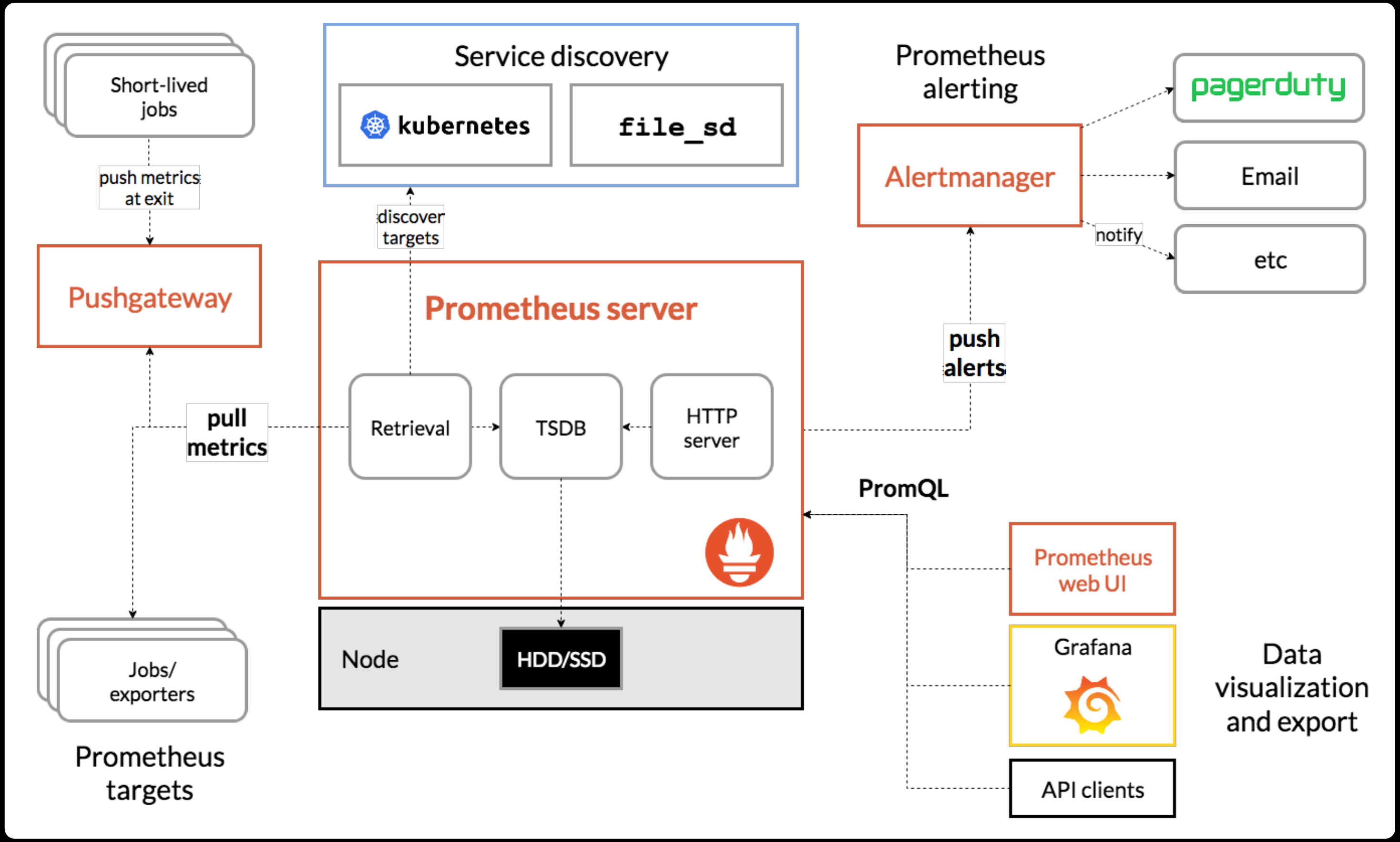
Prometheus collects metrics using a **pull model**. This means the Prometheus server pulls metric data from exporters — agents do not push data to the Prometheus server.



- **Prometheus Server** - Collects metric data.
- **Exporters** - Provide metric data for Prometheus to consume.
- **Client Libraries** - Easily turn your custom application into an exporter that exposes metrics in a format Prometheus can consume.
- **Prometheus Pushgateway** - Allows pushing metrics to Prometheus for certain specific use cases.
- **Alertmanager** - Sends alerts triggered by metric data.
- **Visualization Tools** - Provide useful ways to view metric data. These are not necessarily part of Prometheus.

Prometheus Components

- **Prometheus Server** - Collects metric data.
- **Exporters** - Provide metric data for Prometheus to consume.
- **Client Libraries** - Easily turn your custom application into an exporter that exposes metrics in a format Prometheus can consume.
- **Prometheus Pushgateway** - Allows pushing metrics to Prometheus for certain specific use cases.
- **Alertmanager** - Sends alerts triggered by metric data.
- **Visualization Tools** - Provide useful ways to view metric data. These are not necessarily part of Prometheus.


Short-lived jobs
push metrics at exit
Pushgateway
Service discovery
kubernetes
file_sd
discover targets
Prometheus alerting
pagerduty
Alertmanager
Email
notify
etc
push alerts
Prometheus server
Retrieval
TSDB
HTTP server
pull metrics
PromQL
Node
HDD/SSD
Jobs/ exporters
Prometheus targets
Prometheus web UI
Grafana
API clients
Data visualization and export

Prometheus Use Case: Metric Collection

Judy is the administrator of a large infrastructure consisting of many servers, virtual machines, and components running in the cloud. She wants to be able to monitor the health of on-premises virtual machines as well as instances running in the cloud. Since all these components are interconnected, she does not want to have to search for data in multiple places when there is an outage.

Prometheus allows Judy to collect all this data in one place.

Prometheus Use Case: Visualization

Mark's company has a complex IT infrastructure, and he is periodically on-call to respond to any outages that occur outside normal business hours. When an outage occurs, it often takes him a long time to discover which server or application is down. He wants to create a dashboard with graphs and charts that can show him what is happening at a glance so he can quickly diagnose problems and get back to sleep.

Prometheus allows Mark to build a dashboard that provides statistics on all his servers and applications. When something goes wrong, it is easy to spot where the problem lies.

Prometheus Use Case: Alerting

Sarah's company has been having some problems with their network. She is frequently putting out fires when services become unavailable for customers. She is aware of ways to detect and fix these outages before they become serious, but it would be unrealistic to have someone constantly watch the data to determine when a problem is about to occur. She would like to build some automation to send her an email whenever the network issue is about to happen again.

With Prometheus, Sarah creates an alert that will trigger when the metric data indicates a problem is about to occur. Next time the issue is about to happen, she is able to take steps to prevent it.

Prometheus Limitations

While Prometheus is a great tool for a variety of use cases, it is important to understand when it is not the best tool.

- **100% accuracy** (e.g., per-request billing) - Prometheus is designed to operate even under failure conditions. This means it will continue to provide data even if new data is not available due to failures and outages. If you need 100% up-to-the-minute accuracy, such as in the case of per-request billing, Prometheus may not be the best tool to use.

- **Non time-series data** (e.g., log aggregation) - Prometheus is built to monitor time-series metrics, especially data that is numeric. It is not the best choice for collecting more generic types of data, such as system logs.

Time-Series Data

Prometheus is built around storing **time-series data**.

Time-series data consists of a series of values associated with different points in time.

Single Data Point vs. Time Series

You could track a single data point, such as the current outdoor temperature.

Outdoor temperature: 0C / 31F

However, if you write down the temperature once every hour,
that's a **time series**!

- 08:00AM — -6C / 21F
- 09:00AM — -3C / 26F
- 10:00AM — -2C / 28F
- 11:00AM — 0C / 31F

Time-Series Metric Example

Every metric in Prometheus tracks a particular value over time.

For example, Prometheus might track the available memory for a server, with an entry in the time series for every minute.

```
02-21-2020 09:55AM - node_memory_MemAvailable_bytes=3734269952
02-21-2020 09:56AM - node_memory_MemAvailable_bytes=3734276545
02-21-2020 09:57AM - node_memory_MemAvailable_bytes=3734295563
02-21-2020 09:58AM - node_memory_MemAvailable_bytes=3734263327
```

Time-Series Data in Prometheus

All Prometheus data is fundamentally stored as **time-series data**.

This means Prometheus not only tracks the current **value** of each metric but also **changes** to each metric over time.

Querying

Querying allows you to access and work with your metric data in Prometheus.

You can use **PromQL** (Prometheus Query Language) to write queries and retrieve useful information from the metric data collected by Prometheus.

Querying

You can use Prometheus queries in a variety of ways to obtain and work with data.

- Expression browser
- Prometheus HTTP API
- Visualization tools such as Grafana

Visualization

In Prometheus, **visualization** refers to the creation of visual representations of your metric data, such as charts, graphs, dashboards, etc.

Visualization Methods

There are multiple tools that can help you visualize your metric data.

Some tools are **built into** Prometheus.

- Expression Browser
- Console Templates

Other **external tools** can be integrated with Prometheus to visualize Prometheus data.

- Grafana
- Others

Grafana

Grafana is an open-source analytics and monitoring tool.

Grafana can connect to Prometheus, allowing you to build visualizations and dashboards to display your Prometheus metric data.

Grafana and Prometheus

Grafana is an open-source analytics and monitoring tool.

With Grafana, you can:

- Access Prometheus data using queries.
- Display query results using a variety of different panels (graphs, gauges, tables, etc.).
- Collect multiple panels into a single dashboard.

Exporters

Exporters provide the metric data that is collected by the Prometheus server.

An exporter is any application that exposes data in a format Prometheus can read.

The **scrape_config** in **prometheus.yml** configures the Prometheus server to regularly connect to each exporter and collect metric data.

Prometheus Pushgateway

Prometheus server uses a **pull method** to collect metrics, meaning Prometheus reaches out to exporters to pull data. Exporters do not reach out to Prometheus.

However, there are some use cases where a **push method** is necessary, such as monitoring of batch job processes.

Prometheus Pushgateway serves as a middle-man for these use cases.

- Clients push metric data to Pushgateway.
- Prometheus server pulls metrics from Pushgateway, just like any other exporter.

When to Use Pushgateway

The Prometheus documentation recommends using Pushgateway only for **very specific use cases**.

These usually involve **service-level batch jobs**. A batch job's process exits when processing is finished. It is unable to serve metrics once the job is complete. It should not need to wait for a scrape from Prometheus server in order to provide metrics; therefore, such jobs need a way to push metrics at the appropriate time.

Recording Rules

Recording rules allow you to pre-compute the values of expressions and queries and save the results as their own separate set of time-series data.

Recording rules are evaluated on a schedule, executing an expression and saving the result as a new metric.

Recording rules are especially useful when you have complex or expensive queries that are run frequently. For example, by saving pre-computed results using a recording rule, the expression does not need to be re-evaluated every time someone opens a dashboard.

Recording Rules

Recording rules are configured using YAML. Create them by placing YAML files in the location specified by **rule_files** in **prometheus.yml**.

When creating or changing recording rules, reload their configuration the same way you would when changing **prometheus.yml**.

```
groups:
- name: my_rule_group
  rules:
  - record: my_custom_metric
    expr: up{job=“My Job”}
```

Alertmanager

Alertmanager is an application that runs in a separate process from the Prometheus server. It is responsible for handling alerts sent to it by clients such as Prometheus server.

Alerts are notifications that are triggered automatically by metric data.

For example: A server goes down, and the on-call administrator automatically gets an email notifying them of the problem so they can take action.

Alertmanager

Alertmanager does the following:

- Deduplicating alerts when multiple clients send the same alert.
- Grouping multiple alerts together when they happen around the same time.
- Routing alerts to the proper destination such as email, or another alerting application such as PagerDuty or OpsGenie.

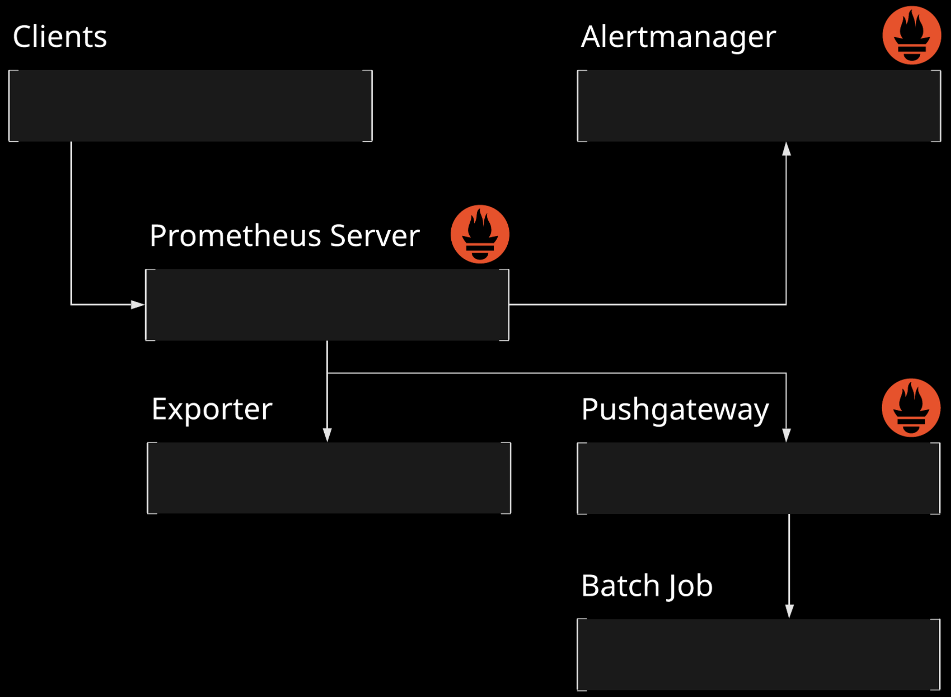
Alertmanager does not create alerts or determine when alerts needs to be sent based on metric data. Prometheus handles that step and forwards the resulting alerts to Alertmanager.

Some Prometheus Server Security Considerations

- Prometheus server does not provide **authentication** out of the box. Anyone who can access the server’s HTTP endpoints has access to all your time-series data.
- Prometheus server does not provide **TLS encryption**. Non-encrypted communications between clients and Prometheus are vulnerable to unencrypted data being read and to man-in-the middle attacks.
- If your Prometheus endpoints are open to a network with potentially untrusted clients, you can add your own security layer on top of Prometheus server using a reverse proxy.

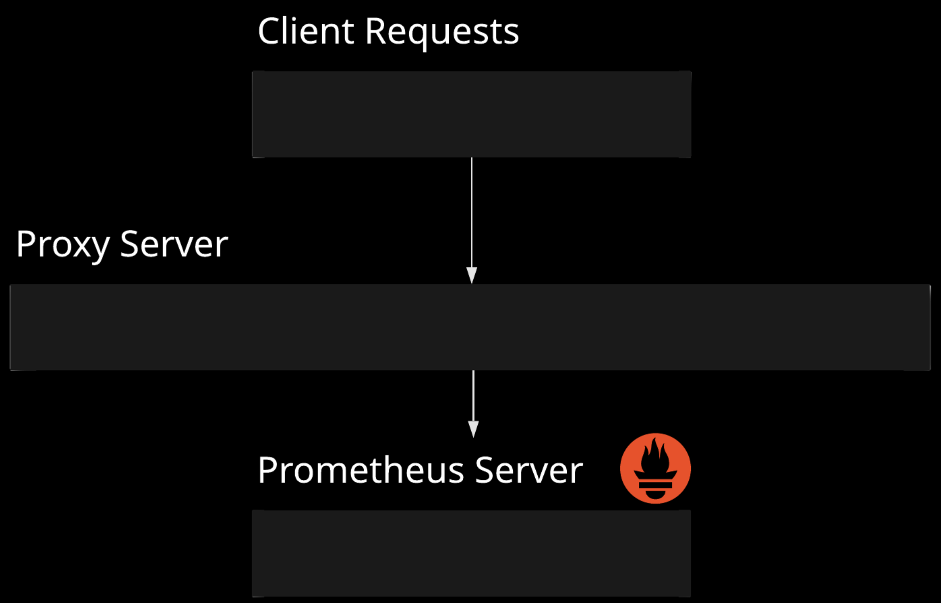
Unsecured Traffic and Endpoints

Be sure to consider all potentially **unsecured traffic** in your Prometheus setup. If your network configuration would allow untrusted users to gain access to sensitive components or data, you may need to take steps to secure your Prometheus setup.



Reverse Proxy

A **reverse proxy** acts as a middleman between clients and a server. You can implement security features on top of Prometheus using a reverse proxy. You can use any simple web server, such as Apache or Nginx, for this purpose.



Client Requests

Proxy Server

Prometheus Server

Alertmanager Security Considerations

- Like Prometheus server, Alertmanager does not provide **authentication** or **TLS encryption**.
- Use a reverse proxy to add your own security layer, if needed.

Pushgateway Security Considerations

- Pushgateway likewise does not provide **authentication** or **TLS encryption**.
- Again, you can add your own security layer with a reverse proxy.

Exporter Security Considerations

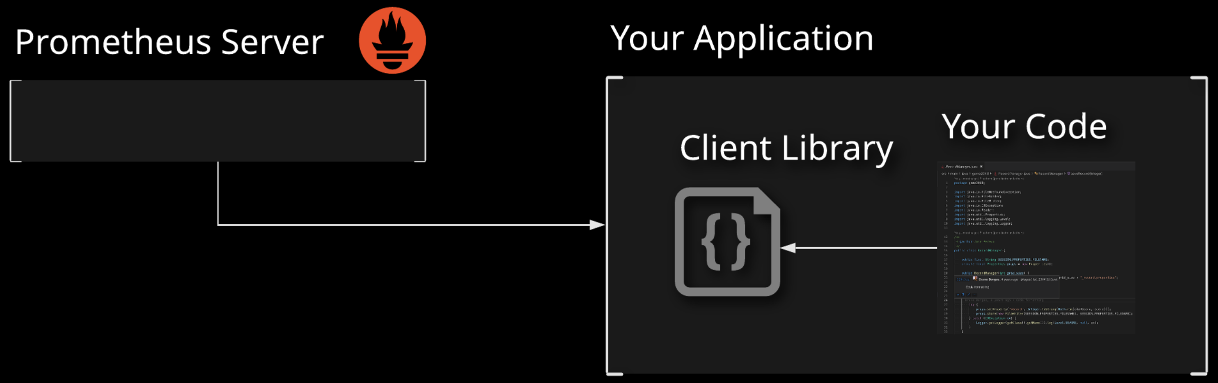
- Every exporter is **different**.
- Many exporters *do* provide authentication and/or TLS encryption.
- Check the documentation for your exporters to learn more about basic security features.
- Without security, data provided by exporters can be read by anyone with access to the **/metrics** endpoint.

Client Libraries

Prometheus **client libraries** provide an easy way to add instrumentation to your code in order to monitor your applications with Prometheus.

Client libraries provide functionality that allows you to:

- Collect and record metric data in your code.
- Provide a **/metrics** endpoint, turning your application into a Prometheus exporter so Prometheus can scrape metrics from your application.

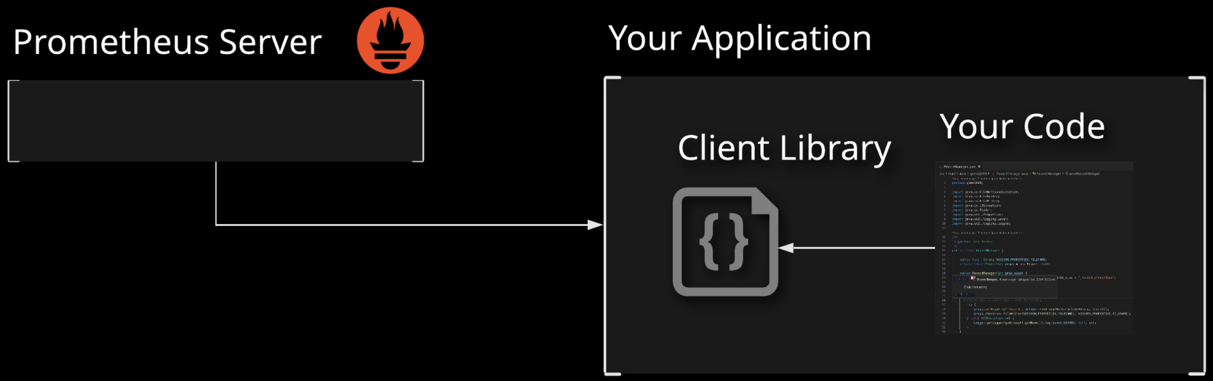


Client Libraries

There are existing client libraries for many popular programming languages and frameworks. You can also code your own client libraries.

Prometheus supports the following official client libraries, although there are many third-party client libraries for other languages.

- Go
- Java/Scala
- Python
- Ruby



What’s Next?

- **DevOps Monitoring Deep Dive** — Learn how Prometheus can function within the context of DevOps.
- **Monitoring Kubernetes with Prometheus** — Use Prometheus to monitor a Kubernetes cluster.
- **AIOps Essentials (Autoscaling Kubernetes with Prometheus Metrics)** — Learn how to use Prometheus to autoscale Kubernetes applications.