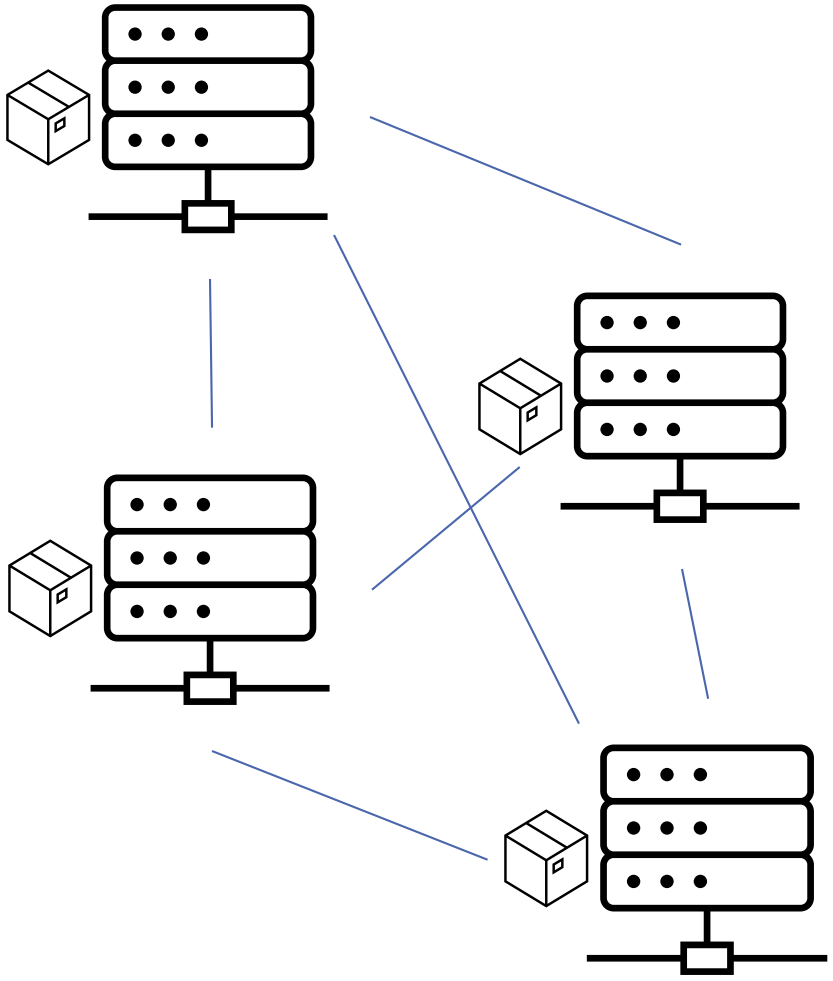
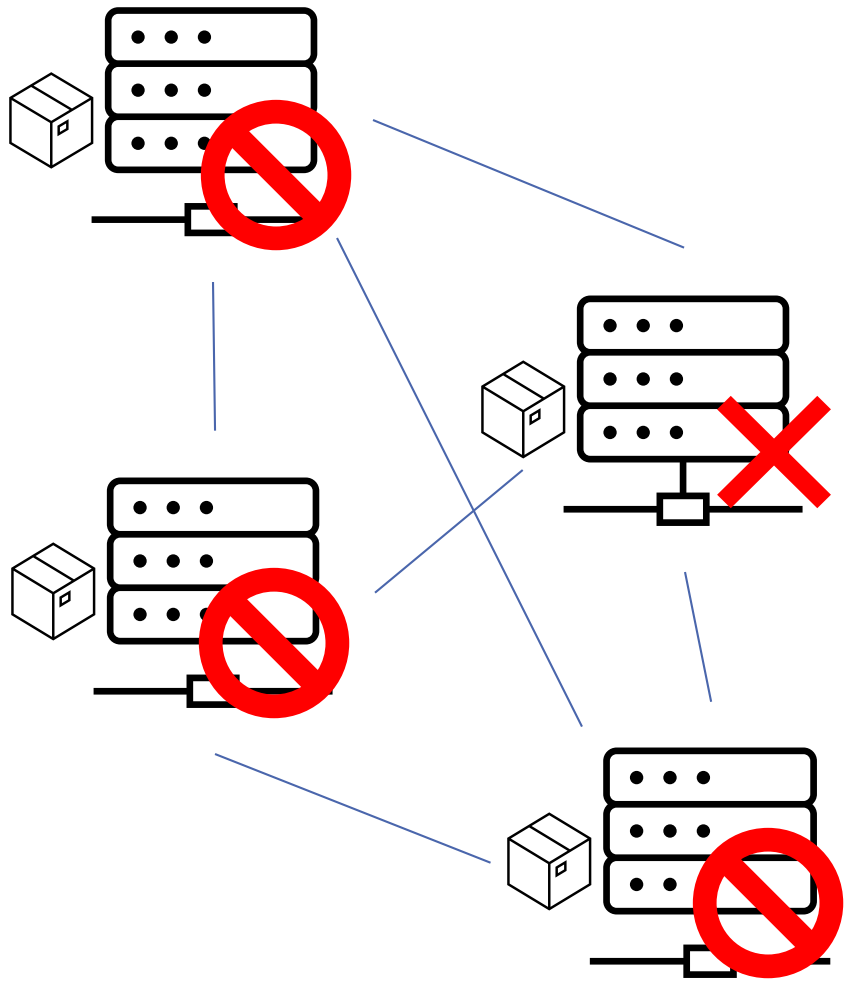
MONITORING

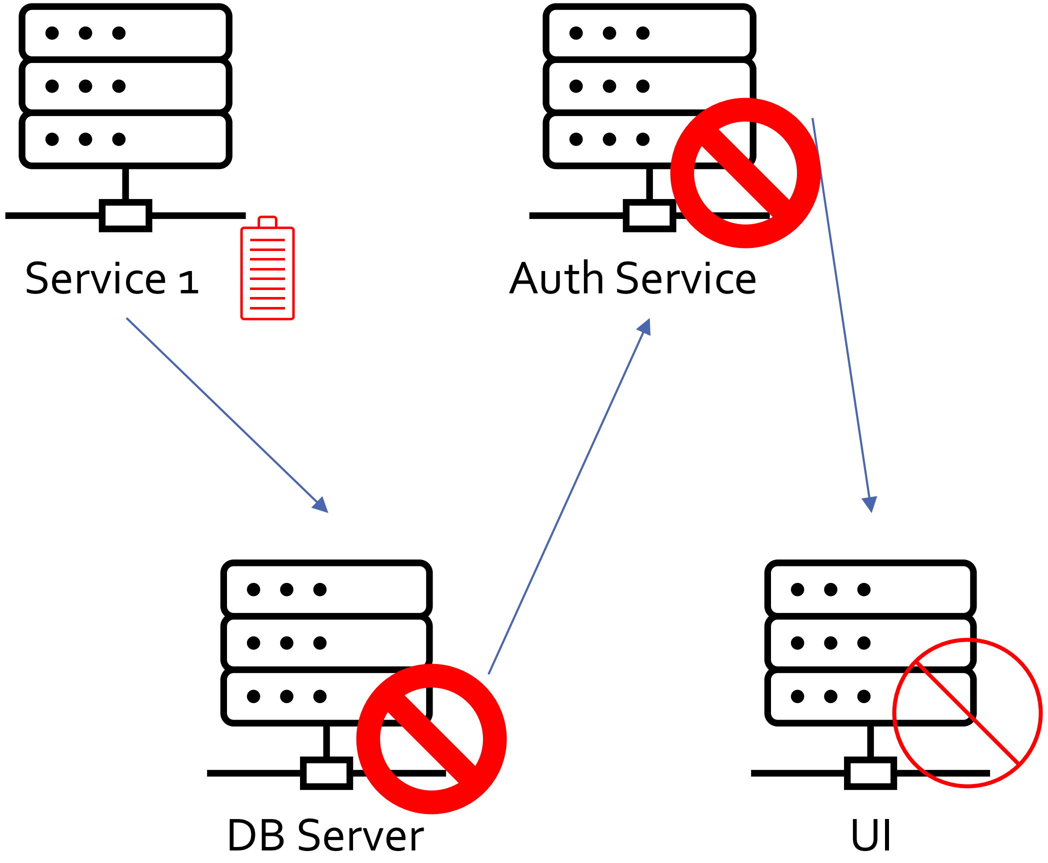


- 100s of running processes
- All Interconnected
- Need to maintain consistent performance

- Need continuous feed back about:
  - Hardware performance
    Running out of resources, hardware malfunction, overloading…
  - Application Performance
    Errors, latency, resource management,…



- In case one service is down, it can disable many others.
- The error shown might not be the root source of the issue.
- Tracing back all connected services and servers to find the root cause of the error is troublesome and time consuming.



- In case Service 1 ran out of memory, the database storage will no longer be managed, and that would disable the authentication service which stores its security keys in the DB Server, and finally users will not be able to authenticate.
- More Efficient Solution: have a tool constantly monitoring all the services to identify what would be the system failure or better yet, anticipate system failures and inform the responsible system admin.
- The error shown on the UI does not reflect the main problem.
- We'll have to trace it back up the chain to find the main problem source.

Categories of Monitoring

Most monitoring is events:

- Receiving a HTTP request
- Sending a HTTP 400 response
- Entering a function
- Reaching the else of an if statement
- Leaving a function
- A user logging in
- Writing data to disk
- …

Categories of Monitoring

A lot of events are generated, reducing the volume of data to something workable can be done through:

- **Profiling**: takes the approach that you can't have all the context for all the events all the time, but you can have some of the context for limited periods of time. (ex: Tcpdump)
- **Tracing**: doesn't look at all events, rather it takes some proportion of events such as one in a hundred that pass through some functions of interest. (ex: sampling one in a hundred user HTTP requests)
- **Logging**: looks at a limited set of events and records some of the context for each of these events. (ex: logging all incoming HTTP requests)
- **Metrics**: largely ignore context, instead tracking aggregations over time of different types of events (ex: number of times you received HTTP requests)

Logs VS Metrics

Logs

- Allow you to collect information about all of one type of event, but can only track a hundred fields of context with unbounded cardinality.

Metrics

- Allow you to collect information about events from all over your process, but with generally no more than one or two fields of context with bounded cardinality

Prometheus

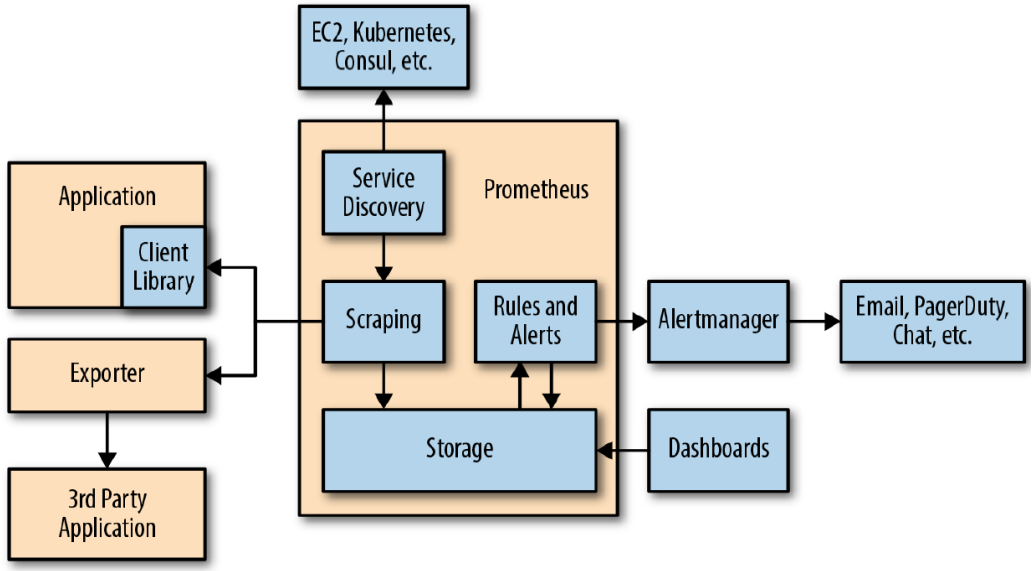
- Is a metrics-based monitoring system, designed to track overall system health, behavior, and performance rather than individual events.
- Prometheus discovers targets to scrape from service discovery. These services can be your own instrumented applications or third-party applications you can scrape via an exporter.
- The scraped data is stored, and you can use it in dashboards using PromQL or send alerts to the Alertmanager, which will convert them into pages, emails, and other notifications.

Why Use Prometheus?

It is an industry standard:

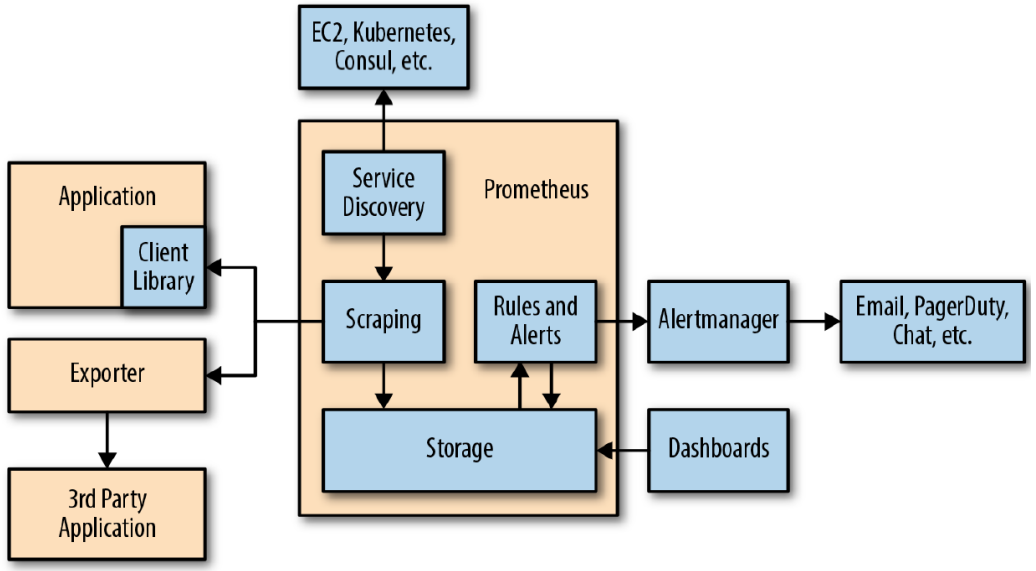
- Prometheus is an **open-source** monitoring software that is very popular in the industry.
- It is easy to customize, and produces metrics **without impacting application performance**.
- It integrates into the DevOps system by monitoring cloud-native applications and infrastructure and watching over hundreds of microservices.
- The Prometheus Node Exporter can be adjusted to **retrieve data from any client**.

Prometheus Architecture



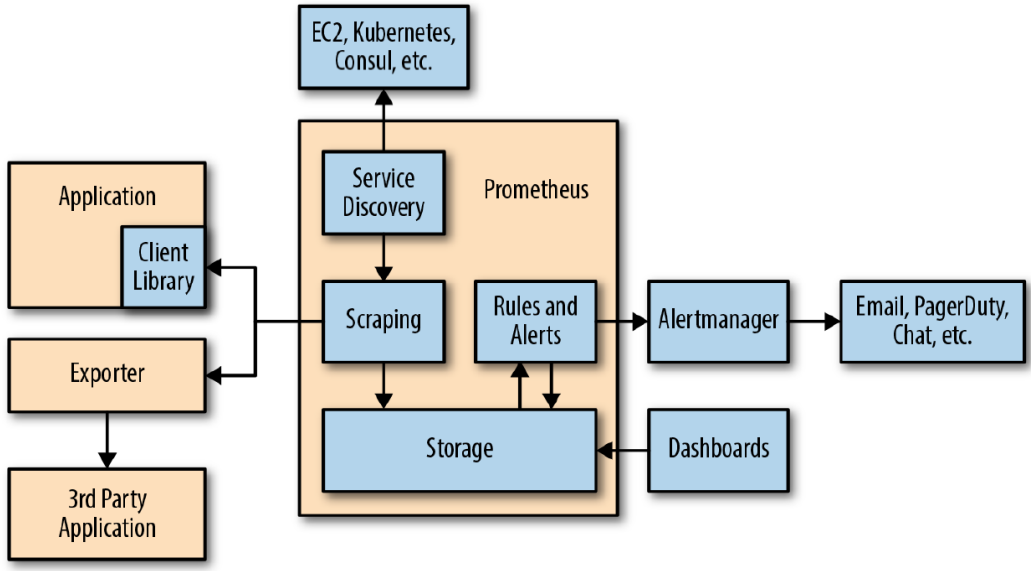
- **Client** Library: used to produce metrics for your code.
- **Exporters**: a piece of software deployed, which gathers data from the application and returns them to Prometheus.
- **Service Discovery**: tells Prometheus where are the instrumented applications and exporters running.

Prometheus Architecture



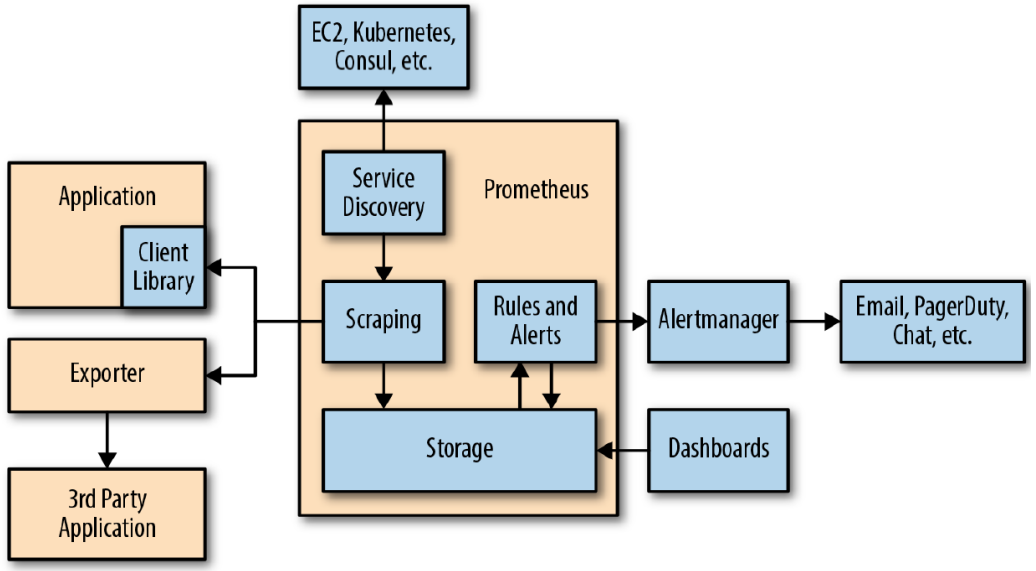
- **Scraping**: sending a HTTP request called a *scrape* to fetch the metrics of the applications monitored.
- **Storage**: Prometheus stores data locally in a custom database, in a non-distributed fashion.
- **Dashboard**: Prometheus has several HTTP APIs that allow you request and evaluate data and queries. These can be used to produce graphs and dashboards.

Prometheus Architecture



- **Recording Rules and Alerts:** rules are recorded, which allows PromQL expressions to be evaluated on a regular basis and their results ingested into the storage engine.
- **Alert Management**: Alertmanager receives alerts from Prometheus servers and turns them into notifications.

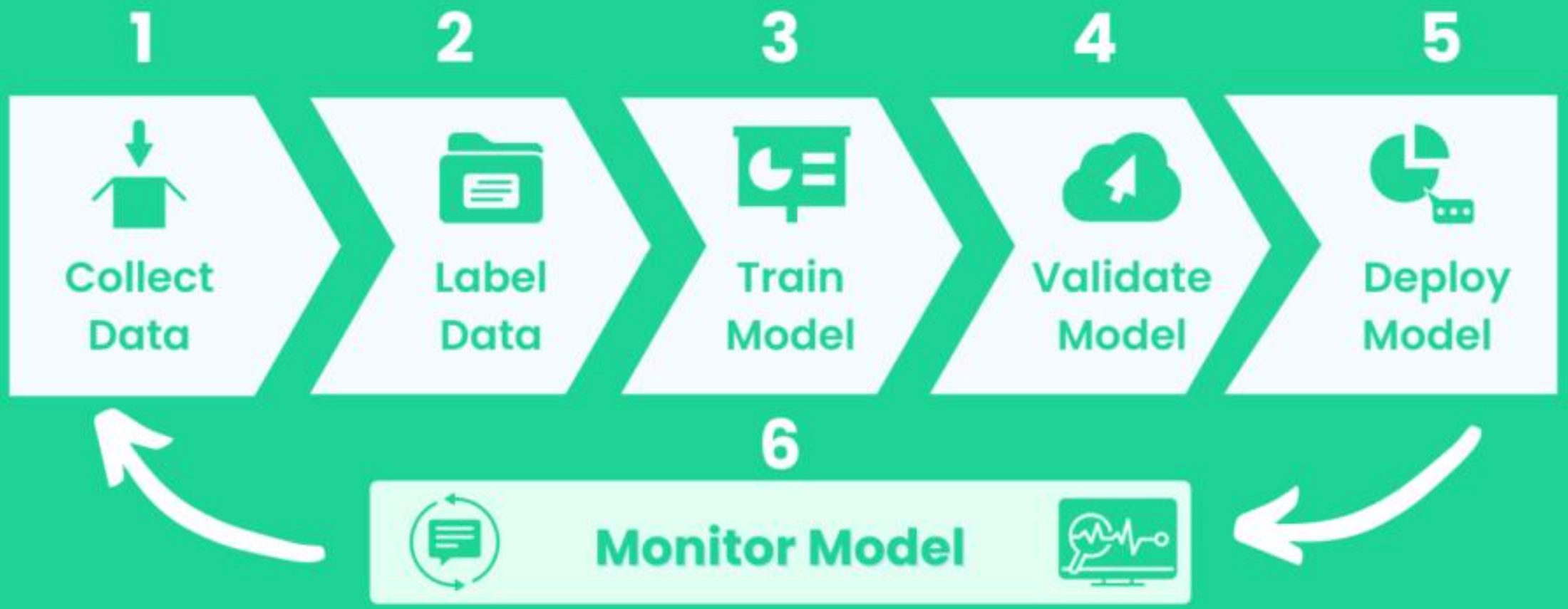
Prometheus Architecture



- **Prometheus is often used in conjunction with Grafana as a dashboard for customizing your visualization of the metrics/logs (https://grafana.com/).**

ML MONITORING

ML Pipeline



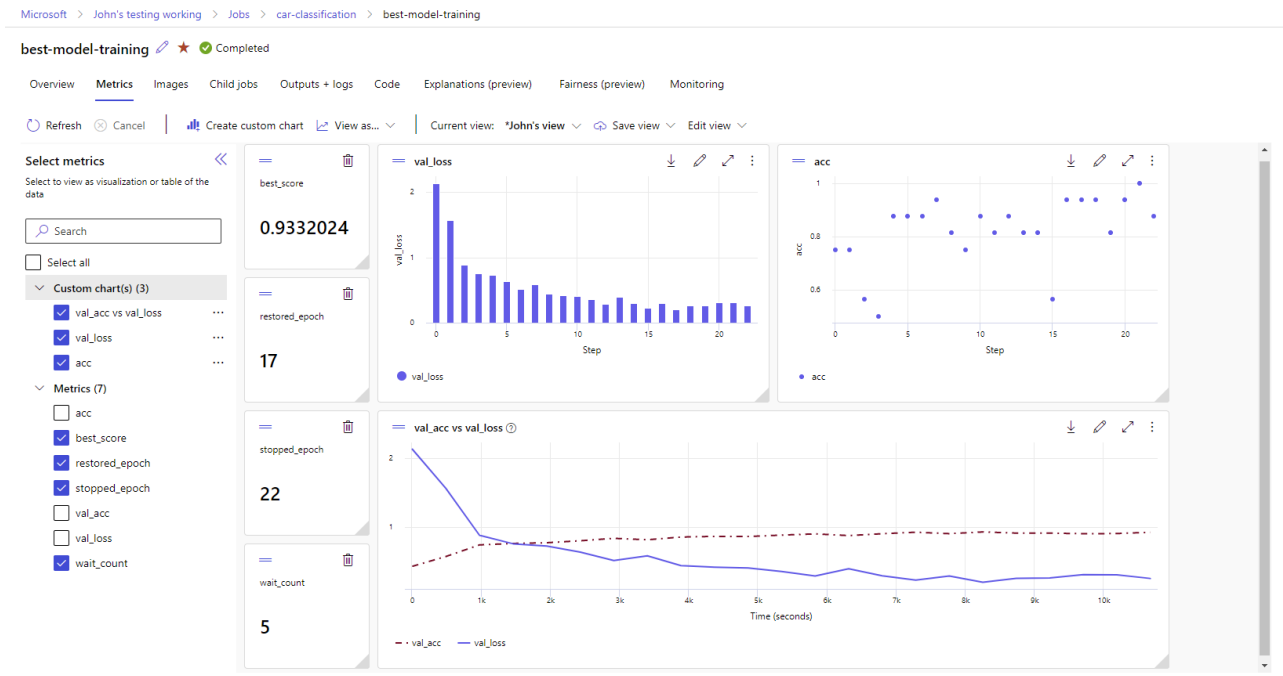
aporia

Monitoring ML Models in Production

Continuously tracking and evaluating ML models' performance

Models can degrade due to data drifts

Timely detection and correction of such issues

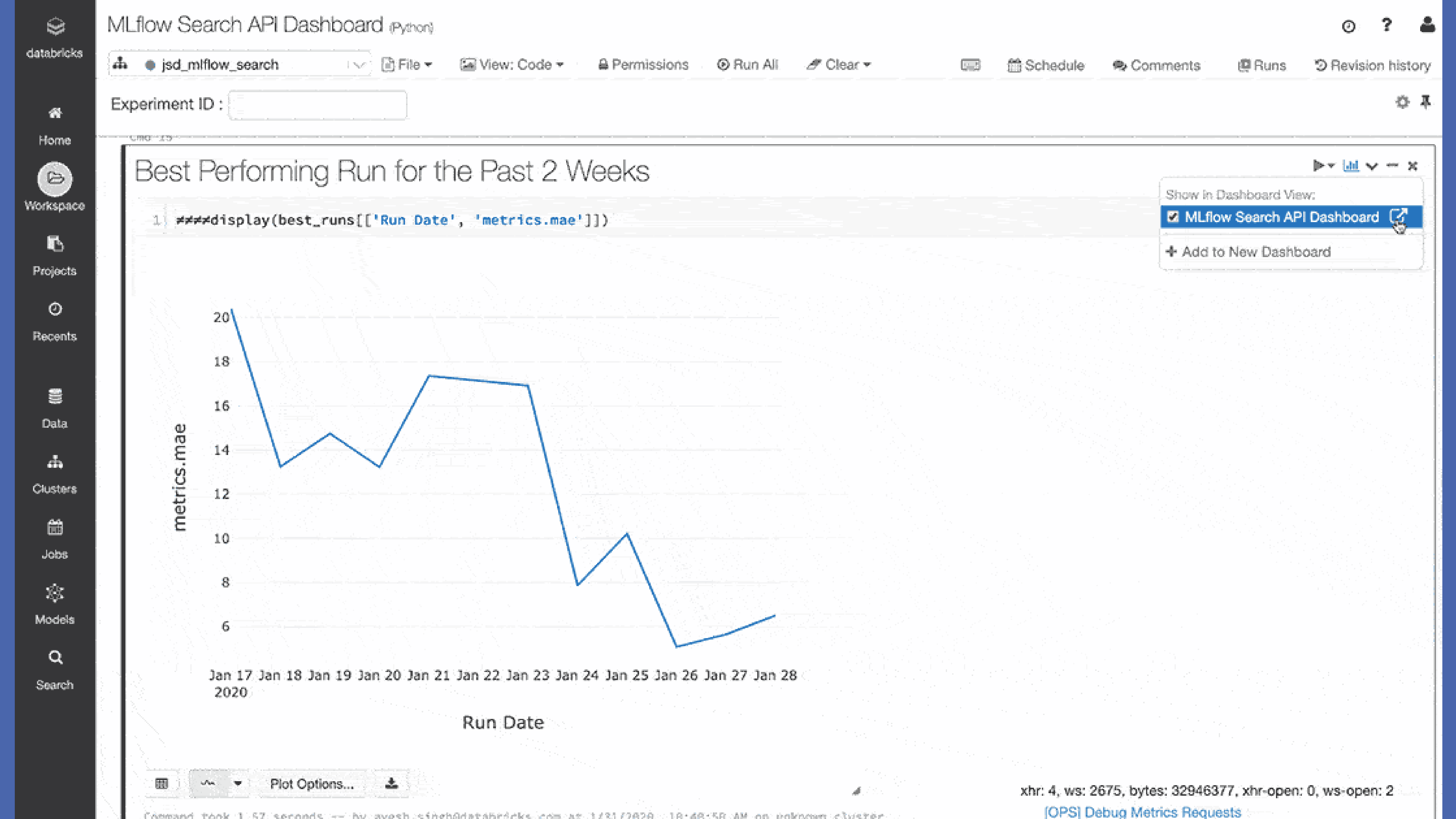


Types of Information to Monitor

- Performance Metrics: Accuracy, precision, recall …
- Prediction Latency: The time it takes for the model to make a prediction
- Data Drift: Changes in the distribution of input data over time
- Model Drift: Changes in model performance over time
- Operational Metrics: System health indicators, CPU and memory usage …

MLflow

- An open-source platform designed to manage the entire machine learning lifecycle
  - Experimentation
  - Reproducibility
  - Deployment
  - Monitoring
- It offers several components
  - MLflow Tracking for logging experiments
  - MLflow Models for packaging and deployment
  - MLflow Projects for collaboration.


databricks
Home
Workspace
Projects
Recents
Data
Clusters
Jobs
Models
Search
MLflow Search API Dashboard (Python)
jsd_mlflow_search
File
View: Code
Permissions
Run All
Clear
Schedule
Comments
Runs
Revision history
Experiment ID :
Best Performing Run for the Past 2 Weeks
####display(best_runs[['Run Date', 'metrics.mae']])
Show in Dashboard View:
MLflow Search API Dashboard
Add to New Dashboard
20
18
16
14
12
10
8
6
metrics.mae
Jan 17 Jan 18 Jan 19 Jan 20 Jan 21 Jan 22 Jan 23 Jan 24 Jan 25 Jan 26 Jan 27 Jan 28
2020
Run Date
Plot Options...
xhr: 4, ws: 2675, bytes: 32946377, xhr-open: 0, ws-open: 2
[OPS] Debug Metrics Requests

Setting up MLflow

- Without Docker

```
pip install mlflow
```

- Then, initiate a tracking server by specifying the file store and artifact location

```
mlflow server --backend-store-uri ./mlruns --default-artifact-root ./mlartifacts --host 0.0.0.0
```

This setup allows you to start tracking your ML experiments and model performance by logging into the local server.

Setting up MLflow

- With Docker
- Without Docker

https://mlflow.org/docs/latest/getting-started/intro-quickstart/index.html