KodeKloud

Observability Fundamentals

What is Observability

Observability – The ability to understand and measure the state of a system based upon data generated by the system

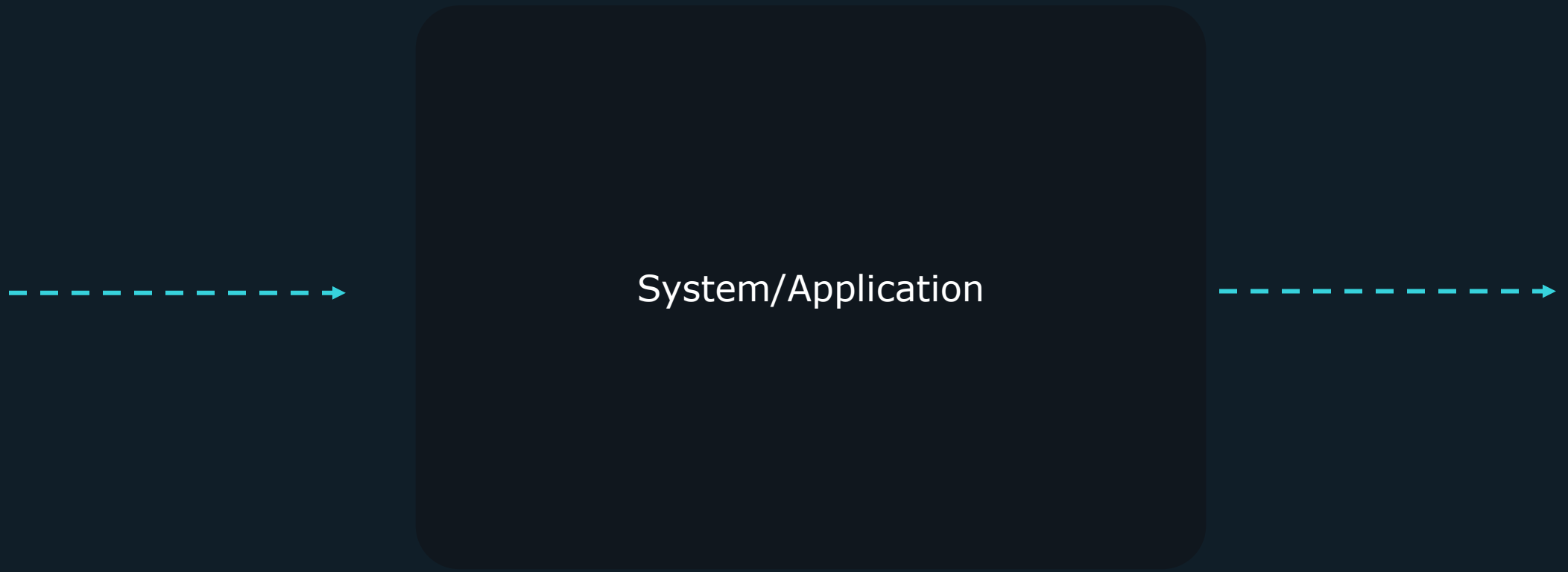
Observability allows you to generate actionable outputs from unexpected scenarios in dynamic environments

Observability will help:

1. Give better insight into the internal workings of a system/application
2. Speed up troubleshooting
3. Detect hard to catch problems
4. Monitor performance of an application
5. Improve cross-team collaboration

Observability

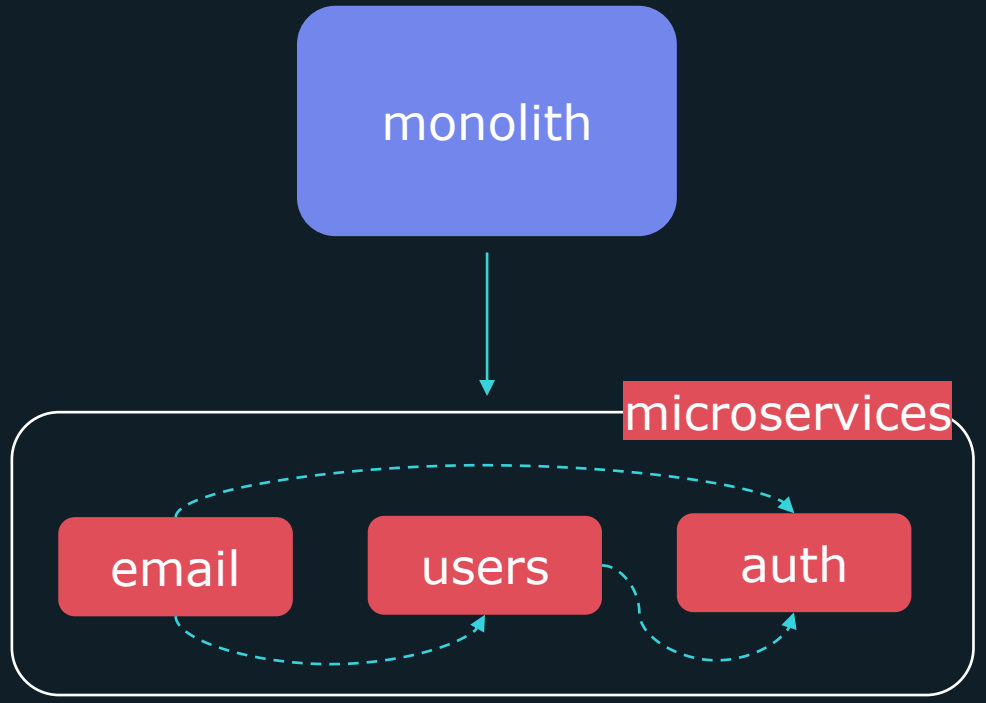
The main purpose of observability it to better understand the internals of your system



Observability

As system architectures continue to get more and more complex, new challenges arise as tracking down issues become far more challenging

There’s a greater need for observability as we move towards distributed systems & microservices based application



Observability

When it comes to troubleshooting issues, we need more information than just what is wrong.

We need to know why our application entered a specific state, what component is responsible and how we can avoid it in the future

- Why are error rates rising
- Why is there high latency
- Why are services timing out

Observability gives you the flexibility to understand unpredictable events

3 pillars of Observability

How do we accomplish observability?

Logging

Metrics

Traces

Logging

Logs are records of events that have occurred and encapsulate information about the specific event

Logs are comprised of:

- Timestamp of when the log occurred
- Message containing information

```
Oct 26 19:35:00 ub1 kernel: [37510.942568] e1000: enp0s3
NIC Link is Down

Oct 26 19:35:00 ub1 kernel: [37510.942697] e1000
0000:00:03.0 enp0s3: Reset adapter

Oct 26 19:35:03 ub1 kernel: [37513.054072] e1000: enp0s3
NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
```

Logs

Logs are the most common form of observation produced by systems

However, they can be difficult to use due to the verbosity of the logs outputted by systems/applications

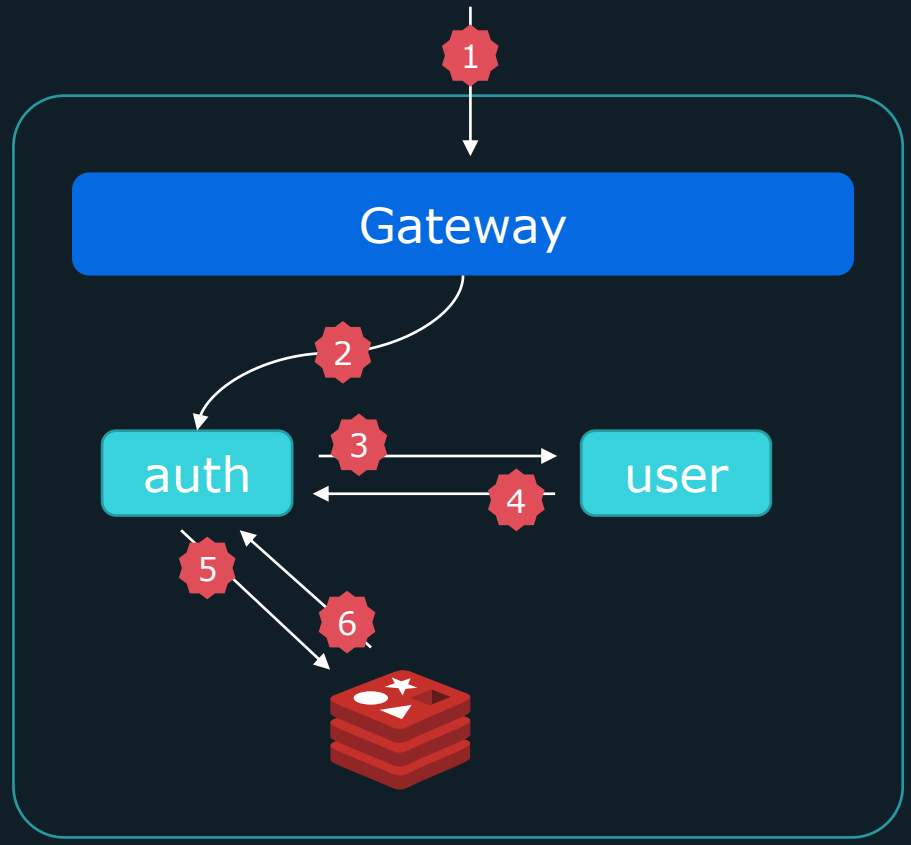
Logs of processes are likely to be interwoven with other concurrent processes spread across multiple systems

Traces

Traces – allow you to follow operations as they traverse through various systems & services

So we can follow an individual request and see it flow through our system hop by hop

Traces help us connect the dots on how processes and services work together



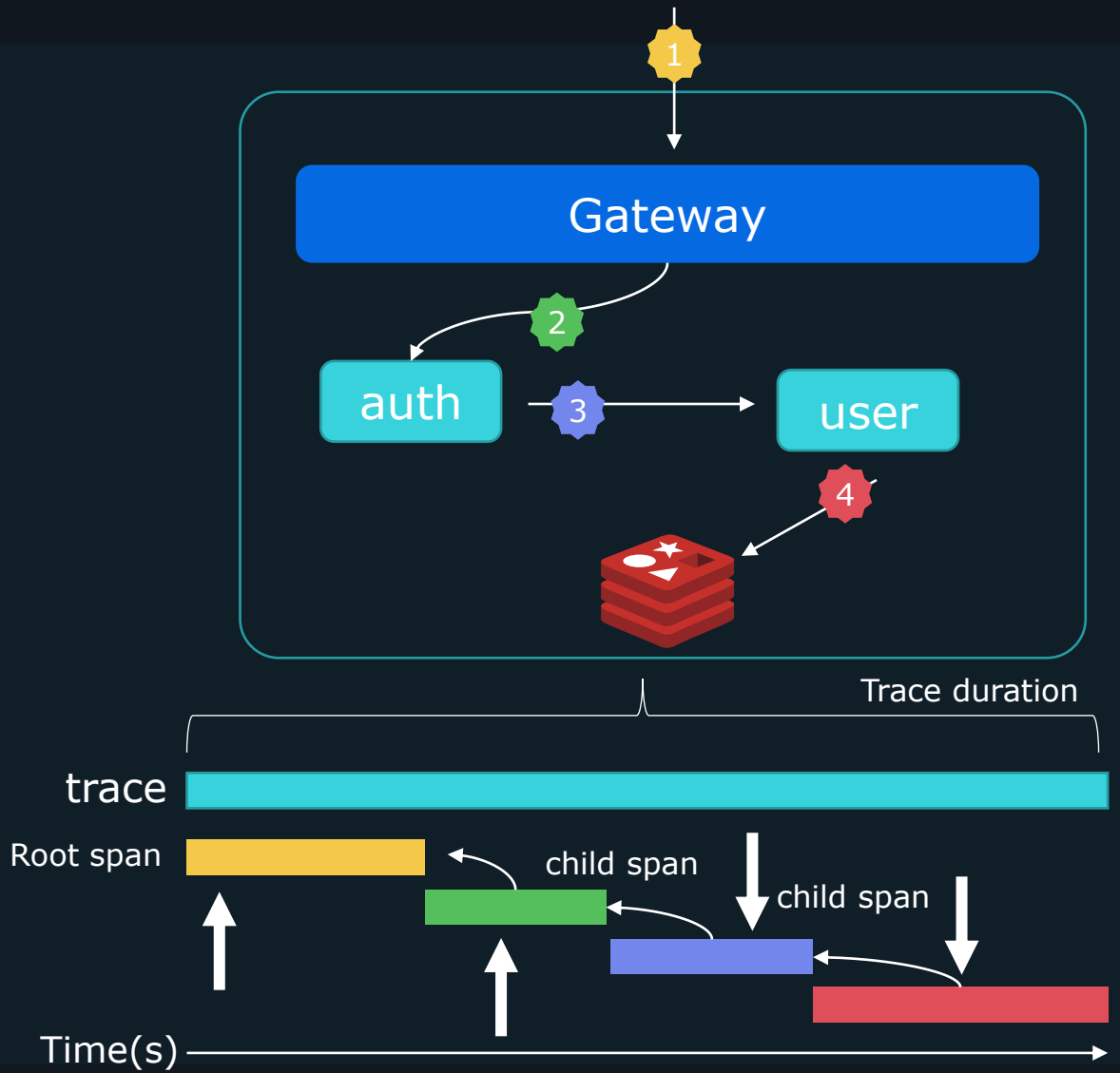
Traces

Each trace has a trace-id that can be used to identify a request as it traverses the system

Individual events forming a trace are called spans

Each span tracks the following:

- Start time
- Duration
- Parent-id

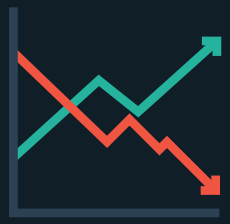


Metrics

Metrics provide information about the state of a system using numerical values

- CPU Load
- Number of open files
- HTTP response times
- Number of errors

The data collected can be aggregated over time and graphed using visualization tools to identify trends over time



Metrics

Metrics contain 4 pieces of information:

1. Metric name
2. Value – most recent or current value of the metric
3. Timestamp for the metric
4. Dimensions – additional information about the metric

Prometheus

Logs

Metrics

Traces

Prometheus is a monitoring solution that is responsible for collecting and aggregating metrics

SLO/SLA/SLI

SLI/SLO/SLA

When designing a system or applications, its important for teams to set specific measurable targets/goals to help organizations strike the right balance between product development and operation work.

These targets help customers & end users quantify the level of reliability they should come to expect from a service

“Application should have 97% uptime in a rolling 30 day window”

Service Level Indicator

Service Level Indicator(SLI) – quantitative measure of some aspect of the level of service that is provided

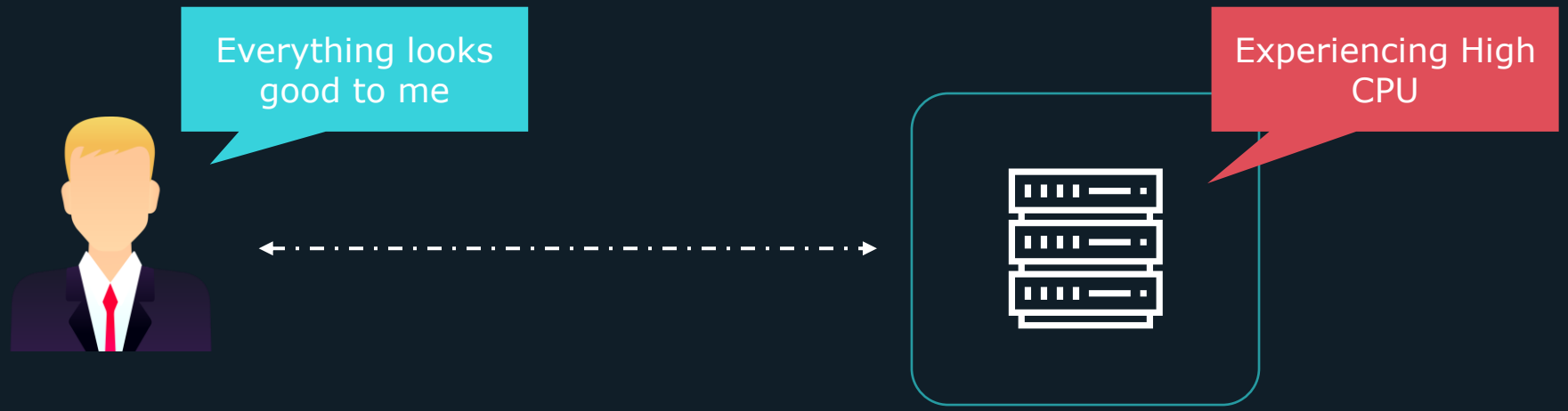
Common SLIs:

- Request Latency
- Error Rate
- Saturation
- Throughput
- Availability

Service Level indicator

Not all metrics make for good SLIs. You want to find metrics that accurately measure a user's experience.

Things like high-cpu/high-memory make for a poor SLI as a user might not see any impact on their end during these events



Service Level Object

Service Level Object(SLO) – target value or range for an SLI

SLI - Latency
SLO – Latency < 100ms

SLI – availability
SLO – 99.9% uptime

SLOs should be directly related to the customer experience. The main purpose of the SLO is to quantify reliability of a product to a customer

SLO

For SLOs it maybe tempting to set them to aggressive values like 100% uptime however this will come at a higher cost

The goal is not to achieve perfection but instead to make customers happy with the right level of reliability

If a customer is happy with 99% reliability increasing it any further doesn’t add any other value

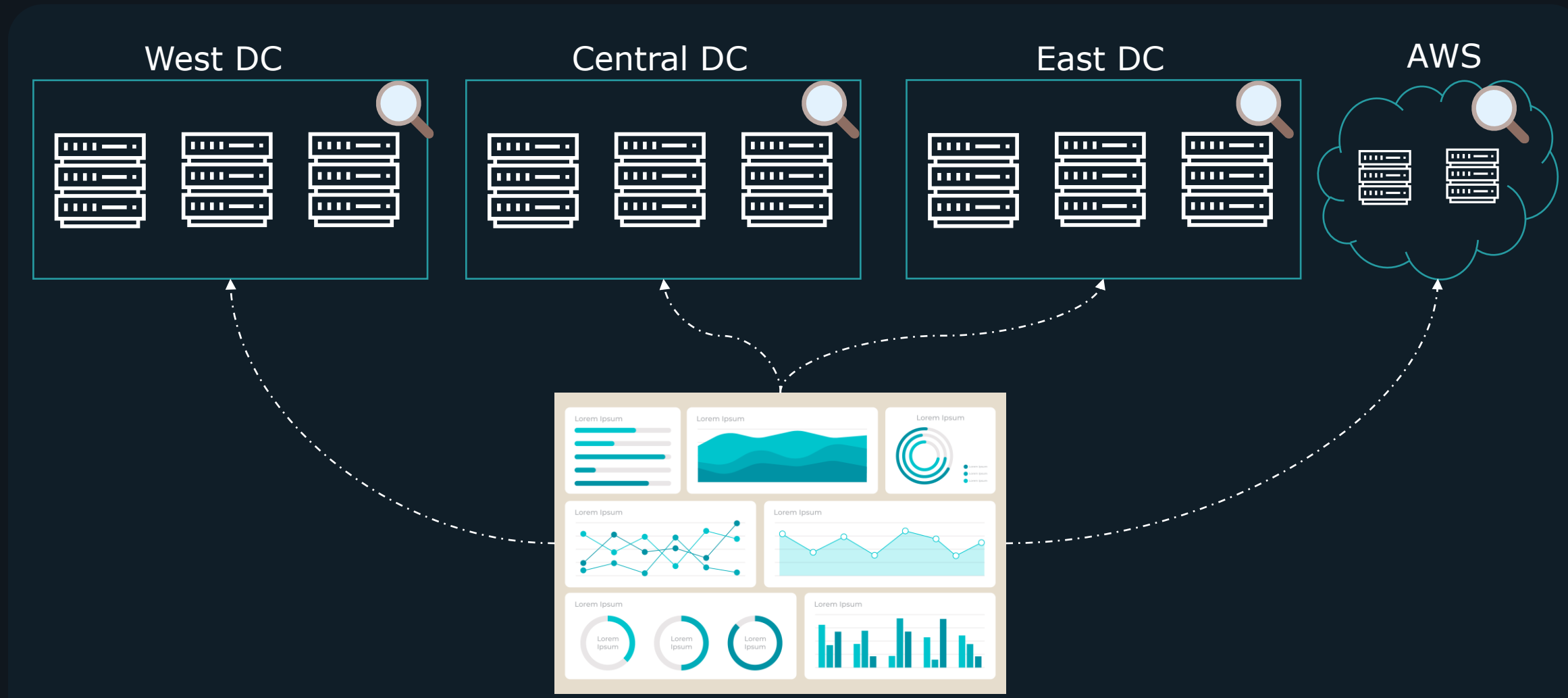
SLA

Service Level Agreement(SLA) – contract between a vendor and a user that guarantees a certain SLO

The consequences for not meeting any SLO can be financial based but can also be variety of other things as well

Prometheus Use Cases

Use Case #1



Usecase #2

Several outages have occurred due to high memory on the server hosting a MySql database.

Operations team would like to be notified through email when the memory gets to 80% max capacity so that proactive measure can be taken

Usecase #3

A new video upload feature has been added to the website, however there is some concerns about users uploading too large of videos.

The team would like to find out at which video length the applications starts to degrade and to do this, they need a chart plotting the avg file size of upload and the average latency per request

Prometheus Basics

What is Prometheus

Prometheus is an open-source monitoring tool that collects metrics data, and provide tools to visualize the collected data

In addition, Prometheus allows you to generate alerts when metrics reach a user specified threshold

Prometheus collects metrics by scraping targets who expose metrics through an HTTP endpoint

Scraped metrics are then stored in a time series database which can be queried using Prometheus' built-in query language PromQl

Prometheus

So what kind of metrics can Prometheus Monitor?

- CPU/Memory Utilization
- Disk space
- Service Uptime
- Application specific data
  - Number of exceptions
  - Latency
  - Pending Requests

Prometheus

Prometheus is designed to monitor time-series data that is numeric

What type of data should Prometheus not monitor

- Events
- System logs
- Traces

Prometheus Background

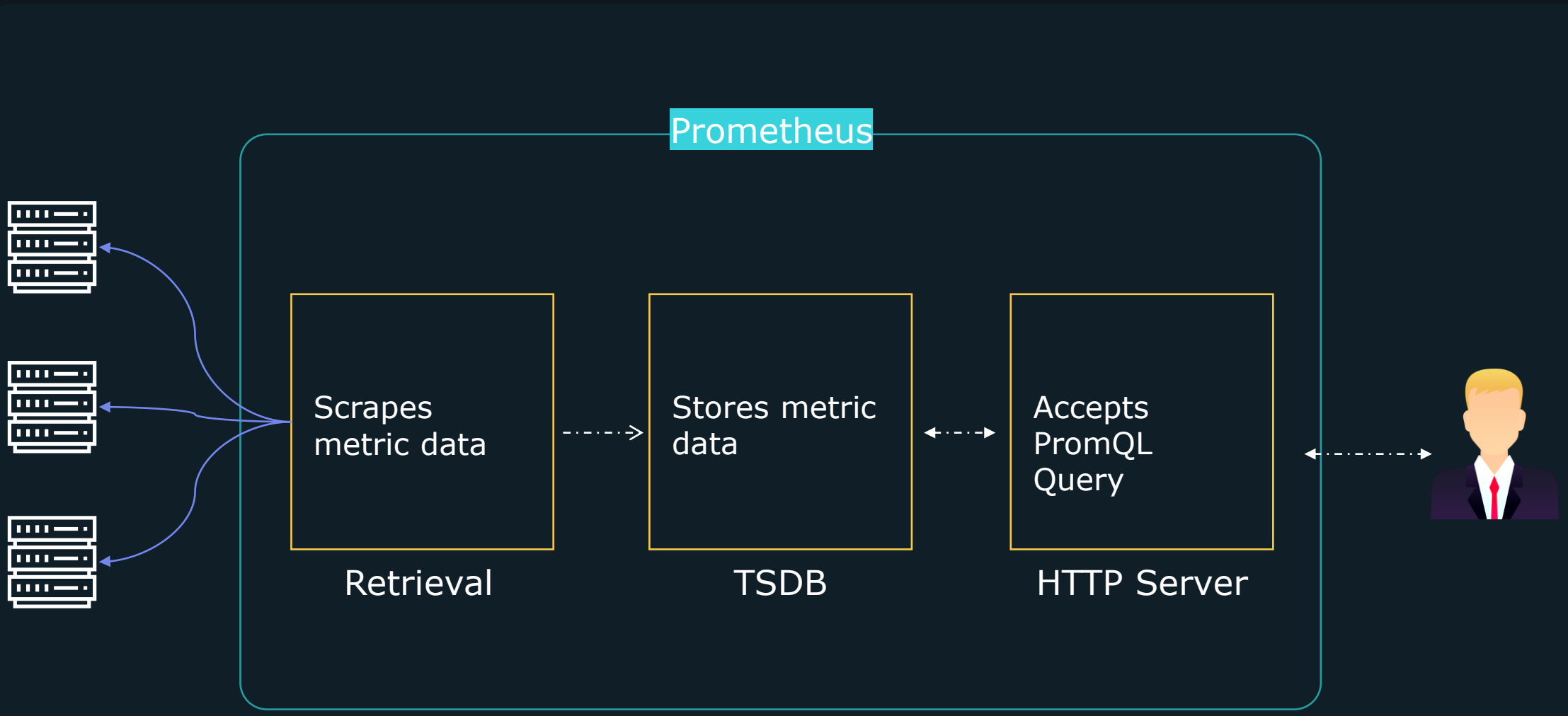
Prometheus was originally sponsored by SoundCloud but in 2016 it joined the Cloud Native Computing Foundation

Prometheus is primarily written GoLang

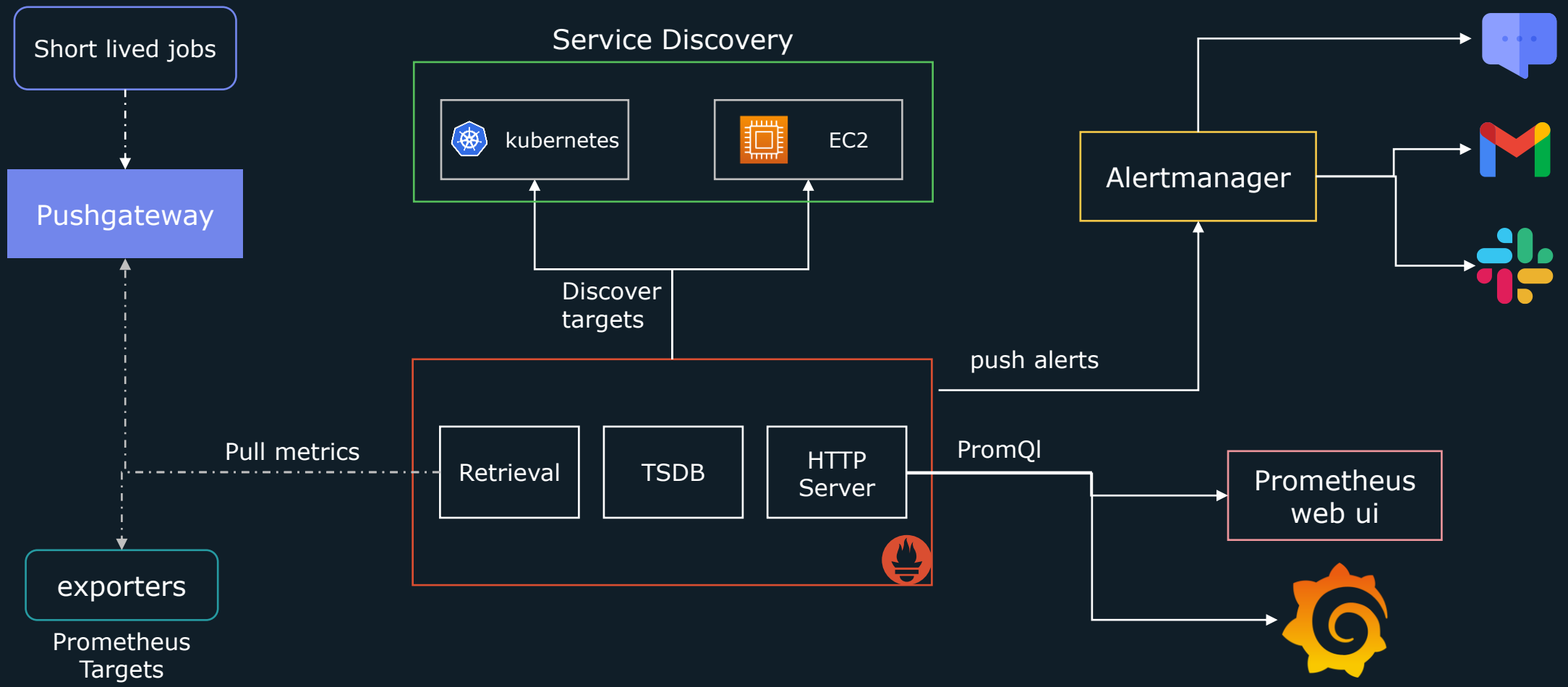
Documentation for Prometheus can be found at prometheus.io/docs

Prometheus Architecture

Prometheus Architecture



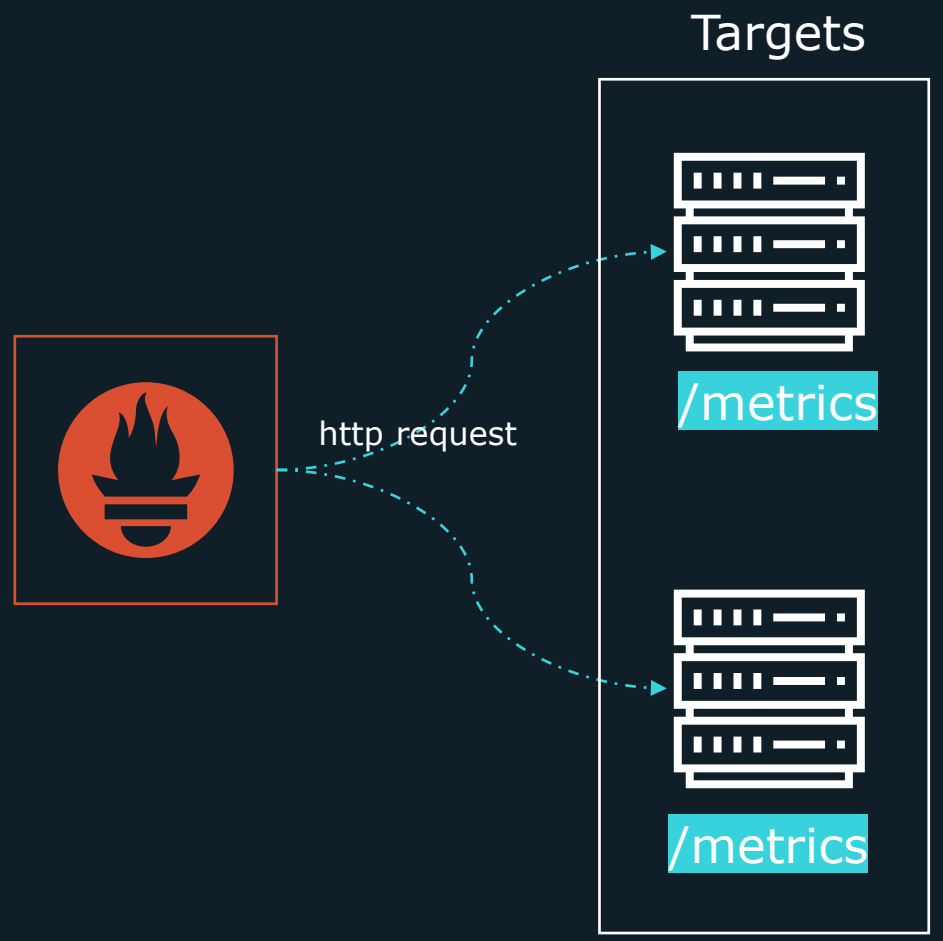
Prometheus Architecture



Collecting Metrics

Prometheus collects metrics by sending http requests to /metrics endpoint of each target

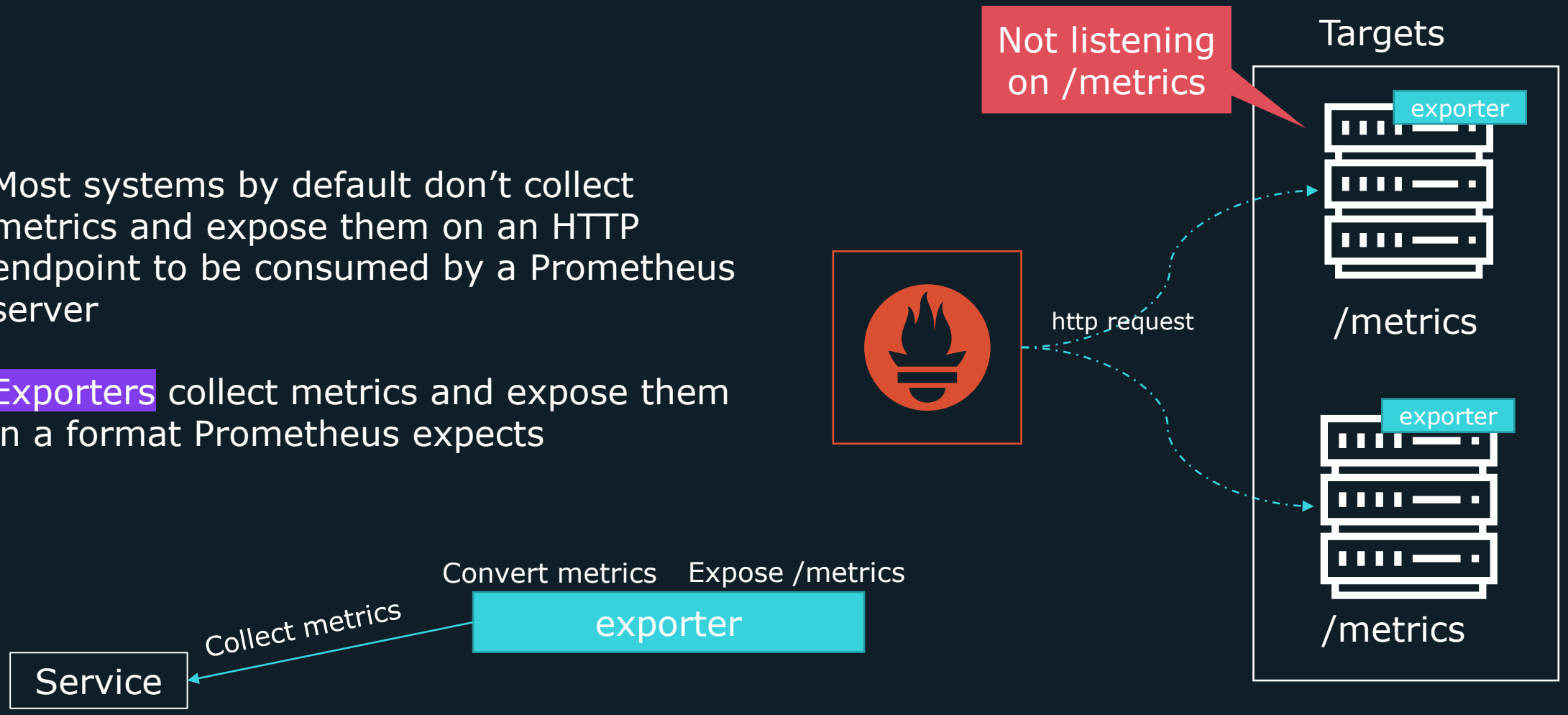
Prometheus can be configured to use a different path other then /metrics



Exporters

Most systems by default don't collect metrics and expose them on an HTTP endpoint to be consumed by a Prometheus server

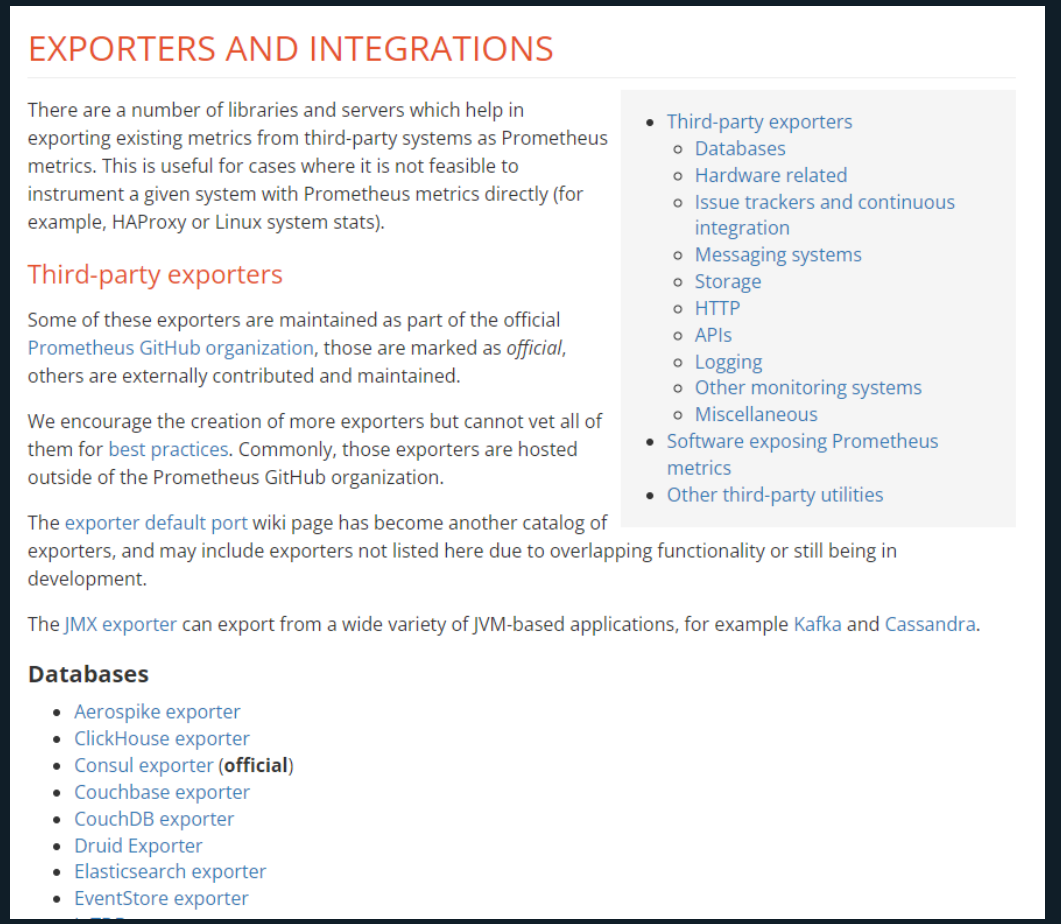
Exporters collect metrics and expose them in a format Prometheus expects



Exporters

Prometheus has several native exporters

- Node exporters(Linux servers)
- Windows
- MySQL
- Apache
- HAProxy



EXPORTERS AND INTEGRATIONS

There are a number of libraries and servers which help in exporting existing metrics from third-party systems as Prometheus metrics. This is useful for cases where it is not feasible to instrument a given system with Prometheus metrics directly (for example, HAProxy or Linux system stats).

- Third-party exporters
  - Databases
  - Hardware related
  - Issue trackers and continuous integration
  - Messaging systems
  - Storage
  - HTTP
  - APIs
  - Logging
  - Other monitoring systems
  - Miscellaneous
- Software exposing Prometheus metrics
- Other third-party utilities

Third-party exporters

Some of these exporters are maintained as part of the official Prometheus GitHub organization, those are marked as *official*, others are externally contributed and maintained.

We encourage the creation of more exporters but cannot vet all of them for best practices. Commonly, those exporters are hosted outside of the Prometheus GitHub organization.

The exporter default port wiki page has become another catalog of exporters, and may include exporters not listed here due to overlapping functionality or still being in development.

The JMX exporter can export from a wide variety of JVM-based applications, for example Kafka and Cassandra.

Databases

- Aerospike exporter
- ClickHouse exporter
- Consul exporter (**official**)
- Couchbase exporter
- CouchDB exporter
- Druid Exporter
- Elasticsearch exporter
- EventStore exporter

Client libraries

Can we monitor application metrics

- Number of errors/exceptions
- Latency of requests
- Job execution time

Prometheus comes with client libraries that allow you to expose any application metrics you need Prometheus to track

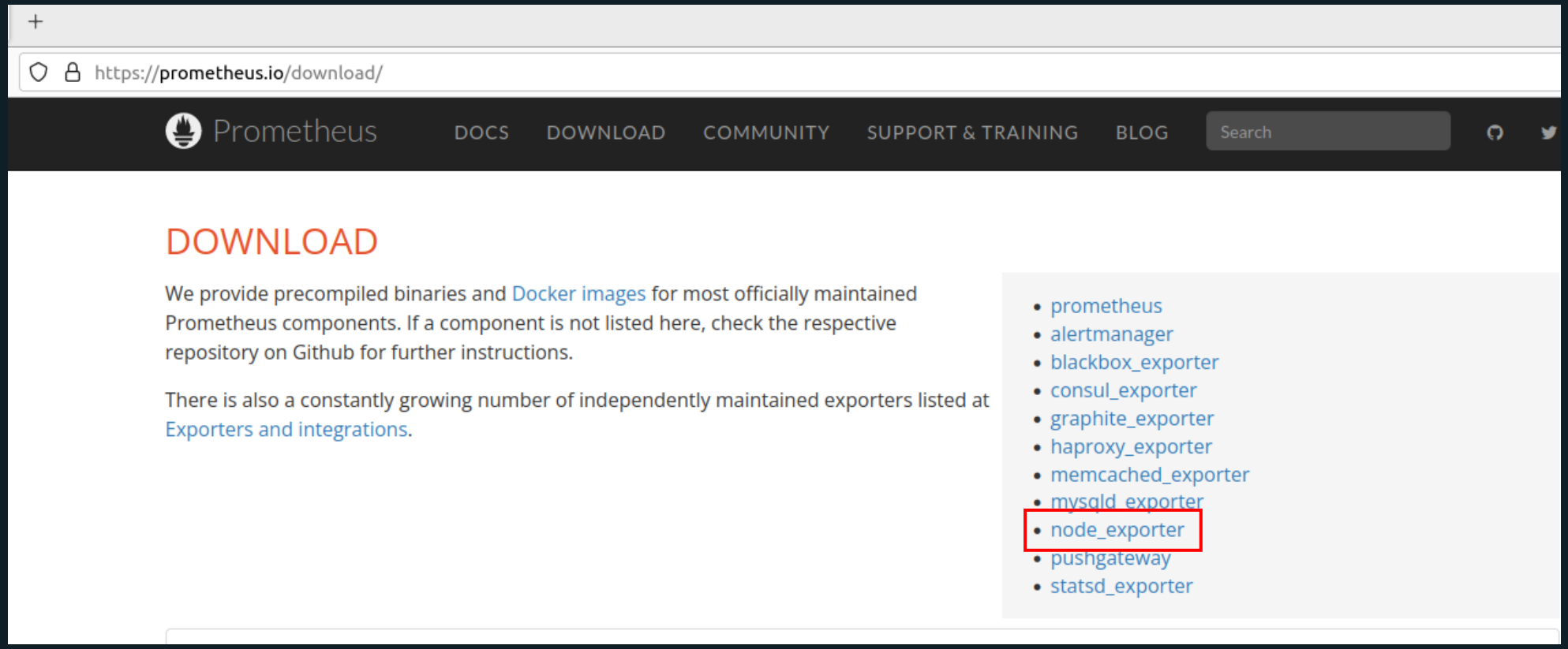
Language support:

- Go
- Java
- Python
- Ruby
- Rust

Node Exporter Installation

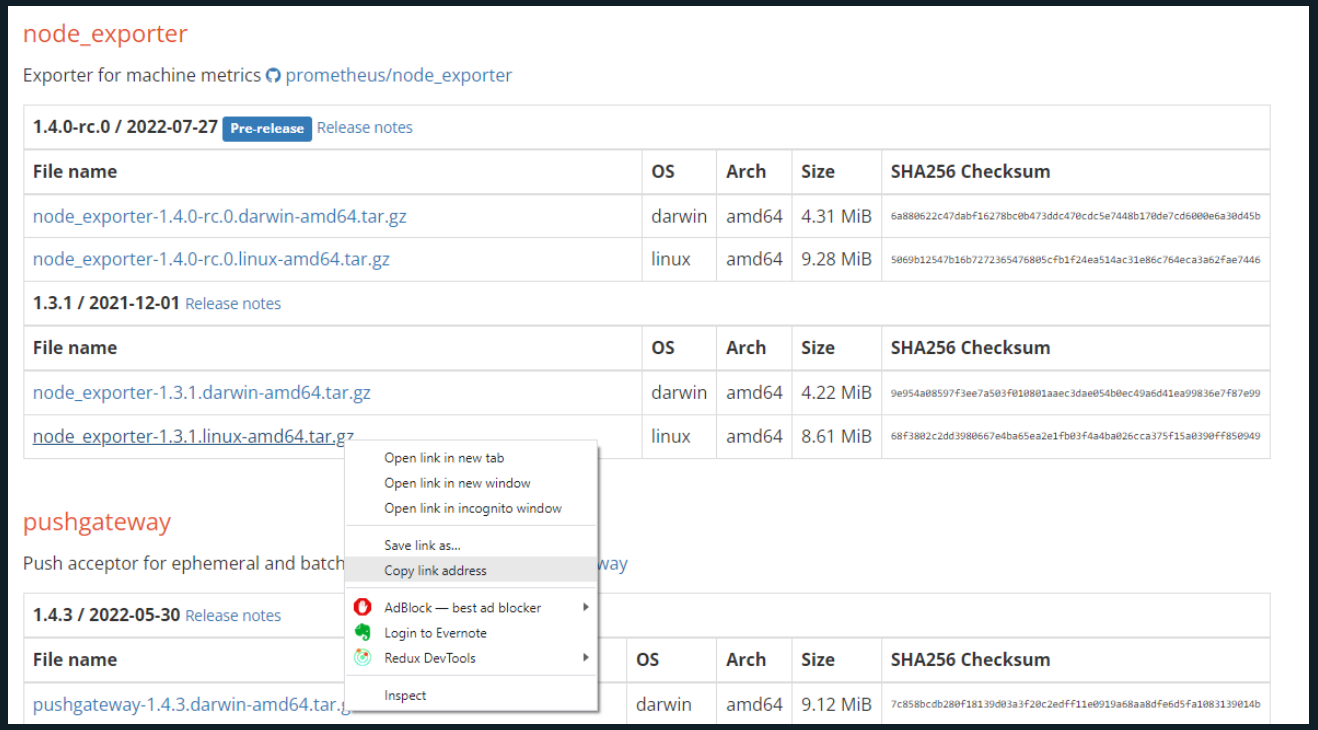
Node Exporter Installation

`http://prometheus.io/download`



Node Exporter Installation

```
Download binary our copy url
```



node_exporter

Exporter for machine metrics prometheus/node_exporter

1.4.0-rc.0 / 2022-07-27 Pre-release Release notes

File name	OS	Arch	Size	SHA256 Checksum
node_exporter-1.4.0-rc.0.darwin-amd64.tar.gz	darwin	amd64	4.31 MiB	6a880622c47dabf16278bc0b473ddc470cdc5e7448b170de7cd6000e6a30d45b
node_exporter-1.4.0-rc.0.linux-amd64.tar.gz	linux	amd64	9.28 MiB	5069b12547b16b72723654768805cfb1f24ea514ac31e86c764eca3a62fae7446

1.3.1 / 2021-12-01 Release notes

File name	OS	Arch	Size	SHA256 Checksum
node_exporter-1.3.1.darwin-amd64.tar.gz	darwin	amd64	4.22 MiB	9e954a08597f3ee7a503f010801aaec3dae054b0ec49a6d41ea99836e7f87e99
node_exporter-1.3.1.linux-amd64.tar.gz	linux	amd64	8.61 MiB	68f3802c2dd3980667e4ba65ea2e1fb03f4a4ba026cca375f15a0390ff850949

Open link in new tab
Open link in new window
Open link in incognito window
Save link as...
Copy link address
AdBlock — best ad blocker
Login to Evernote
Redux DevTools
Inspect

pushgateway

Push acceptor for ephemeral and batch

1.4.3 / 2022-05-30 Release notes

File name	OS	Arch	Size	SHA256 Checksum
pushgateway-1.4.3.darwin-amd64.tar.g	darwin	amd64	9.12 MiB	7c858bcdb280f18139d03a3f20c2edff11e0919a68aa8dfe6d5fa1083139014b

Node Exporter Installation

```
$ wget https://github.com/prometheus/node_exporter/releases/download/v1.3.1/node_exporter-1.3.1.linux-amd64.tar.gz
HTTP request sent, awaiting response... 200 OK
Length: 9033415 (8.6M) [application/octet-stream]
Saving to: ‘node_exporter-1.3.1.linux-amd64.tar.gz’

node_exporter-1.3.1.linux-amd64.tar.gz
100%[=====================================================================
=========>]   8.61M  12.4MB/s    in 0.7s

2022-09-02 15:04:10 (12.4 MB/s) - ‘node_exporter-1.3.1.linux-amd64.tar.gz’
saved [9033415/9033415]
```

Node Exporter Installation

```
$ tar -xvf node_exporter-1.3.1.linux-amd64.tar.gz
node_exporter-1.3.1.linux-amd64/
node_exporter-1.3.1.linux-amd64/LICENSE
node_exporter-1.3.1.linux-amd64/NOTICE
node_exporter-1.3.1.linux-amd64/node_exporter

$ cd node_exporter-1.3.1.linux-amd64

$ ls -l
-rw-r--r-- 1 user2 user2    11357 Dec  5  2021 LICENSE
-rwxr-xr-x 1 user2 user2 18228926 Dec  5  2021 node_exporter
-rw-r--r-- 1 user2 user2      463 Dec  5  2021 NOTICE
```

Node Exporter Installation

```
$ ./node_exporter
ts=2022-09-05T16:51:59.947Z caller=node_exporter.go:115 level=info collector=vmstat
ts=2022-09-05T16:51:59.947Z caller=node_exporter.go:199 level=info msg="Listening on" address=:9100
ts=2022-09-05T16:51:59.947Z caller=tls_config.go:195 level=info msg="TLS is disabled." http2=false
```

```
$ curl localhost:9100/metrics
# TYPE promhttp_metric_handler_requests_in_flight gauge

promhttp_metric_handler_requests_in_flight 1

# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status code.

# TYPE promhttp_metric_handler_requests_total counter

promhttp_metric_handler_requests_total{code="200"} 0

promhttp_metric_handler_requests_total{code="500"} 0

promhttp_metric_handler_requests_total{code="503"} 0
```

Node Exporter Installation Systemd

```
>_
```

```
$ ./node_exporter
```

Runs in Foreground

Doesn’t start on-boot

```
$ systemctl start node_exporter
```

Node Exporter Installation Systemd

```
>_
```

```
$ sudo cp node_exporter /usr/local/bin
```

Create User

```
$ sudo useradd --no-create-home --shell /bin/false node_exporter
```

Permissions

```
$ sudo chown node_exporter:node_exporter /usr/local/bin/node_exporter
```

Node Exporter Installation Systemd

```
>_
```

```
$ sudo vi /etc/systemd/system/node_exporter.service
```

>_ node_exporter.service

```
[Unit]
Description=Node Exporter
Wants=network-online.target
After=network-online.target

[Service]
User=node_exporter
Group=node_exporter
Type=simple
ExecStart=/usr/local/bin/node_exporter

[Install]
WantedBy=multi-user.target
```

Start service after network is up

Start service as part of normal system start-up, whether or not local GUI is active

Node Exporter Installation Systemd

```
$ sudo systemctl daemon-reload
```

Node Exporter Installation Systemd

```
$ sudo systemctl start node_exporter
```

```
$ sudo systemctl status node_exporter
● node_exporter.service - Node Exporter
     Loaded: loaded (/etc/systemd/system/node_exporter.service; disabled; vendor preset: enabled)
     Active: active (running) since Mon 2022-09-05 13:29:45 EDT; 23s ago
   Main PID: 20222 (node_exporter)
      Tasks: 3 (limit: 8247)
     Memory: 2.1M
        CPU: 7ms
     CGroup: /system.slice/node_exporter.service
             └─20222 /usr/local/bin/node_exporter
```

Node Exporter Installation Systemd

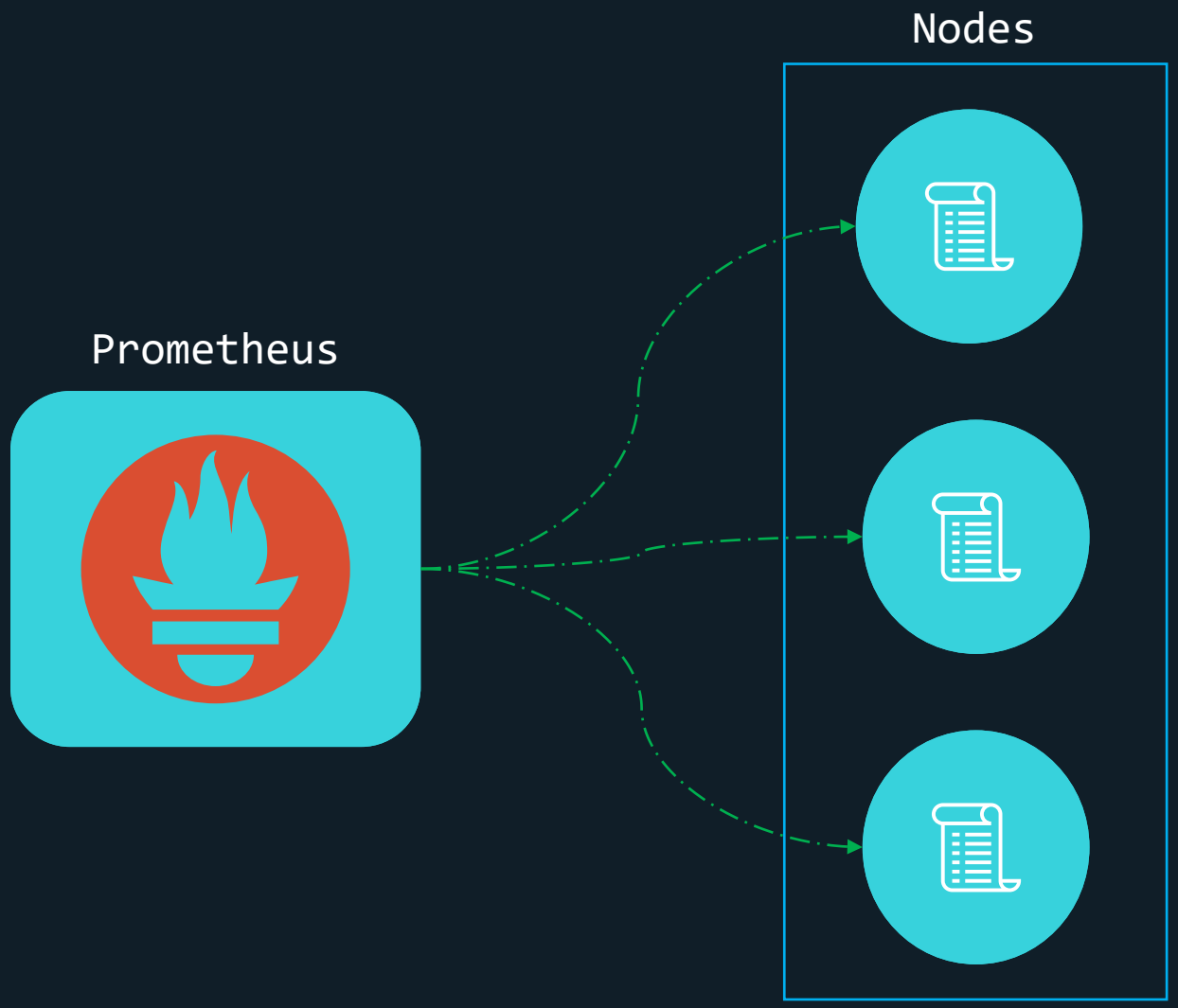
```
$ sudo systemctl enable node_exporter
```

Start service on-boot

```
$ sudo systemctl status node_exporter
● node_exporter.service - Node Exporter
     Loaded: loaded (/etc/systemd/system/node_exporter.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2022-09-05 13:29:45 EDT; 23s ago
   Main PID: 20222 (node_exporter)
      Tasks: 3 (limit: 8247)
     Memory: 2.1M
        CPU: 7ms
     CGroup: /system.slice/node_exporter.service
             └─20222 /usr/local/bin/node_exporter
```

Configuration

Prometheus Configuration



prometheus.yml

```
# my global config

global:


scrape_configs:

  - job_name: "prometheus"

    static_configs:

      - targets: ["localhost:9090"]
```

Prometheus Configuration

prometheus.yml

```yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s

scrape_configs:
  - job_name: 'node'
    scrape_interval: 15s
    scrape_timeout: 5s
    sample_limit: 1000
    static_configs:
      - targets: ['172.16.12.1:9090']

# Configuration related to AlertManager
alerting:

# Rule files specifies a list of files rules are read from
rule_files:

# Settings related to the remote read/write feature.
remote_read:
remote_write:

# Storage related settings
storage:
```

Default parameters for all other config sections

Define targets & configs for metrics collection

A collection of instances that need to be scraped

Configs for scrape job. Takes precedence over global config

Set of targets to scrape

Prometheus Configuration

1. Define a Job with a name of nodes
2. Metrics should be scraped every 30s
3. Timeout of a scrape is 3s
4. Use HTTPS instead of default HTTP
5. Scrape path should be changed from default /metrics to /stats/metrics
6. Scrape two targets at the following IP
    1. 10.231.1.2:9090
    2. 192.168.43.8:9090

>_ prometheus.yml

```
scrape_configs:
  - job_name: 'nodes'
    scrape_interval: 30s
    scrape_timeout: 3s
    scheme: https
    metrics_path: /stats/metrics
    static_configs:
      - targets: ['10.231.1.2:9090', '192.168.43.9:9090']
```

Scrape Config Options

>_ prometheus.yml

```
scrape_configs:
  # How frequently to scrape targets from this job.
    [ scrape_interval: <duration> | default =     <global_config.scrape_interval> ]

  # Per-scrape timeout when scraping this job.
    [ scrape_timeout: <duration> | default = <global_config.scrape_timeout> ]

  # The HTTP resource path on which to fetch metrics from targets.
    [ metrics_path: <path> | default = /metrics ]

  # Configures the protocol scheme used for requests.
    [ scheme: <scheme> | default = http ]

  # Sets the `Authorization` header on every scrape request with the
  # configured username and password.
  # password and password_file are mutually exclusive.
  basic_auth:
    [ username: <string> ]
    [ password: <secret> ]
    [ password_file: <string> ]
```

Restart Prometheus

```
>_
```

```
$ ctrl+c  ->  ./prometheus
```

```
$ kill -HUP <pid>
```

```
$ systemctl restart prometheus
```

Reloading Configuration

After making changes to the Prometheus confiigs, a reload of the configuration needs to take place for changes to take affect

1. Restart Prometheus

   ```
   # systemctl restart prometheus
   ```

2. Send a SIGHUP signal to the Prometheus process

   ```
   # sudo killall -HUP prometheus
   ```

3. Send a POST/PUT request to http://<prometheus>/-/reload
   - This functionality not enabled by default
   - Need to start prometheus with --web.enable-lifecycle flag

Reloading Configuration

>_ Prometheus.service

```
[Unit]
Description=Prometheus
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
    --config.file /etc/prometheus/prometheus.yml \
    --storage.tsdb.path /var/lib/prometheus/ \
    --web.console.templates=/etc/prometheus/consoles \
    --web.console.libraries=/etc/prometheus/console_libraries \
    --web.enable-lifecycle
[Install]
WantedBy=multi-user.target
```

Reload Configuration

```
>_
$ sudo systemctl daemon-reload

$ sudo systemctl restart prometheus

$ curl -X POST http://<prometheus>/-/reload
```

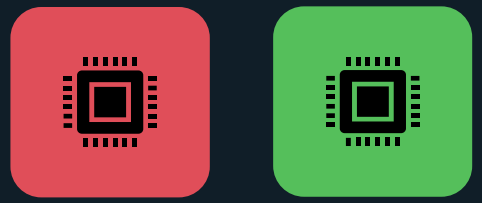
Metrics

Prometheus Metrics

<metric_name>[{<label_1="value_1">,<label_N="value_N">}] <metric_value>

node_cpu_seconds_total{cpu="0",mode="idle"} 258277.86

Labels provide us information on which cpu this metric is for and for what cpu state(idle)



node_cpu_seconds_total{cpu="0",mode="idle"} 258244.86
node_cpu_seconds_total{cpu="1",mode="idle"} 427262.54
node_cpu_seconds_total{cpu="2",mode="idle"} 283288.12
node_cpu_seconds_total{cpu="3",mode="idle"} 258202.33

Prometheus Metrics

```
node_cpu_seconds_total{cpu="0",mode="idle"} 258277.86
node_cpu_seconds_total{cpu="0",mode="iowait"} 61.16
node_cpu_seconds_total{cpu="0",mode="irq"} 0
node_cpu_seconds_total{cpu="0",mode="nice"} 61.12
node_cpu_seconds_total{cpu="0",mode="softirq"} 6.63
node_cpu_seconds_total{cpu="0",mode="steal"} 0
node_cpu_seconds_total{cpu="0",mode="system"} 372.46
node_cpu_seconds_total{cpu="0",mode="user"} 3270.86
```

top

```
top - 14:28:10 up 3 days,  1:49,  1 user,  load
average: 0.24, 0.09, 0.09

%Cpu(s): 32.6 us,  1.4 sy,  0.0 ni, 66.0 id,
0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

Timestamp

When Prometheus scrapes a target and retrieves metrics, it also stores the time at which the metric was scraped as well.

The timestamp will look like this:

1668215300

This is called a unix timestamp, which is the number of seconds that have elapsed since Epoch(January 1st 1970 UTC)

Prometheus Metrics

<metric_name>[{<label_1="value_1">,<label_N="value_N">}] <metric_value>

node_cpu_seconds_total{cpu="0",mode="idle"} 258277.86 Jan 1, 12:51

Labels provide us information on which cpu this metric is for and for what cpu state(idle)

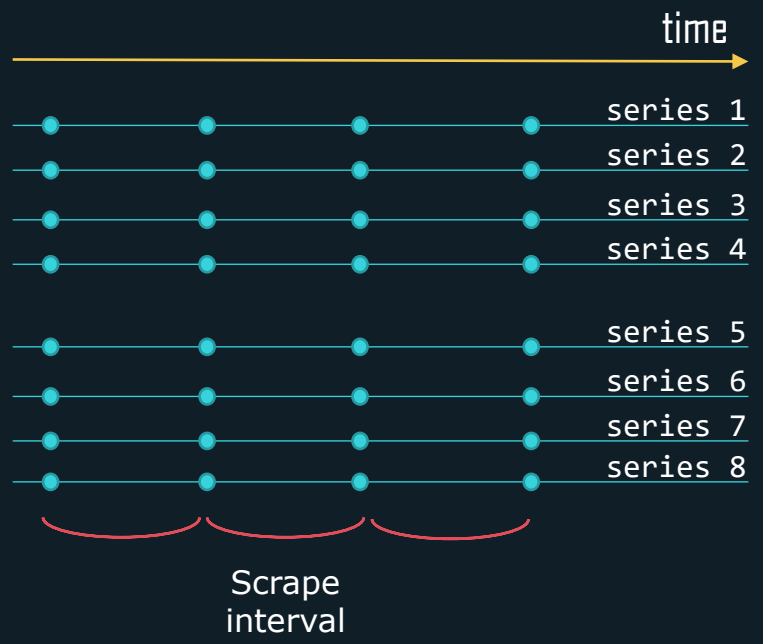
node_cpu_seconds_total{cpu="0",mode="idle"} 258244.86
node_cpu_seconds_total{cpu="1",mode="idle"} 427262.54
node_cpu_seconds_total{cpu="2",mode="idle"} 283288.12
node_cpu_seconds_total{cpu="3",mode="idle"} 258202.33

Prometheus Time Series

Stream of timestamped values sharing the same metric and set of labels

node_filesystem_files{device="sda2", instance="server1"}
node_filesystem_files{device="sda3", instance="server1"}
node_filesystem_files{device="sda2", instance="server2"}
node_filesystem_files{device="sda3", instance="server2"}

node_cpu_seconds_total{cpu="0", instance="server1"}
node_cpu_seconds_total{cpu="1", instance="server1"}
node_cpu_seconds_total{cpu="0", instance="server2"}
node_cpu_seconds_total{cpu="1", instance="server2"}



There are two metrics(node_filesystem_files, node_cpu_seconds_total)

There are 8 total time series(unique combination of metric and set of labels

Metric Attributes

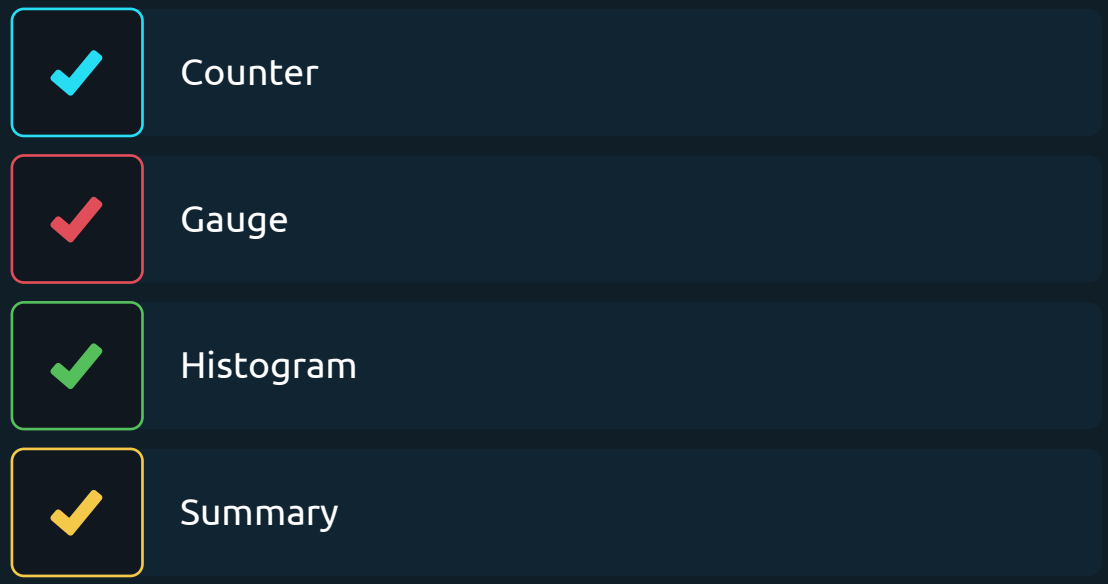
Metrics have a TYPE and HELP attribute

```
# HELP node_disk_discard_time_seconds_total This is the total number of seconds spent by all discards.
# TYPE node_disk_discard_time_seconds_total counter
node_disk_discard_time_seconds_total{device="sda"} 0
node_disk_discard_time_seconds_total{device="sr0"} 0
```

HELP – description of what the metric is

TYPE – Specifies what type of metric(counter, gauge, histogram, summary)

Metric Types



Counter

Counter

- ✓ How many times did X happen
- ✓ Number can only increase

Total # requests

Total # Exceptions

Total # of job executions

Gauge

Gauge

- ✓ What is the current value of X
- ✓ Can go up or down

Current CPU Utilization

Available System Memory

Number of concurrent requests

Histogram

Histogram

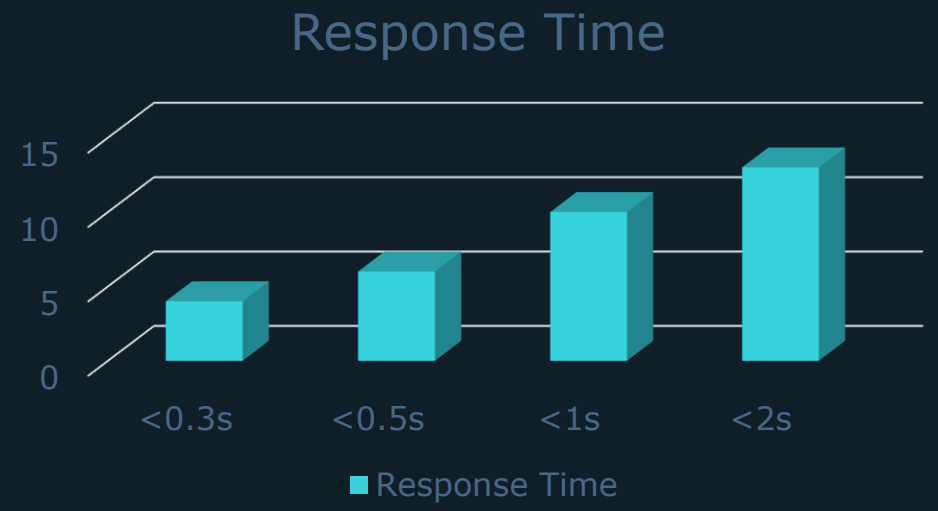
- ✓ How long or how big something is
- ✓ Groups observations into configurable bucket sizes

Response Time

< 1s
< 0.5s
< 0.2s

Request size

< 1500Mb
< 1000Mb
< 800Mb



Summary

Summary

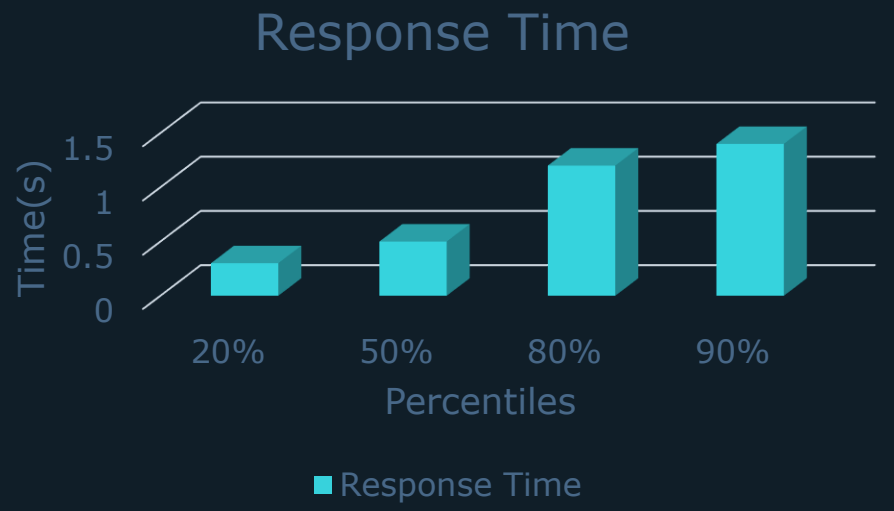
- ✓ Similar to histograms(track how long or how big)
- ✓ How many observations fell below x
- ✓ Don’t have to define quantiles ahead of time

Response Time

20% = .3s
50% = 0.8s
80% = 1s

Request size

20% = 50Mb
50% = 200Mb
80% = 500Mb



Metric Rules

Metric name specifies a general feature of a system to be measured

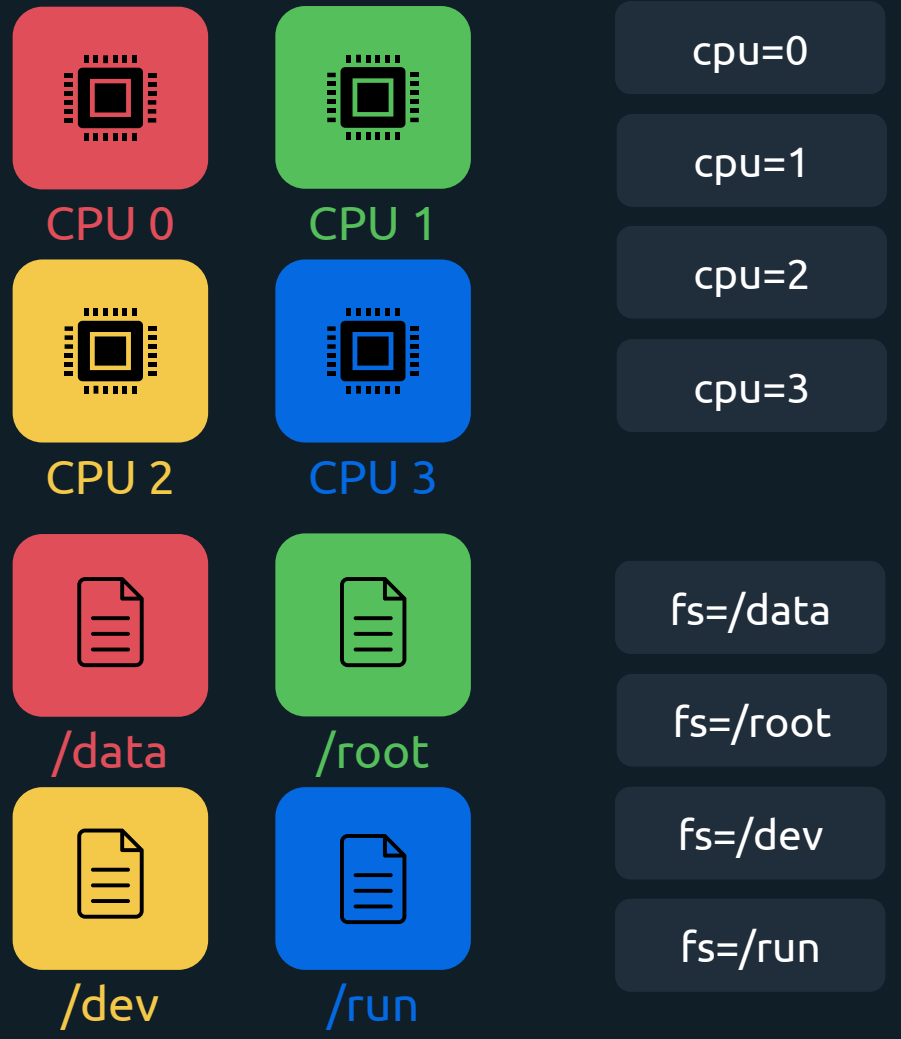
May contain ASCII letters, numbers, underscores, and colons

Must match the regex [a-zA-Z_:][a-zA-Z0-9_:]*

Colons are reserved only for recording rules

Labels

- Labels are Key-Value pairs associated with a metric
- Allows you to split up a metric by a specified criteria
- Metric can have more than one label
- ASCII leters, numbers, underscores
- Must match regex [a-zA-Z0-9_]*



Why Labels

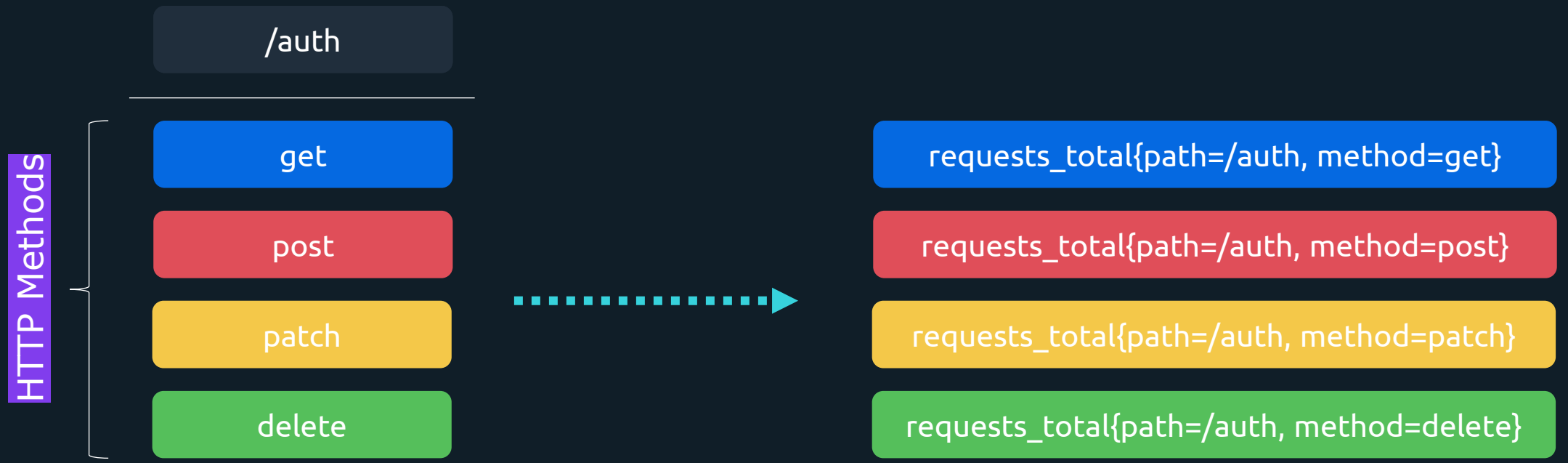
API for e-commerce app

Difficult to calculate total requests across all paths

Sum all requests: sum(requests_total)

/auth	requests_auth_total	requests_total{path=/auth}
/user	requests_user_total	requests_total{path=/user}
/products	requests_products_total	requests_total{path=/products}
/cart	requests_cart_total	requests_total{path=/cart}
/orders	requests_orders_total	requests_total{path=/orders}

Multiple Labels



Internal Labels

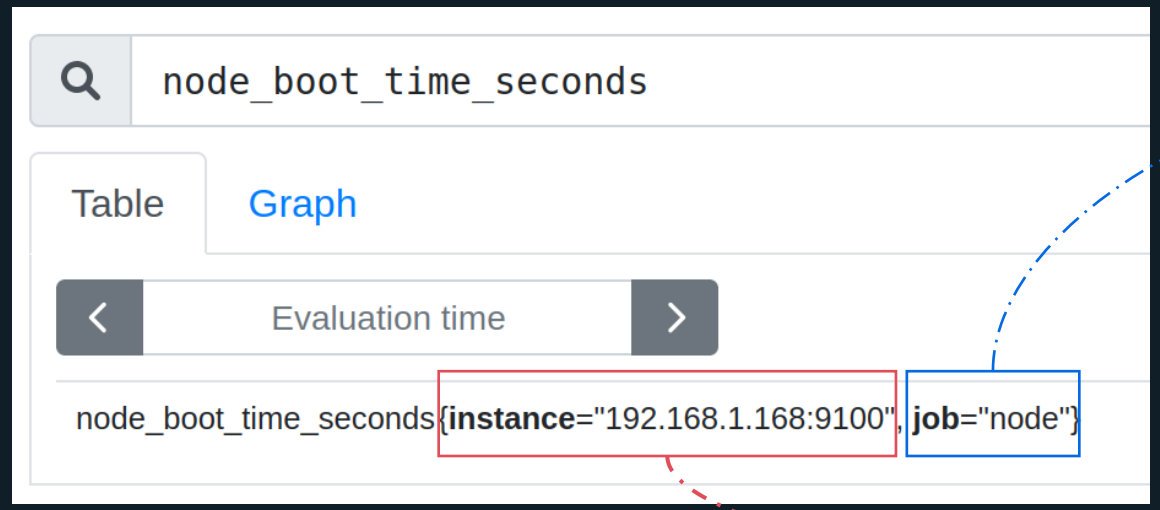
Metric name is just another label

`node_cpu_seconds_total{cpu=0}` = `{__name__=node_cpu_seconds_total, cpu=0}`

```
Labels surrounded by __ are considered
        internal to prometheus
```

Labels

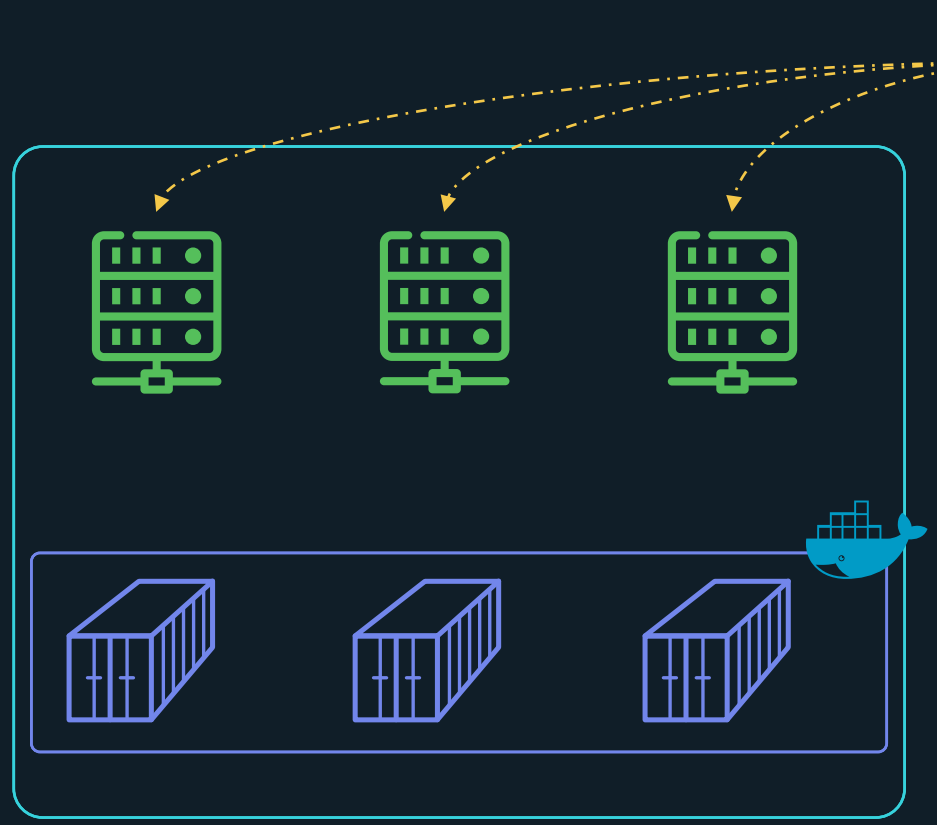
Every metric is assigned 2 labels by default(instance and job)




```
>_
  - job_name: "node"
    scheme: https
    basic_auth:
      username: prometheus
      password: password
    static_configs:
      - targets:
["192.168.1.168:9100"]
```

Monitoring Containers

Container Metrics



cAdvisor

- ✓ Metrics can also be scraped from containerized environments
- ✓ Docker Engine Metrics
- ✓ Container metrics using cAdvisor

Docker Engine metrics

```
$ vi /etc/docker/daemon.json

$ sudo systemctl restart docker

$ curl localhost:9323/metrics
```

daemon.json

```
{
  "metrics-addr" : "127.0.0.1:9323",
  "experimental" : true
}
```

Docker Engine Metrics

>_ Prometheus.yml

```yaml
scrape_configs:
  - job_name: "docker"
    static_configs:
      - targets: ["<ip-docker-host>:9323"]
```

cAdvisor Metrics

```
$ vi docker-compose.yml

$ docker-compose up

$ curl localhost:8080/metrics
```

More Info

https://github.com/google/cadvisor

docker-compose.yml

```yaml
version: '3.4'
services:
  cadvisor:
    image: gcr.io/cadvisor/cadvisor
    container_name: cadvisor
    privileged: true
    devices:
      - "/dev/kmsg:/dev/kmsg"
    volumes:
      - /:/rootfs:ro
      - /var/run:/var/run:ro
      - /sys:/sys:ro
      - /var/lib/docker/:/var/lib/docker:ro
      - /dev/disk/:/dev/disk:ro
    ports:
      - 8080:8080
```

cAdvisor Metrics

>_ Prometheus.yml

```yaml
scrape_configs:
  - job_name: "cAdvisor"
    static_configs:
      - targets: ["<docker-host-ip>:8080"]
```

Docker Metrics vs cAdvisor Metrics

Docker Engine metrics

- ✓ How much cpu does docker use
- ✓ Total number of failed image builds
- ✓ Time to process container actions
- ✓ No metrics specific to a container

cAdvisor metrics

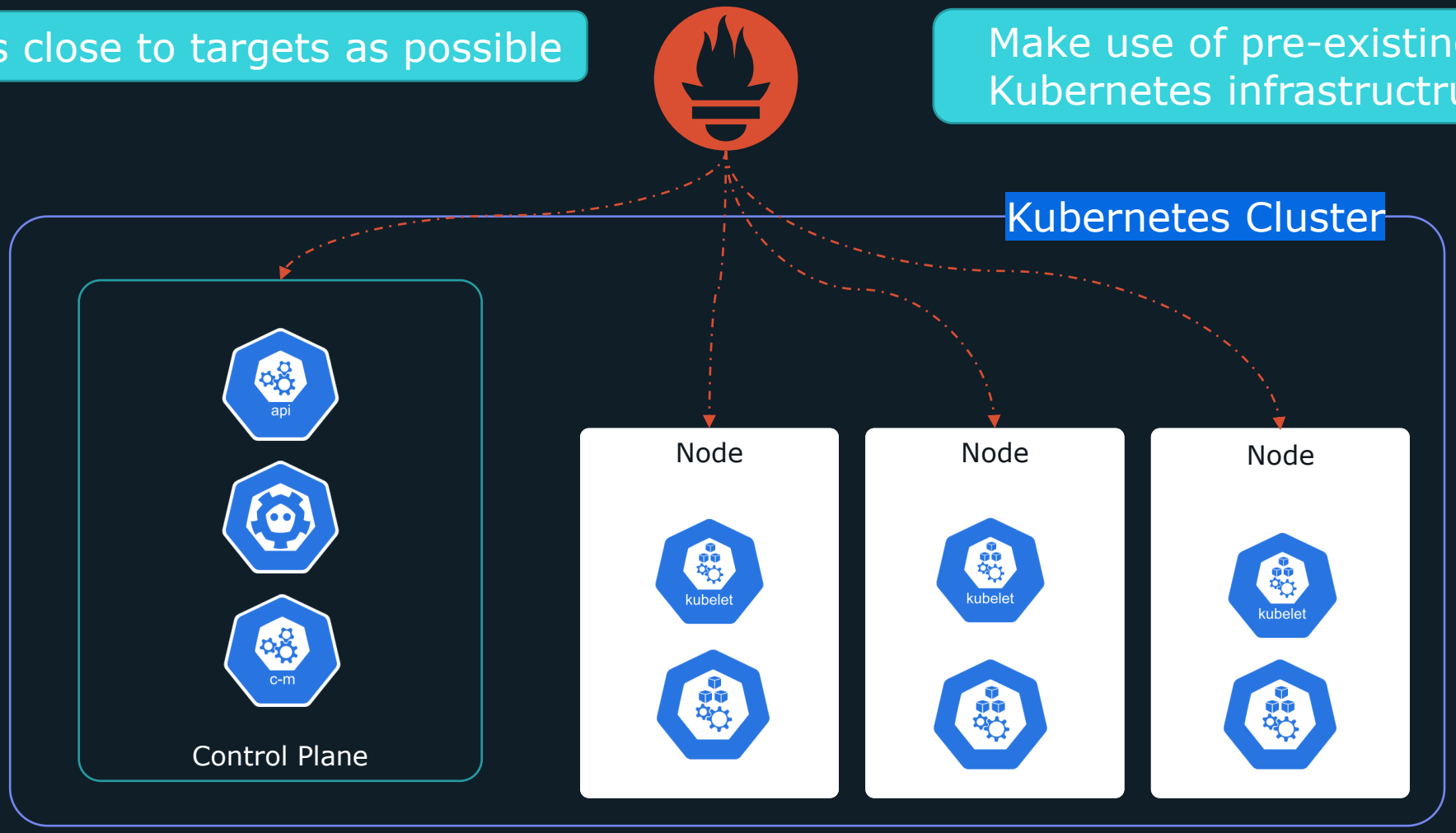
- ✓ How much cpu/mem does each container use
- ✓ Number of processes running inside a container
- ✓ Container uptime
- ✓ Metrics on a per container basis

Kubernetes

Kubernetes

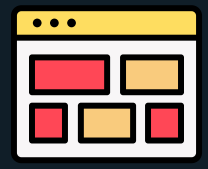
Deploy as close to targets as possible

Make use of pre-existing Kubernetes infrastructrue



Kubernetes

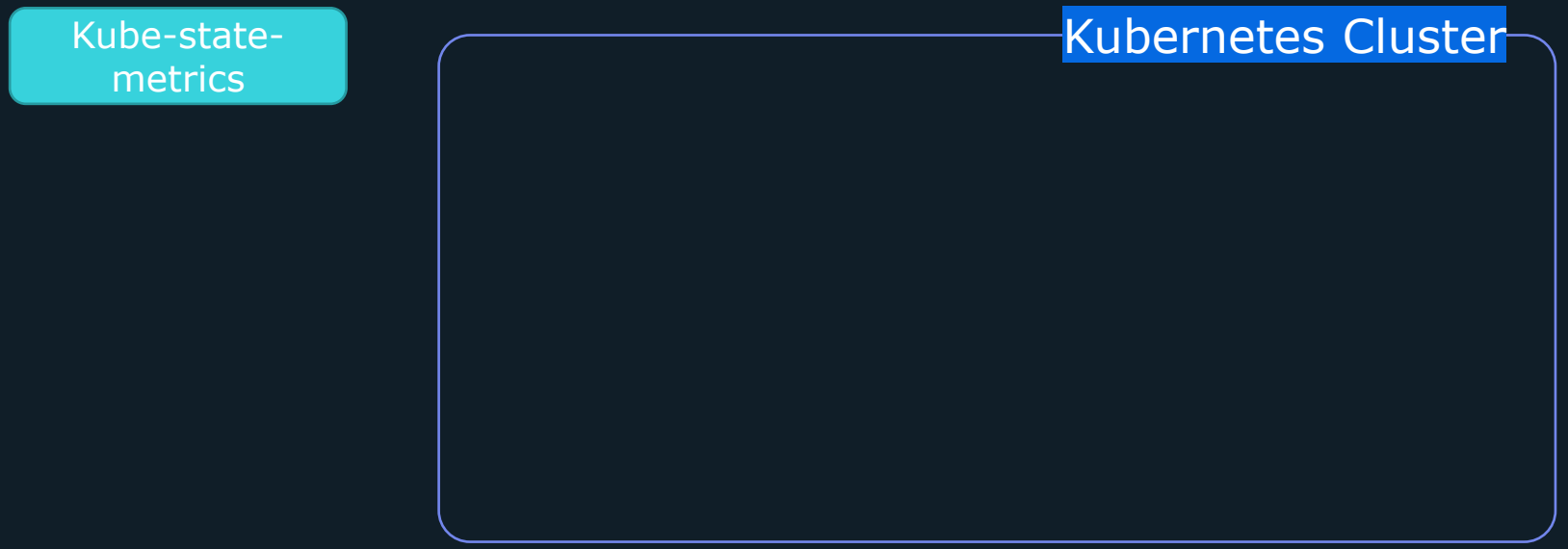
- Monitor applications running on Kubernetes infrastructure



- Monitor Kubernetes Cluster
  - Control-Plane Components(api-server, coredns, kube-scheduler)
  - Kubelet(cAdvisor) – exposing container metrics
  - Kube-state-metrics – cluster level metrics(deployments, pod metrics)
  - Node-exporter – Run on all nodes for host related metrics(cpu, mem, network)

Kube-state-metrics

To collect cluster level metrics(pods, deployments, etc) the kube-state-metrics container must be deployed

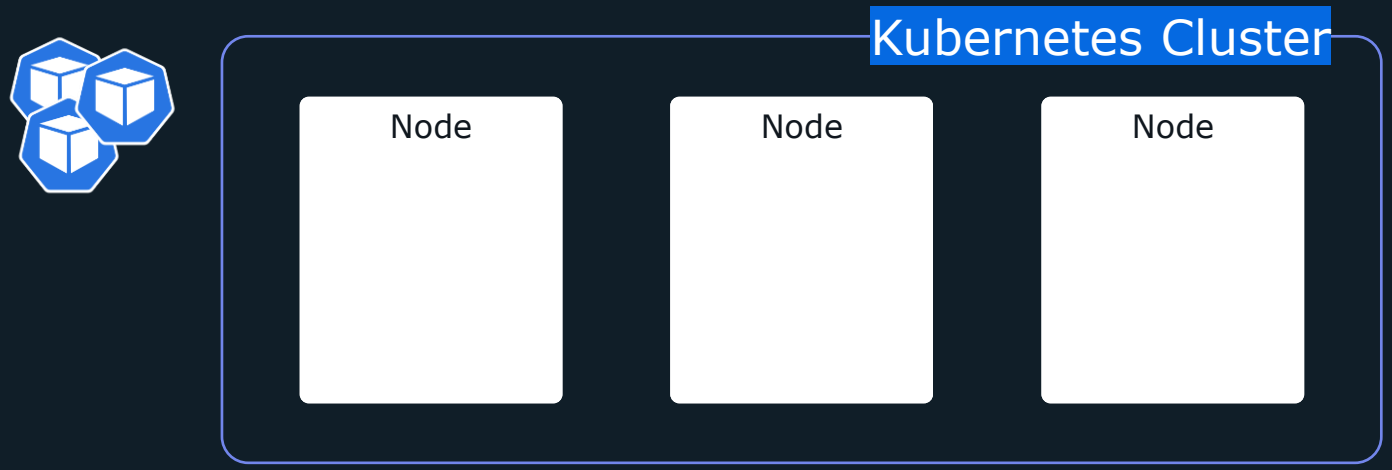


Node Exporter

Every host should run a node_exporter to expose cpu, memory, and network stats

We can manually go in an install a node_exporter on every node

Better option is to use a Kubernetes daemonSet - pod that runs on every node in the cluster

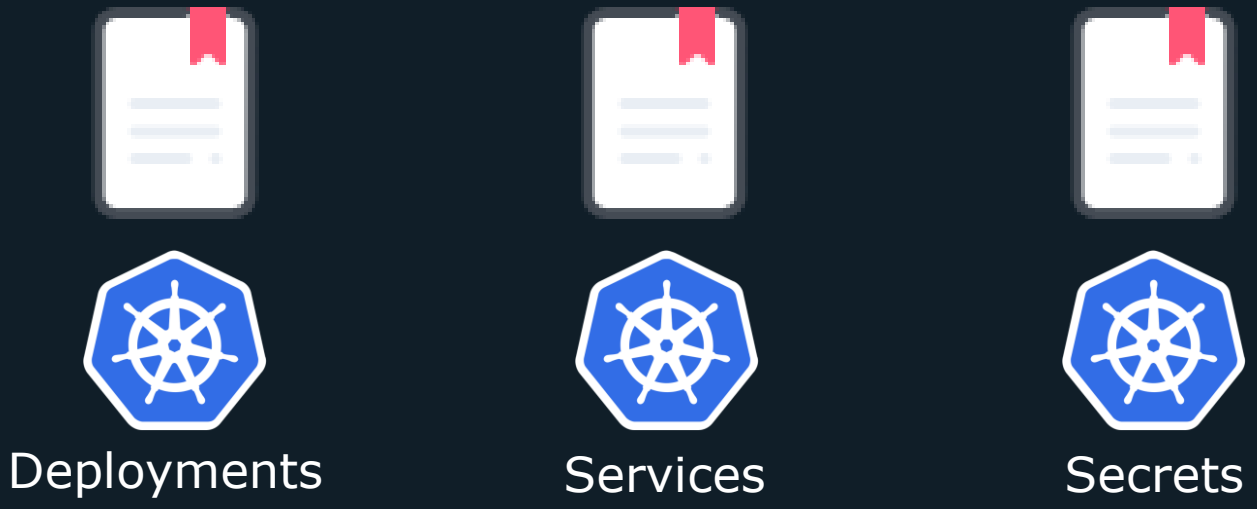


Service Discovery

How to deploy

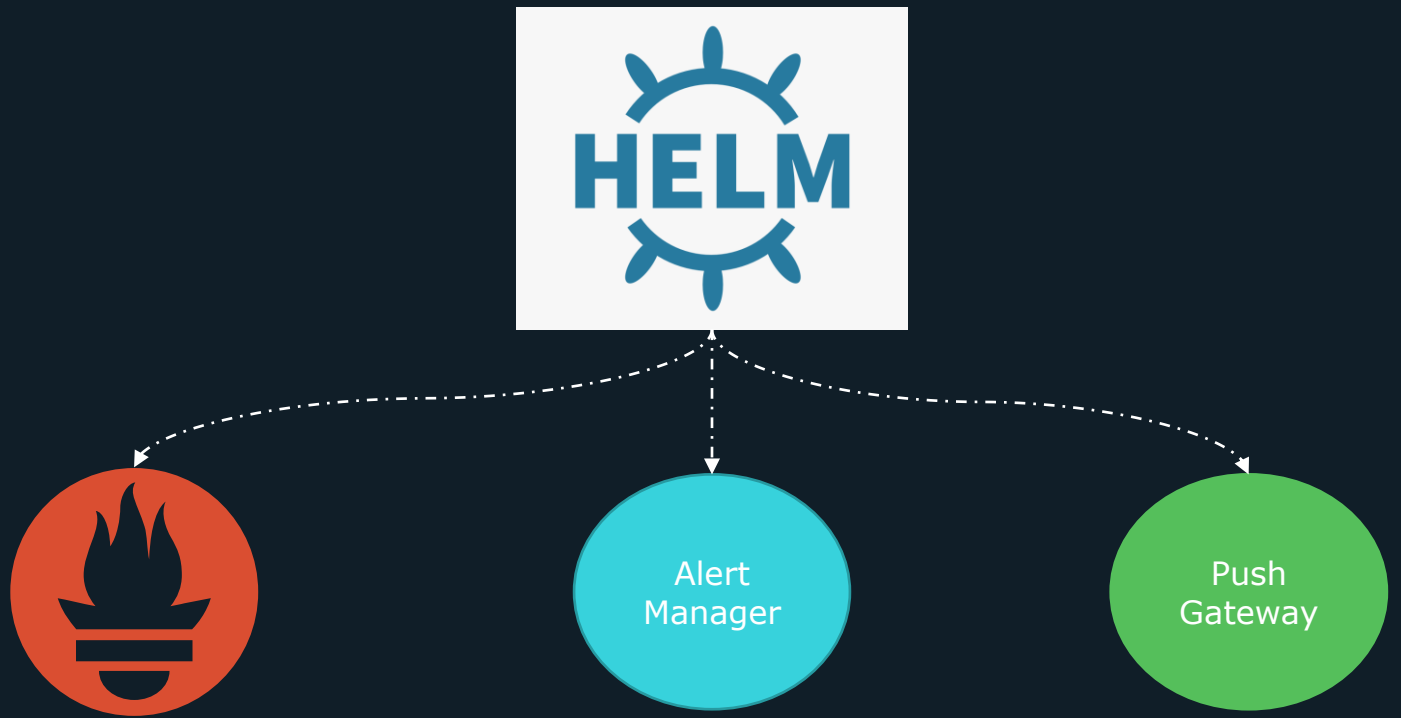
Manually deploy Prometheus on Kubernetes – Create all the deployments, services, configMaps, and secrets

Complex, requires a lot of configuration, not the easiest solution



How to deploy

Best way to deploy Prometheus is using Helm chart to deploy Prometheus operator



What is Helm

- Helm is a package manager for Kubernetes
- All application and Kubernetes configs necessary for an application can be bundled into a package and easily deployed

```
$ helm install
```

Helm Charts

A helm chart is a collection of template & YAML files that convert into Kubernetes manifest files

Helm charts can be shared with others by uploading a chart to a repository

https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack

Repository: Prometheus-community
Chart: kube-Prometheus-stack

Kube-Prometheus-stack

The Kube-Prometheus-stack chart makes use of the Prometheus Operator

A Kubernetes operator is an application-specific controller that extends the K8s API to create/configure/manage instances of complex applications(like Prometheus!)

https://github.com/prometheus-operator/prometheus-operator

Prometheus Operator

The Prometheus operator has several custom resources to aid the deployment and management of a Prometheus instance

- Alertmanager
- Prometheus Rule
- Alertmanager Config
- ServiceMonitor
- PodMonitor

>_ prometheus.yml

```yaml
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  annotations:
    meta.helm.sh/release-name: prometheus
    meta.helm.sh/release-namespace: default
  creationTimestamp: "2022-11-18T01:19:29Z"
  generation: 1
  labels:
    app: kube-prometheus-stack-prometheus
  name: prometheus-kube-prometheus-prometheus
spec:
  alerting:
    alertmanagers:
    - apiVersion: v2
      name: prometheus-kube-prometheus-alertmanager
      namespace: default
      pathPrefix: /
      port: http-web
```