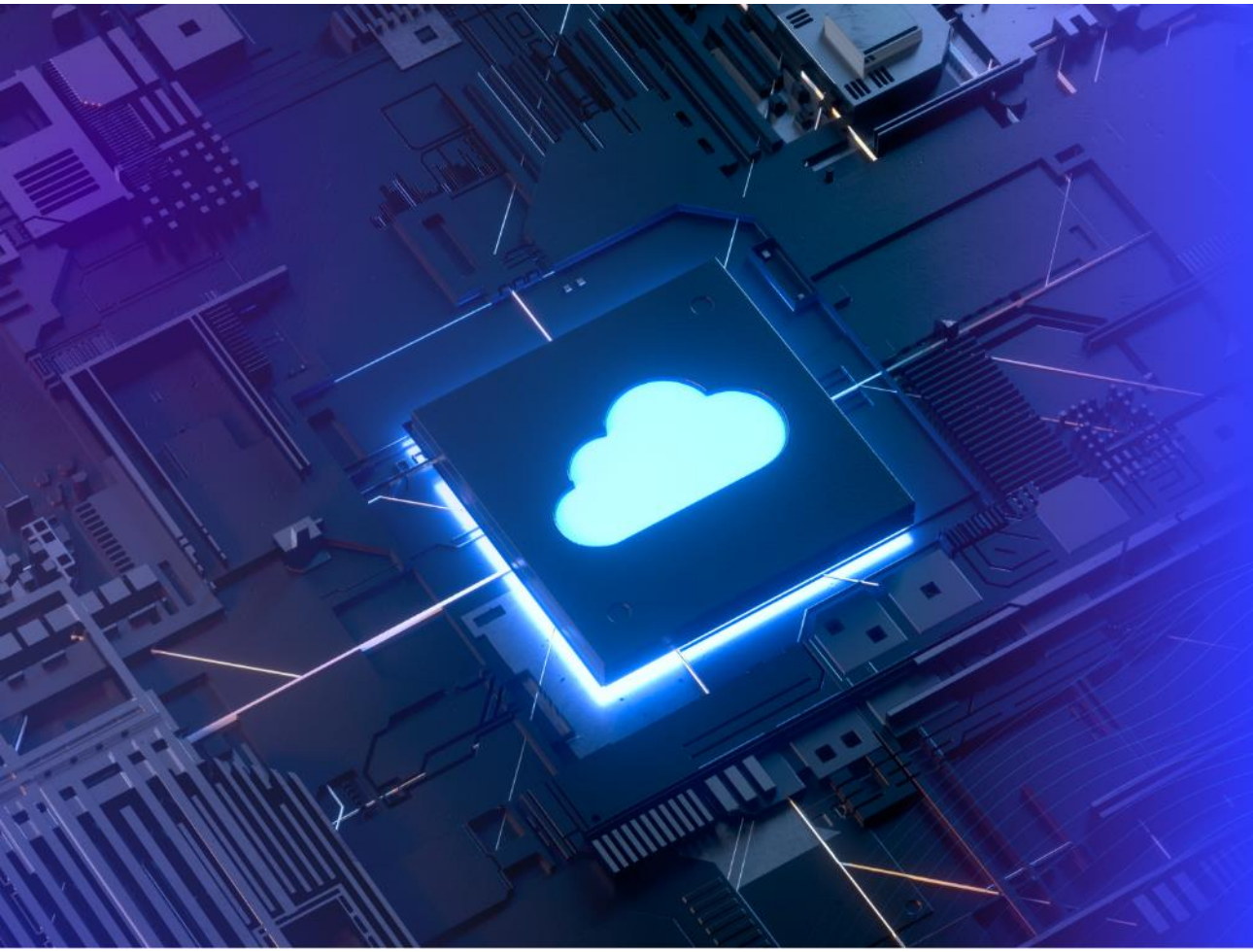
台灣積體電路製造股份有限公司
Taiwan Semiconductor Manufacturing Company, Ltd.



TSMC IT × NCTU CS 課號 5270

CLOUD NATIVE
Development Best Practice

MONITOR & OBSERVATION

資料及平台部 | 詹文志/林家民

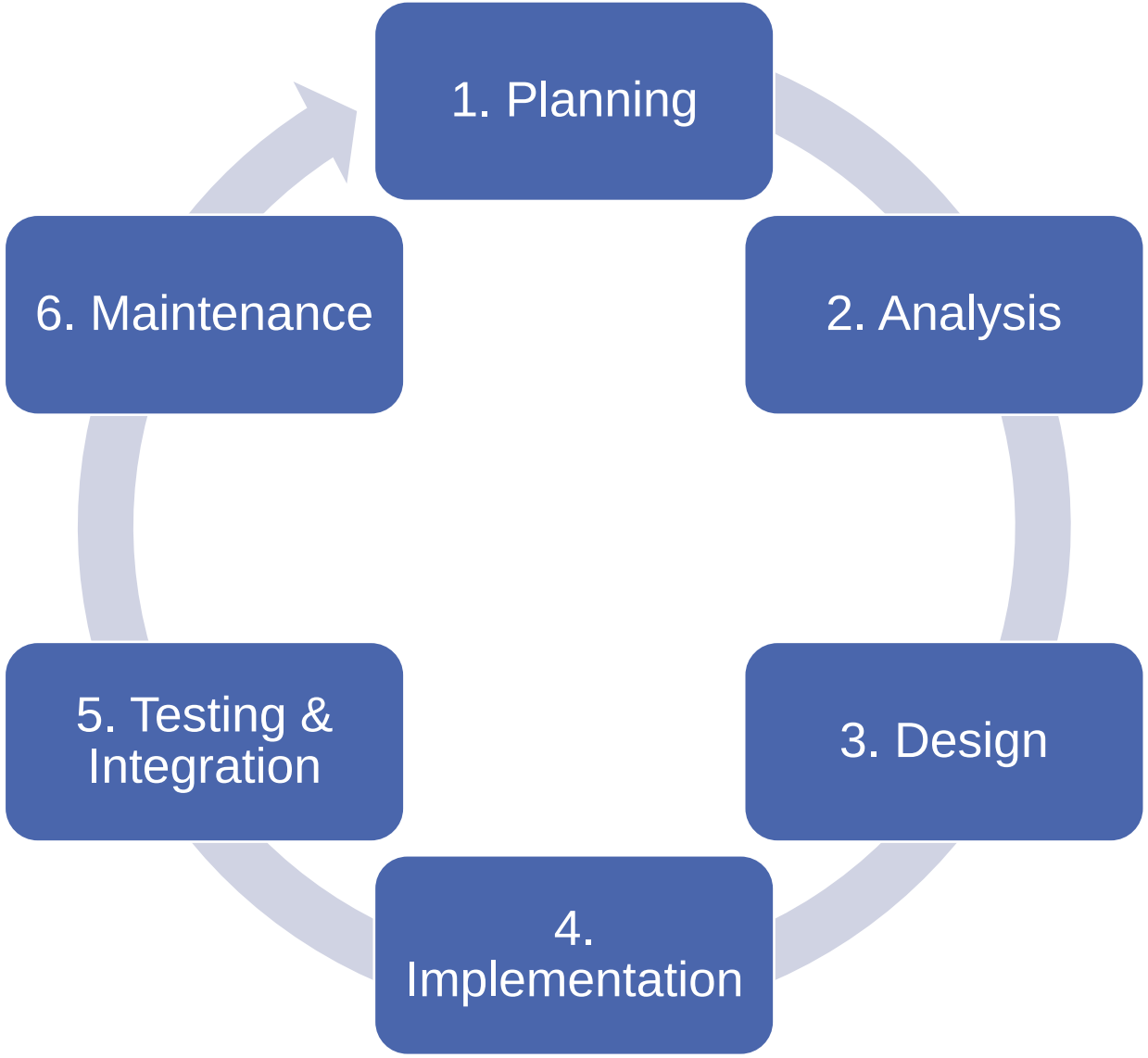
2022

詹文志 自我介紹

- 中德電子, 世界先進 CIM Engineer
- 2004 Join TSMC
  - Oracle Database Administrator.
  - Database monitor system development.
- 2011 中科
  - Operation team
- 2017 南京
  - Operation team.
- 2019 中科
  - IT Monitor/Workflow system development.
  - SRE

林家民 自我介紹

- 2006 - Openfind (網擎資訊)
  - Software Engineer in Search Engine Dept.
- 2011 加入 TSMC
  - F14 MES Operation Team
- 2019
  - Operation Excellent Program
- 2021
  - Platform Team SRE



Software Development Life Cycle (SDLC)

課堂事前準備

1.(Recommended) 安裝 docker engine: https://docs.docker.com/compose/install/
    1. Windows, Mac: 安裝對應的 docker desktop
    2. Linux: 根據 platform & architecture 選擇對應的安裝方式
2.(Recommended) 安裝 docker compose
    1. Windows, Mac: 已經包含在 docker desktop
    2. Linux: https://docs.docker.com/compose/install/
3.(Optional) 安裝 Python

https://hackmd.io/@jeremy-wang-lin/S1oqMEZBc#

Exercise

- Deploy Prometheus, Grafana
  - Dashboard import.
  - Node exporter
  - Create Grafana dashboard
- Export metrics
  - demo code exporter

AGENDA

SRE

Why Monitor ?

What to monitor

How to monitor

K8S Monitor Stacks

- Prometheus
- Grafana
- OpenTelemtry

Site Reliability Engineering (SRE)

- Traditionally, goals of the development and operation teams are often conflict.
- SRE team is a team of people who
  - will quickly become bored by performing tasks by hand
  - have the skill set necessary to write software to replace their previously manual work
- class SRE implements DevOps.
- SRE team is responsible for the **availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning** of their service(s)

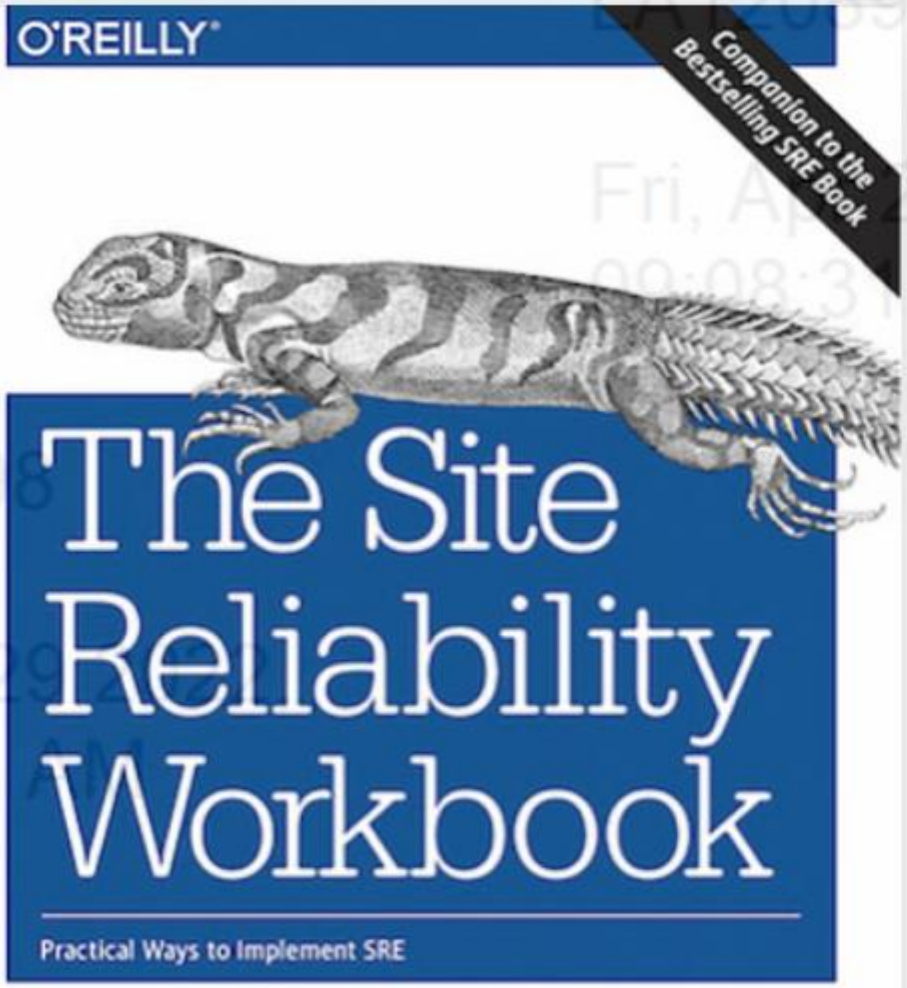
How To Measure The SRE ?

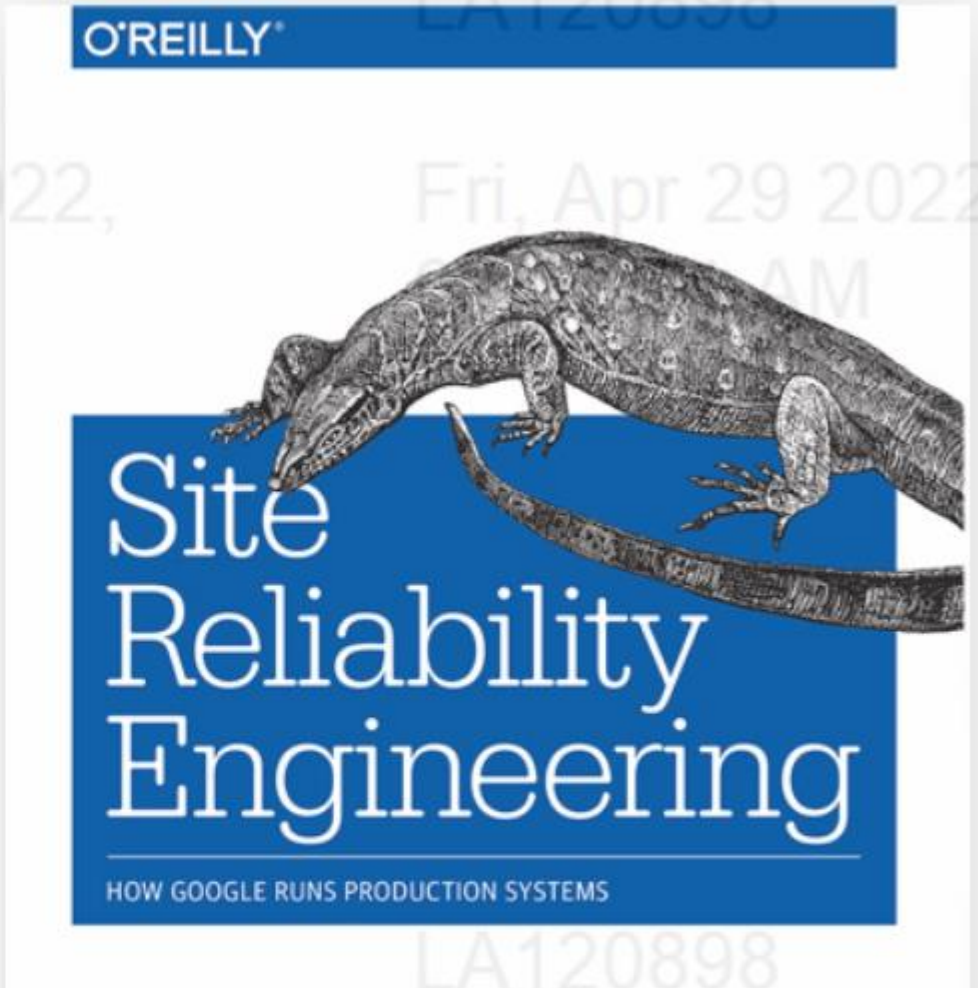
SLI drive SLO which inform SLA

95th percentile latency of service requests past 3 minutes < 300ms

Achieve 99.9% service SLI yearly

Service credits if SLI not achieve 99.5%





https://sre.google/books/

AGENDA

SRE

Why Monitor

What to monitor

How to monitor

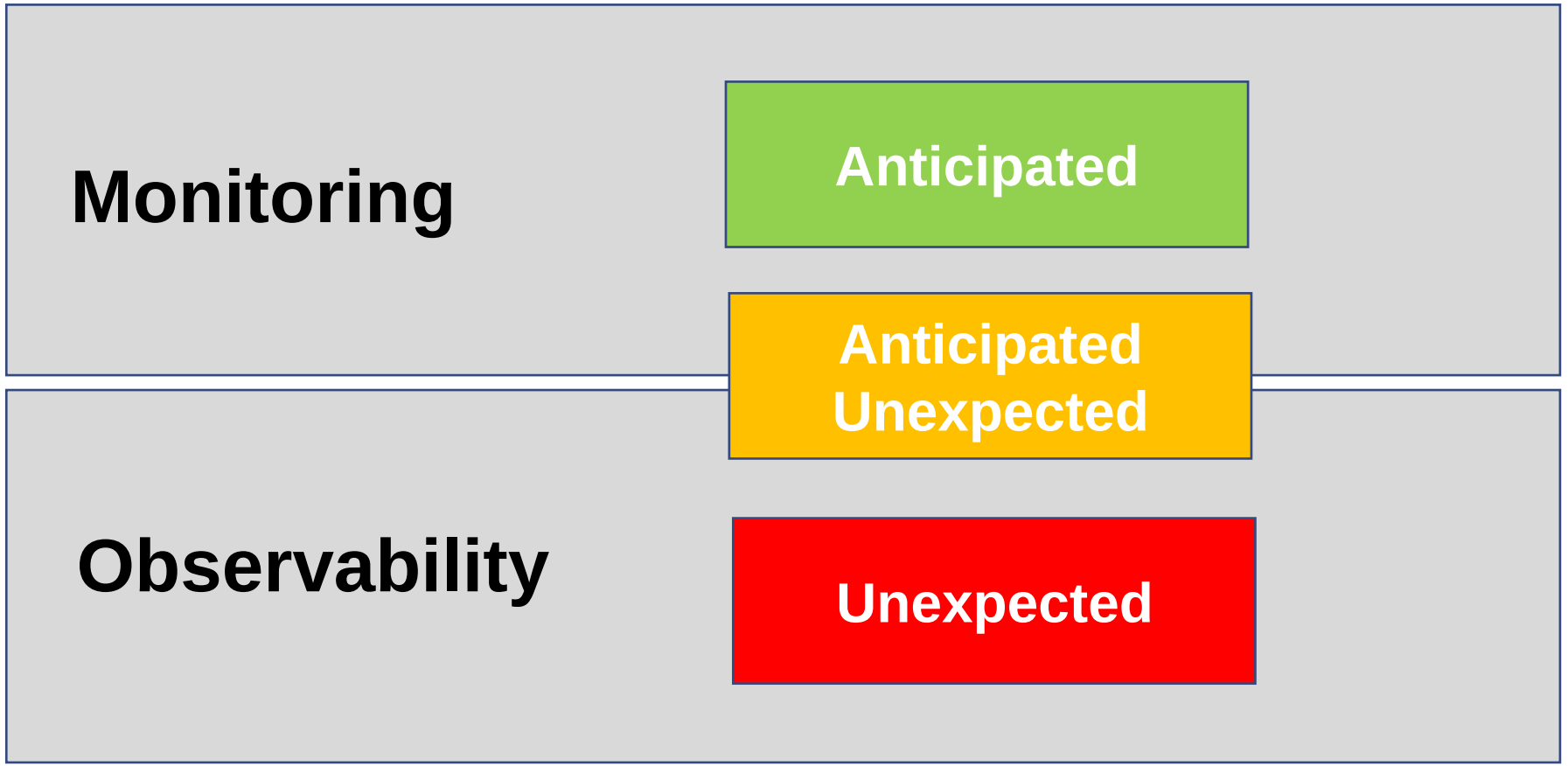
K8S Monitor Stacks

- Prometheus
- Grafana
- OpenTelemtry

Why Monitor?

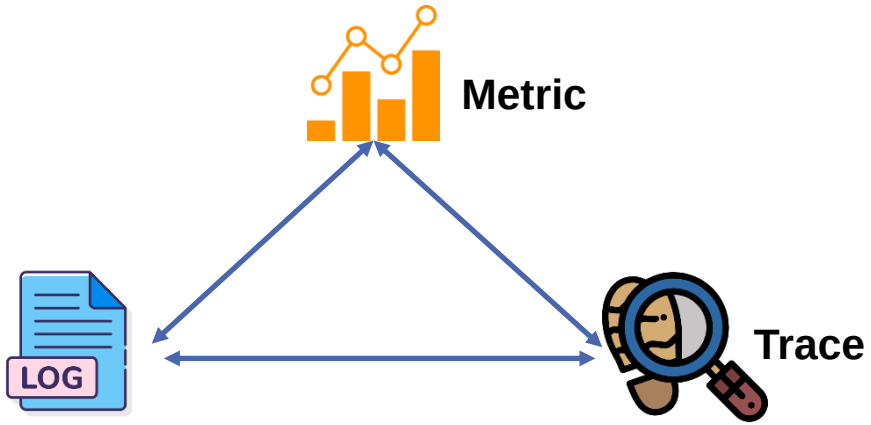
- Analyzing long-term trends
- Comparing over time or experiment groups
- Alerting
- Building dashboards
- Conducting ***ad hoc*** retrospective analysis

Observability



Three Pillars of System Monitor/Observation

Type	Functions	Tools
Metric	Create alerts and dashboards (長眼睛、監控大盤)	Prometheus Grafana
Trace	Identify problem area or failure point (定位問題)	OpenTelemtry Jaeger, Elasticsearch APM, Grafana Loki
Log	Find out the root cause	Elasticsearch Grafana Loki



AGENDA

SRE

Why Monitor

What to monitor

How to monitor

K8S Monitor Stacks

- Prometheus
- Grafana
- OpenTelemtry

Monitoring Methodologies

- There are three common lists or methodologies.
  - USE Method (for resource): **Utilization, Saturation, and Errors**
  - RED Method (for service): **Rate, Errors, and Duration**
  - Google's four golden signals (from the Google SRE book): **Latency, Traffic, Errors, and Saturation**

- **Latency** — Response time, including queue/wait time, in milliseconds.
- **Traffic (Rate)** — Request rate, in requests/sec
- **Errors** — Error rate, in errors/sec
  1. explicit failure: like HTTP 500 error
  2. implicit failure: like wrong or invalid content being returned
  3. policy-based failure: ex, over 10s should be considered error
- **Saturation** — How overloaded something is, which is related to utilization but more directly measured by things like queue depth (or sometimes concurrency). As a queue measurement, this becomes non-zero when you are saturated, often not much before.
- **Utilization** — How busy the resource or system is. Usually expressed 0–100% and most useful for predictions (as Saturation is probably more useful

Monitoring Metrics Example

Metric Category	Specfic metric	AP exposed metrics	Calculation in Prometheus
Latency	avg response time	xxx_duration_seconds (histogram)	rate(xxx_duration_seconds_sum[1m]) / rate(xxx_duration_seconds_count[1m])
	P50 response time		histogram_quantile(0.5, rate(xxx_duration_seconds_bucket[1m]))
	P95 response time		histogram_quantile(0.95, rate(xxx_duration_seconds_bucket[1m]))
	P99 response time		histogram_quantile(0.99, rate(xxx_duration_seconds_bucket[1m]))
Traffic	requests per second	xxx_requests_total (counter)	rate(xxx_requests_total[1m])
Error	error rate	xxx_errors_total (counter) xxx_requests_total (counter)	1. rate(xxx_errors_total[1m]) / rate(xxx_requests_total[1m]) 2. rate(xxx_requests_total{response=~"5.*"}[1m]) / rate(xxx_requests_total[1m])
Saturation	CPU Memory Busy Ratio	…	…

AGENDA

SRE

Why Monitor

What to monitor

How to monitor

K8S Monitor Stacks

- Prometheus
- Grafana
- OpenTelemtry

Probing and Introspection

Probing: Black-box monitoring

- 在 AP 外部探測 AP 的外部特徵

Introspection: White-box monitoring

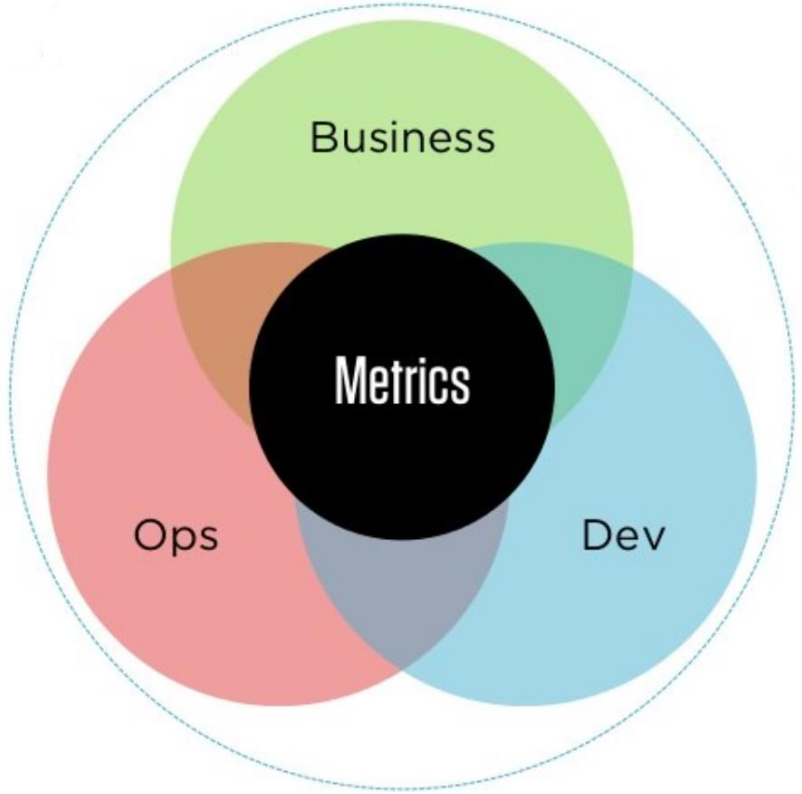
- AP 本身具備提供 AP 內部運作狀態的能力
- Instrument AP and expose metrics for Prometheus scraping
- Metric-Driven Development (MDD)

Metric-Driven Development (MDD)

Define metrics before implementation

Instrumentation-as-Code

Shared view for key metrics



AGENDA

SRE

Why Monitor ?

Observability

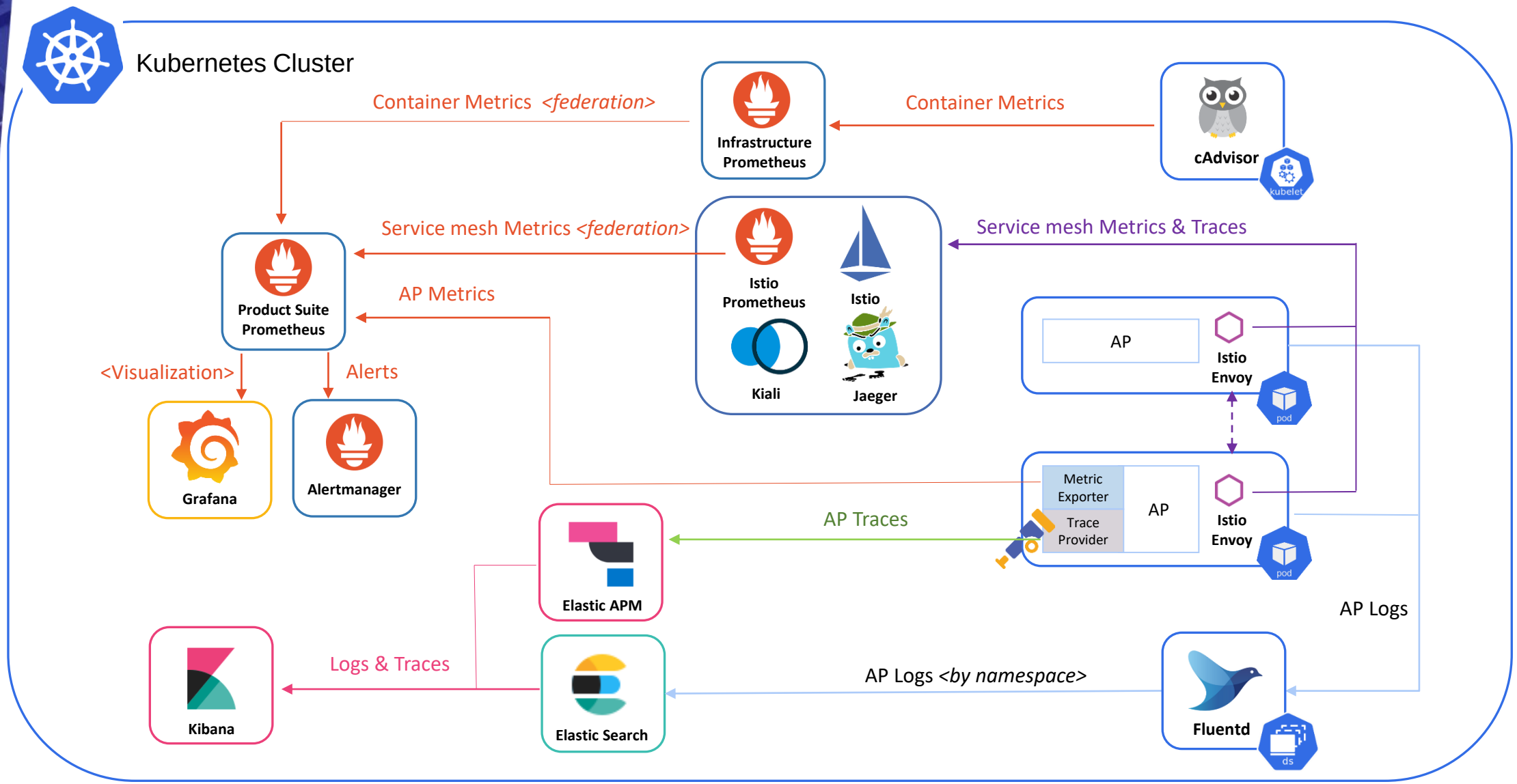
What to monitor

How to monitor

K8S Monitor Stacks

- Prometheus
- Grafana
- OpenTelemetry

Next-gen Kubernetes Monitoring Stack



Prometheus Introduction

Prometheus is an open source systems monitoring and alerting toolkit

- Started at SoundCloud around 2012-2013, and was made public in 2015
- Inspired by Google's Borgmon. Uses time-series data as a data source and support to send alerts based on this data
- Fits very well in the cloud native infrastructure
- Stores time series metrics in memory and on local disk in an efficient custom format.
- Is second graduated project in CNCF (Cloud Native Computing Foundation), after Kubernetes.

Prometheus Pros & Cons

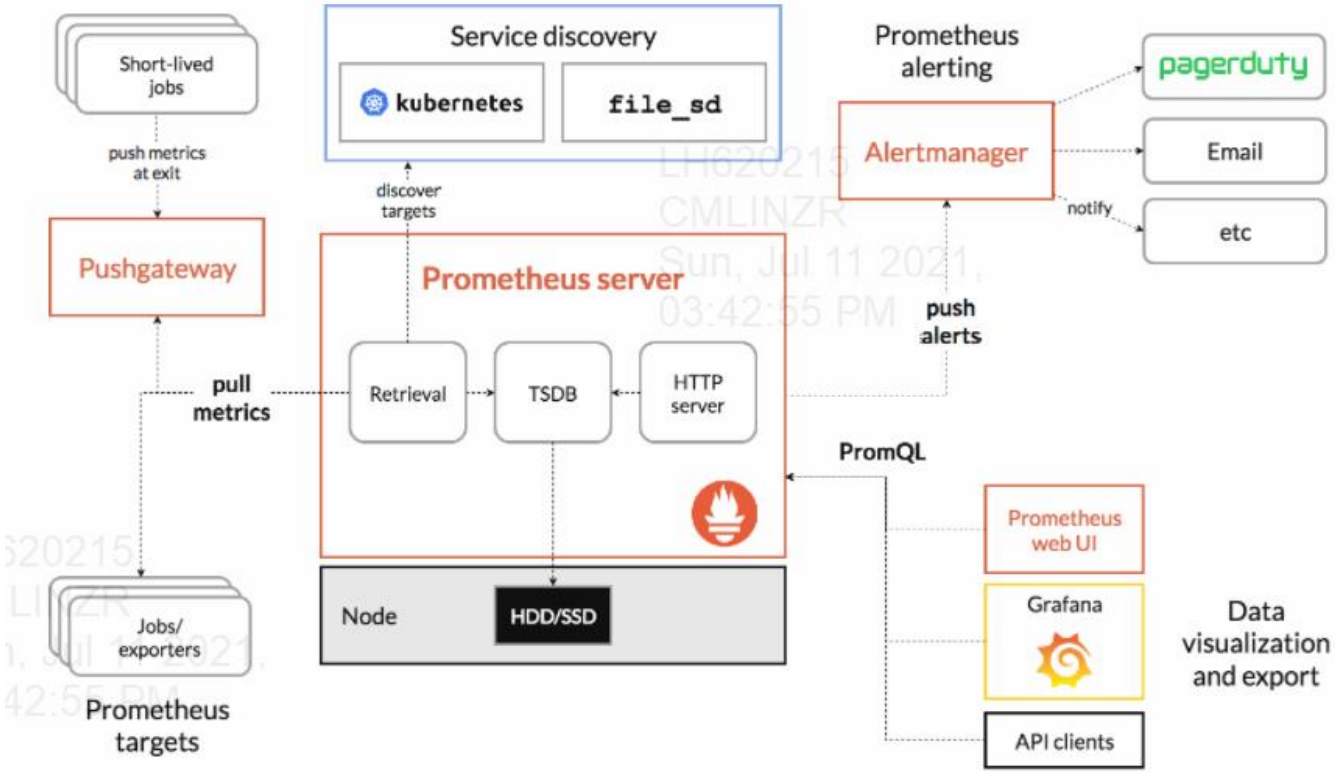
- **Prometheus is TSDB**
- **With Pull/Push mode to collect metrics.**
- **In built Alerting facility**
- **Good client libraries (exporters)**
- **Easy to use monitoring.**
- **Flexible query language**
- **Dimensional data model**
- **Easy to extend.**
- **Service discovery. (Add servers into monitor list)**

- Prometheus doesn't provide a dashboard solution.
- Prometheus is designed to collect and process metrics, not an event logging system.
- There is no option for long-term local storage.

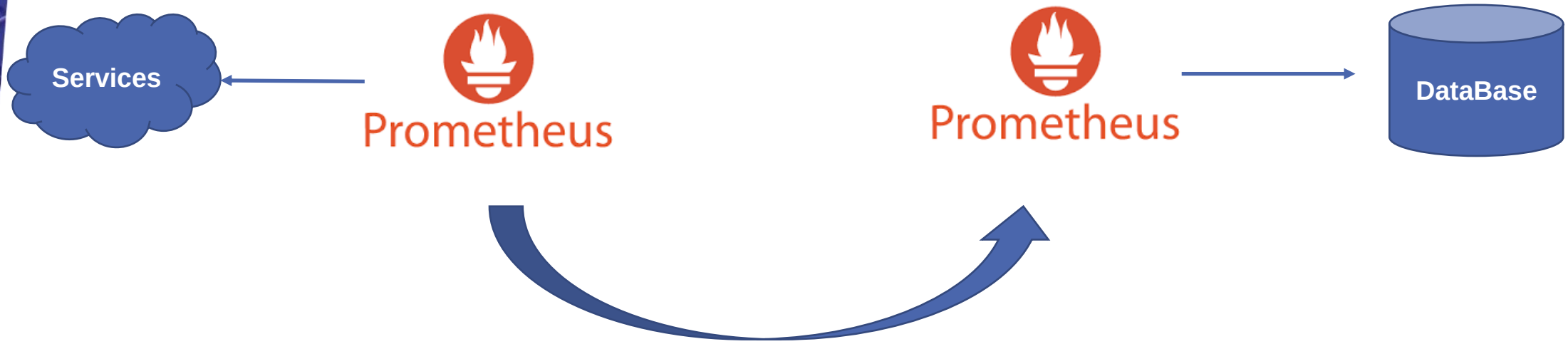
Prometheus Architecture

Prometheus is composed of multiple components

- **Prometheus** pulls metrics from exporter and store in TSDB (time series database) and evaluates alert rules over this data. Alerts are sent to **Alertmanager**.
- **Alertmanager** receives alerts from Prometheus and turns them into notification



Prometheus Federation




```
scrape_configs:
  - job_name: 'federate'
    metrics_path: '/federate'
    params:
      'match[]':
      - '{namespace=~"(APP1|app2)"}'
    static_configs:
      - targets: ['prometheus.app.paasdevc2.com']
```

Prometheus Configuration

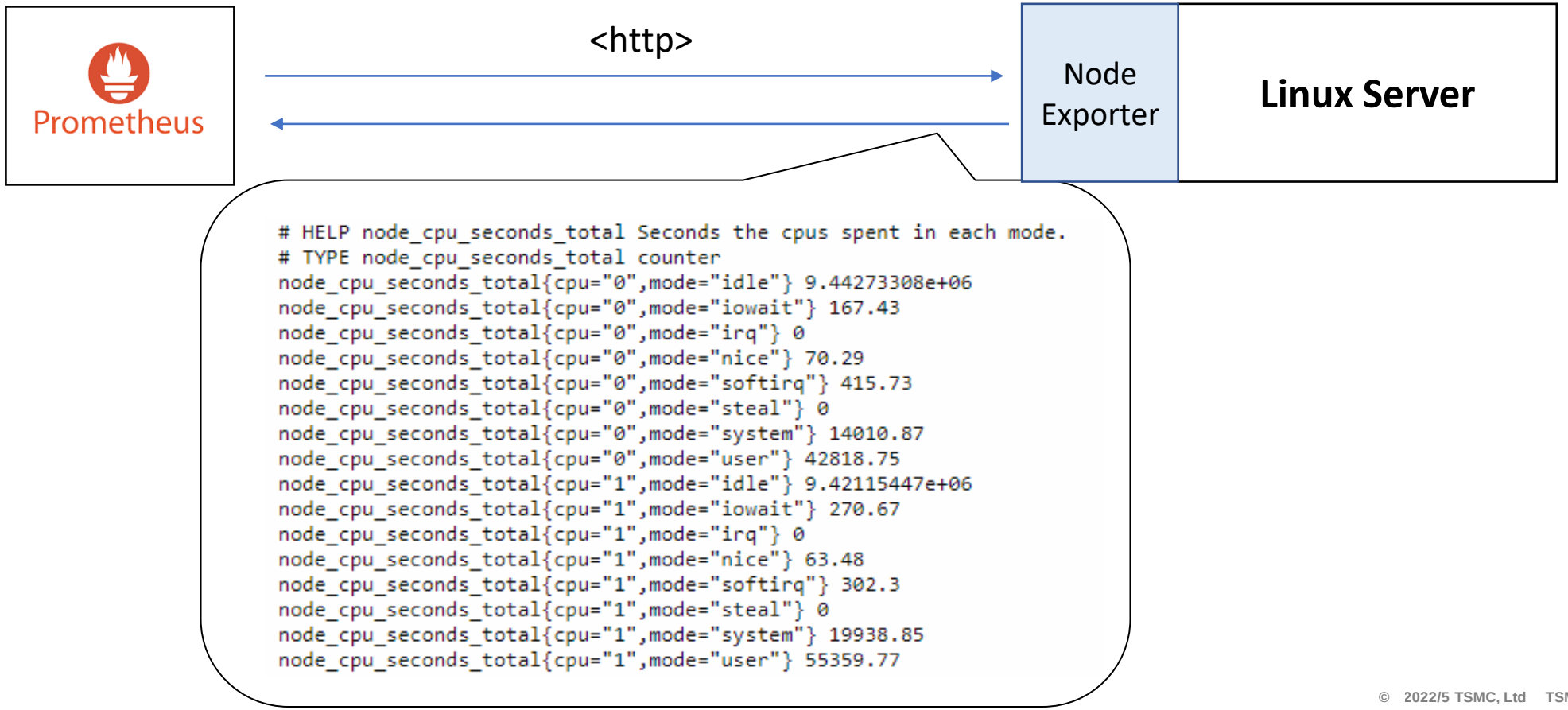
```yaml
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      - alertmanager-0.alertmanager-svc:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  - "alert_rule.yml"

# A scrape configuration containing exactly one endpoint to scrape:
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this c
  - job_name: 'prometheus'
    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.
    static_configs:
    - targets: ['localhost:9090']

  # Get app basic information (cpu/memory) from infra prometheus.
  - job_name: 'federate'
    metrics_path: '/federate'
    params:
      'match[]':
      - '{namespace=~"(tmeet|tccop-fab12-sit)"}'
    static_configs:
      - targets: ['kube-prometheus-prometheus.monitoring.svc.cluster.local:9090']
```

Configuration file: prometheus.yml

scrape_configs defines scrape jobs with method and targets

How Prometheus Works

Prometheus collects metrics by periodically scrapping exporter HTTP endpoints.




```
# HELP node_cpu_seconds_total Seconds the cpus spent in each mode.
# TYPE node_cpu_seconds_total counter
node_cpu_seconds_total{cpu="0",mode="idle"} 9.44273308e+06
node_cpu_seconds_total{cpu="0",mode="iowait"} 167.43
node_cpu_seconds_total{cpu="0",mode="irq"} 0
node_cpu_seconds_total{cpu="0",mode="nice"} 70.29
node_cpu_seconds_total{cpu="0",mode="softirq"} 415.73
node_cpu_seconds_total{cpu="0",mode="steal"} 0
node_cpu_seconds_total{cpu="0",mode="system"} 14010.87
node_cpu_seconds_total{cpu="0",mode="user"} 42818.75
node_cpu_seconds_total{cpu="1",mode="idle"} 9.42115447e+06
node_cpu_seconds_total{cpu="1",mode="iowait"} 270.67
node_cpu_seconds_total{cpu="1",mode="irq"} 0
node_cpu_seconds_total{cpu="1",mode="nice"} 63.48
node_cpu_seconds_total{cpu="1",mode="softirq"} 302.3
node_cpu_seconds_total{cpu="1",mode="steal"} 0
node_cpu_seconds_total{cpu="1",mode="system"} 19938.85
node_cpu_seconds_total{cpu="1",mode="user"} 55359.77
```

Prometheus Exporters for Metric Collection

Official or Community Contributed Prometheus Exporter

- **Databases**
  - MySQL, Redis, Mongo DB, PostgreSQL exporter
- **HTTP**
  - HAProxy, Nginx, Web Driver Exporter
- **Others**
  - JMX, SpringBoot, influxDB, jisti-meet, etc.

https://prometheus.io/docs/instrumenting/exporters/

CONFIG PROMETHEUS ALERTING

Query Prometheus

- **http_requests_total{job="prometheus",group="canary"}**
- **http_requests_total{environment=~"staging|testing|development}**
- **http_requests_total{job="prometheus"}[5m]**
- **rate(http_requests_total[5m]**
- **sum by (app, proc) (**

  instance_memory_limit_bytes - instance_memory_usage_bytes

) / 1024 / 1024

https://prometheus.io/docs/prometheus/latest/querying/examples/

Configure Alert Rules in Prometheus

- **Define rule files in prometheus.yml**

```yaml
# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  - "alert_rules.yml"
  - "second_alert_rules.yml"
```

- **Define alert rule in rule file:**

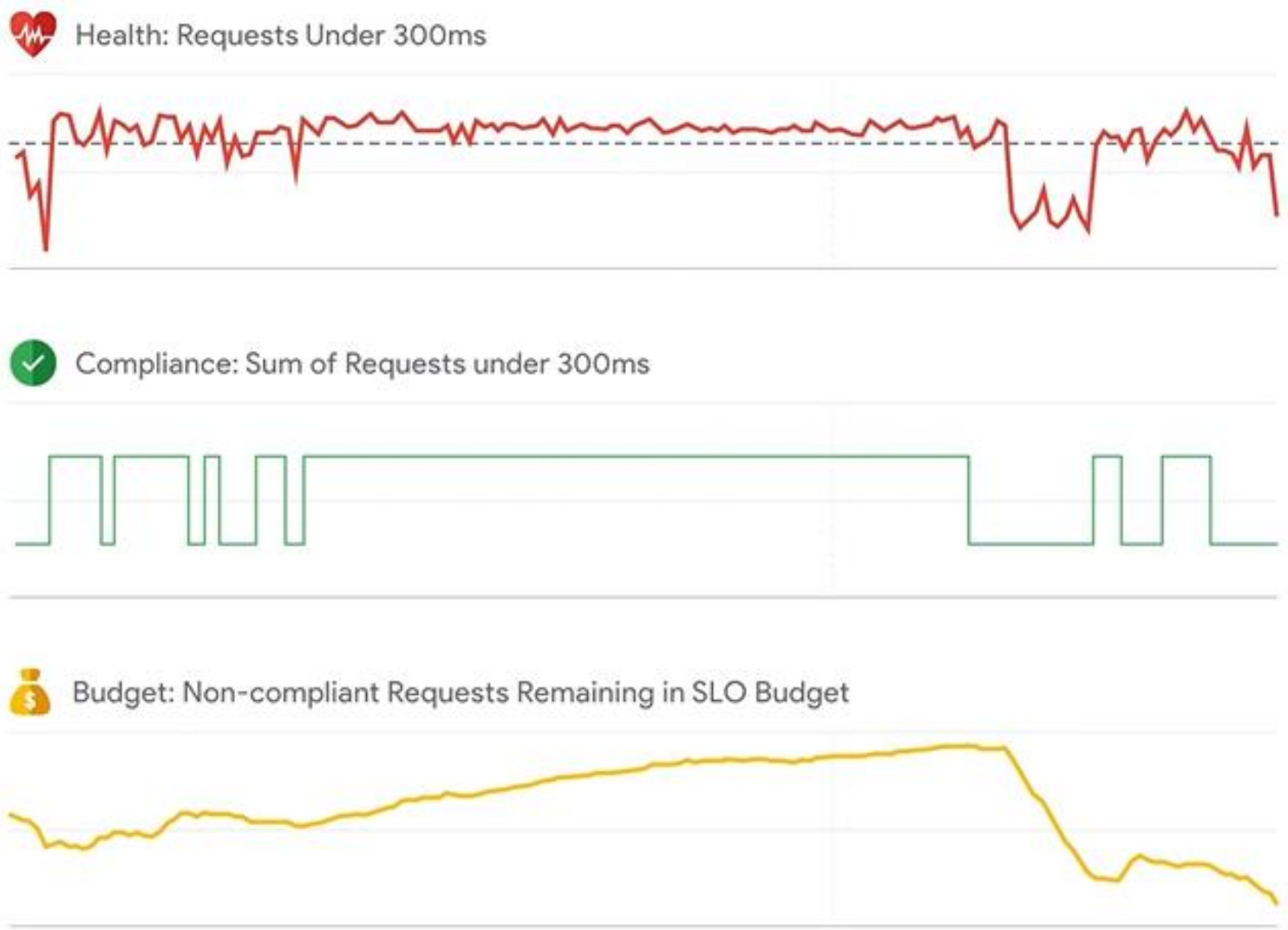
```yaml
groups:
- name:  example
  rules:
  # Alert for any instance that is unreachable for >5 minutes.
  - alert: PrometheusTargetMissing
    expr: up == 0
    for: 3m
    labels:
      severity: critical
    annotations:
      summary: "Prometheus target missing (instance {{ $labels.instance }}"
      description: "A Prometheus target disappered.\n VALUE = {{ $value }}\n LABELS = {{ $labels }}"
  - alert: ServiceError
    expr: >
      (
        sum(rate(istio_requests_total{reporter="source", response_code!~"5.*"}[1m])) by (destination_workload, destination_workload_namespace)
        /
        sum(rate(istio_requests_total{reporter="source"}[1m])) by (destination_workload, destination_workload_namespace)
      ) < 0.95
    for: 5m
    labels:
      severity: ticket
    annotations:
      summary: "Service Successful Rate is less than 95% on {{ $labels.destination_workload }} in namespace {{ $labels.destination_workload_namespace }}"
```

Alerting Considerations

Alert的目標是在發生重大事件時得到通知，因此定義rule時，要考量以下四個屬性

- Precision**精確度**
  - 檢測到的事件比例很重要。 如果每個Alert都有對應重大事件，則精度爲100%。
- Recall**召回率**
  - 檢測到重大事件的比例。如果每個重要事件都會發出報警，則召回率爲100%。
- Detection time**檢測時間**
  - 較長的檢測時間會對錯誤預算(Error budget)產生負面影響。
- Reset time**恢復時間**
  - 解決問題後報警會持續多長時間。較長的恢復時間可能導致混淆或問題被忽略。

Error Budget



SLO 99.9%

Error Budget
43.2 Min/Month

https://asa55.github.io/class-sre-implements-devops/

Multi-window, Multi-Burn-Rate Alerts

增強多消耗率報警，以便僅在我們仍在快速消耗預算時通知我們 - 從而減少誤報的數量。 爲此，我們需要添加另一個參數：一個較短的窗口，用於檢查在觸發報警時是否仍在消耗錯誤預算。一個好的方案是將短窗口設爲長窗口持續時間的1/12。例如，可以在前一小時和前五分鐘超過14.4倍消耗率時發送警報。 只有在消耗了2%的預算後，此警報才會觸發， 但顯示出重置時間來停止報警時間設置五分鐘而不是一個小時

- expr:

(job:slo_errors_per_request:ratio_rate1h{job="myjob"} > (14.4*0.001) and job:slo_errors_per_request:ratio_rate5m{job="myjob"} > (14.4*0.001))

or

(job:slo_errors_per_request:ratio_rate6h{job="myjob"} > (6*0.001) and job:slo_errors_per_request:ratio_rate30m{job="myjob"} > (6*0.001))

Severity: page

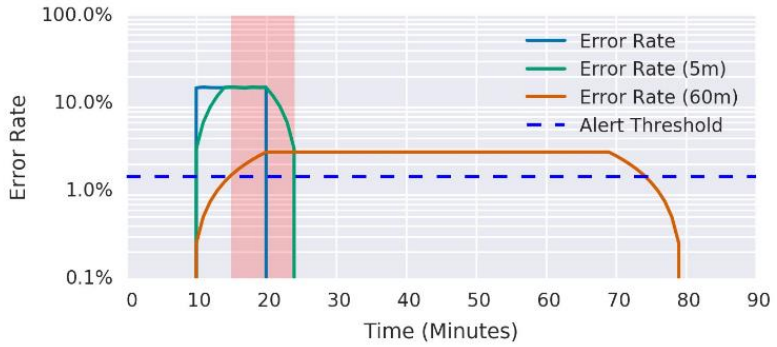
- expr:

(job:slo_errors_per_request:ratio_rate24h{job="myjob"} > (3*0.001) and job:slo_errors_per_request:ratio_rate2h{job="myjob"} > (3*0.001))

or

(job:slo_errors_per_request:ratio_rate3d{job="myjob"} > 0.001 and job:slo_errors_per_request:ratio_rate6h{job="myjob"} > 0.001)

severity: ticket



*burn rate = budget consumed * period / alerting window*

一小時內將2%的預算消耗 *?=2%*30*24/1 ?為14.4*

6小時內的5%的預算消耗 *?=5%*30*24/6 ?為6*

三天內將10%預算消耗 *?=10%*30*24/72 ?為1*

https://sre.google/workbook/alerting-on-slos/

Alertmanager Configuration

```yaml
global:
  smtp_smarthost: HCloudmrelay01:25
  smtp_from: k8s-dapd@tsmc.com
  smtp_require_tls: false
route:
  receiver: default-receiver
  group_by: [cluster, alertname]
  # All alerts that do not match the following child routes
  # will remain at the root node and be dispatched to 'default-receiver'.
  routes:
  # All alerts with service=mysql or service=cassandra
  # are dispatched to the database pager.
  - receiver: 'database-pager'
    group_wait: 10s
    matchers:
    - service=~"mysql|cassandra"
  # All alerts with the team=frontend label match this sub-route.
  # They are grouped by product and environment
  - receiver: 'frontend-pager'
    group_by: [product, environment]
    matchers:
    - team="frontend"

receivers:
- name: default-receiver
  email_configs:
  - to: 'xxx@tsmc.com'
    send_resolved: false
- name: database-pager
  webhook_configs:
  - url: 'http://xxx:8000'
- name: frontend-pager
  slack_configs:
  - api_url: http://tchat-blue-sit.gw.paasprd.tsmc.com.tw/hooks/z8MLktYTtp
    channel: '#grafana-test'
```

- **alertmanager.yaml**

Routing tree setting.
Send to specified receiver when alert label values match.
Check child route before parent
(*post-order tree transversal*)

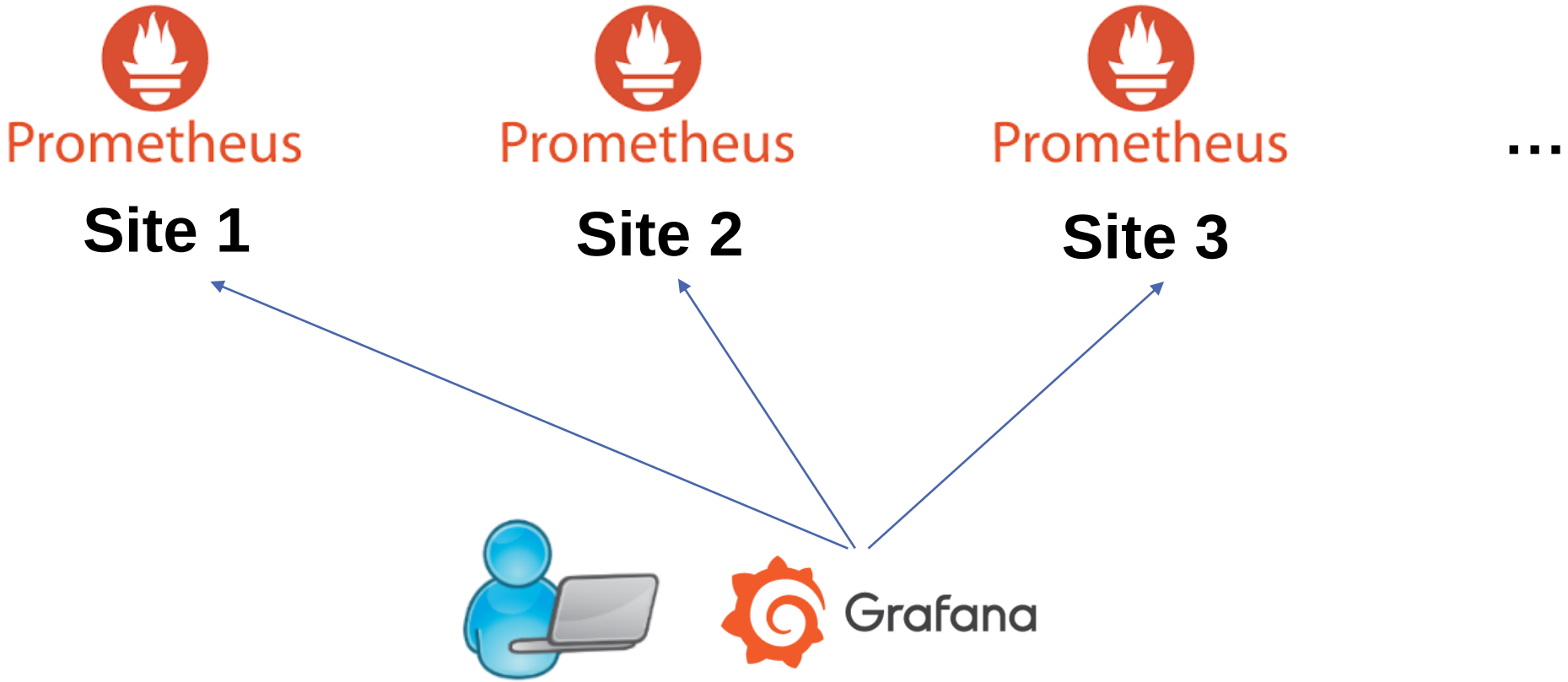
Alertmanager Features

	功能	情境	在哪裡設定
Alert Grouping	將多筆不同 alert 合併成一個	當 network 掛點時，在此網段的 service 所有 alert 都會跟著觸發 (連不到 DB, health ping fail, loading delay.. 等)，此時就可以透過 grouping 功能，整理成一個 alert.	Alert Manager 設定檔 (routing tree 的部分)
Alert Inhibition	某個 alert 已經發了，就不要再發其他 alert	假設某個 cluster 暫時掛點，發了 cluster fail 的 alert 後，就不需要再發其他連不到這個 cluster 的alert	Alert Manager 設定檔
Alert Silence	指定一段時間不要發 alert	Scheduled downtime	Alertmanager web interface

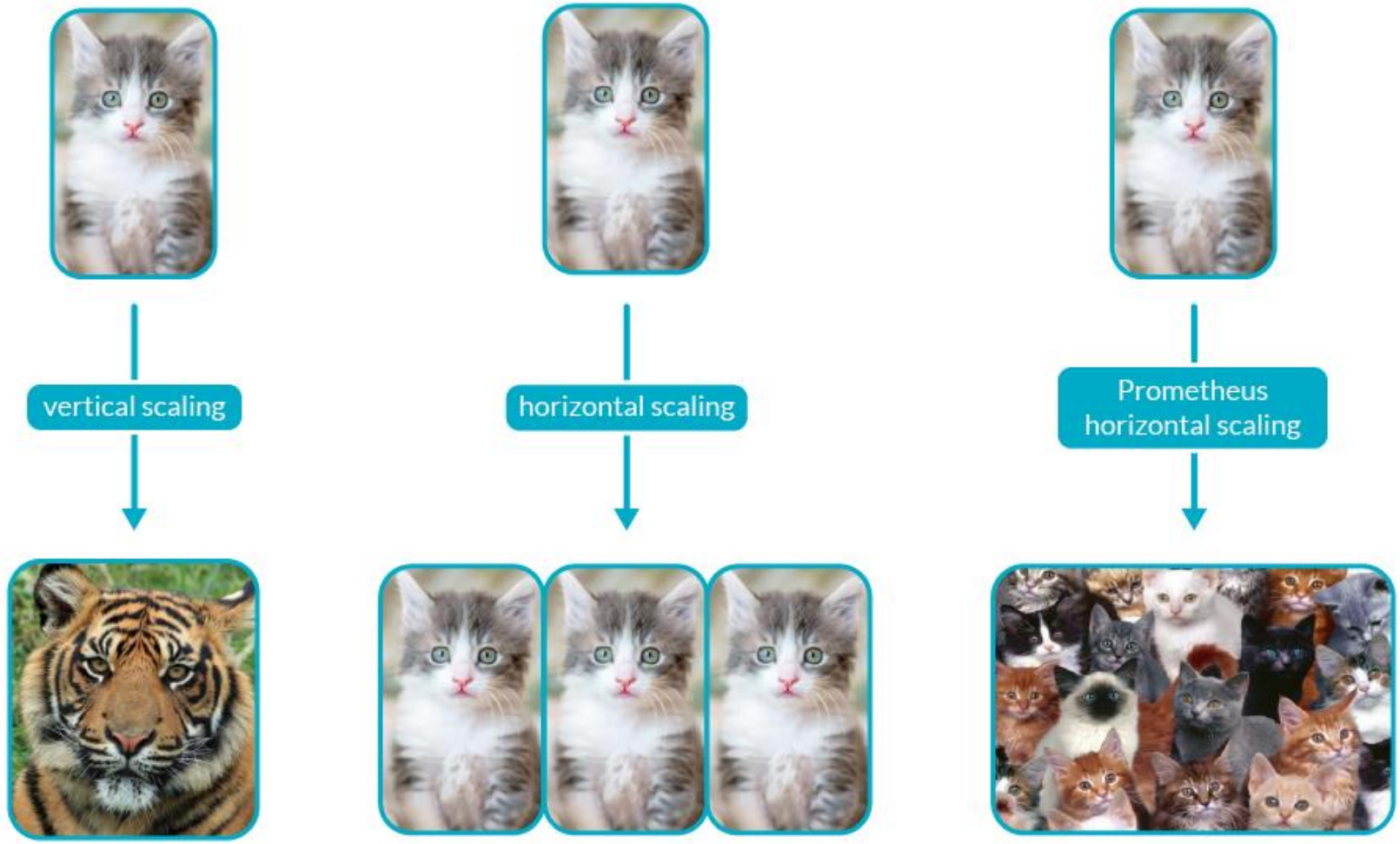
PRACTICE-START UP PRACTICE

PROMETHEUS SCALE OUT

Prometheus Scale Out



Prometheus Horizontal Scaling



https://sysdig.com/blog/challenges-scale-prometheus/

Prometheus Scaling by Thanos or Cortex

- https://thanos.io
- 分散式(其數據可以分散在多個數據中心，且分別存於多個Object Storage中。對於資料的寫入，查詢和告警可分散也可集中。
- Started on Dec 2017
- Joined CNCF sandbox in Aug 2019

- https://cortexmetrics.io
- 集中式(多個Prometheus，所有的data都匯入一個Cortex數據中心，由Cortex集中寫入，查詢和告警。)
- Started in June 2016
- Joined CNCF sandbox Sept 2018

AGENDA

SRE

Why Monitor ?

Observability

What to monitor

How to monitor

K8S Monitor Stacks

- Prometheus
- Grafana
- **OpenTelemetry**

OpenTelemetry History

Google OpenCensus

+

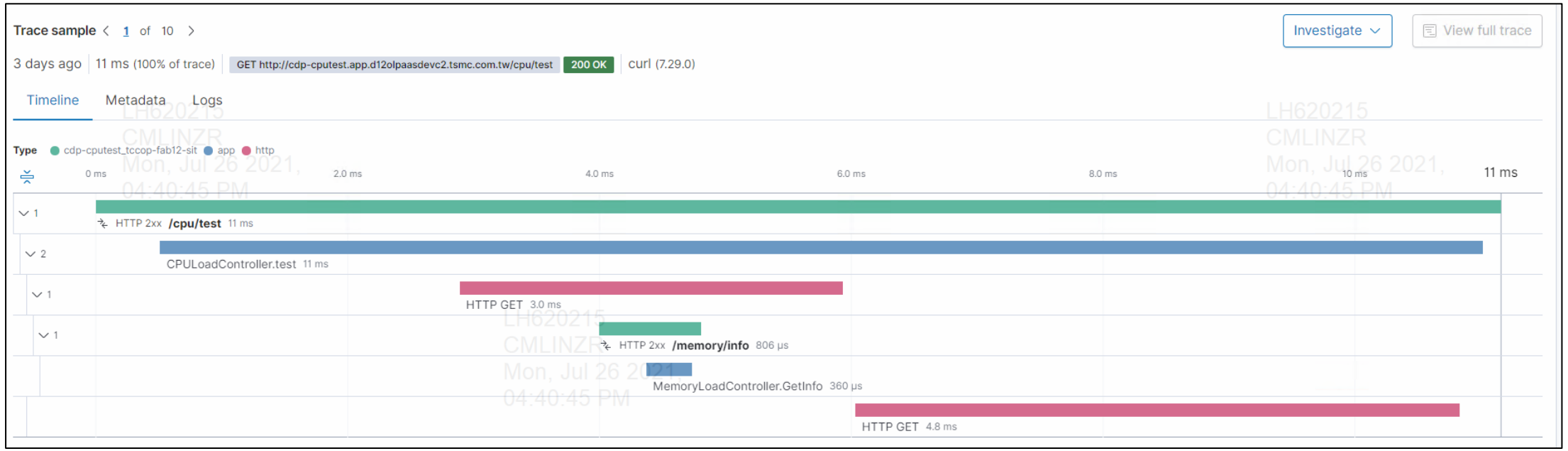
CNCF OpenTracing

=

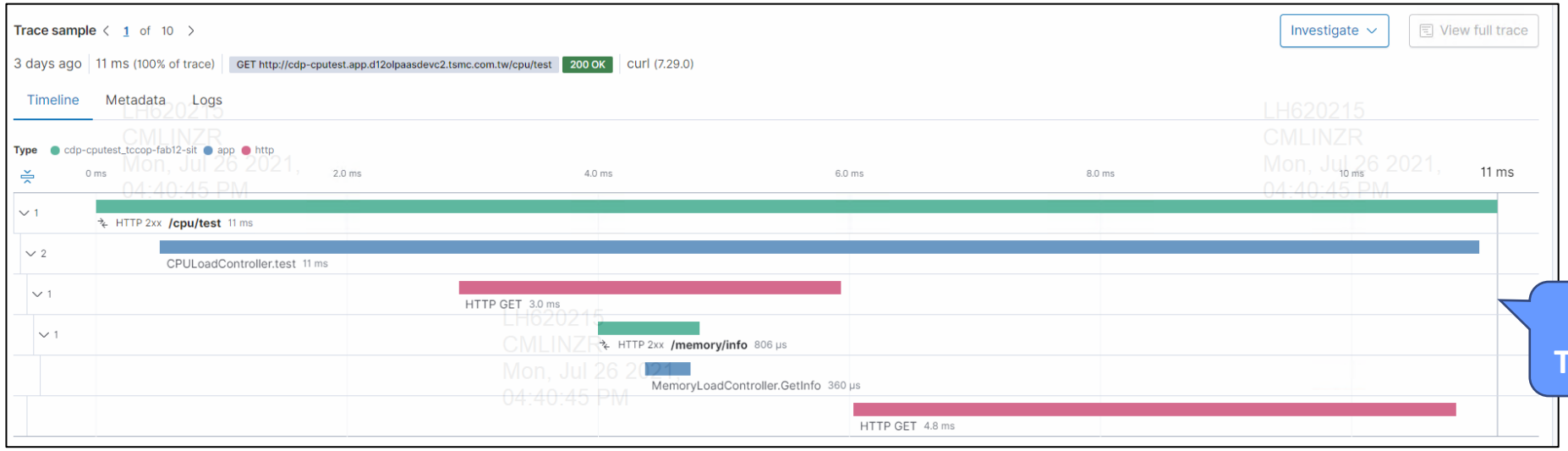
What is trace

Trace represents a single user's journey through an entire app stack

一個 transaction 中間經過的所有路徑。包含 service 內的函式呼叫以及 service 之間的呼叫



Correlate Traces with Logs



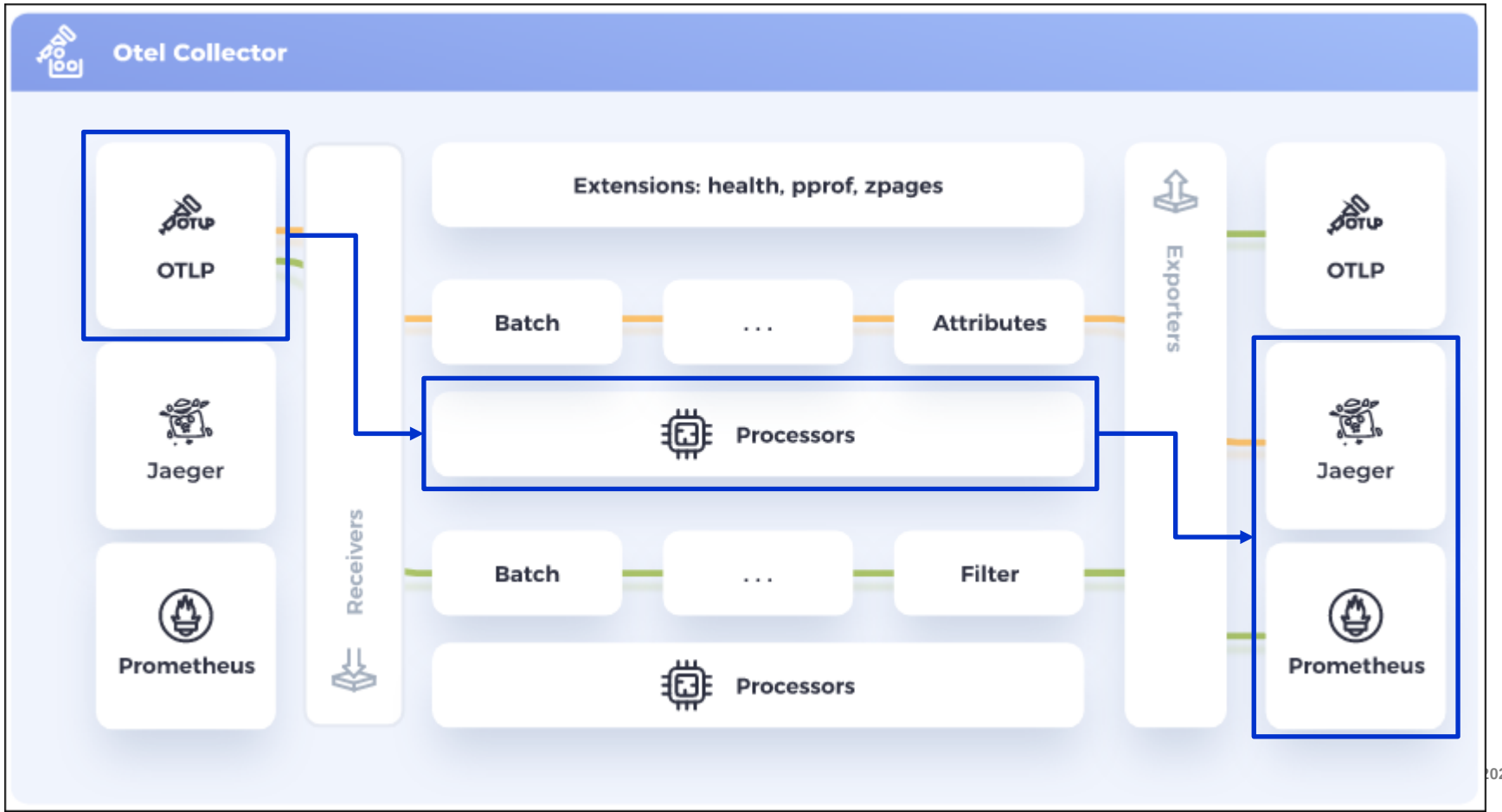
Trace spans generated by Tracing Library - OpenTelemetry

Inject trace and span ID in logs for connecting traces to logs

Otel Collector (1/3)

Otel Collector (Receiver-> Processor-> Exporter)

- OTLP-> Otel Collector-> Elastic APM / Jaeger / Prometheus



Otel Collector (2/3)

Processor-> Span Metrics Processor

- Aggregates Request, Error and Duration (R.E.D) metrics from span data

main ▾ opentelemetry-collector-contrib / processor /

codeboten Bump github.com/aws/aws-sdk-go from 1.40.38 to 1.40.42 (#5230)

..	
attributesprocessor	Upgrade to core latest PR, no other changes (#5195)
cumulativetodeltaprocessor	Upgrade to core latest PR, no other changes (#5195)
deltatorateprocessor	Upgrade to core latest PR, no other changes (#5195)
filterprocessor	Add exclude and include config for filter log processor (#4895)
groupbyattrsprocessor	Upgrade to core latest PR, no other changes (#5195)
groupbytraceprocessor	Upgrade to core latest PR, no other changes (#5195)
k8sprocessor	Upgrade to core latest PR, no other changes (#5195)
metricsgenerationprocessor	Upgrade to core latest PR, no other changes (#5195)
metricstransformprocessor	Upgrade to core latest PR, no other changes (#5195)
probabilisticsamplerprocessor	Upgrade to core latest PR, no other changes (#5195)
resourcedetectionprocessor	Bump github.com/aws/aws-sdk-go from 1.40.38 to 1.40.42 (#5230)
resourceprocessor	Upgrade to core latest PR, no other changes (#5195)
routingprocessor	Upgrade to core latest PR, no other changes (#5195)
spanmetricsprocessor	Upgrade to core latest PR, no other changes (#5195)
spanprocessor	Upgrade to core latest PR, no other changes (#5195)
tailsamplingprocessor	Upgrade to core latest PR, no other changes (#5195)

HTTP POST — Service: **tchat** — Duration: **103.7ms** — Start Time: **1.39s**

Duration

Tags

http.host	localhost:3000
http.method	POST
http.response_content_length	376
http.response_content_length_uncompressed	1057
http.scheme	http
http.status_code	200
http.status_text	OK
http.url	http://localhost:3000/api/v1/method.call/rocketchatSearch.search
http.user_agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/93.0.4577.63 Safari/537.36
internal.span.format	proto
otel.library.name	@opentelemetry/instrumentation-xml-http-request
otel.library.version	0.24.0
otel.status_code	0

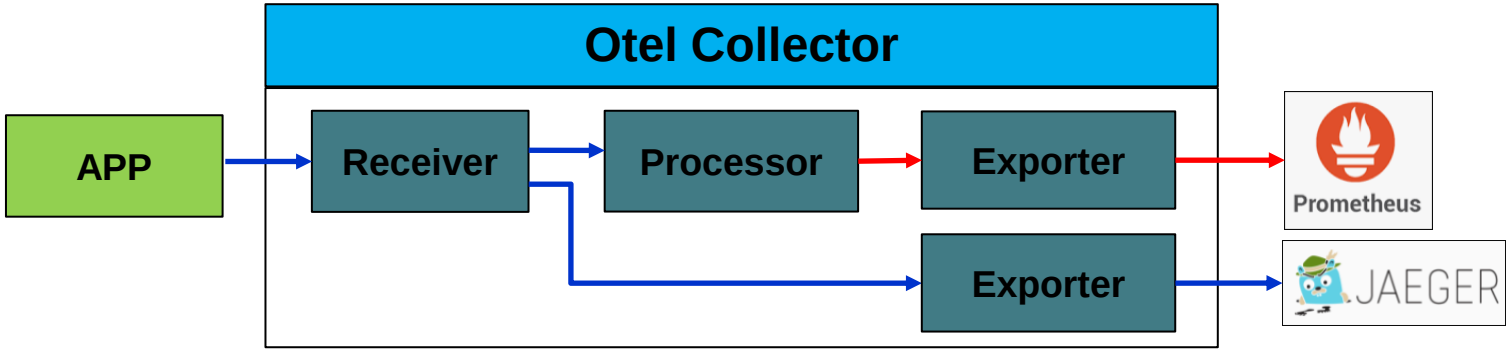
Error

Otel Collector (3/3)

→ Trace info (span data)

→ Metric info

```yaml
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:55680
      http:
        endpoint: 0.0.0.0:55681
        cors_allowed_origins:
        - http://*
        - https://*

  # Dummy receiver that's never used, because
  otlp/spanmetrics:
    protocols:
      grpc:
        endpoint: "localhost:12345"

exporters:
  prometheus:
    endpoint: 0.0.0.0:8889
    namespace: tchat

  jaeger:
    endpoint: jaeger:14250
    insecure: true

  otlp/spanmetrics:
    endpoint: "localhost:55680"
    insecure: true

processors:
  batch:
  spanmetrics:
    metrics_exporter: otlp/spanmetrics
```




```yaml
service:
  pipelines:
    traces:
      receivers: [otlp]
      # spanmetrics will pass on span data untouched to next processor
      # while also accumulating metrics to be sent to the configured 'otlp/spanmetrics' exporter.
      processors: [spanmetrics, batch]
      exporters: [jaeger]

    # This pipeline acts as a proxy to the 'metrics' pipeline below,
    # allowing for further metrics processing if required.
    metrics/spanmetrics:
      # This receiver is just a dummy and never used.
      # Added to pass validation requiring at least one receiver in a pipeline.
      receivers: [otlp/spanmetrics]
      exporters: [otlp/spanmetrics]

    metrics:
      receivers: [otlp]
      # The metrics_exporter must be present in this list.
      exporters: [prometheus]
```

METRICS INSTRUMENTATION

Instrumentation

What is instrumentation?

- 為系統帶上「健康手環」

Why instrument?

- **“The largest payoffs you will get from Prometheus are through instrumenting your own applications”**

Prometheus Metric Types

1. Counter
2. Gauge
3. Histogram
4. Summary

Prometheus Metric Type - Counter

只增不減的計數值 (系統重啟才會重置)。例如：

- http_requests_total
- node_cpu (CPU累積使用時間)

- 一般使用 _total 作為後綴

- 使用範例

1. 以過去五分鐘為樣本，計算 http request 數量的增加率 (平均每秒增加多少)：
   rate(http_requests_total[5m])

   * rate(): 從給定的樣本區間計算「平均每秒的增加量」
2. 列出前 10 大 http request:
   topk(10, http_requests_total)

How to Instrument a Counter Metric

```python
import http.server
import random
from prometheus_client import start_http_server
from prometheus_client import Counter


REQUESTS = Counter('demo_requests_total', 'Demo counter metric to record request count')

ERRORS = Counter('demo_errors_total', 'Demo counter metric to record error count')

class DemoHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        REQUESTS.inc();

        if random.random() < 0.2:
            ERRORS.inc();
            self.send_response(500)
            self.end_headers()
            self.wfile.write(b"Error\n")
        else:
            self.send_response(200)
            self.end_headers()
            self.wfile.write(b"Hello World\n")

if __name__ == "__main__":
    # expose metrics in http endpoint
    start_http_server(8200)

    server = http.server.HTTPServer(('tf15itsre01', 8000), DemoHandler)
    server.serve_forever()
```

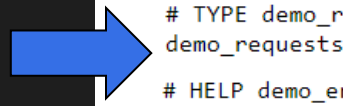
算增長率 (traffic)

rate(demo_requests_total[1m])

算錯誤率 (error)

rate(demo_errors_total[1m]) /
rate(demo_requests_total[1m])

metrics to expose

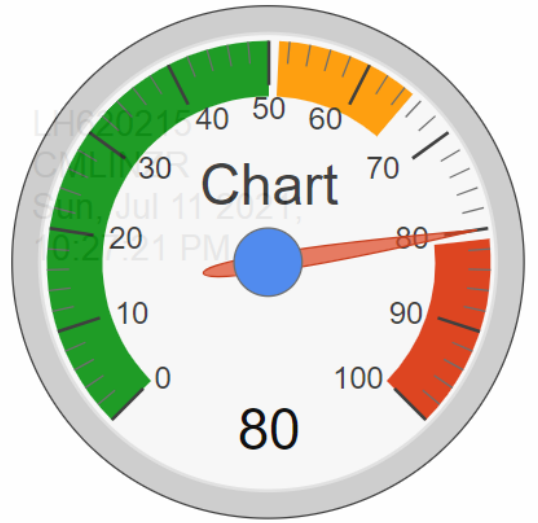


```
# HELP demo_requests_total Demo counter metric to record request count
# TYPE demo_requests_total counter
demo_requests_total 867.0

# HELP demo_errors_total Demo counter metric to record error count
# TYPE demo_errors_total counter
demo_errors_total 174.0
```

TSMC IT × NCTU CS
2022 CLOUD NATIVE
Development Best Practice

Prometheus Metric Type - Gauge

- 反應系統目前狀態。例如：
  - node_memory_MemFree
  - node_memory_MemAvailable



使用範例

1. 直接看系統狀態：
   node_memory_MemFree
2. 計算 CPU 在兩小時內的差異：
   delta(cpu_temp_celsius{host="zeus"}[2h])
3. 使用線性迴歸模型預測 disk 在四小時之後的狀況
   predict_linear(node_filesystem_free{job="node"}[1h], 4 * 3600)

Prometheus Metric Type - Histogram

統計像是「request 執行時間」(latency)、「request/response size 」這類問題的分佈資訊

用於計算平均數以及分位數 (中位數、第90百分位數、第95百分位數)

metric base name: prometheus_http_request_duration_seconds

```
prometheus_http_request_duration_seconds_bucket{handler="/api/v1/query_range",le="0.1"} 499
prometheus_http_request_duration_seconds_bucket{handler="/api/v1/query_range",le="0.2"} 499
prometheus_http_request_duration_seconds_bucket{handler="/api/v1/query_range",le="0.4"} 499
prometheus_http_request_duration_seconds_bucket{handler="/api/v1/query_range",le="1"} 499
prometheus_http_request_duration_seconds_bucket{handler="/api/v1/query_range",le="3"} 502
prometheus_http_request_duration_seconds_bucket{handler="/api/v1/query_range",le="8"} 502
prometheus_http_request_duration_seconds_bucket{handler="/api/v1/query_range",le="20"} 502
prometheus_http_request_duration_seconds_bucket{handler="/api/v1/query_range",le="60"} 502
prometheus_http_request_duration_seconds_bucket{handler="/api/v1/query_range",le="120"} 502
prometheus_http_request_duration_seconds_bucket{handler="/api/v1/query_range",le="+Inf"} 502
prometheus_http_request_duration_seconds_sum{handler="/api/v1/query_range"} 6.075687352000006
prometheus_http_request_duration_seconds_count{handler="/api/v1/query_range"} 502
```

<base name>_bucket 本身是 counter metric.
label le 表示 Less than or Equal to (小於等於).
以第一筆來說，意思是截至目前為止執行時間小於等於 0.1 秒的 request 數量累積有 499 筆.

<base name>_sum: request 執行時間的總和

<base name>_count: request 總數

- 使用範例
  - 計算平均 (以過去 1 分鐘為樣本) ：
    rate(http_request_duration_seconds_sum[1m]) / rate(http_request_duration_seconds_count[1m])
  - 計算第 95 百分位數 (以過去 5 分鐘為樣本)：
    histogram_quantile(0.95,
    rate(prometheus_http_request_duration_seconds_bucket{handler="/api/v1/query_range"}[5m]))

How to Instrument a Histogram Metric

```python
import http.server
import time
import random
from prometheus_client import start_http_server
from prometheus_client import Histogram


LATENCY = Histogram('demo_latency_seconds', 'Demo histogram metric to record latency')

class DemoHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        start = time.time()

        # sleep 0 ~ 1 second
        sleep_time = random.random()
        time.sleep(sleep_time)

        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello World.\n")

        end = time.time()
        LATENCY.observe(end - start)

if __name__ == "__main__":
    # expose metrics in http endpoint
    start_http_server(8200)

    server = http.server.HTTPServer(('tf15itsre01', 8000), DemoHandler)
    server.serve_forever()
```

算平均

rate(demo_latency_seconds_sum[1m]) /
rate(demo_latency_seconds_count[1m])

算分位數

histogram_quantile(0.50,
rate(demo_latency_seconds_bucket[5m]))

metrics to expose

```
# HELP demo_latency_seconds Demo histogram metric to record latency
# TYPE demo_latency_seconds histogram
demo_latency_seconds_bucket{le="0.005"} 1.0
demo_latency_seconds_bucket{le="0.01"} 1.0
demo_latency_seconds_bucket{le="0.025"} 2.0
demo_latency_seconds_bucket{le="0.05"} 8.0
demo_latency_seconds_bucket{le="0.075"} 11.0
demo_latency_seconds_bucket{le="0.1"} 14.0
demo_latency_seconds_bucket{le="0.25"} 32.0
demo_latency_seconds_bucket{le="0.5"} 59.0
demo_latency_seconds_bucket{le="0.75"} 97.0
demo_latency_seconds_bucket{le="1.0"} 123.0
demo_latency_seconds_bucket{le="2.5"} 123.0
demo_latency_seconds_bucket{le="5.0"} 123.0
demo_latency_seconds_bucket{le="7.5"} 123.0
demo_latency_seconds_bucket{le="10.0"} 123.0
demo_latency_seconds_bucket{le="+Inf"} 123.0
demo_latency_seconds_count 123.0
demo_latency_seconds_sum 60.536704778671265
```

Prometheus Metric Type - Summary

跟 Histogram 一樣，用於記錄資料的分佈狀況

直接把分位數 (quantile) 算好

```
# HELP prometheus_rule_group_duration_seconds The duration of rule group evaluations.
# TYPE prometheus_rule_group_duration_seconds summary
prometheus_rule_group_duration_seconds{quantile="0.01"} 0.000441606
prometheus_rule_group_duration_seconds{quantile="0.05"} 0.000446404
prometheus_rule_group_duration_seconds{quantile="0.5"} 0.000612545
prometheus_rule_group_duration_seconds{quantile="0.9"} 0.000780038
prometheus_rule_group_duration_seconds{quantile="0.99"} 0.000823599
prometheus_rule_group_duration_seconds_sum 35.882038880000195
prometheus_rule_group_duration_seconds_count 53132
```

Histogram 可以再進一步做運算，應用上比較靈活

跟 Histogram 的比較說明：
https://prometheus.io/docs/practices/histograms/

Prometheus Client Libraries to Instrument Our Applications

https://prometheus.io/docs/instrumenting/clientlibs/

Official:

- Go
- Java or Scala
- Python
- Ruby

Third-party:

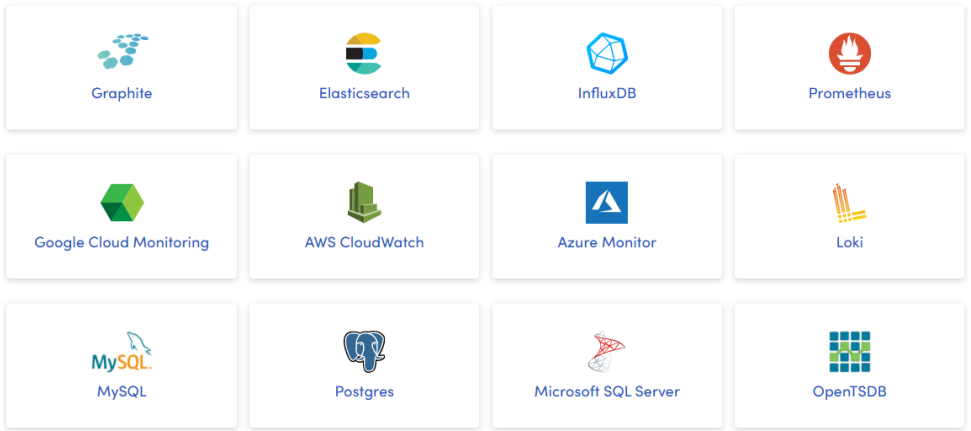
- C
- C++
- .NET / C#
- Node.js

Grafana

Grafana is open source visualization and analytics software. It allows you to query, visualize, alert on, and explore your metrics.

If deploy Grafana with multiple pods (HA), need to use third-party database likes MariaDB.

Data source guides



Hardware recommendations

Grafana does not use a lot of resources and is very lightweight in use of memory and CPU.

Minimum recommended memory: 255 MB Minimum recommended CPU: 1

Some features might require more memory or CPUs. Features require more resources include:

- Server side rendering of images
- Alerting
- Data source proxy

Supported databases

Grafana requires a database to store its configuration data, such as users, data sources, and dashboards. The exact requirements depend on the size of the Grafana installation and features used.

Grafana supports the following databases:

- SQLite
- MySQL
- PostgreSQL

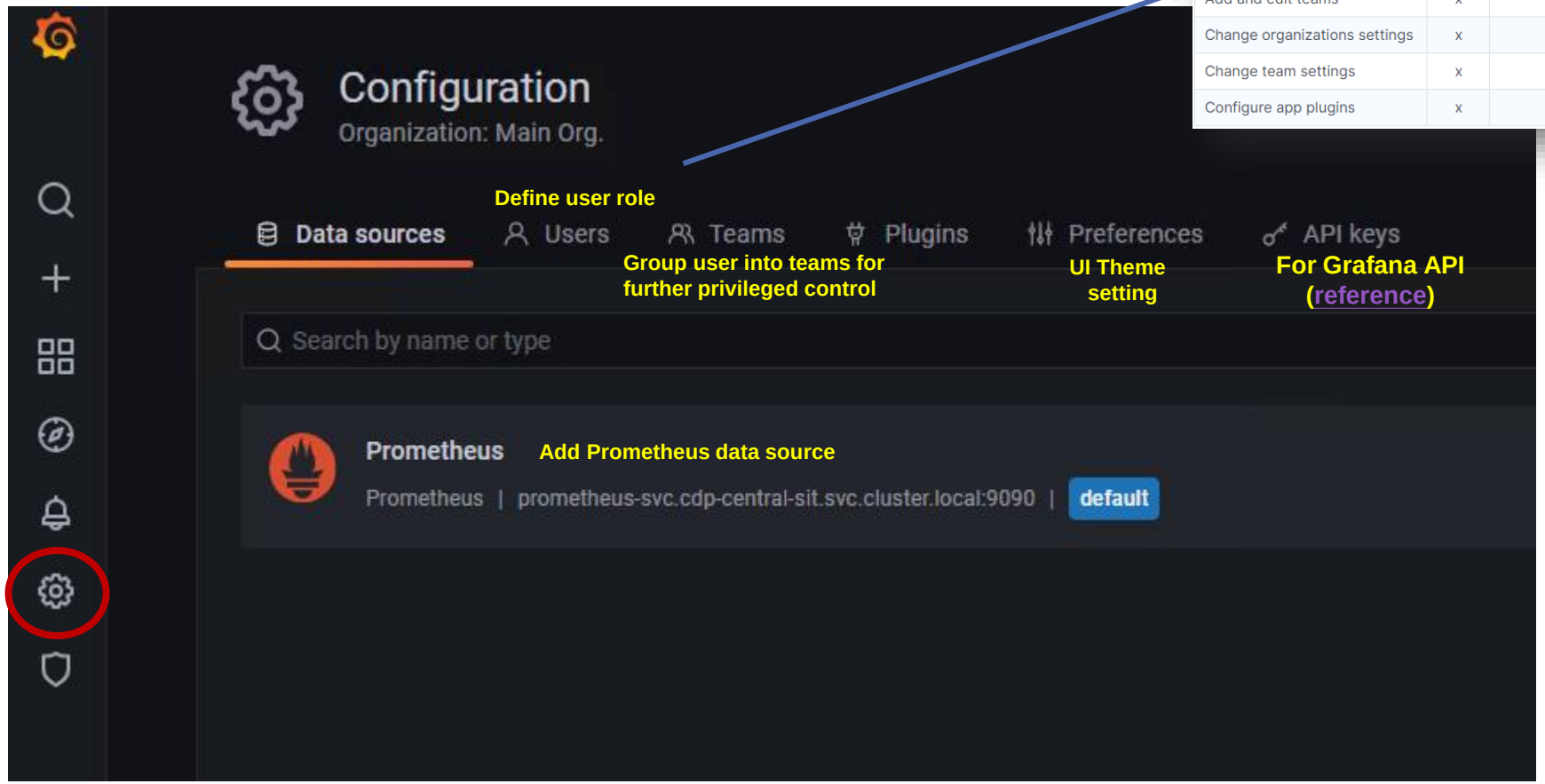
By default, Grafana installs with and uses SQLite, which is an embedded database stored in the Grafana installation location.

Grafana setting

Add data source before creating new dashboard

- Need ‘Admin’ role for configuration edit

	Admin	Editor	Viewer
View dashboards	x	x	x
Add, edit, delete dashboards	x	x	
Add, edit, delete folders	x	x	
View playlists	x	x	x
Create, update, delete playlists	x	x	
Access Explore	x	x	
Add, edit, delete data sources	x		
Add and edit users	x		
Add and edit teams	x		
Change organizations settings	x		
Change team settings	x		
Configure app plugins	x		



Dashboard design and setting

Import dashboard template (Templates)

Create customized dashboard with Graph, Table, Gauge and Stat, etc.

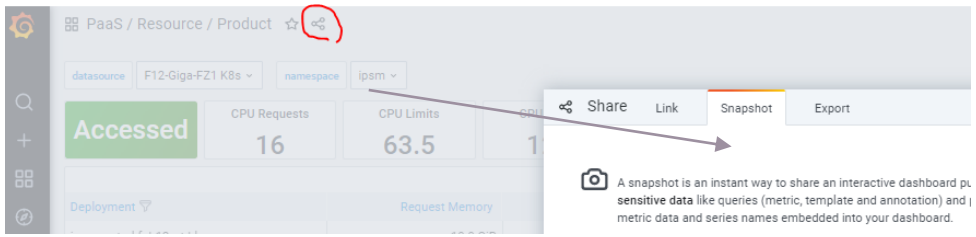
Add variables for multiple data sources (different clusters' Prometheus)

Versions control

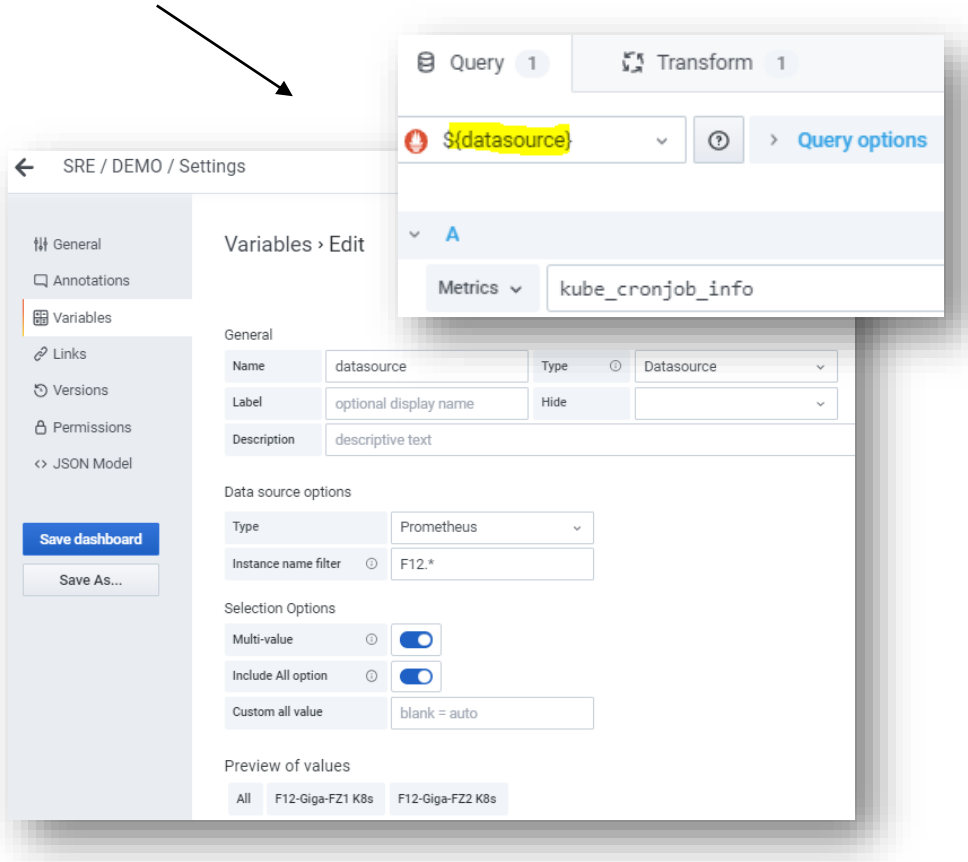
Data Table Export

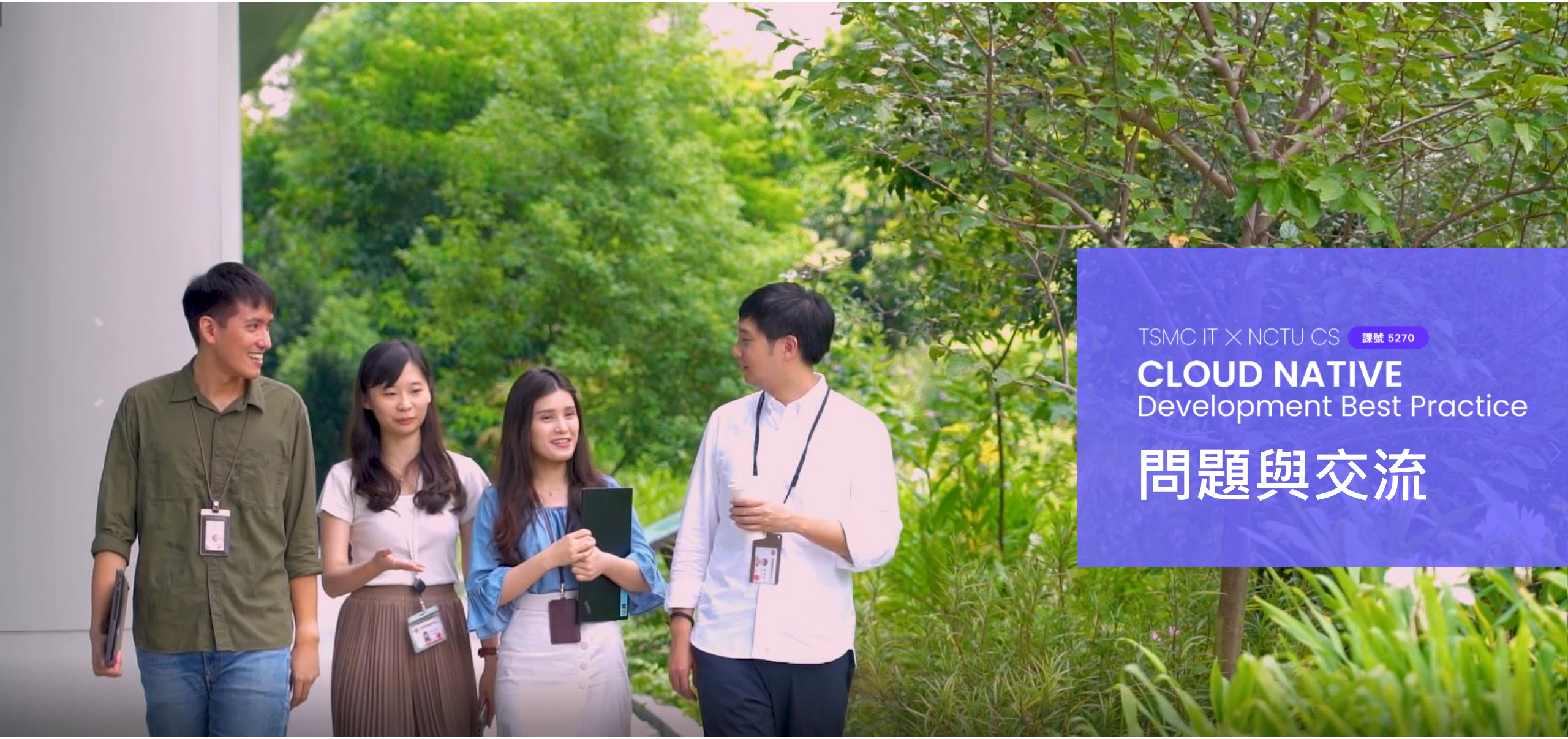
- Table → Inspect → Data → Download csv

Dashboard snapshot



[Optional] Enable Grafana 8 Alert





TSMC IT × NCTU CS 課號 5270

CLOUD NATIVE
Development Best Practice

問題與交流


TSMC IT X NCTU CS
課號 5270
CLOUD NATIVE
Development Best Practice
THANK YOU
FOR YOUR
ATTENTION

Auto Instrument (1/7)

OpenTelemetry auto instrument libraries are the option for someone who doesn't want to modify their application code for generating telemetry data(logs, metrics, and traces)

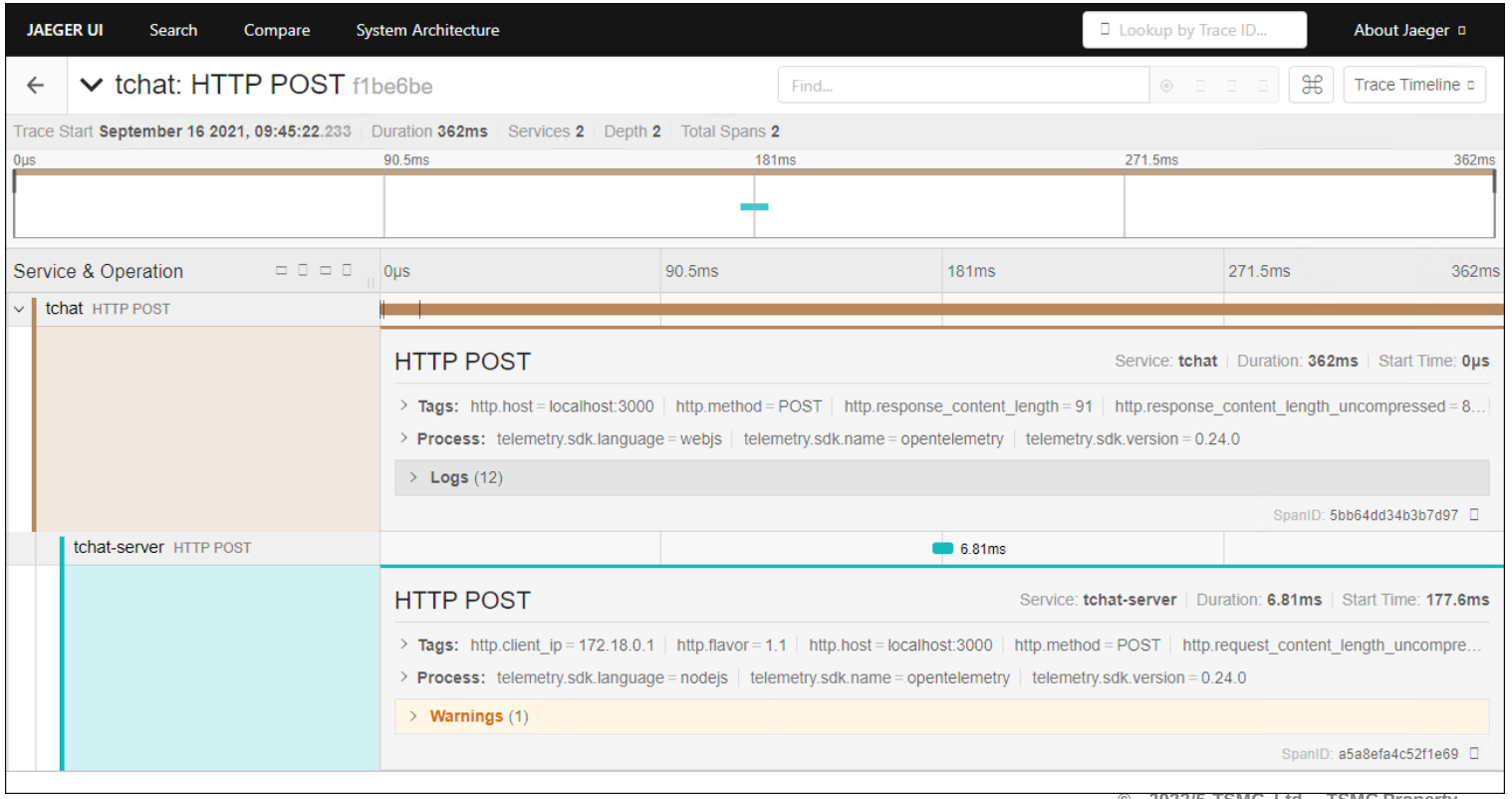
Broad Language Support

- Java
- C#
- Go
- **JavaScript**
- Python
- Rust
- C++
- Erlang
- Elixir

Auto Instrument (2/7)

Completed to adopt auto instrument libraries *(for JavaScript)* to collect trace info between frontend & backend for web application

- Document Load
- User Interaction
- Fetch
- XML HTTP Request
- Auto Instrument Node



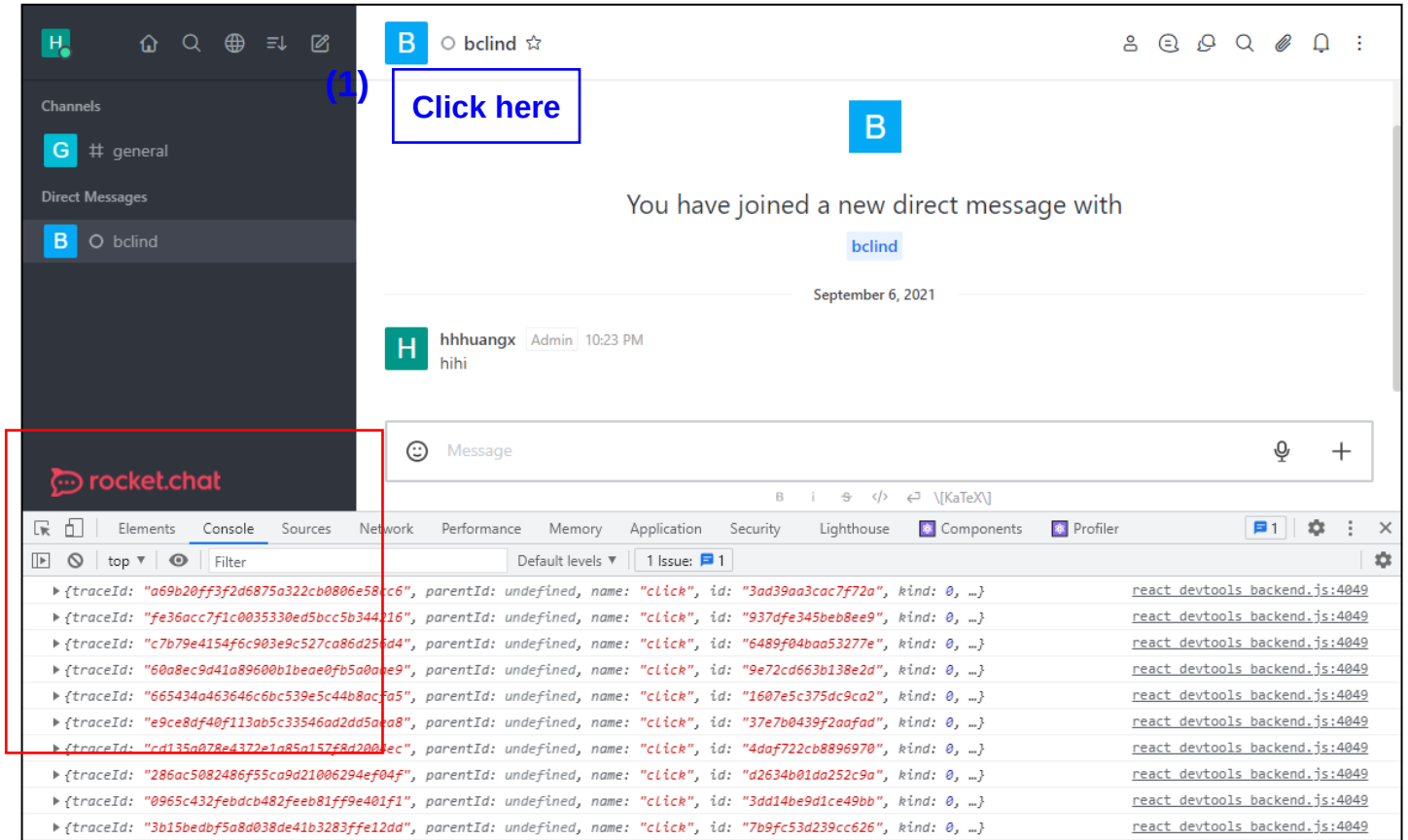
Auto Instrument (3/7)

User Interaction Instrumentation will generate too much trace info

- Ex: When user clicks anywhere, it will auto generate 12 traces

(1)

(2)

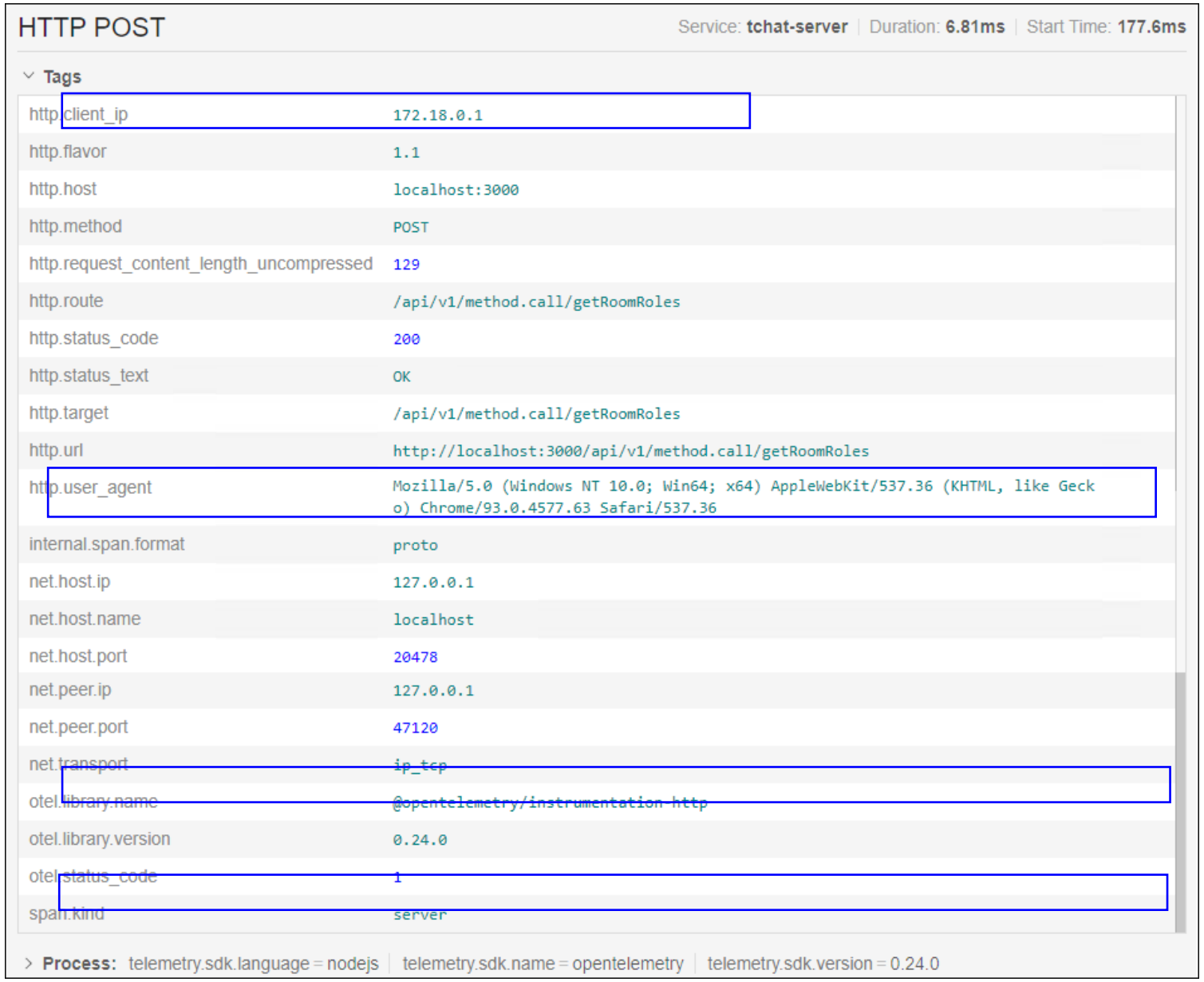


Auto Instrument (4/7)

Span info (frontend)

Auto Instrument (5/7)

Span info (backend)



Auto Instrument (6/7)

Send span data to Otel collector

```
45  export const SERVICE_NAME = 'tchat';
46
47  export const provider = new WebTracerProvider({
48      resource: new Resource({
49        [SemanticResourceAttributes.SERVICE_NAME]: SERVICE_NAME,
50      }),
51  });
52
53  provider.addSpanProcessor(new SimpleSpanProcessor(new ConsoleSpanExporter()));
54
55  const collectorOptions = {
56      url: 'http://localhost:55681/v1/traces',
57      headers: {},
58      concurrencyLimit: 10,
59  };
60  const exporter = new CollectorTraceExporter(collectorOptions);
61
62  provider.addSpanProcessor(new BatchSpanProcessor(exporter, {
63      maxQueueSize: 100,
64      maxExportBatchSize: 50,
65      scheduledDelayMillis: 500,
66      exportTimeoutMillis: 30000,
67  }));
```

Auto Instrument (7/7)

Add new attributes for spans you interest

- Step 1: Import required library *(for JavaScript)*

```
43  import { trace, context } from '@opentelemetry/api';
44
```

- Step 2: Input these program fragments *(for JavaScript)*

```
685         const span = trace.getSpan(context.active());
686         if (span !== undefined) {
687             span.setAttribute('metric_name', 'changeChannel');
688         }
```

- Step 3: New attribute “metric_name” will included in the specific spans

Navigation: /direct/G96rHqfwna4qBqrP8jNDpYfuLKToCjZv2p

Tags

component	user-interaction
event_type	click
http.url	http://localhost:3000/direct/G96rHqfwna4qBqrP8nLAXDp8fuYAKxmwos
http.user_agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML
internal.span.format	proto
metric_name	changeChannel
otel.library.name	@opentelemetry/instrumentation-user-interaction