Monitoring with Prometheus

Key Takeaways

TECHWORLD WITH NANA

Introduction to Prometheus

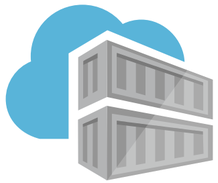
- Prometheus is an **open-source monitoring** system and **alerting** toolkit
- Prometheus is used widely and has an active community

- It gathers, organizes, and **stores metrics as time series data from targets** by "scraping" metrics HTTP endpoints
- Can trigger alerts when specified conditions are observed

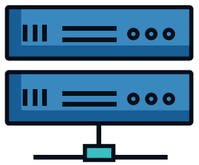
Why we need a monitoring tool - 1

Visibility in different environments

- You need **visibility in all kinds of environments**, but especially in a highly dynamic container environment, which is more challenging to monitor



Container environments

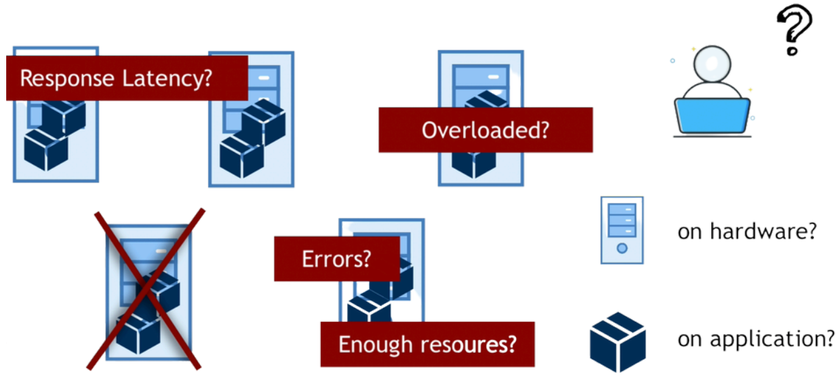


Bare servers

- For that - you need tools like Prometheus, which are designed for monitoring these types of environments

Visibility on different levels

- When you have 100s or 1000s of containers, plus components on multiple levels (infrastructure, platform, application) you need a way to have a visibility and **consistent monitoring across all these components**



- Without visibility, it's a black box for you. When things break inside your complex environment, you have **no idea what is happening**. You don't know what has caused the issue, what is not working.

Why we need a monitoring tool - 2

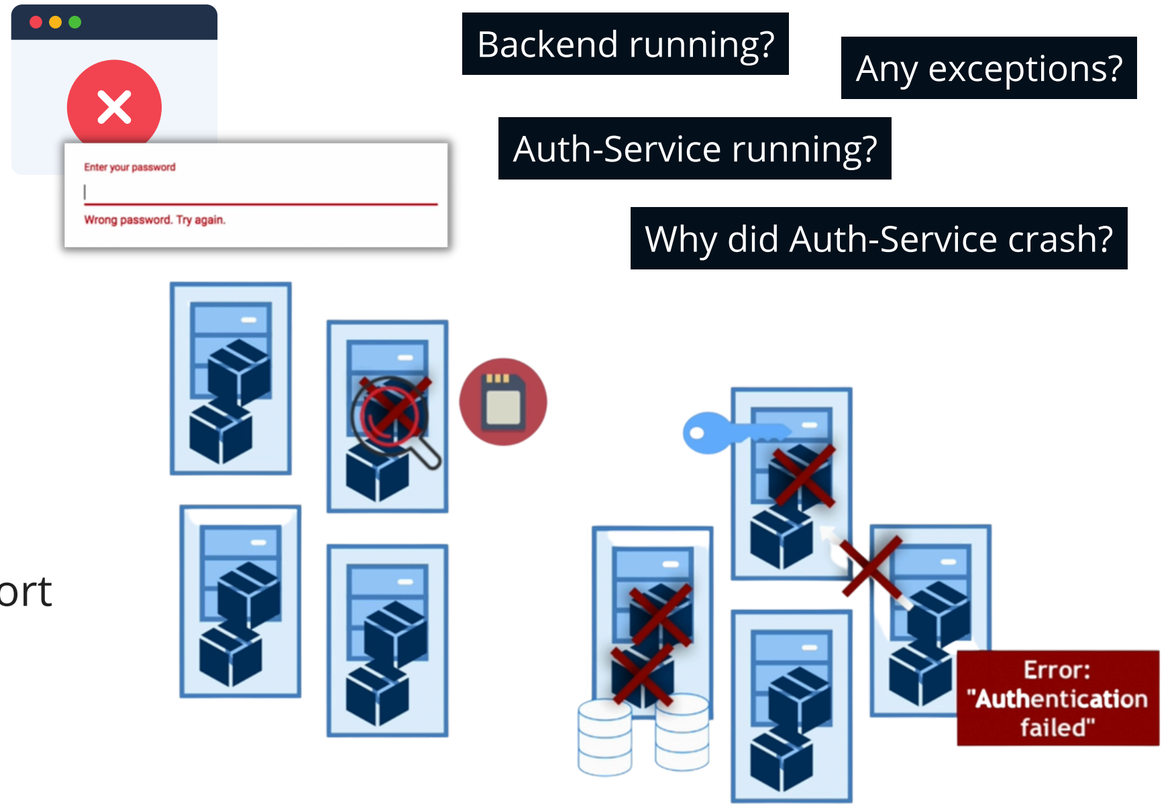
Example Use Case

Problem:

- Many application errors appear on the front-end to the end user
- They **only see the error message**, but the cause can be any of the many components in the back-end

Solution:

- Monitoring can help identify the problem quickly with little effort
- Instead of manually trying to troubleshoot across multiple components, it will **help pin point directly to the root cause**



- Saves you a lot of time and effort
- Company saves a lot of money

Why we need a monitoring tool - 3

With Prometheus, everything is automated. It's constantly monitoring and looking out for any issues in real time and may even identify a potential issue before it happens, so you can prevent it.

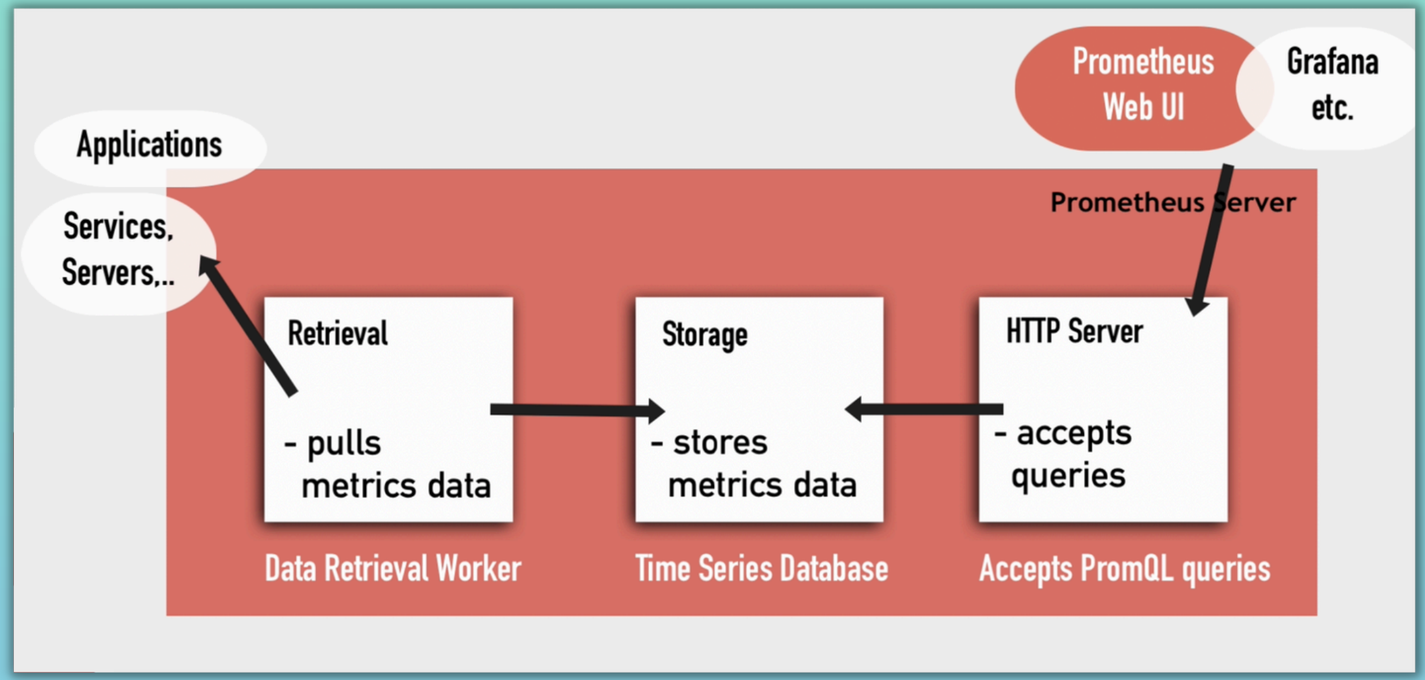
- Constantly monitors all the services
- Triggers alerts when a services crashes
- Helps to identify problems before they happen

Prometheus Architecture - 1

How it all works

Prometheus Server

- Is the **main component**
- Does the actual monitoring work
- Scrapes and stores time series data



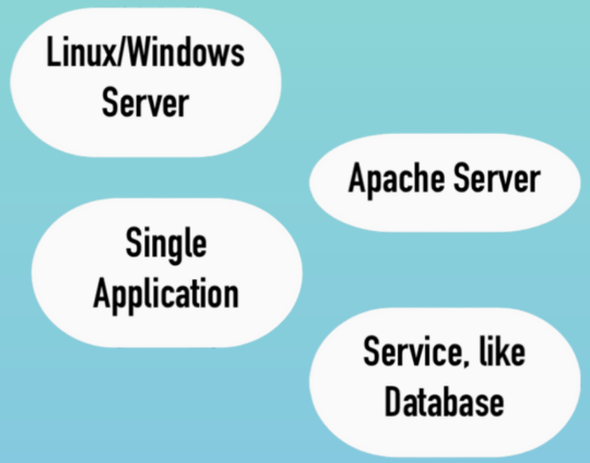
Prometheus Architecture - 2

How it all works: Targets & Metrics

- Prometheus pulls metrics from targets

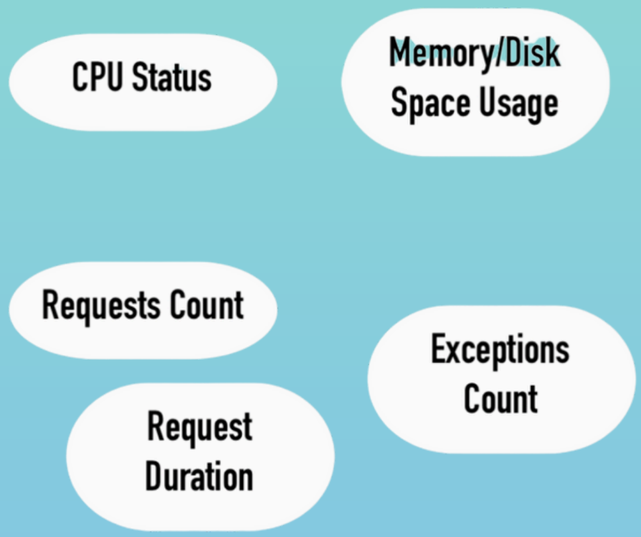
Targets

- **What** does Prometheus monitor?



Metrics

- **Which units** are monitored of those targets?



Prometheus Architecture - 3

How it all works: Metrics

- Metrics play an important role in understanding why your application is working in a certain way

Metric Entries

- Format: Human-readable text-based
- Metric entries consist of: **TYPE** and **HELP attributes**

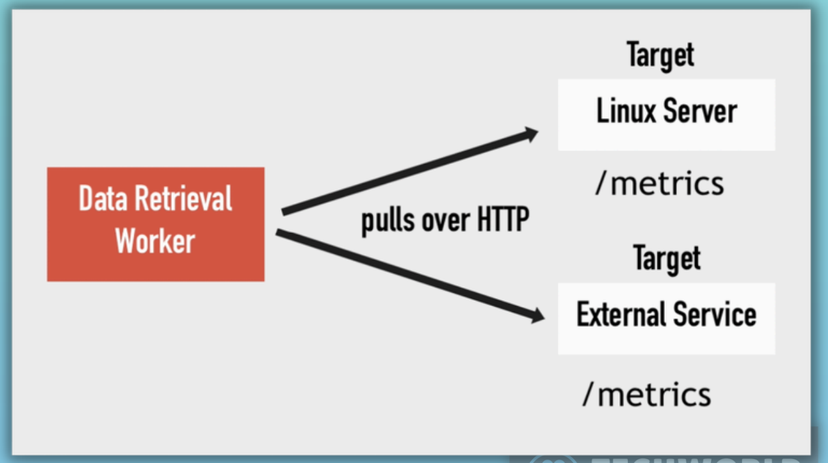
HELP 👉 description of what the metrics is

TYPE 👉 3 metrics types

1) Counter — **...how many times x happened**
2) Gauge — **...what is current value of x now?**
3) Histogram — **...how long or how big?**

How Prometheus collects Metrics Data from Targets

- Prometheus **pulls from HTTP endpoints**
- Targets must expose: [hostaddress]**/metrics**
- Must be in correct format that Prometheus understands



Prometheus Architecture - 4

How it all works: Exporters

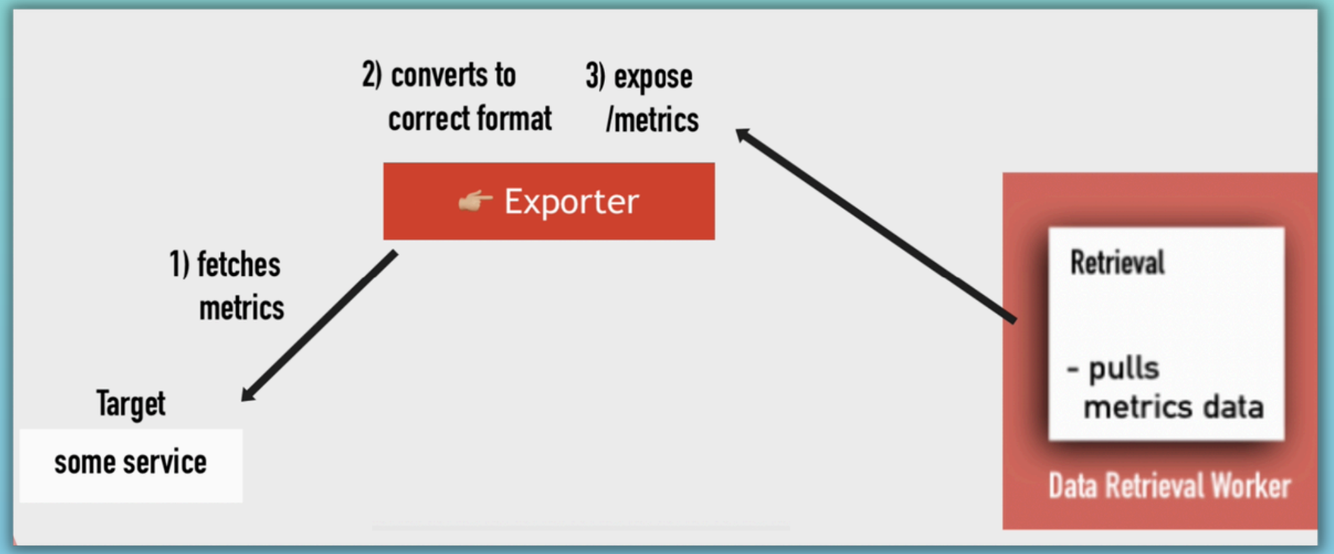
- Some services expose */metrics* endpoints by default
- Others need another component for that:

Official vs Third-Party

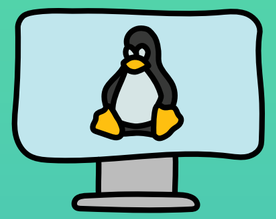
- Some are maintained as part of the official Prometheus organization, others are externally contributed and maintained.

Exporters

- Exporters help in exporting existing metrics from third-party systems as Prometheus metrics
- An exporter is a services that **fetches metrics from a target and converts the data and exposes them as Prometheus metrics**
- Prometheus can then scrape this endpoint as usual



TECHWORLD WITH NANA



Prometheus Architecture - 5

How it all works: Exporters

Example: Monitor a Linux Server

1. Download a **node exporter**
2. Untar and execute
3. Converts metrics of the server
4. Exposes /metrics endpoint
5. Configure Prometheus to scrape this endpoint

Hardware related

- apcupsd exporter
- BIG-IP exporter
- Collins exporter
- Dell Hardware OMSA exporter
- IBM Z HMC exporter
- IoT Edison exporter
- IPMI exporter
- knxd exporter
- Modbus exporter
- Netgear Cable Modem Exporter
- Netgear Router exporter
- Node/system metrics exporter (official)
- NVIDIA GPU exporter
- ProSAFE exporter
- Ubiquiti UniFi exporter

Example: Monitor own applications

- Client libraries let you **define and expose internal metrics** via an HTTP endpoint on your application's instance
- Metrics like:

How many requests?

How many exceptions?

Server resources used?

- Choose a Prometheus client library that matches the language in which your application is written

Exporters are available as Docker Images

TECHWORLD WITH NANA

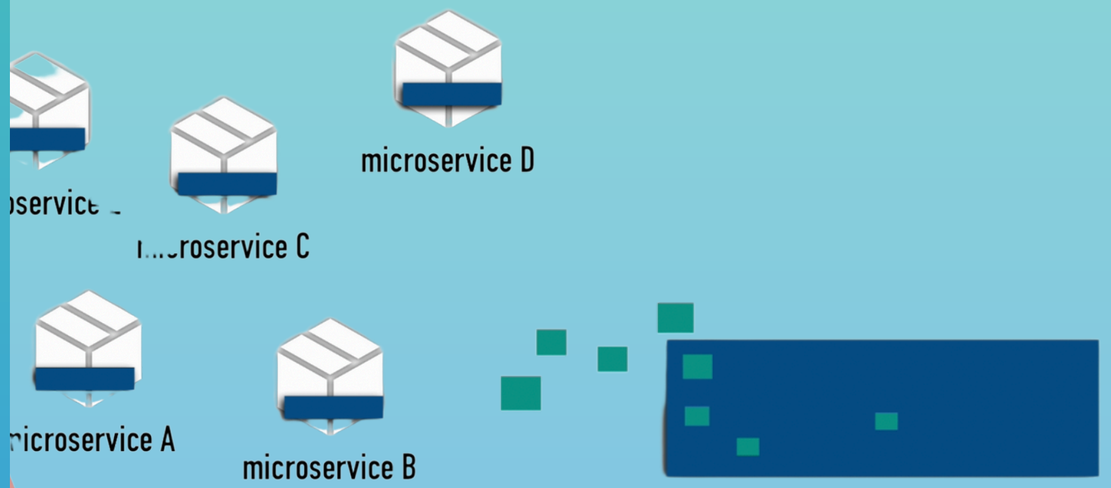
Prometheus Architecture - 6

How it all works: Push vs Pull

- **Important difference** of Prometheus compared to other monitoring systems like Amazon Cloud Watch or New Relic

Others - Push Model

- Services **push to a centralized collection platform**



- High load of network traffic
- Monitoring can become your bottleneck
- Installation of additional software to push metrics

Prometheus - Pull Model

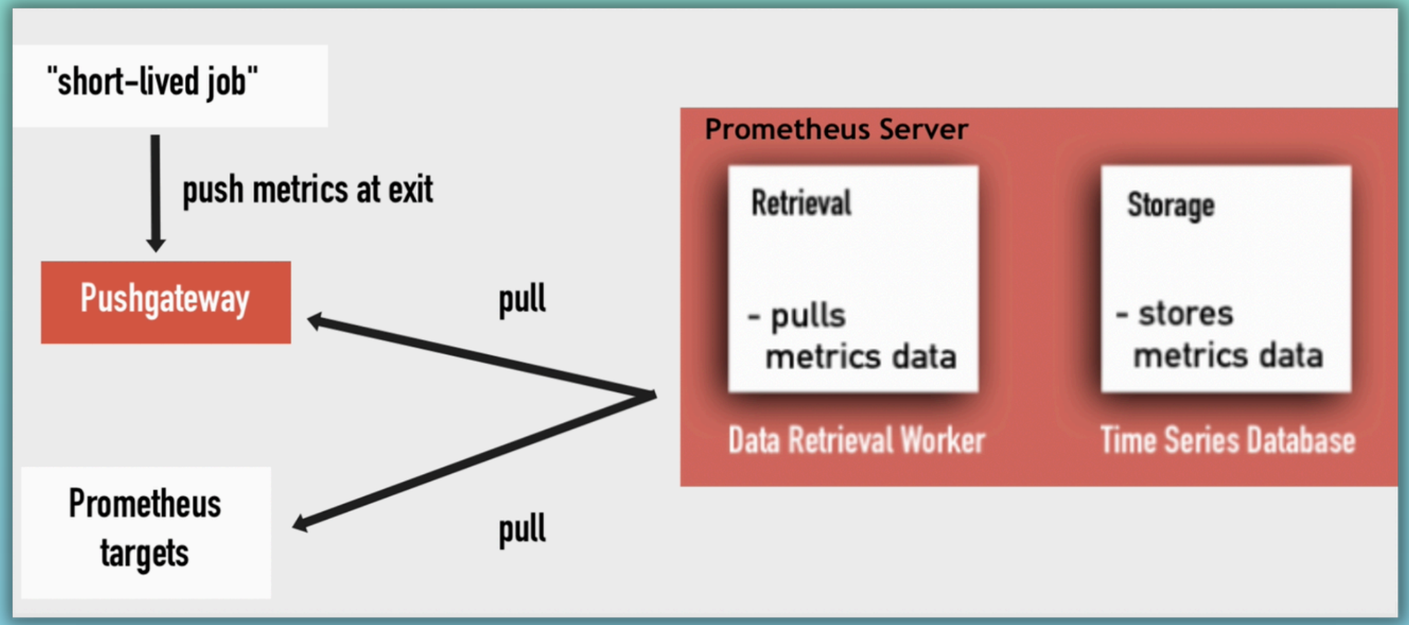
- Prometheus pulls metrics from endpoints

Prometheus Architecture - 7

How it all works: Pushgateway

Pushgateway

- An intermediary service, which **allows you to push metrics** from jobs, which cannot be scraped
- Prometheus recommends using the Pushgateway only in certain limited cases: Usually only valid use case for capturing the outcome of a **service-level batch job**



Prometheus Architecture - 8

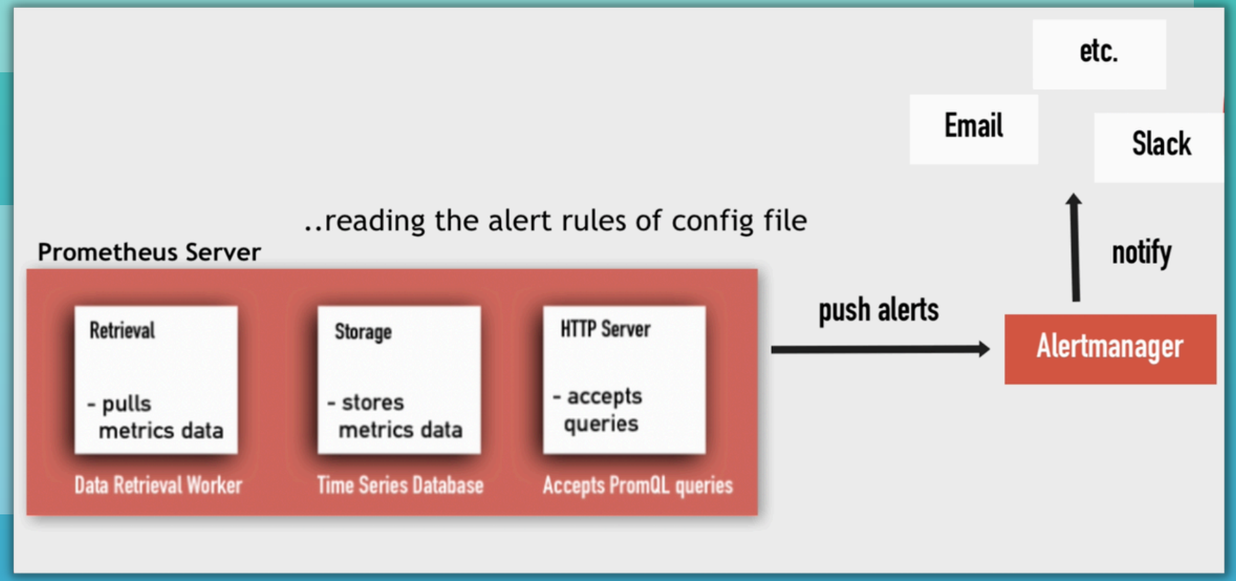
How it all works: Alertmanager

Alertmanager

- The Alertmanager handles alerts sent by Prometheus server
- Takes care of deduplicating, grouping and routing them to the correct receiver integrations

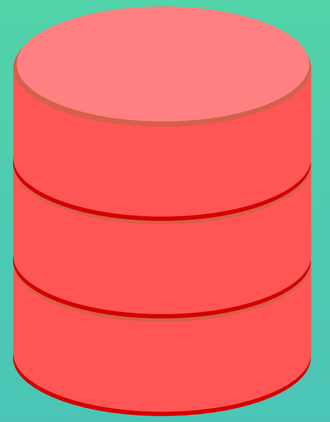
- Receiver of these alerts can be email,

  PagerDuty, Slack etc.



Prometheus Architecture - 9

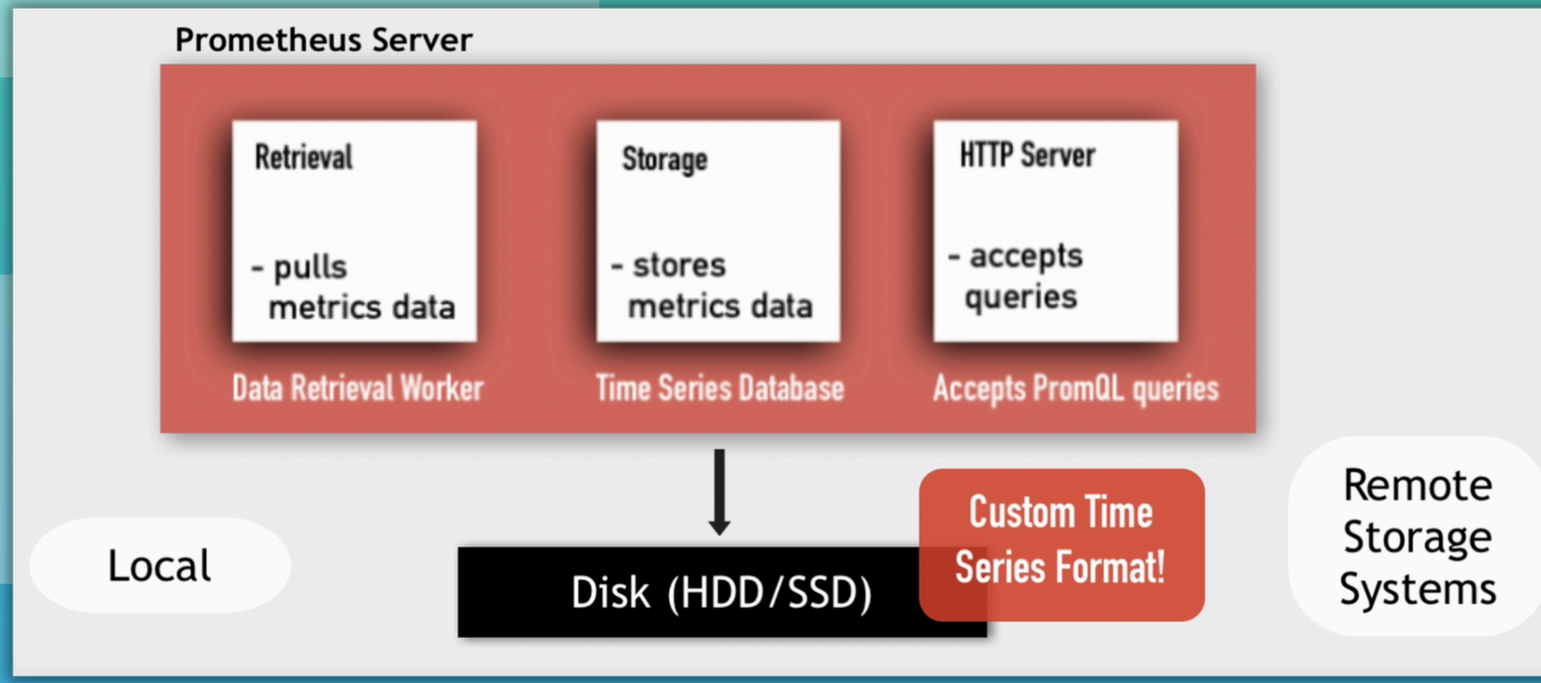
How it all works: Data Storage



Prometheus Data Storage

- Prometheus includes a local on-disk time series database
- But optionally integrates with remote storage systems

- Data in local storage is stored in a custom, highly efficient format

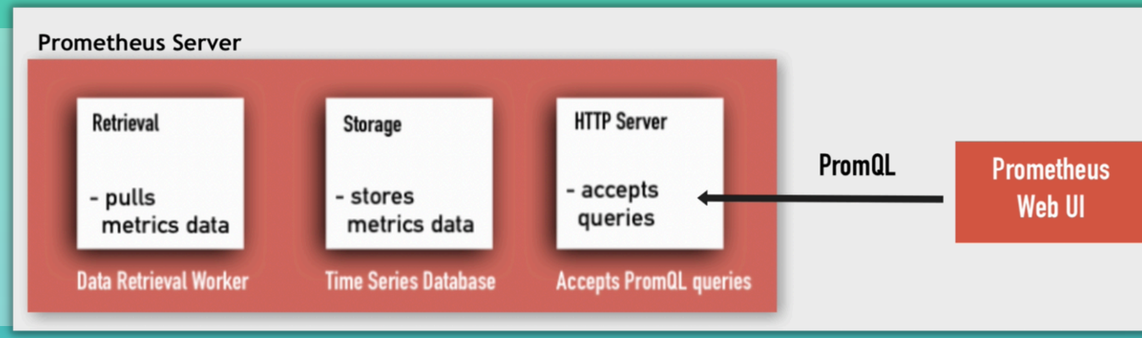


Prometheus Architecture - 10

How it all works: PromQL

Querying Prometheus

- Prometheus provides a functional query language called PromQL
- Let's user select and aggregate time series data in real time



Options to view result

1) Query target directly

2) Prometheus Web UI

3) Or use a more powerful visualization tool, e.g. Grafana

Example Queries:

```
http_requests_total{status!~"4.."}
```

👉 Query all HTTP status codes except 4xx ones

```
rate(http_requests_total[5m])[30m:]
```

👉 Returns the 5-minute rate of the *http_requests_total* metric for the past 30mins

Configuring Prometheus - 1

YAML Config

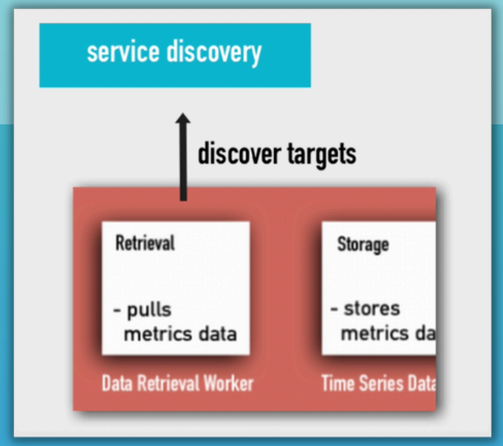
- You write your configuration in a prometheus.yaml file
- Let Prometheus know **what to scrape and when**:

Which targets?

At what interval?

prometheus.yaml

- Targets are discovered via a service discovery mechanism



Example Config File

- **How often** Prometheus will scrape its targets
- **Rules** for aggregating metric values or creating alerts when condition met
- **What resources** Prometheus monitors

```yaml
global:
  scrape_interval:     15s
  evaluation_interval: 15s

rule_files:
  # - "first.rules"
  # - "second.rules"

scrape_configs:
  - job_name: prometheus
    static_configs:
      - targets: ['localhost:9090']
```

Prometheus has its own */metrics* endpoint

Configuring Prometheus - 2

Define your own jobs

- Default values for each job:

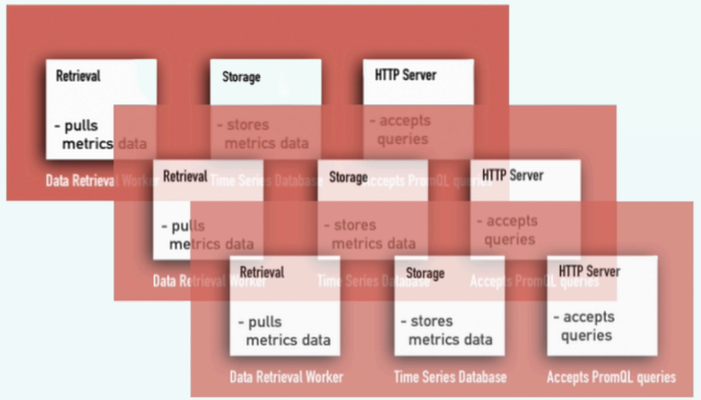
```
metrics_path: "/metrics"
scheme: "http"
```

```
global:
  scrape_interval:     15s
  evaluation_interval: 15s

rule_files:
  # - "first.rules"
  # - "second.rules"

scrape_configs:
  - job_name: "prometheus"
    static_configs:
      - targets: ['localhost:9090']
  - job_name: node_exporter
    scrape_interval: 1m
    scrape_timeout: 1m
    static_configs:
      - targets: ['localhost:9100']
```

Prometheus Characteristics

- Reliable
- Standalone and self-containing
- Works, even if other parts of infrastructure broken
- No extensive set-up needed
- Less complex

- Difficult to scale



- Limits Monitoring

Workaround:

- Increase Prometheus server capacity
- Limit number of metrics

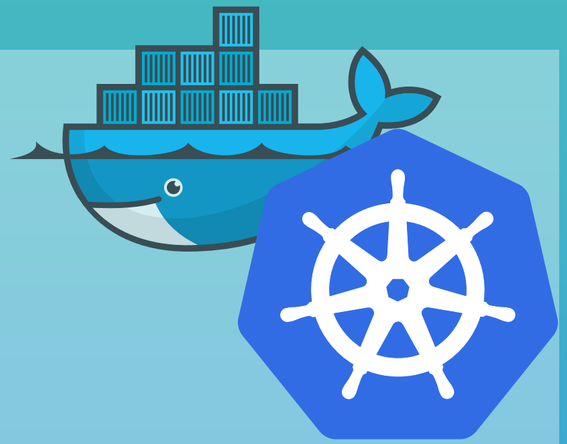
TECHWORLD WITH NANA

Other Prometheus Features

Prometheus Federation

- Allows Prometheus to **scale to environments with tens of data centers and millions of nodes**
- Allows a Prometheus server to scrape data from other Prometheus servers

Prometheus with Docker and Kubernetes



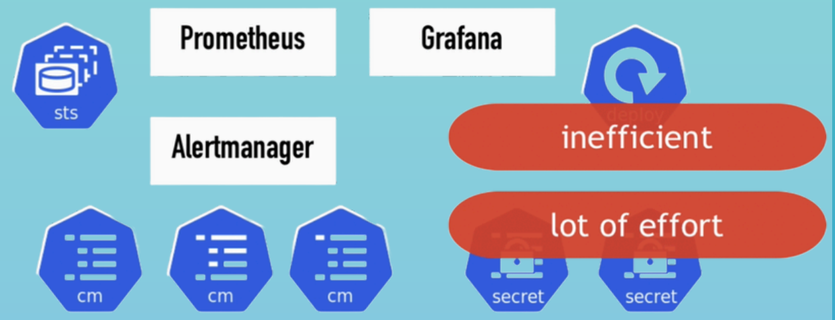
- Fully compatible
- Prometheus components available as Docker images
- Can easily be deployed in container environments like K8s
- **Monitoring of K8s cluster node resources out-of-the-box!**

Deploy Monitoring Stack - 1

3 different ways to deploy the Prometheus monitoring stack

1) Do it yourself

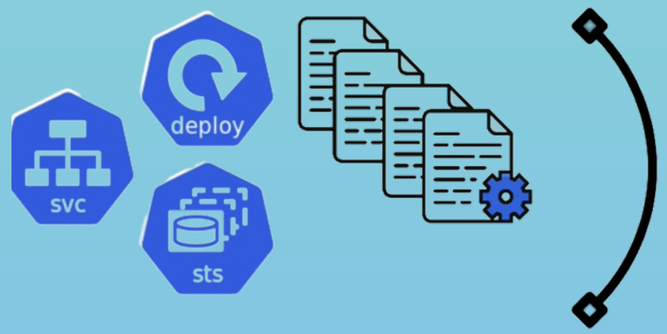
- **Create** all configuration **YAML files yourself**
- Execute them in right order



2) Using an Operator

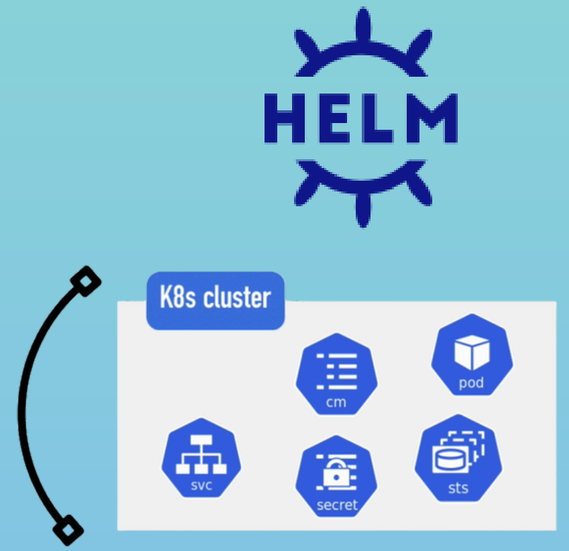
- Manager of all Prometheus components

1. Find Prometheus operator
2. Deploy in K8s cluster



3) Using Helm

- Using Helm chart to deploy operator
- Helm: Manage initial setup
- Operator: Manage setup



Deploy Monitoring Stack - 2

Overview of K8s resources deployed:

3 Deployments

- Prometheus Operator

  created Prometheus and Alertmanager StatefulSet

- Grafana
- Kube State Metrics

  => own Helm chart

  => dependency of this Helm chart

  => scrapes K8s components

1 DaemonSet

- Node Exporter DaemonSet

  => connects to server

  => translates Worker Nodes metrics to Prometheus metrics

  CPU usage ***load on server***

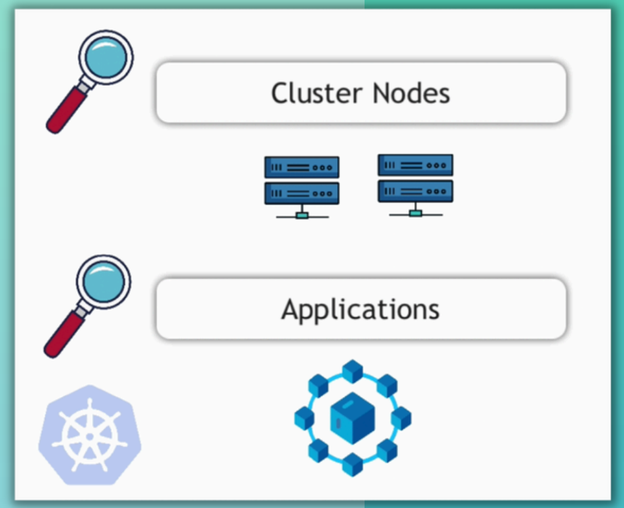
Data Visualization - 1

1st step: Decide what to monitor?

- Notice when something **unexpected** happens
- **Observe** any **anomalies**

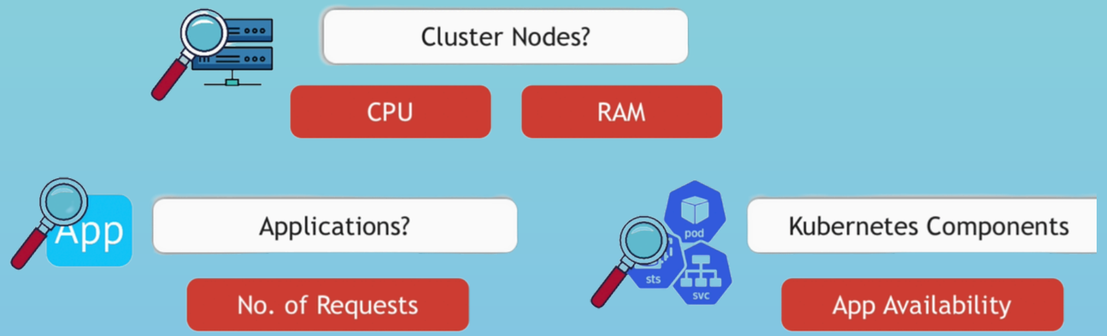
CPU spikes, insufficient storage, high load, unauthorized requests

- Analyze and **react** accordingly



2nd step: How to get this information?

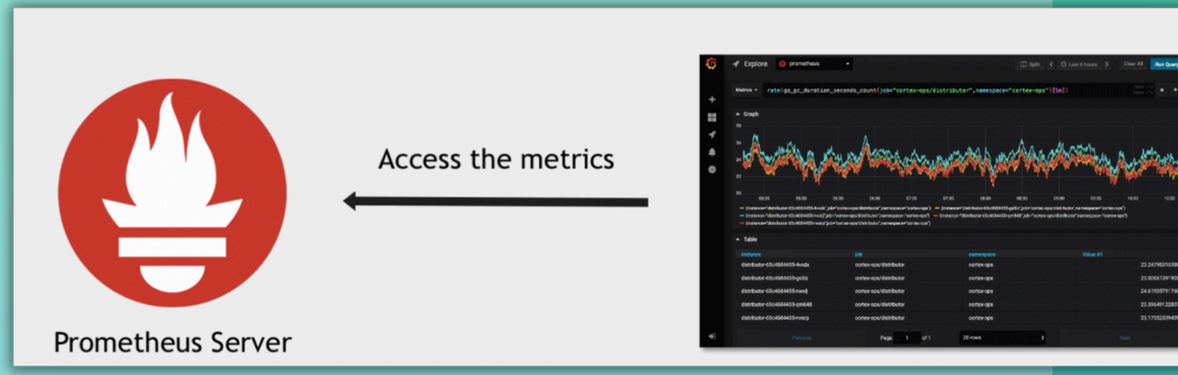
- How to get visibility of these monitoring data
- What data do we have available?



Data Visualization - 2

3rd step: Use a proper data visualization tool

- **Grafana** = a powerful open source visualization and analytics software
- Already deployed with the Prometheus Operator Helm Chart



Data Visualization Tool

TECHWORLD WITH NANA

Grafana

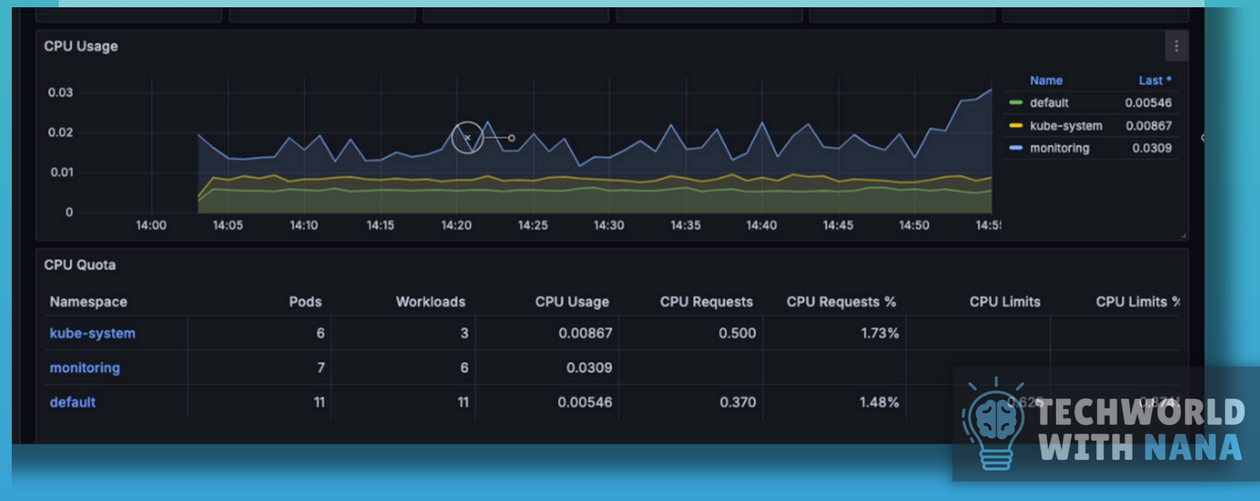
With Grafana you can **create dynamic and reusable dashboards** that allow you to visualize your data in any way you want

Dashboard

- Dashboard is a set of one or more panels
- You can create your own Dashboards
- Organized into one or more rows
- Row is a logical divider within a dashboard
- Rows are used to group panels together

Panel

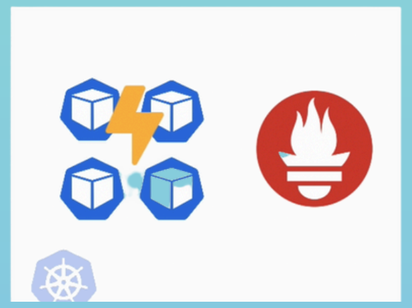
- The basic visualization building block in Grafana
- Composed by a query and a visualization
- Each panel has a query editor **specific to the data source** selected in the panel
- Can be moved and resized within a dashboard

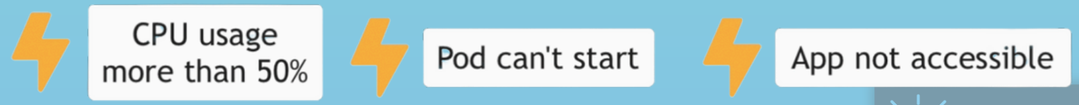


Alerting in Prometheus - 1

- Instead of constantly checking, you want to **get notified** when something happens
- Then you will check your dashboards

- For that we need to configure our monitoring stack to notify us whenever something unexpected happens





Alerting in Prometheus - 2

Configure Alerting

Alerting with Prometheus is separated **into 2 parts**:

1) **Alerting rules in Prometheus server** send alerts to an Alertmanager
2) **Alertmanager then manages (deduplicating, grouping, routing) those alerts**, including sending out notifications

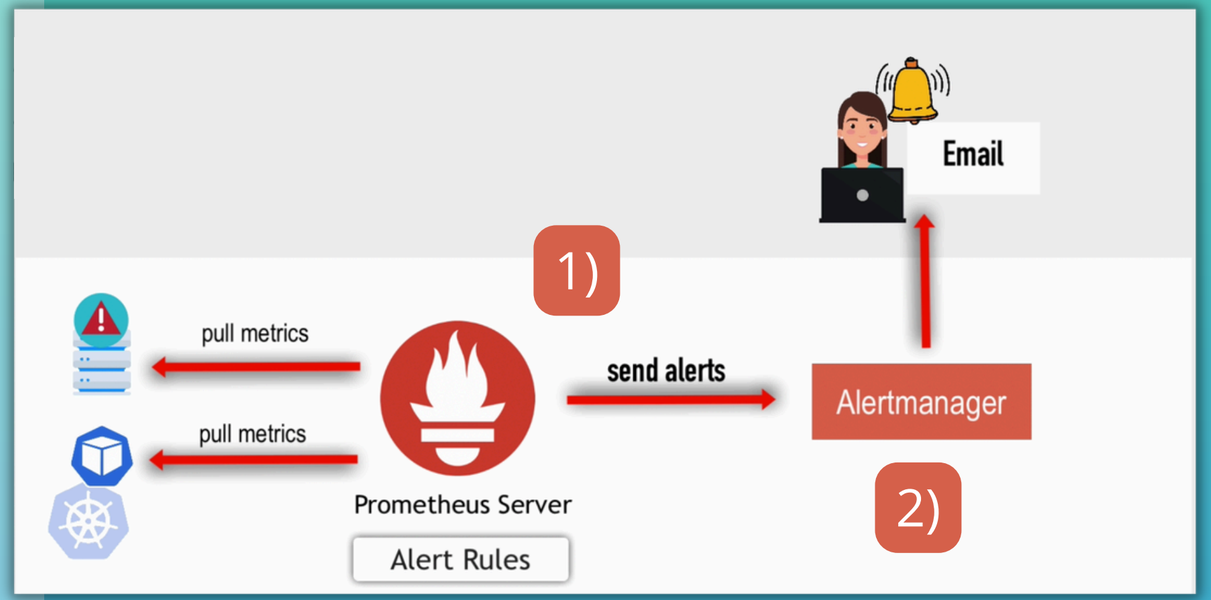
Main steps to setup alerting and notifications:

1. Setup and configure the Alertmanager
2. Configure Prometheus to talk to the Alertmanaer
3. Create alerting rules in Prometheus

Example Alert rules to configure

1st Alert: when CPU usage > 50%

2nd Alert: when Pod cannot start



Prometheus server and Alertmanager have each its own configuration file

TECHWORLD WITH NANA

Alerting in Prometheus - 3

Alertmanager example configuration

```
global:
  resolve_timeout: 5m
  http_config:
    follow_redirects: true
  smtp_hello: localhost
  smtp_require_tls: true
  pagerduty_url: https://events.pagerduty.com/v2/enqueue
  opsgenie_api_url: https://api.opsgenie.com/
  wechat_api_url: https://qyapi.weixin.qq.com/cgi-bin/
  victorops_api_url: https://alert.victorops.com/integrations/g
route:
  receiver: "null"
  group_by:
  - job
  continue: false
  routes:
  - receiver: "null"
    match:
      alertname: Watchdog
    continue: false
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 12h
receivers:
- name: "null"
templates:
```

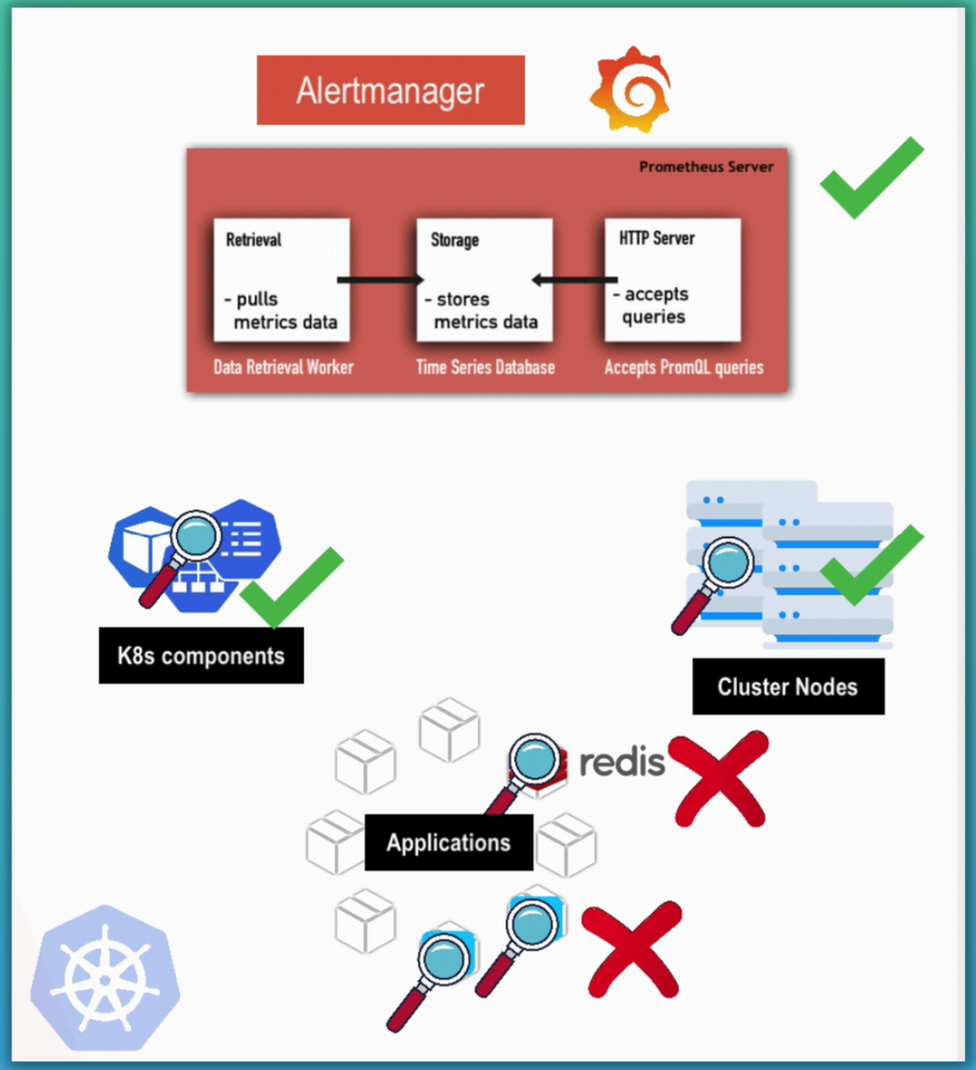
Any Alert

Specific Alerts

Receiver:

- These are the notification integrations
- For each alert you can define own receiver. For example:
  - send all K8s cluster related issues to admin email
  - send all application related issues to developer team's slack channel

Monitor third party and own applications - 1



Still missing:

Configure Third-Party and own application monitoring

- ✅ Monitor Kubernetes components
- ✅ Monitor Resource Consumption on the Nodes
- ✅ Monitor Prometheus Stack itself
- ❌ Monitor third-party applications like Redis
- ❌ Monitor own applications, like your online shop microservices

Monitor third party and own applications - 2

3rd-party example: Redis

- **Monitor Redis on application level**, not on Kubernetes level

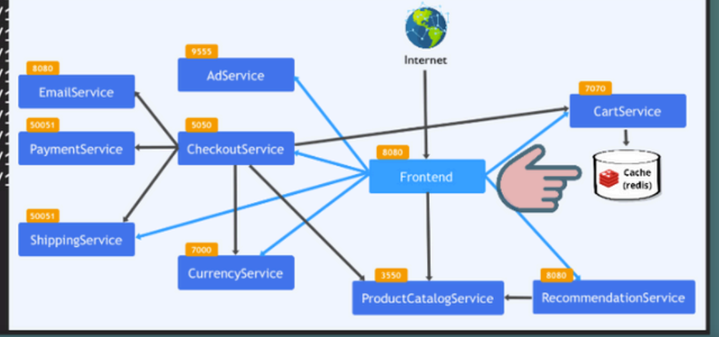
As we learnt, we can do that via an **Exporter**!

Too much load?

Too many connections?

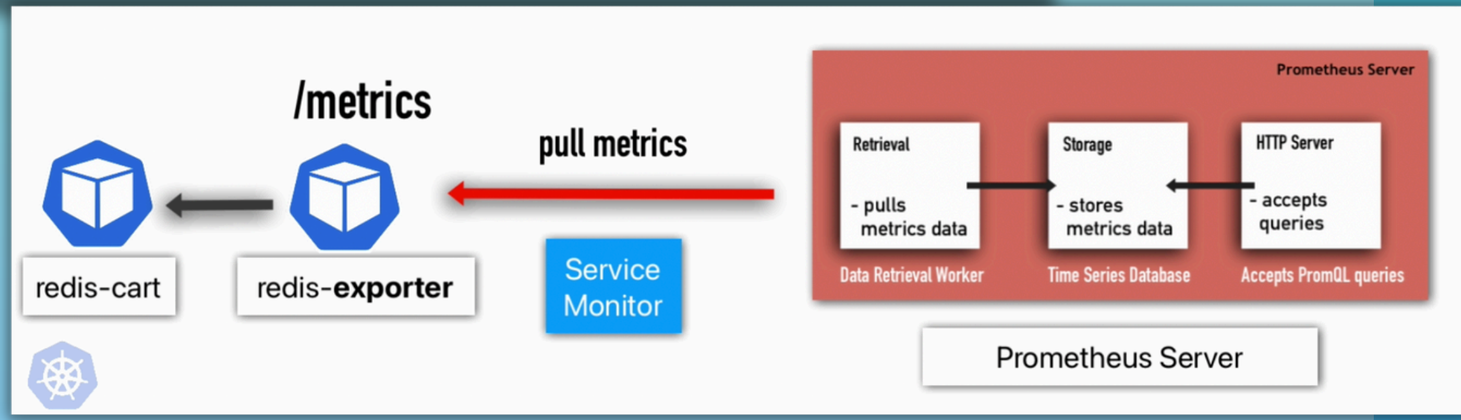
Redis is down?

```
nana@macbook /Users/nana/monitoring
% kubectl get pod
NAME                                 READY   STATUS    RESTARTS   AGE
adservice-7f5dc4b75f-qx5ct           1/1     Running   0          4d2h
cartservice-7986dbd956-mbsnv         1/1     Running   0          4d2h
checkoutservice-86bdcfbbfc-pnhsp     1/1
currencyservice-6f7fd6989-8kzpn      1/1
emailservice-6df48986c8-mzjmt        1/1
frontend-6f7d9ff6bc-qqzt2            1/1
paymentservice-c5776df6f-xs8ld       1/1
productcatalogservice-7856db7589-rrfzl 1/1
recommendationservice-6bd89f88fc-8cg59 1/1
redis-cart-8c5bbbccf-sv4rs           1/1
shippingservice-5cc877bc4c-7q527     1/1
nana@macbook /Users/nana/monitoring
%
```



How to:

1. Deploy redis-exporter
2. Deploy ServiceMonitor (custom K8s resource) to tell Prometheus about this new exporter



Monitor third party and own applications - 3

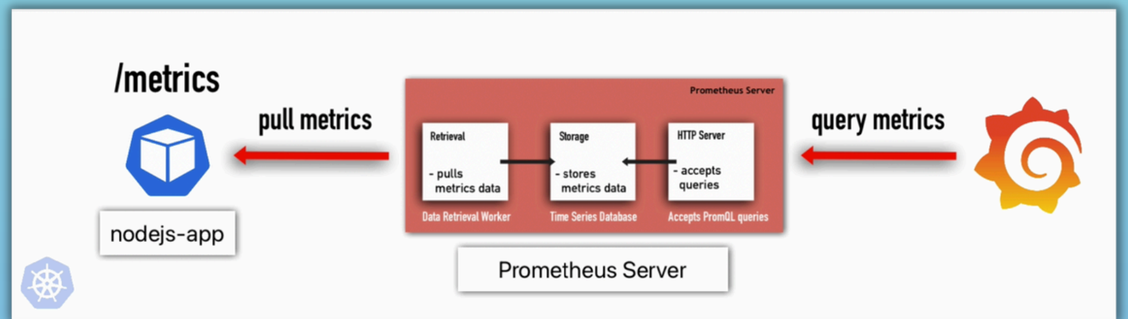
Own application

- No exporter available for your own application
- So we have to define the metrics ourselves

As we learnt, we can do that via **Client Libraries**!

How to (Nodejs application):

1. Expose metrics using Nodejs client library
2. Deploy Nodejs application in the cluster
3. Configure Prometheus to scrape new target (ServiceMonitor)
4. Visualize scraped metrics in Grafana Dashboard



Client Libraries:

- Gives you an abstract interface to expose your metrics
- Libraries implement the Prometheus metric types: Counter, Gauge, Histogram, Summary
- Choose client library that matches the application's language

Best Practices

Official Best Practices:

- Metric and Label Naming: https://prometheus.io/docs/practices/naming/
- Set of guidelines for instrumenting your code: https://prometheus.io/docs/practices/instrumentation/
- Consoles and Dashboards: https://prometheus.io/docs/practices/consoles/
- Alerting: https://prometheus.io/docs/practices/alerting/
- On when to use the Pushgateway: https://prometheus.io/docs/practices/pushing/