# What is Prometheus?

A monitoring tool to monitor **highly dynamic container environments**, can also be used on bare servers with applications directly deployed on it

Prometheus has become **mainstream monitoring tool of choice** **in container and micro services infra structure**

Mainly **automates monitoring** and **generates alerts**

# Where and why is Prometheus used?

Managing Devops manually is becoming more and more complex and therefore needs some automation because, when you have **many servers running containerized applications** where some of them are interconnected, **maintaining such setup smoothly without application downtime is challenging, it becomes even more tough when you have servers deployed across different regions in world, you might have no idea on what’s happening at hardware level or application level regarding issues such as response latency, hardware down time, running out of resources, errors, over loaded containers etc.,**

**Also if one of the container stops running because of some issue, it also effects/stops execution of other containers, in a typical hundreds/thousands of containers we might have to quickly spot where’s it going wrong and Prometheus helps us here!**

# Specific use cases for Prometheus monitoring:

An analogy: there are a group of servers running containerized applications, one specific server ran out of memory and kicked out a container which is responsible for letting other 2 db containers in sync, now the 2 db containers are not running, another container handling authentication requests from UI will not be able to work, as Authentication from UI container is not working, container that is running webapp wont work, from UI perspective only we can see we are not able to login, but we might have to backtrack to see which container is creating the issue (there might be multiple servers running hundreds of containers)

## What does Prometheus offer for above analogy?

* Constantly monitor all the services (like space issue, network loads)
* Alert when crash
* Identify problems before

# Architecture of Prometheus server:

Main components in Prometheus is **Prometheus server,** architecture as follows:

Diagram

Description automatically generated

### Flow of control:

**Data retrieval worker** pulls data from the entity that Prometheus wants to monitor, then stores the data in **time series database**, which is HTTP queried by **Prometheus Web UI or a visualization tool such as Grafana**, these HTTP queries are handled by **PromQL queries**

# What does Prometheus monitor?

Prometheus almost monitors everything such as monitoring linux/windows server or apache server or single application, or any other database.

## Which units are monitored?

There might be many parameters of an entity that Prometheus is monitoring, parameters such as CPU status, request count, request duration, exceptions count, memory/disk usage.

## What are targets, what are metrics:

### Targets:

**entities that are monitored** by Prometheus are called targets

### Metrics:

**parameters of targets which are being monitored** are metrics, metrics get saved in to time series database component discussed in Prometheus architecture above

format: human readable text format

metrics entities: type and help attributes

#### help:

description of what the metrics is

#### type:

divided In to 3 categories

1. counter: how many times ‘x’ happened
2. gauge: what is the current value of ‘x’ now
3. histogram: how long or how big is it

## Collecting metrics data from targets:

**Data retrieval worker** pulls over data from targets through HTTP

1. data is pulled from http end points of targets
2. http end point will be hostaddress/metrics
3. data available at end points must be in correct format in a way that Prometheus understands

## Targets End points and Exporters:

Some components expose /metrics endpoints by default

Many services need other external tool called exporter

Exporter: a script/service that fetches metrics from target, converts them to correct format that Prometheus understands, exposes them at its own exporters /metrics endpoint from where the **data retrieval worker** fetches it for **Prometheus monitor**

Also **Prometheus has list of its own official exporters for mostly used targets such as mysql, elastic search, linux server and so on…**

When you are monitoring a target which does not have its own metrics end point but have an Prometheus offered exporter, then you have to

1. download a node exporter
2. untar and execute
3. exporter will convert metrics of the server, exposes them at its own endpoint
4. configure Prometheus to scrape this end point

**Exporters are also available as docker images,** if you are running a mysql container, you can run docker image of mysql exporter as a **side car container**

## Monitoring your own applications:

You might be having custom requirements of **request count or exception count or server resource usage as metrics** for your own application

For such use case you can use terraform supported client libraries to expose /metrics endpoin0074

## Pull mechanism:

Terraform pulls data from endpoints, whereas other monitoring systems such as **amazon cloud watch** and **new relic** follow push system where applications/servers push data to centralized collection platform of that monitoring tool.

Disadvantages of push system:

1. High load of traffic
2. Which makes monitoring a bottleneck
3. Need to install daemon or other additional software to push metrics to.

Advantages of pull system:

1. Prometheus pull system will just need an endpoint to be installed to make the scraped data available
2. Multiple Prometheus instances can pull metrics data
3. Better detection/insight whether the service is up and running (if prometheus is not able to pull data, there might be something wrong with the service)

### Push gateway- an exception to pull mechanism of prometheus:

Pull mechanism holds good for scenarios where the target is running for a considerable period of time so that when prometheus tried to pull data the instance is up and running, but when the target is short lived then terraform might not be able to pull data as before prometheus pulls data the target might have been dead, for such cases prometheus offers an exception **push gateway** where a target can push metrics to **data retrieval worker**

**Diagram

Description automatically generated**

# Configuring Prometheus – (how does Prometheus know what and when to scrape):

All the configurations directing Prometheus when and what to scrape are configured in prometheus.yml

ex: which targets to scrape, at what interval

Prometheus uses **service discovery** to discover targets

Text

Description automatically generated

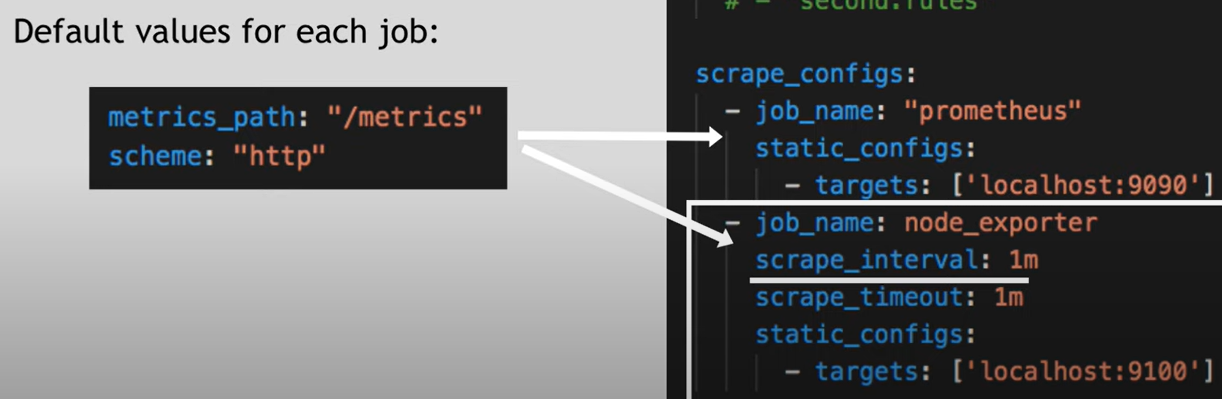
In the above yaml under **global block**-

scrape interval is the default interval for prometheus to scrape targets, it can be overridden at a target level if needed

evaluation interval: how often will it evaluate rules declared in **rules\_file block**

rule\_file: to specify rules such as to pull metric values or to create alerts (ex: CPU usage reached 80%)

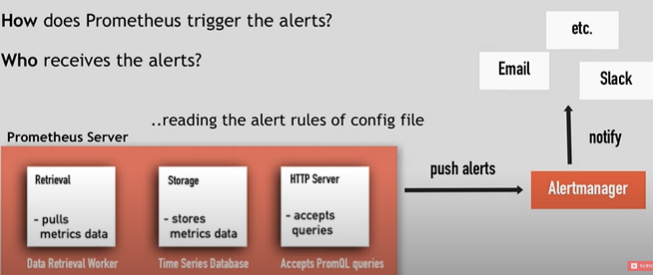
**scrape\_configs block:** this block will have data of which targets does prometheus pull scraped information from, as prometheus has its own /metrics endpoint, in the above image, it is using prometheus endpoint to monitor its health



In the above image, we can see another job node\_exporter, whose scrape interval is overridden to 1m (whereas default scrape interval is 15s) afaik data to be pulled by Prometheus will be available at http:localhost:9100/metrics

# Alert Manager:

Prometheus has a component called alert manager which fires alerts through mail/slack based on the alert rules in the config file



# Prometheus data storage:

Prometheus stores metrics data on local disk in time series format (so you cant write data directly in to relational database), also integrates with remote storage system, it also lets you query stored metrics on targets using server API through PromQL queries

Diagram, timeline

Description automatically generated

You can use PromQL to query using Prometheus web UI t, or Grafana (a powerful visualization tool which uses PromQL under the hood)

Diagram

Description automatically generated

# Prometheus characteristics:

Graphical user interface, text, application

Description automatically generated

# Prometheus with docker and Kubernetes:

