Overview

What is the need for monitoring tools?

* Some of the parameters/metrics that need consistent monitoring include
  + Pod resources,
  + Memory usage,
  + CPU utilization,
  + Network bandwidth,
  + Disk pressure,
  + etc.
* Ideally, these clusters should take **corrective action** as soon as the parameters mentioned above slightly exceed their threshold. If this is not the case, it must at least ensure that **alerts** are generated so that issues arising in the clusters can be taken care of manually.

**Use cases of Monitoring**

1. Scan all servers and see if the memory usage has gone beyond 75% and immediately alert the administrator.
2. Elasticsearch doesn’t accept any new logs because the server ran out of disk space or elasticsearch reached the storage limit that was allocated for it. Monitoring tool should compare continuously the storage space and compare with the elastic search consumption of space of storage and it will see the risk and notify maintainers of the possible storage issue and you can tell the monitoring tool what that critical point is when the alert should be triggered.
3. App have become very slow and then has cascade effect on other services.

**Monitoring Cluster for Memory and CPU**

**Metrics Server offers:**

* A single deployment that works on most clusters.
* Fast autoscaling, collecting metrics every 15 seconds.
* Resource efficiency, using 1 mili core of CPU and 2 MB of memory for each node in a cluster.
* Scalable support up to 5,000 node clusters

**To install the Metrics Server:**

**For Docker Desktop:**

**1. Download and save the below file locally:**

https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

#Add the below line to metrics server's container args, around line 135

- --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname

**- --kubelet-insecure-tls=true**

2. Deploy the Metric Server

kubectl apply -f <https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml>

3. Verify that the Metric Server is started successfully

kubectl get pods -n kube-system

**Note: If you get error, use the YAML provided in the below (its older version 4.5 but works)**

<https://stackoverflow.com/questions/72239288/kubernetes-metrics-server-not-starting-up-locally>

**For MiniKube:**

minikube addons enable metrics-server

**To Verify if metric server has started**

kubectl get pods -n kube-sysem

**# To get the memory and CPU usage.**

kubectl top pods --containers

kubectl top nodes

kubectl top pods --sort-by=cpu

kubectl top pods --sort-by=memory

#We can look at our system pods, CPU and Memory

kubectl top pods --all-namespaces

Note: The **kubectl top** command doesn’t actually collect any **metrics** itself. It queries the Metrics API for the metrics and presents them to you. In most clusters, especially those provided by cloud services, the Metrics API will already be installed.

Prometheus

**What is Prometheus?**

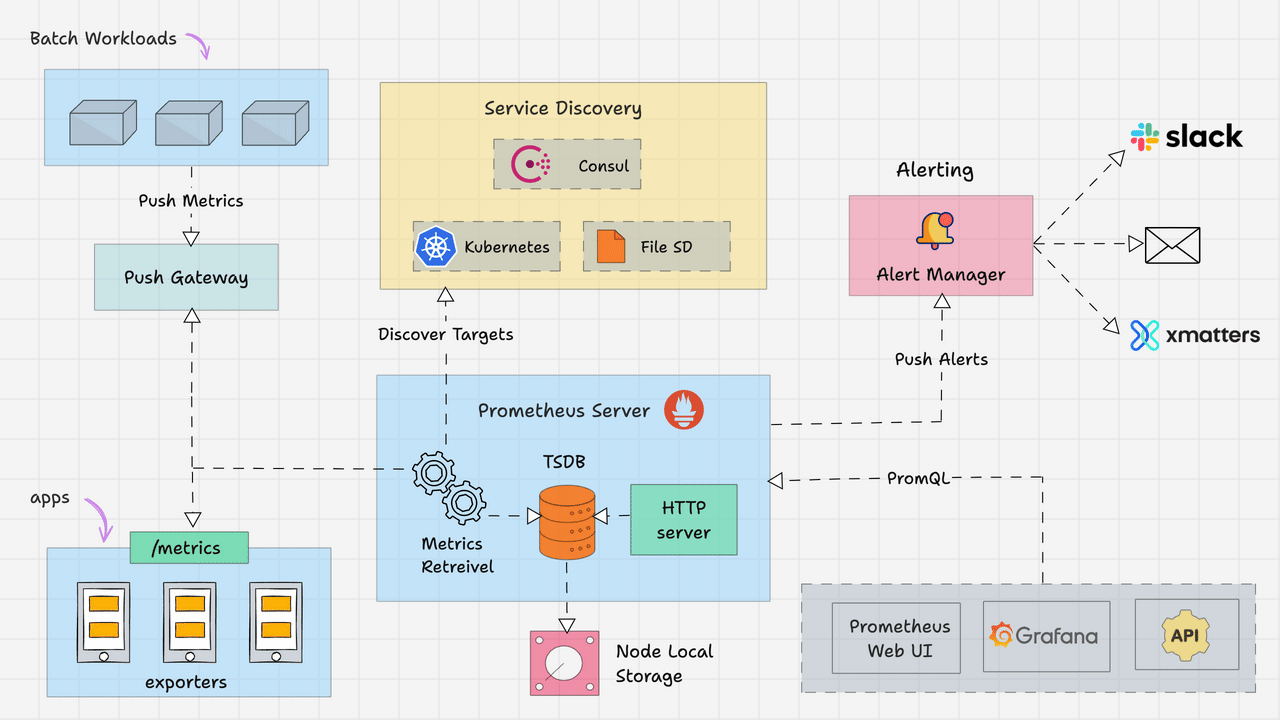
Prometheus is a system monitoring and alerting system.

It was open sourced by SoundCloud in 2012 and is the **second project** both to join and to graduate within Cloud Native Computing Foundation after Kubernetes.

It pulls the real-time metrics, compresses and stores them in a **time-series** database. key-value pairs called as labels can also be stored along with metrics.

**Prometheus Architecture**

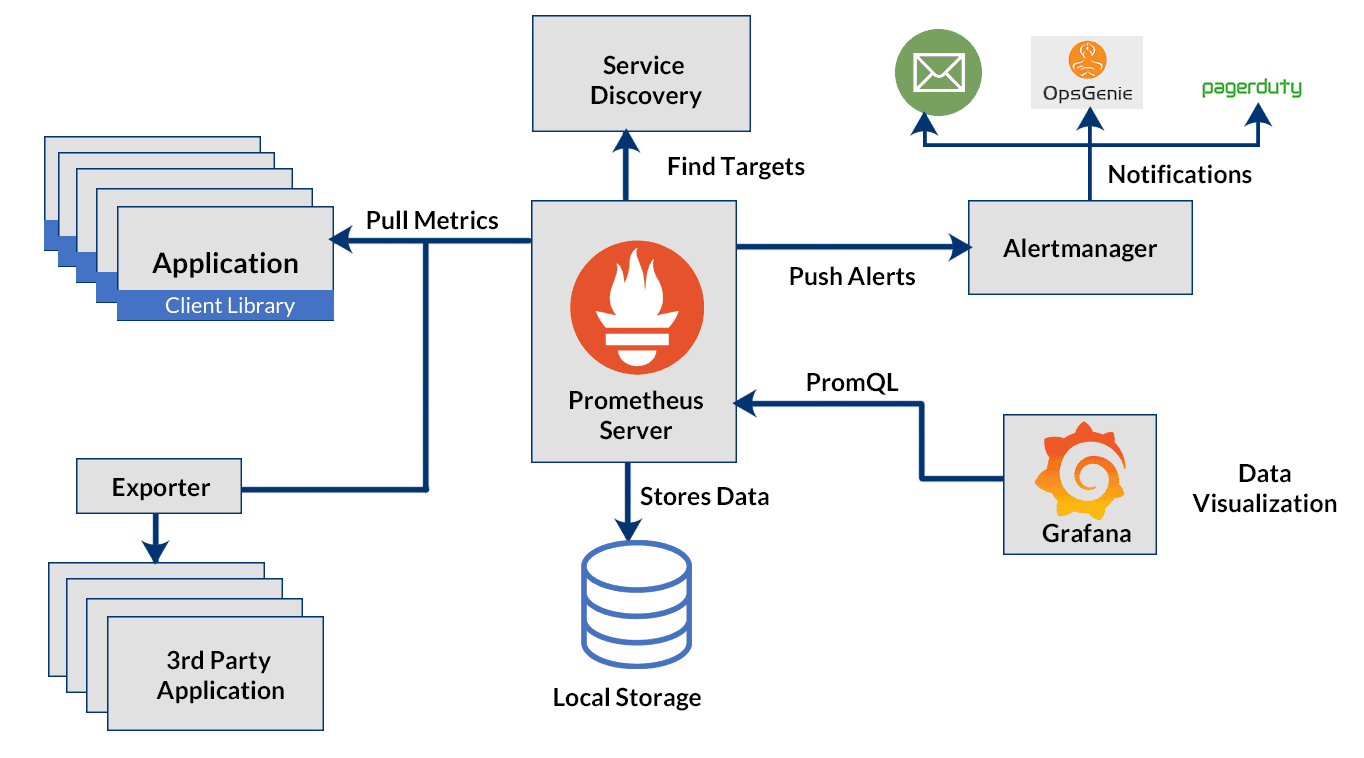
At its core, Prometheus has a main component called Prometheus Server, responsible for the actual monitoring work.



**Main Components of Prometheus Server:**

1. **Data Retrieval Worker** is responsible for all the data **pulling** activities from applications, services, servers etc. using Kubernetes API and pushing them into the database.
2. **Time Series Database** that stores all the metric data like current CPU usage, memory usage etc.
3. **HTTP Server API** meant to accept queries for the stored data. The Server API is used to display the data in a dashboard or a Web UI.

**About Data Retrieval Worker Component:**



**Pull Metrics:** Prometheus on Kubernetes pulls metrics data from the targets. They pull data from HTTP endpoints like the **host address/metrics**. So, for this to work, the targets should expose their **/metrics** endpoint, and the data at the endpoint must be acceptable by Prometheus.

**The most critical Metrics to Monitor**

1. CPU / Memory Requests vs Actual Usage
2. Percentage of Unavailable Pods Out of Desired Replicas
3. Nodes Failing Status Checks
4. Persistent Volume Utilization

**Metrics entries:**

**HELP** attributes description of what the metrics is?

**TYPE**: 3 types

1. **Counter**: How many times X happened. Eg: Number of exceptions the application had.
2. **Gauge**: Metric that can go Up and Down. Eg: What is the current value of CPU usage now
3. **Histogram**: How long something took or how big something was.

**Prometheus Exporter:**

* Not every service exposes their **/metrics** endpoint by defaut.
* An exporter is a software component that sits alongside an application and can accept the HTTP requests that Prometheus sends (to /metrics endpoint), ensure that the data is in a Prometheus-supported format, and return the relevant data to the central server.
* There are many official Prometheus exporters for services like **MySQL, Elastic Search, Linux server, cloud platforms**, etc. You can check it out from [here](https://prometheus.io/docs/instrumenting/exporters/)👈

**To Monitor your own application** for metrics like Number of requests, number of exceptions or server resources used, there are Prometheus client libraries <https://prometheus.io/docs/instrumenting/clientlibs/>. Using client libraries you can expose **/metrics** endpoint in your application and provide relevant information to be monitored.

**Service Discovery:** With exporters attached, each application can return data to Prometheus, but you still need to tell Prometheus where to find the data. Prometheus uses service discovery to **identify targets for scraping data**.

**Prometheus Data Storage**

Prometheus on Kubernetes collects and aggregates all the data in a disk (**custom time-series format**)

**Time-series format** is data collected at different points in time. These data points typically consist of successive measurements made from the same source over a time interval and are used to track change over time.

**Alerts Manager:** The Prometheus has another component that alerts us of a failure in the cluster, resource limitations etc. The Alert Manager is a Gateway and is responsible for firing alerts via different mediums like Slack, Email, OpsGenie, etc.

**Prometheus Querying:** Once the data gets stored in the database, we can query the data using **PromQL**. This query language lets you export data to a graphical interface like Grafana or send alerts using Alertmanager.

Prometheus Lab

**Install Prometheus using Helm:**

1. Start Minikube cluster with 2 cpus and 8GB memory

minikube start --cpus 2 --memory 8192

**Install using HELM**

* 1. Add repos

helm repo add stable https://charts.helm.sh/stable

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm repo update

* 1. Update Prometheus config file

mkdir Prometheus

cd Prometheus

**# Download and View the default configuration**

helm show values prometheus-community/kube-prometheus-stack > prometheus.yaml

**Create prometheus-values.yaml**

prometheus:

prometheusSpec:

scrapeInterval: "16s"

scrapeTimeout: "16s"

**Install Prometheus Stack Chart**

kubectl create namespace monitoring

helm install prometheus prometheus-community/kube-prometheus-stack -n monitoring **-f prometheus-values.yaml**

kubectl get pods -n monitoring

Note:  Helm chart deploys **node-exporter, kube-state-metrics, and alertmanager** along with Prometheus, so you will be able to start monitoring nodes and the cluster state **right away.**

1. **Open Prometheus-UI**

kubectl get services #Note the name of Prometheus Server Service and is running on Port 80.

kubectl port-forward **service/prometheus-kube-prometheus-prometheus** 9090 -n monitoring

Browse <http://localhost:9090>

**OR**

# Edit the Service and change **type** from **ClusterIP** to **LoadBalancer**.

kubectl **patch** service/prometheus-kube-prometheus-prometheus -n monitoring -p '{"spec": {"type": "LoadBalancer"}}'

minikube tunnel

kubectl get services -n monitoring

Browse: http://<ExternalIP>:9090 of Prometheus-server service

1. Status 🡪 Targets 🡪 Look at the **EndPoints**.
2. **Install and Open Grafana in browser:**

kubectl get pod

kubectl port-forward service/prometheus-grafana 80 -n monitoring

Login using username/password = admin/prom-operator

**Reference:**

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

Monitoring MongoDB

Timeline

Description automatically generated with medium confidence

**MongoDb.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: mongodb-deployment

  labels:

    app: mongodb

spec:

  replicas: 1

  selector:

    matchLabels:

      app: mongodb

  template:

    metadata:

      labels:

        app: mongodb

    spec:

      containers:

      - name: mongodb

        image: mongo

        ports:

        - containerPort: 27017

---

apiVersion: v1

kind: Service

metadata:

  name: mongodb-service

spec:

  selector:

    app: mongodb

  ports:

    - protocol: TCP

      port: 27017

      targetPort: 27017

**Deploy the MongoDb Application**

kubectl apply -f mongodb.yaml

**About MongoDb exporter**

**Look at all the official exporters of Prometheus:** [**https://prometheus.io/docs/instrumenting/exporters/**](https://prometheus.io/docs/instrumenting/exporters/)

The MongoDB Exporter collects and exports replicaset, server status, sharding and storage engine metrics.

1. **To view the list of all parameters for the chart and Install the Exporter Chart.**

helm show **values** prometheus-community/**prometheus-mongodb-exporter** > mongodb-values.yaml

1. Note the name of **mongodb service to be used in next step.**

kubectl get svc

1. **To change the configuration of MongoDB Exporter**

# Edit the **mongodb-values.yaml** if necessary. (Not required for this demo)

mongodb:

uri: "mongodb://**mongodb**-**service**:27017"

1. **Install the Helm chart**

helm **install** mongodb-exporter prometheus-community/**prometheus-mongodb-exporter** -f mongodb-values.yaml -n monitoring

1. **Test if the MongoDb Exporter is working or not**

kubectl get svc

kubectl port-forward service/mongodb-exporter-prometheus-mongodb-exporter 9216 -n monitoring

Browse <http://localhost:9216/metrics>

**View the logs of MongoDb Exporter**

kubectl logs mongodb-exporter-prometheus-mongodb-exporter-c9c4bf64-25bl9 -n monitoring

1. **Update Prometheus Config (mongo-values.yaml)**

prometheus:

prometheusSpec:

scrapeInterval: "16s"

**additionalScrapeConfigs**:

- job\_name: **'mongodb-job'**

static\_configs:

- targets: [**'mongodb-exporter-prometheus-mongodb-exporter.**monitoring.svc.cluster.local:9216']

**Note: 'mongodb-exporter-prometheus-mongodb-exporter' is the name of Service.**

1. **Update the Prometheus Chart.**

helm upgrade prometheus prometheus-community/kube-prometheus-stack -f mongo-values.yaml -n monitoring

1. **Visit Prometheus-UI**

kubectl get svc

kubectl port-forward service/prometheus-kube-prometheus-prometheus 9090 -n monitoring

Browse localhost:9090

**OR**

# Edit the Service and change **type** from **ClusterIP** to **LoadBalancer**.

kubectl patch svc prometheus-kube-prometheus-prometheus -n monitoring -p '{"spec": {"type": "LoadBalancer"}}' -n monitoring

minikube tunnel

kubectl get services -n monitoring

Browse: http://<ExternalIP> of Prometheus-server service

Notice that MongoDb exporter is added to targets without explicitly changing the configuration file of Prometheus.

1. **Grafana UI**

kubectl port-forward **service/prometheus-grafana** 80 -n monitoring

Browse <http://localhost:80>

OR

kubectl patch svc prometheus-kube-prometheus-prometheus -n monitoring -p '{"spec": {"type": "LoadBalancer"}}' -n monitoring

1. Login using following details

user: admin

pwd: prom-operator

1. Go to Dashboard 🡪 Click on **Kubernetes / Compute Resource / Pod**

filter pod = mongodb-deployment-XXXXXXXXXX

Note that MongoDb is now being monitored by Prometheus

MySql Exporter

1. **MySQL Deployment and Service**

apiVersion: v1

kind: Service

metadata:

name: mysql

spec:

ports:

- port: 3306

selector:

app: mysql

type: ClusterIP

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: mysql-deployment

spec:

selector:

matchLabels:

app: mysql

strategy:

type: Recreate

template:

metadata:

labels:

app: mysql

spec:

containers:

- image: mysql:5.6

name: mysql

env:

# Use secret in real usage

- name: MYSQL\_ROOT\_PASSWORD

value: password

ports:

- containerPort: 3306

name: mysql

resources:

limits:

cpu: 1

memory: 1Gi

requests:

cpu: 100m

memory: 100Mi

1. helm show values prometheus-community/prometheus-mysql-exporter > mysql-values.yaml -n monitoring
2. **mysql-values.yaml**

mysql:

db: ""

host: "mysql"

# config my.cnf https://dev.mysql.com/doc/c-api/8.0/en/mysql-options.html

additionalConfig:

# - connect-timeout=5

# - debug

pass: "**password**"

port: 3306

protocol: ""

user: "root"

# secret with full config my.cnf

existingConfigSecret:

name: ""

key: ""

# secret only containing the password

existingPasswordSecret:

name: ""

key: ""

1. **prometheus–values.yaml**

prometheus:

prometheusSpec:

scrapeInterval: "17s" # Replace "30s" with your desired interval

additionalScrapeConfigs:

**. . .**

**- job\_name: 'mysql-job'**

**static\_configs:**

**- targets: ['mysql-exporter-prometheus-mysql-exporter.default.svc.cluster.local:9104']**

1. helm install mysql-exporter prometheus-community/prometheus-mysql-exporter -f mysql-values.yaml -n monitoring
2. helm upgrade prometheus prometheus-community/kube-prometheus-stack -f prometheus-values.yaml -n monitoring
3. kubectl port-forward service/prometheus-kube-prometheus-prometheus 9090 -n monitoring

or

kubectl patch svc prometheus-kube-prometheus-prometheus -n monitoring -p '{"spec": {"type": "LoadBalancer"}}' -n monitoring

1. kubectl port-forward service/prometheus-grafana 80 -n monitoring