KUBERNETES

==========

- **Menions:** This is an individual node used in Kubernetes Combination of these minions is called as Kubernetes cluster

- **Master** is the main machine which triggers the container orchestraion It distributes the work load to the Slaves

- **Slaves** are the nodes that accept the work load from the master and handle activites load balancing,autoscalling,high availability etc

❖ **Kubernetes uses various of types of Object**

1 Pod: This is a layer of abstraction on top of a container.This is the samallest object that kubernetes can work on.In the Pod we have a container.

➔ The advantage of using a Pod is that kubectl commands will work on the Pod and the Pod communicates these instructions to the container.In this way we can use the same kubectl irresepective of which technology containers are in the Pod.

2 Service: This is used for port mapping and network load balancing

3 NameSpace: This is used for creating partitions in the cluster.Pods running in a namespace cannot communicate with other pods running in other namespace

4 Secrets: This is used for passing encrypted data to the Pods

5 ReplicationController: This is used for managing multiple replicas of PODs and also perfroming sacling

6 ReplicaSet: This is similar to replicationcontroller but it is more advanced where features like selector can be implemented

7 Deployment: This used for perfroming all activites that a Replicaset can do it can also handle rolling update

8 PersistantVolume: Used to specify the section of storage that should be used for volumes

9 PersistantVolumeClaims: Used to reserver a certain amout of storage for a pod from the persistant volume.

10 Statefulsets: These are used to handle stateful application like data bases where consistency in read write operations has to be maintained.

11 HorrizontalPodAutScaller: Used for auto scalling of pods depending on the load

Kubernetes Architecture

Master Componentes

Container runtime: This can be docker or anyother container technology

apiServer: Users interact with the apiServer using some clinet like ui,command line tool like kubelet.It is the apiServer which is the gateway to the cluster

- ✓ It works as a gatekeeper for authentication and it validates if a specific user is having permissions to execute a specific command.Example if we want to deploy a pod or a deployment first apiServers validates if the user is authorised to perform that action and if so it passes to the next process

ie the "Scheduler"

Scheduler: This process accepts the instructions from apiServer after validation and starts an application on a sepcific node or set of nodes.

- ✓ It estimates how much amount of h/w is required for an application and then checks which slave have the necessary h/w resources and instructs the kubelet to deploy the application

kubelet: This is the actual process that takes the orders from scheduler and deploy an application on a slave.This kubelet is present on both master and slave

controller manager: This check if the desired state of the cluster is always maintained.If a pod dies it recreates that pod to maintain the desired state

etcd: Here the cluster state is maintained in key value pairs.

- ✓ It maintains info about the slaves and the h/w resources available on the slaves and also the pods running on the slaves
- ✓ The scheduler and the control manager read the info from this etcd and schedule the pods and maintain the desired state

Worker components

containerrun time: Docker or some other container technology

kubelet: This process interacts with container run time and the node and it start a pod with a container in it

kubeproxy: This will take the request from services to pod

- It has the intellegence to forward a request to a near by pod.Eg If an application pod wants to communicate with a db pod then kubeproxy will take that request to the nearby pod

Kubernetes can be installed in the following ways Unmanaged K8s setup

1 Kops

2 Kubeadm

3 Kind

Managed K8s setup

1 EKS (AWS)

2 GKE (GCP)

- **KOPS stands for Kubernetes Operations and it is used for setting up Kubernetes on cloud in an automated manner**

Kubernetes on AWS using Kops

1. Launch Linux EC2 instance in AWS (Kubernetes Client)
2. Create and attach IAM role to EC2 Instance.
Kops need permissions to access
S3
EC2
VPC
Route53
Autoscaling
etc..

3. Install Kops on EC2

```
curl -LO https://github.com/kubernetes/kops/releases/download/$(curl -s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag_name | cut -d '"' -f 4)/kops-linux-amd64
chmod +x kops-linux-amd64
sudo mv kops-linux-amd64 /usr/local/bin/kops
```

4. Install kubectl

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl
```

5. Create S3 bucket in AWS
S3 bucket is used by kubernetes to persist cluster state, lets create s3 bucket using aws **cli Note:** Make sure you choose bucket name that is uniqe accross all aws accounts

```
aws s3 mb s3://project.in.k8s --region us-west-2
```

6. Create private hosted zone in AWS Route53
Head over to aws Route53 and create hostedzone
Choose name for example (sai.in)
Choose type as privated hosted zone for VPC
Select default vpc in the region you are setting up your cluster Hit create

7 Configure environment variables.
Open .bashrc file

```
vi ~/.bashrc
```

Add following content into .bashrc, you can choose any arbitary name for cluster and make sure buck name matches the one you created in previous step.

```
export KOPS_CLUSTER_NAME=project.in
export KOPS_STATE_STORE=s3://project.in.k8s
```

Then running command to reflect variables added to .bashrc

```
source ~/.bashrc
```

8. Create ssh key pair

This keypair is used for ssh into kubernetes cluster

```
ssh-keygen
```

9. Create a Kubernetes cluster definition.

```
kops create cluster \
--state=${KOPS_STATE_STORE} \
--node-count=2 \
--master-size=t3.medium \
--node-size=t3.medium \
--zones=us-west-2a \
--name=${KOPS_CLUSTER_NAME} \
--dns private \
--master-count 1
```

10. Create kubernetes cluster

```
kops update cluster --yes --admin
```

Above command may take some time to create the required infrastructure resources on AWS. Execute the validate command to check its status and wait until the cluster becomes ready

11 To check if the cluster is ready

```
kops validate cluster
```

- ✓ For the above above command, you might see validation failed error initially when you create cluster and it is expected behaviour, you have to wait for some more time and check again.

KIND :Kubernetes in Docker

Here the master and slave machines are docker containers

==========

1 Create a AWS ubuntu instance and install docker on it

2 Install Kubectl

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin/kubectl
```

3 Install KIND

```
[ $(uname -m) = x86_64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.20.0/kind-linux-amd64
```

```
chmod +x ./kind

sudo mv ./kind /usr/local/bin/kind
```

4 Create a kind config file

```
  vim config.yml
  # three node (two workers) cluster config
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: worker
- role: worker
```

5 Create the cluster with the above file

```
  kind create cluster --name mycluster --config=config.yml
```

6 Cluster will be created as docker containers

```
  sudo docker container ls
```

7 Go into the master container to run the kubernetes commands

```
  sudo docker exec -it master_container_id bash
```

8 Run any kubernetes command

```
  kubectl get nodes
```

Kubernetes on kubeadm

Kubeadm installation-This is a manaul setup fo Kuberentes and it works on both cloud and on premise

Install, start and enable docker service

```
yum install -y -q yum-utils device-mapper-persistent-data lvm2 > /dev/null 2>&1
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-
ce.repo > /dev/null 2>&1
yum install -y -q docker-ce >/dev/null 2>&1


systemctl start docker
systemctl enable docker
```

```
================================================================================
Disable SELINUX

setenforce 0
sed -i --follow-symlinks 's/^SELINUX=enforcing/SELINUX=disabled/'
/etc/sysconfig/selinux

================================================================================
Disable SWAP

sed -i '/swap/d' /etc/fstab
swapoff -a

================================================================================
Update sysctl settings for Kubernetes networking
```

```
cat >>/etc/sysctl.d/kubernetes.conf<<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sysctl --system

================================================================================
Add Kubernetes to yum repository

cat >>/etc/yum.repos.d/kubernetes.repo<<EOF
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
        https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF

================================================================================
Install Kubernetes
yum install -y kubeadm-1.19.1 kubelet-1.19.1 kubectl-1.19.1

================================================================================
Enable and start Kubernetes service

systemctl start kubelet
systemctl enable kubelet
================================================================================
Repeat the above steps on Master and slaves
================================================================================
```

On Master

```
===========
Initilise the Kubernetes cluster
---------------------------------------

kubeadm init --apiserver-advertise-address=ip_of_master  --pod-network-
cidr=192.168.0.0/16

================================================================================
```

➔ To be able to use kubectl command to connect and interact with the cluster, the user needs kube config file.

```
mkdir /home/ec2-user/.kube
cp /etc/kubernetes/admin.conf /home/ec2-user/.kube/config
chown -R ec2-user:ec2-user /home/ec2-user/.kube

================================================================================
Deploy calico network
kubectl apply -f https://docs.projectcalico.org/v3.9/manifests/calico.yaml

================================================================================
For slaves to join the cluster
kubeadm token create --print-join-command
```

Managed Kubernetes Installtion

EKS (Elastic Kubernetes Service)

==========

1 Create an ubuntu instance on AWS and name it EKS server

2 Create an IAM with admin roles and assign to the EKS server

3 Install Kubectl

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-
release/release/stable.txt)/bin/linux/amd64/kubectl

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin/kubectl
```

4 Install eksctl

Download the eksctl

```
curl --silent --location
"https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -
s)_amd64.tar.gz" | tar xz -C /tmp
```

Give execute permissions on it

```
sudo mv /tmp/eksctl /usr/local/bin
```

Check if it is instlled

```
eksctl version
```

5 To create cluster on EKS

```
eksctl create cluster \
  --region us-east-1 \
--node-type t3.medium \
--nodes 3 \
--name mynew-cluster
```

Kubernetes Setup on GCP using GKE

==========

1 Login into GCP console

2 Click on Navigation menu

3 Click on Kubernetes Engine

4 Click on Create cluster

5 Select Swithc to Standard cluster

6 Click on Create

```
1 To see the list of nodes in the Kubernetes cluster
  kubectl get nodes

2 To get info about the nodes along with ipaddress and docker version etc
  kubectl get nodes -o wide

3 To get detailed info about the nodes
  kubectl describe nodes node_name

================================================================================
Create nginx as a pod and name it webserver
kubectl run --image nginx webserver

To see the list of pods
kubectl get pods

To get  info about the pods along with ipaddress
kubectl get pods -o wide

To get detailed info about the pods
kubeclt describe pods webserver

================================================================================
Create a mysql pod and also pass the necessary environment variables
kubectl run --image mysql:5 db --env MYSQL_ROOT_PASSWORD=intelliqit

Check if the pod is running
kubectl get pods

To delete the mysql pod
kubectl delete pods db

================================================================================
Kubernetes objects are created using definition/manifest files
These files containe mainly four components
---
apiVersion:
kind:
metadata:
spec:
...
```

kind	**:**	**apiversion**
Pod		v1
Service		v1
Namespace		v1
Secret		v1
ReplicationController		v1
ReplicaSet		apps/v1
Deployment		apps/v1
StatefulSet		apps/v1
DaemonSet		apps/v1
PersistantVolume		v1
PersistantVolumeClaim		v1
HorrizontalPodAutoscaller		v1

Create a pod definition file to create an nginx pod

```
1 vim pod-definition1.yml
--
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  namespace: test-ns
  labels:
    author: intelliqit
    type: proxy
    cat: rat
spec:
  containers:
    - name: mynginx
      image: nginx
...

2 To create pods from the above file
  kubectl apply -f pod-defintion2.yml

3 To see the list of pods
  kubectl get pods

4 To delete the pods created from theabove file
  kubectl delete -f pod-defintion1.yml
```

Create a pod definition file to setup a postgres pod

```
1 vim pod-definition.yml

---
apiVersion: v1
kind: Pod
metadata:
  name: postgres-pod
  labels:
    type: db
    author: intelliqit
spec:
  containers:
    - name: mydb
      image: postgres
      env:
        - name: POSTGRES_PASSWORD
          value: intelliqit
        - name: POSTGRES_DB
          value: mydb
        - name: POSTGRES_USER
          value: myuser
```

Create a pod definition file to create a jenkins pod

```
1 vim pod-definition3.yml
---
apiVersion: v1
kind: Pod
metadata:
  name: jenkins-pod
  labels:
    type: ci-cd
    author: intelliqit
spec:
  containers:
    - name: myjenkins
      image: jenkins/jenkins
      ports:
        - containerPort: 8080
          hostPort: 8080
```

Create a httpd pod using a definition file

```
vim pod-definition4.yml
---
apiVersion: v1
kind: Pod
metadata:
  name: httpd-pod
  labels:
    type: webserver
    author: intelliqit
spec:
  containers:
    - name: myhttpd
      image: httpd
      ports:
        - containerPort: 80
          hostPort: 8080
...
```

Namespace: These are logical partitions in the Kubernetes cluster

Create a definition file to create a namespace

```
vim namespace.yml
---
apiVersion: v1
kind: Namespace
metadata:
  name: test-ns
...

To create a namespace from the above file
kubectl apply -f namespace.yml

To see the list of all the namespaces
kubectl get namespace
```

Create a definitition file to create wordpress and launch it on the above namespace

```
vim pod-definition5.yml
---
apiVersion: v1
kind: Pod
metadata:
  name: wordpress-pod
  namespace: test-ns
  labels:
    type: CMS
    author: intelliqit
spec:
  containers:
    - name: mywordpress
      image: wordpress
      ports:
        - containerPort: 80
          hostPort: 8080
...

To create pods from the above file
kubectl apply -f pod-definition5.yml

To check if the pod is created on the above namespace
kubectl get pods -n test-ns
```

ReplicationController

Create a replication controller file to setup httpd with multiple replicas

```
vim replication-controller.yml

---
apiVersion: v1
kind: ReplicationController
metadata:
  name: httpd-rc
  labels:
    type: websrver
    author: intelliqit
spec:
  replicas: 3
  template:
    metadata:
      name: httpd-pod
      labels:
        type: webserver
    spec:
      containers:
        - name: myhttpd
          image: httpd
          ports:
            - containerPort: 80
              hostPort: 8080
...
```

To create replication controller from the above file
kubectl apply -f replication-controller.yml

To see the list of replication controllers
kubectl get rc

To see the pods
kubectl get pods

ReplicaSet

Create a replicaset to setup multiple replicas of tomcat

```
vim replicas-set.yml
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: tomcat-rs
  labels:
    type: appserver
    author: intelliqit
spec:
  replicas: 3
  selector:
    matchLabels:
      type: appserver
  template:
    metadata:
      name: tomcat-pod
      labels:
        type: appserver
    spec:
      containers:
        - name: mytomcat
          image: tomee
          ports:
            - containerPort: 8080
              hostPort: 9090
...
```

To create a replicaset from the above file
kubectl apply -f replica-set.yml

To see the list of replicasets
kubectl get rs

To scale the replicas set we can change the no of replicas in the definition file and
kubectl replace -f replicas-set.yml

Another way of scallinf directly from command prompt is
kubectl scale --replicas 1 -f replica-set.yml

Deployment

Create a deployment definition file for nginx

```
vim deployment1.yml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    type: proxy
    author: intelliqit
spec:
  replicas: 3
  selector:
    matchLabels:
      type: proxy
  template:
    metadata:
      name: nginx-pod
      labels:
        type: proxy
    spec:
      containers:
        - name: mynginx
          image: nginx
          ports:
            - containerPort: 80
              hostPort: 9090

To create a deployment from the above file
kubectl apply -f deployment1.yml

To see the list of deployments
kubectl get deployment

To delete the deployments
kubectl delete -f deployment1.yml
```

Create a deployment definition file to setup mysql

```
vim deployment2.yml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
  labels:
    type: db
spec:
  replicas: 2
  selector:
    matchLabels:
      type: db
  template:
    metadata:
      name: mysql-pod
      labels:
```

```
        type: db
    spec:
      containers:
        - name: mydb
          image: mysql:5
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: intelliqit
...
```

DaemonSet

DaemonSet: This is to run pods on every salve and only one slave per node.

Create a daemonset file to create nginx

```
vim daemonset.yml
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx-daemon
  labels:
    type: proxy
spec:
  selector:
    matchLabels:
      type: proxy
  template:
    metadata:
      name: nginx-pod
      labels:
        type: proxy
    spec:
     containers:
       - name: mynginx
         image: nginx
         ports:
           - containerPort: 80
             hostPort: 8080
...

To create a daemonset from this file
kubectl apply -f daemonset.yml

To see the list of all the objects running in the cluster
kubect get all

================================================================================
=======
Service Objects
======================
1 NodePort:This is used to perform network load balancing

2 LoadBalancer: This will create an ip for the entire cluster and it works only on
managed kubernetes service

3 Clusterip: This is used to fro pods to communicate with other pods in the clsuter
but not with outside world
```

Create a service definition file fro node port object and apply it on pod - definition1.yml

```
vim service1.yml
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  labels:
    author: intelliqit
spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
  selector:
    type: proxy

Create the pod for nginx using the definition file
kubectl apply -f pod-definition1.yml

Create service from the above file
kubectl apply -f service1.yml

To see the list of all object
kubectl get all
```

Now we can access nginx from any machines ip address

Create a service object of the type loadbalancer and apply it on pod - definition3.yml

```
vim service2.yml
---
apiVersion: v1
kind: Service
metadata:
  name: jenkins-service
  labels:
    author: intelliqit
spec:
  type: LoadBalancer
  ports:
    - targetPort: 8080
      port: 8080
      nodePort: 30009
  selector:
    type: ci-cd
    author: intelliqit
...

Create a jenkins pod
kubectl apply -f pod-definition3.yml

Create a loadbalancer service from the above file
kubectl apply -f service2.yml

This will generate a unique public ip for the entire cluster
kubectl get svc
```

Create a service object of the type clusterip and apply it on pod-definition2.yml

```
vim service3.yml
---
apiVersion: v1
kind: Service
metadata:
  name: postgres-service
  labels:
    author: intelliqit
spec:
  ports:
    - targetPort: 5432
      port: 5432
  selector:
    type: db
    author: intelliqit

Create postgres pod
kubectl apply -f pod-definition2.yml

Create service of clusterip type
kubectl apply -f service3.yml

To check if the service is created
kubectl get svc
```

Kompose

This is used to conver t a docker compose file to Kubernetes definition files

Install Kompose into the Kubernetes cluster

https://www.digitalocean.com/community/tutorials/how-to-migrate-a-docker-compose-workflow-to-kubernetes

Create a docker compose file

```
vim docker-compose.yml
---
version: '3'
services:
  db:
    image: mysql:5
    environment:
      MYSQL_ROOT_PASSWORD: intelliqit
    deploy:
      replicas: 2

  wordpress:
    image: wordpress
    ports:
      - 8888:80
    deploy:
      replicas: 3
...

To create Kubernetes defintion files from the above file
kompose convert
```

Kubernetes Project

- ➔ This is a voting app created using python,this app is exposed to the customers and they can cast their vote
- ➔ This info will be registered in a in memory db(temporary db) that we setup using redis From here we have a .net application that filers the data and stores it permenantly in a postgres db and the results can be viewed on an app created using nodejs

Create 5 deployment definition files for all the above object and 4 service definition file

```
vim voting-app-deployment.yml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: voting-app-deployment
  labels:
    name: voting-app
    author: intelliqit
spec:
  replicas: 2
  selector:
    matchLabels:
      name: voting-app
  template:
    metadata:
      name: voting-app-pod
      labels:
        name: voting-app
    spec:
      containers:
        - name: voting-app
          image: dockersamples/examplevotingapp_vote
...

vim result-app-deployment.yml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: result-app-deployment
  labels:
    name: result-app
    author: intelliqit
spec:
  replicas: 2
  selector:
    matchLabels:
      name: result-app
  template:
    metadata:
      name: result-app-pod
      labels:
        name: result-app
    spec:
      containers:
```

```
        - name: result-app
          image: dockersamples/examplevotingapp_result
...

vim redis-app-deployment.yml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-app-deployment
  labels:
    name: redis-app
    author: intelliqit
spec:
  selector:
    matchLabels:
      name: redis-app
  template:
    metadata:
      name: redis-app-pod
      labels:
        name: redis-app
    spec:
      containers:
        - name: redis-app
          image: redis
...

vim postgres-app-deployment.yml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-app-deployment
  labels:
    name: postgres-app
    author: intelliqit
spec:
  selector:
    matchLabels:
      name: postgres-app
  template:
    metadata:
      name: postgres-app-pod
      labels:
        name: postgres-app
    spec:
      containers:
        - name: postgres-app
          image: postgres
          env:
            - name: POSTGRES_PASSWORD
              value: intelliiqt
            - name: POSTGRES_USER
              value: myuser
            - name: POSTGRES_DB
              value: mydb
...
```

```
vim worker-app-deployment.yml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: worker-app-deployment
  labels:
    name: worker-app
    author: intelliqit
spec:
  selector:
    matchLabels:
      name: worker-app
  template:
    metadata:
      name: worker-app-pod
      labels:
        name: worker-app
    spec:
      containers:
        - name: worker-app
          image: dockersamples/examplevotingapp_worker
...

vim voting-app-service.yml
---
apiVersion: v1
kind: Service
metadata:
  name: voting-app-service
  labels:
    author: intelliqit
spec:
  type: LoadBalancer
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
  selector:
    name: voting-app

vim result-app-service.yml
---
apiVersion: v1
kind: Service
metadata:
  name: result-app-service
  labels:
    author: intelliqit
spec:
  type: LoadBalancer
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30009
  selector:
   name: result-app
```

```
vim redis-app-service.yml
---
apiVersion: v1
kind: Service
metadata:
  name: redis-app-service
  labels:
    author: intelliqit
spec:
  ports:
    - targetPort: 6379
      port: 6379
  selector:
   name: redis-app
...

vim postgres-app-service.yml
---
apiVersion: v1
kind: Service
metadata:
  name: postgres-app-service
  labels:
    author: intelliqit
spec:
  ports:
    - targetPort: 5432
      port: 5432
  selector:
    name: postgres-app
...

kubectl apply -f voting-app-deployment.yml

kubectl apply -f voting-app-service.yml

kubectl apply -f result-app-deployment.yml

kubectl apply -f result-app-service.yml

kubectl apply -f redis-app-deployment.yml

kubectl apply -f redis-app-service.yml

kubectl apply -f postgres-app-deployment.yml

kubectl apply -f postgres-app-service.yml

kubectl apply -f worker-app-deployment.yml
```

KUBERNETES

PROMETHEUS

Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud. Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community. Let's dive into its key features and architecture:

1. **Features**:
   - **Multi-dimensional Data Model**: Prometheus uses a time series data model with metrics identified by metric names and key-value pairs (labels).
   - **PromQL**: A flexible query language that leverages this dimensionality for querying and analysis.
   - **No Reliance on Distributed Storage**: Prometheus operates with single server nodes, making them autonomous.
   - **Pull Model for Time Series Collection**: Metrics are collected via HTTP pulls.
   - **Service Discovery or Static Configuration**: Targets (services) are discovered dynamically or configured statically.
   - **Graphing and Dashboarding Support**: Multiple modes for visualizing data.

2. **What Are Metrics?**:
   - Metrics are numerical measurements that record changes over time.
   - For example, a web server might measure request times, while a database could track active connections or queries.
   - Metrics play a crucial role in understanding application behavior and performance.

3. **Components**:
   - The Prometheus ecosystem includes:
     - The main Prometheus server, responsible for scraping and storing time series data.
     - Client libraries for instrumenting application code.
     - A push gateway for short-lived jobs.
     - Special-purpose exporters for services like HAProxy, StatsD, and Graphite.
     - An alertmanager to handle alerts.

4. **Architecture**:
   - Prometheus scrapes metrics from instrumented jobs directly or via an intermediary push gateway.
   - It stores all scraped samples locally and runs rules to aggregate data or generate alerts.

GRAFANA

Grafana is an **open-source platform** for visualizing and analyzing data. It's used for a wide range of purposes, including **performance analysis**, **business intelligence**, and **DevOps dmonitoring**. Organizations of all sizes, from small startups to large enterprises, utilize Grafana to gain insights into their data and make informed decisions[1][2].

Here are some key points about Grafana:

- **Visualization and Analysis**: Grafana allows users to see their data via **charts and graphs** that are unified into one dashboard (or multiple dashboards!) for easier interpretation and understanding[2].
- **Data Integration**: It provides **integrated support** for over **15 popular databases** and monitoring solutions, making it a versatile choice for data analytics[3].