Terraform

IaC – Infrastructure as Code

Create S3 bucket on aws

Api-> make a call

VPC + VPC config+EC2+S3

(CFT) Cloud formation template -> in json and yaml

AWS -> cloud formation template

Azure -> Resource manager

Openstack -> Heat template

Automate infrastructure

Terraform -> API as code ->

Install terraform -> [Install Terraform | Terraform | HashiCorp Developer](https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli)

Like virtual machine -> code space -> [GitHub - iam-veeramalla/terraform-zero-to-hero: Master Terraform in 7 days using this Zero to Hero course.](https://github.com/iam-veeramalla/terraform-zero-to-hero)

Click on code -> codespaces -> Add(+button) -> Not Successful

Cmd

terraform –version

aws cli install and check

if necessary create an Iam role

aws configure

enter access key

secret access key

region us-east-1

aws s3 ls

main.tf

provider "aws" {

region = "us-east-1" # Set your desired AWS region

}

resource "aws\_instance" "example" {

ami = "ami-0c55b159cbfafe1f0" # Specify an appropriate AMI ID

instance\_type = "t2.micro"

}

Terraform template : [aws\_instance | Resources | hashicorp/aws | Terraform | Terraform Registry](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance)

terraform plan -> dry run

terraform apply

In visual studio -> Install plugin hashicorp terraform -> HashiCorp HCL

Terraform destroy

Providers -> EC2 (AWS)

Main.tf -> Provider”aws”{

Region = “-”}

Resource “AWS” ={}

Terraform in 1 video

1. **Provider**: A provider is a plugin for Terraform that defines and manages resources for a specific cloud or infrastructure platform. Examples of providers include AWS, Azure, Google Cloud, and many others. You configure providers in your Terraform code to interact with the desired infrastructure platform.
2. **Resource**: A resource is a specific infrastructure component that you want to create and manage using Terraform. Resources can include virtual machines, databases, storage buckets, network components, and more. Each resource has a type and configuration parameters that you define in your Terraform code.
3. **Module**: A module is a reusable and encapsulated unit of Terraform code. Modules allow you to package infrastructure configurations, making it easier to maintain, share, and reuse them across different parts of your infrastructure. Modules can be your own creations or come from the Terraform Registry, which hosts community-contributed modules.
4. **Configuration File**: Terraform uses configuration files (often with a .tf extension) to define the desired infrastructure state. These files specify providers, resources, variables, and other settings. The primary configuration file is usually named main.tf, but you can use multiple configuration files as well.
5. **Variable**: Variables in Terraform are placeholders for values that can be passed into your configurations. They make your code more flexible and reusable by allowing you to define values outside of your code and pass them in when you apply the Terraform configuration.
6. **Output**: Outputs are values generated by Terraform after the infrastructure has been created or updated. Outputs are typically used to display information or provide values to other parts of your infrastructure stack.
7. **State File**: Terraform maintains a state file (often named terraform.tfstate) that keeps track of the current state of your infrastructure. This file is crucial for Terraform to understand what resources have been created and what changes need to be made during updates.
8. **Plan**: A Terraform plan is a preview of changes that Terraform will make to your infrastructure. When you run terraform plan, Terraform analyzes your configuration and current state, then generates a plan detailing what actions it will take during the apply step.
9. **Apply**: The terraform apply command is used to execute the changes specified in the plan. It creates, updates, or destroys resources based on the Terraform configuration.
10. **Workspace**: Workspaces in Terraform are a way to manage multiple environments (e.g., development, staging, production) with separate configurations and state files. Workspaces help keep infrastructure configurations isolated and organized.
11. **Remote Backend**: A remote backend is a storage location for your Terraform state files that is not stored locally. Popular choices for remote backends include Amazon S3, Azure Blob Storage, or HashiCorp Terraform Cloud. Remote backends enhance collaboration and provide better security and reliability for your state files.

**Setup Terraform for AWS**

1. **Install AWS CLI (Command Line Interface)**:
2. **Create an AWS IAM User**:
3. **Configure AWS CLI Credentials**:
4. **Multiple Providers**

You can use multiple providers in one single terraform project. For example,

1. Create a providers.tf file in the root directory of your Terraform project.
2. In the providers.tf file, define the AWS and Azure providers. For example:

provider "aws" {

region = "us-east-1"

}

provider "azurerm" {

subscription\_id = "your-azure-subscription-id"

client\_id = "your-azure-client-id"

client\_secret = "your-azure-client-secret"

tenant\_id = "your-azure-tenant-id"

}

**Modules**

1. **Modularity**: Terraform modules allow you to break down your infrastructure configuration into smaller, self-contained components. This modularity makes it easier to manage and reason about your infrastructure because each module handles a specific piece of functionality, such as an EC2 instance, a database, or a network configuration.

**Terraform destroy**

Terraform plan

Terraform Registry -> like docker hub registry

[Terraform Registry](https://registry.terraform.io/) -> ec2 instance -> module -> examples -> complete

**Terraform State File**

Terraform is an Infrastructure as Code (IaC) tool used to define and provision infrastructure resources. The Terraform state file is a crucial component of Terraform that helps it keep track of the resources it manages and their current state. This file, often named terraform.tfstate, is a JSON or HCL (HashiCorp Configuration Language) formatted file that contains important information about the infrastructure's current state, such as resource attributes, dependencies, and metadata.

**Advantages of Terraform State File:**

**Resource Tracking:** The state file keeps track of all the resources managed by Terraform, including their attributes and dependencies. This ensures that Terraform can accurately update or destroy resources when necessary.

**Concurrency Control:** Terraform uses the state file to lock resources, preventing multiple users or processes from modifying the same resource simultaneously. This helps avoid conflicts and ensures data consistency.

**Plan Calculation:** Terraform uses the state file to calculate and display the difference between the desired configuration (defined in your Terraform code) and the current infrastructure state. This helps you understand what changes Terraform will make before applying them.

**Resource Metadata:** The state file stores metadata about each resource, such as unique identifiers, which is crucial for managing resources and understanding their relationships.

Statefile -> record, Store

Ec2 -> t2.micro

When you type terraform apply or terraform plan ->

Statefile is heart of terraform -> without state file you will not know to update infrastructure rather than creating it.

Terraform destroy -> will destroy infrastructure

Disadvantage/ drawback ->

1. Github access is compromised statefile is compromised.
2. Everytime change in code updated there should be change in statefile.

In vcs(version control system) -> you have to upload statefile

Lets say a developer takes access of the statefile . Puts a tag on it ec2 and this is pushed back to Github (tag). If the statefile is not pushed to Github. Terraform will not know the logic of tag creation.

If there is difference in terraform file it will tell to delete files.

Note: Everytime a change is made to the code. statefile gets updated. This statefile should be uploaded to terraform after applying.

Remote backend

If you don’t want to store statefile in git you can store it in S3 bucket.

In this statefile won’t get created on Laptop or VM. It will get creted in S3 bucket. This will remove the 2 drawback of terraform.

In this condition when you run terraform apply automatically the state file is updated.

If it gets updated in local. It gets updated in Ec2 instance.

When you run terraform init. It will understand that we are using the logic of remote backend. It will compare between the S3 bucket and the Github repository.

S3 -> remote backend

You can host your terraform remote backend In the terraform cloud.

Azure -> Azure storage

Devops team -> github repository with terraform code. To create EKS or VPC 3 tier architecture -> using remote backend S3 -> Any one will clone on laptop -> when you do terraform apply will apply the logic to S3 bucket -> SO now statefile is updated in S3 bucket

Github

Devops



EC2

Ansible

Ansible is an open source IT automation engine that automates

* provisioning
* configuration management
* application deployment
* orchestration

and many other IT processes. It is free to use, and the project benefits from the experience and intelligence of its thousands of contributors.

**How Ansible works**

Ansible is agentless in nature, which means you don't need install any software on the manage nodes.

For automating Linux and Windows, Ansible connects to managed nodes and pushes out small programs—called Ansible modules—to them. These programs are written to be resource models of the desired state of the system. Ansible then executes these modules (over SSH by default), and removes them when finished. These modules are designed to be idempotent when possible, so that they only make changes to a system when necessary.

For automating network devices and other IT appliances where modules cannot be executed, Ansible runs on the control node. Since Ansible is agentless, it can still communicate with devices without requiring an application or service to be installed on the managed node.

Ansible uses python internally -> yaml file written gets converted to python ->

System

Admins

Configuration



Management

Linux Windows

1. Up to date (Supported)
2. OpenSSh/wget/curlshared -> check packages version -> up to date
3. Java
4. Maintenance

Things to check if are up to date

1. OS
2. System
3. Application
4. Servers

Similar tools -> Puppet, Chef, Salt, Ansible

Puppet, chef, salt are written in ruby

They just need to write one programming that is puppet or chef. It will automate all the things -> Example : Linux, Alpine, Windows, Debian Gnux etc

* Drawback -> have to learn ruby -> complesx way of writing code.
* Have to install software like agent on each device or VM
* New machines created have to be connected to the master.

We write Chef books or cookbooks ->

Ansible playbook

Need to write program in yaml

It is agentless so there is no need to install any software on the target ->

Control Node -> On control nodes Ansible is installed -> Generally VMs are used with this virtual machine you can control configuration of the rest of the machines ->

Manage nodes ->

Installing Ansible -> [Installing Ansible — Ansible Community Documentation](https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html)

First extension to install in visual studio -> YAml, Ansible by red hat

Terraform workspace

EC2 + S3

Jira

dev env -> main.tf

terraform .tfvars

stage ->

prod -> main.tf

main.tf

dev.tfvars

Stage.tfvars

Prod.tfvars

Statefile

So in this condition you have to create a state file for each and every stage.

So workspace will create state file per environment.

Terafform -> dev -> statefile EcfewddcEC2 + S3

Stage -> statefile

Prod -> statefile

Terraform apply -var-file=stage.tfvars

Terraform workspace select dev

Terraform workspace show

Terraform select stage

Hashicorp vault ->

Use information related to Kubernetes

Storing SSH related information

Take a linux instance install hashi corp

: [terraform-zero-to-hero/Day-7/02-vault-integration.md at main · iam-veeramalla/terraform-zero-to-hero · GitHub](https://github.com/iam-veeramalla/terraform-zero-to-hero/blob/main/Day-7/02-vault-integration.md)

Secret Engines : used to store special secret

Example: Kubernetes secret -> configure Kubernetes secret engine by providing cluster details

Key value pair -> use key vault secret engine

Secret engine -> create a new engine -> KV ->

To access this secrets through terraform or ansible -> Create a role inside your hashicorp -> Create an IAM role -> for IAM role grnt the policies ->

Click on access -> policies -> Before creating IAM role in AWS you have to create IAM policy ->

Terraform cloud -> [Profile | Account | HCP Terraform](https://app.terraform.io/app/settings/profile)

**Terraform cloud**

Terraform Cloud is a remote environment that is optimized for the Terraform workflow. It provides features like workspaces and state locking, which allows people in big teams to collaborate.

**State in Terraform**

As an IAC tool, terraform should know the current state of configurations and infrastructure under its management. Terraform stores this information in a file called the state file.

**benefit of Terraform State**

The Terraform State allows Terraform to map real-world resources to your configuration, keep track of metadata, and improve performance when planning changes for complex infrastructures. It is a critical component of Terraform.

**Terraform Backend**

Terraform backend is the platform where the Terraform State Snapshots are stored. By default, Terraform uses a backend called local to store state as a local file on your disk. All other supported backends are some kind of remote storage service.

**provider in Terraform**

Providers in Terraform are plugins that allow Terraform to interact with cloud providers, SaaS providers, and other APIs. For example, if we plan on using Terraform to provision infrastructure on AWS, we will need to declare an AWS provider in our configuration files.

**maintains Terraform Providers**

Providers are distributed separately from Terraform itself. As a Terraform user, anyone can develop their own providers. There are some standard providers that are maintained explicitly by Hashicorp.

**Sentinel**

[Sentinel](https://www.terraform.io/cloud-docs/sentinel) is a policy as a code tool used to enforce standard configurations for resources being deployed by Terraform. It can be used by organizations for compliance and governance purposes.

**Modules in Terraform**

A Terraform module is a standard container for multiple resources used together to provision and configure resources. For example, you can create a “VPC module” for your organization that provisions a standard VPC and other resources like Subnets and Internet Gateways. Modules can be shared publically via the Public module registry and privately via the Private Module registry.

**benefit of using modules in terraform**

Terraform modules allow us to create logical abstraction on the top of a resource set. Using modules allows us to maintain and reuse a standard configuration for resources. They can be versioned and shared with members of your teams to provision resources in a standard way.

Private Module Registry

A Private Module Registry Terraform Cloud feature allows us to share Terraform modules across our organization.

**we export data from one module to another**

We can export data from a module by defining output blocks in the module configuration files. This data can then be transferred as a parameter to the destination module.

**dependencies in Terraform**

Terraform has built-in dependency management. Terraform has two kinds of dependencies between resources- implicit and explicit dependencies.

Implicit dependencies, as the name suggests, are detected by Terraform automatically. This is when the output of a “resource A” is used in “resource B”. Terraform automatically detects that “resource B” needs to be created only after “resource A”

Explicit dependencies can be specified in cases where two resources are internally dependent on each other without sharing any outputs. This can be done by using the depends\_on parameter in the configuration block.

**Provisioners in Terraform**

Provisioners are Terraform resources used to execute scripts as a part of the resource creation or destruction. There are two types of Provisioners in Terraform:

* **local-exec:**Invokes a script on the machine running Terraform.
* **remote- exec:**Invokes a script on a remote resource after it is created.

Provisioners are only meant to be used as a last resort in Terraform.

**external data block in Terraform**

Just like the local-exec provisioner, external data bock can be used to run scripts on machines running Terraform. The difference between a provisioner and the external data block is that the scripts in the external data block can return data in JSON format, whereas provisioners cannot return any outputs. It is important to note that external data blocks are also meant to be a last resort and should not be used if there is a better alternative.

**Terraform cloud can create two different sets of infrastructure using the same working directory**

By using different workspaces. These users can start Terraform runs in two separate workspaces. Each workspace has a state file of its own, so as long as the **resources do not overlap**, both the users can successfully provision two different sets of infrastructure using the same code.

**What happens when multiple engineers start deploying infrastructure using the same state file**

Terraform has a very important feature called **“state locking”**. This feature ensures that no changes are made to the state file during a run and prevents the state file from getting corrupt. It is important to note that not all Terraform Backends support the state locking feature. You should choose the right backend if this feature is a requirement.

**null resource in Terraform**

A terraform null resource is a configuration that runs like a standard terraform resource block but does not create any resources. This may sound like a strange and useless resource, but it **can be useful in various situations to work around limitations in Terraform.**

**How can you use the same provider in Terraform with different configurations**

By using alias argument in the provider block.

**You have a Terraform configuration file with no resources. What happens when you run the terraform apply command?**

Terraform will destroy all the resources. Starting an empty run with terraform apply command is exactly the same as starting the terraform destroy run.

**What happens if a resource was created successfully in terraform but failed during provisioning**

This is an unlikely scenario, but when this happens, the resource is marked as tainted and can be recreated by restarting the terraform run.

**TF\_LOG variable provides the MOST verbose logging**

TRACE is the most verbose and the default value of the TF\_LOG variable.

**import existing resources under Terraform Management**

terraform import command.

**command can be used to preview the terraform execution plan**

terraform plan command generates the execution plan of the changes Terraform will do to the infrastructure.

**command can be used to reconcile the Terraform state with the actual real-world infrastructure**

The terraform apply -refresh-only command is used to reconcile Terraform state with the actual real-world infrastructure. It is the new alternative to the terraform refresh command, which is now deprecated.

**command can be used to switch between workspaces when using Terraform Cloud**

terraform workspace select <workspace-name> command is used to choose a different workspace.

**command is used to perform syntax validation on terraform configuration files**

The terraform validate command is used to verify whether a configuration is syntactically valid and internally consistent.

**command is used to create new workspaces in the Terraform cloud**

terraform workspace new <workspace-name> command is used to create a new workspace

**important terraform commands**

* **terraform init:** Initializes remote backends; downloads providers and remote modules defined in your configuration.
* **terraform init -upgrade:** used to upgrade the existing downloaded providers.
* **terraform plan:**generates the execution plan for the infrastructure creation or updation.
* **terraform apply:**creates or updates the infrastructure after requesting confirmation from user.
* **terraform apply –auto-approve:**creates or updates the infrastructure; user approval stage is skipped.
* **terraform destroy:**deletes the infrastructure after requesting confirmation from user.
* **terraform destroy –auto-approve:**deletes the infrastructure; user approval stage is skipped.
* **terraform fmt**: scans the current directory for configuration files and formats them according to the HCP canonical style and format.
* **terraform fmt –recursive:**scans the current directory as well as the sub directories for configuration files and formats them according to the HCP canonical style and format.
* **terraform show:**provides a human-readable output from a state or plan file.

A screenshot of a computer

Description automatically generated

**basic structure of a Terraform configuration file**

 It is made up of parts like variable for input variables, resource to define resources, provider to determine the cloud provider, and output to extract and display values. Parameters and values defining infrastructure settings and components are provided in each block.

First provisioning project

Steps:

Decide resources needed storage, server, database, Vnet, etc

Version configuration

Inside CI/Cd putting terafform

First make IAM role

Users -> Add users

User name

Secret AWS credential type : Access key – programmatic access

Next permissions

Attach existing policies

Administarative access

Rest whatever the resource is give permission accordingly

Add tags : env test

Create user

Store your access file credentials ->

Inside vs code install plugins Hashicorp Terraform

Make these files -> Ec2.tf

provider ”aws”{

region =”us-east-2”

access\_key =””

secret\_key =””

}

terraform init -cmd

resource “aws\_instance” “ myec2”{

ami = “”

instance\_initiated\_shutdown\_behavior=”t2.micro”

}

Terraform init

Terraform plan

Terraform apply

**List of terraform providers**

1. **AWS Provider**: Allows you to manage Amazon Web Services (AWS) resources.
2. **Google Provider**: Enables management of Google Cloud Platform (GCP) resources.
3. **Azure Provider**: For managing Microsoft Azure resources.
4. **Kubernetes Provider**: Helps you manage Kubernetes clusters and resources.
5. **Docker Provider**: Manages Docker containers and images.
6. **GitHub Provider**: Allows interaction with GitHub repositories and organizations.
7. **Null Provider**: A special provider that does nothing, useful for certain scenarios.
8. **Oracle Cloud Infrastructure Terraform Provider**: For managing Oracle Cloud resources.
9. **Alibaba Cloud Terraform Provider**: Enables management of Alibaba Cloud resources.

**In a workflow, why it is necessary to divide the application in a hierarchical order**

It has several reasons. The very first thing is hierarchical order gives a complete information on all the sub-tasks that are associated. In addition to this, all the previous jobs and stages can be focused clearly, It is a well known fact that the jobs and stages are depended on their previous sections while performing several tasks on Bamboo. SO if the developers don’t clear information regarding the previous sub-tasks, there can be lot of errors. Thus it’s good to consider them in a hierarchical order.

**Multiple jobs can be created from a single job in bamboo**

It is basically a simple approach. For this, the list of tasks can be divided into simpler and well-defined jobs. It is necessary to make sure that every task or job must be accomplished before next job is considered. This is necessary because many times it has been observed that a stage or a job depends on some or one previous jobs. If the same is not completed, there are errors in the next and you need to waste the time locating them. The problem in one job means problem.

Plan in Bamboo

A plan in bamboo is basically something that provides complete information on the configuration for the tasks being performed in a serial or in a random manner. It deals with the information that is related to the Plan variables, Source code repository, build triggers, authorized users who can change or view the plan. At the beginning, a plan has only one stage. However, it is possible to create multiple stages simply by considering a grouping approach. Different jobs can be grouped together and can be made run in a sequence. It is also possible to share build artifacts simply with one another.

Important components that Atlassian bamboo is integrated with

* They are Build tools such as Maven, and second is the Version Controlled System. Both these tools are equally powerful and can simply let you add efficiency in some of the tasks that are associated with the server. They are good enough to deal with both structured as well as unstructured data.

Task handling be made easier in bamboo

Well, the very thing can be done is taking the support of plug-ins. The Atlassian Bamboo supports a large number of plug-in that can make several tasks simpler. In the development environment, the bug handling can be considered which assures smooth operation for the next level to come. It is possible to integrate this approach with maven helps in cutting down a lot of important tasks. It can be integrated with LDAP server to let develpoers keep an eye on everything.

Avoid project failure or breaking in the Atlassian bamboo Environmemnt

They are certain factors that if considered in a proper manner can simply help in cutting down the problem associated. The first thing needs to be done for this is the proper installation of bamboo. It is also important to pay attention to all the units tests without which there are problem that don’t go easily. If a closed eye is kept on all code changes, this task can be made simpler. Synchronization with repository also cut down the chances of failure or breaks.

Monitoring Tools

Prometheus And Grafana

Use to monitor highly dynamic container environments like Kubernetes docker swarm etc ->

however it can also be used in a traditional non-container infrastructure -> where you have just bare servers with applicaions deployed ->

Used to monitor Container & Microservices Infrastructure ->

Why use Prometheus?

Devops becomes more complex.

You have many Server that run containerized application and that there are hundreds of different processes running on that infrastructure and things are interconnected.

You have no idea what is happening on hardware or on application.

Response Latency

Errors

Overloaded

Enough resources

Example: One specific server ran out of memory and kicked off a running container that was responsible for providing database sync between two database pods in a Kubernetes cluster that in turn caused those two database parts to fail that database was used by an authentication service that also stopped working because the database became unavailable and then application that depended on that authentication service couldn’t authenticate users in the ui anymore but from a user perspective all you see is ui can’t login so how do you know what actually went wrong when you don’t have any insight of what’s going on inside the cluster you don’t see that red line of the chain of events as displayed here you just see the Error ”Authentication failed “ .

Things you will check:

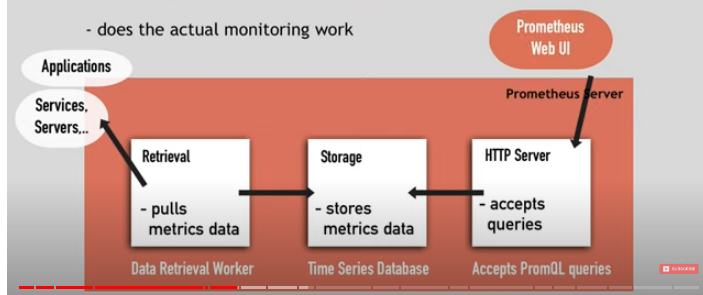
* + Backend running?
  + Any exceptions?
  + Auth-Service running?
  + Why did Auth-Service crash?

Solve this issue

* + Constantly monitor all the services
  + Alert when crash
  + Identify problems before
  + Checking memory usage (notify administrator if greater than 70%)
  + Can’t see the logs for your application because 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 again the monitoring tool would check continuously the storage space and compare with the elastic search consumption of space of storage and it will se 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 for example if you have a very important application that absolutely can have any log data loss you may be very strict and want to take measures as soon as 50-60% capacity is reached or maybe you know adding more storage space eill take long because it’s buratic process so you need solution for this situation.

Main Component: Prometheus Server

* + Does the actual monitoring work



Targets and metrics

What does Prometheus monitor

Linux/Windows

Single Application

Apache Server

Service like database

Those things that Prometheus monitor are called targets and each target has units of monitoring for linux server .

Which Units are monitored of those targets?

CPU Status

Memory/Disk Space Usage

Exceptions count

Request count

Request Duration

**Metrics**

* + Format: Human-readable text-based
  + Metrics entries : TYPE and HELP attributes

HELP -> description of what the metrics is

TYPE -> 3 metrics types

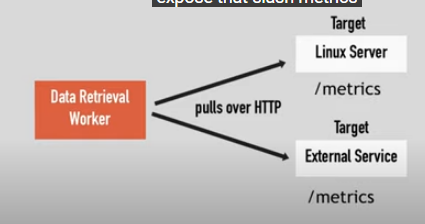
1. Counter -> ….how many times x happened
2. Gauge -> ….what is the current value of x now
3. Histogram -> …how long or how big

…what is the current capacity of disk space

Collecting Metrics Data from Targetes

* + Pulls from HTTP endpoints
  + Hostaddress/metrics
  + Must be in correct format

One targets must expose the slash metrics



Target Endpoints and Exporters

Exposing/metrics endpoints by default

Many services need another component -> Exporter

A screen shot of a computer

Description automatically generated

Monitor a linux Server?

* + Download a node exporter
  + Untar and execute
  + Converts metrics of the server
  + Exposes/ metrics endpoint
  + Configure to scrape this endpoint

Exporter available as docker image

Example: If you want to monitor mysql container in Kubernetes cluster you can deploy a side car container of mysql exporter that will run inside the pod with mysql container connect to it and start translating mysql metrics for Prometheus and making them available at it’s own slash metrics endpoint and again once you add mysql exporter endpoint to Prometheus configuration Prometheus will start collecting those metrics and saving them in its database .

Monitoring your own applications

* + How many requests?
  + How many exceptions?
  + How many server resources are used?

For these there are different languages like node.js java etc using these libraries you can expose the slash metrics endpoint in your application and provide different metrics that are relevant for you on that endpoint.

Using client libraries you can expose/metrics endpoint

Pull Mechanism

Push system of other monitoring systems

Amazon Cloud Watch

New Relic

Applications/Servers push to centralized collection platform of that monitoring tool.

Each service will push there metrics to the monitoring system. It creates a high load of traffic within infrastructure

* + High load of network traffic.
  + Monitoring can become your bottleneck.
  + Install additional software or tools to push metrics

Pull system – more advantages

* + Multiple Prometheus instances can pull metrics data
  + Better detection/ insights if service is up and running

Pushgateway

When targets only runs for a short time?

A screenshot of a video

Description automatically generated

Configuring Prometheus

Prometheus.yml

Which targets?

At what interval?

A screenshot of a computer

Description automatically generated

How often Prometheus will scrape its targets

Global:

Scrape\_interval: 15s

Evaluation\_interval:15s

Rules for aggregating metrics values or creating alerts when condition met

Rule\_files:

# - “first.rules”

# -“second.rules”

What resources Prometheus monitors

Scrape\_configs:

* + Job\_name: Prometheus

static\_configs:

* + targets:[‘localhost:9090’]

Prometheus has its own/metrics endpoint

Define your own jobs

Default values for each job:

* + job\_name: node\_exporter

scrape\_interval: 1m

scrape\_timeout: 1m

static\_configs:

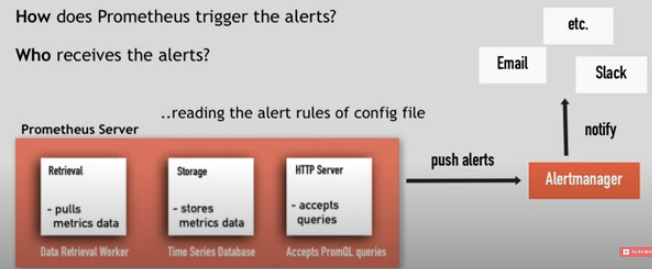
metric\_path: “/metrics”

scheme:”http”

Alert Manager -> responsible for firing alerts

How does Prometheus trigger the alerts?

Who receives the alerts



Promethus Data Storage

Where does prometheus store the data?

Stores data in disk (HDD/SSD)

A screenshot of a computer

Description automatically generated

Querying Prometheus

A diagram of a software development process

Description automatically generated

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 30 mins

My Experience

Learning PromQl

Configuring Prometheus YAML Files

Creating Grafana Dashboards

Complex & not well documented

Prometheus Characteristics

Reliable

Stand-alone and self-containing

Works, even if other parts of infrastructure broken

No extensive set-up needed

Kubernetes,12-1

Jenkins,1-2

Docker,2-3

CI/CD,3-4

Azure,4-5

Github/Gitlab5-6

Plan

Kubernetes

* + automated deployment
  + scaling
  + operation
  + container orchestration
  + container runtime
  + container centric infrastructure
  + balance of load
  + service discovery

**Kubernetes handles container scaling**

**To automatically scale the workload to match demand, a Horizontal Pod Autoscaling in Kubernetes updates a workload resource(such a deployment or stateful set). Horizontal scaling indicates that more pods are added in response to an increase in load.**

**apiVersion: autoscaling/v2beta2**

**kind: HorizontalPodAutoscaler**

**metadata:**

**name: my-hpa**

**spec:**

**scaleTargetRef:**

**apiVersion: apps/v1**

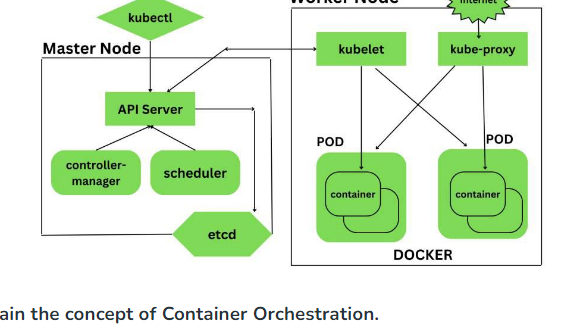
**kind: deployment or Statefulset or ReplicaSet, depending on your workload**

**name: my-deployment**

**Kubelet**

**Kubelet is an important component of Kubernetes that manages containers within pods on a node. It registers the node with the control plane and provides resource information. Kublet keeps an eye on container health and respods to problems-lik isnces of pods that contain a containerized application. Deployment can help efficiently scale the number of replica pods, enable the rollout of updated code in controlled manner, or roll back to the earlier deployment version if necessary.**

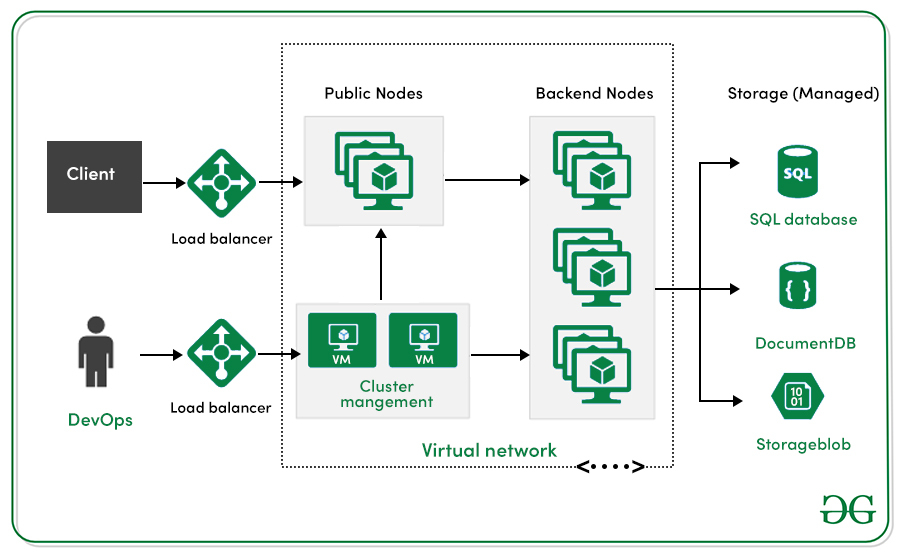
**Kubernetes Architecture.**



Concept of Container orchestration

Container orchestration used to automate the life cycle management of the container.

Provides automatic deployment, scaling and management of containerized applications



Pods

A cluster of one or more Linux containers makes up a Kubernetes pod.

Smallest unit of a Kubernetes application.

Kubectl get pods -n <namespace-name>

Kubernetes handle container scaling

To automatically scale the workload to match demand, a horizontal Pod Autoscaling in kubernetes updates a workload resource (such a deployment or stateful set). Horizontal scaling indicates that more pods are added in response to an increase in load.

*apiVersion: autoscaling/v2beta2*

*kind: HorizontalPodAutoscaler*

Kubelet

Kublet is an important component of Kubernetes that manages containers within pods on a node. It registers the node with the control plane and provides resource information. Kubelet keeps an eye on container health and responds to problems lik isnces of pods that contain a containerized application. Deployments can help to efficiently scale the number of replica pods, enable the rollout of updated code in a controlled manner, or roll back to the earlier deployment version if necessary.

apiVersion: apps/v1

kind: Deployment

StatefulSet and a deployment

StatefulSet Deployment

A collection of identical stateful pods This resource controls identical

are handled by the resource is pods deployment.

called StatefulSet.

Statefulset helpful in managing stateful It enables you to control your

Applications that need persistent storage application’s state and ensure that the

With a dependable network ID. Right number of replicas are always running

Service in Kubernetes

Service is to group a set of pods endpoints into a single resource. We can configure various ways to access the grouping. By default, we can get a stable cluster IP address that the clients inside the cluster can use to contact pods in service.

apiVersion: v1

kind: Service

metadata:

name: my-service

spec:

selector:

Tomcat: deploymentapp

Ports:

* + protocol: TCP

port:80

targetPort: 8080

Kubernetes manage configuration

Kubernetes employs configMaps and Secrets to manage configuration. ConfigMaps store non-sensitive setup data, while Secrets handles sensitive information like passwords. These resources have different configuration from the application code, making updates easier.

ConfigMaps have key-value pairs for different settings that can be accessed as environment variables or mounted files. Sensitive data is securely stored in Secrets, which are applied to the cluster using kubectl. Both ConfigMaps and Secrets are defined in YAML files and applied to the cluster using

Main difference between Jenkins and Bamboo

Jenkins

* [**Open-source**: Jenkins is a widely-used open-source CI/CD tool1](https://www.browserstack.com/guide/jenkins-vs-bamboo).
* [**Extensive plugin support**: It has a large number of plugins available, which allows for extensive customization and integration with other tools1](https://www.browserstack.com/guide/jenkins-vs-bamboo).
* [**Community-driven**: Being open-source, it has a strong community that contributes to its development1](https://www.browserstack.com/guide/jenkins-vs-bamboo).
* [**Flexibility**: Jenkins can be configured to fit a wide range of CI/CD needs1](https://www.browserstack.com/guide/jenkins-vs-bamboo).

**Bamboo**:

* [**Commercial product**: Bamboo is a commercial CI/CD tool developed by Atlassian1](https://www.browserstack.com/guide/jenkins-vs-bamboo).
* [**Integration with Atlassian products**: It offers seamless integration with other Atlassian products like Jira and Bitbucket1](https://www.browserstack.com/guide/jenkins-vs-bamboo).
* [**Built-in capabilities**: Bamboo comes with more out-of-the-box features compared to Jenkins, which can make it easier to use for some teams1](https://www.browserstack.com/guide/jenkins-vs-bamboo).
* [**Subscription-based**: Unlike Jenkins, Bamboo requires a subscription fee](https://www.browserstack.com/guide/jenkins-vs-bamboo)[1](https://www.browserstack.com/guide/jenkins-vs-bamboo).

Kafka -> Data is managed on server-> data is routed for every second

Primary to secondary data transfer -> every second data is updated ->

Example : ola ,Zomato

Kafka can send many messages -> high through out

Kafka provides flexibility -> open source tool

Different microservices -> if you make changes in one place ->

Changes made in one module will be reflected in all module.

Triilion message can be handled by kafka.