AWS & DEVOPS BY VEERA SIR

----------###GIT###----------

*****Installation for Git in your local laptop :

- URL- https://git-scm.com/downloads #it support os like Windows,Linux,Mac.
- yum install git -y # for liunux

-------After completed of installation go to cmd promt check : git --version

- git init -----to intilaise git process

-----introduce yourself to Git----------

------all commands start with git only----------

----------Git commands----------

----------### Tell Git who you are----------

- git config --global user.name --- to check configured name
- git config --global user.email --- to check configured email
- git config --global user.name "veer"
- git config --global user.email "cloud87777@gmail.com"

----------git workflow----------

- git add -----to add file into staging area from local working directory
- git status ----to see the status of the file where it is
- git commit -m "comment"

----------#git changes#----------

- git diff ------ compare changes local directory to staging area
- git diff --staged --comapare chnages staging to local repo
- git diff head ----comapare changes local directory to local repo
- git diff commit id to commit id -----compare chnages in to two commits

---

Revert changes on git

##Revert changes from working directory

- git restore <file_name>

  or

- git checkout -- <file_name> ###like undo command

----Note: even you can remove changes manually. But if we have updated multiple files and don't know which lines to remove this command really helps`

---Revert changes from Staging Area

- git restore --staged <file_name> #to revert changes from Staging area to working directory

--- Revert changes from Local Repository

- git reset HEAD~1 # to revert changes from local repo to working directory
- git restore <file>

#if The git reset HEAD~2 command moves the current branch backward by two commits

Note: if you can give git status or any git commit command everything clear no untracked file no staging files

---------- <<< GITHUB >>>----------

GitHub is an online software development platform. It's used for storing, tracking, and collaborating on software projects. It makes it easy for developers to share code files and collaborate with fellow developers on open-source projects.

- git clone ---- to clone entire repository in first time
- git pull ------to pull the updates and chnages from repository
- git push ------to push the local updates into remote repo

-----git push-----

push the changes by using SSH

To generate SSH Key

- ssh-keygen (give this command into your terminal)
- copy the publickey id_rsa.pub
- paste into github ssh keys
- ➔ settings-->ssh and GPG keys--> Add new ssh key and give any name and paste it your public over there

git local to remote sync (how to add that locally created folder into git hub)

- ➔ after creating repos locally and remote we have to give following commands
- git remote set-url origin git@github.com:User/UserRepo.git
- now it will sync we can start pull push commands

---

push the chnages by using GitHUb Token

###To generate Token

- ➔ settings-->Developer settings-->personal access token -->Token classic --> generate token

- git remote set-url origin https://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@github.com:/user/repository.git
- git remote set-url origin https://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@github.com:/CloudTechDevOps/check.git
- ==========

-----Mapping local repo to remote repository

...####### create a new repository on the command line############

- echo "# check" >> README.md
- git init
- git add README.md
- git commit -m "first commit"
- git branch -M main
- git remote add origin https://github.com/CloudTechDevOps/check.git
- git push -u origin main

########...or push an existing repository from the command line######

- git remote add origin https://github.com/CloudTechDevOps/check.git
- git branch -M main
- git push -u origin main

---------- # Git Branching #----------

BRANCHES:

- ➔ It's an individual line of development for code.
- ➔ we create individual branch in real-time to do test and develope.
- ➔ each developer will work on their own branch.
- ➔ At the end we will merge all branches together.
- ➔ Default branch is Master.

- git branch : to list the branches
- git branch dev : to create a new branch
- git checkout dev : to switch from one branch to another.
- git checkout -b dev : to create and switch from one branch to another.
- git branch -m old new : to rename a branch
- git branch -D dev : to delete a branch

---------- ###### What is Git Merge? ########----------

Git marge will help to combine the changes from two or more branches into a single branch. Developers will work on different branches to improve code or to develop the code after completion we can merge them into a single version of the code.

---------- ## What is Git Rebase? ###----------

Rebase is one of two Git utilities designed to integrate changes from one branch onto another. Rebasing is the process of combining or moving a sequence of commits on top of a new base commit. Git rebase is the linear process of merging.

---------- ###### what is cherry-pick? ######----------

With the "cherry-pick" command, we can pick individual commits from any branch into your current HEAD branch.

Contrast this with the way commit integration normally works in Git: when performing a Merge or Rebase, all commits from one branch are integrated.

---------- #### what is git ignore? ##### ----------

The .gitignore file tells Git which files to ignore when committing your project to the GitHub repository. gitignore is located in the root directory of your repo.

TERRAFORM

What Is Terraform?

Terraform is an IAC tool, used primarily by DevOps teams to automate various infrastructure tasks. The provisioning of cloud resources, for instance, is one of the main use cases of Terraform. It's a cloud-agnostic, open-source provisioning tool written in the Go language and created by HashiCorp.

Terraform allows you to describe your complete infrastructure in the form of code. Even if your servers come from different providers such as AWS or Azure, Terraform helps you build and manage these resources in parallel across providers. Think of Terraform as connective tissue and common language that you can utilize to manage your entire IT stack.

###--- Benefits of Infrastructure-as-Code (IaC) ---###

IaC replaces standard operating procedures and manual effort required for IT resource management with lines of code. Instead of manually configuring cloud nodes or physical hardware, IaC automates the process infrastructure management through source code.

Here are several of the major key benefits of using an IaC solution like Terraform:

Speed and Simplicity. IaC eliminates manual processes, thereby accelerating the delivery and management lifecycles. IaC makes it possible to spin up an entire infrastructure architecture by simply running a script.

Team Collaboration. Various team members can collaborate on IaC software in the same way they would with regular application code through tools like Github. Code can be easily linked to issue tracking systems for future use and reference.

Error Reduction. IaC minimizes the probability of errors or deviations when provisioning your infrastructure. The code completely standardizes your setup, allowing applications to run smoothly and error-free without the constant need for admin oversight.

Disaster Recovery. With IaC you can actually recover from disasters more rapidly. Because manually constructed infrastructure needs to be manually rebuilt. But with IaC, you can usually just re-run scripts and have the exact same software provisioned again.

Enhanced Security. IaC relies on automation that removes many security risks associated with human error. When an IaC-based solution is installed correctly, the overall security of your computing architecture and associated data improves massively.

Install Terraform

Windows

Install Terraform from the Downloads Page (select windows

------ ### Basic Terraform folder structure ### ------

projectname/

|

|-- provider.tf

|-- version.tf

|-- backend.tf

|-- main.tf

|-- variables.tf

|-- terraform.tfvars

|-- outputs.tf

Give Terraform files logical names

Terraform tutorials online often demonstrate a directory structure consisting of three files:

- main.tf: contains all providers, resources and data sources
- variables.tf: contains all defined variables
- output.tf: contains all output resources

The issue with this structure is that most logic is stored in the single main.tf file which therefore becomes pretty complex and long. Terraform, however, does not mandate this structure, it only requires a directory of Terraform files. Since the filenames do not matter to Terraform I propose to use a structure that enables users to quickly understand the code. Personally I prefer the following structure

- provider.tf: contains the terraform block and provider block
- data.tf: contains all data sources

- variables.tf: contains all defined variables
- locals.tf: contains all local variables
- output.tf: contains all output resources

---------- ##### Important Terraform Commands ### --------

Version

- terraform –version ###Shows terraform version installed

Initialize infrastructure

- terraform init ##Initialize a working directory
- terraform init -input=true ##Ask for input if necessary
- terraform init -lock=false ##Disable locking of state files during state-related operations
- terraform plan ###Creates an execution plan (dry run)
- terraform apply ###Executes changes to the actual environment
- terraform apply –auto-approve ###Apply changes without being prompted to enter ”yes”
- terraform destroy –auto-approve ###Destroy/cleanup without being prompted to enter ”yes”

Terraform Workspaces

- terraform workspace new ##Create a new workspace and select it
- terraform workspace select ##Select an existing workspace
- terraform workspace list ##List the existing workspaces
- terraform workspace show ##Show the name of the current workspace
- terraform workspace delete ##Delete an empty workspace

Terraform Import

- terraform import aws_instance.example i-abcd1234(instance id) #import an AWS instance with ID i-abcd1234 into aws_instance resource named “foo”

---## Statefile ##--:

What is state and why is it important in Terraform?

“Terraform must store state about your managed infrastructure and configuration. This state is used by Terraform to map real world resources to your configuration, keep track of metadata, and to improve performance for large infrastructures. This state file is extremely important; it maps various resource metadata to actual resource IDs so that Terraform knows what it is managing. This file must be saved and distributed to anyone who might run Terraform.”

Remote State:

“By default, Terraform stores state locally in a file named terraform.tfstate. When working with Terraform in a team, use of a local file makes Terraform usage complicated because each user must make sure they always have the latest state data before running Terraform and make sure that nobody else runs Terraform at the same time.”

“With remote state, Terraform writes the state data to a remote data store, which can then be shared between all members of a team.”

State Lock:

“If supported by your backend, Terraform will lock your state for all operations that could write state. This prevents others from acquiring the lock and potentially corrupting your state.”

“State locking happens automatically on all operations that could write state. You won’t see any message that it is happening. If state locking fails, Terraform will not continue. You can disable state locking for most commands with the -lock flag but it is not recommended.”

Setting up our S3 Backend

Create a new file in your working directory labeled Backend.tf

Copy and paste this configuration in your source code editor in your backend.tf file.

```
terraform {
  backend "s3" {
    encrypt = true
     bucket = "sample"
    dynamodb_table = "terraform-state-lock-dynamo"
    key    = "terraform.tfstate"
    region = "us-east-1"
  }
}
```

before that we have to create resource are s3 and dynamodb those resource will call in backend.tf

Creating our DynamoDB Table

Create a new file in your working directory labeled dynamo.tf

Copy and paste this configuration in your source code editor in your main.tf file.

```
# S3
resource "aws_s3_bucket" "example" {
  bucket = sample
}



#Dynamodb
   resource "aws_dynamodb_table" "dynamodb-terraform-state-lock" {
     name = "terraform-state-lock-dynamo"
     hash_key = "LockID"
     read_capacity = 20
     write_capacity = 20

     attribute {
       name = "LockID"
       type = "S"
     }
   }

 ## --- DATA source ##-----
```

What is Data Source?

Data source in terraform relates to resources but only it gives the information about an object rather than creating one. It provides dynamic information about the entities we define outside of terraform.

Data Sources allow fetching data about the infrastructure components' configuration. It allows to fetch data from the cloud provider APIs using terraform scripts.

When we refer to a resource using a data source, it won't create the resource. Instead, they get information about that resource so that we can use it in further configuration if required.

How to use Data Source?

For example, we will create an ec2 instance using a vpc and subnet, both of which are created on aws console that is external to terraform configuration.

Step 1: Create a terraform directory and create a file named provider.tf in it. Below code represents the details of the aws provider that we're using, like its region, access key and secret key.

```
provider "aws"{
 region    = "us-east-1"
 access_key = "your_access_key"
 secret_key = "your_secret_key" #keys no  need to configure here it will call from .aws folder from local
}
```

Step 2: In that directory, create another file named demo_datasource.tf and use the code given below.

```
data "aws_vpc" "vpc" {
 id = vpc_id
}

data "aws_subnet" "subnet" {
 id = subnet_id
}

resource "aws_security_group" "sg" {  # here we are creating security grop by calling exicting vpc so we can use data source block
 name = "sg"
 vpc_id = data.aws_vpc.vpc.id
 ingress            = [
 {
   cidr_blocks     = [ "0.0.0.0/0"]
   description     = ""
   from_port       = 22
   protocol        = "tcp"
   security_groups  = []
```

```
    self          = false
    to_port       = 22
  }
  ]
  egress = [
   {
     cidr_blocks     = [ "0.0.0.0/0"]
     description     = ""
     from_port       = 0
     protocol        = "-1"
     security_groups  = []
     self          = false
     to_port       = 0
   }
  ]
}


resource "aws_instance" "dev" {
   ami = data.aws_ami.amzlinux.id
   instance_type = "t2.micro"
   subnet_id = data.aws_subnet.dev.id
   security_groups = [ data.aws_security_group.dev.id ]
tags = {
   Name = "DataSource- Instance"
  }
}
```

In the above block of code, we are using a vpc and a subnet that is already created on AWS using its console. Then using data block, which refers to data sources, that is, a vpc and a subnet. By doing this, we are retrieving the information about the vpc and subnet that are created outside of terraform configuration. Then creating a security group that uses vpc_id that was fetched using data block. Further creating the EC2 instance that uses the subnet_id that was also fetched using data block.

So, in this example, data source is being used to get data about the vpc and subnet that were not created using terraform script and using this data further for creating an EC2.

case-2

we can call AMI id also by using data source block

```
data "aws_ami" "amzlinux" {
most_recent = true
owners = [ "amazon" ]
filter {
  name = "name"
  values = [ "amzn2-ami-hvm-*-gp2" ]
}
filter {
  name = "root-device-type"
  values = [ "ebs" ]
}
filter {
  name = "virtualization-type"
  values = [ "hvm" ]
}
filter {
  name = "architecture"
  values = [ "x86_64" ]
}
}
```

---------- #### Terraform Import ####----------

Why Terraform Import?

Terraform is a relatively new technology and adopting it to manage an organisation's cloud resources might take some time and effort. The lack of human resources and the steep learning curve involved in using Terraform effectively causes teams to start using cloud infrastructure directly via their respective web consoles.

For that matter, any kind of IaC method (CloudFormation, Azure ARM templates, Pulumi, etc.) requires some training and real-time scenario handling experience. Things get especially complicated when dealing with concepts like states and remote backends. In a worst case scenario, you can lose the terraform.tfstate file. Luckily, you can use the import functionality to rebuild it.

Getting the pre-existing cloud resources under the Terraform management is facilitated by Terraform import. import is a Terraform CLI command which is used to read real-world infrastructure and update the state, so that future updates to the same set of infrastructure can be applied via IaC.

The import functionality helps update the state locally and it does not create the corresponding configuration automatically. However, the Terraform team is working hard to improve this function in upcoming releases.

Simple Import

With an understanding of why we need to import cloud resources, let us begin by importing a simple resource – EC2 instance in AWS. I am assuming the Terraform installation and configuration of AWS credentials in AWS CLI is already done locally. We will not go into the details of that in this tutorial. To import a simple resource into Terraform, follow the below step-by-step guide.

1. Prepare the EC2 Instance

Assuming the Terraform installation and configuration of AWS credentials in AWS CLI is already done locally, begin by importing a simple resource—EC2 instance in AWS. For the sake of this tutorial, we will create an EC2 resource manually to be imported. This could be an optional step if you already have a target resource to be imported.

Terraform: Create EC2 Instance in Existing VPC

Go ahead and provision an EC2 instance in your AWS account. Here are the example details of the EC2 instance thus created:

Name: MyVM

Instance ID: i-0b9be609418aa0609

Type: t2.micro

VPC ID: vpc-1827ff72

...

2. Create main.tf and Set Provider Configuration

The aim of this step is to import this EC2 instance into our Terraform configuration. In your desired path, create `main.tf` and configure the AWS provider. The file should look like the one below.

Importing EC2 Instance into Terraform Configuration: Example

```
// Provider configuration
terraform {
 required_providers {
  aws = {
   source  = "hashicorp/aws"
   version = "~> 3.0"
  }
 }
}

provider "aws" {
 region = "us-east-1"
}
```

Run terraform init to initialize the Terraform modules. Below is the output of a successful initialization.

```
Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 3.0"...
- Installing hashicorp/aws v3.51.0...
- Installed hashicorp/aws v3.51.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
```

so that Terraform can guarantee to make the same selections by default when

you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see

any changes that are required for your infrastructure. All Terraform commands

should now work.

If you ever set or change modules or backend configuration for Terraform,

rerun this command to reinitialize your working directory. If you forget, other

commands will detect it and remind you to do so if necessary.

3. Write Config for Resource To Be Imported

As discussed earlier, Terraform import does not generate the configuration files by itself. Thus, you need to create the corresponding configuration for the EC2 instance manually. This doesn't need many arguments as we will have to add or modify them when we import the EC2 instance into our state file.

However, if you don't mind not seeing colorful output on CLI, you can begin adding all the arguments you know. But this is not a foolproof approach, because normally the infrastructure you may have to import will not have been created by you. So it is best to skip a few arguments anyway.

In a moment we will take a look at how to adjust our configuration to reflect the exact resource. For now, append the main.tf file with EC2 config. For example, I have used the below config. The only reason I have included ami and instance_type attribute, is that they are the required arguments for aws_instance resource block.

```
resource "aws_instance" "myvm" {
 ami           = "unknown"(we need to add from state file reference)
 instance_type = "unknown"(we need to add from state file reference)
}
```

4. Import

Think of it as if the cloud resource (EC2 instance) and its corresponding configuration were available in our files. All that's left to do is to map the two into our state file. We do that by running the import command as follows.

import command:

- terraform import aws_instance.myvm <Instance ID>

A successful output should look like this:

aws_instance.myvm: Importing from ID "i-0b9be609418aa0609"...

aws_instance.myvm: Import prepared!

Prepared aws_instance for import

aws_instance.myvm: Refreshing state... [id=i-0b9be609418aa0609]

Import successful!

The resources that were imported are shown above. These resources are now in

your Terraform state and will henceforth be managed by Terraform.

The above command maps the aws_instance.myvm configuration to the EC2 instance using the ID. By mapping I mean that the state file now "knows" the existence of the EC2 instance with the given ID. The state file also contains information about each attribute of this EC2 instance, as it has fetched the same using the import command.

5. Observe State Files and Plan Output

Please notice that the directory now also contains terraform.tfstate file. This file was generated after the import command was successfully run. Take a moment to go through the contents of this file.

Right now our configuration does not reflect all the attributes. Any attempt to plan/apply this configuration will fail because we have not adjusted the values of its attributes. To close the gap in configuration files and state files, run terraform plan and observe the output.

.

.

.

} -> (known after apply)

```
      ~ throughput          = 0 -> (known after apply)
      ~ volume_id           = "vol-0fa93084426be508a" -> (known after apply)
      ~ volume_size         = 8 -> (known after apply)
      ~ volume_type         = "gp2" -> (known after apply)
    }

  - timeouts {}
}

Plan: 1 to add, 0 to change, 1 to destroy.
```

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply"

now.

The plan indicates that it would attempt to replace the EC2 instance. But this goes completely against our purpose. We could do it anyway by simply not caring about the existing resources, and creating new resources using configuration.

The good news is that Terraform has taken note of the existence of an EC2 instance that is associated with its state.

6. Improve Config To Avoid Replacement

At this point, it is important to understand that the terraform.tfstate file is a vital piece of reference for Terraform. All of its future operations are performed with consideration for this state file. You need to investigate the state file and update your configuration accordingly so that there is a minimum difference between them.

The use of the word “minimum” is intentional here. Right now, you need to focus on not replacing the given EC2 instance, but rather aligning the configuration so that the replacement can be avoided. Eventually, you would achieve a state of 0 difference.

Observe the plan output, and find all those attributes which cause the replacement. The plan output will highlight the same. In our example, the only attribute that causes replacement is the AMI ID. Closing this gap should avoid the replacement of the EC2 instance.

Change the value of ami from “unknown” to what is highlighted in the plan output, and run terraform plan again. Notice the output.

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:

~ update in-place

Terraform will perform the following actions:

```
 # aws_instance.myvm will be updated in-place
 ~ resource "aws_instance" "myvm" {
     id                           = "i-0b9be609418aa0609"
   ~ instance_type                = "t2.micro" -> "unknown"
   ~ tags                         = {
       - "Name" = "MyVM" -> null
     }
   ~ tags_all                     = {
       - "Name" = "MyVM"
     } -> (known after apply)
     # (27 unchanged attributes hidden)

     # (6 unchanged blocks hidden)
   }
```

Plan: 0 to add, 1 to change, 0 to destroy.

This time the plan does not indicate the replacement of the EC2 instance. If you get the same output, you are successful in partially importing our cloud resource. You are currently in a state of lowered risk—if we apply the configuration now, the resource will not be replaced, but a few attributes would change.

5 Ways to Manage Terraform at Scale

How to Automate Terraform Deployments and Infrastructure Provisioning

Why DevOps Engineers Recommend Spacelift

7. Improve Config To Avoid Changes

If we want to achieve a state of 0 difference, you need to align your resource block even more. The plan output highlights the attribute changes using ~ sign. It also indicates the difference in the values. For example, it highlights the change in the instance_type value from “t2.micro” to “unknown”.

In other words, if the value of instance_type had been “t2.micro”, Terraform would NOT have asked for a change. Similarly, you can see there are changes to the tags highlighted as well. Let’s change the configuration accordingly so that we close these gaps. The final aws_instance resource block should look as follows:

```
resource "aws_instance" "myvm" {
 ami           = "ami-00f22f6155d6d92c5"
 instance_type = "t2.micro"

 tags = {
    "Name": "MyVM"
 }
}
```

Run terraform plan again, and observe the output.

```
aws_instance.myvm: Refreshing state... [id=i-0b9be609418aa0609]

No changes. Your infrastructure matches the configuration.
```

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

If you have the same output, congratulations, as you have successfully imported a cloud resource into your Terraform config. It is now possible to manage this configuration via Terraform directly, without any surprises.

--------## Meta-arguments ##----------

Meta-arguments in Terraform are special arguments that can be used with resource blocks and modules to control their behavior or influence the infrastructure provisioning process. They provide additional configuration options beyond the regular resource-specific arguments.

Meta-Arguments in Terraform are as follows:

depends_on: Specifies dependencies between resources. It ensures that one resource is created or updated before another resource.

count: Controls resource instantiation by setting the number of instances created based on a given condition or variable.

for_each: Allows creating multiple instances of a resource based on a map or set of strings. Each instance is created with its unique key-value pair.

lifecycle: Defines lifecycle rules for managing resource updates, replacements, and deletions.

provider: Specifies the provider configuration for a resource. It allows selecting a specific provider or version for a resource.

provisioner: Specifies actions to be taken on a resource after creation, such as running scripts or executing commands.

connection: Defines the connection details to a resource, enabling remote execution or file transfers.

variable: Declares input variables that can be provided during Terraform execution.

output: Declares output values that can be displayed after Terraform execution.

locals: Defines local values that can be used within the configuration files.

---

depends_on

Terraform has a feature of identifying resource dependency. This means that Terraform internally knows the sequence in which the dependent resources needs to be created whereas the independent resources are created parallelly.

But in some scenarios, some dependencies are there that cannot be automatically inferred by Terraform. In these scenarios, a resource relies on some other resource's behaviour but it doesn't access any of the resource's data in arguments.

For those dependencies, we'll use depends_on meta-argument to explicitly define the dependency.

depends_on meta-argument must be a list of references to other resources in the same calling resource.

This argument is specified in resources as well as in modules (Terraform version 0.13+)

Example-1

```
#

provider "aws" {
}

resource "aws_s3_bucket" "example" {
  bucket = "qwertyuiopasdfg"

}

resource "aws_instance" "dev" {
   ami = "ami-0440d3b780d96b29d"
   instance_type = "t2.micro"
   depends_on = [ aws_s3_bucket.example] # here depends-on block will help after cration of s3 only ec2 will create if creation of s3 fails ec2 will not create
}
```

Example-2

```
### Create IAM policy
resource "aws_iam_policy" "example_policy" {
  name        = "example_policy"
  description = "Permissions for EC2"
  policy      = jsonencode({
    Version: "2012-10-17",
    Statement: [
      {
        Action: "ec2:*",
```

```
        Effect: "Allow",
        Resource: "*"
      }
    ]
  })
}

### Create IAM role
resource "aws_iam_role" "example_role" {
  name = "example_role"

  assume_role_policy = jsonencode({
   Version = "2012-10-17"
   Statement = [
    {
      Action = "sts:AssumeRole"
      Effect = "Allow"
      Sid    = "examplerole"
      Principal = {
       Service = "ec2.amazonaws.com"
      }
    },
   ]
  })
}

### Attach IAM policy to IAM role
resource "aws_iam_policy_attachment" "policy_attach" {
  name      = "example_policy_attachment"
  roles     = [aws_iam_role.example_role.name]
  policy_arn = aws_iam_policy.example_policy.arn
```

```
}

### Create instance profile using role
resource "aws_iam_instance_profile" "example_profile" {
  name = "example_profile"
  role = aws_iam_role.example_role.name
}

### Create EC2 instance and attache IAM role
resource "aws_instance" "example_instance" {
  instance_type       = var.ec2_instance_type
  ami                 = var.image_id
  iam_instance_profile = aws_iam_instance_profile.example_profile.name
  depends_on = [ aws_iam_role.example_role ] # heare after creating iam role only, ec2 will create and role attach to ec2

}
```

---

count

In Terraform, a resource block actually configures only one infrastructure object by default. If we want multiple resources with same configurations, we can define the count meta-argument. This will reduce the overhead of duplicating the resource block that number of times.

count require a whole number and will then create that resource that number of times. To identify each of them, we use the count.index which is the index number corresponds to each resource. The index ranges from 0 to count-1.

This argument is specified in resources as well as in modules (Terraform version 0.13+). Also, count meta-argument cannot be used with for_each.

example:1

```
resource "aws_instance" "myec2" {
   ami = "ami-0230bd60aa48260c6"
   instance_type = "t2.micro"
   count = 2
   tags = {
   #   Name = "webec2"
     Name = "webec2-${count.index}"
   }
}
```

example:2

```
variable "ami" {
  type    = string
  default = "ami-0440d3b780d96b29d"
}

variable "instance_type" {
  type    = string
  default = "t2.micro"
}

variable "sandboxes" {
  type    = list(string)
  default = [ "sandbox_server_two", "sandbox_server_three"]
}

# main.tf
resource "aws_instance" "sandbox" {
```

```
  ami           = var.ami
  instance_type = var.instance_type
  count         = length(var.sandboxes)

  tags = {
    Name = var.sandboxes[count.index]
  }
}
```

----------## for_each ##----------

As specified in the count meta-argument, that the default behaviour of a resource is to create a single infrastructure object which can be overridden by using count, but there is one more flexible way of doing the same which is by using for_each meta argument.

The for_each meta argument accepts a map or set of strings. Terraform will create one instance of that resource for each member of that map or set. To identify each member of the for_each block, we have 2 objects:

each.key: The map key or set member corresponding to each member.

each.value: The map value corresponding to each member.

This argument is specified in resources (Terraform version 0.12.6) as well as in modules (Terraform version 0.13+)

Example for_each

 # variables.tf

```
variable "ami" {
  type    = string
  default = "ami-0078ef784b6fa1ba4"
}

variable "instance_type" {
  type    = string
```

```
  default = "t2.micro"
}

variable "sandboxes" {
  type    = set(string)
  default = ["sandbox_one", "sandbox_two", "sandbox_three"]
}
```

 # main.tf

```
resource "aws_instance" "sandbox" {
  ami           = var.ami
  instance_type = var.instance_type
  for_each      = var.sandboxes
  tags = {
    Name = each.value # for a set, each.value and each.key is the same
  }
}
```

---

multi provider

---

provider meta-argument specifies which provider to be used for a resource. This is useful when you are using multiple providers which is usually used when you are creating multi-region resources. For differentiating those providers, you use an alias field.

The resource then reference the same alias field of the provider as provider.alias to tell which one to use.

Ex:

```
# Provider-1 for us-east-1 (Default Provider)
provider "aws" {
  region = "ap-south-1"
}
```

```
#Another provider alias
provider "aws" {
 region = "us-east-1"
 alias = "america"
}

resource "aws_s3_bucket" "test" {
 bucket = "del-hyd-naresh-it"

}
resource "aws_s3_bucket" "test2" {
 bucket = "del-hyd-naresh-it-test2"
 provider = aws.america

}
```

----------## Lifecycycle ##----------

What is Terraform Lifecycle Meta-Argument?

The Terraform lifecycle is a nested configuration block within a resource block. The lifecycle meta-argument can be used to specify how Terraform should handle the creation, modification, and destruction of resources. Meta-arguments are arguments used in resource blocks.

Terraform Lifecycle Meta-Argument Example

he lifecycle meta-argument can be used within any resource block like so:

```
Ex:resource "aws_instance" "test" {
   ami = "ami-0440d3b780d96b29d"
   instance_type = "t2.micro"
   availability_zone = "us-east-1b"
```

```
 tags = {
   Name = "test"
 }

 lifecycle {
   create_before_destroy = true    #this attribute will create the new object first and then destroy the old one
 }

# lifecycle {
#   prevent_destroy = true    #Terraform will error when it attempts to destroy a resource when this is set to true:
# }

#   lifecycle {
#     ignore_changes = [tags,] #This means that Terraform will never update the object but will be able to create or destroy it.
#   }
}
```

Managing the Resource Lifecycle Using the Lifecycle Meta-Argument

Controlling the flow of Terraform operations is possible using the lifecycle meta-argument. This is useful in scenarios when you need to protect items from getting changed or destroyed.

A common scenario that requires the use of a lifecycle meta-argument occurs when the Terraform provider itself does not handle a change correctly and so can be safely ignored, rather than the provider attempting to update an object necessarily. With the provider version updates, these “bugs” are slowly ironed out, at which point the lifecycle meta-argument can be removed from the resource.

There are several attributes available for use with the lifecycle meta-argument:

create_before_destroy:

When Terraform determines it needs to destroy an object and recreate it, the normal behavior will create the new object after the existing one is destroyed. Using this attribute will create the new object first and then destroy the old one. This can help reduce downtime. Some objects have restrictions that the use of this setting may cause issues with, preventing objects from existing concurrently. Hence, it is important to understand any resource constraints before using this option.

```
lifecycle {
  create_before_destroy = true
}
```

prevent_destroy

This lifecycle option prevents Terraform from accidentally removing critical resources. This is useful to avoid downtime when a change would result in the destruction and recreation of resource. This block should be used only when necessary as it will make certain configuration changes impossible.

=========

```
lifecycle {
  prevent_destroy = true
}
```

Terraform will error when it attempts to destroy a resource when this is set to true:

Error: Instance cannot be destroyed

resource details...

Resource [resource_name] has lifecycle.prevent_destroy set, but the plan calls for this resource to be destroyed. To avoid this error and continue with the plan, either disable lifecycle.prevent_destroy or reduce the scope of the plan using the -target flag.

==========

##ignore_changes

The Terraform ignore_changes lifecycle option can be useful when attributes of a resource are updated outside of Terraform.

It can be used, for example, when an Azure Policy automatically applies tags. When Terraform detects the changes the Azure Policy has applied, it will ignore them and not attempt to modify the tag. Attributes of the resource that need to be ignored can be specified.

In the example below, the department tag will be ignored:

```
lifecycle {
  ignore_changes = [
    tags["department"]
  ]
}
```

If all attributes are to be ignored, then the all keyword can be used. This means that Terraform will never update the object but will be able to create or destroy it.

```
lifecycle {
  ignore_changes = [
    all
  ]
}
```

----------## Locals ####################

A local value assigns a name to an expressions so you can use the name multiple times within a module. It is helpful to avoid repeating the same values or expressions multiple times in a configuration, but if overused they can also make a configuration hard to read . Locals values are not set by the user input or values in terraform files, instead, they are set 'locally' to the configuration .

Ex:

```
locals {
  bucket-name = "${var.layer}-${var.env}-bucket-hydnaresh"
}

resource "aws_s3_bucket" "demo" {
    # bucket = "web-dev-bucket"
    # bucket = "${var.layer}-${var.env}-bucket-hyd"
```

```
  bucket = local.bucket-name
  tags = {
    # Name = "${var.layer}-${var.env}-bucket-hyd"
    Name = local.bucket-name
    Environment = var.env
  }
}
```

------ ### Provisioners ### ----------

Terraform includes the concept of provisioners as a measure of pragmatism, knowing that there will always be certain behaviors that can't be directly represented in Terraform's declarative model.

Provisioners can be used to model specific actions on the local machine or on a remote machine in order to prepare servers or other infrastructure objects for service.

##File Provisioner:

The file provisioner is used to copy files or directories from the machine executing the terraform apply to the newly created resource. The file provisioner can connect to the resource using either ssh or winrm connections.

The file provisioner can upload a complete directory to the remote machine.

```
resource "aws_instance" "web" {
 # ...

 # Copies the myapp.conf file to /etc/myapp.conf
 provisioner "file" {
  source      = "conf/myapp.conf"
  destination = "/etc/myapp.conf"
 }
}
```

##local-exec Provisioner:

The local-exec provisioner invokes a local executable after a resource is created. This invokes a process on the machine running Terraform, not on the resource.

Basically, this provisioner is used when you want to perform some tasks onto your local machine where you have installed the terraform. So local-exec provisioner is never used to perform any task on the remote machine. It will always be used to perform local operations onto your local machine.

```
resource "aws_instance" "web" {
 # ...

 provisioner "local-exec" {
  command = "echo ${self.private_ip} >> private_ips.txt"
 }
}
```

##remote-exec Provisioner:

As the name suggests remote-exec provisioner is always going to work on the remote machine. With the help of this, you can specify the commands of shell scripts that want to execute on the remote machine. The remote-exec provisioner invokes a script on a remote resource after it is created. This can be used to run a configuration management tool, bootstrap into a cluster, etc. It requires a connection and supports both ssh and winrm.

```
resource "aws_instance" "web" {
 # ...

 # Establishes connection to be used by all
 # generic remote provisioners (i.e. file/remote-exec)
 connection {
  type      = "ssh"
  user      = "ubuntu"  # Replace with the appropriate username for your EC2 instance
  # private_key = file("C:/Users/veerababu/.ssh/id_rsa")
  private_key = file("~/.ssh/id_rsa")  #private key path
  host      = self.public_ip
 }
```

```
  provisioner "remote-exec" {
    inline = [
"touch file200",
"echo hello from aws >> file200",
]
  }
}
```

It can be used inside the Terraform resource object and in that case, it will be invoked once the resource is created, or it can be used inside a null resource which is my preferred approach as it separates this non-terraform behavior from the real terraform behavior.

---------- ## Modules ##----------

What is module?

What are Terraform Modules?

Terraform modules are reusable and encapsulated collections of Terraform configurations. They simplify managing resources, making your Terraform code more manageable and scalable. Modules make defining, configuring, and organizing resources modular and consistent while abstracting away their complexity to make Terraform code more scalable and maintainable.

Benefits of Using Terraform Modules

Using Terraform modules brings several advantages to your infrastructure provisioning process:

Reusability: Modules allow you to organize infrastructure resources and configurations into containers you can repurpose across projects and environments. This reuse saves effort and reduces errors significantly.

Abstraction: Modules simplify resource creation and configuration processes for Terraform configuration files, making them more concise and understandable.

Encapsulation: Modules isolate resources and their dependencies, making it more straightforward for you to manage or modify individual pieces of your infrastructure without impacting others or hindering modularity in its codebase. This improves its modularity.

Versioning: Terraform modules can be versioned, making it easier to track changes and update dependencies in an orderly manner. This ensures that changes made do not cause unintended problems in your infrastructure.

Collaboration: Modules allow your team and the wider community to work more collaboratively by sharing them via Terraform Registry or private module repositories - encouraging best practices and standardizing infrastructure configurations.

Terraform Module Examples

Below are three examples demonstrating how to use Terraform modules.

Example 1: AWS VPC Module

Creating an AWS Virtual Private Cloud (VPC) is fundamental for many infrastructure deployments. Instead of defining the VPC configuration repeatedly, we can create a Terraform module for it.

Module Directory Structure:

```
modules/
  vpc/
        main.tf
        variables.tf
```

VPC Module Code (modules/vpc/main.tf):

```
resource "aws_vpc" "example" {
  cidr_block = var.cidr_block
  tags       = { Name = var.name }
}
```

In this module, we define an AWS VPC resource and allow customization of the VPC and name using input variables.

Input Variables (modules/vpc/variables.tf):

```
variable "cidr_block" {
  description = "The CIDR block for the VPC."
```

```
}

variable "name" {
  description = "The name of the VPC."
}
```

The variables.tf file declares the input variables that can be set when the module is used.

Using the VPC Module (main.tf):

```
module "my_vpc" {
  source     = "./modules/vpc"
  cidr_block = "10.0.0.0/16"
  name       = "my-vpc"
}
```

In the main Terraform configuration, we use the module block to include the VPC module. We specify the module's source directory and provide values for the input variables.

Now, you can easily create multiple VPCs with different configurations by reusing this module.

Example 2: AWS EC2 Instance Module

Creating Amazon Elastic Compute Cloud (EC2) instances is another common task in AWS. Let's create a Terraform module for EC2 instances.

Module Directory Structure:

```
modules/
  ec2/
      main.tf
      variables.tf
```

EC2 Instance Module Code (modules/ec2/main.tf):

```
resource "aws_instance" "example" {
```

```
  ami           = var.ami
  instance_type = var.instance_type
  subnet_id     = var.subnet_id
  key_name      = var.key_name

  tags = {
    Name = var.name
  }
}
```

In this module, we define an AWS EC2 instance resource and allow customization of the AMI, instance type, subnet, key name, and name using input variables.

Input Variables (modules/ec2/variables.tf):

```
variable "ami" {
  description = "The AMI ID for the EC2 instance."
}

variable "instance_type" {
  description = "The instance type for the EC2 instance."
}

variable "subnet_id" {
  description = "The subnet ID for the EC2 instance."
}

variable "key_name" {
  description = "Key pair to associate with the EC2 instance."
}

variable "name" {
  description = "The name of the EC2 instance."
```

}

The variables.tf file declares the input variables that can be set when using the module.

Using the EC2 Instance Module (main.tf):

```
module "my_ec2" {
  source        = "./modules/ec2"
  ami           = "ami-12345678"
  instance_type = "t2.micro"
  subnet_id     = "subnet-01234567"
  key_name      = "my-key-pair"
  name          = "my-ec2-instance"
}
```

In the main Terraform configuration, we use the module block to include the EC2 instance module. We specify the module's source directory and provide values for the input variables.

With this module, you can easily create EC2 instances with different configurations across your infrastructure.

These examples demonstrate how Terraform modules promote code reuse, abstraction, and encapsulation. By following similar patterns, you can create modules for various infrastructure components, including databases, load balancers, and networking resources.

if module source is github

- https://github.com/CloudTechDevOps/Terraform/tree/main/day-11-modules_root_child
- #for my github reference here root_module is source reference

---------- # connection # ----------

Connection Block

You can create one or more connection blocks that describe how to access the remote resource. One use case for providing multiple connections is to have an initial provisioner connect as the root user to

set up user accounts and then have subsequent provisioners connect as a user with more limited permissions.

Connection blocks don't take a block label and can be nested within either a resource or a provisioner.

A connection block nested directly within a resource affects all of that resource's provisioners.

Example:

```
connection {
  type     = "ssh"
  user     = "ubuntu"  # Replace with the appropriate username for your EC2 instance
  # private_key = file("C:/Users/veerababu/.ssh/id_rsa")
  private_key = file("~/.ssh/id_rsa")  #private key path
  host     = self.public_ip
 }
```

----------## Output Values ## ----------

Output values make information about your infrastructure available on the command line, and can expose information for other Terraform configurations to use.

Example:

```
output "instance_public_ip" {
  value = aws_instance.test.public_ip
  sensitive = true
}
output "instance_id"{

  value = aws_instance.test.id
}
output "instance_public_dns" {
  value = aws_instance.test.public_dns
```

```
}
output "instance_arn" {
  value = aws_instance.test.arn
}
```

###TERRAFORM WITH CICD

```
pipeline {
  agent any

  stages {
    stage('clone') {
      steps {
        git branch: 'main', url: 'https://github.com/CloudTechDevOps/Terraform_CICD.git'
      }
    }
    stage('init') {
      steps {
        sh 'terraform init'
      }
    }
    stage('apply') {
      steps {
        sh 'terraform apply -auto-approve'
      }
    }
  }
}
```

MAVEN

What is Maven?

Maven is a build automation tool used primarily for Java projects. Maven can also be used to build and manage projects written in C#, Ruby, Scala, and other languages. The Maven project is hosted by The Apache Software Foundation.

##

mavnen advantages

maven is build tool build the project

build means adding dependencies [lib] for the code

we can package the project by passinfg goals

project management

Maven automate the process

we can generate the reports

maven is written on java programming.

its maintainig by Apache Software Foundadtion.

##

Once generate the archetype with sample project

We can se folder structure

```
my-app
|-- pom.xml
`-- src
  |-- main
  |   `-- java
  |       `-- com
  |           `-- mycompany
  |               `-- app
  |                   `-- App.java
  `-- test
      `-- java
          `-- com
              `-- mycompany
                  `-- app
  |-- target
```

target folder Note: after run the goals we can see jar/war file in target folder

TYPES OF ARTIFACTS:

1. JAR: JAVA ARCHIVE : Backend
2. WAR: WEB ARCHIVE : Frontend + backend
3. EAR: ENTERPRISE ARCHIVE : jar + war

build process tools

JAVA : MAVEN

Nodejs : NPM build

PYTHON : GRADLE

.NET : VS CODE

C, C# : MAKE FILE

Ex:

#######Folder structure########

src/main/java -----sorce code required

src/main/resourc ------resources files keep in side here 3rd party files

src/test/java ------unit test cases code keep here

src/test/resource ------resorces file keep heere 3rd party files to testing

Whenever you create maven project we will get pom.xml

POM.XML FILE:

POM means Project Object Model.

pom.xml - Maven configuration file. Controls the build process for the project

this file will have complete info of the project.

Ex: Name, Tools, Version, Snapshot, Dependencies.

Extension is .xml (Extensible Markup Language)

Note: if we want to pass any goals this file must be on project folder.

without this file maven will not pass any goals.

each project we need to have only one pom.xml

multiple project cant use same pom.xml.

pom.xml two entries

1. dpendencies

2. plugins(configuration plug in)

Dependencies : responsible for downloading the required third party drivers libraries from remote to local

Plugin: nothing but different configuartions to run project eX : compile

example maven archetype

--mvn archetype:generate # to generate and see the sample templates

--mvn archetype:generate -DgroupId=com.apple -DartifactId=java_project -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

maven work with two type of repositories

1.Maven remote : https://mvnrepository.com/artifact

2.Mavne Local repo: .M2

############### life cycles ###################

- clean ------project will be cleaned unnessary files
- validate - validate the project is correct and all necessary information is available
- compile - compile the source code of the project (which converst .java to .class in target foolder)
- test - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- package - take the compiled code and package it in its distributable format, such as a JAR.
- verify - run any checks on results of integration tests to ensure quality criteria are met
- install - install the package into the local repository(.m2)
- deploy - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

example source codes :

############################### sample code from login page ####################

login.jsp

<html>

<head>

<title>Login Form</title>

</head>

<body>

<h3> Login here </h3>

<form action="user_login" method="post">

<table style="width: 20%">

<tr>

<td>UserName</td>

```
                <td><input type="text" name="username" /></td>
            </tr>
                <tr>
                <td>Password</td>
                <td><input type="password" name="password" /></td>
             </tr>
        </table>
        <input type="submit" value="Login" /></form>
</body>
</html>
```

################sample code for hello world ##################################

```
<html>
<body>
<h2>Hello World!</h2>
</body>
</html>
```

-------############## JENKINS CICD ##############----------

what is CICD?

In software engineering, CI/CD or CICD is the combined practices of continuous integration and continuous delivery or, less often, continuous deployment. They are sometimes referred to collectively as continuous development or continuous software development

Why Jenkins?

Jenkins is an open-source automation tool for Continuous Integration (CI) and Continuous Deployment (CD).

It is an open-source tool with great community support.

It is easy to install.

It has 1000+ plugins to ease your work. If a plugin does not exist, you can code it and share it with the community.

It is free of cost.

It is built with Java and hence, it is portable to all the major platforms.

Inventor: kohsuke kawaguchi

its a free and open-source tool.

jenkins written on java.

it can automate entire sdlc.

it is owned by sun micro system as hudson.

hudson is paid version.

later oracle brought hudson and make it free.

later hudson was renamed as jenins.

jenkins runs on 8080 port

For amazon linux 2023 AMi type

==========

Login to server with username and switch to root user

- command : sudo su –
- Provide the password

- jenkin supports JAVA 11/17
- sudo dnf update -y --> it will install or updates the patches
- sudo dnf install java-11-amazon-corretto -y --> to install the openjdk11
- sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo --> Install the repo
- if wget command not found.Please install the wget software using this command "yum install wget -y"
- sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key --> Import the required key
- sudo yum install jenkins -y --> install the Jenkins
- sudo systemctl enable jenkins --> Enable the Jenkins
- sudo systemctl start jenkins --> start Jenkins
- sudo systemctl status jenkins --> check the status of jenkins service

TO CONNECT:

- public_ip:8080 (browser)
- cat /var/lib/jenkins/secrets/initialAdminPassword (server)

---paste password on browser -- > installing plugins --- > user details -- > start

WORKSPACE: job output is going to be stored in workspace

- Default: /var/lib/jenkins/workspace

==========

Process for Amazon linux 2 AMI type

==========

Login to server with username and switch to root user

- command : sudo su –
- Provide the password

- sudo yum update -y --> it will install or updates the patches
- sudo amazon-linux-extras install java-openjdk11 -y --> to install the openjdk11
- sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo --> Install the repo
- if wget command not found.Please install the wget software using this command "yum install wget -y"
- sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key --> Import the required key
- sudo yum install jenkins -y --> install the Jenkins
- sudo systemctl enable jenkins --> Enable the Jenkins
- sudo systemctl start jenkins --> start Jenkins
- sudo systemctl status jenkins --> check the status of jenkins service

---

----------Triggers----------

In Jenkins, a build trigger is a mechanism that initiates the execution of a Jenkins job or pipeline. Triggers are used to start builds automatically based on various events or conditions. There are several ways to trigger builds in Jenkins:

Manual Build Trigger:

Build Now: You can manually trigger a build at any time by clicking the “Build Now” or “Build” button on the Jenkins dashboard for a specific job. This is useful for ad-hoc or on-demand builds.

Scheduled Build Trigger: It will schedule as per the schedule:

Build Periodically: You can schedule builds to run at specific intervals using the “Build periodically” option in the job configuration. You can use cron syntax to define the schedule. For example, to run a build every night at 2:00 AM, you can use the cron expression 0 2 * * *.

SCM (Source Code Management) Trigger:

Poll SCM: If you're using a version control system (e.g., Git, Subversion), you can configure the job to poll the repository for changes. When changes are detected, Jenkins will automatically trigger a build. This is known as the "Poll SCM" build trigger.

Webhooks Trigger:

Webhooks: Many version control systems and external services support webhooks, which allow them to notify Jenkins when code changes occur. Jenkins can listen for these webhook notifications and trigger builds in response.

Dependency Build Trigger:

Build after other projects are built: You can configure a job to be triggered after one or more upstream jobs have been built successfully. This is useful for creating build pipelines or ensuring that certain prerequisites are met before starting a build.

Parameterized Build Trigger:

ex: Job1------job2-----job3

here job1 upstream is job2

job2 downstream is Job1

job2 upstream is Job3

job3 downstream is job2

--so whenever jb1 triggers after that job2 will trigeere agter that job3 will trigger

--if any one downstream job fail means necxt upstream job will not success

--Trigger builds with parameters: You can set up parameterized builds where a build is triggered with specific parameter values. This is useful for customizing builds based on input parameters.

Pipeline Trigger:

Pipeline Trigger: If you're using Jenkins Pipelines (defined in a Jenkinsfile), you can use various triggers within the pipeline script itself. For example, you can set up a webhook trigger, a schedule trigger, or a manual input trigger as part of your pipeline script.

Remote API Trigger:

Jenkins has numerous plugins available that can provide additional trigger mechanisms. For example, the "GitHub Webhook" plugin allows Jenkins to listen to GitHub events and trigger builds accordingly.

The choice of build trigger method depends on your specific use case and requirements.

----------pipeline types----------

1. Declaritive
2. Groovy scripted

Declaritive

Basic Example:

(PASSS:shortcut)

---

single stage:example

```
pipeline {
   agent any

   stages {
      stage('first') {
         steps {
            echo 'Hello first stage'
         }
      }
   }
}
```

---

multi stage example:

```
pipeline {
   agent any
```

```
  stages {
    stage('first') {
      steps {
        echo 'Hello first stage'
      }
    }

    stage('second') {
      steps {
        echo 'Hello second stage'
      }
    }
  }
```

example-1 to build maven by using pipleine (CI) multi stage

```
pipeline {
  agent any

  stages {
    stage('scm') {
      steps {
        git branch: 'main', credentialsId: 'terraform', url:
'https://github.com/nareshdevopscloud/devops_maven.git'
      }
    }
    stage('clean') {
      steps {
        sh "mvn clean"
      }
    }
    stage('install') {
      steps {
```

```
            sh "mvn install"
        }
    }

  }
}
```

example-2

PIPELINE AS A CODE: multiple commands or action inside a single stage.

```
pipeline {
  agent any

  stages {
    stage('one') {
      steps {
        git branch: 'main', credentialsId: 'terraform', url:
'https://github.com/nareshdevopscloud/devops_maven.git'
        sh 'mvn compile'
        sh 'mvn test'
        sh 'mvn clean package'
      }
    }
  }
}
```

Groovy

Basic example:

```
 node{
  stage{"first") {
```

```
  sh file-1
  }
  stage{"second") {
  sh file-2
  }
}
```

----------CI JOB using Groovy----------

```
node {
  stage('git') {
    git branch: 'main', credentialsId: 'terraform', url:
'https://github.com/nareshdevopscloud/devops_maven.git'
  }
  stage('clean') {
    sh "mvn clean"
    }

  stage('install') {
    sh "mvn install"

  }
}
```

Note Using Jenknins deployinmg the Terraform IaC(Infrastructure as Code)

---------- terraform installation----------

```
sudo yum install -y yum-utils shadow-utils
sudo yum-config-manager --add-repo
https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
```

```
sudo yum -y install terraform
```

Deploy Terraform script by using jenkins

----declaritive--------

```
pipeline {
  agent any

  stages {
    stage('checkout') {
      steps {
        git branch: 'main', credentialsId: 'terraform', url:
'https://github.com/nareshdevopscloud/terraform-jenkins.git'
      }
    }
    stage('init') {
      steps {
        sh "terraform init -reconfigure"
      }
    }
    stage('plan') {
      steps {
        sh "terraform plan"
      }
    }
    stage('action') {
      steps {
```

```
            sh "terraform ${action} --auto-approve"
        }
      }
   }
}
```

----------groovy----------

```
node {
   stage('checkout') {
      git branch: 'main', credentialsId: 'terraform', url:
'https://github.com/nareshdevopscloud/terraform-jenkins.git'
   }
   stage('init') {
      sh "terraform init -reconfigure"

      }
   stage('action') {
      sh "terraform ${action} --auto-approve"

      }
}
```

---

approach 3:

Note:

#created jenkins file uploded into github

#we can run job by calling above jenkins file in git hub

- To Implement

- give GITHUB url and in defination section we need to "sleect pipeline script from scm" option and build the job

#Jenkins file

```
pipeline {
  agent any

  stages {
    stage('Checkout Code') {
      steps {
        checkout scm
      }
    }

    stage ("terraform init") {
      steps {
        sh ("terraform init -reconfigure")
      }
    }

    stage ("plan") {
      steps {
        sh ('terraform plan')
      }
    }

    stage (" Action") {
      steps {
```

```
            echo "Terraform action is --> ${action}"
            sh ('terraform ${action} --auto-approve')
        }
    }
}
```

----------jenkins backup and versions ----------

T2.medium

```
#!/bin/bash
sudo dnf install java-17-amazon-corretto-devel -y
sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
sudo yum install jenkins -y
sudo systemctl enable jenkins
sudo systemctl start jenkins
```

In ec2instance:

```
sudo su -
systemctl status jenkins
open jenkins in browser [ public ip:8080 ]
cat /var/lib/jenkins/secrets/initialAdminPassword and pastein browser and set it up
Create a pipeline job with Hello World sample
Apply>Save
Build it
```

JENKINS BACKUP

Goto Dashboard> Manage Jenkins> Plugins> Available Plugins> ThinBackup [Install]

Goto Manage Jenkins >Scroll down to Tools and Actions and click on ThinBackup>Settings

Backup directory : /var/lib/jenkins/backup [Creating a new dir in Jenkins folder]

Backup schedule for full backups : H * * * * [Every hour]

Backup schedule for differential backups : H * * * * [Every hour]

Max number of backup sets : 3

Save

Click on Backup now

Goto ec2instance

cd /var/lib/jenkins/backup

ls

We see a folder [FULL-2024-04-13_07-00] is created

Now go back to Jenkins browser >Dashboard> Delete pipeline job

Goto Manage Jenkins > ThinBackup > Restore

systemctl restart jenkins

log in jenkins again and we see that pipeline job is restored

---

UPDATE JENKINS VERSION

Goto dashboard > Bottom right Jenkins 2.44.0 version is shown

https://www.jenkins.io/download/ and see the latest version of jenkins is 2.453

Goto ec2instance

sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat/jenkins.repo

sudo rpm --import https://pkg.jenkins.io/redhat/jenkins.io-2023.key

sudo yum upgrade

sudo systemctl restart jenkins

login jenkins browser again, we see its updated to 2.453

----------CD----------

- After builld the war file we have to deploy in any application serever like "TOMCAT"
- Create one Ec2 instance if (amazon linux 2) ----note : if you choose linux 2023 commands will different)

----Need to install java

--- for java 11

- sudo amazon-linux-extras install java-openjdk11

---for java 17

- sudo yum install java-17-amazon-corretto.x86_64

Tomcat

----Install tomcat

- wget url https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.83/bin/apache-tomcat-9.0.83.tar.gz
- untar tar -xvzf <apachetomcat-tarfile>
- Rename it if required   mv <apachetomcat-tarfile> tomcat

---tomcat/bin -----

- sh startup.sh   --to start
- sh shutdown.sh -- to stop
- ----to change port number ------ /conf/server.xml
- war file should place in ----------/tomcat/webapps

- custom manager app tomcat --------find / -name context.xml

----after finding the file need to open with vi

- example1 -- vi /opt/tomcat/webapps/manager/META-INF/context.xml

----------add comment it below two line

<!-- -->

- ex: <!-- <Valve className="org.apache.catalina.valves.RemoteAddrValve"
- allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->

--------- 2.vi /opt/tomcat/webapps/host-manager/META-INF/context.xml

- <!--<Valve className="org.apache.catalina.valves.RemoteAddrValve"
- allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->

- username and password to login
- cd tomcat/config/tomcat-user.xml
- vi tomcat-users.xml and add custom username password
- ADD BELOW CONTENT

<role rolename="manager-gui"/>

<role rolename="manager-script"/>

<role rolename="manager-jmx"/>

<role rolename="manager-status"/>

<user username="admin" password="admin" roles="manager-gui, manager-script, manager-jmx, manager-status"/>

<user username="deployer" password="deployer" roles="manager-script"/>

<user username="tomcat" password="s3cret" roles="manager-gui"/>

Maven project Job: CICD deployer on Tomcat

-----step-1 need to configure Maven and jenkins paths in global tool configuration

- manaje jenkins---> tools---> add path for both java and maven

-------step-2 install required plugin

- Dashboard---> Manaje jenkins ----> plugins ---Available plugin

maven

Deployment container

------step-3:need to configure credentials --username : deployer password deployer :--- reference from (<user username="deployer" password="deployer" roles="manager-script"/>)

- Dashboard ---> Manage Jenkins -- > Credentials --- > System ---> Global credentials (unrestricted) --> add credentials

-----step-4 create maven job and give git url for source code to build

------step-5 give build actions goals like test, install etc..

-------step-6 In postbuild actions select war/ear file to give path of War --path **/*.war

-------step-7 we need to add container (tomcat) we have to give configured credentials of tomcat and give url of tomcat

webhook

http://34.228.189.137:8080/github-webhook/

we have to give jenkins URL as payload url

"CI/CD----Maven job---Javaproject"

####pipelinejob:example

```
pipeline {
  agent any

  stages {
    stage('stage-1') {
      steps {
        git branch: 'main', credentialsId: 'terraform', url:
'https://github.com/nareshdevopscloud/project-maven-jenkins-CI-CD.git'
      }
    }

    stage('clean') {
      steps {
       sh  'mvn clean'
      }
    }

    stage('test') {
      steps {
       sh  'mvn test'
```

```
        }
      }
      stage('install') {
        steps {
          sh  'mvn install'
        }
      }

      stage('deployment') {
        steps {

          deploy adapters: [tomcat9(credentialsId: 'tomcat', path: '', url: 'http://100.26.194.72:8080/')],
contextPath: null, war: 'webapp/target/*.war'

        }
      }

  }
}
```

==========

DOCKER

==========

MONOLITHIC: Deploying monolithic applications is more straightforward than deploying microservices. Developers install the entire application code base and dependencies in a single environment.

MICRO SERVICES: Microservices are deployed using VM or Containers. Containers are the preferred deployment route for microservices as containers are lighter, portable, and modular. The microservice code is packaged into a container image and deployed as a container service.

multiple services are deployed on multiple servers with multiple databases.

note: if number odf application are more will go with microservices concept (docker and kubernetes)

BASED ON USERS AND APP COMPLEXITY WE NEED TO SELECT THE ARCHITECTURE.

FACTORS AFFECTIONG FOR USING MICRO SERVICES:

FLEXIBLE

COST

MAINTAINANCE

EASY CONTROL

----- Docker overview ----------##

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

DOCKER IMAGE:

Docker images are read-only templates that contain instructions for creating a container. A Docker image is a snapshot or blueprint of the libraries and dependencies required inside a container for an application to run

CONTAINERS:

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime

its same as a server/vm.

it will not have any operating system.

(Ec2 server=AMI, CONTAINER=IMAGE)

os will be maanged in image

-----docker install process -------

sudo amazon-linux-extras install docker # if linux 2

sudo yum install docker -y # If linux 20203

sudo systemctl start docker

sudo systemctl status docker

Note : By default Docker works with the root user and other users can only access to Docker with sudo commands. However, we can bypass the sudo commands by creating a new group with the name docker and add ec2_user.

#First let's create the docker group

sudo groupadd docker (optional if group is not created)

#Now let's add ec2-user to docker group

sudo usermod -a -G docker ec2-user

#In order to enable the changes, run the following command

newgrp docker

sudo chmod 666 /var/run/docker.sock # to give access docker demon to run docker server

docker --version to check docker version

#If you want to see an extended version of the version details, such as the API version, Go version, and Engine Version, use the version command without dashes. give below command

docker version

----commands:----

---to pull base images from public docker repository

docker pull image <imagename>

docker pull nginx or ubuntu

- docker inspect image nginx ### to check images details
- docker images # to check list of images
- docker run -it imagename /bin/bash --will enter into the container interact terminal
- docker run -dt imagename -we are not enter into the container detach terminal
- docker run -dt --name <name> <imagename>

-------(to give csutome name --name)

- docker ps ## to check running containers
- docker ps -a ## to check both running and stopped containers

To login container

→ docker exec -it <continername or continerid> /bin/bash

1.if you want to come out from connainer without stop give "ctrl+pq"

2.if you give "exit" container also will stop

→ ps -ef --to know how many processors runing if it is in vm many process we can see but it is in container onle few becuase its light weight

- ➔ ps -ef | wc -l #to know number of processors request running backend

how to start container#####

- docker start <containerid>

  or

- docker start <container name>

how to stop container#####

- docker stop <containerid>

  or

- docker stop <container name>

docker kill complete terminated

- docker kill <containername>
- docker pause cont_name : to pause container
- docker unpause cont_name: to unpause container
- docker inspect cont_name: to get complete info of a container

danger commands##

- docker system prune -a to remove all images
- docker container prune -- delete all stoped containers
- docker rm -f $(docker ps -a -q) ---delete all runnig containers Note: please dont use danger command
- docker rm <containername or continerid> ### to remover stopped container
- docker rmi <imagename> ##to delete image
- docker rm -f <continer id> delete runing continer

--Enabeling port--

- docker run -p <HOST_PORT>:<CONTAINER:PORT> IMAGE_NAME
- docker run -dt -p 3000:3000 --name project-2 saturday #along port expose

==========

-------- pull jenkins----------

- docker pull jenkins/Jenkins
- docker run -dt -p <local host port>:<container run port> --name <name of the container> <name of the image>
- docker run -dt -p 8081:8080 --name jenkins container jenkins/jenkin
- ➔ access the jenkins with public ip and port number (8081)

==========

----------# Docker file run commands #----------

Dockerfile is a simple text file that consists of instructions to build Docker images.

Docker file instruiction

- ➔ From : to pull base image
- ➔ Run : to excutive commands
- ➔ CMD : To Provide defaults for an executing container
- ➔ Entrypoint : To configure a container that will run as an container
- ➔ Workdir : to set working directory
- ➔ Copy: files form local to the container
- ➔ Add: TO copy files and loader file but little advance add command we can add url also
- ➔ Expose : informs docker that the container listen on the specified network port runtime
- ➔ Env: To set env variabels

below command to run docker file

- docker build -t check .
- docker run -p <HOST_PORT>:<CONTAINER:PORT> IMAGE_NAME
- docker run -dit --name demo -p 8080:80 check
- ➔ ex: docker run -dt -p 3000:3000 --name project-2 saturday

Ex:1 docker pull httpd (readymade)

FROM httpd:2.4

COPY . /public-html/ /usr/local/apache2/htdocs/ # copy present directory files into destination patha tha is /usr/local/apache2/htdocs/

Ex:2 httpd with ubuntu (impliment from scratch)

FROM ubuntu #pull ubuntu base image from docker hub

RUN apt update

RUN apt-get -y update

RUN apt-get -y install apache2 #by using RUN instrcution install apache2

COPY index.html /var/www/html #copying index.html file into /var/www/html

EXPOSE 80 #expose default port number for application

CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"] #Run apache2 srever while running container from image

Note:

-D FOREGROUND This is not a docker command this is Apache server argument which is used to run the web server in the background. If we do not use this argument the server will start and then it will stop

Ex:3 httpd with centos

FROM centos:7

RUN yum update -y && yum install -y httpd

COPY index.html /var/www/html/

EXPOSE 80

#httpdserver

CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]

==========

IMAGE PUSH

==========

######### docker push to Docker private repository #########

- ➔ docker login first
- ■ docker tag <image> dockerusername/<image>
- ■ docker push <dockerusername>/<image>
- ■ docker pull <dockerusername>/<image>

==========

######### docker push to AWS ECR #########

- ■ Retrieve an authentication token and authenticate your Docker client to your registry.

Use the AWS CLI:

- ■ aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin 992382358200.dkr.ecr.ap-south-1.amazonaws.com

---Note: If you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.

----Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions here . You can skip this step if your image is already built:

- ■ docker build -t maventomcat .

-----After the build completes, tag your image so you can push the image to this repository:

- ■ docker tag maventomcat:latest 992382358200.dkr.ecr.ap-south-1.amazonaws.com/maventomcat:latest
- ■ Run the following command to push this image to your newly created AWS repository:
- ■ docker push 992382358200.dkr.ecr.ap-south-1.amazonaws.com/maventomcat:latest

==========

----- ## Docker file single stage and multi stage ##--------

==========

scenario :1 single stage after run mavne manually, deploy war file on tomcat webapp by using docker file

```
FROM tomcat:latest

COPY webapp/target/webapp.war /usr/local/tomcat/webapps/webapp.war

RUN cp -R  /usr/local/tomcat/webapps.dist/*  /usr/local/tomcat/webapps
```

```
# scenario :2 multi satge --run Maven Image by using docker file and deploy on tomcat webapp

FROM maven:3.8.4-eclipse-temurin-17 AS build

RUN mkdir /app

WORKDIR /app

COPY . .

RUN mvn package


FROM tomcat:latest

COPY --from=build /app/webapp/target/webapp.war /usr/local/tomcat/webapps/webapp.war # copying files fromsta ge -1 docker file maven into tomcat path

RUN cp -R  /usr/local/tomcat/webapps.dist/*  /usr/local/tomcat/webapps # adding dependecnes to run base tomcat page
```

```
# scenario :3 multi stage from ubuntu scratch by using docker file


FROM ubuntu:latest as builder

RUN apt-get update && \

    apt-get install -y openjdk-8-jdk wget unzip


ARG MAVEN_VERSION=3.9.6

RUN wget https://dlcdn.apache.org/maven/maven-3/3.9.6/binaries/apache-maven-3.9.6-bin.tar.gz && \

   tar -zxvf apache-maven-${MAVEN_VERSION}-bin.tar.gz && \

   rm apache-maven-${MAVEN_VERSION}-bin.tar.gz && \

   mv apache-maven-${MAVEN_VERSION} /usr/lib/maven
```

```
ENV MAVEN_HOME /usr/lib/maven

ENV MAVEN_CONFIG "$USER_HOME_DIR/.m2"

ENV PATH=$MAVEN_HOME/bin:$PATH

RUN mkdir -p /app

COPY . /app

 WORKDIR /app

 RUN mvn install

FROM tomcat:latest

COPY --from=builder /app/webapp/target/webapp.war /usr/local/tomcat/webapps/webapp.war

RUN cp -R  /usr/local/tomcat/webapps.dist/*  /usr/local/tomcat/webapps
```

==========

Custom Docker file name

if we create multiple docker files like Dockerfile and Dockerfile1

- ➔ Below command example to run
- docker build -f <Dockerfilename> -t <imagename>.
- docker build -f Dockerfile1 -t image .

==========

RUN DOCKER FILE DIRECTLY TAKING SOURCE FROM GITHUB

- docker build -t <imagename> <github projecturl>

Example:

- docker build -t mvntomgit https://github.com/CloudTechDevOps/project-1-maven-jenkins-CICD-docker-eks-.git

######### CMD vs Entrypoint ###

```
FROM centos:centos7

CMD ["yum", "install", "-y", "httpd"]
```

------note: condition:1 after run command docker run image ----it will install httpd

------note: condition -2 if we run docker run image yum install -y git it will overwrite new command it will download git only and ignores httpd

-------but i need only allow required attribute change like httpd, git ex : docker run image httpd or git instead of docker run image yum install -y httpd or git

Entrypoint

FROM centos:centos7

ENTRYPOINT ["yum", "install", "-y", "git"]

note: condition:1 after run command docker run image ----it will install git

condition -2 if we run docker run image httpd ----it will install git and httpd also

FROM centos:centos7

ENTRYPOINT ["yum", "install", "-y"]

CMD ["git"]

Docker network

- ➔ bridge
- ➔ host
- ➔ none
- ■ docker run -dt --name container1 ubuntu
- ■ docker network ls
- ➔ create two containers
- ➔ container1
- ➔ container2
- ■ docker inspect <containername>

----container1: 172.17.0.2

----container2: 172.17.0.3

----------Note:defualt network is bridge

- ➔ login to container
- ■ docker exec -it <containernmae> /bin/bash

##--- need to install ping libraries by using below command (if ping command will not work)

- ■ apt-get update -y
- ■ apt install iputils-ping

--------check both containers ips and try to access each other

host type

Docker network host, also known as Docker host networking, is a networking mode in which a Docker container shares its network namespace with the host machine. The application inside the container can be accessed using a port at the host's IP address

- ■ docker run -dt --name <container> --network host <image>
- ➔ ex:docker run -d --network host nginx

#None type

none network in docker means when you don't want any network interface for your container. If you want to completely disable the networking on a container, you can use the --network none flag when starting the container.

- docker run -dt --name <container> --network None ubuntu

#to remove network : docker network rm networkname

create custom network # (optional)

- docker network create <networkname> # to maitain private connection one container from another container

- docker run -dt --name <container-name> --network <created-networkname> <image-name>

➔ ex: docker run -dt --name container3 --network private ubuntu

##

----------Docker volumes ----------

Volumes are a mechanism for storing data outside containers. All volumes are managed by Docker and stored in a dedicated directory on your host, usually /var/lib/docker/volumes for Linux systems.

to create volume

- docker volume create <volume-name>
- docker volume ls ##to check list of volumes
- docker inspect volume <volumename>
- docker volume rm <volumename> ## to remove volume

--------to check created volume path

➔ cd /var/lib/docker/volumes

----run container from ubuntu image along with created volume

- docker run -ti --name=container1 -v volume1:/volume1 ubuntu
- docker run -ti --name=container1 -v wed:/wed ubuntu
- docker run -ti --name=container2 --volumes-from container1 ubuntu

→echo "Share this file between containers">/volume1/Example.txt

--------docker compose installation ------------

- sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

→ sudo chmod +x /usr/local/bin/docker-compose

- - - - - - - - -- - - - - - - - -- - - - - -- - - - - -

→docker-compose version

```
services:
 mydb:
  environment:
   MYSQL_ROOT_PASSWORD: test
  image: mysql
 mysite:
  image: wordpress
  links:
  - mydb:site
  ports:
  - published: 8080
   target: 80
version: '3'
```

filename : docker-compose.yaml

- docker-compose -f filename.yaml up ###need to give this command for custom file name
- docker-compose up -----#to run docker compose
- docker-compose ps -----#to see containers status

############ list of commands on Docker compose ######

Docker-Compose commands.

- ➔ To pull docker images.
- docker-compose pull
- ➔ To create all containers using docker-compose file.
- docker-compose up
- ➔ To create all containers using docker-compose file with detached mode.
- docker-compose up -d
- ➔ To stop all running containers with docker-compose.
- docker-compose stop

----------To view the config file.

- docker-compose config

--→ To remove all stopped containers.

→docker-compose rm

- ➔ To view the logs of all containers.
- docker-compose logs
- ➔ To view all images.
- docker-compose images

➔ To view all contaiers created by docker-compose file.

- docker-compose ps

➔ To restart containers with docker-compose file.

- docker-compose restart

----------jenkins with docker ----------

jenkins Docker

Pre-requistes:

1. Jenkins is up and running

2. Docker installed on Jenkins instance. Click here to for integrating Docker and Jenkins

3. Docker and Docker pipelines plug-in are installed

4. Repo created in ECR, Click here to know how to do that.

plugins :: docker ,ecr

Jenkins pipeline to automate the following:

- Automating builds

- Automating Docker image builds

- Automating Docker image upload into AWS ECR

- Automating Docker container provisioning

```
pipeline {
  agent any
  options {
    skipStagesAfterUnstable()
  }
  stages {
    stage('Clone repository') {
      steps {
```

```
        git branch: 'main', credentialsId: 'git', url: 'https://github.com/nareshdevopscloud/jenkins-dockerbuild.git'
      }
    }
  stage('maven install') {
      steps {
          sh 'mvn install'
    }
    stage('Build') {
      steps {
        sh '''docker build -t test2 .
            docker rm -f $(docker ps -aq)
            docker run -d -p 8081:80 test
            docker system prune -a
            '''

      }
    }

    stage('Deploy') {
      steps {
        sh '''
        aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 987075663466.dkr.ecr.us-east-1.amazonaws.com
        docker tag test:latest 987075663466.dkr.ecr.us-east-1.amazonaws.com/test2:latest
        docker push 987075663466.dkr.ecr.us-east-1.amazonaws.com/test2:latest
        '''
      }
      }
    }
  }
```

---------- Kubernetes----------

Kubernetes is a container orchestration system that was initially designed by Google to help scale

containerized applications in the cloud. Kubernetes can manage the lifecycle of containers, creating and

destroying them depending on the needs of the application, as well as providing a host of other features.

Kubernetes has become one of the most discussed concepts in cloud-based application development,

and the rise of Kubernetes signals a shift in the way that applications are developed and deployed.

In general, Kubernetes is formed by a cluster of servers, called Nodes, each running Kubernetes agent

processes and communicating with one another. The Master Node contains a collection of processes

called the control plane that helps enact and maintain the desired state of the Kubernetes cluster, while

Worker Nodes are responsible for running the containers that form your applications and services.

A Kubernetes cluster is composed of two separate planes:

Kubernetes control plane—manages Kubernetes clusters and the workloads running on them. Include components like the API Server, Scheduler, and Controller Manager.

Kubernetes Workernode that can run containerized workloads. Each node is managed by the kubelet, an agent that receives commands from the control plane. --

==========

Master Node or Control Node compomnents

API Server -------

Provides an API that serves as the front end of a Kubernetes control plane. It is responsible for handling external and internal requests—determining whether a request is valid and then processing it. The API can be accessed via the kubectl command-line interface or other tools like kubeadm, and via REST calls.

Scheduler :------

This component is responsible for scheduling pods on specific nodes according to automated workflows and user defined conditions, which can include resource requests

etcd :-----

A key-value database that contains data about your cluster state and configuration. Etcd is fault tolerant and distributed.

Controller:-----

It receives information about the current state of the cluster and objects within it, and sends instructions to move the cluster towards the cluster operator's desired state.

==========

Worker Node Components

kubelet: ----

Each node contains a kubelet, which is a small application that can communicate with the Kubernetes control plane. The kubelet is responsible for ensuring that containers specified in pod configuration are running on a specific node, and manages their lifecycle.. It executes the actions commanded by your control plane

Kube Proxy :---

All compute nodes contain kube-proxy, a network proxy that facilitates Kubernetes networking services. It handles all network communications outside and inside the cluster, forwarding traffic or replying on the packet filtering layer of the operating system.

Container Runtime Engine :---

Each node comes with a container runtime engine, which is responsible for running containers. Docker is a popular container runtime engine, but Kubernetes supports other runtimes that are compliant with Open Container Initiative, including CRI-O and rkt.

==========

Core Components

#Nodes: ----

Nodes are physical or virtual machines that can run pods as part of a Kubernetes cluster. A cluster can scale up to 5000 nodes. To scale a cluster's capacity, you can add more nodes.

#POD:--------

Pods—pods are the smallest unit provided by Kubernetes to manage containerized workloads. A pod typically includes several containers, which together form a functional unit or microservice.

==========

What is Minikube?

Minikube is a tool that sets up a Kubernetes environment on a local PC or laptop

minikube quickly sets up a local Kubernetes cluster on macOS, Linux, and Windows. We proudly focus on helping application developers and new Kubernetes users.

---Installation Process Minikube ---

- curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
- sudo install minikube-linux-amd64 /usr/local/bin/minikube
- to start minikube---minikube start
- to check status ----minikube status
- to update --- minikube update-context

What is Kubectl?

kubectl is the Kubernetes-specific command line tool that lets you communicate and control Kubernetes clusters. Whether you're creating, managing, or deleting resources on your Kubernetes platform, kubectl is an essential tool.

----------kubectl Installation----------

- 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 #Make the kubectl binary executable
- sudo mv ./kubectl /usr/local/bin/kubectl

-----EKSCTL installation----

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

EKS cluster Setup Process

Create an IAM Role and attache it to EC2 instance

`Note: create IAM user with programmatic access if your bootstrap system is outside of AWS`

IAM user should have access to

IAM

EC2

VPC

CloudFormation

Create your cluster and nodes

```sh
eksctl create cluster --name cluster-name  \
--region region-name \
--node-type instance-type \
--nodes-min 2 \
--nodes-max 2 \
--zones <AZ-1>,<AZ-2>
```

example:

```
eksctl create cluster --name headless \
  --region ap-south-1 \
--node-type t2.small \
```

after create cluster to update it

- aws eks update-kubeconfig --region <region> --name <cluster name>
- ➔ ex: aws eks update-kubeconfig --region ap-south-1 --name naresh

------To delete the EKS clsuter

- eksctl delete cluster naresh --region ap-south-1

----------Kubernetes workloads----------

Imperative:

- kubectl run pod --image nginx

---------- pods ----------

- kubect apply -f pod <.yaml>
- kubectl get pods
- kubectl get pods -o wide # to see the
- kubectl delete pod <pod name>
- kubectl describe pods <name of the pod> # to know more details about pod

- kubectl exec <pod-name> -it -- /bin/sh

---Pod.yaml---

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp
 labels:
    app: webapp
    type: front-end
spec:
 containers:
 - name: nginx-container
  image: nginx
```

==========

#Kuberentes Service#:

-→Kubernetes, a Service is a method for exposing a network application that is running as one or more Pods in your cluster.

→ git hub link for dervice files : https://github.com/CloudTechDevOps/Kubernetes/tree/main/day-3-services

--Types:

-→ ClusterIP : This is the default type for service in Kubernetes.

As indicated by its name, this is just an address that can be used inside the cluster.

- NodePort: A NodePort differs from the ClusterIP in the sense that it exposes a port in each Node.

When a NodePort is created, kube-proxy exposes a port in the range 30000-32767:

- LoadBlancer :This service type creates load balancers in various Cloud providers like AWS, GCP, Azure, etc., to expose our application to the Internet.

##Kubernetes has several port configurations for Services:

#Port: the port on which the service is exposed. Other pods can communicate with it via this port.

#TargetPort: the actual port on which your container is deployed. The service sends requests to this port and the pod container must listen to the same port.

#NodePort: exposes a service externally to the cluster. So the application can be accessed via this port externally. By default, it's automatically assigned during deployment.

==========

Horizonal Pod Auto Scaling

- In Kubernetes, a Horizontal Pod Autoscaler automatically updates a workload resource (such as a Deployment or StatefulSet), with the aim of automatically scaling the workload to match demand.
- Horizontal scaling means that the response to increased load is to deploy more Pods. This is different from vertical scaling, which for Kubernetes would mean assigning more resources (for example: memory or CPU) to the Pods that are already running for the workload.
- If the load decreases, and the number of Pods is above the configured minimum, the HorizontalPodAutoscaler instructs the workload resource (the Deployment, StatefulSet, or other similar resource) to scale back down.
- ➔ git hub link for hpa files : "https://github.com/CloudTechDevOps/Kubernetes/tree/main/day-4-horizonalScaling"

==========

Metric Server deployment

- The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster, and it isn't deployed by default in Amazon EKS clusters.
- we need to deploy by following process

#Deploy the Metrics Server with the following command:

- kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
- ➔ Verify that the metrics-server deployment is running the desired number of Pods with the following command.
- kubectl get deployment metrics-server -n kube-system
- kubectl top pod # to see pod load
- kubectl top node # to see node load

----------ingress----------

What is an Ingress?

➔ In Kubernetes, an Ingress is an object that allows access to your Kubernetes services from outside the Kubernetes cluster. You configure access by creating a collection of rules that define which inbound connections reach which services.

- kubectl create namespace ingress-nginx # create name space for ingress-nginx
- kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.2.1/deploy/static/provider/cloud/deploy.yaml ----to install ingress controller yaml

Create Below deployment file git hub link
https://github.com/CloudTechDevOps/Kubernetes/tree/main/day-5-ingress

- create deployemnt file for path-1
- create deployment file for path-2
- create ingress reosurce file

Note:

-------after deploy the above files just run below command

- kubectl get ingress

------we are able to seec load blancer link and acces by giving path along

---------- RBAC process ----------

- aws configure --profile IAMuser
- AKIA6ODUZCK4H2GTR5JT
- 9AYwm23X/gNFyl73B86kMxJDJNWvj5uIKpl1MO9X
- aws eks update-kubeconfig --name naresh --profile IAMuser #user
- aws eks update-kubeconfig --name naresh #for defulat one

➔ step-1 user need to create and generate keys
➔ step-2 create kuberenetes Role
➔ step-3 cretae kuberentes role binding to bind role and group
➔ step-4 add usr arn into config map file.

Role

==========

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

namespace: default

name: developer-role

rules:

```
 - apiGroups: [""] # "" indicates the core API group ["apps"]
   resources: ["ConfigMap"]
   verbs: ["get", "list"]
 - apiGroups: [""] # "" indicates the core API group ["apps"]
   resources: ["pods"]
   verbs: ["get", "list",]
 - apiGroups: ["apps"]
   resources: ["deployments"]
   verbs: ["get", "list"]


================================================
# Role Binding to map role and group
==========================================
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
 name: read-pods
 namespace: default
subjects:
 - kind: Group
   name: "developer"
   apiGroup: rbac.authorization.k8s.io
roleRef:
 kind: Role
 name: developer-role
 apiGroup: rbac.authorization.k8s.io


#edit command and add belwow user details
```

- kubectl edit cm aws-auth -n kube-system

```
#aws auth config
```

==========

- userarn: arn:aws:iam::992382358200:user/eks

  username: eks

  groups:

  - developer

==========

- kubectl get rb
- kubectl get rolebinding
- kubectl api-resources

to add user into aws-auth file

- kubectl edit cm aws-auth -n kube-system
- kubectl get cm -n kube-system

Please edit the object below. Lines beginning with a '#' will be ignored,

and an empty file will abort the edit. If an error occurs while saving this file will be

reopened with the relevant failures.

----for example below reference ------

apiVersion: v1

data:

 mapRoles: |

  - groups:

    - system:bootstrappers

    - system:nodes

    rolearn: arn:aws:iam::992382358200:role/eksctl-naresh-nodegroup-ng-bbb93ed-NodeInstanceRole-9GWNpfucPXRt

    username: system:node:{{EC2PrivateDNSName}}

 mapUsers: |

  - userarn: arn:aws:iam::992382358200:user/eks

    username: eks

```
    groups:
    - developer
kind: ConfigMap
metadata:
 creationTimestamp: "2024-03-22T02:22:59Z"
 name: aws-auth
 namespace: kube-system
 resourceVersion: "344678"
 uid: 8167447c-eb81-4108-8653-690369d98c4f
```

==========

schedulers

schedule

Scheduling overview

- A scheduler watches for newly created Pods that have no Node assigned. For every Pod that the scheduler discovers, the scheduler becomes responsible for finding the best Node for that Pod to run on. The scheduler reaches this placement decision taking into account the scheduling principles described below.

1. Node Selector

2. Nodeaffinity

3. Daemonset

4. Taint and Toleration

1.Nodeselector

NodeSelector is the simplest recommended form of node selection constraint. You can add the nodeSelector field to your Pod specification and specify the node labels you want the target node to have. Kubernetes only schedules the Pod onto nodes that have each of the labels you specify.

to label the node

- kubectl label nodes <node-name> <label-key>=<label-value>

Example:

➔ kubectl label nodes ip-192-168-43-22.ap-south-1.compute.internal size=large

to unlabel

- kubectl label nodes <node-name> <label-key>=<label-value>-
- kubectl label nodes ip-192-168-43-22.ap-south-1.compute.internal size=large-

to list

- kubectl get nodes --show-labels

----------Labels are casesensitve

Ex:Below Pod node selector

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp
 labels:
    app: webapp
    type: front-end
spec:
 containers:
 - name: nginx-container
  image: nginx
 nodeSelector:
  size: Large
```

Note:if pod label is matching it will schedule on to labled node only

→ if my pod label is not matching it will not schedule on any node always trying to schedule on labeld node only otherwise it will not scheduled

==========

2.Node affinity

→ Node affinity is conceptually similar to nodeSelector, allowing you to constrain which nodes your Pod can be scheduled on based on node labels. There are two types of node affinity:

→ A.requiredDuringSchedulingIgnoredDuringExecution: The scheduler can't schedule the Pod unless the rule is met. This functions like nodeSelector, but with a more expressive syntax.

→ B.preferredDuringSchedulingIgnoredDuringExecution: The scheduler tries to find a node that meets the rule. If a matching node is not available, the scheduler still schedules the Pod.

a.requiredDuringSchedulingIgnoredDuringExecution

***it will schedule if matches the pod and node label only otherwise it will not schedule

Ex: a.Schedule a Pod using required node affinity

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
spec:
 affinity:
  nodeAffinity:
   requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
    - matchExpressions:
      - key: disktype
        operator: In
        values:
        - ssd
 containers:
 - name: nginx
  image: nginx
  imagePullPolicy: IfNotPresent
```

b.preferredDuringSchedulingIgnoredDuringExecution:

******if label match it will create in matched node or another node

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
spec:
 affinity:
  nodeAffinity:
   preferredDuringSchedulingIgnoredDuringExecution:
   - weight: 1
     preference:
      matchExpressions:
      - key: disktype
        operator: In
        values:
        - ssd
 containers:
 - name: nginx
  image: nginx
  imagePullPolicy: IfNotPresent
```

==========

3. Daemonset:

- A Daemonset is another controller that manages pods like Deployments, ReplicaSets, and StatefulSets. It was created for one particular purpose: ensuring that the pods it manages to run on all the cluster nodes.pod is going to schedule all available nodes

ex : if we have three nodes same pod is going to schedule on three nodes

```
apiVersion: apps/v1
```

```
kind: DaemonSet
metadata:
 name: nginx
spec:
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: test-nginx
    image: nginx
    ports:
    - containerPort: 8080
    resources:
     limits:
      cpu: 100m
      memory: 200Mi
     requests:
      cpu: 50m
      memory: 100Mi
```

==========

#Taint and tolleration

- Taints are the opposite – they allow a node to repel a set of pods. Tolerations are applied to pods. Tolerations allow the scheduler to schedule pods with matching taints.

Two types:

1.Noschedule:

2.Noexecutive

Below Commands are for Node Taint and Untaint proces :

- kubectl taint node ip-192-168-43-22.ap-south-1.compute.internal app=blue:NoSchedule #taint
- kubectl taint node ip-192-168-43-22.ap-south-1.compute.internal app=blue:NoSchedule- # untaint

- kubectl taint node ip-192-168-43-22.ap-south-1.compute.internal app=blue:Noexecutive taint
- kubectl taint node ip-192-168-43-22.ap-south-1.compute.internal app=blue:Noexecutive- # untaint

- kubectl describe node ip-192-168-45-254.ap-south-1.compute.internal | grep Taints #to list tainted nodes

toleration pod example:

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
 labels:
  env: test
spec:
 containers:
 - name: nginx
  image: nginx
  imagePullPolicy: IfNotPresent
 tolerations:
 - key: "app"
  operator: "Equal"
  value: "blue"
  effect: "NoSchedule"
```

IMP Note:

*Toleration pod only create into specfic tainted node if labels match

*The taint effect defines how a tainted node reacts to a pod without appropriate toleration. It must be one of the following effects;

*NoSchedule—The pod will not get scheduled to the node without a matching toleration. (willnot schedule new pods on tainted node but runinng pods will not delete also after enable taint to nodes)

*NoExecute—This will immediately evict all the pods without the matching toleration from the node (no new pods will schedule will and also delete runinng pods also after enable taint to nodes)

==========volume==========

What is a Kubernetes volume?

A Kubernetes volume is a directory containing data accessible to containers in a given pod,

the smallest deployable unit in a Kubernetes cluster.

Within the Kubernetes container orchestration and management platform,

volumes provide a plugin mechanism that connects ephemeral containers with persistent data storage.

A Kubernetes volume persists until its associated pod is deleted.

When a pod with a unique identification is deleted, the volume associated with it is destroyed.

If a pod is deleted but replaced with an identical pod, a new and identical volume is also created.

Step - 1

a) Create an EKS cluster with the clusteconfig file I provided.

b) Install helm in your local machine. Below is the link -> https://helm.sh/docs/intro/install/

c) Connect to your EKS cluster. Check the connection.

Three types of create volume approaches

1. Static hots path (pv,pvc,deployment) not recomanded if node will delete volumes also delete

2. Static cloud ebs(pv,pvc,deployment) static approach no effect even node will be deleted still volum persistent but ec2 volume need to create manually

3. Dynamic volume provision create by Storage class (pvc,storage class) here pv is going to create by Storage class

-----helm install-----

- https://github.com/helm/helm/releases
- wget https://get.helm.sh/helm-v3.14.0-linux-amd64.tar.gz
- tar -zxvf helm-v3.14.0-linux-amd64.tar.gz
- mv linux-amd64/helm /usr/local/bin/helm
- chmod 777 /usr/local/bin/helm # give permissions
- helm version

#Note: after create cluster we have to give Iam ec2 full access or admin access to node group IAM role then only ebs volume able to create by node

Step - 2

Install CSI driver in EKS cluster by following below steps.

a) After connection execute the below commands.

- helm repo add aws-ebs-csi-driver https://kubernetes-sigs.github.io/aws-ebs-csi-driver
- helm repo update

#install aws ebs driver to kubernets

- helm upgrade --install aws-ebs-csi-driver --namespace kube-system aws-ebs-csi-driver/aws-ebs-csi-driver

#ReadWriteMany#

If you need to write to the volume, and you may have multiple Pods needing to write to the volume

where you'd prefer the flexibility of those Pods being scheduled to different nodes,

and ReadWriteMany is an option given the volume plugin for your K8s cluster, use ReadWriteMany.

#ReadWriteOnce#

If you need to write to the volume but either you don't have the requirement that multiple pods should be able to write to it,

or ReadWriteMany simply isn't an available option for you, use ReadWriteOnce.

#ReadOnlyMany

If you only need to read from the volume, and you may have multiple Pods needing to read from the volume

where you'd prefer the flexibility of those Pods being scheduled to different nodes,

and ReadOnlyMany is an option given the volume plugin for your K8s cluster, use ReadOnlyMany.

If you only need to read from the volume but either you don't have the requirement that multiple pods should be able to read from it,

or ReadOnlyMany simply isn't an available option for you, use ReadWriteOnce.

In this case, you want the volume to be read-only but the limitations of your volume plugin have forced you to

choose ReadWriteOnce (there's no ReadOnlyOnce option). As a good practice, consider the containers.volumeMounts.readOnly

==========

#PVC recalim policies

reclaim policy : delete # this is defualt one ,which means when ever pvc deleted pv will delete automatically cretaed by storage class but ebs volume will not delete if you want to delete delete it manually

- reclaim policy : retain # it will persistant

#Types Of Volume bindings

- volumebinding : Immediate # this is default one ,pv is created immdeialty
- volumebinding : WaitForFirstConsumer #pv will create only any pod will claim the stoarage only

==========

Grafana and Prometheus:

Prometheus collects and stores metric data as time-series data, while Grafana is an analytics and visualization web application that can ingest data from various sources and display it in customizable charts.

For more details:

#Refer below blog

- https://medium.com/@veerababu.narni232/deployment-of-prometheus-and-grafana-using-helm-in-eks-cluster-22caee18a872
-

----------HELM Charts----------

Helm is a tool that automates the creation, packaging, configuration, and deployment of Kubernetes applications by combining your configuration files into a single reusable package.

-----helm install-----

- https://github.com/helm/helm/releases
- wget https://get.helm.sh/helm-v3.14.0-linux-amd64.tar.gz
- tar -zxvf helm-v3.14.0-linux-amd64.tar.gz
- mv linux-amd64/helm /usr/local/bin/helm
- chmod 777 /usr/local/bin/helm # give permissions

- helm version

- helm create firstproject helloworld #create sample helmp chart application—helloworld
- helm install <FIRST_ARGUMENT_RELEASE_NAME> <SECOND_ARGUMENT_CHART_NAME> # run helm

➔ whenever creates the sample project we will get template of kuberentes structure like below

----------so we can place our values and images

```
1 helloworld
2 ├── charts
3 ├── Chart.yaml
4 ├── templates
5 │   ├── deployment.yaml
6 │   ├── _helpers.tpl
7 │   ├── hpa.yaml
8 │   ├── ingress.yaml
9 │   ├── NOTES.txt
10 │   ├── serviceaccount.yaml
11 │   ├── service.yaml
12 │   └── tests
13 │       └── test-connection.yaml
14 └── values.yaml
```

- helm list -a # to list helm chart
- helm upgrade firstproject helloworld
- helm delete <chart> # to delete chart

For more details:

#Refer below blog

- https://medium.com/@veerababu.narni232/writing-your-first-helm-chart-for-hello-world-40c05fa4ac5a
-

----------ArgoCD----------

Argo CD is a declarative continuous delivery tool for Kubernetes. It can be used as a standalone tool or as a part of your CI/CD workflow to deliver needed resources to your clusters.

-------Install process

- kubectl create namespace argocd
- kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
- kubectl get pods -n argocd
- kubectl get svc -n argocd
- kubectl edit svc argocd-server -n argocd
- kubectl get svc -n argocd

#access it nodeIp:Nodeport

- kubectl get secrets -n argocd

user= admin

- kubectl edit secret argocd-initial-admin-secret -n argocd # to get intial credential to login argocd
- echo bnFabGx3emtCNjB5dFZQSA== | base64 --decode

##For more details:

#Refer below blog

- https://medium.com/@veerababu.narni232/a-complete-overview-of-argocd-with-a-practical-example-f4a9a8488cf9
-

==========

#Statefulset

➔ StatefulSet is the workload API object used to manage stateful applications.

➔ Manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods.

➔ Like a Deployment, a StatefulSet manages Pods that are based on an identical container spec. Unlike a Deployment, a StatefulSet maintains a sticky identity for each of its Pods. These pods are created from the same spec, but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling.

Using StatefulSets :

➔ StatefulSets are valuable for applications that require one or more of the following.

Stable, unique network identifiers.

Stable, persistent storage.

Ordered, graceful deployment and scaling.

Ordered, automated rolling updates.

--------for more details

#Refer below blog

- https://medium.com/@veerababu.narni232/what-are-stateful-applications-2a257d876187
-

==========

#Headless service

- A Headless Service is a variation of the ClusterIP Service, where the clusterIP field is set to None. Unlike traditional services, Headless Services do not use a single Service IP to proxy connections to the Pods. Instead, they allow you to directly connect to Pods without any load balancing intermediary.
- Use Cases for Headless Services
- Headless Services are particularly useful in the following scenarios:
- Service Discovery: Some service discovery mechanisms, such as Kubernetes DNS-based service discovery, require direct access to individual Pods. Headless Services provide a convenient way to achieve this.
- Stateful Applications: Applications that require direct access to individual Pods, such as databases or distributed storage systems, can benefit from using Headless Services.
- Custom Load Balancing: If you need to implement custom load balancing logic or use a specific load balancing mechanism, Headless Services allow you to directly access the Pods without relying on Kubernetes' built-in load balancing.

-----Accessing Pods using Headless Services

-------example through DNS name:

- Once you have created a Headless Service, you can access the Pods directly using their DNS names or IP addresses.

- The DNS name for each Pod follows the pattern <pod-ip>.<namespace>.pod.cluster.local.

➔ ##For example, if you have a Pod with the IP address 10.0.0.5 in the default namespace, you can access it using the DNS name like below

- 10-0-0-5.default.pod.cluster.local. -----dns name

Also

- You can also access the Pods directly by their IP addresses,
- which can be useful for applications that require direct IP-based communication.
- 10-0-0-5 --ip of the target pod

Conclusion

➔ Headless Services in Kubernetes offer a unique approach to accessing individual Pods directly, without relying on a load balancer or a single Service IP. This type of service is invaluable in scenarios where direct Pod access is required, such as service discovery mechanisms, stateful applications like databases, or when implementing custom load balancing logic.

➔ By setting the clusterIP field to None, Headless Services bypass the traditional load balancing layer and instead provide a direct connection to individual Pods. This is achieved through the assignment of DNS records for each Pod, allowing you to access them by their DNS names or IP addresses.

- kubectl run -i --tty --image busybox:1.28 dns-validate # create pod and access servcice from pod by using nslookup.
- example : nslookup<servcie name> nslookup <cluster ip enables servcie name>

➔ Server: 10.100.0.10

➔ Address 1: 10.100.0.10 kube-dns.kube-system.svc.cluster.local

you will get result cluster IP only not pod Ips

- nslookup<servcie name> nslookup <headless servcie name>

result

Name: mysql

Address 1: 192.168.11.161 mysql-0.mysql.default.svc.cluster.local

Address 2: 192.168.61.95 mysql-1.mysql.default.svc.cluster.local

========== Probes ==========

Kubernetes, the industry-leading container orchestration tool, offers mechanisms to implement robust health checks for your applications. These checks, termed “probes,” act as guardians of your application's well-being, continuously monitoring the health status of your pods and their hosted applications

For More deatils Refer blog

- https://medium.com/@veerababu.narni232/probes-in-kubernetes-5133ebe03475
-

Note:

pleae click below git hub link for all yml files

- https://github.com/CloudTechDevOps/Kubernetes

**********SonarQube**********

Code Quality check tool

#SonarQube ----------other tools (veracode, coverity, codescence)

SonarQube is a very popular code quality management tool that is used widely for code analysis to identify code smells, possible bugs, and performance enhancements. SonarQube supports many popular programming languages like Java, JavaScript, C#, Python, Kotlin, Scala etc. It also provides test and code coverage.

Benfits:

Improve quality.

Grow developer skills.

Continuous quality management.

Reduce risk(vulnerabilities).

Scale with ease.

Code quality and smell

code smell : A maintainability issue that makes your code confusing and difficult to maintain.

Code vulnerabilities: Code vulnerabilities are software flaws that open opportunities for potential application misuse, exploits, or breaches that result in sensitive information disclosure, data leaks, ransomware attacks, and other cyber security issues.

---

threecompontes

------>sonarquber server

----->rules --apply rules on code

--defult data base embedded H2 database

------>scanner

gather rules

scan the source code

send the reports to database

STEUP:

required 4gb ram and 2 cpus

dependency: java11 or 17

req: t2.medium

port: 9000

#Launch an t2.medium instance with allow port 9000

- amazon-linux-extras install java-openjdk11 -y #if amzon linux 2
- sudo dnf install java-11-amazon-corretto # if amazon linux 2023
- cd /opt/ #switch to opt directory
- sudo wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-8.9.6.50800.zip
- sudo unzip sonarqube-8.9.6.50800.zip

sonarqube has to run with user only

- sudo useradd sonar
- sudo chown sonar:sonar sonarqube-8.9.6.50800 -R
- sudo chmod 777 sonarqube-8.9.6.50800 -R

➔ passwd sonar (create password)

- su - sonar (log in as sonar)

#run this on server manually

- sh /opt/sonarqube-8.9.6.50800/bin/linux-x86-64/sonar.sh start
- sh /opt/sonarqube-8.9.6.50800/bin/linux-x86-64/sonar.sh status

- to start ----- sh sonar.sh start or ./sonar.sh start

- to check status ----- sh sonar.sh status

#to access

public-ip:9000

user=admin & password=admin

note : after login we need to give our custom password

---

sonarQube on Docker

docker run -d --name sonar -p 9000:9000 sonarqube:lts-community

---

For More details Refer blog

- https://medium.com/@veerababu.narni232/sonarqube-step-by-step-static-code-analysis-implementation-9d788890b506

----------configure with jenkins----------#####

prerequistes :

create token in sonarqube ---->after login go to adminstration---security--user--create token

step-1

--Required plugins--

- ➔ sonarqube scanner for Jenkins
- ➔ qulaity gates if required (optional)

step-2

open your sonarqube—generate the token

go to credential and select secret text mode paste you token and give name

go to system configurations paste url and select key and give name ex: 'SonarQube' (note: this name only we have to call like withSonarQubeEnv('SonarQube'))

step-3

--create jenkins job and paste below script mavne and sonar

--passing with envvironment varaibles without hards coded values (sonar url and token)

```
pipeline {
  agent any

  stages {
    stage('scm') {
      steps {
        git branch: 'main', url: 'https://github.com/nareshdevopscloud/project-1-maven-jenkins-CICD-docker-eks-.git'
      }
    }
    stage('clean') {
      steps {
        sh 'mvn clean
      }
    }
    stage('code quality') {
      steps {
        withSonarQubeEnv('SonarQube'){
        sh 'mvn install sonar:sonar'

      }
    }
```

```
    }
  }
}
```

---------- jfrog----------

Why Only Jfrog

Jfrog Artifactory?

What is Artifact

what is artifact Repository?

why artifact Repository?

Type of packages it supports:

How to setup Artifactory server on AWS

Artifact Deployment from maven to Jfrog

Artifactory Integration with jenkins

repository management tool

==========

- Repository management tools helps development teams create, maintain, and track their software packages.

Options

---

1. Jfrog Artifact
2. Nexus
3. Apache Archiva
4. Nuget
5. github
6. s3

Why Only Jfrog

But jfrog supports wide range of formets and types.

Eg:-

python repo

chef repo

puppet repo

Apt repo

yum repo

docker repo

rpm repo

maven repo...etc

- it is a repository management tool

- A universal artifact repository manager

- JFrog Artifactory is a repository manager that supports all available software package types

- Artifactory, the first-in-class binary repository management

Jfrog Artifactory?

==========

Jfrog Artifactory is a tool used in devops methodology to store artifacts (readily deployable code)

What is Artifact

The files that contain both compiled code and resources that are used to compile them are know as artifact.

- They are readily deployable files.

- source code --> Build Tools --> Compilation --> Binary code --> Dependencys/resources --> Artifact

- in java an artifacts would be jar, war, ear...etc file

- in .net an artifacts would be .dll file

what is artifact Repository?

- An artifact repository is a repository which can store multiple different versions of artifacts.each time the war or tar.gz file is created. it stored in a server dedicated for the artifacts.

in real-time in the above process if you have any error in test env, we will rollback to version control to fix it. instead of that if you store artifacts in repo we can rollback to prevision version.

Author

---

https://jfrog.com

- they have number of products like
  - Jfrog Artifactory (Very Popular)
  - Jfrog Pipelines
  - Jfrog x-ray
  - Jfrog connect
  - JFrog Container Registry ...etc

Written in

- Jfrog developed in java. it is platform independent(run in windows, mac, Linux)

Releases

- Free, Professional and Enterprice

Free vs Pro

---

https://jfrog.com/community/download-artifactory-oss/

Type of packages it supports:

---

https://www.jfrog.com/confluence/display/JFROG/Package+Management

Note:- what kind of repo = what kind of package

How to setup Artifactory server on AWS

---

1. Pre-requisites:

An AWS T2.medium (4GB RAM)EC2 instance (Linux)

➔ Open port 8081 and 8082 in the security group

- wget https://releases.jfrog.io/artifactory/artifactory-rpms/artifactory-rpms.repo -O jfrog-artifactory-rpms.repo;
- sudo mv jfrog-artifactory-rpms.repo /etc/yum.repos.d/;
- sudo yum update && sudo yum install jfrog-artifactory-oss
- systemctl start artifactory.service
- systemctl status artifactory.service

➔ http://<PUBLIC_IP_Address>:8081

After Login we need to Provide

Username as admin

Password as password

#Note:we have to change it password after loggedin

########## APPROACH-1 #########

Artifact Deployment from maven to Jfrog without jenkins

==========

1. Add Deployment element to maven pom.xml file.

```
 <distributionManagement>
   <repository>
     <id>central</id>
     <name>NareshIT</name>
     <url>http://34.238.157.19:8081/artifactory/naresh-repo</url>
   </repository>
   <snapshotRepository>
     <id>central</id>
     <name>NareshIT</name>
     <url>http://34.238.157.19:8081/artifactory/naresh-repo</url>
   </snapshotRepository>
</distributionManagement>
```

2. Add Jfrog Credentials to apache-maven-3.8.6-->--conf-->settings.xml file.

```
        <servers>
          <server>
                <id>naresh</id>
                <username>admin</username>
                <password>naresh_123</password>
          </server>
        </servers>
```

3. Navigate to maven structure where pom.xml and src locates, and give below command.

- mvn Deploy

###############APPROACH-2##########

Artifactory Integration maven integration with Jfrog by using jenkins

==========

pre-requisites

A Artifactory server

A Jenkins Server

Integration Steps

Login to Jenkins to integrate Artifactory with Jenkins

1. Install "Artifactory" plug-in

Manage Jenkins -> Jenkins Plugins -> available -> artifactory

2. Configure Artifactory server credentials

Manage Jenkins -> Configure System -> Artifactory (or) Jfrog

3. Artifactory Servers

Server ID : test

URL : Artifactory Server URL (http://localhost:8081/)

Username : admin

Password : default password is "password" only. (login with default password later you can change)

Test the connection

Approach-1

Create a Freestyle Project

- Create a new job
  -> Job Name : ex:Job1
- Source code management
  -> Github URL : <githuburl>
- Build Environment
  -> Maven3-Artifactory Integration : ` <provide Artifactory server and repository details
- Build --> Invoke Artifactory Maven3
- Goals: clean deploy
- Execute job

Approach-2

Create a Maven Project

- Create a new job
  ->. Job Name : ex:Job2
- Source code management
  -> Github URL : <github>
- Build Environment(optional)
  -> Resolve artifacts from Artifactory : <provide Artifactory server and repository details>
- Build - Goals: clean install
  ->. Post-build Actions
- Deploy Artifacts to Artifactory : <provide Artifactory server and repository details>
- Execute job

Approach-3 (recomended)

step-1 Install Artifactory plugin

step-2 Configure Artifactory server credentials

Manage Jenkins -> Configure System -> Artifactory (or) Jfrog (Note: after configure serverID it will take care of URL and authentication of JFROG)

Step-3 create pipeline job

```
pipeline {
  agent any

  stages {
    stage('stage-1') {
      steps {
        git branch: 'main', credentialsId: 'terraform', url:
'https://github.com/nareshdevopscloud/project-maven-jenkins-CI-CD.git'
      }
    }

    stage('clean') {
      steps {
       sh  'mvn clean'
      }
    }

    stage('test') {
      steps {
       sh  'mvn test'
```

```
    }
  }
  stage('install') {
    steps {
      sh  'mvn install'
    }
  }

  stage('Push artifacts into artifactory') {
    steps {
     rtUpload (
      serverId: 'jfrog_server',
      spec: '''{
          "files": [
           {
            "pattern": "*.war",
             "target": "JFrog/"
           }
         ]
       }'''
      )
   }
  }
  stage('deployment') {
    steps {

     deploy adapters: [tomcat9(credentialsId: 'tomcat', path: '', url: 'http://18.204.210.126:8181/')],
contextPath: null, war: 'webapp/target/*.war'

    }
  }
}
```

}

---------- Trivy ----------

Trivy scans local and remote container images, supports multiple container engines, as well as archived and extracted images. It works on raw filesystem and remote git repositories. With Trivy, you can scan whenever and wherever you need to.

wget rpm -ivh https://github.com/aquasecurity/trivy/releases/download/v0.48.3/trivy_0.48.3_Linux-64bit.rpm

trivy verison

trivy image <name of image>

it will start security check and vulnerbalities

----------ansible----------

What is Ansible?

Ansible is an open source, command-line IT automation software application written in Python. It can configure systems, deploy software, and orchestrate advanced workflows to support application deployment, system updates, and more. Ansible's main strengths are simplicity and ease of use.

#How Ansible works?

In Ansible, there are two categories of computers: the control node and managed nodes. The control node is a computer that runs Ansible. There must be at least one control node, although a backup control node may also exist. A managed node is any device being managed by the control node.

- ➔ sudo yum update -y
- ➔ sudo yum install ansible -y

log in to main server (Ansible)

==========

Connection process (SSH)

==========

generate keys

ssh-keygen --copy public ip(id_rsa.pub) and paste it into target server path .ssh/authorizedkeys

so now Ansible server able to ping target server

==========

Inventory File Process

==========

What is Inventory File?

Inventory defines the managed nodes you automate, with groups so you can run automation tasks on multiple hosts at the same time

Create our own inventory file vi inventory and add target private ips

or

We can add target private ips into defualt path of inventory fiel

- path--- vi /etc/ansible/hosts

==========

Ad-hoc commands

==========

-# sample ad-hoc coomand ----------

Ad-hoc commands are commands which can be run individually to perform quick functions

Ansible ad-hoc commands

- ansible -i inventory all -a "yum install git -y" -b # this command is to define our own path inventory
- ansible all -a "yum install git -y" -b # this command is for deaflt path of inventory so no need to give "-i inventory argument"

---here '-b' is beacome a root

---- '-a' is ad-hoc argument

'--- all' is apply for all hosts # we can replace with all if we want to call sepcific ip

Example

[web]

192.168.1.2

[test]

192.168.1.3

if i want to run any adhoc command give below example

ansible web -a "yum install git -y" -b

Exampes:

- ansible -i inventory all -a "yum install maven -y" -b # with define inventory
- ansible -all -a "git --version" -b #default inventory
- ansible -i inventory all -a "touch file100" -b

Ansible Module

Ansible modules are units of code that can control system resources or execute system commands. Ansible provides a module library that you can execute directly on remote hosts or through playbooks. You can also write custom modules.

----ping module-------

ansible -i inventory all -m ping

-----stat module ----

ansible -i inventory all -m stat -a "path=/var/www/html"

----------user----------

ansible -i inventory all -m user -a "name=naresh" -b

-----------setup--------

ansible -i inventory all -m setup

----------file----------

ansible -i inventory all -m file -a "name=demo state=touch"

----------copy----------

ansible -i inventory all -m copy -a "src=file1 dest=~"

------yum or apt module ----- ##Very ImP

(basic state's

name = (httpd install)

state=latest

state=present

state=absent)

ansible all -m yum -a "name=httpd state=latest" -b

ansible all -m yum -a "name=httpd state=present" -b

ansible all -m yum -a "name=httpd state=absent" -b

(name = systemctl or service

state=stopped

state= started

state=restrted

state= absent)

Examples :

ansible -i inventory all -m yum -a "name=httpd state=latest" -b #to install httpd

ansible -i inventory all -m service -a "name=httpd state=started" -b #to start service

ansible -i inventory all -m service -a "name=httpd state=stopped" -b #to stop service

ansible -i inventory all -m yum -a "name=httpd state=absent" -b #to uninstall service

ansible -i inventory all -m service -a "name=httpd state=restarted" -b #to restart service

========== Host Key verification put false by default it is true ==========

sudo vi /etc/ansible/ansible.cfg #create config file

[defaults]

host_key_checking = False

----------Ansible playbook----------

Ansible Playbooks are lists of tasks that automatically execute for your specified inventory or groups of hosts. One or more Ansible tasks can be combined to make a play—an ordered grouping of tasks mapped to specific hosts—and tasks are executed in the order in which they are written

```
---
- name: first playbook
  hosts: all
  become: yes
  tasks:
    - name: install httpd software
      yum:
        name: httpd
        state: latest
    - name: start web server
      service:
```

```
      name: httpd
      state: started
```

```
# to execute plyabook
ansible-playbook -i inventory test-playbook.yaml
```

======== Deploy through Copy ========

```
---
- name: first playbook
  hosts: all
  become: yes
  tasks:
    - name: install httpd software
      yum:
        name: httpd
        state: latest
    - name: start web server
      service:
        name: httpd
        state: started
    - name: copying the files
      copy:
        src: index.html
        dest: /var/www/html/index.html
    - name: restart server
      service:
```

```
      name: httpd
      state: restarted

==============================================
----------Global Variable----------
==============================================
---
- name: first playbook
 hosts: all
 become: yes
 vars:
  - a: httpd
  - b: present
  - c: started
  - d: restarted



 tasks:
  - name: install httpd software
    yum:
      name: "{{a}}"
      state: "{{b}}"
  - name: start web server
    service:
      name: "{{a}}"
      state: "{{c}}"
  - name: copying the files
    copy:
      src: index.html
      dest: /var/www/html/index.html
  - name: restart server
```

```
    service:
     name: "{{a}}"
     state: "{{d}}"
```

========== TAGS ==========

TAGS: To execute or skip specific tasks

```
- name: first playbook
  hosts: all
  tasks:
    - name: installing git
      yum:
       name: git
       state: present
      tags: a
    - name: installing httpd
      yum:
        name: maven
        state: present
      tags: b
    - name: create user
      user:
        name: test
        state: present
      tags: c
```

SINGLE TAG: ansible-playbook name.yml --tags a

MULTI TAG: ansible-playbook name.yml --tags b,c

SKIP TAG: ansible-playbook name.yml --skip-tags "c"

MULTI SKIP TAGS: ansible-playbook name.yml --skip-tags "c,d"

Dynamic inventory file process

Prerequistes

python3

boto3

==== Install python and boto3 ====

yum update -y #update

yum install python -y # install python

----install pip

wget https://bootstrap.pypa.io/get-pip.py #download pip

python get-pip.py #install pip

---install boto3

pip install boto3 #to install boto3

pip show boto3   #to verify boto3

You might already have this collection installed if you are using the ansible package. It is not included in ansible-core. To check whether it is installed, run ansible-galaxy collection list.

To install it, use: ansible-galaxy collection install amazon.aws. You need further requirements to be able to use this inventory plugin, see Requirements for details.

To use it in a playbook, specify: amazon.aws.aws_ec2.

sudo vi /etc/ansible/ansible.cfg

```
[defaults]
enable_plugins = aws_ec2
inventory =./aws_ec2.yml
host_key_checking = False
```

sudo vi /etc/ansible/aws_ec2.yml

```
---
plugin: amazon.aws.aws_ec2
regions:
 - ap-southeast-1
filters:
 tag:Name:
  - dev
    #add multiple tag names here if required
```

for all

```
# if we want to add inventory groups
plugin: amazon.aws.aws_ec2
regions:
 - ap-southeast-1
keyed_groups:
 # add hosts to tag_Name_value groups for each aws_ec2 host's tags.Name variable.
 - key: tags.Name
   prefix: tag_Name_
   separator: ""
groups:
 # add hosts to the group dev if any of the dictionary's keys or values is the word 'dev'.
 development: 'dev' in (tags|list)"
filters:
 tag:Name:
   - 'dev'

ansible-inventory -i aws_ec2.yml --list

ansible-inventory -i aws_ec2.yml --graph


-------sample playbook inventory all no without group


---
- name: first playbook
 hosts: all
 become: yes
 tasks:
   - name: install httpd software
     yum:
       name: httpd
```

```
      state: latest
  - name: start web server
    service:
      name: httpd
      state: started
```

--------sample playbook with inventory group

```
---
- name: first playbook
 hosts: tag_Name_dev
 become: yes
 tasks:
  - name: install httpd software
    yum:
      name: httpd
      state: latest
  - name: start web server
    service:
      name: httpd
      state: started
```

Reference ansible offcial document

https://docs.ansible.com/ansible/latest/collections/amazon/aws/docsite/aws_ec2_guide.html