rds (relational data service)
---

rds stands for relational data service, it a database service which manages the lifecycle operations for the relational database management systems like
1. mysql
2. oracle
3. postgress sql
4. ms sql server
5. db2
6. aurora db

it is recomended to create rds instance within private subnet. for high availability aws encourages the users to create subnetgroup in which the mysql/rds instance will be provisioned. if there is an downtime in availability zone, aws automatically bringsup the rds instance on another availability zone within subnet group based on the backup.

#1 create a subnetgroup
#2 provision the rds instance by selecting the database
#3 while provisioning the rds instance we need to provide the following inputs
- vpc
- type of database
- machine shape
- storage
- subnet group
- enable public ip (not recommended) (endpointname - within vpc)
- backup configuration
- username/password
- database configuration
- security group (3306)

what we should do to enable mysql access to ec2 instance?
1. goto nacl rules add mysql/aurora with port 3306 to be accessible within vpc cidr
2. security group of ec2 instance allow ec2 to communicate over 3306

3. install mysql-client on ec2 and try to the rds instance
yum install https://dev.mysql.com/get/mysql80-community-release-el7-3.noarch.rpm
yum install mysql-community-client

Load balancer
---
it is often recommended to host the application on multiple ec2 instances of different availability zones, so that we can achieve high availability. how to route the application requests across multiple ec2 instances of the application.
Here is what we use Load Balancer,
load balancer is used for distributing the traffic between multiple ec2 instances, through which we can achieve fault tolerance, resilience and high availability
- load balance is a traffic distributor between ec2 instances
- it is available per region level
- spans across all the availability zones of the region
- load balancers are high available, scalable and resilience, we dont need to create multiple load balancers aws cloud platform takes care of managing them.
- health checks = will be conducted by the lbrs periodically, to identify which ec2 instances are available to route the traffic
- supports auto-scaling groups = auto-scaling group will takes care of scaling-out/scale-out the ec2 instances based on their threadshold values. For eg... auto-scaling group has provisioned a new ec2 instance to handle load, this ec2 instance will gets automatically registered with load balancer and will start routing the traffic to the instance

There are 3 types of load balancers are there in aws platform
1. application load balancer = The application load balancer helps us in routing the traffic based on rules as url patterns in forwarding the traffic to appropriate target groups.
- Target Group = setup group of instances that are hosted with our application as one group
The group can be created with using
- instance id
- ip addresses
- Listener = in Lbr we need to add listener configuring to receive a specific protocol request on a port number
- then we need to add routing rules based on urls, to route the request across the application of the different target groups

2. network load balancer
The network routes the traffic to the ec2 instances based on protocol/portno to fixed set of ec2 instances based on the ip address only. It operations at layer 3 of the OSI Model

3. Gateway load balancer
These acts as Routers/Gateways at the network level, they operate at very low level on the network. Intecepting data packets and detecting fraud, firewall aspects on the network. These acts as security gateways in monitoring, controlling and detecting the network traffic

Load balancer = is used for distributing the traffic to mulitple backend servers/ec2 instances. Through load balaner we can achieve High Availability, Fault Tolerance and Resilence.

1. These are create at region level and spans across all the availability zones of that region
2. aws takes care of the load balancers to be HA, Fault Tolerant and Resilent
3. Load balancer should be configure with listener, in receiving and routing the traffic to the Target Groups
4. These periodically runs health check to determine which backend servers are healthy to route the traffic
5. They support auto-scaling group, so that the new instances will be register automatically
6. Route53 service to map domain name to loadbalancer

While creating an aws instance, it asks for either provide our own ssh keys or generate a keypair from aws.
The aws generates the keypair and allows us to download it as ".pem".format

.pem = stands for privacy enhanced mail. it is a text file containing the contents in a Base32 format. This file contains both public/private key as part of it which we can download and use it. It works on all the platforms
- linux
- mac
- windows

.ppk/.pub = putty private key / public key are the special formats that are create by the putty tool for using them as ssh keys for connecting the remote server. putty doesnt support ".pem" format. so if we have a pem file we can convert the pem into ppk and pub files by using puttygen.

IAM Service
Identity and Authorization Management Service

Using IAM Service we can create users and authorize them to access the aws services of aws cloud platform
1. Identity
2. Authorization
3. Users (to grant access to the aws cloud platform)
4. Groups (allocate users to the groups)
5. Create Policies
6. Create Roles
7. Multi-Factor authentication (MFA)

Wrong Myth about IAM Service

The users/groups we create in IAM are used for gainging the access to the aws cloud platform in managing the aws services through
1. aws cloud console
2. aws cli
3. sdk
4. api
the IAM user can perform the lifecycle operations on the services/resources of aws like
- provision
- deprovision
- scaleup
- scaledown
- scale out
- scale in
...

But these users are not meant to access the resources that are provisioned.
- For e.g.. we need to access an ec2 instance remotely using
- ssh keypair
- operating system user/password as part of that ec2 instance
similarly for an rds instance we create a database username and password with which we login into database for managing
---
There are 2 types of aws account users are there
1. aws account administrator

aws account administrator gains an unretricted access to all the services of aws platform, inturn he can create aws account users and can grant access to the aws services to those users.

2. aws account user
is an non-admin or role based user, based on roles assigned he can manage the aws services onbehalf of aws account administrator.
few points to remember:-
1. aws account user by default when created by an aws account administrator will gain access to the aws services through api only. aws account administrator generates a api access key pair and issues to the aws account user, using this keypair aws account user can access the aws service through
- api
- sdk
- cli only
usually we create these aws account users to manage aws services through iac automation tools like terrafrom etc.
2. always an aws account administrator will be registered through email address and while logging in you need to choose in cloud console administrator login or account user login based on which it prompts for email address.
an aws account user will not register in aws cloud console through email address, he has to use a username to gain access to the account
3. aws account administrator can enable access to the aws cloud console for an account user as well.
4. aws account administrator while creating aws account user attaches policies which are nothing but permissions to the aws services based on which he gains access to the platform.

Identity and Authorization Management Service (IAM)
IAM service is used for creating users/groups and delegating the access to the aws services of cloud platform to the users.
- users
- groups
- policies
- roles
- MFA

#1 users
There are 2 types of users are there in aws
1. Aws Account Administrator = He is the Root user of the aws account, has full-access to the services of his account, and he create and delegate permissions to the aws account users

2. Aws Account User
Non-Admin has access to the services of the aws account based on the permission given by the aws account administrator.
by default he dont have aws management console access unless provided by the aws account administrator
An aws account user will be generated with aws access key comprises of aws access key and secret code using which he can gain access to the aws services either through
- api
- cli
- sdk

---

groups
- set of aws account users are grouped together so that managing the group of users and permissions would be easier than an individual
- we can create different groups like administrators, developers, qaengineering, devopsengineers, we can assign policies (permissions) to the group level. Now we can organize the users into these groups based on their roles. Any change required in terms of permission granted can be achieved easily just by modifying the policy attached to the group, so that all the users would gets effected.
- a user can be part of multiple groups as well
- groups cannot be nested
- we can create a user with no group as well, in such case we need to assign permissions/ policies directly to the user.

MFA = Multi-Factor Authentication
The Multi-Factor authentication device
- physical device (courier) = unique code
- application installed on your mobile = unique code

For a user we need to explicity enable multi-factor authentication, so that aws prompts for the access code/unique code generated by the MFA device to login into the aws management console

---

Policies
Policies are the access grants or restrictions we want to enforce on the aws account users / groups. through policies we delegate access rights/permissions to the aws services of an aws account.

Policies are json text files describing the information about the resources, type of access to

those resources. and once creates those are attached to user/groups

There are 2 types of policies are there
1. aws managed policies
These are pre-defined policies created and provided by the aws for the convinience of the aws account administrator, so he can quickly use any of he policies out the available rather than creating one of his own each time.
aws has pre-defined policies almost for all the resources/services in aws cloud platform with fixed set of permissions in each policy

e.g.. policies
AWSEC2FullAccess
AWSEC2ReadonlyAccess
aws maintains these policies and very rarely it is going to makes changes into these policies

2. customer managed policies
aws account administrator can create his own policies for granting access to the aws services to an aws acount user. These customer managed policies are of 2 types
2.1 customer managed policies at account level = These policies are available within the aws account only and managed by customer. These policies can be reused for creating groups and users of the account

2.2 inline policies = if we have a policy that we want to apply for a specific user/group only once, then instead of defining it as customer managed policy we can create inline and cannot be reused.

How does an aws policy can be created and looks like?
aws policy is a text describing resources and their permissions. we can create aws policy by writing directly the json text or we can use aws policy generator tool available in aws console.

IAM Service
Identity and Authorization Management Service = is used for delegating the accessing permissions on aws services of the cloud platform to the aws account users
1. users
2. groups
3. MFA (Multi-Factory authentication)
4. create policies (permissions) attach group/user
5. roles

Policies
are the way through which we delegate access permissions to the aws account users/groups.
There are 2 types of policies are there
1. aws managed policies
These are pre-defined policies attached with permission defined by aws for the use of aws account users. There are global policies available for all the accounts. Aws takes care of managing these policies and very rarely the are going to modify these policies

2. customer managed policies
2.1 customer account-level policies
- these policies are created by the customer and are available to aws account level

2.2 inline policies
- these are onetime policies that are attached directly to the user/group and cannot be reused.

Policies are json text files describing resources and the action allowed on these resources, account users can write policies manually and can create in aws (if familiar). otherwise we can take the help of aws policy generator tool

Policy json

In an policy json comprises of multiple policies as part of it each targeting different resources and their permissions, which are nothing policy statements
Each Policy Statement contains 3 parts of it
1. Effect (Allow/Deny)
2. Action = provision, list
3. Resource = ec2
finally we can apply conditions on when does this policy should be applied
here the resource name are identified as arn (amazon resource name). amazon has given for each resource a unique name to identify and enforce policy on it.
arn:aws:s3 = s3 bucket

```
{
"Version" : "2021-04-07",
"Statement": [
{                              --- the user attached with this policy can get the list of all s3
buckets of the account
"Effect": "Allow",
"Action": "s3:ListAllMyBuckets",
"Resource" : "arn:aws:s3:::*"
},
{                        --- use can store/access objects from all the buckets of the account
"Effect": "Allow",
"Action" [
"s3:PutObject",
```

"s3:GetObject"
],
"Resource" : "arn:aws:s3:::*"
}
]
}

What are aws roles, what is the use of them?
We can allow one account user of an aws account to gain temporary access to the another aws account by creating and issuing roles to the another account.
Roles are attached with policies and we assume a user is in a role to authorize him in accessing our account.
---

scp command:-
scp -i ~/Key/java_serverkp.pem ~/Key/java_serverkp.pem ubuntu@ip:/home/ubuntu/.ssh
---

Terraform

There are various cloud providers are available in the market, like google cloud platform, amazon web services cloud, oracle container infrastructure, Microsoft Azure, Digital Ocean, Pivotal Cloud Foundary etc. across all of these providers they provide cloud console management tool, which is nothing but a web-based user interface that users can login and access the cloud platform of the provider.

By using cloud console we can provision various different infrastructure resources offered by the respective cloud vendor, by navigating through the web pages and filling the form wizards etc. The process of creating the infrastructure resources through cloud console applications are manual and requires human interaction, due to which we run into lot of problems.
1. takes lot of time in provisioning the infrastructure
2. often errorneous
3. efforts are duplicated in setting up similar type of infrastructure

To overcome the above challenges we need to adopt iac tools.
Infrastructure as code (iac) = we define the infrastructure resources interms of code and upon executing the code, will creates the resources are the respective environments.

There are various iac tools which are also called as provisioning automation tools are available in the market like
1. terraform
2. openstack heat
3. cloud formation

What is the benefits of using iac tools?
The application that adopts iac/provisioning automation tools has better rate of delivery of the application as the efforts of setting up the infrastructure is not there. There are lot of advantages are there in adopting and using iac tools.
1. selfservice
Most of the time the application being developed are deployed manually in the production environment, to facilitate the deployment process we have a small number of sysadmins formed as a team, will be handling the delivery responsibility of our application. If the teams grows more in number, the dependencies on the sysadmins team will be more and becomes a big bottleneck in the delivery process of our application. Instead if we can automate the delivery process of our application through iac tools, the developer he himself can take the responsibility of delivering the application whenever it is required.

2. Speed and Safety
Instead of manually setting up the infrastructure which is time taking as well as error prone, if we can automate the process of creating infrastructure through iac tools, we can build infrastructure with no time and the repeatability of the infrastructure without errors will be very high

3. Documentation
If we present the final state of our infrastructure interms of code, we dont need to be dependent on sysadmin
's to understand our deployment architecture. If for any reason a sysadmin has left from the job and is replaced by another sysadmin with little or no dependency the new admin can take up responsibility easily by understand the infrastructure we expressed in code, which indicates iac

acts as a good documentation engine as well.

4. Version Control
As we express the infrastructure resources interms of code, we can push the iac source files into the source code versioning repositories, by which we get a chance of maintaining the versioning of our infrastructure. we can easily understand how did we evolved to the current state of infrastructure by going through the history of changes we made in scm tools. Any point of time if we want rollback our infrastructure state to previous we can pull that from the history and can be applied.

5. Validation
If we express the infrastructure resources interms of code, for every change we make in our infrastructure interms of code, can be reviewed by the peers. along with that we can apply set of automation tests also to verify the changes we made are safe before applying on production environment.

6. Reusablity
We can write iac code as libraries/code module can be stored. so while creating infrastructure across various application, we can reuse the code modules we created to speed up the infrastructure coding.

7. Collaboration and Sharing
As we expressed our infrastructure interms of code, once we push our code changes in scm repositories, the other memebers of our team can pull the source code and can use it / make further changes on our infrastructure.

iac (infrastructure as code) tools let us express the infrastructure requirements interms of code, so that we can quickly create the infrastructure out of the code. it is studied/proven that application who adopted iac tools has the better rate of delivery of the application when compared with others.
benefits:-
1. developer byhimself can handle the entire process of development to delivery of the application without dependening on the sysadmins
2. speed and safety / repeatability
3. versioning
4. documentation
5. validation
6. Reusability
7. Collaboration and Sharing

By now we come across lot of tools that addresses various different aspects of provisioning and deployment automation of an application, like terraform, openstack heat, ansible, puppet, chef, docker, packer etc. how do we distinguish and identify which tools has to be used when
let us derive the boundaries and factors to be consider in using these tools

1. Configuration Management versus Provisioning Automation Tools
There are Software Configuration management tools like ansible, chef, puppet and salt are there to install and configure software, out of which few of the tools like ansible supports creating infrastructure as well similary we have provisioning automation tools like Terraform and OpenStack heat which are going to help us in performing software configuration management automation to some extent.
Looks from the above both the tools can be used inter-changeably, but it is not recommended.

If we are using server templating tools like packer or docker most of the software configuration management requirements are handled by these tools we dont need to use ansible/puppet/chef here but to provision infrastructure in running these images/templates we need iac tools

If we are not user server templating in running the application then for software configuration management we need to use ansible|puppet|chef etc and for provisioning automation use iac tools both in combination.

2. Mutable Infrastructure versus immuntable infrastructure
While using software configuration management tools in provisioning an infrastructure, if we made any changes in the infrastructure post provisioning and try to re-apply the changes, they we go and update the existing infrastructure incrementally
But in case of iac tools they destroy the current infrastructure that has been created and tries to create new infrastructure resources from scratch.

Each of them has their own advantages and dis-advantages
For eg.. we used configuration management tool for managing the software installation and configuration or infrastructure provisioning.
-> apache2 server -> rewriting
-> apache2 modules
- http proxy
- mod rewrite
- reverse proxy
- mod redirect ---> production

---

3. Procedural language versus declarative language
In procedural langauges we need to write the code and express the steps in building the

infrastructure through programming, whereas in a declartive langauges we define the final state of the system by expressing interms of resource definitions, we need tell how to achieve the final state, the underlying tools like terraform reads these definitions and build the final state

---

4. Master verus Masterless, Agent vs Agentless
These tools are available in different architectures few are
1.requires central master sever where we publish changes and pulled by all the fleet servers of our infrastructure and apply through agents
2. masterless and agentless systems - where we can use any computer as a control node on using which we can connect to the fleet servers and apply the changes

Master and Agent systems
#1 dis-advantages:-
1. Dedicated Machine should be allocated which is going to be extra infrastructure overhead and for high availability we need to one more server to the Master
2. Migrating the master from machine to another is an headache
3. to allow all the nodes of the fleet to connect and pull changes from master we need to open lot of ports which is going to be a security breach

Masterless and Agentless systems:-
light weight but cannot schedule the changes

We can express the infrastructure requirements of an application interms of code using iac so that we can achieve more frequent delivery of the application when compared with otherway.
advantages:-
1. selfservice
2. versioning
3. documentation
4. speed and safety
5. validation
6. collaboration and sharing
7. code reusablity

How can we compare various different tools that exists in provisioning automation and software configuration management automation?
1. choose wisely between software configuration management automation tools from provisioning tools
if we are using packer or docker we just need provisioning automation tools we dont need software configuration management automation tools as most of the configuration management is handled by server templating tools
if we are not using server templating tools then a combination of both software configuration management and provisioning tools has to be used.

2. Mutable vs Immutable Infrastructure
The software configuration management tools are muttable, rather than destroying the existing infrastructure when a change is made, they update the existing infrastructure to bring the system to final state
In case of provisioning automation tools always they destroy the existing infrastructure and build new from scratch which is termed as immutable

3. Procedural vs declarative programming

Software configuration management automation tools are procedural where we have to write the program to achieve the final system state.

For eg.. give a case where we used ansible to create ec2 instance by using the below playbook
task:
- name: provision ec2 instance
amazon.aws.ec2:
instance_type: t2.micro
image: ami-292933
count: 5
with the above playbook configuration ansible provisions 5 ec2 instances on aws. now we want to upgrade to 15 instances so if we make a change in count of instances to 15 in playbook as below.
task:
- name: provision ec2 instance
amazon.aws.ec2:
instance_type: t2.micro
image: ami-292933
count: 15
and rerun the playbook, now ansible creates 15 more instances totally making 20, so its the responsibility of the devops engineer to identify the existing infrastructure and write your configuration files to achieve the final system state.

Whereas in case of provisioning automation tools, these tools keep track of the existing

infrastructure and tries to compare with the manifest/configuration we provided and determins what best we do to build the infrastructure to the final state
The above can be achieved easily using terraform as below,

```
resource aws_ec2_instance.javaservers {
ami: ami-2922233
instance_type: t2.micro
count: 5
}
```

The above configuration creates 5 ec2 instances, now if we want 15 instances totally, we dont need to worry about how many instances are there in existing infrastructure and count for remaining, rather simply specify the desired instances we want to terraform/provisioning automation tool as below

```
resource aws_ec2_instance.javaservers {
ami: ami-2922233
instance_type: t2.micro
count: 15
}
```

Now terraform queries the existing infrastructure and determines the plan of achieving the final system state now it understands there are already 5 instances in aws and it brings up 10 more instances to make it 15

4. Master versus Masterless
We have a Master and slave architecture, where all the manifest are uploaded to the master node and slaves pull the manifest and apply periodically from the master
1. Extra infrastructure cost in maintaning the Master node:
We need a dedicated machine and for HA we need a backup instance through which we should apply manifest on the fleet/slaves
2. Migrating the master is a pain point
3. less secured = we need to open ports on all the slaves allowing them to connect to the master

5. Agent and Agentless
To have the nodes of the infrastructure to connect to the master we need a piece of software to be running across all the nodes of the infrastructure which is called agent software.
1. How to install the agents across all the nodes of the infrastructure, again we need software configuration management tools to have this software configuration management tool to use (chicked-egg problem)
2. upgrading the agent softwares across all the nodes of the infrastructure
3. trouble shooting the infrastructure is going to be very difficult
---
From the above we can cleary distinct that for infrastructure provisioning it is highly recommended to use iac tools rather than software configuration management tools.
There are lot of infrastructure automation tools are available like
1. terraform
2. cloud formation
3. openstack heat

Terraform = is an infrastructure automation tool that is built by hashicorp. Terraform works in provisioning the infrastructure across various different cloud providers like aws cloud platform, google cloud, pcf, oci and azure etc. Its an cross-platform infrastructure provisioning automation tool

Terraform = is a infrastructure provisioning automation tool, that works with various different cloud providers in provisioning the infrastructure.

How does it works?

Cloud Providers = There are various different cloud providers/platform available in the market like aws, gcp, azure and oci etc.
These cloud platforms offers different services/resources as part of them. To enable provisioning of the services and resources of these cloud providers we use cloud consoles, but human interaction is required in provisioning the infrastructure on the cloud platforms.

We are looking for seemless automation of integrating infrastructure with application programs deployment,which cannot be achieved unless we have an ability of provisioning the infrastructure through programs rather than cloud consoles.

To facilitate provisioning the infrastructure through programs, every cloud provider offers api's.
cloud api's = Those are programmatic interfaces through which people can interact through an application in provisioning the infrastructure on that provider.

Terraform
Terraform inorder to automate the process of creating the infrastructure across various cloud platforms, it should use the cloud platform api's as the cloud providers allows to interact with cloud platform only through api's externally

The apis are different from one cloud provider from the another, because their underlying services and resources they offer are completely different

Then how can terraform can provision infrastructure across various cloud providers, here comes provider plugins.

provider plugin = each provider has their own provider plugin for e.g.. aws has aws provider plugin and azure has azure provider plugin. The provider plugin knows the services of the cloud platform and the corresponding apis that should be invoked to create a respective service.

Now the devops engineer will write a terraform configuration describing the resources we want to create with a cloud provider and passes the terraform configuration to terraform engine.
Terraform (CLI):
1. Terraform reads the terraform configuration and validates the configuration is valid or not
2. identifies the respective cloud provider we have specified in terrafrom configuration
3. downloads the provider plugin into the workspace
4. passes the terraform configuration to the provider plugin asking to plan and apply the infrastructure.

Now the provider plugin goes through the resource definitions we expressed in terraform configuration and identifies the appropriate apis to be invoked on cloud provider makes a network call to the cloud provider by passing the relevant inputs in creating the resource on that provider.

With the Terraform we are able to achieve unified workflow in provisioning the infrastructure across all the cloud providers.

What are the components in Terraform?
1. Terraform CLI = reads the terraform configuration, validates and invokes the respective cloud provider plugin

2. Terraform Configuration = Its written in HCL (Hashi crop language) which contains resource definition that has to be created on cloud platform of a provider, this is written by terraform developer
3. Provider Plugin = Each provider writes a plugin byhimself to get the resources of platform to be provisioned through Terraform. The provider plugin reads the resource definitions and makes an api call of their provider
4. Cloud Provider api = its an programmatic interface through which people can create/provision infrastructure on a cloud platform

---

Note:-
Terraform is a cross-platform infrastructure automation tool that works across multiple cloud providers in provisioning infrastructure, but the terraform configuration is not portable across the multiple cloud providers.

For eg.. we wrote a terraform configuration to create an ec2 instance on aws, the same configuration cannot be applied on azure/pcf because those cloud platforms doesnt offer a ec2 resource.

Download and Install Terraform CLI
Terraform CLI is a Command-Line utility program (single binary) that works across all the platforms, we just need to download and add to the System Path of our operating system.

The machine on which we have Terraform CLI installed and available is called Terraform Control Node, from here we can apply terraform configuration

---

Aws setup to use with Terraform

In order to provision infrastructure using terraform on a cloud provider we need an cloud provider account lets say an aws cloud platform account.

There are 2 types of aws users are there
1. aws account administrator
2. aws account user

dont configure terraform to work with aws account administrator, as it provides an unrestricted access to the cloud platform, if something goes wrong in our terraform configuration, an unexpected behaviour might result on the cloud platform which could cause damage like huge billing or might delete existing infrastructure.

So always create an Aws account user with limited access being enfored through IAM Policies and permit to be used as part of terraform to provision infrastructure. Now terraform/provider plugin to allow access to the aws provider apis we need to generate api keys / access keys of aws user and pass it as input to the terraform allowing to access the apis onbehalf of the user.

Now aws generates access key id and access key code using which we need to access the aws apis. Now we need to pass the access_key_id and access_key_code to terraform enabling it to access the aws cloud apis on behalf of us. We can download the access keys as a csv file from cloud console

There are 3 ways in passing the access_key_id and access_key_code information to terraform allow it to access aws apis

#1 shared file : we need to create .aws directory under the user home directory of the computer

and create afile name credentials. in this file we need to place access_key and code, since we have multiple user keys or mutiple accounts information we have place them under profile name

credentails
[finance] - profile
aws-access-key-id=
aws-access-key-code=

[hr]
aws_access_key_id=
aws_secret_access_key=
Now in terraform configuration we specify which profile access keys should be used for connecting to cloud apis in provisioning the infrastructure
---
#2 through environment variables
Before running the terraform to create infrastructure resources, we need to set the access_key_id and secret_access_key as environment variables

set AWS_ACCESS_KEY_ID=
set AWS_SECRET_ACCESS_KEY=

#3 we can embed the aws access key and secret as part of terraform configuration file itself
This is not high recommended, since we push the terrafroms into git repositories, so that our access keys will be exposed.

[DONT USE THIS]
provider "aws" {
access_key = ""
secret_key = ""
}

How to provision infrastructure through Terraform?

To provision the infrastructure resources through terraform on a cloud platform we need to write terraform configuration file. The terraform configuration file is 2 formats
1. HCL = hashicorp language
2. JSON = Terraform supports Json format in writing resource declarations
Preferrably uses HCL to write the configuration file

In terraform configuration file there are basically 2 types of delcarations are there
1. resource declaration
2. datasource declaration

1. resource declaration = it represents the resource we want to provision on the cloud platform. it contains the resource configuration with which we wanted to create the resource on the platform

resource "TYPE" "NAME" {
resource configuration
}
A resource declaration contains 2 things
1. TYPE = indicates the type of the resource we want to provision. for every cloud provider the resource type always starts with cloudprovider_resourceType
aws_ec2_resource = ec2 instance
2. NAME = used for referring the resource in another resources (to express resource

dependencies)

```
resource "aws_security_group" "sg1" {
allow {
protocol:
cidr:
port:
}
}

resource "aws_ec2_instance" "javaserverec2" {
instance_type = "t2.micro"
ami = "ami-3933444"
security_group_id = aws_security_group.sg1
}
```

In the above example we used resource name sg1 to refer and use that security group in creating ec2 instance. by referring the resourceName terraform identifies the resource dependencies automatically.

When we ask terraform to create infrastructure by using terraform configuration above, terraform generates an acyclic graph representing resources and their dependencies and creates in the order of dependencies automatically.

Each provider has different types of resources so there are various resource declarations with different configurations so the only way to configure the resources of the provider refer to provider resource configuration documentation provided by the terraform.

2. datasource declarations
will be used for querying the existing resources that are available on cloud platfrom, so that using these references we can create resources.
For eg.. we want to provision an ec2 instance on an existing subnet of a vpc. we can query the subnet and vpc information using datasource and can pass it as an input to ec2 resource declaration.

How to write Terraform configuration file?
In order to provision infrastructure using terraform we need to create terraform configuration as below.
1.create a project/workspace directory
d:\> ec2 (directory)
2. within this directory we need to write terraform configuration file
The terraform configuration file must and should be written with an extension as *.tf which indicates terraform file
usually people write the resource declarations in a terraform file preferrable with name as main.tf
3. we can write any number of *.tf files under the project directory with any name and when we ask terraform to create infrastructure it takes all the *tf files merges into one single file logically and applies the infrastructure

to let terraform identify the resource/datasource declarations we declared should be executed on which cloud platfrom we need to write providers declaration in the file

main.tf

```
terraform {
required_providers {
aws = {
source = "hashicorp/aws"
version = "3.36.0"
}
}
}
provider "aws" {
region = "ap-south-1"
profile = "default"
}

resource "aws_ec2_instance" "javaserverec2" {
instance_type = "t2.micro"
ami = "ami-0d758c1134823146a"
}
```

How to run terraform configuration?

1. be in the directory of the project
2. terraform init

when we issue terraform init, it reads all the *.tf files and consolidates, the validates the configuration is valid or not. If we valid immediately downloads the terrform provider plugin and stores inside the project directory under .terraform

we can safely run terraform init for any number of times

3. terraform plan

before applying the infrastructure configuration we can verify whether the infrastructure we want create is valid or not by running plan

Terraform goes to the aws cloud platform verifies what resources are there and compares with the terraform configuration we provided and generates a report displaying which resources are existing, which will be destroyed and what resources will be new created.

if plan generated looks good, then we can run

4. terraform apply

This will creates or executes the terrform configuration on cloud provider resulting in creating the final state of the infrastructure.

How to work with variables in Terraform?
Variables are the placeholders which are meant for storing the values. In terraform configuration file we write configuration values pertaining to the infrastructure/resource we want to provision, for eg.. for "aws_instance" resource we write instance_type = "t2.micro", this is applicable in development environment, but when we switch to a qa/production environment we cannot provision an instance of "t2.micro" shape we need a bigger machine.
So each time when we want a different configuration with which we want provision the infrastructure resource we need to modify or rewrite the configuration file. Instead of this we can use variables in configuration file and we can pass inputs to those variables at the time of applying the terraform configuration.

There are 3 parts in working with variables.
1. how to declare variables?
2. how to refer variables in terraform configuration file (hcl)
3. how to pass the values to those variables?

1. How to declare variables?
Inside the terraform configuration we can declare a variable in variable declaration block.

```
variable "aws_region" {
type = string
description = "region in which we want to provision the infrastructure"
default = "ap-south-1"
}
```

In the above default value is defined, then the variable is considered as optional, and even we didnt supply the value terraform will not prompt for the value, rather it applies the configuration using default value.

It is often recommended to keep your variable declarations separate from terraform configuraton *.tf by writing them in another file of the project directory with extension *.tf, the common best practise is to keep the variables in a file "variables.tf"
There are different types of variables are supported by the terraform
- string
- list
- map
- bool
- number
- set
- object

```
variable "subnets" {
type = list
description = "subnet cidrs"
default = ["10.0.1.0/24","10.0.2.0/24"]
}

variable "ec2object" {
type = object( {
instance_type = string
ami = string
Name = string
})
}
```

```
variable "tags" {
type = map
default = {
Name = "javaec2"
Organization = "railreservation"
}
}
```

2. How to use the variables we declared?
main.tf

```
terraform {
required_providers {
aws = {
source = "hashicorp/aws"
version = "3.3.6.0"
}
}
}
provider aws {
region = var.aws_region
profile = "default"
}

resource "aws_instance" "javaec2" {
instance_type = var.ec2object.instance_type
ami = var.ec2object.ami
tags = {
Name = var.tags['Name']
}
}
```

3. How to pass the values for the variables while applying the terraform configuration?
When we declare a variable without any default value and if we refer the variable in terraform configuration as shown below.

variables.tf

```
variable "aws_region" {
type = string
}
```

main.tf

```
provider "aws" {
region = var.aws_region
profile = "default"
}
```

when we apply the above terraform configuration without passing any value or without default declaration, then during apply terraform prompts user on the command-prompt asking to supply the value that variable. Here the problem is human intervention, because of this we cant achieve automation. Instead of this we can supply values directly at the time of applying the terraform

configuration

Note:- when we supply values at the time of applying the terraform configuration, the default values will be overwritten.

There are 4 ways in passing the values for the input variables we defined in Terraform configuration.
1. While running the terraform apply we can pass values for the variables using -var switch
terraform apply -var aws_region=ap-south-2 -var ec2object={instance_type=t2.micro, ami=am033933}

2. rather than passing the variables in command-prompt while applying, we can declare values for the variables in *.tfvars and pass them as input while applying terraform

variables.tf
---
variable "aws_region" {
type = string
}

standard best practise or convention followed is define the values in input.tfvars file and place it inside the project directory

inputs.tfvars
---
aws_region=ap-south-1
tags = {
Name = javaec2
Organization = Finance
}

Now while running terraform apply we need to pass the variable file as input
terraform apply -tfvars=inputs.tfvars

3. instead of passing the *.tfvars file manually while running the terraform apply we can follow standard convention in naming this file as *.auto.tfvars in the project directory.
If there appears a filename with *.auto.tfvars terraform automatically picks the file values and applies we dont need to pass variable file.

4. we can pass variable values as input through environment variables
for e.g.. we defined a variable aws_region in inputs.tf
variable "aws_region" {
type = string
}

before running the terraform apply we need to set value for this variable through environment variable at command-prompt as
export TF_VAR_aws_region=ap-south-1
every variable while defining the value through environment variable should be prefixed with TF_VAR_variableName = value

terraform apply = now terraform looks for environment variable and will run without prompting for the value.

What are output variables, how do we define them in terraform?
Output variables are similar to a function return value in programs, where those can be used as input into another resource or can be used for displaying the output on CLI.
We can declare output variables in *.tf files only and it is usually recommended to declare output variables in a separate file other than main.tf, and the standard naming convention people follows is outputs.tf. we declare an output variable using output block.

outputs.tf
output "javaec2_publicip" {
value = expression [pointing to a resource attribute]
description = "information about the output value"
sensitive = true/false (should display or not in console)
}
output "javaec2_publicip" {
value = aws_instance.javaec2.public_ip
description = "java ec2 instance public ip address"
}

ec2-on-public-subnet
|-variables.tf
|-outputs.tf
|-main.tf
|-inputs.tfvars

inputs.tfvars
aws_region = "ap-south1"
localnews_vpc_cidr = "10.0.0.0/16"

variables.tf
---
variable "aws_region" {
type = string
description = "aws region"
}
variable "localnews_vpc_cidr" {
type = string
}

main.tf
---
terraform {
required_providers {
aws {
source = "hashicorp/aws"
version = "3.36.0"
}
}
}

provider "aws" {
region = var.aws_region
profile = "default"
}

```
resource aws_vpc "localnewsvpc" {
cidr_block = var.localnews_vpc_cidr
tags {
Name = "localnewsvpc"
}
}

resource aws_subnet "localnews_private_subnet" {
cidr_block = "10.0.1.0/24"
vpc_id = aws_vpc.localnewsvpc.id
availability_zone = "ap-south-1a"

tags {
Name = "localnews_private_subnet"
}
}


resource aws_subnet "localnews_public_subnet" {
cidr_block = "10.0.2.0/24"
vpc_id = aws_vpc.localnewsvpc.id
availability_zone = "ap-south-1a"

tags {
Name = "localnews_public_subnet"
}
}

resource aws_internet_gateway "localnews_internet_gateway" {
vpc_id = aws_vpc.localnewsvpc.id
tags {
Name = "localnews_internet_gateway"
}
}

resource aws_route_table "localnew_internet_gateway_route_table" {
vpc_id =  aws_vpc.localnewsvpc.id

route {
cidr_block = "0.0.0.0/0"
gateway_id = aws_internet_gateway.localnews_internet_gateway.id
}
}

resource aws_route_table_association "localnews_route_table_to_public_subnet_association" {
subnet_id = aws_subnet.localnews_public_subnet.id
route_table_id = aws_route_table.localnew_internet_gateway_route_table.id
}

resource aws_security_group "localnews_ec2_security_group" {
name = "localnews_ec2_sg"
vpc_id = aws_vpc.localnewsvpc.id
ingress {
```

```
description = "allow ssh access"
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"] // source ip
from_port = 22
to_port = 22
}
egress {
description = "global"
protocol = "-1"
cidr_blocks = ["0.0.0.0/0"] // destination ip
from_port = 0
to_port = 0
}
}

resource "aws_network_nacl" "localnews_network_nacl" {
vpc_id = aws_vpc.localnewsvpc.id
ingress {
rule_no = 100
protocol = "tcp"
action = "Allow"
cidr_block = "0.0.0.0/0"
from_port = 22
to_port = 22
}

egress {
rule_no = 100
protocol = "tcp"
action = "Allow"
cidr_block = "0.0.0.0/0"
from_port = 22
to_port = 22
}
}

resource "aws_key_pair" "localnews_key_pair" {
key_name = "localnews_key_pair"
public_key = "ssh key"
}

resource aws_instance "localnews_ec2" {
vpc_id = aws_vpc.localnewsvpc.id
subnet_id = aws_subnet.localnews_public_subnet.id
security_group.id = aws_security_group.localnews_ec2_security_group.id
key_pair_name = aws_key_pair.localnews_key_pair.name
instance_type = "t2.micro"
ami = "ami-29393"
tags {
Name = "localnews_ec2"
}
}
```

Terraform is an iac tool that is used for provisioning the cloud infrastructure, but what about software installtations and configuration we want to perform on the infrastructure to achieve final system state?
Terraform doesnt helps us in installating and configuration of the softwares on the infrastructure.

How to solve this problem?
There are 2 solutions for this.
1. use software configuration management tools like ansible, puppet, chef or salt for installation and configuration of software on the provisioned infrastructure
2. use containerization tools and ami images to build infrastructure and run containers

Instead of we manually performing the above steps of running ansible/puppet/chef scripts after provisioning the infrastructure terraform would help us in executing post provisioning automatically by using provisioners.
Terraform supports 3 types of provisioners:-
1. file provisioner = copies the files or directories from the path specified on to the resource
2. local-exec provisioner = runs a the provisioner scripts on the local machine where terraform control node is there
3. remote-exec provisioner = copies the provisioner script on to the resource and will execute.

provisioners can be declared at 2 levels
1. at the resource level
2. independently triggered at the last state of the infrastructure

Cloud Computing = is an on-demand availability of computing resources to the end-users. They day to day infrastructure requirements of running your software applications on the computing resources can be made available through cloud computing.
There are 3 services offered by the cloud computing vendors.
iaas = infrastructure as a service
paas = platform as a a service
saas = software as a service

depends on the cloud computing vendor they provide all of the above services are could be few of them as well.

iaas = infrastructure as a service provides the infrastructure aspects of running the software applications. The cloud vendors provides virtual resources using virtualization technology rather than providing physical machines to us so that they can serve the machines based on the requirements of the users.

in iaas we can create a machine of a customized shape based on the requirement of your business.We never endup in paying more for un-used resources of the machine. We can always can increase or decrease the hardware resources of our machine on demand within couple of minutes.

To manage and provide virtual machines on demand the cloud providers uses some softwares called hypervisors and special softwares for managing these hypervisors as well.
We can signup for using a machine for few hours as well and even for months/years there is no upfront comittment of usage of machines.

All the cloud services are offered in 2 billing models
1. pay-per usage model
When we take a machine from cloud vendor, if we dont use it and turn it off we never need to pay. pay for only the hours of usage.
Pay for the amount of data transfer you do the network rather than paying for the communication line
Pay for the storage allocated to you.

2. fixed rate
You upfront reserver a computer of fixed shape and endup paying the same cost for the whole year irrespective of the usage.

Terraform is an iac automation tool that helps in provisioning infrastructure but it dont configure or install any software on the machines that are provisioned. To install and configuration software on these infrastructure we need to use Software configuration management tools like
- chef
- puppet
- ansible

post provisioning of the infrastructure now devops engineer has to run these software configuration management scripts manually to have the software being installed and configured. but this requires a manual interaction and results in lot of time and human mistakes (we might forget to apply these on infrastructure)

To overcome the above challenges Terraform has provided provisioners, which would be executed automatically at the end of the provisioning of infrastructure, so that we can execute software configuration management scripts to apply software installation and configuration process.
Terraform supports 3 types of provisioners
1. file provisioner = used for copying files on to the remote resource
2. local_exec provisioner = running shell or any software configuration management scripts locally on terraform control node
3. remote_exec provisioner = it copies the shell script or software configuration management script on to the remote provisioned resource and executes on the remote machine.

We can write the provisioners at 2 different places in terraform configuration file
1. local to the resource level
2. independently written and executed at the last

1. local to the resource level

```
resource "aws_instance" "javainstance" {
ami = "ami-3938444555"
instance_type = "t2.micro"

connection {
type = "ssh"
timeout = "5m"
host = self.public_ip
user = "ubuntu"
private_key = file(~/.ssh/id_rsa)
}
provisioner "file" {
source = "installjdk.sh"
destination = "/tmp/installjdk.sh"
}
provisioner "remote-exec" {
inline = [
"chmod +x /tmp/installjdk.sh",
"/tmp/installjdk.sh"
]
}
}
```

connection block = represents the connection configuration to connect to the machine for executing provisioner.

What is cloud computing?
Cloud Computing is an on-demand supply of computing resources to the end-users. The cloud computer providers offers 3 types of services.
1. iaas = infrastructure as a service
2. paas = platform as a service
3. saas = software as a service

The cloud providers offers the computing resources whether it could be a machine|network| software on below principles
1. on-demand = whenever we request a machine or a network or a software application they provide immediately with little or no latency.

2. elastic = the resource they offer are flexible where we can scaleup these resources or scaledown these resource with minimal effect or no effect on resource availability.

3. billing
There are 2 billing models the cloud providers are going to offer
1.pay per usage = each of the resources we are using will be billed based on the usage of them, we dont need to pay for the whole month even we use the resources per a day or 2
The pay per usage model differs from resource type to resource

fixed shapes
Debain Ubuntu 20.04 | 2.4 ghz | 4 gb | 10 gb = 0.001$/per sec = 24/7 - 30day
Windows 10 | 2.4 ghz | 4 gb | 10 gb = 5$ / per hour

2.fixed rate
If we are using the resources with a sustained usage then the cummulative cost in pay-per-usage model will be little high, so cloud providers offers fixed pricing model where irrespective of hours of usage of the machine they fix price for each of the resource on monthly bases.
So that we are being charged on fixed amount rather than usage.

The above services are offered by the cloud providers using different technologies like virtualization and bare metal models which indicates they dont serve physical systems , rather they provide virtual resources to the end user.
---

1. iaas = infrastructure as a service (almost every cloud providers serves iaas)
The cloud providers offers the physical resources like machine, network, routers, firewalls etc on-demand for usage. Like how do we buy a computer from a computer market and use it. here also we are going to get a compute machine baked with an operating system with network resources attached.

We can install our own software packages and we can deploy or run our own applications on these compute machines. They provide a dedicated ssh access to these machines using which we can access the compute instances locally from our computer.
---

2. platform as a service
To run software applications that are built by the end-user we not only need a compute machine, we need middleware software to be installed on that computer.

Middleware = The software that we use underlying to make the end-user application work on a computer is called "Middleware" software.

For e.g..
1. jdk software
2. .net sdk
3. python
4. nodejs
5. npm
6. database server like mysql server, mssql server, oracle database server
7. mongo db
9. tomcat, weblogic server, websphere server, glassfish server, jboss wildfly server
10. Message queues (kafka) | (rabitmq) | (mqserver)

Now if we take a iaas compute machine, we cannot directly run end-user software on the machine, because we need middleware software to be installed. To have them install on compute instance, the end-user has to manually setup the software packages as the same way he would install locally on his machine.

Manually installing, configuring and managing the middleware softwares has lot of problems.
1. To setup the middleware software on the compute instances takes lot of time, we may to do several activities to have these middleware software running on the compute instance like, configuring firewalls, security rules, writing configuration files etc, this delays the deployment and delivery of the application

2. periodically software updates and patches will be released by middleware vendors, it is highly critical to ensure these patches are applied on the compute instances, where we are using the middleware software. uptaking these patches are not so easy
2.1 we need to ensure the patch that is released by the middleware is safe to use and will not break the system we need to check. So we have to setup test env equal to our prod and apply the patch on test env and perform through testing then apply the patches on the prod env by taking the prod env backup.

2.2 periodically apply these patches is going to be quite difficult job, certifying and applying the job requires a separate team to be maintained which is going to be costly

3 when we want to scale the compute instances, we not only need to take another compute instance from the cloud provider, again we need to re-setup the middleware software and do necessary configurations to make both machines work as a cluster.

Instead of we taking care of installing/configuring/managing the middleware software cloud providers offers paas services to us. For each of the middleware software there is a paas offering. For eg..

1. java as a cloud service = a compute instance with jdk software being installed will be provided to us directly
2. weblogic server as a cloud service = the provision a compute instance with weblogic server installed and configured for ready to usage.
3. mysql server as a cloud service = they provision a machine with msql server database installed configured with firewall, security rules so that the database is not comprised by the attacks
3.1 periodical backup of mysql server database will be taken care
3.2 crash recovery
3.3 backup and restoration in event of failure
3.4 on heavy loads scale the mysql server on mutiple machines

3.5 including licensing fees
---
saas = software as a service

To serve an organization requirement we need to usually build software applications.
Financial system, of a company (automobiles) like audits, accounts, balance sheet.
- We need build a software system that can take care of handling financial aspects of the company

being an automobile business vendor, he himself cannot build an software application for their organization, they need outsource the software requirements to another software services company. To have the software application being build by the outsourced company we need to do the following.
1. our business team regularly should be in touch with outsourced company in providing our requirements.
2. allocating budget and planned release of financial resources to the other company based on software progress
3. monitoring the development process
4. verification has to be done regularly to ensure the people build the software as per the expectition

On top of the above we need to invest big way money for building software application and wait for a long-term period to get it build and delivered. where the investment that made becomes dead util the software application build comes under use.

Financial management system software is an Enterprise need that every requires to maintain their financial records. There are few technofunctional experts who has both financial domain knowledge and software development knowledge, using their enourmous amount of experience knowledge they build common software system we can be categorized as product that can be installed and used directly by enterprises in the market which are called "ERP" Systems.
- salesforce
- oracle fusion apps
- ebusiness suite
- peoplesoft
- jd adwards
- sap
- tally

Now our company dont need to build software from scratch, rather we can procure license for any of the ERP Softwares and can adopt and use it in our organization.

To adopt these ERP there is a lot of difficulties or challenges are involved.
1. To run these ERP softwares on the organization infrastructure, we need high-grade server machines with huge computing capacity required. The cost of setting up such an infrastructure is very high
2. The licensing cost of these ERP very high where a moderate or small company may not afford it.
3. Requires consultants in adopting these ERP Systems in our organization, because they need special installation and configurations to run on a machine
4. ERP systems has to be customized based on the organization requirement, which is called implementation/adoption of the ERP requires special manpower trained on this job

5. 24X7 monotoring | backup | recovery and regular maintainance activities has to be conducted to ensure these systems works. This requires a crew in doing this job
From the above we cannot understand clearly these can be used only Enterprises only. To rescue from this problem saas (software as a service) comes into picture.

Software as a Service (SaaS) = provides these enterprise software systems pre-installed and pre-configured on the iaas layer.

SaaS works in 2 ways
- multi-tenancy model = one erp system will be shared to multiple users on a multi-tenancy model, so that each enterprise which is using the erp system believes they have their own dedicated erp system for their use even though the underlying infrastructure and erp system is shared across multiple enterprises.

The advantage of multi-tenant model here is
1. keeping the cost of using the ERP is very low
2. Small and Moderate enterprises can make use of ERP in running their business

- private cloud
- a dedicated machine is alloted with saas implementation on it so that your organization can use it without sharing with others.

In generate with SaaS model we have plenty of advantages
3. No installation/configuration is required since cloud provider takes of them
4. backup / recovery and restoration in the event of crash will be taken care by cloud provider
5. autoscaling based on traffic and hanlding performance issues at peak load.

By the above we can understand a cloud provider offers various services and manages their lifecycle operations.
1. provisioning
2. scaleup
3. scaledown
4. scaleout
5. scalein
6. backup
7. restore
8. deprovisioning

local-exec provisioner runs a shell script locally on the terraform control node

---

```
resource "aws_instance" "localprovec2" {
instance_type = "t2.micro"
ami = "ami-3939304"

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

We can write provisioners at 2 different levels.
1. local to the resource level
2. independently outside at the last whereever we want to execute

2. independent provisioners why?
if we want to perform an operation after a group of resources has been created, then rather than writing local to the resource we can write independently.

for having provisioners to be independent out of a resource, terraform as provided a special resource called "null_resource" = indicates it doesnt create any resource, but used for attaching provisioners to be executed.

we need to specify a trigger point telling the provisioners of the provisioner should be executed only after something was populated/changed on the resource.

```
resource "aws_instance" "ec2" {
instance_type = "t2.micro"
ami = "ami-29383"
}

resource "null_resource" "install_java" {
triggers {
public_ip = aws_instance.ec2.public_ip
}
connection {
type = "ssh"
timeout = "3m"
host = aws_instance.ec2.public_ip
user = "ubuntu"
private_key = file(privatekey_loc)
}
provisioner "remote-exec" {
inline = [
"chmod +x /tmp/install-java.sh,
/tmp/install-java.sh"
]
}
}
```

Terraform Modules

Terraform modules are group of resources that can be used reused by importing as modules in

various terraform configurations. We can create our own local modules and can reuse in creating different infrasturctures or there are lot of pre-defined modules published as part of terraform registry, we can use those modules to create the infrastructure quickly.

By default every terraform configuration has one default module called root. within the root module we can call child modules. now while calling the child modules we can pass inputs and child modules upon completing execution can return outputs also similar to a function call in programming.

The things we learnt till now are going to be same, but the way we are organizing the code differs.
How to work on modules?

ec2apache2
|-modules
|-services
|-network
|-vpc
|-main.tf
|-variables.tf
|-output.tf
|-subnet
|-main.tf
|-variables.tf
|-output.tf

|-global (root) (main terraform configuration, here we use modules)
|-variables.tf
|-input.tfvars
|-output.tf
|-main.tf

ec2apache2/modules/services/network/vpc/variables.tf

```
variable "vpc_name" {
type = string
}
variable "cidr_block" {
type = string
}
```

here in main.tf we should not write terraform block and provider block it should contain only resource declarations since these resource are used as part of root module, root contains those declarations.

ec2apache2/modules/services/network/vpc/main.tf

```
resource "aws_vpc" "global_news_vpc" {
cidr_block = var.cidr_block
tags = {
Name = var.vpc_name
}
}
```

ec2apache2/modules/services/network/vpc/outputs.tf

```
output "vpc_id" {
value = aws_vpc.global_news_vpc.id
}
```

ec2apache2/modules/services/network/subnet/variables.tf

```
variable "cidr_block" {
type = string
}

variable "vpc_id" {
type = string
}
```

ec2apache2/modules/services/network/subnet/main.tf

```
resource "aws_subnet" "global_news_subnet" {
vpc_id = var.vpc_id
cidr_block = var.cidr_block
availability_zone = "ap-south-1b"
}
```

ec2apache2/modules/services/network/subnet/output.tf

```
output "subnet_id" {
value = aws_subnet.global_news_subnet.id
}
```

[root module]
ec2apache2/global/main.tf

```
terraform {}
provider "aws" {}

module "vpc_module" {
source = "../modules/services/network/vpc"
cidr_block = "10.0.0.0/16"
vpc_name = "ec2_module_vpc"
}

module "subnet_module" {
source = "../modules/services/network/subnet"
cidr_block = "10.0.1.0/24"
vpc_id = vpc_module.vpc_id
}

module "subnet_module" {
source = "../modules/services/network/subnet"
cidr_block = "10.0.2.0/24"
vpc_id = vpc_module.vpc_id
}
```

What is cloud computing?
Providing the computing the resources/services on demand to the end-user is called "Cloud Computing". The computing services that are offered by the cloud provided can be classified into 2 categories

1. fully-managed services (saas)
2. customer-managed services (iaas)

The cloud providers takes care of managing the lifecycle aspects of the services based on the above 2 categories like
1. provisioning = creating the request service and provided to the end-user
2. de-provisioning = terminating the service
3. backup = periodical backup of the services to be taken automtically so that in event crash it can be used for restoring.
4. restore = in the failure of the service, cloud provided takes care of restoring the service by using the backup created earlier.
5. scaleup = increasing the computing resources of the instance or service like RAM, CPU, Storage, network capacity etc with minimal or no loss of the service
6. scaledown = decreasing the computing resources
7. scalein = removing the additional service/compute instance allocated when it is of no use
8. scaleout = adding an additional compute instance/service for High Availability or performance reason and loading balancing, clustering, traffic management and rotation of service on new instance will be taken care by could provider if it is fully-managed.
9. start = start the service instance
10. stop = stop using the service instance temporarily
11. restart = reboot the service instance

They offer 3 types of services.
1. iaas = infrastructure as a service, here only infrastructure resources on-demaind like compute instance, network, router, firewalls, gateways are provided to the end-user. These are not managed services, the end-user by himself should be responsible for managing the lifecycle aspects of the services.

2. paas = platform as a service, here along with compute instance, the middleware software required for running end-user applications also will be provided by cloud provider.So the end-user can quickly get his application deployed and running on the service instanced without bothering in managing the paas

3. saas = fully-managed service instances, where an end-user can get a readily available software to use it
usually the saas services are the ERP, EIS systems are build as SaaS models.

There are multiple cloud providers are there in the market, out of which the most popular cloud providers are
1. aws (amazon web services) [third-party sellers] (iaas)
3. gcp (google cloud platform)
[Machine Learning | Artificial Intelligence platform | Integration-Tier]
2. azure (microsoft cloud platform) (iaas/paas)
4. oci (oracle cloud infrastructure) (iaas|paas|saas)
5. Pivotal cloud Foundry

Different cloud providers offers different services to the market.

Every cloud provided offers cloud computing resources to the customers. How can the customers would be able to view and manage the resource they bought from a cloud provider? For this every cloud provider offers various different technics in managing their resources/ services underlying.

1. cloud console system = Mostly web console access so that end-user can interactively create services
2. cloud software development kit (sdk) = Cloud providers provides their own api libraries, using them we can build cloud provider services programmatically.
3. cloud cli = can be used for quickly grabbing the computing resources from the cloud providers. These will comes in handy when we want to perform one-time administrative activities on the services.
4. cloud api (rest)

8:00 am - 10:00 am
15 breaks (mins)

11:30 am - 12:30 am (1 hour)

devops
10:15 am = 11:15 am (1 hour)
4:00 pm - 5:00 pm

There are 4 ways a cloud provider allows us to access the cloud services.
1. cloud console a web application through which we can consume the services
2. cloud sdk of the provider
3. cloud cli
4. through cloud provider rest api

Aws provides lot of cloud services, an about 100 or even more. if those services has to be used we should identify and locate them to use, which is going to be difficult. AWS has grouped all the related services into domains based on the nature of the services.

In aws there are total 7 domains offerings are there
1. Compute
2. Storage
3. Database
4. Networking
5. Mangement Tools
6. Security
7. Messaging

#1 Compute Domain

The name itself says all the computing resources related services falls under this domain. There are total 5 services are offered under Compute domain.

1.1 ec2 (elastic cloud compute) = its also known as "compute instance" which is nothing but a machine of the desired configuration (shape) will be provisioned by the aws and provides to the end-user. while creating the ec2 instance, aws asks for which operating system to be installed on that instance, based on which it creates the instance with that operating system and gives to us. Now the end-user has to install middle-ware software and has to run his applications on top of it.

1.2 lamda = "serverless technology of aws". we can write functions that executes for a short interval of time. give this function to aws lamda, so that it executes the function by taking care of concurrency and scalability aspects of it.

1.3 elastic beanstalk = aws has predefined stack of middleware softwares being listed as part beanstalk which it automate in installing as part of the provisioning of ec2 instance. so, end-user can pick the pre-defined list of beanstalks one of it and can ask aws to provision.
AWS takes of provisioning the ec2 instance with that middleware software directly, so that end-user can directly run his software application.

1.4 elastic load balancer = distributes the application traffic between multiple ec2 instances

1.5 asg (auto scaling group) = based on the rules and threasholds we configure the asg takes care of scaleout and scalein of ec2 instances

#2 Storage Domain

All the services related to storing the data comes under Storage Domain. There are 2 types of storage services are supported by aws
1. object storage = an individual file or a doc or an image can be stored and access from an object storage database using an id generated.
2. block storage = huge volumes of data can be stored on a memory partition just like an harding of a computer.

There are total 6 storage services offered by aws grouped into one of the above 2 categories
1. s3 (simple storage service) = it is an object storage system provided by aws. we can store files|documents|images in the s3 storage service. We create buckets and we store objects in the buckets in an organized manner. The buckets can be private or public, if it is a public bucket anyone can access the objects we stored inside it. if it is a private bucket only permitted users can access. We can apply authorization rules like read|write|delete objects.

2. Cloud Front = is a content delivery network service which is used for caching the static resources like images, documents, files, css, js etc which are static resources local to the geographic locations of the users. So that whenever the request comes for a static resource, the cdn server takes of service the request based on geographic location quickly by reducing the load on the backend server. This improves the performance of our applicaiton.

3. Elastic block storage = is an block volume that is attached to an ec2 instance and acts as an harddisk for ec2 instance

4. Glacier = ec2 and rds (relational database service) instances has the data as part of them. Glacier is used for storing/backing up the ec2 or rds service data into a magnetic tapes (very cheap of cost). we can schedule periodical automatic backup of ec2|rds services data into glacier.

5. Snowball = snowball is a migrataionl service, where a special device named snowball is used for migrating the data from a datacenter to a aws infrastructure.

6. Storage Gateway = The snapshot of the data in database can be backedup to storage gateway periodically. so that if any crash in the database the data can be quickly restored back to database from storage gateway.

#3 Database domain

There are total 7 domains in aws
1. Compute
2. Storage
3. Database
4. Networking
5. Management
6. Security
7. Messaging

1. Compute = we offer computing related services under compute domain. There are total 5 services are there in compute domain
1.1 ec2
1.2 lamda
1.3 elastic beanstalk
1.4 elastic loadbalancer
1.5 auto scaling group (asg)
2. Storage = all the data storage management services are part of this domain. Total 6 services are there
We can classify the Storage Types into 2 categories
- object storage
- block storage

2.1 s3 (object storage)
2.2 cloud front (content delivery network) [caching services for the static data]
2.3 elastic block storage = harddisk for ec2 instance
2.4 snowball = data migration service to move from data center to aws infrastructure
2.5 glacier = magnetic tape backup for ec2 and rds instances
2.6 storage gateway = rds database snapshots are backed up into storage gateway

3. Database domain = All the database services are part of this domain. Total 5 service offerings are there as part of this domain.

1. RDS (Relational database service)
RDS by itself is not a database software, rather it is a managed service that manages the databases like mysql, oracle, postgress sql.
You can place a request to RDS asking for provisioning a mysql server | oracle | postgress database instances by specifying the parametes like storage capacity, scalability aspects, back & restore settings and failover configuration settings. So that RDS takes care of provisioning the instance and returns to based on the above configuration.
2. Aurora db = is a database that is built by amazon on top of mysql server database. and compared with mysql server the aurora db is 5 timers faster interms of data transfer and other aspects.
3. Dynamo db = it is an no-sql database system used for storing semi-structured data.
4. Elastic Cache = used for storing data in a distrbuted cache system
5. Redshift = database warehouse service for generating analytics and data reporting
---
4. Networking
Networking domain offers various networking services and security services in managing aws resources. It offers 3 services as part of it.

1. vpc (virtual private cloud) = virtual private cloud network is a virutal isolated network a user can create in aws, so that all the resources within that vpc can only can communicate.
2. Direct Connect = aws provides an leased internet connection dedicated to your aws resources as a replacment to aws internet connection

3. Route53 = is a domain naming service registry provided by aws for binding domain name with ip address.

---

5. Management Tools

There are 7 domains are in aws
1. compute
2. storage
3. database
4. network
5. management tools
6. security
7. messaging

1. compute
1.1 ec2
1.2 lamda
1.3 elastic beanstalk
1.4 elastic loadbalancer
1.4 asg
2. storage
2.1 s3
2.2 cloud front
2.3 elastic block storage
2.4 snowball
2.5 glacier
2.6 storage gateway
3. database
3.1 rds = relational database service, it is not an database. it manages the mysql server, oracle postgress sql database in provisioning, scalability, storage, back & recovery aspects. we can think of it as a pass for database
3.2 aurora = its a database built on top of mysql server and is 5X faster than mysql server
3.3 dynamo db = no-sql database provided by amazon used for storing semi-structured data
3.4 elastic cache = distributed caching services
3.5 RedShift = dataware house help in build data analytics and data reports

4. Network
All the networking related resources are offered as services, there are 3 services are there
1. vpc = virtual private cloud network, helps us in creating an isolated network for our aws resources.
2. direct connect = dedicated internet leased connection service established by aws to your resources instead of using a shared internet service
3. Route53 = is a dns server hosted by amazon for providing dns services to the aws resources
---
5. Management Tools
The services under this domain helps us in managing the aws infrastructure
1. Cloud Watch = This is an ec2 monitoring tool, that sends notifications to the aws account user, when the certain threashold limits are reached by the ec2 instance
2. Cloud Formation = Templatized Service, that helps us in creating cloned enviromnents, so that we quickly create production environments from the exting test or stage environments we have created. (Test To Production)
3. Cloud Trail = Logging service for your api requests and responses
4. CLI = alternate to cloud console
5. OpsWork = its an configuration management tool that works with puppet and chef in provisioning and configuring aws instances.
We create stack in opswork providing configuration related to ec2 instance and related chef| puppet scripts that should be used for configuring the instance post provisioning. The OpsWork takes care of applying those configuration scripts automatically after the instance has been provisioned.

6. Trusted Advisor = Billings and Recommendations

---

6. Security Domain
IAM = Identity and Authorization Management

athena = insurance software application (aws account) [project] [project manager]
-> deployed tested and should be deployed in production environment (aws cloud)
-> aws users
-> groups
-> policies

KMS = Key Management Service

7. Messaging
It provides all the messaging related services in build application
SMS Mailing Service =

5. Management Tools

It holds all the services related to managing the aws infrastructure
1. Cloud Watch = it monitors aws ec2 instances and generates notifications when certain threashold levels are reached by the instances
2. Cloud Formation = its an Templatization service, used for creating cloned environments of the aws resources.
3. Cloud Trail = its an logging api for tracking api request and responses
4. CLI = alternate to cloud console
5. OpsWork = Configuration Management Tool helps us in provisioning and configuring the aws resource using chef or puppet
6. Trusted Advisor = generates billing recommendations

6. Security

Services related to securing and protected aws infrastructure resources are available as part of this domain
1. IAM = identity and authorization management
2. KMS = Key Management Service

7. Messaging Domain

Messaging related services are provided as part of this domain

SES (Simple Email Service) = acts as an relay server is sending bulk emails to the users
SNS (Simple Notification Service) = Service that generates events when an operation has been taken place on any resource in aws infrastructure.
For e.g.. when we upload a big image file into s3 bucket the SNS service generates an event and publishes, so that lamda can look at the notification and may perform operation like compression.
SQS (Simple Queuing Service) = its an queuing service for storing the SNS generated notifications untill those are consumed.

AWS stands for Amazon Web Service cloud, it offers various iaas,paas and saas services ondemand to the end-users.
In order to use the aws cloud platform services we should have an aws account. aws account is the primary account holder|administrator who has complete access to the aws infrastructure. aws account holder into to use aws services should login into aws cloud console, the he can provision resources and can perform lifecycle operations on the resources. all the resources that are provisioned are entitled to the aws account owner and he only can administer them like
- stop/start/restart
- scale out/in
- scale up/down
aws generates the billing against the aws account we are using for provisioning resources.

when we are signing up for an aws account, we need email address and debit/credit card, so that aws authorizes your credit/debit card by deducting 1 rupee from your card and revert it back with 24 hours - 3 working days. and every month bill will be generated based on usage and the amount will be automatically deducted from your debug/credit card that you provided while registration.

When we create new aws account, they provide free-tier access for 1 year on 80 products are provided with threashold usage limits for each of the services separately.

aws cloud console = https://console.aws.amazon.com/console/home

---

AWS Architechture

aws has distributed its services into regions where a region is a country in which the aws services are being offered. So that customers can choose and avail services nearest to their location by choosing these regions.

advantages:-
- The network latency in accessing the resources will be very less when choose the nearest region for using the resources
- it takes less time to transfer the data between the customers computes and aws resources if we choose nearby region.

aws is offering its services in different countries/geographical locations and each country may have one or more regions.
for e.g.. us has 4 regions
us-east-1
us-east-2
us-west-1
us-west-2

Aws root account | aws account administrator

To work with aws infrastructure, we need aws root account, all the services/resources we created within aws cloud will be entitled to the aws root account. root account is the administrator has fullaccess to all the resources/services of the aws infrastructure.
He can perform all the lifecycle activities on the resources like
- provision
- deprovision
- start/stop/restart
- scale up/down
- scale out/in

A monthly bill will be generated against the root account based on the usage of the resources.

AWS Architecture

Regions
---

aws cloud infrastructure has been spread across various geographic locations which are called "Regions". A user can choose to avail the services within any of these regions, so that all the services will be provisioned and allotted within the region selected.
A Region is a country or a geographic location where aws operate in providing the services, so that users has to select the region based on their usage traffic and the customers who are using your business services, for better performance and faster data transfers.

There are total 13 regions are currently available with aws few of them are
us-east-1
us-east-2
us-west-1
us-west-2
ap-south-1 ... etc
upon login you have to choose appropriate region from top right of your aws console page.

Availability Zones

"Availability Zones" are the various different data centers aws operates within the Regions. Each region will have atleast 2 availability zones, and can have even more as well. The availability zones are provided by aws for High Availability of aws resource for the customers.

For e.g.. when we are creating 2 ec2 instances for our application, it is recommended to distribute these 2 instances across 2 availability zones of that region. So if one availability zone goes down your services are still available from the 2nd availability zone, in such a way we can achieve high availability.

The resources under the same region but spread across multiple availability zones can communicate directly with each other, aws takes of inter-linking the data centers of the same region with high speed internet connectivity so that we face little/no latency in communication between the resources of different availability zones of a region.

by default the resources across the regions cannot communicate with each other
one the aws account owner logins he choose only the region, but availability zones will be picked up while creating the resources within that region

---

Network Domain
---
1. vpc (virtual private cloud)

When we create resources under an region within multiple availability zones, by default all of these resource that belongs to one aws account user can communite with each other. So how to control the accessibility aspects of the resources within the region. To help us with this aws has provided vpc

vpc stands for virtual private cloud is used for create isolated network with an aws account. all the resource that belongs to a vpc can communicate with each other. by default the resources of one vpc cannot communicate with other vpc even thought those belongs to
- same aws user account
- same region
- within same/multiple available zones.

aws architecture
---

aws cloud infrastructure has been distributed across various different geographies which are caled "Regions".

Region

Aws account user should choose a region upon login to the aws cloud console to provision the resources under the region. choose the region that is nearest point of usage, so that they will no network latency and delay in network transmission.
There are currently 13 regions are available in aws cloud infrastructure

Availability Zones

Availability zones are the data centers of the aws within the regions. each region has atleast 2 availability zones and could more as well.
Aws brought atleast 2 availability zones per Region to support High Availability of services. When have to distribute our resources across the Availability Zone of a Region,so that if one Availability Zone was down due to some reason, always we have the other Availability zones within the same region in which our services are running, so that complete loss of service will not happen

Edge Locations

Edge locations are the co-located locations around the Aws Region which are a kind of mini or small datacenters. There are certain shared services in aws like
1. cloud front
2. Route53
These services should be accessible globally, for example a Route53 dns server should be used for resolving a domain name to an ip address. If the Route53 services is available with a distant location from the customer, it takes more latency time in discovering the ip address for a given domain name.
So, aws has hosted such global services across the edge locations so that the performance of these resources will be high with no latency

Scope of Services in aws

Aws services can be broadly classified into 3 scopes.

1. Global = A change in these services will affect the entire aws infrastructure at account level. These services are served across the AWS Regions.
1.1 AWS IAM
1.2 Cloud Front
1.3 Amazon Route53

If we add a static resources in our project, it would be cached globally across all the aws regions and their edge locations globally
If we modify an IAM Policy for a AWS User or a Group it would be reflected across all the AWS Regions.

2. Regional (Regions) = These services are accessible within the Region in which those are created, those are confined to that Region only.
2.1 Amazon DynamoDB

2.1 Amazon S3
2.2 Elastic Load balancer
2.4 Virtual Private Cloud

For eg.. we have provisioned an s3 storage service within an Region us-east-1. The service is accessible only within that Region only

3. Availability Zone = These are the services who has the scope of accessibility or presence within the availability zone only.
1. Amazon Elastic block storage
2. Amazon ec2
3. Subnets

When we create an ec2 instance it is confined to that availability zone only, and the scope of the resource is within that availablity zone. Elastic block storage is an harddisk for that ec2 instance and is scoped to the same availability zone of ec2

Networking domain

---

virtual private cloud (vpc) = vpc is an isolated network of resources of an aws account. the resources of one vpc are isolated from other vpc networks.
1. vpc are created at account level and per region of an account
2. per a vpc of a region, all availability zones of that region are included as part of it
3. per account aws allows max 5 vpc (its a soft limit)
4. by default a resource of one vpc cannot communiate with other resources of another vpc. we can use vpc peering for establishing communication across the resources of the vpc.

What the purpose of vpc?
vpc's are used in various different ways.
For example
per business domain in a company to host all their services in a isolated manner people use vpc. for eg.. Company may have different business domains like financial domain, and retail marketing domain. The services/resources hosted in financial domain should not become accessible to other services of retail marketing domain. How keep them isolated from each other.
Place them in 2 separate vpcs.
1. financial vpc
2. sales and retail marketing vpc

We can use vpc for creating multiple parallel isolated environments.
For eg. test vpc and production vpc so that all the test environment related services/resources will be hosted in testvpc and production related services/resources will be kept isolated from test by keeping them in productionvpc.

edge locations
---
edge locations are the co-located mini datacenters around the "Aws Region" to serve global aws services to the customers with no latency and high speed data transfer
- cloud front
- aws route53

aws scope of services
we can categorize the aws services into 3 scopes
1. global
These services are target to the aws infrastructure level, and a change in these services affects all the regions of the aws account level. For eg.. if we change an user policy in IAM service, the policy will be effect for all the regions of the aws account

similar if we remove an cached static file from cloud front cdn network, the it would be reflected across all the edge locations whereever the static content is served.

1. AWS IAM
2. AWS Route53
3. AWS Cloud Front

2. Regional
These services are scoped at Regional level, for eg.. if we create an vpc in us-east-1 the presence of the vpc is only for that region, and we cannot see the vpc in other region like us-west-1.

1. S3
2. Elastic Load Balancer
3. vpc
4. Dynamo db

3. Available Zone
If we create a aws resource of a service in an availability zone, the presence of the resource is to the availability zone only, the same resource will not be visible to another availability zone.
For eg.. we created an ec2 instance in ap-south-1 under availablity zone az_1 if moved to az_2 of the same region the ec2 instance will not appear to manage.

1. Elasitic block storage
2. ec2
3. elastic cache
4. subnets
---
vpc
---
vpc stands for virtual private cloud, used for creating isolated network of resources. vpc are used enforcing traffic restrictions and controlling the accessibility/visibility of the resources within a region.

- by default for all the regions of the aws account, one default vpc will be created by the aws.
- vpc are scoped to the region level
- vpc are created for an aws account at region scope
- at max 5 vpc are allowed per aws account, and its a soft limit
- all the resources within the vpc can communicate with each other by default
- the resources across the vpc cannot communicate with each other by default, we need to

establish vpc peering
- vpcs spans across the availability zone of a region

what is the pupose of vpc?
we can create isolated networks for business domain level or parallel environments using vpc.
---

subnet
---
vpc is an large isolated network that is scoped to the Region level, so we can break/divide the vpc into smaller networks using subnet.

we can create sub-networks within a vpc using subnet. by distributing the resources into multiple subnets of the vpc we can enfore traffic restrictions and accessibility aspects of the resources based on subnet

1. subnets are created within an availablity zone
2. per vpc we can create upto 200 max subnets
3. it is recommended to create atleast 2 subnets per vpc in different availability zones

Why do we need to create a subnet, what is the purpose of it?
There are 2 reason for creating a subnetwork
1. to enforce traffic various different traffic restrictions on different resources we use subnetwork
2. using subnets we can map aws resources to the availablity zones

while creating an vpc or a subnet we need specify the CIDR Notation defining the range of ip address to be allocted to the resources of the vpc or subnet