**Terraform**

**Terraform --> popular infrastructure as a code**

**Ansible --> configutation management, configutation as a code**

**manual**

**shell script**

**Ansible --> preferred**

**version control**

**if you keep code in Git, we can maintain history and you can get back previous versions**

**consisten infrastructure**

**same code will work with all environments, so infrastructure is consistent across all environments.**

**inventory management**

**cost optimisation**

**create when required, stop or delete when not required**

**automatic dependency management**

**we can create resources, terraform will take care of dependency management**

**modules**

**you can create terraform code as modules, other projects can use it without writing from the scratch**

**declartive way of creative infrastructure**

**what is declartive? whatever you write you will get provided if you write correct syntax. easy syntax, no need to follow sequence**

**used for hybrid cloud and many other providers**

**AWS, Azure, GCP, etc.**

**terraform --> it will check all the paths you have given in environment variables, if it is available then your command will work**

**1. install terraform**

**2. install aws cli**

**resource "what-resource" "name-your-resource-your-wish" {**

**}**

**terraform requires AWS provider configuration to create AWS infra**

**.sh, .yaml, .tf**

**1. create IAM admin user**

**2. create access key and secret key**

**3. aws configure**

**sg --> optional, then default security group will be attached**

**where to run terraform command, wherever .tf file exists you have to run terraform command in that directory**

**terraform init --> intialize the provider,**

**.terraform folder is created and provider downloaded**

**terraform plan --> just a plan, will not create actually**

**terraform apply --> now it will create resource**

**variables:-**

**variable "name-of-your-variable"{**

**type = data-type**

**default = "default-value" # we can always override**

**}**

**Expression:-**

**expression ? "this will run if true" : "this will run if false"**

**conditions**

**condition ? "true" : "false"**

**loops:-**

**count based loops --> useful mostly to iterate lists**

**foreach --> useful to iterate maps**

**dynamic loop**

**function:-**

**it will do some work**

**but we need to give some input**

**it will give some output**

**locals:-**

**locals is just like variables, but it have some extra capabilities. you can keep functions and expressions inside locals and use them**

**data sources:-**

**it is useful to query the data dynamically from providers**

**state and remote state:-**

**whatever we write it should create --> declarative**

**declarative = desired state**

**current state = terraform.tfstate**

**desired state == current state**

**terraform will check current state, if current state is equal to desired state then it will not take any action...**

**pushing to github**

**another devops-2 cloned the repo and tried to run again**

**devops-1 and devops-2**

**both are running terraform apply**

**duplicates may come, some resources got error as already exist..**

**we need to a central state file to check the infra exists or not...**

**1. collabaration**

**terraform will have no idea about the infra created, if multiple persons are working on same infra, it will try to create duplicates and errors may also come**

**2. security**

**tfstate is a crucial, if you keep it in local it may be deleted or updated by mistake.**

**how you create multiple env with terraform**

**multiple env:-**

**tfvars**

**workspaces**

**diff repos**

**mongodb-dev, redis-dev**

**mongodb-dev.daws76s.online**

**same code but different configuration**

**diff buckets to maintain state**

**web-dev or web-prod or web-qa**

**if key starts with web**

**workspaces**

**------------------**

**terraform.workspace --> dev**

**terraform.workspace --> prod**

**create different repo for different env**

**1. pros --> same code**

**cons --> same code for multiple env, you need to be very careful because whatever changes you do that will apply to prod**

**2. pros --> same code**

**cons --> same code for multiple env, you need to be very careful because whatever changes you do that will apply to prod**

**terraform is maintaining same bucket that may cause errors and difficult to maintain, difficult to maintain variables**

**3. pros --> since everything diff, you no need to worry**

**cons --> code duplication, you need to maintain 2 repos**

**provisioners:-**

**2 types of provisioners**

**1. local-exec --> local to where you are terraform command**

**2. remote-exec --> this will run inside the server**

**provisioners are useful to integrate terraform with configuration management tools like ansible**

**server creation --> infra creation**

**server configuration --> configuration management**

**creation time --> this local exec will run when server is create**

**destroy time --> at the time of destroy**

**remote-exec:-**

**1. first you should connect to server**

**2. then you can run anything inside the server**

**Modules development:- (29 note)**

**DRY --> Don't repeat yourself**

**code reuse, no need to write infra for every project from the scratch**

**ec2.tf**

**variables.tf**

**module "name" {**

**}**

**1. customised modules for our organisation**

**2. open source modules**

**1. module developer --> who develops modules**

**2. module user --> who consumes modules**

**VPC --> Virtual private cloud:-**

**physical space**

**AC**

**power backup**

**network connections**

**security gaurds**

**network field engineer**

**linux admin team**

**an isolated network with in AWS.**

**Village --> VPC:-**

**streets --> admin, proper organisation and maintaince**

**subnets -->**

**HR department**

**Databases**

**Fianance**

**Public affairs**

**minister offices --> no direct access**

**PM office --> no direct access**

**Arch --> entry point = internet gateway**

**roads = routes**

**public/private**

**route tables --> routes**

**public route table --> routes --> attatch with public subnets**

**private route table --> routes --> attatch with private subnets**

**database route table --> routes --> attatch with database subnets**

**CIDR --> classless inter domain routing**

**------**

**ipv4 --> 192.168.1.1 --> 32 bits = 4 octates**

**0 1 2 3 4 5 6 7 8 9 ---> 10 numbers**

**0,1 --> 2 numbers**

**- - - - - - - -**

**1st --> 2^0 --> 1**

**2nd --> 2^1 --> 2**

**3rd --> 2^2 --> 4**

**--> 8**

**--> 16**

**--> 32**

**--> 64**

**8th --> 2^7 --> 128**

**1+2+4+8+16+32+64+128 = 255**

**11000000.10101000.00000001.00000001**

**2^3 = 8**

**2^5 = 32**

**2^7 = 128**

**decimal --> binary**

**15.197.142.173 --> facebook IP**

**home network --> facebook network --> particular server**

**IP = Network + Server IP**

**10.0.0.0/16 --> first 16 bits are for network, next 16 bits are for servers**

**10.0.0.1**

**10.0.0.2**

**10.0.0.3**

**10.0.0.4**

**\_ \_ = 2^2**

**0 0**

**0 1**

**1 0**

**1 1**

**\_ \_ \_ = 2^3**

**0 0 0**

**0 0 1**

**0 1 0**

**0 1 1**

**1 0 0**

**1 0 1**

**1 1 0**

**1 1 1**

**32 bits --> 2^32 == 403 crores**

**IP address**

**NAT --> network address translation**

**subnetting**

**-------------**

**549906 --> village pin code**

**street --> street 1**

**home number --> 1-12 --> street 1 and house 12**

**1-12**

**549906**

**IP = network+host**

**10.0.0.0/16 --> 2^16 IP are possible**

**public subnet**

**private subnet**

**database subnet**

**10.0.1.0/24 --> 2^8 = 256 IP are possible**

**10.0.1.0**

**10.0.1.1**

**10.0.1.2**

**.**

**.**

**.**

**.**

**10.0.1.255**

**10.0.2.0/24 --> 2^8 = 256 IP are possible**

**10.0.2.0**

**10.0.2.1**

**10.0.2.2**

**.**

**.**

**.**

**.**

**10.0.2.255**

**10.0.1.0/20 --> 20 bits network, 2^12 = 4096 servers you can configure**

**10.0.1.0/32 --> 10.0.1.0**

**16-28**

**10.0.0.0/28 --> 28 bits are for network, 2^4 = 16 servers**

**10.0.0.0/16 --> 16 bits are for network, 16 bits are for hosts --> 65k**

**VPC:-**

**internet gateway --> VPC**

**public subnet**

**private subnet**

**public route table**

**private route table**

**Regions and AZS**

**ap-south-1**

**ap-south-1a --> west mumbai --> psunami**

**ap-south-1b --> east mumbai --> far from sea**

**ap-south-1c --> south**

**NAT gateway and VPC peering**

**30 Note**

**HA**

**atleast 2 AZ resources be there**

**az-1 public subnet az-2 public subnet**

**az-1 10.0.1.0/24**

**az-2 10.0.2.0/24**

**az-1 private subnet az-2 private subnet**

**az-1 10.0.11.0/24**

**az-2 10.0.12.0/24**

**az-1 database subnet az-2 database subnet**

**az-1 10.0.21.0/24**

**az-2 10.0.22.0/24**

**NAT gateway :-**

**we need to update packages, we need to download something. We should connect internet**

**outgoing**

**incoming traffic is disabled...**

**NAT gateway should be in public subnet because it has internet connectivity, then we add i in the route tables..**

**it is chargeable**

**VPC Peering:-**

**By default we cant connect 2 vpc. it is possible through VPC if we want**

**condition --> Both VPC should have different CIDR.**

**Requestor VPC --> I want to connect with you please accept my request.**

**Acceptor VPC**

**VPCs in same region same account**

**VPC in another region same account**

**VPC is same region another account**

**VPC in different region another account**

**Requestor VPC --> roboshop**

**Acceptor VPC --> Default**

**1. create VPC**

**2. create igw and attach**

**3. create all subnets**

**4. create route tables**

**5. create routes**

**6. associate with subnets**

**7. create eip**

**8. create nat**

**9. add nat gateway route in private and database subnets**

**10. peering connection**

**11. routes**

**common tags**

**Project = roboshop**

**Terraform = true**

**Environment = dev**

**Name = something**

**vpc tags, subnet tags, nat gateway tags, Name=another thing**

**merge common tags and resource tags**

**roboshop-dev -->**

**roboshop-prod**

**I will fetch az from AWS using data source and I will get first 2 AZ.**

**slice(list, 0, 2) --> 0,1**

**roboshop-dev-us-east-1a**

**31 Note**

**user can decide peering required or not**

**if required**

**1. they have to give peering VPC ID**

**2. if they are not giving peering VPC ID, we will consider default VPC**

**Requestor - roboshop**

**Acceptor - user provided VPC or default**

**roboshop-dev-mongodb**

**1. accept connections only from catalogu and user**

**create a vpn in default VPC**

**connect to vpn in your laptop, then only you can access instances...**

**32 Note**

**sg module create and create all security groups**

**roboshop-dev-mongodb**

**mongodb:**

**- catalogue**

**- user**

**source: catalogue IP address**

**source sg id: sg-018af35c4f2bb2497 --> catalogue**

**port: 27017**

**mongodb sg, catalogue sg**

**roboshop-dev-mongodb**

**sbnr-1234, sbnt-3456 --> for AWS it is list, but for us it is string**

**[]**

**1. create one ec2 in public subnet, login to that ec2 first and from there connect to mongodb**

**2. vpn installation, connect to private servers through vpn**

**33 Note**

**private**

**--------**

**1. first connect to public instance**

**2. from public you can connect private**

**vpn**

**----**

**inbound**

**mongodb --> vpn security group**

**mongodb**

**--------**

**1. vpn**

**2. catalogue**

**3. user**

**34 Note**

**1. how to develop module**

**2. how to use module**

**3. SSM paramter store to store the configuration.**

**one project exports configuration**

**another project can refer configuration**

**4. SG -->**

**VPC and peering**

**All SG**

**VPN**

**EC2 and records**

**create one ansible server, this server will provision playbooks to all the EC2**

**Load Balancer**

**Target groups**

**rules**

**launch template**

**auto scaling**

**auto scaling policy**

**one manager, team leaders, team members**

**clitens will send requirements to manager**

**1. UI requirements**

**2. backend requirements**

**3. database requirements**

**UI team leader, backend leader, database leader**

**team leader --> team members**

**manager --> load balancer**

**running 2 nginx servers**

**instances will get requests from load balancer**

**ingress rule should have a security which is attached to load balancer.**

**client --> manager**

**target groups --> teams**

**ui, backend, database**

**health check**

**1. instance up**

**2. is instance busy serving other requests**

**35 Note**

**vpc**

**push**

**----------**

**ansible --> nodes**

**manual login, they disturb configuration**

**ansible-pull --> every 30min download the configuration from git directly and run it, anyhow we implemented idempotency so no problem**

**every platform will have its own solution to maintain configuration and secrets**

**SSM paramter store --> services will refer this..**

**server provisioning is through ansible, so it is ansible responsibility to get the configuration from paramter store**

**ansible --> AWS paramter store**

**ansible use boto3 and botocore python modules to connect**

**app and web tier:-**

**--------------------**

**1. if traffic increases, autoscaling will create instance**

**1. when new instance created, use user\_data or provisioner with ansible to configure the server --> 5min**

**2. when you are deploying, website is running**

**a. first we create one new instance**

**b. provision the instance using ansible**

**c. stop the instance**

**d. take AMI --> 10min**

**e. update autoscaling using new AMI**

**slowly old instances will be replaced by new instances**

**in future if traffic increases, autoscaling uses AMI to add the instances**

**36 Note**

**client --> Business analyst --> team manager --> team leads --> team members**

**team manager --> only listen to business analyst**

**team manager**

**ui team lead --> if ui work, he will redirect to UI team lead**

**backend team lead --> if backend work, he will redirect to backend team lead**

**UI team --> group**

**--------**

**3 members**

**HR --> HR should continously check if the resources**

**overload --> have to take another resource**

**less work --> have to terminate**

**if they resign --> they have to replace**

**Team Manager --> APP ALB**

**Team Manager Listens --> listens to only business analyst, port no 80. no encryption required.**

**Rules**

**if UI --> UI team**

**if Backend --> Backend team**

**HR --> autoscaling**

**Team/Groups --> target groups**

**employees --> instances**

**hiring template --> launch template**

**context path**

**daws76s.online/roboshop**

**daws76s.online/amazon**

**host path**

**roboshop.daws76s.online**

**amazon.daws76s.online**

**\*.app-dev.daws76s.online**

**team create -->**

**create catalogue target group**

**HR --> Autoscaling**

**create one instance**

**provision with ansible/shell**

**stop the instance**

**take AMI**

**delete the instance**

**now create launch template with AMI**

**roboshop-dev-catalogue-YYYY-MM-DD-hh-mm**

**catalogue.app-dev.daws76s.online --> redirect to catalogue target group**

**37 Note**

**catalogue deploy**

**new version of catalogue should be released, new launch template version also should be released....**

**autoscaling is refreshed using new launch template**

**autoscaling provision ec2 instances into a target group directly**

**RollingUpdate:-**

**4 instances are there now, 4 new instances should be created and replace old**

**1 new instance --> once it is up, 1 old instance will be terminated**

**2 new instance --> once it is up 2nd old instance will be terminated**

**target group**

**instance launch**

**provision**

**stop**

**ami**

**delete instance**

**launch template**

**autoscaling --> target group**

**LB --> listener and rules**

**\*.app-dev.daws76s.online --> APP ALB**

**catalogue.app-dev.daws76s.online --> catalogue target group**

**http://catalogue.app-dev.daws76s.online/categories**

**\*.app-dev.daws76s.online --> internal-roboshop-dev-app-alb-1986419658.us-east-1.elb.amazonaws.com.**

**instance-1 --> handling some requests**

**instance-1 got termination request, dereigstration will start**

**complete pending requests and loadbalancer should not give any new requests**

**WEB**

**--------------**

**1. web alb, create security group first. listener 443. rule**

**web-dev.daws76s.online**

**before creating listener you need to create certificate. https**

**\*.daws76s.online --> create https certificate**

**2. 443 give access from internet**

**godaddy**

**---------**

**login to godaddy account, create records there...**

**How HTTPS works?**

**ACM**

**CDN**

**38 Note**

**app\_alb\_web**

**source: web\_sg\_id**

**ingress rule: 80**

**web-dev**

**https://web-dev.daws76s.online**

**1. web ALB roboshop-dev-web-alb-1134442274.us-east-1.elb.amazonaws.com.**

**2. 443 listener will invoke**

**3. rules will evaluate and traffic will be redirected to web tg**

**4. traffic reaches web and give us the website**

**https://web-dev.daws76s.online/api/catalogue/categories**

**1. web ALB roboshop-dev-web-alb-1134442274.us-east-1.elb.amazonaws.com.**

**2. 443 listener will invoke**

**3. rules will evaluate and traffic will be redirected to web tg**

**4. traffic reaches web and give us the website**

**5. api/catalogue/\* --> catalogue.app-dev.daws76s.online**

**http://catalogue.app-dev.daws76s.online**

**1. APP ALB**

**2. 80 listener will invoke**

**3. rules will evaluate and traffic will be redirected to catalogue tg**

**4. it will connect to and fetch products**

**Cloudfront**

**---------------------------**

**ISP also cache servers, they download movies in cache servers**

**cdn --> get it from web alb(origin)**

**39 Note**

**Cloudfront:-**

**useful to cache static content**

**dynamic content should not be cachced**

**https://web-dev.daws76s.online/media/graph.png**

**https://web-dev.daws76s.online/media/instana\_icon\_square.png**

**https://web-dev.daws76s.online/images/STAN-1.jpg**

**https://web-dev.daws76s.online/media/instana\_icon\_square.png**

**/images/\* --> then apply cache policy**

**/media/\* --> then apply cache policy**

**default --> get the content from web-dev.daws76s.online**

**deployment**

**------------**

**changed some content**

**remove cache from all edge locations**

**cloudfront will get latest again and cache it**

**https://web-cdn.daws76s.online**

**1. get the cdn url from route53**

**2. behaviours will be evaluated**

**https://web-cdn.daws76s.online/api/catalogue/categories --> web-alb**

**web-dev.daws76s.online/api/catalogue/categories**

**listeners and rules in web alb**

**443, web-dev --> web**

**/api/catalogue/\* --> catalogue.app-dev.daws76s.online --> APP ALB**

**80 catalogue. --> catalogue component**

**/categories --> connect to mongodb and fetch the categories**

**SG**

**----**

**VPC based service, where resource is created inside VPC**

**non VPC service, resources are created outside VPC**

**1. CDN --> WEB ALB**

**web alb should accept traffic on 443 from internet.**

**2. Web ALB --> WEB**

**WEB should accept traffic on 80 from web-alb**

**3. web-alb --> app-alb**

**app-alb should accept traffic on 80 from web and vpn**

**4. app-alb --> catalogue**

**catalogue should accept traffic on 8080 from app-alb**

**5. app-alb --> cart**

**cart should accept traffic on 8080 from app-alb**

**cart --> catalogue through app-alb**

**app-alb should accept traffic on 80 from cart**

**all components are accepting traffic on port no 80 from app-alb**

**6. mongodb should accept traffic on 27017 from catalogue**

**vpc, sg, vpn, databases, web-alb, app-alb -->**

**project infra --> we need project infra to create applications**

**through SSM parameter store**

**catalogue**

**--------------**

**CICD**

**CI**

**----**

**clone**

**build**

**scan the code**

**static scanning --> sonarqube**

**sast --> fortify, coding is based on security standards**

**dast --> web inspect, app will be tested with all vulnerabilities**

**open source scan --> nexus IQ, blackduck scan thrird party libraries**

**image scanning --> ECR scanning, scanning docker images**

**create artifact, generally zip file**

**store the zip file --> nexus, jfrog, s3**

**unit testing --> junit**

**CD**

**----**

**application infra --> developers or devops engineers**

**-----------------**

**create instance**

**deploy the latest version**

**stop the instance**

**take ami**

**refresh autoscaling**

**do the functional testing --> selenium**

**shift-left**

**------------**

**GIT**

**--------------**

**main/master --> depicts/points the code running currently in production**

**create a copy of file --> do the changes --> test those changes --> then do it main file**

**create another branch from main/master**

**--------------------------------------**

**do the changes here**

**run CICD --> deploy in DEV**

**test the application**

**merge into master --> deploy in PROD**

**we write functions --> functionality**

**input and output**

**code coverage --> how much unit testing covered**