URL To The Doc →

https://tinyurl.com/terraawsworkshop

Objective → To understand how to build real world aws application use cases/troubleshoot/configuration with terraform.

Session Logistics →

1. Understanding the case study and service involved + terraform projects + configuration/scale/security/…
2. https://github.com/orgs/awsterraworkshop/repositories
3. Lunch Break timings – 15.30 - 16.30 IST
4. Breather breaks – about 90-100 minutes

5. https://docs.google.com/presentation/d/144lPlndwvk8Kq1cCjJY-9HSW75TYp1Kxm2dT3s_F5kM/edit?slide=id.g35f391192_00#slide=id.g35f391192_00

Pre-requisites →

1. AWS Basics
2. Terraform - pro level
3. Git, kubernetes, api ..etc..
4.

Introductions →

1. Your quick intro
2. Your familiarity with aws, terraform
3. Questions

Labs →

1. Go to lms.springpeople.com
2. Official email id and login with password or try forgot password
3. Click on the left hand side bar and choose cloud labs
    a. One Windows vm which will your development vm
    b. AWS Account - administrator (region)
4. Connect with your windows-vm and login to aws and go to iam.
5. Go to uses and generate aws accesskey and secretaccesskey for the user.
6. aws configure
7. Region name will be mentioned on the lab. Confirm with aws s3 ls

8. Create a directory on the desktop - aws-workshop and create a sub-dir case-1 and open it in the vs-code. Create a file main.tf and confirm that terraform extension is added.

Case - 1 →

Resources to provision →

- Vpc, public subnets, ig, rt

Handson - 1 → just initialise the provider - aws with the region name.

terraform validate

503 clear

504 terraform init

Time till – 14.06

HandsON2 → Create vpc, ig, public subnets.

HandsOn3 → Added the launch template, ami and sgs.

HandsOn4 → create lb, tgt group and associate tg with asg.

Handson 5 → ensure to update keypair name in launch template which you can download a ppk file.

Scenario - 1 → connect with ec2 instance and check. – connect over ssh using putty and check localhost.

HandsON → Deploy the autoscaling policies.

Time till – 17.31

Scenario – 2 – try out manual/autoscaling.

sudo dnf -y install stress-ng
sudo stress-ng --cpu 5 --timeout 180

HandsOn – try out manual and autoscaling both.

Scenario – 3 – security → we have used chained security groups with minimum post openings. One can manipulate nacl.

Scenario – 4 → application update →

Update the shell script and change the default version of launch template and roll out using asg.

Scenario 5 → one can create custom alarms. Try to add an alarm using cloudwatch when number of instance in asg goes above 4.

Scenario 6 → adding a route53 record into a public zone for lb access.

DAY - 2 →

Connect with the lab →

1. https://lms.springpeople.com/cloudlabs/learner/
2. Connect with the vm and aws account

Data Persistency →

- Ec2 instances takes down root volumes by design when they are terminated
- One can ebs, s3, efs - persist data depending upon the kind of data that you have

Ex – provision, connect with a vm go to /data and create some files. Reboot the vm and check if data persisted. Now terminate the ec2 you should see autoscaling creating a new instance with a new volume.

Ex – Try out the persistence with efs across ec2 instances.

Case - 2 →

Ex – Create a new dir case-2 in aws-workshop folder and open it in vscode and initialise aws as a provider.

Time till – 14.42

Ssm →

- Is primary a capability to connect, manage updates/patches/changemanagemnt/compliance/secretpassing …
- Agent based utility
- VM (ssm agent + ec2 profile with ssm permissions) → SSM api (AWS)
- Most of the prebuilt amis available in aws ec2 come preinstalled with ssm agent
- https://docs.aws.amazon.com/systems-manager/latest/userguide/manually-install-ssm-agent-linux.html
- At bare minimum an ec2 instance must have policy - AmazonSSMManagedInstanceCore

HandsOn – Provision ec2 and bring it under ssm. Look at fleet manager controls.

Connectivity →

- Ssm lets you connect with ec2 instances for without opening ssh at all
- Using session manager or ssm connect one can get into it.

Scenario - 1 – remote port 22 from your ec2 sg for all and using ssn connect with the vm.

Change management → One can plan change management using tradition change management utilities.

Scenario - 2 – execute a run command on the vms using any option you would like to try.

Compliance →

- Lets you declare if instances are meeting certain update standards.

Scenario 3 – Execute a aws-basepatch on ec2 instance and see if you get compliance report.
Time till – 17.29

Case - 3 →

Resources →

- A vpc
- Ig
- Public subnets
- Private subnets
- Security groups
- Public routes

HandsON – 1 – For case3 create vpc, public/private subnets, sg.

DAY - 3 →

Ex – Connect with the lab aws and vm and check if all is well.

HandsON – provision nat-gw and create all sgs.

HandsOn – case3

- Check if scaling really kicks in
- Check if private instances have internet
- See if you can provision nat gw in each subnet
- Make backend scalable and expose with a nlb

https://github.com/awsterraworkshop/case-3

Case - 9 →

VPC peering – is bidirectional, peer to peer network.

Transit Gateway →

- Hub-spok – acts as a transit providers to any number of networks (vpc, peering connections, dx, vpn connection..)
- Operates at a regional level
- For across the region connections - use transit gateway peering

https://aws.amazon.com/transit-gateway/faqs/

https://medium.com/awesome-cloud/aws-difference-between-vpc-peering-and-transit-gateway-comparison-aws-vpc-peering-vs-aws-transit-gateway-3640a464be2d

Time till – 15.10

Case - 9 →

https://github.com/awsterraworkshop/case-9

Resources to build →

- Vpc-1
  - Pub-subs
  - private-subs
- Vpc-2
  - Pub-subs
  - Private-subs

- TG
- Vms

HandsON – Create vpc1 vpc2 with public pvt subnets, rt, ssm permissions.

HandsON – Provision tg and pub/pvt vms respectively.

Trying pinging vpc1 pub vm from vpc2 pub vm over private ip and vice-versa you will see thats not working.

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/ec2_transit_gateway_route_table

Time till – 17.48

Ex – Try out the tg, attachments, tg route table, vpc route tables and check the communication between vpc1 and vpc2 pub vms.

Ex – Add a vpc3.tf which creates largely the same structure using 10.30.0.0/16. Now make sure that each vpc can communicate privately with other two.

DAY - 4 →

Ex – Provision the labs and connect.

Ex – Try out inter region tg peering.

https://docs.aws.amazon.com/vpc/latest/tgw/tgw-peering.html

Ex – Create a tg rt for vpc1 and ensure vpc1 can only communicate with ohio.

*create a tg rt, propagation, attachment, associate with tg rt.

Case - 7 →

aws secretsmanager put-secret-value --region us-east-1 --secret-id arn:aws:secretsmanager:us-east-1:619512840514:secret:topsecret1-cB5dB0 --secret-string "myrealtopsecret1valuehere"

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/secretsmanager_secret_version

Ex – Create a kms key, secret manager and try out updation, retrieval for it.

Ex – tryout the versioning with aws secretmanager.

```
 599  aws secretsmanager
get-secret-value --region us-east-1
--secret-id
arn:aws:secretsmanager:us-east-1:61
9512840514:secret:topsecret1-cB5dB
0
  600  aws secretsmanager
put-secret-value --region us-east-1
--secret-id
arn:aws:secretsmanager:us-east-1:61
9512840514:secret:topsecret1-cB5dB
```

0 --secret-string "myrealtopsecret1valuehere--version3"
  601 aws secretsmanager get-secret-value --region us-east-1 --secret-id arn:aws:secretsmanager:us-east-1:619512840514:secret:topsecret1-cB5dB0

Ex – Tryout aws secrets resource policies to Allow/Deny permissions.

Ex – See if you can enable replication for secrets to another region - ohio. Change the value in one region and see if that works.

Case - 4 →

*https://docs.aws.amazon.com/whitepapers/latest/aws-vpc-connectivity-options/aws-direct-connect-site-to-site-vpn.html

Ex – Create a vpc for case4 and cg vm in ohio which we will use a customer location to create vpc connection with.

Ex – bring up vpn and see the configuration for your choice of vpn site to site server.

https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html

Ex – try out enabling logging vpc flow log.

https://docs.aws.amazon.com/vpn/latest/s2svpn/monitoring-cloudwatch-vpn.html

Time till – 18.40

DAY - 5 →

Ex – Please connect with your labs.

Case - 10 →

Serverless → have no infra layer to be managed by us, no provisioning, scale, logging …etc..

*https://aws.amazon.com/serverless/

Lambda →

- A serverless - event driven computing platform built for backends.
- You create functions which have a piece of code, functions execute when triggers takes place.
- A function can max run for 15 minutes
- Lambda must not be used to deploy frontend, batch process, ml, big data …etc..
- Lambda functions run in a black box
- Lambda functions are by design stateless and if you want data to be persistent you must use your own stateful resources (s3, databases …etc..)
- Run time is provided
- A lambda function can max have memory between 128mb to 10 GB with a block size of 128mb

- Cpu is allocated in proportion to the memory
- One can operate at zero cost
- Idle cost of operation in lambda is zero - lambda charges only on the duration of function run time
- Lambda scales horizontally
- https://aws.amazon.com/lambda/faqs/
- Permissions →
    - Execution role – iam role – to which services your lambda can talk
    - Invocation permissions – resource policies – who can invoke lambda
- Handler determines who is your lambda main function - event(payload) and context(settings of lambda to override)
- Lambda functions writes logs in cloudwatch by default

Ex – Go to lambda, create a lambda, look at triggers, checkout configuration options, logs…etc.

Apigateway →

- You define everything for api frontend
- Allows you to decouple your microservice making them scalable easier to operate, independent …etc..
- Openapi3 compatible
- It is serverless
- Proxy, validation (auth, api, pathparam, query param, body, header …), transformations, usage plan, latency ….etc..
- https://aws.amazon.com/api-gateway/faqs/
- Apis are deployed in stages - stages are like environments

Ex – Deploy the example api in apigateway and see how it works and integrates with the lambda/backends.

Ex – Create lambda, try out handler, try out execution role permissions, logging.

https://github.com/awsterraworkshop/case-10

Ex – Create an api, stage, enable logging, apigateway permissions etc…

Ex – try out adding a post method and integration along with passing of header, body, path params, query params to the lambda function via apigateway.

Ex – Add another lambda function in the same for a different resource newdata with lambda integration as proxy.

Usage Plan → to limit the hits in a time period and burst number.

Ex – try out apikey and usage plans to see how we have enabled them only on POST method for data resource only.

*header - x-api-key -

*

https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/api_gateway_authorizer

DAY - 6 →

Ex – Please provision the lab and connect with it.

*ci/cd →

Case - 5 →

Github – git based scm which is community based

Github Actions – is a ci/cd (continuous integration and continuous deliver/deployment) capability offered by github.

Advantages →

- Reduces dependencies on middle infra (ci servers/artifact managers/cd systems..etc..)
- Is used in conjunction with artifact managers, container image repositories, cloud platforms.

Ex – Create a public repo in your personal github account and clone it locally and make a push into it.

Time till – 12.31

How it works →

- You create a .yml file in .github/workflows directory
- In the .yml you will define number of jobs, triggers, steps of the job, job

dependencies, github actions apps/actions ….

- Workflow is the name you give to whatever you write in .yml file
- https://docs.github.com/en/actions
- Name – whatever name you wish
- Trigger – is the event upon which the execution starts
    - Push, pull-requests, schedule …etc…
- Jobs – jobs are executed on runners
    - One can write multiple jobs
    - Jobs run in parallel by default
    - But if required one can declare dependency between them
- Runner – is the infrastructure on which job steps will be executed

Ex – https://github.com/awsterraworkshop/case-5/tree/main/.github/workflows. Write simple workflows and try to understand the structure. Add a new branch to your repo called dev and push into it but in the trigger only execute the workflow if pushed into main.

Github actions variables/secrets →

- Variables are not encrypted and are part of logs in plain text
- Secrets are supposed to kept safe and are never part of logs in plain text – must be used to credentials, certificates …etc..

Ex – create a varialbe and a secret and try to access in job.

https://github.com/awsterraworkshop/case-5/tree/main/.github/workflows

Actions/Apps →

- Reusable configuration units which can be used to clone the repo, install binaries, authentication with cloudplatform, set up build utilities …etc…
- actions/something
- https://docs.github.com/en/actions/how-tos/create-and-publish-actions/manage-custom-actions
- https://github.com/aws-actions/configure-aws-credentials
- https://github.com/google-github-actions/auth
- https://github.com/marketplace?type=actions
-

Ex – go through the github actions and write one actions which installs docker, kubectl awscli and authenticates with it, install terraform.

https://github.com/awsterraworkshop/case-5

Ex – Write a simple dockerfile and build it using github actions and see if you can publish it your dockerhub account.

Case - 8 →

EKS best practices –

- Expose eks endpoint only privately if not required but then to even run kubectl you will have to be in the same vpc
- One can enable public and private access both for the api server if required
- Eks – control plane(master) + data plane(nodes)
- Data plane can be put in public subnets/private subnets – but they will always need internet access to download images from dockerhub
- Best way to manage data pool is to put them in a private subnet with nat gateway
- https://www.eksworkshop.com/

Ex – https://github.com/awsterraworkshop/case-8 - Create a vpc and set it up to initialise eks cluster creation.

- One can created multiple nodegroups of different types and sizes

Ex – Install eksctl and initialise a cluster creation – eksctl create cluster --name neweks --version 1.32 --region us-east-1 --nodegroup-name neweks-ng1 --node-type t2.medium --nodes 1 --nodes-max 2 --managed --zones us-east-1a,us-east-1b

Adding an iam user to eks cluster with kubernetes authorisation → aws permissions + kubernetes auth

Config-map based authorisation →

kubectl get clusterroles
  527 clear
  528 cd
  529 kubectl get configmaps -n kube-system
  530 clear
  531 kubectl edit configmap -n kube-system aws-auth

API based →

*https://docs.aws.amazon.com/eks/latest/userguide/access-entries.html

```
terraform {
 required_providers {
   aws = {
     source  = "hashicorp/aws"
   }
 }
}
provider "aws" {
   region = "us-east-2"
}
data "aws_eks_cluster" "eks2" {
   name = "eks2"
   region = "us-east-2"
}
output "cluster-endpoint" {
   value = data.aws_eks_cluster.eks2.endpoint
 }

# Create an iam user

resource "aws_iam_user" "eks2-user" {
   name = "eks2-user"
}
resource "aws_iam_user_policy_attachment"
"eks2-user-policy-attachment" {
```

```
    user       = aws_iam_user.eks2-user.name
    policy_arn = "arn:aws:iam::aws:policy/AdministratorAccess"
}

resource "aws_eks_access_entry" "eks2-user-access" {
    principal_arn = aws_iam_user.eks2-user.arn
    cluster_name = data.aws_eks_cluster.eks2.name
    type = "STANDARD"
    kubernetes_groups = [ "system:masters" ]
}
```

*https://docs.aws.amazon.com/eks/latest/userguide/grant-k8s-access.html

aws eks describe-cluster --name eks2 --region us-east-2 --query 'cluster.accessConfig.authenticationMode' --output text

DAY - 7 →

Ex – Provision the labs and connect.

Case - 8 →

Eks provisioning with terraform →

Ex – Clean up case-8 previously done work, clone again from https://github.com/awsterraworkshop/case-8 and start with it.

aws eks update-kubeconfig
--name=demo-eks-cluster
--region=us-east-1

kubectl get nodes

Using ECR in EKS →

- If a node can pull an image from ecr private repo will largely depend upon the role that you have allocated to that machine.
- One can set the role and its permissions at node group level.
- https://docs.aws.amazon.com/AmazonECR/latest/userguide/repository-policy-examples.html

Case – 5 →

Github actions usage to deploy on eks cluster →

https://github.com/awsterraworkshop/case-5/blob/main/.github/workflows/eks-deploy.yml

https://github.com/awsterraworkshop/case-5

Ex – Try out deploying to eks using github actions.

Ex – Execute a simple terraform project from github runner. – create an iam user.

*remote state backend

Case - 6 →

AWS DevOps Capabilities →

- One can build and run entire devops pipelines using native aws services
- Aws code commit, code build , code deploy, code pipeline, code artifact…etc..
- https://aws.amazon.com/devops/
- Code commit – is a git enabled code repository.
- CodeBuild – fully managed build service
  - Interact with code repo, cloning, building. Testing, deploy, artifact management ….etc…
  - buildspec.yml – you can write build steps
  - https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html

- CodeDeploy →
  - Fully managed deployment service
  - Ec2, ecs, lambda …etc..
  - Deploy strategies
  - appspec.yml
  - https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file.html
- CodePipeline → it self doesnt really do anything but it acts as an orchestrator
- CodeArtifact → artifact manager for your build outcomes

Case - 6 →

1. Create a github repo in your account, keep it public and clone it locally.
2. https://github.com/awsterraworkshop/case6-buildfiles

3. Put all the files and make a commit.
4. https://github.com/awsterraworkshop/case-6/tree/master
5. Clone the above as case-6 terraform project and in main.tf you need to update your github repo, connection arn and eks cluster name.
6. Before you do terraform apply, ensure to update cluster name in buildspec-deploy.yml and also update the deployment.yaml raw url and update the image name nginx in deployment.yaml file.
7. Do terraform apply and replace the ecr url with latest tag in buildspec-deploy.yml file.
8. Very if deployment is created.
9.

Time till – 17.05

Rollout deployment →

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: webapp
 labels:
   app: webapp
spec:
 replicas: 5
 strategy:
   type: RollingUpdate
   rollingUpdate:
     maxUnavailable: 1
     maxSurge: 3
 selector:
   matchLabels:
     app: webapp
 template:
   metadata:
     labels:
       app: webapp
   spec:
     containers:
       - name: app
         image:
777669575376.dkr.ecr.us-east-1.amazonaws.com/case6-ecr:latest
         imagePullPolicy: Always
         ports:
           - containerPort: 80
```

```yaml
---
apiVersion: v1
kind: Service
metadata:
 name: webapp
spec:
 type: LoadBalancer
 selector:
   app: webapp
 ports:
   - port: 80
     targetPort: 80
```

Ex – Try it out by updating the current image with a totally different image like tomcat:9 and do a deployment watch, also set the number of replicas-8.

kubectl get deployments –watch

Q/A –

*terraform state list