:Introduction:

The exercise consists of deploying the AWS EC2 (Elastic Compute Cloud) Service and DynamoDB by means of IaC (Infrastructure as a code).

To accomplish this, below resources have been used:

1. ***Localstack*** – To Mock the AWS environment in your local system due to constraint of accessing and consume real AWS services.By the way, Localstack is best if you want to test Various AWS services in your local system.Its community edition is free and provides most of the mocked AWS Services like EC2,DynamoDB,SNS,IAM,S3 et***c***
2. ***Terraform*** – This is the best in industry IaC tool through which Small scale to Large Scale Infrastructure can be deploy by use of Code natively called “tf” Scripts.
3. ***Docker and Docker Consume*** – This is a Container in which out Localstack will run as a service.

:Basic Architecture of exercise:

REGION

Dynamo

DB

VPC

PUBLIC SUBNET

EC2 Instance

Internet Gateway

Internet

Route Table

IAM

AWS SECRET MANAGER

:Goal:

The goal is to deploy the **EC2** on **AWS** that can connect to **Dynamo DB**.

The deployment should be accomplish using IaaC (Terraform) over AWS (Here Localstack)

Following measures should be taken into consideration:

1. **Security** – All the security measures should be taken into consideration so that any attack or unauthorized access to resources cannot prevails.
2. **Solution** – Solution should be easy, precise and to the mark. No redundant resources should be used and IaC code must be short and understandable.

My Solution:

Environment Setup on Windows:

1. Install Docker , Docker-compose.
2. Prepare the “docker-compose.yml” file that contains the localstack configuration and create the localstack container within docker using below command-

*docker-compose up -d*

1. This will create all the mock AWS services (if SERVICES tag specify in yaml file then only selected services will be up ) in the container.

To view the logs run

*docker compose logs -f*

:Bottleneck:

I’ve also tried to download the localstack packages using “pip” and start it using localstack CLI but services are not coming up after finish –

Accessing the URL : <http://localhost:4566/health> gives following output.

{"services": {"dynamodbstreams": "available", "ec2": "available", "iam": "available", "sts": "available", "kinesis": "available", "s3": "available", "dynamodb": "avaialble"}}

The status should be “running”.

After googling of more than 2 hours , I conclude that Some compatibility issue with Windows with the latest version 0.14 .Even I tried with few old version i.e. 0.10.0 before but no luck

Finally, 0.12.6 version works with docker-compose

Accessing the URL : <http://localhost:4566/health> gives following output.

{"services": {"dynamodbstreams": "running", "ec2": "running", "iam": "running", "sts": "running", "kinesis": "running", "s3": "running", "dynamodb": "running"}}

Terraform Scripts Implementation:

Following Scripts have been created for various resources (checked into the git)-

1. **vpc\_creation.tf** : Creates terraform code for VPC and Public Subnet.
2. **AssumePolicy.tf**: Creates permission AssumeRole that defines two policies :
3. A trust policy that defines who can assume the role
4. Permission what can be done with the role.

Here , if a role has assumeRole policy attach then this role allow EC2 instance to assume the role and use attached permissions.Hence , all the applications running inside the EC2 has READ access.Moreover , if this policy is attached to a role then any user who have this role can pass this readOnly DynamoDB roles.

1. **IAM.tf :** Creates specified IAM users and access keys
2. **IAM\_Policies**.tf : Creates Read/Write IAM policies on DynamoDB specific table for sake of simplicity.
3. **IAM\_Roles**.tf : Creates roles and attach policies that are created earlier.
4. **ec2\_instance.tf** : Creates an ec2 instance with predefined type ,region and ami.
5. **dynamoDB.tf** : Creates simple dynamo DB table without autoscaling.
6. **Attach\_Roles\_Policies.tf** : Assign read roles normal Users and Read/Write roles to admin Users created earlier
7. **Security\_group.tf** : Creates a custom security group that would be assigned to EC2 instance
8. **Main.tf** : Configure a provider when “terraform init” executes . This contains the region , access keys , secret keys and endpoints to EC2,DynamoDB and IAM services.
9. **variables**.tf : Stores variables used in all the scripts.
10. **outputs**.tf : show what would be shown as output after executing terraform code.
11. Running Below terraform commands will deploy the solution

*terraform workspace new adidas*

*terraform init*

*terraform validate*

*terraform plan*

*terraform apply*

:Bottleneck**:**

**During EC2 instance creation through terraform on localstack**

Localstack community edition have some limitations to mock EC2 service.After so many failed attempts came to know that mocking EC2 provisioning on localstack exhibits errors.

Error: reading EC2 Instance (i-b48cffc5156363285) attribute: InvalidParameterValue: Value (disableApiStop) for parameter attribute is invalid. Unknown attribute.

│ status code: 400, request id: 7a62c49f-347e-4fc4-9331-6e8eEXAMPLE

│

│ with aws\_instance.ec2-instance,

│ on ec2\_instance.tf line 2, in resource "aws\_instance" "ec2-instance":

│ 2: resource "aws\_instance" "ec2-instance" {

:Security Considerations (Implemented):

1. AWS Secret Manger has been used to store access key and secret key in json format which is fetched run time on demand.
2. Separate roles have been created “ADMIN” and “NORMAL” for users to access DynamoDB.
3. Application within the EC2 can use temporary credentials to access DynamoDB using AssumeRole which has default time duration 900 secs.
4. Any IAM User is authorized to pass only read access roles to other resources.

:Security Best Practices that should be taken into consideration:

1. DynamoDB VPC Endpoint access can be used to provide maximum security to Organization data.All the communication between the EC2 and DynamoDB is on private channel separated from Internet Gateway as seen below :

REGION

VPC

PUBLIC SUBNET

EC2 Instance

Internet

Route Table

Internet Gateway

Dynamo

DB

IAM

AWS SECRET MANAGER

PRIVATE SUBNET

Route Table

EC2 Instance

1. AWS KMS can be used as a centralized storage for encrypted access and secret keys.
2. AWS KMS is integrated with AWS SDK to provide client side data encryption.
3. Multi Factor Authentication can be used for IAM Users as an additional Security measures
4. Provide least privileges with minimum time to any of the resource.