

From this lab onwards, we will be working with aws services! In this lab, we will learn how to make use of aws cli to explore and deploy IAM resources that we learnt in the previous lecture.

As explained in the Lab Demonstration video, we are making use of localstack as the mocking framework to work with AWS.

This framework is exposed at <http://aws:4566>. Hence, throughout this lab, we will be making use of the command option and parameter like this: `--endpoint http://aws:4566`

First, let's explore the AWS CLI configuration.

What is the exact version of the CLI installed on the iac-server?

Run: `aws --version`

```
iac-server $ aws --version
aws-cli/2.2.20 Python/3.8.8 Linux/5.4.0-1103-gcp exe/x86_64.ubuntu.18 prompt/off
iac-server $
```

Which command should be used to interact with Identity and Access Management in AWS using the CLI?

`iam , aws iam`

Which subcommand with iam can be used to list all the users created in aws?

If unsure, use the CLI help by running: `aws iam <subcommand> help`

Run: `aws iam list-users help`

Run: `aws iam list-users`

```
iac-server $ aws iam list-users
An error occurred (InvalidClientTokenId) when calling the ListUsers operation: The security token included in the request is invalid.
iac-server $
```

This is because, we have not provided the `--endpoint` option.

Now, run the same command but with the `--endpoint http://aws:4566` as the option, like this:

`aws --endpoint http://aws:4566 iam list-users`

```
iac-server $ aws --endpoint http://aws:4566 iam list-users
{
  "Users": [
    {
      "Path": "/",
      "UserName": "jill",
      "UserId": "wswktt4um15fo92nuqq",
      "Arn": "arn:aws:iam::000000000000:user/jill",
      "CreateDate": "2023-04-29T11:33:14.775000+00:00"
    },
    {
      "Path": "/",
      "UserName": "jack",
      "UserId": "k9qe1w0gnh7k2912ixwq",
      "Arn": "arn:aws:iam::000000000000:user/jack",
      "CreateDate": "2023-04-29T11:33:15.693000+00:00"
    }
  ]
}
```

Now let's add a few more users! To add more users, we need to make use of the create-user sub-command.

However, we also need to pass in a mandatory option with this command for it to work?

Use: `aws iam create-user help`, for an example. : `--user-name`

Create a new user called mary using the AWS CLI.

Make sure to use the --endpoint http://aws:4566 option with the command.

Run: `aws --endpoint http://aws:4566 iam create-user --user-name mary`

```
iac-server $ aws iam create-user --user-name mary
iac-server $ aws --endpoint http://aws:4566 iam create-user --user-name mary
{
  "User": {
    "Path": "/",
    "UserName": "mary",
    "UserId": "29qimr62494t157ff0ak",
    "Arn": "arn:aws:iam::000000000000:user/mary",
    "CreateDate": "2023-04-29T11:38:13.016000+00:00"
  }
}
iac-server $
```

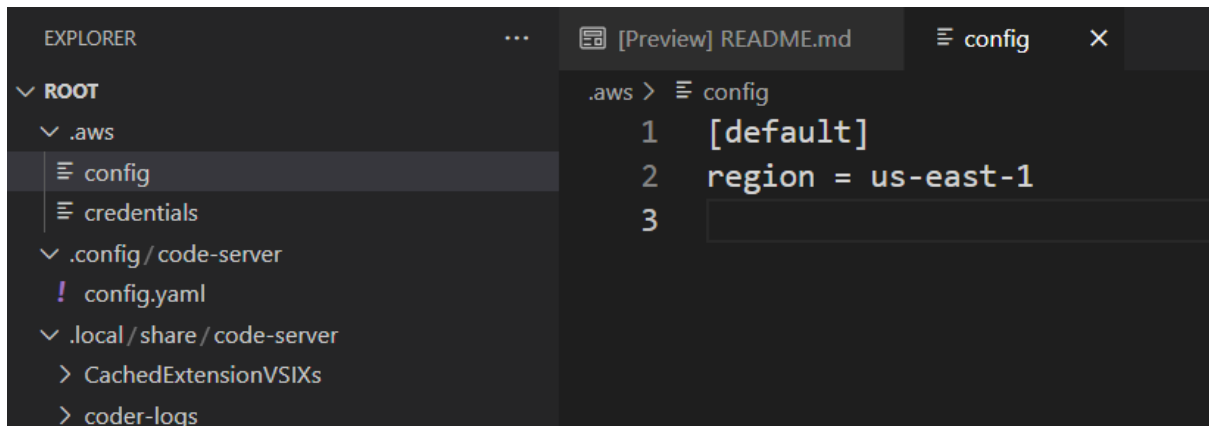
Now, inspect the newly created user mary and find out its ARN (Amazon Resource Name).

Run: `aws --endpoint http://aws:4566 iam list-users`

```
iac-server $ aws --endpoint http://aws:4566 iam list-users
{
  "Users": [
    {
      "Path": "/",
      "UserName": "jill",
      "UserId": "wwswktt4um15fo92nuqq",
      "Arn": "arn:aws:iam::000000000000:user/jill",
      "CreateDate": "2023-04-29T11:33:14.775000+00:00"
    },
    {
      "Path": "/",
      "UserName": "jack",
      "UserId": "k9qe1w0gnh7k2912ixwq",
      "Arn": "arn:aws:iam::000000000000:user/jack",
      "CreateDate": "2023-04-29T11:33:15.693000+00:00"
    },
    {
      "Path": "/",
      "UserName": "mary",
      "UserId": "29qimr62494t157ff0ak",
      "Arn": "arn:aws:iam::000000000000:user/mary",
      "CreateDate": "2023-04-29T11:38:13.016000+00:00"
    }
  ]
}
```

What is the default region that has been configured for use with the AWS CLI?

Inspect the configuration file `/root/.aws/config` file.

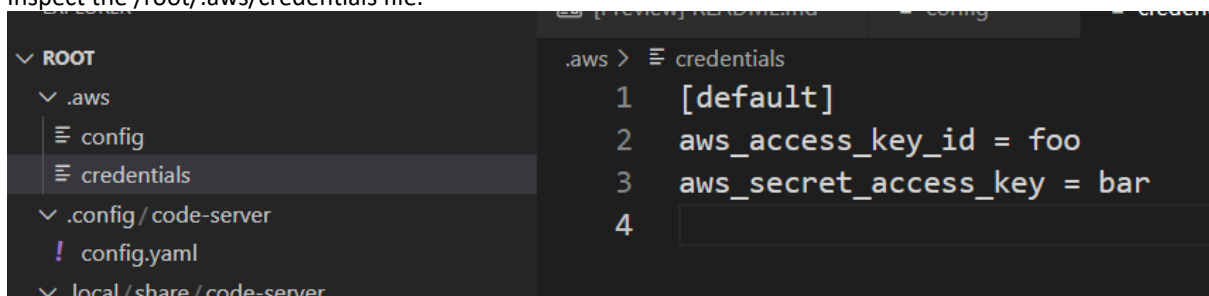
A screenshot of a code editor showing a file explorer on the left and a configuration file on the right. The file explorer shows a project structure with folders like .aws, .config, and .local. The .aws folder is expanded, showing config and credentials files. The config file is open in the editor, showing a [default] section with a region = us-east-1.

```
EXPLORER
...
[Preview] README.md
config
x

ROOT
  .aws
    config
    credentials
  .config / code-server
    config.yaml
  .local / share / code-server
    CachedExtensionVSIXs
    coder-logs

.aws > config
1 [default]
2 region = us-east-1
3
```

What is the `aws_access_key_id` used in the configuration?
Inspect the `/root/.aws/credentials` file.

A screenshot of a code editor showing a file explorer on the left and a credentials file on the right. The file explorer shows the same project structure as before, but the credentials file is now open. The credentials file shows a [default] section with aws_access_key_id = foo and aws_secret_access_key = bar.

```
ROOT
  .aws
    config
    credentials
  .config / code-server
    config.yaml
  .local / share / code-server

.aws > credentials
1 [default]
2 aws_access_key_id = foo
3 aws_secret_access_key = bar
4
```

What is the value of `aws_secret_access_key` used?

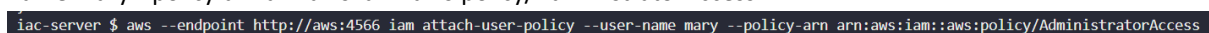
Now that we have a few users created, let's grant them privileges.

Let's start with mary, grant her full administrator access by making use of the policy called `AdministratorAccess`.

Make use of the subcommand `attach-user-policy`.

The ARN of the `AdministratorAccess` policy is `arn:aws:iam::aws:policy/AdministratorAccess`.

Use the subcommand `attach-user-policy`. Run: `aws --endpoint http://aws:4566 iam attach-user-policy --user-name mary --policy-arn arn:aws:iam::aws:policy/AdministratorAccess`

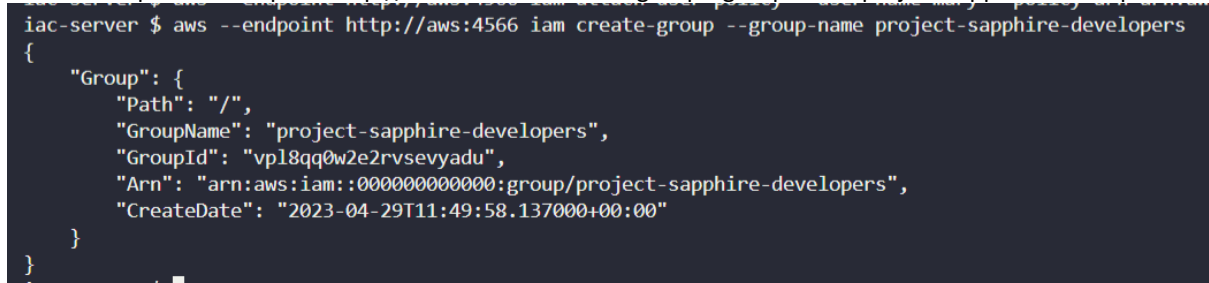
A terminal screenshot showing the command: iac-server \$ aws --endpoint http://aws:4566 iam attach-user-policy --user-name mary --policy-arn arn:aws:iam::aws:policy/AdministratorAccess

jack and jill are developers and are part of a project called `project-sapphire`.

Create a new IAM Group called `project-sapphire-developers`.

Use the subcommand `create-group` to create the group.

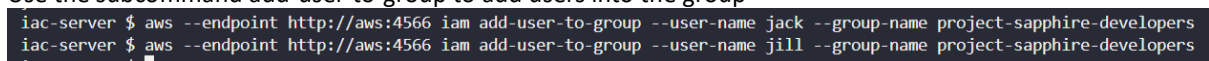
Create Group: `aws --endpoint http://aws:4566 iam create-group --group-name project-sapphire-developers`

A terminal screenshot showing the command: iac-server \$ aws --endpoint http://aws:4566 iam create-group --group-name project-sapphire-developers. The output is a JSON object representing the created group.

```
iac-server $ aws --endpoint http://aws:4566 iam create-group --group-name project-sapphire-developers
{
  "Group": {
    "Path": "/",
    "GroupName": "project-sapphire-developers",
    "GroupId": "vp18qq0w2e2rvsevyadu",
    "Arn": "arn:aws:iam::000000000000:group/project-sapphire-developers",
    "CreateDate": "2023-04-29T11:49:58.137000+00:00"
  }
}
```

Add the IAM users called jack and jill, who are developers to the new IAM Group called `project-sapphire-developers`.

Use the subcommand `add-user-to-group` to add users into the group

A terminal screenshot showing two commands: iac-server \$ aws --endpoint http://aws:4566 iam add-user-to-group --user-name jack --group-name project-sapphire-developers and iac-server \$ aws --endpoint http://aws:4566 iam add-user-to-group --user-name jill --group-name project-sapphire-developers

What privileges are granted for jack and jill who are part of the group project-sapphire-developers? Check for their permissions individually and the ones granted to the group.

```
iac-server $ aws --endpoint http://aws:4566 iam list-attached-group-policies --group-name project-sapphire-developers
{
  "AttachedPolicies": []
}
iac-server $ aws --endpoint http://aws:4566 iam list-attached-user-policies --user-name jack
{
  "AttachedPolicies": []
}
```

Both jack and jill need complete access to the EC2 service.

Attach the AmazonEC2FullAccess policy with the ARN: arn:aws:iam::aws:policy/AmazonEC2FullAccess to the group project-sapphire-developers.

```
iac-server $ aws --endpoint http://aws:4566 iam attach-group-policy --group-name project-sapphire-developers --policy-arn arn:aws:iam::aws:policy/AmazonEC2FullAccess
iac-server $ aws --endpoint http://aws:4566 iam list-attached-group-policies --group-name project-sapphire-developers
{
  "AttachedPolicies": [
    {
      "PolicyName": "AmazonEC2FullAccess",
      "PolicyArn": "arn:aws:iam::aws:policy/AmazonEC2FullAccess"
    }
  ]
}
```

IAM with terraform:

In this lab, we will learn how to deploy AWS resources using Terraform with the same mocking service that we use in the AWS CLI lab.

Let's start off by creating an IAM User called mary but this time by making use of Terraform. In the configuration directory /root/terraform-projects/IAM, create a file called iam-user.tf with the following specifications:

Resource Type: aws_iam_user

Resource Name: users

Name: mary

Once the file has been created, run terraform init.

iam-user.tf

```
resource "aws_iam_user" "users" {
  name = "mary"
}
```

Great! We now have a configuration file with a simple resource block for creating an IAM User with Terraform!

Let's check if everything is in order for us to create this resource.

Run terraform plan within this configuration.

Move to the directory first cd /terraform-projects/IAM/ and then run terraform validate.

```
Commands will detect it and remind you to do so if necessary.
iac-server $ terraform validate
Success! The configuration is valid.

iac-server $ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

-----

Error: configuring Terraform AWS Provider: validating provider credentials: retrieving caller identity from STS: operation error STS:
GetCallerIdentity, failed to resolve service endpoint, an AWS region is required, but was not found

on <empty> line 0:
(source code not available)
```

As we learned in the lecture, the AWS provider requires a region to be defined. Either in the provider block or by setting a value for the variable called region. This has not been done so far.

Let's do that now. We will add the argument region in our provider block called aws. We can do this via other means as well (like the ones we saw in the lecture). However, we will be making use of the provider block to define additional arguments to make use of the mocking framework. We will see those in the later questions of this lab.

Add a new file called provider.tf containing a provider block for aws. Inside this block add a single argument called region with the value ca-central-1. You don't have to run a terraform plan or apply at this stage.

Provider.tf

```
provider "aws" {  
  region = "ca-central-1"  
}
```

iam-user.tf

```
resource "aws_iam_user" "users" {  
  name = "mary"  
}
```

Since we are making use of the mocking framework, the credentials defined using aws configure (stored within the file /root/.aws/credentials) do not work as it is.

We have now updated the provider.tf file with additional arguments to make it work. Take a look.

The endpoint argument is similar to the one we saw with the AWS CLI where we used the --endpoint http://aws:4566. Here we have defined it to make it work with the IAM service.

Please note that these additional configurations are not needed when working directly with the AWS Cloud. It is only needed by the lab as it is using an AWS mock framework.

Provider.tf

```
provider "aws" {  
  region = "us-east-1"  
  skip_credentials_validation = true  
  skip_requesting_account_id = true  
  
  endpoints {  
    iam = "http://aws:4566"  
  }  
}
```

iam-user.tf

```
resource "aws_iam_user" "users" {  
  name = "mary"  
}
```

```
iac-server $ terraform apply
```

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:

+ create

Terraform will perform the following actions:

```
# aws_iam_user.users will be created
+ resource "aws_iam_user" "users" {
  + arn           = (known after apply)
  + force_destroy = false
  + id           = (known after apply)
  + name         = "mary"
  + path         = "/"
  + tags_all     = (known after apply)
  + unique_id    = (known after apply)
}
```

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

Do you want to perform these actions?

Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

aws_iam_user.users: Creating...

aws_iam_user.users: Creation complete after 1s [id=mary]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Great! We have added one user called mary. However, project_sapphire has 5 more people who need access to the AWS Account!

Let's use the count meta-argument and the new variables.tf file created in the configuration directory to create these additional users!

Inspect the newly created variables.tf file and answer the subsequent questions.

Now, update the iam-user.tf to make use of the count meta-argument to loop through the project-sapphire-users variable and create all the users in the list.

You may want to make use of the length function to get the length of the list.

Variables.tf

```
variable "project-sapphire-users" {
  type = list(string)
  default = ["mary", "jack", "jill", "mack", "buzz", "mater"]
}
```

Provider.tf

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

```
skip_requesting_account_id = true

endpoints {
  iam          = "http://aws:4566"
}
}
```

iam-user.tf

```
resource "aws_iam_user" "users" {
  name  = var.project-sapphire-users[count.index]
  count = length(var.project-sapphire-users)
}
```

In this lab, we will work on configuration directories that have been created under `/root/terraform-projects/S3-Buckets`.

Main.tf

```
resource "aws_s3_bucket" "marvel-cinematic-universe" {
  bucket = "mcu-202011121359"
}
```

Provider.tf

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "4.15.0"
    }
  }
}

provider "aws" {
  region          = var.region
  s3_use_path_style = true
  skip_credentials_validation = true
  skip_requesting_account_id = true

  endpoints {
    s3          = "http://aws:4566"
  }
}
```

Terraform.tfvars

```
region = "us-east-1"
```

```
variables.tf
variable "region" {
}
```

The main.tf file is empty. Use it to create a new S3 with the following specifications:

resource name: `dc_bucket`

bucket name: `dc_is_better_than_marvel`

Once the resource block is complete, run a `terraform init`, `plan` and `apply` to try and create the bucket.

If unsure, refer to the documentation. The documentation tab is available at the top right panel.

Main.tf

```
resource "aws_s3_bucket" "dc_bucket" {  
  bucket = "dc_is_better_than_marvel"  
}
```

Provider.tf

```
terraform {  
  required_providers {  
    aws = {  
      source = "hashicorp/aws"  
      version = "4.15.0"  
    }  
  }  
}  
  
provider "aws" {  
  region = var.region  
  s3_use_path_style = true  
  skip_credentials_validation = true  
  skip_requesting_account_id = true  
  
  endpoints {  
    s3 = "http://aws:4566"  
  }  
}
```

Terraform.tfvars

```
region = "us-east-1"
```

variables.tf

```
variable "region" {  
}
```

That's right! The bucket name we used does not conform to a DNS Name standard as it uses underscores.

Let's fix that now and change the bucket name so that it uses dashes (-) instead of underscore(_).

resource name: dc_bucket

bucket name: dc-is-better-than-marvel

Once the resource block is complete, run a terraform init, plan and apply to try and create the bucket.

Main.tf

```
resource "aws_s3_bucket" "dc_bucket" {  
  bucket = "dc-is-better-than-marvel"  
}
```

Let's move on to the next configuration directory called Pixar.

Same as the directory called DC, we have the provider.tf, variables.tf, terraform.tfvars and an empty main.tf file that is already created.

We have a file called woody.jpg stored at /root that has to be uploaded to a bucket called pixar-studios-2020. This bucket already exists though and was created using the AWS CLI.

Let's do that now and upload this image to the s3 bucket! Update the main.tf file with the following specifications:

Bucket: pixar-studios-2020

Key: woody.jpg

Source: /root/woody.jpg

Once ready, proceed to run terraform init, plan and apply.

Main.tf

```
resource "aws_s3_object" "upload" {  
  bucket = "pixar-studios-2020"  
  key = "woody.jpg"  
  source = "/root/woody.jpg"  
}
```

Provider.tf

```
terraform {  
  required_providers {  
    aws = {  
      source = "hashicorp/aws"  
      version = "4.15.0"  
    }  
  }  
}  
  
provider "aws" {  
  region = var.region  
  s3_use_path_style = true  
  skip_credentials_validation = true  
  skip_requesting_account_id = true  
  
  endpoints {  
    s3 = "http://aws:4566"  
  }  
}
```

Terraform.tfvars

```
region = "us-east-1"
```

variables.tf

```
variable "region" {  
}
```

In this lab, we will work with DynamoDB tables using terraform.

The configuration directories have been created under /root/terraform-projects/DynamoDB.

We have already created a resource block for a DynamoDB table inside /root/terraform-projects/DynamoDB/project-sapphire-user-data/.

But something is wrong with this configuration. Try running a terraform plan or validate and identify the cause of the failure.

Main.tf

```
resource "aws_dynamodb_table" "project_sapphire_user_data" {  
  name = "userdata"  
  billing_mode = "PAY_PER_REQUEST"  
  hash_key = "UserId"  
  
  attribute {  
    name = "Name"  
    type = "S"  
  }  
}
```

Provider.tf

```
provider "aws" {
  region          = var.region
  s3_force_path_style = true
  skip_credentials_validation = true
  skip_requesting_account_id = true

  endpoints {
    dynamodb = "http://aws:4566"
  }
}
```

Variables.tf

```
variable "region" {
  default = "us-west-2"
}
```

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

```
iac-server $ terraform plan
```

Refreshing Terraform state in-memory prior to plan...

The refreshed state will be used to calculate this plan, but will not be persisted to local or remote state storage.

Warning: Argument is deprecated

Use s3_use_path_style instead.

Warning: Attribute Deprecated

Use s3_use_path_style instead.

Error: 2 errors occurred:

- * all attributes must be indexed. Unused attributes: ["Name"]

- * all indexes must match a defined attribute. Unmatched indexes: ["UserId"]

on main.tf line 1, in resource "aws_dynamodb_table" "project_sapphire_user_data":
1: resource "aws_dynamodb_table" "project_sapphire_user_data" {

Attribute should be defined for primary key .

That's right! At a minimum, the Primary Key should be defined as an attribute when creating a DynamoDB table.

In this configuration, we should add the attribute for UserId for it to work.

Let's fix that now! Update the main.tf file so that it uses an attribute for the Primary/Hash Key.

Note that the UserId should be a number.

Once ready, run a terraform init, plan and apply.

Update type as Number for the UserID in the main.tf.

Main.tf

```
resource "aws_dynamodb_table" "project_sapphire_user_data" {
  name      = "userdata"
  billing_mode = "PAY_PER_REQUEST"
  hash_key   = "UserId"

  attribute {
    name = "UserId"
    type = "N"
  }
}
```

Another table has been created using the configuration in the directory `/root/terraform-projects/DynamoDB/project-sapphire-inventory`.

Main.tf

```
resource "aws_dynamodb_table" "project_sapphire_inventory" {
  name      = "inventory"
  billing_mode = "PAY_PER_REQUEST"
  hash_key   = "AssetID"

  attribute {
    name = "AssetID"
    type = "N"
  }
  attribute {
    name = "AssetName"
    type = "S"
  }
  attribute {
    name = "age"
    type = "N"
  }
  attribute {
    name = "Hardware"
    type = "B"
  }
  global_secondary_index {
    name      = "AssetName"
    hash_key   = "AssetName"
    projection_type = "ALL"
  }
  global_secondary_index {
    name      = "age"
    hash_key   = "age"
    projection_type = "ALL"
  }
  global_secondary_index {
    name      = "Hardware"
    hash_key   = "Hardware"
    projection_type = "ALL"
  }
}
```

Provider.tf

```
provider "aws" {
  region          = var.region
  s3_force_path_style = true
  skip_credentials_validation = true
  skip_requesting_account_id = true

  endpoints {
    dynamodb = "http://aws:4566"
  }
}
```

Variable.tf

```
variable "region" {
  default = "us-west-2"
}
```

Now, let's add an item to this table called inventory. Use the following specifications and update the main.tf file:

Resource Name: upload

Table = Use reference expression to the table called inventory

Hash Key = Use reference expression to use the primary key used by the table inventory

Use the below data for item:

```
{
  "AssetID": {"N": "1"},
  "AssetName": {"S": "printer"},
  "age": {"N": "5"},
  "Hardware": {"B": "true" }
}
```

when ready, run terraform init, plan and apply

main.tf

```
resource "aws_dynamodb_table" "project_sapphire_inventory" {
  name         = "inventory"
  billing_mode = "PAY_PER_REQUEST"
  hash_key     = "AssetID"

  attribute {
    name = "AssetID"
    type = "N"
  }
  attribute {
    name = "AssetName"
    type = "S"
  }
  attribute {
    name = "age"
    type = "N"
  }
  attribute {
    name = "Hardware"
    type = "B"
  }
  global_secondary_index {
    name       = "AssetName"
    hash_key   = "AssetName"
  }
}
```

```

    projection_type = "ALL"

}
global_secondary_index {
    name          = "age"
    hash_key      = "age"
    projection_type = "ALL"

}
global_secondary_index {
    name          = "Hardware"
    hash_key      = "Hardware"
    projection_type = "ALL"

}
}

resource "aws_dynamodb_table_item" "upload" {
    table_name = aws_dynamodb_table.project_sapphire_inventory.name
    hash_key   = aws_dynamodb_table.project_sapphire_inventory.hash_key
    item = <<EOF
    {
        "AssetID" : {"N": "1"},
        "AssetName": {"S": "printer"},
        "age": {"N": "5"},
        "Hardware": {"B": "true" }
    }
    EOF
}

```