

Build an API Gateway and integrate it to a Lambda function using Terraform

Level: Intermediate

AWS Lambda   Amazon Web Services   Terraform   Amazon API Gateway



0h 44m 17s left



K ▾

Validation

Lab Credentials

User Name ⓘ

Whiz\_User\_80425.20701170



Password ⓘ

c6eb9c93-6ff6-4282-a616-406929857994



Access Key ⓘ

AKIAYKGOODAZ5HD2PLVY



Secret Key ⓘ

NkVCV5RwCxxM9EogRhm91AdlYdT59gGSCKb97cWY






Lab Resources


- 1. [Download Zip](#)

Support Documents

No Support Documents Found

## Need help?

-  How to use Hands on Lab
-  Troubleshooting Lab
-  FAQs

[Submit Feedback](#)[Share](#)[Lab Overview](#)[Lab Steps](#)[Lab Validation](#) Cloud Architect, Cloud Administrator Compute, Serverless, Infrastructure

# Lab Steps

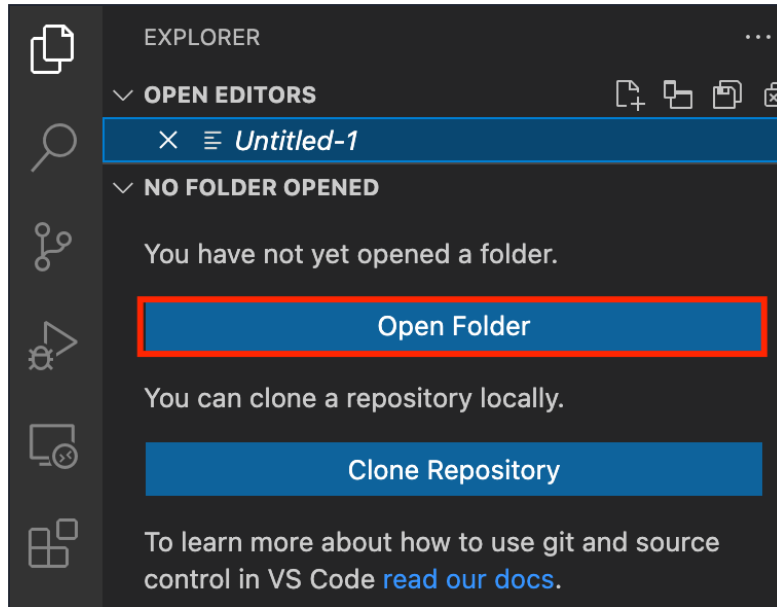
## Task 1: Sign in to AWS Management Console

1. Click on the **Open Console** button, and you will get redirected to AWS Console in a new browser tab.
2. On the AWS sign-in page,
  - Leave the **Account ID** as default. Never edit/remove the 12 digit Account ID present in the AWS Console. otherwise, you cannot proceed with the lab.
  - Now copy your **User Name** and **Password** in the Lab Console to the **IAM Username** and **Password** in AWS Console and click on the **Sign in** button
3. Once Signed In to the AWS Management Console, Make the default AWS Region as **US East (N. Virginia) us-east-1**.

## Task 2: Setup Visual Studio Code

1. Open the visual studio code.
2. If you have already installed and using Visual Studio code, open a new window.
3. A new window will open a new file and release notes page (only if you have installed or updated Visual Studio Code recently). Close the Release notes tab.

4. Open Terminal by selecting View from the Menu bar and choose Terminal.
5. It may take up to 2 minutes to open the terminal window.



6. Once the terminal is ready, let us navigate to the **Desktop**.

```
cd Desktop
```



7. Create a new folder by running the below command.

```
mkdir task_10106_api
```



8. Change your present working directory to use the newly created folder by running the below command:

```
cd task_10106_api
```



9. Get the location of the present working directory by running the below command:

```
pwd
```



10. Note down the location, as you will open the same in the next steps.
11. Now click on the first icon Explorer present on the left sidebar.
12. Click on the button called Open folder and navigate to the location of folder **task\_10106\_api**
13. (Optional) Click on Authorize button for allowing Visual Studio Code to use the **task\_10096\_api** folder. This will only be asked when you have been using Visual



Studio code for a while as you are allowing a new folder to be accessed by VSC.

14. Visual Studio Code is now ready to use.

### Task 3: Create a variable file

In this task, you will create a variable file where you will declare all the global variables with a short description and a default value.

1. To create a variable file, expand the folder **task\_10106\_api** and click on the **New File** icon to add the file.
2. Name the file as **variables.tf** and press Enter to save it.
3. **Note: Don't change the location of the new file, keep it default, i.e. inside the task\_10106\_api folder.**
4. Paste the below contents in **variables.tf** file.

```
variable "access_key" {  
    description = "Access key to AWS console"  
}  
variable "secret_key" {  
    description = "Secret key to AWS console"  
}  
variable "region" {  
    description = "AWS region"  
}
```



5. In the above content, you are declaring a variable called, access\_key, secret\_key, and region with a short description of all 3.
6. After pasting the above contents, save the file by pressing **ctrl + S**.
7. Now expand the folder **task\_10106\_api** and click on the **New File** icon to add the file.
8. Name the file as **terraform.tfvars** and press Enter to save it.
9. Paste the below content into the **terraform.tfvars** file.

```
region = "us-east-1"  
access_key = "<YOUR_ACCESS_KEY>"  
secret_key = "<YOUR_SECRET_KEY>"
```



10. In the above code, you are defining the dynamic values of variables declared earlier.
11. Replace the values of access\_key and secret\_key by copying from the lab page.



- After replacing the values of `access_key` and `secret_key`, save the file by pressing **Ctrl + S**.



```

variables.tf  terraform.tfvars X
terraform.tfvars > secret_key
1  region = "us-east-1"
2
3  access_key = "[REDACTED]"
4
5  secret_key = "[REDACTED]"

```

## Task 4: Create an IAM role for Lambda in the main.tf file

In this task, you will create a **main.tf** file where you will add details of the provider and resources.

- To create a **main.tf** file, expand the folder **task\_10106\_api**, and click on the New File icon to add the file.
- Name the file as **main.tf** and press **Enter** to save it.
- Paste the below content into the **main.tf** file.

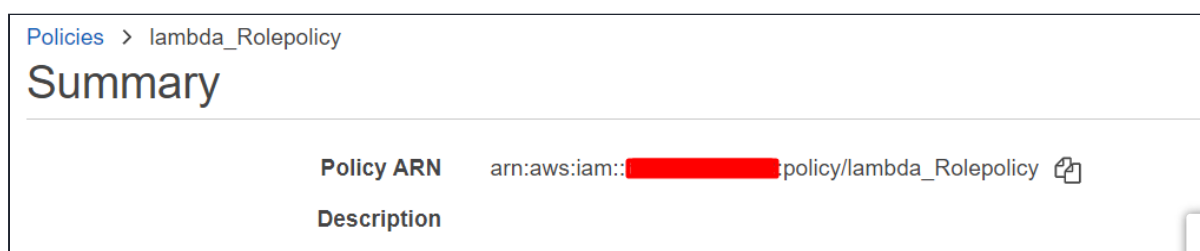
```

provider "aws" {
  region      = "${var.region}"
  access_key  = "${var.access_key}"
  secret_key  = "${var.secret_key}"
}

```



- In the above code, you are defining the provider as **AWS**.
- Next, we want to tell Terraform to create an IAM role for the Lambda function.
- In AWS Console, navigate to the **IAM** by clicking on **Services** on the top and search for **IAM**.
- Click** on the **Policies** on the left navigation panel and search **lambda\_Rolepolicy**. **Copy the ARN** and save it in notepad.



Policies > lambda\_Rolepolicy

### Summary

Policy ARN	arn:aws:iam::[REDACTED]:policy/lambda_Rolepolicy
Description	

- Paste the following code into the **main.tf** file.



```
resource "aws_iam_role" "iam_for_lambda" {
  name = "iam_for_lambda"
  assume_role_policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Effect": "Allow",
      "Sid": ""
    }
  ]
}
EOF
}
```

9. We will also describe the policy to the IAM role created. This policy will provide Lambda function access to CloudWatch logs.

10. Paste the below content into the **main.tf** file after the provider. Paste the policy arn copied earlier.

```
resource "aws_iam_role_policy_attachment" "lambda_policy" {
  role      = aws_iam_role.iam_for_lambda.name
  policy_arn = "<PASTE_POLICY_ARN_HERE>"
}
```



```

main.tf
5   secret_key = "${var.secret_key}"
6   }
7   resource "aws_iam_role" "iam_for_lambda" {
8       name = "iam_for_lambda"
9
10      assume_role_policy = <<EOF
11      {
12          "Version": "2012-10-17",
13          "Statement": [
14              {
15                  "Action": "sts:AssumeRole",
16                  "Principal": {
17                      "Service": "lambda.amazonaws.com"
18                  },
19                  "Effect": "Allow",
20                  "Sid": ""
21              }
22          ]
23      }
24      EOF
25  }
26
27  resource "aws_iam_role_policy_attachment" "lambda_policy" {
28      role      = aws_iam_role.iam_for_lambda.name
29      policy_arn = "<PASTE_POLICY_ARN_HERE>"
30  }

```

11. Save the file by pressing **Ctrl + S**.

## Task 5: Create a lambda function in main.tf file

In this task, we are going to create a lambda function.

1. Firstly you will download [lambda\\_function.zip](#) and upload it locally to your folder **task\_10106\_api**.
2. Name the zip file as **lambda\_function.zip**.



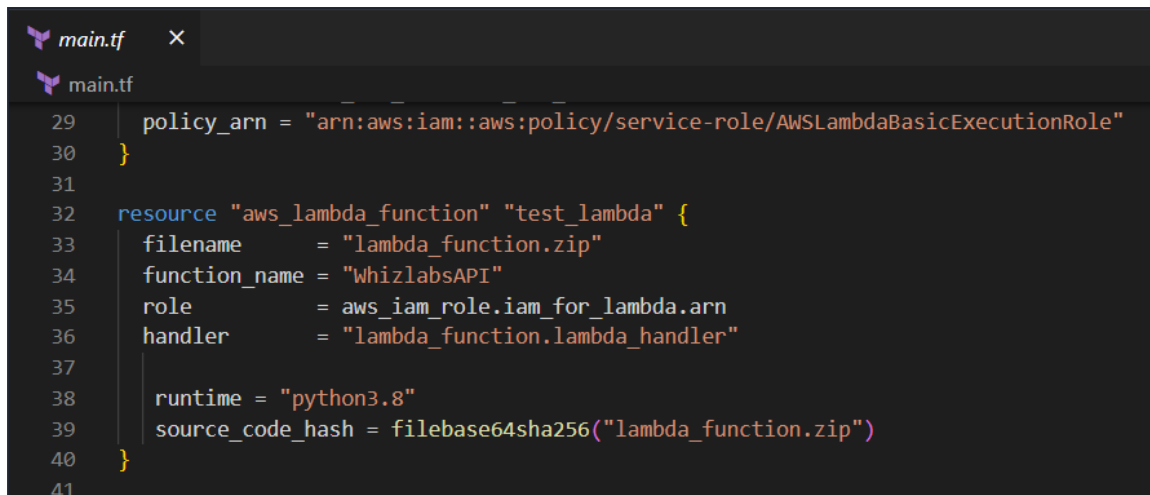
3. To create a lambda function, paste the following contents in the **main.tf** file

```

resource "aws_lambda_function" "test_lambda" {
    filename      = "lambda_function.zip"
    function_name = "WhizlabsAPI"
    role          = aws_iam_role.iam_for_lambda.arn
    handler       = "lambda_function.lambda_handler"
    runtime       = "python3.8"
    source_code_hash = filebase64sha256("lambda_function.zip")
}

```





```

29 | policy_arn = "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
30 | }
31 |
32 | resource "aws_lambda_function" "test_lambda" {
33 |     filename      = "lambda_function.zip"
34 |     function_name = "WhizlabsAPI"
35 |     role          = aws_iam_role.iam_for_lambda.arn
36 |     handler       = "lambda_function.lambda_handler"
37 |
38 |     runtime = "python3.8"
39 |     source_code_hash = filebase64sha256("lambda_function.zip")
40 | }
41 |

```

4. Save the file by pressing **Ctrl + S**.

## Task 6: Create a REST API, it's method and resource in main.tf file

In this task, we are going to create a Rest API and its method and resource and paste it in the **main.tf** file.

1. To create a REST API, add another block just below the upload object code in the **main.tf** file.

```

resource "aws_api_gateway_rest_api" "testAPI" {
    name          = "WhizAPI"
    description   = "This is my API for demonstration purposes"
    endpoint_configuration {
        types      = ["REGIONAL"]
    }
}

```



2. In the above code, we have declared the name, description, and type of the rest API created.

3. To create a resource for the Rest API, paste the following contents in the **main.tf** file.

```

resource "aws_api_gateway_resource" "testresource" {
    parent_id     = aws_api_gateway_rest_api.testAPI.root_resource_id
    path_part     = "whizapi"
    rest_api_id   = aws_api_gateway_rest_api.testAPI.id
}

```



4. In the above code, we have declared the path part, parent id, and the REST API associated with it.

5. For creating the method, paste the following code in the **main.tf** file:







```
resource "aws_api_gateway_method" "testMethod" {
  rest_api_id  = aws_api_gateway_rest_api.testAPI.id
  resource_id  = aws_api_gateway_resource.testresource.id
  http_method  = "GET"
  authorization = "NONE"
}
```

6. To create a method response, paste the following code in the **main.tf** file.



```
resource "aws_api_gateway_method_response" "response_200" {
  rest_api_id = aws_api_gateway_rest_api.testAPI.id
  resource_id = aws_api_gateway_resource.testresource.id
  http_method = aws_api_gateway_method.testMethod.http_method
  status_code = "200"
  response_models = {
    "application/json" = "Empty"
  }
}
```

```
main.tf x
main.tf
41
42
43 resource "aws_api_gateway_rest_api" "testAPI" {
44     name          = "WhizAPI"
45     description   = "This is my API for demonstration purposes"
46     endpoint_configuration {
47         types      = ["REGIONAL"]
48     }
49 }
50
51 resource "aws_api_gateway_resource" "testresource" {
52     parent_id     = aws_api_gateway_rest_api.testAPI.root_resource_id
53     path_part     = "whizapi"
54     rest_api_id   = aws_api_gateway_rest_api.testAPI.id
55 }
56
57 resource "aws_api_gateway_method" "testMethod" {
58     rest_api_id   = aws_api_gateway_rest_api.testAPI.id
59     resource_id   = aws_api_gateway_resource.testresource.id
60     http_method   = "GET"
61     authorization = "NONE"
62 }
63
64 resource "aws_api_gateway_method_response" "response_200" {
65     rest_api_id   = aws_api_gateway_rest_api.testAPI.id
66     resource_id   = aws_api_gateway_resource.testresource.id
67     http_method   = aws_api_gateway_method.testMethod.http_method
68     status_code   = "200"
69     response_models = {
70         "application/json" = "Empty"
71     }
72 }
73
```

7. Save the file by pressing **Ctrl + S**.



## Task 7: Create a gateway integration and deploy the API in the main.tf file

1. To create a gateway integration, paste the following code in the **main.tf** file.

```
resource "aws_api_gateway_integration" "MyDemoIntegration" {  
  rest_api_id      = aws_api_gateway_rest_api.testAPI.id  
  resource_id      = aws_api_gateway_resource.testresource.id  
  http_method      = aws_api_gateway_method.testMethod.http_method  
  integration_http_method = "POST"  
  uri              = aws_lambda_function.test_lambda.invoke_arn  
  type             = "AWS"  
  passthrough_behavior = "WHEN_NO_TEMPLATES"  
}
```

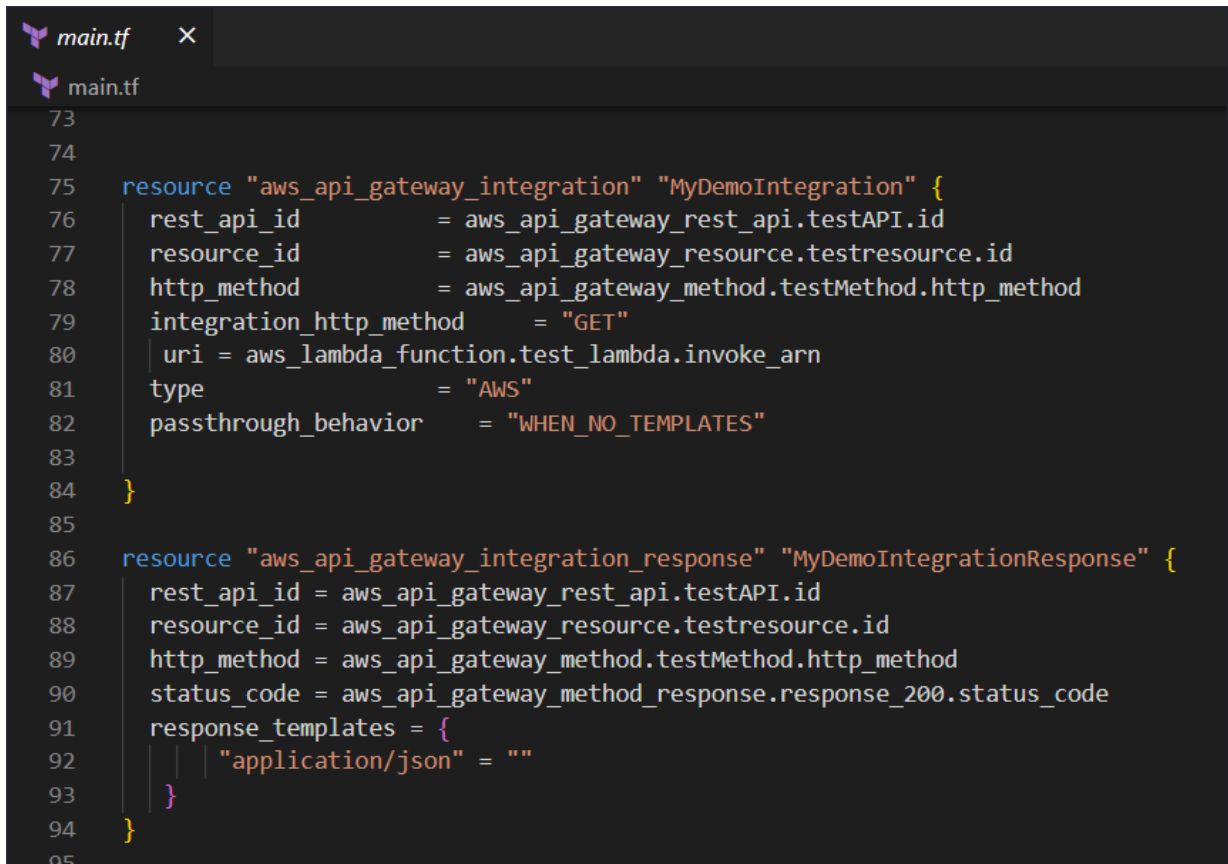


2. In the above code, we have declared the rest API, resource, method and lambda function invoke arn.

3. To add the gateway integration response, paste the following in the **main.tf** file

```
resource "aws_api_gateway_integration_response" "MyDemoIntegrationResponse" {  
  rest_api_id = aws_api_gateway_rest_api.testAPI.id  
  resource_id = aws_api_gateway_resource.testresource.id  
  http_method = aws_api_gateway_method.testMethod.http_method  
  status_code = aws_api_gateway_method_response.response_200.status_code  
  response_templates = {  
    "application/json" = ""  
  }  
  depends_on = [  
    aws_api_gateway_integration.MyDemoIntegration  
  ]  
}
```





```

main.tf x
main.tf
73
74
75 resource "aws_api_gateway_integration" "MyDemoIntegration" {
76     rest_api_id      = aws_api_gateway_rest_api.testAPI.id
77     resource_id      = aws_api_gateway_resource.testresource.id
78     http_method      = aws_api_gateway_method.testMethod.http_method
79     integration_http_method = "GET"
80     uri              = aws_lambda_function.test_lambda.invoke_arn
81     type             = "AWS"
82     passthrough_behavior = "WHEN_NO_TEMPLATES"
83 }
84
85
86 resource "aws_api_gateway_integration_response" "MyDemoIntegrationResponse" {
87     rest_api_id = aws_api_gateway_rest_api.testAPI.id
88     resource_id = aws_api_gateway_resource.testresource.id
89     http_method = aws_api_gateway_method.testMethod.http_method
90     status_code = aws_api_gateway_method_response.response_200.status_code
91     response_templates = {
92         "application/json" = ""
93     }
94 }
95

```

4. To deploy the API, add the below contents in the **main.tf** file:

```

resource "aws_api_gateway_deployment" "testdep" {
    rest_api_id = aws_api_gateway_rest_api.testAPI.id
    triggers = {
        redeployment = sha1(jsonencode([
            aws_api_gateway_resource.testresource.id,
            aws_api_gateway_method.testMethod.id,
            aws_api_gateway_integration.MyDemoIntegration.id,
        ]))
    }
    depends_on = [aws_api_gateway_integration.MyDemoIntegration]
    lifecycle {
        create_before_destroy = true
    }
}

```



```

main.tf
95
96 resource "aws_api_gateway_deployment" "testdep" {
97     rest_api_id = aws_api_gateway_rest_api.testAPI.id
98     triggers = {
99         redeployment = sha1(jsonencode([
100             aws_api_gateway_resource.testresource.id,
101             aws_api_gateway_method.testMethod.id,
102             aws_api_gateway_integration.MyDemoIntegration.id,
103         ]))
104     }
105
106     depends_on = [aws_api_gateway_integration.MyDemoIntegration]
107     lifecycle {
108         create_before_destroy = true
109     }
110 }
111

```

5. For deployment, there is a need to create a stage as well. To create the stage for the deployment, paste the following contents in the **main.tf** file:

```

resource "aws_api_gateway_stage" "teststage" {
    deployment_id = aws_api_gateway_deployment.testdep.id
    rest_api_id   = aws_api_gateway_rest_api.testAPI.id
    stage_name    = "whizstage"
}

```



```

main.tf
110
111
112
113 resource "aws_api_gateway_stage" "teststage" {
114     deployment_id = aws_api_gateway_deployment.testdep.id
115     rest_api_id   = aws_api_gateway_rest_api.testAPI.id
116     stage_name    = "whizstage"
117 }
118

```

6. The last thing we need in the **main.tf** file is to provide API gateway the permission to invoke the lambda function. Add the following code in the **main.tf** file:

```

resource "aws_lambda_permission" "api_gw" {
    statement_id = "AllowExecutionFromAPIGateway"
    action      = "lambda:InvokeFunction"
}

```



```
function_name = "${aws_lambda_function.test_lambda.function_name}"
principal     = "apigateway.amazonaws.com"
source_arn    = "${aws_api_gateway_rest_api.testAPI.execution_arn}/*/*"

}
```

## Task 8: Create an Output file

In this task, you will create an `output.tf` file where you will add details of the provider and resources.

1. To create an **output.tf** file, expand the folder **task\_10106\_api**, and click on the **New File** icon to add the file.
2. Name the file as **output.tf** and press **Enter** to save it.
3. Paste the below content into the **output.tf** file. In the above code, we will extract the API Invoke URL.

```
output "api_invoke_url" {
  value =
"${aws_api_gateway_stage.teststage.invoke_url}/${aws_api_gateway_resource.testresource.pat
}
```



## Task 9: Confirm the installation of Terraform by checking the version

1. In the Visual Studio Code, open Terminal by selecting View from the Menu bar and choose Terminal.
2. If you are not in the newly created folder change your present working directory by running the below command.

```
cd task_10106_api
```



3. To confirm the installation of Terraform, run the below command to check the version:

```
terraform version
```



4. If you are getting output as command not found: terraform, this means that terraform is not installed on your system, To install terraform follow the official guide link provided in the Prerequisite section above.

## Task 10: Apply terraform configurations



## 1. Initialize Terraform by running the below command,

```
terraform init
```



```
PS C:\Users\██████\Desktop\task_10106_api> terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v4.48.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

**Note:** terraform init will check for all the plugin dependencies and download them if required, this will be used for creating a deployment plan

## 2. To generate the action plans run the below command

```
terraform plan
```



```
PS C:\Users\██████\Desktop\task_10106_api> terraform plan

Terraform used the selected providers to generate the following execution p
+ create

Terraform will perform the following actions:

# aws_api_gateway_deployment.testdep will be created
+ resource "aws_api_gateway_deployment" "testdep" {
  + created_date = (known after apply)
  + execution_arn = (known after apply)
  + id           = (known after apply)
  + invoke_url   = (known after apply)
  + rest_api_id  = (known after apply)
  + triggers     = (known after apply)
}

# aws_api_gateway_integration.MyDemoIntegration will be created
+ resource "aws_api_gateway_integration" "MyDemoIntegration" {
  + cache_namespace      = (known after apply)
  + connection_type      = "INTERNET"
  + http_method          = "GET"
  + id                   = (known after apply)
```

3. To create all the resources declared in the main.tf configuration file, run the below command.

```
terraform apply
```



4. Approve the creation of all the resources by entering **yes**.

```
Plan: 12 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + api_invoke_url = (known after apply)

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_api_gateway_rest_api.testAPI: Creating...
```

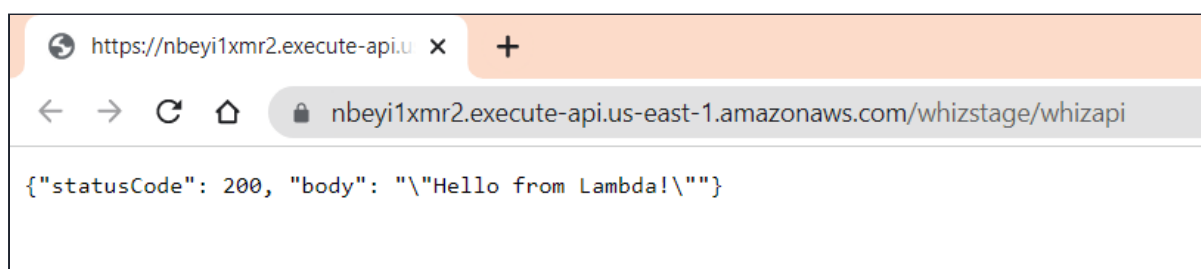
- **Note:** This process will take around 4-5 minutes.

5. Ids' of all the resources created by terraform will be visible there.

```
Outputs:

api_invoke_url = "https://nbeyi1xmr2.execute-api.us-east-1.amazonaws.com/whizstage/whizapi"
```

6. Copy the **api\_invoke\_url** and **paste** it in the browser to see if the lambda is successfully integrated into the Lambda function.



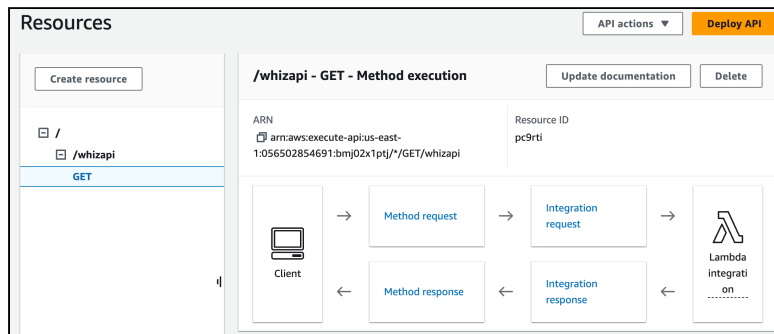
## Task 11: Check the resources in AWS Console

1. Make sure you are in the **US East (N. Virginia) us-east-1** Region.
2. Navigate to **API Gateway** by clicking on **Services** on the top, then click on **API Gateway** in the **Networking and Content Delivery** section.
3. Click on **APIs** in the left navigation menu.



APIs (1)						
<div> <input type="text" value="Find APIs"/> <div> <div>&lt;</div> <div>1</div> <div>&gt;</div> <div>⚙️</div> </div> </div>						
	Name ▲	Description ▼	ID ▼	Protocol ▼	Endpoint type ▼	Created ▼
<input type="radio"/>	WhizAPI	This is my API for demonstration purposes	nbeyi1xmr2	REST	Regional	2022-12-22

4. Select the **API** and click on **GET**. This is the execution for the API Gateway integrated with the lambda function.



## Do you know?

Building an API Gateway and integrating it with a Lambda function using Terraform involves creating and managing cloud resources programmatically. Terraform is an Infrastructure as Code (IaC) tool that allows developers and DevOps teams to define, provision, and manage cloud infrastructure in a declarative manner. This process automates the setup and configuration of the API Gateway, which acts as a front-end for your Lambda function, enabling external clients to invoke the function securely and efficiently.

### Task 12 : Validation of the Lab

1. Once the lab steps are completed, please click on the **Validation** button on the left side panel.
2. This will validate the resources in the AWS account and displays whether you have completed this lab successfully or not.
3. Sample output :



Validate My Lab

### Create an IAM role for Lambda

Check whether an IAM role for Lambda is created or not.

### Create an AWS Lambda Function

Check whether a Lambda Function is created with runtime as python or not

### Create a REST API Gateway

Check whether a REST API gateway is created or not

### Create Resource and Method for API Gateway

Check whether a resource and a GET method integrated with lambda function is created or not

### Deploy and Invoke REST API Gateway

Check whether a Rest API gateway is deployed and can invoke the URL or not

## Task 13: Delete AWS Resources

1. To delete the resources, open Terminal again.
2. Run the below command to delete all the resources. Enter **yes** to confirm the deletion.

```
terraform destroy
```



```

Enter a value: yes

aws_iam_role_policy_attachment.lambda_policy: Destroying... [id=iam_for_lambda-20221222142010061100000001]
aws_api_gateway_integration_response.MyDemoIntegrationResponse: Destroying... [id=agir-nbeyilxmr2-0ecknx-GET-200]
aws_lambda_permission.api_gw: Destroying... [id=AllowExecutionFromAPIGateway]
aws_api_gateway_stage.teststage: Destroying... [id=ags-nbeyilxmr2-whizstage]
aws_iam_role_policy_attachment.lambda_policy: Destruction complete after 1s
aws_api_gateway_integration_response.MyDemoIntegrationResponse: Destruction complete after 1s
aws_api_gateway_method_response.response_200: Destroying... [id=agmr-nbeyilxmr2-0ecknx-GET-200]
aws_api_gateway_stage.teststage: Destruction complete after 1s
aws_api_gateway_deployment.testdep: Destroying... [id=couqv1]

```

3. You can see the **Destroy complete!** message.

```

aws_lambda_function.test_lambda: Destruction complete after 0s
aws_iam_role.iam_for_lambda: Destroying... [id=iam_for_lambda]
aws_api_gateway_resource.testresource: Destruction complete after 1s
aws_api_gateway_rest_api.testAPI: Destroying... [id=nbeyilxmr2]
aws_api_gateway_rest_api.testAPI: Destruction complete after 0s
aws_iam_role.iam_for_lambda: Destruction complete after 1s

Destroy complete! Resources: 12 destroyed.

```

# Completion and Conclusion

- You have successfully set up the Visual Studio Code editor.
- You have successfully created variables.tf and terraform.tfvars files.
- You have successfully created an IAM role for the lambda function in the main.tf file.
- You have successfully created a lambda function in the main.tf file
- You have successfully created a rest API, its method, and its response in the main.tf file
- You have successfully created a gateway integration in the main.tf file.
- You have successfully created an output file.
- You have successfully checked all the resources created by opening the Console.
- You have deleted all the resources.

## End Lab

1. Sign out of AWS Account.
2. You have successfully completed the lab.
3. Once you have completed the steps, click on **End Lab** from your whizlabs dashboard

[About Us](#) [Subscription](#) [Instructions and Guidelines](#) [FAQ's](#) [Contact Us](#)



© 2023, Whizlabs Software Pvt. Ltd.

