QATIPv3 AWS Lab3
VPC deployment using Terraform

Contents

Lab Objectives

In this lab, you will:

- Deploy a VPC in the us-west-2 region
- Create a Public and Private subnet
- Create an Internet Gateway and a NAT Gateway
- Create Routing Tables and Security Groups
- Restrict inbound traffic to the Public Subnet, allowing any traffic from the Private Subnet but only SSH traffic from any other source
- Restrict inbound traffic to the Private subnet, allowing any traffic from the Public subnet only
- Allow unrestricted outbound traffic

Teaching Points

This lab will take you through coding a multi-component cloud deployment into AWS using Terraform. You will deploy a relatively simple VPC, but even this has

several components that need either to exist already or be created as part of the deployment. Elements required in this lab are the VPC itself, Subnets, Routing Tables, Gateways and Security Groups. This lab breaks the deployment into phases, each a separate task below, to demonstrate component interdependencies:

Task 1: VPC with 2 subnets
Task 2: Task 1 components plus an Internet Gateway and a NAT Gateway
Task 3: Task 2 components plus 2 Routing Tables and 2 Security Groups
Task 4: Task 3 components plus 2 EC2 instances for testing

Solution

The solution main.tf to this lab can be found in awslabs/solutions/03. Try to use this only as a last resort if you are struggling to complete the step-by-step processes.

Start Lab

1. Ensure you have completed Lab0 before attempting this lab.
2. In the IDE terminal pane, enter the following commands…

   cd ~/environment/awslabs/03

3. This shifts your current working directory to awslabs/labs/03. **Ensure all commands are executed in this directory**
4. Close any open files and use the Explorer pane to navigate to and open the empty **main.tf** file in awslabs/03.

Task 1. Create the basic VPC

Task 1 Objectives

The aim of this task is to:

1. Create a VPC resource **aws_vpc.test_vpc**, named **Test-VPC** in **us-west-2** using CIDR **10.1.0.0\16**
2. On this VPC, create a subnet resource **aws_subnet.public_subnet**, named **Public-Subnet** in **us-west-2a** using CIDR **10.1.1.0\24**

3. Create a second subnet resource **aws_subnet.private_subnet**, named **Private-Subnet** in **us-west-2a** using CIDR **10.1.2.0\24**
4. Destroy your deployment ahead of enhancing your code in Task 2.

Try it yourself

If you feel comfortable doing so, then attempt to complete this task without referencing the step-by-step instructions below. You can verify your attempt by comparing your code with the "Task1.tf" file in awslabs/solutions/03.

Note. Provider version 4.53.0 should be used throughout this lab.

References

https://registry.terraform.io/providers/hashicorp/aws/4.53.0
https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/vpc
https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/subnet

Step by Step

1. Review AWS Terraform documentation: https://registry.terraform.io/providers/hashicorp/aws/4.53.0
2. Click `**Use Provider**`
3. Copy the code block into main.tf using the IDE. For convenience, the code is listed below:

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "4.53.0"
    }
  }
}

provider "aws" {
  # Configuration options
}
```

4. Click on the "**Documentation**" link and from within the `**Example Usage**` section, find the code relating to `**Configure the AWS Provider**`. Note that

information regarding the AWS region is needed, `us-east-1` being used in the sample code.

5. We will be creating our resources in `**us-west-2**`. Copy the code and modify to include the `**us-west-2**` region. For convenience, the completed code for configuring the AWS provider is listed below:

```
provider "aws" {
 region = "us-west-2"
}
```

6. Review the documentation for creating a VPC ...

   https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/vpc

7. We want to identify our VPC by name, so we are interested in the '**Basic usage with tags'** example code. Scroll down the document to evaluate the `i**nstance_tenancy'** argument. We do not need our instances to have dedicated tenancy and therefore this argument is not needed.

8. Copy the code block into main.tf, change the resource block identifier from `**main**` to `**test_vpc**`, the cidr block from `**10.0.0.0/16**` to `**10.1.0.0/16**`, remove the `**instance_tenancy**` argument and change the value of the Name tag from `**main**` to `**Test-VPC**` For convenience, the modified code is listed below:

```
resource "aws_vpc" "test_vpc" {
 cidr_block = "10.1.0.0/16"
 tags = {
  Name = "Test-VPC"
 }
}
```

9. We now need to add 2 subnets to this VPC. Review the documentation for sample code on the aws_subnet resource, selecting `Version 4.53.0'

   https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/subnet

10. We need 2 subnets, so the simplest option is to create 2 copies of this resource block. What if we wanted many more though ? We will discuss more efficient options in a later lab. For now, though, copy the code block twice into main.tf.

11. Note that there is no availability zone specified in the example code. The default behaviour is therefore to randomly chose an availability zone within the region. We want to mandate a specific availability zone, `us-west-2a` so we will need to add this argument.

12. Edit the first block: change the resource block identifier from `main` to `public_subnet`, the vcp_id argument value from `aws_vpc.main.id` to `aws_vpc.test_vpc.id`, the cidr block from `10.0.1.0/24` to `10.1.1.0/24`. Add an `availability_zone` argument with the value `us-west-2a`. Finally, change the Name tag from `Main` to `Public-Subnet`

13. Edit the second block: change the resource block identifier from `main` to `private_subnet`, the vcp_id argument value from `aws_vpc.main.id` to `aws_vpc.test_vpc.id`, the cidr block from `10.0.1.0/24` to `10.1.2.0/24` Add an `availability_zone` argument with the value `us-west-2a`. Finally, change the Name tag from `Main` to `Private-Subnet` The modified blocks should now be as follows…

```
resource "aws_subnet" "public_subnet" {
 vpc_id     = aws_vpc.test_vpc.id
 cidr_block = "10.1.1.0/24"
 availability_zone = "us-west-2a"
 tags = {
  Name = "Public-Subnet"
 }
}

resource "aws_subnet" "private_subnet" {
 vpc_id     = aws_vpc.test_vpc.id
 cidr_block = "10.1.2.0/24"
 availability_zone = "us-west-2a"
 tags = {
  Name = "Private-Subnet"
 }
```

`}`

14. Save your changes and run `**terraform init**`
15. Run `**terraform plan**` Note any errors and fix if appropriate. Refer to `Task 1 Solution Code` in the Solution section if necessary
16. Run `**terraform apply**`, entering `**yes**` when prompted. 3 resources should be added.
17. Switch to the Console
18. Ensure you have the Oregon region selected. Use the drop-down list at top right of screen to change current region if not.
19. Use the Search bar to search for `**VPC**`
20. In the `**Resources by Regions**` area you should see `2` next to `US West`, indicating there are 2 VPCs in this region, the default VPC created in every region and the one you have just deployed.
21. Click on `VPCs` to navigate to the `**Your VPCs**` view for the Oregon region.
22. Verify you see the VPC we have just deployed, `**Test-VPC**`
23. Select the `**Subnets**` option on the left menu and verify you see the 2 subnets we have just deployed.

Task 2. Provision an Internet Gateway and a NAT Gateway

Important – If you attempted Task 1 yourself

If you attempted the previous task yourself, then your code, whilst hopefully achieving the objectives specified, may vary slightly from the solution provided for this lab. You are encouraged to continue attempting each task without precise guidance if you feel comfortable doing so. An alternative approach is to 'reset' your code at the start of each task to align it with the solution code prior to moving forward.

To do this now:

1. Destroy any resources you created in Task1

2. Clear the contents of your current main.tf file
3. Navigate to awslabs\solutions\03 and copy the contents of Task1.tf into your empty main.tf
4. Save main.tf
5. Run **Terraform apply** followed by **yes**

Task 2 Objectives

The aim of this task is to enhance your new VPC as follows:

1. Create an Internet Gateway resource `**aws_internet_gateway.lab_igw**` named `**Lab-IGW**`
2. Create an Elastic IP resource `**aws_eip.static_ip**` named `**NAT-GW-Static-IP**`
3. Create a Public NAT Gateway resource `**aws_nat_gateway.lab_nat_gw**` on your public subnet. Name it `**Lab-NAT-GW**` and associate `**aws_eip.static_ip**` with it

Try it yourself

If you feel comfortable doing so, then attempt to complete this task without referencing the step-by-step instructions below. You can verify your attempt by comparing your code with the "Task2.tf" file in awslabs/solutions/03.

Note. Provider version 4.53.0 should be used throughout this lab.

References

https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/internet_gateway
https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/eip
https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/nat_gateway

Step by Step

1. Review the Terraform Registry documentation regarding the creation of an AWS Internet Gateway resource `aws_internet_gateway`

   https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/internet_gateway

2. Copy the `**Example Usage**` code block into main.tf
3. Rename the resource identifier to `**lab_igw**`
4. Rename the vpc_id argument value to reference your VPC `**aws.test_vpc.id**`

5. Rename the resource to `**Lab-IGW**`

   The modified block should now be as shown below

```
resource "aws_internet_gateway" "lab_igw" {
 vpc_id = aws_vpc.test_vpc.id

 tags = {
  Name = "Lab-IGW"
 }
}
```

6. Review the Terraform Registry documentation regarding the creation of an AWS Elastic IP resource `aws_eip`

   https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/eip

7. Copy the `**Example Usage - Single EIP associated with an instance**` code block into main.tf

8. Change the resource block identifier from "lb" to **"static_ip"**

9. Remove the **'instance = aws_instance.web.id'** argument

10. Add a tags block and Name the resource `**NAT GW Static IP**` The modified block should now be as shown below

```
resource "aws_eip" "static_ip" {
 vpc      = true
 tags = {
  Name = "NAT GW Static IP"
 }
}
```

11. Review the Terraform Registry documentation regarding the creation of an AWS NAT Gateway resource `aws_nat_gateway`
    https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/nat_gateway

12. Copy the `**Example Usage - Public NAT**` code block into main.tf

13. Change the resource block identifier to `**lab_nat_gw**`
14. Change the `**allocation_id**` argument value to `**aws_eip.static_ip.id**` This associates the EIP `**NAT GW Static IP**` with this NAT Gateway
15. Change the `**subnet_id**` argument value to `**aws_subnet.public_subnet.id**`. This places the NAT Gateway onto the `**Public-Subnet**`
16. Change the resource name to `**Lab-NAT-GW**`
17. Remove the comments
18. Change the `depends_on` argument value to `**[aws_internet_gateway.lab_igw, aws_eip.static_ip]**` This is to ensure that both the EIP and the IGW are created before attempting to create the NAT GW. The modified block should now be as shown below

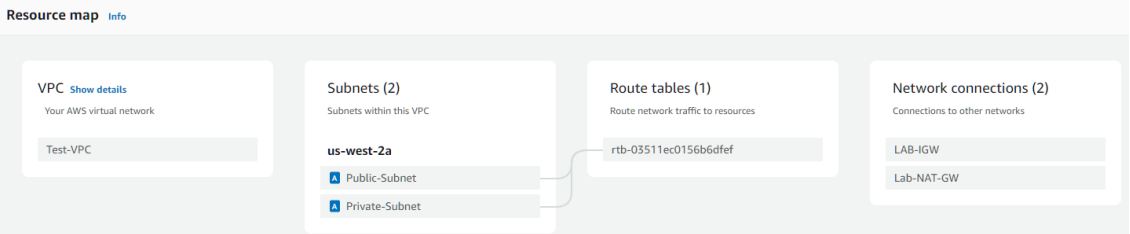
```
resource "aws_nat_gateway" "lab_nat_gw" {
 allocation_id = aws_eip.static_ip.id
 subnet_id     = aws_subnet.public_subnet.id

 tags = {
  Name = "Lab-NAT-GW"
 }

 depends_on = [aws_internet_gateway.lab_igw, aws_eip.static_ip]
}
```

19. Save your changes and run `**terraform plan**` Note any errors and fix if appropriate. Refer to `Task 2 Solution Code` if necessary. This shows the entire main.tf as it should be to date.
20. Finally, run `**terraform apply**`, entering `**yes**` when prompted. 6 resources are created.
21. Switch to the AWS Console
22. Navigate to the VPCs console for Oregon and verify the existence of the new Internet Gateway, Elastic IP, and NAT Gateway along with the 3 Task 1 resources.
23. We now have a VPC with 2 subnets, an Internet Gateway, and a NAT Gateway with a static public IP address. What we do not yet have is routing between the subnets and the gateways. Every VPC has a default `**main**` routing table

automatically created and all VPC subnets are associated with this main routing table unless specifically associated with an alternative routing table. The VPC `**Resource map**` allows you to visualize this



24. As well as network connectivity, we must also consider firewall rules, needed to permit inbound and outbound traffic flows. Every VPC has a default Security Group containing these rules. The default rules on this Security Group are to allow unrestricted outbound access to any destination IP address but to only allow inbound traffic from the IP addresses of EC2 instances that are using this Security Group.

25. In Task 3 you will enhance your code to include dedicated routing tables and security Groups for `Test-VPC`

Task 3. Provision Routing tables and Security Groups

Important – If you attempted Task 2 yourself

If you attempted a previous task yourself, then your code, whilst hopefully achieving the objectives specified, may vary slightly from the solution provided for lab. You are encouraged to continue attempting each task without precise guidance if you feel comfortable doing so. An alternative approach is to 'reset' your code at the start of each task to align it with the solution code, prior to moving forward.

To do this now:

1. Destroyed any resources you have created
2. Clear the contents of your current main.tf file
3. Navigate to awslabs\solutions\03 and copy the contents of Task2.tf into your empty main.tf
4. Save main.tf
5. Run **Terraform apply** followed by **yes**

Task 3 Objectives

The aim of this task is to enhance your new VPC as follows:

1. Create a routing table resource `**aws_route_table.public_route_table**` named `**Public-Route-Table**`
2. Create a routing table resource `**aws_route_table.private_route_table**` named `**Private-Route-Table**`
3. Set the default route on `**Public-Route-Table**` as your internet gateway `**aws_internet_gateway.lab_igw**`
4. Set the default route on `**Private-Route-Table**` as your NAT gateway `**aws_nat_gateway.lab_nat_gw**`
5. Create a Route Table Association resource `**aws_route_table_association.pub_to_ig**` to associate `**PublicSubnet**` with `**Public-Route-Table**`
6. Create a Route Table Association resource `**aws_route_table_association.priv_to_nat**` to associate `**PrivateSubnet**` with `**Private-Route-Table**`
7. Create a Security group resource `**aws_security_group.internal**` named `**Internal SG**`
8. Create a Security group resource `**aws_security_group.external**` named `**External SG**`
9. Configure `**Internal SG**` with rules that allow unrestricted inbound traffic from `**PublicSubnet**` and unrestricted outbound traffic to all destinations. **Note.** In production we would only open selective ports between the public and private subnets
10. Configure `**External SG**` with rules that allow TCP port 22 inbound traffic from any source IP, unrestricted inbound traffic from `**PrivateSubnet**`, and unrestricted outbound traffic to all destinations. **Note.** In production we would be more selective regarding ports, as appropriate to the solution being deployed onto the EC2 instances on `PublicSubnet`

Try it yourself

If you feel comfortable doing so, then attempt to complete this task without referencing the step-by-step instructions below. You can verify your attempt by comparing your code with the "Task3.tf" file in awslabs/solutions/03.

Note. Provider version 4.53.0 should be used throughout this lab.

References

https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/route_table
https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/route_table_association
https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/security_group

Step by Step

1. Review the Terraform Registry documentation regarding the creation of an AWS Route Table `aws_route_table`
   https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/route_table

2. Copy the `Example Usage` code block into main.tf

3. Change the resource block identifier from 'example' to `**public_route_table**`

4. Change the vpc_id argument value from 'aws_vpc.example.id' to reference your VPC **aws_vpc.test_vpc.id**

5. Change the cidr_block to make this the default route `**0.0.0.0/0**`

6. Change the gateway_id to reference your Internet Gateway **aws_internet_gateway.lab_igw.id**

7. Remove the route block relating to IPv6

8. Rename the resource to `**Public-Route-Table**`

9. Add depends_on argument with a value of `**[aws_internet_gateway.lab_igw]**`

The modified block should now be as shown below

```
resource "aws_route_table" "public_route_table" {
 vpc_id = aws_vpc.test_vpc.id

 route {
  cidr_block = "0.0.0.0/0"
  gateway_id = aws_internet_gateway.lab_igw.id
 }
 tags = {
  Name = "Public-Route-Table"
 }
 depends_on = [aws_internet_gateway.lab_igw]
}
```

10. Copy this entire aws_route_table resource block and paste a copy into the end of main.tf

11. Rename the new resource block to `private_route_table`

12. Change the gateway_id to aws_nat_gateway.lab_nat_gw.id

13. Rename the resource to `Private-Route-Table`

14. Change the depends_on argument value to `[aws_nat_gateway.lab_nat_gw]`

The modified block should now be as shown below

```
resource "aws_route_table" "private_route_table" {
 vpc_id = aws_vpc.test_vpc.id

 route {
  cidr_block = "0.0.0.0/0"
  gateway_id = aws_nat_gateway.lab_nat_gw.id
 }
 tags = {
  Name = "Private-Route-Table"
```

```
  }
  depends_on = [aws_nat_gateway.lab_nat_gw]
 }
```

15. Review the Terraform Registry documentation regarding the creation of an AWS Route Table Association `**aws_route_table_association**`

    https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/route_table_association

16. Copy the `Example Usage` code block that relates to an association between a route table and a subnet into main.tf

17. Change the resource block identifier from a to `**pub_to_ig**`

18. Associate the subnet_id argument with your Public subnet **aws_subnet.public_subnet.id**

19. Associate route_table_id to your Public routing table **aws_route_table.public_route_table.id**

20. Add depends_on argument with a value of `**[aws_internet_gateway.lab_igw]**` The modified block should now be as shown below..

```
resource "aws_route_table_association" "pub_to_ig" {
 subnet_id      = aws_subnet.public_subnet.id
 route_table_id = aws_route_table.public_route_table.id
 depends_on = [aws_internet_gateway.lab_igw]
}
```

21. Copy the entire **aws_route_table_association** resource block and paste into the end of main.tf

22. Rename the new resource block to `**priv_to_nat**`

23. Associate the subnet_id argument with your Private subnet `**aws_subnet.private_subnet.id**`

24. Associate route_table_id to your Private routing table
    aws_route_table.private_route_table.id

25. Change the depends_on argument to `**[aws_nat_gateway.lab_nat_gw]**' The modified block should now be as shown below

```
resource "aws_route_table_association" "priv_to_nat" {
 subnet_id     = aws_subnet.private_subnet.id
 route_table_id = aws_route_table.private_route_table.id
 depends_on = [aws_nat_gateway.lab_nat_gw]
}
```

26. Review the Terraform Registry documentation regarding the creation of an AWS Security Group resources `**aws_security_group**`

    https://registry.terraform.io/providers/hashicorp/aws/4.53.0/docs/resources/security_group

27. Copy the `Example Usage` Basic Usage code block into main.tf

28. Change the resource block identifier from 'allow tls' to `**internal_sg**`

29. Change the name to `**Internal-SG**`

30. Change the description to `**SG for internal instances**`

31. Change the vpc_id to reference your VPC **aws_vpc.test_vpc.id**

32. Change the ingress rule description to **"All"**

33. Change both the from_port and the to_port to `**0**`

34. Change the protocol to `**"-1"**`

35. Change the cidr_blocks to reference your Public subnet range `**["10.1.1.0/24"]**`

36. Remove the ipv6_cidr_blocks entry

37. Remove the ipv6_cidr_blocks entry from the egress rule, leaving the remainder unchanged

38. Change the Name tag to `"Internal-SG"` The modified block should now be as shown below

```
resource "aws_security_group" "internal_sg" {
 name        = "Internal-SG"
 description = "SG for internal instances"
 vpc_id      = aws_vpc.test_vpc.id

 ingress {
  description      = "ALL"
  from_port        = 0
  to_port          = 0
  protocol         = "-1"
  cidr_blocks      = ["10.1.1.0/24"]
 }

 egress {
  from_port        = 0
  to_port          = 0
  protocol         = "-1"
  cidr_blocks      = ["0.0.0.0/0"]
 }
 tags = {
  Name = "Internal-SG"
 }
}
```

39. Copy the entire Internal aws_security_group resource block that you have just created and paste a copy into the end of main.tf

40. Change the new resource block identifier to `external_sg`

41. Change the name to `External-SG`

42. Change the description to `SG for front facing instances`

43. Change ingress cidr_blocks reference your Private subnet range `10.1.2.0/24`

44. Duplicate the ingress block and paste a copy before the egress block

45. Change the new ingress rule description to `"SSH"`

46. Change both the from_port and the to_port to `22`

47. Change the protocol to `tcp`

48. Change cidr_blocks to apply to all incoming traffic `0.0.0.0./0`

49. Leave the egress rule unchanged

50. Change the Name tag to `"External SG"` The modified block should now be as shown below…

```
resource "aws_security_group" "external_sg" {
 name        = "External-SG"
 description = "SG for front facing instances"
 vpc_id      = aws_vpc.test_vpc.id

 ingress {
  description      = "ALL"
  from_port        = 0
  to_port          = 0
  protocol         = "-1"
  cidr_blocks      = ["10.1.2.0/24"]
 }

 ingress {
  description      = "SSH"
  from_port        = 22
  to_port          = 22
  protocol         = "tcp"
  cidr_blocks      = ["0.0.0.0/0"]
 }
```

```
  egress {
    from_port       = 0
    to_port         = 0
    protocol        = "-1"
    cidr_blocks     = ["0.0.0.0/0"]
  }

  tags = {
    Name = "External-SG"
  }
}
```

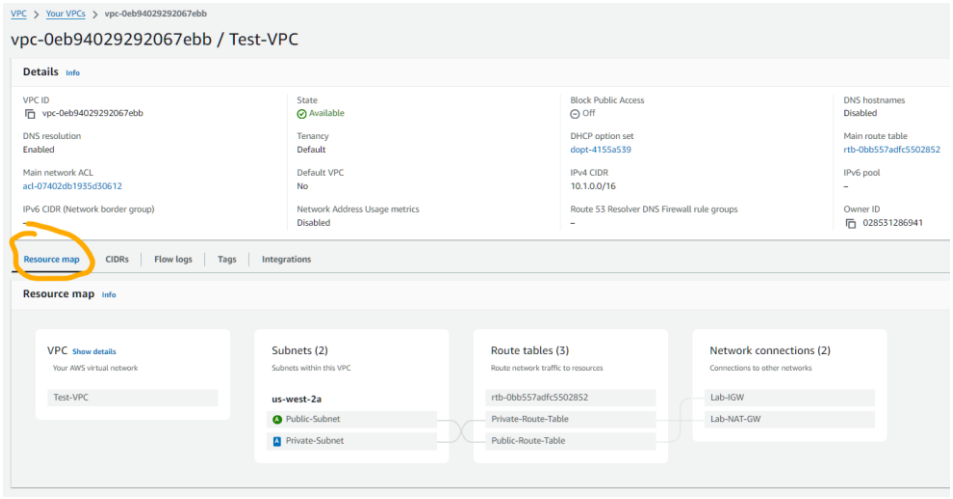
51. Save main.tf and run `terraform plan` Note any errors and fix if appropriate. Refer to `Task 3 Solution Code` in the `Solution` section if necessary. This shows the entire main.tf as it should be to date.

52. Run `terraform apply`, entering `yes` when prompted. 6 resources are created.

53. Switch to the AWS Console

54. Navigate to the VPCs console for Oregon and verify the existence of the added resources.

55. Use the VPC `Resource map` to verify we now have the desired networking configuration (you may need to refresh the page)..



PublicSubnet <---> Public-Route-Table <---> Lab-IGW

PrivateSubnet <---> Private-Route-Table <---> Lab-NAT-GW

Note: You can hover over each subnet to highlight its current network connectivity.

56. In Task 4 you will create EC2 instances on each subnet to test the VPC network connectivity and Security Group rules

Task 4. Create EC2 Instances

Important – If you attempted Task 3 yourself

If you attempted a previous task yourself, then your code, whilst hopefully achieving the objectives specified, may vary slightly from the solution provided for this lab. You are encouraged to continue attempting each task without precise guidance if you feel comfortable doing so. An alternative approach is to 'reset' your code at the start of each task to align it with the solution code, prior to moving forward.

To do this now:

1. Ensure you have destroyed any resources
2. Clear the contents of your current main.tf file
3. Navigate to awslabs\solutions\03 and copy the contents of Task3.tf into your empty main.tf
4. Save main.tf
5. Run **terraform apply** followed by **yes**

Step-by-step

1. Using the boilerplate code below, attempt this task without step-by-step guidance. Refer to the solution main.tf in awslabs/solutions/03 only if needed.

2. During this task you are required to:

3. Use the **AWS EC2 Dashboard** to create a regional key-pair called `**Oregon_lab_keypair**` This key-pair is to be used to access all regional EC2

instances. Download the key-pair in .pem format and retain for lab testing later.

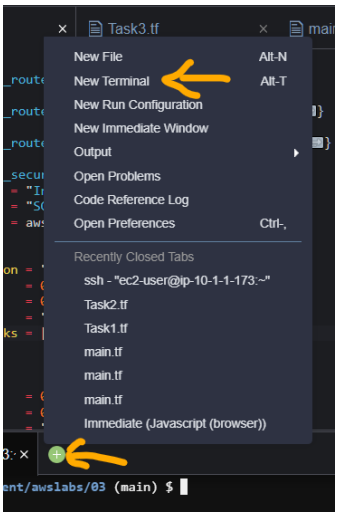
4. Create 2 EC2 Instances of size `t3.micro` using `ami-06e85d4c3149db26a`
5. Both instances should be in availability zone `us-west-2a`
6. Name the first instance `PubVM`
7. Place `PubVM` on subnet `PublicSubnet`
8. Add `PubVM` to the security group `External SG`
9. Allocate `PubVM` a dynamic public IP address
10. Name the second instance `PrivVM`
11. Place `PrivVM` on subnet `PrivateSubnet`
12. Add `PrivVM` to the security group `Internal SG`
13. Do not allocate a public IP address to `PrivVM`
14. **Save** and **apply** the now complete main.tf
15. Do not destroy the deployment as we will move onto testing next.

```
resource "aws_instance" "" {
ami           = ""
instance_type = ""
key_name = "Oregon_lab_keypair"
availability_zone = ""
subnet_id =
associate_public_ip_address =
vpc_security_group_ids = ["${}"]

tags = {
Name = ""
}
depends_on = []
}
```

Task 5. Testing

1. Using the EC2 Dashboard, note the external IP address of `PubVM` and private IP address of `PrivVM`
2. In the IDE, use the `File/Upload Local Files` menu to upload `**Oregon_lab_keypair.pem**` from your local machine to awslabs/03 folder
3. Open a second terminal session on your IDE…



4. In the new terminal session, switch to the lab3 working directory..
   cd ~/environment/awslabs/03
5. Change permissions on the uploaded pem file

   chmod 400 Oregon_lab_keypair.pem
6. Connect to PubVM using:

   ssh -i Oregon_lab_keypair.pem ec2-user@{public IP of PubVM}

   Example: ssh -i Oregon_lab_keypair.pem ec2-user@1.2.3.4
7. Enter **yes** at connection security prompt.
8. Ping PrivVM from within the PubVM ssh session

   ping -c 3 {private IP of PrivVM}
9. Pinging PrivVM from PubVM should succeed because there is a route from the public subnet to the private subnet and the private security group ingress policy allows the traffic.

10. Modify main.tf with an erroneous ingress entry within the internal security group resource block, changing the cidr_blocks entry from `10.1.1.0/24` to `**10.1.3.0/24**`
11. Save main.tf then switch to your initial terminal session to apply the change with **terraform apply** followed by **yes**
12. Switch back to the ssh session and repeat the ping command.
13. This ping attempt should fail
14. Undo the change made to main.tf, resetting cidr_blocks back to `**10.1.1.0/24**`
15. Save main.tf and apply the change
16. Repeat the ping attempt
17. The pings should succeed.
18. Type **exit** to close your SSH session to PubVM. Close the ssh terminal, acknowledging the message.

Task 6. Lab Clean-Up

1. Run **terraform destroy** followed by **yes** to remove all lab resources

***** Congratulations, you have completed this lab *****