# Conclusion

Besides the creation of key pairs (AWS, n.d.), every resource created in this assignment is through code/.yml files. The usage of CloudFormation designer and the submission of .yml files through CloudFormation UI was done initially to get an understanding of the concept of CloudFormation and how to setup one. Writing code for setting up a CloudFormation seems to be the ideal approach opposed to using the CloudFormation UI’s create stack option to get the benefits of reusability and testability. A high-level diagram of the network created in this assignment is in Figure 1. The code written for this assignment can be reused to create any number of networks that can be configured differently with respect to the IP range, availability zones or ami images of instances since these are all configurable through parameters. Tables 1 to 8 in appendix show the parameter combination of each type differing by each instance. Moreover, various features offered by the AWS is used to get an understanding of the capabilities of CloudFormation. Few of the features learned and applied in this assignment are:

* Fn::ImportValue: - This is an intrinsic function in aws to import a value into a stack exported by a different stack.
* Conditions: Conditions are boolean expressions that can be derived on some logic and be applied to a resource to determine whether it needs to be created or not.
* Mappings: Mappings define a set of {key – value} mappings which can be used to arrive at a value based on a key.

These features are helpful when we are developing a bigger complex structure of resources that will help us not only reduce the number of lines, also more readable and neat looking code.

One of the key considerations taken while developing the solution was breaking up of code into individual .yml files rather than having a single lengthy file. This proves to be easier while troubleshooting giving more flexibility to execute a single stack in isolation to repeatedly test it without having to come over from the very beginning. The grouping of resources into different stacks are planned as depending on the relations between them. For example, the VPC and the Internet Gateway are kept in a single stack. This is done so under the notion that there will be only a single VPC in a network and will be associated with an Internet Gateway. The dependency between resources across stacks can still be managed through the use of Import and Export capability offered by AWS. Another benefit of splitting a complex solution into multiple files is the degree of reusability. Each stack written is configurable through parameters which changes the behaviour of the template using the features such as the ones listed above so that the same templates can be used over again for different instances of the same type of resource. The Subnet.yml for instance is called five times with varying set of parameters. Creating public routes with InternetGateway in the case of public subnet demoSubnetL1 for associating instance J and creating a route with NATGateway in the case of private subnet demoSubnetR1 for instance A is an ideal example of this.

Another consideration was the security of the resources within the network created. Only instances in the public subnet demoSubnetL1 are publicly accessible. Instance J is the only public instance that is available to the public and can be accessed from outside. While instances like F and A that are in private subnets have their SecurityGroups configured with rules to allow access from outside only through instance J. The DB instance D, is made available to J through A, further distancing away from public access. This way we are able to keep resources secure while maintaining the desired level of communication possible.

The flexibility offered by AWS in order to achieve a solution is impressive. The NATGateway implementation to make internet available to a private instance is an example. Rather than assigning it an InternetGateway, we can restrict the inbound traffic while making internet available to the instances inside a private subnet. We also have an enormous collection of ami images available to spin up an instance. There are also images provided by third parties. To get the nginx web server onto an instance, an image made available in the AWS Marketplace by bitnami (Anon., n.d.) is used which is built on top of Debian operating system. There is also an option for individual contributions to the AWS community like the third-party contributions into which any private individual can create and make it available to the public. The DBInstance also supports a variety of relational databases. The database engine used in this assignment is Postgres (The PostgreSQL Global Development Group, n.d.); there are others such as SQL Server, MySQL and MariaDB.

Even though creating a resource directly through the AWS user interface may seem simple and straightforward, when it comes to a group of resources that need to configured to interact each other with some level of security implemented we will end up in a number of tasks switching between views and repeating a number of steps against different resources. All the private subnets in this example needed the same configuration in this assignment except the additional NATGateway route for demoSubnetR2. These kinds of repetitions can easily be avoided when we have CloudFormation templates instead which enables reusability of templates. The requirement to recreate a similar structure or duplicating the same structure is again a monotonous human activity which could also lead to inconsistencies if we do not perform the exact steps in the same order. Meanwhile, having a set of CloudFormation templates that is proven and stabilised in time after each trial and tests will be more consistent. The further usage of Nested stacks (AWS, n.d.) to combine multiple stacks in this assignment makes the usage easier having only a single step to create a network.

From the GRC (Governance, Risk and Compliance) point of view, using a template for a specific resource enables to enforce several standards such as naming of resources, restricting parameter values to ranges or an available number of types. This also reduces the risks in creating resources with parameters which may not be acceptable as per the organisational policies. The use of ingress/egress rules to restrict access of resources only from intended sources respecting the protocol reduces the risk of any unauthorised access to resources that could end up in any undesirable incidents. Moreover, usage of templates holds evidence that standards are being followed and can be verified and reviewed by the auditors and be modified to make it compliant to future changes.

From the experience doing this assignment, I feel CloudFormation is a great way to automate setting up a cluster of resources that are meant to work together. Although there are other alternatives to create resources directly through the user interface of AWS quickly, the benefits we get by writing a CloudFormation is that we arrive at an optimal solution refined over time that can be reused any number of times across teams or organisations.

# References

Anon., n.d. *NGINX Open Source packaged by Bitnami.* [Online]   
Available at: https://aws.amazon.com/marketplace/pp/prodview-lzep7hqg45g7k?sr=0-1&ref\_=beagle&applicationId=AWS-EC2-Console  
[Accessed 10 11 2022].

AWS, n.d. *Create key pairs.* [Online]   
Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/create-key-pairs.html  
[Accessed 10 11 2022].

AWS, n.d. *create-stack.* [Online]   
Available at: aws cloudformation create-stack   
[Accessed 10 11 2022].

AWS, n.d. *Working with nested stacks.* [Online]   
Available at: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-nested-stacks.html  
[Accessed 10 11 2022].

The PostgreSQL Global Development Group, n.d. [Online]   
Available at: https://www.postgresql.org/download/linux/debian/  
[Accessed 10 11 2022].

# Appendix

|  |  |
| --- | --- |
| **Resource** | **Description** |
| VPC | VPC resource |
| demoGateway1 | Internet gateway |
| NATGatewayR | NAT gateway; placed in demoSubnetL1 |
| ElasticIP | Elastic IP associated to the NAT gateway |
| demoSubnetL1 | public subnet for instance J |
| demoSubnetL2 | private subnet for instance F |
| demoSubnetR1 | private subnet for instance A |
| demoSubnetR2a | first private subnet for DB instance D |
| demoSubnetR2b | second private subnet for DB instance D |
| demoDBSubnetGroup | Subnet group for DB instance |
| demoSecurityGroup1 | security group for instance J |
| demoSecurityGroup2 | security group for instance F; allow only comms from J |
| demoSecurityGroupR1 | security group for instance A; allow only comms from F |
| demoSecurityGroupD | security group for DB instance D; allow only comms from A |
| J | Jumpbox; public facing ubuntu OS |
| F | A web server with nginx running in Debian; private instance |
| A | A web server with nginx running in Debian; private instance |
| D | DBInstance with PostgreSql |

Table 1. List of resource created

|  |  |  |
| --- | --- | --- |
| **Parameters** | **Description** | **VPC** |
| VPCId | Name of VPC Id Output. This variable is also exported in the output containing the ID value generated so that it can be used in the subsequent stacks like each of Subnets that requires the VPCId | demoVPC1ID |
| GatewayId | Name of Gateway Id Output. This variable is also exported to output so that it can be used in subsequent stacks | demoGatewayID |

Table 2. Parametes in VPC.yml

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameters** | **Description** | **Default** | **demoSubnetL1** | **demoSubnetL2** |
| VPCId | Name of variable for VPC Id from Output; This should match the same value from the VPC stack |  | demoVPC1ID | demoVPC1ID |
| GatewayId | Name of variable for Gateway Id; Could also be the NAT Gateway in case of private subnet |  | demoGatewayID | demoGatewayID |
| SubnetId | Name of the variable for Subnet Id to Output. Exported to output |  | demoSubnetL1 | demoSubnetL2 |
| RouteTableId | Name of the variable for RouteTable Id to Output. Exported to output |  | demoRouteTableL1 | demoRouteTableL2 |
| CidrBlock (Default: 10.0.0.0/24) | IP range for subnet | 10.0.0.0/24 | 10.0.0.0/24 | 10.0.1.0/24 |
| IsPublic | Set whether the subnet is public | true | true | false |
| AvailabilityZone | Availability zone of subnet | us-east-1a |  |  |
| HasNATGateway | Whether we need (NAT) Gateway | false |  |  |

Table 3. Parametes in Subnet.yml

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameters** | **Description** | **demoSubnetR1** | **demoSubnetR2a** | **demoSubnetR2b** |
| VPCId | Name of variable for VPC Id from Output; This should match the same value from the VPC stack | demoVPC1ID | demoVPC1ID | demoVPC1ID |
| GatewayId | Name of variable for Gateway Id; Could also be the NAT Gateway in case of private subnet | NATGatewayR | demoGatewayID | demoGatewayID |
| SubnetId | Name of the variable for Subnet Id to Output. Exported to output | demoSubnetR1 | demoSubnetR2a | demoSubnetR2b |
| RouteTableId | Name of the variable for RouteTable Id to Output. Exported to output | demoRouteTableR1 | demoRouteTableR2a | demoRouteTableR2b |
| CidrBlock (Default: 10.0.0.0/24) | IP range for subnet | 10.0.2.0/24 | 10.0.3.0/24 | 10.0.4.0/24 |
| IsPublic (Default: true) | Set whether the subnet is public | false | false | false |
| AvailabilityZone | Availability zone of subnet |  |  | us-east-1b |
| HasNATGateway  (default: false) | Whether we need (NAT) Gateway | true |  |  |

Table 3a. Parameters in Subnet.yml for remaining subnets continued

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameters** | **Description** |  | **NATGatewayR** |
| PublicSubnetId | Name of Subnet Id variable from Output created by subnet stack | | demoSubnetL1 |
| NATGatewayId | Name of the NAT GatewayId to Output meant to be used by subsequent private subnet stack | | NATGatewayR |

Table 4. Parameters in NATGateway.yml

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parameters** | **Description** | **Default** | **J** | **F** | **A** |
| SecurityGroup  EgressIP1 | First Egress rule's IP address range | 0.0.0.0/0 | 22 | 0.0.0.0/0 | 10.0.3.0/24 |
| SecurityGroup  EgressPort1 | First Egress rule's Port | 0 | 10.0.1.0/24 | -1 | 5432 |
| SecurityGroup  EgressProtocol2 | Second Egress rule's IP protocol (icmp/tcp) | icmp | tcp | tcp | tcp |
| SecurityGroup  EgressIP2 | Second Egress rule's IP address range | 0.0.0.0/0 | 0.0.0.0/0 | 10.0.0.0/24 | 0.0.0.0/0 |
| SecurityGroup  EgressPort2 | Second Egress rule's Port | 0 | 80 | 80 | 80 |
| SecurityGroup  EgressProtocol3 | Third Egress rule's IP protocol (icmp/tcp) | icmp | tcp |  | tcp |
| SecurityGroup  EgressIP3 | Third Egress rule's IP address range | 0.0.0.0/0 | 10.0.2.0/24 |  | 10.0.4.0/24 |
| SecurityGroup  EgressPort3 | Third Egress rule's Port | 0 | 22 |  | 5432 |
| SecurityGroup  IngressProtocol1 | First Ingress rule's IP protocol (icmp/tcp) |  | tcp | tcp | tcp |
| SecurityGroup  IngressIP1 | First Ingress rule's IP range | 0.0.0.0/0 | 0.0.0.0/0 | 10.0.0.0/24 | 10.0.0.0/24 |
| SecurityGroup  IngressPort1 | First Ingress rule's port |  | 22 | 22 | 22 |
| SecurityGroup  IngressProtocol2 | First Ingress rule's IP protocol (icmp/tcp) |  | icmp | tcp | tcp |
| SecurityGroup  IngressIP2 | First Ingress rule's IP range | 0.0.0.0/0 | 0.0.0.0/0 | 10.0.0.0/24 | 10.0.0.0/24 |
| SecurityGroup  IngressPort2 | First Ingress rule's port |  | -1 | 80 | 80 |

Table 5. Parameters in Instance.yml

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameters** | **Description** |  | **D** |
| VPCId | Name of VPC Id from Output | | demoVPC1ID |
| InstanceName | Name of the db instance | | demoSecurityGroupD |
| DBInstanceIdentifier | db instance identifier | | demoDB |
| VPCSecurityGroupsName | Name of the variable for security group | | demoSecurityGroupD |
| DBUsername | username of the db | | {{resolve:ssm-secure:MasterUsername}} |
| DBPassword | Password of the db | | {{resolve:ssm-secure:MasterPassword}} |
| DBSubnetGroupName | Subnet group's name | | demoDBSubnetGroup |
| DBSubnetGroupDescription | Subnet group's description | | DB Subnet for RDS |
| Subnet1Id | First subnet Id |  | demoSubnetR2a |
| Subnet2Id | Second subnet Id | | demoSubnetR2b |

Table 6. Parameters in DBInstance.yml

|  |  |  |
| --- | --- | --- |
| **Conditions** | **demoSubnetL1** | **demoSubnetR2b** |
| CreatePublicRoute | true | false |
| CreateNATRoute | false | false |

Table 7. Conditions in Subnet.yml

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Conditions** |  | **J** | **F** | **A** |
| HasDefaultEgress2 |  | false | true | false |
| CreateEgress2 |  | true | false | true |
| HasDefaultEgress3 | | false | true | false |
| CreateEgress3 |  | true | false | true |

Table 8. Conditions in Instance.yml

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Protocol** | **Source** | **J** | **F** | **A** | **D** |
| SSH | Public/outside | Yes | No | No | No |
| HTTP | J | - | Yes | Yes | No |
| SSH | F | No | - | No | No |
| HTTP | A | No | No | - | Yes |

Table 9. Communication possible between instances while testing

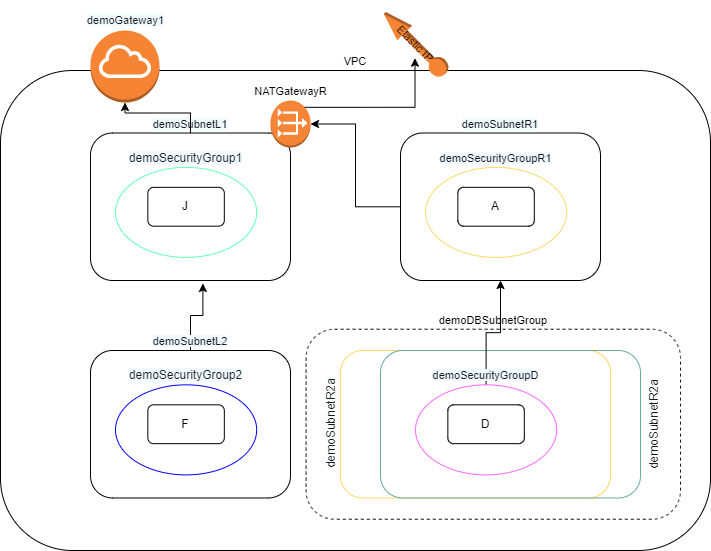


Figure 1. Implementation of the network at a high level.

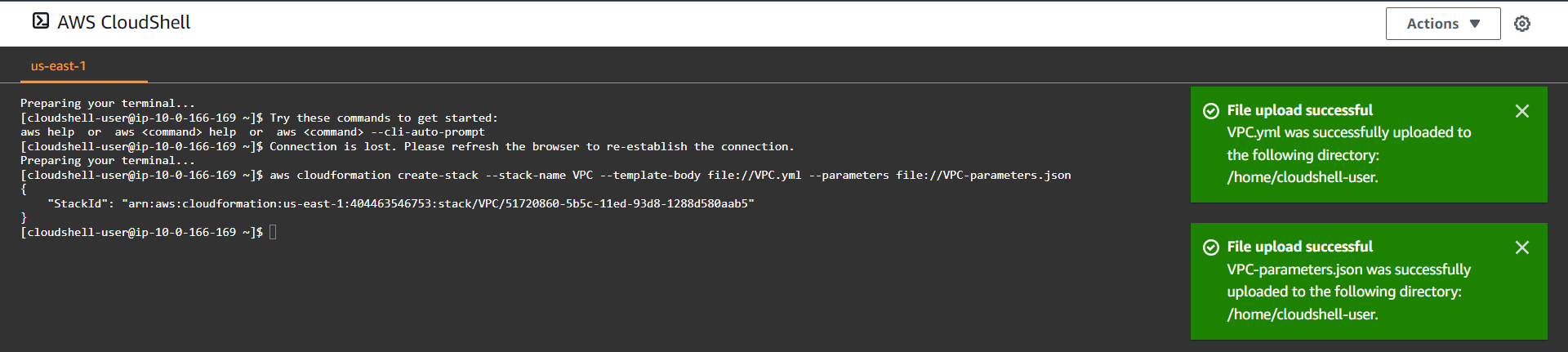


Figure 2. Command in cloud shell to setup VPC after uploading both the VPC.yml and the parameters file

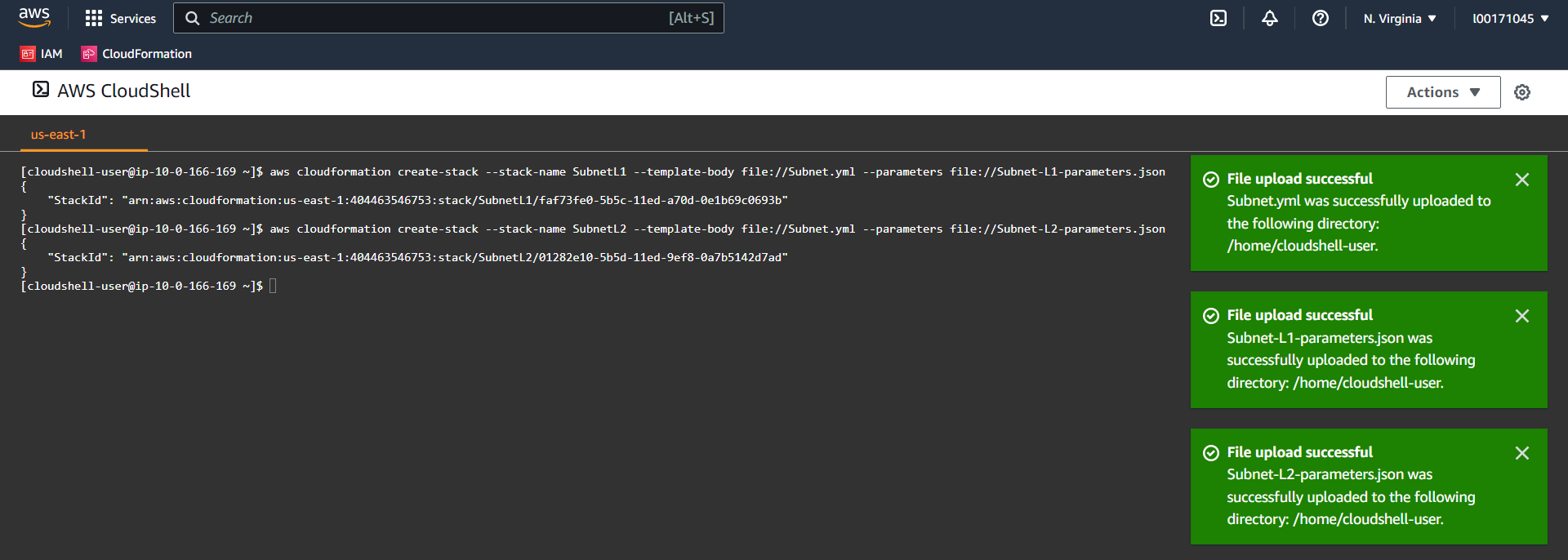


Figure 3. Two commands to setup left hand-side subnets for both jumpbox and a server (I have used bitnami’s nginx ami). Files uploaded: single Subnet.yml and two parameter files each

Text

Description automatically generated

Figure 4. Setting up NAT Gateway

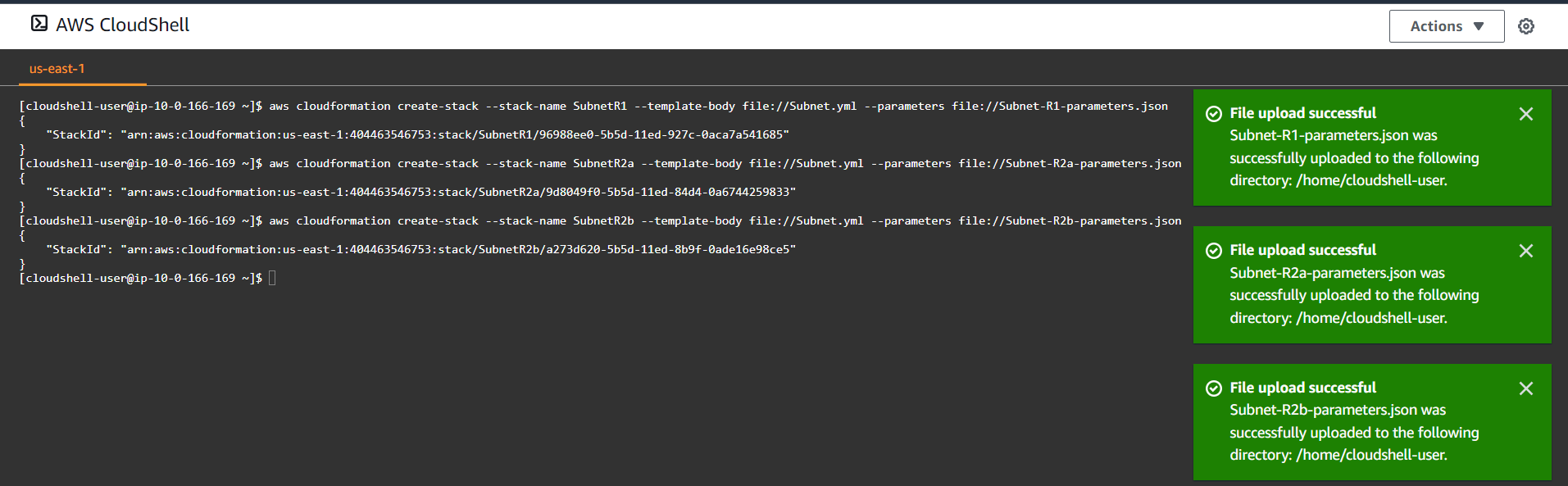


Figure 5. Three commands to setup right hand side subnets for app server (have used binami’s nginx ami). Files uploaded: single one for server and two for DBSubnet (required for DBInstance)

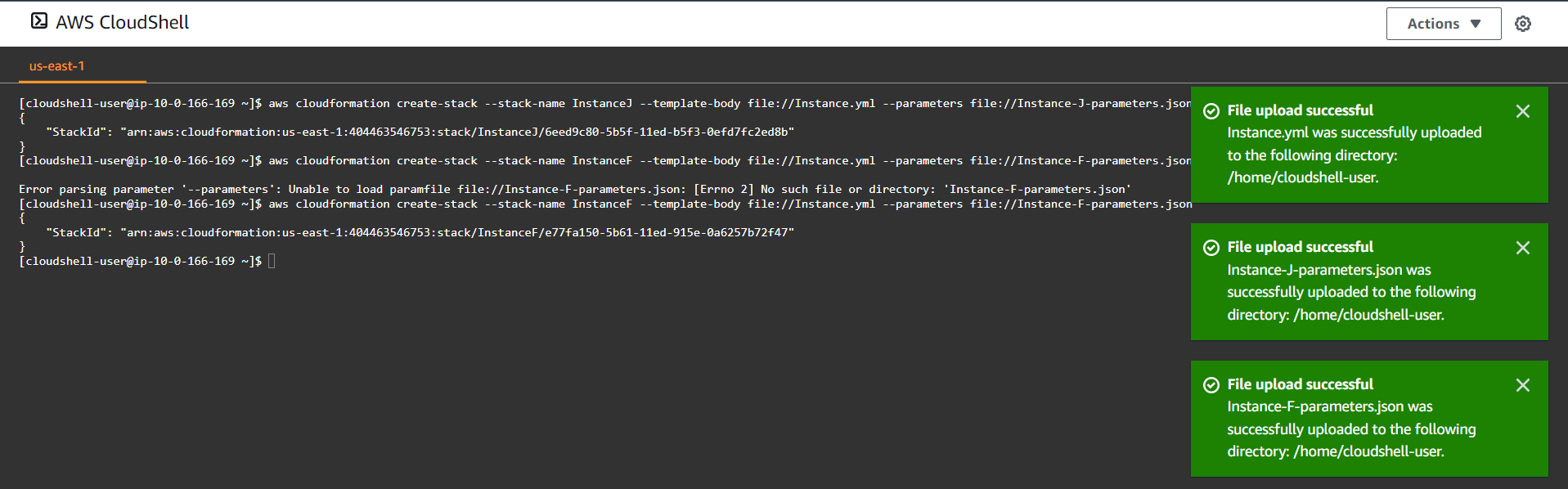


Figure 6. Two commands for creating the left hand-side instances J & F: Files uploaded: 3, One Instance.yml for the instance in general and two parameter files for each instance

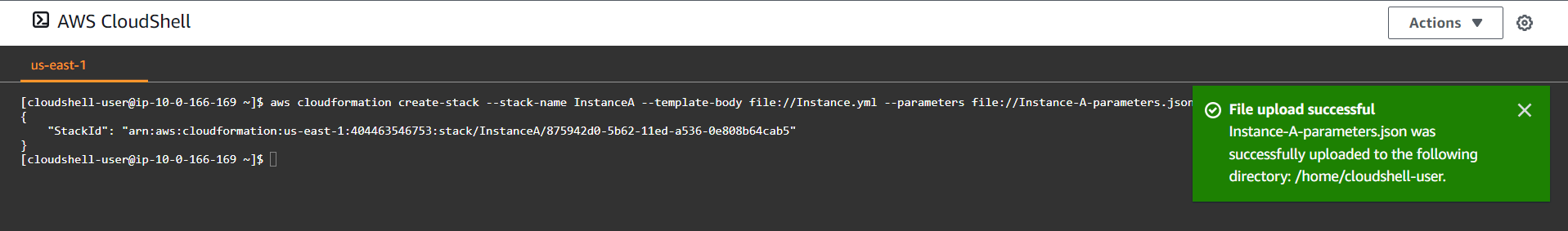


Figure 7. Command to create right hand-side app server (bitnami’s nginx ami). Files uploaded: one parameter file for instance A

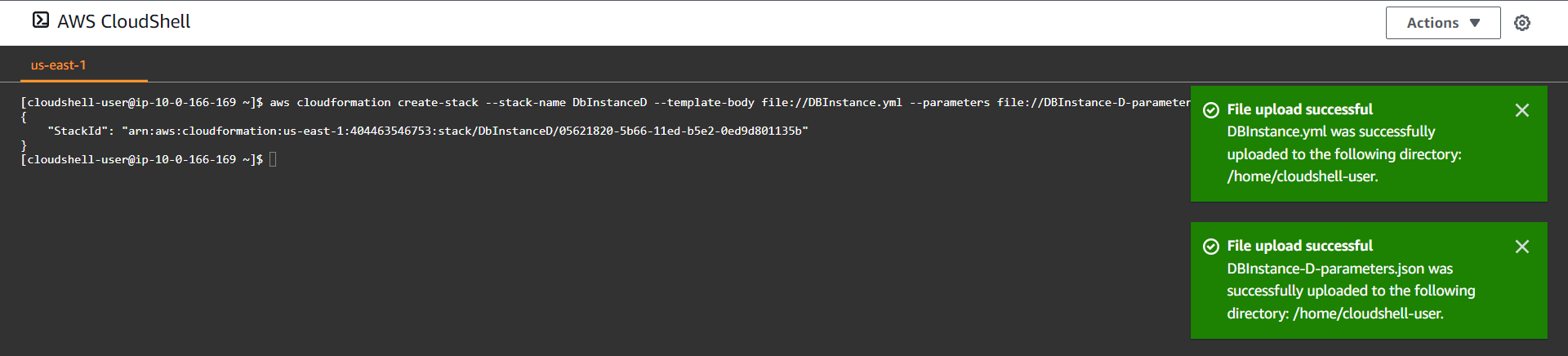


Figure 8. Command to create DBInstance. Files uploaded: one DBInstance.yml and another parameter file for single instance

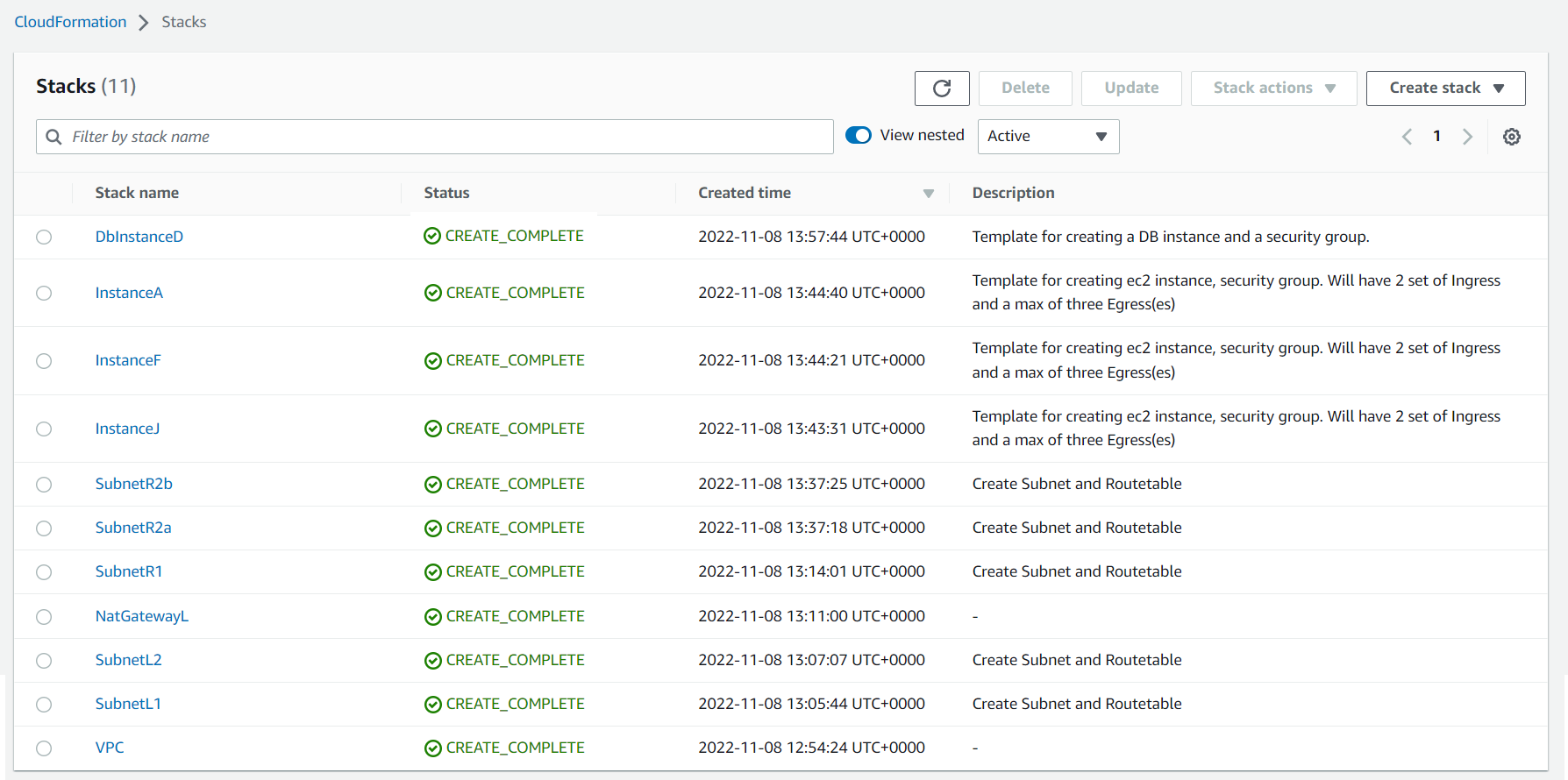


Figure 9. CloudFormation Stacks

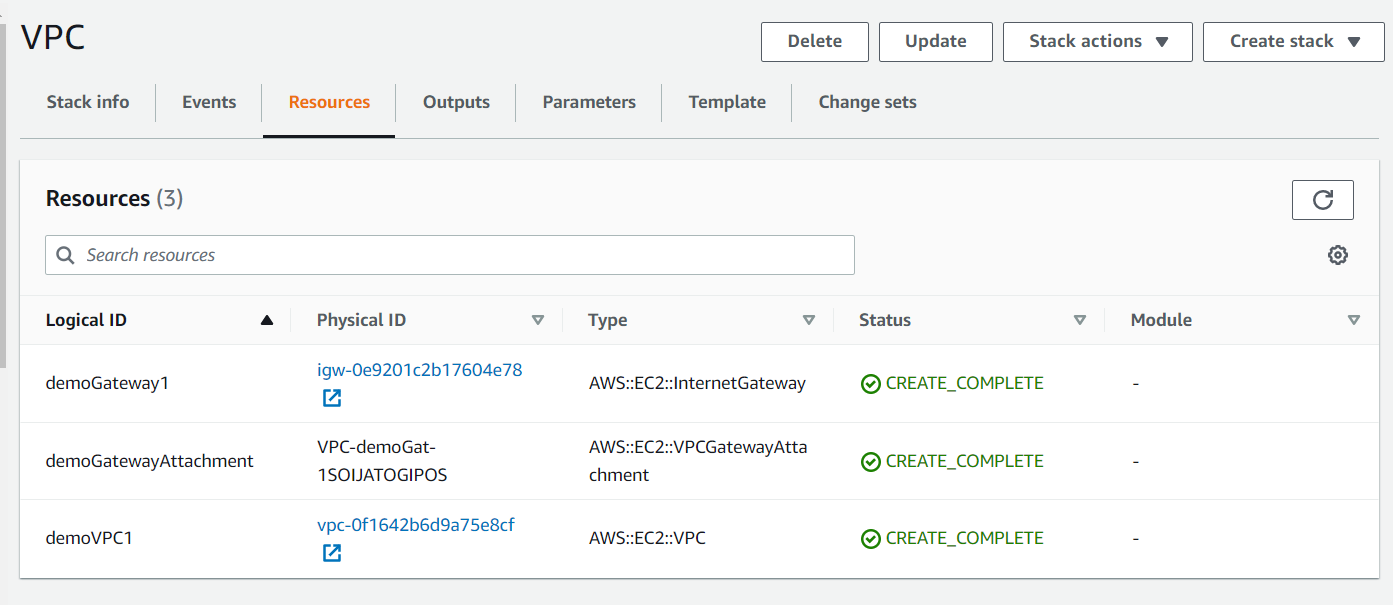


Figure 10a. VPC stack resources

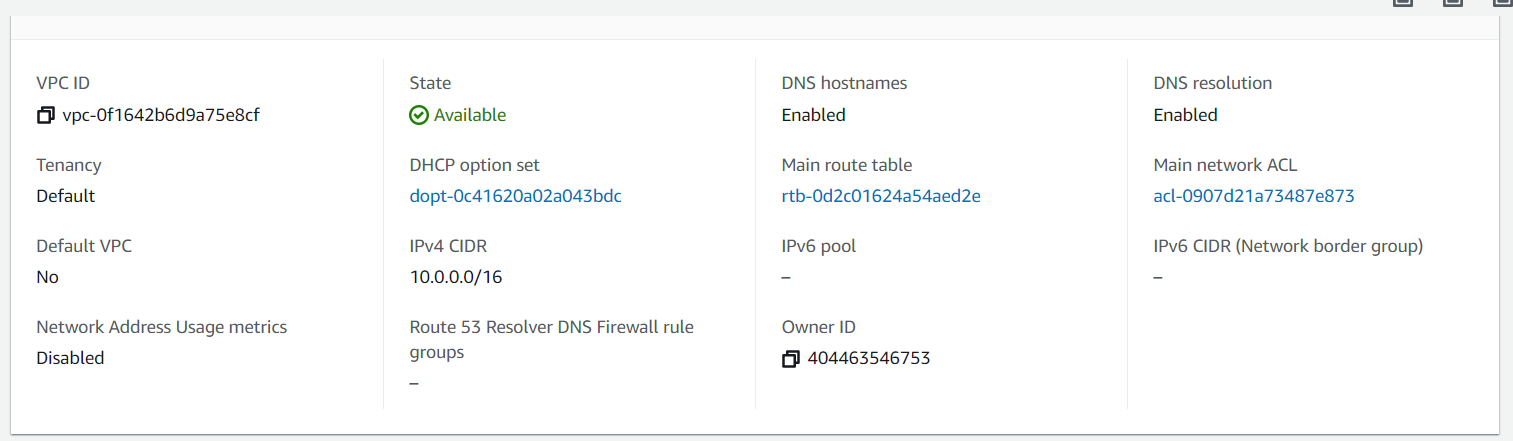


Figure 10b. VPC resource

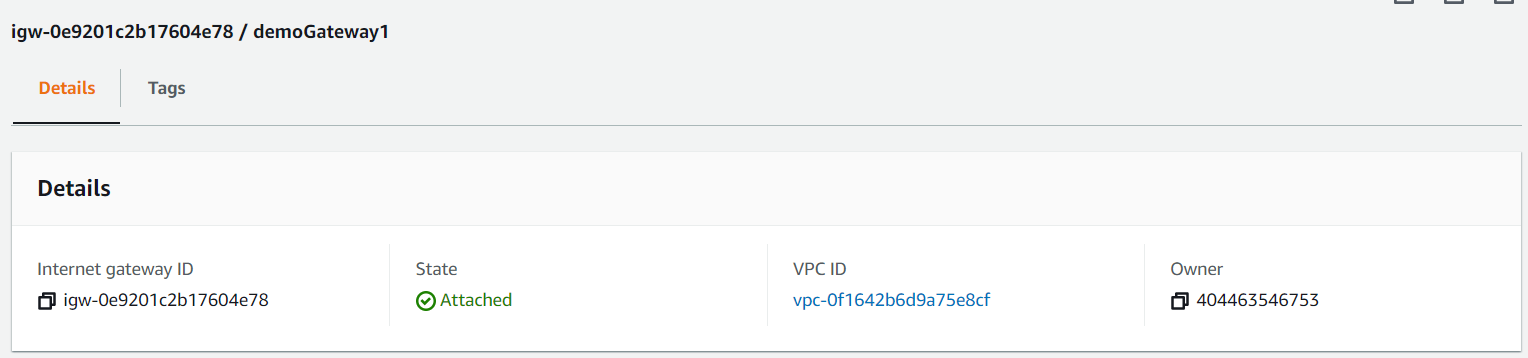


Figure 10c. InternetGateway, demoGateway associated to VPC resource

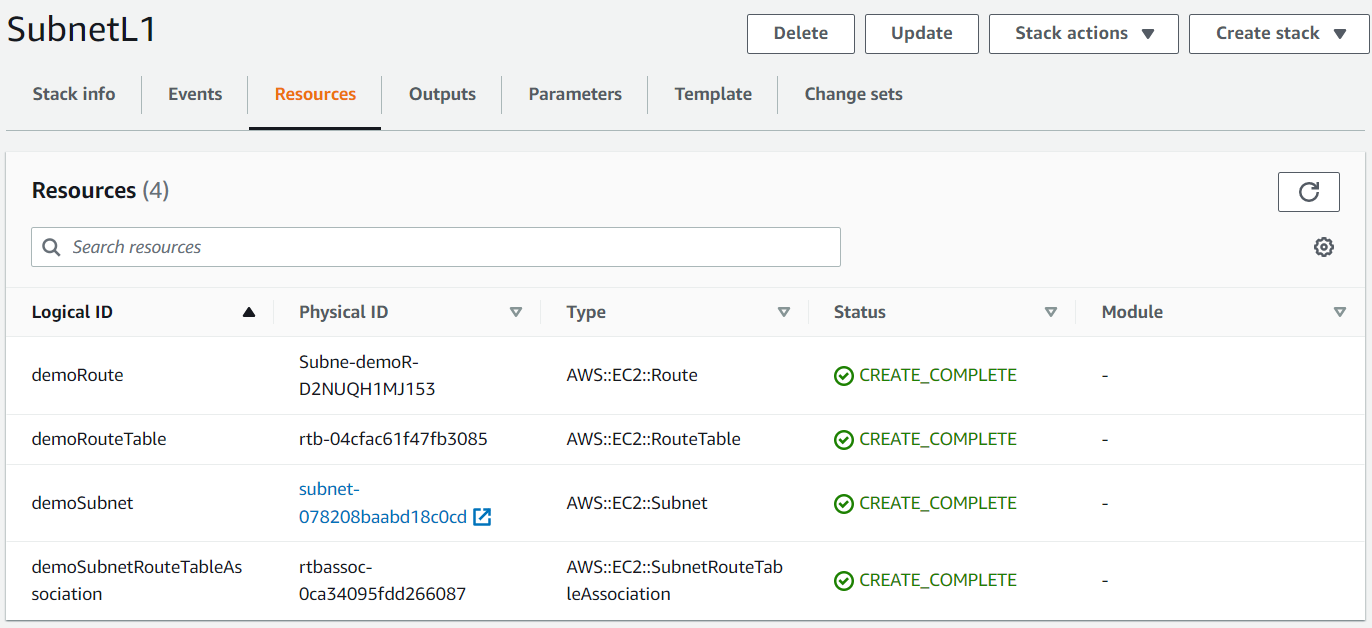


Figure 11a. Resources in SubnetL1 stack

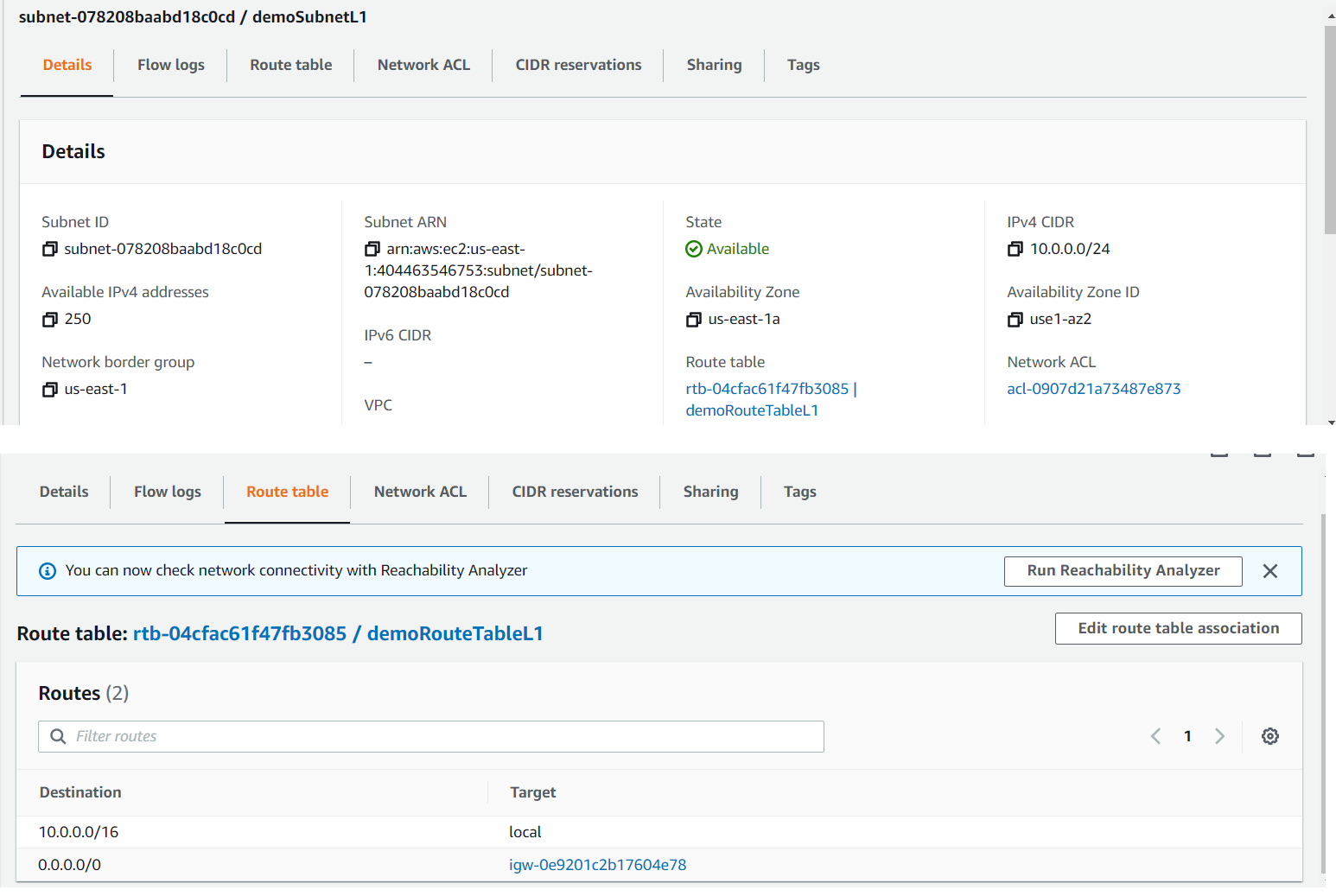


Figure 11b. demoSubnetL1 and associated route

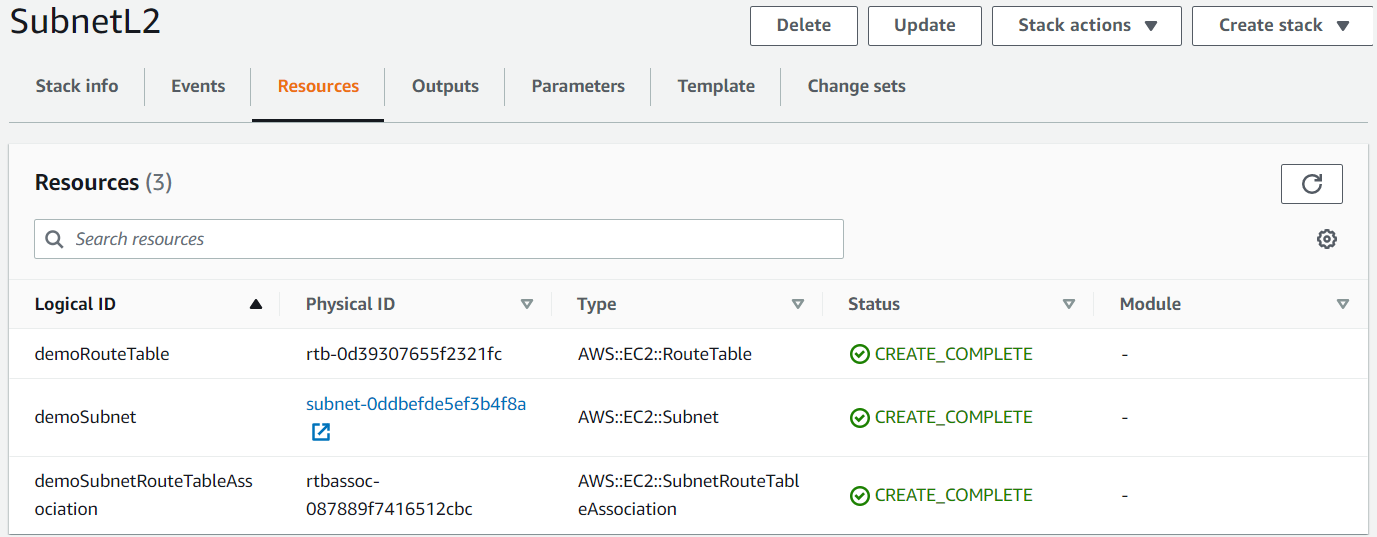


Figure 12a. Resources in SubnetL2 stack



Figure 12b. Subnet demoSubnetL2 and associated route (no other routes than the vpc inherited one)

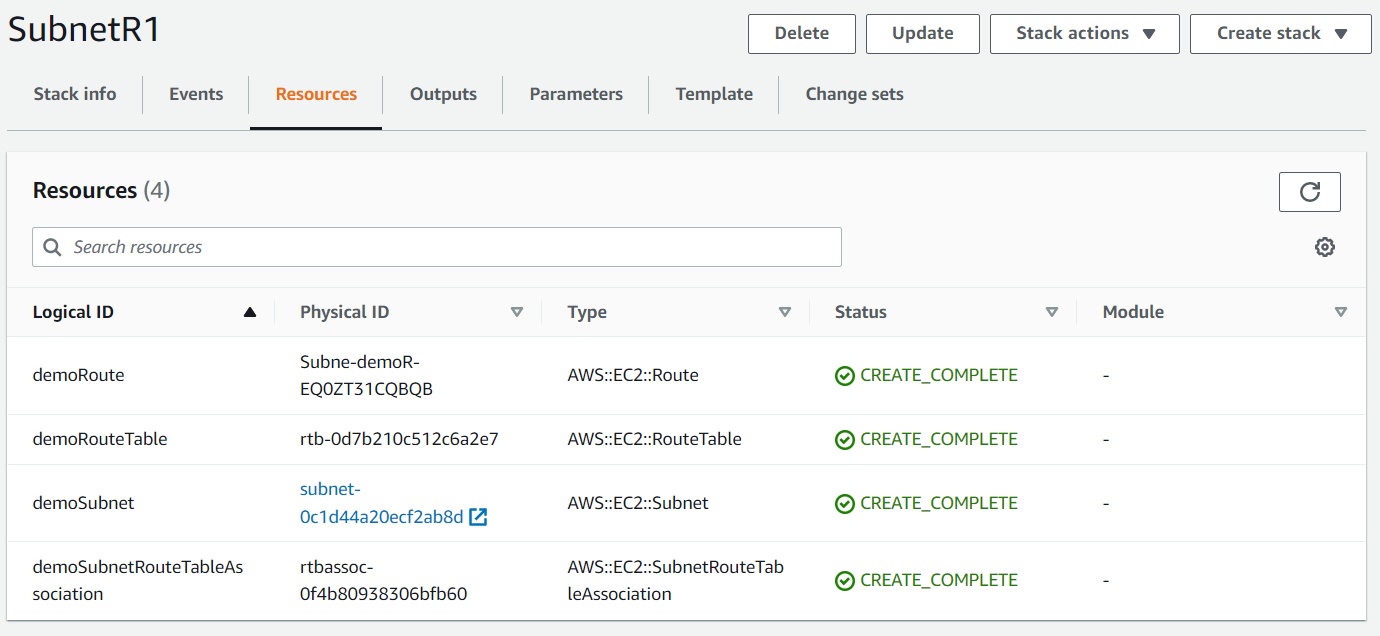


Figure 13a. Resources in the SubnetR1 stack

Graphical user interface, text, application, email

Description automatically generated

Figure 13b. Subnet demoSubnetR1’s details

Graphical user interface, text, application

Description automatically generated

Figure 13c. Subnet demoSubnetR1’s associated routes



Figure 14a. Resources in SubnetR2a

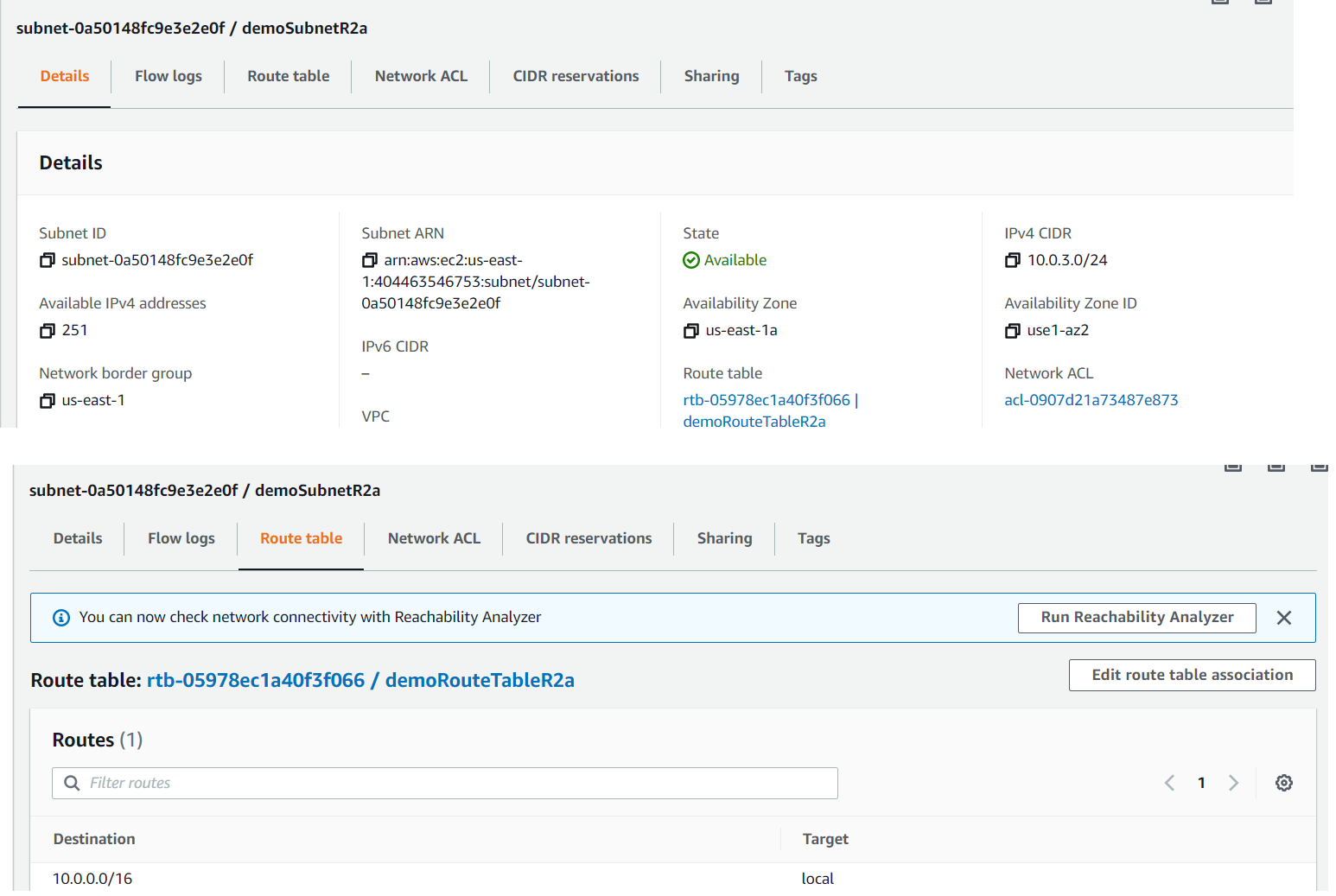


Figure 14b. Subnet demoSubnetR2a and its associated routes (no other routes than the VPC inherited one)

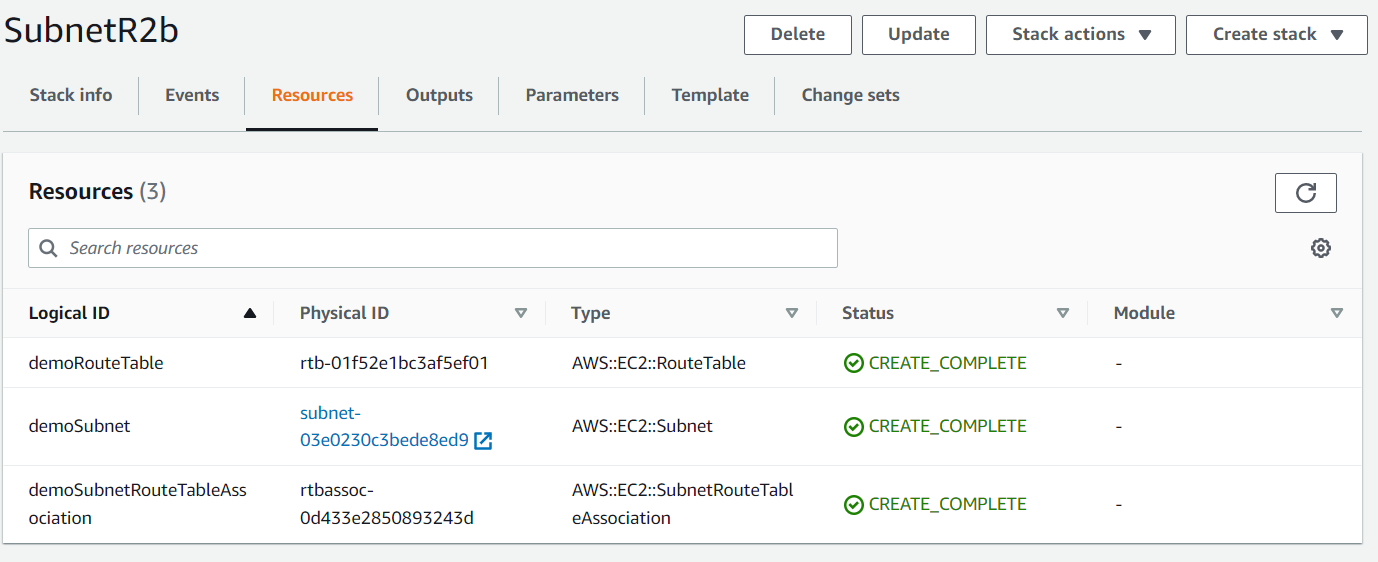


Figure 14c. Resources in SubnetR2b

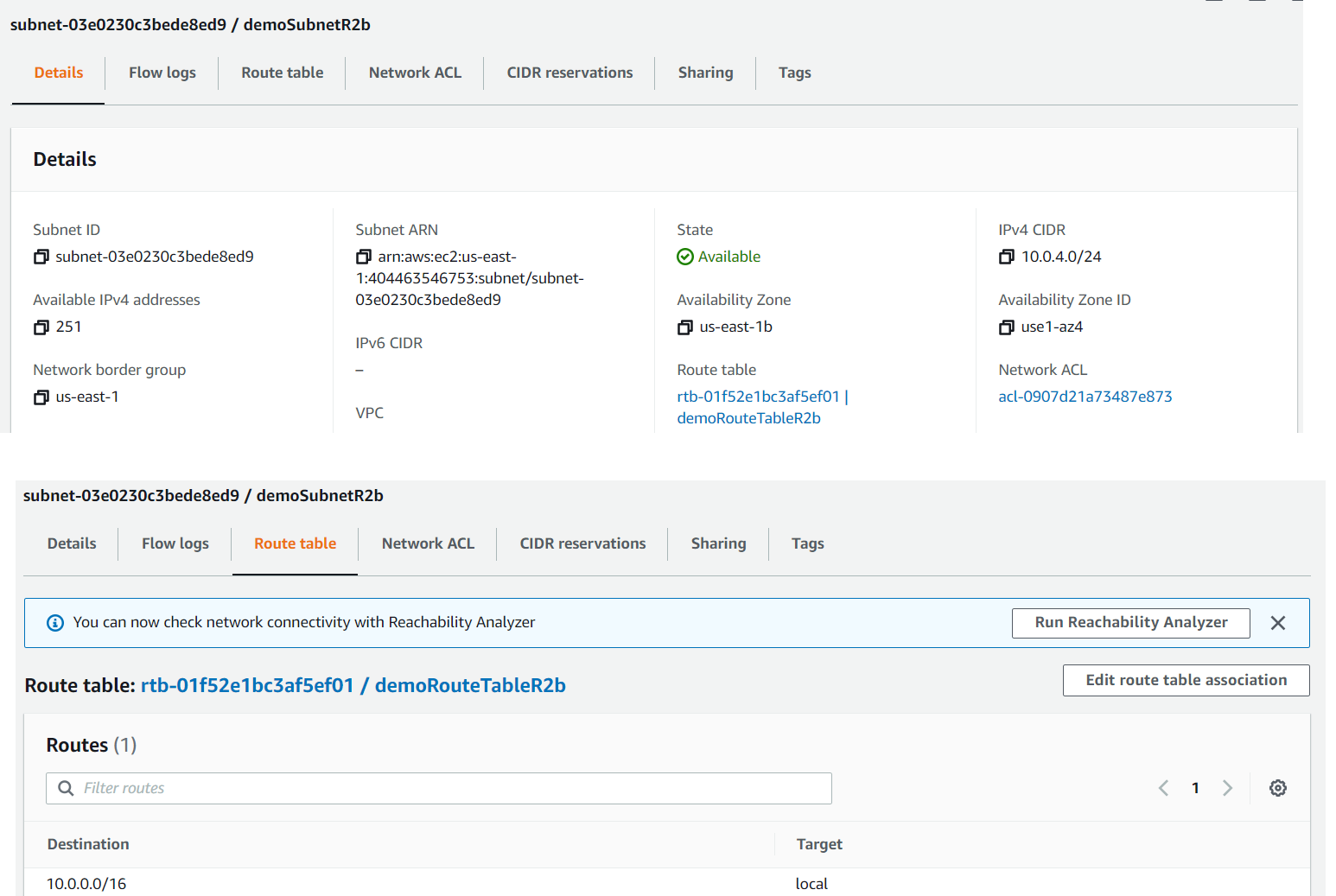


Figure 14d. Subnet demoSubnetR2b and its associated routes (no other routes than the VPC inherited one)

Graphical user interface, text, application, email

Description automatically generated

Fig 15. NAT gateway in public subnet (demoSubnetL1) associated with an EIP

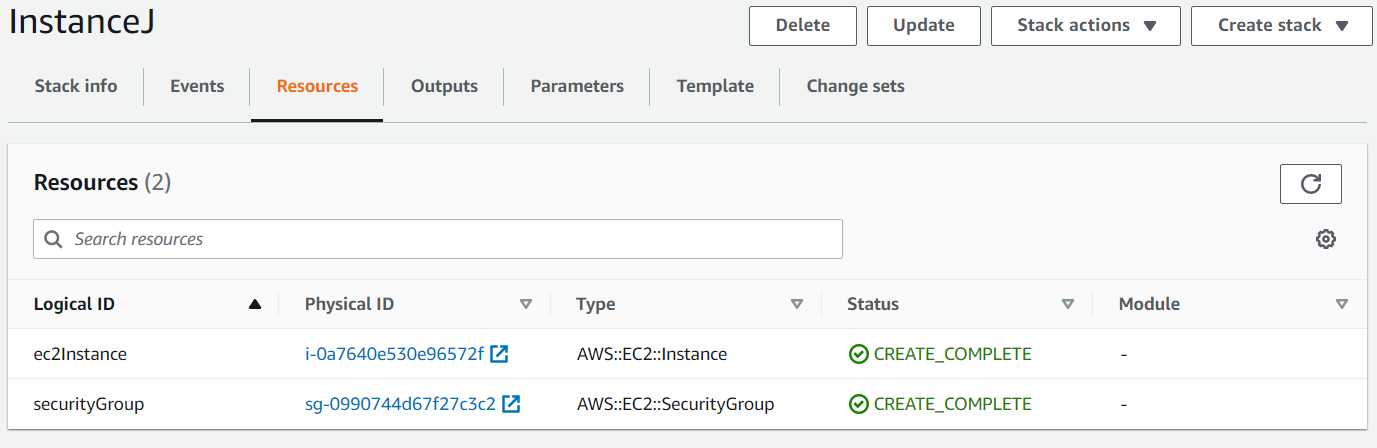


Figure 16a. Resources in InstanceJ stack

Graphical user interface, text, application

Description automatically generated

Figure 16b. Instance details of resource J

Graphical user interface, text, application

Description automatically generated

Figure 16c. routes allowed via security group in instance J

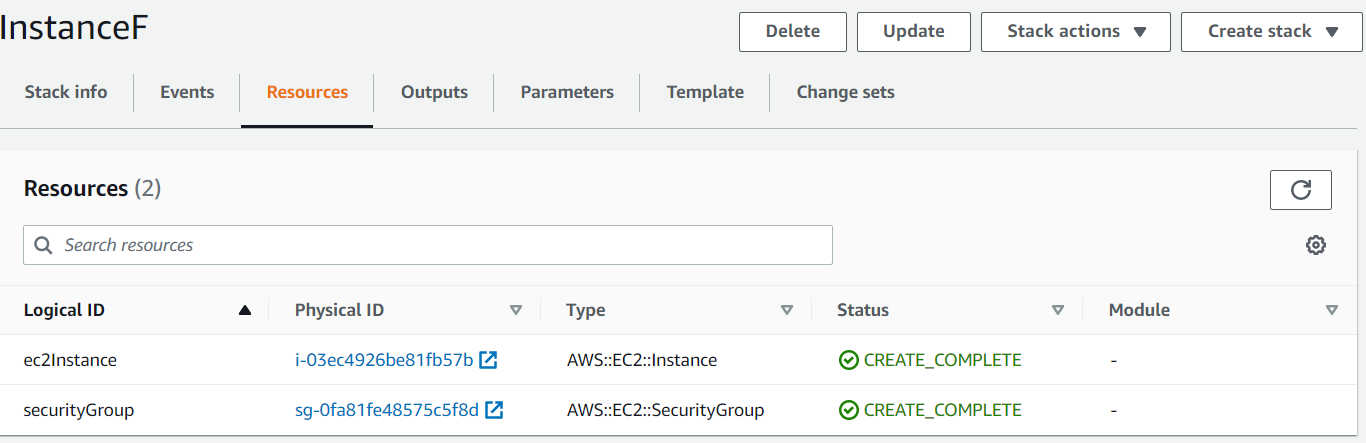


Figure 17a. Resources in stack InstanceF



Figure 17b. Instance details of resource F

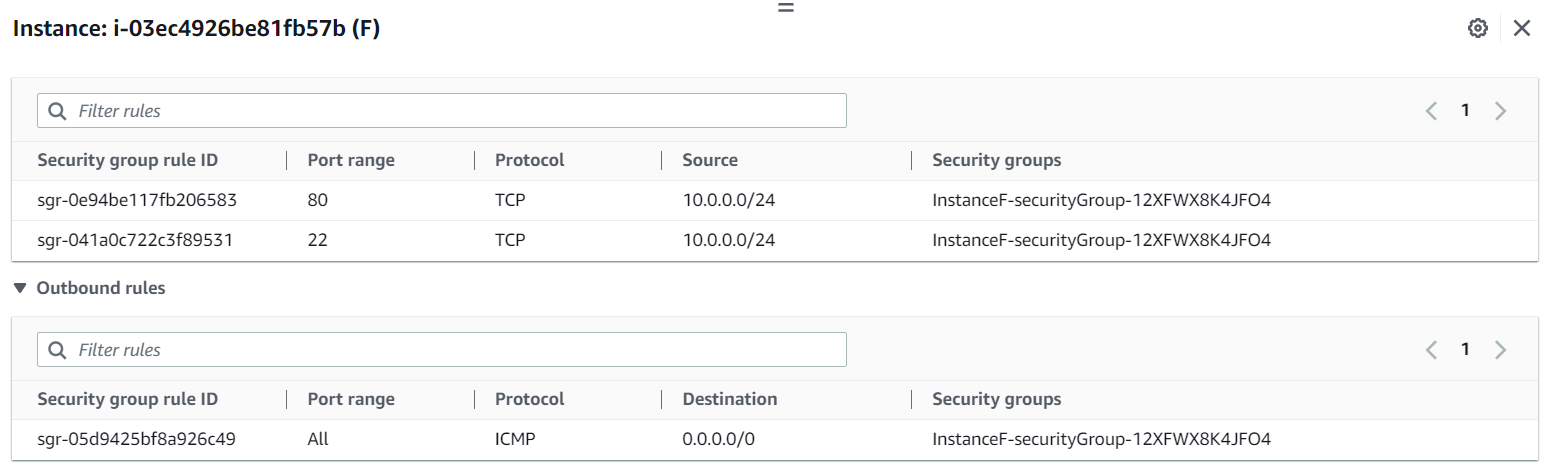


Figure 17c. Routes allowed via security group in instance F

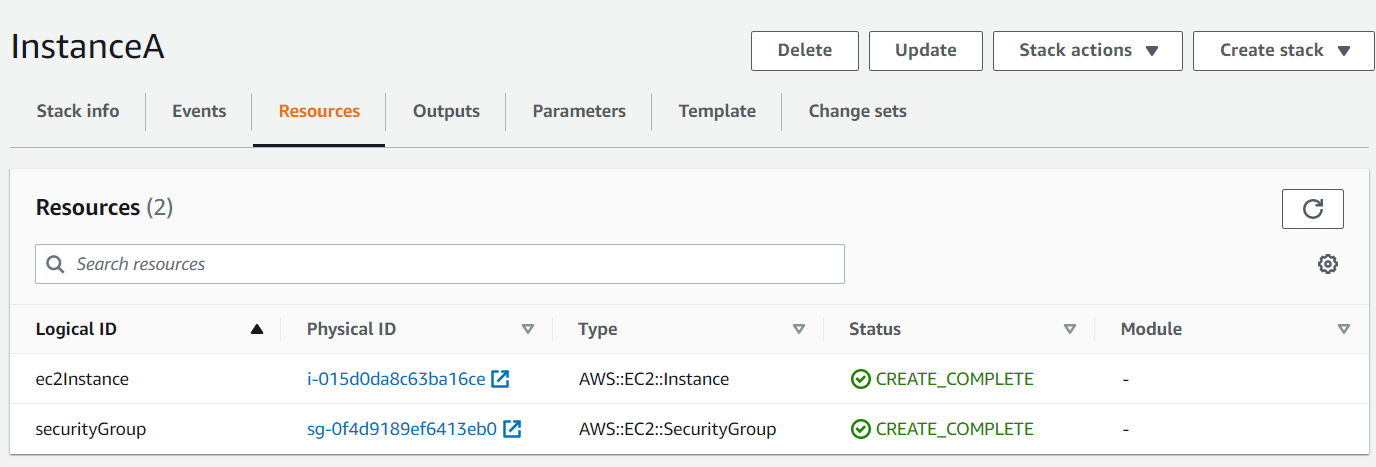


Figure 18a. Resources in stack InstanceA

Graphical user interface, text, application, email

Description automatically generated

Figure 18b. Details of instance A

Graphical user interface, text, application

Description automatically generated

Figure 18c. Routes allowed via security group in instance A

A screenshot of a computer

Description automatically generated

Figure 19a. Resources in stack DbInstanceD

Graphical user interface, text, application, email

Description automatically generated

Figure 19b. Security group of DBInstance D, allowing communication only via port 5432

Graphical user interface, application, Teams

Description automatically generated

Figure 19c. Subnet group for DBInstance D, holding two subnets in different Availability zones

Graphical user interface, text, application

Description automatically generated

Figure 19d. DBInstance details

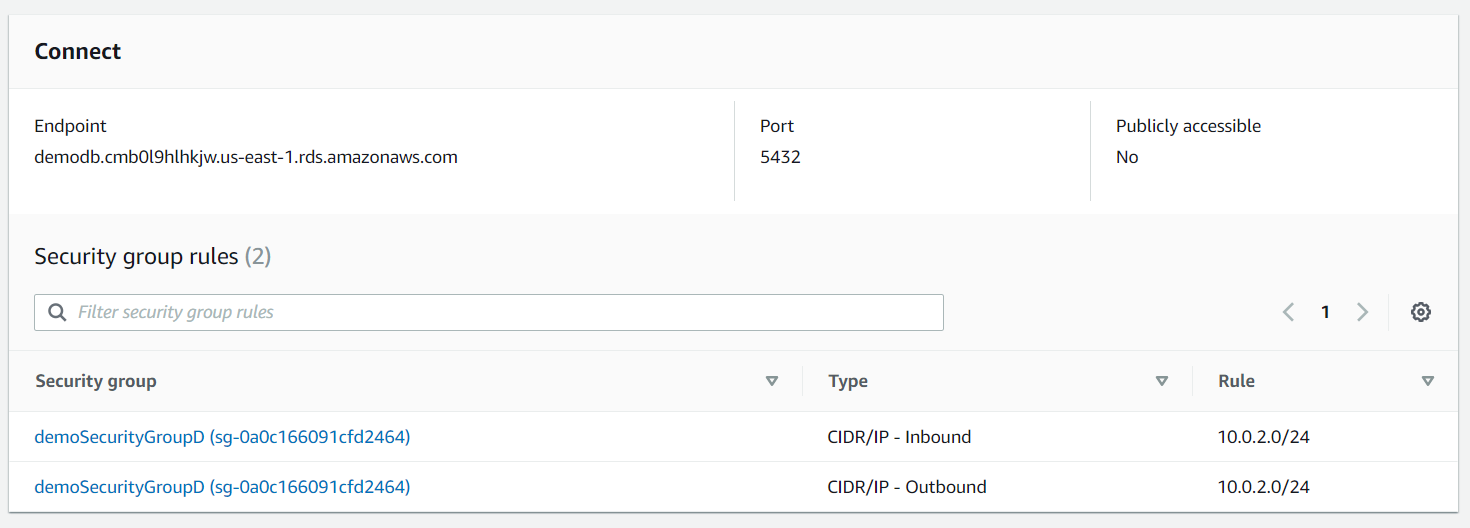


Figure 19e. Security group association of DBInstance D, allowing only communication to Instance A’s subnet

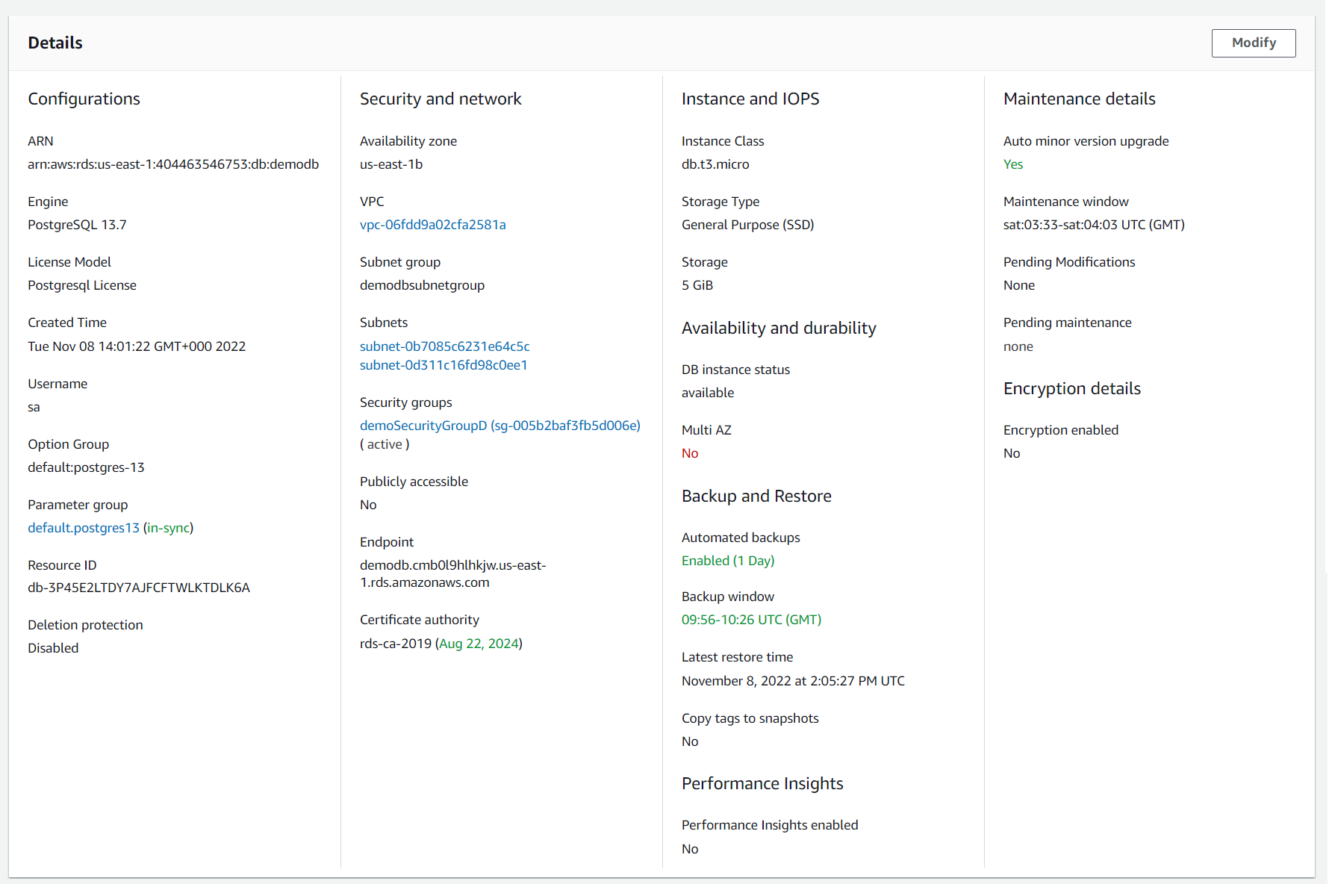


Figure 19f. DBInstance details

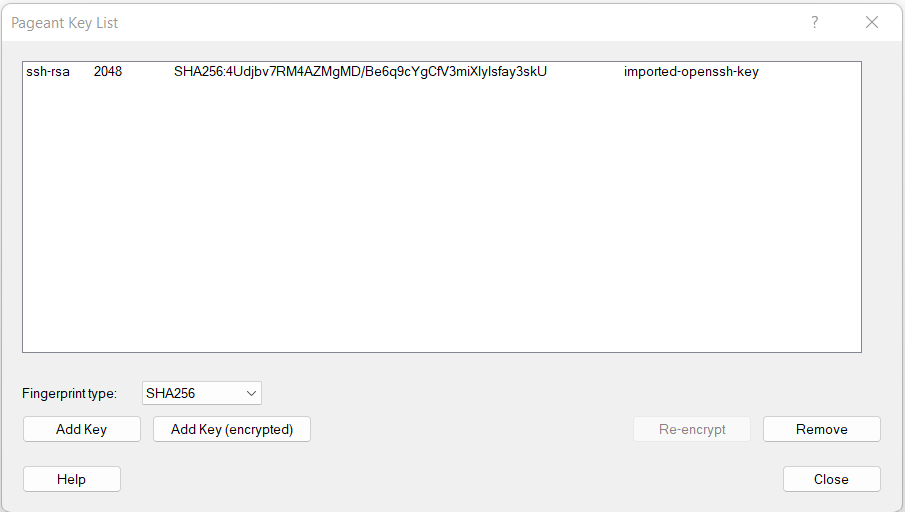


Figure 20a. Added ppk into Pageant Key list

Graphical user interface, application

Description automatically generated

Figure 20b. Public IP for instance J for SSH connection via PuTTY

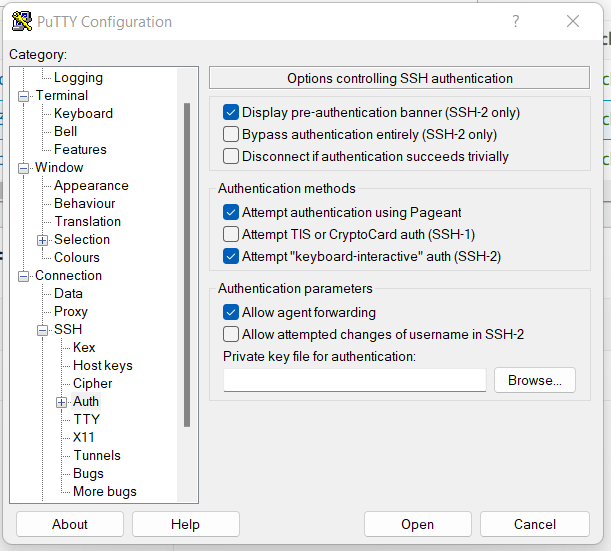


Figure 20c. Select allow agent forwarding

Text

Description automatically generated

Figure 21a. SSH into instance J

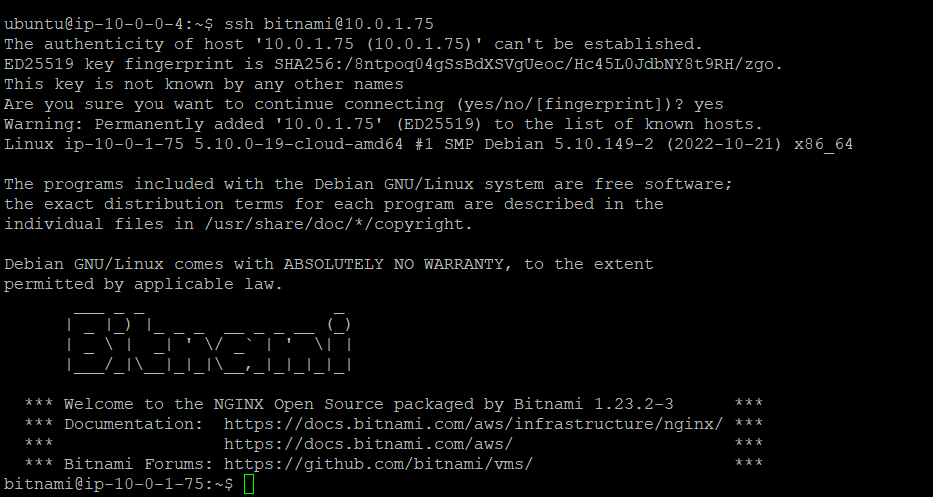


Figure 21b. SSH into instance F from instance J’s ssh/PuTTY terminal

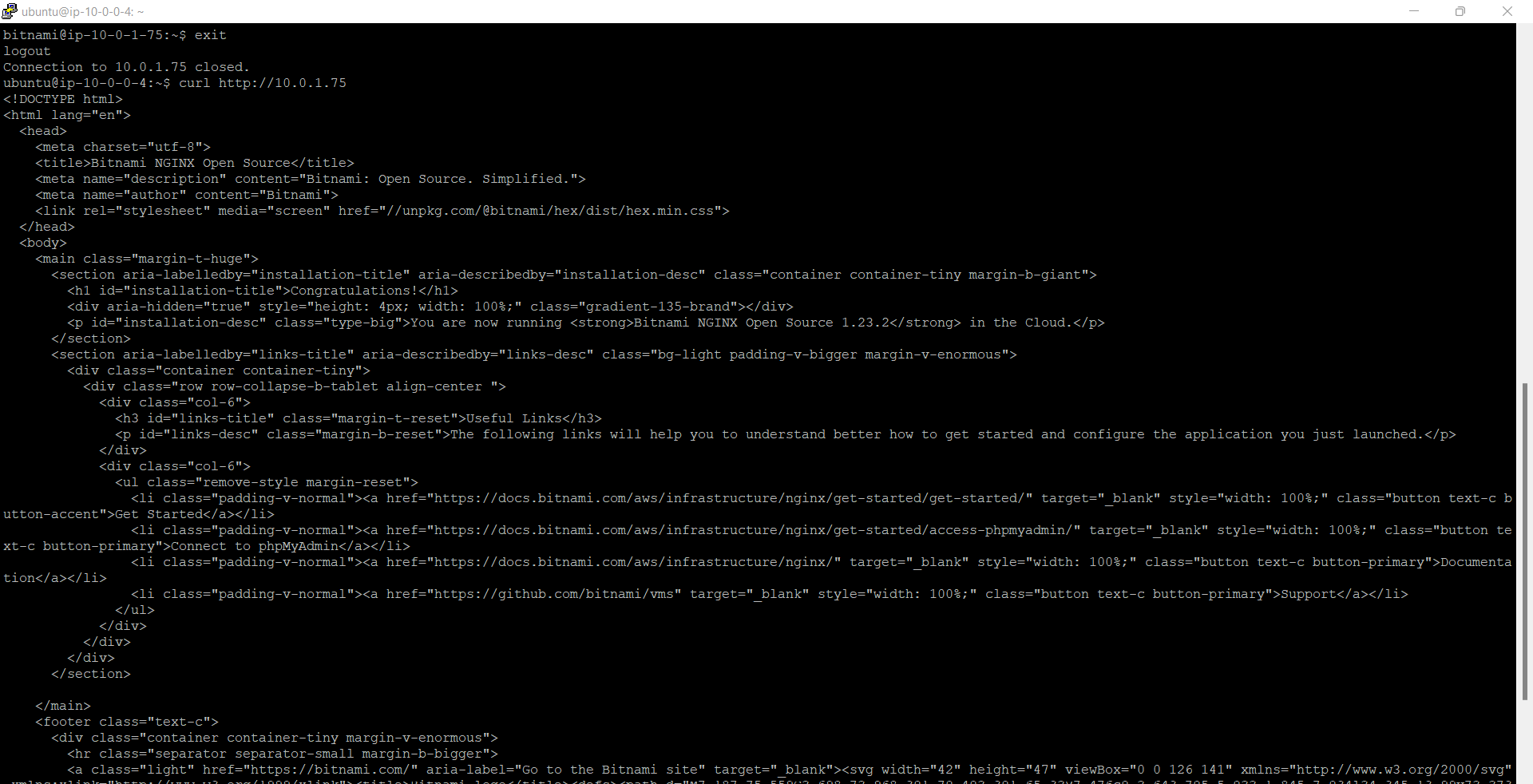


Figure 21c. curl to nginx’s landing page in instance F from SSH terminal of instance J

Text

Description automatically generated

Fig 22a. Ssh into Instance A from ssh terminal of instance J

Text

Description automatically generated

Figure 22b. sudo apt-get update on server A (via ssh terminal of Instance J) and install postgresql-client in A. And connect to DBInstance’s postgres engine using its endpoint

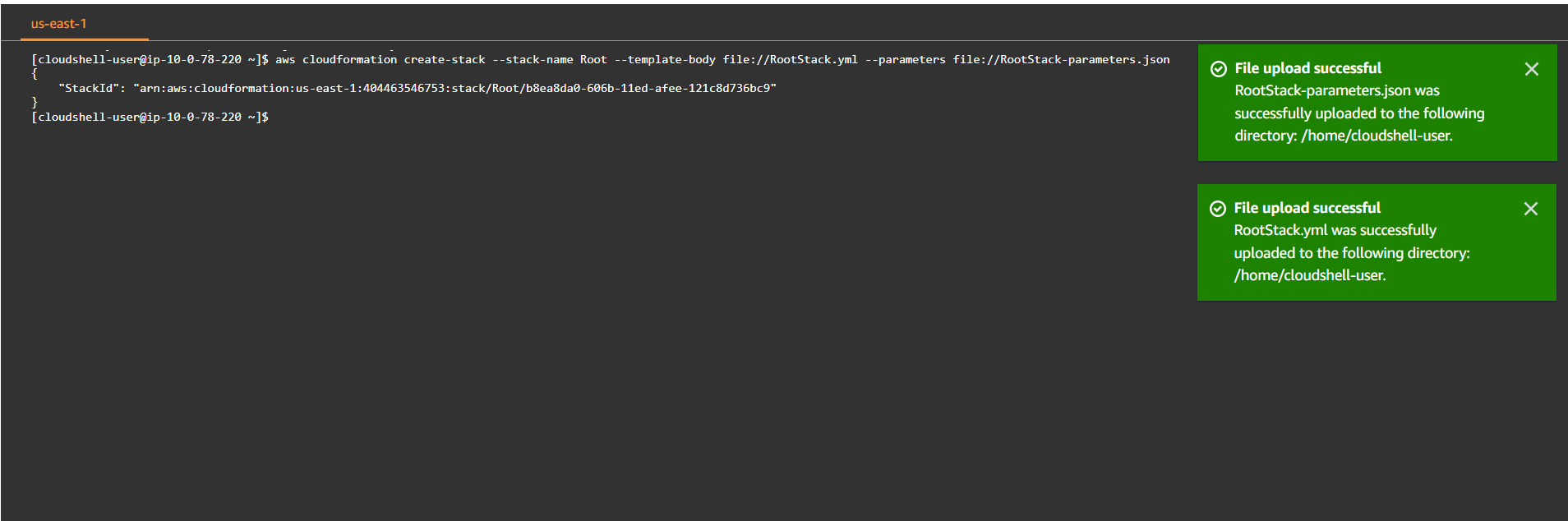


Figure 23 a. Creating nested stack

Graphical user interface, text, application, email

Description automatically generated

Fig 23b. Nested stack