15 – Creating Virtual Private Cloud (VPC) with CloudFormation

---

Table of Contents

Overview

This lab exercise is a bit different in several regards:

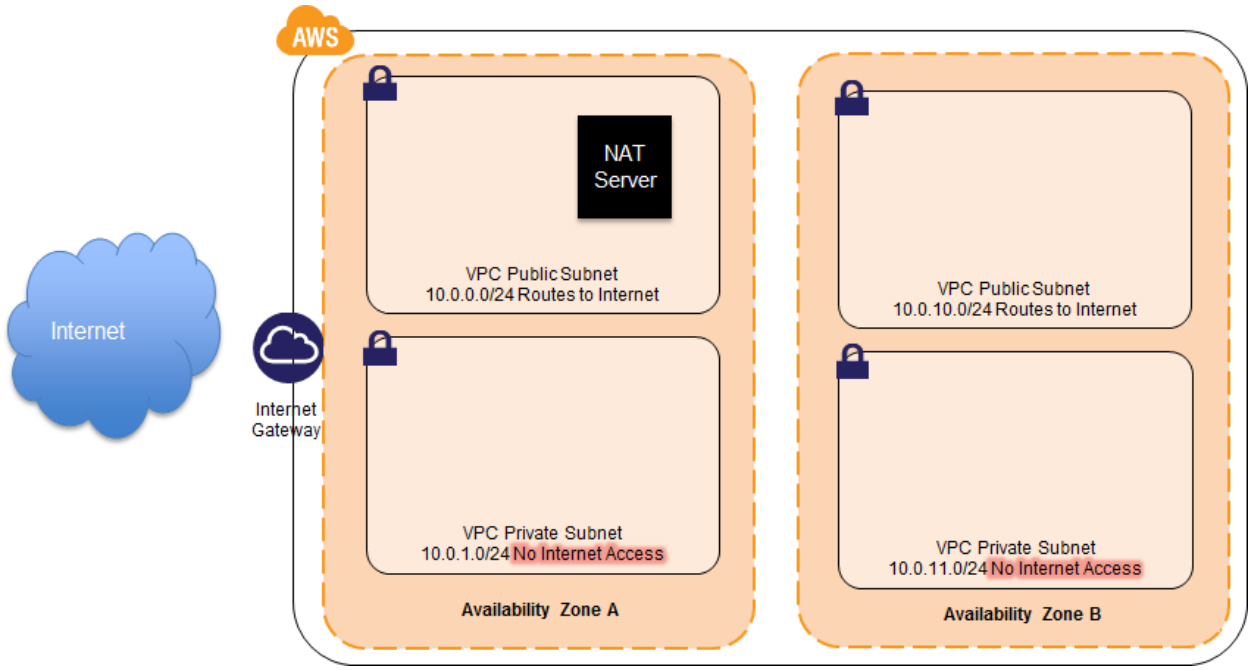
First, we assume that you either completed the *How to Create a VPC* lab, or that you have equivalent experience.

Second, we also assume that you completed the CloudFormation lab (or that you dream JSON).

But most of all this is a walkthrough of a template rather than "learn how to build it". There are no exercises as such required; although the walk-through shows how to build the template, with explanations for each step.

The End Goal

What we plan to create is a four-subnet VPC that spans two Availability Zones, as depicted below. The final VPC will also include a NAT that allows servers in the private subnets to communicate with the Internet in order to download packages and updates. There is no way to reach these servers on the private subnet from the Internet, though.



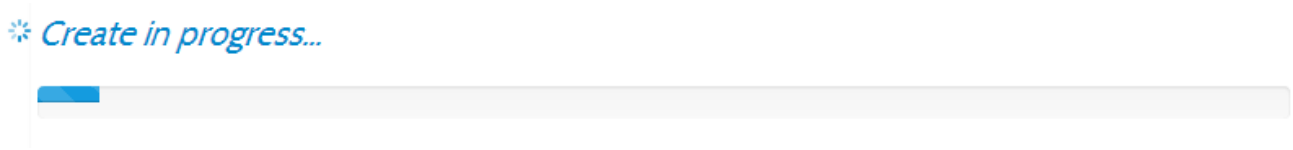
Let's look at the details of this VPC:

- There are two Availability Zones, in order to provide redundancy and therefore High Availability
- Each Availability Zone has one public subnet (10.0.0.0 and 10.0.10.0)
- Each Availability Zone has one private subnet (10.0.1.0 and 10.0.11.0)
- All subnets can talk to each other freely
- Only the Public subnets are accessible from the Internet
- Servers in the private subnet can only make outbound calls to the Internet via the NAT server. No inbound traffic is accepted
- We will further restrict traffic via security groups

Start your *qwikLAB*™

1) Start your *qwikLAB*™
   Use the 'Start Lab' button to start your lab.
   (Hint: If you are prompted for a token, please use one you've been given or have purchased.)

   You will see the lab creation in progress.

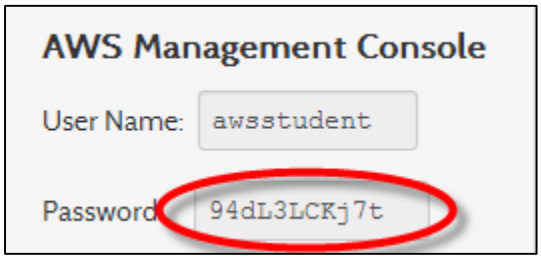


2) Note a few properties of the lab.

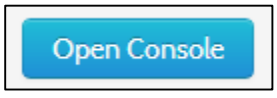
   a. **Duration -** The time the lab will run for before shutting itself down.
   b. **Setup Time -** The estimated lab creation time on starting the lab.
   c. **AWS Region** - The AWS Region the lab resources are being created in.

3) Copy the Password provided.

   a. Hint: selecting the value shown and using Ctrl+C works best



4) Click the 'Open Console' button.



5) Login to the AWS Management Console

Enter the User Name '**awsstudent**' and paste the password you copied from the lab details in *qwikLAB*™ into the Password field.

Click on the 'Sign in using our secure server' button.

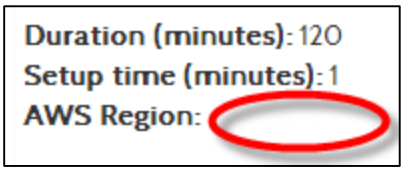
In this step you logged into the AWS Management Console using login credentials for a user provisioned via AWS Identity Access Management in an AWS account by *qwikLAB*™.

AWS Management Console

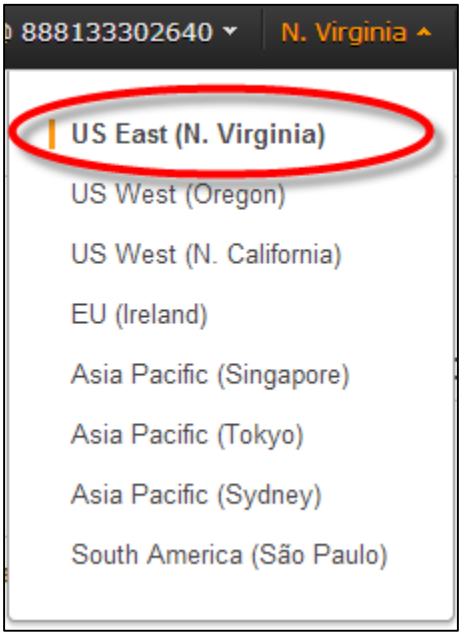
Once logged in, select "VPC" from the service console.

Confirm your AWS Region

Note the AWS Region set for your lab in *qwikLAB*™



Select or confirm that the same AWS Region is already set in the AWS Management Console



Choose an Editor

You can use your preferred text editor to author the JSON CloudFormation template described in this lab.

Create the Template Basics

We'll sketch in the framework as per below. This bare-bones template won't actually validate as a valid CloudFormation template because it is missing the Resources section; however the point is that none of this is specific to a VPC per-se.

Note the comma after the Description section. Ordinarily the final object in a template does not have a comma; however given that this template won't validate anyways there is no reason to make you go back and add one as we add sections.

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",

  "Description" : "VPC with 4 subnets across 2 Availability Zones.",

}
```

Add Input Parameters

We need to collect some information from the user, in order to successfully create the VPC.

The first choice is which instance type that you want the NAT to run on. These AMIs serve just one purpose: allow instances on the Private subnets to call out to the Internet to download updates. Traffic from the Internet is not permitted to make inbound connections. Because NAT instances are appliances, we won't even bother assigning a SSH key pair that enables us to log in to the instance: should the instance misbehave, we will simply replace it with another, more cooperative, instance. M1.small instances will serve all except the largest networks.

The CorporateCidrIP parameter defaults to the entire Internet. If you want to restrict inbound access to the VPC (typically, only from your corporate office), here's the place to enter that range.

Finally, you need to enter the Availability Zones that were collected above. Make certain that the default values listed below align with your actual account.

```
  "Parameters" : {

    "NATInstanceType" : {
      "Description" : "NAT EC2 instance type",
      "Type" : "String",
      "Default" : "m1.small",
      "AllowedValues" : [ "m1.small","m1.medium" ]
    },

    "CorporateCidrIp" : {
      "Description" : "Your Company's CidrIp (to restrict traffic to be
authorized ONLY from corporate office)",
```

```
      "Type" : "String",
      "Default" :  "0.0.0.0/0"
    }
  },
```

Add a Mappings Section

One of the objects in a CloudFormation template is the Mappings section. This section is a series of lookup tables that in the first case tell us whether an instance type supports 32-bit or 64-bit operating systems, and in the second case what AMI we should launch in a given region.

The power of this section is self-evident: a single template works across multiple regions.

This particular map is a list of NAT AMIs that are available around the world. We assembled the list by visiting each region, selecting EC2 -> AMIs -> Amazon Images, and typing *vpc* into the search string at the top of the list.

```
"Mappings" : {
  "AWSInstanceType2Arch" : {
    "m1.small"   : { "Arch" : "64" },
    "m1.medium"  : { "Arch" : "64" },
    "m1.large"   : { "Arch" : "64" },
    "m1.xlarge"  : { "Arch" : "64" },
    "m2.xlarge"  : { "Arch" : "64" },
    "m2.2xlarge" : { "Arch" : "64" },
    "m2.4xlarge" : { "Arch" : "64" },
    "c1.medium"  : { "Arch" : "64" },
    "c1.xlarge"  : { "Arch" : "64" }
  },
  "AWSRegionArch2AMI" : {
    "us-east-1"      : {"64" : "ami-f619c29f"},
    "us-west-2"      : {"64" : "ami-52ff7262"},
    "us-west-1"      : {"64" : "ami-3bcc9e7e"},
    "eu-west-1"      : {"64" : "ami-e5e2d991"},
    "ap-southeast-1" : {"64" : "ami-02eb9350"},
    "ap-southeast-2" : {"64" : "ami-ab990e91"},
    "ap-northeast-1" : {"64" : "ami-14d86d15"},
    "sa-east-1"      : {"64" : "ami-0039e61d"}
  }
},
```

We'll reference this section when we define the NAT instance in the template.

Add Resources Section

Let's get down to the business of defining the actual VPC and its component elements. We'll start by roughing in the section, as follows:

```
Resources : {
}
```

So the entire template now looks like this:

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
```

```
  "Description" : "VPC with 4 subnets across 2 Availability Zones.",

  "Parameters" : {

    "NATInstanceType" : {
      "Description" : "NAT EC2 instance type",
      "Type" : "String",
      "Default" : "m1.small",
      "AllowedValues" : [ "m1.small","m1.medium" ]
    },

    "CorporateCidrIp" : {
      "Description" : "Your Company's CidrIp (to restrict traffic to be
authorized ONLY from corporate office)",
      "Type" : "String",
      "Default" :  "0.0.0.0/0"
    }

  },

  "Mappings" : {
          "AWSInstanceType2Arch" : {
            "m1.small"   : { "Arch" : "64" },
            "m1.medium"  : { "Arch" : "64" },
            "m1.large"   : { "Arch" : "64" },
            "m1.xlarge"  : { "Arch" : "64" },
            "m2.xlarge"  : { "Arch" : "64" },
            "m2.2xlarge" : { "Arch" : "64" },
            "m2.4xlarge" : { "Arch" : "64" },
            "m3.xlarge"  : { "Arch" : "64" },
            "m3.2xlarge" : { "Arch" : "64" },
            "c1.medium"  : { "Arch" : "64" },
            "c1.xlarge"  : { "Arch" : "64" }
          },
          "AWSRegionArch2AMI" : {
            "us-east-1"      : {"64" : "ami-f619c29f"},
            "us-west-2"      : {"64" : "ami-52ff7262"},
            "us-west-1"      : {"64" : "ami-3bcc9e7e"},
            "eu-west-1"      : {"64" : "ami-e5e2d991"},
            "ap-southeast-1" : {"64" : "ami-02eb9350"},
            "ap-southeast-2" : {"64" : "ami-ab990e91"},
            "ap-northeast-1" : {"64" : "ami-14d86d15"},
            "sa-east-1"      : {"64" : "ami-0039e61d"}
          }
        },

  "Resources" : {

  }
}
```

Define the VPC Address Space

The default address space for this template is hard-coded to 10.0.0.0/16 corresponding to the default value in the VPC Wizard in the AWS Management Console. You can either change the value in the

template or even make it an input parameter. Because it's what the AWS wizard uses, and because this is also a really common setting, we'll use the default.

```
"VPC" : {
   "Type" : "AWS::EC2::VPC",
   "Properties" : {
     "CidrBlock" : "10.0.0.0/16"
   }
 },
```

Set Up the Internet Gateway

In order to talk to the Internet, the VPC needs a gateway. First we'll define it, and then we'll map the gateway to the VPC.

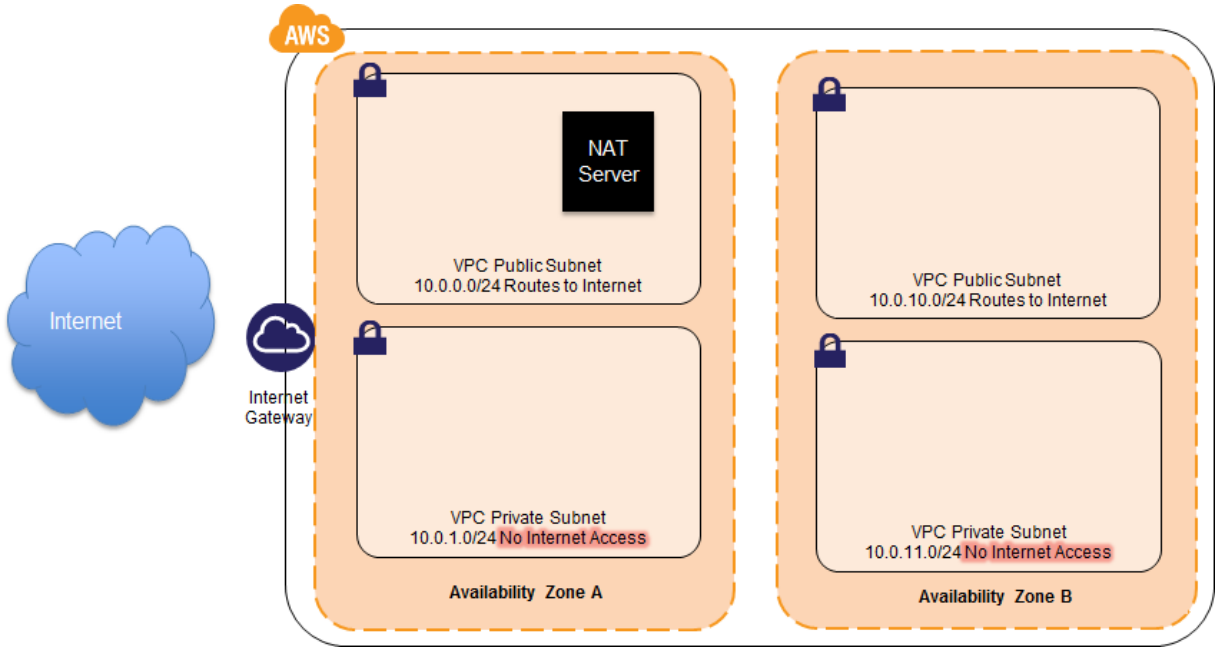
Note the "Ref" property here, CloudFormation will look up the VPC ID that was assigned upon creation, and automatically map it to the Gateway.

```
"InternetGateway" : {
   "Type" : "AWS::EC2::InternetGateway",
   "Properties" : {
   }
 },

 "AttachGateway" : {
    "Type" : "AWS::EC2::VPCGatewayAttachment",
    "Properties" : {
      "VpcId" : { "Ref" : "VPC" },
      "InternetGatewayId" : { "Ref" : "InternetGateway" }
    }
 },
```

Define the Subnets

Let's refer to the original diagram again. As illustrated, we need multiple subnets that map to Availability Zones in a manner that each AZ contains both a public and private subnet.



Each of the subnets uses the "Ref" keyword to refer to the VPC previously created by the template. The subnets also make use of the intrinsic functions Fn::Select and Fn::GetAZs. The former selects one item from a list by the item's zero-based index. The latter returns a list of Availability Zones accessible to the user inside the specified region or inside the current region if none is specified. These functions used together allow this template to be run in any region, and the subnets will be created inside the first two Availability Zones in that region. Alternately, the template could be modified to allow subnet Availability Zones to be user-specified as input parameters and referenced with the "Ref" keyword.

```
    "PublicSubnet1" : {
      "Type" : "AWS::EC2::Subnet",
      "Properties" : {
        "VpcId" : { "Ref" : "VPC" },
        "CidrBlock" : "10.0.0.0/24",
        "AvailabilityZone" : { "Fn::Select" : [ "0", { "Fn::GetAZs" : "" } ]
}
      }
    },
    "PublicSubnet2" : {
      "Type" : "AWS::EC2::Subnet",
      "Properties" : {
        "VpcId" : { "Ref" : "VPC" },
        "CidrBlock" : "10.0.10.0/24",
        "AvailabilityZone" : { "Fn::Select" : [ "1", { "Fn::GetAZs" : "" } ]
}
      }
    },
```

```
    "PrivateSubnet1" : {
      "Type" : "AWS::EC2::Subnet",
      "Properties" : {
        "VpcId" : { "Ref" : "VPC" },
        "CidrBlock" : "10.0.1.0/24",
        "AvailabilityZone" : { "Fn::Select" : [ "0", { "Fn::GetAZs" : "" } ]
}
      }
    },

    "PrivateSubnet2" : {
      "Type" : "AWS::EC2::Subnet",
      "Properties" : {
        "VpcId" : { "Ref" : "VPC" },
        "CidrBlock" : "10.0.11.0/24",
        "AvailabilityZone" : { "Fn::Select" : [ "1", { "Fn::GetAZs" : "" } ]
}
      }
    },
```

Create Route Tables

We'll create two rules: one for the public subnet, and another for the private subnet. These will be referenced in just a moment…

Depending on how you set up your VPC, there can be as few as one route table, or as many as one per subnet.

```
    "PublicRouteTable" : {
      "Type" : "AWS::EC2::RouteTable",
      "Properties" : {
        "VpcId" : {"Ref" : "VPC"}
      }
    },

    "PrivateRouteTable" : {
      "Type" : "AWS::EC2::RouteTable",
      "Properties" : {
        "VpcId" : {"Ref" : "VPC"}
      }
    },
```

Define Routes

Next, define the routes that each subnet type (public or private) will use to communicate with the Internet. Note that the public subnet communicates directly with the Internet Gateway, while the private subnet communicates with the (yet-to-be-defined) NAT server, which is a router which permits outbound traffic, but that rejects inbound traffic except for responses to the outbound requests.

```
    "PublicRoute" : {
      "Type" : "AWS::EC2::Route",
      "Properties" : {
        "RouteTableId" : { "Ref" : "PublicRouteTable" },
        "DestinationCidrBlock" : "0.0.0.0/0",
        "GatewayId" : { "Ref" : "InternetGateway" }
      }
```

```
},

"PrivateRoute" : {
  "Type" : "AWS::EC2::Route",
  "Properties" : {
    "RouteTableId" : { "Ref" : "PrivateRouteTable" },
    "DestinationCidrBlock" : "0.0.0.0/0",
    "InstanceId" : { "Ref" : "NAT" }
  }
},
```

Map the Subnets to the Route Tables

The first of several details that we need to specify:

```
"PublicSubnetRouteTableAssociation1" : {
  "Type" : "AWS::EC2::SubnetRouteTableAssociation",
  "Properties" : {
    "SubnetId" : { "Ref" : "PublicSubnet1" },
    "RouteTableId" : { "Ref" : "PublicRouteTable" }
  }
},

"PublicSubnetRouteTableAssociation2" : {
  "Type" : "AWS::EC2::SubnetRouteTableAssociation",
  "Properties" : {
    "SubnetId" : { "Ref" : "PublicSubnet2" },
    "RouteTableId" : { "Ref" : "PublicRouteTable" }
  }
},

"PrivateSubnetRouteTableAssociation1" : {
  "Type" : "AWS::EC2::SubnetRouteTableAssociation",
  "Properties" : {
    "SubnetId" : { "Ref" : "PrivateSubnet1" },
    "RouteTableId" : { "Ref" : "PrivateRouteTable" }
  }
},
"PrivateSubnetRouteTableAssociation2" : {
  "Type" : "AWS::EC2::SubnetRouteTableAssociation",
  "Properties" : {
    "SubnetId" : { "Ref" : "PrivateSubnet2" },
    "RouteTableId" : { "Ref" : "PrivateRouteTable" }
  }
},
```

Create Network ACLs

You need to create network ACLs, and then drill down to create rules – and finally associate them with subnets.

```
"PublicSubnetAcl" : {
  "Type" : "AWS::EC2::NetworkAcl",
  "Properties" : {
    "VpcId" : {"Ref" : "VPC"}
  }
},
```

```
"PrivateSubnetAcl" : {
  "Type" : "AWS::EC2::NetworkAcl",
  "Properties" : {
    "VpcId" : {"Ref" : "VPC"}
  }
},
```

Create Network ACL Rules

Following are rules that make the network "wide open". In fact, the network is controlled by routing rules and (mostly) security groups. We strongly recommend that you do not use additional ACL rules, because it is very expensive in performance terms to inspect each packet this way.

There is a pair of ACL entries: one for ingress, and one for egress. It is perfectly acceptable to apply one pair of rules to all four subnets; we are setting this up in a more granular way – or at least so that you can be more granular if you need to.

```
"PublicInSubnetAclEntry" : {
  "Type" : "AWS::EC2::NetworkAclEntry",
  "Properties" : {
    "NetworkAclId" : {"Ref" : "PublicSubnetAcl"},
    "RuleNumber" : "32000",
    "Protocol" : "-1",
    "RuleAction" : "allow",
    "Egress" : "false",
    "CidrBlock" : "0.0.0.0/0",
    "Icmp" : { "Code" : "-1", "Type" : "-1"},
    "PortRange" : {"From" : "1", "To" : "65535"}
  }
},

"PublicOutSubnetAclEntry" : {
  "Type" : "AWS::EC2::NetworkAclEntry",
  "Properties" : {
    "NetworkAclId" : {"Ref" : "PublicSubnetAcl"},
    "RuleNumber" : "32000",
    "Protocol" : "-1",
    "RuleAction" : "allow",
    "Egress" : "true",
    "CidrBlock" : "0.0.0.0/0",
    "Icmp" : { "Code" : "-1", "Type" : "-1"},
    "PortRange" : {"From" : "1", "To" : "65535"}
  }
},

"PrivateInSubnetAclEntry" : {
  "Type" : "AWS::EC2::NetworkAclEntry",
  "Properties" : {
    "NetworkAclId" : {"Ref" : "PrivateSubnetAcl"},
    "RuleNumber" : "32000",
    "Protocol" : "-1",
    "RuleAction" : "allow",
    "Egress" : "false",
    "CidrBlock" : "0.0.0.0/0",
    "Icmp" : { "Code" : "-1", "Type" : "-1"},
    "PortRange" : {"From" : "1", "To" : "65535"}
  }
```

```
},

"PrivateOutSubnetAclEntry" : {
  "Type" : "AWS::EC2::NetworkAclEntry",
  "Properties" : {
    "NetworkAclId" : {"Ref" : "PrivateSubnetAcl"},
    "RuleNumber" : "32000",
    "Protocol" : "-1",
    "RuleAction" : "allow",
    "Egress" : "true",
    "CidrBlock" : "0.0.0.0/0",
    "Icmp" : { "Code" : "-1", "Type" : "-1"},
    "PortRange" : {"From" : "1", "To" : "65535"}
  }
},
```

Associate the ACLs with Subnets

This is starting to sound pedantic; however you must wire all the parts together.

```
"PublicSubnetAclAssocation1" : {
  "Type" : "AWS::EC2::SubnetNetworkAclAssociation",
  "Properties" : {
    "SubnetId" : { "Ref" : "PublicSubnet1" },
    "NetworkAclId" : {"Ref" : "PublicSubnetAcl"}
  }
},

"PublicSubnetAclAssocation2" : {
  "Type" : "AWS::EC2::SubnetNetworkAclAssociation",
  "Properties" : {
    "SubnetId" : { "Ref" : "PublicSubnet2" },
    "NetworkAclId" : {"Ref" : "PublicSubnetAcl"}
  }
},

"PrivateSubnetAclAssocation1" : {
  "Type" : "AWS::EC2::SubnetNetworkAclAssociation",
  "Properties" : {
    "SubnetId" : { "Ref" : "PrivateSubnet1" },
    "NetworkAclId" : {"Ref" : "PrivateSubnetAcl"}
  }
},

"PrivateSubnetAclAssocation2" : {
  "Type" : "AWS::EC2::SubnetNetworkAclAssociation",
  "Properties" : {
    "SubnetId" : { "Ref" : "PrivateSubnet2" },
    "NetworkAclId" : {"Ref" : "PrivateSubnetAcl"}
  }
},
```

Create a NAT Server

There is one remaining piece of work before this template will work, with some additional security groups that we can define later. Remember that the NAT is a virtual appliance that acts as a secure

outbound router that servers on private subnets can use to facilitate outbound calls to the Internet in order to download updates, etc.

We will define the NAT as an instance in Public Subnet 1, with security group rules that are appropriate to its mission.

Because a NAT is an instance, it is also a single point of failure (SPOF). You might want to consider strategies to monitor and replace this instance if it becomes problematic.

Our initial definition for the NAT references other resources that still need to be created.

```
    "NAT" : {
      "Type" : "AWS::EC2::Instance",
      "Properties" : {
        "InstanceType" : { "Ref" : "NATInstanceType" },
        "ImageId" : { "Fn::FindInMap" : [ "AWSRegionArch2AMI", { "Ref" :
"AWS::Region" },
                      { "Fn::FindInMap" : [ "AWSInstanceType2Arch", { "Ref" :
"NATInstanceType" }, "Arch" ] } ] },
        "SubnetId" : { "Ref" : "PublicSubnet1" },
        "SourceDestCheck" : "false",
        "DisableApiTermination" : "true",
        "SecurityGroupIds" : [ { "Ref" : "NATSecurityGroup" } ],
        "Tags" : [
          { "Key" : "Name", "Value" : { "Fn::Join" : ["", [
              "NAT-", { "Ref" : "VPC"} ]] } }
        ]
      }
    },
```

Assign an Elastic IP Address to the NAT

In order for the NAT to communicate with the Internet, it needs a public IP address. This address will be allocated and assigned by CloudFormation via the following snippet:

```
    "NATIP" : {
      "Type" : "AWS::EC2::EIP",
      "Properties" : {
        "Domain" : "vpc",
        "InstanceId" : { "Ref" : "NAT" }
      }
    },
```

Define a NAT security Group

Instances require security groups that will define which ports are open or closed. If you fail to define—and reference—a group, then the default group will be used. That is most definitely not the group that you want to assign to something as sensitive as the NAT server!

Note that this Security Group will only accept traffic from members of the Private Security Group, so you will need to define that group too.

```
   "NATSecurityGroup" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "GroupDescription" : "NAT Security Group",
```

```
        "VpcId" : {"Ref" : "VPC"},
        "SecurityGroupIngress" : [
          { "IpProtocol" : "-1", "FromPort" : "1", "ToPort" : "65535",
"SourceSecurityGroupId" : { "Ref" : "PrivateSG" } },
          { "IpProtocol" : "icmp", "FromPort" : "-1", "ToPort" : "-1",
"SourceSecurityGroupId" : { "Ref" : "PrivateSG" } }
        ]
      }
    },
```

Define the Private Security Group

This security group is typically one of several that are applied to servers in the Private subnets. We'll come back to it later and further restrict who is able to access servers in this group.

```
    "PrivateSG" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "GroupDescription" : "Servers in the Private Subnets",
        "VpcId" : {"Ref" : "VPC"},
        "SecurityGroupIngress" : [
          { "IpProtocol" : "tcp", "FromPort" : "22", "ToPort" : "22",
"CidrIp" : "0.0.0.0/0" },
          { "IpProtocol" : "tcp", "FromPort" : "3389", "ToPort" : "3389",
"CidrIp" : "0.0.0.0/0" },
          { "IpProtocol" : "icmp", "FromPort" : "-1", "ToPort" : "-1",
"CidrIp" : "0.0.0.0/0" }
        ]
      }
    }
```

Add an Output Section

Very useful information about the environment belongs in this section. A maximum of 10 output parameters are allowed, which seems small for a busy VPC. Fortunately CloudFormation allows you to concatenate strings to group the output into logical blocks of information.

```
"Outputs" : {

    "VPC" : {
      "Description" : "VPC",
      "Value" : {"Ref" : "VPC"}
    },

    "PublicSubnets" : {
      "Description" : "Public Subnets",
      "Value" : { "Fn::Join":["", [
            {"Ref" : "PublicSubnet1" }, ", ",
            {"Ref" : "PublicSubnet2" }
        ]]}
    },

    "PrivateSubnets" : {
      "Description" : "Private Subnets",
      "Value" : { "Fn::Join":["", [
            {"Ref" : "PrivateSubnet1" }, ", ",
```

```
            {"Ref" : "PrivateSubnet2" }
        ]]}
    }

}
```

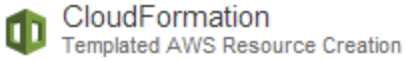
Try Launching the Template

There is a lot of code in this template, so we posted a ready-to-go version at

http://us-east-1-aws-training.s3.amazonaws.com/self-paced-lab-15/VPC1.json

You can either download and launch locally, or launch the template directly from Amazon CloudFormation.

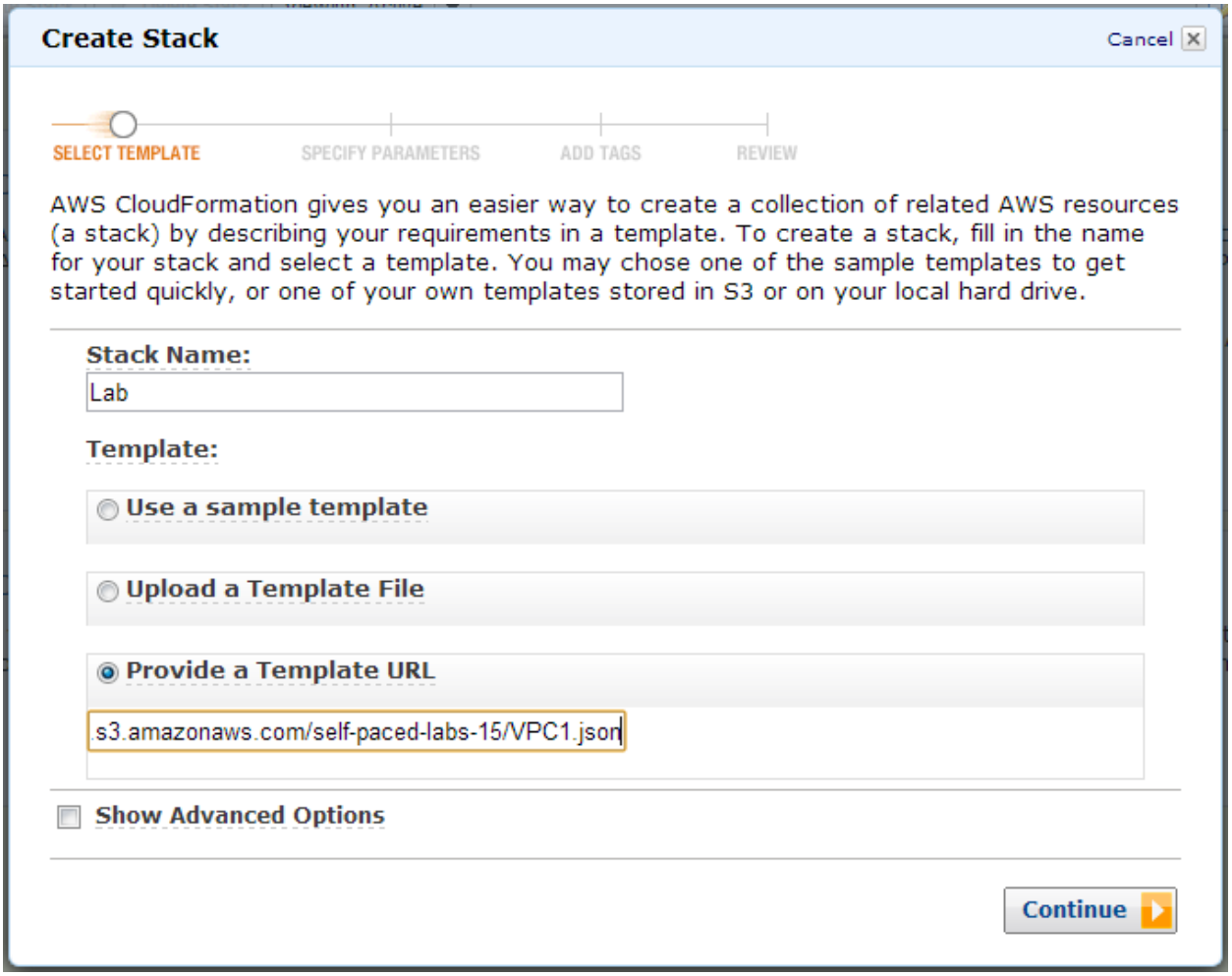
In the AWS Console, navigate to the CloudFormation section.



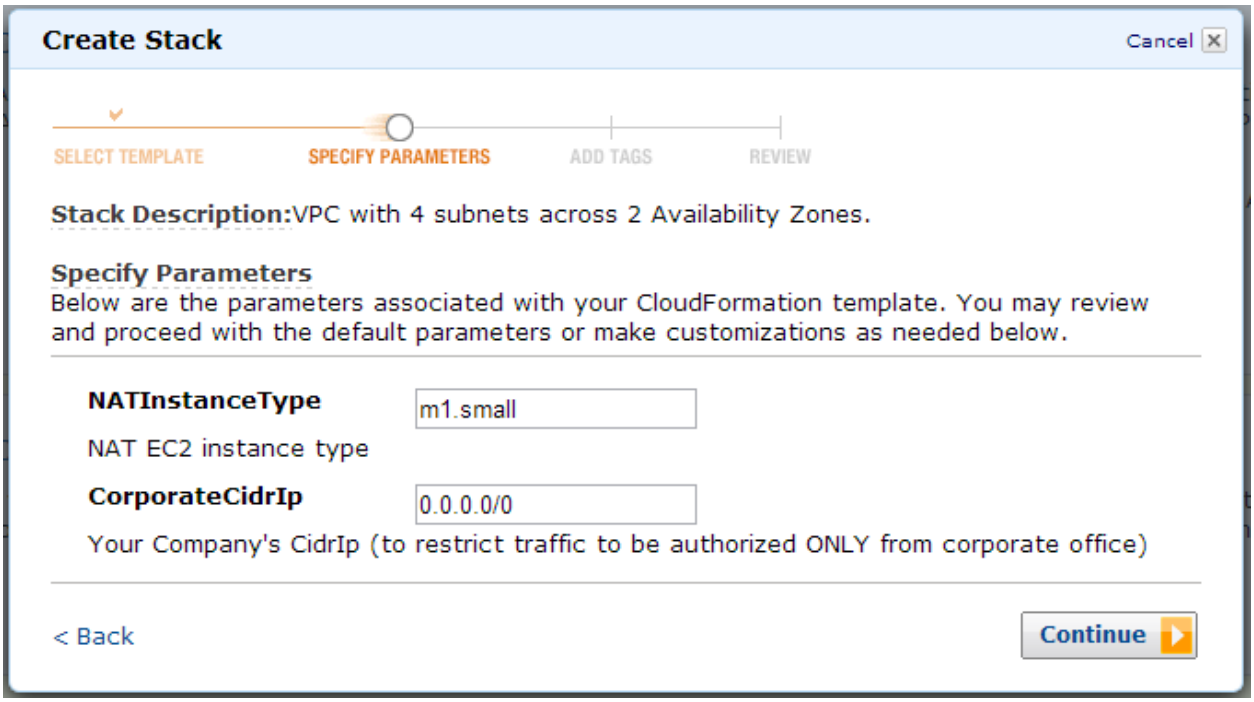
Click on "Create New Stack".

Name the stack, and then either upload your template file from your local hard drive, or else reference the copy that we created for you in Amazon S3:

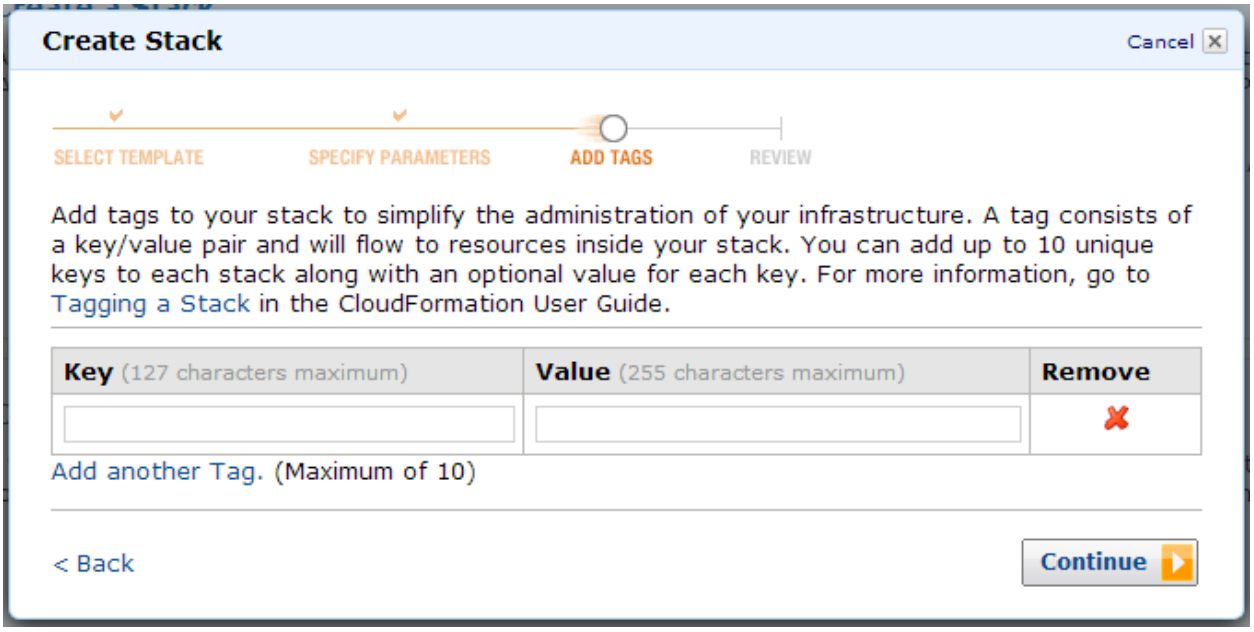
http://us-east-1-aws-training.s3.amazonaws.com/self-paced-lab-15/VPC1.json



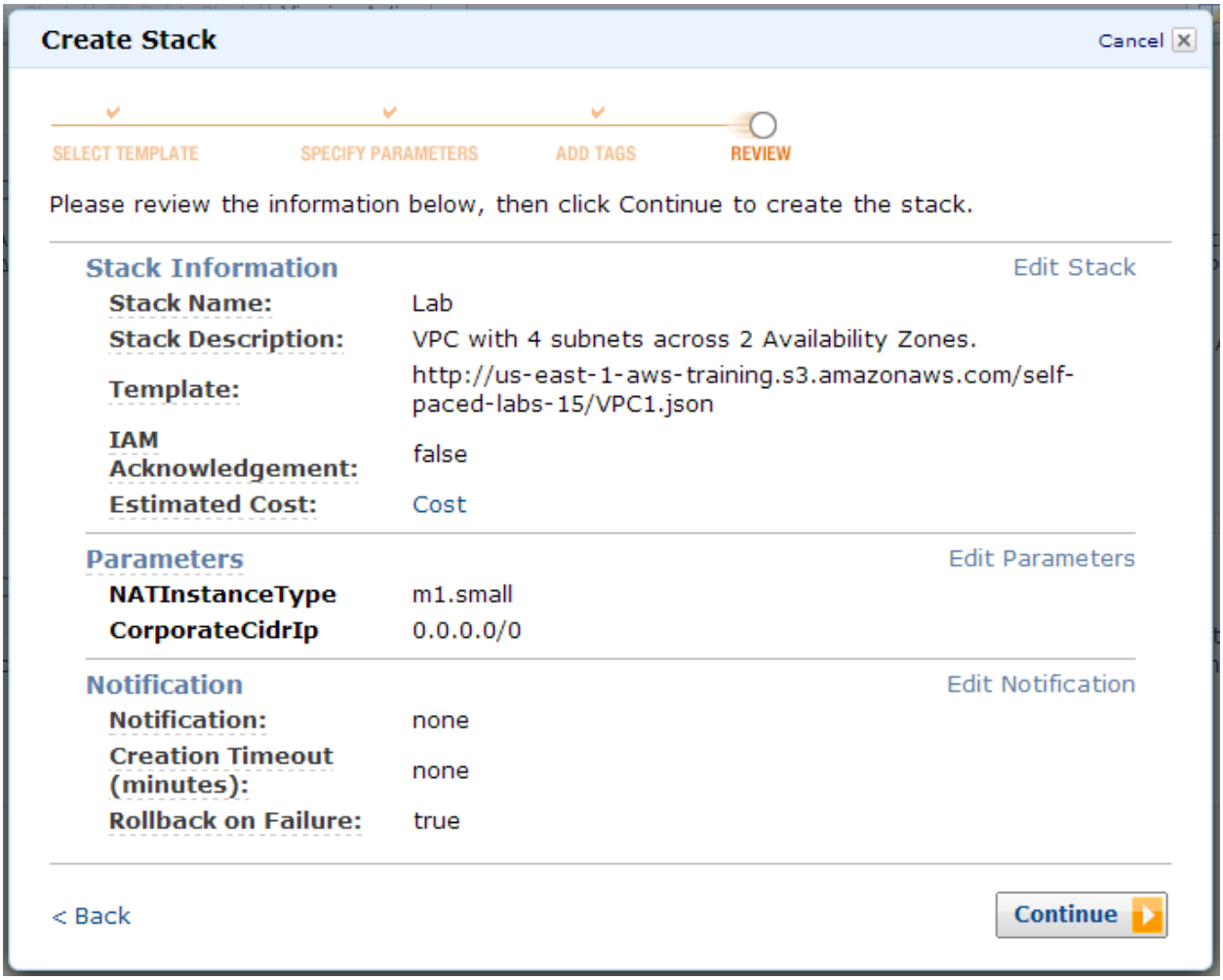
If any parameters need to be changed, this is your opportunity to do so



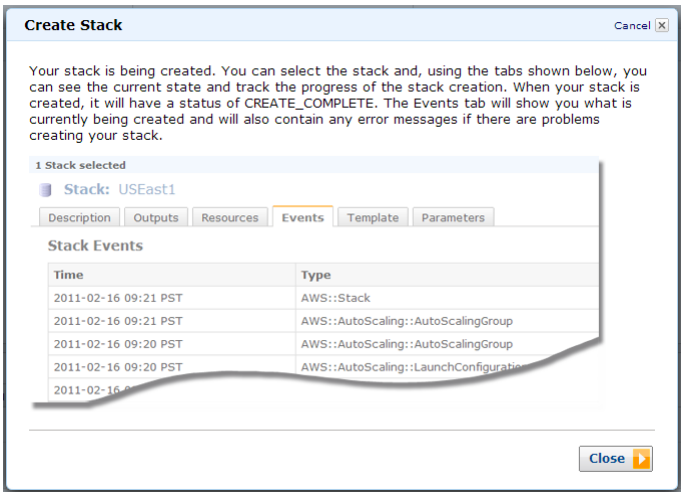
Optionally add tags that will propagate to resources created in your CloudFormation stack.



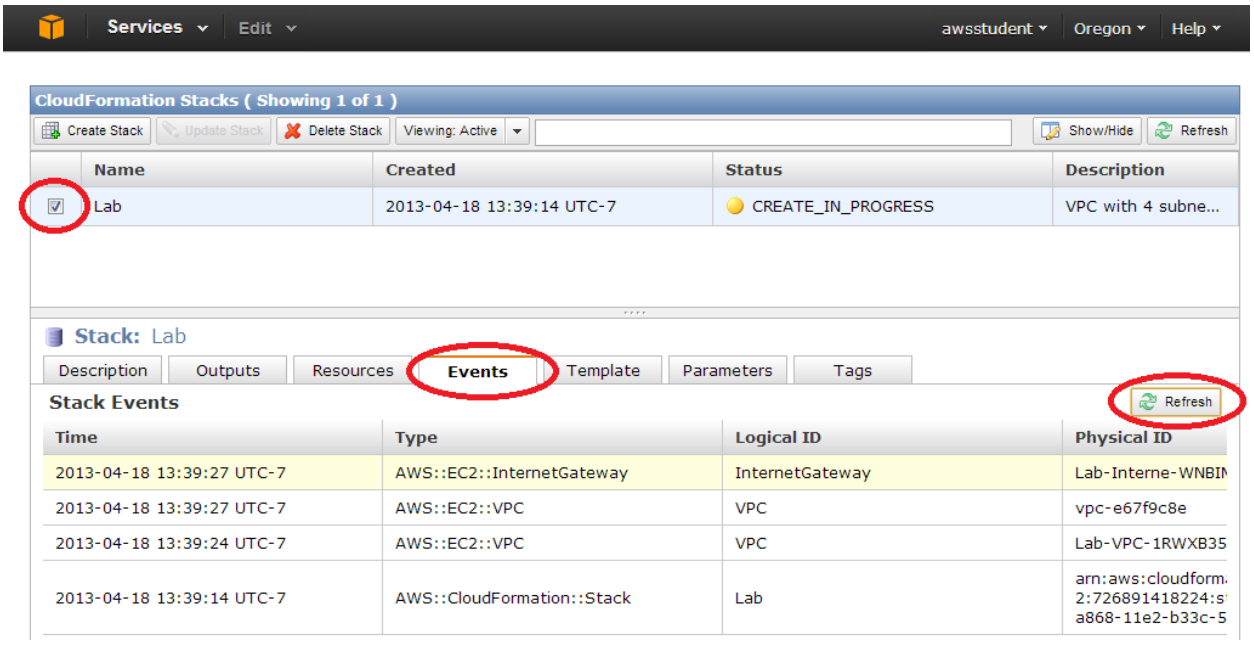
Review the parameters and continue.



Click “Close”.



Select the CloudFormation stack, and click on Events. You can periodically click Refresh in order to update the console with the latest status. Note: the console does not self-refresh for CloudFormation, although it does for some other services.



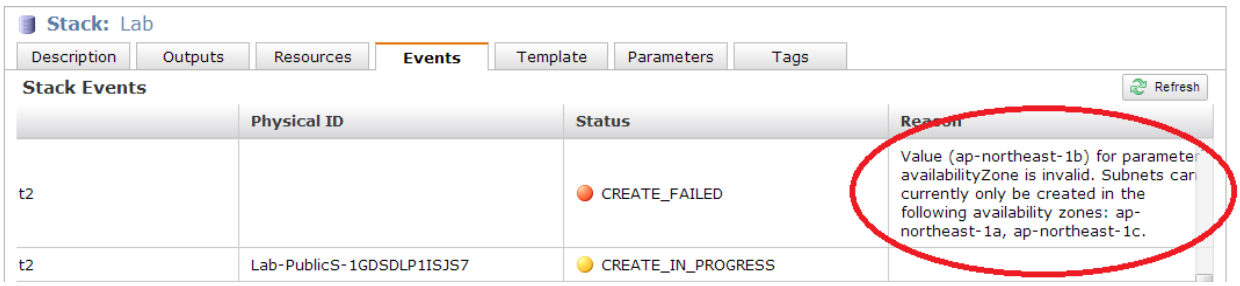
Inspect the Results

Switch to the VPC service area of the AWS Management Console. Here you'll see all the elements that you described in the template. You could even create a second VPC, with assurance that it will be an exact replica of the first VPC!

Troubleshooting

In certain regions (such as ap-northeast-1 at time of writing), VPC subnets might not be available in all Availability Zones. The template uses Fn::Select to arbitrarily select the first and second Availability Zones in the list which might result in selection of an Availability Zone that does not allow creation of a VPC subnet.

As pictured below, this might result in an error message such as "*Value (ap-northeast-1b) for parameter availabilityZone is invalid. Subnets can currently only be created in the following availability zones: ap-northeast-1a, ap-northeast-1c.*"

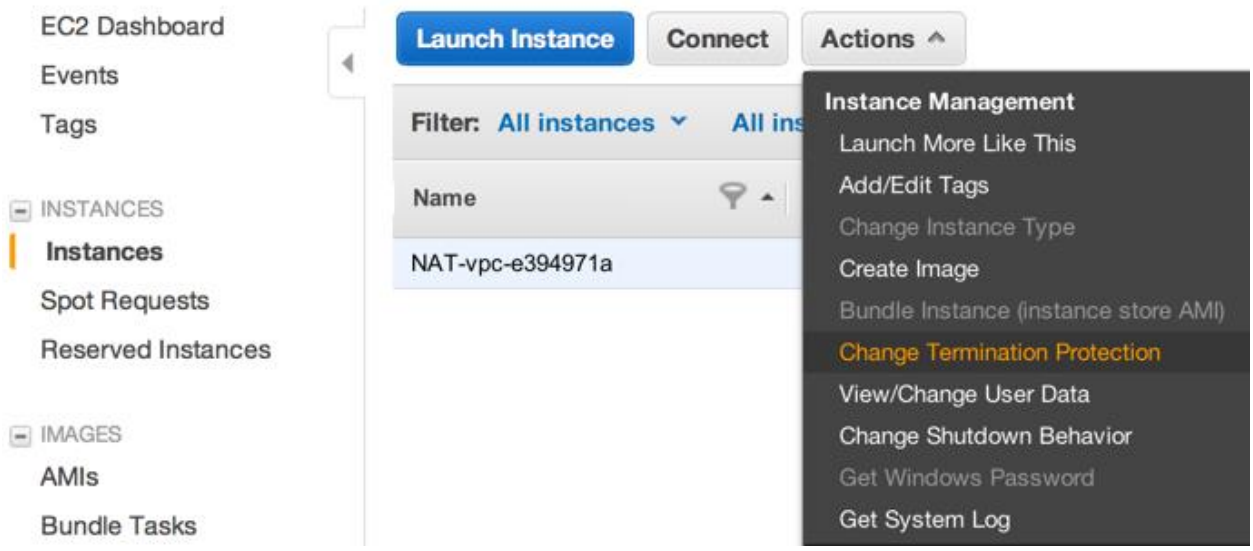


If stack creation fails for this reason, the Fn::Select indexes can be changed to select alternate Availability Zones, or the Availability Zones can be added to the template as user-specified input parameters and referenced using the Ref keyword.

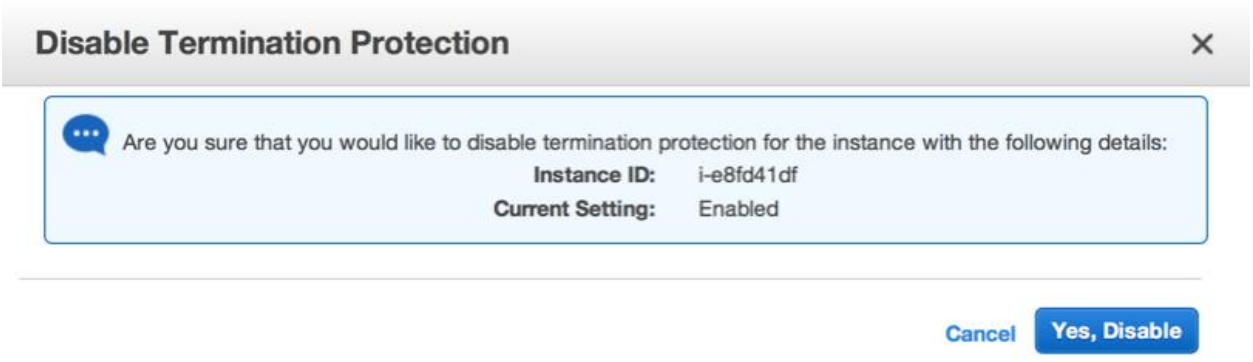
Delete the Stack

In a few minutes the stack will report "Create Complete". You could click on "Delete Stack", but the delete will fail because the NAT instance has Termination Protection enabled.

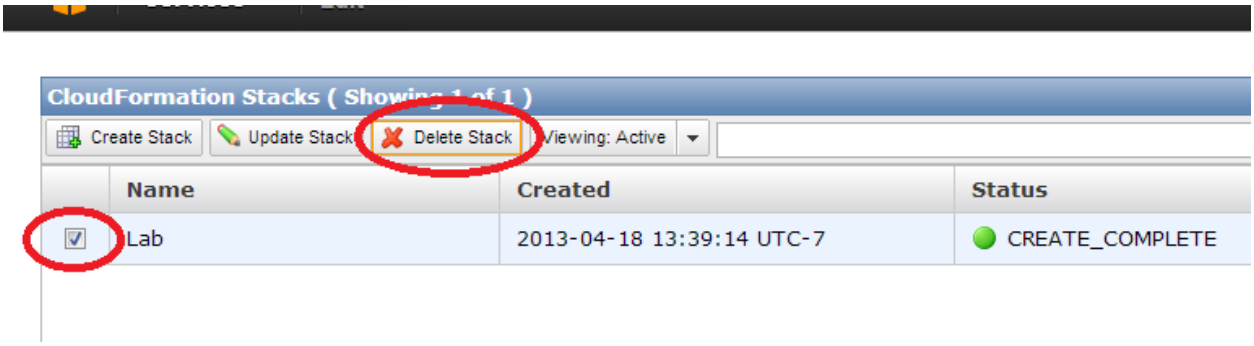
Switch to the EC2 section of the AWS Management Console and disable Termination Protection for the NAT. Select the appropriate instance (likely, the only instance in your console), and then right-click to display the menu shown in the screen shot below.



Choose Change Termination Protection and click Yes, Disable in the dialog.



Now navigate back to CloudFormation and delete the stack.



Add More Security Groups

The remainder of this workbook is a walk-through of various security groups that make sense in an enterprise environment. There are several core tenets in the way that these groups are constructed:

- VPC permits inbound and outbound rules. We are focusing on inbound rules only; so if you want to be ultra conservative then you will need to add outbound rules on your own.
- Earlier in this manual we mentioned that Network ACLs are expensive from a performance point of view. What we didn't mention is that they are also confusing, because now you have rules in multiple places. Accordingly almost all of our rules are in Security Groups. The single exception is that there are Routing Rules that apply to each subnet (to define Internet access).
- Remember that EC2 instances in the VPC can have multiple security groups associated with them. A logical example of this is a server that is a member of the Private Security Group and also the Domain Controller Security Group.
- There are two key ways to restrict a security groups in a manner that limits who can connect to an instance.
    - The most common way is by source IP address, with 0.0.0.0/0 meaning "anyone"
    - You can also restrict traffic to packets flowing from another security group. You'll see that technique here, when we specify that only Bastion Security Group members are allowed to connect via SSH or RDP to certain servers.

Create a Bastion Security Group

Bastion servers sit in the public subnet(s), and act as a door in to the back end servers. They are obviously a valuable chess piece, and deserve protection. In this case, we're assuming that anyone in the Corporate CIDR block is allowed access.

```
    "BastionSG" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "GroupDescription" : "Bastion access from Corporate Network",
        "VpcId" : {"Ref" : "VPC"},
        "SecurityGroupIngress" : [
          { "IpProtocol" : "tcp", "FromPort" : "22", "ToPort" : "22",
"CidrIp" : { "Ref" : "CorporateCidrIp" } },
          { "IpProtocol" : "tcp", "FromPort" : "3389", "ToPort" : "3389",
"CidrIp" : { "Ref" : "CorporateCidrIp" } },
          { "IpProtocol" : "icmp", "FromPort" : "-1", "ToPort" : "-1",
"CidrIp" : { "Ref" : "CorporateCidrIp" } }
```

```
        ]
      }
    },
```

Rewrite the Private Security Group Rules

We defined the Private Security Group earlier; but now that there is a Bastion Security Group let's restrict SSH and RDP access to Bastion Servers only. With this change, you might even decide to place Bastion servers in one of the private subnets.

```
    "PrivateSG" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "GroupDescription" : "Servers in the Private Subnets",
        "VpcId" : {"Ref" : "VPC"},
        "SecurityGroupIngress" : [
          { "IpProtocol" : "tcp", "FromPort" : "22", "ToPort" : "22",
"SourceSecurityGroupId" : { "Ref" : "BastionSG" } },
          { "IpProtocol" : "tcp", "FromPort" : "3389", "ToPort" : "3389",
"SourceSecurityGroupId" : { "Ref" : "BastionSG" } },
          { "IpProtocol" : "icmp", "FromPort" : "-1", "ToPort" : "-1",
"SourceSecurityGroupId" : { "Ref" : "BastionSG" } }
        ]
      }
    },
```

Define a Windows Active Security Group Environment

Active Directory needs a variety of ports open. In the two groups below, things are setup as follows:

- The Domain Controller only accepts traffic from Domain Members, except that …
- Because the Domain Controller also needs to serve as a DNS server, it accepts DNS requests from anyone that can reach it.
- The Domain Member Security Group has no inbound rules whatsoever. We'll use multiple security groups to allow access as appropriate by specific server function.

```
    "DomainControllerSG" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "VpcId" : {"Ref" : "VPC"},
        "GroupDescription" : "Domain Controller",
        "SecurityGroupIngress" : [
           { "IpProtocol" : "icmp", "FromPort" : "-1", "ToPort" : "-1",
"CidrIp" : "0.0.0.0/0" },
          { "IpProtocol" : "tcp", "FromPort" : "53", "ToPort" : "53",
"CidrIp" : "0.0.0.0/0" },
          { "IpProtocol" : "udp", "FromPort" : "53", "ToPort" : "53",
"CidrIp" : "0.0.0.0/0"},
          { "IpProtocol" : "udp", "FromPort" : "123", "ToPort" : "123",
"SourceSecurityGroupId" : { "Ref" : "DomainMemberSG" } },
          { "IpProtocol" : "tcp", "FromPort" : "135", "ToPort" : "135",
"SourceSecurityGroupId" : { "Ref" : "DomainMemberSG" } },
          { "IpProtocol" : "tcp", "FromPort" : "464", "ToPort" : "464",
"SourceSecurityGroupId" : { "Ref" : "DomainMemberSG" } },
```

```
          { "IpProtocol" : "udp", "FromPort" : "464", "ToPort" : "464",
"SourceSecurityGroupId" : { "Ref" : "DomainMemberSG" } },
          { "IpProtocol" : "-1", "FromPort" : "49152", "ToPort" : "65535",
"SourceSecurityGroupId" : { "Ref" : "DomainMemberSG" } },
          { "IpProtocol" : "tcp", "FromPort" : "389", "ToPort" : "389",
"SourceSecurityGroupId" : { "Ref" : "DomainMemberSG" } },
          { "IpProtocol" : "udp", "FromPort" : "389", "ToPort" : "389",
"SourceSecurityGroupId" : { "Ref" : "DomainMemberSG" } },
          { "IpProtocol" : "tcp", "FromPort" : "636", "ToPort" : "636",
"SourceSecurityGroupId" : { "Ref" : "DomainMemberSG" } },
          { "IpProtocol" : "tcp", "FromPort" : "3268", "ToPort" : "3269",
"SourceSecurityGroupId" : { "Ref" : "DomainMemberSG" } },
          { "IpProtocol" : "tcp", "FromPort" : "88", "ToPort" : "88",
"SourceSecurityGroupId" : { "Ref" : "DomainMemberSG" } },
          { "IpProtocol" : "udp", "FromPort" : "88", "ToPort" : "88",
"SourceSecurityGroupId" : { "Ref" : "DomainMemberSG" } },
          { "IpProtocol" : "tcp", "FromPort" : "445", "ToPort" : "445",
"SourceSecurityGroupId" : { "Ref" : "DomainMemberSG" } },
          { "IpProtocol" : "-1", "FromPort" : "137", "ToPort" : "139",
"SourceSecurityGroupId" : { "Ref" : "DomainMemberSG" } },
          { "IpProtocol" : "tcp", "FromPort" : "1024", "ToPort" : "65535",
"SourceSecurityGroupId" : { "Ref" : "DomainMemberSG" } }
        ]
      }
    },

    "DomainMemberSG" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "VpcId" : { "Ref" : "VPC" },
        "GroupDescription" : "Domain Members",
        "SecurityGroupIngress" : []
      }
    }
```

Expand the Outputs Section

You'll need the ID of many of these VPC features in order to launch EC2 instances, etc. Here are some additional output parameters:

Enhance the Subnets Section

Keeping track of which subnet is in what Availability Zone is a real pain. Accordingly we're listing the Availability Zone with each subnet, as below.

```
    "PublicSubnets" : {
      "Description" : "Public Subnets",
      "Value" : { "Fn::Join":["", [
                  {"Ref" : "PublicSubnet1" }, " (in ", {"Fn::GetAtt" : [
"PublicSubnet1", "AvailabilityZone" ] }, "), ",
                  {"Ref" : "PublicSubnet2" }, " (in ", {"Fn::GetAtt" : [
"PublicSubnet2", "AvailabilityZone" ] }, ")"
        ]]}
    },

    "PrivateSubnets" : {
```

```
      "Description" : "Private Subnets",
      "Value" : { "Fn::Join":["", [
                  {"Ref" : "PrivateSubnet1" }, " (in ", {"Fn::GetAtt" : [
"PrivateSubnet1", "AvailabilityZone" ] }, "), ",
                  {"Ref" : "PrivateSubnet2" }, " (in ", {"Fn::GetAtt" : [
"PrivateSubnet2", "AvailabilityZone" ] }, ")"
        ]]}
    },
```

Add Security Groups

Let's add the various security group IDs too. You could, of course, add an almost endless quantity of data; however we are trying to find a balance between useful information and "just data".

```
    "PrivateSG" : {
      "Description" : "Private Security Group",
      "Value" : { "Ref" : "PrivateSG" }
    },

    "BastionSG" : {
      "Description" : "Bastion Security Group",
      "Value" : { "Ref" : "BastionSG" }
    },

      "DomainControllerSG" : {
      "Description" : "Domain Controller Security Group",
      "Value" : { "Ref" : "DomainControllerSG" }
    },

    "DomainMemberSG" : {
      "Description" : "Domain Member Security Group",
      "Value" : { "Ref" : "DomainMemberSG" }
    }
```

Try the New Template

You can launch this template from:

http://us-east-1-aws-training.s3.amazonaws.com/self-paced-lab-15/VPC2.json

The resulting VPC will be the same as before, except with additional security groups and also with new output parameters.

Summary

There are lots of component parts to a VPC template; however as you learned in this workbook, it's not difficult to create repeatable processes, no matter how complicated the overall structure.

From here, inspect the VPC in the AWS Management console, and then try launching servers in multiple subnets and security groups to test the rules.

But above all else, use this template as a recipe for your own templates.

End Lab

Sign-out of the AWS Management Console.

Click the End Lab button in *qwikLAB*™.

Give the lab a thumbs-up/down, or enter a comment and click Submit

