Linux Academy

Hands-On Training

Creating a Virtual Private Cloud with CloudFormation and Launching an EC2 Instance

Contents

Introduction

In this lab, your goal is to build an entire VPC, including subnets, internet gateways, and routing tables. After building the VPC, the CloudFormation template should then launch an EC2 instance into a public VPC and return the public IP address as output.

Goals

- Build a VPC
- Build a public subnet with an internet gateway route attached to it
- Launch an EC2 instance into the subnet
- Return the public IP address of the instance as output

Getting Started

Click **Start Lab** on the Linux Academy lab page, and then log into the AWS console using the provided link and credentials. We will be using AWS's CloudFormation service; however, before we proceed with deploying anything with CloudFormation, we need to build our stack.

We recommend building the template in a text document before moving it to CloudFormation.

Template Declaration

Although a template declaration is not required to build a JSON CloudFormation template, it allows for us to include a description of the template for management purposes, and will be included in this lab. 2010-09-09 is the latest and only version of the template at this time.

The following is the start of the CloudFormation template:

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "Building A VPC From Scratch With CloudFormation",
```

A comma is included after our description declaration because we are going to define additional sections of the template.

Resources Declaration

The resources section of a CloudFormation template is the only portion required to get the template to run. In the resources declaration, each component of the VPC will be built. An understanding of a VPC is very important for this lab, although we will walk through most concepts in the text. If you feel confident in

your abilities, we encourage you to review instead the listed components, read their linked references and try to create the template yourself. If not, each section will be added, part by part, below.

The following resource types will be used:

- AWS::EC2::VPC
- AWS::EC2::Subnet
- AWS::EC2::InternetGateway
- AWS::EC2::VPCGatewayAttachment
- AWS::EC2::RouteTable
- AWS::EC2::Route
- AWS::EC2::SubnetRouteTableAssociation
- AWS::EC2::NetworkAcl
- AWS::EC2::SubnetNetworkAclAssociation
- AWS::EC2::Instance

VPC

```
"Resources" : {

 "VPC" : {
  "Type" : "AWS::EC2::VPC",
  "Properties" : {
   "EnableDnsSupport" : "true",
   "EnableDnsHostnames" : "true",
   "CidrBlock" : "10.0.0.0/16",
   "Tags" : [
    { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName" } },
    { "Key" : "Network", "Value" : "Public" }
   ]
  }
 },
```

In the above declaration, **VPC** is the unique, logical name for the resource we're defining. All names must be unique, and the names used in this lab are intended to be descriptive. The properties included are *EnableDnsSupport*, which will allow Amazon's DNS server to resolve DNS hostnames to their IPs, *EnableDnsHostnames*, which means instances in the VPC will get DNS hostnames, and *CidrBlock*, which names the CIDR block the VPC should cover.

Tags are added to assist in identifying and categorizing resources. More information can be found here.

Public Subnet

```
"PublicSubnet" : {
 "Type" : "AWS::EC2::Subnet",
 "Properties" : {
  "VpcId" : { "Ref" : "VPC" },
  "CidrBlock" : "10.0.0.0/24",
  "Tags" : [
   { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName" } },
   { "Key" : "Network", "Value" : "Public" }
  ]
 }
},
```

The *VpcId* specifies the VPC ID in which the subnet belongs. Because we do not know the ID of our subnet, currently, we use the intrinsic ref function that allows us to pull that information at runtime.

Internet Gateway/VPC Gateway Attachment

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

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

For a public subnet to function, we need an Internet gateway, defined in the *InternetGateway* resource. We also need to assign the Internet gateway to a VPC, using the same *VpcId* property as before, this time in the **GatewayToInternet** resource, using the **AWS::EC2::VPCGatewayAttachment** resource type.

Route Table/Route to Internet/Subnet Route Table Association

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

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

"PublicSubnetRouteTableAssociation" : {
 "Type" : "AWS::EC2::SubnetRouteTableAssociation",
 "Properties" : {
  "SubnetId" : { "Ref" : "PublicSubnet" },
  "RouteTableId" : { "Ref" : "PublicRouteTable" }
 }
},
```

Again, we are referencing the VPC in the **PublicRouteTable** resource. This resource is what will create the routing table.

To connect to the Internet, a public subnet needs to have an Internet gateway and a route that will allow traffic to pass. To achieve this, we need to create a new route in the routing table. This is done with the **PublicRoute** resource above.

Because, by default, all subnets use the default routing table, we need to ensure our route is attached to the subnet and made public. Within the **PublicRoute** resource, the *RouteTableId* calls to the *PublicRouteTable* (defined above), the *DestinationCidrBlock* is the block used for a destination match, and the *GatewayId* is the ID of the Internet gateway attached to the VPC. In **PublicSubnetRouteTableAssociation**, *SubnetId* is simply the ID of the subnet, while *RouteTableId* is the same for the route table.

Instance

```
    "PublicInstance" : {
     "Type" : "AWS::EC2::Instance",
     "DependsOn" : "GatewayToInternet",
     "Properties" : {
      "InstanceType" : "t1.micro",
      "ImageId"  : "ami-fb8e9292",
      "NetworkInterfaces" : [{
       "AssociatePublicIpAddress" : "true",
       "DeviceIndex"           : "0",
       "DeleteOnTermination"     : "true",
       "SubnetId"              : { "Ref" : "PublicSubnet" }
      }]
     }
     }
}
```

Finally, we need to have our public instance launch inside of the VPC we just created. Using the resource type **AWS::EC2::Instance**, we then define that the resource *DependsOn* the **GatewayToInternet** resource. Additionally, the properties defined lay out the type of instance we want to launch, the ID of the image, and networking information. Within the *NetworkInterfaces* embedded property, we indicate through the *AssociatedPublicIpAddress* property that this network interface receives a public IP address. The *DeviceIndex* defines the interface's position in the attachment order, and *DeleteOnTermination* notes that that network interface should be deleted when the instance is terminated. The *SubnetId* works as in previous code blocks.

Save your template as **vpc.template**.

The Full Template

The full template can be viewed below. It is also available for download here.

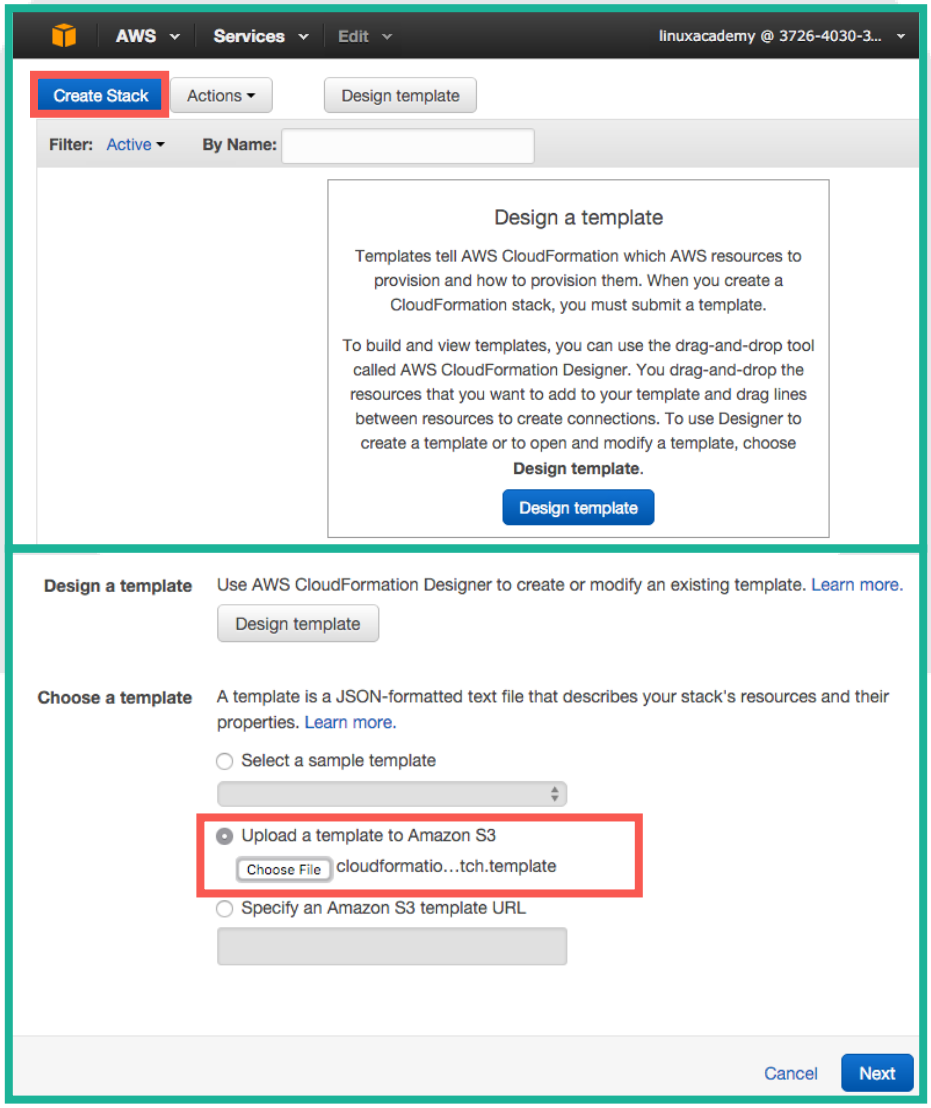
```
{
 "AWSTemplateFormatVersion" : "2010-09-09",

 "Description" : "Building A VPC From Scratch With CloudFormation",

 "Resources" : {


  "VPC" : {
   "Type" : "AWS::EC2::VPC",
   "Properties" : {
    "EnableDnsSupport" : "true",
    "EnableDnsHostnames" : "true",
    "CidrBlock" : "10.0.0.0/16",
    "Tags" : [
     { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName" } },
     { "Key" : "Network", "Value" : "Public" }
    ]
   }
  },

  "PublicSubnet" : {
   "Type" : "AWS::EC2::Subnet",
   "Properties" : {
    "VpcId" : { "Ref" : "VPC" },
    "CidrBlock" : "10.0.0.0/24",
    "Tags" : [
     { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName" } },
     { "Key" : "Network", "Value" : "Public" }
    ]
   }
  },

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

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

},

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

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

"PublicSubnetRouteTableAssociation" : {
  "Type" : "AWS::EC2::SubnetRouteTableAssociation",
  "Properties" : {
    "SubnetId" : { "Ref" : "PublicSubnet" },
    "RouteTableId" : { "Ref" : "PublicRouteTable" }
  }
},



"PublicInstance" : {
  "Type" : "AWS::EC2::Instance",
  "DependsOn" : "GatewayToInternet",
  "Properties" : {
    "InstanceType" : "t1.micro",
    "ImageId"  : "ami-fb8e9292",
```

```
        "NetworkInterfaces" : [{
         "AssociatePublicIpAddress" : "true",
         "DeviceIndex"           : "0",
         "DeleteOnTermination"   : "true",
         "SubnetId"              : { "Ref" : "PublicSubnet" }
        }]
      }
      }
  }
  }
```

Building the CloudFormation Stack

From the CloudFormation dashboard located in the AWS console, select **Create Stack**. From there, select the **Upload a template to Amazon S3** radio button, and upload the template you just created. Press **Next**.



Name your stack. We choose *vpc-from-scratch* as the name of our own. Select **Next**.

Here, you can define your tags for the resources in your stack. This is useful if you are using multiple stacks for various apps. Press **Next**, then press **Create**. You will be taken to your stack listing, where you can see your stack is being created.

Final Steps

When the stack has been created (the status will change to *CREATE_COMPLETE*), navigate to the VPC dashboard, and compare the work you have done in the template to the VPC located on the Your VPCs dashboard. Afterward, view the created EC2 instance to ensure the information there also matches up with your template settings.

When done, we encourage you to delete the stack and start over, using the remainder of the lab time to experiment with CloudFormation templates.