Slalom AWS Introduction Workshop

Lab 3 Instructions

Introduction

Welcome to **Lab 3** of the Slalom AWS Introduction Workshop! This document outlines 10 steps, walking you through the process of using CloudFormation to build and improve upon the architecture developed in Lab 1.

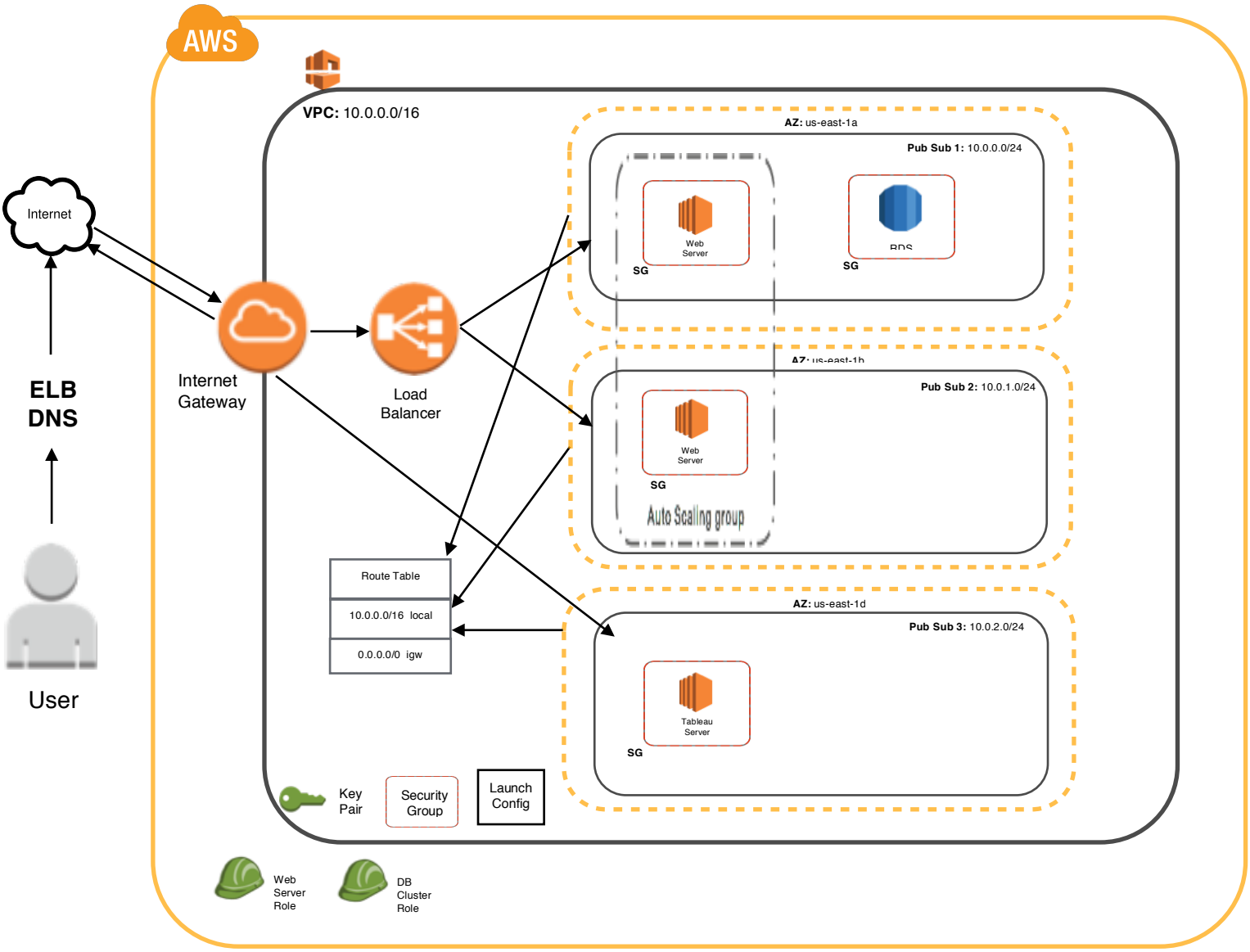
In this lab we will be focusing on a tool developed by AWS for automated infrastructure, **AWS CloudFormation**. Starting with a base template I have provided for you, we will walk through the steps required to add a few additional pieces including a new public subnet, an RDS instance and a Tableau server. Don't worry if things don't always go to plan, I have also provided templates to cover each step of the way!

At the end of each step you will find an architectural summary of the AWS resources that you have built so far.

The next two pages contain a brief summary of what is achieved within each step of the process. Please reach out to Bruce, Jeff, Todd or James for clarification on any of the content covered within this document!

The image at the top of the following page outlines the final architecture of our system following the completion of this lab. Notice the **new subnet** (Pub Sub 3), **RDS instance** (inside Pub Sub 1) and **Tableau Server** (inside Pub Sub 3).

Lab 3 - Final Architecture



Step Overview

Step 1: Logging in to the AWS Console

By now you should have signed up for your own AWS account. This step simply directs you to log in to your account and select the appropriate AWS Region.

Step 2: Setting Up Our Templates Directory and Introducing JSON

A zipped directory was provided to you in the email containing this instruction document. In this step we will set up a directory for the templates on your Desktop. CloudFormation templates are written in **J**ava**S**cript **O**bject **N**otation (JSON) format. This step also examines two important sections of a template and how JSON is used within a template.

Step 3: Getting Familiar with CloudFormation Designer

Amazon CloudFormation Designer is a useful way of visualizing the content within any AWS CloudFormation Template. We will use it throughout this lab to examine the resources that we've added as our architecture grows.

Step 4: Creation of Additional Public Subnet

Within a VPC, you can create subnets to logically divide the space within your VPC into smaller logical sections. Subnets can be both public (AWS resources created within are reachable over the public Internet) or private (AWS resources created within cannot be reached over the Internet). Here we are adding a third public subnet to our architecture for our Tableau server to live within, as well as the necessary associations to our public route table.

Step 5: DB Cluster Subnet Group and Security Group

When you create an RDS (Relational Database Service) instance, you need to set up a DB Subnet Group for it to live within. Within this step we are creating the DB subnet group for our future RDS instance and also also adding a dedicated security group for our RDS.

Step 6: Create an IAM Policy, Role and DB Cluster Parameter Group

AWS Identity and Access Management (IAM) is an AWS best practice that helps you to define what a resource can and cannot do in AWS. In our example, we will be creating an IAM Policy and Role for our future RDS Instance to adopt. This Policy/Role combination will allow the DB to load in a dataset stored in Amazon S3. The second part of this step, creating the DB Cluster Parameter Group, will allow us to ensure that our RDS Instance uses the newly created IAM Role when reading from S3.

Step 7: Create RDS Cluster and RDS Instance

In this step we are going to be creating an RDS Cluster running AWS's very own Aurora service and a single RDS Instance which lives within this cluster.

Step 8: Create a Tableau Server

Tableau is one of the more frequently employed tools for Business Intelligence (BI) purposes. Within this step, we are going to be taking a pre-configured version of Tableau server and launching it on an AWS EC2 instance. In addition, we will be creating a security group and ensuring the correct security rules are set.

Step 9: Accept Terms and Conditions in AWS Marketplace

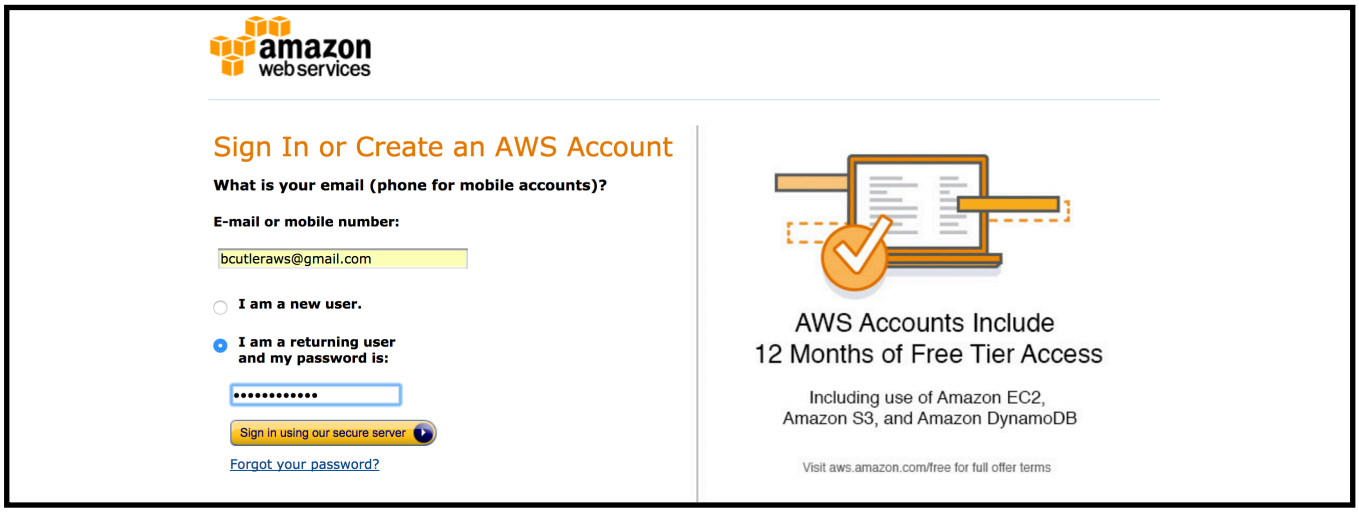
As we are using a pre-loaded version of Tableau server, we have to accept some Terms and Conditions in the AWS Marketplace before we can successfully launch our EC2 Tableau instance. As you will see, the AWS Marketplace is a way for you to launch EC2 instances with pre-loaded versions of a number of popular technologies.

Step 10: Launch our CloudFormation Template

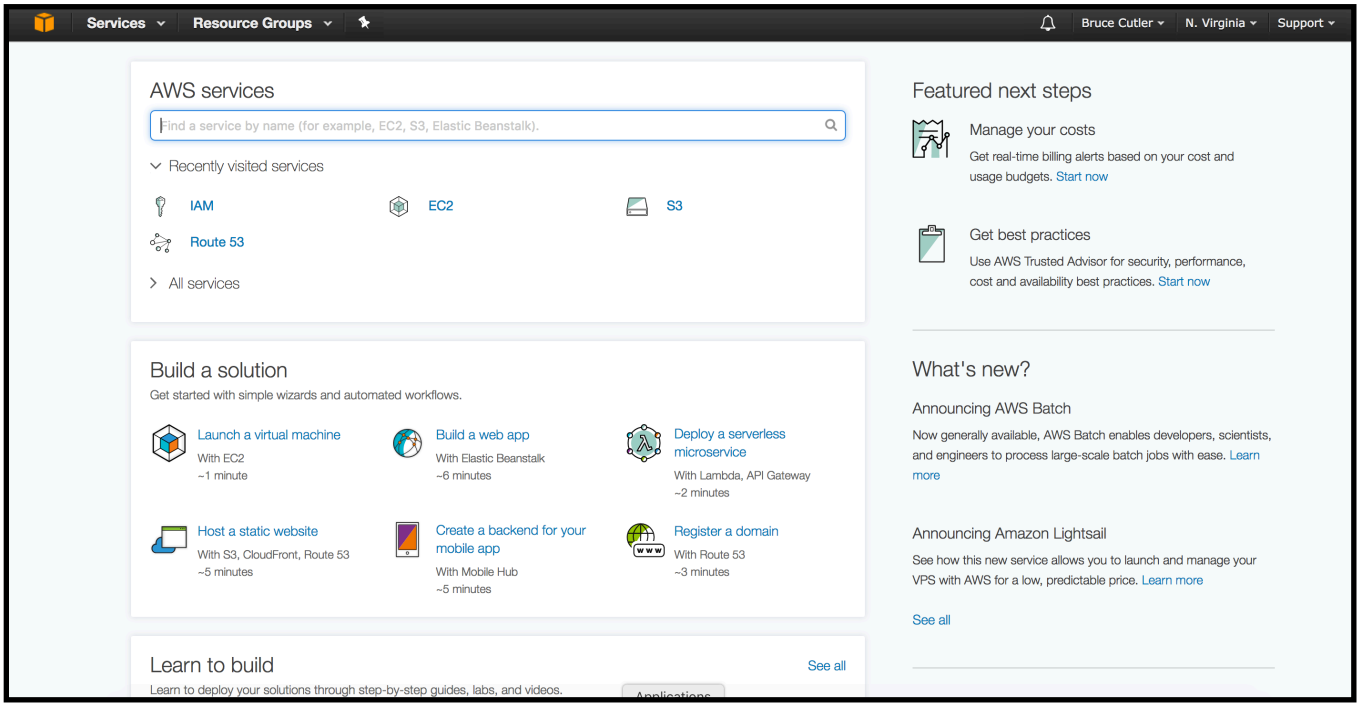
Every step up until now has been preparation for this step. At this point we are ready to launch our AWS CloudFormation Template to create a "Stack" of resources.

Step 1: Logging in to the AWS Console

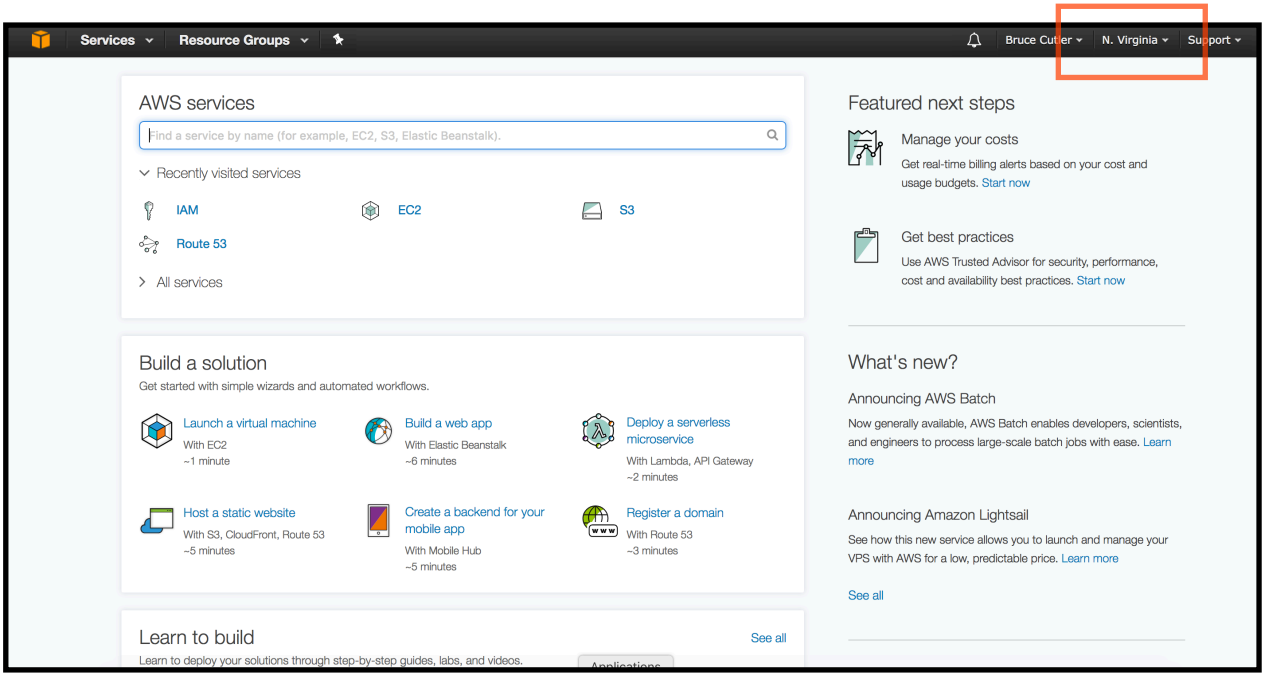
1. In a web browser, navigate to **https://console.aws.amazon.com/console/home**



2. Log in with the email / password combination you signed up for your account with. This will bring you to the AWS Console home page:



3. At the top right of the AWS Console screen, ensure that the **N. Virginia** is selected as your AWS Region



NOTE:
Before we begin, please make sure you have deleted all of the resources outlined within the document "**Slalom AWS Intro - Resource Deletion**" that I recently sent to the group.

As we start Lab 3, the only resource that should exist within your AWS Account is the EC2 KeyPair, which can be found by going to **Services —> EC2 —> Key Pairs**

Step 2: Setting Up Our Templates Directory and Introducing JSON

1. You should all have received a zipped directory (**slalom-templates.zip**) in the email with these instructions. Unzip that file to your Desktop, so that the **slalom-templates** directory lives there

2. Navigate into the **slalom-templates** directory. We want to create a copy of the **Lab3-Base.json** file to serve as our working copy. So copy and paste that file within your directory, giving it the name **Lab3-Working.json**

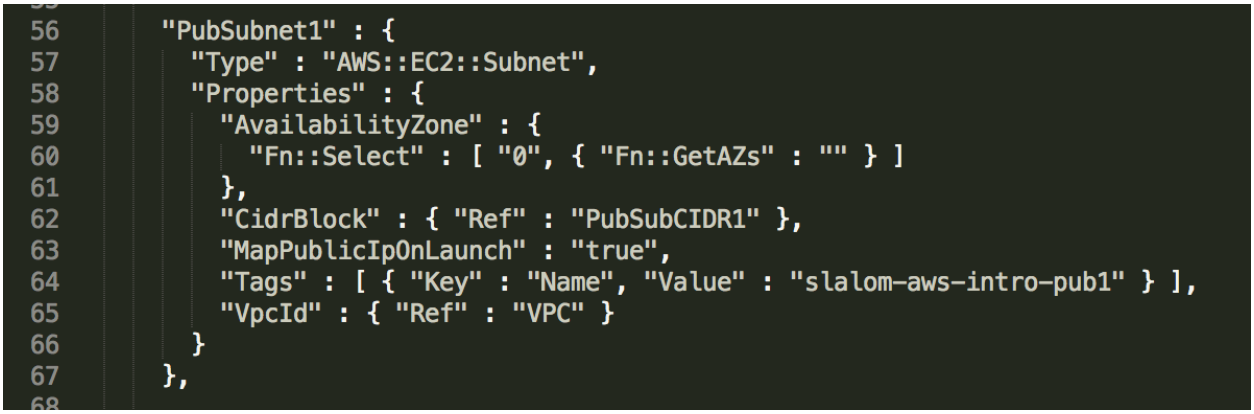
3. At this point, lets go ahead and open **Lab3-Working.json** in our text editor. If you haven't had a chance to download an editor yet, I recommend SublimeText2. With the file open, make sure that you have line numbers turned on so that you can identify where you are in the code.

4. Take a minute to scroll through the file. It might look like nonsense at this point, but this is a CloudFormation Template written in **JSON** notation (**J**ava**S**cript **O**bject **N**otation) that replicates the work that you performed in Lab 1.

5. Scroll down to **Line 9**. You should see the word **"Parameters"**. Parameters allow you to define variable like attributes that you can use in your CloudFormation template. **Lines 9-38** in **Lab3-Working.json** define parameters. We will give these parameters values when we run our template in Step 10.

6. Now Scroll down to **Line 46**. You should see the word **"Resources"**. The resources section of the template is where AWS infrastructure such as EC2 instances, security groups and load balancers are actually created. The resources section usually makes up the majority of a CloudFormation template.

7. Now lets take a closer look at an individual resource. For example, the creation of a Public Subnet between **lines 56-67**:



Examining the above image:

Line 56:
This is a name we choose to identify our subnet in the template. We can name resources whatever we would like, but there can be no duplicates inside a template. Think of the name as being like a Social Security Number for an AWS resource in your template. It makes sense to call them something that we can easily identify when we are trying to use them at a later stage in our template.

Line 57:
This is an important line as the **"Type"** property tells CloudFormation exactly what we are trying to create. In this case, we are telling CloudFormation to create a new Subnet.

Lines 58-67:
Line 58 opens the **"Properties"** section for our new subnet. Every resource has a number of properties that you can supply to help define it. In this case, our new Subnet needs to know properties such as where to be created (AvailabilityZone on Line 59), how big to be (CidrBlock on Line 62) and which VPC to live within (VpcId on Line 65)

You'll notice use of the word **"Ref"** within lines 62 and 65. **Ref** is a function that allows us to reference parameters (from lines 9-38) or other resources we may have already created (VPC on lines 47-54).

8. Hopefully the above has helped you to understand a little of how CloudFormation is put together. In the following steps we are going to make some additions to the **Lab3-Working.json** template.

   I will provide the code that you will need to add, explain what it's doing and let you know where it should be placed in your file. As you add the code in, take a moment to examine it to help you understand what's being added.

9. As a final note about JSON notation, be careful as you copy the code in! Ensure you pick up everything in the provided code snippets, particularly commas and brackets.

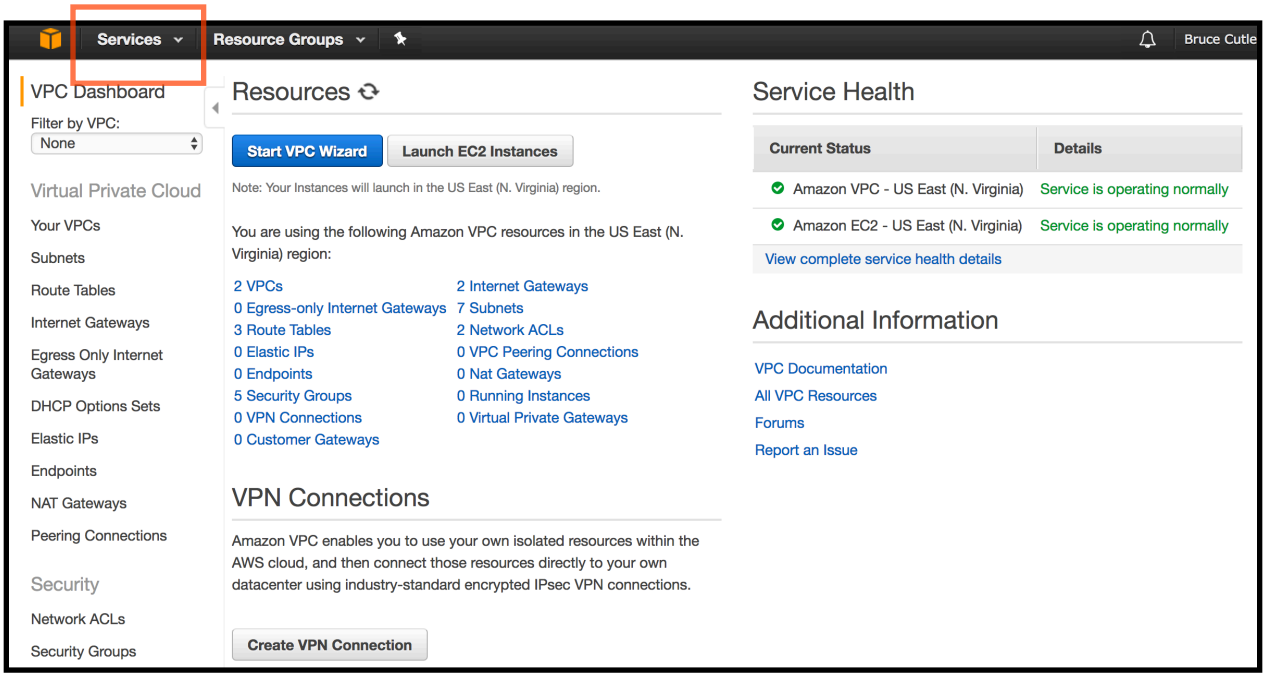
NOTE:

If at any stage of the Lab you need to catch up to the end of a step, you can simply copy the file with the matching name to Lab3-Working.json.

For example:

If I got a bit stuck with Step 5 and wanted to reboot with a fresh copy ready to start Step 6 then I would simply copy and paste **Lab3-Step5.json** and rename as **Lab3-Working.json**

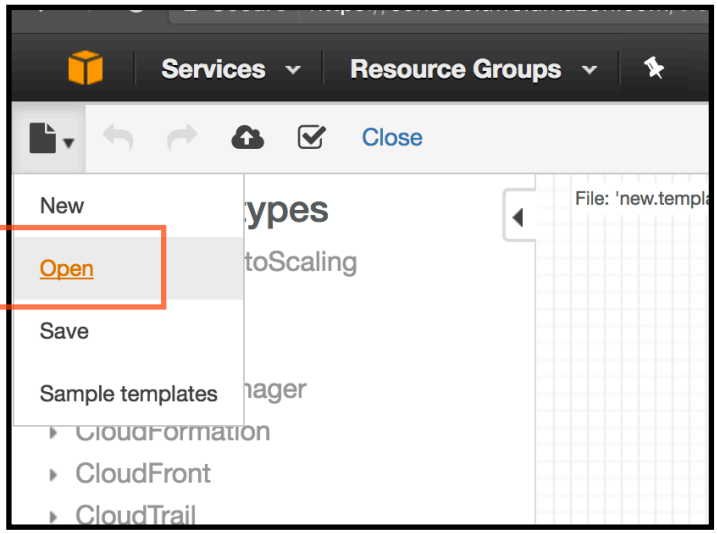
Step 3: Getting Familiar with the CloudFormation Designer



1. In the AWS Console, navigate to **Services —> CloudFormation:**

2. Click on the blue **Design Template** button towards the middle of the screen

3. At the top left of the screen, select the symbol that looks like a sheet of paper and click **Open**

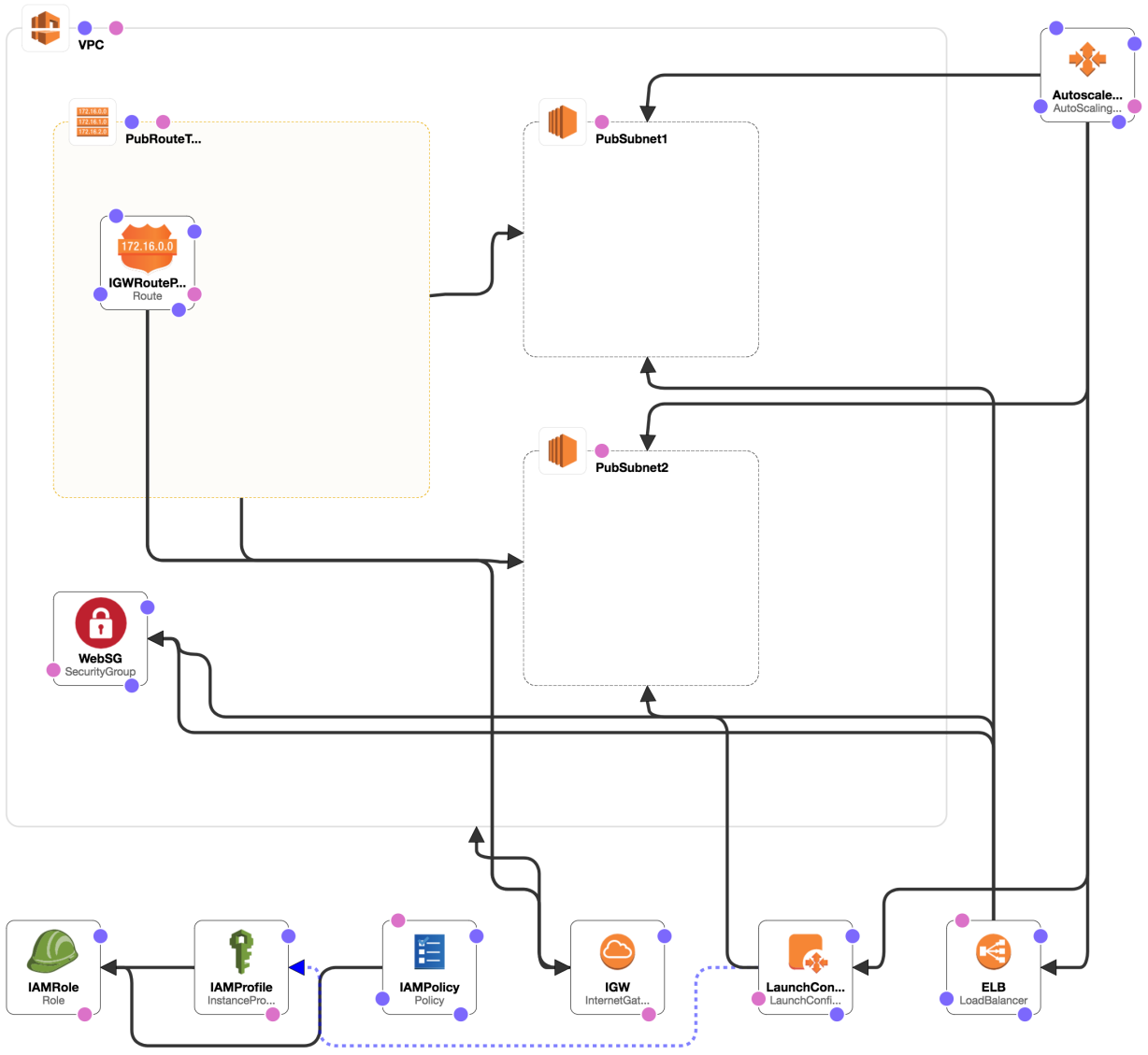


4. Select **Local File** and navigate to the **slalom-templates** directory on your Desktop

5. Select the **Lab3-Working.json** template and select Open

6. The following image should show up in the CloudFormation template designer:

CloudFormation Designer – Step 3



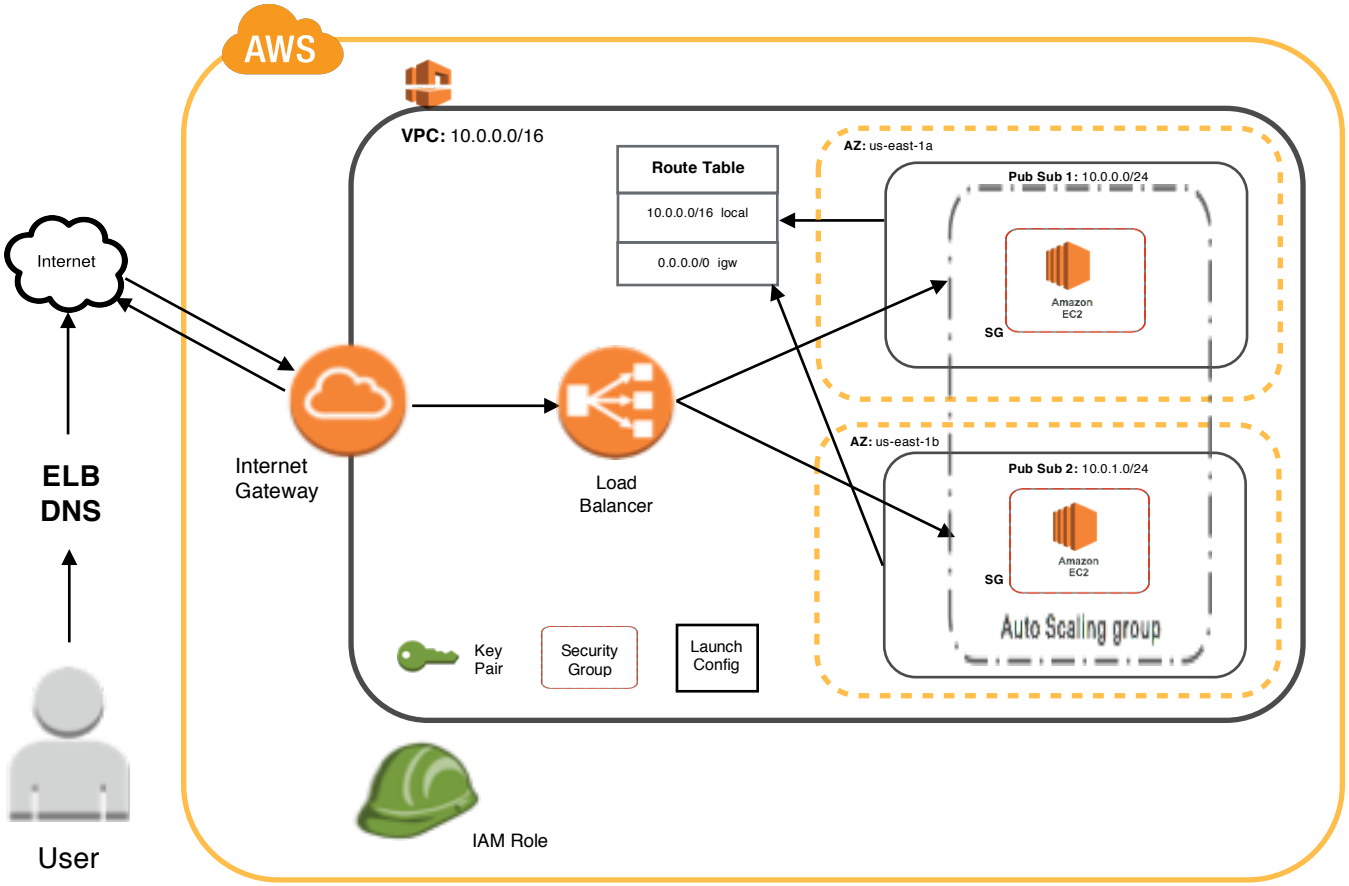
The above image is a visual representation of the file we were scrolling through in **Step 2**! Looking at some of the components, we see PubSubnets, WebSG and IAMRoles to name but a few resources.

7. In the next 5 steps you will be asked to navigate to the CloudFormation Designer to validate that what you have added to your CloudFormation template matches the image provided in the document.

   If it does, awesome!

   If it doesn't, don't worry! I have provided templates that put you in the correct place for every step of the way. As I mentioned in **Step 2**, simply copy and paste the template that represents the last step you were trying to complete and rename as **Lab3-Working.json**. If I got stuck on Step 5 and wanted a version with Step 5 complete ready to start Step 6, I'd simply copy/paste **Lab3-Step5.json** and rename as **Lab3-Working.json**

System Architecture: Step 3



Step 4: Creation of Additional Public Subnet

Within this step we are going to create a third public subnet. This public subnet will be used to host our Tableau server.

1. In order to help us define a subnet, we need to first **add a Parameter** that will be used to describe how big that subnet should be. Navigate to the “Parameters” section starting at **Line 9** in your **Lab3-Working.json** file

2. Click to the right of the comma on **Line 21** and press return

3. Starting on **Line 22**, copy and paste in the following code snippet:

```
"PubSubCIDR3" : {
  "Type" : "String",
  "Description" : "This is the range of addresses for PublicSubnet3"
},
```

4. Take a second to correct any alignment issues and remove any blank lines created after your new parameter. This should leave you with the following on **lines 22-25**:

```
18      "PubSubCIDR2" : {
19        "Type" : "String",
20        "Description" : "This is the range of addresses for PublicSubnet2"
21      },
22      "PubSubCIDR3" : {
23        "Type" : "String",
24        "Description" : "This is the range of addresses for PublicSubnet3"
25      },
26      "MyKeyPair" : {
27        "Type" : "AWS::EC2::KeyPair::KeyName"
28      },
```

5. Next, lets add in the resource for our new AWS Subnet. Moving to the “**Resources**” section of our template, jump to the end of **line 84** and press return twice

6. Starting on **line 86**, copy and paste the following code snippet:

```
"PubSubnet3" : {
 "Type" : "AWS::EC2::Subnet",
 "Properties" : {
  "AvailabilityZone" : {
   "Fn::Select" : [ "2", { "Fn::GetAZs" : "" } ]
  },
  "CidrBlock" : { "Ref" : "PubSubCIDR3" },
  "MapPublicIpOnLaunch" : "true",
  "Tags" : [ { "Key" : "Name", "Value" : "slalom-aws-intro-pub3" } ],
  "VpcId" : { "Ref" : "VPC" }
 }
},
```

7. Unlike parameters, for ease of resource identification I've chosen to leave a blank line between resources (lines 85 and 98 for example). This should leave you with the following on **lines 86-97:**

```
85
86      "PubSubnet3" : {
87        "Type" : "AWS::EC2::Subnet",
88        "Properties" : {
89          "AvailabilityZone" : {
90            "Fn::Select" : [ "2", { "Fn::GetAZs" : "" } ]
91          },
92          "CidrBlock" : { "Ref" : "PubSubCIDR3" },
93          "MapPublicIpOnLaunch" : "true",
94          "Tags" : [ { "Key" : "Name", "Value" : "slalom-aws-intro-pub3" } ],
95          "VpcId" : { "Ref" : "VPC" }
96        }
97      },
98
```

8. Finally, let's associate our new subnet with the same route table as our other two Public Subnets

9. Navigate to the end of line 136, press return twice and copy in the following on **line 138**:

```
"PubRtbAssoc3" : {
 "Type" : "AWS::EC2::SubnetRouteTableAssociation",
 "Properties" : {
  "RouteTableId" : { "Ref" : "PubRouteTable1" },
  "SubnetId" : { "Ref" : "PubSubnet3" }
 }
},
```

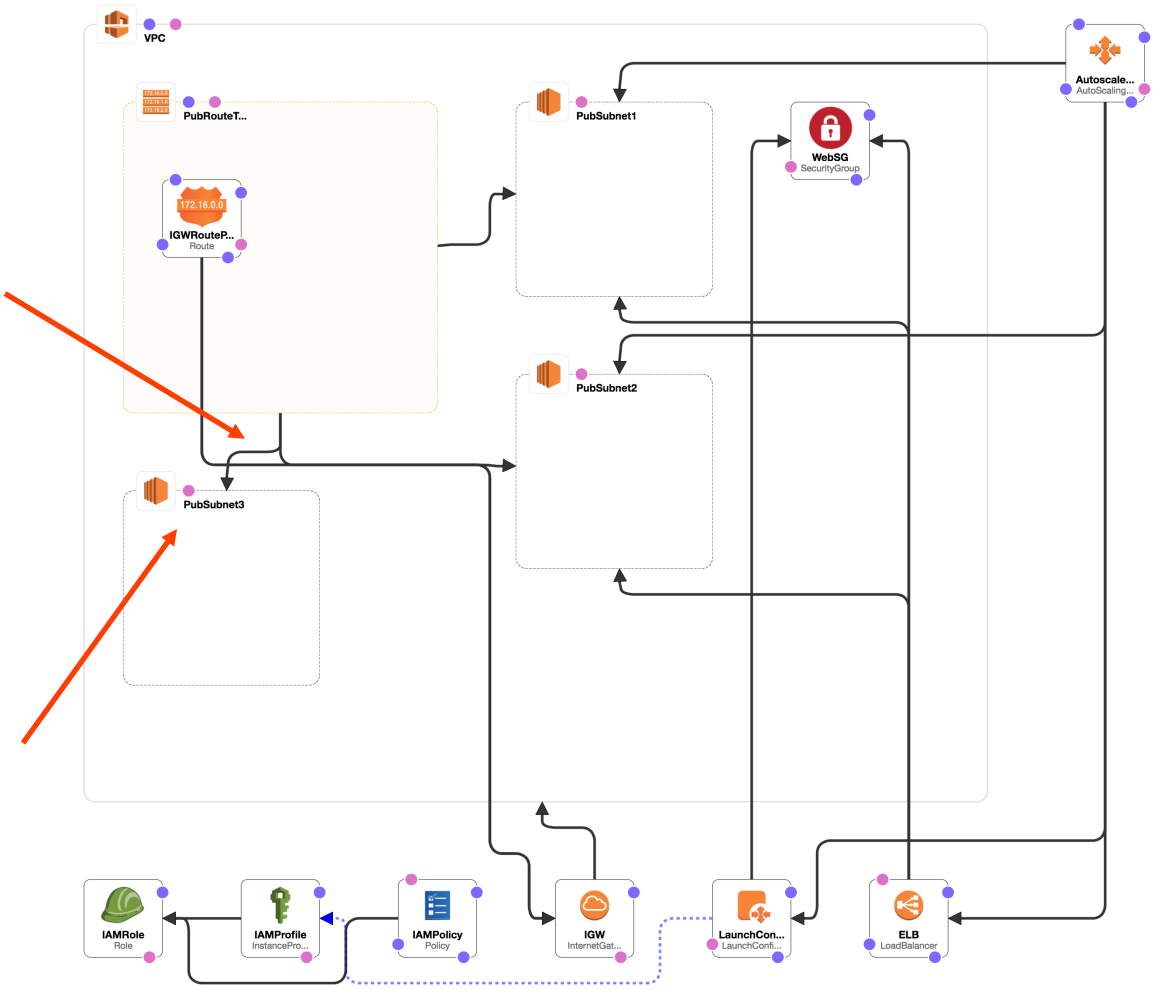
10. To summarize, in the above three steps we have:
    - Added a parameter which will be used to help us define the size of our subnet
    - Added the resource for a new AWS subnet
    - Associated our new subnet with the same route table used by our other two public subnets

11. Save your **Lab3-Working.json** template

12. Next, navigate to the CloudFormation Designer (see Step 3 if you need a reminder on how) and **open the Lab3-Working.json** template

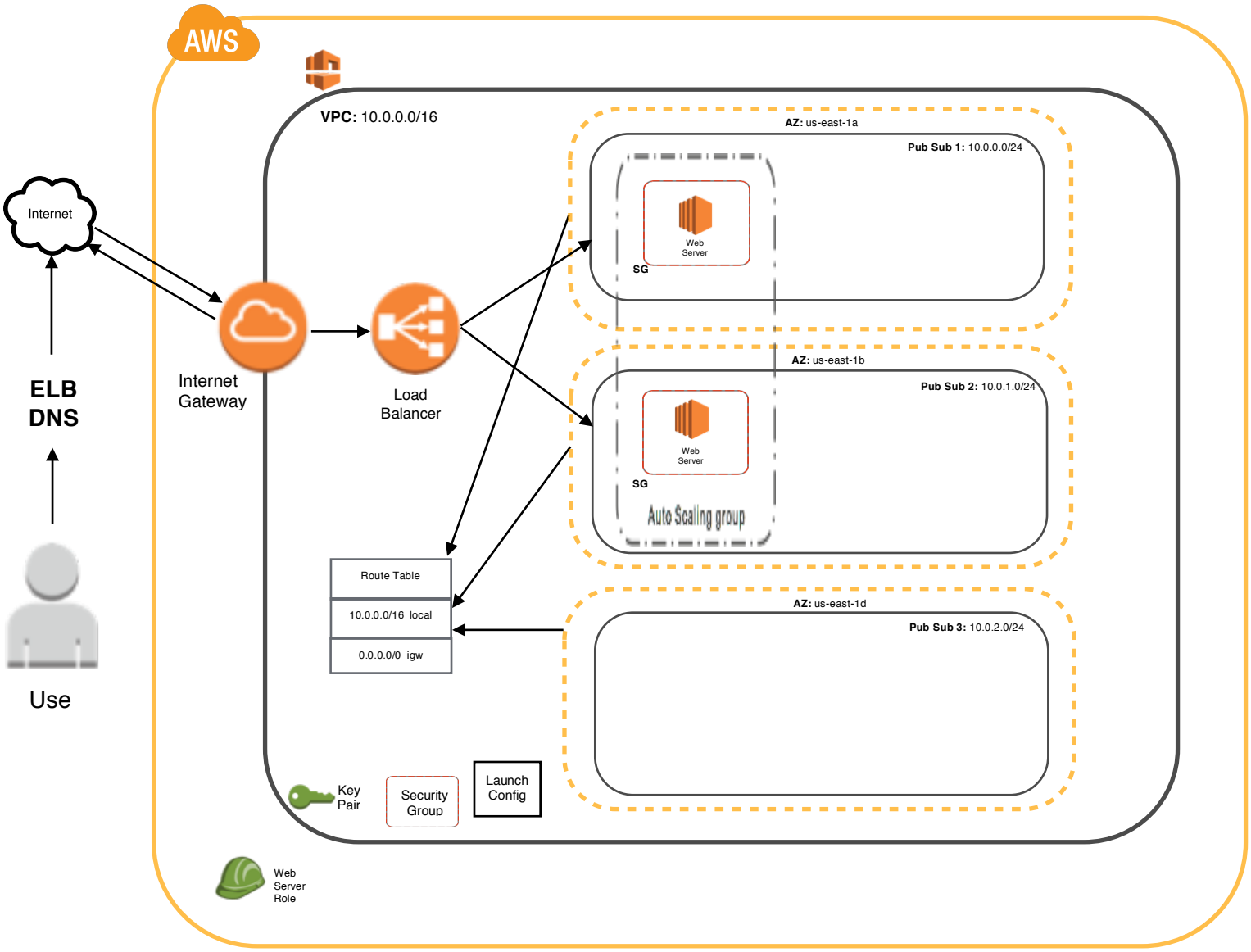
13. If the additions within this Step have worked out properly, you should see the following when you open your JSON template:

CloudFormation Designer – Step 4



14. I have added the red arrows to show the new components that should be visible in the diagram. If your diagram looks like the above, great! If not, remember that you can always start again from the Base template and re-do the above steps, or get ready for Step 5 by copy/pasting **Lab3-Step4.json** and renaming as **Lab3-Working.json**

System Architecture: Step 4



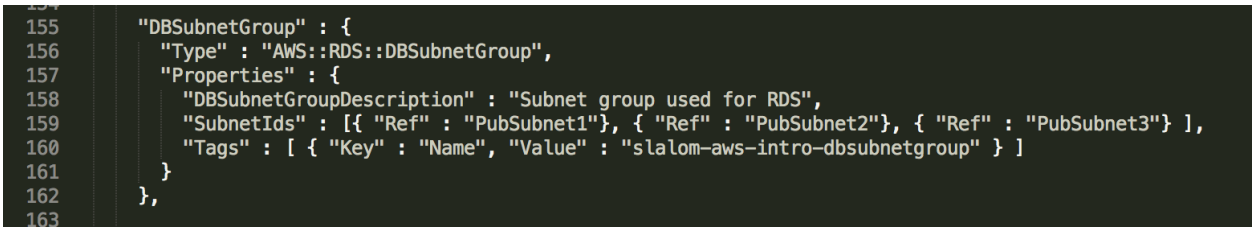
Step 5: DB Subnet Group and Security Group

1. Within your **Lab3-Working.json** file, lets first add in a subnet group for our future RDS instance. Navigate to the end of **Line 153** and press return twice.

2. Starting on **line 155**, add the following code:

```
"DBSubnetGroup" : {
  "Type" : "AWS::RDS::DBSubnetGroup",
  "Properties" : {
    "DBSubnetGroupDescription" : "Subnet group used for RDS",
    "SubnetIds" : [{ "Ref" : "PubSubnet1"}, { "Ref" : "PubSubnet2"}, { "Ref" : "PubSubnet3"} ],
    "Tags" : [ { "Key" : "Name", "Value" : "slalom-aws-intro-dbsubnetgroup" } ]
  }
},
```

3. This should leave you with the following on **lines 155-162**:



4. Next, we will add in a security group for our future RDS instance. Navigate to the end of **line 162** and press return twice

5. Add the following code starting on **line 164**:

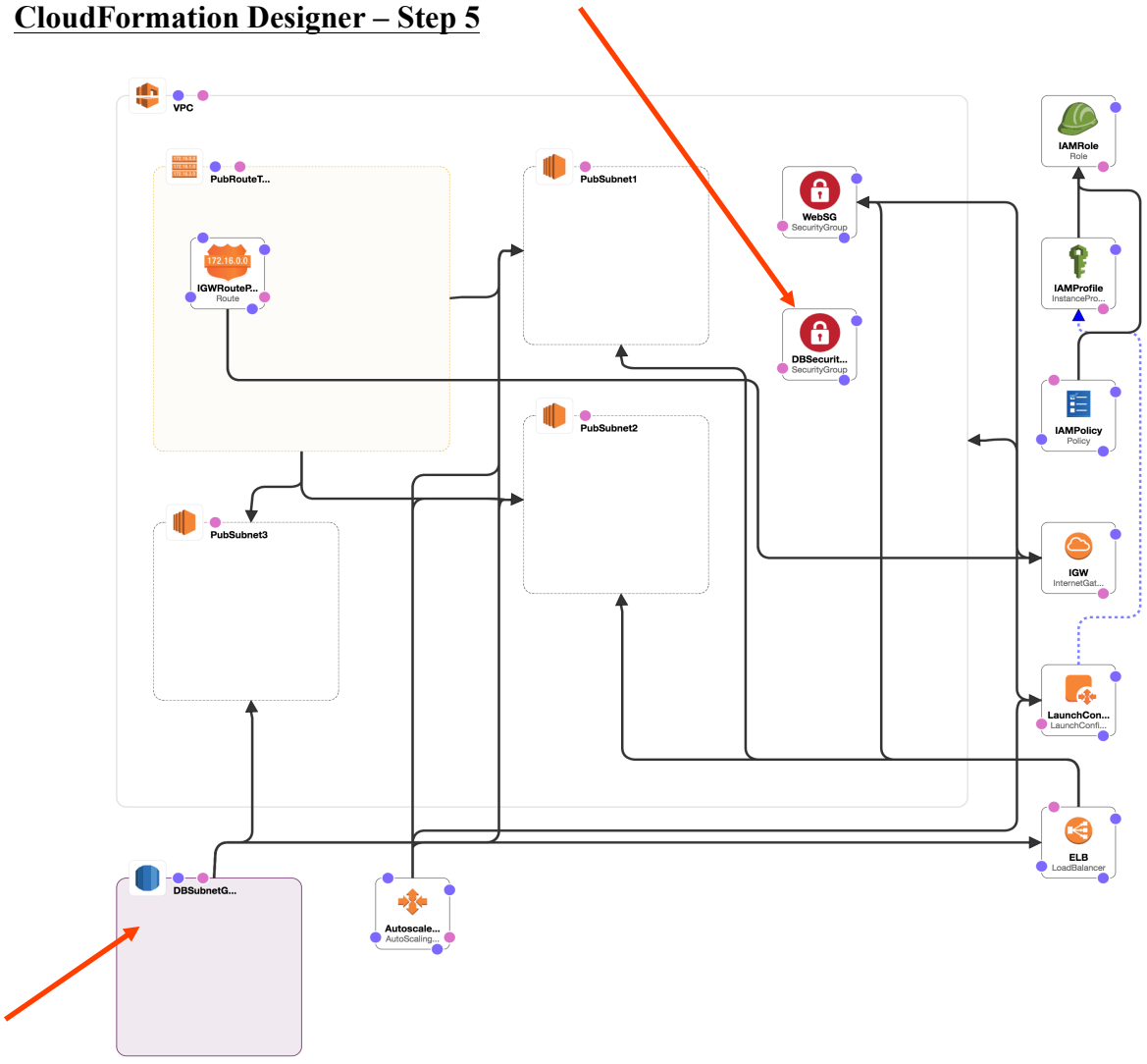
```
"DBSecurityGroup" : {
  "Type" : "AWS::EC2::SecurityGroup",
  "Properties" : {
    "GroupDescription" : "VPC Security group used for RDS",
    "VpcId" : { "Ref" : "VPC" },
    "Tags" : [ { "Key" : "Name", "Value" : "slalom-aws-intro-dbsecuritygroup" } ]
  }
},
```

6. After the above two additions, you should have the following between **lines 155-171**:

```
155     "DBSubnetGroup" : {
156       "Type" : "AWS::RDS::DBSubnetGroup",
157       "Properties" : {
158         "DBSubnetGroupDescription" : "Subnet group used for RDS",
159         "SubnetIds" : [{ "Ref" : "PubSubnet1"}, { "Ref" : "PubSubnet2"}, { "Ref" : "PubSubnet3"} ],
160         "Tags" : [ { "Key" : "Name", "Value" : "slalom-aws-intro-dbsubnetgroup" } ]
161       }
162     },
163
164     "DBSecurityGroup" : {
165       "Type" : "AWS::EC2::SecurityGroup",
166       "Properties" : {
167         "GroupDescription" : "VPC Security group used for RDS",
168         "VpcId" : { "Ref" : "VPC" },
169         "Tags" : [ { "Key" : "Name", "Value" : "slalom-aws-intro-dbsecuritygroup" } ]
170       }
171     },
172
```

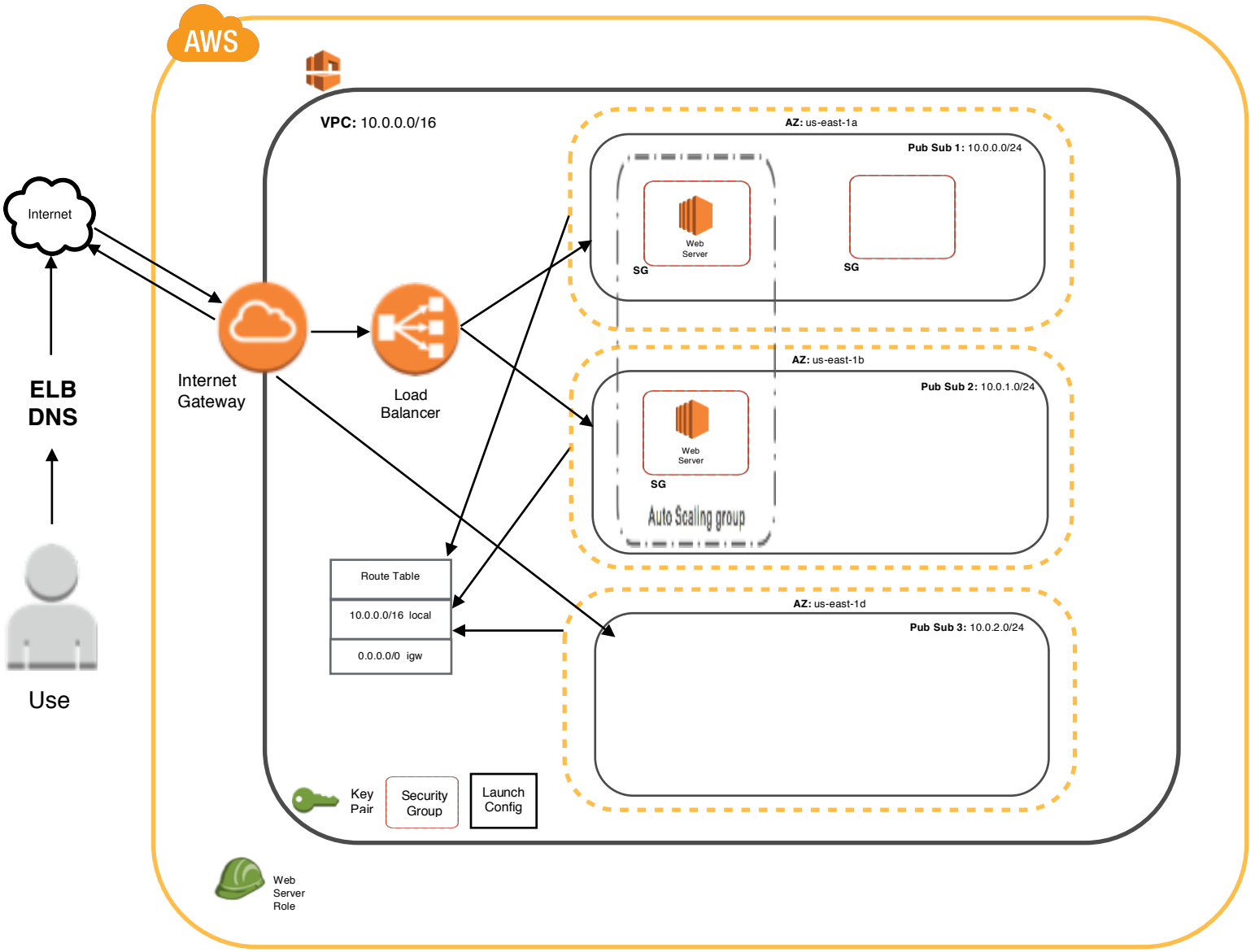
7. Save your **Lab3-Working.json file**, navigate to the CloudFormation Designer and open the file there. You should see the following (red arrows indicate new additions):

CloudFormation Designer – Step 5



8. Again, hopefully things are going to plan and your template looks like the above in the designer. However, if it doesn't then feel free to copy/paste **Lab3-Step5.json** and rename as **Lab3-Working.json** file

System Architecture: Step 5



Step 6: Create an IAM Policy, Role and DB Cluster Parameter Group

1. Remembering the purpose of AWS Identity and Access Management (to limit which AWS services a resource can talk to), we need to create an IAM policy and role that will be used by our future RDS database instance.

2. Navigate to the end of **line 171**, press return twice and enter the following code block:

```
"IAMRoleAurora" : {
  "Type" : "AWS::IAM::Role",
  "Properties" : {
    "RoleName" : "slalom_aurora_s3read",
    "AssumeRolePolicyDocument" : {
      "Version" : "2012-10-17",
      "Statement" : [
        {
          "Action" : "sts:AssumeRole",
          "Principal" : {
            "Service" : "rds.amazonaws.com"
          },
          "Effect" : "Allow",
          "Sid" : ""
        }
      ]
    }
  }
},

"IAMPolicyAurora" : {
  "Type" : "AWS::IAM::Policy",
  "Properties" : {
    "PolicyName" : "slalom_aurora_s3read_policy",
    "Roles" : [{ "Ref" : "IAMRoleAurora" }],
    "PolicyDocument" : {
      "Version" : "2012-10-17",
      "Statement" : [
        {
          "Action": [
            "s3:GetObject",
            "s3:GetObjectVersion"
          ],
          "Effect" : "Allow",
          "Resource" : "arn:aws:s3:::aws-slalom-brown-bag-sample"
        }
      ]
    }
  }
},
```

3. Before we go any further, lets discuss what we've just added:

   - The top block **(lines 173-191)** creates our **AWS IAM Role** for Aurora, our future RDS Instance
   - This block contains a small **IAM Policy Document** that allows our chosen service, AWS Relational Database Service (RDS) to use this IAM role

   - The second code block **(lines 193-212)** creates our AWS IAM Policy
   - This policy contains a statement, which allows it to Get Objects from storage in Amazon S3
   - This IAM Policy is associated to our new IAM Role using the Ref function on **line 197**

4. After adding these two blocks, we should have the following:

```
172
173     "IAMRoleAurora" : {
174       "Type" : "AWS::IAM::Role",
175       "Properties" : {
176         "RoleName" : "slalom_aurora_s3read",
177         "AssumeRolePolicyDocument" : {
178           "Version" : "2012-10-17",
179           "Statement" : [
180             {
181               "Action" : "sts:AssumeRole",
182               "Principal" : {
183                 "Service" : "rds.amazonaws.com"
184               },
185               "Effect" : "Allow",
186               "Sid" : ""
187             }
188           ]
189         }
190       }
191     },
192
193     "IAMPolicyAurora" : {
194       "Type" : "AWS::IAM::Policy",
195       "Properties" : {
196         "PolicyName" : "slalom_aurora_s3read_policy",
197         "Roles" : [{ "Ref" : "IAMRoleAurora" }],
198         "PolicyDocument" : {
199           "Version" : "2012-10-17",
200           "Statement" : [
201             {
202               "Action": [
203                 "s3:GetObject",
204                 "s3:GetObjectVersion"
205               ],
206               "Effect" : "Allow",
207               "Resource" : "arn:aws:s3:::slalom-aws-intro"
208             }
209           ]
210         }
211       }
212     },
213
```

5. Our second task within this Step is to create a DB Cluster Parameter Group. Navigate to the end of **line 212** and press return twice

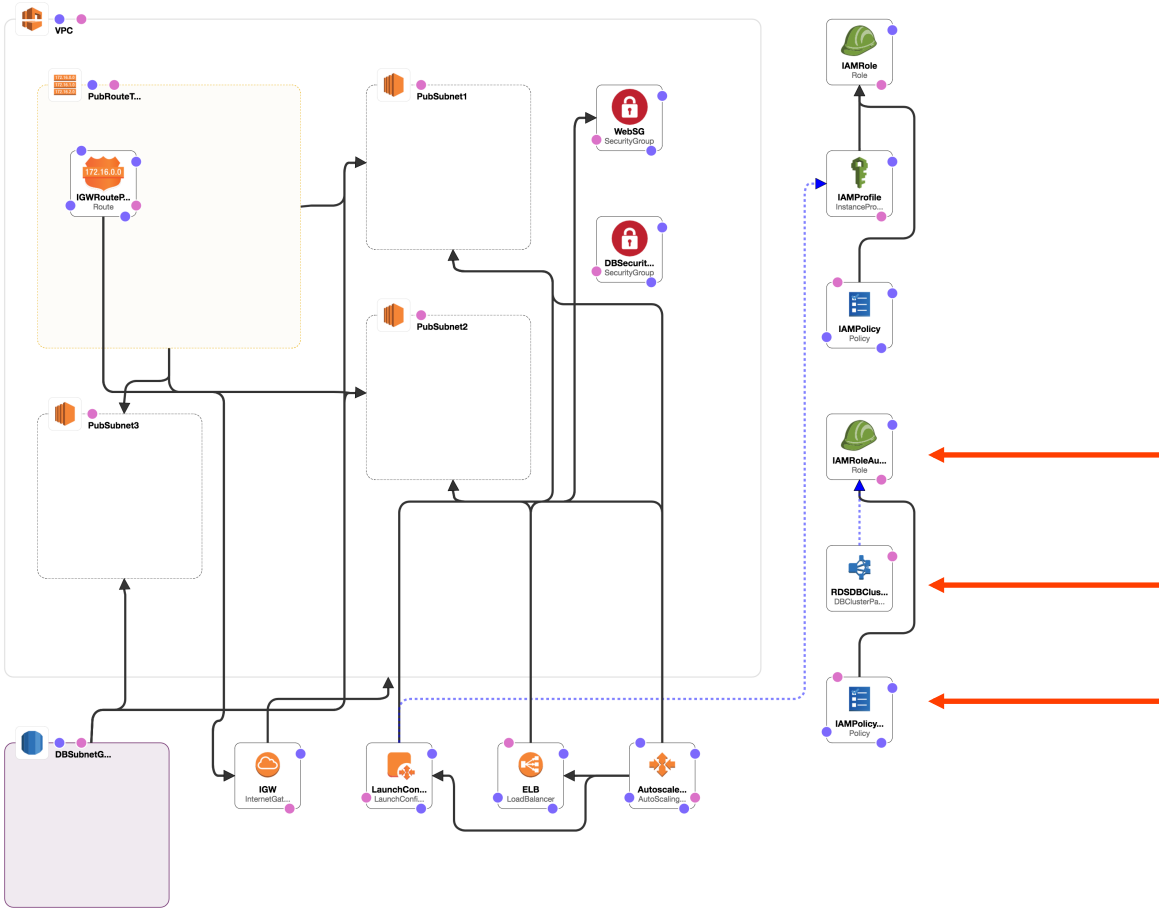
6. Add the following code block, starting on **line 214**:

```
"RDSDBClusterParameterGroup" : {
 "Type" : "AWS::RDS::DBClusterParameterGroup",
 "Properties" : {
  "Parameters" : {
   "aurora_load_from_s3_role" : { "Fn::GetAtt" : [ "IAMRoleAurora", "Arn" ] }
  },
  "Family" : "aurora5.6",
  "Description" : "Cluster parameter group for Aurora"
 }
},
```

7. The DB Cluster Parameter Group that we have just created will be attached to our RDS DB Cluster. Most important of all is **line 218**, which ensures that our DB cluster will use our newly created IAM role to load data from Amazon S3

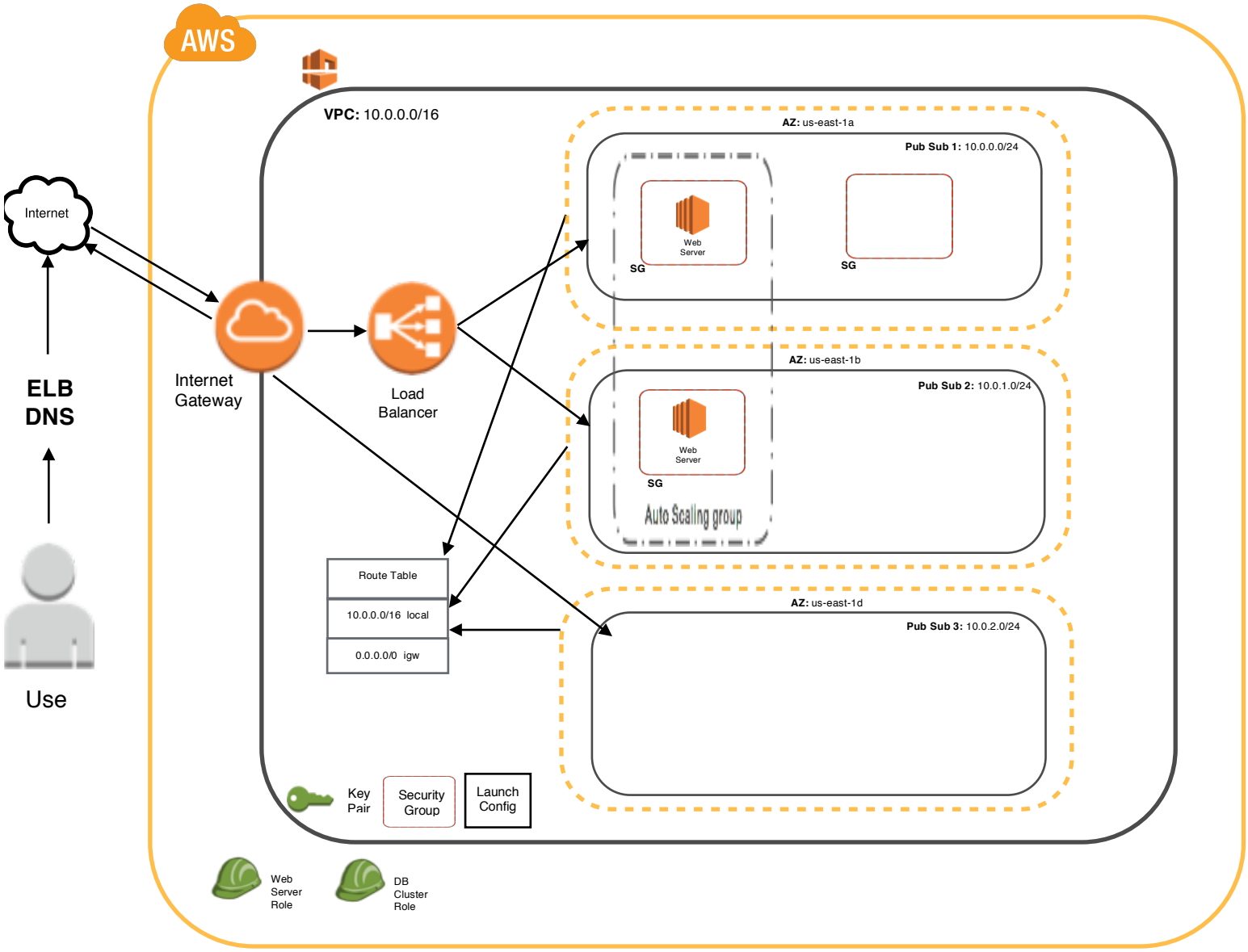
8. Again, save your **Lab3-Working.json** template file and open it in the CloudFormation Designer. It should hopefully match the following:

CloudFormation Designer – Step 6



9. Hopefully things are looking good and you see the three new resources we created above! If not, feel free to copy/paste **Lab3-Step6.json** and rename as your **Lab3-Working.json** template file

System Architecture: Step 6



Step 7: Create RDS Cluster and RDS Instance

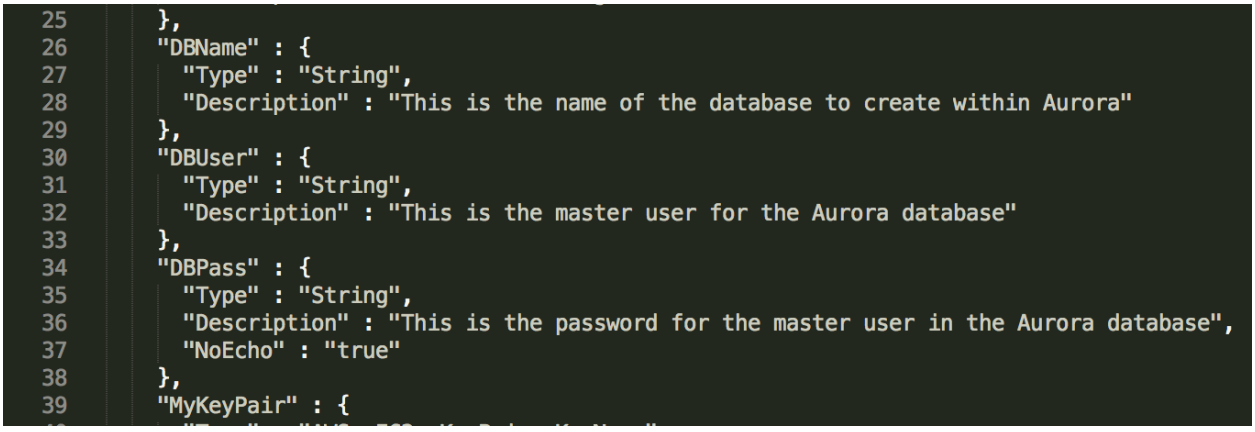
1. We are now at the stage where we are ready to define our RDS Cluster and RDS Instance. Rather than using a familiar engine such as OracleDB or MySQL, we are going to be using AWS's very own DB engine called **Aurora**.

2. Before we define the new resources, we need to add three new parameters for our DB Cluster and RDS Instance. In **Lab3-Working.json**, navigate up to the **"Parameters"** section in **lines 9-42**

3. Click to the right of **line 25** and press return once. Add in the following block of three parameters for DBName, DBUser and DBPass:

```
"DBName" : {
  "Type" : "String",
  "Description" : "This is the name of the database to create within Aurora"
},
"DBUser" : {
  "Type" : "String",
  "Description" : "This is the master user for the Aurora database"
},
"DBPass" : {
  "Type" : "String",
  "Description" : "This is the password for the master user in the Aurora database",
  "NoEcho" : "true"
},
```

4. This should leave the following:



```
25      },
26      "DBName" : {
27        "Type" : "String",
28        "Description" : "This is the name of the database to create within Aurora"
29      },
30      "DBUser" : {
31        "Type" : "String",
32        "Description" : "This is the master user for the Aurora database"
33      },
34      "DBPass" : {
35        "Type" : "String",
36        "Description" : "This is the password for the master user in the Aurora database",
37        "NoEcho" : "true"
38      },
39      "MyKeyPair" : {
```

5. Examining the three parameters that we just added between **lines 26-38**, notice that the **DBPass** parameter has an option called **NoEcho**. This ensures that the password that you enter when we run our template in **Step 10** will be hidden from other people.

6. Now that we have our parameters in place, we can navigate back to the resources section of our template. Navigate to the end of **line 236**, press return twice and enter the following:

```
"RDSCluster" : {
  "Type" : "AWS::RDS::DBCluster",
  "Properties" : {
    "DatabaseName" : { "Ref" : "DBName" },
    "MasterUsername" : { "Ref" : "DBUser" },
    "MasterUserPassword" : { "Ref" : "DBPass" },
    "Engine" : "aurora",
    "DBSubnetGroupName" : { "Ref" : "DBSubnetGroup" },
    "DBClusterParameterGroupName" : { "Ref" : "RDSDBClusterParameterGroup" },
    "VpcSecurityGroupIds" : [ { "Ref" : "DBSecurityGroup" } ]
  }
},

"RDSDBInstance" : {
  "Type" : "AWS::RDS::DBInstance",
  "Properties" : {
    "DBSubnetGroupName" : {
      "Ref" : "DBSubnetGroup"
    },
    "Engine" : "aurora",
    "DBClusterIdentifier" : {
      "Ref" : "RDSCluster"
    },
    "PubliclyAccessible" : "true",
    "AvailabilityZone" : { "Fn::GetAtt" : [ "PubSubnet1", "AvailabilityZone" ] },
    "DBInstanceClass" : "db.t2.small"
  }
},
```

7. This should leave you with the following between **lines 238-265**

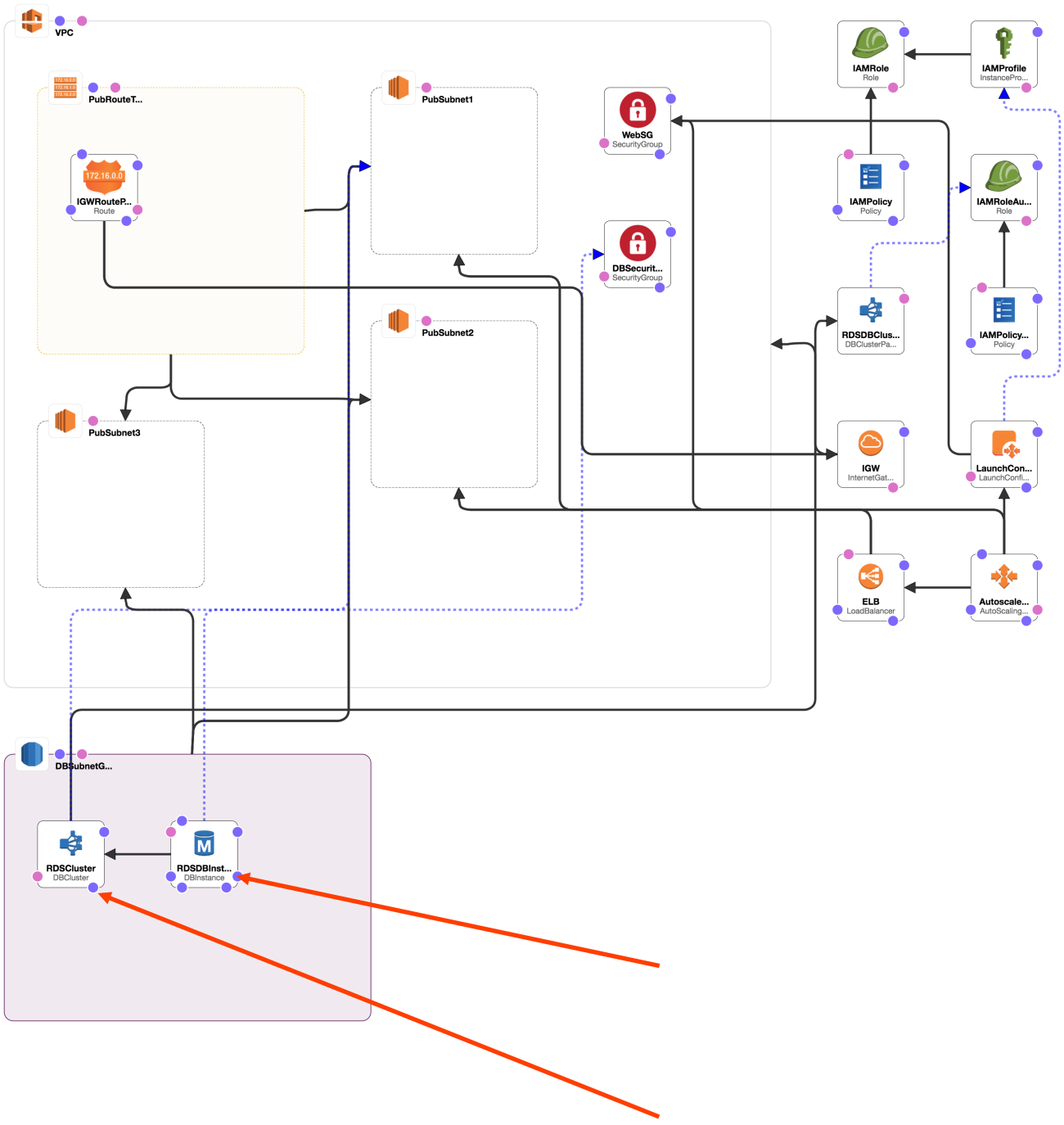
```
237
238     "RDSCluster" : {
239       "Type" : "AWS::RDS::DBCluster",
240       "Properties" : {
241         "DatabaseName" : { "Ref" : "DBName" },
242         "MasterUsername" : { "Ref" : "DBUser" },
243         "MasterUserPassword" : { "Ref" : "DBPass" },
244         "Engine" : "aurora",
245         "DBSubnetGroupName" : { "Ref" : "DBSubnetGroup" },
246         "DBClusterParameterGroupName" : { "Ref" : "RDSDBClusterParameterGroup" },
247         "VpcSecurityGroupIds" : [ { "Ref" : "DBSecurityGroup" } ]
248       }
249     },
250
251     "RDSDBInstance" : {
252       "Type" : "AWS::RDS::DBInstance",
253       "Properties" : {
254         "DBSubnetGroupName" : {
255           "Ref" : "DBSubnetGroup"
256         },
257         "Engine" : "aurora",
258         "DBClusterIdentifier" : {
259           "Ref" : "RDSCluster"
260         },
261         "PubliclyAccessible" : "true",
262         "AvailabilityZone" : { "Fn::GetAtt" : [ "PubSubnet1", "AvailabilityZone" ] },
263         "DBInstanceClass" : "db.t2.small"
264       }
265     },
266
```

8. The first code block has added an RDS DBCluster resource to the template. This cluster acts as a logical grouping for a collection of RDS Instances and is a feature employed heavily by AWS's Aurora engine. You can see our new parameters being used on **lines 241, 242 and 243** to configure the cluster. In addition, the DB Cluster Parameter Group and DB Security Group we created in the previous step are also referenced on lines 246 and 247.

9. The second code block creates an Aurora RDS Instance within our new RDS DB Cluster. The AvailabilityZone options ensures that the DB is created in Public Subnet 1.

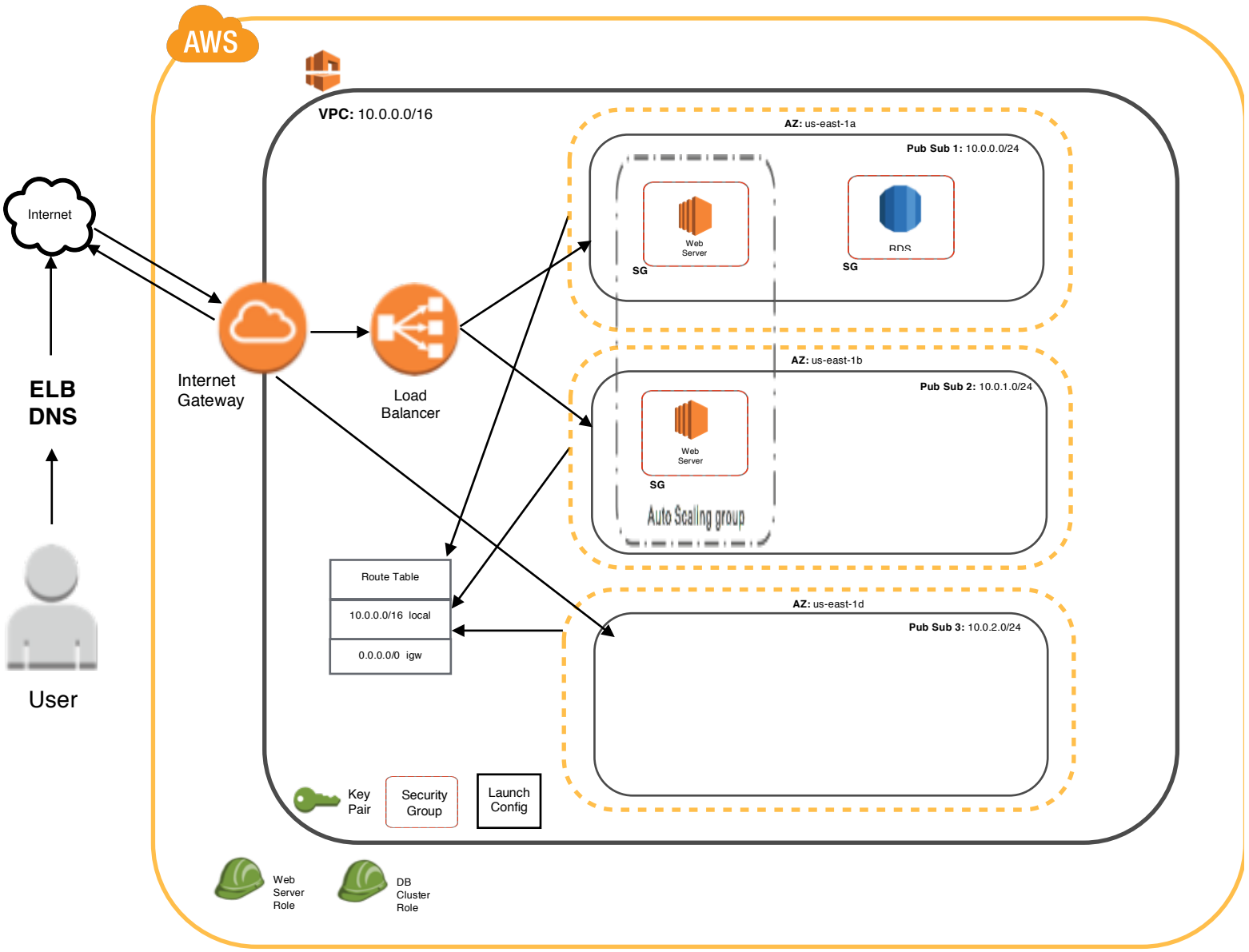
10. Save your **Lab3-Working.json** template and load it into the CloudFormation designer. It should hopefully look like the image shown at the top of the next page

CloudFormation Designer – Step 7



11. Our architecture is certainly growing in complexity and this is certainly not an easy task! If you need a refresh of your template if it doesn't quite look right, remember that you can copy/paste **Lab3-Step7.json** and name it **Lab3-Working.json** before continuing to Step 8.

System Architecture: Step 7



Step 8: Create a Tableau Server

1. You'll be glad to hear that this is the final addition required to our CloudFormation Template! In this step we are going to add in an EC2 instance that will serve as a Tableau server and a security group for this EC2 instance with some predefined security rules

2. Navigate to the right of **line 265** in your **Lab3-Working.json** template and press return twice

3. Enter the following code block to create a security group for our Tableau server:

```
"TableauSG" : {
  "Type" : "AWS::EC2::SecurityGroup",
  "Properties" : {
    "GroupDescription" : "Security group for the Tableau server",
    "SecurityGroupIngress" : [
     {
       "IpProtocol" : "tcp",
       "FromPort" : "3389",
       "ToPort" : "3389",
       "CidrIp" : "0.0.0.0/0"
     },
     {
       "IpProtocol" : "tcp",
       "FromPort" : "443",
       "ToPort" : "443",
       "CidrIp" : "0.0.0.0/0"
     },
     {
       "IpProtocol" : "tcp",
       "FromPort" : "80",
       "ToPort" : "80",
       "CidrIp" : "0.0.0.0/0"
     }
    ],
    "SecurityGroupEgress" : [
     {
       "IpProtocol" : "-1",
       "FromPort" : "0",
       "ToPort" : "0",
       "CidrIp" : "0.0.0.0/0"
     }
    ],
    "VpcId" : { "Ref" : "VPC" },
    "Tags" : [ { "Key" : "Name", "Value" : "slalom_aws_intro_tableau" } ]
  }
},
```

4. Examining the above code block, we have:
   - Created an AWS Security Group
   - Added three inbound tcp (Ingress) rules for ports 3389, 443 and 80
   - Added one outbound (Egress) rules for all ports and protocols

5. Next, navigate to the end of **line 302**, press return twice and enter the following on line 304:

```
"TableauServer" : {
  "Type" : "AWS::EC2::Instance",
  "Properties" : {
    "IamInstanceProfile" : { "Ref" : "IAMProfile" },
    "ImageId" : "ami-24f27932",
    "InstanceType" : "c3.4xlarge",
    "KeyName" : { "Ref" : "MyKeyPair" },
    "SecurityGroupIds" : [{ "Ref" : "TableauSG" }],
    "SubnetId" : { "Ref" : "PubSubnet3" },
    "Tags" : [ { "Key" : "Name", "Value" : "slalom_aws_intro_tableau" } ]
  }
},
```

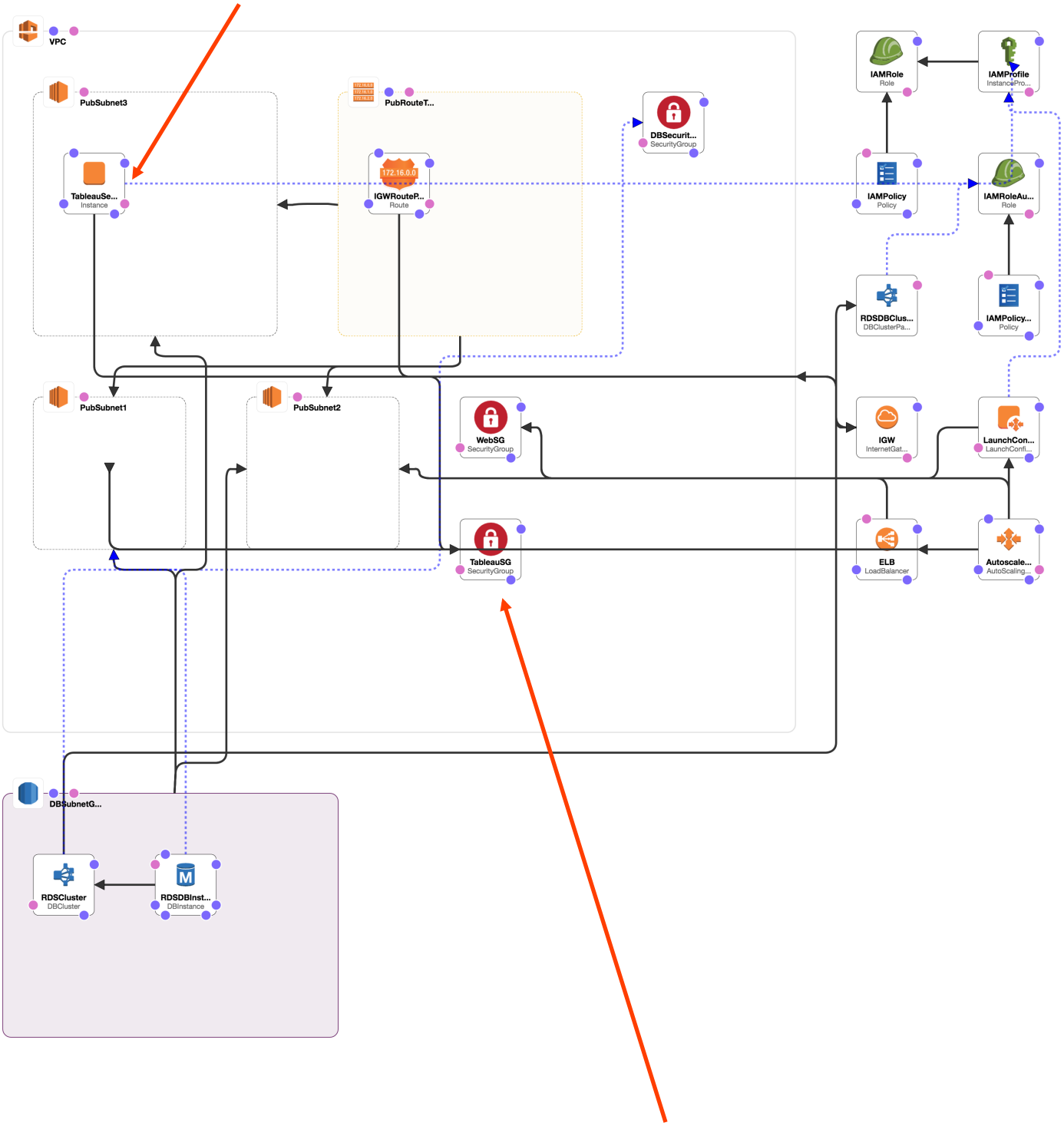
5. Examining the above code block we have:
   - Created an EC2 instance
   - The image we are using (ami-24f27932) is pre-configured with Tableau Server 10.2
   - We are referencing a few other parameters and resources using the **Ref** function

6. If everything has been entered correctly, your code should match the image at the top of the following page:

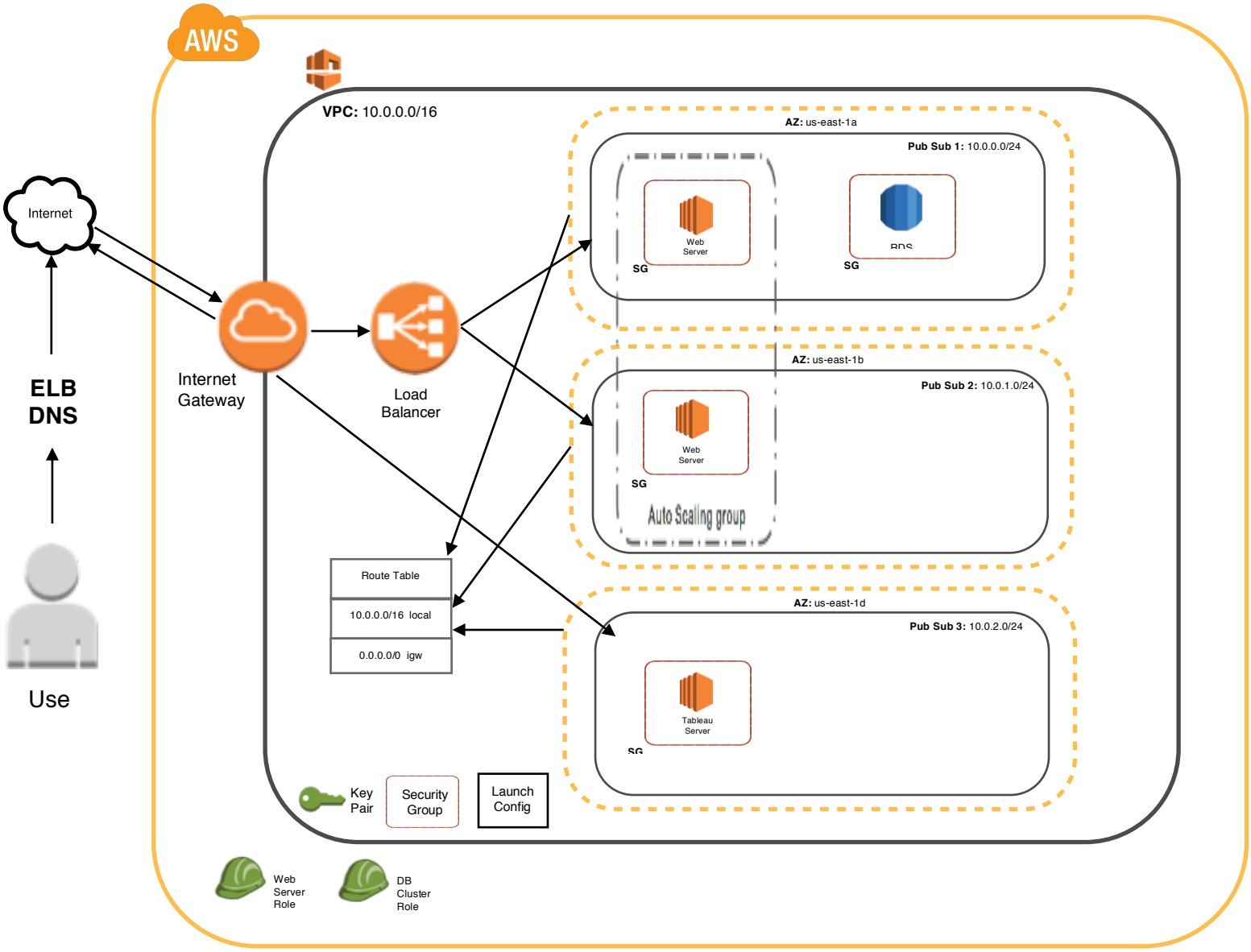
```
266
267     "TableauSG" : {
268       "Type" : "AWS::EC2::SecurityGroup",
269       "Properties" : {
270         "GroupDescription" : "Security group for the Tableau server",
271         "SecurityGroupIngress" : [
272           {
273             "IpProtocol" : "tcp",
274             "FromPort" : "3389",
275             "ToPort" : "3389",
276             "CidrIp" : "0.0.0.0/0"
277           },
278           {
279             "IpProtocol" : "tcp",
280             "FromPort" : "443",
281             "ToPort" : "443",
282             "CidrIp" : "0.0.0.0/0"
283           },
284           {
285             "IpProtocol" : "tcp",
286             "FromPort" : "80",
287             "ToPort" : "80",
288             "CidrIp" : "0.0.0.0/0"
289           }
290         ],
291         "SecurityGroupEgress" : [
292           {
293             "IpProtocol" : "-1",
294             "FromPort" : "0",
295             "ToPort" : "0",
296             "CidrIp" : "0.0.0.0/0"
297           }
298         ],
299         "VpcId" : { "Ref" : "VPC" },
300         "Tags" : [ { "Key" : "Name", "Value" : "slalom_aws_intro_tableau" } ]
301       }
302     },
303
304     "TableauServer" : {
305       "Type" : "AWS::EC2::Instance",
306       "Properties" : {
307         "IamInstanceProfile" : { "Ref" : "IAMProfile" },
308         "ImageId" : "ami-24f27932",
309         "InstanceType" : "c3.4xlarge",
310         "KeyName" : { "Ref" : "MyKeyPair" },
311         "SecurityGroupIds" : [{ "Ref" : "TableauSG" }],
312         "SubnetId" : { "Ref" : "PubSubnet3" },
313         "Tags" : [ { "Key" : "Name", "Value" : "slalom_aws_intro_tableau" } ]
314       }
315     },
316
```

7. Lets save our final version of **Lab3-Working.json** and load it into the CloudFormation designer. It should look like the following image:

CloudFormation Designer – Step 8



System Architecture: Step 8



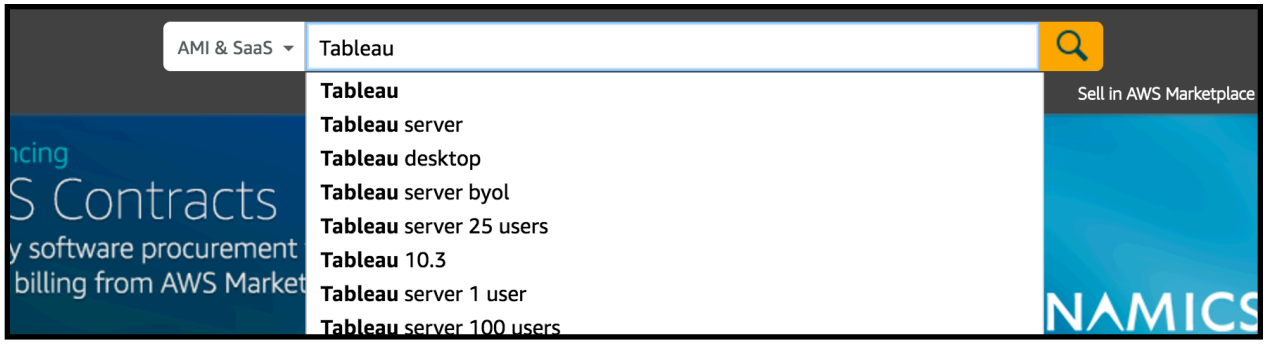
Step 9: Accept Terms and Conditions in the AWS Marketplace

In Step 8 we created an AWS EC2 instance running a pre-configured version of Tableau server. This pre-configured version is packaged within an Amazon Machine Image (AMI).

Before we can proceed to running our **Lab3-Working.json** template, we must accept the terms and conditions for the Tableau software in the AWS Marketplace. The AWS Marketplace is home to a number of pre-configured software types that can be launched in Amazon AWS.

1. Navigate to **https://aws.amazon.com/marketplace**

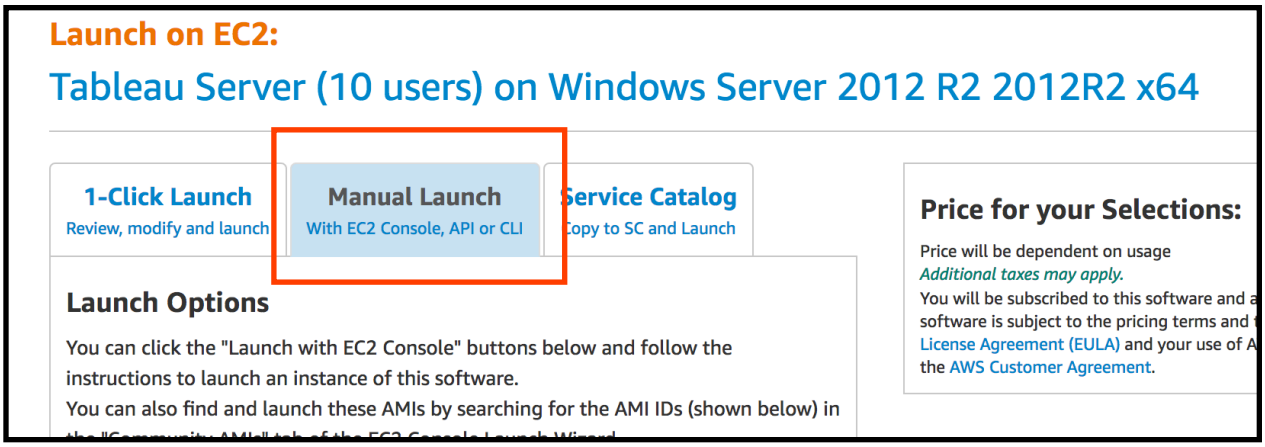
2. In the search bar, enter “Tableau” and proceed to search



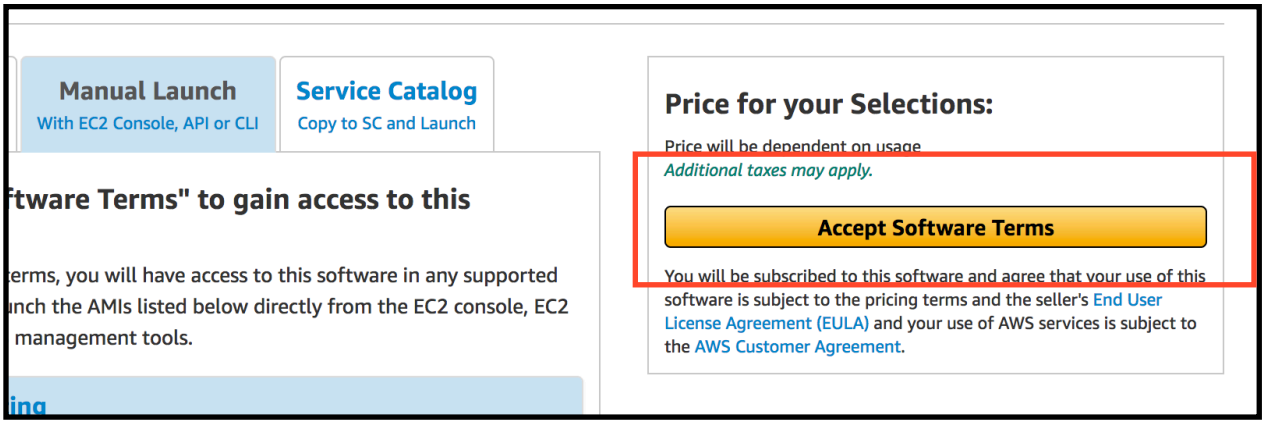
3. Of the options returned by the search, select **“Tableau Server (10 Users)”**

4. Ensuring that the region is set to **US East**, proceed to click the yellow **Continue** button

5. In the following page, select the **Manual Launch** Tab



6. In the box to the right of the page, select the **Accept Software Terms** button



7. You should receive a message to let you know you have successfully subscribed to Tableau Server (10 Users)

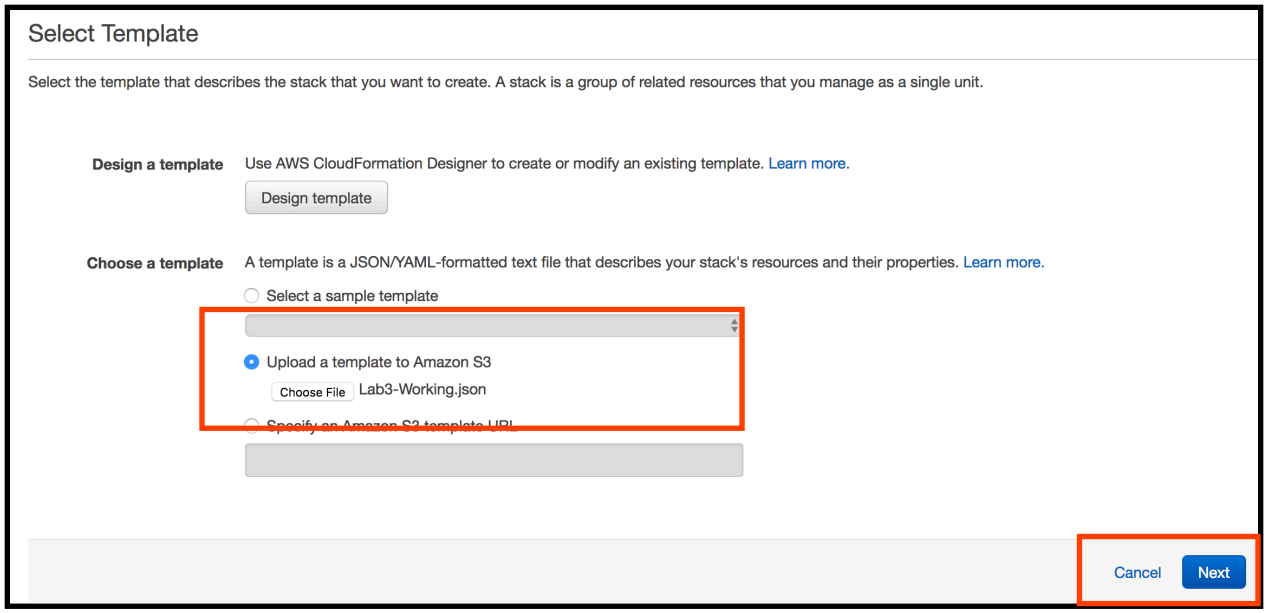
Step 10: Launch our CloudFormation Template

1. Ensure that you have all changes saved in your **Lab3-Working.json** CloudFormation template

2. Navigate to **Services —> CloudFormation**

3. Click the blue **Create New Stack** button in the middle of the screen

4. On the next screen, select the **"Upload a template to Amazon S3"** option and click the **Choose File** button. Navigate to and select your Lab3-Working.json file and click Open.



5. Once you have your file loaded, click the blue **Next** button

6. The following screen is the **Specify Details** screen. This is where we populate all of the parameters from lines 9-56 in the template with their values. Within this screen, enter the following:

   Stack Name: slalom-aws-intro-lab3

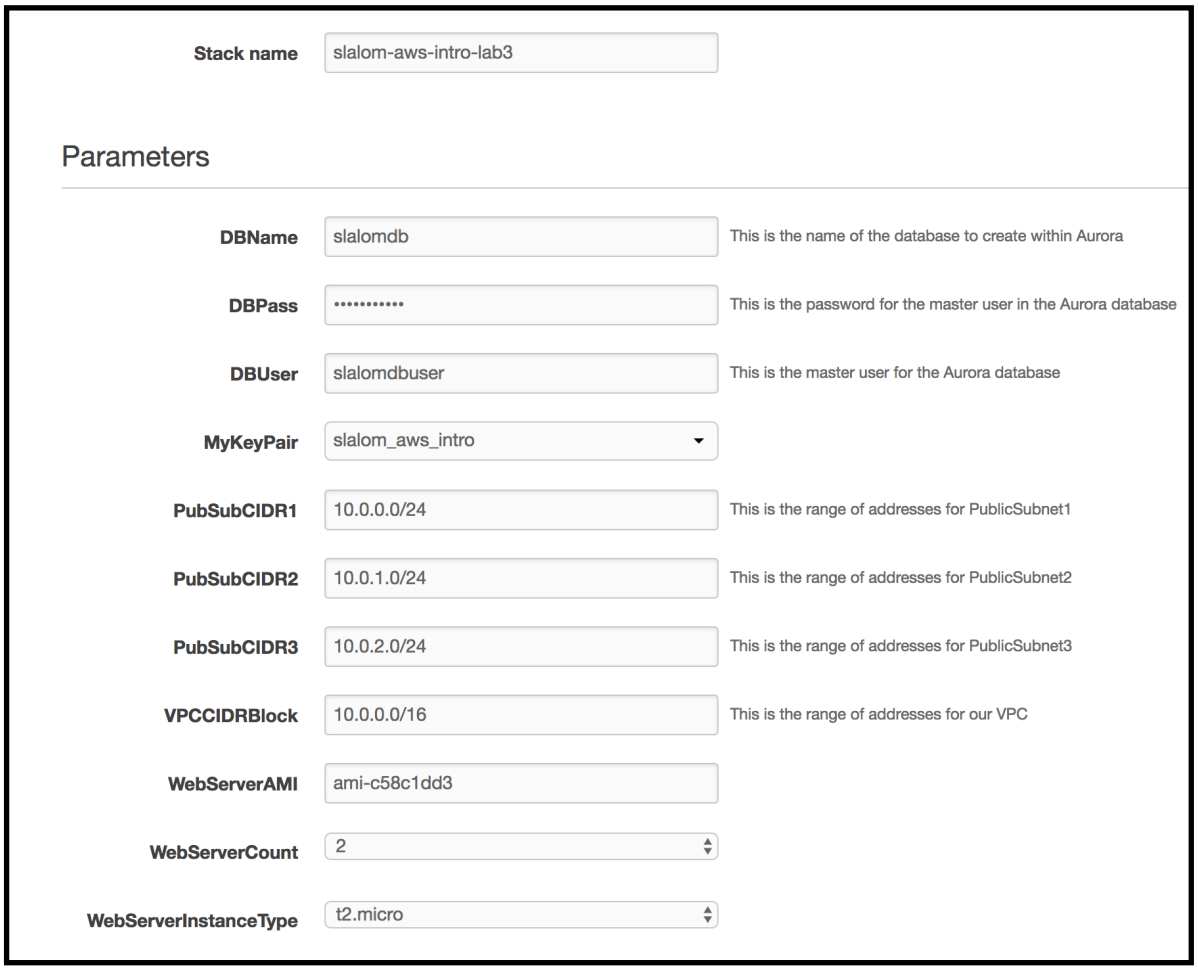
   DBName: slalomdb
   DBPass: <Choose your own password>
   DBUser: slalomdbuser

   MyKeyPair: Select the KeyPair that we created in Lab1

   PubSubCIDR1: 10.0.0.0/24
   PubSubCIDR2: 10.0.1.0/24
   PubSubCIDR3: 10.0.2.0/24

   VPCCIDRBlock: 10.0.0.0/16

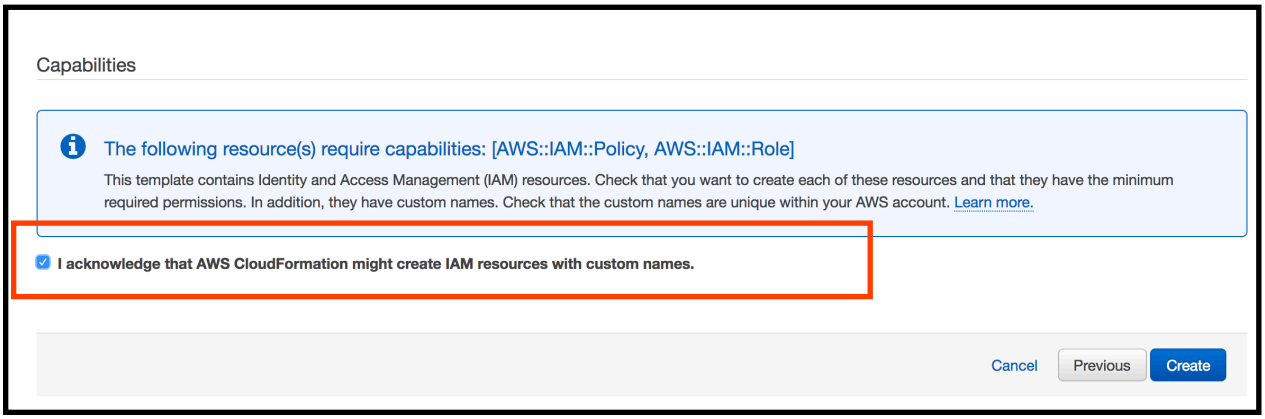
   WebServerAMI: ami-c58c1dd3
   WebServerCount: 2
   WebServerInstanceType: t2.micro



7. Once you have the above values entered, select the blue **Next** button

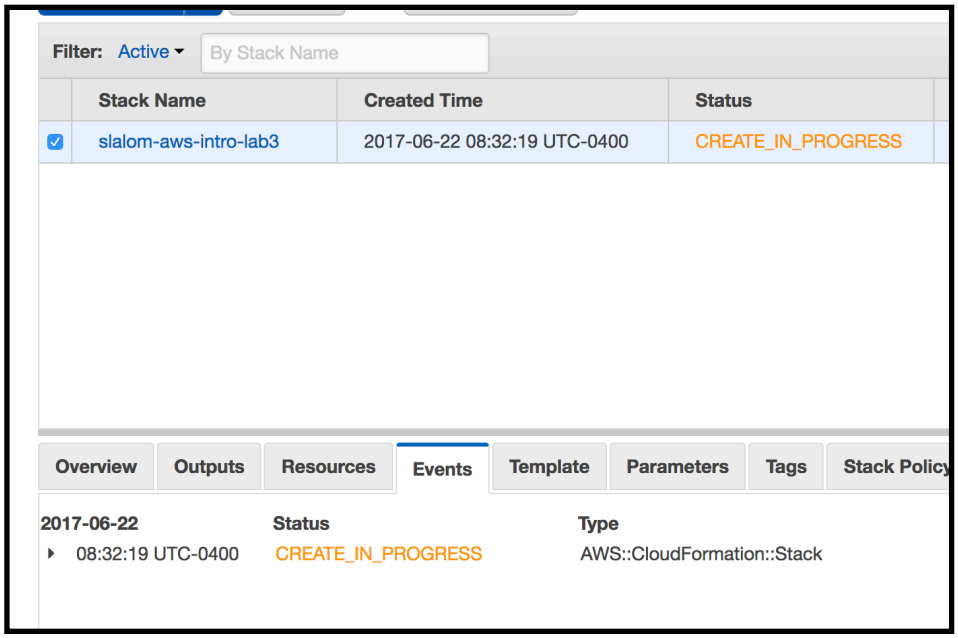
8. On the following screen, we don't need to configure any Options for our stack so click the blue **Next** button

9. On the **Review** screen, check that everything looks ok in your configuration and scroll to the bottom. Within the Capabilities section, select the checkbox that says **"I acknowledge that AWS CloudFormation might create IAM resources with custom names"**. This is required because our template is creating IAM Policies and Roles.

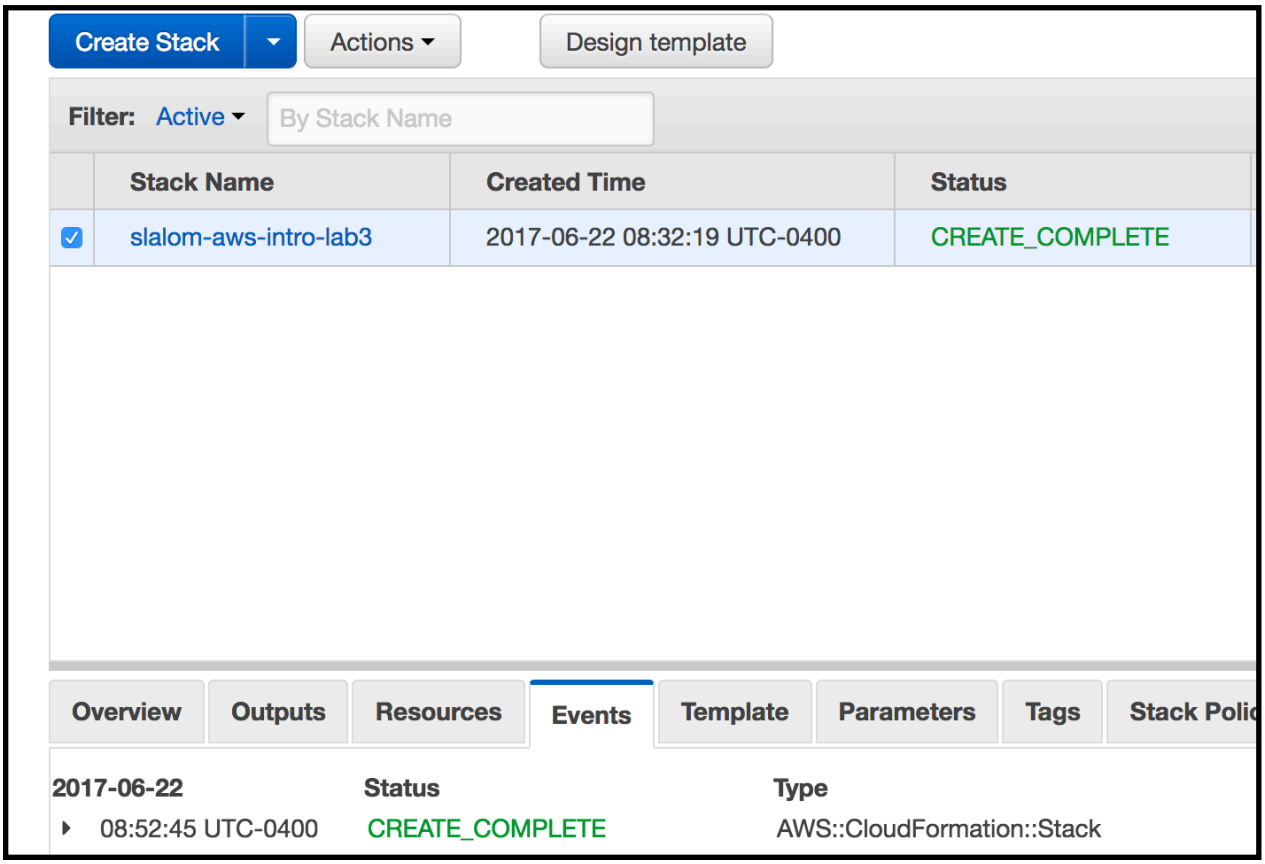


10. Once you are ready, click the blue **Create** button

11. You will be redirected to the following screen, where you will start to see the Events tab populate with notifications as resources are created in AWS



12. If everything works as expected (expect it to take ~20 mins), you should see that the CloudFormation stack creation has been successful

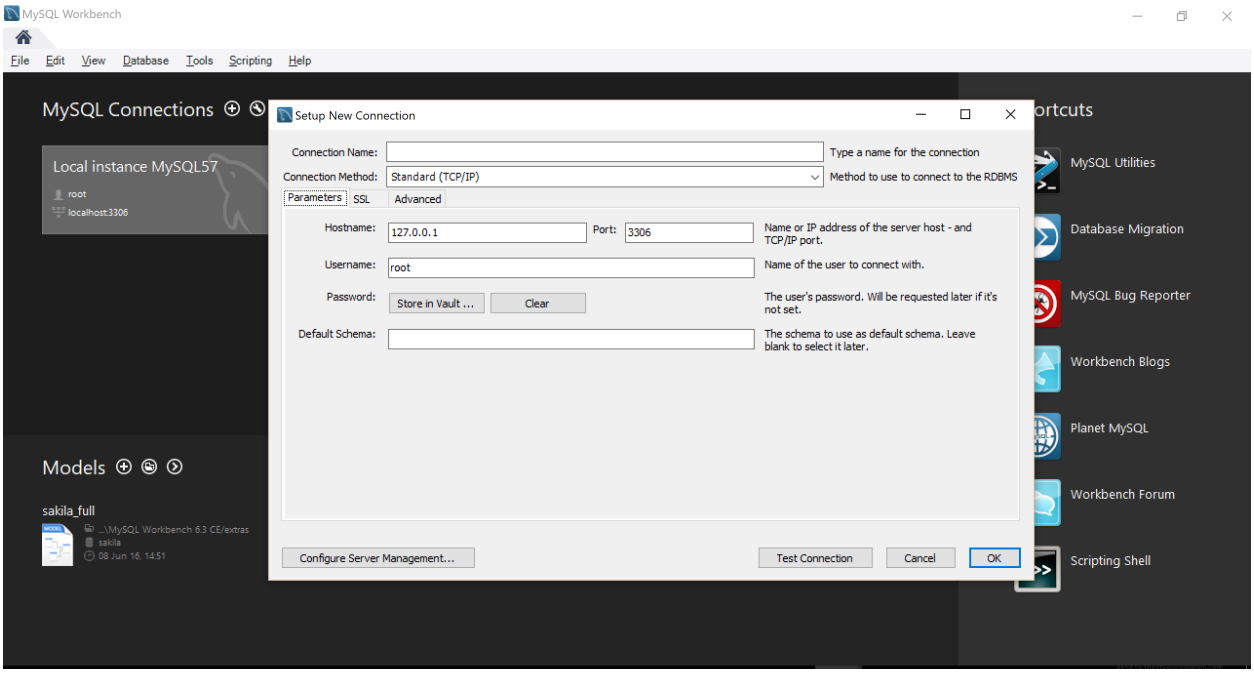


13. You can test out your newly created web servers by navigating to **Services —> EC2 —> Load Balancers**. Copy the full DNS name from the load balancer and put it in your web browser. The hello page with the Slalom logo should appear.

Step 11: Move Data into RDS

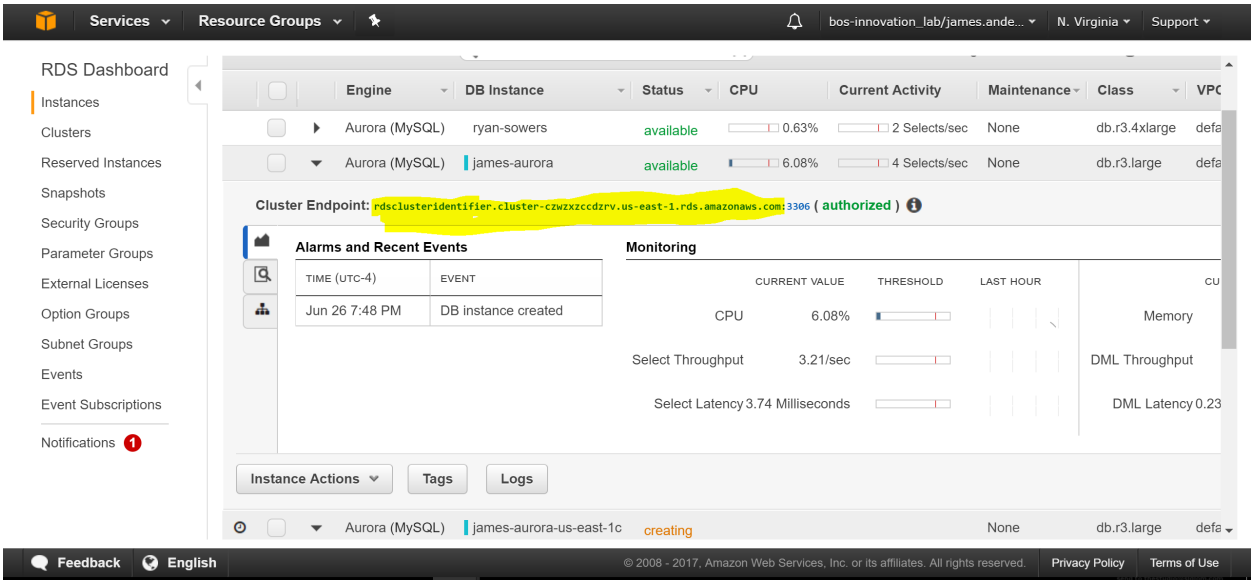
1. Once your stack is up and running, on your local machine, open up **MySQL Workbench**

2. In the Connections screen, **hit the + button** to add a new connection, and the **Setup New Connection** screen will appear



3. Give your connection a name (“My Aurora DB”, for example)

4. In the hostname, add the public endpoint of the RDS cluster that was created in **Step 10**. You can find that in the RDS management console, by following **Services → RDS**



5. Change the username to the username that you set up in **step 10**.

6. Click on the “Store in Vault…” button, and enter the password you created in **step 10**.

7. Click on the **“Test Connection”** button, and you should receive a success message

8. Once you have the connection set up, click on “OK”, then select the connection to initiate a new session

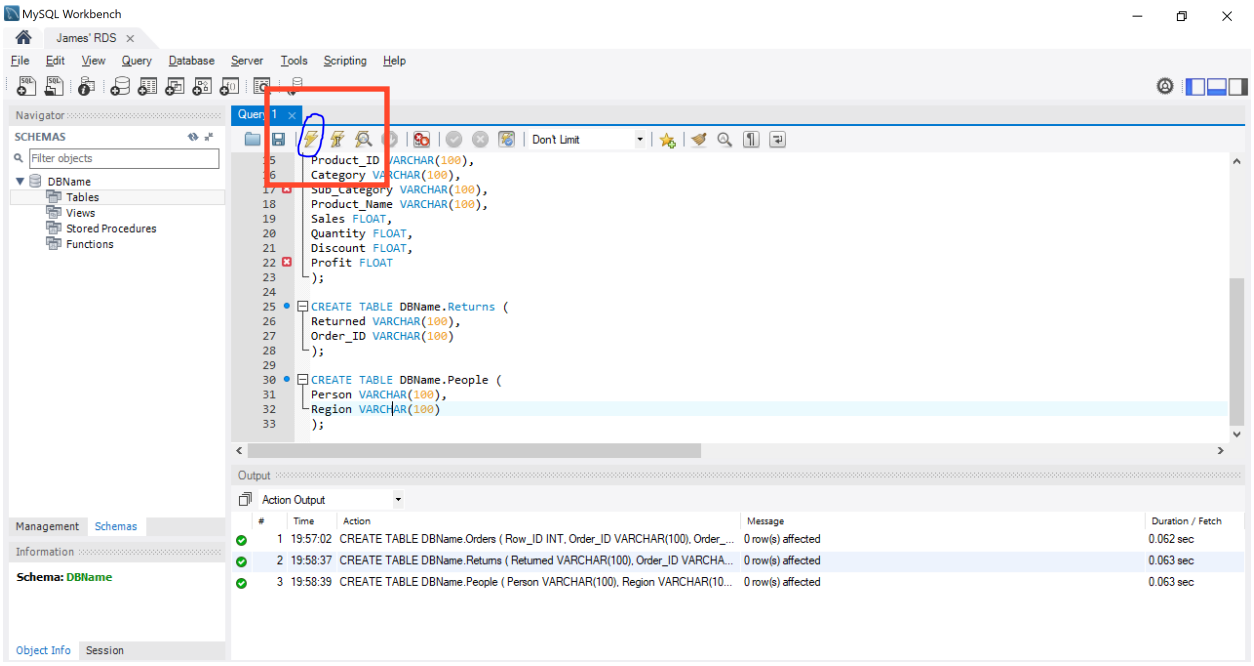
9. In the query window, copy and paste the following DDL (Data Definition Language) script into the editor:

```
CREATE TABLE DBName.Orders (
Row_ID INT,
Order_ID VARCHAR(100),
Order_Date DATE,
Ship_Date DATE,
Ship_Mode VARCHAR(100),
Customer_ID VARCHAR(100),
Customer_Name VARCHAR(100),
Segment VARCHAR(100),
Country VARCHAR(100),
City VARCHAR(100),
State VARCHAR(100),
Postal_Code VARCHAR(100),
Region VARCHAR(100),
Product_ID VARCHAR(100),
Category VARCHAR(100),
Sub_Category VARCHAR(100),
Product_Name VARCHAR(100),
Sales FLOAT,
Quantity FLOAT,
Discount FLOAT,
Profit FLOAT
);

CREATE TABLE DBName.Returns (
Returned VARCHAR(100),
Order_ID VARCHAR(100)
);

CREATE TABLE DBName.People (
Person VARCHAR(100),
Region VARCHAR(100)
);
```

10. To execute all the queries, click on the **lightning bolt**



11. Before we can copy data from S3, we need to associate the IAM roles created in step 7 to our RDS Cluster. In order to do that, you must login to the Management Console, and click on RDS

12. From there, click on the “Clusters” tab on the left, and highlight the new cluster you have created.

13. Click on “Manage IAM Roles” at the top, and add both the IAM roles created in step 7 to the cluster, and click done

14. Once all the tables are created and the IAM roles have been associated, we can now copy the data from a public S3 bucket over to the database. In order to this, execute the below SQL statements:

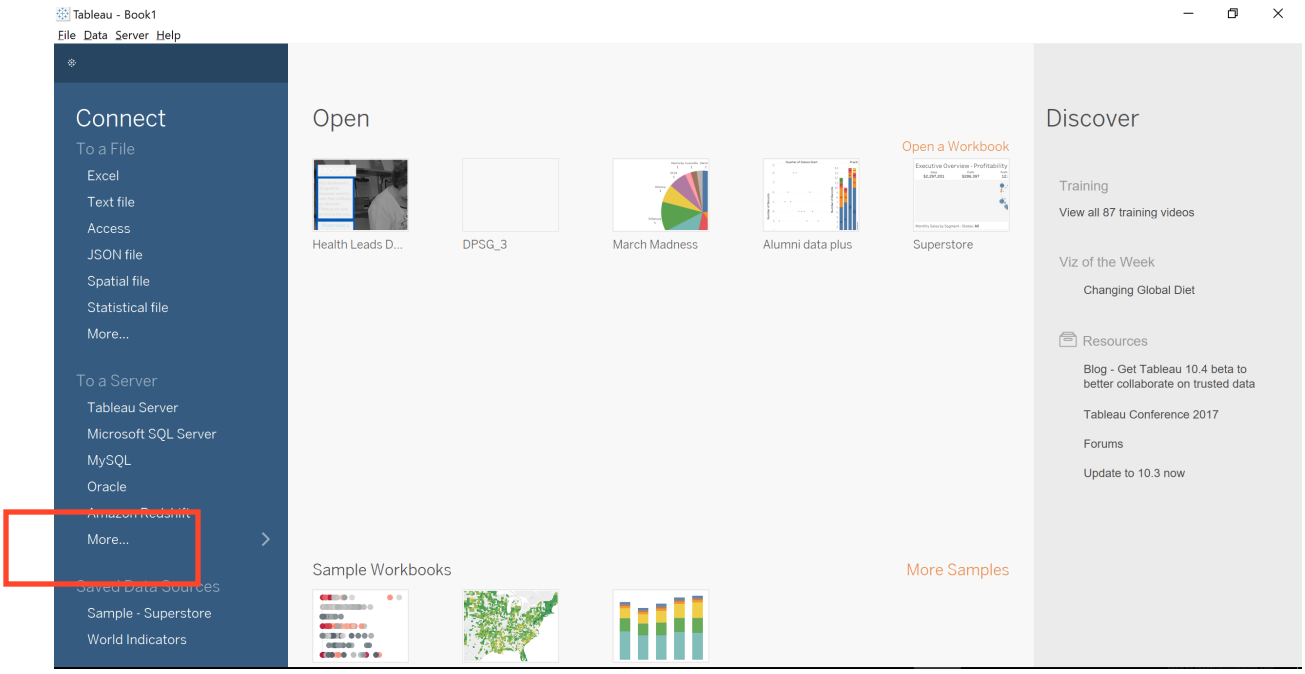
```
LOAD DATA FROM S3 's3://aws-slalom-brown-bag-sample/Orders.csv'
INTO TABLE DBName.Orders
IGNORE 1 LINES;
LOAD DATA FROM S3 's3://aws-slalom-brown-bag-sample/Returns.csv'
INTO TABLE DBName.Returns
IGNORE 1 LINES;
LOAD DATA FROM S3 's3://aws-slalom-brown-bag-sample/People.csv'
INTO TABLE DBName.People
IGNORE 1 LINES;
```

15. Once these queries are finished running, the data will be in your tables for querying!

Step 12: Create Tableau Data Source

1. Now that you have data in RDS, you can use it to build Tableau Dashboards for your teams! In order to begin, open up Tableau Desktop on your machine

2. Once you open up Tableau Desktop, the main screen will look like below:

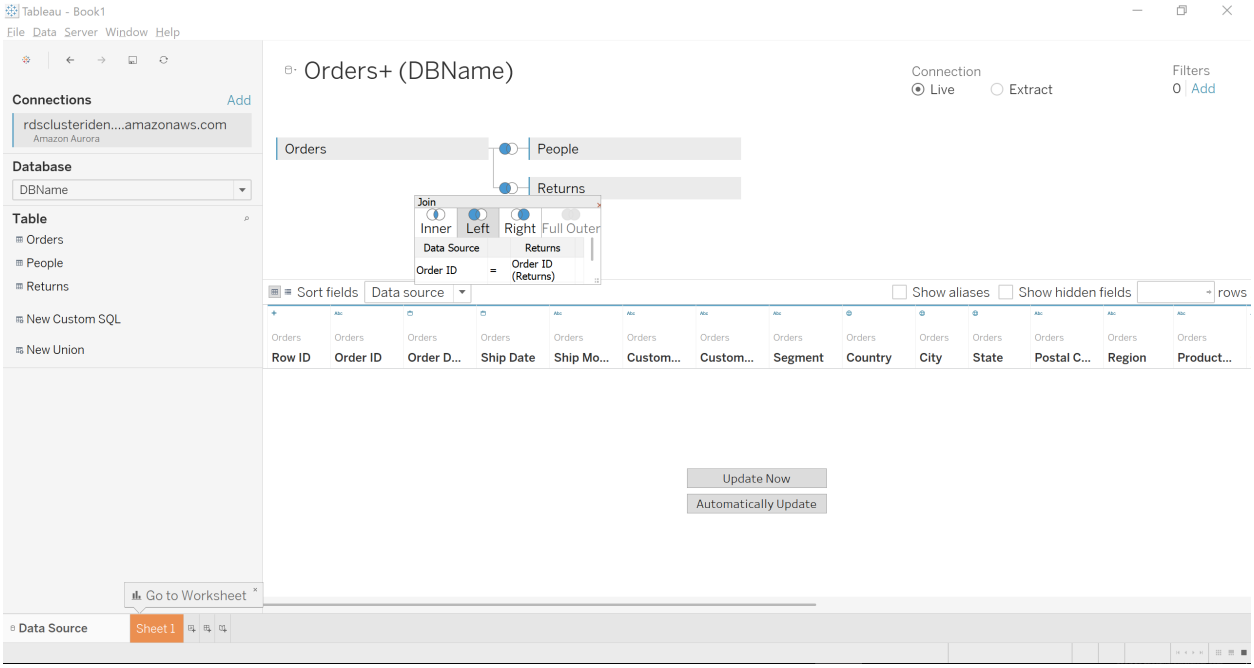


3. In order to connect to your Aurora Database, click on **"More…"** under the **To a Server**, and select the "Amazon Aurora" option

4. Using the same connection information used to set up the MySQL workbench connection, sign into the Aurora DB instance

5. Once you get to the Data Source setup window, select the DBName database on the left drop down, and the 3 tables you set up in the previous step will show up

6. First, pull in the "Orders" table to the Data Source pane. Then, add the "Returns" and "People" tables. Tableau will automatically set up the join conditions, but they will be Inner Joins. In order to change them to Left Outer Joins, simple click on the bubbles, and select the Left Join (shown below):

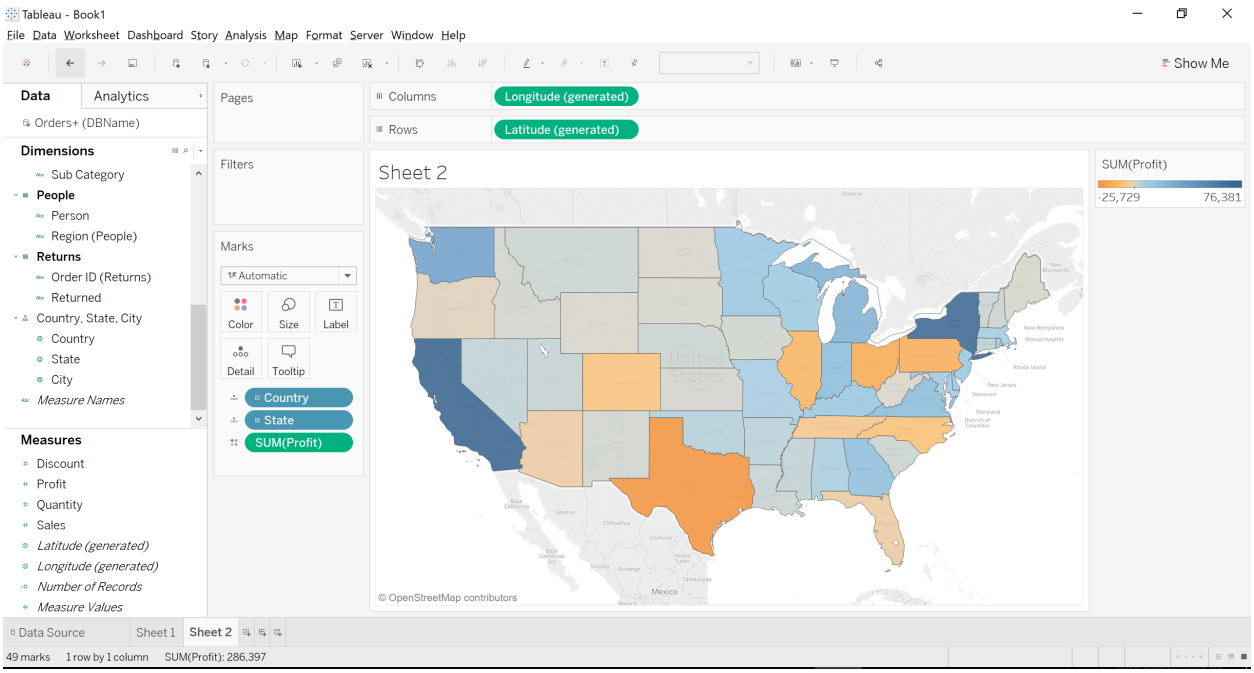


7. Once you have your data source set up, click on **Sheet 1** to start visualizing!

Step 13: Create Tableau Dashboard

1. When you get to **Sheet 1**, pull the following fields into the sheet in order to create a map of the USA, with a heat map showing highest profit:
    a. Profit to “Color”
    b. Country to “Detail”
    c. State to “Detail”
    d. Under the “Show Me” heading at the top left, select the “Filled Map” option

2. If correct, the corresponding map will look like the below map:



3. Once you have this map on a sheet, click on “**Dashboard/New Dashboard**” at the top pane

4. From the left, Drag “**Sheet 1**” into the Dashboard page to add the sheet to the Dashboard page.

5. Once you have your dashboard ready, you can publish it to your new Tableau Server!