Slalom AWS Introduction Workshop

Lab 2 Instructions

Introduction

Welcome to Lab 2 of the Slalom AWS Introduction Workshop! This document outlines 13 steps, walking you through the creation of the AWS resources required to extend and improve upon the architecture created in Lab 1.

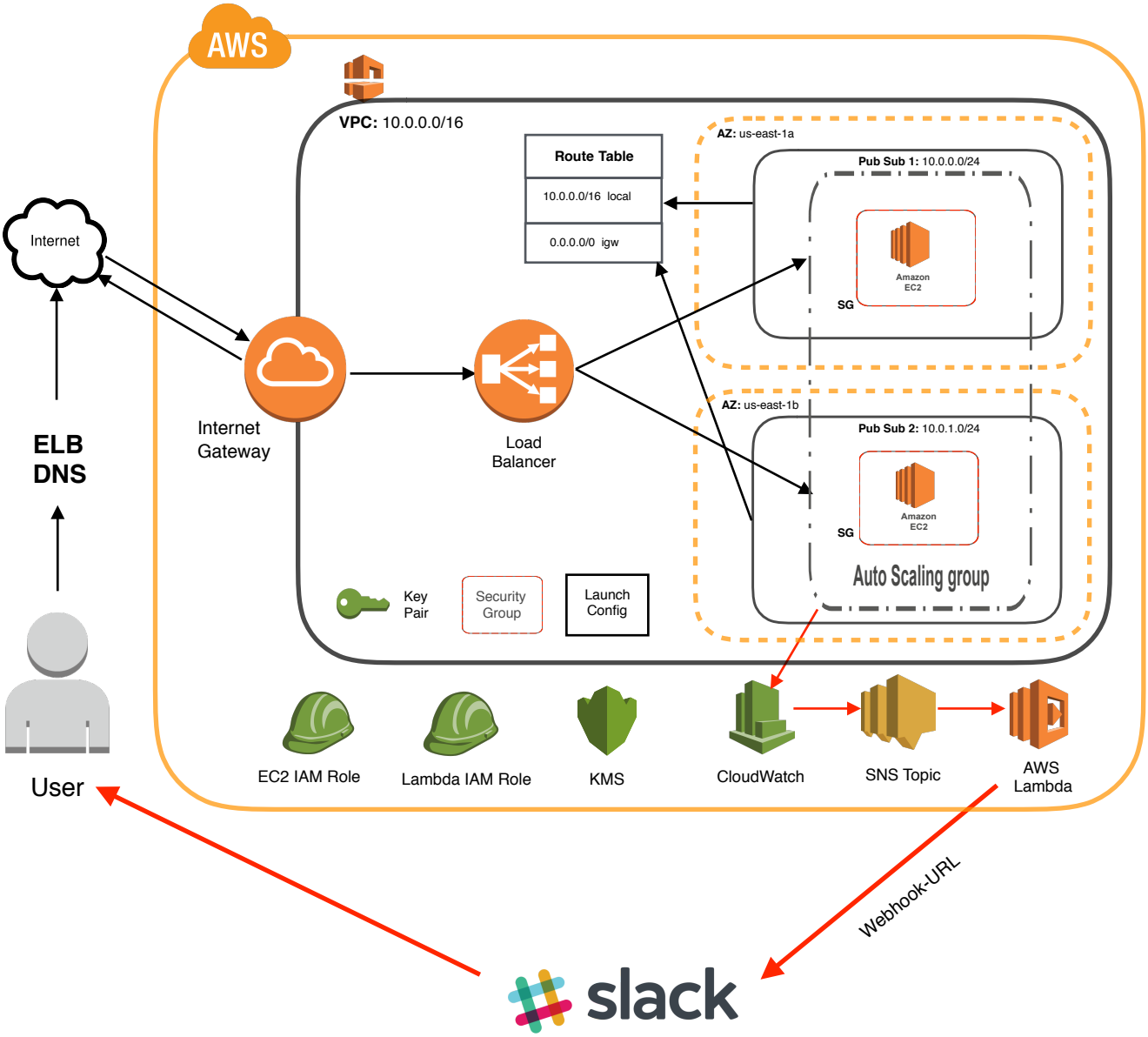
In this lab we will be using services such as AWS Simple Notification Service (SNS), AWS Key Management Service (KMS), AWS CloudWatch and AWS Lambda to create an alert from your autoscaling group that will post to your own personal Slack channel.

At the end of each step (excluding Step 1) 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 or Todd 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.

Lab 2 - Final Architecture



Step Overview

Step 1: Logging in to the AWS Console

This step simply directs you to log in to your account and select the appropriate AWS Region

Step 2: Create your own private Slack channel

Slack is a cloud-based team collaboration tool. Although on the surface it may seem similar to any other messaging platform, it's capabilities extend far beyond that. In this lab we will see that Slack has deep integrations with a number of technologies, including AWS. In this step, you create your own private Slack channel.

Step 3: Add the Webhooks-Integration to your private Slack channel

As mentioned above, Slack has a number of deep integrations with third party tools. Within this step, you will install the Webhooks-Integration within your Slack channel. The resulting webhook URL (a URL that is unique to your Slack channel) can be used by these third party tools to post messages and statuses to Slack.

Step 4: Create an SNS Topic

Amazon Simple Notification Service (SNS) is one of AWS's oldest services. SNS is a fast, flexible messaging service that allows you to push notifications to a number of different AWS services, email users or third party services.

Step 5: Create an encrypted key using AWS Key Management Service (KMS)

AWS Key Management Service (KMS) is a service that makes it easy to create and manage encryption keys that you can use to encrypt your data. In this step, you will create a new encrypted key that will be used to encrypt the webhook URL generated in Step 3

Step 6: Create an IAM Role for running AWS Lambda functions

Remembering back to Lab 1, we used AWS Identity and Access Management (IAM) to create an IAM Role that we assigned to our EC2 instances in the autoscaling group. As a refresher, AWS IAM Roles and Policies can be used to help restrict how an AWS Resource can communicate with other AWS Resources. In this Step, we will create a

new IAM Role that we will eventually assign to an AWS Lambda function that we will create later on in the Lab.

Step 7: Use the AWS IAM Policy Generator to create and assign an IAM Policy

We will use the IAM Policy generator to generate an IAM policy that we will apply to the IAM Role we just created in Step 6. This policy will contain a permission which will allow it to decrypt the key we created in Step 5.

Step 8: Update your Auto Scaling Group Instance Count

Step 9 will require us to create CloudWatch alarms for our Auto Scaling Group. In order to set up this alarm, we must have some active EC2 instances in our group. This step will increase the Min and Desired capacities to allow for alarm creation to occur in Step 9.

Step 9: Set up an AWS CloudWatch Alarm

AWS CloudWatch is a service that provides monitoring capabilities within AWS. You can monitor important statistics about a number of AWS services. In our case, we are going to create a CloudWatch alarm that will be monitoring the CPU utilization levels on the EC2 instances within your autoscaling group.

Step 10: Create and configure an AWS Lambda function using provided blueprints

AWS Lambda is a service provided to allow serverless compute capabilities. Instead of having an EC2 instance that runs all of the time, AWS Lambda allows you to create a code function that will be executed at certain times. In our case, we are using a Lambda function blueprint provided by AWS that contains code to send a notification to Slack.

Step 11: Recreate the ELB from Lab 1

As part of the cleanup for Lab 1, you were asked to delete the ELB. In this step, you will recreate it again so that we can test the functionality of our webpage once again.

Step 12: Increase the min and desired capacity of your AutoScaling Group

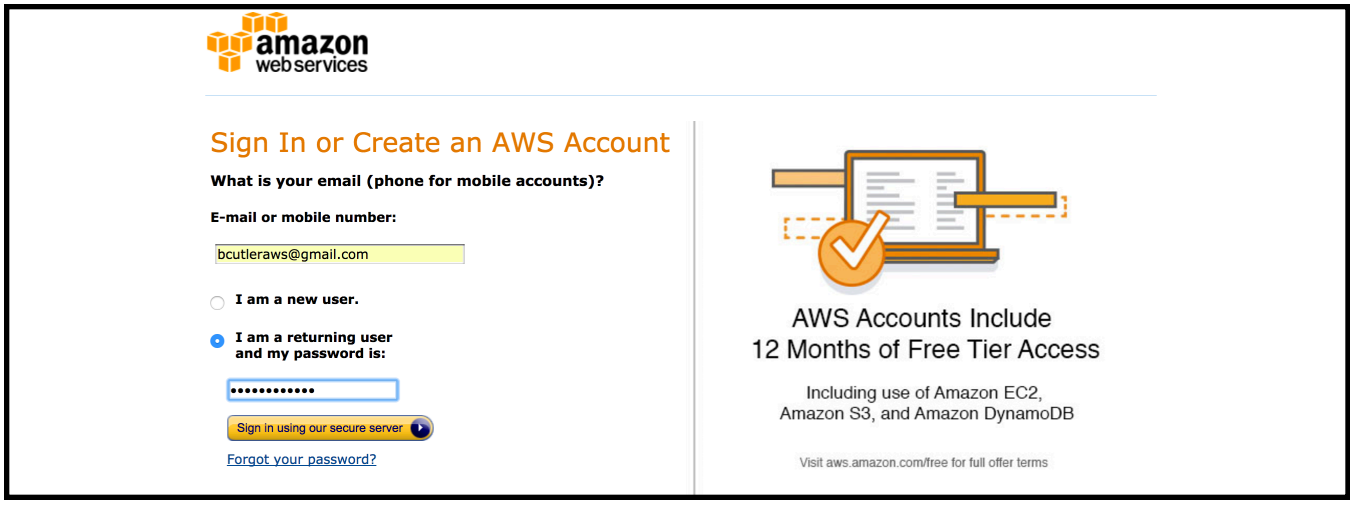
Also as part of the cleanup for Lab 1, you were asked to decrease the min and desired capacity values to 0. This ensured that any EC2 instances in your autoscaling group were terminated. By turning the value back up to 2, this will create 2 brand new EC2 instances as part of your autoscaling group.

Step 13: Test it out!

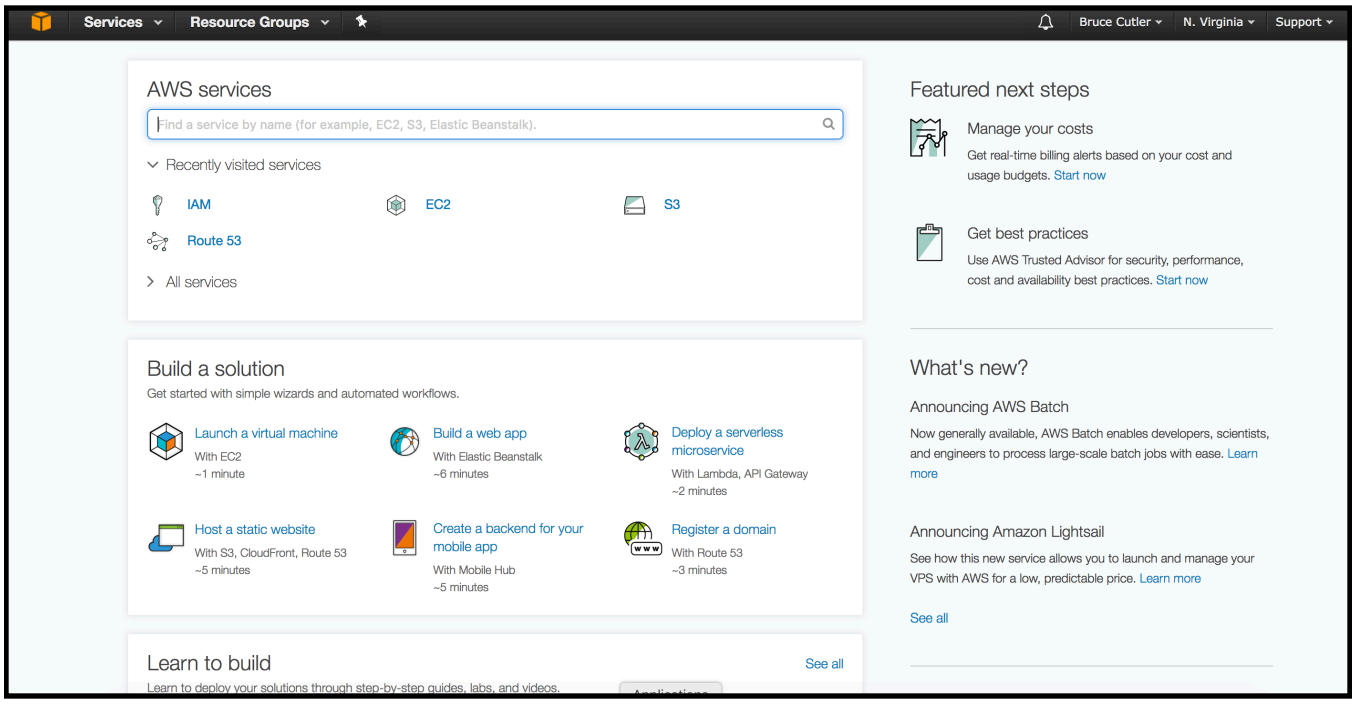
Once we've build all of the above components, we can use the DNS value from the load balancer in a browser to view our simple webpage.

Step 1: Logging in to the AWS Console

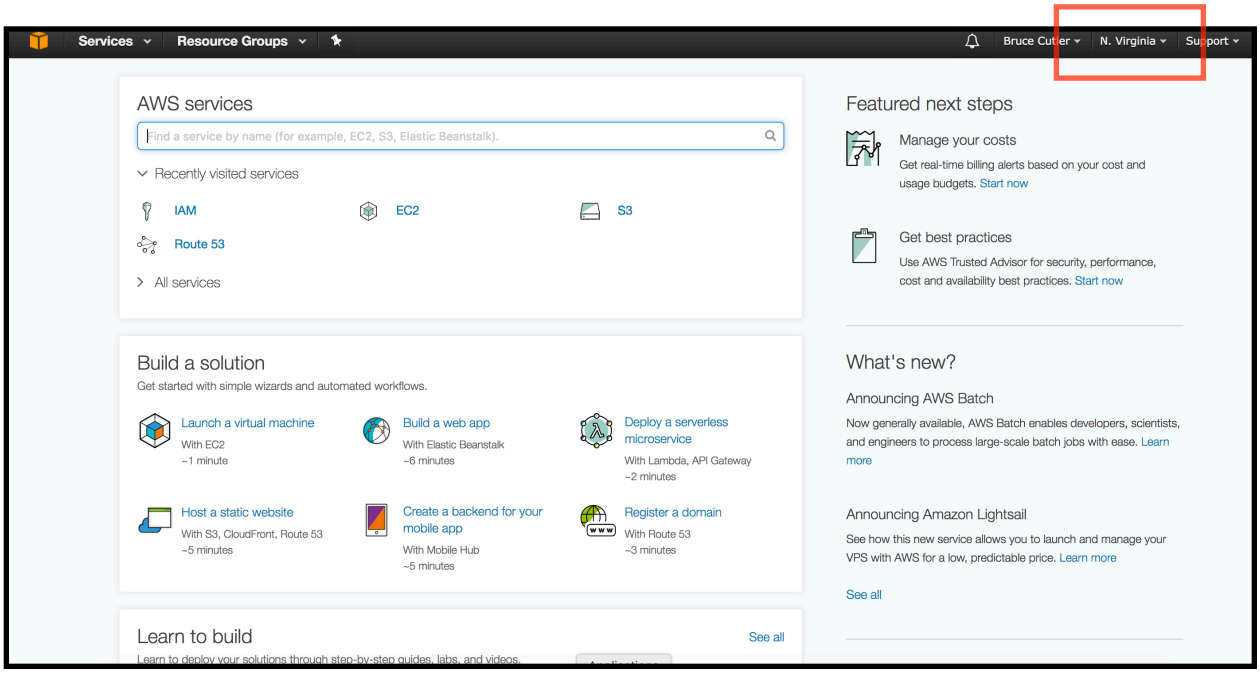
1. In a web browser, navigate to **https://console.aws.amazon.com/console/home**



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:

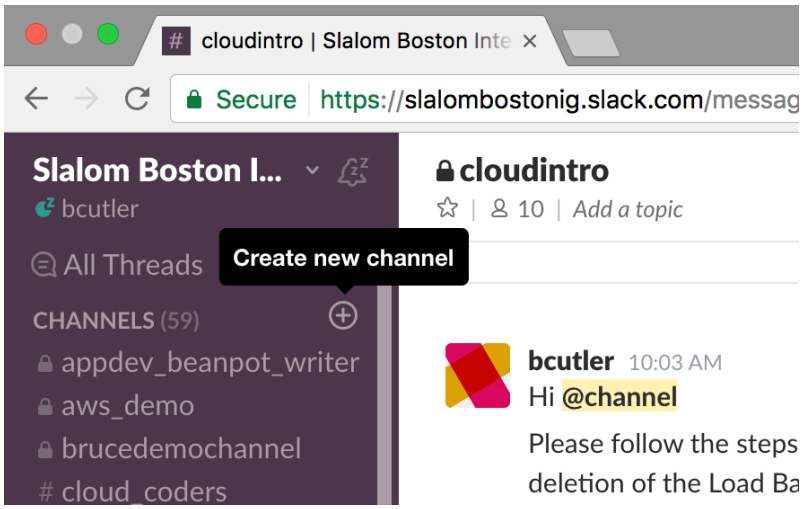


3. At the top right of the AWS Console screen, ensure that the **N. Virginia** is selected as your AWS Region



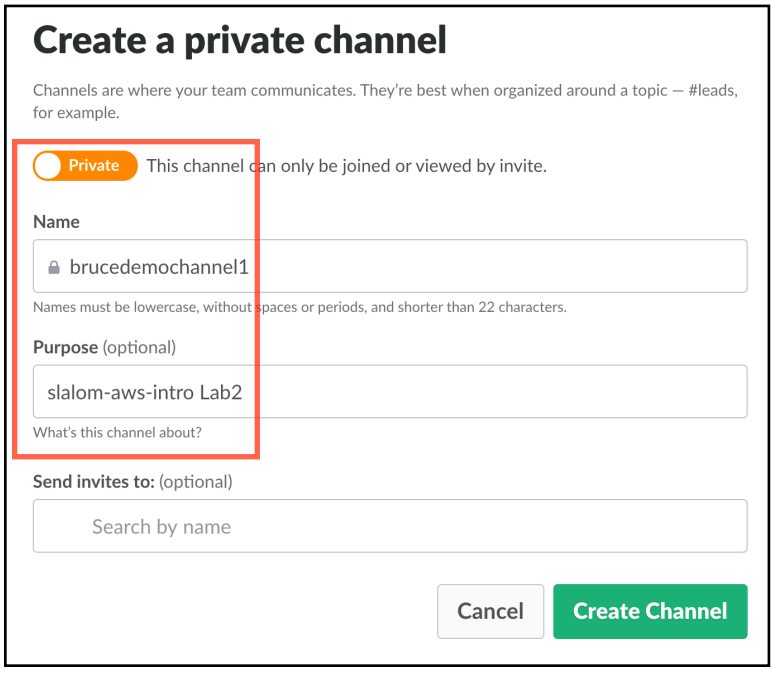
Step 2: Create your own private Slack channel

1. Whether you use Slack from your browser or using the Slack Desktop app, click on the **(+)** button beside Channels**:**



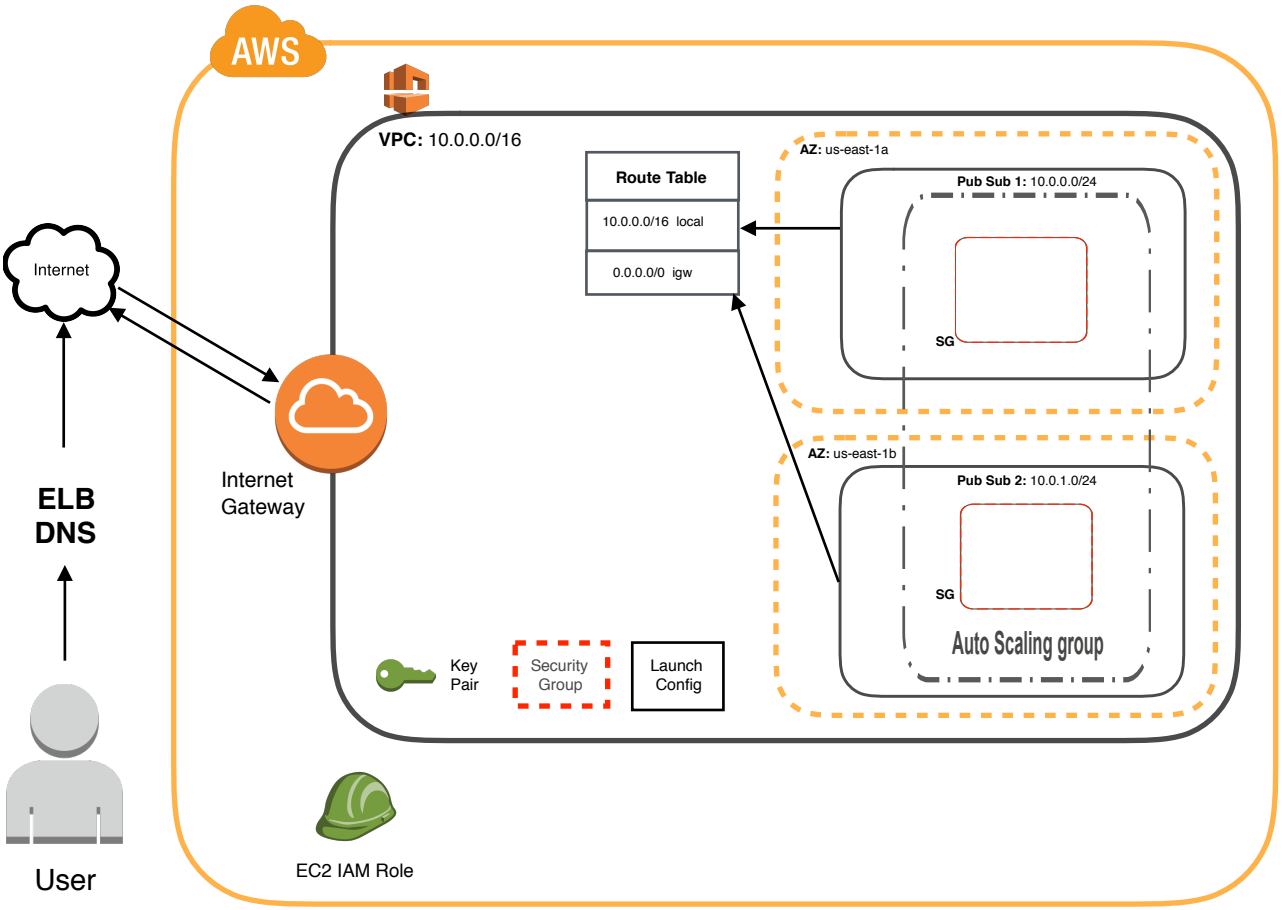
2. In the Create Channel screen that appears, ensure that you:

   - Change the channel type to **Private**
   - Give it a unique **Name**
   - Provide a **Purpose** description for the channel



3. Proceed to click **Create Channel** once the above 3 conditions have been satisfied

System Architecture: Step 2



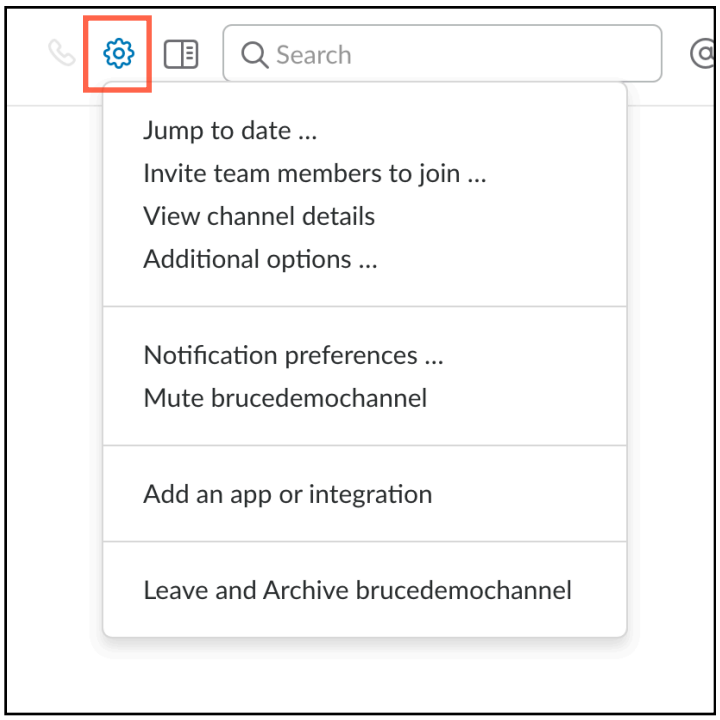
slack

Step 3: Add the Incoming WebHooks integration to your Slack Channel

Adding in the Incoming WebHooks integration will allow you to generate a URL that uniquely identifies your private Slack channel. This URL can then be used by other services, such as AWS, to push messages to that channel.

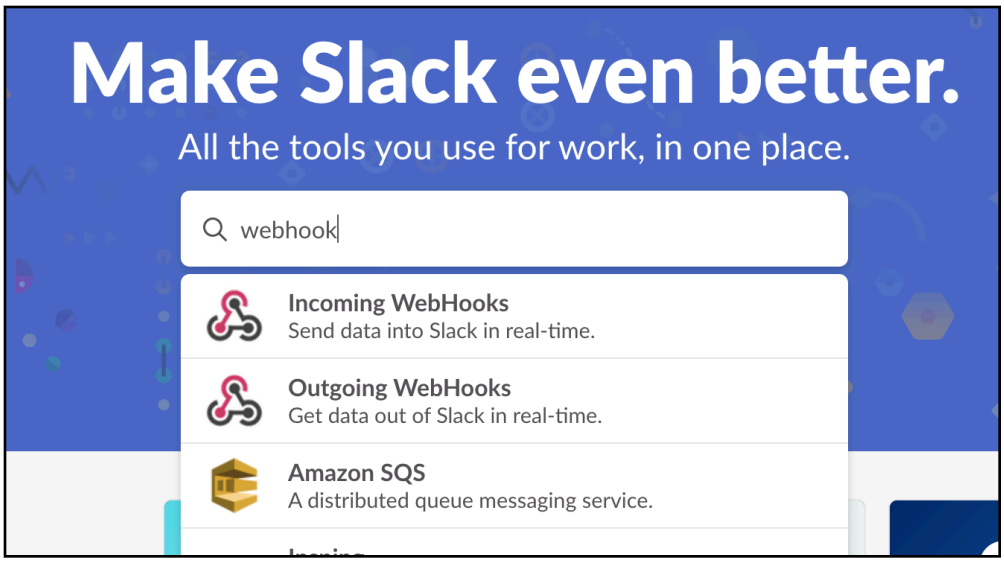
1. Ensure you have your new Slack channel selected from the list of available channels

2. Navigate to the top right of the Window and select the cog symbol, indicating settings



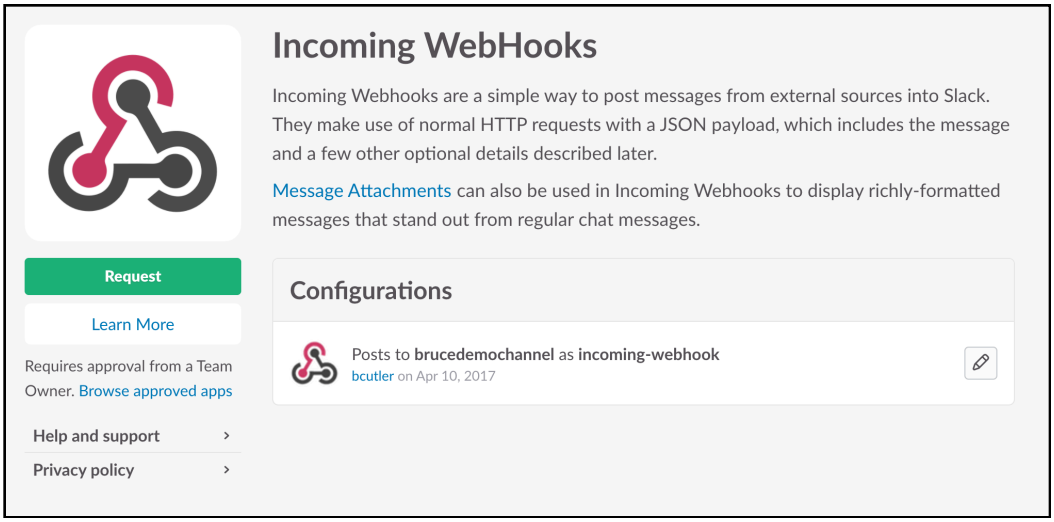
3. Select the **Add an app or integration** menu item

4. A new page will load with a search bar. Enter "**webhook**" and you should see the following:

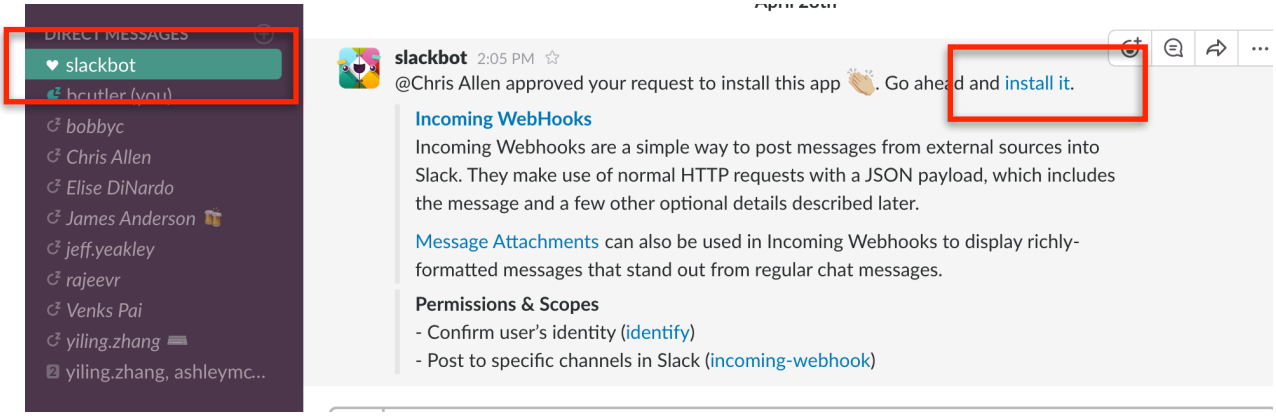


5. Click on **Incoming WebHooks** in the search results

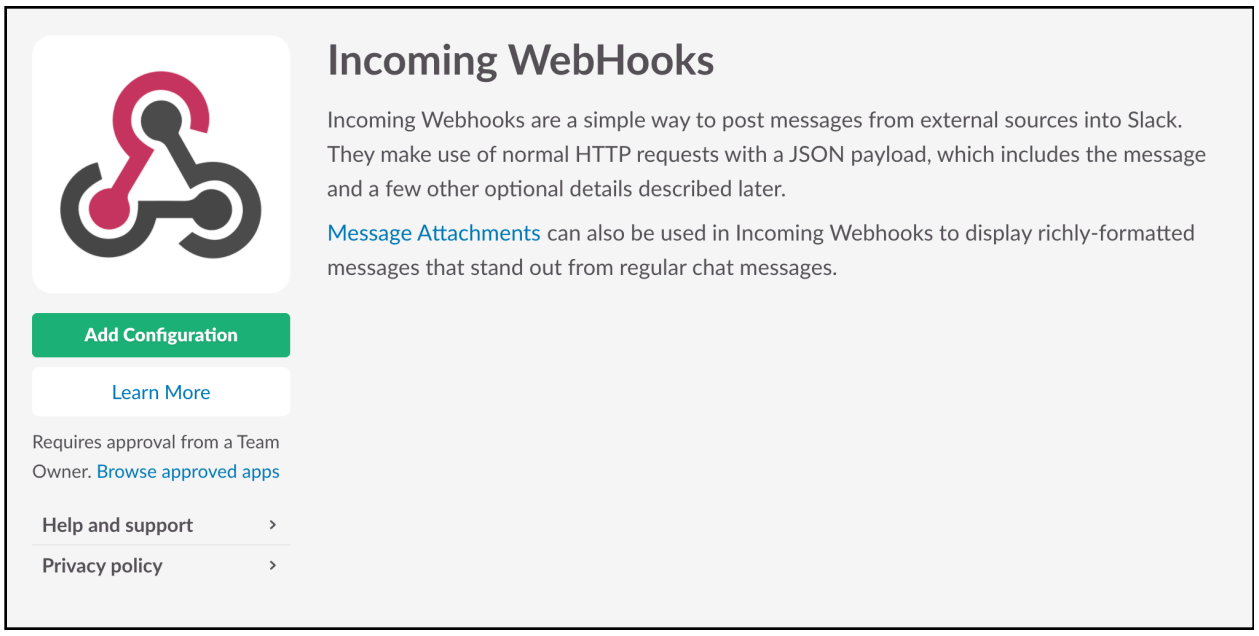
6. In the page that loads, select the green "**Request**" button. This will alert our Slalom Slack Team Owners that we would like to add this new functionality.



7. Once the Slack team owner has approved your request (I've alerted our Slack team owners, but this could take a few minutes), you will be notified by the Slackbot channel. Within that notification message there will be a link to Install the integration. Click on the **Install** link

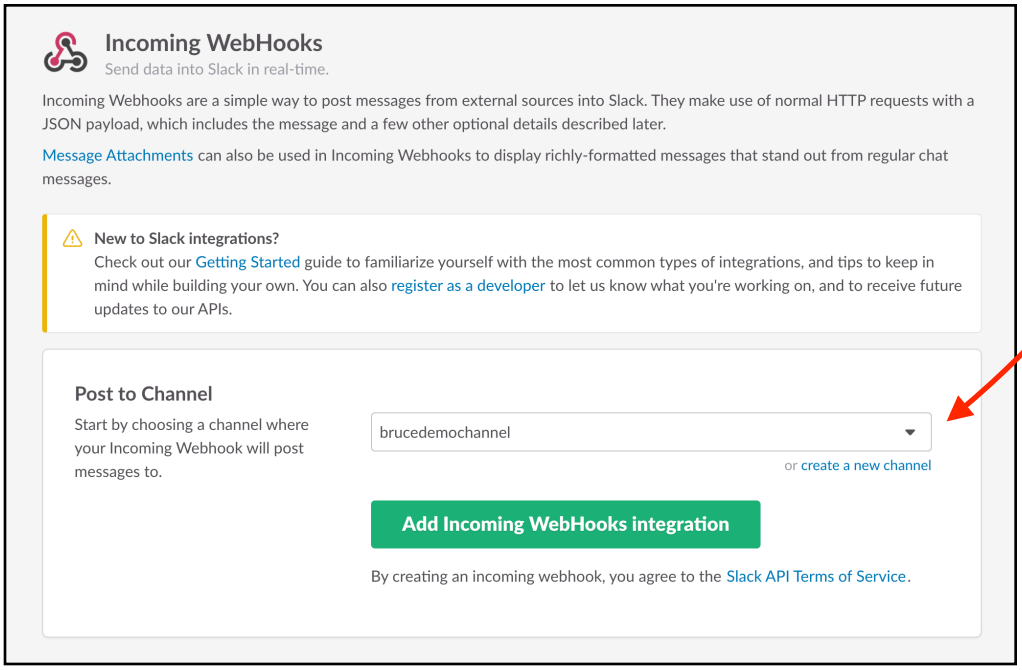


8. The following page will load. Click the green **Add Configuration** button



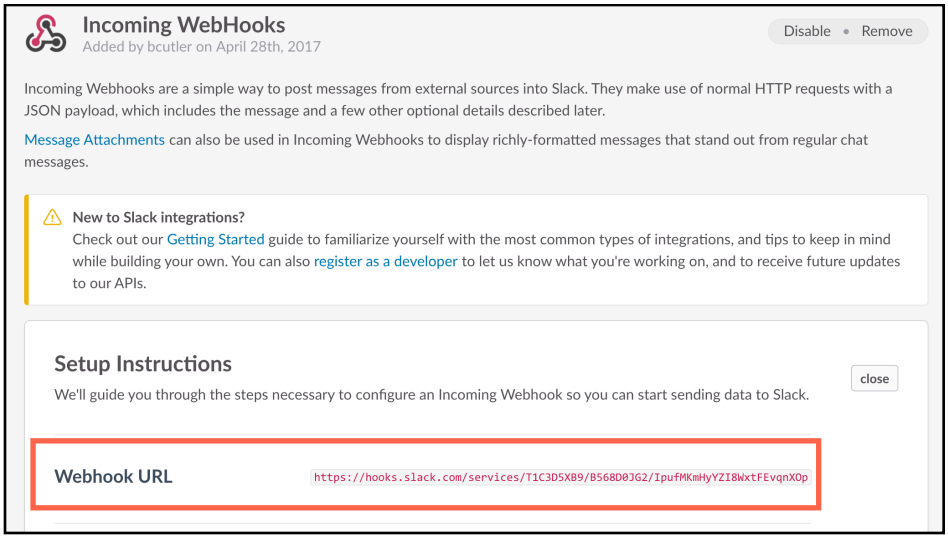
9. In the page that loads, ensure that you select your new channel created in **Step 2**. Private channels can be found towards the bottom of the dropdown list.

   Click the green **Add Incoming WebHooks Integration** button once you have your channel selected.



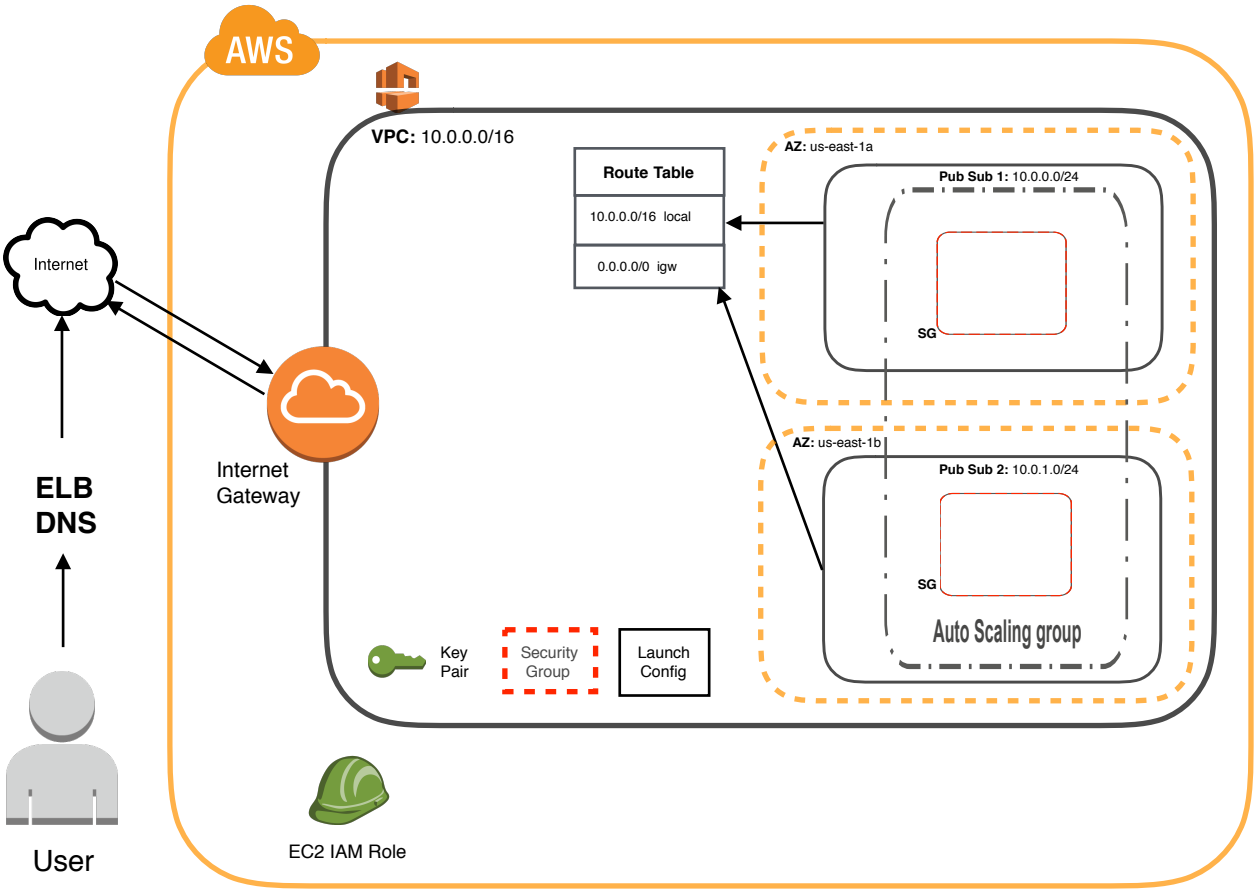
10. A configuration page will load which allows you to customize a number of different aspects of the WebHook integration you have just added.

    On this page, a particularly important piece of information is the **Webhook URL**, as highlighted below. This is URL uniquely identifies your Slack channel, allowing external services to post messages and statuses to Slack



11. Open up a blank text document (Notepad, Microsoft Word etc.) and copy the **Webhook URL** to it so that we can use it easily at a later stage of the lab.

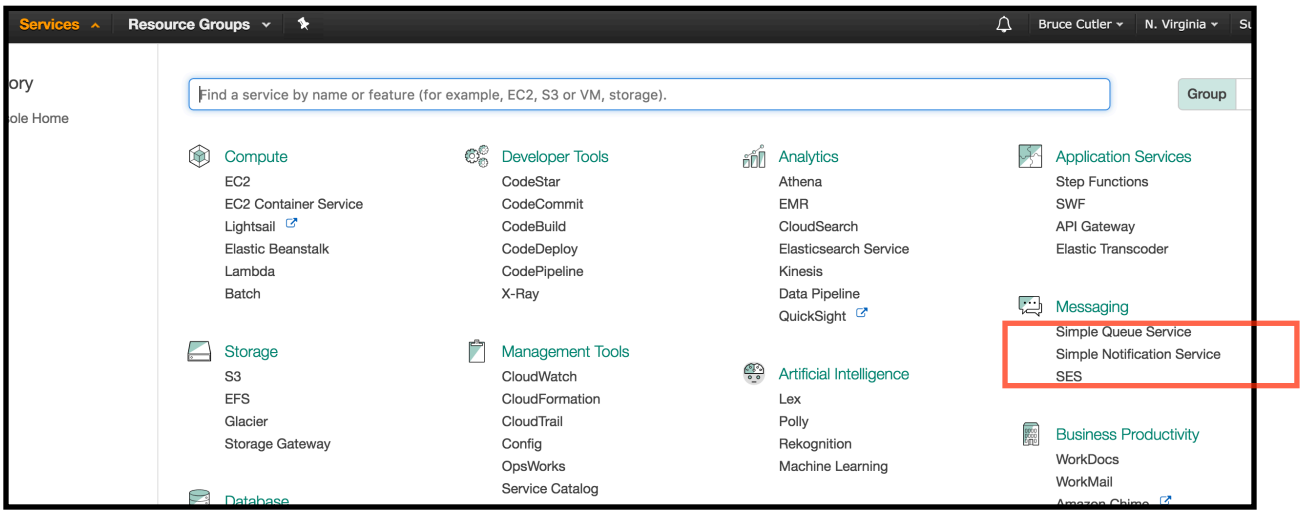
System Architecture: Step 3



Webhook URL: https://hooks.slack.com/….

Step 4: Create an SNS Topic

1. Navigate to **Services —> Simple Notification Service**

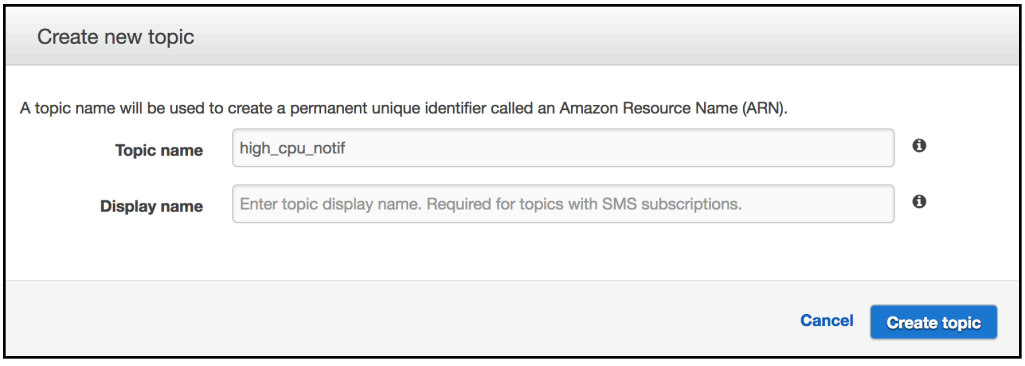


2. Click the blue **Get Started** button

3. Select the **Create Topic** link

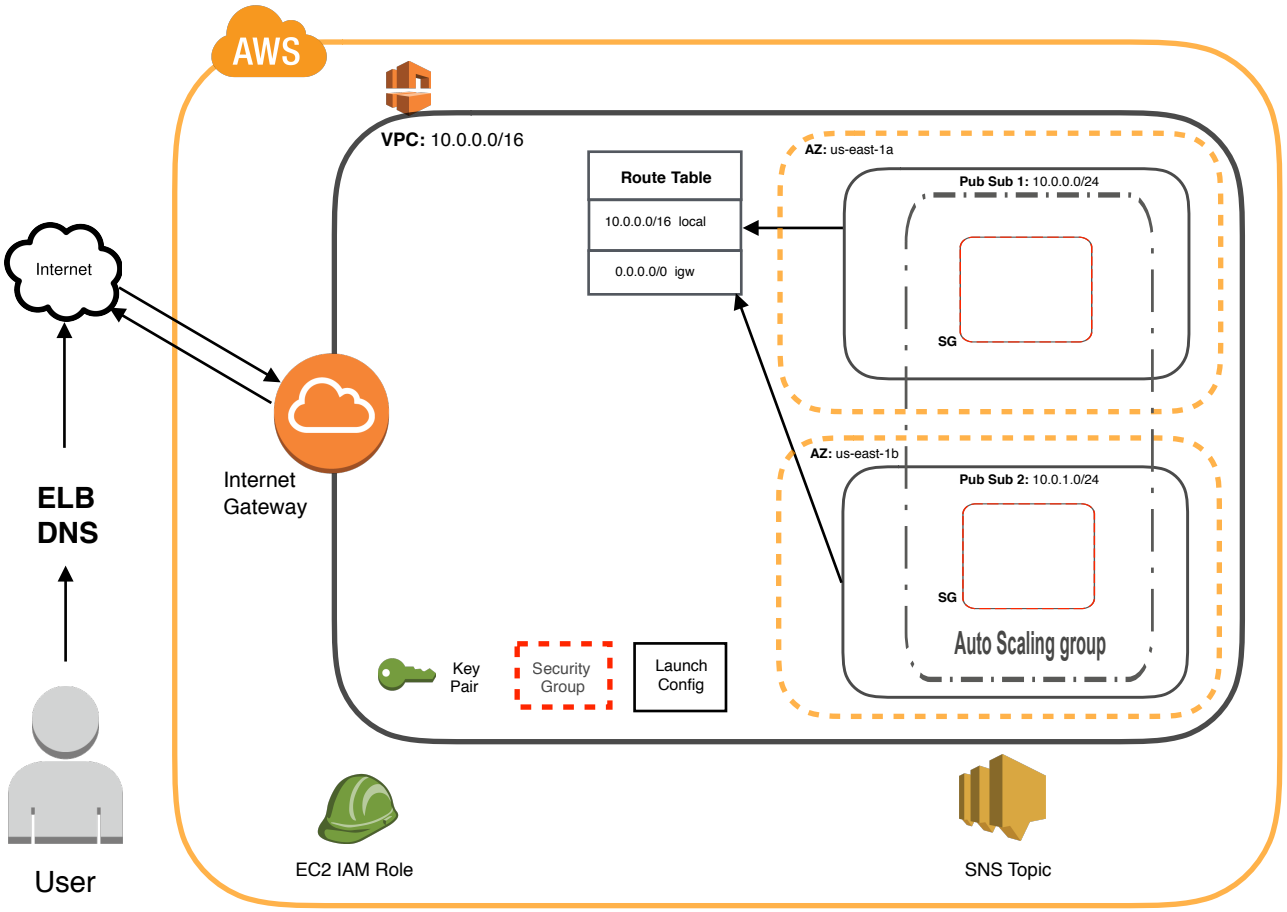
3. In the Create new topic popup that appears, enter the following and click the blue **Create Topic** button:

   Topic Name: high_cpu_notif
   Display name: < Leave as default value >



4. After the topic has been created, click on **Topics** in the left hand menu. You should see your new SNS Topic, **high_cpu_notif** listed

System Architecture: Step 4

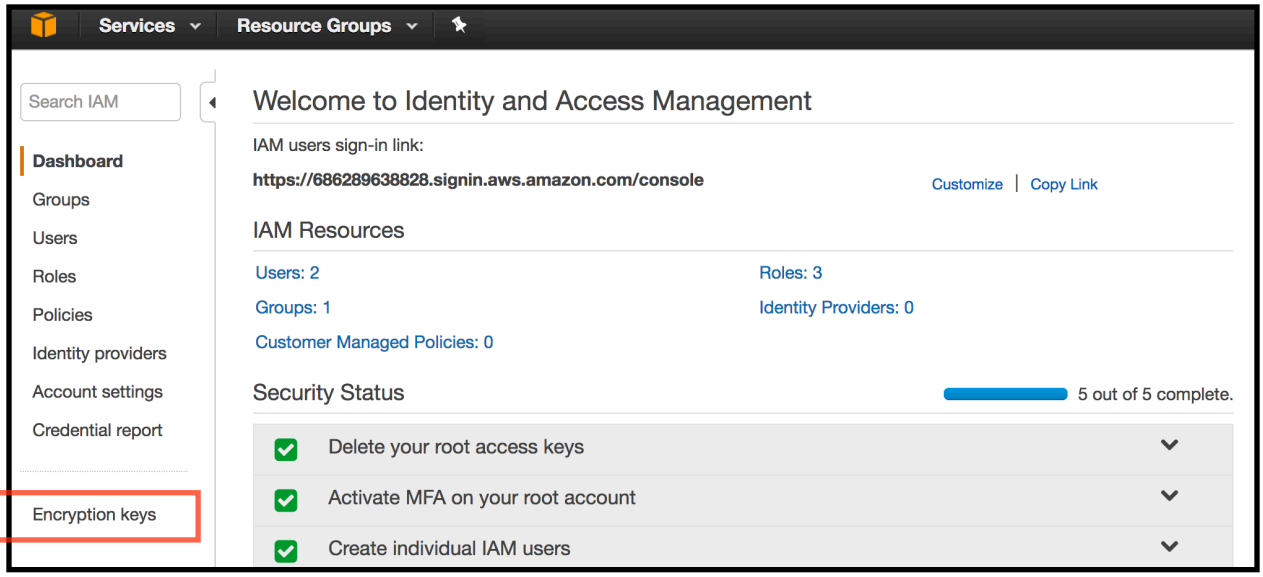


Webhook URL: https://hooks.slack.com/….

Step 5: Create an encryption key using Amazon Key Management Service (KMS)

1. Navigate to **Services —> IAM**

2. Select the **Encryption keys** link in the left menu

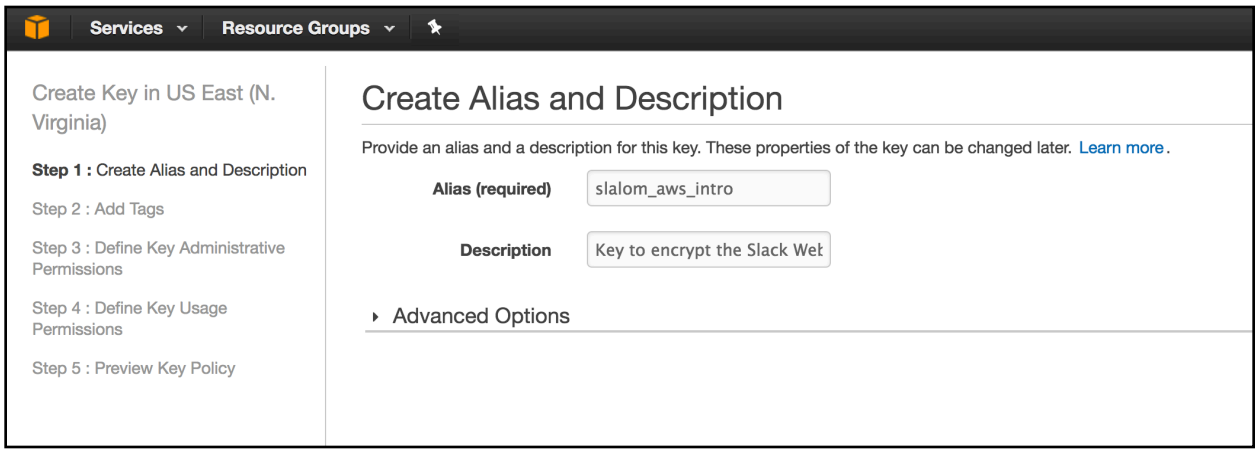


3. Select the blue **Get Started Now** button

4. Select the blue **Create Key** button

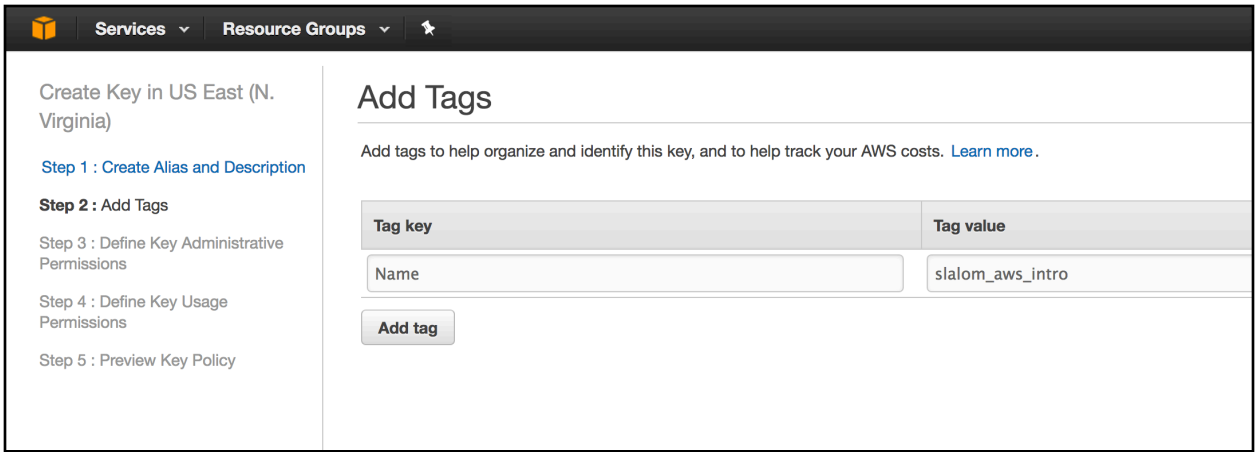
5. In **Step 1** of the wizard that appears , enter the following Alias and Description:

   Alias: slalom_aws_intro
   Description: Key to encrypt the Slack Webhook URL



6. In **Step 2**, enter the following tag:

   key = Name
   value = slalom_aws_intro

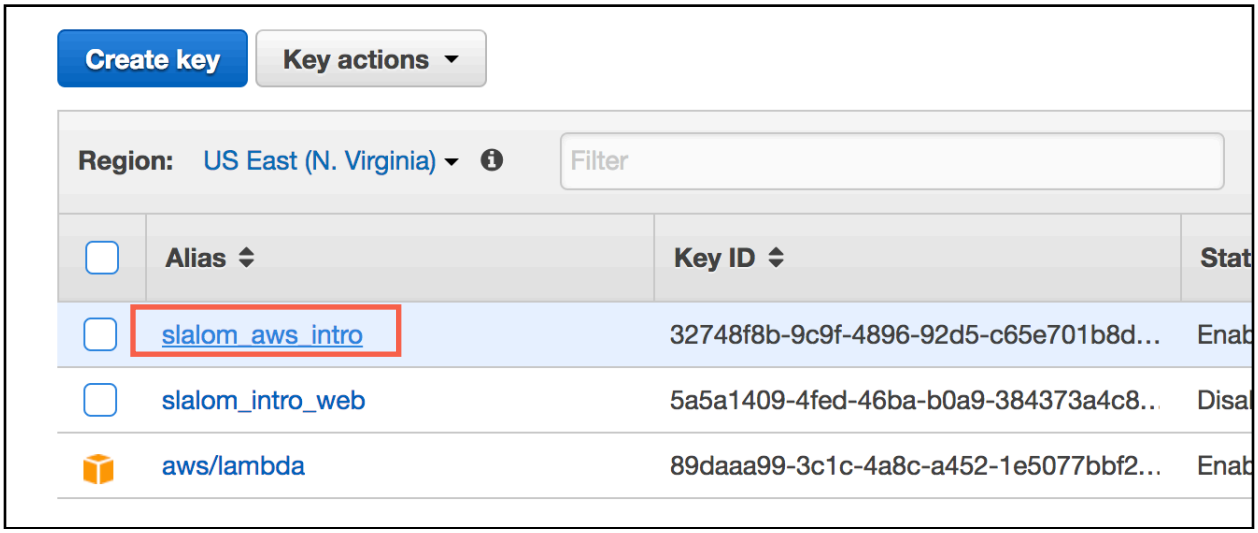


7. In **Step 3**, click the blue **Next Step** button

8. In **Step 4**, click the blue **Next Step** button

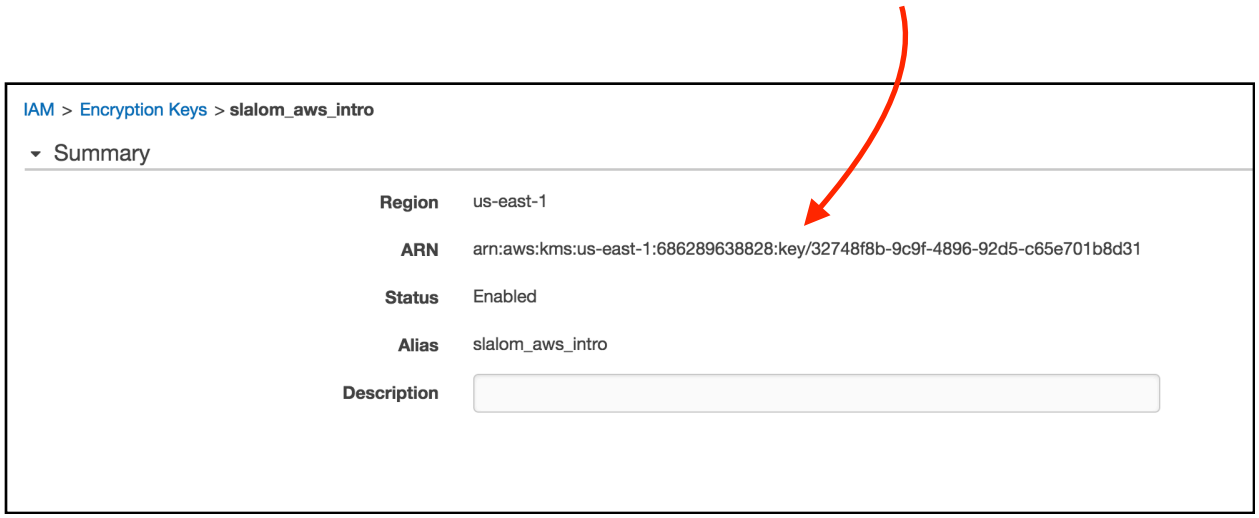
9. **Step 5** offers you a preview of the Key's attached policy. This simply defines how the key can be used by various users and AWS services. No action is required here, so click the blue **Finish** button to complete key creation.

10. You should see your newly created **slalom_aws_intro** key. Click on the key name in the Alias column

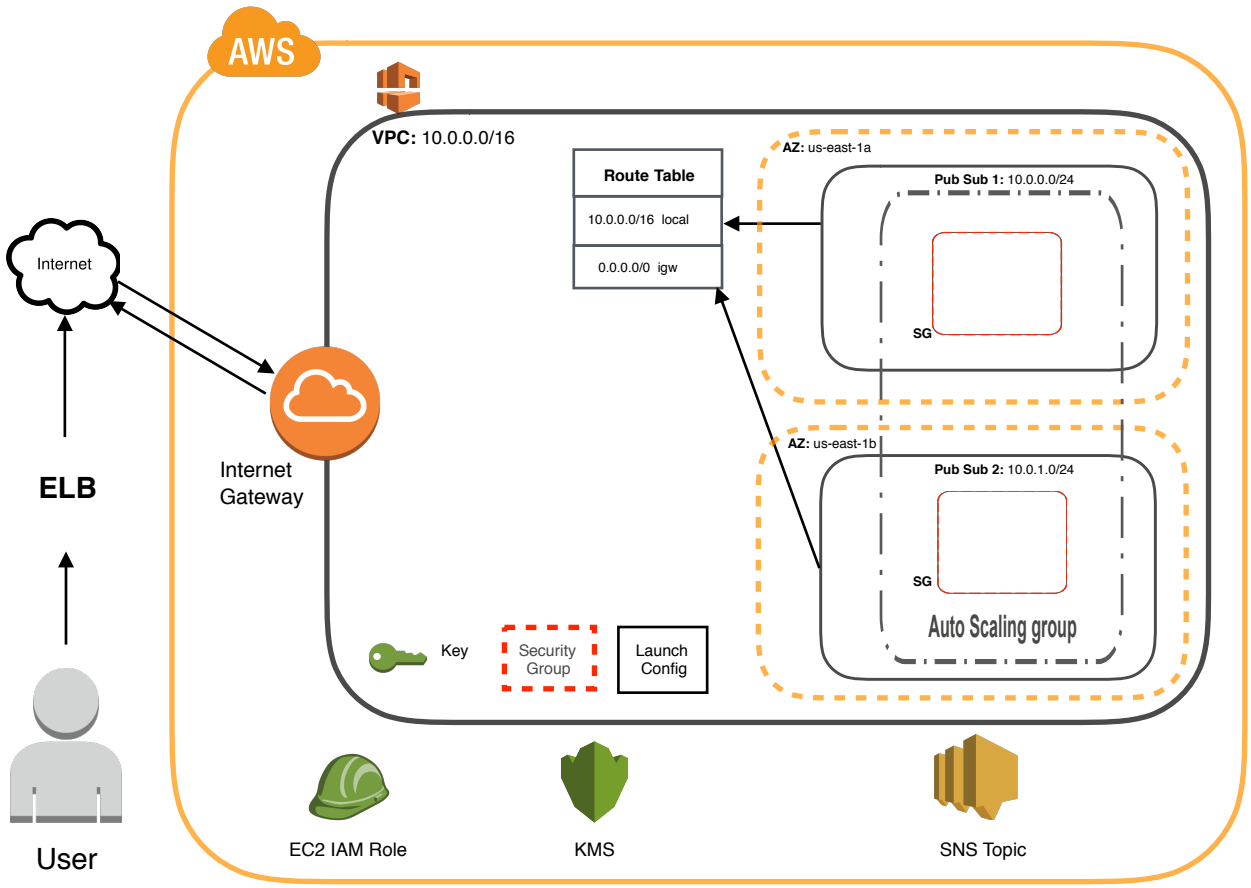


11. In the page that loads, **copy the value in the ARN field** “arn:aws:kms:us-east…..” to the text file that you were asked to create earlier (See Step 2 #11 for reference).

    Again, this is an important piece of text that will be used to configure another service later within the lab.



System Architecture: Step 5



Webhook URL: https://hooks.slack.com/….

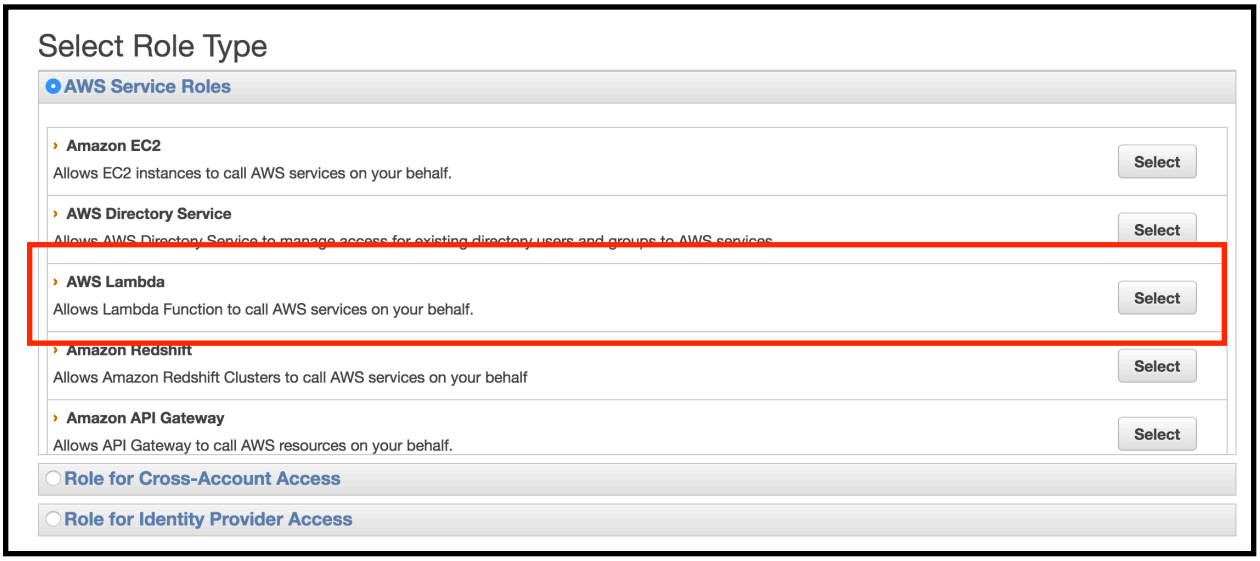
Step 6: Create An IAM Role for Running AWS Lambda Functions

1. Navigate to **Services —> IAM**

2. In the left menu, select **Roles**. You should see the slalom_web_server role that we created in Lab1

3. Select the blue **Create New Role** button

4. In the Select Role Type screen, click the **Select** button in the AWS Lambda section:

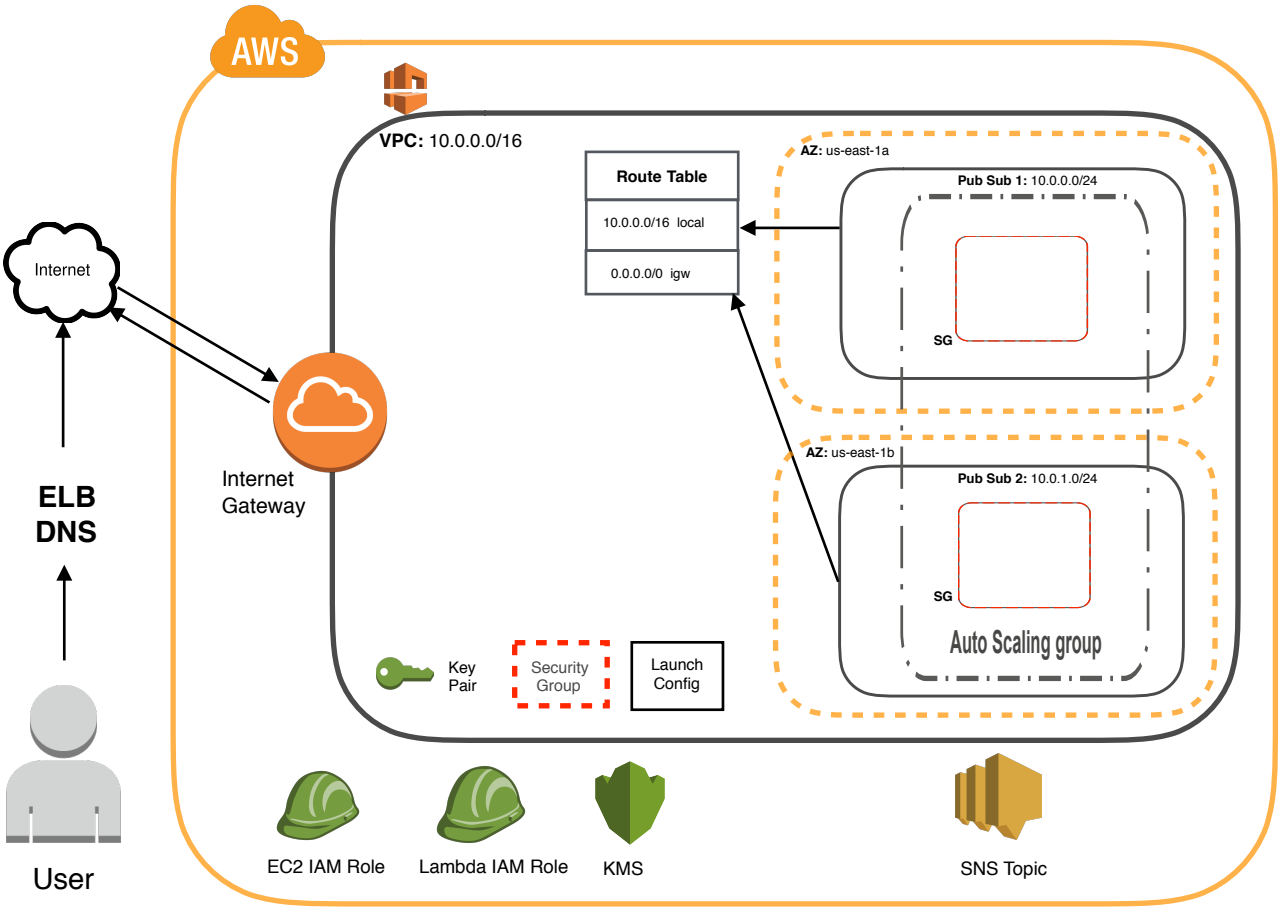


5. In the Attach Policy screen, don't select any policy. We will add this in ourselves in the following Step. Click the blue **Next Step** button:

6. Enter **slalom_lambda_role** as the Role Name and click **Next Step**

7. Click the **Create Role** button to create the **slalom_lambda_role** IAM Role

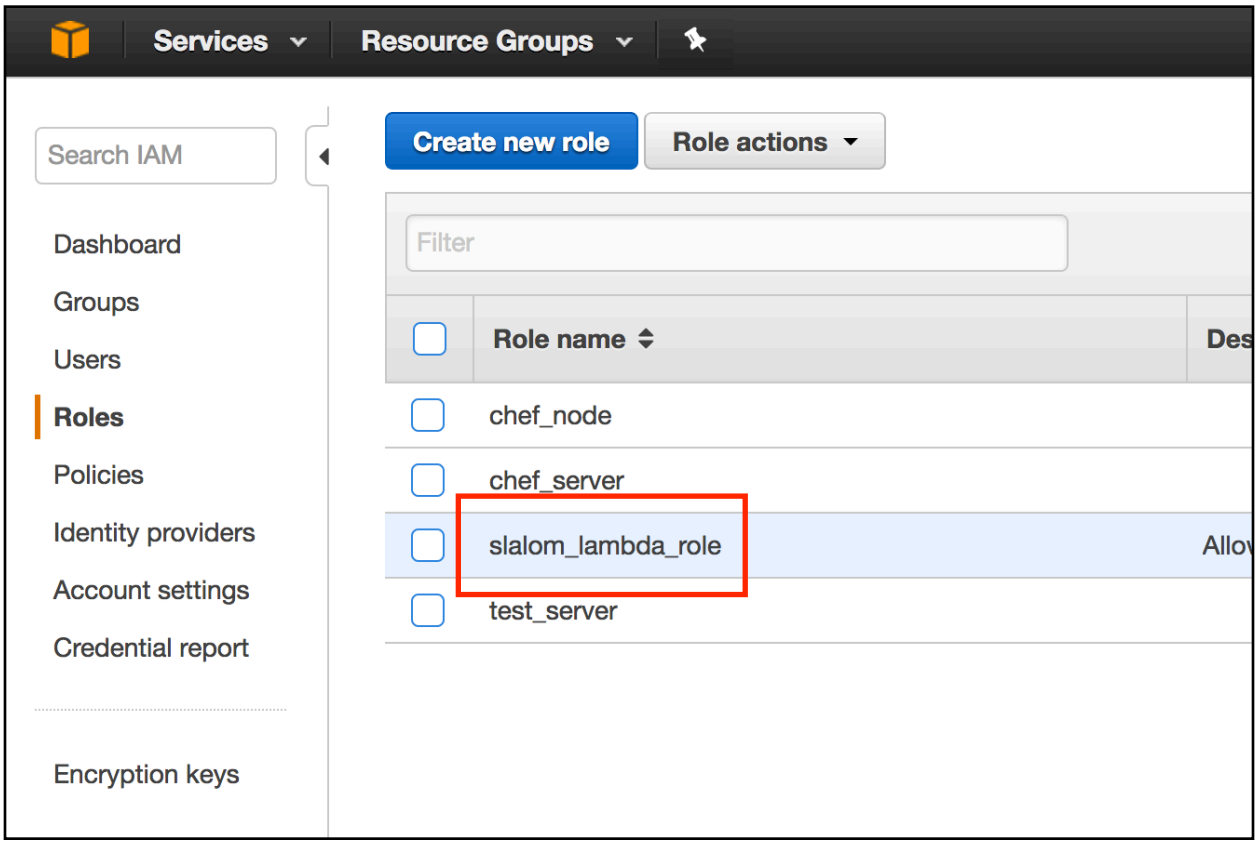
System Architecture: Step 6



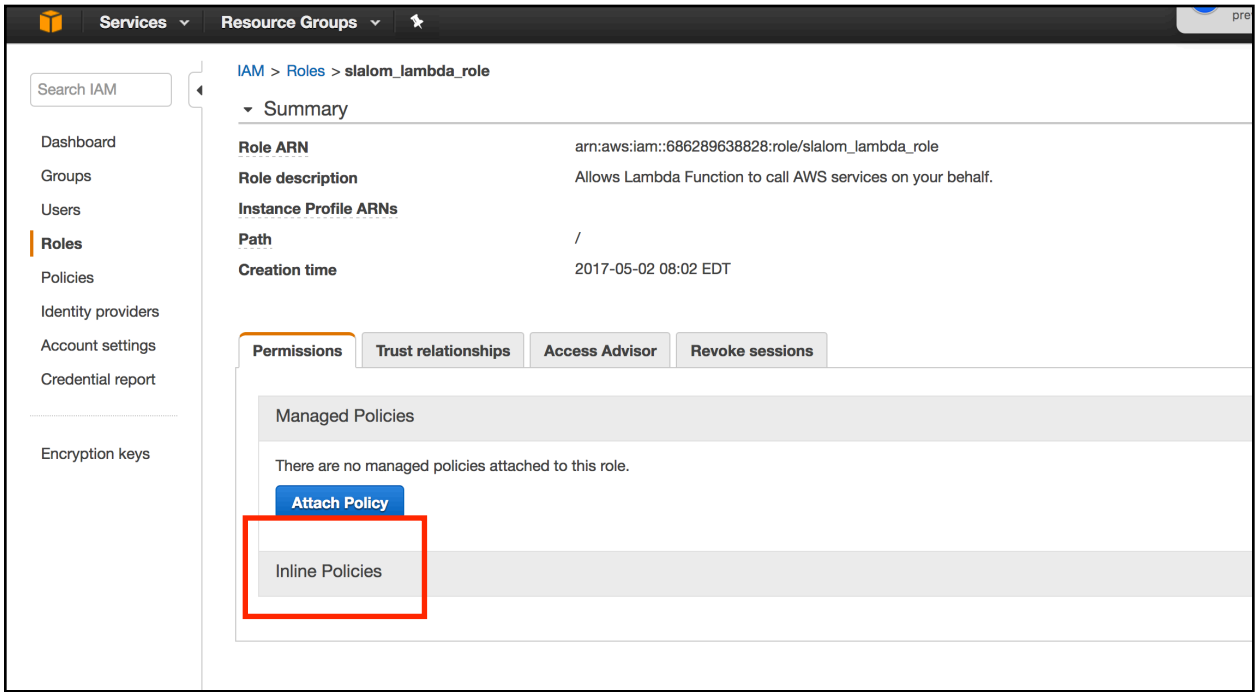
Webhook URL: https://hooks.slack.com/….

Step 7: Create and assign an IAM Policy to your new IAM Role

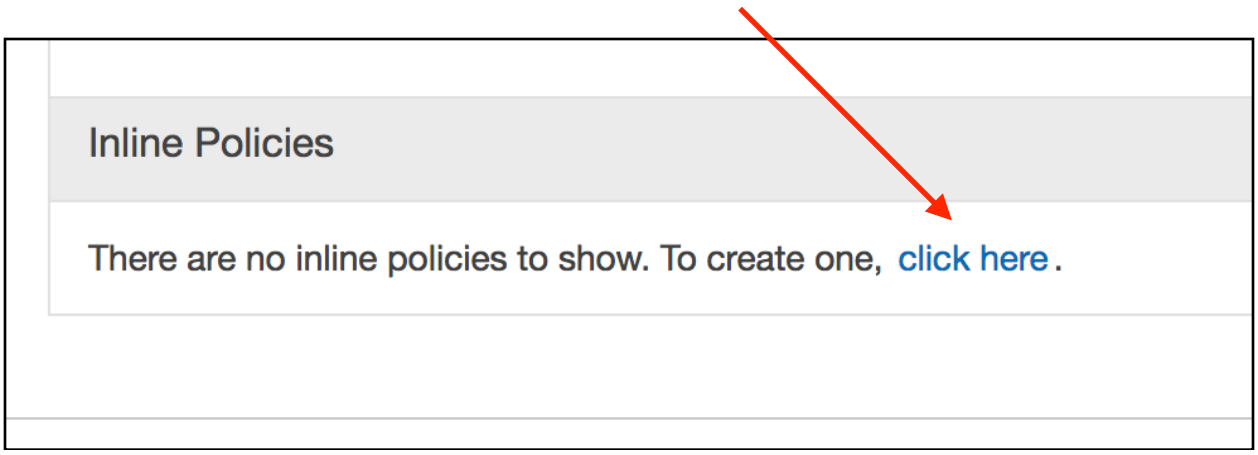
1. Navigate to **Services —> IAM**
2. In the left menu, click the **Roles** link
3. Click on the name of your new role, **slalom_lambda_role**



4. The following page should open. Click on the **Inline Policies** section as shown:



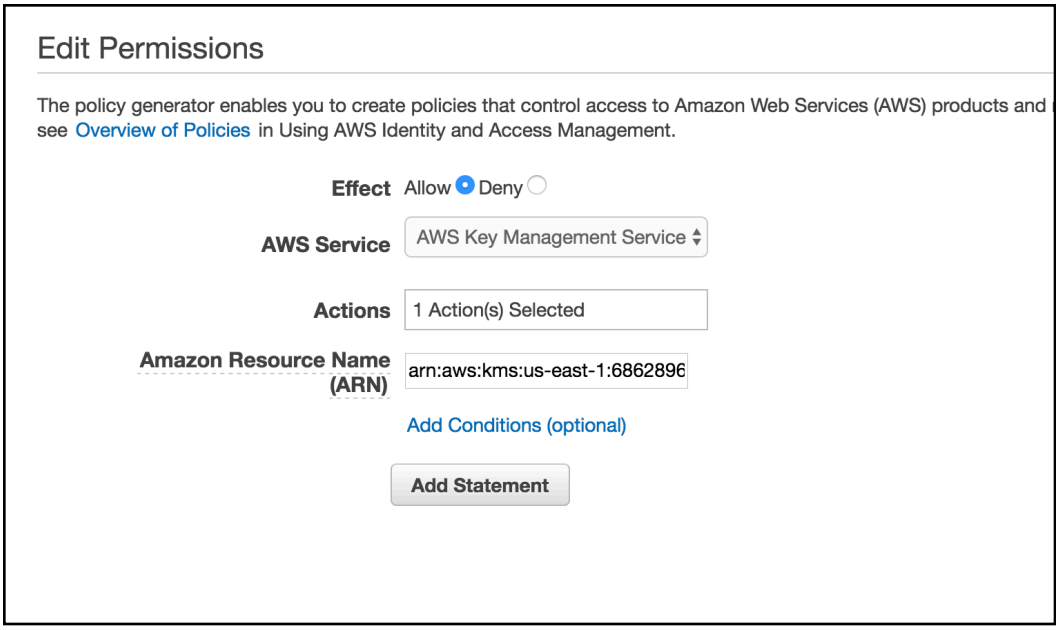
5. The message lets us know that we don't currently have any inline policies attached to the new Role you created in Step 6. Let's use the **click here** link to create one.



6. Keep the **Policy Generator** selected and click the grey **Select** button

7. In the next page, ensure the following are set:

   Effect: Allow
   AWS Service: AWS Key Management Service
   Actions: Decrypt
   Amazon Resource Name: < Paste in the ARN for your Key from Step 5 #11 >

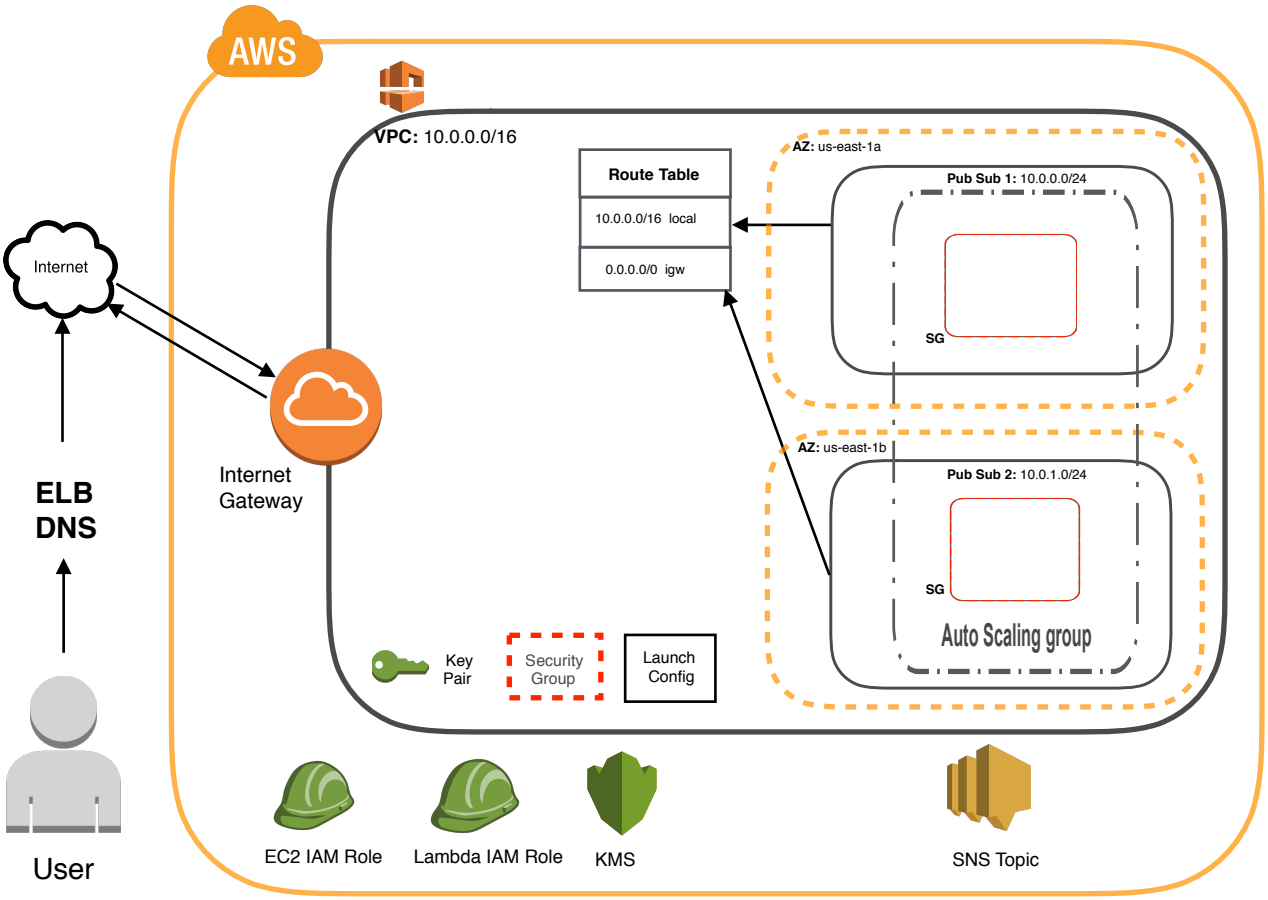


8. Once the above have been added, click the grey **Add Statement** button

9. Click the blue **Next Step** button

10. Click the blue **Apply Policy** button. This will add your new custom policy to the IAM Role

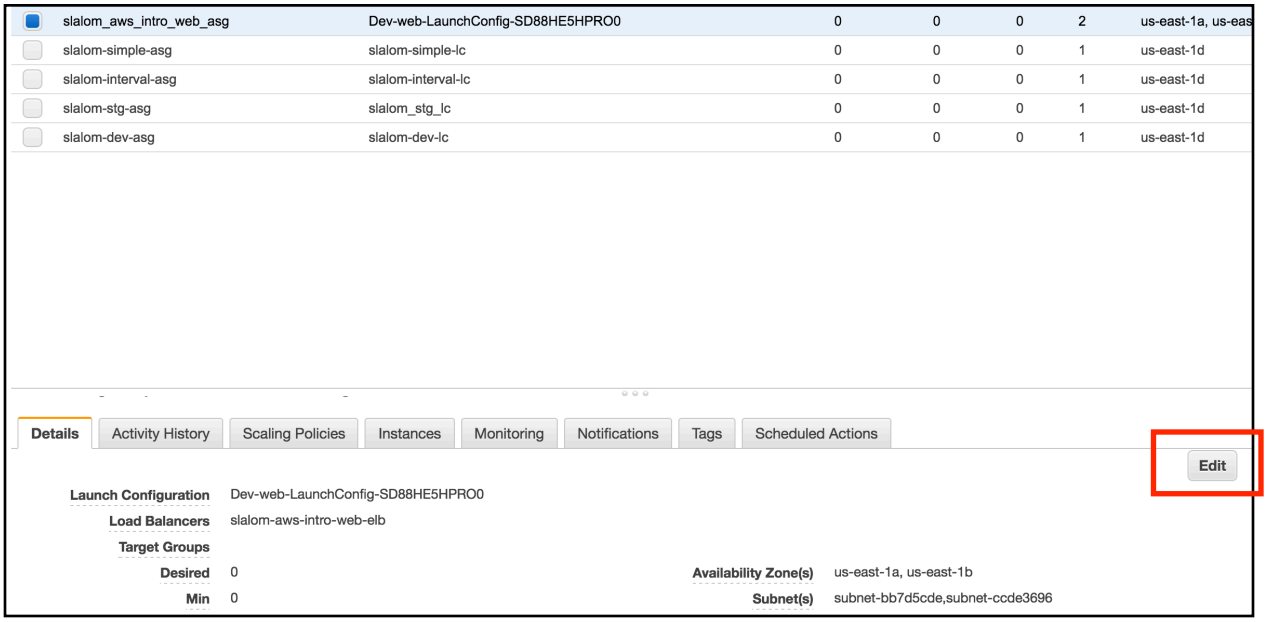
System Architecture: Step 7



Webhook URL: https://hooks.slack.com/….

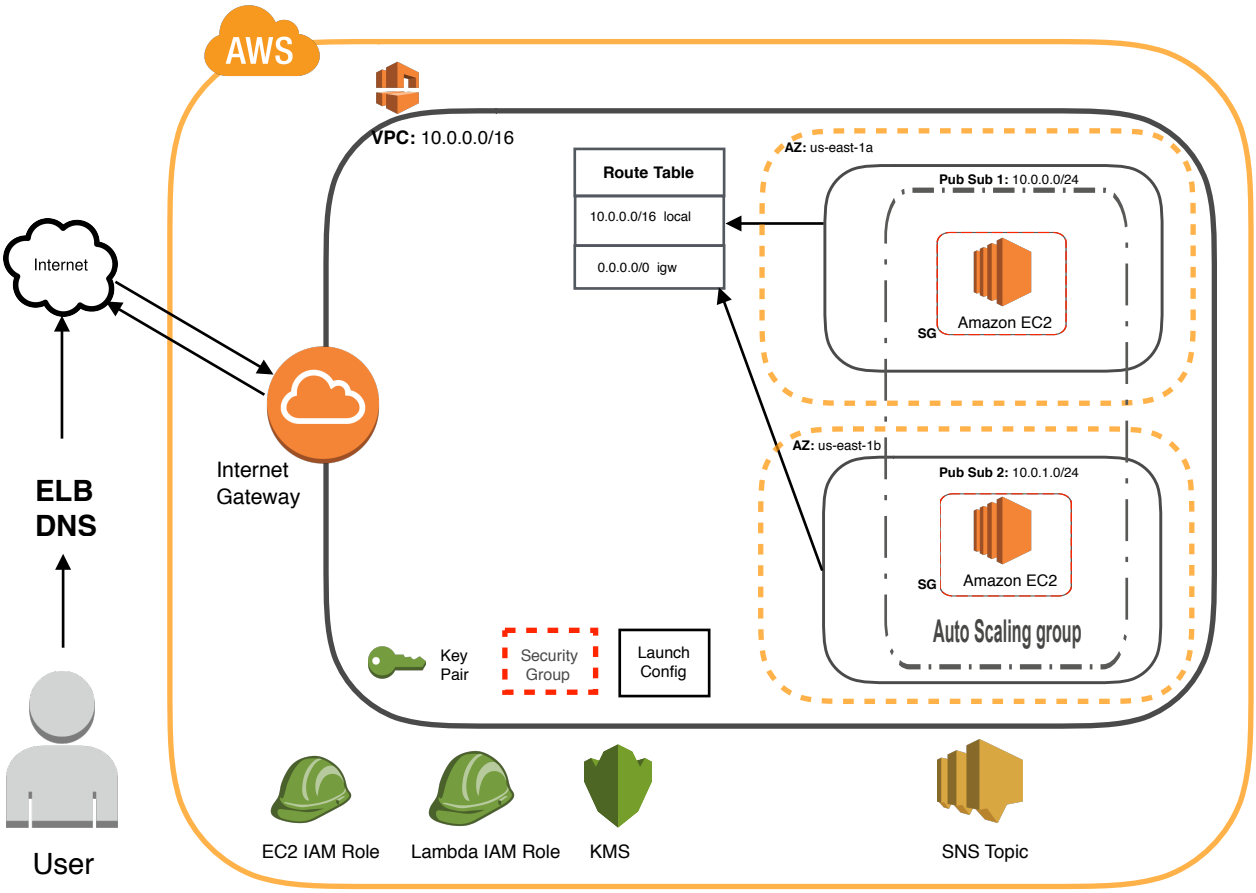
Step 8: Update your Auto Scaling Group Instance Count

1. Navigate to **Services —> EC2**
2. In the left menu, select **Auto Scaling Groups**
3. Select your **slalom_aws_intro_web_asg** group
4. In the box at the bottom, select the grey **Edit** button



5. Change the value of **Desired** and **Min** to 2. This will create two EC2 instances within your ASG. We need active EC2 instances in our ASG to allow us to create our CloudWatch alarm in Step 8.
6. Click the blue **Save** button

System Architecture: Step 8

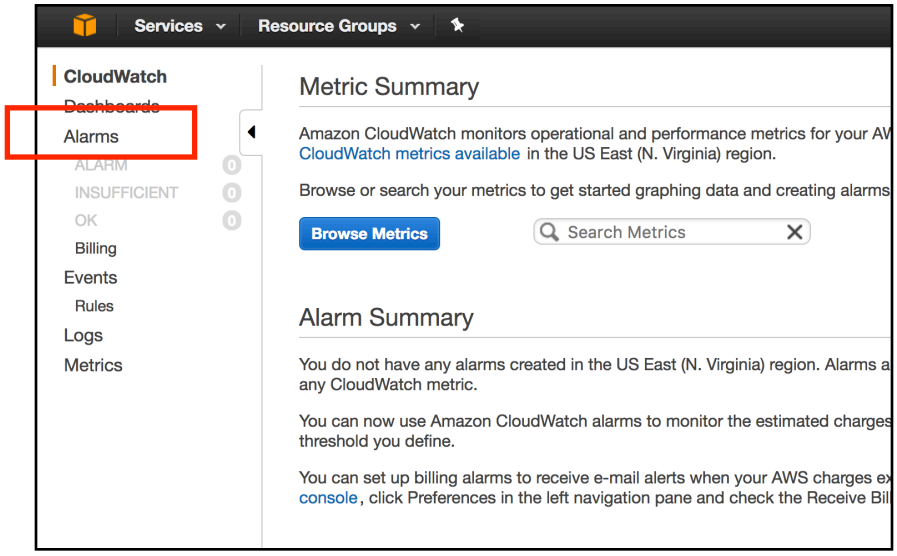


slack

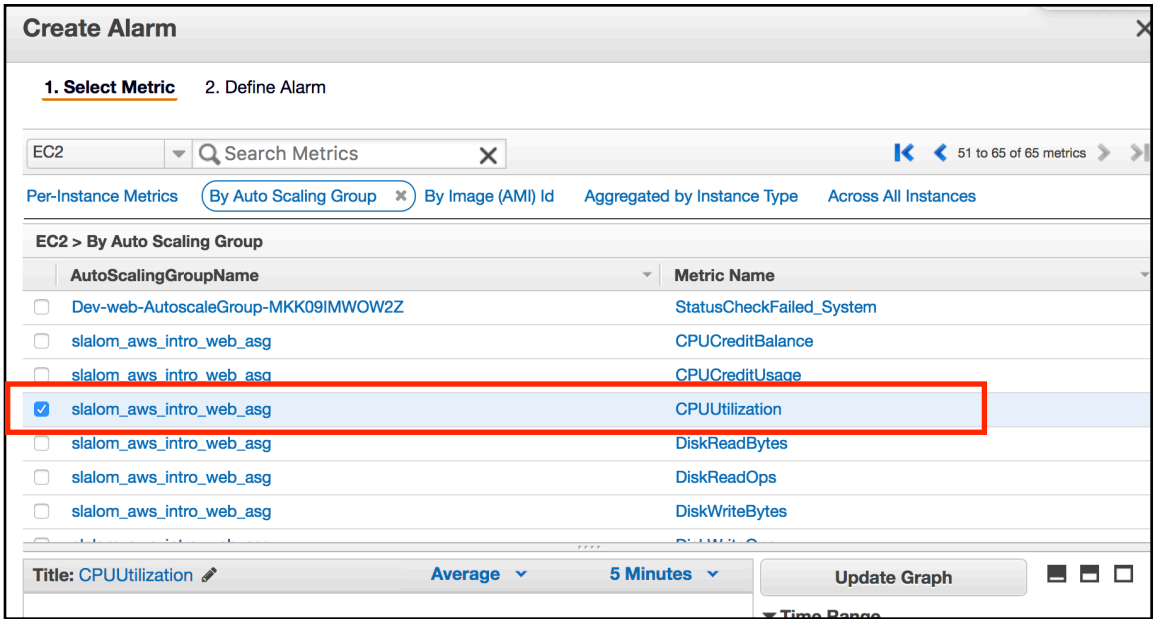
Webhook URL: https://hooks.slack.com/….

Step 9: Create an AWS CloudWatch Alarm

1. Navigate to **Services —> CloudWatch**
2. In the left menu, select **Alarms**



3. Select the blue **Create Alarm** button
4. In the following screen, select the **By Auto Scaling Group** option
5. In the box the opens, select the option with **slalom_aws_intro_web_asg** in the AutoScalingGroupName column and **CPUUtilization** in the Metric Name column



6. Click the blue **Next** button

7. On the following screen for Alarm Threshold, ensure the following information is configured to match the image below:

   Name: high_cpu_alarm

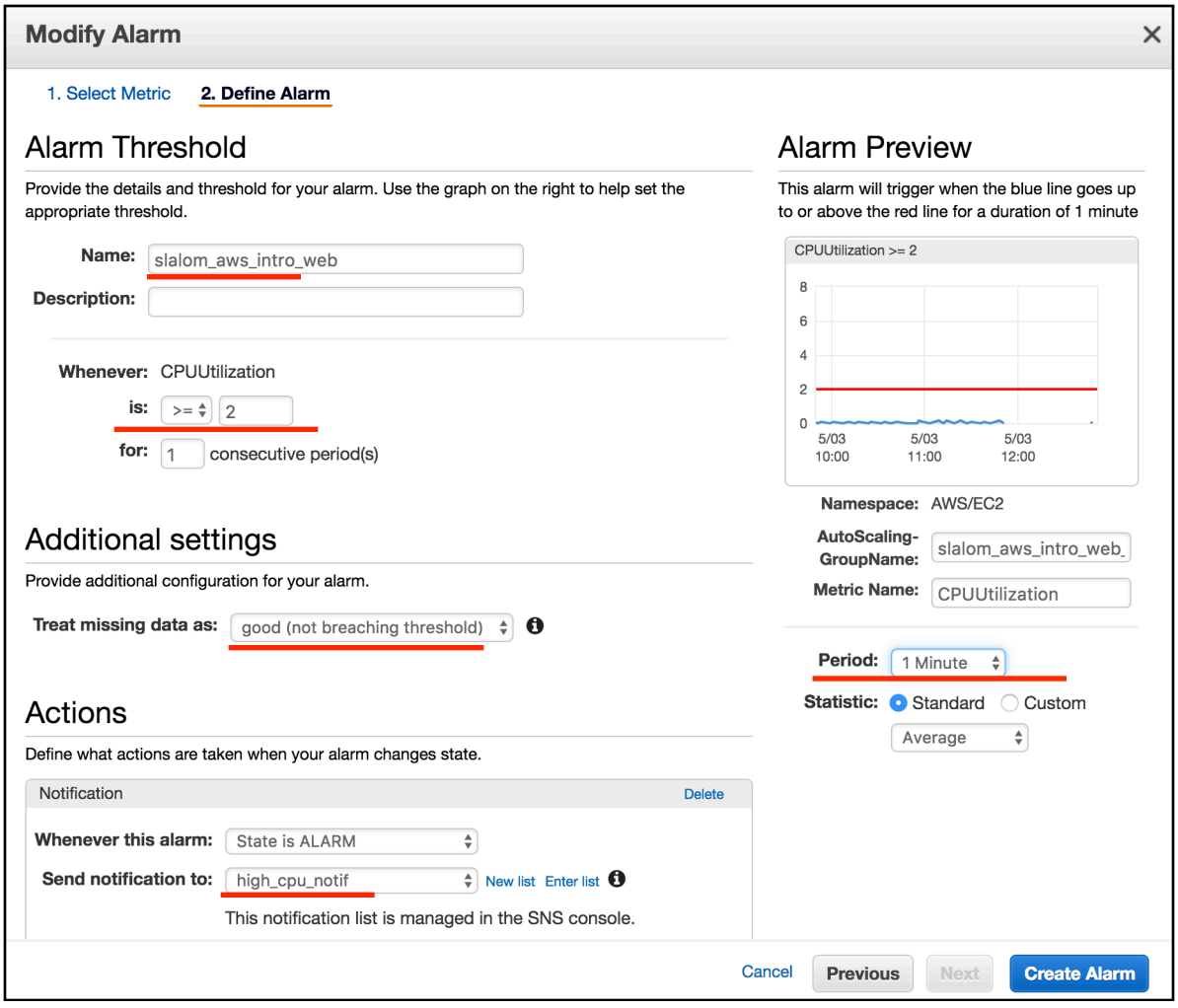
   Whenever: CPU Utilization is >= 2 for 1 consecutive period

   Treat missing data as: good (not breaching threshold)

   Send notification to: high_cpu_notif

   Period: 1 minute

NOTE: This alarm is configured with an unrealistically low threshold of 2%, but it will allow us to demonstrate this feature



8. Click the blue **Create Alarm** button

9. Navigate back to **Services —> EC2**

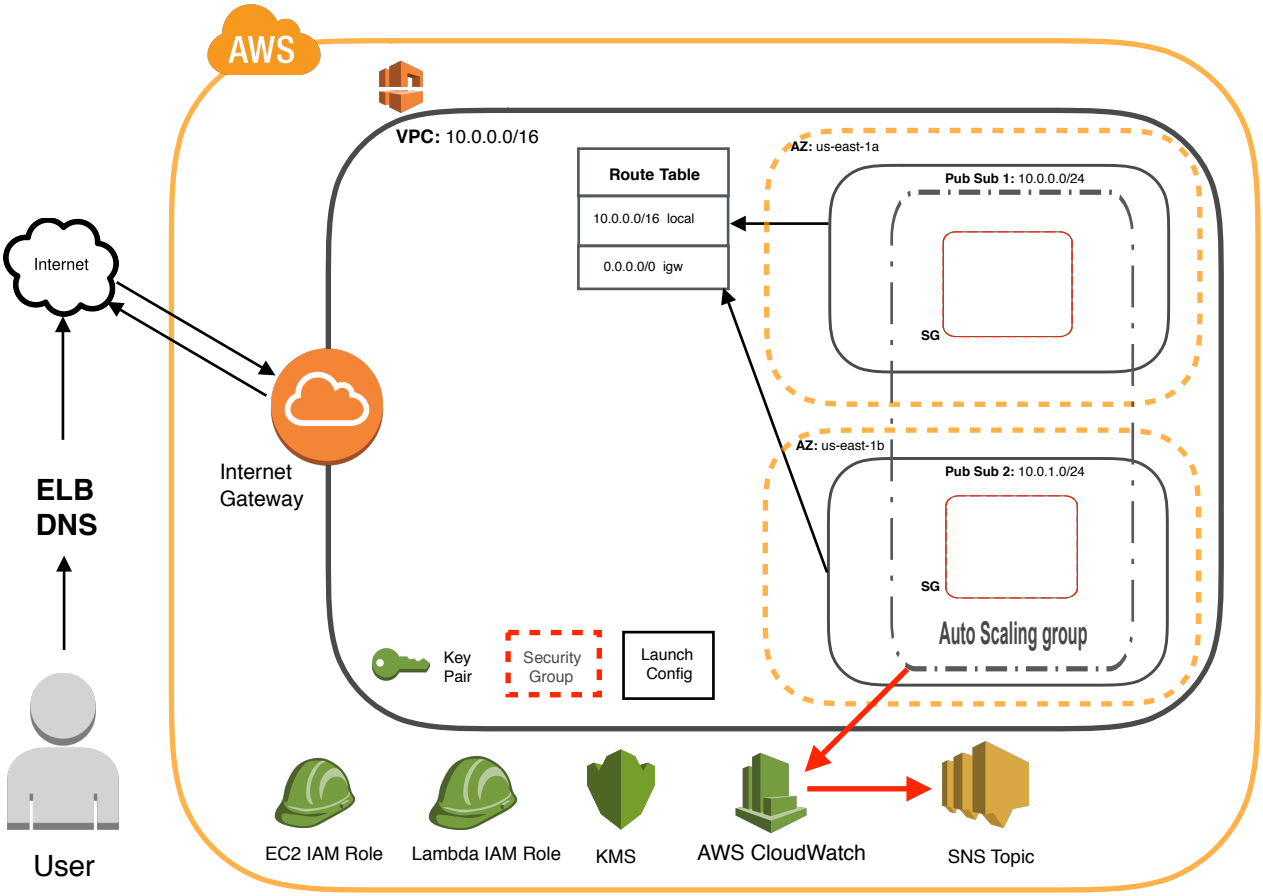
10. Click on **Auto Scaling Groups** in the left menu

11. Select your **slalom_aws_intro_web_asg** group in the list and click the grey Edit button at the bottom

12. Change the **Desired** and **Min** values back to 0

13. Click the blue **Save** button

System Architecture: Step 9



slack

Webhook URL: https://hooks.slack.com/….

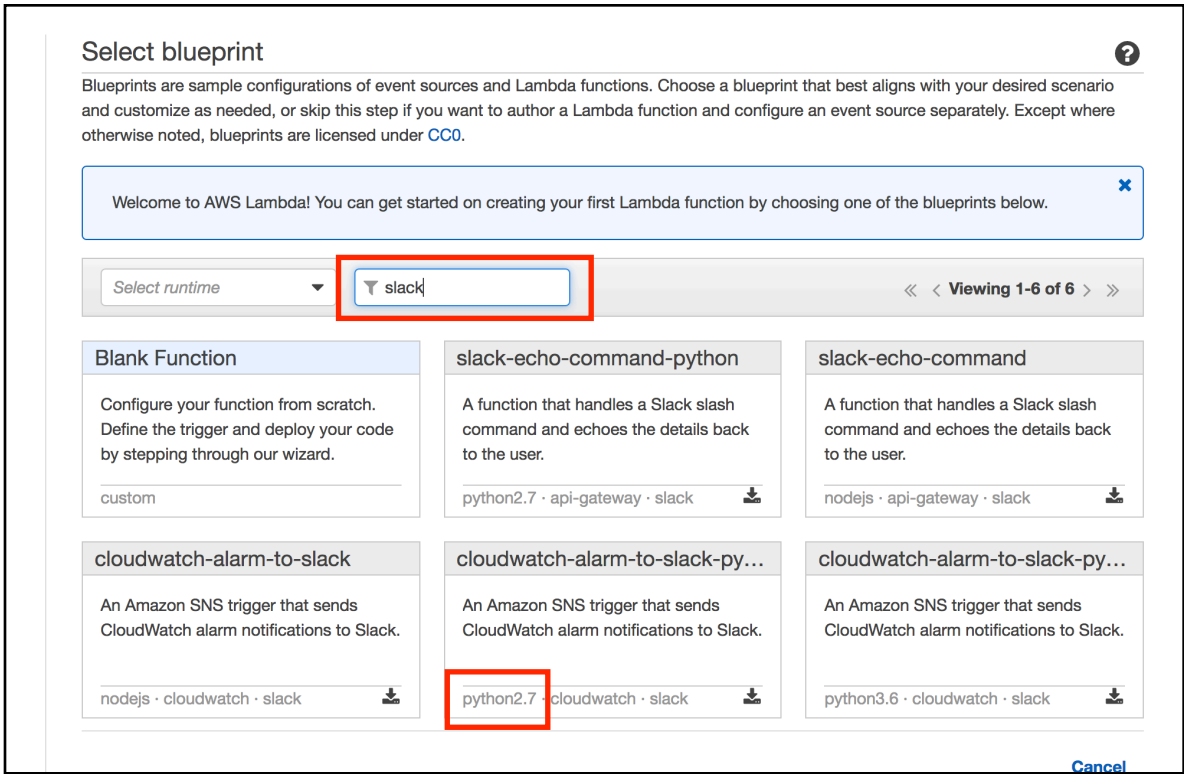
Step 10: Create an AWS Lambda function to post to Slack

1. Navigate to **Services —> Lambda**

2. Click the blue **Get Started Now** button

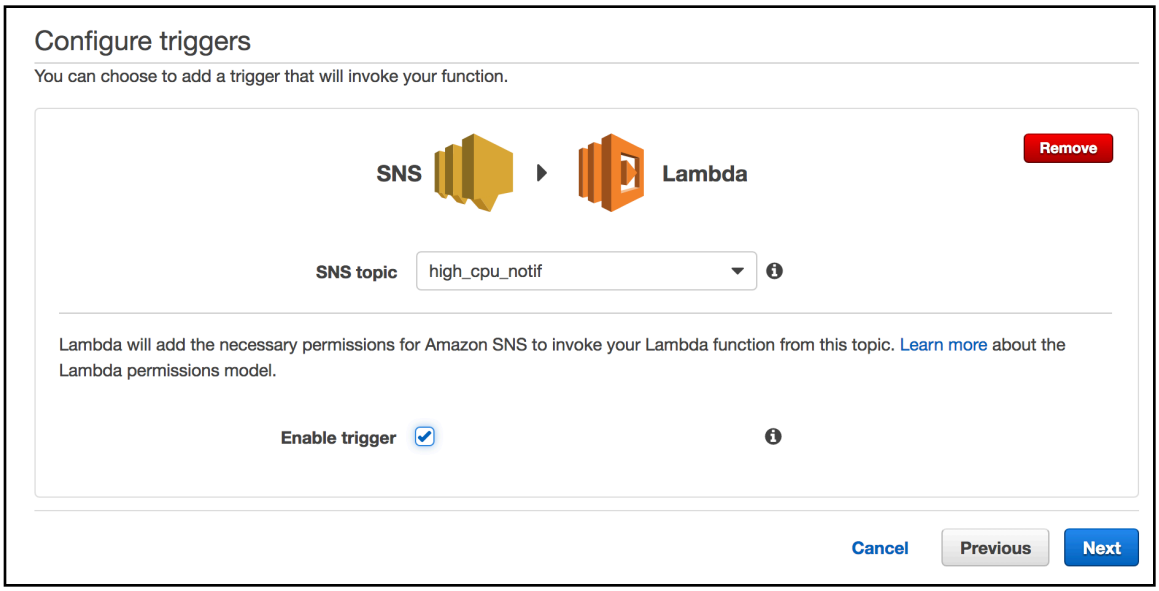
3. In the filter box, type in **slack**

3. Select the **cloudwatch-alarm-to-slack-py**… for **python 2.7** option



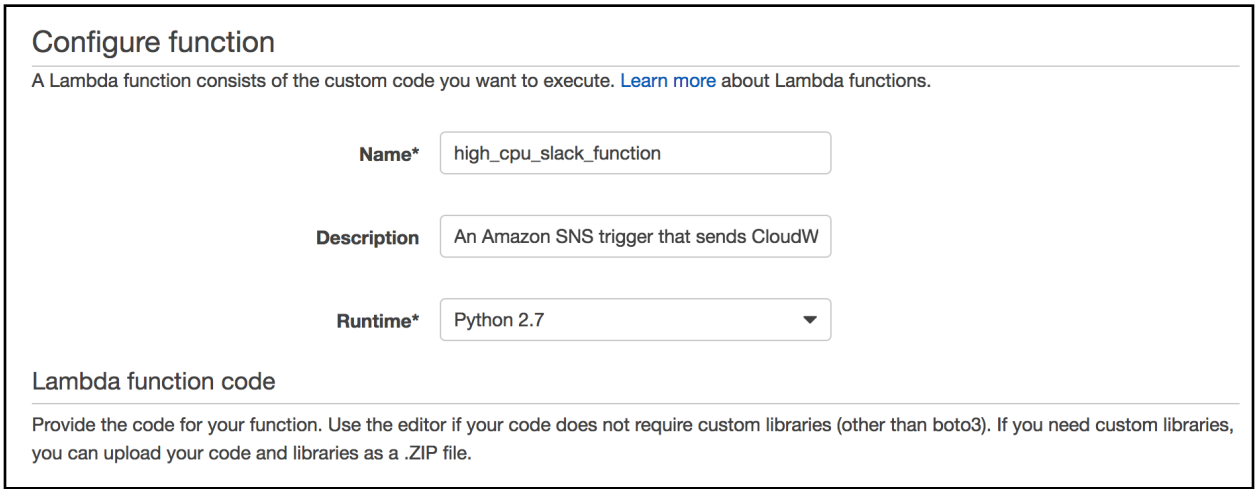
4. In the following page, ensure that the following is set:

   SNS topic: high_cpu_notif
   Enable Trigger: selected



5. Click the blue **Next** button

6. At the top of the Configure function page, set the **Name** to **high_cpu_slack_function**

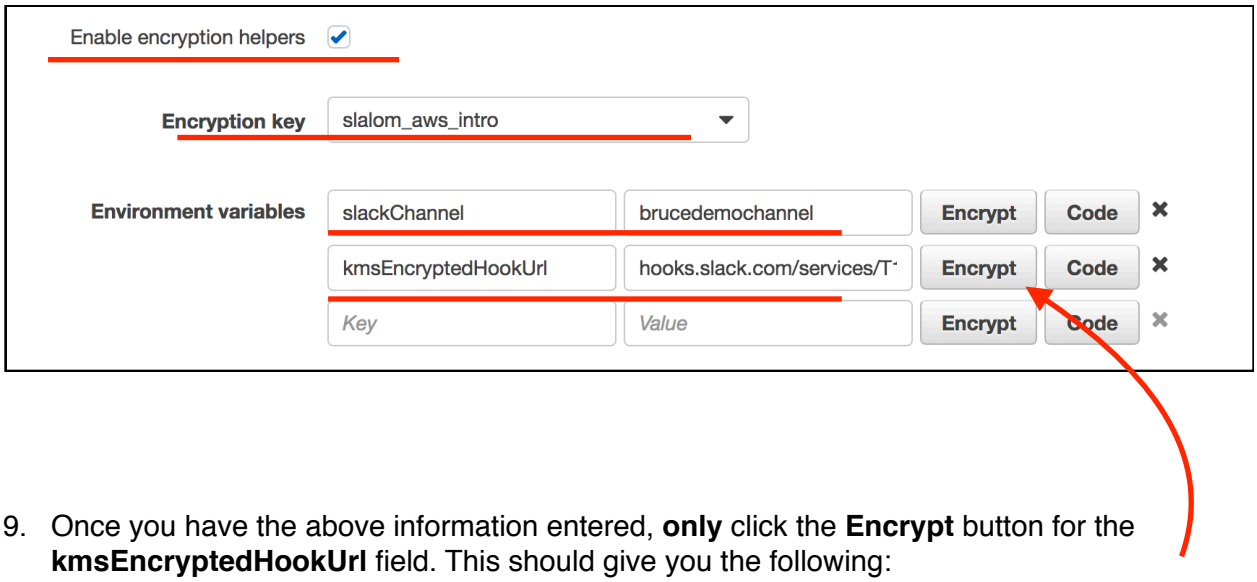


7. Scroll down the page to below the function code. You should see an option for **Enable encryption helpers**. Click the box to **select this option**

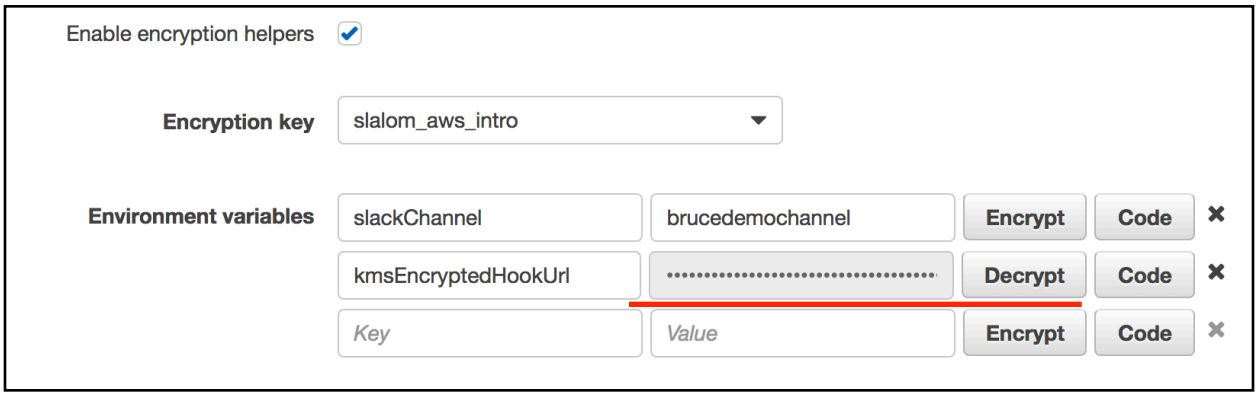
8. With the Enable encryption headers checkbox checked, enter the following information below.

   NOTE: Remember that your slack WebHook URL should be within the text file you created earlier (if you forgot to put your WebHook URL in a text file, ask Bruce and he will show you how to find it again):

   Encryption key: slalom_aws_intro
   slackChannel: < Your private slack channel name > (don't include the hashtag)
   kmsEncryptedHookURL: < The webhook URL you created in Step 2. **Do not include the https:// at the start** >



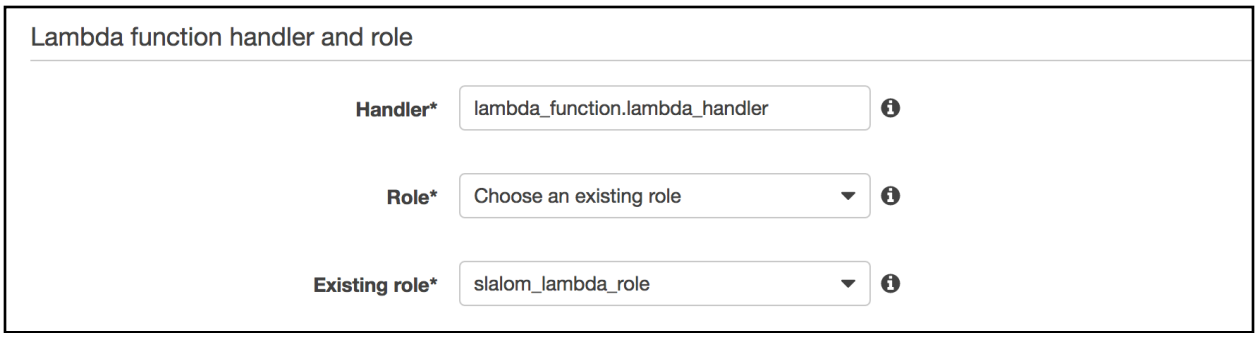
9. Once you have the above information entered, **only** click the **Encrypt** button for the **kmsEncryptedHookUrl** field. This should give you the following:



10. Scroll down further to the **Lambda function handler and role section**

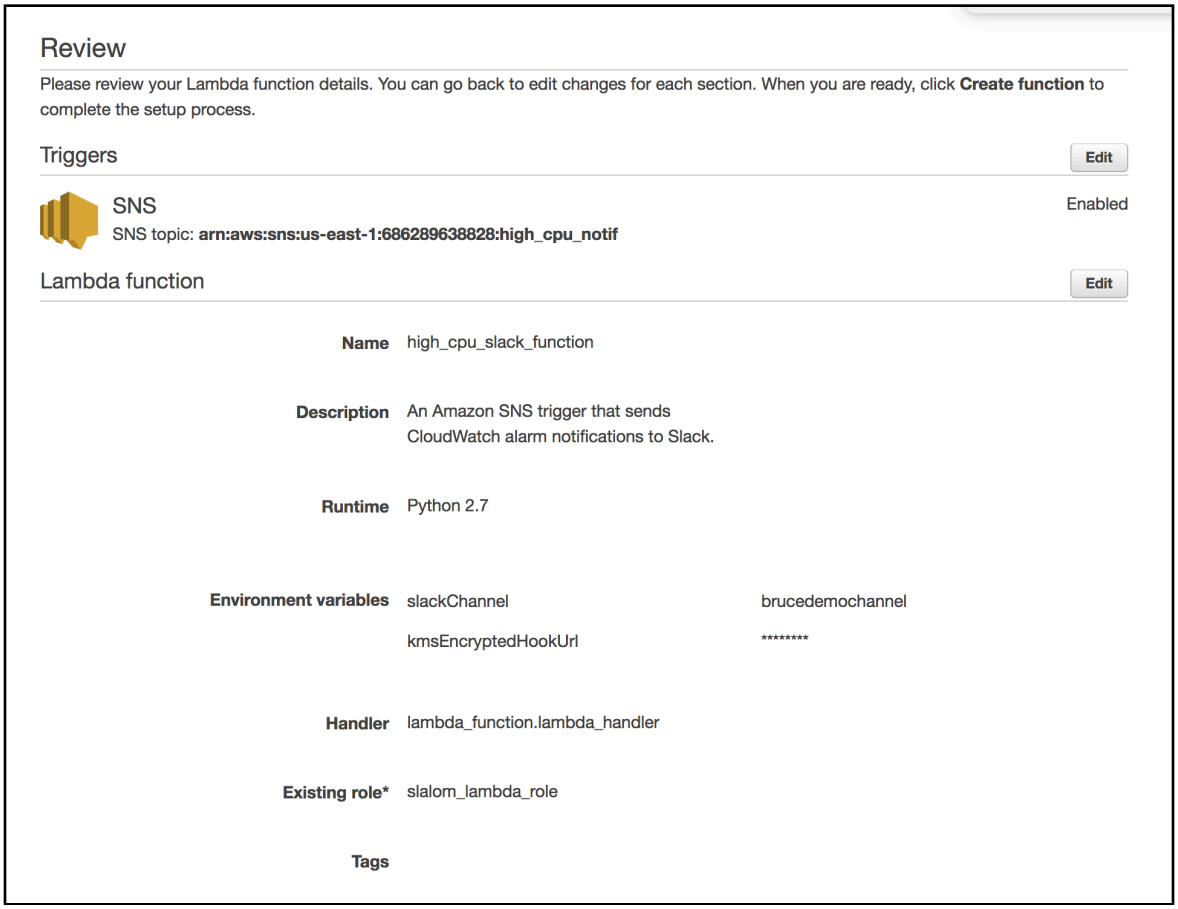
11. Within this section, ensure the following are set:

    Role: Choose an existing role
    Existing Role: slalom_lambda_role



12. Click the blue **Next** button at the bottom right of the page

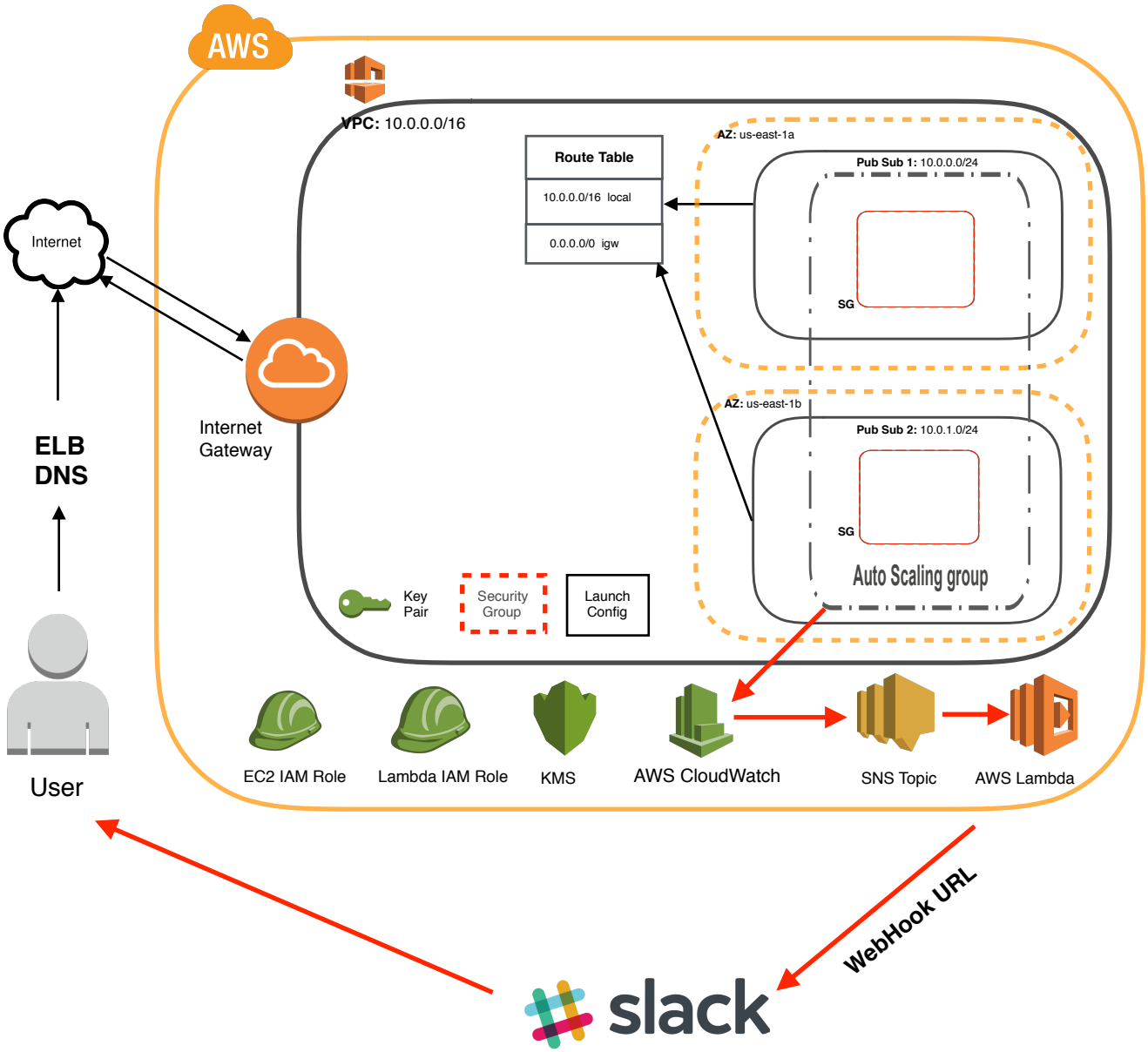
13. Review the summary page, checking that the options match the following (other than channel name, which will be set to your own channel):



14. Click the blue **Create function** button

15. You should see a success page, letting you know that the function was created

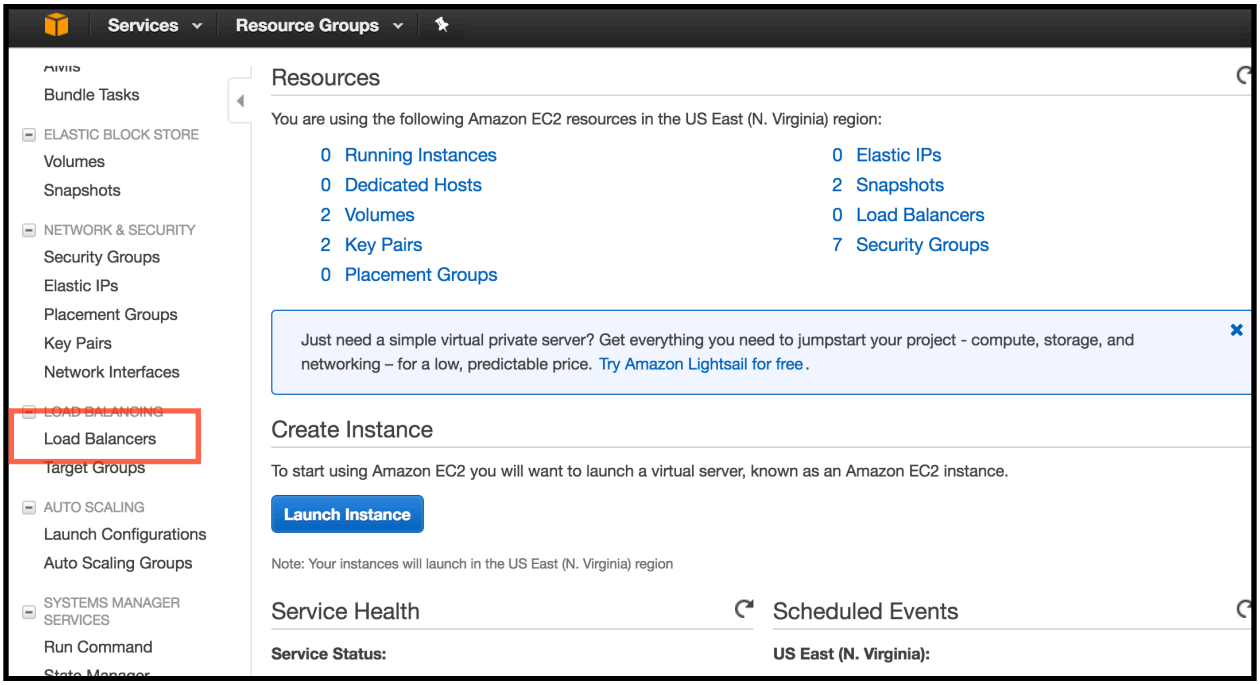
System Architecture: Step 10



Step 11: Create a Load Balancer

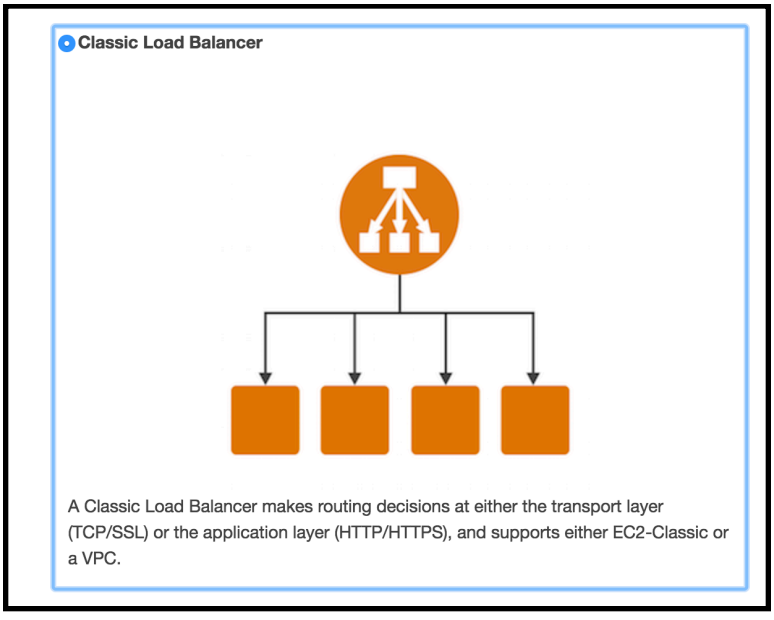
1. Navigate to **Services —> EC2**

2. In the left menu, select **Load Balancers**



3. Select the blue **Create Load Balancer** button

4. In the following screen, select the **Classic Load Balancer** option and click the blue **Continue** button

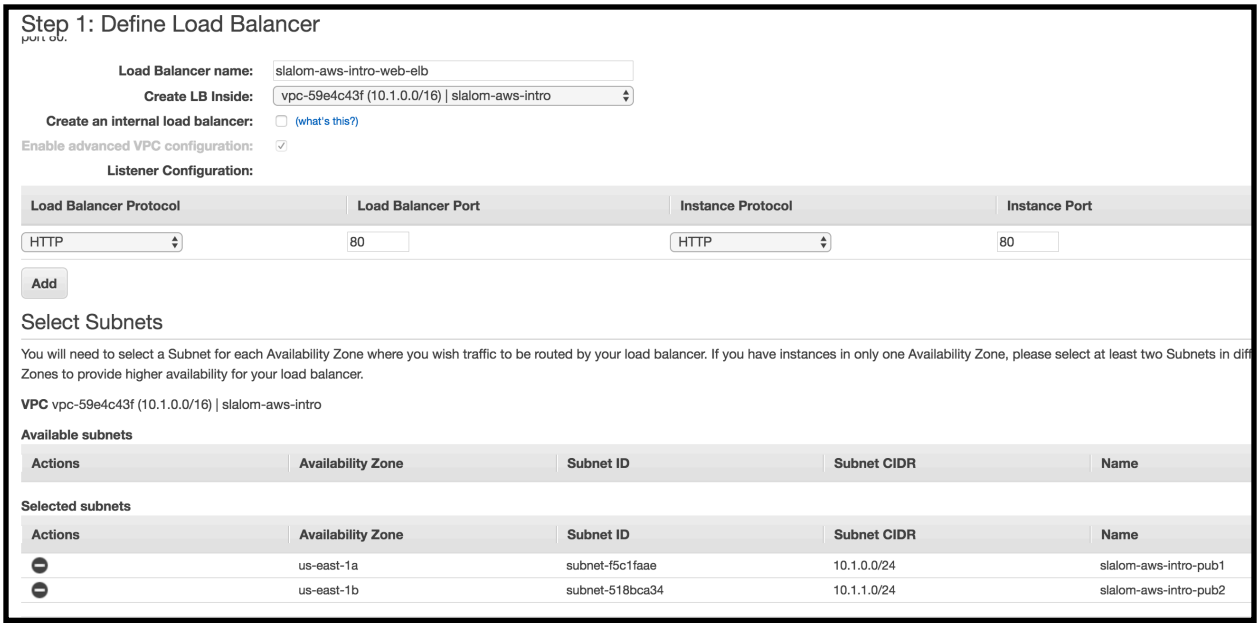


5. In the Define Load Balancer page, enter the following information:

Load Balancer Name: slalom-aws-intro-web-elb
Create LB Inside: slalom-aws-intro

6. In the Select Subnets section at the bottom, click the + symbols beside both **slalom-aws-intro-pub1** and **slalom-aws-intro-pub2**

   This should leave you with the following setup:



7. Click the **Next: Assign Security Groups** button at the bottom right

8. Use the **Select an existing security group** option and select **slalom_aws_intro_webserver** from the list of security groups

9. Click the **Next: Configure Security Settings** button

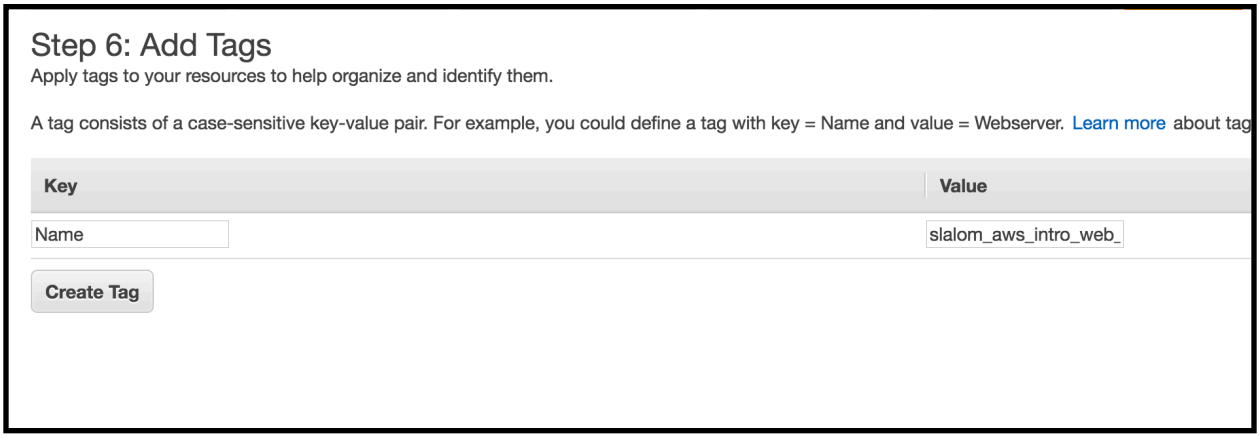
10. A warning message will appear, informing us that we can improve the load balancers security by using a secure listener. For the purposes of this workshop we will be using HTTP instead of HTTPS, so we can continue by clicking **Next: Configure Health Check**

11. We can leave the configured health check as it is, so click **Next: Add EC2 Instances**

12. We don't have any EC2 instances running currently, so continue to **Next: Add Tags**

13. Add a tag with the following values:

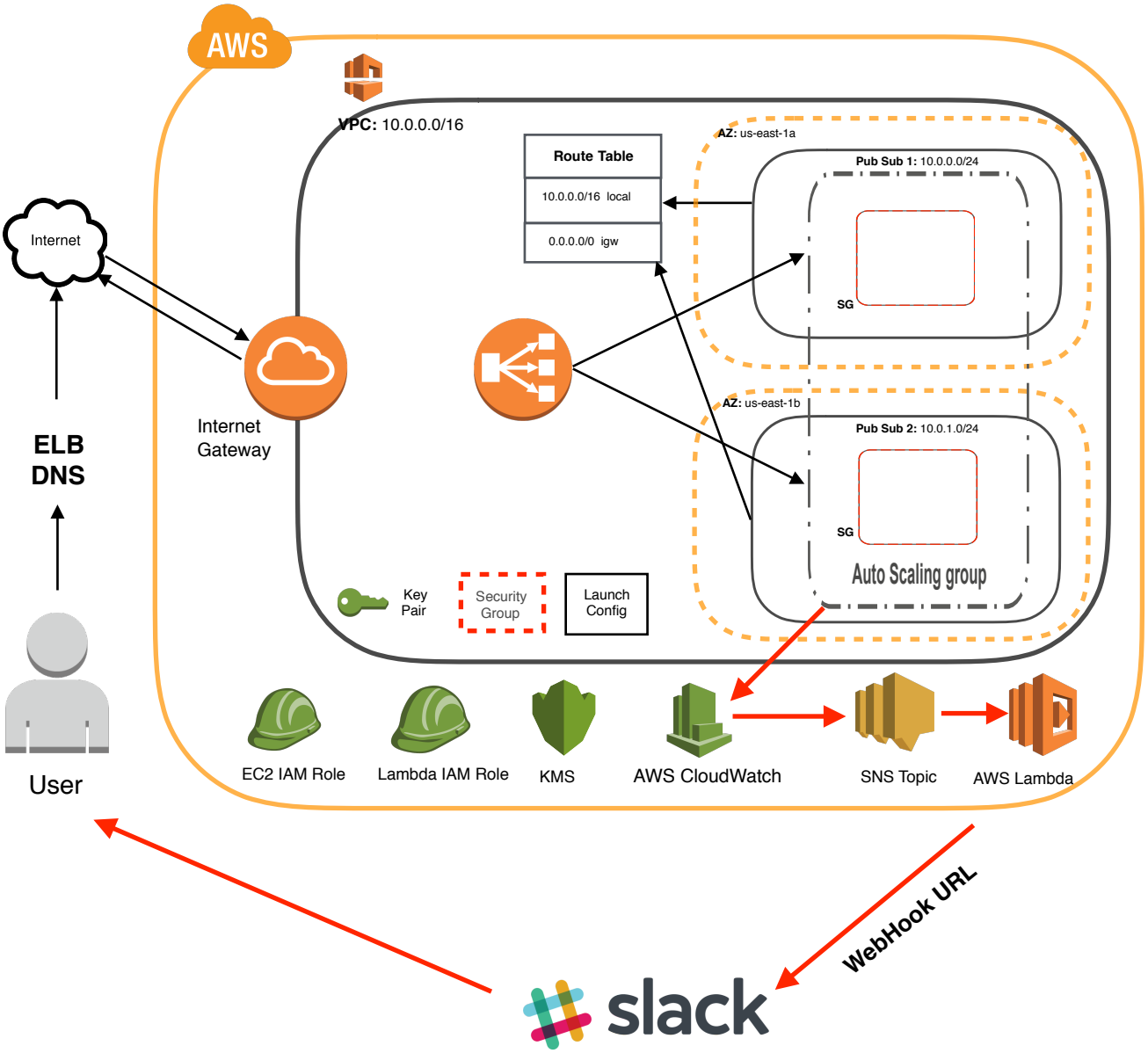
Key = Name
Value = slalom_aws_intro_web_elb



14. Click the blue **Review and Create** button at the bottom right

15. Click the blue **Create** button at the bottom right

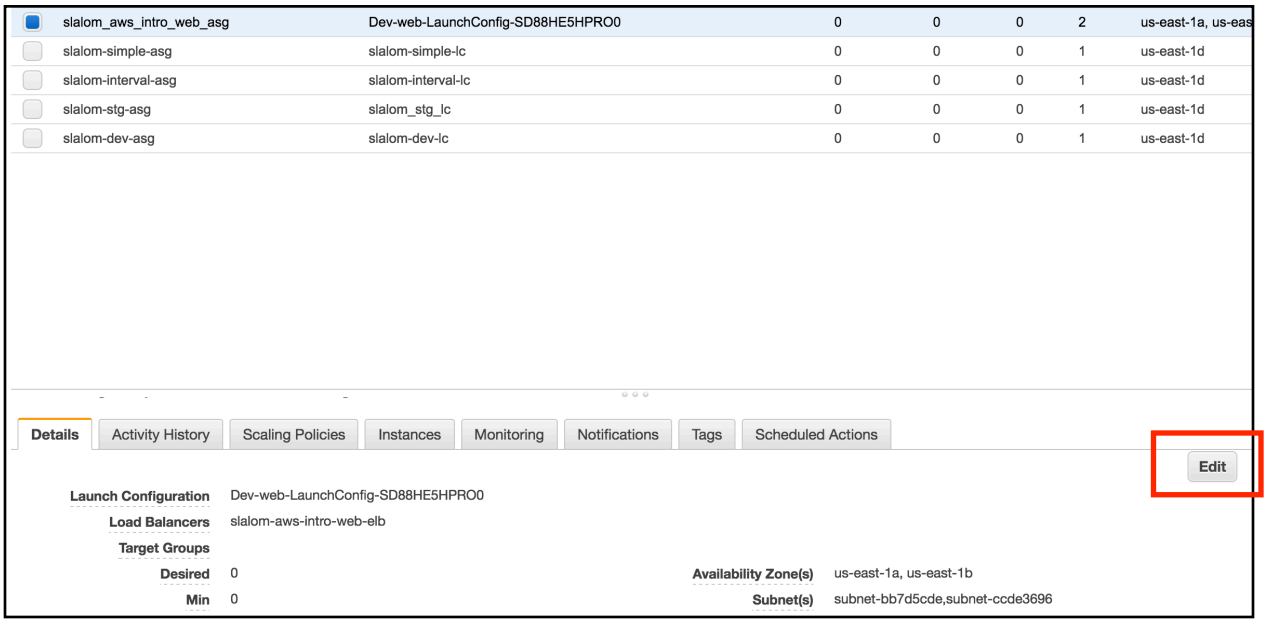
System Architecture: Step 11



Step 12: Update your Auto Scaling Group Instance Count

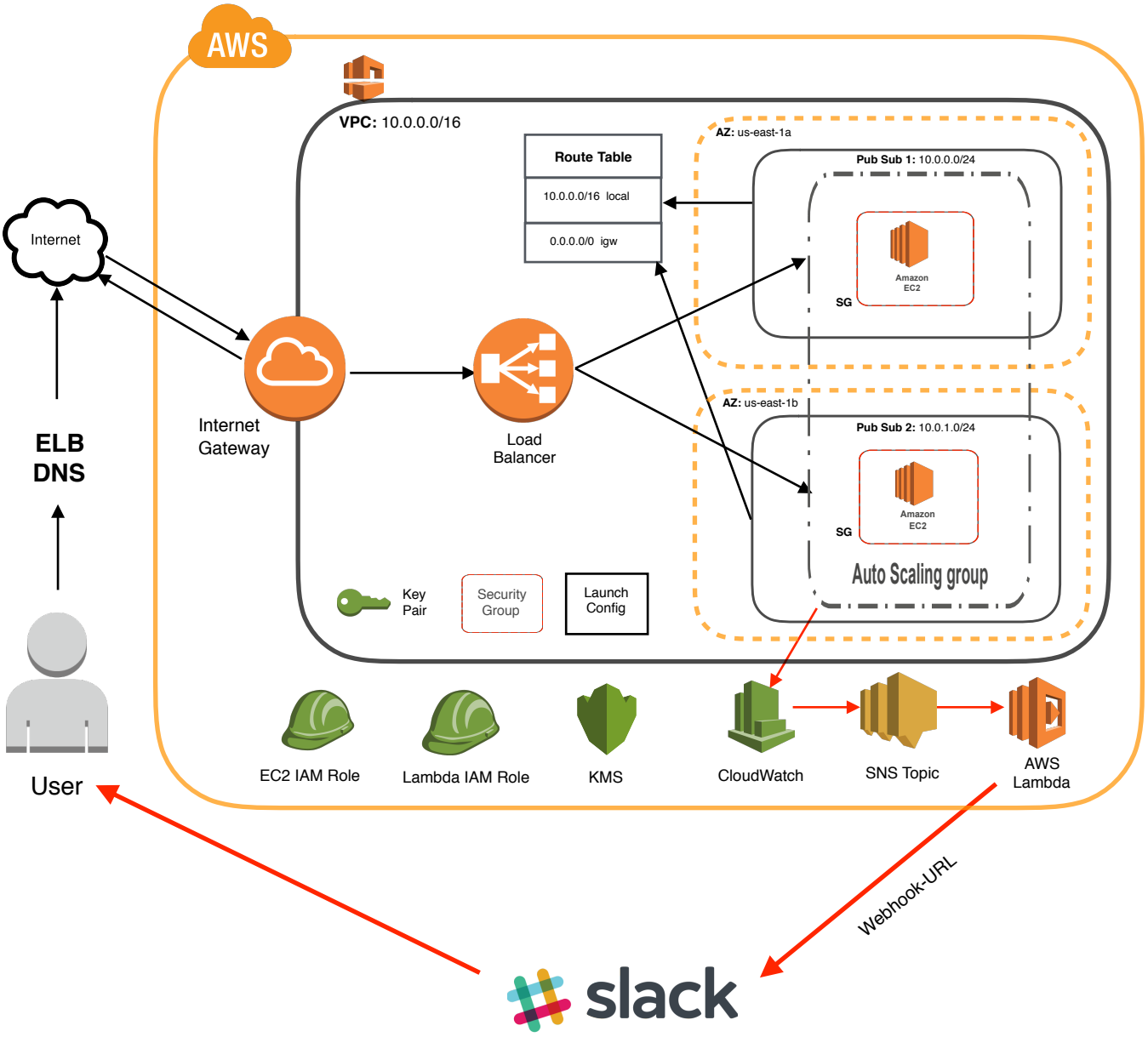
We now have all the pieces in our architecture in place to allow us to receive alerts. In this step, we will again increase the Min and Desired capacity of our Auto Scaling Group. Doing so will launch new EC2 instances, and their CPU's will spike during launch. This spike should be sufficient to trigger our CloudWatch alarm.

1. Navigate to **Services —> EC2**
2. In the left menu, select **Auto Scaling Groups**
3. Select your **slalom_aws_intro_web_asg** group
4. In the box at the bottom, select the grey **Edit** button



5. Click in the load balancers text box and select the **slalom-aws-intro-web-elb** that was just created
6. Change the value of **Desired** and **Min** to 2. This will create 2 EC2 instances within your ASG
7. Click the blue **Save** button

System Architecture: Step 12

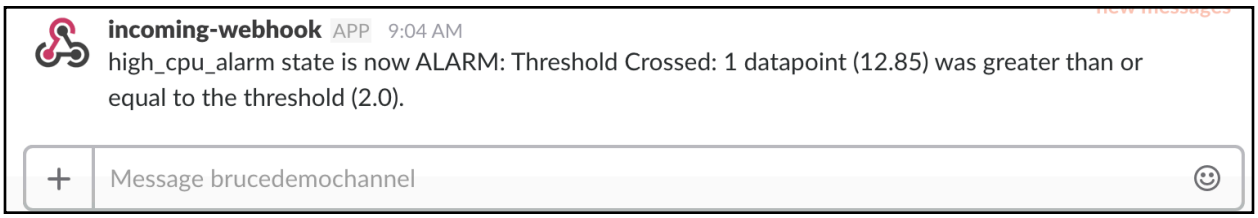


Step 13: Test it out!

As you have just increased the **Min** and **Desired** capacity from your autoscaling group, new instances EC2 instances will now be created within your Auto Scaling Group.

When new EC2 instances are created, their CPU Utilization spikes for a period time as they are booted. This spike in CPU should be enough to trigger our CloudWatch alarm which is configured to listen for CPU Utilization over 2% over a 1 minute period.

Keep an eye on your private Slack channel - You should receive an alert like the following:



System Architecture: Step 13