**Setup an Alexa Skill Demo utilizing Spring-Boot and running on AWS Elastic Beanstalk for testing**

**Michael Mc Mahon**

## 1.1 Introduction

As a training exercise I have set a goal to implement a custom skill for Alexa using an AWS HTTPS web service that will use Spring Boot as the framework. This approach can be helpful when you have an existing REST services already deployed and you want to leverage that infrastructure investment. An additional concern could be that privacy and security requirements will not allow off-premise or cloud-based infrastructure.

## 1.2 High Level Alexa Design

Alexa Infrastructure

Machine generated alternative text:
Interaction Model 
(Front-end) 
amazon 
alexa 
Hosted Service 
(Back-end) 

Alexa Skill structure

Machine generated alternative text:
Alexa, ask History Buff what happened on May 5th 
Wake word Starting 
Phrase 
Skill 
Invocation Name 
Utterance 

In this demo I will create a skill that will lookup trivia about either a specific year, or a random year and a user can interact with the skill by saying things such as:

*“Alexa, ask michaels demo to tell me trivia about a random year?”*

*“Alexa, ask michaels demo what happened in the year 1984?”*

This first demo Alexa Skill is based in part on the [Alexa Spring-Boot article](https://medium.freecodecamp.org/implementing-an-alexa-skill-with-spring-boot-also-why-would-you-do-such-a-thing-9992c0797646) by Rafael Fiol and the a free third-party [NumbersAPI](http://numbersapi.com/#42) by David and Mack to lookup the trivia information.

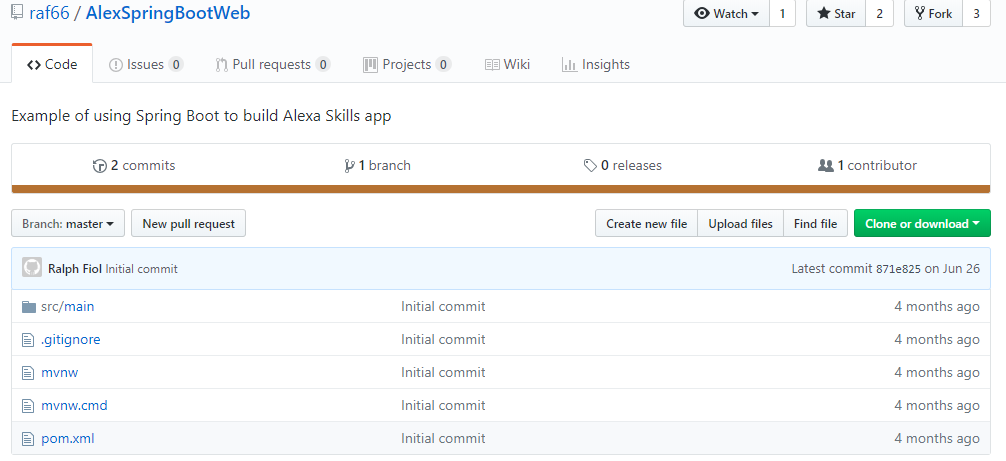
I have attempted to documents in as much detail as possible the process I followed to implement this Spring-Boot Alexa Skill.

## 1.3 Prerequisites

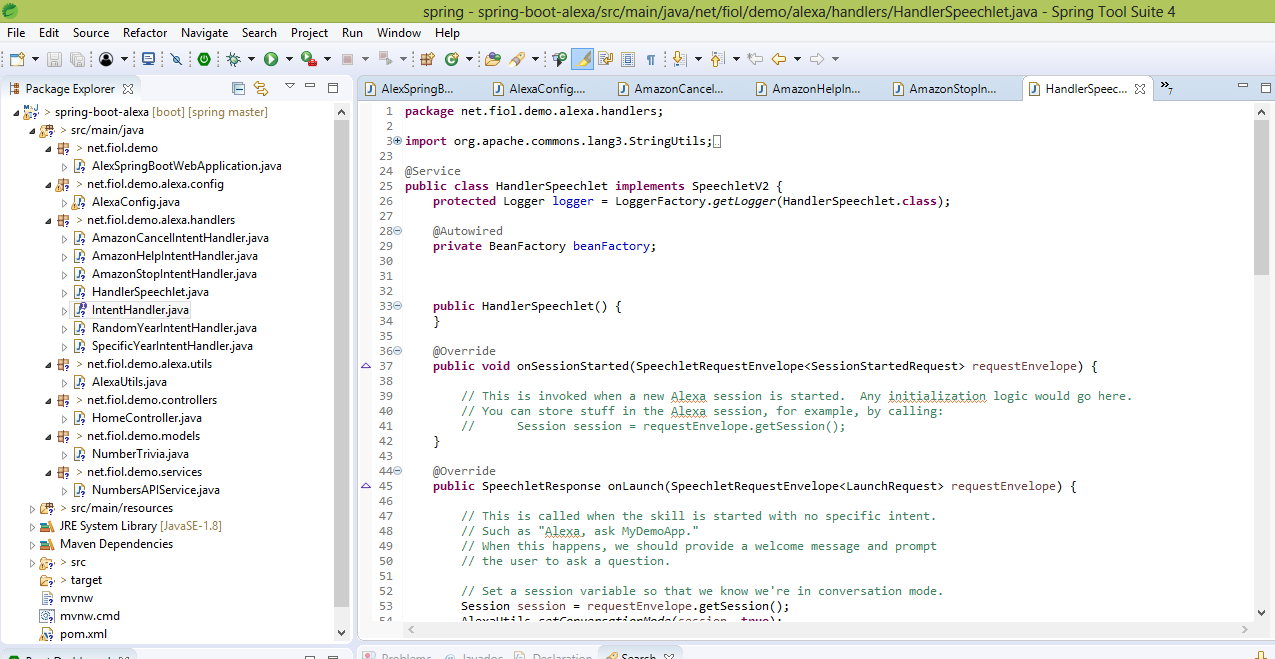
* A suitable IDE such as [Spring Tool Suite 4](http://spring.io/tools4)
* [Apache Maven](https://maven.apache.org/) build automation tool
* A developer account in the [Alexa Developer Console](https://developer.amazon.com/alexa)
* An [AWS account](https://console.aws.amazon.com/) to setup Elastic Beanstalk
* [OpenSSL](https://www.openssl.org/) or [OpenSSL for Windows](https://slproweb.com/products/Win32OpenSSL.html) to generate a self-signed certificate

## 1.4 Process

Download the source code from my repository or from the original Rafael Fiol GitHub repository <https://github.com/raf66/AlexSpringBootWeb>



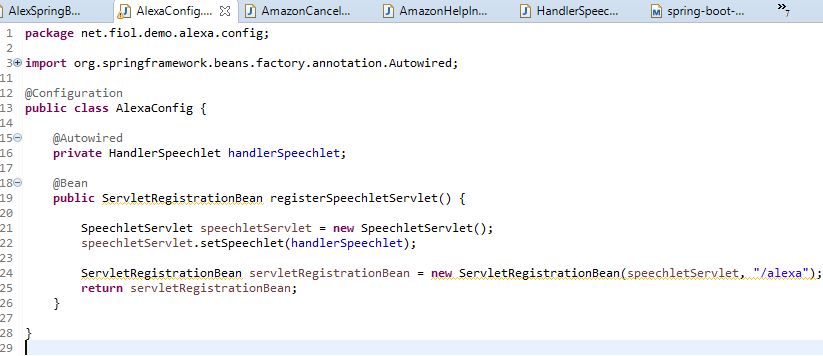
Open and review the code in Spring Tool Suite 4



Note that the Alexa Skills Kit SDK is referenced in “pom.xml” file.



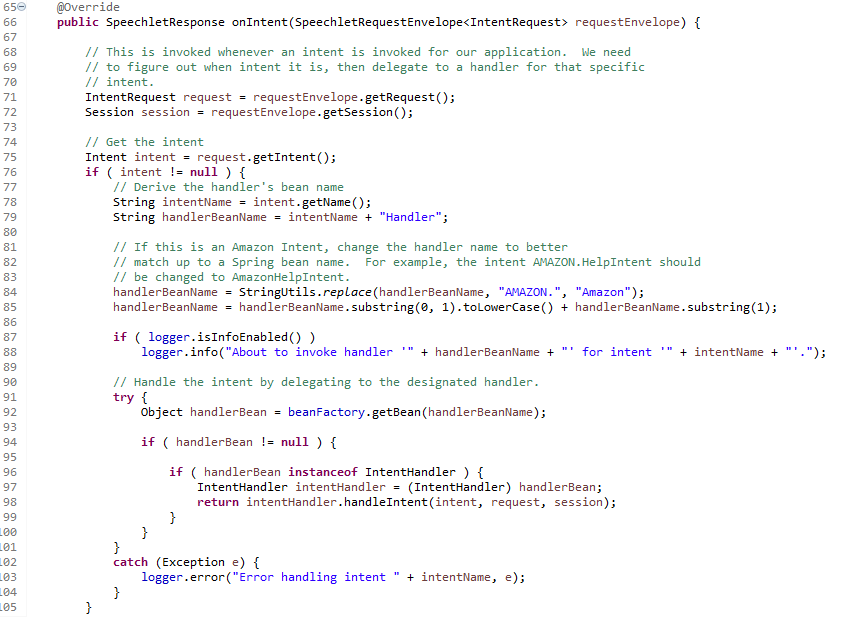
The SDK includes a servlet called SpeechletServlet which handles all of the complicated handshake communications with Alexa, such as verifying that the request was sent by Alexa, validating the signature, and checking the timestamp. The servlet is loaded through the AlexaConfig.java file.



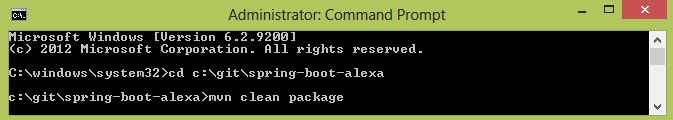
The servlet delegates the actual interaction business logic to a HandlerSpeechlet which will be invoked with every Alexa interaction and contains four methods:

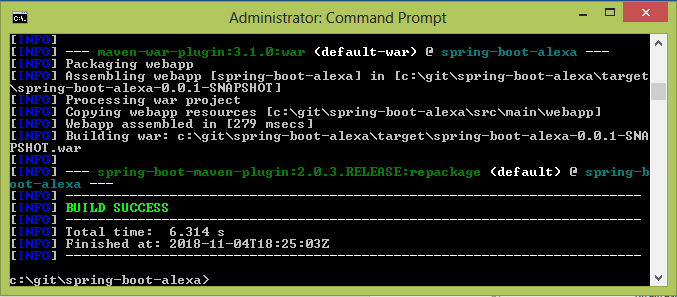
* onSessionStarted()
* onLaunch()
* onIntent()
* onSessionEnded()

The OnIntent() is where most of the work happens as this is what is invoked when the user says something meaningful to the Alexa Skill.

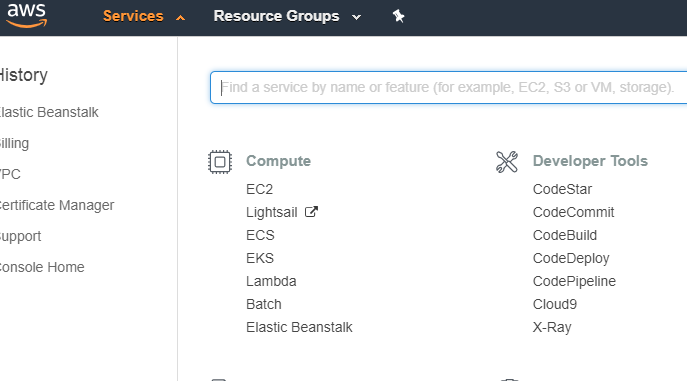


Open terminal window and change to code directory and build WAR package - **mvn clean package**

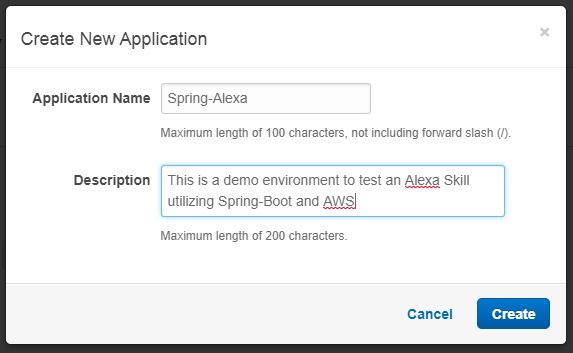




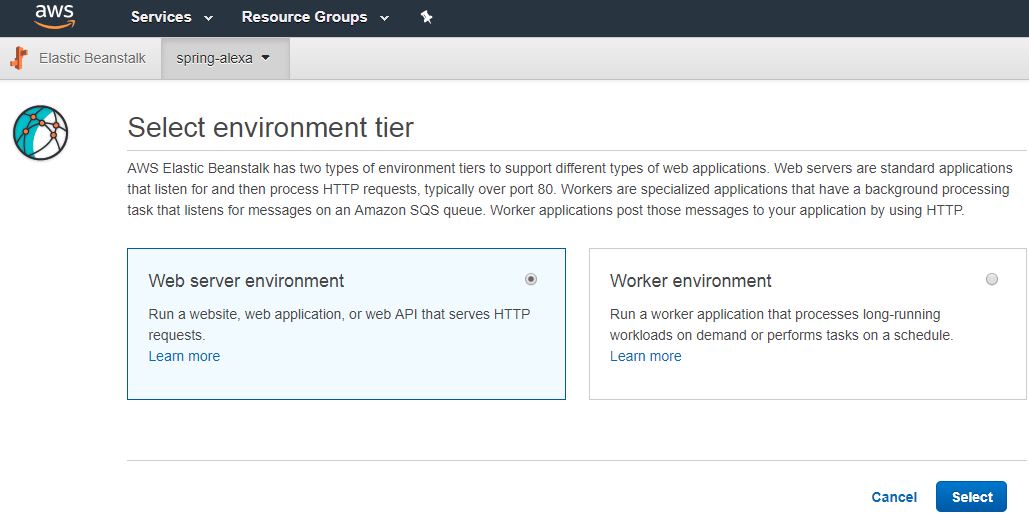
Now we create our AWS Elastic Beanstalk Environment so Log into AWS Console and select Elastic Beanstalk



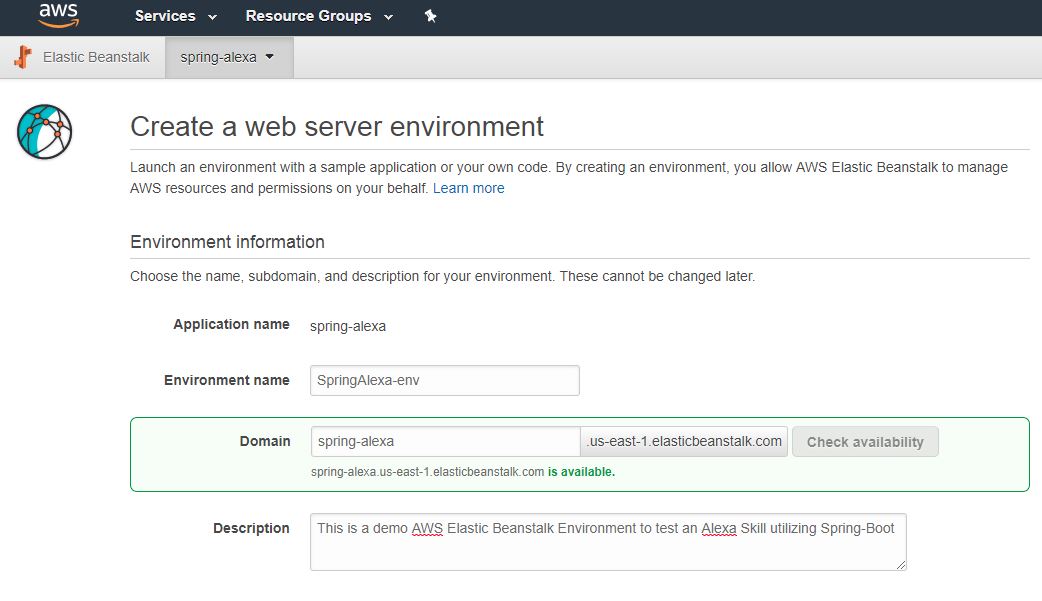
Create a new Elastic Beanstalk application



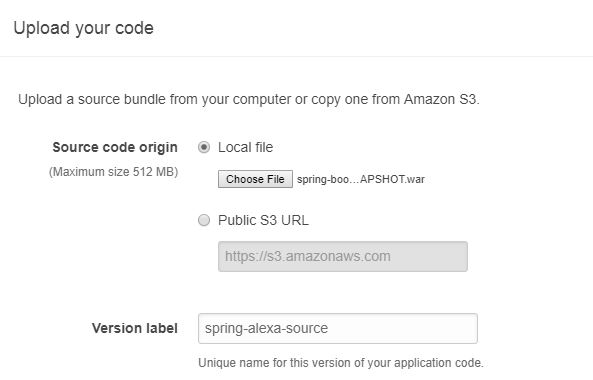
Select the Web Server Environment Tier



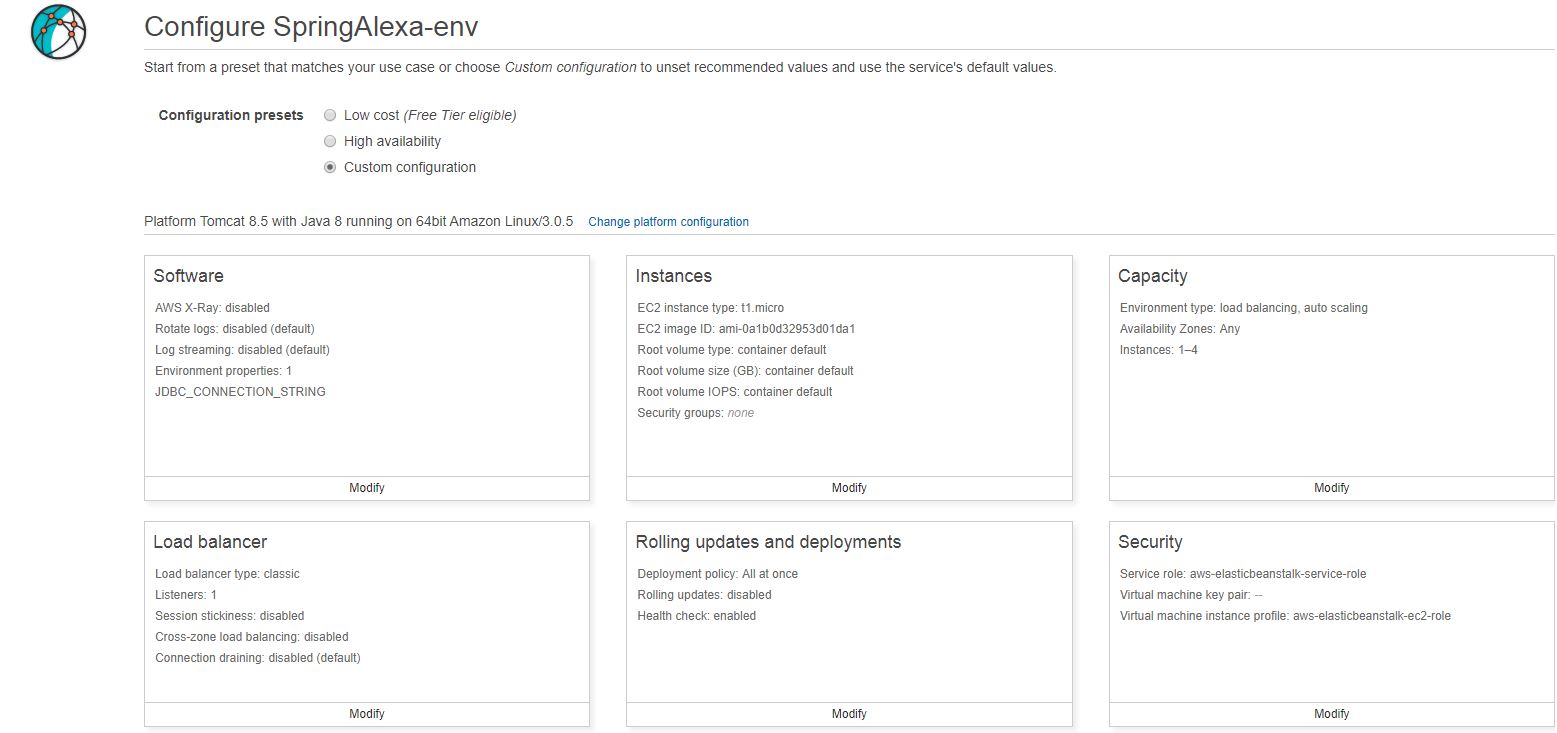
Create new web server environment and check Domain Name is available.



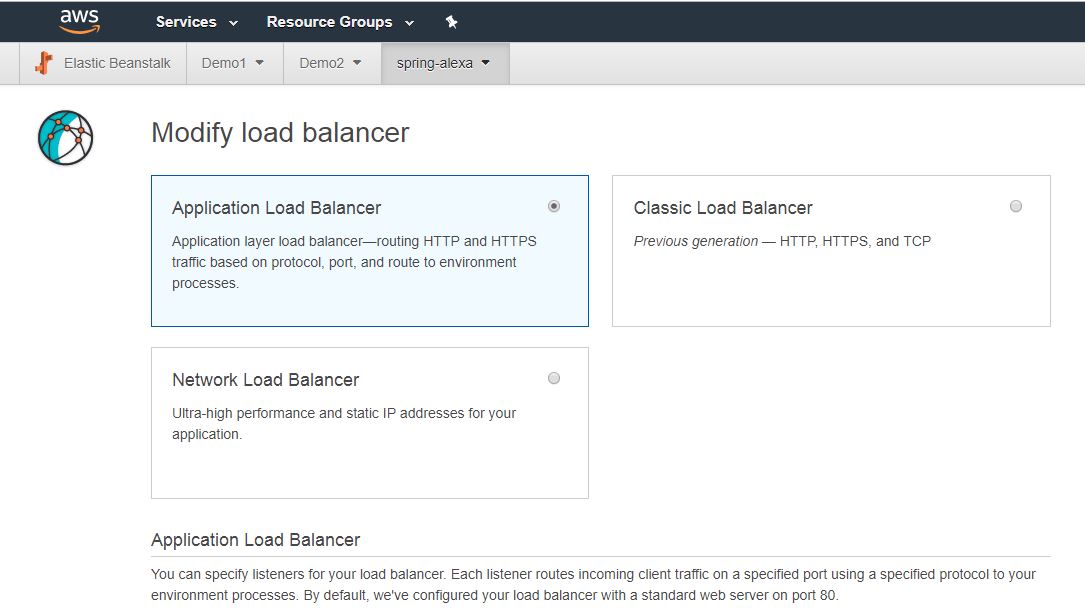
Upload the WAR file previously packaged and located in **<Your Location>\spring-boot-alexa\target\spring-boot-alexa-0.0.1-SNAPSHOT.war**



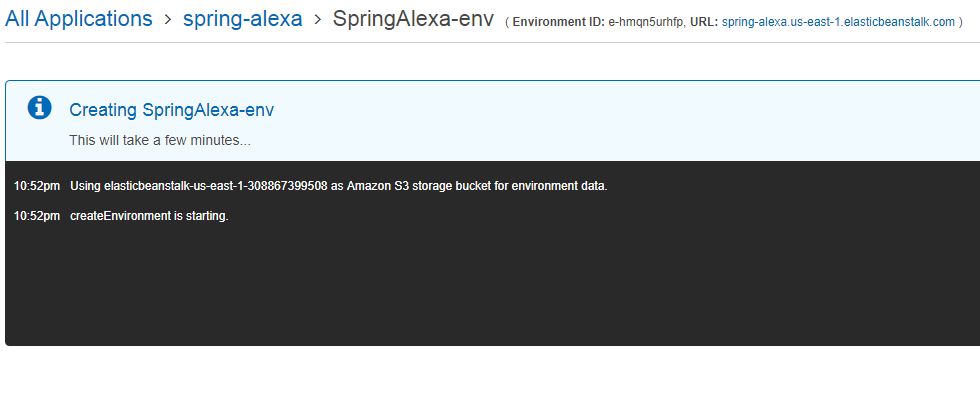
Configure the Environment – I used a custom setup as the load balancer is not created by the Low Cost Preset configuration and is needed to enable the HTTPS web service



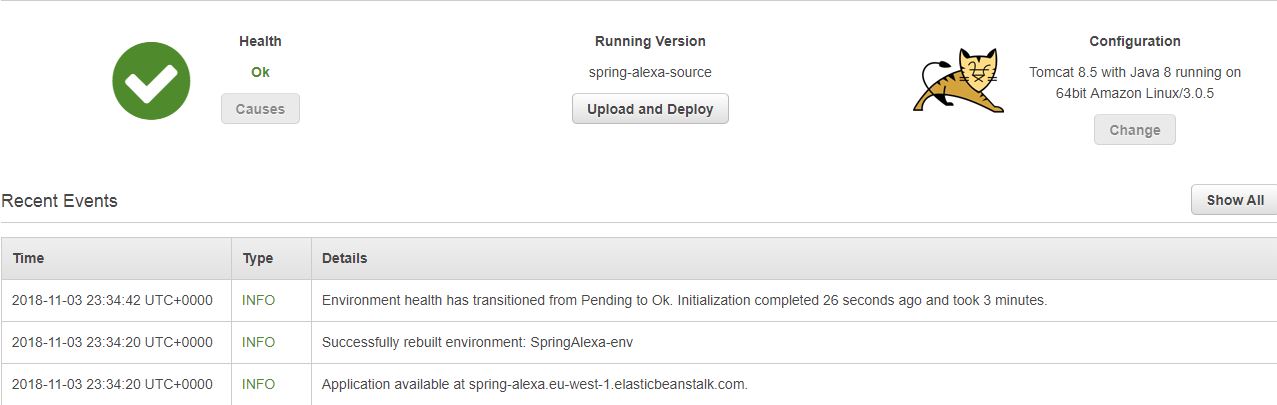
Open the load balancer and select the Application Load Balancer which I found worked best



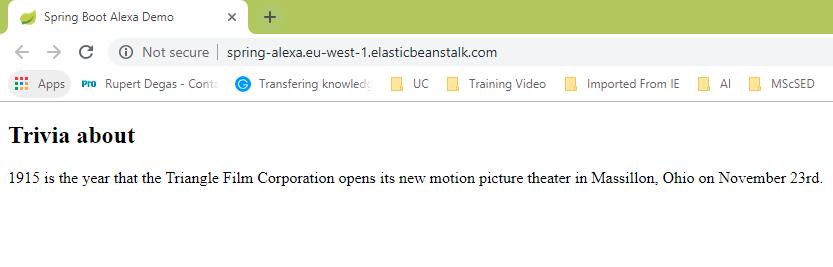
Create the Elastic Beanstalk environment



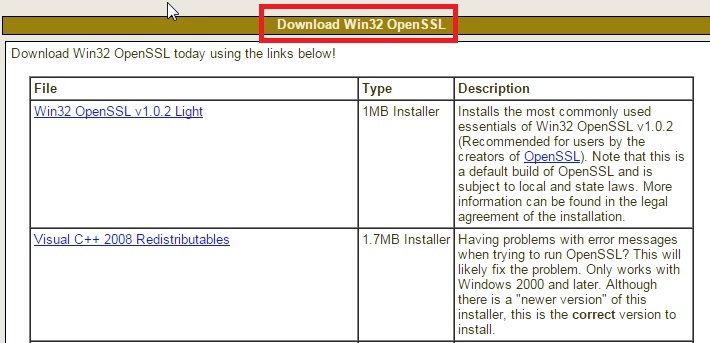
Environment launches if all went well



Test the application is now working to clicking on the assigned URL and you should receive a trivia answer.

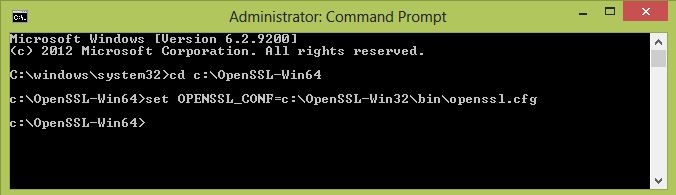


Now it is necessary to create and configure the self-signed certificate. Since I am using windows, I needed to use a Windows Binary distribution. Open the following link in your web browser: https://wiki.openssl.org/index.php/Binaries. On the table Third Party OpenSSL Related Binary Distributions, there are a few distributions. Select the "OpenSSL for Windows" and follow the link: https://slproweb.com/products/Win32OpenSSL.html. Scroll to the section “Download Win32 OpenSSL". Select one of the non-light editions of the installer and download it.



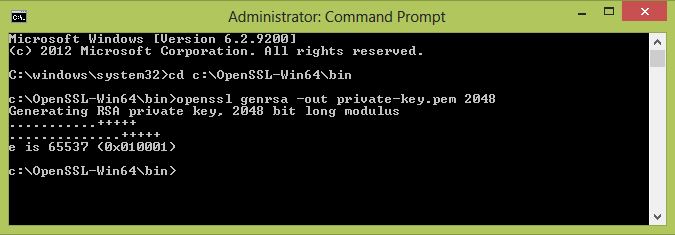
In Windows, open a command prompt and type the following command at the prompt and press Enter: **cd <Your Location>\OpenSSL-Win64**. The line changes to C:\OpenSSL-Win64

Type the following command at the prompt and press Enter**: set OPENSSL\_CONF=c:\OpenSSL-Win32\bin\openssl.cfg**

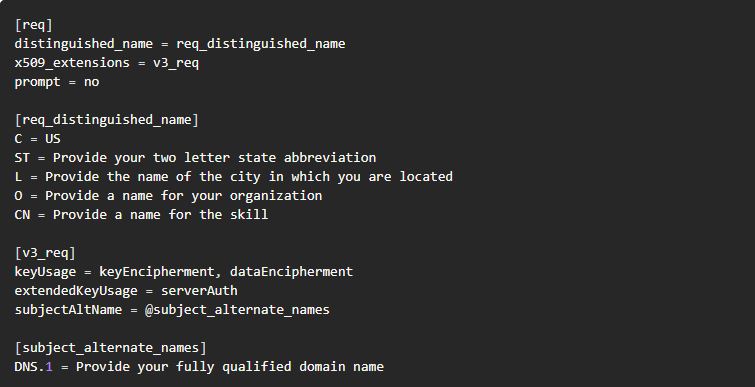


**It is necessary to restart your computer after this step**

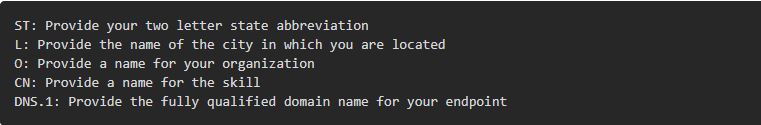
After restart open the command prompt and navigate to **C:\OpenSSL-Win64\bin** then type the following command at the prompt and press Enter: **openssl genrsa -out private-key.pem 2048**



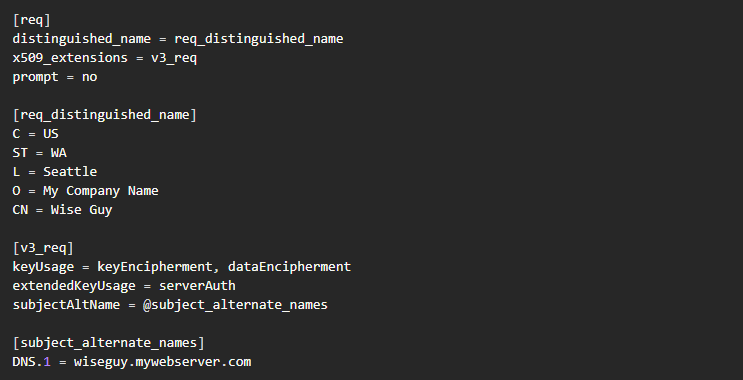
Use a text editor to create a configuration file in the following form and save it as a .cnf file (for instance, configuration.cnf):



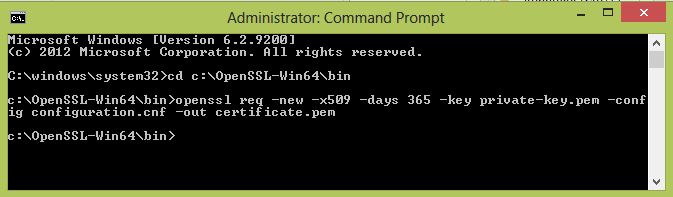
You will need to replace the following content in the configuration file with your own values:



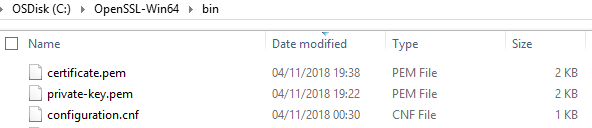
Note that you must provide the AWS Elastic Beanstalk domain name for your endpoint in the DNS.1 section hence the reason we waited until after the AWS setup to create the cert. See below for a completed sample configuration file.



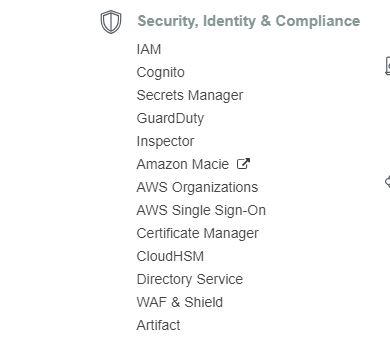
Use the following command to generate a certificate. Specify the names you used for your private-key.pem and configuration.cnf files: **openssl req -new -x509 -days 365 -key private-key.pem -config configuration.cnf -out certificate.pem**. This produces a self-signed certificate in a file called certificate.pem.



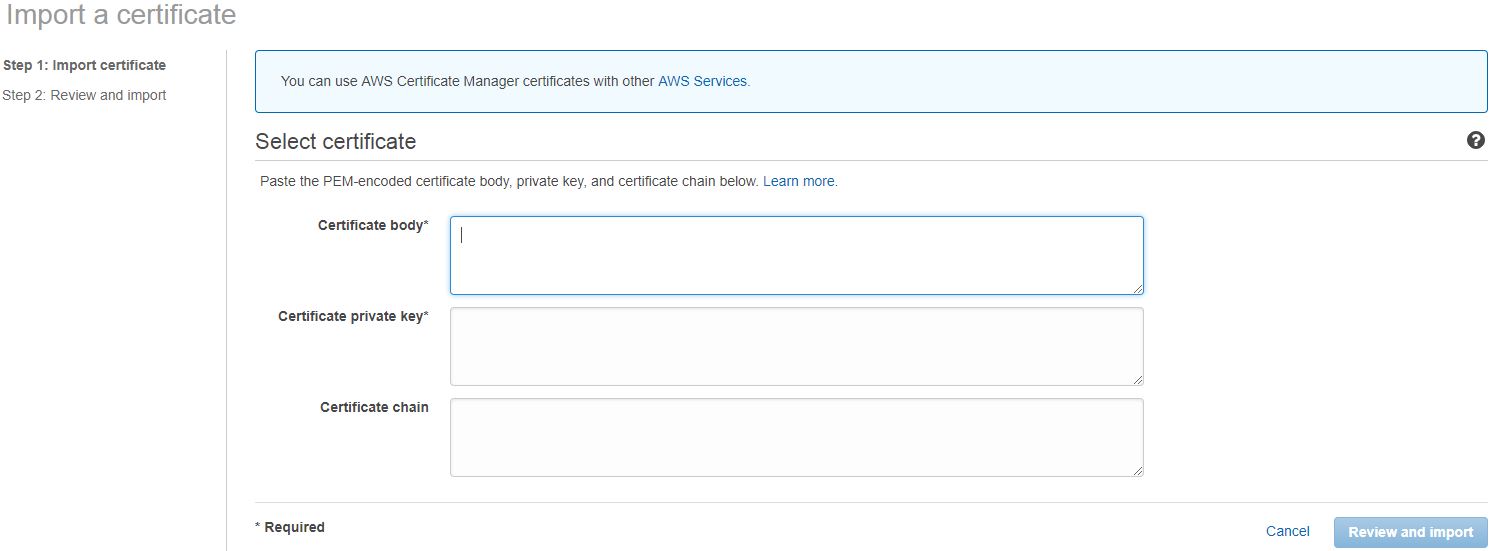
These files are now located in the OpenSSL-Win64\bin directory



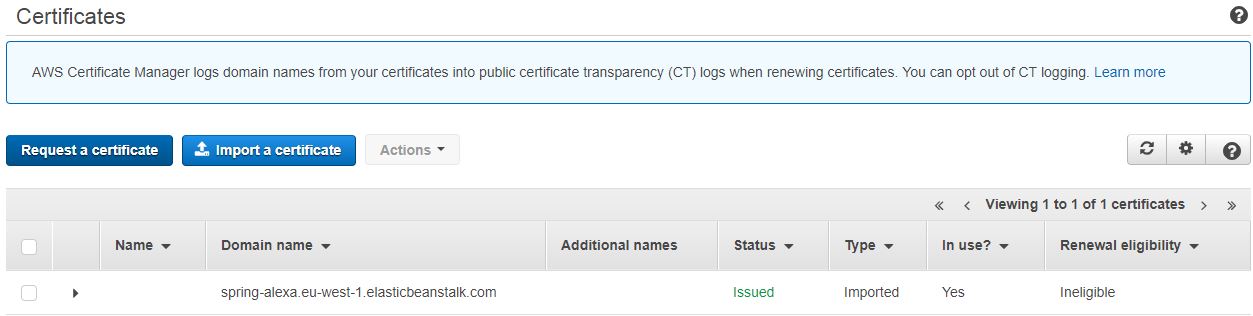
The Certificate and Key now need to be imported into AWS Certificate Manager. You need to select Certificate Manager from the AWS Console.



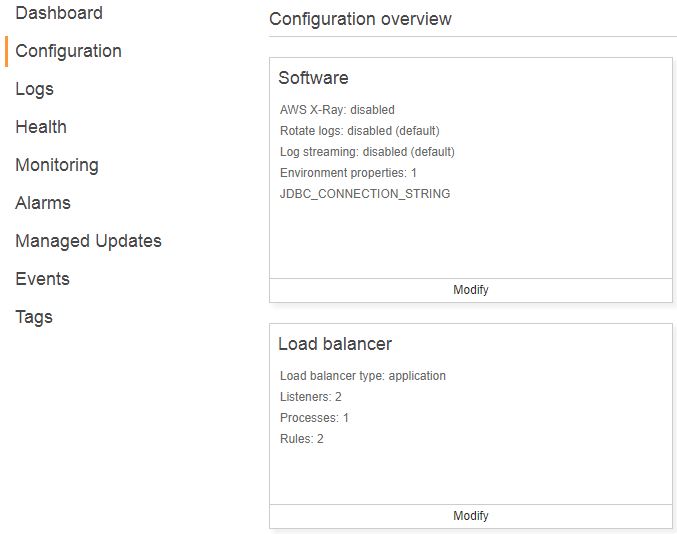
Open the certificate.pem and the private-key.pem in a text editor and paste everything between START and END into the appropriate fields.



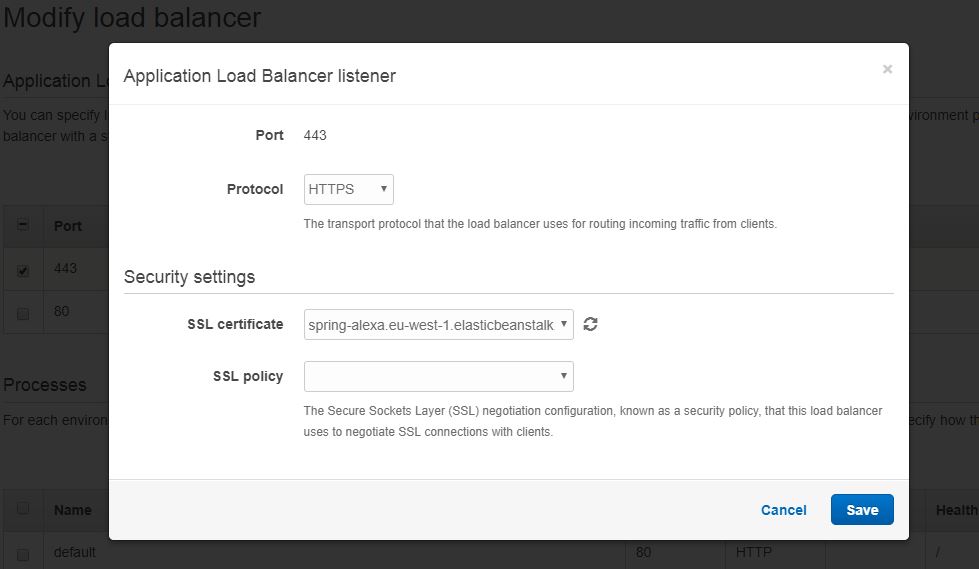
Then Click ‘Review and Import’



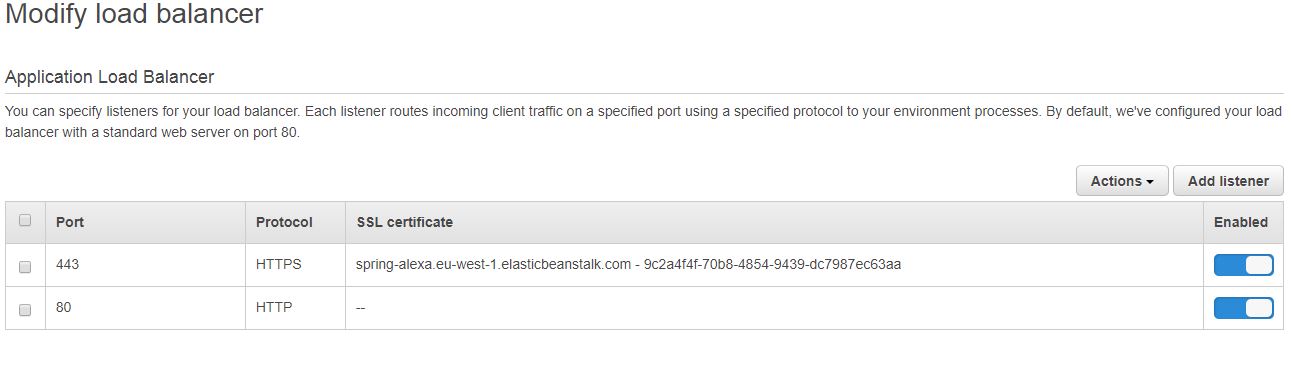
Now, you need to terminate the SSL certificate on the load balancer in your web server environment. Enter the web server environment and click on configuration and select the load balancing section.



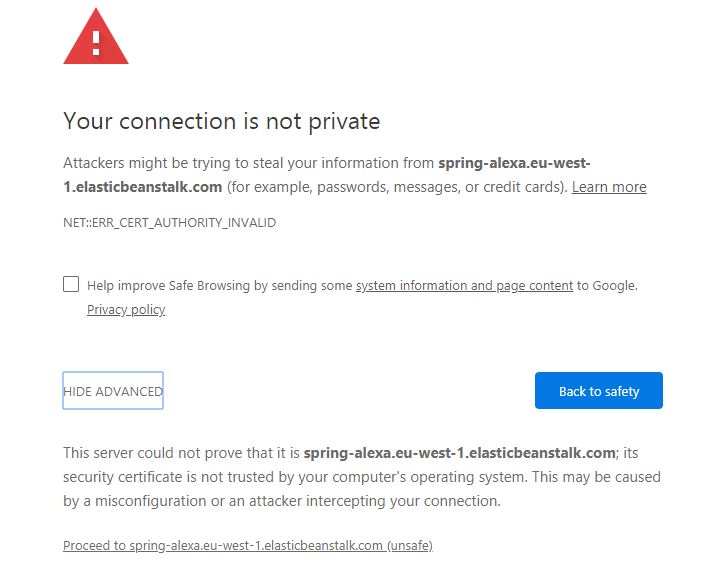
Click on Add a Listener and select protocol HTTPS and enter port 443 and select your newly uploaded certificate from the 'SSL certificate ID dropdown.



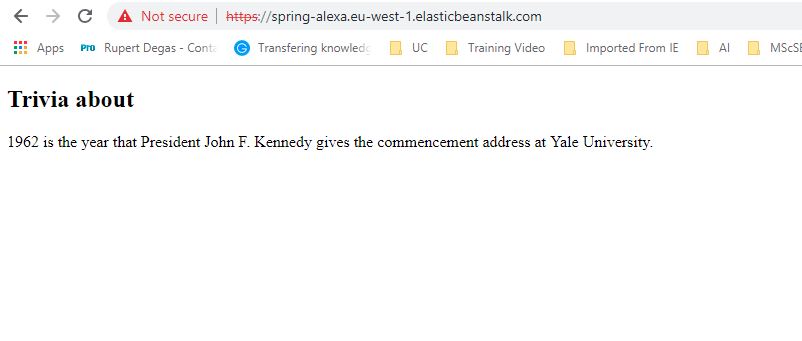
Now that the certificate is installed



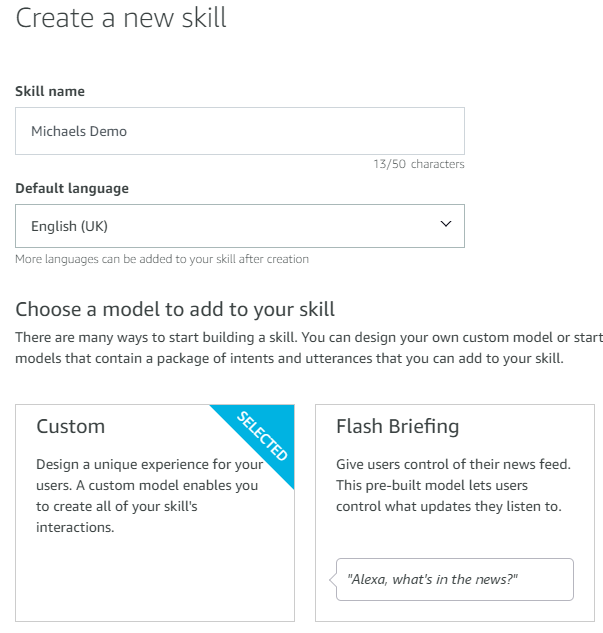
Browse to the URL and see if it works. You should be greeted by an image similar to below. This is because the certificate is not signed by an authority trusted by your browser. To get around this, simply click on 'Advanced' and choose to proceed to the URL.



All being well, you should now be able to access your URL over SSL.



Now move to the Alexa Developer Console. Set up a new custom skill within the console. I’ve named my skill “Michaels Demo”, and under the Invocation menu, I set the Skill Invocation Name to “michaels demo”.



Every Alexa Skill sentence start with the wake word (“Alexa”) and the Skill Invocation Name (“my demo app”).

*"Alexa, ask my demo app ..."*

Everything after the Skill Invocation Name is known as an utterance which needs to be listed in the developer console, and these need to be mapped to an intent. A single intent normally has many utterances, representing the variations a user might say. For example:

*"tell me trivia about a random year."*

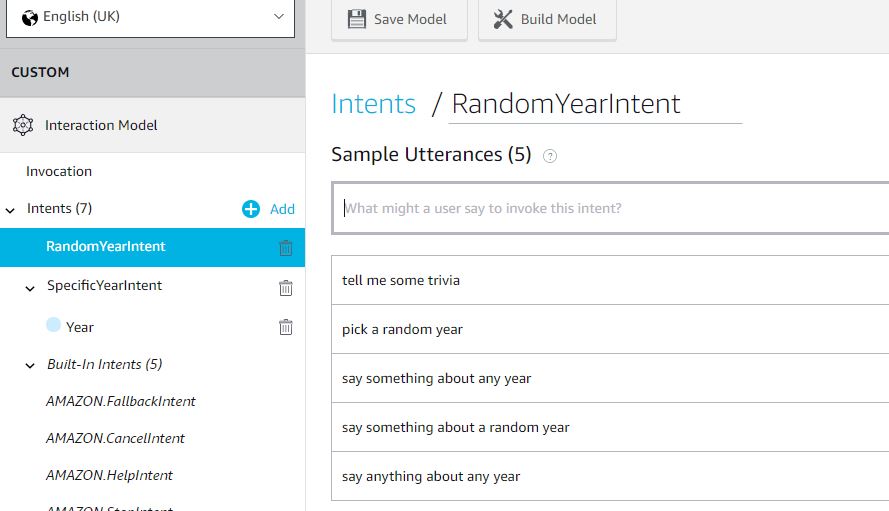
*"say something about any year."*

*"pick a random year."*

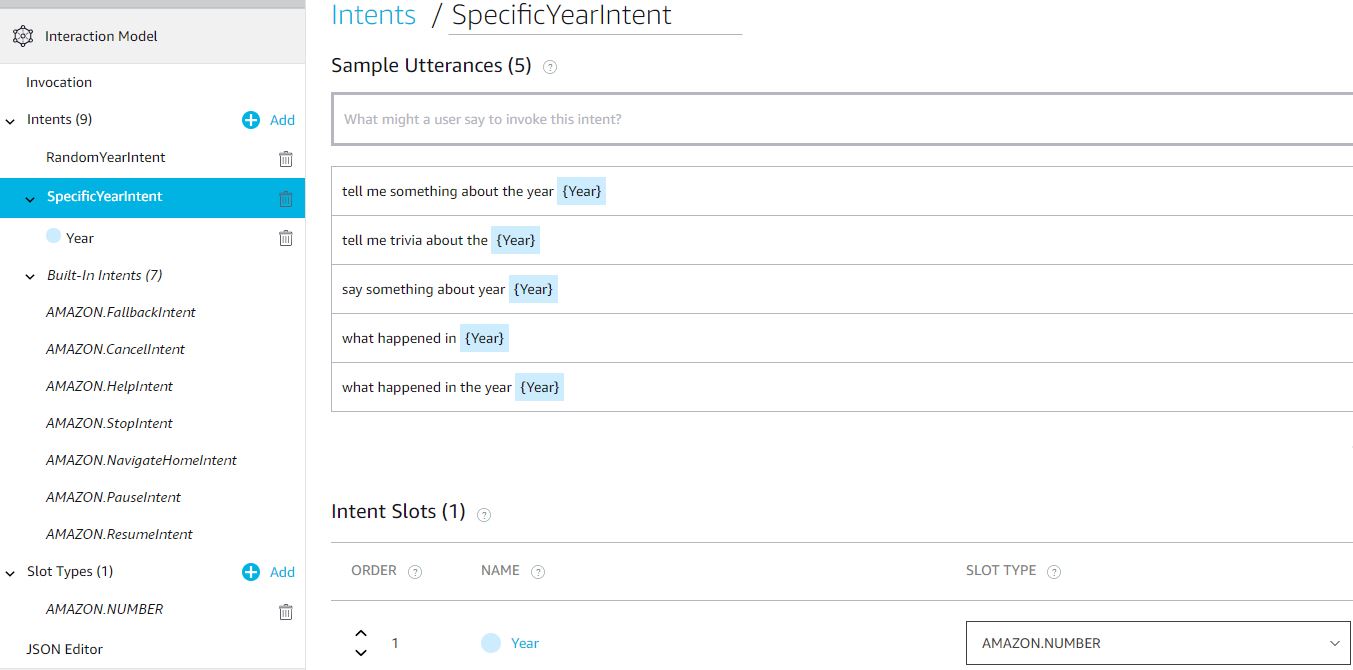
*"tell me some trivia."*

*"say anything about any year."*

All of these utterances mean the same thing, and can map to a single intent which in the app is called the “RandomYearIntent”.

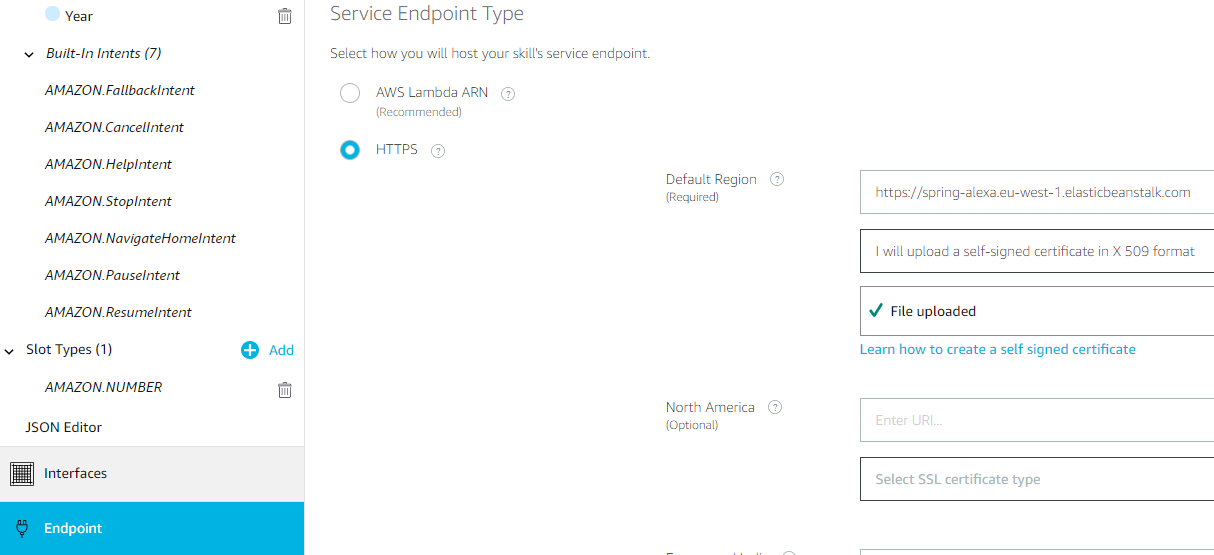


We will also implement a “SpecificYearIntent” which allows the user to say any year. For example: *“Alexa, ask my michaels demo, what happened in the year 1984?”* Since the year is highly variable the utterance is defined using a slot, as: *“Alexa, ask michaels demo, what happened in the year {Year}?”* where {Year} represents a number and is defined as a Slot named “Year”, which is of type “AMAZON.NUMBER”



In the Spring-Boot application the “HandlerSpeechlet” determines which intent is being invoked and calls specialized handlers for each intent i.e. for each intent, there is a class that implements the interface so for the intent “RandomYearIntent” there is a class named RandomYearIntentHandler etc. But in order to call our backend application we must first navigate to Build > Custom > Endpoint in Alexa Skill Configuration:

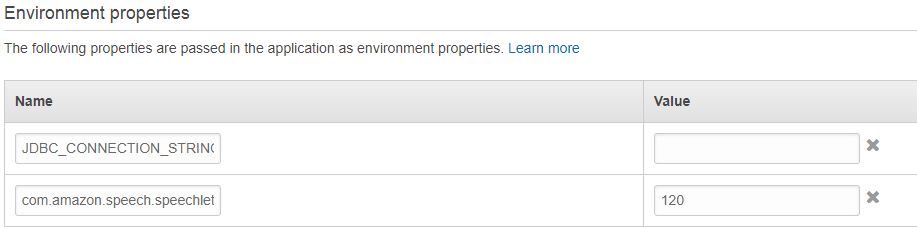
* For the Service Endpoint Type, select HTTPS.
* Enter the endpoint in the region to configure, such as Default Region.
* Under the endpoint field, select the option I will upload a self-signed certificate.
* Click the Upload Certificate box and choose the .pem file for your certificate that you generated previously. The command shown above generated a certificate in a file called certificate.pem.



It is now necessary to set some Java System Properties for Elastic Beanstalk

com.amazon.speech.speechlet.servlet.timestampTolerance: Defines the tolerance, in seconds, to allow when verifying the timestamp on a request. Every request sent to your web service by Alexa includes a timestamp. Your service should allow a tolerance of no more than 150 seconds (two and a half minutes). This means that your service should only accept requests in which the request timestamp is within 150 seconds of the current time.

1. Log in to the AWS Management Console. Navigate to Elastic Beanstalk and then navigate to your new environment.
2. In the left menu, click Configuration, then open the Software Configuration section.
3. Under Environment Properties, scroll to the bottom of the list and find the blank rows after any existing properties.
4. In a blank row, enter the following com.amazon.speech.speechlet.servlet.timestampTolerance.
5. Property Value: Enter the value you want for the property.
6. Click the plus button to add the new property.
7. Click Apply to save the changes and update your environment.



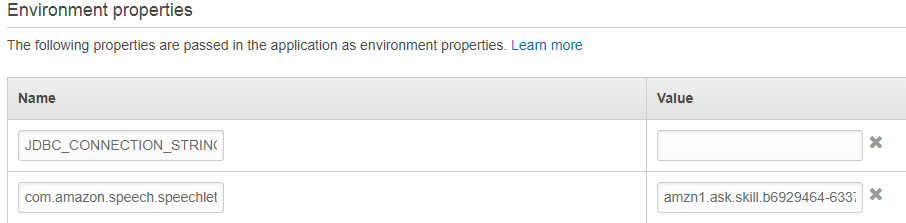
com.amazon.speech.speechlet.servlet.supportedApplicationIds: Defines a comma-separated list of application IDs supported by your service. Before your web service accepts a request, you should verify that the request is actually intended for your service. This protects your endpoint from someone else discovering your endpoint address, configuring their own skill with that endpoint, and using that configuration to send requests to your service. To do this validation, every request sent by Alexa includes an application ID. You can check this ID to ensure that the request was intended for your service.

Open the developer console to your list of skills. Each skill has the skill ID which is show by clicking the link ‘View Skill ID’ shown below the Skill Name.

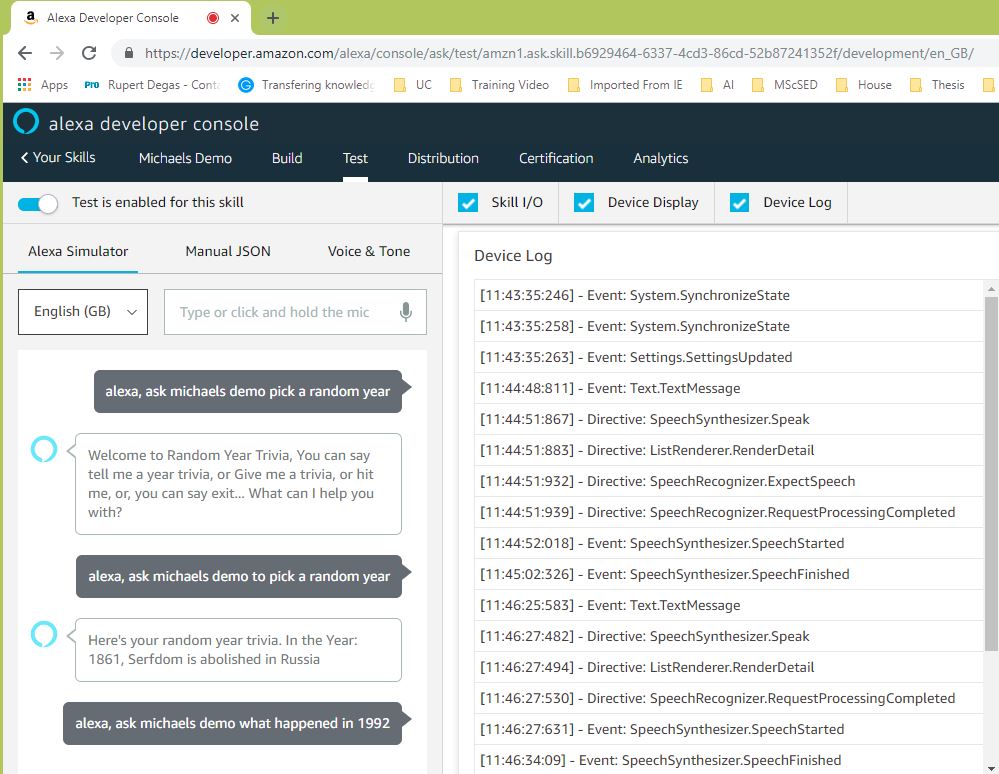


The skill ID will be displayed in the format “amzn1.ask.skill.XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX”

1. Log in to the AWS Management Console. Navigate to Elastic Beanstalk and then navigate to your new environment.
2. In the left menu, click Configuration, then open the Software Configuration section.
3. Under Environment Properties, scroll to the bottom of the list and find the blank rows after any existing properties.
4. In a blank row, enter com.amazon.speech.speechlet.servlet.supportedApplicationIds
5. Property Value: Enter the ID value.
6. Click the plus button to add the new property.
7. Click Apply to save the changes and update your environment.



It is now possible to test the Alex Skill



## 1.4 Conclusion

I am still having some issues with the Endpoint connection between Alexa Skill and AWS Beanstalk which I will continue working on and update this doc as soon as this issue is resolved.

**References**

Configure Your Web Service to Use a Self-signed Certificate | Custom Skills. (2018). Retrieved from <https://developer.amazon.com/docs/custom-skills/configure-web-service-self-signed-certificate.html>

Create New Skill. (2018). Retrieved from <https://developer.amazon.com/alexa/console/ask/create-new-skill>

Deploy a Web Service for a Custom Skill to AWS Elastic Beanstalk | Custom Skills. (2018). Retrieved from <https://developer.amazon.com/docs/custom-skills/deploy-a-web-service-for-an-alexa-skill-to-aws-elastic-beanstalk.html>

Dixon, J. (2018). Installing a self-signed SSL certificate on Elastic Beanstalk. Retrieved from <https://joedixon.co.uk/installing-a-self-signed-ssl-certificate-on-elastic-beanstalk>

Fiol, R. (2018). How to implement an Alexa Skill with Spring Boot – freeCodeCamp.org. Retrieved from <https://medium.freecodecamp.org/implementing-an-alexa-skill-with-spring-boot-also-why-would-you-do-such-a-thing-9992c0797646>

Fradet, B. (2018). Create a new application in Elastic Beanstalk and upload the WAR file into it. Retrieved from <https://github.com/snowplow/snowplow/wiki/Create-a-new-application-in-Elastic-Beanstalk-and-upload-the-WAR-file-into-it>

Generate a Certificate Signing Request (CSR) using OpenSSL on Microsoft Windows system. (2018). Retrieved from <https://knowledge.digicert.com/solution/SO27347.html>

Introduction to Alexa | Codecademy. (2018). Retrieved from <https://www.codecademy.com/courses/learn-alexa/lessons/intro-to-alexa/>