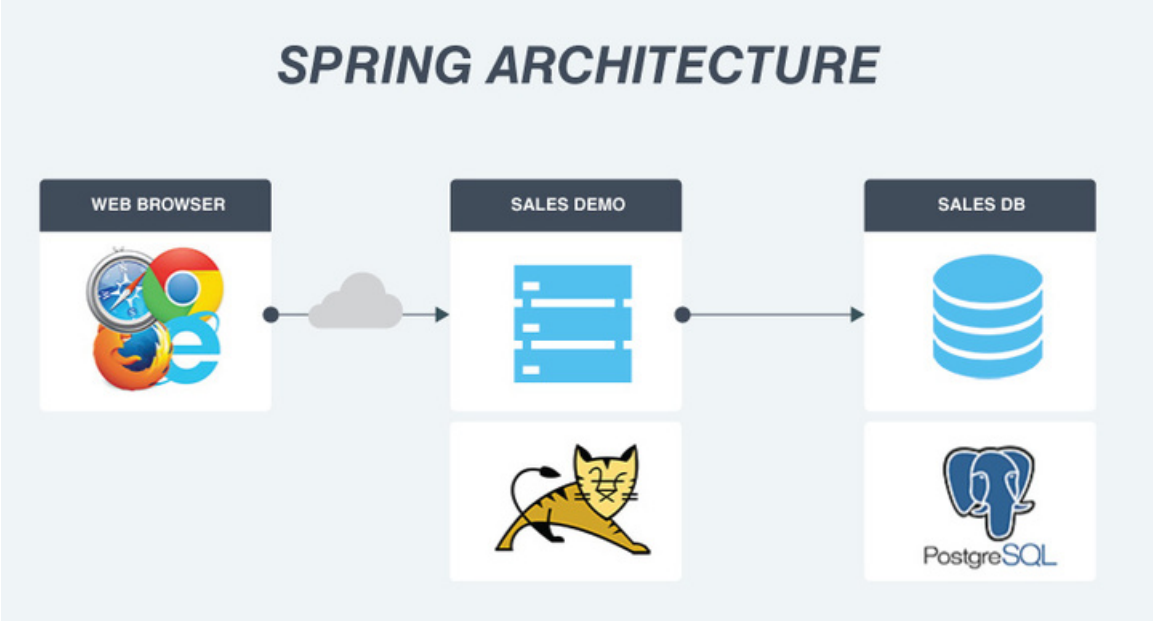
Migrating Spring App to MicroServices App on AWS

The company I am working for has recently gone through a migration of refactoring our code base from a monolithic application (Java Spring WAR) into a MicroServices Application hosted on the Amazon PAAS (specifically Beanstalk and CloudFront). As part of this blog post I have provided a simple Sales Demo application and will discuss the steps of that will be required for this refactoring to bring this into a Beanstalk/CloudFront Applcation.

For the purposes of this blog I will be using a SalesTax demo application and the code can be found here (https://github.com/shannonlal/salesdemo). This website will allow users a list of products followed by the ability to create an order and apply sales tax. The following is a diagram of the Spring Architecture:



The above architecture is a pretty standard Spring architecture for most monolithic web applications. In our migration we broke up our code and separated the backend services from the front end content JSPs(Now HTML), CSS and JS. The following is a diagram illustrating our model of how we controlled access:

Amazon Web Services

I am going to start by explaining at a high-level what these different components in AWS are and how we integrate them together.

Route 53

Route 53 is a Domain Name Service(https://aws.amazon.com/route53/) which allows you to route traffic to different internal AWS services. In our model we used Route 53 to host our DNS servers (for example www.mycompany.com).

S3

Amazon S3 (https://aws.amazon.com/s3/) is a simple storage service which allows you to store content (html, css, js files in buckets in the cloud). In this demo we will be using Amazon S3 to host the static content (html, css, and JS).

Beanstalk

Beanstalk (https://aws.amazon.com/elasticbeanstalk/)is an application stack which will be used to host our individual services. Beanstalk has access to multiple stacks (Tomcat, PHP, Node, Ruby, Go, .Net). In this demo we will be using Beanstalk to host our different web services (as Spring WARS running on Tomcat).

RDS

Amazon Relational Database Service (RDS https://aws.amazon.com/rds/) will be used to host our database. We will create an RDS database and our web services will be used to connect to the database.

CloudFront

Amazon CloudFront is the glue that will tie all your different services together under one common url. We will define an origin (which will correspond to our url defined in Route 53 www.mycompany.com). When the user hits this url Route53 will route the traffic to CloudFront. CloudFront will host the content and push it to edge locations around the world. In CloudFront you are able to redirect traffic based on URL patterns. For example anyone coming to the default pattern (/*) can be redirected to a bucket in S3 which hosts your static content (i.e. html, css, images). If they come to say an API url (/api/users/authenticate) you can route them to a Beanstalk service in the backend.

Infrastructure Security

In our production systems we have all our web services hidden behind different VPCs and have implemented network rules to restrict access to our backend services. I do not think I will have time to address this in this blog but will try to talk about this in my next blog.

Application Security

One major component I have not included in the Sales Demo is Spring Security. In our application we removed our Spring Security and replaced access control using an API Gateway. I will discuss this concept briefly at the end of this blog.

NOTE: AWS is a very sophisticated and complex ecosystem and provides multiple ways to integrate these different services. The model I will be discussing will be is similar to the model which we implemented at our company.

SalesTax Application Overview

The SalesTax Demo application will look like a traditional Spring Application with once exception. The JSP pages do not follow the traditional Spring MVC model with data being passed from the controller and then the JSP pages rendering the view. Instead we are using Angular to manage displaying the content, which will make REST calls to the backend controllers. The reason that we are doing this is so that we can migrate our static content (html, css, js files) to S3 buckets and have our backend services run in beanstalk.

Controllers

Name	Description
ViewController	The View Controller will handle the requests for viewing different HTML pages
APIController	The API Controller will handle the requests to update orders, view orders and products

Services

Name	Description
Product Service	The Product Service will handle requests to view product information
Order Service	The Order Service will handle viewing, creating and updating orders

DAO

Name	Description
AbstractDao	This is a generic DAO which will handle calls for creating, updating, retrieval and deletion of the different entities

Entities

Name	Description
Product	The Product defines the characticts of the product and its tax information
Item	The item represents a product and its quantity for an order
Order	The Order represents a list of items which have been purchased

JSP Pages

Name	Description
Index.jsp	The main langing page which will load the javascript libraries and load the angular modules
Orders.jsp	The orders.jsp will display a list of orders that are defined in the system
Products.jsp	The products.jsp will display a list of products in the system
Updateorder.jsp	The update order jsp page will allow the user to create and remove items from the order

Configure SalesDemo in BeanStalk

The first step will be to build the application and deploy it into a beanstalk instance. To checkout the code please run the following command:

git clone https://github.com/shannonlal/salesdemo step0

You can import the project into your IDE (Eclipse, NetBeans, STS, etc) or you can just build this from the command line. To build the project run the following commands:

mvn clean install

If you want, you can deploy the application locally in tomcat to verify that it deployed correctly. **Note: The initial application uses Derby as an in memory DB so no JDBC connections are required. At a later step we will connect to an RDS instance in Amazon.**

Once this is done you have a war file create called target/salesdemo.war.

a. Login in Amazon Console and click on Elastic Beanstalk:

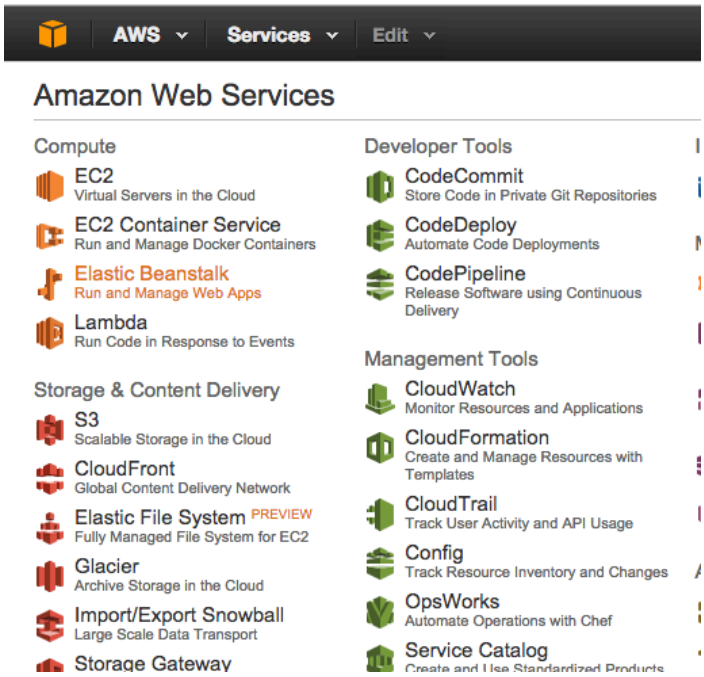


Figure 1 - Login to Amazon Console

b. Select the platform as Tomcat

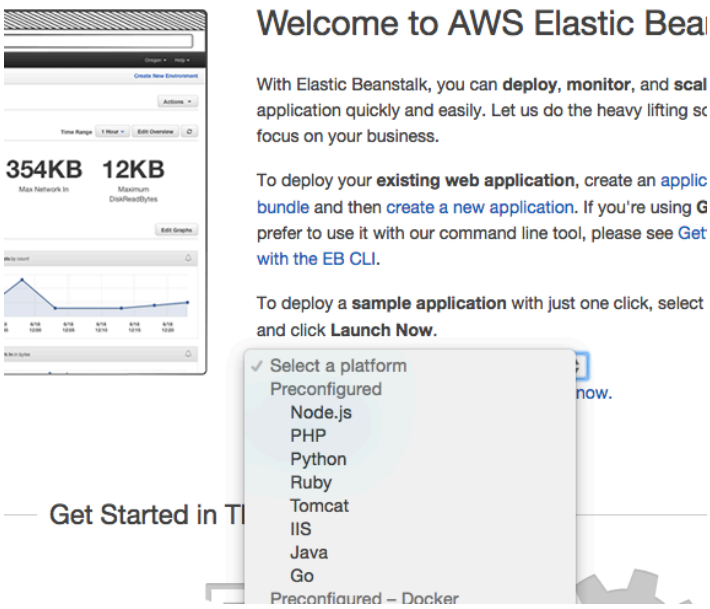


Figure 2 - Select Platform

c. Select Environment as Web Server

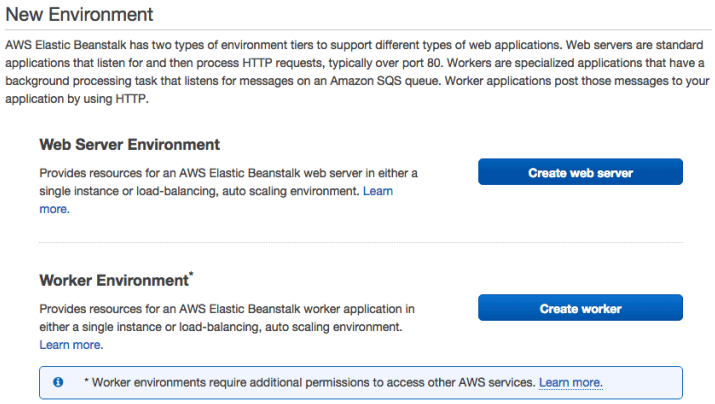


Figure 3 - Select Environment

d. Select Environment Type as Single Instance

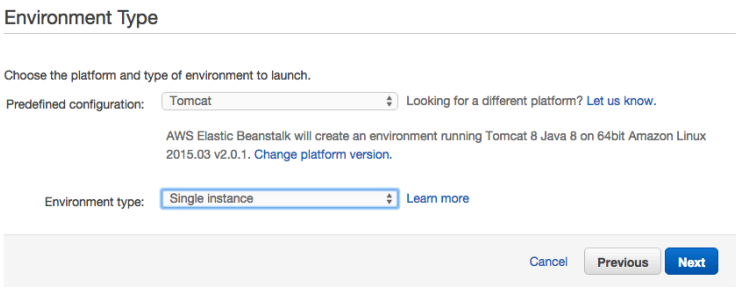


Figure 4 - Environment Type

e. Select Upload and Sales Demo WAR

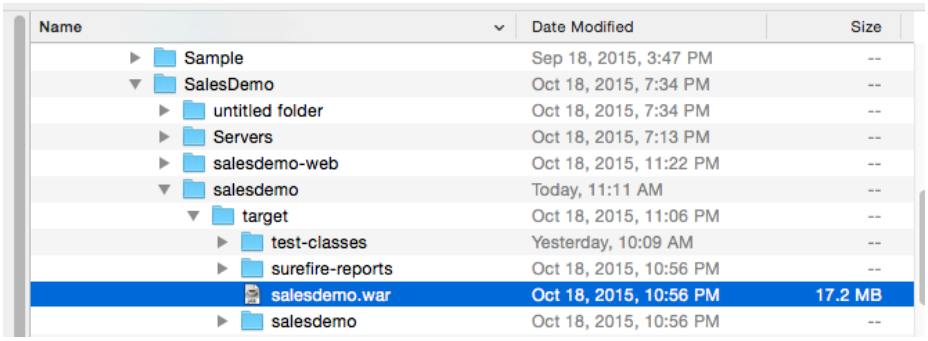


Figure 5 - Upload the WAR

f. Provide the Environment Name

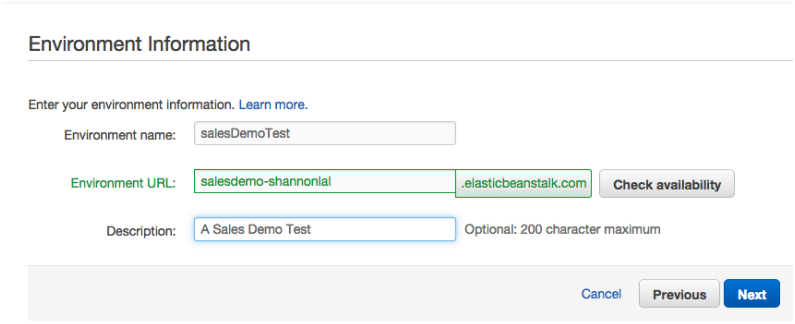
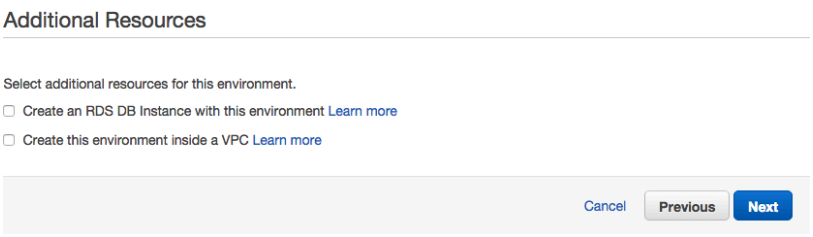


Figure 6 - Environment Name

g. Proceed to the next step



h. Specify Instance Information

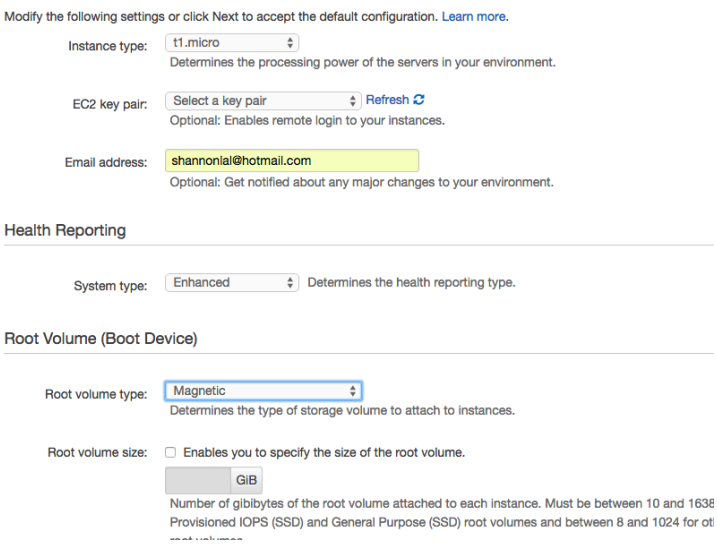


Figure 7 - Select Instance Type

i. Select the create the default role for the Elastic Beanstalk

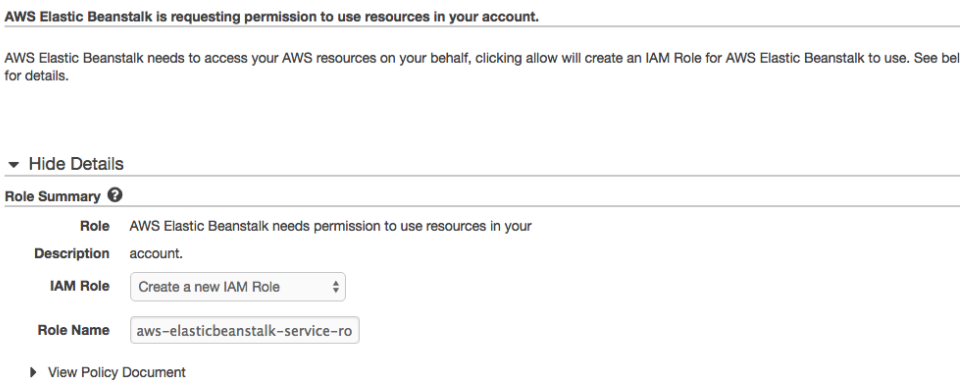


Figure 8 - Create default Role

j. Launch the new Instance

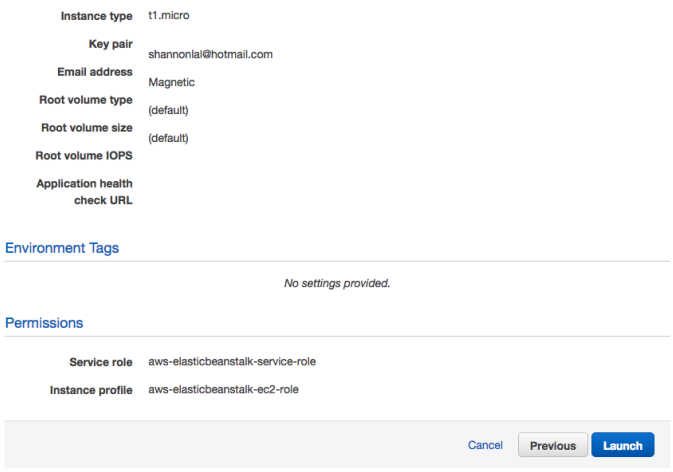
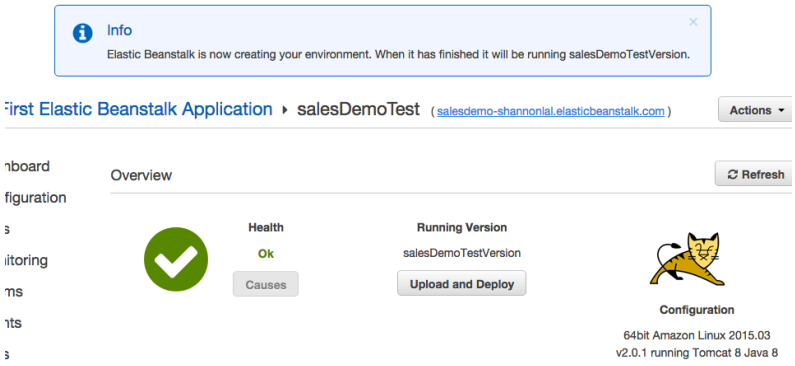


Figure 9 - Launch Instance

k. View Elastic Beanstalk Instance



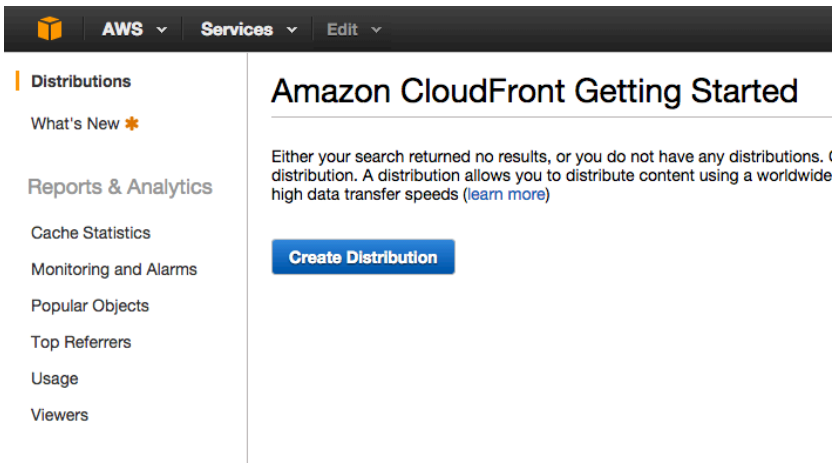
Configure CloudFront to point to your application

Login into the Amazon Console and click on the CloudFront link. At this point you have two options. If you already have your own domain name you can add it to Route 53. The following link provides detailed instructions on how to do this (http://docs.aws.amazon.com/gettingstarted/latest/swh/website-hosting-intro.html). If you do not you can just create a CloudFront Origin and it will give you a url.

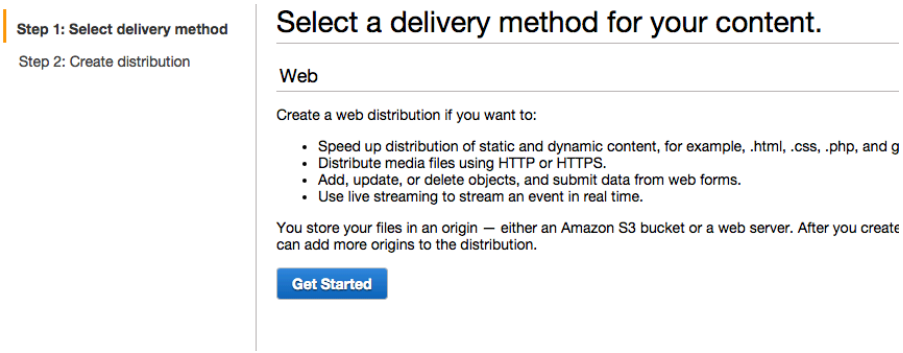
The following steps define how to setup CloudFront to point to your Beanstalk application.

1. Login to Amazon Cloud and select CloudFront

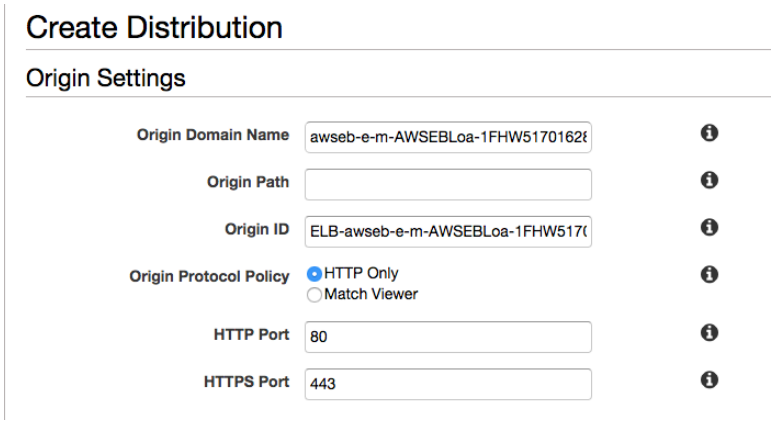
2. Create a new Distribution



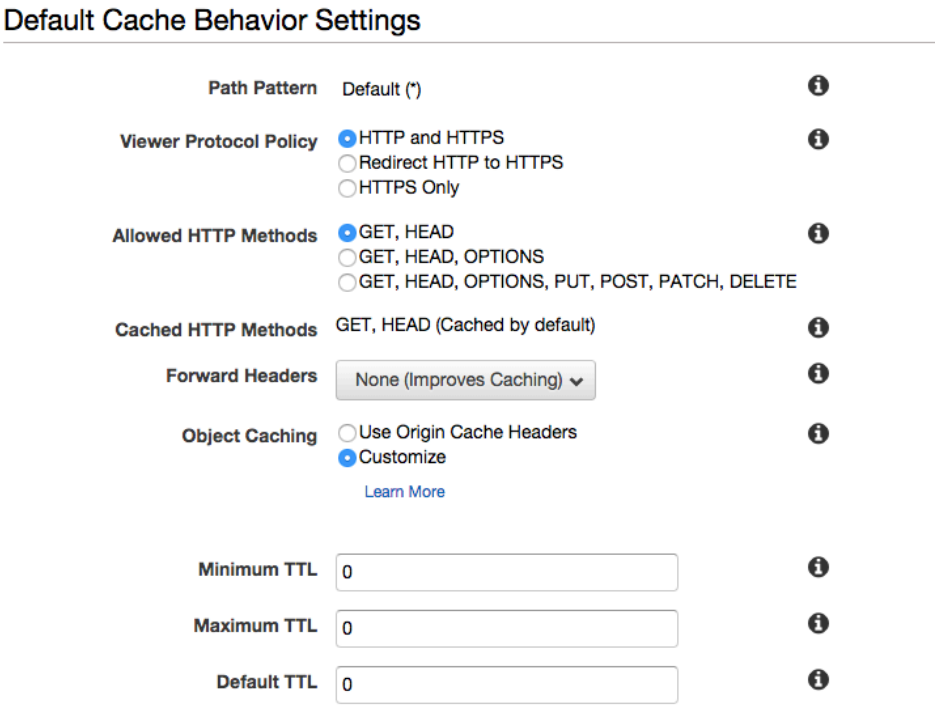
3. Select Web Distribution



4. Select the default ELB

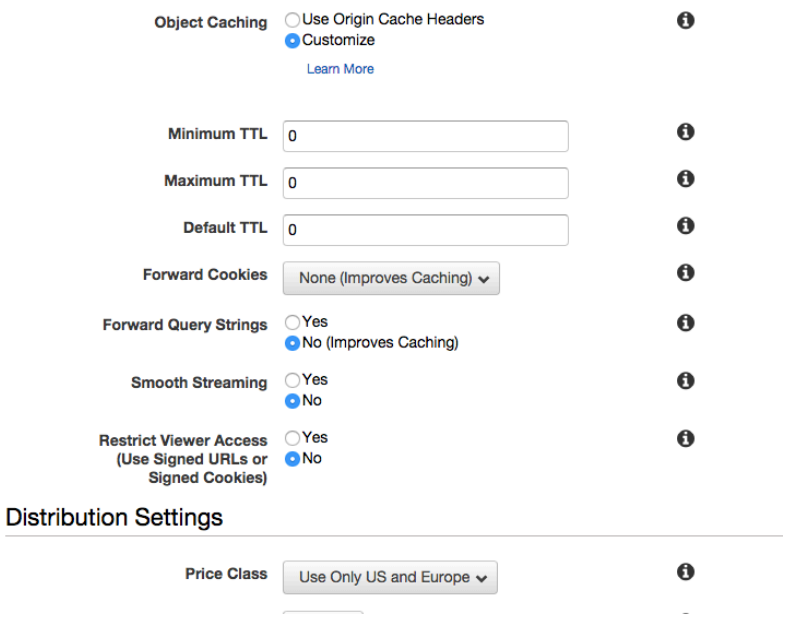


5. Disable Caching

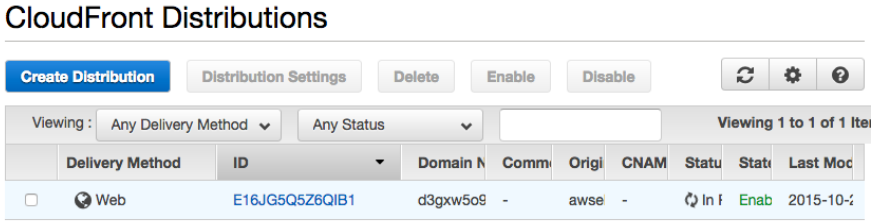


NOTE: This is recommended during the development stages

6. Accept the defaults



7. Check in status



Once your status has changed to deployed, in the Domain Name column you will see the url to you instance of Cloud Front. The format will be something like (https://xxxxxxxxxx.cloudfront.net). When you open this url it will load a page which should be the Sales Demo home page. CloudFront will redirect your request to your Elastic BeanStalk instance.
Note: When you first create a Cloud Distribution and Origin, it will automatically create a default Behavior (/*) for you. In later steps when we want to map to different services (i.e. static content on S3 or other Beanstalk Applications) you will need to create new Origins and Behaviors. This will be discussed later in the demo.

Create an RDS PostgreSQL Instance and connect to your Beanstalk instance

We are going to setup a publicly accessible RDS (Postgres) instance which you will be able to connect to from your local DB tools (pgAdmin or WorkBench).

Git clone https://github.com/shannonlal/salesdemo step1

You can import the project into your IDE (Eclipse, NetBeans, STS, etc) or you can just build this from the command line. To build the project run the following commands:

mvn clean install

Note: If you run the application locally you will need to pass the JDBC URL, Username and Password as environment variables through the command line. This is how parameters can be configured when setting up Beanstalk instances in Amazon

1. Create Local Postgres. The following are the instructions on how to create a local instance of PostGres: https://wiki.postgresql.org/wiki/Detailed_installation_guides

2. Create a user for your database:
   a. Click on create Login Role

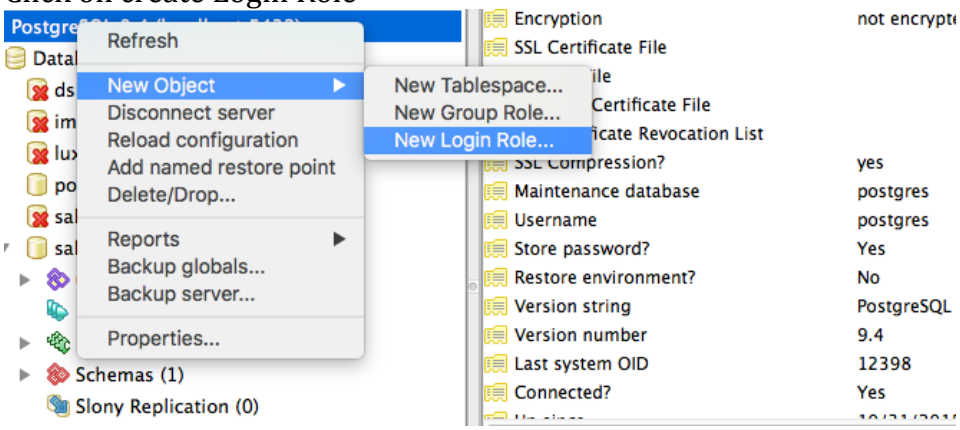


Figure 10 Create User Role

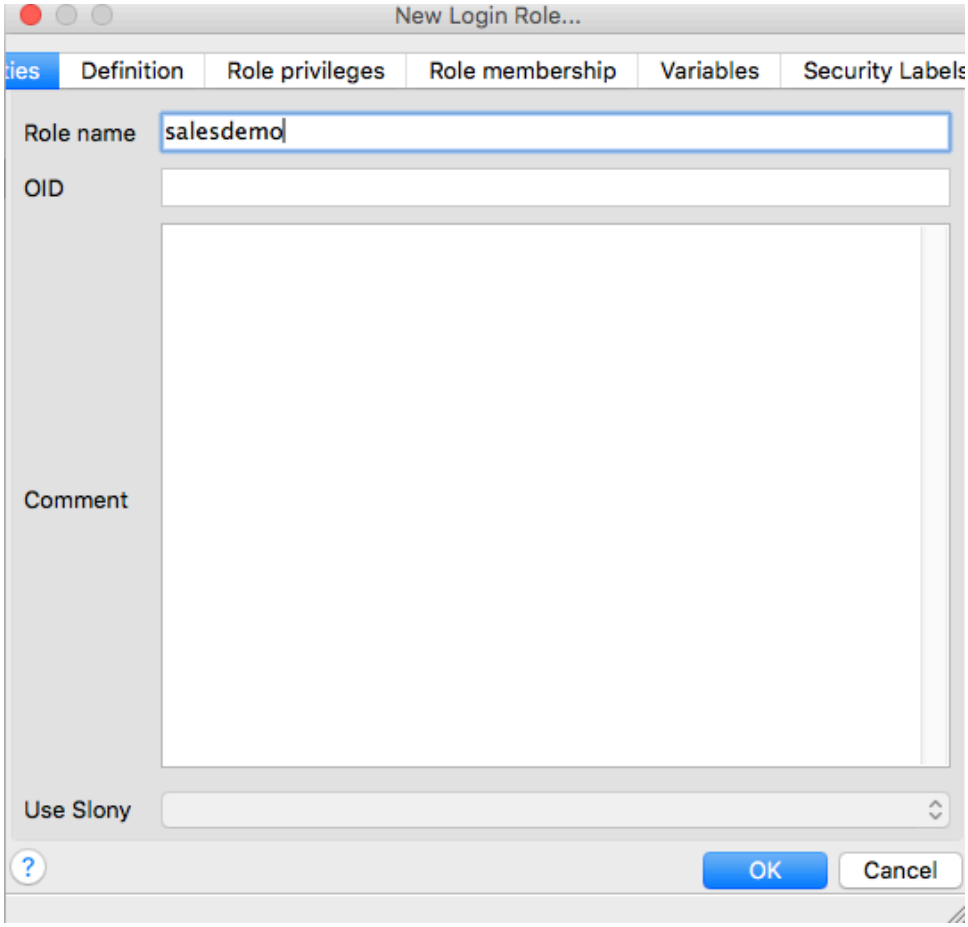


Figure 11 - Define User role

b. Define a role and set the permissions

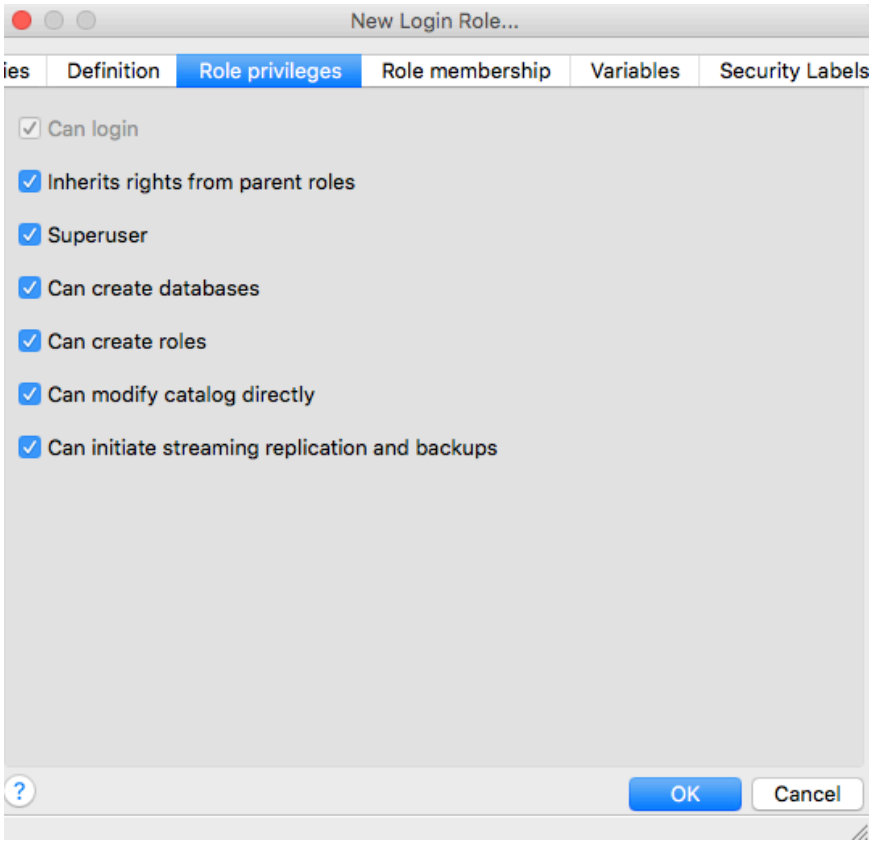
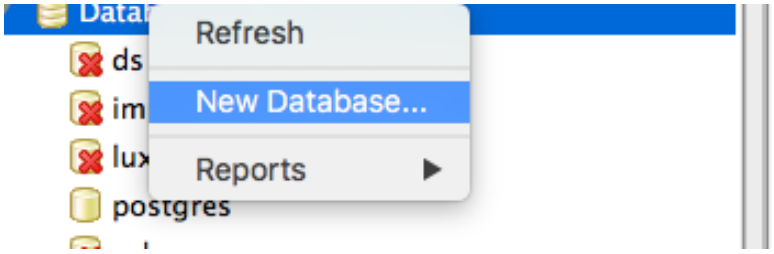


Figure 12 Specify the User Role

c. Create a new instance of the database



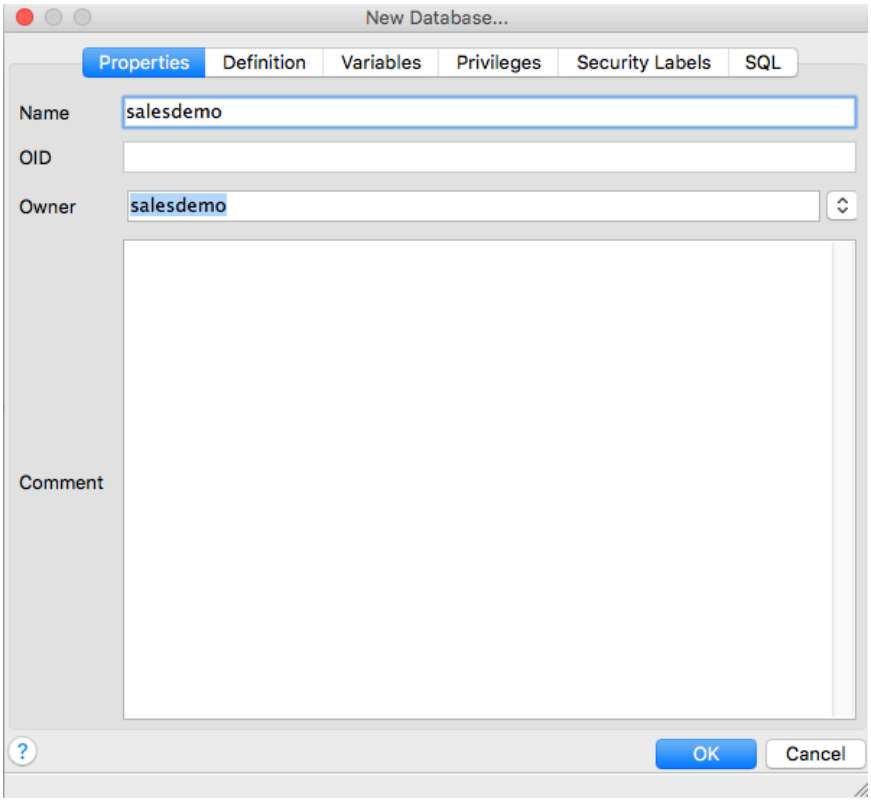


Figure 13 Create the DB Instance

3. Connect to local DB instance using pgAdmin and run the sql script found in the following directory:
   src/resources/sql/createSalesTax-DB-Derby.sql

4. Test Connecting to local tomcat to local PostGreSQL Server
   This is an optional step but is useful if you would like to verify your JDBC Connection. The configuration of this demo site only requires you to specify the jdbc url, user and password to connect to the local and remote database.

   To specify these parameters it is recommended to append these parameters as parameters on the command line. When running applications in Beanstalk you can pass environment variables as part of the configuration. This allows you to change settings when the server is in production. You can obviously embedded the parameters in property file, which is included as part of the WAR, but you will have to load up a new WAR if you which to change any DB settings. The following is a sample command line string which you will need to append to Tomcat or(in Eclipse, Netbeans, etc).

-DDB-URL="jdbc:postgresql://localhost:5432/salesdemo" -DDB-User="salesdemo" -DDB-Password="sales"

You can file this file in src/main/resources/config/server-config.txt

5. Create an RDS instance in the Amazon Cloud
    a. The following stesps will illustrate how to connect to create an RDS instance in the cloud. Log into the Amazon Cloud console and click on the link for RDS.

    b. Click on Launch new instance

    c. Click on PostgreSQL and click Select

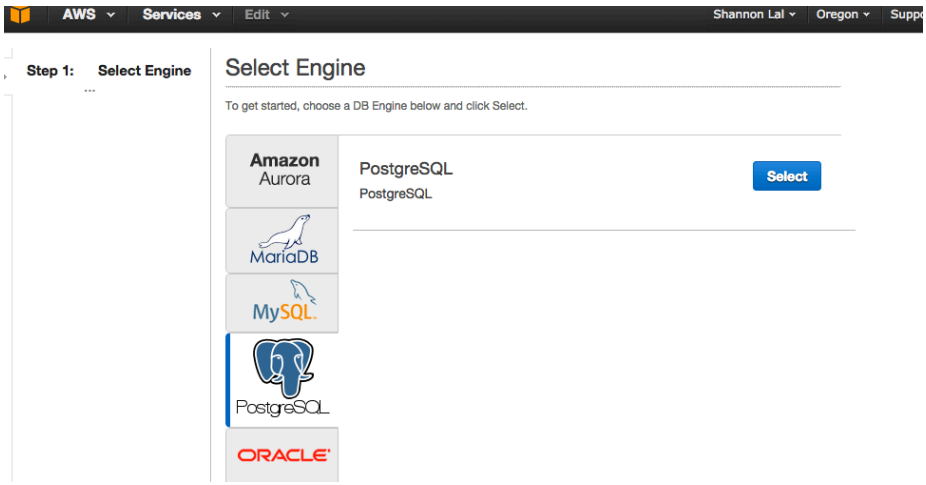


Figure 14 Create PostgreSQL RDS Instance

    d. Click on No, for Multi-AZ and then click Next

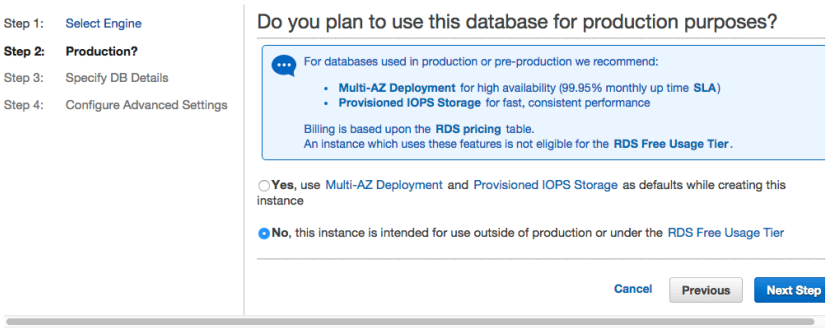


Figure 15 RDS Environment Selection

e. Select the defaults and then enter a DB Instance Name, Username and Password and then Click Next

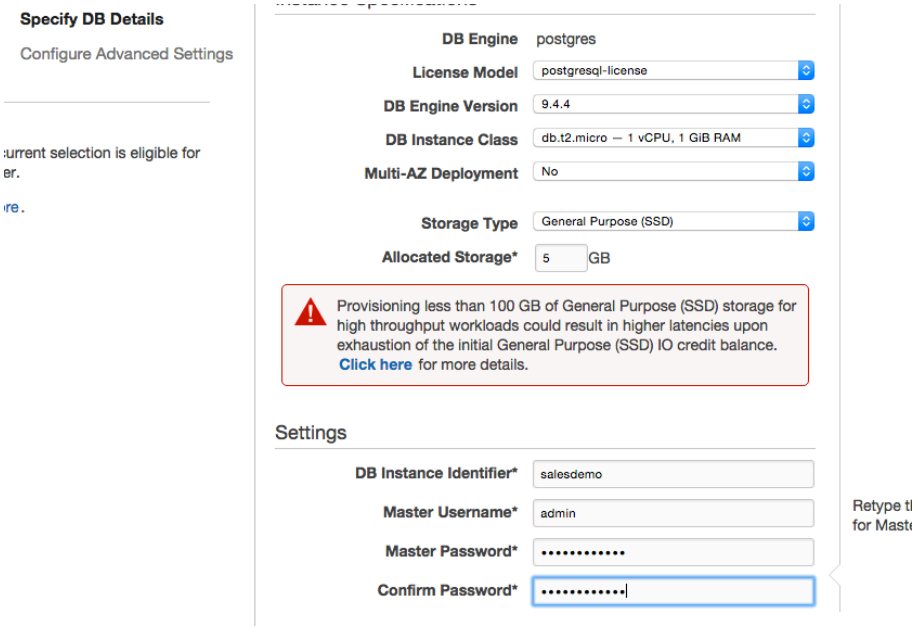
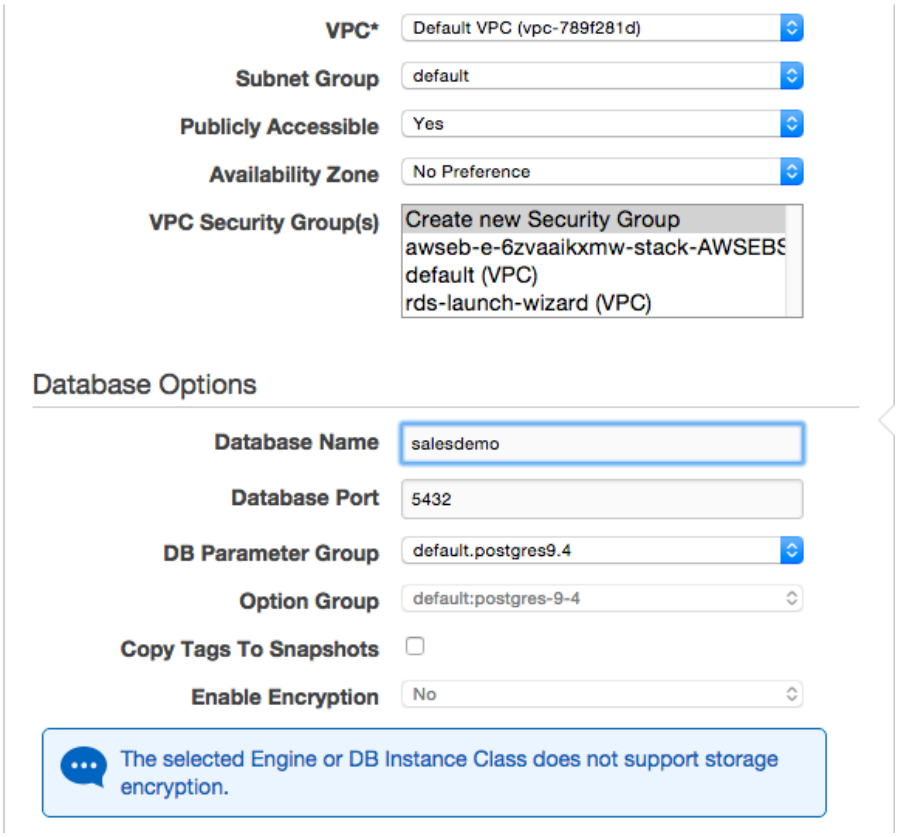
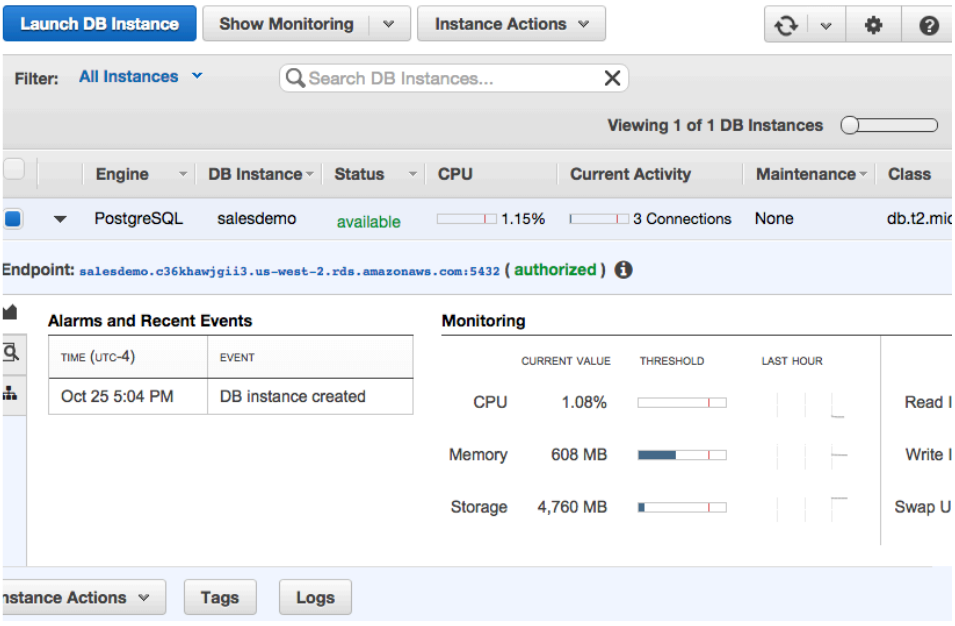


Figure 16 RDS Parameters

f. Select the defaults; however, ensure that you select Publicly Accessible as this will allow you to access the DB Instance from your Application. Click on Launch DB



g. Click on View your DB Instance and wait until the status changes to available

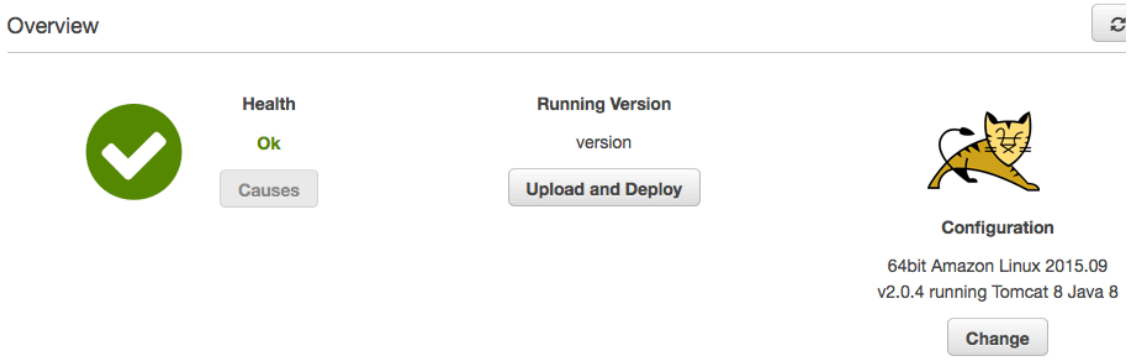


h. Once your DB Instance is available it will provide you with the EndPoint URL which you can use to connect to. For example:

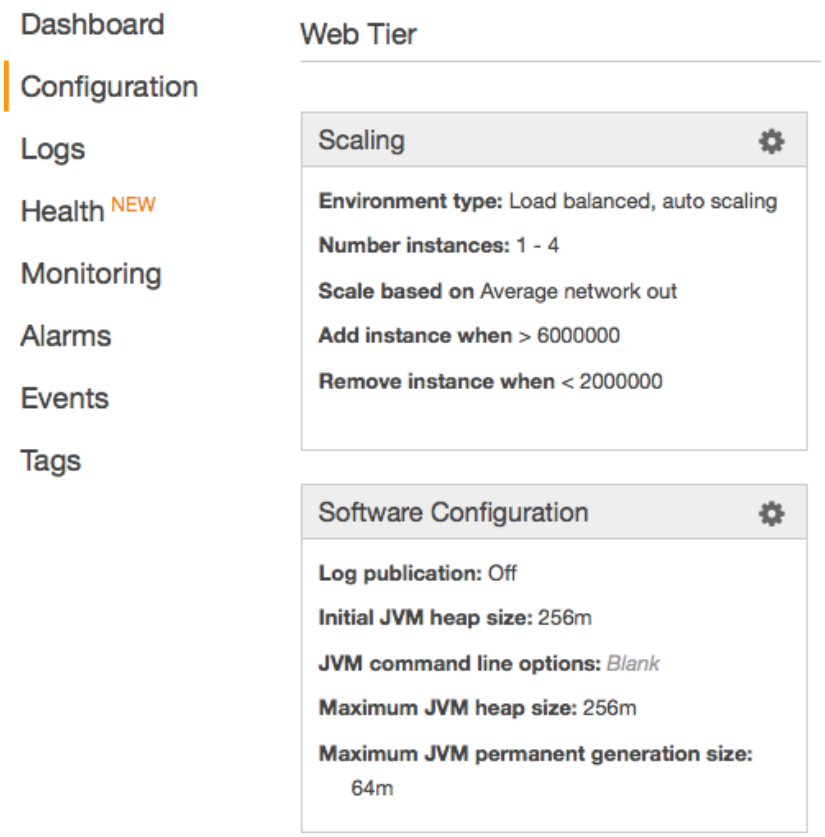
salesdemo.c36khawjgii3.us-west-2.rds.amazonaws.com

6. Connect to publicly accessible Instance
   The following instructions will be on how to test connecting to your PostGreSQL database that has been created in the Amazon Cloud.

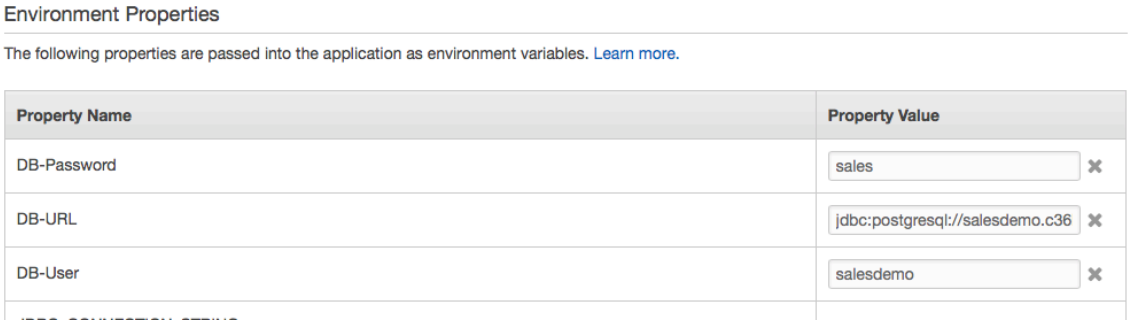
   a. Open pgAdmin (or any SQL Admin Tool)
   b. You will need to provide the Host (Endpoint provided by Amazon), Username (Username you specified when creating your instance) and Password(Password you specified when creating your instance)
   c. In your SQL tool open up a new SQL Script Page and past the contents of the createSalesDemo-Postgres.sql file into the pgAdmin Script page
   d. Execute the script
7. Optional. Re-Run Local Tomcat and connect to Remote DB Instance in Amazon.
   a. In Tomcat or Eclipse modify your DB-URL to the new URL for your Amazon DB Instance in cloud.
   b. Start your Application Server and load the main page. If it works correctly you should be able to get the list of products from the DB Instance in the cloud.
8. Log into AWS Console and Click on Beanstalk.
9. Click on your environment and then click on Upload to deploy the new WAR



10. Click on Configuration and then Software Configuration



11. Under Environment Properties add the 3 properties for DB-URL, DB-User and DB-Password as illustrated below:



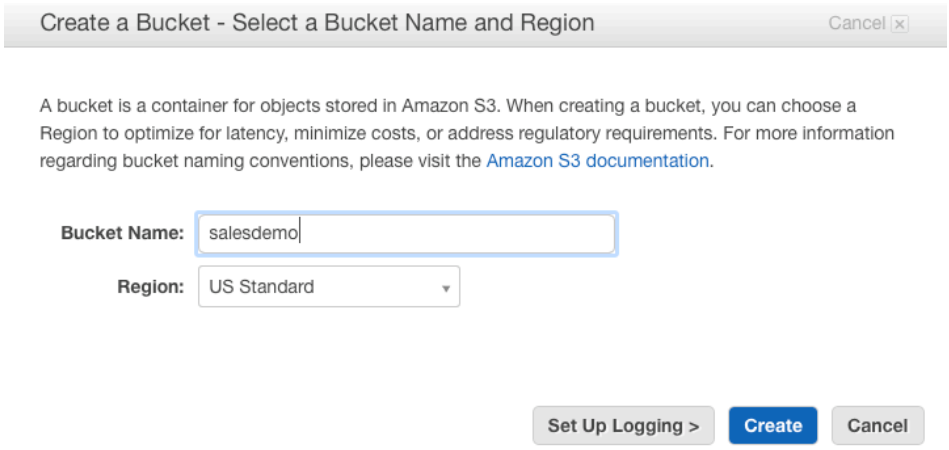
12. Reload your Beanstalk instance from the browser and you should now be seeing the data from the RDS instance hosted in the cloud

Create an S3 Bucket and deploy Static Content to it

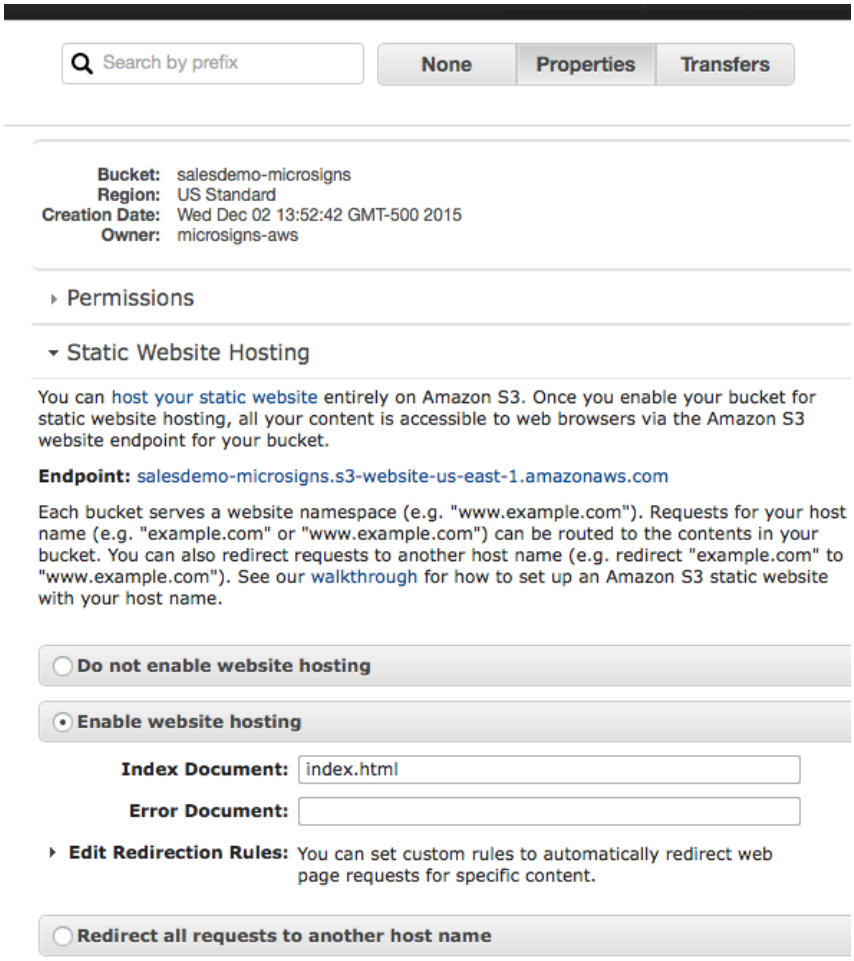
In this step we are going to create an S3 bucket and will move our Static Content (html, css, images, etc) to it. To get the latest code for this we will need to pull down the latest changes from the git. Run the following command

Git clone https://github.com/shannonlal/salesdemo step1

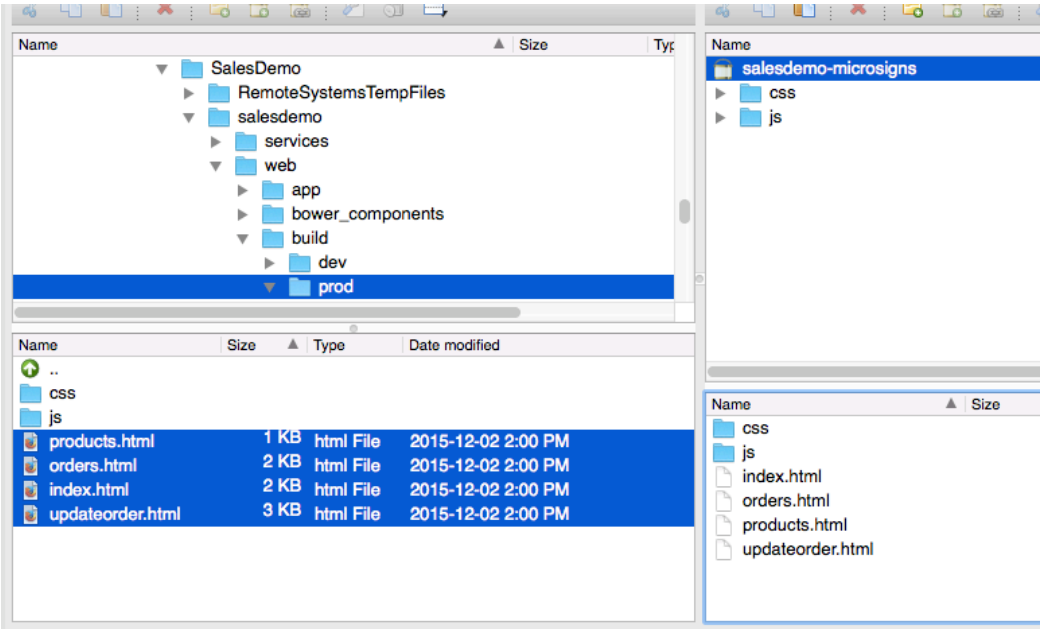
Log back into the Amazon Console and click on S3. Click on Create Bucket and create a new bucket.



Once your bucket is created, click on Properties (upper right corner) and click on Static Website Hosting to enable hosting of content.



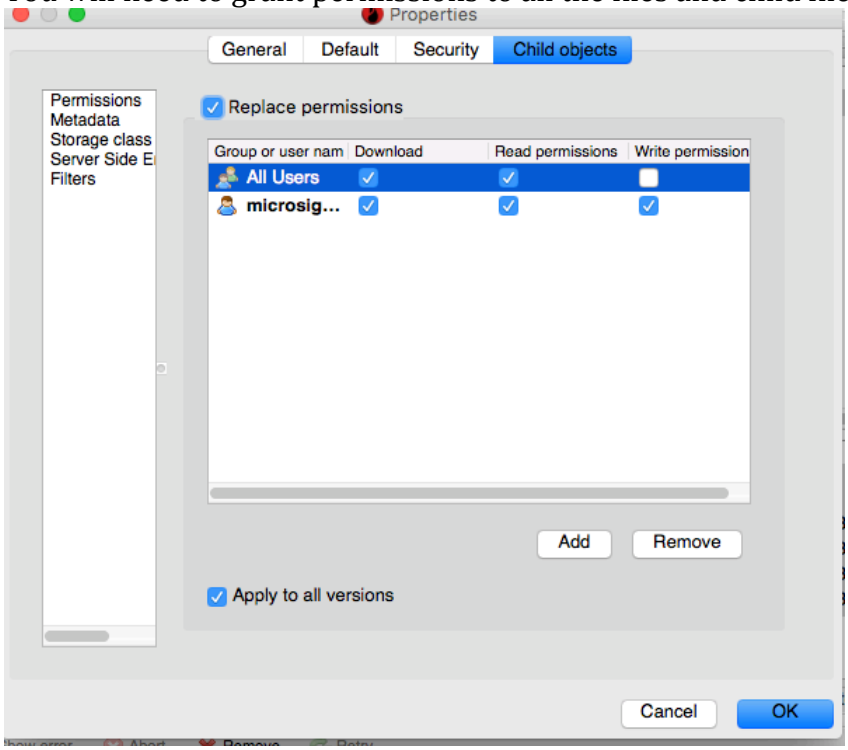
At this point your S3 bucket is ready to store content. You will a tool to transfer content from your local machine to your S3 bucket in the cloud. I use a tool called Dragon Disk (http://www.dragondisk.com/) but there are other tools.



Transfer the files from your local machine over to your S3 bucket. The files you will need to transfer will be under the following directory:

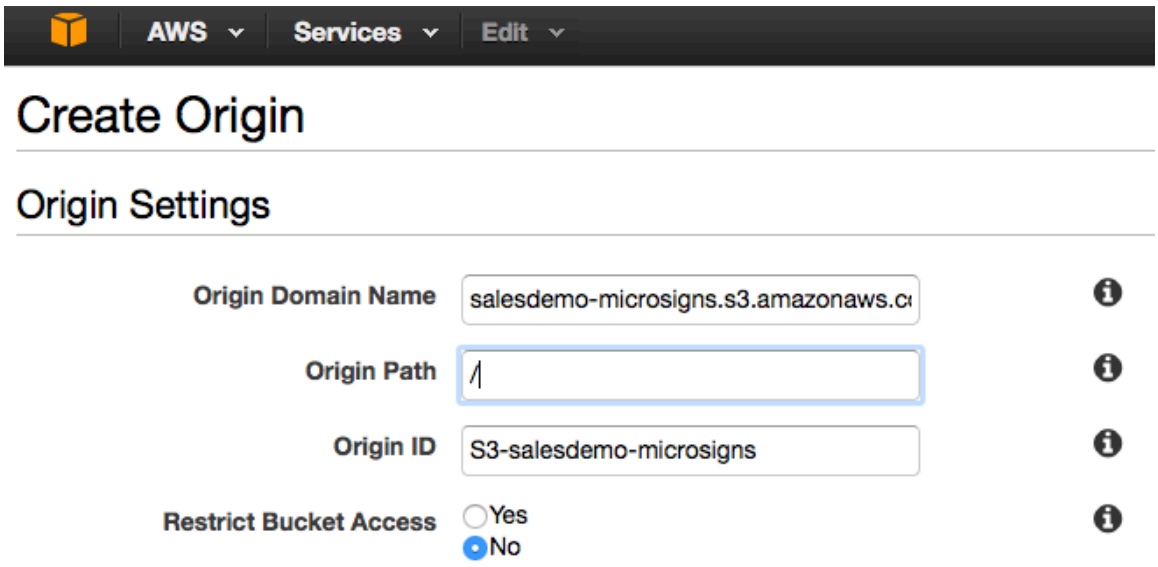
web/build/prod/

You will need to grant permissions to all the files and child files in the directory.

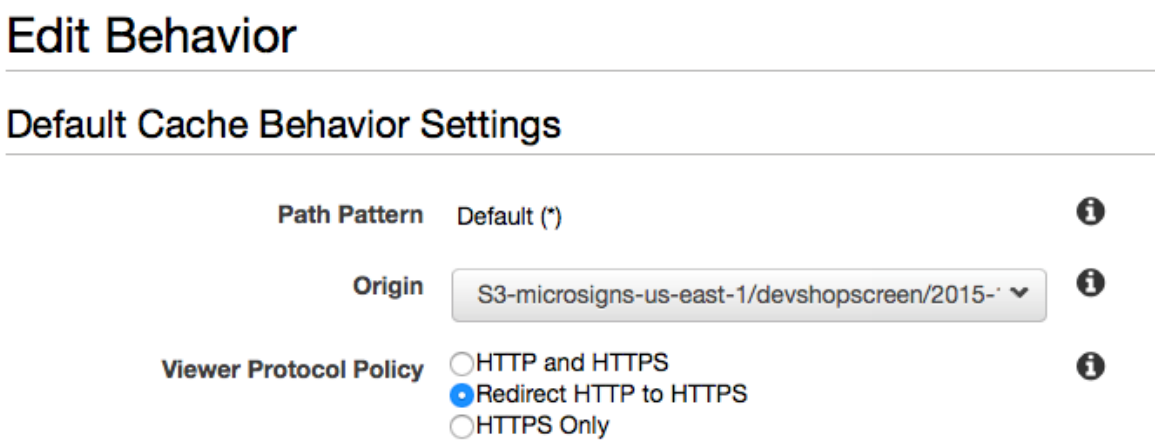


The final step now will be to map your CloudFront default directory to your content hosted on S3. You will also want to only direct api calls (i.e. /api/*) to CloudFront to your Beanstalk application.

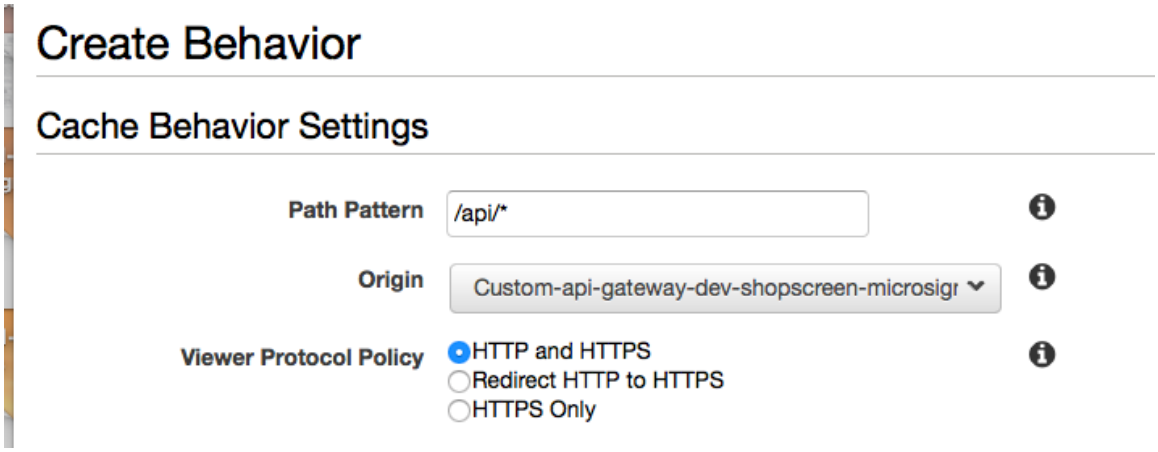
In the Amazon Console log back into Cloud Front and click on your distribution. Click on create on Create Origin.



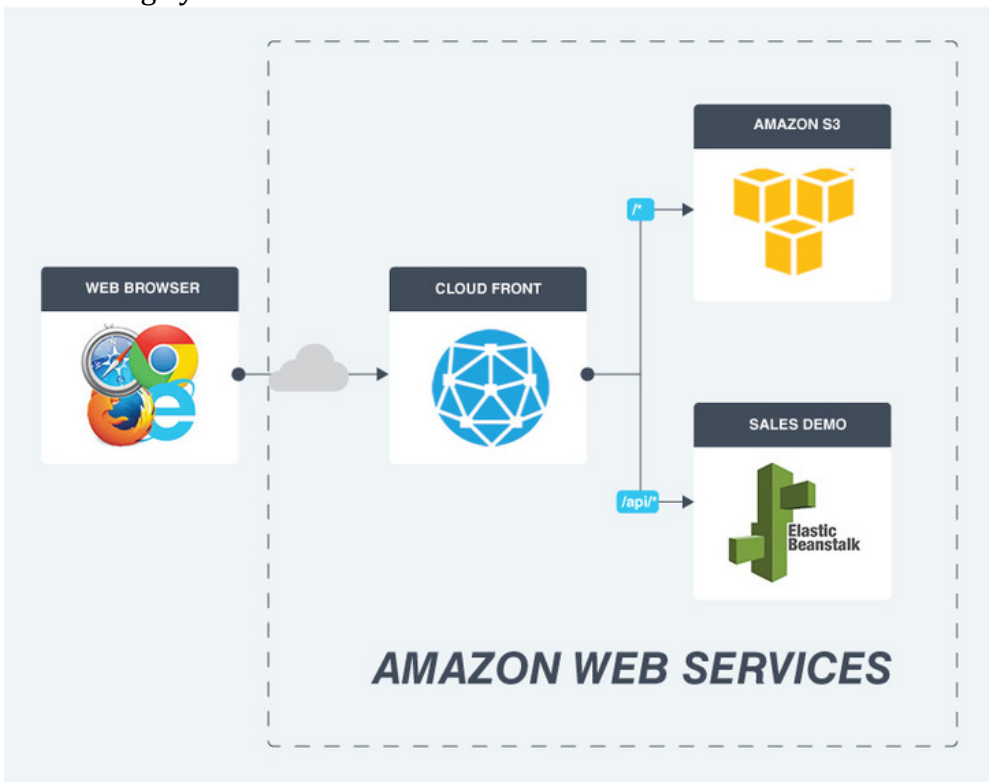
Click on the Behavior tab and click on the default behavior. Change the Origin so it now points to your newly created S3 bucket. Click on Yes, Edit.



On the behavior tab click on Create Behavior. We will create a new behavior so that all api requests (/api/*) get redirected to your BeanStalk Application



At this stage your CloudFront environment should look as follows:



Redeploy Application

Once CloudFront has been updated and your static content is hosted in S3. The only thing left to do is rebuild the sales demo application and redeploy it into Beanstalk. At this stage the all the static content has been moved to the web directory and the backend functionality is in the services directory. To rebuild your application run the maven command in services directory

mvn clean install

Log back into the Amazon Console and redeploy your Beanstalk application with the new WAR.

The above architecture is a good starting point for anyone who is looking at migrating their Spring application to a cloud based MicroServices. When you start looking about moving your application to a production system I would suggest you look at incorporating an API Gateway. There are a series of open source and commercially available API Gateways (Amazon released their API Gateway in July 2015, membrane-soa.org/, etc). The API Gateway will sit in between CloudFront and your backend services and will handle authentication, and it will redirect your requests to the appropriate Beanstalk instance. I have included a picture of the API Gateway below.