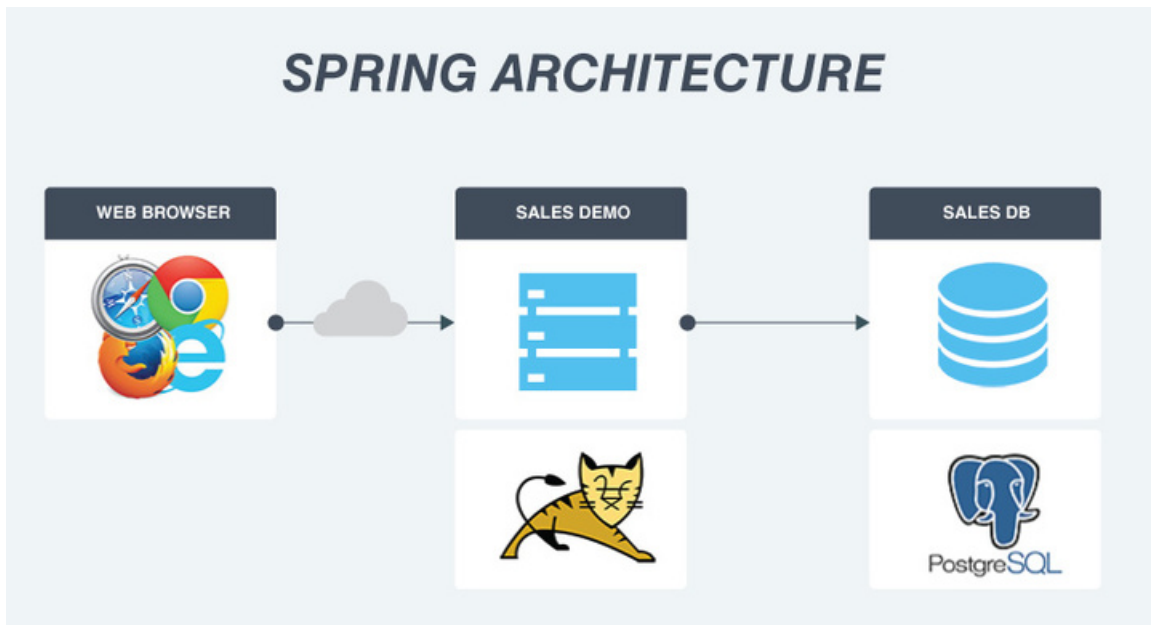


Migrating Spring App to MicroServices App on AWS

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 Application.

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 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 one 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 characteristics 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 landing 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:

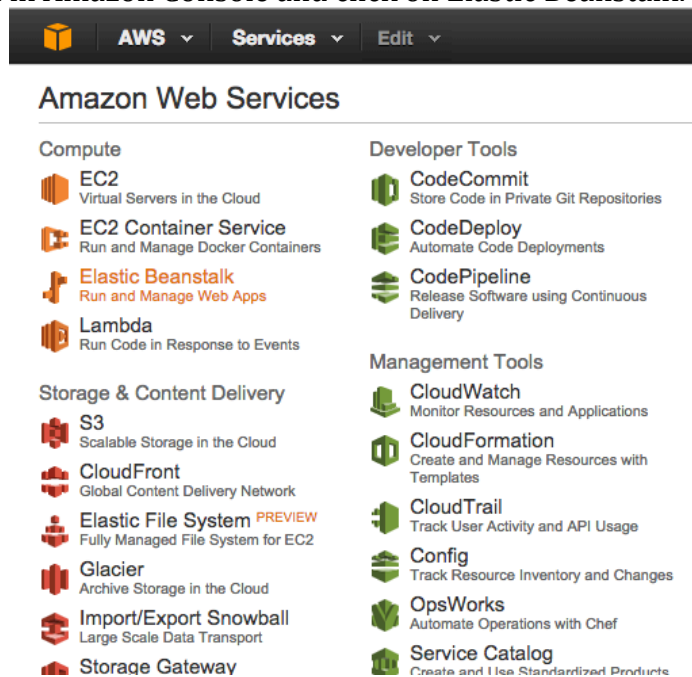


Figure 1 - Login to Amazon Console

- b. Select the platform as Tomcat

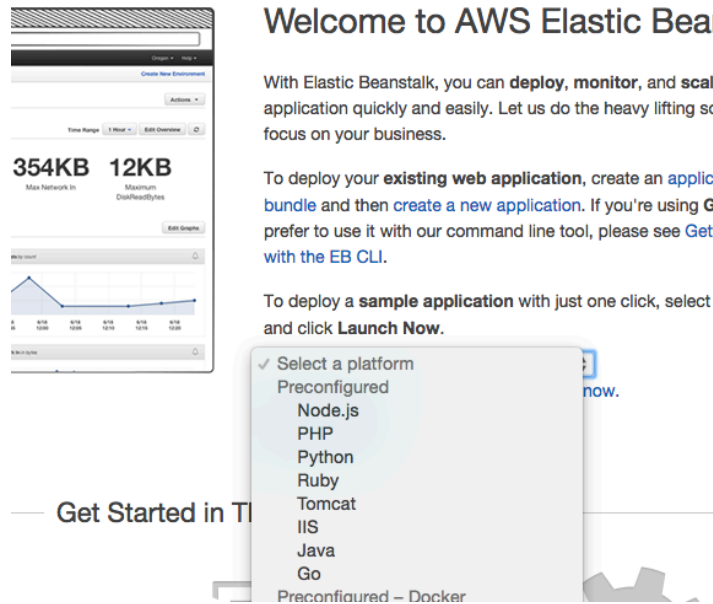


Figure 2 - Select Platform

c. Select Environment as Web Server

New Environment

AWS Elastic Beanstalk has two types of environment tiers to support different types of web applications. Web servers are standard applications that listen for and then process HTTP requests, typically over port 80. Workers are specialized applications that have a background processing task that listens for messages on an Amazon SQS queue. Worker applications post those messages to your application by using HTTP.

Web Server Environment

Provides resources for an AWS Elastic Beanstalk web server in either a single instance or load-balancing, auto scaling environment. [Learn more.](#)

Create web server

Worker Environment*

Provides resources for an AWS Elastic Beanstalk worker application in either a single instance or load-balancing, auto scaling environment. [Learn more.](#)

Create worker

* Worker environments require additional permissions to access other AWS services. [Learn more.](#)

Figure 3 - Select Environment

d. Select Environment Type as Single Instance

Environment Type

Choose the platform and type of environment to launch.

Predefined configuration: Tomcat Looking for a different platform? [Let us know.](#)

AWS Elastic Beanstalk will create an environment running Tomcat 8 Java 8 on 64bit Amazon Linux 2015.03 v2.0.1. [Change platform version.](#)

Environment type: Single instance [Learn more](#)

Cancel

Previous

Next

Figure 4 - Environment Type

e. Select Upload and Sales Demo WAR

Name	Date Modified	Size
Sample	Sep 18, 2015, 3:47 PM	--
SalesDemo	Oct 18, 2015, 7:34 PM	--
untitled folder	Oct 18, 2015, 7:34 PM	--
Servers	Oct 18, 2015, 7:13 PM	--
salesdemo-web	Oct 18, 2015, 11:22 PM	--
salesdemo	Today, 11:11 AM	--
target	Oct 18, 2015, 11:06 PM	--
test-classes	Yesterday, 10:09 AM	--
surefire-reports	Oct 18, 2015, 10:56 PM	--
salesdemo.war	Oct 18, 2015, 10:56 PM	17.2 MB
salesdemo	Oct 18, 2015, 10:56 PM	--

Figure 5 - Upload the WAR

f. Provide the Environment Name

Environment Information

Enter your environment information. [Learn more.](#)

Environment name: salesDemoTest

Environment URL: salesdemo-shannonlal.elasticbeanstalk.com Check availability

Description: A Sales Demo Test Optional: 200 character maximum

Cancel Previous Next

Figure 6 - Environment Name

g. Proceed to the next step

Additional Resources

Select additional resources for this environment.

☐ Create an RDS DB Instance with this environment [Learn more](#)

☐ Create this environment inside a VPC [Learn more](#)

Cancel Previous Next

h. Specify Instance Information

Modify the following settings or click Next to accept the default configuration. [Learn more.](#)

Instance type:
Determines the processing power of the servers in your environment.

EC2 key pair: [Refresh](#)
Optional: Enables remote login to your instances.

Email address:
Optional: Get notified about any major changes to your environment.

Health Reporting

System type:
Determines the health reporting type.

Root Volume (Boot Device)

Root volume type:
Determines the type of storage volume to attach to instances.

Root volume size: ☐ Enables you to specify the size of the root volume.
 GiB
Number of gibibytes of the root volume attached to each instance. Must be between 10 and 16385 for Provisioned IOPS (SSD) and General Purpose (SSD) root volumes and between 8 and 1024 for other root volumes.

Figure 7 - Select Instance Type

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

AWS Elastic Beanstalk is requesting permission to use resources in your account.

AWS Elastic Beanstalk needs to access your AWS resources on your behalf, clicking allow will create an IAM Role for AWS Elastic Beanstalk to use. See below for details.

▼ Hide Details

Role Summary ⓘ

Role	AWS Elastic Beanstalk needs permission to use resources in your account.
Description	account.
IAM Role	<input type="text" value="Create a new IAM Role"/>
Role Name	<input type="text" value="aws-elasticbeanstalk-service-ro"/>

► View Policy Document

Figure 8 - Create default Role

j. Launch the new Instance

Instance type	t1.micro
Key pair	shannonia@hotmail.com
Email address	Magnetic
Root volume type	(default)
Root volume size	(default)
Root volume IOPS	
Application health check URL	

[Environment Tags](#)

No settings provided.

[Permissions](#)

Service role	aws-elasticbeanstalk-service-role
Instance profile	aws-elasticbeanstalk-ec2-role

Cancel Previous **Launch**

Figure 9 - Launch Instance

k. View Elastic Beanstalk Instance

Info

Elastic Beanstalk is now creating your environment. When it has finished it will be running salesDemoTestVersion.

First Elastic Beanstalk Application ▶ salesDemoTest (salesdemo-shannonia1.elasticbeanstalk.com) **Actions**

Overview **Refresh**

Health

Ok

Causes

Running Version

salesDemoTestVersion

Upload and Deploy

Configuration

64bit Amazon Linux 2015.03
v2.0.1 running Tomcat 8 Java 8

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/swl/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

Amazon Web Services

Compute



-  **EC2**
Virtual Servers in the Cloud
-  **EC2 Container Service**
Run and Manage Docker Containers
-  **Elastic Beanstalk**
Run and Manage Web Apps
-  **Lambda**
Run Code in Response to Events

Storage & Content Delivery








-  **S3**
Scalable Storage in the Cloud
-  **CloudFront**
Global Content Delivery Network
-  **Elastic File System** **PREVIEW**
Fully Managed File System for EC2
-  **Glacier**
Archive Storage in the Cloud
-  **Import/Export Snowball**
Large Scale Data Transport
-  **Storage Gateway**
Integrates On-Premises IT Environments with Cloud Storage

Databases




Developer Tools


-  **CodeCommit**
Store Code in Private Git Repositories
-  **CodeDeploy**
Automate Code Deployments
-  **CodePipeline**
Release Software using Continuous Delivery

Management Tools

-  **CloudWatch**
Monitor Resources and Applications
-  **CloudFormation**
Create and Manage Resources with Templates
-  **CloudTrail**
Track User Activity and API Usage
-  **Config**
Track Resource Inventory and Changes
-  **OpsWorks**
Automate Operations with Chef
-  **Service Catalog**
Create and Use Standardized Products
-  **Trusted Advisor**
Optimize Performance and Security

2. Create a new Distribution

 **AWS**  **Services**  **Edit**

Distributions
What's New 

Reports & Analytics
Cache Statistics
Monitoring and Alarms
Popular Objects
Top Referrers
Usage
Viewers

Amazon CloudFront Getting Started

Either your search returned no results, or you do not have any distributions. (distribution. A distribution allows you to distribute content using a worldwide high data transfer speeds ([learn more](#)))

Create Distribution

3. Select Web Distribution

Step 1: Select delivery method
Step 2: Create distribution

Select a delivery method for your content.

Web

Create a web distribution if you want to:

- Speed up distribution of static and dynamic content, for example, .html, .css, .php, and g
- Distribute media files using HTTP or HTTPS.
- Add, update, or delete objects, and submit data from web forms.
- Use live streaming to stream an event in real time.

You store your files in an origin — either an Amazon S3 bucket or a web server. After you create can add more origins to the distribution.

Get Started

4. Select the default ELB

Create Distribution

Origin Settings

Origin Domain Name	<input type="text" value="awseb-e-m-AWSEBLoa-1FHW5170162t"/>	
Origin Path	<input type="text"/>	
Origin ID	<input type="text" value="ELB-awseb-e-m-AWSEBLoa-1FHW5170"/>	
Origin Protocol Policy	<input checked="" type="radio"/> HTTP Only <input type="radio"/> Match Viewer	
HTTP Port	<input type="text" value="80"/>	
HTTPS Port	<input type="text" value="443"/>	

5. Disable Caching

Default Cache Behavior Settings

Path Pattern	Default (*)	
Viewer Protocol Policy	<input checked="" type="radio"/> HTTP and HTTPS <input type="radio"/> Redirect HTTP to HTTPS <input type="radio"/> HTTPS Only	
Allowed HTTP Methods	<input checked="" type="radio"/> GET, HEAD <input type="radio"/> GET, HEAD, OPTIONS <input type="radio"/> GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE	
Cached HTTP Methods	GET, HEAD (Cached by default)	
Forward Headers	<input type="text" value="None (Improves Caching)"/>	
Object Caching	<input type="radio"/> Use Origin Cache Headers <input checked="" type="radio"/> Customize Learn More	
Minimum TTL	<input type="text" value="0"/>	
Maximum TTL	<input type="text" value="0"/>	
Default TTL	<input type="text" value="0"/>	

NOTE: This is recommended during the development stages

6. Accept the defaults

Object Caching ☐ Use Origin Cache Headers ☒ Customize [Learn More](#)

Minimum TTL

Maximum TTL

Default TTL

Forward Cookies

Forward Query Strings ☐ Yes ☒ No (Improves Caching)

Smooth Streaming ☐ Yes ☒ No

Restrict Viewer Access (Use Signed URLs or Signed Cookies) ☐ Yes ☒ No

Distribution Settings

Price Class

7. Check in status

CloudFront Distributions

<div> Create Distribution Distribution Settings Delete Enable Disable </div>									
<div> Viewing: Any Delivery Method Any Status </div>									
Viewing 1 to 1 of 1 Item									
	Delivery Method	ID	Domain N	Comm	Origi	CNAM	Statu	Statu	Last Mod
<input type="checkbox"/>	Web	E16JG5Q5Z6QIB1	d3gwx5o8	-	awse	-	In f	Enab	2015-10-4

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

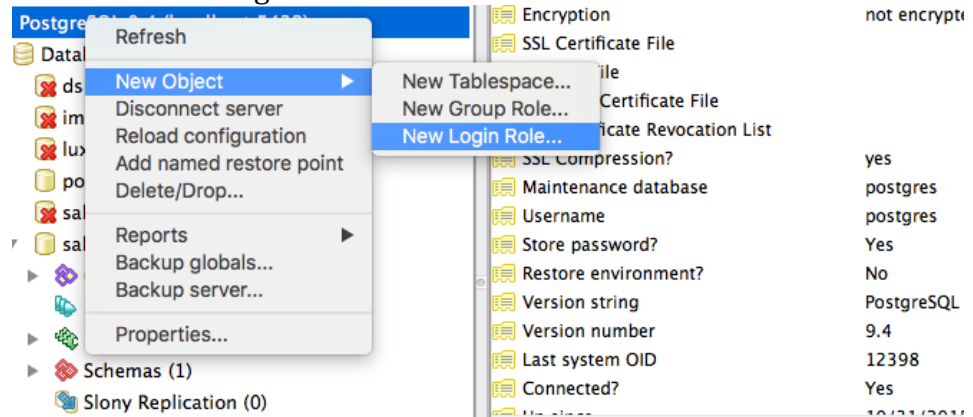
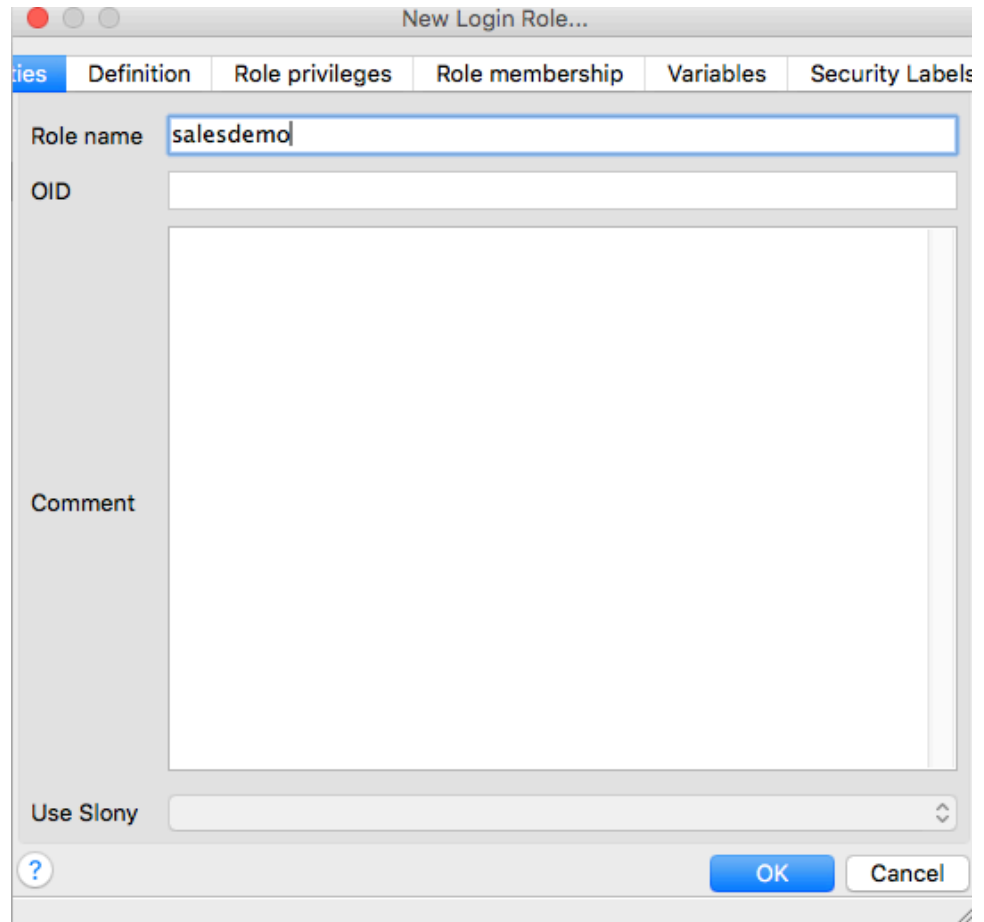


Figure 10 Create User Role



The image shows a 'New Login Role...' dialog box with a tabbed interface. The 'Definition' tab is selected. It contains a 'Role name' field with the text 'salesdemo', an 'OID' field, a large 'Comment' text area, and a 'Use Slony' checkbox. The dialog has 'OK' and 'Cancel' buttons at the bottom right.

ies	Definition	Role privileges	Role membership	Variables	Security Labels
Role name	salesdemo				
OID					
Comment					
Use Slony					

Figure 11 - Define User role

- b. Define a role and set the permissions

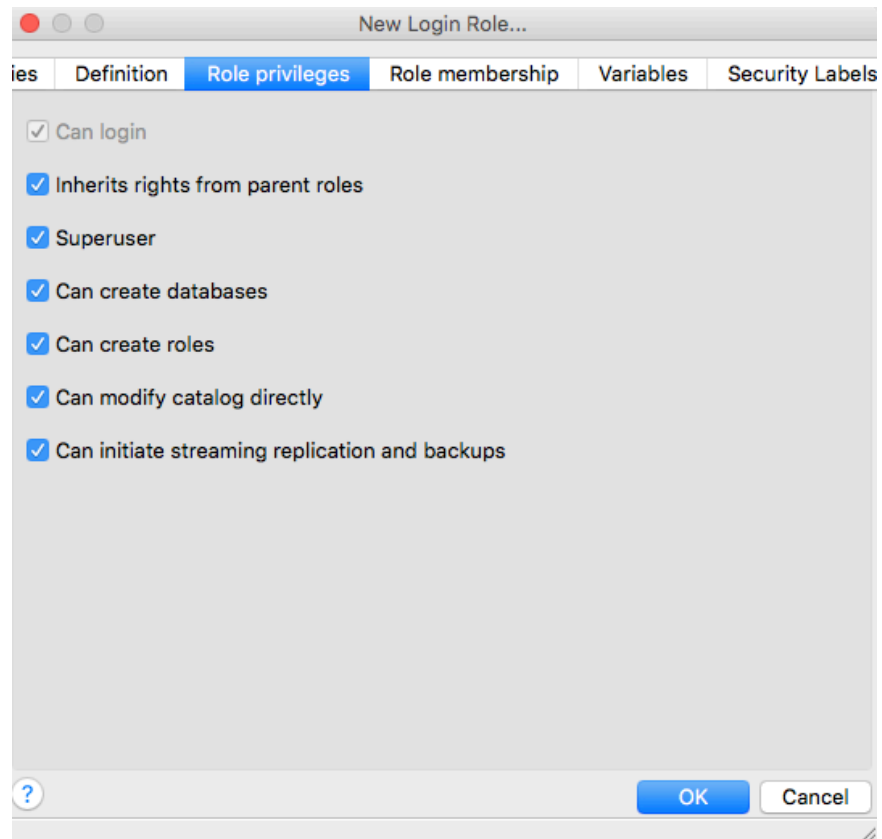
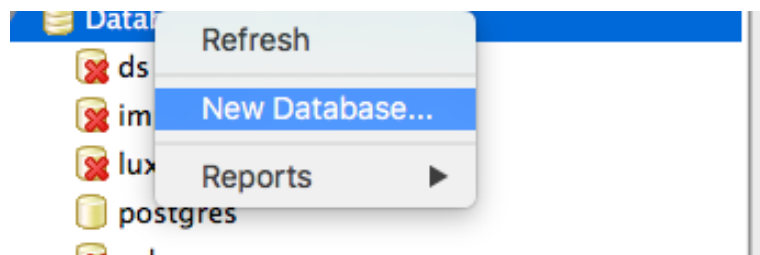


Figure 12 Specify the User Role

c. Create a new instance of the database



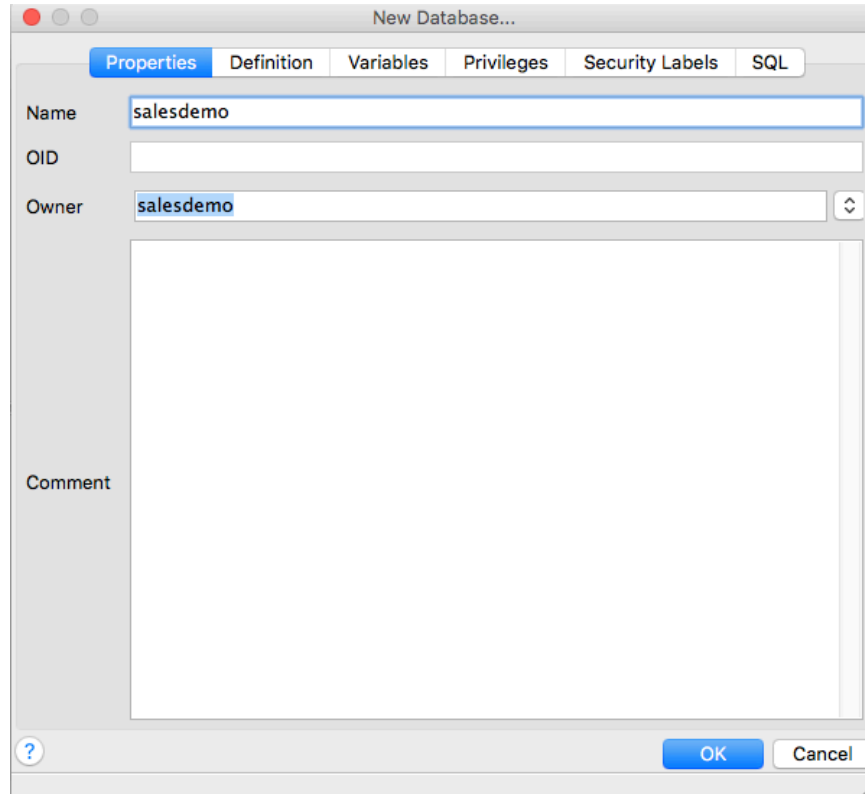


Figure 13 Create the DB Instance

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 embed 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 wish 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 steps 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

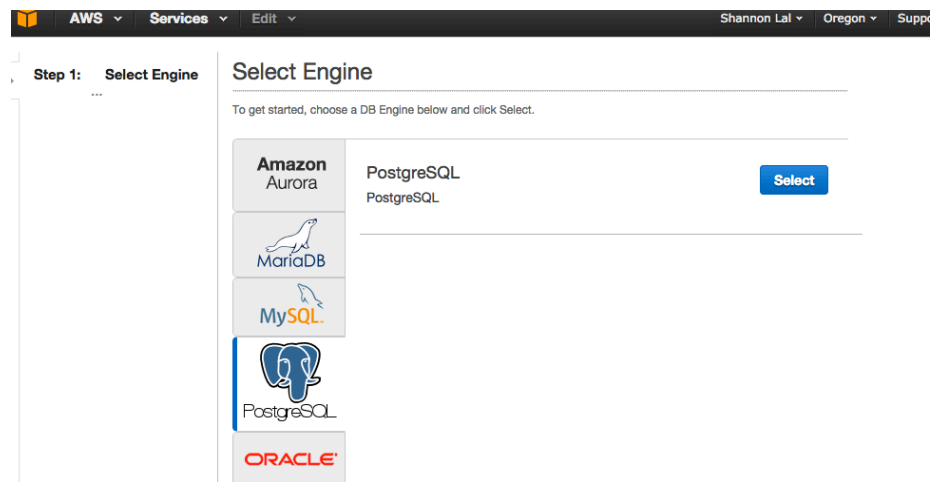


Figure 14 Create PostgreSQL RDS Instance

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

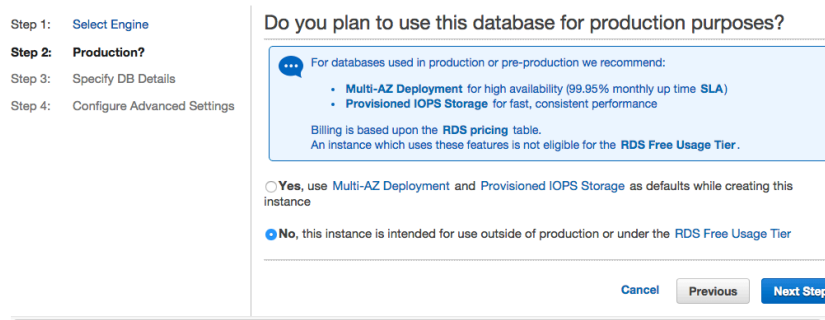


Figure 15 RDS Environment Selection

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

Specify DB Details
Configure Advanced Settings

Current selection is eligible for...
...re.

DB Engine postgres
License Model postgresql-license
DB Engine Version 9.4.4
DB Instance Class db.t2.micro — 1 vCPU, 1 GiB RAM
Multi-AZ Deployment No
Storage Type General Purpose (SSD)
Allocated Storage* 5 GB

Warning: Provisioning less than 100 GB of General Purpose (SSD) storage for high throughput workloads could result in higher latencies upon exhaustion of the initial General Purpose (SSD) IO credit balance. [Click here](#) for more details.

Settings

DB Instance Identifier* salesdemo
Master Username* admin
Master Password*
Confirm Password*

Launch DB

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

VPC* Default VPC (vpc-789f281d)

Subnet Group default

Publicly Accessible Yes

Availability Zone No Preference

VPC Security Group(s) Create new Security Group
awseb-e-6zvaikxmw-stack-AWSEBS
default (VPC)
rds-launch-wizard (VPC)

Database Options

Database Name salesdemo


Database Port 5432

DB Parameter Group default.postgres9.4

Option Group default:postgres-9-4

Copy Tags To Snapshots ☐

Enable Encryption No

 The selected Engine or DB Instance Class does not support storage encryption.

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

Launch DB Instance **Show Monitoring** **Instance Actions**

Filter: All Instances Search DB Instances... X

Viewing 1 of 1 DB Instances

Engine	DB Instance	Status	CPU	Current Activity	Maintenance	Class
PostgreSQL	salesdemo	available	1.15%	3 Connections	None	db.t2.micro

Endpoint: salesdemo.c36khawjgi13.us-west-2.rds.amazonaws.com:5432 (authorized)

Alarms and Recent Events

TIME (UTC-4)	EVENT
Oct 25 5:04 PM	DB instance created

Monitoring

	CURRENT VALUE	THRESHOLD	LAST HOUR	
CPU	1.08%			Read I
Memory	608 MB			Write I
Storage	4,760 MB			Swap U

Instance Actions **Tags** **Logs**

- 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

Overview



Health

Ok

Causes

Running Version

version

Upload and Deploy



Configuration

64bit Amazon Linux 2015.09
v2.0.4 running Tomcat 8 Java 8

Change

10. Click on Configuration and then Software Configuration

Dashboard
Configuration
Logs
Health NEW
Monitoring
Alarms
Events
Tags

Web Tier

Scaling

Environment type: Load balanced, auto scaling
Number instances: 1 - 4
Scale based on Average network out
Add instance when > 6000000
Remove instance when < 2000000

Software Configuration

Log publication: Off
Initial JVM heap size: 256m
JVM command line options: Blank
Maximum JVM heap size: 256m
Maximum JVM permanent generation size: 64m

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

Environment Properties

The following properties are passed into the application as environment variables. [Learn more.](#)

Property Name	Property Value
DB-Password	sales
DB-URL	jdbc:postgresql://salesdemo.c36
DB-User	salesdemo

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.

Create a Bucket - Select a Bucket Name and Region Cancel [X]

A bucket is a container for objects stored in Amazon S3. When creating a bucket, you can choose a Region to optimize for latency, minimize costs, or address regulatory requirements. For more information regarding bucket naming conventions, please visit the [Amazon S3 documentation](#).

Bucket Name:

Region:

Set Up Logging > Create Cancel

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

Q Search by prefix

NonePropertiesTransfers

Bucket: salesdemo-microsigns

Region: US Standard

Creation Date: Wed Dec 02 13:52:42 GMT-500 2015

Owner: microsigns-aws

Permissions

Static Website Hosting

You can [host your static website](#) entirely on Amazon S3. Once you enable your bucket for static website hosting, all your content is accessible to web browsers via the Amazon S3 website endpoint for your bucket.

Endpoint: salesdemo-microsigns.s3-website-us-east-1.amazonaws.com

Each bucket serves a website namespace (e.g. "www.example.com"). Requests for your host name (e.g. "example.com" or "www.example.com") can be routed to the contents in your bucket. You can also redirect requests to another host name (e.g. redirect "example.com" to "www.example.com"). See our [walkthrough](#) for how to set up an Amazon S3 static website with your host name.

☐ Do not enable website hosting

☒ Enable website hosting

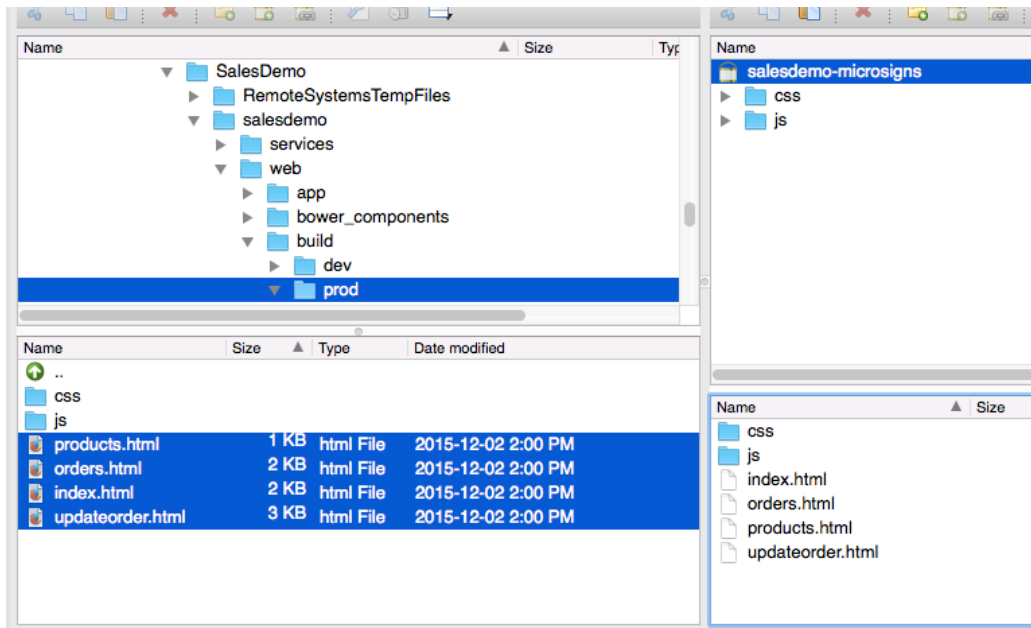
Index Document:

Error Document:

Edit Redirection Rules: You can set custom rules to automatically redirect web page requests for specific content.

☐ Redirect all requests to another host name

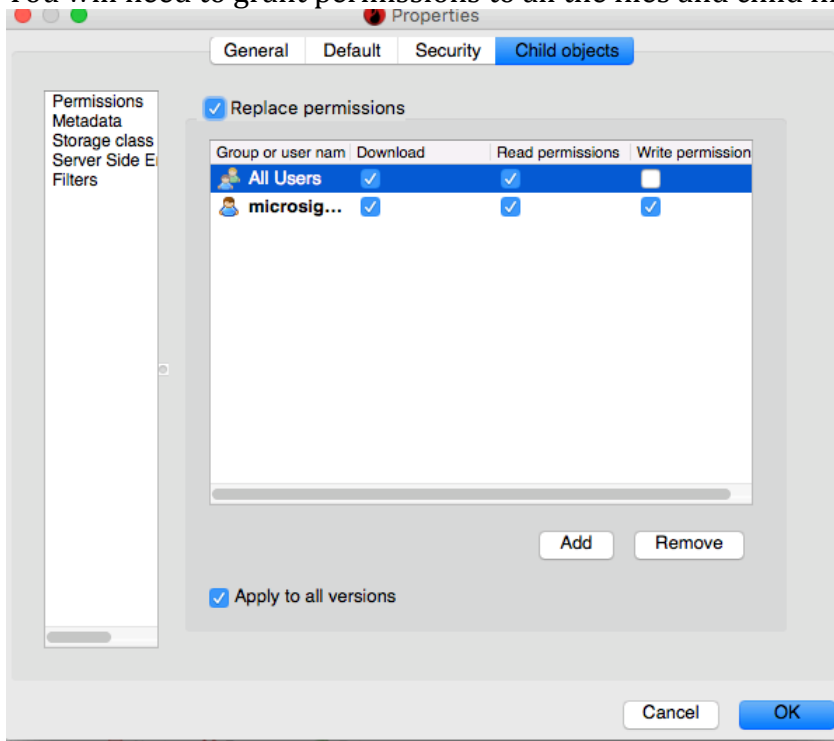
At this point your S3 bucket is ready to store content. You will a tool to transfer content from your local machine to your S3 bucket in the cloud. I use a tool called Dragon Disk (<http://www.dragondisk.com/>) but there are other tools.



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.



Create Origin

Origin Settings

Origin Domain Name salesdemo-microsigns.s3.amazonaws.com ⓘ

Origin Path / ⓘ

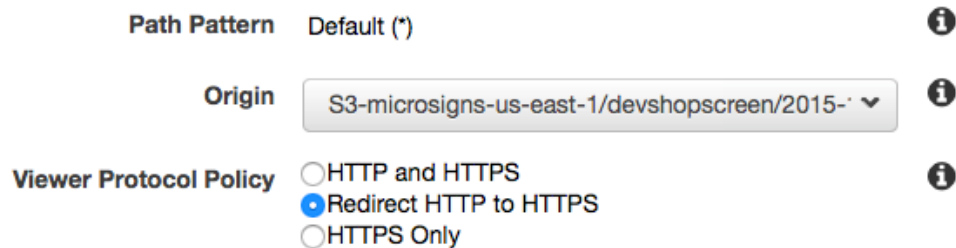
Origin ID S3-salesdemo-microsigns ⓘ

Restrict Bucket Access ☐ Yes ☒ No ⓘ

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.

Edit Behavior

Default Cache Behavior Settings



Path Pattern Default (*) ⓘ

Origin S3-microsigns-us-east-1/devshopscreen/2015- ⓘ

Viewer Protocol Policy ☐ HTTP and HTTPS ☒ Redirect HTTP to HTTPS ☐ HTTPS Only ⓘ

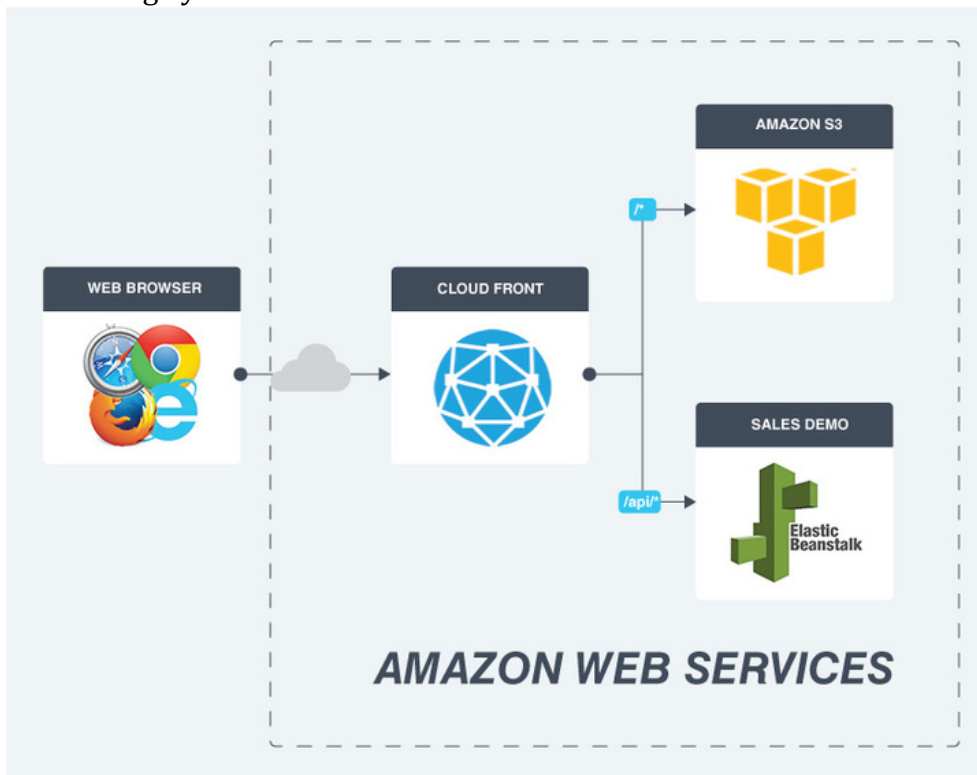
On the behavior tab click on Create Behavior. We will create a new behavior so that all api requests (/api/*) get redirected to your BeanStalk Application

Create Behavior

Cache Behavior Settings

Path Pattern	<input type="text" value="/api/*"/>	
Origin	<input type="text" value="Custom-api-gateway-dev-shopscreen-microsigr"/>	
Viewer Protocol Policy	<input checked="" type="radio"/> HTTP and HTTPS <input type="radio"/> Redirect HTTP to HTTPS <input type="radio"/> HTTPS Only	

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.

