

Deploying in Elastic Beanstalk

Another option that Amazon AWS provides to deploy the sample application is [Elastic Beanstalk](#). Beanstalk allows you to deploy a [WAR](#) file in a [Tomcat](#) (versions 6 or 7) environment. The system automatically manages load balancing and scalability of the application, while giving you full access to all Amazon AWS components.

We have to give a warning about the integration between Beanstalk and Play Framework. While Beanstalk has many benefits and it works fine with a Play Framework application, it is not the optimal platform to execute Play applications. By default, Play expects to be run using its embedded [Netty](#) server. Using a WAR file to run the application inside a JEE container reduces the performance and, in some Servlet engines, you lose access to some of the Play API that these engines don't support, like Websockets.

That said, it is possible to deploy Play in Beanstalk, and in some scenarios it is the best option. If you don't use certain API and you want Amazon AWS to configure your EC2 instances and the scalability of your application, please use Beanstalk; it works and you can focus on the application, not on configuring AWS. But you should be aware of the trade off you are making.

The following steps assume that both DynamoDB and RDS MySQL are properly set up in Amazon AWS. Check the previous chapter for more information on the steps to follow to configure them.

Generating the WAR file

Play Framework doesn't provide a native tool to generate WAR files from the application source. Thankfully, the community has taken to the challenge and [Damien Lecan](#) has created a plugin to that end, [Play2War](#). This section describes the steps to integrate the plugin with our existing code base and how to generate the WAR file from it

Adding Play2War to source

[Integrating](#) Play2War with an existing Play application is very straightforward. Play2War runs as a SBT plugin, so first of all we need to edit *project/plugins.sbt* and add the following line:

```
addSbtPlugin("com.github.play2war" % "play2-war-plugin" % "1.0")
```

This line allows calling Play2War commands via SBT, both by Play and via the terminal.

The second step is to edit *project/Build.scala* and add the Play2War configuration to the project, so Play Framework knows how to generate the War file when asked to. The resulting configuration looks like the following:

```
1 import com.github.play2war.plugin._
2
3 object ApplicationBuild extends Build {
4
5   val appName          = "play-sample-app"
6   val appVersion       = "1.0-SNAPSHOT"
7
8   val appDependencies = Seq(
9     javaCore,
10    javaJdbc,
11    javaEbean,
12    "org.webjars" %% "webjars-play" % "2.1.0-2",
```

```

13     "org.webjars" % "bootstrap" % "2.3.2",
14     "com.amazonaws" % "aws-java-sdk" % "1.4.5",
15     "com.feth" %% "play-authenticate" % "0.2.5-SNAPSHOT",
16     "be.objectify" %% "deadbolt-java" % "2.1-SNAPSHOT",
17     "mysql" % "mysql-connector-java" % "5.1.25"
18 )
19
20 val main = play.Project(appName, appVersion, appDependencies)
21   .settings(Play2WarPlugin.play2WarSettings: _*)
22   .settings(
23     resolvers += Resolver.url("Objectify Play Repository (release)", url("http://scha
24     resolvers += Resolver.url("Objectify Play Repository (snapshot)", url("http://scha
25     resolvers += Resolver.url("play-easymail (release)", url("http://joscha.github.com
26     resolvers += Resolver.url("play-easymail (snapshot)", url("http://joscha.github.co
27     resolvers += Resolver.url("play-authenticate (release)", url("http://joscha.github
28     resolvers += Resolver.url("play-authenticate (snapshot)", url("http://joscha.githu
29
30     Play2WarKeys.servletVersion := "3.0"
31   )
32
33 }

```

As you can see in line 1, we are importing the Play2War plugin which we integrate with our project in line 21. Line 30 sets the *Servlet* environment for which we are creating the War file. In this case we target *Servlets 3.0*, which has substantial improvements over *Servlets 2.5*.

Play2War provides some other [configuration](#) options that are not covered in this guide. Please check them to ensure full compatibility with the JEE container of your choice.

Configuration

One concern when creating a War file from the sample application is how to manage critical configuration values. This includes passwords and secret keys used to connect to SaaS services, which we don't want to commit to source control.

Fortunately, the default way that Play uses to manage these keys is still valid when creating a War file. As it was mentioned in the first chapter, any configuration property provided as JVM option via the *-D* flag overrides the corresponding entries in *application.conf*.

This means that we can store critical information in the JVM options string, instead of adding it to *conf* files where it could be committed to source control by mistake.

As it is explained later on, Beanstalk supports this and provides a simple way to add JVM options to a Java container.

Generating the WAR

To generate the War file, open a terminal window and go to the root folder of the sample application. In there, execute:

```
$play war
```

to execute the SBT plugin that builds the War file. The resulting War is stored at the *target* folder

that is created at the root of the application. In this example the file is at:

\$target/play-sample-app-1.0-SNAPSHOT.war

The output of executing Play2War is:

```
pvillega@pvillega-Inspiron-530: ~/Dropbox/Projectes/amazon-article/play-sample-app
pvillega@pvillega-Inspiron-530:~/Dropbox/Projectes/amazon-article/play-sample-app$ play war
[info] Loading global plugins from /home/pvillega/.sbt/plugins
[info] Loading project definition from /home/pvillega/Dropbox/Projectes/amazon-article/play-sample-app/project
[info] Set current project to play-sample-app (in build file:/home/pvillega/Dropbox/Projectes/amazon-article/play-sample-app/)
[info] Packaging /home/pvillega/Dropbox/Projectes/amazon-article/play-sample-app/target/scala-2.10/play-sample-app_2.10-1.0-SNAPSHOT-sources.jar ...
[info] Updating {file:/home/pvillega/Dropbox/Projectes/amazon-article/play-sample-app/)play-sample-app...
[info] Done packaging.
[info] downloading http://repo1.maven.org/maven2/com/github/play2war/play2-war-core-servlet30_2.10/1.0/play2-war-core-servlet30_2.10-1.0.jar ...
[info] [SUCCESSFUL ] com.github.play2war#play2-war-core-servlet30_2.10;1.0!play2-war-core-servlet30_2.10.jar (2775ms)
[info] downloading http://repo1.maven.org/maven2/com/github/play2war/play2-war-core-common_2.10/1.0/play2-war-core-common_2.10-1.0.jar ...
[info] [SUCCESSFUL ] com.github.play2war#play2-war-core-common_2.10;1.0!play2-war-core-common_2.10.jar (625ms)
[info] Done updating.
[info] Wrote /home/pvillega/Dropbox/Projectes/amazon-article/play-sample-app/target/scala-2.10/play-sample-app_2.10-1.0-SNAPSHOT.pom
[info] Generating Scala API documentation for main sources to /home/pvillega/Dropbox/Projectes/amazon-article/play-sample-app/target/scala-2.10/api...
[info] Packaging /home/pvillega/Dropbox/Projectes/amazon-article/play-sample-app/target/scala-2.10/play-sample-app_2.10-1.0-SNAPSHOT.jar ...
[info] Done packaging.
model contains 97 documentable templates
[info] Scala API documentation generation successful.
[info] Packaging /home/pvillega/Dropbox/Projectes/amazon-article/play-sample-app/target/scala-2.10/play-sample-app_2.10-1.0-SNAPSHOT-javadoc.jar ...
[info] Done packaging.
[info] Build WAR package for servlet container: 3.0
[info] Packaging /home/pvillega/Dropbox/Projectes/amazon-article/play-sample-app/target/play-sample-app-1.0-SNAPSHOT.war ...
[info] Packaging done.
[success] Total time: 272 s, completed 30-Jun-2013 15:22:17
pvillega@pvillega-Inspiron-530:~/Dropbox/Projectes/amazon-article/play-sample-app$
```

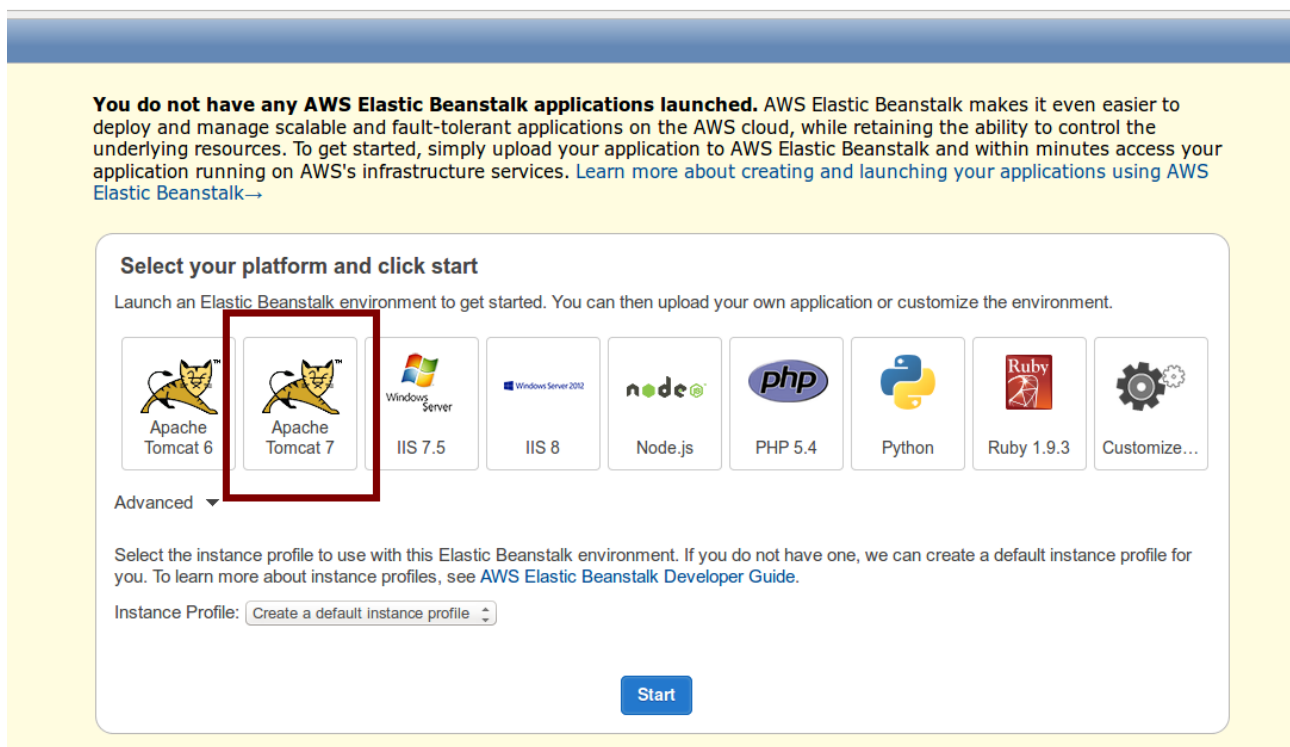
As you can see, the Play2War plugin generates a set of jars from the source code of the application and stores them inside the War file. All the configuration required to run the file inside a container is added as per the Servlet version we selected in *Build.scala*.

At this stage we have created a fully functional War file, ready to be deployed in Beanstalk.


Creating the Beanstalk instance

As we want to deploy the War file in Elastic Beanstalk, first we have to create an instance via the [Elastic Beanstalk Console](#). An instance consists on a servlet container in which we deploy the War file. As mentioned before, Amazon manages many aspects of the container, simplifying the maintenance of the application.

When accessing the console, the first screen allows us to select a container for the application:



We select *Tomcat 7* as our target container and press *Start*. Beanstalk starts creating the environment, which may take several minutes to be ready.

 Services ▾ Edit ▾

My First Elastic Beanstalk Application

Elastic Beanstalk Application Details

Overview

Events

Versions

Application Description:

This is the sample application provided by Amazon Web Services for demonstrating AWS Elastic Beanstalk.

Created on:

2013-06-30 15:35 GMT+0100

Edit Application Description

Delete This Application

My First Elastic Beanstalk Application Environments

Default-Environment

Successfully running version Sample Application.

Environment Details

Overview

Logs

Monitoring

Events

URL:

<http://Default-Environment-5kb8hjacs9.elasticbeanstalk.com>

Running Version:

Sample Application

Container Type:

64bit Amazon Linux running Tomcat 7

Changed on:

2013-06-30 15:41 GMT+0100

Deploy a Different Version

Edit Configuration

As you can see in the *Events* view, Beanstalk is creating all the necessary components for the application, including *auto-scaling* and *CloudWatch Alarms*, so you don't have to worry about managing them:

My First Elastic Beanstalk Application

Elastic Beanstalk Application Details

Overview

Events

Versions

Below are the most recent 1000 events for this application. Click [here](#) to learn how you can retrieve all events.

Viewing:

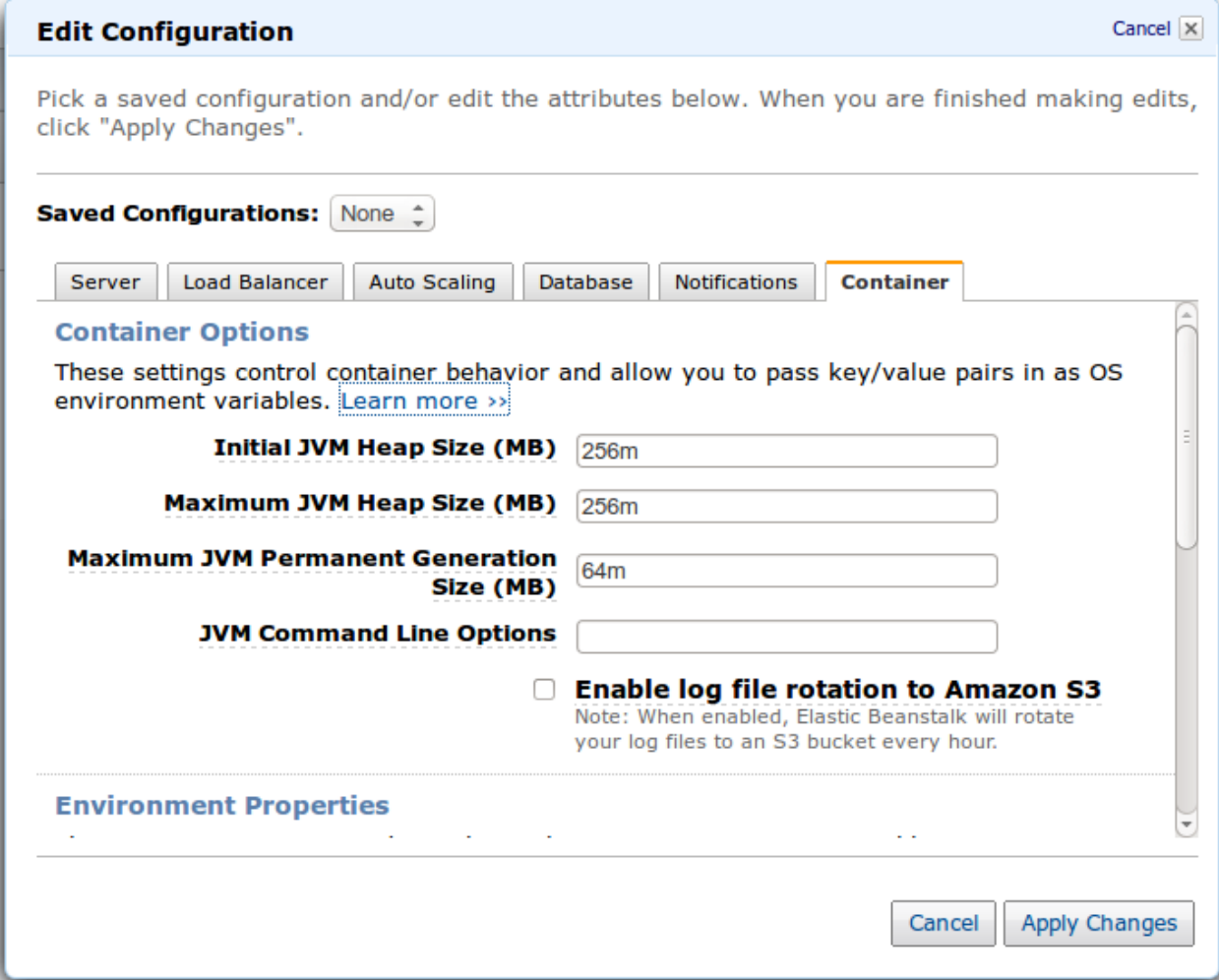
All Event Types

Event Time	Event Type	Event Details	Environment
2013-06-30 15:37 GMT+0100	INFO	Created CloudWatch alarm named: awseb-e-37si3kwu6s-stack-AWSEBCloudwatchAlarmHigh-163IED4ZNDQA2	Default-Environment
2013-06-30 15:37 GMT+0100	INFO	Created CloudWatch alarm named: awseb-e-37si3kwu6s-stack-AWSEBCloudwatchAlarmLow-JOCQXHODWLVN	Default-Environment
2013-06-30 15:37 GMT+0100	INFO	Created Auto Scaling group policy named: arn:aws:autoscaling:eu-west-1:497420748264:scalingPolicy:51c9f7c1-8a1e-4321-85a3-5edc41513850:autoScalingGroupName/awseb-e-37si3kwu6s-stack-AWSEBAutoScalingGroup-1Y2OC56F2D7KY:policyName/awseb-e-37si3kwu6s-stack-AWSEBAutoScalingScaleDownPolicy-28NQ7I57HJSX	Default-Environment
2013-06-30 15:37 GMT+0100	INFO	Created Auto Scaling group policy named: arn:aws:autoscaling:eu-west-1:497420748264:scalingPolicy:6e03211c-22ec-4c5f-a7f0-8fbbbf938c6e:autoScalingGroupName/awseb-e-37si3kwu6s-stack-AWSEBAutoScalingGroup-1Y2OC56F2D7KY:policyName/awseb-e-37si3kwu6s-stack-AWSEBAutoScalingScaleUpPolicy-XYLERLUHAVP9	Default-Environment
2013-06-30 15:37 GMT+0100	INFO	Waiting for EC2 instances to launch. This may take a few minutes.	Default-Environment
2013-06-30 15:37 GMT+0100	INFO	Created Auto Scaling group named: awseb-e-37si3kwu6s-stack-AWSEBAutoScalingGroup-1Y2OC56F2D7KY	Default-Environment
2013-06-30 15:36 GMT+0100	INFO	Created Auto Scaling launch configuration named: awseb-e-37si3kwu6s-stack-AWSEBAutoScalingLaunchConfiguration-14NG28XL7OZQW	Default-Environment
2013-06-30 15:36 GMT+0100	INFO	Created security group named: awseb-e-37si3kwu6s-stack-AWSEBSecurityGroup-1T07IAIO6FD2S	Default-Environment
2013-06-30 15:36 GMT+0100	INFO	Created load balancer named: awseb-e-3-AWSEBLoa-11EU5ELBZH5EY	Default-Environment
2013-06-30 15:35 GMT+0100	INFO	Using elasticbeanstalk-eu-west-1-497420748264 as Amazon S3 storage bucket for environment data.	Default-Environment

Once the initial deployment is ready, we can start configuring our application.

In the Overview tab, the *Edit Configuration* link gives access to several settings for the application. These include the *Container Settings* area, where we can modify the heap of the application and,

more importantly, provide JVM options:



Edit Configuration Cancel

Pick a saved configuration and/or edit the attributes below. When you are finished making edits, click "Apply Changes".

Saved Configurations: None

Server **Load Balancer** **Auto Scaling** **Database** **Notifications** **Container**

Container Options

These settings control container behavior and allow you to pass key/value pairs in as OS environment variables. [Learn more >>](#)

Initial JVM Heap Size (MB)

Maximum JVM Heap Size (MB)

Maximum JVM Permanent Generation Size (MB)

JVM Command Line Options

☐ **Enable log file rotation to Amazon S3**
Note: When enabled, Elastic Beanstalk will rotate your log files to an S3 bucket every hour.

Environment Properties

Cancel Apply Changes

We can use the JVM options to provide configuration settings to the application, like the JDBC url to our RDS instance. Remember that in a Play application, options provided via *-D* flags in JVM override settings in *application.conf*. For example, we can provide the following string:

```
-Ddb.default.url="<jdbc url>" -Ddb.default.user=user -Ddb.default.password=password
```

to tell the application to use the RDS instance created in the previous chapter.

After applying the changes, the application will restart and we are ready to deploy the War file in Tomcat.

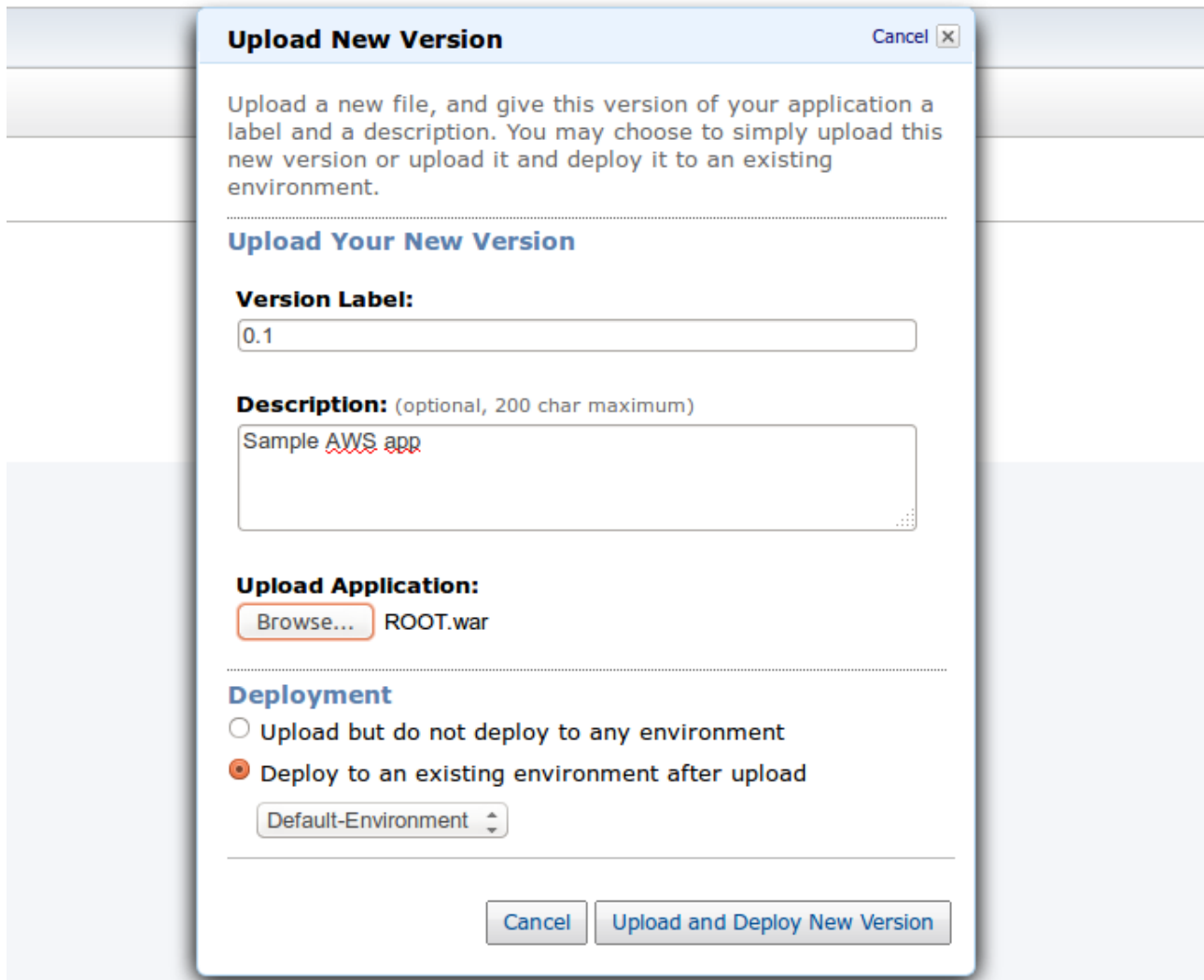
Deploying to Beanstalk

Deploying to Beanstalk is a simple process. Beanstalk uses the concept of *Versions* to manage the application. Each new version is a new War file uploaded to the server. When deploying, we can add comments to facilitate tracking of changes to the application. At any moment we can download the War file associated to a version, to do local testing, and we can revert to previous versions if needed.

When we created the Beanstalk environment, we selected Tomcat 7 as the container. To facilitate deployment to Tomcat 7, first you need to rename the War generated by Play2War to *ROOT.war*. This will turn our sample application into the *root* application of the container.

Once ready, select *Upload new version* in the Elastic Beanstalk dashboard. It opens a pop up

window where all the details of the version can be added, before deployment:



The screenshot shows a dialog box titled "Upload New Version" with a "Cancel" button in the top right corner. The dialog contains the following sections:

- Upload Your New Version**
 - Version Label:** A text input field containing "0.1".
 - Description:** (optional, 200 char maximum). A text area containing "Sample AWS app".
 - Upload Application:** A "Browse..." button followed by the text "ROOT.war".
- Deployment**
 - Two radio buttons: "Upload but do not deploy to any environment" (unselected) and "Deploy to an existing environment after upload" (selected).
 - A dropdown menu showing "Default-Environment".

At the bottom of the dialog are two buttons: "Cancel" and "Upload and Deploy New Version".

After pressing *Upload and Deploy* the War file is uploaded and the server is restarted to deploy the new release. This process takes a few minutes, as the War file being uploaded can be around 60Mb and the server needs to deploy the War and restart after the upload finishes.

If everything works as expected, a new version of the application is shown in the dashboard. Otherwise, check the logs to find the reason why the application is not deploying properly.

A common error when deploying to Beanstalk is to have a mismatch between JDK versions in the development environment and the container. Beanstalk uses JDK 1.6. If you create your War file using a newer version, like JDK 1.7, application deployment fails due to class version issues. Be aware of this limitation when generating the War file.

Testing the application

At this stage we have the application deployed and running. If we access the url provided by Beanstalk, which in this example is <http://default-environment-5kb8hjacs9.elasticbeanstalk.com/>, we can see the homepage:

AWS Sample Application

Sample Play application that shows integration with AWS services

Sign up »

The application allows logged in users to browse trending photos from Flickr. The trending tags are retrieved asynchronously by a background job and stored in DynamoDB. On user request, stored photos are displayed. User details are stored using MySQL in RDS.

Extensible

This sample application contains a basic logic that can be easily extended to fulfill your needs

Secured

User authentication and authorization included thanks to Play Authenticate and Deadbolt 2, state of the art modules for the Java version of Play Framework

Integrated

The application connects with 3rd party services, a standard requirement of any modern application

© 2012-2013 [@pvillega](#). Licensed under Apache License, Version 2.0.

Authentication system based on [Play Authenticate](#) and its sample application. Styles by [Twitter Bootstrap](#) · Provider icons by [Paul Robert Lloyd](#)

Beanstalk has full access to all AWS services. This means that you can extend your application as required, usually with minimal effort as AWS provides convenient interfaces between Beanstalk and other AWS components.

For example, to modify the generated domain linked to your application (which is not very user friendly) and use your own custom domain you can follow the steps described in the [documentation](#), which use [Amazon Route 53](#) to provide this mapping.

Next steps

This concludes the article about deploying a Play Framework application in Amazon AWS. Through it we have seen how to create a Play Framework application that uses Amazon AWS services, how to deploy it in EC2 using Ansible and how to run a War version of this application in Elastic Beanstalk.

The application is a good starting point from which to build your own application. Feel free to clone and fork the [Github project](#) and develop your own. You will see that by using these three tools (Amazon AWS, Ansible and Play Framework) your productivity increases and you are able to create amazing websites with minimal effort.

If you have any questions, feel free to contact me at [@pvillega](#).

Cheers!