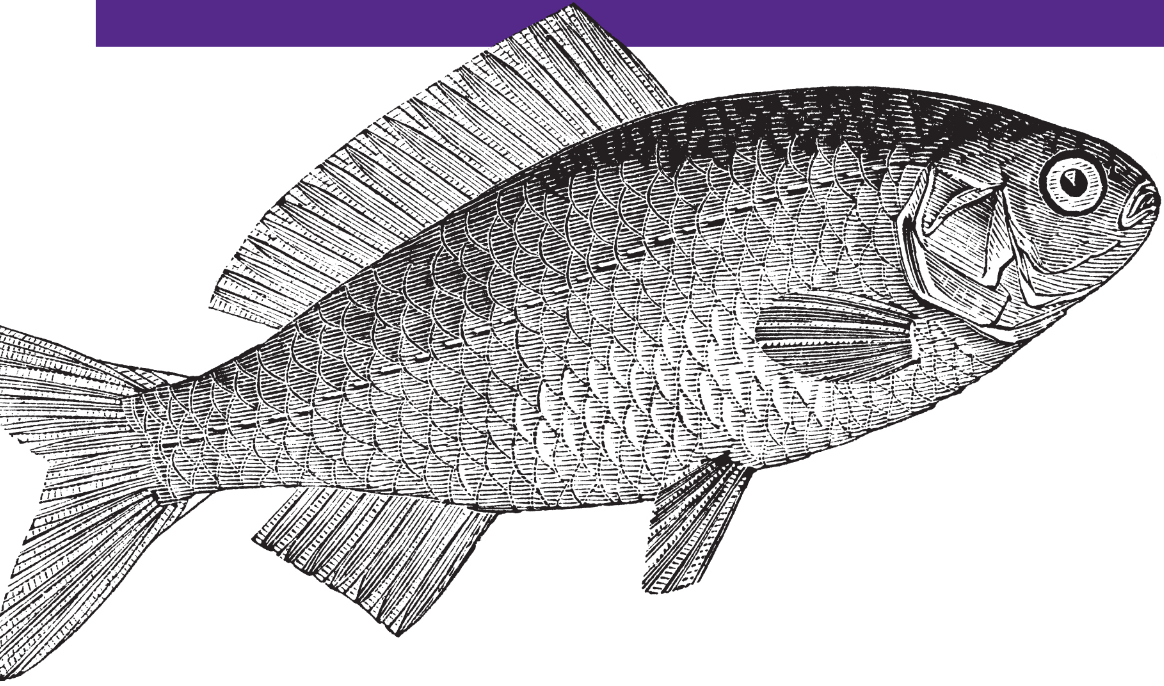
*Simple Cloud Scaling for Java Developers*

# Elastic Beanstalk



*Jurg van Vliet, Flavia Paganelli, Steven van Wel & Dara Dowd*

O'REILLY®

# Elastic Beanstalk

While it's always been possible to run Java applications on Amazon EC2, Amazon's Elastic Beanstalk makes the process easier—especially if you understand how it works beneath the surface. This concise, hands-on book not only walks you through Beanstalk for deploying and managing web applications in the cloud, but also shows you how to use this AWS tool in other phases of development.

Ideal if you're a developer familiar with Java applications or AWS, *Elastic Beanstalk* provides step-by-step instructions and numerous code samples for building cloud applications on Beanstalk that can handle lots of traffic. Learn how to use Beanstalk with the Eclipse IDE, Hudson for continuous integration, and several AWS tools for load balancing, auto scaling, storage, and other services.

- Understand how Beanstalk provides an entry into Infrastructure as a Service (IaaS)
- Design your Java web application for the cloud—and for Beanstalk
- Get an overview of AWS services that power Beanstalk, and learn how to use them independently
- Use Beanstalk to set up your development, testing, production, and staging environments
- Learn advanced hacking techniques for customizing Beanstalk

**US $29.99** **CAN $34.99**
ISBN: 978-1-449-30664-9

# Elastic Beanstalk

# Elastic Beanstalk

*Jurg van Vliet, Flavia Paganelli, Steven van Wel, and Dara Dowd*

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

**Elastic Beanstalk**
by Jurg van Vliet, Flavia Paganelli, Steven van Wel, and Dara Dowd

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

**Editors:** Mike Loukides and Julie Steele
**Production Editor:** Teresa Elsey
**Cover Designer:** Karen Montgomery
**Interior Designer:** David Futato
**Illustrator:** Robert Romano

**Printing History:**
July 2011: First Edition.

ISBN: 978-1-449-30664-9

[LSI]

1311259188

# Table of Contents

# Preface

Thank you for picking up a copy of this book. Amazon Elastic Beanstalk is one of Amazon AWS's services. It offers a platform for easy deployment of web applications. The first version of Elastic Beanstalk handles Java applications running in a Tomcat container. Deploying an application has been made as easy as uploading your WAR to your Application Environment.

Elastic Beanstalk is difficult, and barely understood. But it has been a huge hit with the media following cloud trends. We have seen headlines shouting that Amazon AWS was "in the PaaS business," taking on Heroku and Google App Engine. These comparisons are not so interesting, except that they show that expectations are high. There is the idea that the cloud will end all problems, including building and especially deploying applications to large-scale infrastructures.

There is a huge gap between developing web applications in Java and running them on AWS infrastructures that can handle huge traffic. This gap contains things like installing Linux, configuring Tomcat, etc. But it also includes many AWS services, like EC2, Auto Scaling, Elastic Load Balancing, and S3. Elastic Beanstalk tries to hide these details, but it allows you to take over at any level, whenever you require. In a way, it tries to provide an "easy entrance" to AWS. So, the task at hand is to explain something that has been intentionally left out, because it is often a source of frustration.

We very recently finished our first book, *Programming Amazon EC2*. Just before the deadline for that book, Elastic Beanstalk was introduced. We wrote about it briefly, without getting into much detail. But Elastic Beanstalk was the logical next topic to address. We also had plans to build a Scala application called heystaq then, and we decided to use Elastic Beanstalk to deploy it. That became our first real experience with Beanstalk.

The authors of the book have been working together in different ways. We were drawn together to build a prototype of heystaq. We participated in an AWS Hackathon in Amsterdam in April 2011 to create something cool.

heystaq is a tool to visualize AWS infrastructures. We set out to build it in Scala for several reasons: the two most important are scalability and availability of the AWS Java

SDK. We have enough Java experience, but Scala was new. And, of course, this project was to be built on Elastic Beanstalk.

## Audience

Elastic Beanstalk has intrigued us from the moment it was introduced. It is a service that automates many of the intricacies of building and running Java web applications. We set out to show how you can use Elastic Beanstalk. In the process we had to deal with many Java-related tools like Eclipse and Hudson, as well as introduce other Amazon AWS services like EC2 and Elastic Load Balancing.

It will definitely help to either have a good understanding of Amazon AWS or be intimate with building Java applications. If you are not familiar with either, you should at least be able to coerce Eclipse into building your Java app, or be intimate with building with a tool like maven. If all these terms mean nothing to you, you are probably looking for another book.

With this book we want to help you understand Elastic Beanstalk and show you how to use it in your work.

## Conventions Used in This Book

The following typographical conventions are used in this book:

*Italic*
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

**`Constant width bold`**
: Shows commands or other text that should be typed literally by the user.

*`Constant width italic`*
: Shows text that should be replaced with user-supplied values or by values determined by context.

This icon signifies a tip, suggestion, or general note.

This icon indicates a warning or caution.

## Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Elastic Beanstalk* by Jurg van Vliet and Flavia Paganelli (O'Reilly). Copyright 2011 I-MO BV, 978-1-449-30664-9."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

## Safari® Books Online

Safari Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at *http://my.safaribooksonline.com*.

## How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

*http://oreilly.com/catalog/0636920020561/*

To comment or ask technical questions about this book, send email to:

*bookquestions@oreilly.com*

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

## Acknowledgments

Many people played a part in this book. Some small, others bigger. We want to thank Rodica Buzescu and Werner Vogels for organizing the Hackathon during the Next Web conference in Amsterdam. Also the conversations with Matt Wood and Stephanie Cuthbertson helped in writing the underlying stories of this book.

While working with Elastic Beanstalk and writing the book, we were fortunate to have contact with the Amazon Elastic Beanstalk Product team. We were shown solutions to problems that were not released yet, helping us write a book that contains all features available at the moment of publication, even if only for a day. Saad Ladki, thank you for helping, and thank you for a great service.

Without technical reviewers, writing a book is impossible. They help catch errors, but more importantly are very frank when it doesn't make sense. Thanks Wilfred Springer, Eric Bowman, and Saad Ladki for reviewing this book.

Julie and Mike convinced us to write this book first, although there were many other topics we shared an interest in. Thank you for insisting on doing this first; we had a great time working with Beanstalk. And, of course, the rest of the O'Reilly team was instrumental in making this book a success!

CHAPTER 1

# Up and Running with Elastic Beanstalk

Applications deployed on Elastic Beanstalk are “cloud citizens.” Therefore, they have to submit to “cloud law.” The resulting requirements are sensible for any application expecting to grow above a certain level of traffic. We’ll start this chapter with a description of these requirements, illustrated with examples of applications. Then we will choose an application that is ready for the cloud, and ready for Elastic Beanstalk, and deploy it. The sample application is also available under GPL license, so you can try the full example yourself. You can even use the application, because it’s a generic URL shortener, ready to use. At the end of this chapter you should know how to deploy a Java application on Elastic Beanstalk.

## What Is Elastic Beanstalk?

But what is Elastic Beanstalk, actually? Elastic Beanstalk is one of the many Amazon Web Services, and its purpose is to let developers and engineers easily deploy and run web applications in the cloud, but in a way that they are highly available and scalable. It stands next to other AWS services (like EC2 instances, Elastic Load Balancers, and Auto Scaling —if you are not familiar with these concepts, you can fast forward to Chapter 2, where they are explained in more detail), and uses sensible defaults that you can modify to adapt to your application needs.

Perhaps Beanstalk’s most important feature is deployment. Deployment has always been quite a hassle, even using tools like Hudson or Jenkins for continuous integration. Moving an application around from one environment to another is usually difficult.

At the moment of writing, Amazon only provides Elastic Beanstalk for applications that can be deployed on a Tomcat container, so it’s mainly used for Java web applications, though anything you can run as a WAR inside Tomcat should work.

Elastic Beanstalk helps you with running your application in several ways:

*Deploying*
: You can upload and manage different versions of your application, and switch between them in different environments (e.g., development, test, production environments). We will do an example deployment in this chapter.

*Provisioning*
: When you deploy your application, Elastic Beanstalk will provision the necessary instances, load balancers, and other resources you might have configured.

*Monitoring*
: When your application is running in production, you want to know if something goes wrong, and ideally fix it right away. Elastic Beanstalk will do regular health checks to make sure your application is running, and if not, try to find the cause and solve it. The solution might be to launch a new instance to replace one that is failing, or replace the load balancer if the cause is there. It might even tell you that your security settings are incorrectly configured and that you need to open up port 80, for example. This kind of automated troubleshooting is invaluable to get work off your hands.

  You also have access to all the monitoring metrics provided by Amazon CloudWatch, such as request count, CPU usage, and inbound and outbound network traffic. At all moments, you can see the health status of your application.

*Autoscaling*
: You might have a game application that runs smoothly on one server most of the time. But perhaps the majority of users play during the weekends, so then you need two or three instances. With Elastic Beanstalk, you have triggers for adding or removing instances depending on load, and you can tailor them to your needs (e.g., "increase the number of servers if the average CPU usage goes above 50%").

*Sending notifications*
: Elastic Beanstalk will send you notifications when important events from any of the above activities take place, such as new servers being launched, thresholds being surpassed, or new deployments occurring.

*Managing your running app*
: From the Beanstalk console, you can administer your versions and environments and view the Tomcat logs. You can also restart all your Tomcat instances in one go, or rebuild the whole infrastructure. You can, as well, configure your application and the underlying infrastructure by choosing a different instance type with more or less memory, changing JVM settings and environment variables, or enabling SSH access to your instances.

Beanstalk introduces some terms that we will use throughout the book:

*Application Version*
: A version is the deployable code. For JVM-based applications, that means a WAR file. It has a label and a description. You can see where it is deployed (in what environments, see below), and you can download the file itself if necessary.

*Environment*
: An environment has a deployed version on specific instances, load balancers, auto scaling groups, etc. You can deploy one of the existing versions to any environment inside the application. Typically you could create an environment for production and another one for testing, but you can create as many as you need —and as your budget allows, of course. You can access your environment in a URL of the type *http://<cname>.elasticbeanstalk.com*, where *<cname>* is a value that you choose.

  An environment can be in different health statuses: green (OK), yellow (it hasn't responded within the last 5 minutes), red (it hasn't responded for more than 5 minutes), gray (unknown).

*Events*
: Events tell you what is going on with your environments. Events could be informative, warnings, or errors, such as "environment x has been successfully launched," "instance x is using 90% CPU," or "instance x did not start correctly." You can view the events in the web console, or you can get them sent to you by email.

*Application*
: An application in Beanstalk is a collection of environments, versions, and everything else related to them, like events. You would normally create an Elastic Beanstalk application for each of your applications, but this is not required.

Let's see, then, how to find out if your application can be immediately deployed using Elastic Beanstalk.

## Which Apps Run on Elastic Beanstalk?

As we discovered while trying to run many different Java applications on Beanstalk, most are not immediately ready for running on the cloud. Generally the reason is that they use local—filesystem—storage. This is not a good idea for two reasons: being able to scale, and being prepared for failures.

One big advantage of the cloud is *elasticity*: being able to launch new servers (instances) when usage increases, and shrinking down (terminating servers) when usage subsides. This is what we call *scaling out* and *scaling in*. With the AWS services we get a virtually unlimited number of resources, and with services like Auto Scaling, we can make sure we always have what we need. Not less, but also not more. If we are using several instances and their usage is under a certain threshold (that we can determine), one of them is going to be terminated, because it's not needed anymore.

This means that our instances must be "disposable." So, we can't use local storage unless it's on a temporary basis and we don't count on it when the instance goes away. If there is data that needs to be saved permanently, it has to be in proper persistent storage, such as a relational database (Amazon provides RDS service for this), SimpleDB and/or S3.

Local storage on an AWS instance comes in two forms. There is an EBS volume that acts as the root device, and there is *ephemeral* storage (the real local hard drive). Both are unreliable for storage that needs to be persisted.

Even if you decide to use only one instance and never autoscale, you have to be ready for your instance to die. Think, for example, that if something fails in the underlying hardware of your instance, Elastic Beanstalk will decide to terminate the instance and launch another one, and everything on that instance will be lost.

There is a configuration option on the instances called Termination Protection. This prevents the instance from being terminated, so that the local storage remains accessible. If you use this option to avoid losing your local storage, you will still have to do some work to recover your data.

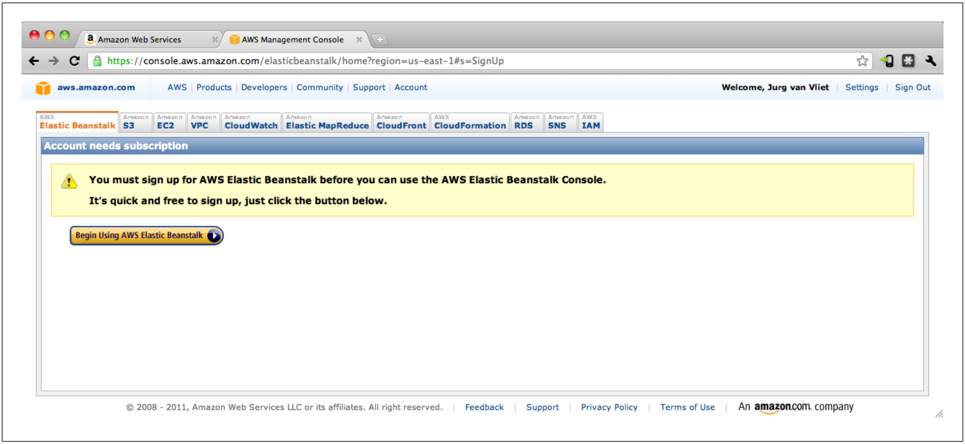
There is another way of protecting your data, and that is to prevent the root volume itself from being deleted when the instance is terminated. You can read more details about it in this article by Eric Hammond.

## Sign Up

Before we can do anything, we have to sign up to Elastic Beanstalk. If you are already an AWS user with some experience, you'll have no trouble with this. You have been exposed to the countless number of services and accompanying acronyms. If you are new to AWS, we suggest you take the easy way, and follow along with us.

Elastic Beanstalk requires you to sign up for a number of other AWS services. Beanstalk uses services like EC2 (compute), EBS (storage), ELB (load balancing), and S3 (another type of storage). Of course, Amazon makes signing up to services as easy as possible. And the only thing you have to do is sign up for Elastic Beanstalk. The rest of the services are automatically added to your arsenal of AWS tools.

But, where to sign up for Beanstalk? If you are already familiar with AWS, you have probably seen the AWS Console. The very first (leftmost) tab in that console is home to Elastic Beanstalk. And having never used Beanstalk before, you probably see something like Figure 1-1.

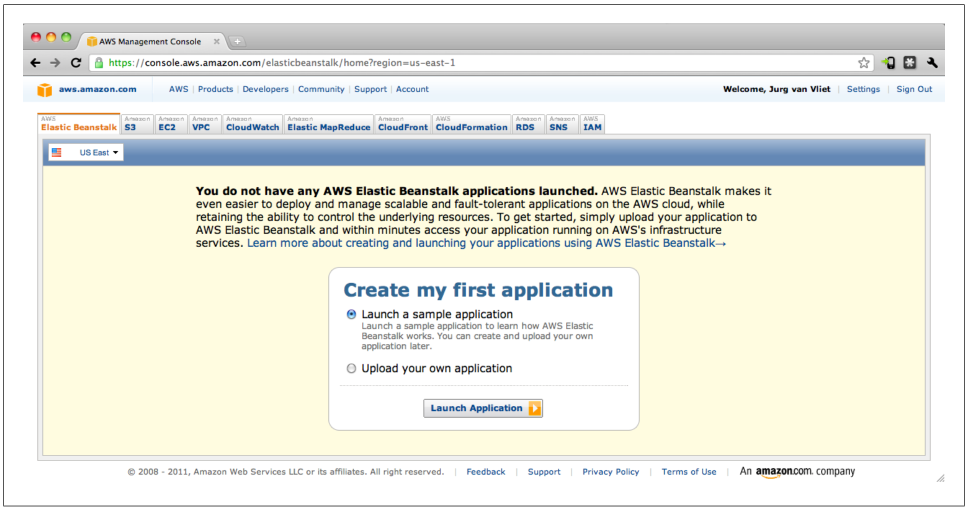

Amazon Web Services
AWS Management Console
https://console.aws.amazon.com/elasticbeanstalk/home?region=us-east-1#s=SignUp
aws.amazon.com
AWS | Products | Developers | Community | Support | Account
Welcome, Jurg van Vliet | Settings | Sign Out
Elastic Beanstalk S3 EC2 VPC CloudWatch Elastic MapReduce CloudFront CloudFormation RDS SNS IAM
Begin Using AWS Elastic Beanstalk
© 2008 - 2011, Amazon Web Services LLC or its affiliates. All right reserved. | Feedback | Support | Privacy Policy | Terms of Use
An amazon.com. company

**Account needs subscription**

**You must sign up for AWS Elastic Beanstalk before you can use the AWS Elastic Beanstalk Console.**

**It's quick and free to sign up, just click the button below.**

*Figure 1-1. Elastic Beanstalk home before signing up*


AWS Management Console
https://console.aws.amazon.com/elasticbeanstalk/home?region=us-east-1
aws.amazon.com
AWS | Products | Developers | Community | Support | Account
Welcome, Jurg van Vliet | Settings | Sign Out
Elastic Beanstalk S3 EC2 VPC CloudWatch Elastic MapReduce CloudFront CloudFormation RDS SNS IAM
US East
Launch Application
© 2008 - 2011, Amazon Web Services LLC or its affiliates. All right reserved. | Feedback | Support | Privacy Policy | Terms of Use
An amazon.com. company

**You do not have any AWS Elastic Beanstalk applications launched.** AWS Elastic Beanstalk makes it even easier to deploy and manage scalable and fault-tolerant applications on the AWS cloud, while retaining the ability to control the underlying resources. To get started, simply upload your application to AWS Elastic Beanstalk and within minutes access your application running on AWS's infrastructure services. Learn more about creating and launching your applications using AWS Elastic Beanstalk→

**Create my first application**

[✓ Launch a sample application]
Launch a sample application to learn how AWS Elastic Beanstalk works. You can create and upload your own application later.

Upload your own application

*Figure 1-2. Elastic Beanstalk home again*

You can also go to the product page for Elastic Beanstalk and follow the instructions. You will need an Amazon.com account, and you will have to provide a credit card and a phone number.

If you are successful in signing up, you'll be notified by email. And the Beanstalk home (the Elastic Beanstalk tab in the AWS Console) will look like Figure 1-2.

For the example we are going to use in this chapter, we will also use SimpleDB, an easy-to-use data store service from Amazon. You will need to sign up to that service as well, in a similar way. You can go to the SimpleDB product page for that. If you don't want to use SimpleDB, you can use a database compatible with Hibernate, such as MySQL, and configure it to use it.

So, ready to go. Now we have to find a useful app, ready to be deployed on Beanstalk.

## Candidates for Running on Elastic Beanstalk

The TIOBE index shows us that Java is the undisputed leader in programming languages over the last decade. So there must be numerous readily available (preferably open source) Java applications. There are, but not all are really useful to us. And, of those that are useful, a lot are just not ready for Elastic Beanstalk in particular, or the cloud in general.

Our first two candidates were JIRA and Liferay. The first is the popular “Bug, Issue and Project Tracking” software. The second is an enterprise open source portal, in use by many high-profile corporations. We could get both to work on AWS with Beanstalk, but not in 15 minutes. Both JIRA and Liferay use local file storage, which you can’t rely on with Beanstalk (see “Which Apps Run on Elastic Beanstalk?” on page 3).

Another very interesting example was Nuxeo, an open source platform for enterprise content management. Nuxeo uses a relational database, but it does not really work with MySQL, or at least using it with MySQL (a relational database we can use with Amazon’s RDS without installation setup) is discouraged. This disqualifies it for our purpose, because we only had 15 minutes to start with.

With some effort we can make these apps run, but they are not 100% cloud-ready, in our humble opinions. We can’t use them out of the box, so to speak. Later in the book, we’ll see how we can extend Beanstalk or massage it to do other things. For now, make sure your app does not use local file storage and uses a database you can set up easily, like Amazon SimpleDB (no setup) or Amazon RDS.

## Hystqio, Our Pick

These days everyone wants their own URL shortener. We didn’t have one yet, and we thought we could easily build one and run it on Elastic Beanstalk, preferably for free within AWS’s Free Usage Tier.

We found a URL shortener called Shorty, written by Viral Patel, at the beginning of 2010. He used it to demonstrate Struts 2 and Hibernate. We asked if we could use it to show how Beanstalk works. And he said we could. (We also got his permission to GPL the codebase.) You can download his application and follow his explanation on his blog.

There were a couple of things we changed:

- We used a more impressive-looking hash generator (instead of counting up, we generate the short codes randomly).

- We added the alternative of using SimpleDB for storage apart from Hibernate (because it is easy to use and less expensive than using MySQL on AWS, probably free).
- We used Maven to help us create a WAR.

The full modified source code and Maven project can be found in our GitHub repository. And you can see it running (on Elastic Beanstalk) at *http://hy.stq.io* (Figure 1-3).

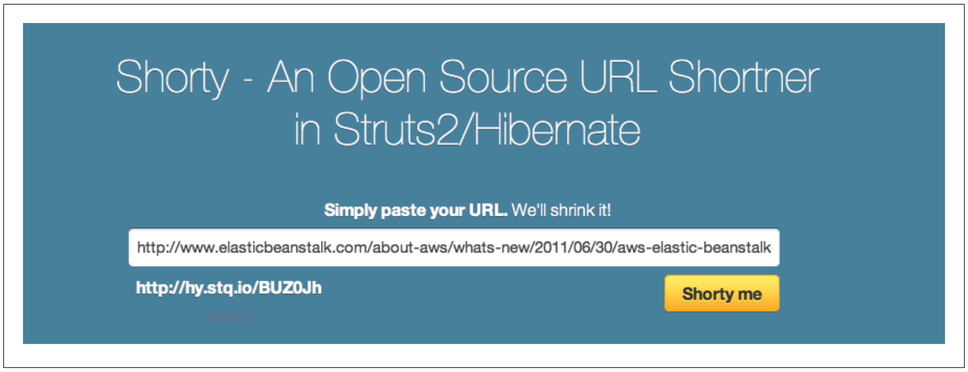

Shorty - An Open Source URL Shortner in Struts2/Hibernate
Simply paste your URL. We'll shrink it!
http://www.elasticbeanstalk.com/about-aws/whats-new/2011/06/30/aws-elastic-beanstalk
http://hy.stq.io/BUZ0Jh
Shorty me

*Figure 1-3. Hystqio*

## The Hystqio Code

We created a project with the structure shown in Figure 1-4. This is the default structure for a web application when using Maven. We created the *pom.xml* file with all the dependencies of the project, including, for example, hibernate, the MySQL connector library, and struts. For using SimpleDB, we added the AWS Java library:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>net.nineapps.hystqio</groupId>
  <artifactId>hystqio</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>hystqio Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>

    […]

    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk</artifactId>
      <version>1.1.9</version>
    </dependency>
  </dependencies>
```

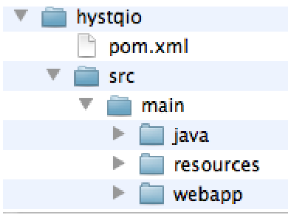


*Figure 1-4. Hystqio project structure*

```
</project>
```

We won't delve into the details of Struts or Hibernate. You can learn more about them on Viral's page. But we want to show you the modified hashing function and the class for accessing SimpleDB. You can also skip the rest of this section and go directly to "Deploy Hystqio to Elastic Beanstalk" on page 12, if you want to focus on the deployment.

### Creating the hash for the short URLs

The hashing function simply generates random strings of six characters, which can be numbers or upper and lowercase letters. We need to make sure a string has not yet been used, but, choosing from 62 characters, we have in the order of tens of billions of strings ($10^{10}$), so we have a long way to go. We'll output some warnings when we start getting repeated strings, and then maybe we can use longer codes or just start reusing them.

This is how we generate the short codes:

```
public class HystqioUtils {

  private static final String CHARSET = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGH
IJKLMNOPQRSTUVWXYZ";
  private static final Random random = new Random(System.currentTimeMillis());

  /**
   * Randomly generates a short code of 6 characters
   * @return
   */
  public static String generateShortCode() {
    StringBuilder buffer = new StringBuilder();
    for(int i=0; i<6; i++) {
      int r = random.nextInt(CHARSET.length());
      buffer.append(CHARSET.charAt(r));
    }
    return buffer.toString();
  }


[…]
```

### SimpleDB as a database

For storage we created an interface that determines what we require to be able to persist our URLs:

```
/**
 * Interface for persisting and retrieving links
 * in the persistence layer (be it RDS, SimpleDB, plain MySQL, etc.)
 */
public interface LinkDAO {

  Link get(String shortCode);
  Link add(Link link);
  void incrementClicks(Link link);
}
```

We have two classes implementing this interface: `HibernateLinkDAO` and `SimpleDBLink DAO`. The Hibernate class is not interesting in this context, and the SimpleDB class uses the AWS SDK for Java for saving and retrieving the short codes in SimpleDB. We have created a domain—something that can be compared to a table for relational databases—called "links" in SimpleDB for storing the URLs with their short code, number of clicks, and date on which they were created.

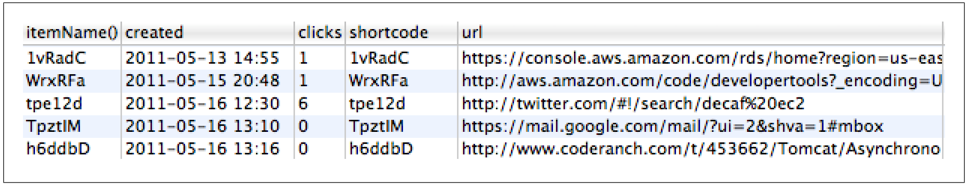
For accessing the SimpleDB API, we first initialize the `AmazonSimpleDB` object, passing the credentials provided in our AWS account (and saved in a properties file):

```
private void initSimpleDBService() {
  ResourceBundle bundle = ResourceBundle.getBundle ("aws");

  AWSCredentials credentials = new BasicAWSCredentials(
      bundle.getString("accessKey"), bundle.getString("secretKey"));

  simpleDB = new AmazonSimpleDBClient(credentials);
}
```

Then we can use the SimpleDB API for *Get Attributes*, *Put Attributes* and *Select* for getting a specific link, adding a new one or modifying, and checking if a specific URL is already in the data store. The main methods for this are listed below:

```
/**
 * Data access object which uses SimpleDB
 * to persist the links.
 *
 */
public class SimpleDBLinkDAO implements LinkDAO {

[…]

  public Link get(String shortCode) {

    AmazonSimpleDB simpleDB = getSimpleDBService();

    Link link = new Link();
    GetAttributesResult result = simpleDB.getAttributes(new GetAttributesRequest
(LINKS, shortCode));
    for (Attribute attribute : result.getAttributes()) {
      if (URL_ATTRIBUTE.equals(attribute.getName())) {
        link.setUrl(attribute.getValue());
      } else if (SHORTCODE_ATTRIBUTE.equals(attribute.getName())) {
        link.setShortCode(attribute.getValue());
      } else if (CLICKS_ATTRIBUTE.equals(attribute.getName())) {
        link.setClicks(Long.parseLong(attribute.getValue()));
      } else if (CREATED_ATTRIBUTE.equals(attribute.getName())) {
        link.setCreated(string2Date(attribute));
      }
    }

    return link;
  }

  public void incrementClicks(Link link) {

    AmazonSimpleDB simpleDB = getSimpleDBService();

    List<ReplaceableAttribute> attribs = new ArrayList<ReplaceableAttribute>();

    boolean clicksOverwritten;
    int attempts = 0;

    do {
      try {

        long oldClicks = numberOfClicks(link, simpleDB);

        // add 1 to number of clicks
        attribs.add(new ReplaceableAttribute(CLICKS_ATTRIBUTE, "" + (oldClicks + 1),
true));

        // use UpdateCondition so we only update
        // if the previous value for clicks was the same that we just retrieved before
        simpleDB.putAttributes(new PutAttributesRequest(LINKS, link.getShortCode(),
attribs,
            new UpdateCondition(CLICKS_ATTRIBUTE, "" + oldClicks, true)));

        clicksOverwritten = false;

      } catch (AmazonServiceException e) {
        log.error(e.getErrorCode());
        if ("ConditionalCheckFailed".equals(e.getErrorCode())) {
          clicksOverwritten = true;
          attempts++;
        } else{
          throw e;
        }
      }
    } while (clicksOverwritten && attempts < 10);

    if (clicksOverwritten && attempts >= 10) {
      log.error("WARNING: Could not update the number of clicks.");
    }

  }

  private long numberOfClicks(Link link, AmazonSimpleDB simpleDB) {
    long oldClicks = 0;

    // get the current number of clicks
    // do a consistent read to get the latest written value
    GetAttributesRequest request = new GetAttributesRequest(LINKS, link.getShort
Code());
    request.setConsistentRead(true);
    GetAttributesResult result = simpleDB.getAttributes(request);

    for (Attribute attribute: result.getAttributes()) {
      if (CLICKS_ATTRIBUTE.equals(attribute.getName())) {
        oldClicks = Long.parseLong(attribute.getValue());
        break;
      }
    }
    return oldClicks;
  }

  public Link add(Link link) {

    SimpleDateFormat format = dateFormat();

    AmazonSimpleDB simpleDB = getSimpleDBService();

    // Check if the URL has already been shortened
    // by doing a consistent read
    SelectRequest request = new SelectRequest(
        "select shortcode from links where url = '" +link.getUrl()+ "'", true);

    SelectResult result = simpleDB.select(request);

    String shortcode = null;

    // if so, get the already existing short URL
    if (result.getItems().size() > 0) {
      // we assume there is only one item
      Item item = result.getItems().get(0);
      for (Attribute attribute : item.getAttributes()) {
        if (SHORTCODE_ATTRIBUTE.equals(attribute.getName())) {
          shortcode = attribute.getValue();
          break;
        }
      }
    }

    // if the url was not yet in the database, generate a new short url
    if (shortcode == null) {
      shortcode = generateShortcode();
    }
    link.setShortCode(shortcode);

    List<ReplaceableAttribute> attribs = new ArrayList<ReplaceableAttribute>();
    attribs.add(new ReplaceableAttribute(URL_ATTRIBUTE, link.getUrl(), true));
    attribs.add(new ReplaceableAttribute(SHORTCODE_ATTRIBUTE, link.getShortCode(),
true));
    attribs.add(new ReplaceableAttribute(CREATED_ATTRIBUTE, format.format(new java.
util.Date()), true));
    attribs.add(new ReplaceableAttribute(CLICKS_ATTRIBUTE, "0", true));

    simpleDB.putAttributes(new PutAttributesRequest(LINKS, link.getShortCode(),
attribs));

    return link;
  }

[…]

}
```

As a result, the "links" SimpleDB domain contents will look similar to Figure 1-5.



| itemName() | created | clicks | shortcode | url |
|---|---|---|---|---|
| 1vRadC | 2011-05-13 14:55 | 1 | 1vRadC | https://console.aws.amazon.com/rds/home?region=us-eas |
| WrxRFa | 2011-05-15 20:48 | 1 | WrxRFa | http://aws.amazon.com/code/developertools?_encoding=U |
| tpe12d | 2011-05-16 12:30 | 6 | tpe12d | http://twitter.com/#!/search/decaf%20ec2 |
| TpztIM | 2011-05-16 13:10 | 0 | TpztIM | https://mail.google.com/mail/?ui=2&shva=1#mbox |
| h6ddbD | 2011-05-16 13:16 | 0 | h6ddbD | http://www.coderanch.com/t/453662/Tomcat/Asynchrono |

*Figure 1-5. "links" SimpleDB domain*

If you want to use the shortener with a relational database using Hibernate, change the value of the `persistence` parameter in the *src/main/webapp/WEB-INF/web.xml* to `hibernate`. Your database can then be configured in *src/main/resources/hibernate.cfg.xml*.

## Building Hystqio

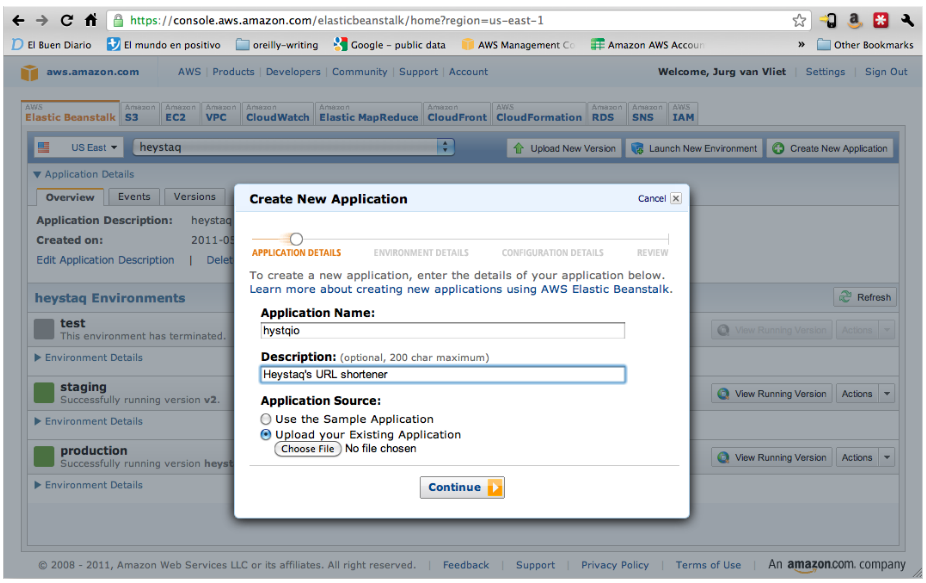
Building is the easiest part. Just run Maven:

```
$ mvn package
```

and you will get *hystqio.war* in the target directory.

## Deploy Hystqio to Elastic Beanstalk

So we have a WAR and we have Elastic Beanstalk: we are ready to run! Go to the Elastic Beanstalk tab of the AWS Console and tap on "Create New Application" (Figure 1-6). Choose "Upload your Existing Application" and select your WAR.


https://console.aws.amazon.com/elasticbeanstalk/home?region=us-east-1
aws.amazon.com
AWS | Products | Developers | Community | Support | Account
Welcome, Jurg van Vliet | Settings | Sign Out
Elastic Beanstalk | S3 | EC2 | VPC | CloudWatch | Elastic MapReduce | CloudFront | CloudFormation | RDS | SNS | IAM
US East | heystaq
Upload New Version | Launch New Environment | Create New Application
Application Details
Overview | Events | Versions
heystaq Environments
test
staging
production
Environment Details
Refresh
View Running Version | Actions
APPLICATION DETAILS | ENVIRONMENT DETAILS | CONFIGURATION DETAILS | REVIEW
Cancel
Continue
© 2008 - 2011, Amazon Web Services LLC or its affiliates. All right reserved. | Feedback | Support | Privacy Policy | Terms of Use | An amazon.com company

**Create New Application**

To create a new application, enter the details of your application below. Learn more about creating new applications using AWS Elastic Beanstalk.

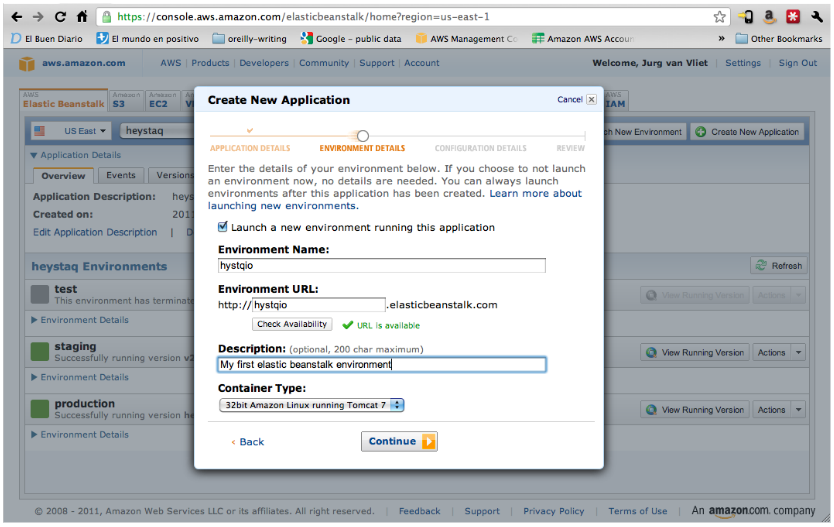
Application Name: hystqio

Description: (optional, 200 char maximum) Heystaq's URL shortener

Application Source:
Use the Sample Application [✓ Upload your Existing Application]
Choose File No file chosen

Figure 1-6. *Creating an Elastic Beanstalk application*

Next, you are prompted to create a new environment. Create one and give it a name, choose a convenient available URL, and choose a container type (Figure 1-7).


https://console.aws.amazon.com/elasticbeanstalk/home?region=us-east-1
aws.amazon.com
Welcome, Jurg van Vliet | Settings | Sign Out
US East | heystaq
Create New Application
heystaq Environments
test
staging
production
Refresh
View Running Version | Actions
APPLICATION DETAILS | ENVIRONMENT DETAILS | CONFIGURATION DETAILS | REVIEW
Cancel
Back
Continue
Check Availability
© 2008 - 2011, Amazon Web Services LLC or its affiliates. All right reserved. | Feedback | Support | Privacy Policy | Terms of Use | An amazon.com company

**Create New Application**

Enter the details of your environment below. If you choose to not launch an environment now, no details are needed. You can always launch environments after this application has been created. Learn more about launching new environments.

[✓] Launch a new environment running this application

Environment Name: hystqio

Environment URL: http://hystqio.elasticbeanstalk.com
✓ URL is available

Description: (optional, 200 char maximum) My first elastic beanstalk environment

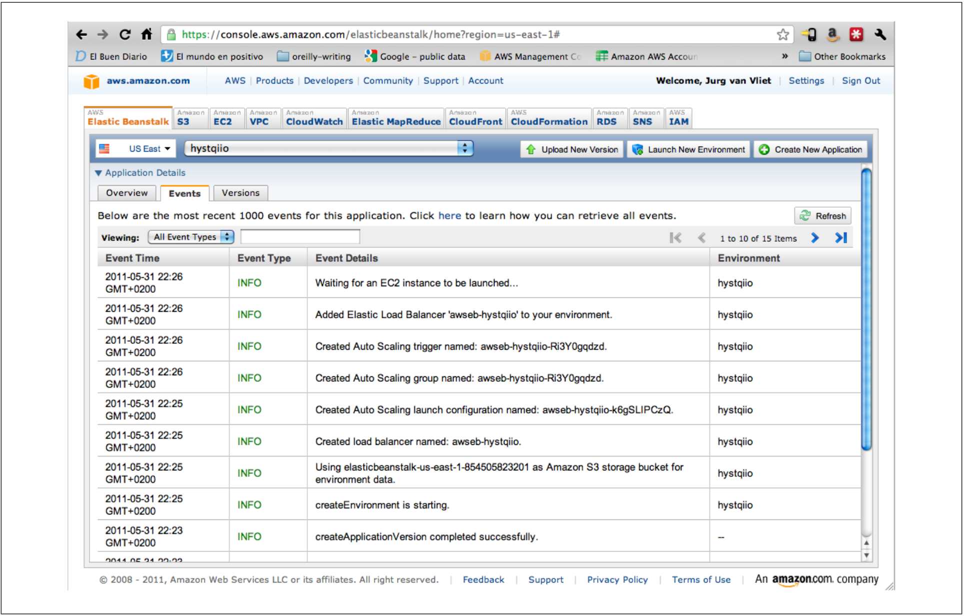
Container Type: 32bit Amazon Linux running Tomcat 7

Figure 1-7. *Create an environment*

In the third step, we can leave the default options. The instance type is the size of the machine we want to use. Micro is fine for a start, and we can benefit from the free tier. The key pair would be used to give us SSH access to the instance, but for now we only want it to run; we'll worry about getting onto the machine later in this book. Take into account, though, that if you are running a real application, you should assign a key pair to be able to log in to your instances when the need arises.

Configure an email address if you want to receive notifications about the changes in status and events happening in your application, such as restarts, instances being removed or added when autoscaling, health check failures, etc. All these options can be changed later on.

When you finish the wizard, you'll see that it takes a few minutes to launch the environment. You can check what is going on in the Events tab (Figure 1-8). In the end, if all goes well, your environment will be in "green" status, with the text "Successfully running version First Release." Tap on "View Running Version" or go directly to *http://<cname>.elasticbeanstalk.com* to see your app running.



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

| Event Time | Event Type | Event Details | Environment |
|---|---|---|---|
| 2011-05-31 22:26 GMT+0200 | INFO | Waiting for an EC2 instance to be launched... | hystqiio |
| 2011-05-31 22:26 GMT+0200 | INFO | Added Elastic Load Balancer 'awseb-hystqiio' to your environment. | hystqiio |
| 2011-05-31 22:26 GMT+0200 | INFO | Created Auto Scaling trigger named: awseb-hystqiio-Ri3Y0gqdzd. | hystqiio |
| 2011-05-31 22:26 GMT+0200 | INFO | Created Auto Scaling group named: awseb-hystqiio-Ri3Y0gqdzd. | hystqiio |
| 2011-05-31 22:25 GMT+0200 | INFO | Created Auto Scaling launch configuration named: awseb-hystqiio-k6gSLIPCzQ. | hystqiio |
| 2011-05-31 22:25 GMT+0200 | INFO | Created load balancer named: awseb-hystqiio. | hystqiio |
| 2011-05-31 22:25 GMT+0200 | INFO | Using elasticbeanstalk-us-east-1-854505823201 as Amazon S3 storage bucket for environment data. | hystqiio |
| 2011-05-31 22:25 GMT+0200 | INFO | createEnvironment is starting. | hystqiio |
| 2011-05-31 22:23 GMT+0200 | INFO | createApplicationVersion completed successfully. | -- |

*Figure 1-8. Elastic Beanstalk events*

## Conclusion

In this chapter, we showed you how easy it is to deploy a Java application on Elastic Beanstalk by using an open source URL shortener. In the process, we introduced the basic components of Beanstalk, and explained how you should design your application to be Elastic Beanstalk–ready.

In the coming chapter, we will explain in more detail how all the underlying AWS services interact when you deploy an application on Beanstalk.

CHAPTER 2

# Elastic Beanstalk Explained

To understand Elastic Beanstalk, you need to know how it works. It uses a number of Amazon AWS services, and it adds application deployment on top of that. Because of the nature of Elastic Beanstalk, you can't treat it as a black box. You need to have a basic understanding of the underlying AWS services, and how Elastic Beanstalk makes them work in concert. In this chapter, we'll present an overview of the services that power Elastic Beanstalk, and illustrate how to use them independently. At the end of this chapter you'll know the components that interacted to make Hystqio (the example of the previous chapter) work.

This is a lot of information at once, so don't worry if it doesn't look as simple as you thought. Later on, you can come back to this chapter as a reference for all the AWS services.

## Elastic Beanstalk and AWS

Elastic Beanstalk is not a Google App Engine. And, even though it resembles Heroku a bit, it is quite different from that as well. The purpose of Beanstalk is not to be a simple solution that will scale infinitely. But, to borrow Amazon's marketing, it is something that is "impossible to outgrow."

Google App Engine (GAE) is a Platform as a Service, or PaaS. It is designed to manage everything for your application; it promises you won't have to worry about anything anymore. GAE completely hides anything resembling servers, IP addresses, load balancers, backups, etc. Heroku is similar, except that it is built on top of Amazon AWS. Heroku wants you to "forget servers," and promises to "run everything."

Elastic Beanstalk is not a PaaS. You can forget about servers for a while, but if you are not happy with them, you can take control. Load balancing is also taken care of, but it can be tweaked and adjusted as you see fit. As a matter of fact, you can take over all individual AWS services underlying Elastic Beanstalk.

We think Beanstalk is best understood as an entry into IaaS (Infrastructure as a Service). It helps you to get onto the AWS cloud quickly. With sensible defaults, you can get a long way into the life of your application. But there comes a time when you want or need to customize the individual components that make up Elastic Beanstalk.

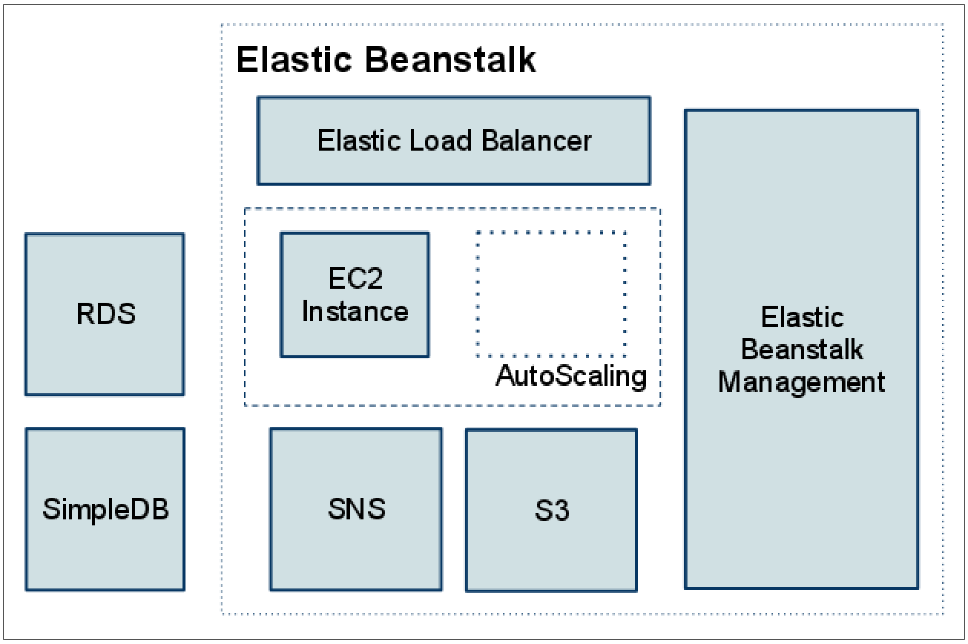


*Figure 2-1. Elastic Beanstalk versus Amazon AWS*

The underlying services, like *Elastic Load Balancing*, for example, can be configured in many different ways. When traffic increases, you need to find an *instance type* suitable for your workload. You might need to add some memcached, or other infrastructure components. And if you use *RDS* (relational databases on Amazon AWS), you need to work with *security groups* to configure permissions to access your database. Or you might need to change the Tomcat version that is part of the default *images*. Even though Beanstalk will get you quite far, sooner or later you'll want to replace certain elements of the Beanstalk infrastructure components.

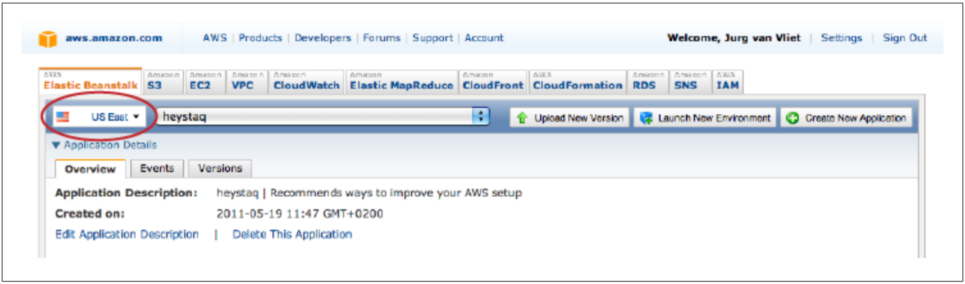
Elastic Beanstalk brings together AWS services like EC2, Auto Scaling, and S3 for the purpose of deploying an elastic Java application (see Figure 2-1). In order to work with your Beanstalk infrastructure, you need to be familiar with these services. In this chapter we'll introduce them. This is a very quick overview; it is basically a summary of the first half of Programming Amazon EC2. We wrote that book to help you build applications on Amazon AWS, and it covers everything in much more detail. But with this chapter, you should be able to find your way around Beanstalk.

## Regions and Availability Zones

Amazon AWS is organized in *regions*, which determine where in the world your resources will be hosted. Whatever you do with AWS, you have to first choose a region. Currently there are two regions in the United States, two in Asia, and one in Europe. Undoubtedly Amazon will open more regions in the near future. At this moment, Elastic Beanstalk is only available in the US East region. This was the first region that was opened, and it is also the region where new services are first made available.

Every region has a number of *availability zones*. These zones are designed to be physically separated, but still part of one (data) network. The purpose of different availability zones is to make your infrastructure more resilient to failures related to power and network outages. Availability zones are an extremely powerful and important feature of AWS. As the outage of April 21, 2011 has shown us, if you work properly with different zones, the harm done to your application will be minimal or nonexistent in even the worst-case scenarios.

If you work with Amazon AWS, it is good to realize that the tools operate by default in the US East region (Figure 2-2). If you don't specify otherwise, everything you do will be in this region. There is no default availability zone, though. If you don't specify an availability zone when creating a new resource, Amazon AWS will choose one for you.



Application Description: heystaq | Recommends ways to improve your AWS setup
Created on: 2011-05-19 11:47 GMT+0200

*Figure 2-2. Region selector in the AWS Console*

## Working with AWS Services

AWS comes with a mature set of tools to operate your infrastructure. There are commercial tools available, but we mostly work with the tools AWS provides. There are the command-line tools and the AWS Console, a browser-based management interface. For Elastic Beanstalk and other services, there is also the AWS Toolkit, an Eclipse plugin. We will talk more about it in Chapter 3. And there is a full web services API to access all the services, with existing libraries for the most commonly used programming languages.

AWS provides a very complete web services API to access all its services, so that for each service you can programmatically access all the functionality provided by AWS. An example of this was already shown in Chapter 1 with SimpleDB, and we'll use the Elastic Beanstalk API in Chapter 3 for automating deployments.

Every service comes with its own place in the Console, and with its own set of command-line tools. It is important to realize that the command-line tools always implement 100% of the available features. The Console does not. AWS is organized in product teams, building the services. Each product team builds its own set of tools. This leads to minor, but annoying inconsistencies.

## Command-Line Tools

Installing the command-line tools takes a few minutes, but it's good to have them. Even though most EC2 functionality is supported by the web-based Amazon AWS Console, you will occasionally have to use the command line for some features that are not yet implemented in the console. Plus, later on you might want to use them to script tasks that you want to automate. Running the command-line tools is not difficult if you set up the environment properly.

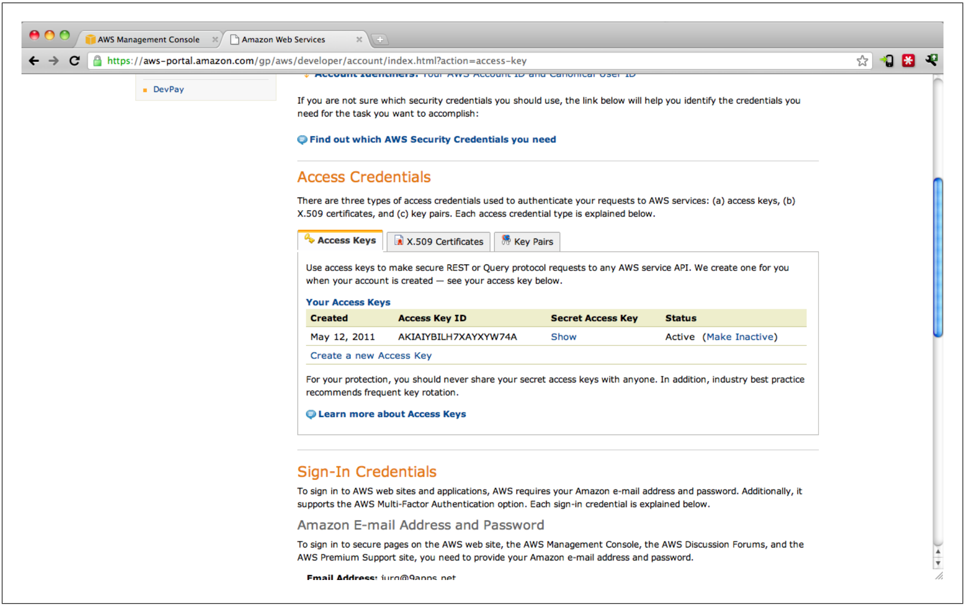
Access to Amazon AWS is protected in a couple of different ways. There are three types of access credentials (you can find these in the "Account" section if you look for "Security Credentials" at *aws.amazon.com*):

1. *Access Keys*, for REST and Query protocol requests
2. X.509 certificates, to make secure SOAP protocol requests
3. *Key Pairs*, in two different flavors, for protecting CloudFront content and for accessing your EC2 instances

For the command-line tools, you will need X.509 credentials. An AWS account doesn't come with X.509 certificates, but you can create them, or upload your own. We ask Amazon AWS to create our X.509 certificates, and immediately download both the Access Key ID and the Secret Access Key (Figure 2-3).

With our downloaded certificates and the access key, we can set the environment variables. For this we create a bash script we call `initaws`, like the one listed below. (For Windows, we would have created a BAT script). For the examples in this book, we will use the command-line tools for EC2, Elastic Beanstalk, and IAM, but you can install other tools in the same way if you need them. Specify the directory where you downloaded your private key and certificate in the EC2_PRIVATE_KEY and EC2_CERT environment variables:

```
#!/bin/bash
export JAVA_HOME=/System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home
export EC2_HOME=/Users/jurg/src/ec2-api-tools-1.3-46266
export AWS_ELB_HOME=/Users/jurg/src/elasticbeanstalk-cli
export AWS_IAM_HOME=/Users/jurg/src/IAMCli-1.2.0
```


AWS Management Console
Amazon Web Services
https://aws-portal.amazon.com/gp/aws/developer/account/index.html?action=access-key
DevPay

Account Identifiers: Your AWS Account ID and Canonical User ID

If you are not sure which security credentials you should use, the link below will help you identify the credentials you need for the task you want to accomplish:

Find out which AWS Security Credentials you need

Access Credentials

There are three types of access credentials used to authenticate your requests to AWS services: (a) access keys, (b) X.509 certificates, and (c) key pairs. Each access credential type is explained below.

Access Keys | X.509 Certificates | Key Pairs

Use access keys to make secure REST or Query protocol requests to any AWS service API. We create one for you when your account is created — see your access key below.

Your Access Keys

| Created | Access Key ID | Secret Access Key | Status |
|---|---|---|---|
| May 12, 2011 | AKIAIYBILH7XAYXYW74A | Show | Active (Make Inactive) |

Create a new Access Key

For your protection, you should never share your secret access keys with anyone. In addition, industry best practice recommends frequent key rotation.

Learn more about Access Keys

Sign-In Credentials

To sign in to AWS web sites and applications, AWS requires your Amazon e-mail address and password. Additionally, it supports the AWS Multi-Factor Authentication option. Each sign-in credential is explained below.

Amazon E-mail Address and Password

To sign in to secure pages on the AWS web site, the AWS Management Console, the AWS Discussion Forums, and the AWS Premium Support site, you need to provide your Amazon e-mail address and password.

Email Address: jurg@9apps.net

*Figure 2-3. Amazon AWS credentials*

```
export PATH="$EC2_HOME/bin:$AWS_IAM_HOME/bin:$AWS_ELB_HOME/bin:$PATH"

export EC2_KEY_DIR=/Users/jurg/.ec2
export EC2_PRIVATE_KEY=${EC2_KEY_DIR}/pk-4P54TBID4E42U5ZMMCIZWBVYXXN6U6J3.pem
export EC2_CERT=${EC2_KEY_DIR}/cert-4P54TBID4E42U5ZMMCIZWBVYXXN6U6J3.pem
export AWS_CREDENTIAL_FILE=${EC2_KEY_DIR}/credentials.txt
```

Notice that we also indicate the location of a file that contains the access keys (*credentials.txt*), like so:

```
AWSAccessKeyId=AKIAIYBIJOUYDEMQW74A
AWSSecretKey=Hg4unEkgQwDMWNo+wqujnEV4yUYwx7nUOkUxwH7t
```

This is needed by some of the command-line tools, such as IAM.

You can then run your bash script in your terminal with `source initaws`. Let's see if this worked by invoking a command, `ec2-describe-regions`, from the EC2 tools:

```
$ ec2-describe-regions

REGION	eu-west-1	ec2.eu-west-1.amazonaws.com
REGION	us-east-1	ec2.us-east-1.amazonaws.com
REGION	ap-northeast-1	ec2.ap-northeast-1.amazonaws.com
REGION	us-west-1	ec2.us-west-1.amazonaws.com
REGION	ap-southeast-1	ec2.ap-southeast-1.amazonaws.com
```

For Elastic Beanstalk, you can make a call to list the existing applications (even if you don't have any yet), such as:

```
$ elastic-beanstalk-describe-applications

No applications found.
```

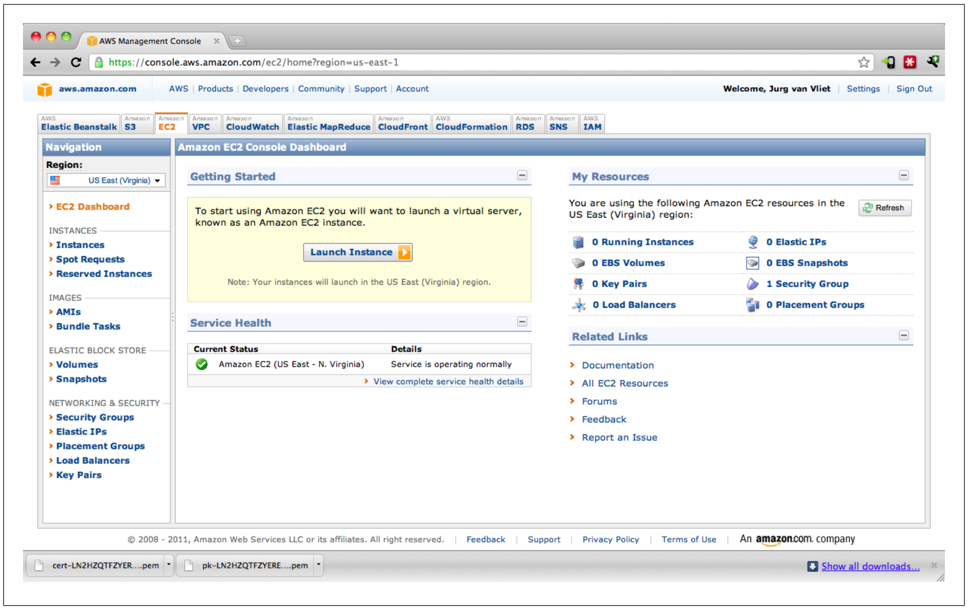
For testing IAM tools, you can run:

```
$ iam-accountgetsummary

Groups: 0
Users: 3
UsersQuota: 5000
GroupsQuota: 100
GroupPolicySizeQuota: 5120
AccessKeysPerUserQuota: 2
UserPolicySizeQuota: 2048
GroupsPerUserQuota: 10
ServerCertificates: 2
SigningCertificatesPerUserQuota: 2
ServerCertificatesQuota: 10
```

We'll take a look at the Amazon AWS Console next.

## The AWS Console

What is there to say about the Amazon AWS Console? We've used it ever since it was launched. There are things we would like to see different, but it is a very complete tool (Figure 2-4).



To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Note: Your instances will launch in the US East (Virginia) region.

| Current Status | Details |
|---|---|
| ✓ Amazon EC2 (US East - N. Virginia) | Service is operating normally |

› View complete service health details

You are using the following Amazon EC2 resources in the US East (Virginia) region:

| | |
|---|---|
| 0 Running Instances | 0 Elastic IPs |
| 0 EBS Volumes | 0 EBS Snapshots |
| 0 Key Pairs | 1 Security Group |
| 0 Load Balancers | 0 Placement Groups |

© 2008 - 2011, Amazon Web Services LLC or its affiliates. All right reserved. | Feedback | Support | Privacy Policy | Terms of Use | An amazon.com. company

*Figure 2-4. Amazon AWS Console*

You already know where to find the home of Amazon Elastic Beanstalk. If you've already explored the other tabs (services) in the Console, you will have noticed that the first application created all sorts of other things. Let's see what those are...

## Elastic Compute Cloud (EC2)

When you create a Beanstalk application, the servers are *EC2 instances*, and they are configured behind a load balancer to expose them to the outside world. There are a couple of things that are hidden by Beanstalk but that are necessary for understanding what is going on. You need to understand how storage is organized (Elastic Block Store) and how the instances are launched from a particular *image*. We'll briefly cover the different assets, and give them a place in the Beanstalk-orchestrated infrastructure.

### Instances

An instance is the virtual counterpart of a server. It is probably called an instance because it is launched from an (immutable) *image*. We can think of an image as an—object-oriented programming—class, and the instances launched from it are instances of that class.

Instances come in types. You can think of a type as the "size" of the machine. The default type for Elastic Beanstalk is *Micro*, which supports both 32- and 64-bit architectures. The other types that run 32-bit architectures are *Small* and *High-CPU Medium*. All the others are exclusively 64-bit instances. This is important because it shows you that scaling up (scaling by using a bigger server) is quite constrained for 32-bit instances. If you decide to use a 32-bit AMI, you will only be able to launch micro, small, and large instances, but if your AMI is 64 bits, you have currently nine different instance types to choose from.

A *Micro* instance costs approximately US$0.02 per hour. On the other end, a *Quadruple Extra Large* instance is available for approximately US$2.40 per hour.

An instance provides a "predictable amount of dedicated compute capacity." For example, the *Large* instance provides:

- 7.5 GB memory
- 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each)
- 850 GB instance storage
- I/O Performance: High

Amazon AWS describes an EC2 Compute Unit like this: "One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor. This is also the equivalent to an early-2006 1.7 GHz Xeon processor referenced in our original documentation."

## AMIs

An Amazon Machine Image, or AMI, is like a boot CD. It contains the root image with everything necessary to start an instance. There are many publicly available AMIs, and you can create your own preconfigured for your needs. The available operating systems include various flavors of Linux, Windows, and OpenSolaris. Often AMIs are simply referred to as images.

There are two different kinds of AMIs. The "old" kind of AMI is stored on S3. Launching an instance from an S3-backed AMI (as they are called) gives you an instance with the root device in the machine itself. Instances launched from an S3-backed AMI cannot be stopped and started; they can only be restarted or terminated. You probably won't want to use these.

The other, newer kind of AMI, probably used by most people, is stored in EBS, or Elastic Block Store. The most important difference for now is that the root device is an EBS volume (EBS will be described in detail later) and it can survive the instance itself. Because of this, an EBS-backed instance can be stopped and started, making it much easier to only use the instance when you need it. A stopped instance does not cost you anything, apart from the EBS storage used.

Beanstalk comes with default (EBS) AMIs. These AMIs contains everything necessary to run a WAR in a Tomcat environment, with Apache Server running on a Linux operating system. And they tightly integrate with the deployment mechanism of Beanstalk. You can use your own images, as we'll see in Chapter 4, but you'll have to find a way to play nice with Beanstalk's way of working.

## Elastic Block Store

EBS is AWS's way of persisting data. These "network volumes" are somewhere in between NAS (networked attached storage) and SAN (storage area networks). They are extremely versatile, and have reasonable performance. (Some people disagree with this remark on performance.)

Beanstalk does not use EBS directly, but the instances are EBS-based. What is important to realize is that the instance storage is not persisted this way. If your instance dies, your EBS volume is lost. This is the reason you can't use local file storage in your application, as we have seen before.

## Security Groups

The Security Groups are sometimes called "firewall." We tend to think of security groups as a combination of a firewall and vlans. A security group is its own little network; instances within a group can freely communicate without constraints. Groups can allow connections to other groups; again, unconditionally. And you can selectively open a group to an (outside) address or group of addresses. It is straightforward to create DMZ (demilitarized zones) for a database or group of application servers.

Beanstalk will create the necessary security groups for you. But if you want to use RDS, or other infrastructure components, you have to manage access based on these security groups. A memcached instance will only become available to the Beanstalk instances if you allow access from the security group that Beanstalk created for you.

## Elastic Load Balancers

Wikipedia says that *load balancing* is something like "a technique to distribute workload evenly across two or more computers, network links, CPUs, hard drives, or other resources, in order to get optimal resource utilization, maximize throughput, minimize response time, and avoid overload." And, Wikipedia notes, it can also "increase reliability through redundancy." To finish the description, Wikipedia claims load balancing "is usually provided by a dedicated program or hardware device."

Amazon Elastic Load Balancing is all of the above. It is a load balancing service. It distributes load evenly across availability zones, and in those zones evenly across instances. ELB checks the health of the instances, and it will not route traffic to unhealthy instances. You have the ability to use something they call *sticky sessions*, which you can use to force a particular session to one instance. You would need this if the instances keep session data in a nonshared location, such as local memory.

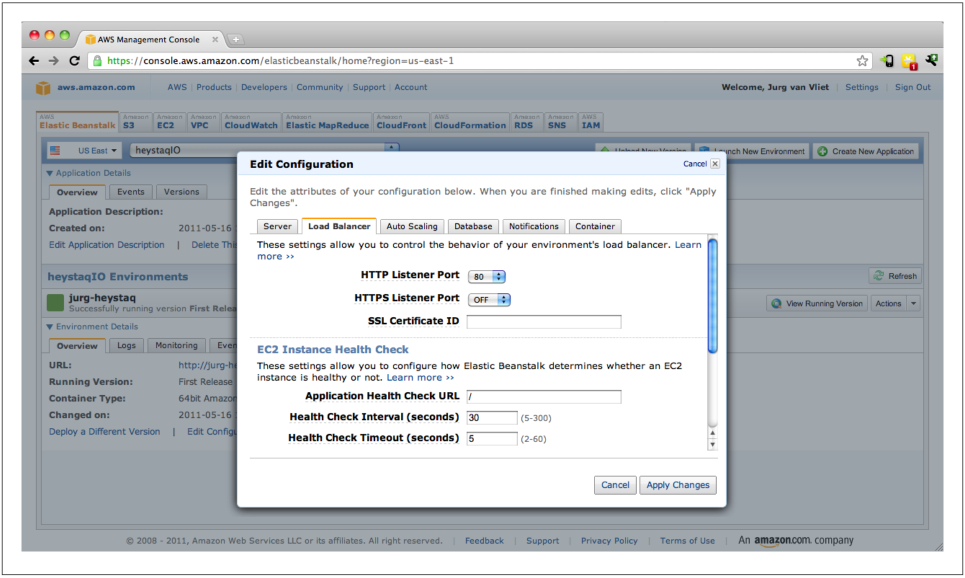
But it is not a "dedicated program" or a "hardware device"; it is a load balancing service. As a service, it can automatically scale its capacity depending on incoming traffic. As a result, an ELB is not referenced by an IP address but by a (fully qualified) domain name. It is said an ELB scales best with "slowly" increasing/decreasing traffic. What we have seen so far is that spikes are handled quite well.

Because an ELB is a service, with underlying instances that change, it is important to set a low TTL on your CNAME records pointing to the ELB. Amazon AWS recommends a TTL of 60, which is not always supported by the different DNS providers. We always use Route53, giving us the control and performance we need for working with ELBs.

The ELB is less hidden in Beanstalk than in the Console. But, as in the Console, it only supports a subset of features in Beanstalk (Figure 2-5). If your app does not expose itself through HTTP or HTTPS, you will have to modify your ELB using the command-line tools.

## Key Pairs

*Key pairs* is one of the ways AWS handles security. It is also the only way to get into your fresh instance the first time you launch it. You can create an SSH key pair and


Edit Configuration

Edit the attributes of your configuration below. When you are finished making edits, click "Apply Changes".

These settings allow you to control the behavior of your environment's load balancer. Learn more ››

HTTP Listener Port: 80

HTTPS Listener Port: OFF

SSL Certificate ID:

EC2 Instance Health Check

These settings allow you to configure how Elastic Beanstalk determines whether an EC2 instance is healthy or not. Learn more ››

Application Health Check URL: /

Health Check Interval (seconds): 30 (5-300)

Health Check Timeout (seconds): 5 (2-60)

*Figure 2-5. AWS Elastic Beanstalk Configuration*

pass it on to the instance you launch. The public key will be stored in the instance in the right place, while you keep the private key to log in to your instance.

You can create a key pair through the Amazon Web Console. Go to Key Pairs and click "Create Key Pair." Give it a name and store the downloaded private key somewhere safe—you won't be able to download it again.

As you saw in Chapter 1, Beanstalk can give the instances it launches a key pair. If you do that, you can access the instances to get to the log files directly instead of through the Beanstalk console, for example.

## Other AWS Services

There are many more EC2-related services and assets. Many of these you will not use immediately, but it is good to at least know of their existence.

If you start/stop (or terminate) an instance, you lose the IP addresses and associated domain names. When using Beanstalk, this is not a big problem, because we have an Elastic Load Balancer exposing our instances. But if you want to point directly to an instance, you can't afford to lose the IP address or domain name. It would require changing your DNS all the time. You can remedy this with an Elastic IP Address. You can request an Elastic IP Address, and associate it with any instance (in the same region). This Elastic IP Address is yours until you release it again, regardless of what happens to your instances.

If you use an instance, you probably use it in the “normal” way. You pay per hour, and only for what you need. You can also use what are called *spot instances*. These are instances you bid on. You specify a price for which you want to run an instance from a particular image. You can, for example, use this for doing work that you can easily schedule to do later, like processing large amounts of traffic data, creating PDFs, or sending large batches of mail.

The last important service in working with AWS is the ability to take point-in-time snapshots of your EBS volumes. For Beanstalk this is not immediately useful, but for a lot of other use cases you can use this to make backup/restore easier. See Programming Amazon EC2 for more details.

## Auto Scaling

The big promise of the cloud is elasticity, the ability to scale up and down according to traffic. This is what AWS (and lately many others) calls *auto scaling*. You automatically launch more instances when your traffic increases, and terminate them again when the traffic decreases.

Auto Scaling is an important part of Beanstalk. In Chapter 1 we briefly touched the subject, but we allowed Beanstalk to choose sensible defaults. In this case it scales based on the ingoing network traffic, aggregated by the group of instances.

You can also set the alarms that cause the Auto Scaling to initiate scaling activities on other CloudWatch metrics. We most often scale on CPUUtilization. But you can also scale on other metrics, as the default Beanstalk deployment showed us.

What exactly the optimum Auto Scaling configuration is depends on your application, and how the instances perform under stress.

## CloudWatch

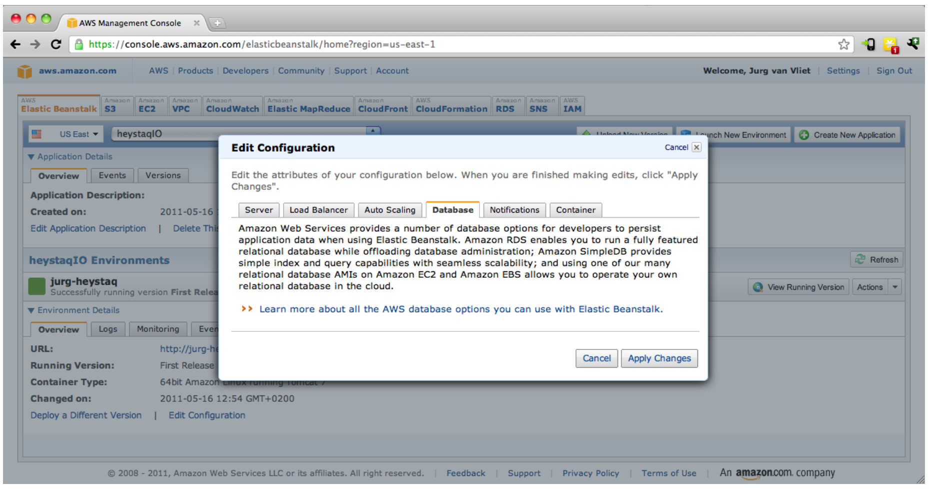
CloudWatch has come a long way since we published Programming Amazon EC2 in early 2011. The core of the service is still the same: two weeks of data with various metrics. By default, the measurements are taken every 5 minutes, but you can enable detailed metrics to take them every minute. You can configure it per EC2 instance. (Elastic Beanstalk gives you this choice as well.) CloudWatch is very helpful in understanding behavior and performance. Most other services (RDS, ELB) come with CloudWatch metrics as well.

The first addition that we are very happy with is CloudWatch support in the Console. It will take you a while to get intimate with this part of the Console. But it is very much easier than just the cmdline; a list of numbers is not very informative for most of us. CloudWatch in the Console gives you a good, visual way to delve into your metrics.

The other addition is more exciting, and it came as a bit of a surprise. You can add your own custom metrics. If you regularly feed your data into CloudWatch, all the features can be used to work with this stream. You can visualize, but you can also add alerts. In this way you can add application-level metrics that are not possible with plain CloudWatch. Imagine, for example, if your application is a multiplayer game, and you want to know the number of users logged in and playing at any moment of the day.

## Database

In the Beanstalk configuration panel, you see a "Database" tab, but it currently doesn't provide any functionality. There are not many apps without a database. And AWS does have interesting database services, such as RDS. But unfortunately this tab does nothing yet (Figure 2-6).



Application Description:
Created on: 2011-05-16
URL: http://jurg-h
Running Version: First Release
Container Type: 64bit Amazon Linux running Tomcat 7
Changed on: 2011-05-16 12:54 GMT+0200

© 2008 - 2011, Amazon Web Services LLC or its affiliates. All right reserved.

**Edit Configuration**

Edit the attributes of your configuration below. When you are finished making edits, click "Apply Changes".

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

Amazon Web Services provides a number of database options for developers to persist application data when using Elastic Beanstalk. Amazon RDS enables you to run a fully featured relational database while offloading database administration; Amazon SimpleDB provides simple index and query capabilities with seamless scalability; and using one of our many relational database AMIs on Amazon EC2 and Amazon EBS allows you to operate your own relational database in the cloud.

>> Learn more about all the AWS database options you can use with Elastic Beanstalk.

Cancel | Apply Changes

*Figure 2-6. Amazon Beanstalk configuration*

## Simple Notification Service

When deploying the app in Chapter 1, you could give Elastic Beanstalk an email address to send you notifications. If you did that, you received a confirmation mail. And, if you confirmed, you received notifications by mail. This mechanism is built with Amazon Simple Notification Service, or SNS.

SNS is a notification service with topics and subscribers. In this case, a subscriber is an email address, but you can also have subscribers with a URL as an endpoint. It is interesting to realize that Auto Scaling uses SNS underwater to have notifications sent based on certain events (alerts) CloudWatch picked up.

One thing you can't do in Elastic Beanstalk is add another email address. But you can add one to the topic using the SNS tab in the Console.

## S3

S3 is AWS's storage service. We describe it as the most revolutionary of all of AWS's services. But it only plays a small part in the default Elastic Beanstalk setup. You can configure your Beanstalk environment to rotate its logfiles to an S3 bucket (a sort of directory) every hour. Later, you can use other services to process the log files, at your convenience.

S3 is also the place where the WARs are kept. You can upload files with the same name every time, and Beanstalk will prepend some code to make them unique.

## Identity and Access Management

AWS Identity and Access Management allows you to create users with a limited set of rights to specific AWS resources.

We could, for example, create a user that can only access S3 services, and use it in a script that uploads new versions of an application to S3. We could even restrict it to have access to one specific bucket, the `deploys` bucket, and then we'll be sure we won't mess with other content stored in S3 by mistake. And if someone gets hold of the credentials of this user, he won't be able to do as much harm as with the general AWS credentials: it won't be possible to use them to terminate servers, launch new ones, etc.

## Hystqio on Elastic Beanstalk

So now that we've talked about all the underlying AWS services, we have a clearer idea of what happened in the previous chapter when deploying an application. Figure 2-7 depicts most of the AWS services being used. When we launch an environment, Elastic Beanstalk takes a predefined AMI installed with the Linux operating system, Tomcat, and Apache Server and launches a new instance of the type that you specify. It then takes the WAR that we provide (the version), which is uploaded to S3, renames it to *ROOT.war*, and copies it to the webapps directory of the Tomcat installation. At any moment you can log in to this instance and see for yourself, by using a key pair. The Tomcat logs are periodically published in S3, and you can see them in the AWS Console.

Beanstalk will also set up an elastic load balancer and make it respond to the URL you provided before. By default, it will launch one instance and configure Auto Scaling to scale between one and four instances, depending on network traffic. CloudWatch will be monitoring the behavior of your infrastructure and gathering metrics, while Simple Notification Service will keep you informed of all the relevant "events." If a threshold is passed, CloudWatch will send alarms, which will be attended by Auto Scaling, and

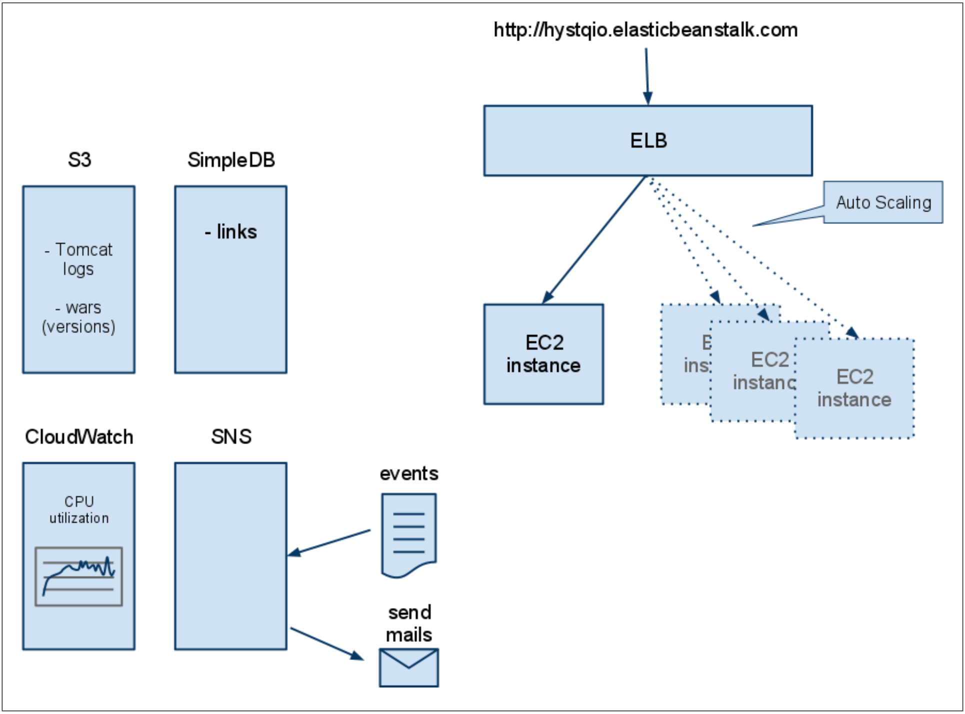


*Figure 2-7. Hystqio deployed with Elastic Beanstalk*

this might cause new instances to be launched, or old ones to be terminated. SNS will also listen to these alarms and send you emails about them.

Finally, in the example we showed, we are using SimpleDB for this particular application for our persistent storage of links.

## Conclusion

Though this chapter is relatively long, it does not cover every detail of all services involved. We tried to show the place of the services, in the context of Elastic Beanstalk. But there are many more things to say about each of them.

You can learn more about these services through customizing your Beanstalk environments or the individual assets directly. You can always see all the resources being used from the Amazon Console. This will also help you understand what is going on "behind the scenes."

In Chapter 3 we will get very practical again and show different ways in which you can use Elastic Beanstalk in your projects.

CHAPTER 3

# Working with Elastic Beanstalk

In this chapter we want to show how you can work with Elastic Beanstalk in your every day —work— life. Developing applications can be facilitated when using Elastic Beanstalk, especially when you work with Eclipse. Your continuous integration testing can be configured to integrate with Elastic Beanstalk as well. Running production environments will use many of the concepts we explored in Chapter 2. We'll use Elastic Beanstalk Configurations to quickly spin up entire environments for staging or performance testing purposes. And, in the end, we'll take one of these environments and start looking for limits with serious load tests.

## Different Environments with Elastic Beanstalk

In the most limited life cycle you have one environment: the production environment. But in most cases, you also have at least a development environment. If your audience (or app) grows, you will find the need to include one or two environments to minimize the impact of rolling out features and functionality. Often, an application ends up with four different kinds of environments, each with their peculiarities:

*Development*
: Most of the time the local machine is the development environment of choice. But if you cooperate/collaborate with different (groups of) people, you want to share your work. Especially when you work with multi-disciplinary teams; not everyone has Git and Tomcat running on their laptops.

*Test*
: Before you can do any sort of production deployment, you need to go through several cycles of testing and fixing. This is usually a separate process, and you don't want development changes to interfere too much with it. So you want a separate environment. Testing usually means functional, but sometimes might include load and stress tests. Usually this environment is permanently available, but a lot smaller (with smaller and perhaps fewer instances) than production.

*Staging*
: Staging is, at least in structure, 100% identical to production. This is the place to practice deployments one last time. This is usually an environment you don't have 100% of the time. With clouds like AWS in general, and with Beanstalk in particular, it is very easy to build a full infrastructure for just a couple of hours. With minimal cost you can do stress tests on an infrastructure that is identical to production. Even though there is nothing like production traffic, this is also the place to try different settings for Auto Scaling.

*Production*
: This is the real deal. Staging has seen all testable scenarios, hopefully. But real users are something very different. A production system, therefore, is not something you build and run. It is something you evolve, and if you are not careful, it runs you.

We'll introduce an application we have been building the last couple of months. The application is called heystaq, and it is intended to become a system you can use to visualize, track, and improve your AWS infrastructure. Because we didn't have much time, we built the first version during a Hackathon organized by The Next Web, sponsored by Amazon AWS. We had two days to build a working prototype of the visualization part.

heystaq will gather data about your AWS infrastructure. It will allow users to define metrics they can track over time and that can be compared to those of other similar apps on this platform. It will be possible to measure an application's performance relative to benchmarks for that type of application.

So for heystaq we need to build a scalable system, and to build something fast, we needed an environment where we could rapidly add and test features. Elastic Beanstalk was an interesting tool for this. The Java VM is a good choice as a runtime for an application like this. Java also has the added benefit that we can use the AWS SDK, the official Java library for accessing the complete AWS API. We chose to work with Scala, however. We were drawn to its features of less code and expressive power. The only drawback was that we know Java, but we hadn't seen Scala before the Hackathon.

## Interacting with Elastic Beanstalk

Elastic Beanstalk comes with several different tools, most of which we have already seen. You have the Console, where you can interact with your applications and environments. You have the command-line tools, basically doing the same thing as the Console, but which can be used for scripting. And, as with all other AWS services, you can do everything you want through the API. This API can be used directly, or through several SDKs (Java, PHP, .NET) AWS ships.

Because you can do everything with the API and command-line tools, you can extend Elastic Beanstalk in interesting ways. You can integrate Elastic Beanstalk in your development environment; for example, by writing an Eclipse plugin. You can also do continuous integration with Hudson, automatically deploying successful versions to Elastic Beanstalk. We used the AWS Java SDK for this.

Because Elastic Beanstalk orchestrates different AWS services, there is an additional way to interact. You can directly manipulate the assets (instances, ELBs, etc.) that Elastic Beanstalk manages. You can bypass, overwrite, and adjust everything that Elastic Beanstalk does.

## Developing with Eclipse

Our development environment and that of many Java developers includes the Eclipse IDE. Probably because of the popularity of this tool, Amazon developed an AWS Eclipse plugin —the AWS Toolkit for Eclipse. This plugin includes tools to help you work with Elastic Beanstalk among other AWS services. Basically, you can deploy your application from your local development environment directly onto Amazon AWS on Beanstalk.

We will show how to use Eclipse with the AWS Toolkit to launch both a Java application and a Scala-based application on Elastic Beanstalk.

### Setting Up Your Environment

The Beanstalk part of the AWS plugin for Eclipse requires the WTP (Web Tools Platform) plugins. The easiest way to set this up, then, is to download the WTP platform and then install the AWS plugin using the update site. That's fine for Java, but if you want to develop in Scala, read the following section.

#### Scala

For developing with Scala in Eclipse, there is a plugin called Scala IDE. It provides syntax highlighting, and so on. And it's supposed to require Eclipse classic, so we downloaded Eclipse Classic 2.6.2, and installed Scala IDE using the updates URL.

We then installed the WTP tools from the update URL and the AWS Toolkit.

## Configure your environment

Next, you will have to configure the AWS Toolkit to use your AWS credentials. Notice the AWS icon in your toolbar, and click the arrow to get the menu shown in Figure 3-1. Enter your AWS credentials as shown in Figure 3-2 and save.

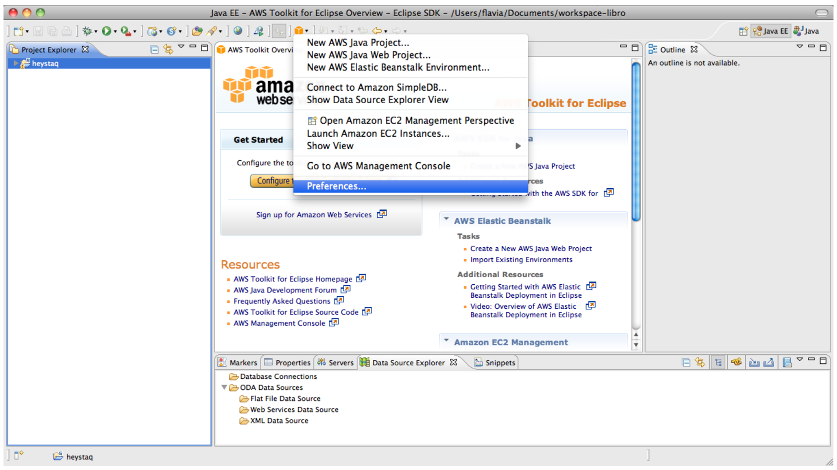

Java EE – AWS Toolkit for Eclipse Overview – Eclipse SDK – /Users/flavia/Documents/workspace-libro
New AWS Java Project...
New AWS Java Web Project...
New AWS Elastic Beanstalk Environment...
Connect to Amazon SimpleDB...
Show Data Source Explorer View
Open Amazon EC2 Management Perspective
Launch Amazon EC2 Instances...
Show View
Go to AWS Management Console
Preferences...

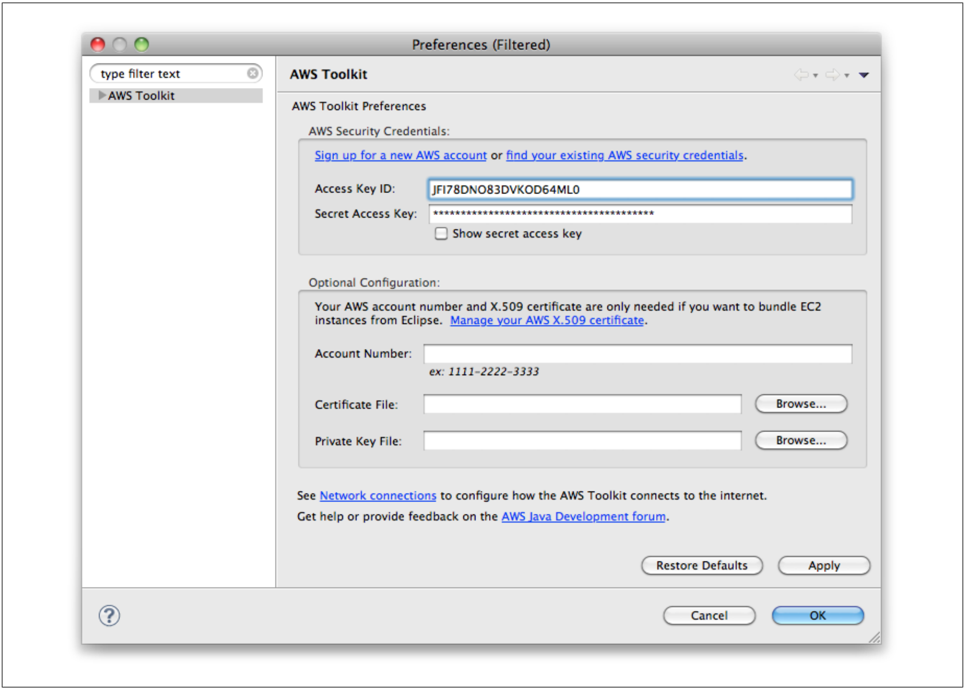
*Figure 3-1. AWS Toolkit preferences*

Create or import a Dynamic Web Project, and you are ready to launch your application on AWS.

## Deploying Your Application

Deploying our application on AWS will consist of creating a new Elastic Beanstalk environment. Right-click on your project, Run As, Run on Server. Define a new server, as shown in Figure 3-3. Elastic Beanstalk currently supports Tomcat 6 and 7. Give it a name or accept the given one, and go on to the next screen to configure the Elastic Beanstalk application and environment (Figure 3-4). If you have already created the application from the AWS console, you can just select it; otherwise create a new one. Give a name and description to the new environment and continue.

If you already have an environment on which you want to deploy a new version (for example, because you created it before in the AWS console), you can choose “Import an existing environment into the Servers view” and the plugin will retrieve all the existing environments.


Preferences (Filtered)

type filter text

AWS Toolkit

AWS Toolkit Preferences

AWS Security Credentials:

Sign up for a new AWS account or find your existing AWS security credentials.

Access Key ID: JFI78DNO83DVKOD64ML0

Secret Access Key: ****************************************

Show secret access key

Optional Configuration:

Your AWS account number and X.509 certificate are only needed if you want to bundle EC2 instances from Eclipse. Manage your AWS X.509 certificate.

Account Number:

ex: 1111-2222-3333

Certificate File:

Private Key File:

See Network connections to configure how the AWS Toolkit connects to the internet.

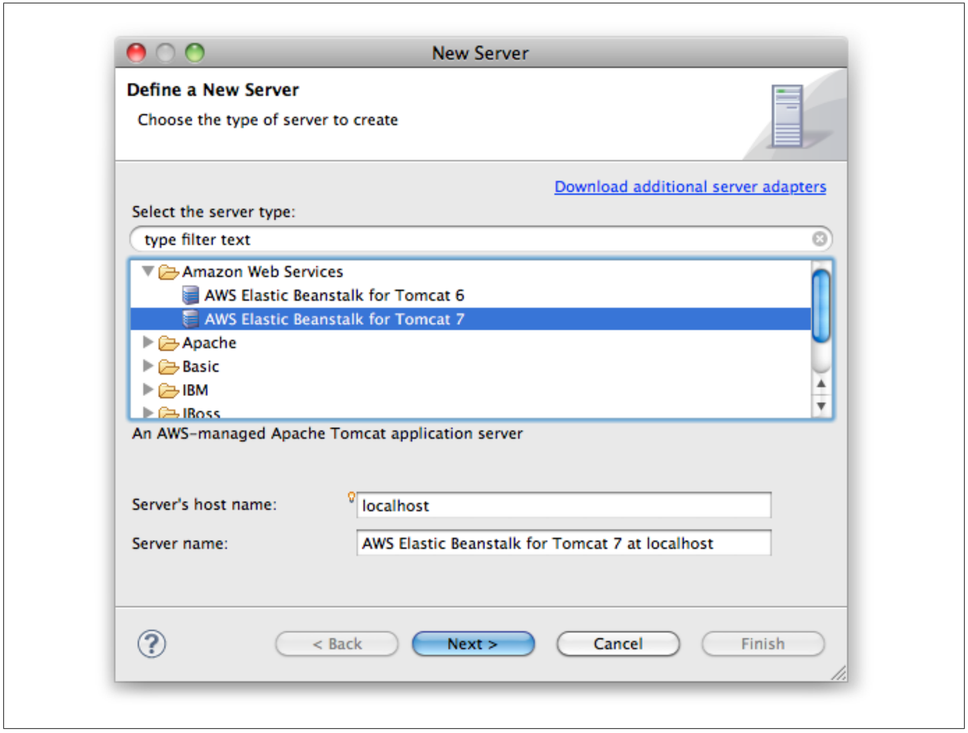
Get help or provide feedback on the AWS Java Development forum.

*Figure 3-2. Entering AWS credentials in Eclipse*

The next step is shown in Figure 3-5. If you will want to SSH to the server(s) you are deploying, you need to use a key pair. You can create one on the fly within this wizard, and the private part of the key will be stored in the local directory you indicate.

Be careful not to remove a key pair on this screen (it's very easy to do so without any warning or confirmation of what you are doing). In general, the plugin needs a bit of polishing to get rid of these kinds of user interface issues and bugs.

The *cname* will become part of the URL of your application. In this example, we chose "heystaq-flavia", so we will find our application in *http://heystaq-flavia.elasticbeanstalk.com*. The application health check URL will be used by Elastic Beanstalk to know if the deployment succeeded, and also to monitor and, if necessary, remove "sick" servers for which this check fails, and replace them with new ones. You will get notifications about the environment start, termination, restart, rebuild, etc. at the email address you provide.


New Server
Define a New Server
Choose the type of server to create
Download additional server adapters
Select the server type:
type filter text
Amazon Web Services
AWS Elastic Beanstalk for Tomcat 6
AWS Elastic Beanstalk for Tomcat 7
Apache
Basic
IBM
JBoss
< Back
Next >
Cancel
Finish

An AWS-managed Apache Tomcat application server

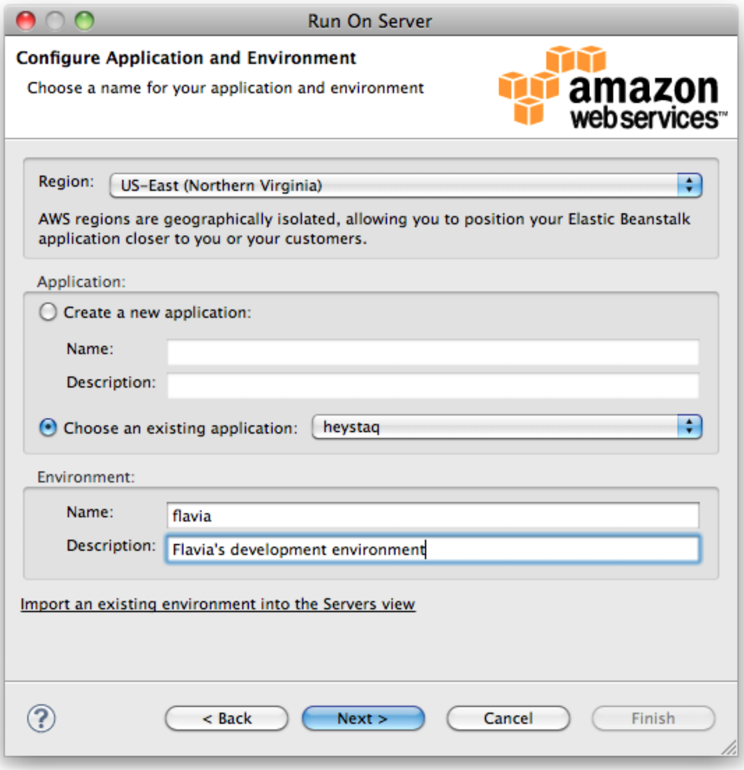
Server's host name: localhost

Server name: AWS Elastic Beanstalk for Tomcat 7 at localhost

*Figure 3-3. Define a new server in AWS*

In the final step, confirm that you want your project resources configured on the server (Figure 3-6). Choose a version for this WAR (Figure 3-7) and wait a few minutes to see the magic happen (Figure 3-8).

So what exactly happened? Eclipse compiled and built your application into a WAR, then made an API call to the Elastic Beanstalk service to create a new environment in the heystaq application and deploy the aforementioned WAR in that environment. This resulted in the creation of one EC2 instance (a micro instance, which is the default), which was added to a new Elastic Load Balancer, and Auto Scaling was configured to have a minimum of one and a maximum of four instances, depending on network utilization. You can see the full configuration in Eclipse (Figure 3-9).


Run On Server

Configure Application and Environment

Choose a name for your application and environment

Region: US-East (Northern Virginia)

AWS regions are geographically isolated, allowing you to position your Elastic Beanstalk application closer to you or your customers.

Application:

Create a new application:

Name:

Description:

[✓ Choose an existing application:] heystaq

Environment:

Name: flavia

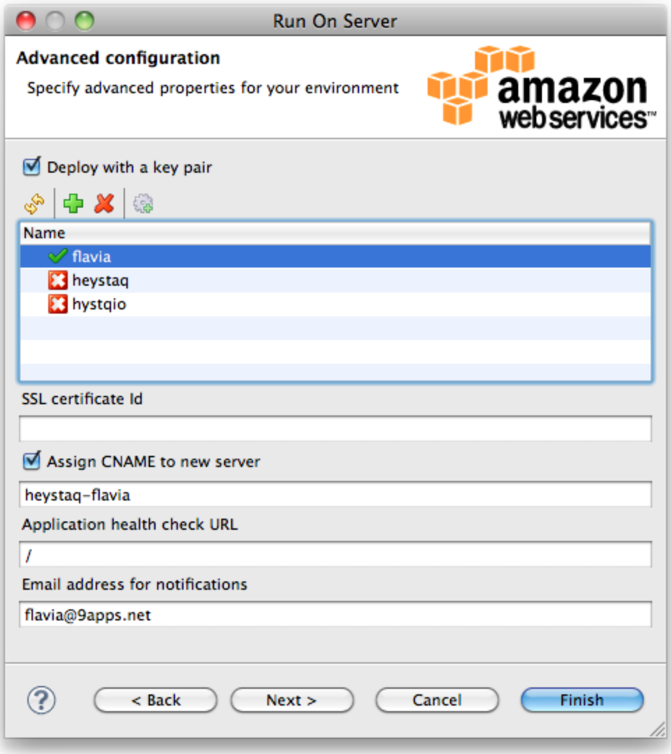
Description: Flavia's development environment

Import an existing environment into the Servers view

*Figure 3-4. Create an environment from Eclipse*

Once you've configured your environment, you can always see its status from the Servers view (Window → Show View → Other... → Server → Servers). There you will see your server listed, and you can open it to view and modify its configuration and show logs and recent events (Figure 3-10).

If for some reason your application fails to deploy, look at the Elastic Beanstalk events. If the problem seems to be in the application, you can look at the logs from Eclipse or the AWS console, or you can also SSH to the instance. (To make your life easier, you can always first make sure your application runs by deploying it on a local Tomcat server.)


Run On Server

Advanced configuration

Specify advanced properties for your environment

☑ Deploy with a key pair

| Name |
|---|
| ✓ flavia |
| heystaq |
| hystqio |

SSL certificate Id

☑ Assign CNAME to new server

heystaq-flavia

Application health check URL

/

Email address for notifications

flavia@9apps.net

*Figure 3-5. Environment configuration*

Another interesting thing you can do with the AWS plugin is show all the instances you have. If you launched on Beanstalk with a key pair for which you have the private key, you can open a terminal to it using Secure Shell directly from Eclipse (Figure 3-11). Or you can always do it manually, of course:

```
$ ssh -i .ssh/flavia.pem ec2-user@ec2-75-101-224-101.compute-1.amazonaws.com
```

Notice that the user name created for you is ec2-user.

When you are done with your environment, you can also terminate it from Eclipse, by right-clicking on the server and selecting Stop. The AWS plugin will save your environment configuration, even after terminating the environment.

In this way you will be able to start and stop a whole environment on AWS with your latest changes at any moment.

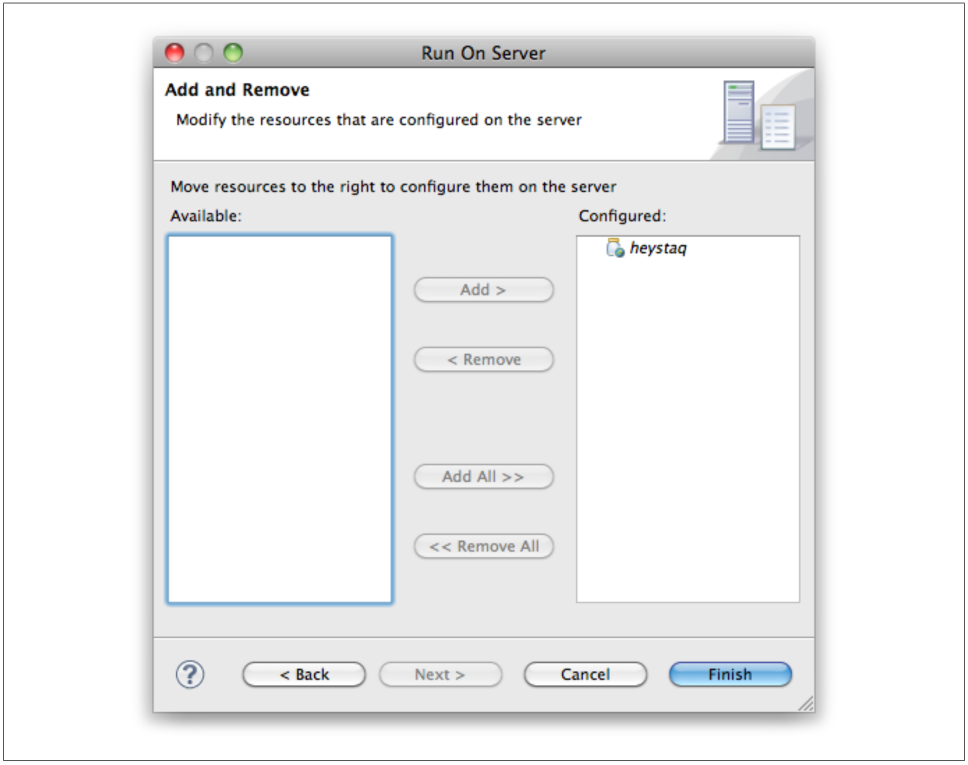
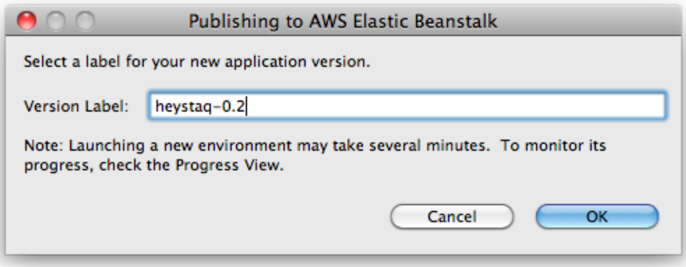


*Figure 3-6. Configure your project resources on your server*

## Continuous Integration with Hudson

When Extreme Programming methodology was created in 1999, one of the practices it recommended was *continuous integration*. Continuous integration aims at having frequent builds and unit testing of the software —preferably, on every commit to the code repository. Hudson is a tool for automating precisely these tasks, notifying and showing the results, and it is widely used in the Java community.

We can integrate Hudson with Elastic Beanstalk in such a way that whenever we have a successful build—one that compiles and where all the unit tests pass—we deploy it in a test environment. This test environment could be used to execute automated web testing, for example, or just for the development and test teams to be able to access the latest running version of the application.

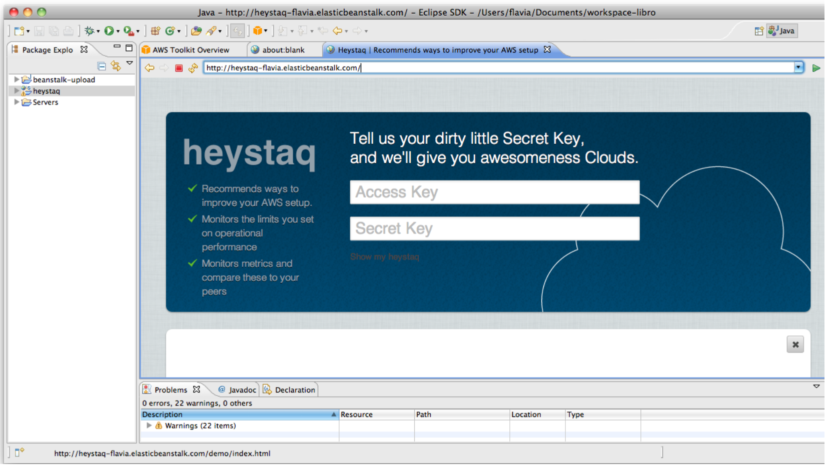

Publishing to AWS Elastic Beanstalk

Select a label for your new application version.

Version Label: heystaq-0.2

Note: Launching a new environment may take several minutes. To monitor its progress, check the Progress View.

*Figure 3-7. Give a version to your application*


Java – http://heystaq-flavia.elasticbeanstalk.com/ – Eclipse SDK – /Users/flavia/Documents/workspace-libro

http://heystaq-flavia.elasticbeanstalk.com/

heystaq

Tell us your dirty little Secret Key,
and we'll give you awesomeness Clouds.

- Recommends ways to improve your AWS setup.
- Monitors the limits you set on operational performance
- Monitors metrics and compare these to your peers

0 errors, 22 warnings, 0 others

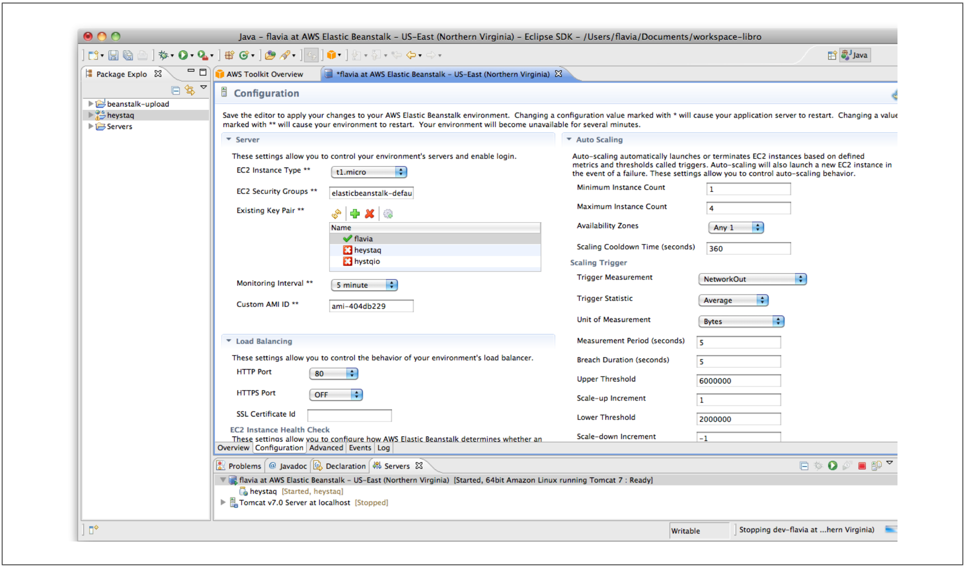
| Description | Resource | Path | Location | Type |
|---|---|---|---|---|
| Warnings (22 items) | | | | |

http://heystaq-flavia.elasticbeanstalk.com/demo/index.html

*Figure 3-8. Heystaq deployed from Eclipse*

In what follows we are going to show how to:

- Install Hudson on an EC2 instance.
- Create a small Java application that deploys WARs to Elastic Beanstalk.
- Combine both of the above to do “continuous deployments.”


Java – flavia at AWS Elastic Beanstalk – US-East (Northern Virginia) – Eclipse SDK – /Users/flavia/Documents/workspace-libro
Package Explo
beanstalk-upload
heystaq
Servers
AWS Toolkit Overview
*flavia at AWS Elastic Beanstalk – US-East (Northern Virginia)
Java
Overview
Configuration
Advanced
Events
Log
Problems
Javadoc
Declaration
Servers

Configuration

Save the editor to apply your changes to your AWS Elastic Beanstalk environment. Changing a configuration value marked with * will cause your application server to restart. Changing a value marked with ** will cause your environment to restart. Your environment will become unavailable for several minutes.

Server

These settings allow you to control your environment's servers and enable login.

| | |
|---|---|
| EC2 Instance Type ** | t1.micro |
| EC2 Security Groups ** | elasticbeanstalk-defau |
| Existing Key Pair ** | Name: ✓ flavia; heystaq; hystqio |
| Monitoring Interval ** | 5 minute |
| Custom AMI ID ** | ami-404db229 |

Load Balancing

These settings allow you to control the behavior of your environment's load balancer.

| | |
|---|---|
| HTTP Port | 80 |
| HTTPS Port | OFF |
| SSL Certificate Id | |

EC2 Instance Health Check

These settings allow you to configure how AWS Elastic Beanstalk determines whether an

Auto Scaling

Auto-scaling automatically launches or terminates EC2 instances based on defined metrics and thresholds called triggers. Auto-scaling will also launch a new EC2 instance in the event of a failure. These settings allow you to control auto-scaling behavior.

| | |
|---|---|
| Minimum Instance Count | 1 |
| Maximum Instance Count | 4 |
| Availability Zones | Any 1 |
| Scaling Cooldown Time (seconds) | 360 |

Scaling Trigger

| | |
|---|---|
| Trigger Measurement | NetworkOut |
| Trigger Statistic | Average |
| Unit of Measurement | Bytes |
| Measurement Period (seconds) | 5 |
| Breach Duration (seconds) | 5 |
| Upper Threshold | 6000000 |
| Scale-up Increment | 1 |
| Lower Threshold | 2000000 |
| Scale-down Increment | -1 |

```
flavia at AWS Elastic Beanstalk – US-East (Northern Virginia)  [Started, 64bit Amazon Linux running Tomcat 7 : Ready]
    heystaq  [Started, heystaq]
Tomcat v7.0 Server at localhost  [Stopped]
```

Writable | Stopping dev-flavia at ...hern Virginia)

*Figure 3-9. Viewing the environment configuration*

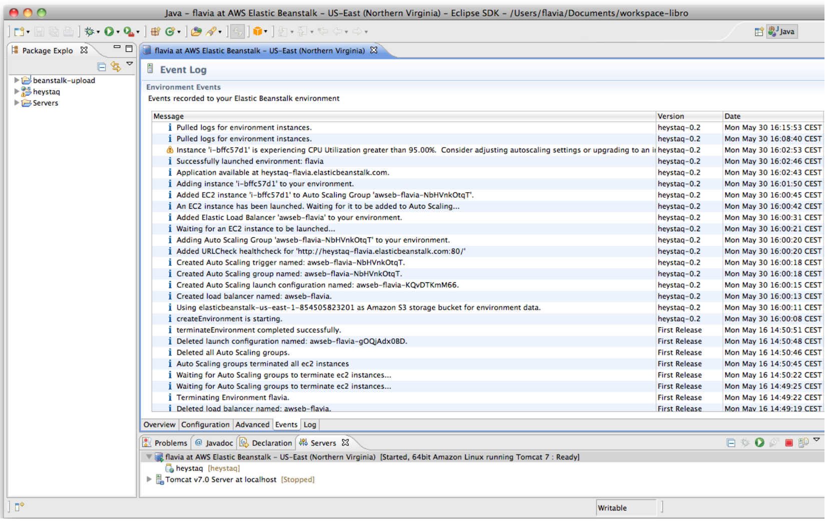
## Launch an EC2 Instance

Hudson is a Java application that runs in a servlet container like Tomcat, so we could also think of deploying it using Elastic Beanstalk. But this application requires a home directory to save all the job configurations and results, which means we can't use it as it is on Beanstalk. As we explained in Chapter 1, an instance should be "disposable." When it fails, for any reason, Elastic Beanstalk will detect it, and replace it with another one. If such a thing occurs, the storage in the instance is lost. So we are going to install Hudson on an EC2 instance without Elastic Beanstalk.

The first thing we need is to launch a new EC2 instance. Go to the Amazon AWS console, EC2 tab, and choose a region. We chose US-east-1 region to be close to the Elastic Beanstalk setup, but that is not necessary. You might want to have your Hudson application closer to the development team to have less latency when accessing the user interface.

Tap on "Launch Instance." We have to choose an AMI. We use Ubuntu, and we choose the Long Term Support 10.04 from Canonical, 32 bits: ami-3e02f257. (You can find the Ubuntu Canonical AMIs at *http://alestic.com*). From the Community AMIs we search for the AMI id and select it (Figure 3-12).

Now choose an instance type. A micro instance will probably be too slow (it only has 613 MB of memory and is better suited for short periodic CPU bursts; see *http://aws.amazon.com/ec2/instance-types/*), but you can always start small and scale up if necessary. You can give the instance a name tag, like "hudson", to recognize it later on, and use an existing key pair for which you have the private key, so you can log in to

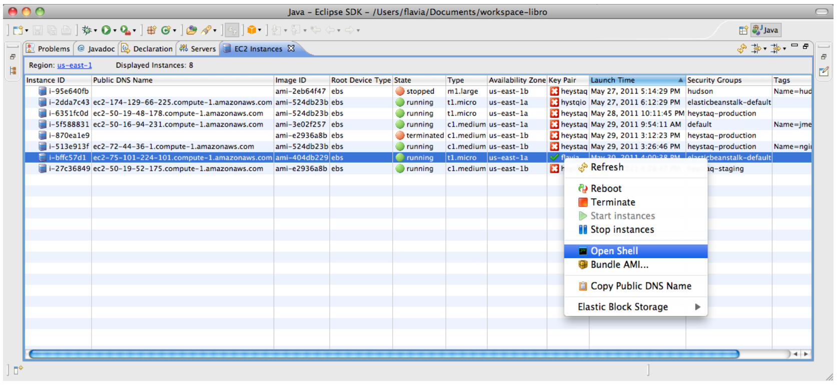

Java – flavia at AWS Elastic Beanstalk – US-East (Northern Virginia) – Eclipse SDK – /Users/flavia/Documents/workspace-libro
Package Explo
beanstalk-upload
heystaq
Servers
flavia at AWS Elastic Beanstalk – US-East (Northern Virginia)
Event Log
Environment Events

Events recorded to your Elastic Beanstalk environment

| Message | Version | Date |
|---|---|---|
| Pulled logs for environment instances. | heystaq-0.2 | Mon May 30 16:15:53 CEST |
| Pulled logs for environment instances. | heystaq-0.2 | Mon May 30 16:08:40 CEST |
| Instance 'i-bffc57d1' is experiencing CPU Utilization greater than 95.00%. Consider adjusting autoscaling settings or upgrading to an i | heystaq-0.2 | Mon May 30 16:02:53 CEST |
| Successfully launched environment: flavia | heystaq-0.2 | Mon May 30 16:02:46 CEST |
| Application available at heystaq-flavia.elasticbeanstalk.com. | heystaq-0.2 | Mon May 30 16:02:43 CEST |
| Adding instance 'i-bffc57d1' to your environment. | heystaq-0.2 | Mon May 30 16:01:50 CEST |
| Added EC2 instance 'i-bffc57d1' to Auto Scaling Group 'awseb-flavia-NbHVnkOtqT'. | heystaq-0.2 | Mon May 30 16:00:45 CEST |
| An EC2 instance has been launched. Waiting for it to be added to Auto Scaling... | heystaq-0.2 | Mon May 30 16:00:42 CEST |
| Added Elastic Load Balancer 'awseb-flavia' to your environment. | heystaq-0.2 | Mon May 30 16:00:31 CEST |
| Waiting for an EC2 instance to be launched... | heystaq-0.2 | Mon May 30 16:00:21 CEST |
| Adding Auto Scaling Group 'awseb-flavia-NbHVnkOtqT' to your environment. | heystaq-0.2 | Mon May 30 16:00:20 CEST |
| Added URLCheck healthcheck for 'http://heystaq-flavia.elasticbeanstalk.com:80/' | heystaq-0.2 | Mon May 30 16:00:20 CEST |
| Created Auto Scaling trigger named: awseb-flavia-NbHVnkOtqT. | heystaq-0.2 | Mon May 30 16:00:18 CEST |
| Created Auto Scaling group named: awseb-flavia-NbHVnkOtqT. | heystaq-0.2 | Mon May 30 16:00:18 CEST |
| Created Auto Scaling launch configuration named: awseb-flavia-KQvDTKmM66. | heystaq-0.2 | Mon May 30 16:00:15 CEST |
| Created load balancer named: awseb-flavia. | heystaq-0.2 | Mon May 30 16:00:13 CEST |
| Using elasticbeanstalk-us-east-1-854505823201 as Amazon S3 storage bucket for environment data. | heystaq-0.2 | Mon May 30 16:00:11 CEST |
| createEnvironment is starting. | heystaq-0.2 | Mon May 30 16:00:08 CEST |
| terminateEnvironment completed successfully. | First Release | Mon May 16 14:50:51 CEST |
| Deleted launch configuration named: awseb-flavia-gOQjAdx0BD. | First Release | Mon May 16 14:50:48 CEST |
| Deleted all Auto Scaling groups. | First Release | Mon May 16 14:50:46 CEST |
| Auto Scaling groups terminated all ec2 instances | First Release | Mon May 16 14:50:45 CEST |
| Waiting for Auto Scaling groups to terminate ec2 instances... | First Release | Mon May 16 14:50:22 CEST |
| Waiting for Auto Scaling groups to terminate ec2 instances... | First Release | Mon May 16 14:49:25 CEST |
| Terminating Environment flavia. | First Release | Mon May 16 14:49:22 CEST |
| Deleted load balancer named: awseb-flavia. | First Release | Mon May 16 14:49:19 CEST |

flavia at AWS Elastic Beanstalk – US-East (Northern Virginia) [Started, 64bit Amazon Linux running Tomcat 7 : Ready]
heystaq [heystaq]
Tomcat v7.0 Server at localhost [Stopped]

Figure 3-10. View events from Eclipse


Java – Eclipse SDK – /Users/flavia/Documents/workspace-libro
Problems
Javadoc
Declaration
Servers
EC2 Instances

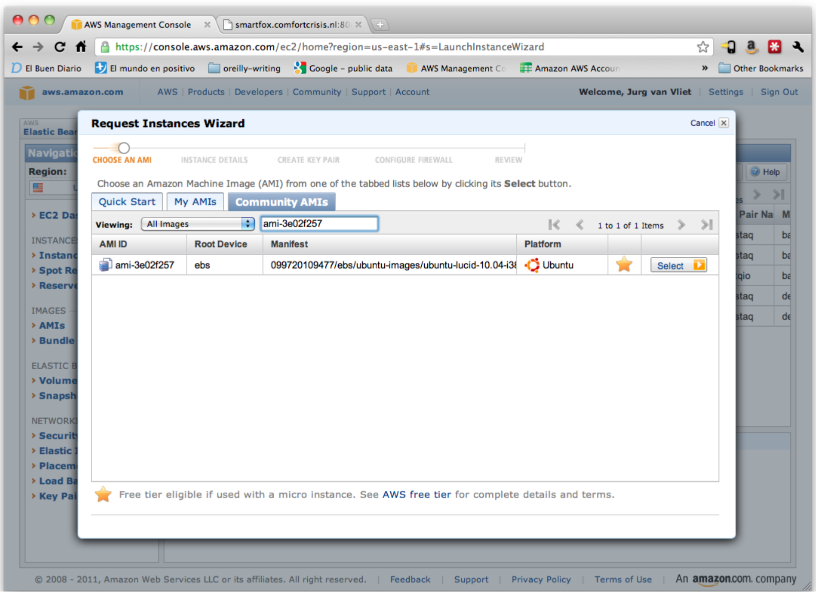
Region: us-east-1    Displayed Instances: 8

| Instance ID | Public DNS Name | Image ID | Root Device Type | State | Type | Availability Zone | Key Pair | Launch Time | Security Groups | Tags |
|---|---|---|---|---|---|---|---|---|---|---|
| i-95e640fb |  | ami-2eb64f47 | ebs | stopped | m1.large | us-east-1b | heystaq | May 27, 2011 5:14:29 PM | hudson | Name=hud |
| i-2dda7c43 | ec2-174-129-66-225.compute-1.amazonaws.com | ami-524db23b | ebs | running | t1.micro | us-east-1a | hystqio | May 27, 2011 6:12:29 PM | elasticbeanstalk-default |  |
| i-6351fc0d | ec2-50-19-48-178.compute-1.amazonaws.com | ami-524db23b | ebs | running | t1.micro | us-east-1a | heystaq | May 28, 2011 10:11:45 PM | heystaq-production |  |
| i-5f588831 | ec2-50-16-94-231.compute-1.amazonaws.com | ami-3e02f257 | ebs | running | c1.medium | us-east-1a | heystaq | May 29, 2011 9:54:11 AM | default | Name=jme |
| i-870ea1e9 |  | ami-e2936a8b | ebs | terminated | c1.medium | us-east-1b | heystaq | May 29, 2011 3:12:23 PM | heystaq-production |  |
| i-513e913f | ec2-72-44-36-1.compute-1.amazonaws.com | ami-524db23b | ebs | running | c1.medium | us-east-1b | heystaq | May 29, 2011 3:26:46 PM | heystaq-production | Name=ngi |
| i-bffc57d1 | ec2-75-101-224-101.compute-1.amazonaws.com | ami-404db229 | ebs | running | t1.micro | us-east-1a | flavia | May 30, 2011 4:00:38 PM | elasticbeanstalk-default |  |
| i-27c36849 | ec2-50-19-52-175.compute-1.amazonaws.com | ami-e2936a8b | ebs | running | c1.medium | us-east-1b | h | [illegible] | q-staging |  |

Figure 3-11. Listing EC2 instances from Eclipse

the instance later. If you don't have a key pair, create a new one and download the private key.

Create a security group for this instance—we called it "hudson"—so you can restrict access to it if necessary. The rest of the options you can leave at the defaults. While you wait for your instance to launch, modify your hudson security group to accept SSH connections and TCP connections to port 8080 from anywhere (0.0.0.0/0); see Figure 3-13. Hudson will be running on port 8080, and you will need to SSH to the instance to install it...which brings us to the next section.



Choose an Amazon Machine Image (AMI) from one of the tabbed lists below by clicking its **Select** button.

Viewing: All Images ami-3e02f257 1 to 1 of 1 Items

| AMI ID | Root Device | Manifest | Platform | | |
|---|---|---|---|---|---|
| ami-3e02f257 | ebs | 099720109477/ebs/ubuntu-images/ubuntu-lucid-10.04-i3 | Ubuntu | ★ | Select |

Free tier eligible if used with a micro instance. See AWS free tier for complete details and terms.

Figure 3-12. Launch an Ubuntu server instance

## Install Hudson

Now we can log in to the instance and install Hudson.

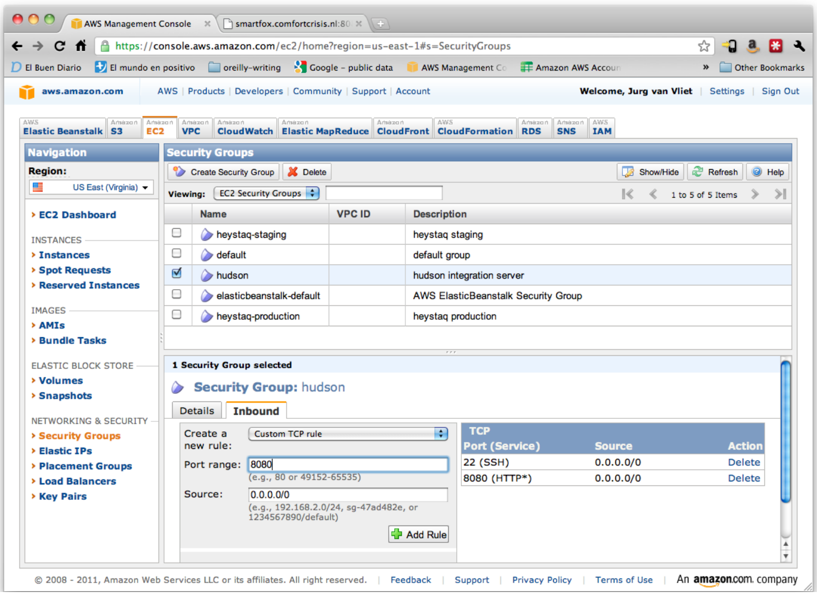
SSH using the key pair you provided to the instance at launch, the public DNS of the instance, and the user ubuntu. You should do something like this:

```
$ ssh -i heystaq.pem ubuntu@ec2-184-72-128-29.compute-1.amazonaws.com
```

Now install Hudson as a daemon (we follow the instructions here *http://wiki.hudson-ci.org/display/HUDSON/Installing+Hudson+on+Ubuntu*):

```
$ sudo su

$ echo "deb http://hudson-ci.org/debian binary/" > /etc/apt/sources.list.d/hudson.list

$ apt-get update

$ apt-get install hudson

$ exit
```

Now we can access Hudson at the public DNS in port 8080, in our case *http://ec2-184-72-128-29.compute-1.amazonaws.com:8080* (Figure 3-14).



| Name | VPC ID | Description |
|---|---|---|
| heystaq-staging | | heystaq staging |
| default | | default group |
| hudson | | hudson integration server |
| elasticbeanstalk-default | | AWS ElasticBeanstalk Security Group |
| heystaq-production | | heystaq production |

1 Security Group selected

Security Group: hudson

Details | Inbound

Create a new rule: Custom TCP rule

Port range: 8080 (e.g., 80 or 49152-65535)

Source: 0.0.0.0/0 (e.g., 192.168.2.0/24, sg-47ad482e, or 1234567890/default)

Add Rule

| TCP Port (Service) | Source | Action |
|---|---|---|
| 22 (SSH) | 0.0.0.0/0 | Delete |
| 8080 (HTTP*) | 0.0.0.0/0 | Delete |

© 2008 - 2011, Amazon Web Services LLC or its affiliates. All right reserved. | Feedback | Support | Privacy Policy | Terms of Use | An amazon.com company

*Figure 3-13. Modifying the security group for the Hudson server*

## Install Hudson Plugins

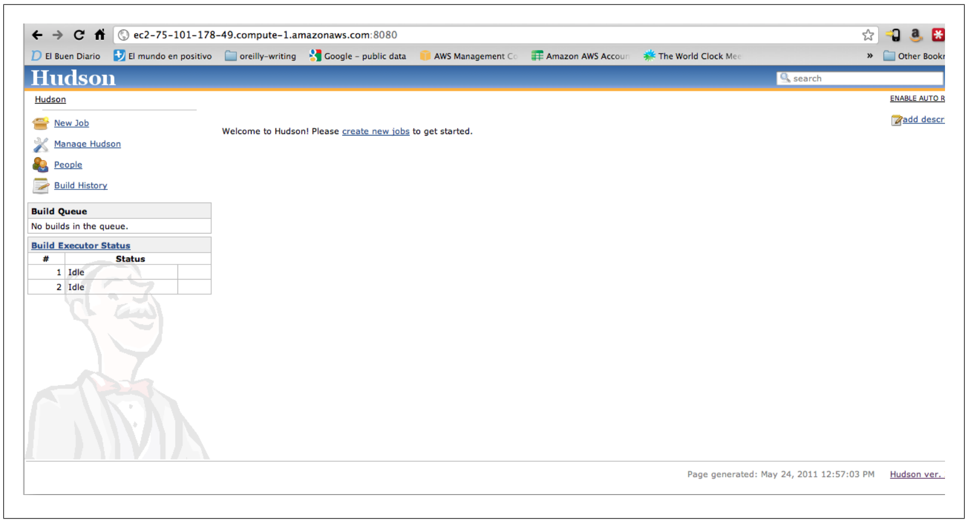
We are going to need some Hudson plugins to set up our example. We have our sources in a GitHub repository, and git is not yet supported by default, so we will need a plugin. We will be using the Maven plugin to do the builds, and we will need a third plugin that can run a script after a successful Maven build.

For setting up the git plugin, we go to Manage Hudson → Manage Plugins, and in the list of available plugins, we find the Hudson Git Plugin. Select it and click Install.

We are going to need a private key for git. SSH to your Hudson EC2 instance, sudo as the Hudson user, and create the SSH key without a passphrase following the instructions:

```
$ sudo -u hudson -i

$ cd ~/.ssh

$ ssh-keygen -t rsa -C "your_email@youremail.com"
```

This created the private and public parts of an SSH key in *~/.ssh/id_rsa* and *~/.ssh/id_rsa.pub*. We then copy the content of the public key and add it to the GitHub repository as a deployment key (Admin → Deploy Keys → Add another deploy key, as shown in *http://help.github.com/deploy-keys/*).



Welcome to Hudson! Please create new jobs to get started.

Build Queue

No builds in the queue.

Build Executor Status

| # | Status |
|---|---|
| 1 | Idle |
| 2 | Idle |

Page generated: May 24, 2011 12:57:03 PM

Figure 3-14. Hudson installed

We also need to install git on the server:

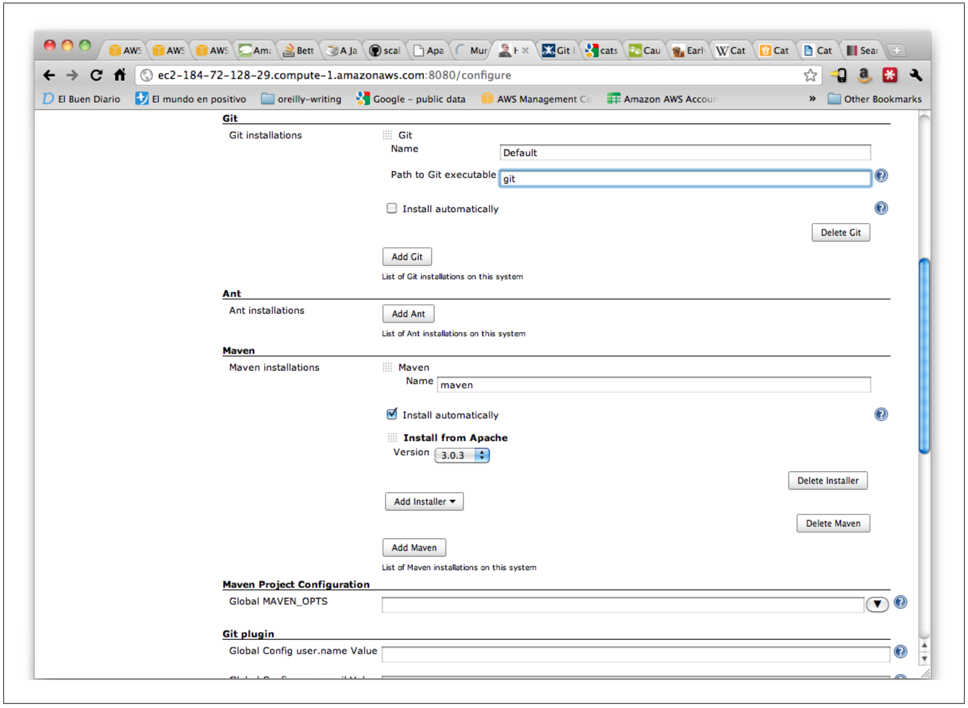
```
$ sudo apt-get -y install git-core gitosis

$ git config --global user.email "your_email@youremail.com"

$ git config --global user.name "Hudson deployment"
```

You can test the git installation by sudo-ing as the Hudson user and trying to clone your repository, like we do here:

```
$ sudo -u hudson -i

$ cd /tmp

$ git clone git@github.com:flavia/heystaq.git
```

Finally, go to your Hudson user interface, click on Manage Hudson → Configure System. There, make sure that the value for "Path to Git executable" is "git". For Maven, click on "Install automatically." We are using version 3.0.3 of Maven. (Figure 3-15)

The third plugin that we will need is called M2 Extra Steps Plugin (*http://wiki.hudson-ci.org/display/HUDSON/M2+Extra+Steps+Plugin*). Even though it doesn't have any more support from the developer, it proved to do what we needed, which is to execute a script after a successful Maven build. Install from Manage Hudson → Manage Plugins page.


ec2-184-72-128-29.compute-1.amazonaws.com:8080/configure

**Git**

Git installations

Git
Name: Default
Path to Git executable: git
☐ Install automatically

List of Git installations on this system

**Ant**

Ant installations

List of Ant installations on this system

**Maven**

Maven installations

Maven
Name: maven
☑ Install automatically
**Install from Apache**
Version: 3.0.3

List of Maven installations on this system

**Maven Project Configuration**

Global MAVEN_OPTS

**Git plugin**

Global Config user.name Value

*Figure 3-15. Configuring Git and Maven in Hudson*

## A Java App to Deploy a WAR to Beanstalk

Being a book about deploying JVM-based applications, we chose Java to write this one. Apart from that, the official AWS Java libraries—or AWS SDK for Java—are very complete and up to date.

We built this small application using Eclipse, creating an "AWS Java project," which is basically a Java project that includes the AWS Java libraries and its dependencies, and creates a properties file with the AWS credentials.

Our class receives a WAR file and does three things:

- Uploads the file to S3.
- Creates a new application version in Elastic Beanstalk pointing to this WAR.
- Deploys this version to a Beanstalk environment.

The `DeployNewVersion` class follows:

```
/**
 * This class deploys a given war as a new version in the Heystaq test environment.
 *
 * It will first upload the given war to S3,
 * then add it to Elastic Beanstalk as a new version with the current date,
 * and finally deploy it in the Elastic Beanstalk "test" environment.
 *
 */
public class DeployNewVersion {

private static final Log log = LogFactory.getLog(DeployNewVersion.class);

  private AWSCredentials credentials;

  private Date now;
  private String versionLabel;
  private String bucketName;
  private String warName;

  private Upload upload;
  private AWSElasticBeanstalk beanstalk;

  public DeployNewVersion() throws Exception {

    now = new Date();

    // Get credentials from the properties file
    credentials = new PropertiesCredentials(DeployNewVersion.class
                .getResourceAsStream("AwsCredentials.properties"));

    // the version label will be of the form "heystaq-test_20110520T1450"
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyyMMdd'T'HHmm");
    versionLabel = "heystaq-test_" + dateFormat.format(now);

    // the S3 bucket where we upload the wars is called "hudson-wars"
    bucketName = "hudson-wars";

    // the name that we give to the S3 resource containing the war is
"heystaq-test_20110520T1450.war"
    warName = versionLabel + ".war";

    beanstalk = new AWSElasticBeanstalkClient(credentials);

  }

  /**
   *
   * @param fileName the file location of the war to upload
   */
  public void execute(String fileName) {

    // first upload the war to S3
        File warToUpload = new File(fileName);
        if (!warToUpload.exists()) {
          System.out.println("Sorry, the file "+fileName+" doesn't exist");
          System.exit(1);
        }

        // A ProgressListener will receive updates about the uploading
        // of the file, and we can know when it is finished
        ProgressListener progressListener = new ProgressListener() {
            public void progressChanged(ProgressEvent progressEvent) {
                if (upload == null) return;

                System.out.print(".");

                switch (progressEvent.getEventCode()) {
                case ProgressEvent.COMPLETED_EVENT_CODE:
                    // done uploading war! now create the new version in Beanstalk
                  System.out.println();
                  System.out.println("War has been uploaded.");
                createApplicationVersion();
                deployVersion();
                    break;
                case ProgressEvent.FAILED_EVENT_CODE:
                    try {
                        AmazonClientException e = upload.waitForException();
                        System.out.println("Unable to upload file to Amazon S3: "
+ e.getMessage());
                    } catch (InterruptedException e) {}
                    break;
                }
            }
        };

        TransferManager transferManager = new TransferManager(credentials);

        PutObjectRequest request = new PutObjectRequest(
                bucketName, warName, warToUpload)
            .withProgressListener(progressListener);

        // create the bucket if it doesn't already exist
        createS3Bucket(transferManager);

        System.out.println("Uploading war to S3...");
        upload = transferManager.upload(request);

  }

  /**
   * Create an Elastic Beanstalk application version for this war,
   * indicating its location in S3.
   */
  private void createApplicationVersion() {
    SimpleDateFormat friendlyDateFormat = new SimpleDateFormat("dd MMM yyy HH:mm");

    CreateApplicationVersionRequest request =
      new CreateApplicationVersionRequest("heystaq", versionLabel);
    request.setDescription("War created by Hudson maven on " + friendlyDateFormat.
format(now));
    request.setSourceBundle(new S3Location(bucketName, warName));

    CreateApplicationVersionResult result = beanstalk.createApplicationVersion
(request);
```

```
    System.out.println();
    System.out.println("Version " + result.getApplicationVersion().getVersionLabel()
+ " has been created.");
    System.out.println("Version details: " + result.getApplicationVersion());

  }

  /**
   * Deploy the newly created version to the test environment.
   */
  private void deployVersion() {

    UpdateEnvironmentRequest request = new UpdateEnvironmentRequest();
    request.setEnvironmentName("test");
    request.setVersionLabel(versionLabel);

    UpdateEnvironmentResult result = beanstalk.updateEnvironment(request);
    System.out.println();
    System.out.println("SUCCESS! Version deploy requested. Test environment is now "
+ result.getStatus());

    System.exit(0);
  }

    private void createS3Bucket(TransferManager transferManager) {
        try {
            if (transferManager.getAmazonS3Client().doesBucketExist(bucketName)
== false) {
                transferManager.getAmazonS3Client().createBucket(bucketName);
            }
        } catch (AmazonClientException ace) {
            System.out.println("Unable to create a new Amazon S3 bucket: " +
ace.getMessage());
        }
    }

  /**
   * @param args one argument containing the location of the war to deploy
   */
  public static void main(String[] args) throws Exception {

    if (args.length < 1) {
      System.out.println("Required argument: location of war to upload");
      System.exit(1);
    }

    new DeployNewVersion().execute(args[0]);

  }
}
```

You can, of course, reuse this class by using the names of your application, environment, and bucket, and, of course, using your own credentials in the properties file:

```
#Insert your AWS Credentials from http://aws.amazon.com/security-credentials
#Thu May 26 15:36:10 CEST 2011
accessKey=AKIAIYBIJOUYDEMQW74A
secretKey=Hg4unEkgQwDMWNo+wqujnEV4yUYwx7nUOkUxwH7t
```

We tried to simplify the class as much as possible to focus on the Elastic Beanstalk API calls. You can find a better version of it that you can use as a library in our public repository.

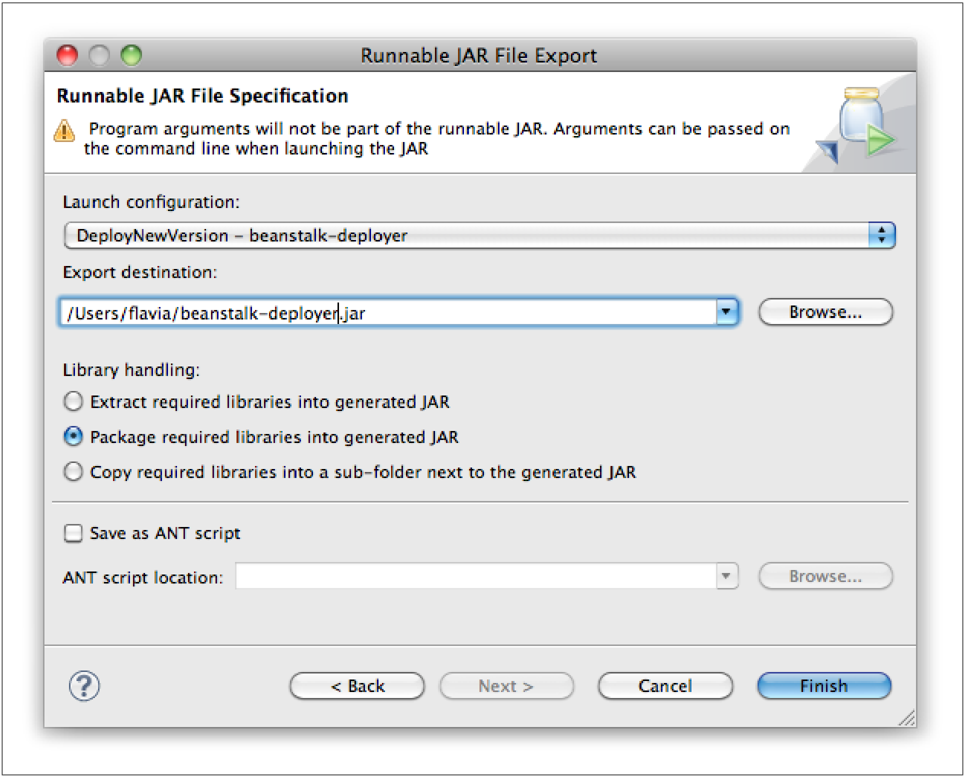
If you use a different region than us-east-1 (the "US Standard" region, used by default), you will need to change the endpoint URL of your services by calling, for example `TransferManager.getAmazonS3Client().setEndpoint("s3-eu-west-1.amazonaws.com")`. Elastic Beanstalk at the moment only supports us-east-1 region. The full list of endpoints is listed in the Amazon AWS website.

Ideally you could use IAM to create a user with just enough rights to run this script (i.e., rights to the specific bucket) and to create and deploy versions in your application. Unfortunately, at the moment of writing, this is still not possible, since Elastic Beanstalk is still not integrated with IAM, so any IAM-generated credentials won't work on Beanstalk.

To create a standalone application, you can export it as a runnable jar from Eclipse, including all the needed jars (Figure 3-16).

Take into account that the deployer doesn't create the environment if it doesn't exist, so you will need to create the environment in the AWS console before running it (or modify the code to create the environment). You can then execute it in this way:

```
$ java -jar beanstalk-deployer.jar ~/heystaq/target/heystaq-0.0.1-SNAPSHOT.war
```


Runnable JAR File Specification

Program arguments will not be part of the runnable JAR. Arguments can be passed on the command line when launching the JAR

Launch configuration: DeployNewVersion – beanstalk-deployer

Export destination: /Users/flavia/beanstalk-deployer.jar

Library handling:

( ) Extract required libraries into generated JAR

[✓ Package required libraries into generated JAR]

( ) Copy required libraries into a sub-folder next to the generated JAR

[ ] Save as ANT script

ANT script location:

*Figure 3-16. Exporting as a runnable jar*

## Create a Hudson Job

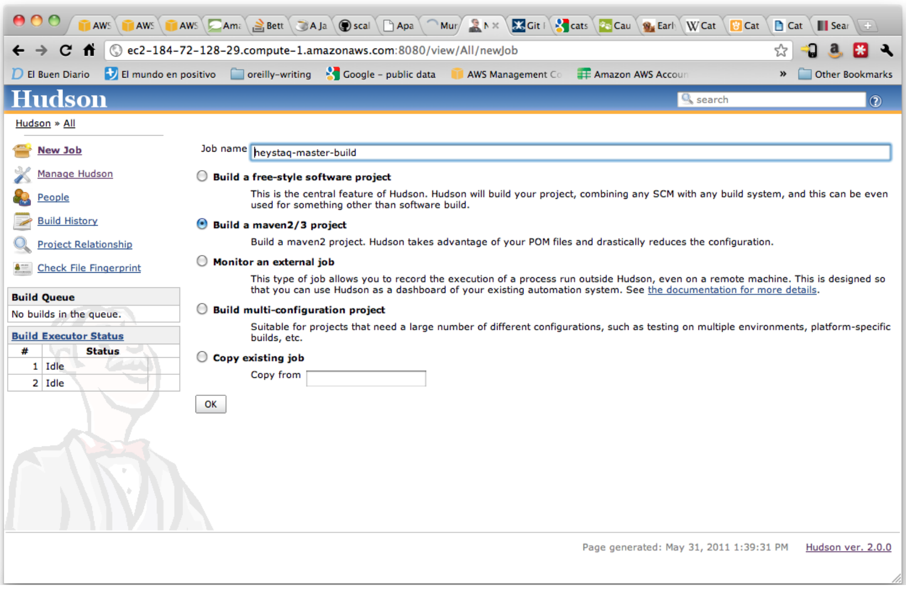
Create a new Hudson job, and make it a "maven 2/3 type project" (Figure 3-17). We named it "heystaq-master-build" because we are going to be building the master branch of the heystaq project.

We choose Git for source code management and give the URL of our repository, `git@github.com:flavia/heystaq.git`, and master as the branch to build. Fill in your "Goals and options," for example with `clean test package`.

The other thing we need to configure is the "M2 Extra Build Steps." We add a "post-build step" and we choose to execute shell, only if the build succeeds. And what we run is:

```
java -jar deploy-version.jar \
   /var/lib/hudson/jobs/heystaq-master-build/lastSuccessful/com.heystaq\$heystaq/
archive/com.heystaq/heystaq/0.0.1-SNAPSHOT/heystaq-0.0.1-SNAPSHOT.war
```

This is shown in Figure 3-18.



New Job
Manage Hudson
People
Build History
Project Relationship
Check File Fingerprint

**Build Queue**
No builds in the queue.

**Build Executor Status**

| # | Status |
|---|---|
| 1 | Idle |
| 2 | Idle |

Job name: heystaq-master-build

( ) **Build a free-style software project**
This is the central feature of Hudson. Hudson will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

[✓ **Build a maven2/3 project**]
Build a maven2 project. Hudson takes advantage of your POM files and drastically reduces the configuration.

( ) **Monitor an external job**
This type of job allows you to record the execution of a process run outside Hudson, even on a remote machine. This is designed so that you can use Hudson as a dashboard of your existing automation system. See the documentation for more details.

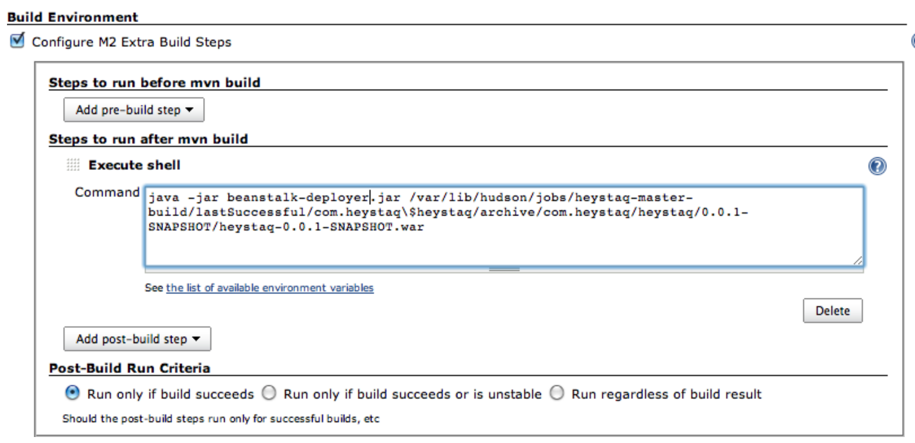
( ) **Build multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

( ) **Copy existing job**
Copy from

OK

Page generated: May 31, 2011 1:39:31 PM Hudson ver. 2.0.0

*Figure 3-17. Create a Hudson Maven job*



**Build Environment**

[✓] Configure M2 Extra Build Steps

**Steps to run before mvn build**

Add pre-build step ▾

**Steps to run after mvn build**

**Execute shell**

Command
```
java -jar beanstalk-deployer.jar /var/lib/hudson/jobs/heystaq-master-
build/lastSuccessful/com.heystaq\$heystaq/archive/com.heystaq/heystaq/0.0.1-
SNAPSHOT/heystaq-0.0.1-SNAPSHOT.war
```

See the list of available environment variables

Delete

Add post-build step ▾

**Post-Build Run Criteria**

[✓ Run only if build succeeds] Run only if build succeeds or is unstable Run regardless of build result

Should the post-build steps run only for successful builds, etc

*Figure 3-18. Configure the Hudson job to deploy the WAR after build*

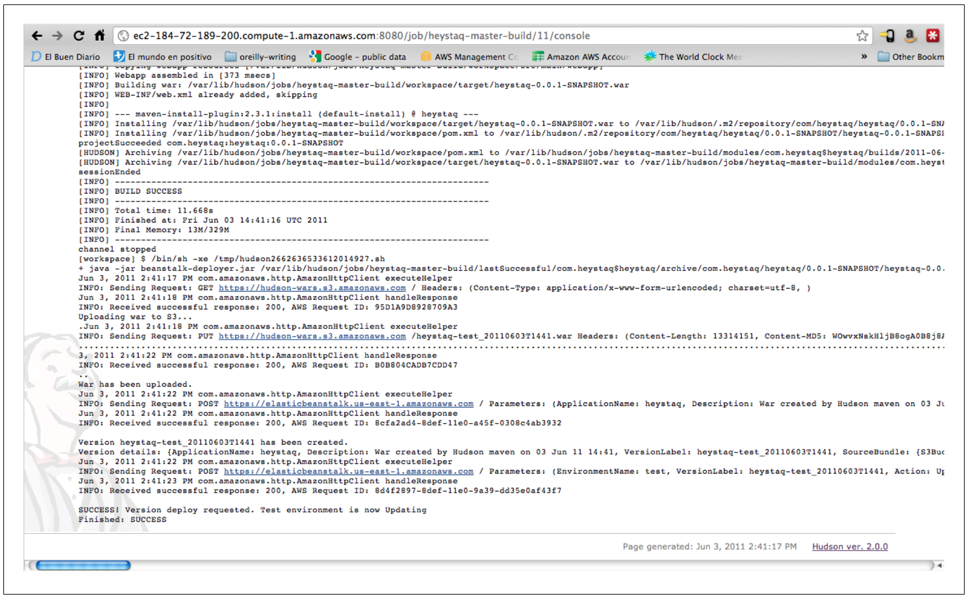
Since we are going to need to call the *beanstalk-deployer.jar* from the Hudson instance, we need to upload it there first, of course:

```
$ scp -i heystaq.pem beanstalk-deployer.jar ubuntu@ec2-184-72-128-29.compute-1.
amazonaws.com:/home/ubuntu
```

Then SSH to the instance and move the *jar* to the job workspace directory:

```
$ sudo mv /home/ubuntu/beanstalk-deployer.jar /var/lib/hudson/jobs/heystaq-master-
build/workspace/
```

Save the job, and we are ready to run! Go to the job, click on Build Now, and check the Console Output of the build (Figure 3-19 and Figure 3-20).



```
[INFO] Webapp assembled in [373 msecs]
[INFO] Building war: /var/lib/hudson/jobs/heystaq-master-build/workspace/target/heystaq-0.0.1-SNAPSHOT.war
[INFO] WEB-INF/web.xml already added, skipping
[INFO]
[INFO] --- maven-install-plugin:2.3.1:install (default-install) @ heystaq ---
[INFO] Installing /var/lib/hudson/jobs/heystaq-master-build/workspace/target/heystaq-0.0.1-SNAPSHOT.war to /var/lib/hudson/.m2/repository/com/heystaq/heystaq/0.0.1-SNA
[INFO] Installing /var/lib/hudson/jobs/heystaq-master-build/workspace/pom.xml to /var/lib/hudson/.m2/repository/com/heystaq/heystaq/0.0.1-SNAPSHOT/heystaq-0.0.1-SNAPSH
projectSucceeded com.heystaq:heystaq:0.0.1-SNAPSHOT
[HUDSON] Archiving /var/lib/hudson/jobs/heystaq-master-build/workspace/pom.xml to /var/lib/hudson/jobs/heystaq-master-build/modules/com.heystaq$heystaq/builds/2011-06-
[HUDSON] Archiving /var/lib/hudson/jobs/heystaq-master-build/workspace/target/heystaq-0.0.1-SNAPSHOT.war to /var/lib/hudson/jobs/heystaq-master-build/modules/com.heyst
sessionEnded
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 11.668s
[INFO] Finished at: Fri Jun 03 14:41:16 UTC 2011
[INFO] Final Memory: 13M/329M
[INFO] ------------------------------------------------------------------------
channel stopped
[workspace] $ /bin/sh -xe /tmp/hudson2662636533612014927.sh
+ java -jar beanstalk-deployer.jar /var/lib/hudson/jobs/heystaq-master-build/lastSuccessful/com.heystaq$heystaq/archive/com.heystaq/heystaq/0.0.1-SNAPSHOT/heystaq-0.0.
Jun 3, 2011 2:41:17 PM com.amazonaws.http.AmazonHttpClient executeHelper
INFO: Sending Request: GET https://hudson-wars.s3.amazonaws.com / Headers: (Content-Type: application/x-www-form-urlencoded; charset=utf-8, )
Jun 3, 2011 2:41:18 PM com.amazonaws.http.AmazonHttpClient handleResponse
INFO: Received successful response: 200, AWS Request ID: 95D1A9D8928709A3
Uploading war to S3...
.Jun 3, 2011 2:41:18 PM com.amazonaws.http.AmazonHttpClient executeHelper
INFO: Sending Request: PUT https://hudson-wars.s3.amazonaws.com /heystaq-test_20110603T1441.war Headers: (Content-Length: 13314151, Content-MD5: WOwvxNskHljB8ogA0B8j8
...................................................................................................................................
3, 2011 2:41:22 PM com.amazonaws.http.AmazonHttpClient handleResponse
INFO: Received successful response: 200, AWS Request ID: B0B804CADB7CDD47
..
War has been uploaded.
Jun 3, 2011 2:41:22 PM com.amazonaws.http.AmazonHttpClient executeHelper
INFO: Sending Request: POST https://elasticbeanstalk.us-east-1.amazonaws.com / Parameters: (ApplicationName: heystaq, Description: War created by Hudson maven on 03 Ju
Jun 3, 2011 2:41:22 PM com.amazonaws.http.AmazonHttpClient handleResponse
INFO: Received successful response: 200, AWS Request ID: 8cfa2ad4-8def-11e0-a45f-0308c4ab3932

Version heystaq-test_20110603T1441 has been created.
Version details: {ApplicationName: heystaq, Description: War created by Hudson maven on 03 Jun 11 14:41, VersionLabel: heystaq-test_20110603T1441, SourceBundle: {S3Buc
Jun 3, 2011 2:41:22 PM com.amazonaws.http.AmazonHttpClient executeHelper
INFO: Sending Request: POST https://elasticbeanstalk.us-east-1.amazonaws.com / Parameters: (EnvironmentName: test, VersionLabel: heystaq-test_20110603T1441, Action: Up
Jun 3, 2011 2:41:23 PM com.amazonaws.http.AmazonHttpClient handleResponse
INFO: Received successful response: 200, AWS Request ID: 8d4f2897-8def-11e0-9a39-dd35e0af43f7

SUCCESS! Version deploy requested. Test environment is now Updating
Finished: SUCCESS
```

Page generated: Jun 3, 2011 2:41:17 PM    Hudson ver. 2.0.0

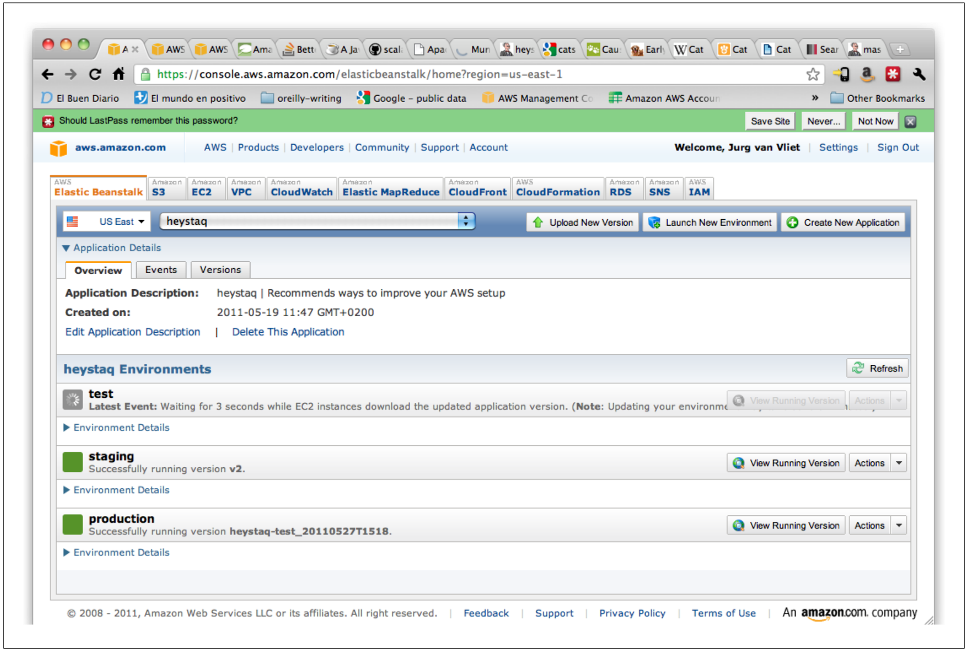
*Figure 3-19. The job's output shows that the version has been deployed*

## Create an Image

Before you run happily away from your computer because your Hudson server is running and deploying stuff on Beanstalk, do one last thing: create an AMI for your instance.

Realize that ultimately, an EC2 instance runs on physical hardware, and hardware can fail. If your server's disk suddenly breaks down, you will need to repeat all the above steps, unless you make an image from which to launch an instance again. This is very easy: just go to the AWS console, to the EC2 tab, and select your instance. Create Image is one of the Instance actions (Figure 3-21). It's advisable to first stop the instance, and then start making the AMI.

Give it a name that will help you identify it—we normally use the instance tag name, an indication of the operating system, and the date. In this case we created `hudson_(32bit_Ubuntu)_20110531`.



Should LastPass remember this password?

Welcome, Jurg van Vliet

heystaq

**Application Description:** heystaq | Recommends ways to improve your AWS setup

**Created on:** 2011-05-19 11:47 GMT+0200

Edit Application Description | Delete This Application

**heystaq Environments**

**test**
**Latest Event:** Waiting for 3 seconds while EC2 instances download the updated application version. (**Note:** Updating your environme

**staging**
Successfully running version **v2**.

**production**
Successfully running version **heystaq-test_20110527T1518**.

© 2008 - 2011, Amazon Web Services LLC or its affiliates. All right reserved.

*Figure 3-20. Test environment deploying a new version*

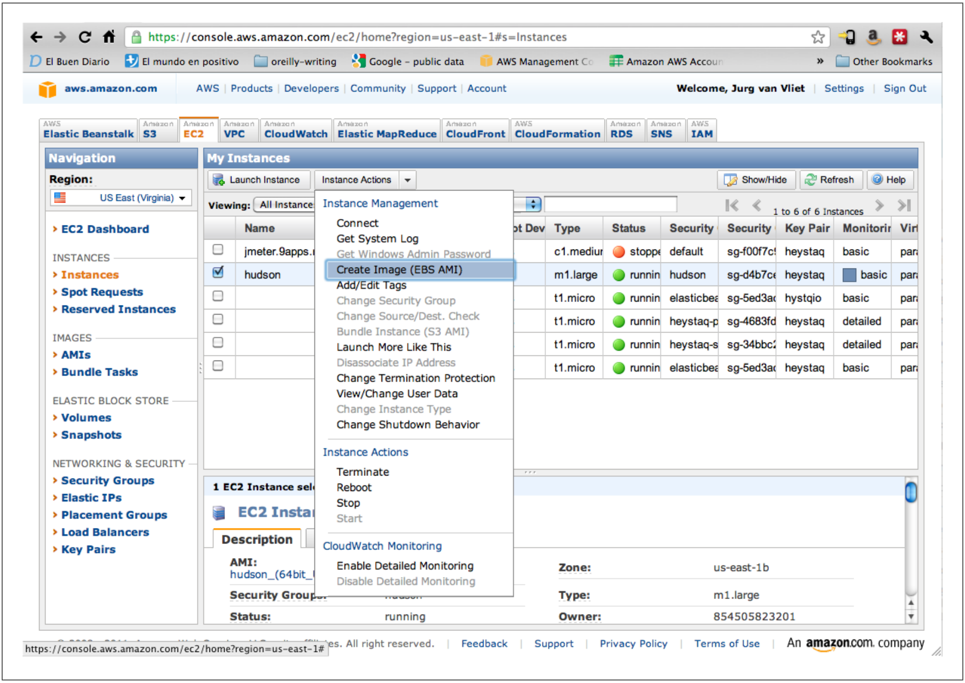
To complete this setup, you might configure regular backups of the Hudson home directory if you want to keep the history of the builds and archives. The simplest way to do this would be to create images (AMIs) of the instance regularly. Another solution would be to use an EBS volume for the Hudson home directory, and make regular backups of it using S3 *snapshots*. This is explained in detail in Chapter 2 of Programming Amazon EC2. You can find a free sample of the first chapters at http://oreilly.com/catalog/0636920013228.

## Setting Up Production

Unlike the other environments, the production environment is supposed to always be there. Depending on the application and the traffic pattern, you have to make several choices and adjust settings for security, performance, or functionality. Most of these configuration settings can be done through the Console, but we will also need the command-line tools.

### Server (EC2 Instances)

In general, when running an Auto Scaling group behind an ELB (Elastic Load Balancer), you have to balance the number of instances with the number of instances behind the ELB. This is somewhat vague, as there is no way to easily verify the number of instances



| | Name | ot Dev | Type | Status | Security | Security | Key Pair | Monitorir | Vir |
|---|---|---|---|---|---|---|---|---|---|
| ☐ | jmeter.9apps. | | c1.mediur | ● stoppe | default | sg-f00f7c! | heystaq | basic | para |
| ☑ | hudson | | m1.large | ● runnin | hudson | sg-d4b7ce | heystaq | ▨ basic | para |
| ☐ | | | t1.micro | ● runnin | elasticbea | sg-5ed3ac | hystqio | basic | para |
| ☐ | | | t1.micro | ● runnin | heystaq-p | sg-4683fd | heystaq | detailed | para |
| ☐ | | | t1.micro | ● runnin | heystaq-s | sg-34bbc2 | heystaq | detailed | para |
| ☐ | | | t1.micro | ● runnin | elasticbea | sg-5ed3ac | heystaq | basic | para |

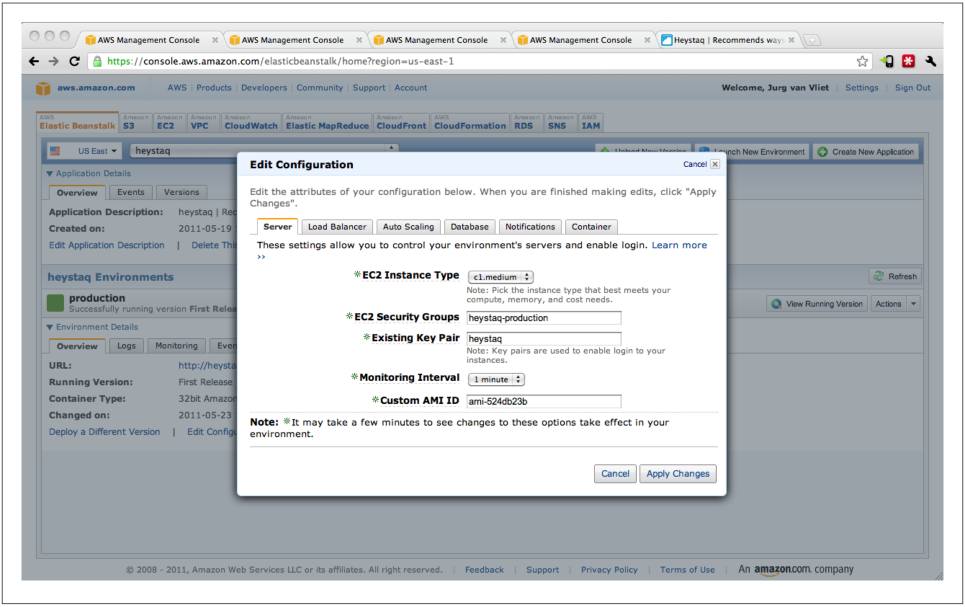
| | | | |
|---|---|---|---|
| AMI: hudson_(64bit_ | | Zone: | us-east-1b |
| Security Group: | hudson | Type: | m1.large |
| Status: | running | Owner: | 854505823201 |

Figure 3-21. Create an AMI in the AWS console

the ELB uses internally. But, in general, we choose the smallest instance that provides good performance. Usually we start with the instance type c1.medium for web apps. (See Figure 3-22.)

As you have probably seen in the Beanstalk part of the Console, there are 32-bit and 64-bit instance types. Normally, choosing either one makes it quite difficult to easily scale up or down between the different types of instances. Doing this with Beanstalk is easy, though it will cost a little bit of downtime. Because AWS has created images for most available instance types, it is relatively painless to move from c1.medium (32-bit) to m1.large (64-bit), for example.

We also want to change the security group, which by default is shared across environments and even applications. You can create a custom security group in the AWS Console (under the EC2 tab) and give it the minimum rules (HTTP and SSH). We also have Beanstalk launch the instances with a key pair, so we can get on in case of emergency. And we set the monitoring interval at 1 minute, incurring a little bit of extra cost. Changing security group or instance type forces Beanstalk to replace the instances as well. This will result in some downtime.


Edit Configuration

Edit the attributes of your configuration below. When you are finished making edits, click "Apply Changes".

These settings allow you to control your environment's servers and enable login. Learn more >>

*EC2 Instance Type: c1.medium
Note: Pick the instance type that best meets your compute, memory, and cost needs.

*EC2 Security Groups: heystaq-production

*Existing Key Pair: heystaq
Note: Key pairs are used to enable login to your instances.

*Monitoring Interval: 1 minute

*Custom AMI ID: ami-524db23b

Note: *It may take a few minutes to see changes to these options take effect in your environment.

*Figure 3-22. Configure Elastic Beanstalk environment*

After your new instances have come up, you can log in with SSH:

```
$ ssh -i heystaq.pem ec2-user@@ec2-50-19-33-104.compute-1.amazonaws.com
```

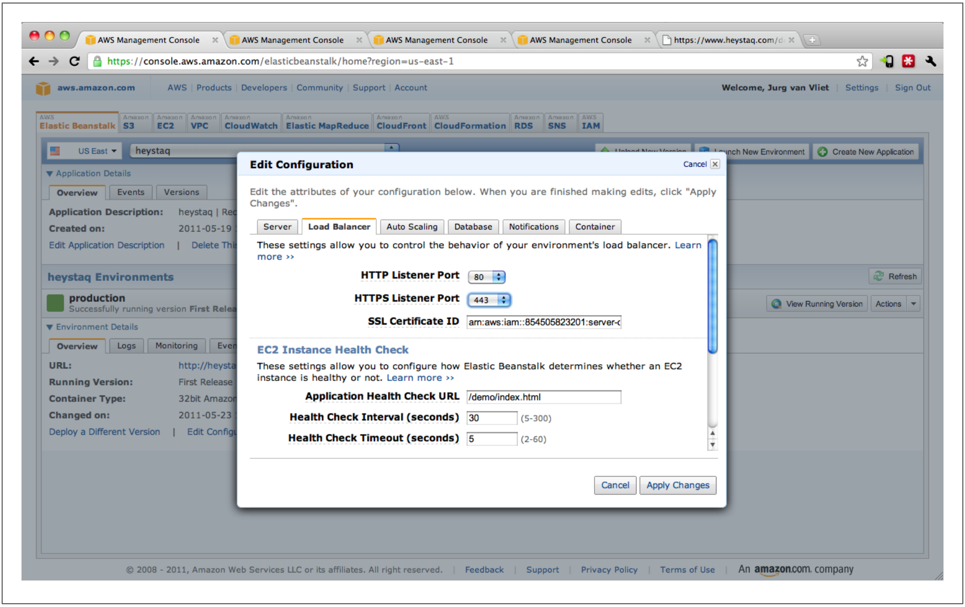
Have a look around your instance. It is a fairly sophisticated environment that integrates with the Beanstalk system. In Chapter 4 we'll take a look at what goes on inside, but for now let's stick to modifying configuration settings.

## Elastic Load Balancer

The default settings for the Elastic Load Balancer are fine. But we want to have the ELB route our HTTPS traffic as well. For that we need to add a certificate and have the ELB use that for traffic on port 443 or 8443, depending on your choice. To add the certificate, we do the following:

```
$ iam-servercertupload -b www_heystaq_com.crt -c www_heystaq_com.ca-bundle -k
www.heystaq.com.key -s www_heystaq_com
$ iam-servercertlistbypath
arn:aws:iam::854505823201:server-certificate/www_heystaq_com
```

Then, with the certificate ID we obtained, we go to the Load Balancer settings and add it as SSL Certificate ID (see Figure 3-23). Set your HTTPS port to 443 or 8443 and click "Apply Changes." Wait for the ELB to be changed, and test! This change does not give you any downtime.



**Edit Configuration**

Edit the attributes of your configuration below. When you are finished making edits, click "Apply Changes".

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

These settings allow you to control the behavior of your environment's load balancer. Learn more ››

| | |
|---|---|
| HTTP Listener Port | 80 |
| HTTPS Listener Port | 443 |
| SSL Certificate ID | arn:aws:iam::854505823201:server-c |

**EC2 Instance Health Check**

These settings allow you to configure how Elastic Beanstalk determines whether an EC2 instance is healthy or not. Learn more ››

| | | |
|---|---|---|
| Application Health Check URL | /demo/index.html | |
| Health Check Interval (seconds) | 30 | (5-300) |
| Health Check Timeout (seconds) | 5 | (2-60) |

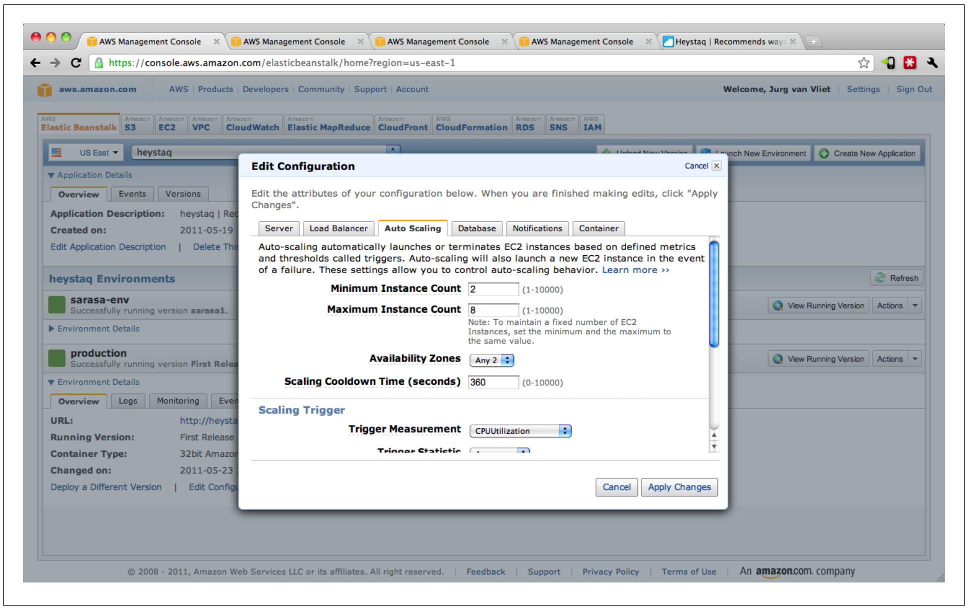
*Figure 3-23. Configure Elastic Beanstalk environment*

## Auto Scaling

The next important thing to configure is Auto Scaling (see Figure 3-24 and Figure 3-25). Even though it depends on the particular application you run, there are some sensible defaults to start with.

For starters, we want to have at least two instances, divided over two different zones. We configure it like this because we want to use availability zones for resilience. In theory, you should be able to choose more than just two zones, especially when you choose to start with a minimum of three instances and scale up and down with three. We only want two instances, for now, and scale with two at a time. Changing the number of instances will not result in downtime; you can't leave the environment running without any instances.

The default metric for scaling that Beanstalk chooses, NetworkOut, is not what we want. We always choose CPUUtilization as the metric for the auto scaling alarms. The main reason for choosing another metric is that we have no idea yet how NetworkOut or any of the others behave. CPUUtilization is a relative measure, and we can start conservatively, scaling out with 30% utilization, and scaling back in with 10% utilization. In the last section of this chapter, we'll do some stress testing on a clone of this environment, and adjust the configuration if necessary.



**Edit Configuration**

Edit the attributes of your configuration below. When you are finished making edits, click "Apply Changes".

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

Auto-scaling automatically launches or terminates EC2 instances based on defined metrics and thresholds called triggers. Auto-scaling will also launch a new EC2 instance in the event of a failure. These settings allow you to control auto-scaling behavior. Learn more »

| | | |
|---|---|---|
| Minimum Instance Count | 2 | (1-10000) |
| Maximum Instance Count | 8 | (1-10000) |
| | | Note: To maintain a fixed number of EC2 Instances, set the minimum and the maximum to the same value. |
| Availability Zones | Any 2 | |
| Scaling Cooldown Time (seconds) | 360 | (0-10000) |

Scaling Trigger

| | |
|---|---|
| Trigger Measurement | CPUUtilization |

Cancel | Apply Changes

Figure 3-24. Configure Elastic Beanstalk environment

## Container

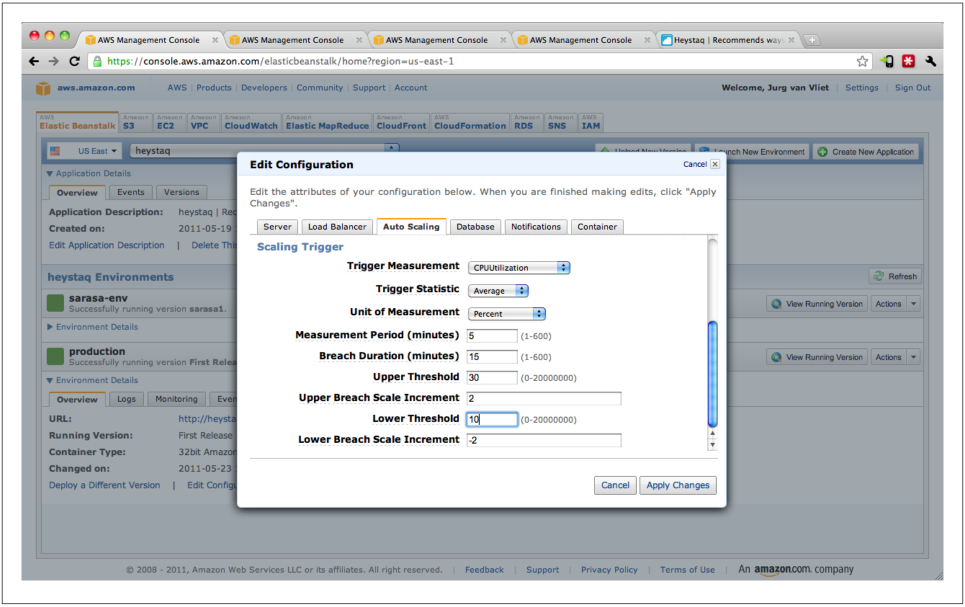
Choosing the name container (see Figure 3-26) hints at several types of containers, but for now we only have Tomcat. When changing instance types, the container configuration stays the same. This is a shame, because a medium instance already has significantly more memory to use than the default 256 megabytes.

For now we only change the JVM Heap Size environment variables. We'll give the JVM 512 megabytes to work with. We can set more options through the Java Command Line Options environment variable. But we'll work with those when we do stress testing in the staging environment.

And we do enable log file rotation to S3. In case of problems, it is necessary to find out what went wrong.

Note that you can pass custom environment variables to the JVM. This allows your application to get its configuration from somewhere else, for example. It is important to keep the configuration outside of the WAR, because in case of an emergency with some other component, you need to be able to change these configuration settings quickly.

Applying these changes does not replace the instances. Restarting Tomcat, however, might cause a little bit of downtime or unexpected behavior in your application.


Edit Configuration

Edit the attributes of your configuration below. When you are finished making edits, click "Apply Changes".

Scaling Trigger

Trigger Measurement: CPUUtilization
Trigger Statistic: Average
Unit of Measurement: Percent
Measurement Period (minutes): 5 (1-600)
Breach Duration (minutes): 15 (1-600)
Upper Threshold: 30 (0-20000000)
Upper Breach Scale Increment: 2
Lower Threshold: 10 (0-20000000)
Lower Breach Scale Increment: -2

Figure 3-25. Configure Elastic Beanstalk environment

## Configurations

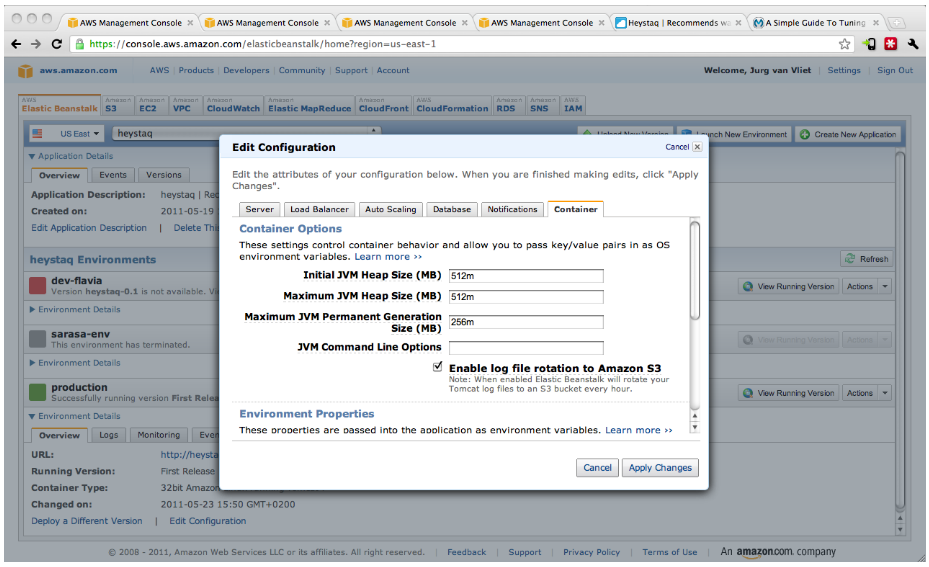
We have our production environment running. The traffic is not huge yet, so we have some time to do stress testing and tune the environment some more. It would be convenient if we could easily create a clone of this environment for this.

Beanstalk has a feature that makes this very easy: configurations. When we started writing this book, it was still hidden and poorly documented. We had to use the command line and figure out how this feature worked. We'll show you how we got it to work using the command line, but we'll also show screenshots illustrating where to find everything you need.

Listing your Elastic Beanstalk applications is simple:

```
$ elastic-beanstalk-describe-applications

ApplicationName | ConfigurationTemplates | DateCreated | DateUpdated
| Description | Versions
-------------------------------------------------------------------------------------
heystaq | Default | Thu May 19 11:47:17 +0200 2011 | Thu May 19 11:47:17 +0200 2011
| heystaq | Recommends ways to improve your AWS setup | v2, v1, heystaq-0.1,
First Release
hy.stq.io | Default | Mon May 16 17:10:08 +0200 2011 | Mon May 16 17:10:08 +0200 2011
| heystaq URL shortener | Initial Version
```

The application we are working with is heystaq; let's see what environments we have:


Edit Configuration

Edit the attributes of your configuration below. When you are finished making edits, click "Apply Changes".

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): 512m

Maximum JVM Heap Size (MB): 512m

Maximum JVM Permanent Generation Size (MB): 256m

JVM Command Line Options:

[✓] Enable log file rotation to Amazon S3
Note: When enabled Elastic Beanstalk will rotate your Tomcat log files to an S3 bucket every hour.

Environment Properties

These properties are passed into the application as environment variables. Learn more ››

Figure 3-26. Configure Elastic Beanstalk environment

```
$ elastic-beanstalk-describe-environments --application heystaq

ApplicationName | CNAME | DateCreated | DateUpdated | Description | EndpointURL
| EnvironmentId | EnvironmentName | Health | SolutionStackName | Status |
TemplateName | VersionLabel
-------------------------------------------------------------------------------------
heystaq | flavia.elasticbeanstalk.com | Mon May 23 15:53:44 +0200 2011 | Mon May 23
19:01:29 +0200 2011 | Flavia's development environment | awseb-dev-flavia-403863737.
us-east-1.elb.amazonaws.com | e-ibunpb3p2e | dev-flavia | Green | 64bit Amazon Linux
running Tomcat 7 | Ready | N/A | v2
heystaq | heystaq-production.elasticbeanstalk.com | Mon May 23 14:45:18 +0200 2011 |
Mon May 23 16:20:14 +0200 2011 | heystaq | Recommends ways to improve your AWS setup
| awseb-production-1751105411.us-east-1.elb.amazonaws.com | e-fni73snhht | production
| Green | 32bit Amazon Linux running Tomcat 7 | Ready | N/A | First Release
```

We see that we have two environments. Now, let's see what resources our production environment has:

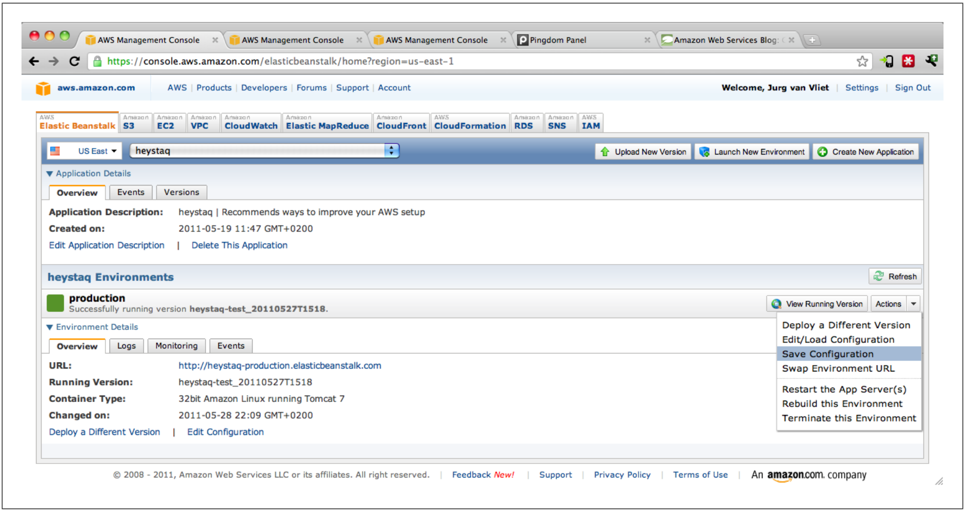
```
$ elastic-beanstalk-describe-environment-resources --environment-name production

AutoScalingGroups | EnvironmentName | Instances | LaunchConfigurations |
LoadBalancers | Triggers
-------------------------------------------------------------------------------------
awseb-production-HZqOCzokdi | production | i-c12ffbaf, i-694f9b07 | awseb-production-
bSmSJM21GS | awseb-production | awseb-production-HZqOCzokdi
```

We see two instances (remember we set the minimum number of instances in Auto Scaling to two), a launch configuration, which defines which image we are using, which instance type, key pair, security groups, etc. And the triggers for Auto Scaling.

We can now create a configuration, or configuration template, as it's called in the command-line tools:

```
$ elastic-beanstalk-create-configuration-template \
    --application-name heystaq \
    --template-name Production \
    --environment-id e-fni73snhht
```

And all this has been made easy in the console, as we can see in Figure 3-27.



https://console.aws.amazon.com/elasticbeanstalk/home?region=us-east-1

**Application Description:** heystaq | Recommends ways to improve your AWS setup
**Created on:** 2011-05-19 11:47 GMT+0200
Edit Application Description | Delete This Application

**heystaq Environments**

**production**
Successfully running version **heystaq-test_20110527T1518.**

**URL:** http://heystaq-production.elasticbeanstalk.com
**Running Version:** heystaq-test_20110527T1518
**Container Type:** 32bit Amazon Linux running Tomcat 7
**Changed on:** 2011-05-28 22:09 GMT+0200
Deploy a Different Version | Edit Configuration

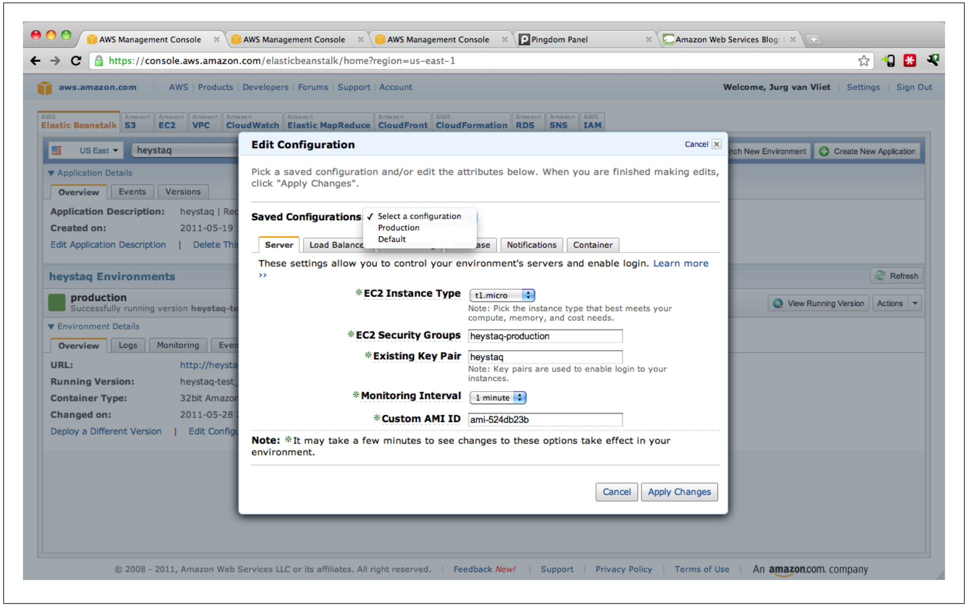
© 2008 - 2011, Amazon Web Services LLC or its affiliates. All right reserved. | Feedback *New!* | Support | Privacy Policy | Terms of Use | An amazon.com. company

*Figure 3-27. Save Elastic Beanstalk environment configuration*

Inspecting the application, we see we now have two configuration templates, Default and Production:

```
$ elastic-beanstalk-describe-applications \
    --application heystaq
ApplicationName | ConfigurationTemplates | DateCreated | DateUpdated | Description
| Versions
-------------------------------------------------------------------------------------
heystaq | Production, Default | Thu May 19 11:47:17 +0200 2011 | Thu May 19 11:47:17
+0200 2011 | heystaq | Recommends ways to improve your AWS setup | v2, v1, heystaq-
0.1, sarasa1, sarasa_0.1, First Release
```

Now, let's use this configuration template and create a copy of the production environment for staging in the next section. The latest Elastic Beanstalk updates also include editing/saving configuration in the console (see Figure 3-28). This is very convenient, as we'll see in the next section.



Edit Configuration

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

Saved Configurations: ✓ Select a configuration / Production / Default

These settings allow you to control your environment's servers and enable login. Learn more >>

*EC2 Instance Type: t1.micro
Note: Pick the instance type that best meets your compute, memory, and cost needs.

*EC2 Security Groups: heystaq-production

*Existing Key Pair: heystaq
Note: Key pairs are used to enable login to your instances.

*Monitoring Interval: 1 minute

*Custom AMI ID: ami-524db23b

**Note:** *It may take a few minutes to see changes to these options take effect in your environment.

*Figure 3-28. Editing/saving configurations*

AWS has another service, which is called CloudFormation. It solves the problem that configuration templates solve in a more general way, for all sorts of AWS entities (EC2 instances, Elastic Load Balancers, S3, even Elastic Beanstalk itself). You can read more about it in the online Amazon AWS documentation.

## Staging New Versions

A staging environment is only used occasionally. We want to test a new version, or new settings, so we need a “clean” copy of production. It is not necessary to have this environment around all the time. We’ll use our freshly created configuration template to easily launch a new environment that we can terminate again when we are done.

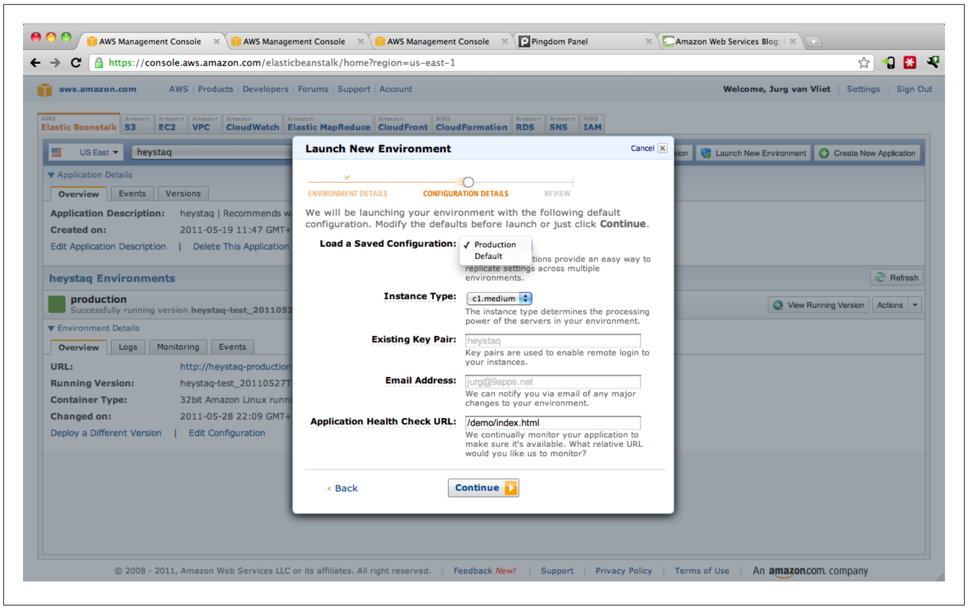
We want to stress test the environment we are running, so we’ll be using JMeter to generate some load. We will first tune the environment based on CloudWatch information we can read from the Console, but later we’ll create a custom AMI with some additional tools so we can monitor exactly the behavior of Tomcat.

### Launching the Environment

With our previously created configuration option, this is all it takes to launch a new environment with the proper version deployed:

```
$ elastic-beanstalk-create-environment \
    --application-name heystaq \
    --version-label v2 \
    --environment-name staging \
    --template-name Production \
    --cname-prefix staging-heystaq \
    --template-name Production

ApplicationName | CNAME | DateCreated | DateUpdated | Description | EndpointURL |
EnvironmentId | EnvironmentName | Health | Resources | SolutionStackName | Status
| TemplateName | VersionLabel
-----------------------------------------------------------------------------------
heystaq | heystaq-staging.elasticbeanstalk.com | Tue May 24 12:17:11 +0200 2011
| Tue May 24 12:17:12 +0200 2011 | N/A | N/A | e-nyzfqbdbgy | staging | Grey |
LoadBalancerListenersProtocolhttpPort80DomainLoadBalancerNameawseb-staging | 32bit
Amazon Linux running Tomcat 6 | Launching | Production | v2
```

This will launch a clone of the production environment, ready for us to work with. In the console, you can now choose the Configuration you want to use (see Figure 3-29).


Launch New Environment

We will be launching your environment with the following default configuration. Modify the defaults before launch or just click **Continue**.

Load a Saved Configuration: ✓ Production / Default

Instance Type: c1.medium

Existing Key Pair: heystaq

Email Address: jurg@9apps.net

Application Health Check URL: /demo/index.html

*Figure 3-29. Choose configuration*

## Looking for Limits

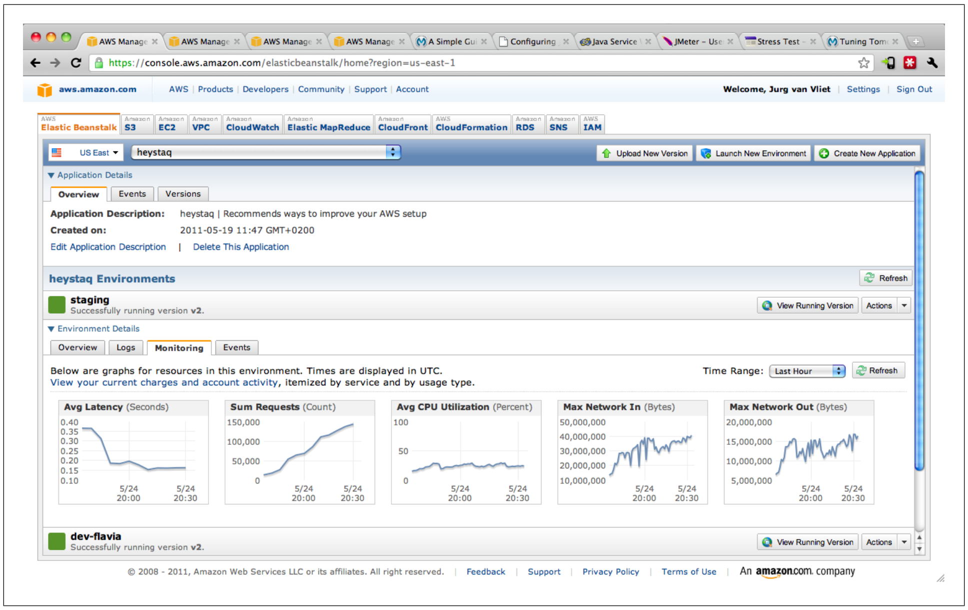
The first thing we want to do is figure out where the limits of this system are. For this we'll use JMeter, to generate some sort of realistic traffic. There is one thing important to know when putting strain on a system with an Elastic Load Balancer, and that is to take a fair "ramp up" time. In JMeter this is called `ramp_time`. And because we don't want to wait, we make the Auto Scaling in the staging environment a bit more "trigger happy" by kicking in after 5 minutes of breach, instead of 15.

The test we run is on four different URLs. We run this test with 32 threads, or users, in parallel. We use a medium instance to generate this load, and we can start several JMeter jobs doing these tests, continuously.

With the current, fairly standard environment, we are handling over 700 requests per second. This is on 12 medium instances, running a small Scala app on Tomcat 7, backed by SimpleDB. We only tested the REST API, which is backed by SimpleDB. If we run the same tests on the main page, we get to a much higher number of requests per second.

## The Environment

The instances run fine. In the Console we can see the performance of the environment at a glance in the Monitoring tab (Figure 3-30).


https://console.aws.amazon.com/elasticbeanstalk/home?region=us-east-1

Welcome, Jurg van Vliet

Application Description: heystaq | Recommends ways to improve your AWS setup

Created on: 2011-05-19 11:47 GMT+0200

heystaq Environments

staging
Successfully running version v2.

Below are graphs for resources in this environment. Times are displayed in UTC.
View your current charges and account activity, itemized by service and by usage type.

Time Range: Last Hour

dev-flavia
Successfully running version v2.

© 2008 - 2011, Amazon Web Services LLC or its affiliates. All right reserved.

*Figure 3-30. Monitoring Elastic Beanstalk environment*

It is weird to see the latency (on the Elastic Load Balancer) dropping when the Auto Scaling group has four or more instances In Service. If you want to see more detail, you can delve in the CloudWatch tab in the Console. There you can look at more fine-grained and configurable charts, and there are many more metrics that could be of interest.

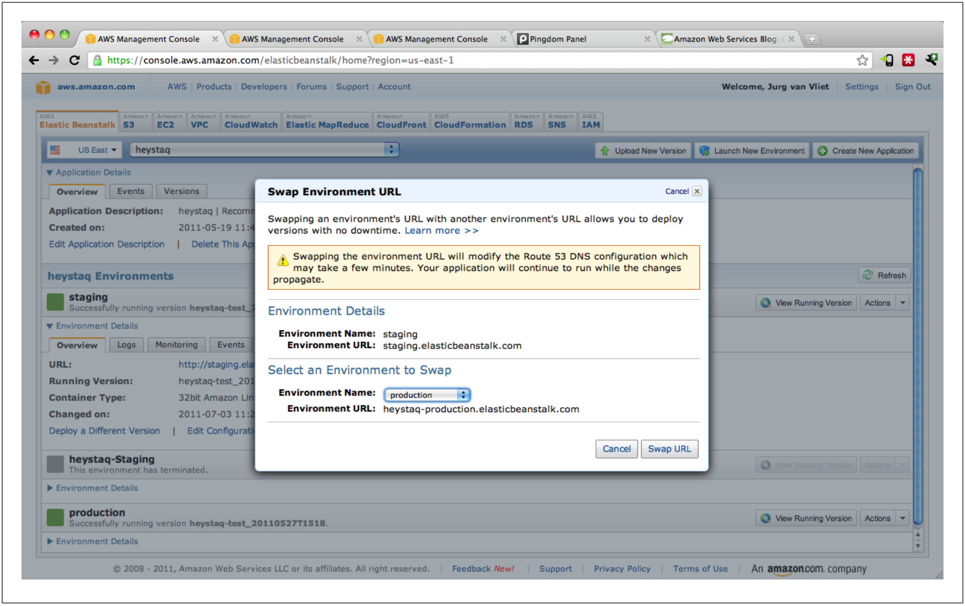
We already gave the JVM a lot more memory than it has by default. But there is one more memory management feature we would like to test under load, and that is "incremental garbage collection." Beanstalk makes this easy, because we can add `-Xincgc` to the JVM Command-Line Options in the environment configuration.

## Updating the Production Environment

Preferably, our production environment is never down. And with Auto Scaling and multiple application server instances, it should be possible to update an environment without too much service degradation. But, at the time we are working with this, updating a Beanstalk environment is quite disruptive.

Deploying a new version is relatively painless with our application, as the Tomcat restart is pretty quick. But changing configuration settings that require instances to be replaced hurts a lot. Beanstalk first deregisters the instance(s) from the Auto Scaling group, and then launches replacement instances. With our apps, this takes a couple of minutes. We did not find a way around this.

There is a way out, and that is to use URL Swapping (Figure 3-31). Instead of tearing apart the staging environment that has just been tested, we can promote it to production. You can do this by telling Elastic Beanstalk to swap the URL of the production environment with that of the staging environment.



**Swap Environment URL**

Swapping an environment's URL with another environment's URL allows you to deploy versions with no downtime. Learn more >>

Swapping the environment URL will modify the Route 53 DNS configuration which may take a few minutes. Your application will continue to run while the changes propagate.

Environment Details

Environment Name: staging
Environment URL: staging.elasticbeanstalk.com

Select an Environment to Swap

Environment Name: production
Environment URL: heystaq-production.elasticbeanstalk.com

Successfully running version heystaq-test_20110527T1518.

This environment has terminated.

© 2008 - 2011, Amazon Web Services LLC or its affiliates. All right reserved.

*Figure 3-31. URL swapping*

## Conclusion

In this chapter we experienced Elastic Beanstalk in its daily use, for the different stages of software. For developing, it's very convenient to have an easy and quick way to share our work with others. We think it's still better to have a local environment because it takes relatively too much time to deploy on Beanstalk. But we showed a nice way to publish changes continuously with Hudson, if we want to publish on every commit, for example.

In the production environment, you will probably have to make configuration changes and tweak settings such as those of Auto Scaling and of Tomcat itself. Most of these changes can be done from the AWS Console, and they can be saved in a configuration. For staging, we can use this configuration to replicate the production environment, stress test, and tweak some more.

CHAPTER 4

# Hacking Elastic Beanstalk

The purpose of this chapter is to explore where Elastic Beanstalk ends, and where we can begin to adjust the system ourselves. We'll start by delving into the way the Elastic Beanstalk instances integrate with the Elastic Beanstalk Service. If we understand this, we can slowly start to customize the image that runs our application (and the instances that are launched from it); we'll change the logging, replace the OpenJDK with the Sun JDK, and replace Apache with Nginx. An interesting way to change an infrastructure is to take things out, which is exactly what we'll do in the end: we'll make the Elastic Load Balancer bypass Apache or Nginx altogether.

So, we understand how Elastic Beanstalk works and have sort of mastered the fundamentals. It is time to go a little bit further. Why not get our hands dirty and change those fundamentals? Chapter 2 introduced the concepts underlying the AWS services Elastic Beanstalk uses. There is not time to get into the details of working with every one of those things. If you want more details on how to create an AMI, for example, we suggest you read *Programming Amazon EC2*.

Building on top of Elastic Beanstalk, we can do all sorts of interesting things. Perhaps you want to use Nginx instead of Apache. Or you are contemplating just ignoring Apache for the Tomcat traffic. You might be using some features on the Sun JDK that you are used to, and they are not implemented (yet) in the OpenJDK.

Well, the good news is, we can do all these things. And, even better, the *hostmanager* (the part of the image that AWS added) is available under the Amazon Software License. You have no control over the actual hostmanager to Beanstalk communication, so doing these things comes at a cost. You will have to keep an eye on changes to this service, and maintain your custom images yourself.

## The Instance

Elastic Beanstalk comes with default AMIs, for 32-bit and 64-bit instances. These images launch into instances that basically contain two things:

1. Everything related to running Tomcat (6 or 7)
2. The hostmanager, which is used to communicate with the Beanstalk environment

The hostmanager is a Ruby application. It takes care of starting and stopping necessary applications, handling deploys, and other Beanstalk-related tasks, like restarting the application servers. The Tomcat environment is a standard install, on top of the Open JDK.

If you know your way around Linux (CentOS/RH in particular), you can inspect the instances. If you launch a separate instance, you can make your changes and create a custom AMI. The difficulty is that you can't easily deploy your WAR and test if your changes work.

We launched an environment with one instance, and made changes there so we could test this immediately. Sometimes when we broke the instance Beanstalk automatically replaced it. So you have to make a bit of haste. Once we were happy with the changes, we replayed the changes to the separate instance and created our AMI. Changing the environment configuration will show you if you are successful or not.

It is a bit difficult to work with Elastic Beanstalk Instances like this. Another way to prevent accidental termination by an eager Elastic Beanstalk is to use Termination Protection. You can enable Termination Protection in the Console.

If you use Termination Protection, Elastic Beanstalk will still replace your instance if it does not show up healthy. But it can't terminate it, so you don't lose changes while working on it.

## Logging

The logging (Tomcat logs) is verbose in the default images. If you want to change this, you'll have to create a custom image. If you want to change the logging, you have to edit `/opt/tomcat7/conf/logging.properties`. We changed this file to this:

```
# ElasticBeanstalk Tomcat Logging
handlers = 1monitor.java.util.logging.FileHandler, 2tail.java.util.logging.FileHandler

# catalina.log for logrotate
1monitor.java.util.logging.FileHandler.level = WARNING
1monitor.java.util.logging.FileHandler.count = 1
1monitor.java.util.logging.FileHandler.pattern = ${catalina.base}/logs/monitor_
catalina.log
1monitor.java.util.logging.FileHandler.append = true
1monitor.java.util.logging.FileHandler.formatter=java.util.logging.XMLFormatter

2tail.java.util.logging.FileHandler.level = WARNING
2tail.java.util.logging.FileHandler.count = 1
2tail.java.util.logging.FileHandler.pattern = ${catalina.base}/logs/tail_catalina.log
2tail.java.util.logging.FileHandler.append = true
2tail.java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter
```

Now, create the image and change the environment configuration.

Since we are hanging around in */opt/tomcat7/conf* anyway, you can see many other files. If you are familiar with Tomcat, you can find your way easily.

One thing we noticed is that there are only two AMIs, one 32-bit and one 64-bit. If you want to use other instance types for your particular app, you will definitely want to have a look at `server.xml`, to change the number of threads, for example.

## Sun JDK

There are things we can't see from outside the instances. We can't see what the memory usage is, for example. By logging in, we can see the instances themselves are fine. The memory appears to be OK as well. But we have a very limited test suite, so it doesn't tell us very much.

There is an interesting tool we often use in other Tomcat environments, and that is VisualVM. In theory, it would be pretty straightforward to enable this type of profiling information in Tomcat, but it is not that easy.

First, it is available in the Sun JDK, and experimental in the OpenJDK that powers Beanstalk. But we still wanted to give this a try. Launching a separate medium instance from the Beanstalk AMI gives us something to work on. And after some time we were able to replace the OpenJDK with the Sun JDK by doing the following:

```
$ rpm -e --nodeps java-1.6.0-openjdk.i686

$ wget -O jdk-6u25-linux-i586-rpm.bin \
    http://download.oracle.com/otn-pub/java/jdk/6u25-b06/jdk-6u25-linux-i586-rpm.bin

$ sh jdk-6u25-linux-i586-rpm.bin

$ sed -i 's/\/usr\/lib\/jvm\(\/jre\)*/\/usr\/java\/jdk1.6.0_25/g' \
    /etc/java/java.conf \
    /etc/profile.d/aws-apitools-common.sh
```

We posted this solution in the AWS forum, and one of our colleagues (dhavala, going by the name of Kris) came up with an alternative way of installing the Sun JDK:

```
Re: installing Sun JDK
Posted by: Kris
Posted on: May 30, 2011 3:45 AM
 in response to: truthtrap
 Reply
When you remove OpenJDK using rpm -e --nodeps, you end up removing
some symbolic links that are not created upon installing Sun JDK bin.

Here are the commands for Tomcat, 64bit (similar to truthtrap's)
```

```
cd ~
wget -O jdk-6u25-linux-x64-rpm.bin http://download.oracle.com/otn-pub/
java/jdk/6u25-b06/jdk-6u25-linux-x64-rpm.bin
sudo chmod +x jdk-6u25-linux-i586-rpm.bin
sudo ./jdk-6u25-linux-i586-rpm.bin
sudo alternatives --install /usr/bin/java java /usr/java/default/bin/
java 20000
sudo update-alternatives --config java
sudo ln -s /usr/java/default/jre /usr/lib/jvm/jre
sudo ln -s /usr/share/java /usr/lib/jvm-exports/jre

(Optional) While you are at it, install PSI-Probe in the Dev
environments to monitor your JVMs.

Just copy probe.war to /usr/share/tomcat6/webapps, start Tomcat using:
/etc/init.d/tomcat6 start
```

We created the image, and it works perfectly. Now, supposedly, adding the following JVM Command-Line Options should make Tomcat ready for VisualVM style scrutiny, but we did not get this to work. This is not a "we leave this to the reader"; we have a book to finish. But if you know how to make this work, let us know:

```
-Dcom.sun.management.jmxremote=true \
-Dcom.sun.management.jmxremote.port=8086 \
-Dcom.sun.management.jmxremote.ssl=false \
-Dcom.sun.management.jmxremote.authenticate=false
```

## Nginx

Apache is quite a beast. It does everything, basically, but at a cost. For heavy lifting (like our heystaq API calls), this is fine, but for other calls, the overhead of Apache is not always necessary. So, why not replace Apache with Nginx?

Replacing Apache with Nginx is a little bit more difficult than replacing the JDK. To do this we not only have to change the OS installation, but we also have to change the hostmanager.

We compiled most of the changes you have to make into this script:

```
#!/bin/sh

# install Nginx
yum -y install nginx
sed -i 's/ 1;/ 4;/g' /etc/nginx/nginx.conf
echo 'MAKE SURE TO REMOVE THE server ENTRY FROM /etc/nginx/nginx.conf'

# add Nginx to the infamous Beanstalk hostmanager
cd /opt/elasticbeanstalk/srv/hostmanager/lib/elasticbeanstalk/hostmanager
cp utils/apacheutil.rb utils/nginxutil.rb
sed -i 's/Apache/Nginx/g' utils/nginxutil.rb
sed -i 's/apache/nginx/g' utils/nginxutil.rb
sed -i 's/httpd/nginx/g' utils/nginxutil.rb
cp init-tomcat.rb init-tomcat.rb.orig
sed -i 's/Apache/Nginx/g' init-tomcat.rb
sed -i 's/apache/nginx/g' init-tomcat.rb

# create the right proxies (Beanstalk and hostmanager)
echo 'proxy_redirect            off;
proxy_set_header          Host            $host;
proxy_set_header          X-Real-IP       $remote_addr;
proxy_set_header          X-Forwarded-For $proxy_add_x_forwarded_for;
client_max_body_size      10m;
client_body_buffer_size   128k;
client_header_buffer_size 64k;
proxy_connect_timeout     90;
proxy_send_timeout        90;
proxy_read_timeout        90;
proxy_buffer_size         16k;
proxy_buffers             32              16k;
proxy_busy_buffers_size   64k;' > /etc/nginx/conf.d/proxy.conf

echo 'server {
 listen 80;
 server_name _;
 access_log /var/log/httpd/elasticbeanstalk-access_log;
 error_log /var/log/httpd/elasticbeanstalk-error_log;

 #set the default location
 location / {
  proxy_pass         http://127.0.0.1:8080/;
 }

 # make sure the hostmanager works
 location /_hostmanager/ {
  proxy_pass         http://127.0.0.1:8999/;
 }
}' > /etc/nginx/conf.d/beanstalk.conf
```

Make sure to remove the `server` entry in `/etc/nginx/nginx.conf`; otherwise it won't work.

## The Infrastructure

Not everything happens on the instances, of course. We can also hack the infrastructure. In the previous section we replaced Apache with Nginx, for example. We can easily ignore Apache altogether, and tell the load balancer to connect its port 80 to the Tomcat ports on the instances.

There is one thing we need to do for that, and that is make sure the Tomcat instances accept incoming connections on that port. Until a few weeks ago, we had to open up the security group to the world, but Amazon released a feature that allows us to open it up to a special security group that each Elastic Load Balancer has. You can find this security group in the Console, and it has a form like `amazon-elb/amazon-elb-sg`, where `amazon-elb` is the owner alias.

And now, you can change the ELB from the command line to connect port 80 to 8080 (for easy testing you can make two distinct connections, 80 to 80 and 8080 to 8080):

```
# first remove the old listener
$ elb-delete-lb-listeners awseb-staging -lb-ports 80

# and then bypass the apache to point directly to 8080
elb-create-lb-listeners awseb-staging --listener "lb-port=80,instance-port=8080,
protocol=http"
```

In general it is best to hide as much of your instances as possible, but this particular feature just wasn't there yet. We expect the Elastic Beanstalk product team to implement these features in updates.

We could have removed Apache altogether, but that would have broken the hostmanager. This app runs on port 8999, but Beanstalk talks to `/_hostmanager`. Apache proxies this traffic as well. We chose to ignore Apache, and leave it be.

## Conclusion

This chapter shows that, with a few recipes, you can get inside Elastic Beanstalk and customize—or hack—the provided images to your needs. It's not so straightforward, but it's possible. The next thing would be to create your own AMIs for Beanstalk, but that goes beyond being a user to being a contributor to Beanstalk.

We hope that by reading this book you learned how to use Elastic Beanstalk in standard and more advanced ways. If you understand what is happening underneath the surface of your application, the potential of what you can do with Elastic Beanstalk and AWS is big!

## About the Authors

**Jurg van Vliet** graduated from the University of Amsterdam in computer science. After his internship with Philips Research, he worked for many web startups and media companies. Passionate about technology, he wrote for many years about it and its effects on society. He became interested in the cloud and started using AWS in 2007. After merging his former company, 2Yellows, with a research firm, he decided to start 9Apps, an AWS boutique that is an AWS solution provider and silver partner of Eucalyptus, together with Flavia. Give Jurg a scalability challenge, and he will not sleep until he solves it—and he will love you for it.

**Flavia Paganelli** has been developing software in different industries and languages for over 14 years, for companies like TomTom and Layar. She moved to the Netherlands with her cat after finishing an MSc in computer science at the University of Buenos Aires. A founder of 9Apps, Flavia loves to create easy-to-understand software that makes people's lives easier, like the Decaf EC2 smartphone app. When she is not building software, she is probably exercising her other passions, like acting or playing capoeira.

**Steven van Wel** is the cofounder of Marvia. Previously, Steven founded his own design/ad agency. He has his hands in every aspect of the product, but he spends most of his time making sure clients are happy users and its servers run as fast as possible. A runner, Steven spends his off-work time developing real-estate property.

**Dara Dowd** has been developing software for close to a decade for companies in Japan and the Netherlands. He still hopes to one day score the winning goal for Ireland in a World Cup final.

## Colophon

The animal on the cover of *Elastic Beanstalk* is a gold carp.

The cover image is from *Johnson's Natural History*. The cover font is Adobe ITC Garamond. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSansMonoCondensed.