FREE eBook

LEARNING elastic-beanstalk

Free unaffiliated eBook created from
Stack Overflow contributors.

#elastic-beanstalk

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: elastic-beanstalk

It is an unofficial and free elastic-beanstalk ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official elastic-beanstalk.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with elastic-beanstalk

Remarks

This section provides an overview of what elastic-beanstalk is, and why a developer might want to use it.

It should also mention any large subjects within elastic-beanstalk, and link out to the related topics. Since the Documentation for elastic-beanstalk is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Detailed instructions on getting elastic-beanstalk set up or installed.

Read Getting started with elastic-beanstalk online: https://riptutorial.com/elastic-beanstalk/topic/7568/getting-started-with-elastic-beanstalk

Chapter 2: Deploy a Java Web Application to Elastic Beanstalk

Introduction

Elastic Beanstalk is a Cloud PaaS provider (Platform as a Service), meaning applications can be deployed to the platform without the fuss of manually setting up a deployment environment.

Java applications are easily deployable to Elastic Beanstalk both via web interface or via command-line tools.

Examples

Deploying a Spring Boot Application to Elastic Beanstalk

Objectives:

1. Using the Spring CLI, we will create a new Spring Boot application to be deployed to Elastic Beanstalk.
    - We will edit the generated Spring Boot application so that it will:
        - Create a jar named `aws-eb-demo.jar`.
        - Listen on port 5000.
        - Include a single web page named `index.html`.
2. Using the Elastic Beanstalk CLI, we will:
    - Initialize the project for deployment to Elastic Beanstalk.
    - Create a new deployment environment and deploy the Spring Boot application in a single step.
    - Edit and redeploy the application to the same environment.
3. We will edit the Spring Boot application to add monitoring, management, and logging features.
4. We will redeploy the Spring Boot application to its target environment.

Prerequisites:

- Install the Spring CLI
- Install the Elastic Beanstalk CLI

Part 1: Create a Java application

Let's create a new Java application using the Spring CLI. The Spring CLI provides a convenient way to quickly get started with cloud-ready applications that leverage Spring Boot and the Spring Framework.

Steps

1. Create a Spring Boot web application with the Spring CLI and `cd` into the folder.

```
$ spring init -d=web -name=aws-eb-demo aws-eb-demo
$ cd aws-eb-demo
```

2. Initialize the project with source control, then commit the initial revision. This will allow us to track changes as they are made and revert to a previously working state if necessary. This is always a good practice before you start making changes to your application.

   For this example, we will use Git:

```
$ git init
$ git add .
$ git commit -m "initial commit"
```

3. For convenience in deployment, edit `pom.xml` and add the `<finalName>` setting under the `<project>`/`<build>` section:

```
<build>
    <finalName>${project.artifactId}</finalName>
    ...
</build>
```

   The `<finalName>` setting will cause the built artifact to have the name `aws-eb-demo.jar` without the version. This ensures that deployment scripts won't have to be changed every time the application iterates to a new version.

4. Configure the application to listen on port `5000` by adding the following property to `src/main/resources/application.properties`:

```
server.port=5000
```

   By default, the Spring Boot web application listens for requests over port `8080`. However, the AWS load balancer that fronts Elastic Beanstalk applications expects them to be listening on port `5000`. Without this setting, we will get the error `502 Bad Gateway` when attempting to access our application over the web.

5. Give your application a static home page by creating a file at `src/main/resources/static/index.html` with the following sample content:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>AWS EB Demo</title>
</head>
<body>
    <h1>Hello from Elastic Beanstock</h1>
</body>
</html>
```

6. Test the web application locally.

```
$ mvn spring-boot:run
```

   Open your browser to `http://localhost:5000` and verify that the home page is served up.

   When finished testing, type `Ctrl+C` in the running terminal to shutdown the application.

7. Package the web application.

```
$ mvn package -DskipTests
```

   The application jar will be created at `target/aws-eb-demo.jar`.

8. Commit your changes to source control.

```
$ git add .
$ git commit -m "prepared for initial deployment"
```

Part 2: Deploy the Java application to Elastic Beanstalk

With our application tested and ready to deploy for the first time, we can now use the Elastic Beanstalk CLI to initialize our deployment configuration, create the deployment environment, and push it to the cloud.

Steps

1. Using the Elastic Beanstock CLI, initialize your application for deployment.

```
$ eb init
```

   a. When prompted to "Select a default region", accept the default.

   b. When prompted to "Select an application to use", accept the default to create a new application.

c. When prompted to "Enter Application Name", accept the default or enter `aws-eb-demo`.

d. When prompted to "Select a platform", choose `Java`.

e. When prompted to "Select a platform version", choose `Java 8`.

f. When prompted to deploy using AWS CodeCommit, accept the default (`n`).

i. When prompted to set up SSH for your instance, select 'y'. This will allow you to use the Elastic Beanstalk CLI to shell into the virtual machine where your application instance will be deployed.

j. When prompted to "Select a keypair", choose the default (`Create new KeyPair`). Alternatively, you may select an existing keypair.

k. When prompted to "Type a keypair name", accept the default or type a name.

l. When prompted to "Enter a passphrase", leave empty or type a passphrase that you will remember. Type the same passphrase a second time and hit enter.

2. Edit the newly created Elastic Beanstock deployment manifest by opening `.elasticbeanstock/config.yml` and adding the following setting to the bottom of the file:

```
deploy:
  artifact: 'target/aws-eb-demo.jar'
```

3. Using the Elastic Beanstock CLI, deploy your application to a new environment.

```
$ eb create
```

a. When prompted to "Enter Environment Name", accept the default value (`aws-eb-demo-dev`).

b. When prompted to "Enter DNS CNAME prefix", accept the default value (`aws-eb-demo-dev`).

c. When prompted to "Select a load balancer type", accept the default (`classic`).

It may take up to 5 minutes for Elastic Beanstalk to complete the deployment. Meanwhile, Elastic Beanstalk is doing the following for you:

- creating the deployment environment
- creating the load balancer
- preparing a security group
- setting up auto-scaling
- launching one or more EC2 instances
- launching the application

While this work is being done, it is safe to type `Ctrl+C`.

4. While waiting, you may use the Elastic Beanstalk CLI to check the application's deployment status.

```
$ eb status
```

5. When the `Health:` field shows `Green` and the `Status:` field shows `Ready`, you may browse to the application using the address shown in the `CNAME` field.

   For example, if you chose the region `us-west-2 : (US West (Oregon)`, then your application will be deployed at the following URL:

   http://aws-eb-demo-dev.us-west-2.elasticbeanstalk.com

   If the application's status is `Ready`, but the `Health:` field is not `Green`, see the application logs to diagnose the problem:

```
$ eb logs
```

Part 3: Modify the Spring Boot application to include production-ready features.

For our application to be truly production-ready, we need smanagement, monitoring, logging and security. Fortunately, Spring Boot comes with these features out of the box. They simply need to be added as dependencies to our project.

For details about the production-ready features provided by Spring Boot, see the Spring Boot Reference Guide, Part V, Spring Boot Actuator: Production-ready features.

Steps

1. Open the file `pom.xml` (found in the root directory of our project), and add the following dependencies under the `<project>`/`<dependencies>` section:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-hateoas</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

   a. The `spring-boot-starter-actuator` dependency includes production-ready management and monitoring features, called `actuators`. These features include ready-made REST endpoints that allow administrators to quickly access runtime information about the application, such as health information, environment properties, configuration settings, log

files, and so forth.

b. The `spring-boot-starter-hateoas` dependency allows Spring Boot to list all actuators under a single convenient endpoint using `HAL`-style links that point to each actuator endpoint.

c. The `spring-boot-starter-security` dependency gives our application the facilities to secure the actuator endpoints so they can only be accessed by an authenticated user having the `ACTUATOR` role.

2. While still editing `pom.xml`, add a custom execution to the `spring-boot-maven-plugin` so that it will include information about the build with the packaged application artifact.

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <executions>
        <execution>
            <goals>
                <goal>build-info</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

3. Customize the Spring Boot configuration so that all actuators will be listed under the path `/management`. Also, configure the user name and password of the administrator that will be able to see the actuator endpoints. Finally, configure the application logging so that logging statements will be written to a file in addition to the console.

   Open `src/main/resources/application.properties` and add the following settings:

```
management.context-path=/management

security.user.name=admin
security.user.password=password1

logging.file=logs/application.log
server.tomcat.accesslog.enabled=true
```

4. Edit `.gitignore` and add the following line at the top of the file so that local testing logs will be ignored by source control.

```
logs/
```

5. Test the web application locally.

```
$ mvn spring-boot:run
```

   a. In a browser, open `http://localhost:5000/management` and verify that a JSON object providing links to available Spring Boot actuators is served up.

b. Open `http://localhost:5000/management/info` and verify that build information is showing.

b. Open `http://localhost:5000/management/health` and verify that the application status is `UP`. When not authenticated as a user having the `ACTUATOR` role, the status field should be the only health field displayed.

c. Open `http://localhost:5000/management/logfile` and verify that it requires authentication. Type in the credentials of the administator (`admin:password1`) and verify that it grants you access.

d. Open `http://localhost:5000/management/health` again and verify that additional health details are showing. When authenticated as a user having the `ACTUATOR` role, the health endpoint will show additional health details regarding the resources the application is using.

e. Try out some of the other actuator endpoints. In particular, you may be interested in `env`, `metrics`, `auditevents`, and `mappings`.

When finished testing, type `Ctrl+C` in the running terminal to shutdown the application.

6. Perform a clean rebuild and repackage the web application.

```
$ mvn clean package -DskipTests
```

As before, the application jar will be created at `target/aws-eb-demo.jar`.

7. Commit your changes to source control.

```
$ git add .
$ git commit -m "added management and monitoring features"
```

Part 4: Deploy the modified application

Re-deploying is usually much faster than the initial deployment because the environment has already been prepared. The CLI simply needs to upload the new jar to the environment's EC2 instances and restart the application.

Steps

1. With the Elastic Beanstalk CLI, redeploy new jar.

```
$ eb deploy
```

Once the upload is complete, you may safely type `Ctrl+C`

2. Verify that the application has deployed successfully.

```
$ eb status
```

3. In a browser, go to `http://aws-eb-demo-dev.us-west-2.elasticbeanstalk.com/management` and verify that everything works as it did when it was tested locally.

Read Deploy a Java Web Application to Elastic Beanstalk online: https://riptutorial.com/elastic-beanstalk/topic/9207/deploy-a-java-web-application-to-elastic-beanstalk

Chapter 3: Setting JVM Options

Remarks

```
-bundle.zip          //--> bundle file which will be uploaded
|--.ebextensions      //--> the file name must be exactly ".ebextensions"
   |--jvm.config     //--> config file that will set the JVM options on deployment (upload)
|--java_app.jar      //Here, I am using Spring Boot
```

```
option_settings:                                    //--> must have line
  aws:elasticbeanstalk:application:environment:     //--> namespace used to set JVM values
    XX:MaxPermSize: 256m                            //--> set XX:MaxPermSize to 256m
    Xmx: 1024m                                      //--> set Xmx to 1024m
    Xms: 512m                                       //--> set Xms to 512m
```

Examples

Setting JVM Options Via Ebextensions Config

To setup JVM options inside elastic beanstalk, your bundle file must be like this:

```
-bundle.zip
|--.ebextensions //do not forget the dot at the beginning of the name
   |--jvm.config
|--java_app.jar
```

And, the *jvm.config* file must be set like this:

```
option_settings:
  aws:elasticbeanstalk:application:environment:
    XX:MaxPermSize: 256m
    Xmx: 1024m
    Xms: 512m
```

When you upload your bundle, first those settings will be read and applied to the server's (Ec2 instance's) JVM and you must see the health as "OK".

Source

Read Setting JVM Options online: https://riptutorial.com/elastic-beanstalk/topic/7569/setting-jvm-options

Credits

S. No	Chapters	Contributors
1	Getting started with elastic-beanstalk	Community
2	Deploy a Java Web Application to Elastic Beanstalk	Robert Thornton
3	Setting JVM Options	webmaster