



Developing Applications with Cloud Foundry

Lab Instructions

Application Deployment using Pivotal Cloud
Foundry

Version 1.6.b

Pivotal

ved.

Copyright Notice

- Copyright © 2016 Pivotal Software, Inc. All rights reserved. This manual and its accompanying materials are protected by U.S. and international copyright and intellectual property laws.
- Pivotal products are covered by one or more patents listed at <http://www.pivotal.io/patents>.
- Pivotal is a registered trademark or trademark of Pivotal Software, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies. The training material is provided “as is,” and all express or implied conditions, representations, and warranties, including any implied warranty of merchantability, fitness for a particular purpose or non-infringement, are disclaimed, even if Pivotal Software, Inc., has been advised of the possibility of such claims. This training material is designed to support an instructor-led training course and is intended to be used for reference purposes in conjunction with the instructor-led training course. The training material is not a standalone training tool. Use of the training material for self-study without class attendance is not recommended.
- These materials and the computer programs to which it relates are the property of, and embody trade secrets and confidential information proprietary to, Pivotal Software, Inc., and may not be reproduced, copied, disclosed, transferred, adapted or modified without the express written approval of Pivotal Software, Inc.

Pivotal

© Copyright 2016 Pivotal. All rights reserved.

This Page Intentionally Left Blank

Table of Contents

1. Lab Setup	1
1.1. About this Lab	1
1.2. Steps	1
2. Treasure Hunt	6
2.1. Introduction	6
2.2. Resources	6
2.3. Steps	6
3. Deploy an Application Using the CLI	8
3.1. About this Lab	8
3.2. Steps	8
3.2.1. Connect	8
3.2.2. Push	9
3.2.3. Observe	9
3.2.4. Scale	9
3.2.5. Bonus	10
4. Deploy an Application Using Eclipse or STS	11
4.1. About this Lab	11
4.2. Steps	11
4.2.1. Check IDE	11
4.2.2. Deploy	12
4.2.3. Observe the running application	12
4.3. Bonus	13
5. Logging and Troubleshooting	15
5.1. About This Lab	15
5.2. Steps	15
5.2.1. Debugging Push Problems - IDE Users	16
5.2.2. Debugging Push Problems - Using <code>cf</code>	17
6. Manifests and Environment Variables	19
6.1. About This Lab	19
6.2. Steps	19
6.2.1. Setting Environment Variables	19
6.2.2. Optional - Scaling Application Instances	20
7. CloudFoundry Services	22
7.1. About this Lab	22
7.2. Steps	22
8. Service Brokers	25

8.1. Goal	25
8.2. Exercise	25
8.2.1. Overview	25
8.2.2. Prerequisites	26
8.2.3. Configure the Haash Broker	26
8.2.4. Running the Haash Client	27
8.3. Beyond the class	28
9. Using a Buildpack	30
9.1. About this Lab	30
9.2. Steps	30
10. Customizing a Buildpack	32
10.1. About this Lab	32
10.2. Steps	32
11. Integrating with 3rd Party Log Management Tools	34
11.1. About this Lab	34
11.2. Prerequisites	34
11.3. Setup PaperTrail	34
11.4. Create and Bind User-Provided Service	34
11.5. View Log Output	35
12. Integrating with APM tools	36
12.1. About this Lab	36
12.2. Prerequisites	36
12.3. Configure New Relic APM Service	36
13. Blue/Green Deployment	38
13.1. About this Lab	38
13.2. Steps	38
13.2.1. Push "Blue"	38
13.2.2. Alter the application	39
13.2.3. Push "Green"	39
13.2.4. Alter Routes	40
14. Continuous Delivery	42
14.1. About this Lab	42
14.2. Prerequisites	42
14.3. Setup Git Project	42
14.4. Install Jenkins	42
14.5. Configure Jenkins	43
14.6. Create the Initial Build Job	44
15. Session Management	47
15.1. About this Lab	47
15.2. Steps	47
15.2.1. Push	47
15.2.2. Restart Application	47

15.2.3. Bind the Redis session-replication Service	48
15.2.4. Retry Killing the Application	48
15.2.5. Examine the Application	48
16. Services and Environment in Java Applications	50
16.1. About this Lab	50
16.2. Steps	50
17. Appendix: Treasure Hunt Solution	52
17.1. Answers	52

Chapter 1. Lab Setup

1.1. About this Lab

Goals

Prepare your computer for the lab exercises

- Install the Cloud Foundry Command Line Interface
- Copy Lab projects to your computer
- Setup a trial account on Pivotal Web Services (or use a private PCF installation)
- Install an IDE: Eclipse, STS, or IntelliJ
- Install the Cloud Foundry Plugin for Eclipse
- Establish a GitHub Account (optional)

Prerequisites

To complete these steps, you will need:

- An Internet connection.

Estimated time to complete: 30 minutes

1.2. Steps

1. Obtain a Cloud Foundry account. Two options:
 - a. If you are running Pivotal CF on-premise you may already have an account so there may nothing more to do. If not, ask an organization manager to "invite" you to become a member of a Cloud Foundry organization. Your organization may have established an alternate process for you to follow to obtain an account. (Typically, an "organization manager" must send you an "invitation" to join an existing organization.)

- b. Alternatively, use Pivotal Web Services (PWS). Go to <https://console.run.pivotal.io/register> and go through the registration process for a trial account. If you already have a PWS account you can skip this step.

2. Install Cloud Foundry Command Line Interface (CLI)

Go to <https://github.com/cloudfoundry/cli/releases> or <https://console.run.pivotal.io/tools> and download the installer for your platform. Run the installer.

3. Copy lab files from GitHub

- a. Go to <https://github.com/S2EDU/CloudFoundryStudentFiles>. Click the “Download Zip” button to download the project files we will use in this course.
- b. Unzip the files to an easy-to-find location on your local file system (for simplicity we suggest C:\ on a Windows system). For the remainder of this course, we will simply refer to the projects by name, so you should remember this location.

4. Install an IDE (Eclipse, Spring Tool Suite or IntelliJ)

Some of the exercises work best when run from an IDE, such as Eclipse, Spring Tool Suite (STS) or IntelliJ, but again if you are not familiar with these tools, you may prefer to simply use the CLI and your favorite editor (such as Wordpad, Notepad++, JEdit, TextWrangler, XCode, gEdit or even `vi` or `emacs`). If so you are done, please ignore the rest of these instructions.

- a. For Eclipse, Go to <https://www.eclipse.org/downloads/> and download Eclipse. Select the latest version, download it, and install it. If you already have Eclipse 4.3 or higher, you can skip this step.
- b. For STS Go to <http://spring.io/tools/sts> and download STS. Select the latest version. If you already have STS version 3.6.0 or higher, you can skip this step.
- c. IntelliJ also includes integration with Cloud Foundry. You may use it if you prefer, although the instructions in this course will only refer to Eclipse. If you already have IntelliJ Ultimate Edition V13 or later, skip this step. Otherwise download the Ultimate Edition for your platform from <http://www.jetbrains.com/idea/download/>. This comes with a free 30 day trial license and Cloud Foundry integration built-in. The free Community Edition *does not* support Cloud Foundry deployment.

5. Install IDE Support for Cloud Foundry

- a. Eclipse / STS: Install Cloud Foundry Integration
 - i. If using Eclipse or STS, select Eclipse Marketplace from the Help menu.

- ii. In the marketplace dialog, enter “Cloud Foundry” in the Find field, and click Go.
- iii. Find the entry for "Cloud Foundry Integration for Eclipse" in the search results. The most recent versions of STS have Cloud Foundry installed by default. There are three possibilities:
 - A. The Cloud Foundry plugin is not installed, click the Install button now.
 - B. If Cloud Foundry is already installed there will be two buttons Update and Uninstall. If the Update button is disabled there is nothing more to do. Go to the next section "Add a “Cloud Foundry” Server" below.
 - C. If the Update button is enabled, you may wish to update now. Click Update to continue. If you *do not* wish to update, go to the next section "Add a “Cloud Foundry” Server" below.
- iv. In the next dialog just click Confirm
- v. Finally, accept the license agreement and click Finish. The install/update process will take a bit of time, and you will be prompted to restart Eclipse to make the changes take effect.
- vi. Full details: [Install to Eclipse](#)

b. IntelliJ: Enable Cloud Foundry Plugin

You may have enabled Cloud Foundry when you ran IntelliJ the first time. If not, or not sure, go to Preferences / Plugins (under IDE Settings) now and enable it.

6. Add a “Cloud Foundry” Server

a. Eclipse or STS

- i. Within Eclipse or STS, Select the “Servers” view; this is typically located in the lower left corner of the workspace. (If it is not present, go to Window / Show View / Other / Server / Servers.)
- ii. Within the Servers view, right click, select New / Server and browse to Pivotal / Cloud Foundry. Enter the email and password for your account.
- iii. By default the URL specifies PWS (run.pivotal.io). If using a different installation, click on “Manage Cloud”, then “Add” and then specify a name for your CF instance (call it whatever you want) and its API URL. If you are not sure of the API URL, you will need to obtain it from an administrator familiar with your CF setup.
- iv. If prompted, it is fine to take the default values for Organization / Space or make a selection from the choices offered. If unsure, accept the defaults. We will explain "organization" and "space" more fully

later in the course.

v. Click Finish. We now have a 'server' in Eclipse/STS that points to a Cloud Foundry environment.

b. IntelliJ

i. Open the Preferences/settings dialog and go to Cloud (under IDE Settings) to configure your Cloud Foundry setup.

ii. Click the + to add a new cloud and select Cloud Foundry. Specify your Cloud Foundry username, password, organization and preferred space. Be sure to leave Version V2 selected.

iii. The Provider defaults to PWS. If you are using your own CF instance you will need to supply its API URL (you were asked for this during step 3).

iv. Click "Test connection" to see if all the details are correct. Fix them if necessary.

v. Click OK to finish.

7. Import projects into your IDE

a. Eclipse or STS

i. From the project explorer, right click, import, Maven, existing maven projects

ii. Navigate to the folder where you extracted the student files (step 2) and Finish.

iii. The projects will be imported.

iv. Note that depending on the version of Eclipse / STS, your version of Java, and other factors, you may encounter errors. These will need to be fixed on a case-by-case basis before proceeding.

b. IntelliJ

i. From File menu select "Import Project" and navigate to wherever the course labs were installed during the previous step.

ii. Click "Import project from external model" and import as Maven projects.

iii. After clicking Next, select the "Search for projects recursively" tick-box.

iv. Click Next again and several projects should be listed.

v. Click Next and select a suitable SDK (Java 7 or 8 is required).

vi. Click Next one last time. You can leave the project name and location. Click Finish.

vii. Repeat the procedure, but this time explicitly select the spring-music project within CloudFoundryStudentFiles. IntelliJ should know automatically it is a gradle project so just click Next and then Finish. Import may take a while. When prompted, ask for the project to be added to a New Window.

8. Register a GitHub account (optional, recommended).

Go to <https://github.com> and signup for a free account. If you already have a GitHub account, you can skip this step. If you are not familiar with git, this step can be ignored.

Congratulations! You have setup everything you will need for the rest of the course. The hard part is over!

Chapter 2. Treasure Hunt

2.1. Introduction

In this lab you will look at various aspects of Cloud Foundry by using the *online documentation*. There is no coding required and you will not actually be using Cloud Foundry. This is just a "treasure hunt" to become familiar with the documentation.

Goals

What you will learn:

- Explore Cloud Foundry and Pivotal CF documentation sites

Estimated time to complete: 25 minutes

2.2. Resources

Open the following URLs in a browser:

- <http://docs.cloudfoundry.org/>
- <http://docs.pivotal.io/pivotalcf>
- <http://docs.run.pivotal.io>

Create bookmarks / favorites in your browser for each one of these.

2.3. Steps

Answer the following questions based on <http://docs.cloudfoundry.org> - note that most of these concepts have not yet been discussed in class:

1. What languages / frameworks does Cloud Foundry appear to support (see “buildpacks”)?
2. What kinds of things can be specified in a “Manifest”?

3. When designing applications for the cloud, what are some practices to avoid? (Hint: see section on "App design for the cloud")
4. What are the names of some of the architectural components within Cloud Foundry?
5. What are some of the “log types” that you can expect to find in Cloud Foundry logs?

Answer the following questions based on <http://docs.pivotal.io/pivotalcf/> - note that many of these concepts have not yet been discussed in class:

6. What kinds of things can be defined in the “Apps Manager”?
7. What is the purpose of a “Buildpack”?

Answer the following questions based on <http://docs.run.pivotal.io> - note that many of these concepts have not yet been discussed in class:

8. What kind of services are presently available in the services marketplace?
9. BONUS When building a custom buildpack, what is the purpose of the “DETECT” script?
- 10.BONUS: If you were to “push” a new web application named “weeble” to Pivotal web services, what URL could you use to access it?

(Answers are in the appendix)

Chapter 3. Deploy an Application Using the CLI

3.1. About this Lab

Goals

What you will learn:

1. Push a simple Java/Spring application to Cloud Foundry
2. Observe and scale a running application
3. You need to know the three URLs for either your Cloud Foundry installation or for PWS. Refer back to the presentation for details on the *System Domain*, *Apps Domain* and *Apps Manager* URLs.

Prerequisites

To complete these steps, you will need:

- A Cloud Foundry account (either on your company's CF installation or Pivotal Web Services)
- The Cloud Foundry student files
- The Cloud Foundry CLI installed

See the "Lab Setup" in the "Welcome, Agenda, and Setup" module for instructions.

Estimated time to complete: 30 minutes

3.2. Steps

3.2.1. Connect

Check setup and login to your Cloud Foundry instance.

1. Open a command prompt on your system. Run the `cf` command. If you do not see a detailed output of Cloud Foundry commands, return to the prerequisites.

2. Use `cf login` to connect to Cloud Foundry

- a. Use the `-a` option to specify the “api” of the Cloud Foundry instance you wish to connect to. For Pivotal Web Services, this is api.run.pivotal.io. For a private CF installation you will need to find out what the System Domain URL is - this is installation dependent. Once you know the System Domain, then the API URI is `api.<system-domain>`.
- b. Do NOT use the `-p` option for password. Instead, let the interface prompt you for your password.

3.2.2. Push

Push the Spring Music application.

3. Change folders to the location where you downloaded the `CloudFoundryStudentFiles` and go into the `spring-music` folder.
4. Run the `cf push` command to push this application to Cloud Foundry
 - a. Use any value you like for the application name. We suggest something brief and easy to type.
 - b. Use the `-n` option to specify the host. This value must be unique within the domain used by Cloud Foundry.
 - c. Use the `-p` option to specify the package to upload. This is a Java application, and a WAR file has already been built for you. It is located in “pre-built/spring-music.war”.
5. Observe the push log output. Can you identify the different stages of the process?

3.2.3. Observe

4. When the process is complete, identify the URL of the application. Open a browser and connect to this URL to make sure that the application comes up.
5. Run the `cf logs` command for your app.
6. Return to the browser, click the links, make changes to the data, then return to the command line and observe the generated log activity.

3.2.4. Scale

7. Open the Apps Manager associated with your Pivotal CF instance. For Pivotal Web Services, this is at <http://run.pivotal.io>. For a private CF installation you will need to find out what the System Domain URL is - this is installation dependent. Once you know the System Domain, then the Apps Manager URI is `console.<system-domain>`.
8. Login and navigate to the organization and space that your application was pushed to.
9. Open the details page associated with your application. Change the number of instances to 4 and save your changes. From the browser, refresh the application's web page repeatedly and notice that the application does not go down while it is being scaled. Return to the command prompt and observe the activity that occurs as Cloud Foundry scales your application horizontally.
10. Back on the Apps Manager, scroll down to the "instances" section to observe the running instances.
11. Scale back to 1 instance.
12. Unless you have time to do the bonus below, stop the app.

Congratulations, you have completed this exercise! If you have time, look at the bonus.

3.2.5. Bonus

There are other application properties we can modify. Change the memory settings, and experiment with adding / removing routes.

When you are done, stop the application.

Chapter 4. Deploy an Application Using Eclipse or STS

4.1. About this Lab

Goals

What you will learn:

- Use an IDE to deploy an application
- Push a simple Java/Spring application to Cloud Foundry
- Observe a running application

Prerequisites

To complete these steps, you will need:

- A Cloud Foundry account (either on your company's CF installation or on Pivotal Web Services)
- The Cloud Foundry student files
- An IDE installed (with Cloud Foundry support) - Eclipse or STS may be used.



Note

If you wish to use IntelliJ, you may, but the instructions below are all Eclipse specific.

See the "Lab Setup" in the "Welcome, Agenda, and Setup" module for instructions.

Estimated time to complete: 30 minutes

4.2. Steps

4.2.1. Check IDE

Open your IDE. Make sure that you have a “Cloud Foundry” Server and that you have imported all of the lab projects, specifically the `spring-mvc-demo` project. If you are missing either of these, please return to the Setup lab and make sure you complete the later steps on "Add a “Cloud Foundry” Server" and "Import projects into your IDE".

4.2.2. Deploy

Deploy the `spring-mvc-demo` project on Cloud Foundry.

1. Select the `spring-mvc-demo` project (there may be an outer folder, if so open the folder to find the project within it).
2. Drag the `spring-mvc-demo` project and drop on the “Pivotal Cloud Foundry” server. (Or, right-click on the project, select Run As / Run on Server, and select the Cloud Foundry server)
3. When prompted, enter a name. We recommend you choose a short name that is easy to type when working with CF’s command line interface. Leave “buildpack” empty, and check “Save to Manifest”. Next.
4. Alter the value of subdomain. The default value (based on application name) is unlikely to be unique within the `cfapps.io` domain, so create a longer name by prefixing/suffixing `spring-mvc-demo` with another value, such as your initials or name, such as `keyser.soze.spring-mvc-demo` - the goal is to make Deployed URL unique across the entire `cfapps.io` domain. Click Next.
5. Click “next” through the remaining screens. You won’t need to change any values, but note the type of things that can be controlled during deployment.



Note

A 400 error encountered at this stage typically indicates a non-unique application name. Remove your application and try to create a unique name.

4.2.3. Observe the running application

1. Double click on the “Pivotal Cloud Foundry” server. Go to the “Applications and Services” tab. Click on your `spring-mvc-demo` application.
2. Click on the “Mapped URLs” link to see your application. If desired open your app in another browser.
3. Note the Apps Manager is updated with the application’s activity.

If you have reached this point, congratulations, you have completed the lab!

4.3. Bonus

If you have time, see if you can complete the following:

1. Scale the application:
 - a. In the console view, use the “close console” button to close all consoles that may be open. There may be several open, so click this button several times.
 - b. Change the number of instances from 1 to 2. Click Restart.
 - c. Note the “Instances” section is updated with two running instances (it may take a moment for both instances to restart).
 - d. Right click on one instance and select “Show Recent Logs”
 - e. In the console view, you will see the logs for the instance you selected. Now right-click on the other instance. The console will refresh with its logs.
 - f. Unfortunately you cannot view the logs for more than one instance at a time. In the browser, refresh the web page. By fetching the logs for each instance in turn, see if you can determine which instance handled the request
 - g. Change instances back to 1. You should see a message in the console telling you an instance is stopping. After a while only one instance should remain in the Instances table.
2. Explore the manifest.
 - a. Find the “Manifest” prompt and click the “Save” button. Click “OK” if prompted about merging.
 - b. Right-click the spring-mvc-demo project and select Refresh. Notice that a “manifest.yml” file is now present within the project.
 - c. Open manifest.yml. Explore the settings and format of this file.
3. Alter memory setting.
 - a. Within Your IDE, alter the memory allocation for spring-mvc-demo to 128 MB. Note that this will not be enough memory to run the application.
 - b. Click Update and Restart. Observe the console to see what kinds of problems will be reported. Does the

application ever start? Do you get a clear message about memory?

Chapter 5. Logging and Troubleshooting

5.1. About This Lab

Goals

What you will learn:

- Become familiar with accessing / reading logs
- Diagnose application problems using CF tools

Prerequisites

To complete these steps, you will need:

- A Cloud Foundry account (either on your company's CF installation or on Pivotal Web Services)
- The Cloud Foundry student files
- The Cloud Foundry CLI installed
- Optionally: an IDE installed (with Cloud Foundry support)

See the "Lab Setup" in the "Welcome, Agenda, and Setup" module for instructions.

Estimated time to complete: 20 minutes

5.2. Steps

In this section you will push an application to Cloud Foundry that has a fatal flaw. By accessing logs you will attempt to diagnose the issue, repair the application, and re-push.



Warning

If you have trouble viewing logs and events using `cf` commands, view the logs in the Apps Manager console instead. This is usually due to proxy or firewall restrictions on the websocket port 4443. The typical errors are:

```
Error dialing loggregator server: dial tcp 1.2.3.4:4443: connection timed out.  
  
websocket.Dial wss://loggregator.run.pivotal.io:4443/tail/?... connection timed  
out.
```

5.2.1. Debugging Push Problems - IDE Users

If you prefer to use an IDE, continue with this section. If you prefer to use the `cf` command-line interface, skip to the [next section](#).

1. If you have not done so already, import the `cf-spring-mvc-trouble` project into your IDE.
If you have any other projects open, you might like to close them all so you don't work with the wrong project by mistake.
2. Deploy the `cf-spring-mvc-trouble` application on Cloud Foundry
 - a. Be sure to alter the value of subdomain to produce a unique route.
 - b. Watch the console / log output carefully during the deployment. Note that the application appears to be having difficulty starting. Run the following commands to see if you can diagnose the issue (replace “[app]” with your application’s name):

- i. `cf events [app]`
- ii. `cf logs [app] --recent`
- iii. `cf logs [app]`

If you are having problems with `cf logs` - see warning at start of section.

3. Once you have a general feeling for the problem, return to the IDE and examine this Java class: `gopivotal.cf.workshop.Config` - an easy way to do this is to use Ctrl-T or Cmd-T to popup the Open Type dialog and enter `Config` - pick the `Config` class in the current project. Find the lines of code marked with the “BAD CODE” comment.

This is a Spring application and the `@PostConstruct` method is automatically called on startup. Do you see why it is failing?

4. Comment out the code that is causing the problem. (Java comments are lines strating with the “//” characters.) Save your work (File / Save), return to the “Pivotal Cloud Foundry” tab, and use “Update and Restart” to repush and restart the application.



Note

When using an IDE (Eclipse/STS/IntelliJ) there is no need to ‘build’ the application. Saving causes an automatic build on any modified files.

5. The application should now start successfully. Run the `cf events` and `cf logs` commands again and note the differences in the logged output.

Congratulations, you have finished this exercise.

5.2.2. Debugging Push Problems - Using `cf`

In this section we will do the lab using the `cf` utility.

1. In your terminal or command window, make sure that `CloudFoundryStudentFiles/cf-spring-mvc-trouble` is the current directory.
2. Deploy the `cf-spring-mvc-trouble` application on Cloud Foundry. There is no manifest so you will need to specify the `-i`, `-m`, `-n` and `-p` flags. The application you need to deploy is `pre-built/cf-spring-mvc-trouble-0.0.1-SNAPSHOT.war`.
 - a. Be sure to specify the value of subdomain (`-n` option) to produce a unique route.
 - b. Watch the console / log output carefully during the deployment. Note that the application appears to be having difficulty starting. Run the following commands to see if you can diagnose the issue (replace “[app]” with your application’s name):

- i. `cf events [app]`
- ii. `cf logs [app] --recent`
- iii. `cf logs [app]`

If you are having problems with `cf logs` - see warning at start of section.

3. Once you have a general feeling for the problem, examine the Java code in the following file:
`gopivotal/cf/workshop/Config.java`. An easy way to do this is to enter
 - `more src/main/java/com/gopivotal/cf/workshop/Config.java` (Linux/MacOS)
 - `more src\main\java\com\gopivotal\cf\workshop\Config.java` (Windows)

in your command/terminal window to view the file. Use the tab key auto-complete feature to avoid typing the path in full. Or feel free to use the editor of your choice.

Note the lines of code marked with the “BAD CODE” comment:

```
@PostConstruct
public void consumeAllMemory() {

    //      TODO: LOOK HERE!  LOOK HERE!  LOOK HERE!
    //      BAD CODE!  BAD CODE!  BAD CODE!
    //      The code you see here will consume all available memory allocated
    //      to the application instance. Comment out this code and re-push

    List<Double> list = new ArrayList<Double>();
    while (true) {
        list.add(Math.random());
    }
    //      End of bad code.
}
```

This program has been deliberately setup to fail - the `consumeAllMemory` method sits in an infinite loop until all memory is consumed and the application fails to start.

4. Lets assume you contacted a developer colleague and got them to remove the offending code. The new application is `pre-built/fixed-cf-spring-mvc-trouble-0.0.1-SNAPSHOT.war`.

Rerun your `cf push` command but specify this file instead (using the `-p` option).

5. The application should now start successfully. Run the `cf events` and `cf logs` commands again and note the differences in the logged output.

Congratulations, you have finished this exercise.

Chapter 6. Manifests and Environment Variables

6.1. About This Lab

Goals

What you will learn:

- Set application behavior via manifest and CLI options
- Gain experience pushing and debugging applications

Prerequisites

To complete these steps, you will need:

- A Cloud Foundry account (either on your company's CF installation or on Pivotal Web Services)
- The Cloud Foundry student files
- The Cloud Foundry CLI installed
- An IDE installed (with Cloud Foundry support)

See the "Lab Setup" in the "Welcome, Agenda, and Setup" module for instructions.

Estimated time to complete: 20 minutes

6.2. Steps

6.2.1. Setting Environment Variables

In this section you will set environment variables via the manifest and observe how they are accessed in (Java) code.

1. Open the `cf-spring-mvc-demo` folder and locate the manifest.



Note

STS does not use manifests when pushing so *you can only use the CLI to do the next few steps*.

2. Add an environment variable to the manifest using your favorite editor. Name the variable `EVTEST` and give it the value `FromManifest`. Save your work.
3. Push the `cf-spring-mvc-demo` application to Cloud Foundry *using the `cf` CLI*.
4. If you understand Java code and Spring MVC Controllers, while the application is being pushed, examine the logic in the `CloudFoundryWorkshopController` Java class. Specifically look at the `environment()` method to see how `System.getenv()` is used to obtain the variables.

Also open the `env.html` page (under `src/main/resources/templates`) and see how the `environmentVariables` are accessed and put in the page using Thymeleaf markup.

5. Once the application has started, access the main page, then select the `Environment Variables` menu. The `EVTEST` variable should be present and set to `FromManifest`.

6.2.2. Optional - Scaling Application Instances

In this section you will scale the application to multiple instances and note the effect.

1. Using either your IDE, CLI, or Apps Manager, alter the number of instances to 2.
2. When scaling is complete, access the main page of the web app. Note the value of the application's port. Use the browser's refresh button several times. Does the port value remain the same?
3. Using the CLI or STS, tail the logs for this app. Refresh the home page several times. Do you see entries for both `App/0` and `App/1`?



Note

If you are unable to run `cf logs` due to proxy or firewall restrictions, view logs in Apps Manager instead.

4. Scale instances back to 1. (You will find this is much quicker using the CLI or Apps Manager)

5. Stop the application.

Congratulations, you have finished this exercise.

Chapter 7. CloudFoundry Services

7.1. About this Lab

In this lab, you'll practice creating, binding and managing services using both the CLI and Apps Manager.

Goals

What you will learn:

- Bind a service to an application

Prerequisites

To complete these steps, you will need:

- A Cloud Foundry account (either on your company's CF installation or on Pivotal Web Services)
- The Cloud Foundry student files
- The Cloud Foundry CLI installed
- Optional: An IDE installed (with Cloud Foundry support). This lab can be done using the CF utility only.
- Additionally, the "Spring Music" application should have already been pushed to Cloud Foundry. Push it now if needed. See the earlier lab "Deploy an Application Using the CLI" if you need guidance.
- Finally, you will need some database client software to connect to the new database that you will provision. Many JDBC compatible tools are suitable. Some suggestions are DBVisualizer (<http://www.dbvis.com/>), SquirrelSQL (<http://squirrel-sql.sourceforge.net>), HeidiSQL(<http://www.heidisql.com/>), JackDB (<https://www.jackdb.com>). JackDB is especially useful if your firewall prevents you from making connections to databases in the public cloud. Install / use the tool of your choice.

Estimated time to complete: 30 minutes

7.2. Steps

1. Obtain a listing of all of the services / plans available in the marketplace. This listing can vary depending on the Cloud Foundry environment you are connected to:

```
kkrueger-mbp15:~ kkrueger$ cf marketplace
Getting services from marketplace in org kkrueger-net / space development as kkrueger@gopivotal.com...
OK

service      plans      description
blazemeter    free-tier, basic1kmr, pro5kmr, pp10kmr, hv40kmr  The JMeter Load Testing Cloud
cleardb       spark, boost, amp, shock                        Highly available MySQL for your Apps.
cloudamqp     lemur, tiger, bunny, rabbit, panda             Managed HA RabbitMQ servers in the cloud
cloudforge    free, standard, pro                             Development Tools In The Cloud
elephantsql   turtle, panda, hippo, elephant                 PostgreSQL as a Service
ironmq        pro_platinum, pro_gold, large, medium, small, pro_silver  Powerful Durable Message Queueing Service
ironworker    large, pro_gold, pro_platinum, pro_silver, small, medium  Scalable Background and Async Processing
loadimpact    lifree, li100, li500, li1000                   Automated and on-demand performance testing
memcachedcloud  25mb, 100mb, 250mb, 500mb, 1gb, 2-5gb, 5gb      Enterprise-Class Memcached for Developers
mongolab      sandbox                                         Fully-managed MongoDB-as-a-Service
newrelic      standard                                       Manage and monitor your apps
rediscloud    25mb, 100mb, 250mb, 500mb, 1gb, 2-5gb, 5gb, 10gb, 50gb  Enterprise-Class Redis for Developers
searchify     small, plus, pro                               Custom search you control
searchly      small, micro, professional, advanced, starter, business, enterprise  Search Made Simple. Powered-by Elasticsearch
sendgrid      free, bronze, silver, gold, platinum           Email Delivery. Simplified.
```

Figure 7.1.

2. Provision a new MySQL service instance. On PWS this service is known as “cleardb”.
 - a. Use the “spark” plan since our storage requirements are modest.
 - b. Give the service any name you like, but remember the name for later.

```
kkrueger-mbp15:~ kkrueger$ cf create-service cleardb spark mydb
Creating service mydb in org kkrueger-net / space development as kkrueger@gopivotal.com...
OK
```

Figure 7.2.

3. Bind spring-music to the service you have just provisioned. If you don't know where to get "Spring Music" ask your Instructor.

```
kkrueger-mbp15:~ kkrueger$ cf bind-service smkk mydb
Binding service mydb to app smkk in org kkrueger-net / space development as kkrueger@gopivotal.com...
OK
TIP: Use 'cf push' to ensure your env variable changes take effect
```

Figure 7.3.

4. Restage the app to make sure it uses the new service instance.
5. Use a browser to open the application's URL. Navigate through the application and see there is data there. (Note that this application has an ‘embedded’ database that is used when no other DB is available, this is why it worked before the service was bound. So how can we tell if it is using the ‘bound’ database or the

‘embedded’ database? Next step...)

6. Obtain the DB credentials. You can do this via the Apps Manager: Find the details of your application, go to Bound Services, click on credentials.
7. Open your database client, create a MySQL connection and point it to the MySQL you just provisioned. Examine the table data containing the music albums.



Note

To do this step, the default MySQL port 3306 must be accessible. Your company firewall may block access to this port.

8. Alter the music data using your database client. Refresh the web page and notice the changes are present. Alternatively, alter the album data using the web application and use the database client to observe the changes.

Congratulations, you have finished this exercise.

Chapter 8. Service Brokers

8.1. Goal

In this section, students will deploy a custom service broker.

1. The `cf-haash-broker` implements a `HashMap` as a `Service`
2. Deploy the Service Broker as an app to Cloud Foundry
3. Register the Broker with the Cloud Controller
4. Make the single plan in the catalog “public”
5. Create an instance of your service
6. Push and bind a test app to your service instance
7. Test the app’s interaction with the service

8.2. Exercise

1. Review the documentation on [Creating Custom Services](#)

8.2.1. Overview

The Service Broker for this lab is for a service called `HashMap as a Service` (HaaSh). It wraps a minimal REST API around a Java `HashMap` implementation, and each service instance creation event results in a newly allocated `Map`.

The service-broker is in the folder `cf-haash-broker` and a client that will use that service can be found in `cf-haash-client`. The code for both has been written for you, you just have to configure and deploy them.

These instructions just require the `cf` command-line tool, a Terminal or CMD window and your favorite text-editor

- Windows: Notepad or Wordpad will do. Notepad++ (<https://notepad-plus-plus.org/>) is better.

- On MacOS or Linux: You can use `vi` (if you know it) or whatever GUI editor you have available.

8.2.1.1. Using an IDE (Optional)

Alternatively you might prefer to use an IDE.

To use Eclipse or STS, you'll need to install the Gradle plugin first. You can find it under `Help#Eclipse Marketplace`, then search for Gradle. The "Gradle Integration for Eclipse 3.7.xxx" by Pivotal works well.

Once the Gradle plugin is installed, you use `File#Import#Gradle#Gradle Project`, then select the project folder to import. Then click the "Build Model" button, select the project, and click Finish.

8.2.2. Prerequisites

If you do not have one already, open a Terminal window (MacOS, Linux) or CMD window (Windows).

If you have not already done so, download the `CloudFoundryStudentFiles` from github.

- If you have `git` installed, change to a suitable directory and run `git clone https://github.com/S2EDU/CloudFoundryStudentFiles`
- Otherwise, goto <https://github.com/S2EDU/CloudFoundryStudentFiles> and on the right-hand side click the Download ZIP button. Then unzip in a suitable location.

In your Terminal/CMD window, make `CloudFoundryStudentFiles` your current directory.

If not using an IDE, the Gradle build tool is required. If you do not have gradle installed, there is a script in `CloudFoundryStudentFiles` (`gradlew` or `gradlew.bat`) that loads Gradle from the Internet and then runs a build.

If this course is running on-site, you must be able to access Maven repositories through your corporate firewall. If this is not possible, skip this lab.

8.2.3. Configure the Haash Broker

- Change into the `cf-haash-broker` project directory under `CloudFoundryStudentFiles`.
- Edit the `src/main/resources/import.sql` file and replace each occurrence of `XX` with your initials, like this:

```
insert into services (id, name, description, bindable) values ('<first-initial><last-initial>', '<first-initial><last-initial>', '<first-initial><last-initial>');
insert into plans (id, name, description, service_id) values ('<first-initial><last-initial>', 'basic', 'Basic Plan', '<first-initial><last-initial>');
```


- Edit the `cf-haash-broker` dashboard client code, add your initials to the String `id` field. The `DashboardClient` class is in the `src/main/java/io/pivotal/pcf/haash/model` folder.

```
public class DashboardClient {  
    private final String id = "Haash-XX";  
    private final String redirectURI = "http://example.com";  
    private final String secret = "secret";  
}
```

- Build the Haash Broker - in the `cf-haash-broker` folder by running `gradle assemble`.
 - If you do not have Gradle installed use the script mentioned in prerequisites: This script takes a long time to run the first time due to the downloads. So, in your Terminal/CMD window run `../gradlew assemble` (Linux/MacOS) or `..\gradlew.bat` (Window).
- Push the broker as an app to Cloud Foundry using `cf`
 - Use the name: `<first-initial><last-initial>-haash-broker`
 - The file you need to deploy is `build/libs/haash-broker-0.0.1-SNAPSHOT.jar`.
- Register the broker with the cloud controller. You must first login as the admin user to run this command, using the password from the Elastic Runtime tile in Ops Manager, from the credentials tab for UAA/admin.

```
$ cf create-service-broker ...
```

For the service broker name, use your initials `<first initial><last initial>-broker`. The username & password you need to create the service broker are in: `src/main/resources/application.properties`

- Next, you need to make the plan public:

```
$ cf enable-service-access ...
```

You will need to figure out the service and plan names. You can do this by running the following command:

```
$ cf service-access
```

- Verify you can see your new service in the marketplace. There is a `cf` command for this.
- Create a new instance of your service using the CF commands. Name the instance `<first-initial><last-initial>-haash-service`.

8.2.4. Running the Haash Client

- Change into the `cf-haash-client` directory.
- There is no need to build the Haash Client application. It is already built for you and the jar you need to deploy is in the `pre-built` directory.
- Create a manifest for your Haash Client.
- Be sure to bind the Haash Client to the service instance you created above.
- Push the Haash Client to Cloud Foundry and name it: `<first-initial><last-initial>-haash-client`
- To access your client you need to be able to send HTTP GET and PUT requests.
 - Use a browser plugin: RESTClient for Firefox or Postman for Chrome. There is also a "REST Client" plug-in for Eclipse/STS.
 - Command Line: MacOS and Linux users can conveniently do this using `curl` or `wget`. Windows users can download `wget.exe` from <https://eternallybored.org/misc/wget/>.
- Access your haash client to be sure it is working and using your haash-service
 - Using `curl`, you can add key,value pairs using:

```
$ curl <your haash-client>/HaaSh/<key> -d '<some value>' -X PUT
```

- Now retrieve the values:

```
$ curl <your haash-client>/HaaSh/<key>
```

- Or using `wget`:

```
$ wget <your haash-client>/HaaSh/<key> --body-data '<some value>' --method PUT
$ wget <your haash-client>/HaaSh/<key>
```

8.3. Beyond the class

There are many community resources to help you build your own more robust service broker.

- Spring developers can use the [Spring Boot CF Service Broker](#)
- You following repositories can be used as a reference:

- [S3 CF Service Broker](#)
- [Spring Boot CF Service Broker Mongo](#)

Chapter 9. Using a Buildpack

9.1. About this Lab

In this lab you will learn how to deploy a Static HTML site by using an external buildpack.

Prerequisites

To complete these steps, you will need:

- A Cloud Foundry account (either on your company's CF installation or on Pivotal Web Services)
- The Cloud Foundry student files
- The Cloud Foundry CLI installed

See the "Lab Setup" in the "Welcome, Agenda, and Setup" module for instructions.

Estimated time to complete: 15 minutes

9.2. Steps

Pushing a Static HTML Application

In this section you will push an existing static HTML application to Cloud Foundry. CF does not have a built-in buildpack for static HTML sites, so you will need to specify an external buildpack that utilizes Nginx.

1. Locate the `cf-static-demo` folder in `CloudFoundryStudentFiles` on your file system.
2. Edit the `manifest.yml`. Give your application a unique host, and specify `https://github.com/cloudfoundry-community/staticfile-buildpack.git` as the buildpack.
3. Use CLI `cf push` to push the application to your Cloud Foundry environment.



Note

Some administrators may choose to disallow external buildpacks, in which case you cannot specify an alternate selection. If this is the case in your environment, you may simply use Pivotal Web Services for this exercise.

4. When the push is complete, use a browser to access your running application. If it is not running, attempt to debug using `cf logs <app>--recent` and other techniques.

Congratulations, you have finished this exercise.

Chapter 10. Customizing a Buildpack

10.1. About this Lab

Goals

What you will learn:

- Fork the Java buildpack to make a minor change.

Prerequisites

To complete these steps you will need:

- A Cloud Foundry account (either on your company's CF installation or on Pivotal Web Services)
- The Cloud Foundry student files
- Either the Cloud Foundry CLI or an IDE installed (with Cloud Foundry support)
- Most importantly: a GitHub account. *Unlike other labs, a github account is **mandatory** for this lab.*

Estimated time to complete: 30 minutes

10.2. Steps

Forking the Java Buildpack

In this section you will fork the Java buildpack, make a minor change to the desired Java version, then use your modified buildpack to push an application to Cloud Foundry, and observe the results

1. Push an existing Java application (eg. spring-mvc-demo, spring-music, etc.) to Cloud Foundry (if needed).
 - a. Observe the console output during the push process. Note the version of Java being used. Our goal in this lab will be to specify a different Java version.
2. Fork the Java Buildpack.

- a. Using a browser, go to <https://github.com/cloudfoundry/java-buildpack> (if you are not signed into GitHub, do so now).
 - b. Click the “Fork” button to create a fork of this repository under your own GitHub account. (You may clone a copy of this repository locally, though it is not necessary for the steps here)
3. Alter the Java version
- a. While still in the browser viewing your forked copy of the Java buildpack, drill-down to `config/open_jdk_jre.yml`. Click the edit button to modify this file in place.
 - b. Find the line (near line 20) that indicates the Java version: `version: 1.8.0_+`
 - c. Alter this line to deliberately specify an older Java version: `version: 1.8.0_31`
 - d. Save your work using the ‘Commit’ button.
4. Re-push the application using the altered Buildpack
- a. This step is performed differently depending on whether you have an existing manifest file or not. Assuming you do:
 - i. Alter the manifest file. Add a line to indicate your new buildpack (change “yourGitId” and “java-buildpack” as needed to match). Be sure your indentation is correct:

```
buildpack: https://github.com/[yourGitId]/[java-buildpack]
```
 - ii. Re-push the application. If using Eclipse ensure that it is picking up the values you specified in the manifest.
 - b. If you did not have a manifest file, and you used the CLI, repush using the `-b` option to specify the buildpack (alter “yourGitId” and “java-buildpack” as needed):

```
-b https://github.com/[yourGitId]/[java-buildpack]
```
 - c. Observe the console output during the push process. Note the version of Java being used. It should be the version you specified above.

Congratulations, you have finished this exercise.

Chapter 11. Integrating with 3rd Party Log Management Tools

11.1. About this Lab

Setup a Syslog drain to a PaperTrail, third-party log management solution.

11.2. Prerequisites

See the "Lab Setup" in the "Welcome, Agenda, and Setup" module for instructions.

Estimated time to complete: 25 minutes

11.3. Setup PaperTrail

1. Go to <https://papertrailapp.com> and establish a free trial account.
2. Once logged in, click on "Add System" to add a new source. Follow the instructions at <http://docs.cloudfoundry.org/devguide/services/log-management-thirdparty-svc.html#papertrail> to setup your project with the correct IP addresses.

11.4. Create and Bind User-Provided Service

1. Create a user-provided service using the command `cf cups log-drain -l syslog://<URL-FROM-PAPERTRAIL>`, replacing `<URL-FROM-PAPERTRAIL>` with the value provided by Papertrail. You may name the service whatever you like.
2. Bind the service to any of your existing applications used in prior labs.
3. Restart the application.
4. Access the URL of your application and generate logging activity. This varies depending on the application you are using; click on available links, refresh the web page, etc.

11.5. View Log Output

1. Return to Papertrail. Use the "Dashboard" button to find the main log output. Experiment with the search feature at the bottom of the page, look for terms such as "GET" or "hibernate".

Chapter 12. Integrating with APM tools

12.1. About this Lab

Integrate multiple instances of a running application with a third-party Application Performance Monitoring tool.

12.2. Prerequisites

To complete these steps you will need a Pivotal Web Services account. Another Cloud Foundry environment may or may not work depending on the existence of the New Relic service in the marketplace. Typically, Pivotal CF does not include such a marketplace service. See the "Lab Setup" in the "Welcome, Agenda, and Setup" module for instructions.

Estimated time to complete: 15 minutes

12.3. Configure New Relic APM Service

1. From the PWS Apps Manager, go to the marketplace and provision a new New Relic service instance. Name the instance anything you like (suggestion: "new-relic").
2. Determine if you already have the spring-mvc-demo app deployed on Cloud Foundry from an earlier lab. Make sure it has a minimum of **1 GB** of memory - otherwise it will crash. Push and/or scale it now. Bind it to the "new-relic" service you just provisioned. Restage the application. Scale it to 2 instances.
3. Once the application starts, open it's URL in a browser. Refresh it several times. Notice that the instance index, id, and port changes depending on which of the 2 instances you are hitting.
4. Return to the PWS Apps Manager. From the list of services in your application's space, locate the new-relic service and click on the "Manage" link.



Note

New Relic will use OAuth for authentication, so you will be prompted with a login screen from Pivotal Web Services. Essentially, it is verifying with PWS the account information.

5. Agree to the New Relic terms of service, or temporarily defer.

6. You should now see a page that lists your Cloud Foundry applications (only one should be in the list at the moment, the `spring-mvc-demo` app). Click on this entry to obtain detailed monitoring information. (Note that it may take several moments for detailed logging information to appear.)
7. If you have reached this point, congratulations! You have successfully bound your application to an APM via a Cloud Foundry service. If you have time, investigate the APM's capabilities. (Note that the specific capabilities of the New Relic APM is outside the scope of this course.)

Chapter 13. Blue/Green Deployment

13.1. About this Lab

Goals

What you will learn:

- Practice performing a zero-downtime deployment

Prerequisites

To complete these steps, you will need:

- A Cloud Foundry account (either on your company's CF installation or on Pivotal Web Services)
- The Cloud Foundry student files
- The Cloud Foundry CLI installed

See the "Lab Setup" in the "Welcome, Agenda, and Setup" module for setup details. Note that the instructions are written from the perspective of Pivotal Web Services, but will function with on any CF instance with minor adjustment.

Estimated time to complete: 20 minutes

13.2. Steps

13.2.1. Push "Blue"

Push the cf_bluegreen application to Cloud Foundry using the CLI (do not use an IDE for this first step).

1. Give your app the name "blue" for the application's name. 512M should be sufficient.
2. Give your app a host name of "XXX-always-available", except replace "XXX" with some value to ensure a unique route. Remember this host for later steps.

3. For the path, specify the "pre-built/blue.war". (This WAR file has been built for you using Maven and the project's source files.)
4. After the application starts, open a browser and access the app: `http://XXX-always-available.cfapps.io` (except using the URL based on your host name). Notice the large blue box on the screen. Leave this browser window open.

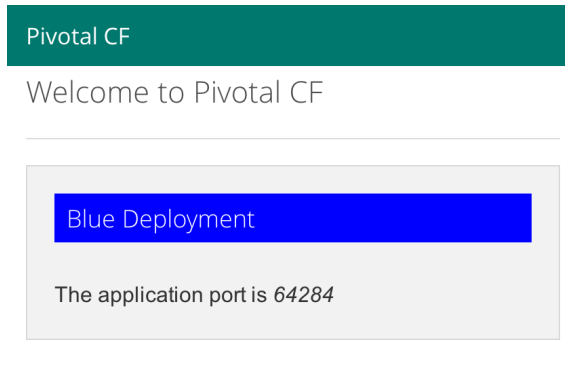


Figure 13.1.

13.2.2. Alter the application

1. Using your IDE, open the `src/main/resources/templates/index.html` page.
2. Find the tag containing "Blue Deployment". Comment this tag out using XML comment `<!-- -->` tags.
3. Notice the tag containing "Green Deployment" that is presently commented out. Remove the comments. Save your work.
4. If using Eclipse, the application will be built for you automatically and you are ready for the next step.

13.2.3. Push "Green"

Push a separate copy of the `cf_bluegreen` application to Cloud Foundry

1. For this step, we recommend you use an IDE to avoid the need to build the Java application. If you prefer to

use the CLI, you will need to build the application using Maven, which you may not have installed ("mvn clean install" is the command). If for any reason you cannot build the application, an existing WAR "green.war" is available for you in the pre-built sub folder.

2. Push the app using "green" for the application's name. 512M should be sufficient. As always, make sure your deployed URL is unique by customizing the host.
3. After the application starts, open a separate browser window/tab and access the green app via its route. Notice the large green box on the screen.

13.2.4. Alter Routes

1. Return to the original browser window/tab. Refresh it multiple times, notice it is still pointing to the old 'blue' deployment.
2. Using either the CLI, IDE, or Apps Manager, map the same "XXX-always-available" to the "green" app. IF using the CLI you may receive a warning that the route is already in use; this is expected.
3. Return to the original browser window/tab. Refresh this page several times. Notice that the same route alternates randomly between the 'blue' and 'green' apps; this is expected since the route is temporarily mapped to both applications.
4. Alter the blue app by removing its route.
5. Refresh the browser. The browser is now pointing to the new green instance with no downtime.

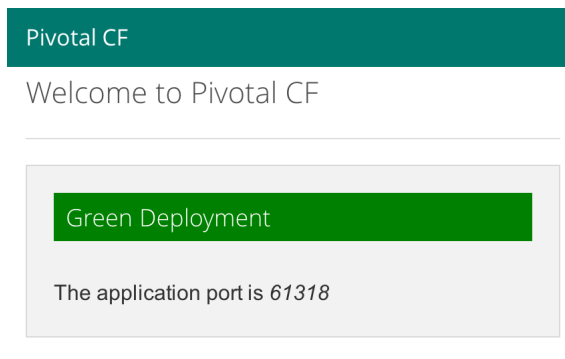


Figure 13.2.

The conclusion: no downtime would be experienced by any user of the `cf_bluegreen` application when accessed via its "always-reliable" route. Congratulations! You have conducted a zero-downtime deployment! You can stop and/or delete both applications now.

Chapter 14. Continuous Delivery

14.1. About this Lab

Setup and run a basic Continuous Delivery pipeline with Git, Jenkins, and Cloud Foundry.

14.2. Prerequisites

To complete these steps you will need a Cloud Foundry account (either on your company's CF installation or on Pivotal Web Services) and a GitHub account. See the "Lab Setup" in the "Welcome, Agenda, and Setup" module for instructions. Unlike other labs, a github account is *mandatory* for this lab.

14.3. Setup Git Project

In this section you will fork an existing Java/Maven project into your personal GitHub repository. We will then use this project as something to deploy to CF via a Continuous Delivery pipeline.

1. If you haven't already, locate the "Cloud Foundry Student Files" example project on GitHub: <https://github.com/S2EDU/CloudFoundryStudentFiles>
2. Use the "Fork" button to make your own private copy of the project on your own GitHub repository.

14.4. Install Jenkins

1. Navigate to <https://jenkins-ci.org/> and on the right side of the page download the installer for your OS.
2. Run the installer.
3. On completion of the install, you should see the Jenkins dashboard appear in your browser at <http://localhost:8080>.

Alternatively, feel free to download the .war file. It comes with a built-in web server and can be started with "java -jar jenkins.war." When you're done with the lab, simply stop jenkins with a Ctrl-C and throw away the .war file.

14.5. Configure Jenkins

1. Click on the "Manage Jenkins" link, then click the "Manage Plugins" link.

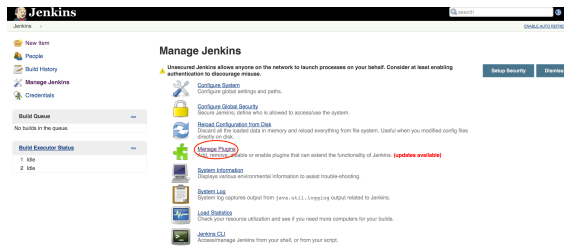


Figure 14.1. Manage Jenkins

2. Click on the "Available" tab, and find the GitHub plugin. (You can search using the Filter box in the top right corner.) Select it and click "Install without restart".

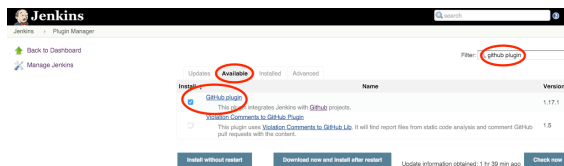


Figure 14.2. Github Plugin

3. Also install the Cloud Foundry Plugin.

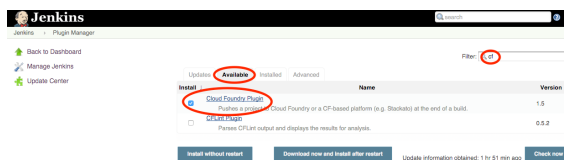


Figure 14.3. Cloud Foundry Plugin

4. From the "Manage Jenkins" page, click on "Configure System". Scroll down to the "Maven" section, and click the "Add Maven" button.

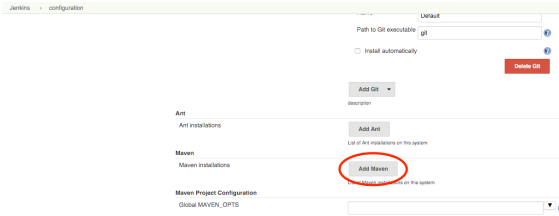


Figure 14.4. Add Maven Support

Enter a name for the Maven installation, and leave the "Install automatically" box checked.

5. Save your changes.

14.6. Create the Initial Build Job

1. From the Jenkins dashboard, click "New Item" (in top left corner), give it a name and select "Build a Maven project" Then click "OK".
2. Under "Source Code Management", select "Git", and supply your forked repository URL.

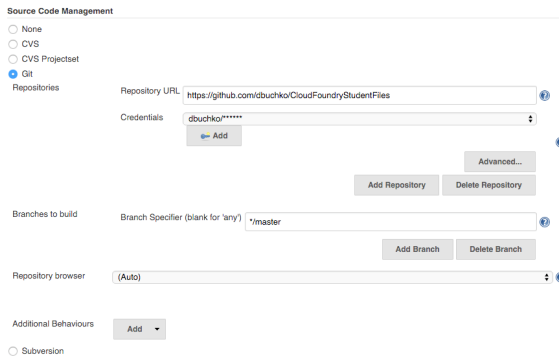


Figure 14.5. Configure GitHub Repository URL

3. Under "Build Triggers", select "Poll SCM". In the "Schedule", enter the CRON formatted string such as `H/5 * * * *`. This will poll your Github repository every 5 minutes for changes, and if any are detected, will

execute the build.

Build Triggers

- ☐ Build whenever a SNAPSHOT dependency is built
- ☐ Build after other projects are built
- ☐ Build periodically
- ☐ Build when a change is pushed to GitHub
- ☒ Poll SCM

Schedule

H/5 * * * * *

Would last have run at Tuesday, March 15, 2016 10:30:20 o'clock AM EDT; would next run at Tuesday, March 15, 2016 10:35:20 o'clock AM EDT.

Ignore post-commit hooks ☐

Figure 14.6. Configure Build Triggers

- Under "Build", add the project path to the "Root POM", so it becomes `cf-cloud-info-solution/pom.xml`.

Build

Root POM

cf-cloud-info-solution/pom.xml

Goals and options

Advanced...

Figure 14.7. Configure Path to mvn pom.xml file

- Under "Post Steps", select the radio button "Run only if build succeeds".
- Under "Post-build Actions", click the "Add post-build action", and select "Push to Cloud Foundry".
- Fill in the parameters to target and log into the Cloud Foundry instance you'll be using. You will have to add your credentials. Test the connection to make sure you can connect.

Post-build Actions

☒ Push to Cloud Foundry

Target

https://api.run.pivotal.io

Credentials

dbuchko@pivotal.io*****

Organization

pivotalcd

Space

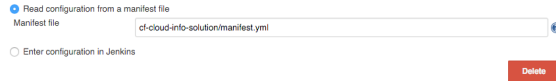
dbuchko


Allow self-signed certificate ☐

Test Connection

Figure 14.8. Configure Post-Build Actions

- Make sure that "Read configuration from a manifest file" is selected, and change the path to the manifest file to `cf-cloud-info-solution/manifest.yml`.



☒ Read configuration from a manifest file
Manifest file 

☐ Enter configuration in Jenkins

Figure 14.9. CF Push Configuration

9. Save the config and try running the build by clicking "Build Now". The first build will fail, because the Jenkins Cloud Foundry plugin does not recognize the `${random-word}` directive in the manifest.
 10. In your forked repo, edit the manifest file in the `cf-cloud-info-solution` directory and replace the `${random-word}` in the host parameter with a unique value using your own initials. eg. `cloud-info-solution-db123` where `db123` are your initials with a random number. Commit the change to git, wait until the polling detects it, and watch the magic. Verify the build in Jenkins now succeeds.
 11. Log in to your PWS console, and verify that the application has been deployed to your space, and is running.
- Congratulations, you have finished this exercise!

Chapter 15. Session Management

15.1. About this Lab

In this lab you will learn how to enable an HTTP Session to survive instance failure

Prerequisites

To complete these steps, you will need a Cloud Foundry account, the Cloud Foundry student files, the Cloud Foundry CLI installed, and an IDE installed (with Cloud Foundry support). If you are missing any of these return to the "Lab Setup" in the "Welcome, Agenda, and Setup" module for instructions.

Estimated time to complete: 20 minutes

15.2. Steps

You will be given an application that uses the `HttpSession` which will lose information if restarted. Then you will fix it in Cloud Foundry by adding a Redis session replication service.

15.2.1. Push

Locate the `cf-session-management` application and push it to Cloud Foundry. You may use CLI or Eclipse.

1. If using the CLI, edit the `manifest.yml` and specify a unique `host` (add your name or initials).
2. Once it's running, go to its URL in the browser to see it working. The application is very simple. Enter a name and description (anything you like) and click "Submit". The name and description are displayed in the results page.
3. Refresh the page several times. Notice that the values are retained. You can examine the code if your are curious how the session is used.

15.2.2. Restart Application

1. Click on the "Stop!" link to kill the application (the code behind this actually kills the JVM, simulating a real application crash).

2. Go back to the previous page in the browser and refresh that page. You will get a CF error:

```
404 Not Found: Requested route ('cf-session-demo2.cfapps.io') does not exist.
```

3. Wait a moment for Cloud Foundry's Health Manager to notice the missing instance and cause it to be restarted. You can monitor the restart via the CLI, Eclipse plugin, or Apps Manager if you like.
4. Refresh the page again, note that the previously submitted data has been lost. This is expected since the session data is stored in memory; it is lost when the application instance stops. In the next steps we will use a service to cause the session data to be persisted external to the instance.

15.2.3. Bind the Redis session-replication Service

1. Using either the CLI, Apps Manager, or Eclipse, provision a new Redis service instance `session-replication`. You must use this name for the service instance to obtain the desired session behavior.

If you are running on PWS the service is called `rediscloud`. Select the 30mb plan (it's free!).

2. Bind the service to the application. You can do this via CLI, Apps Manager, or Eclipse.
3. Push the application to Cloud Foundry again. Though the application code did not change, we need to force the application to be restaged. (If using Eclipse you can use the "Update and Restart" button as a shortcut).



Note

You must run `cf push` again or the new service will not be used.

15.2.4. Retry Killing the Application

1. Return to your browser and check the application is working again. Go to the Home page and re-enter some name and description data.
2. Redo all the steps in [Section 15.2.2. "Restart Application"](#). However, this time the name and description should be retained by the Redis-backed session even after the application instance has been stopped.

15.2.5. Examine the Application

1. If you have time, take a few minutes to examine the application code. This is a Spring Boot application using

Spring MVC Controllers. You might like to look at the code to see how it works. In particular: `FormController` and `DisplayController`. The `FormController` loads values into the session and the `DisplayController` pulls them out.

The Spring Boot initialization class is `ConfigApplication` defines `main()` and `WebApplication` allows this application to also run as a WAR.

Congratulations, you have finished this exercise.

Chapter 16. Services and Environment in Java Applications

16.1. About this Lab

Goals

What you will learn:

1. Access Cloud specific application data

Prerequisites

To complete these steps, you will need:

- A Cloud Foundry account (either on your company's CF installation or on Pivotal Web Services)
- The Cloud Foundry student files
- The Cloud Foundry CLI installed
- An IDE installed (with Cloud Foundry support)

See the "Lab Setup" in the "Welcome, Agenda, and Setup" module for instructions.

Estimated time to complete: 20 minutes

16.2. Steps

In this lab you will implement a simple one-page web-site that displays information about the execution environment. By using Spring Profiles, the application should run both locally and on Cloud Foundry.

1. Import the `cf-cloud-info` application into your IDE as an existing Maven project (if you have not done so already). You will find it in the `CloudFoundryStudentFiles` folder.
2. There is no Cloud meta-data when running locally and no profile defined either. The profile "default" is activated automatically if no other profile is explicitly specified.

Look in `io.pivotal.cf.ConfigDefault` and you will see three Java Config @Bean methods defined. These return empty objects containing 'null' information when no cloud environment is detected. (TODO 01).

3. Run the application locally as a Spring Boot application - in Eclipse/STS right-click on the project and select `Run As ... -> Spring Boot App`. Then go to <http://localhost:8080>. You should see a page containing environment information, but the rest of the page below is empty. (TODO 02)



Note

If you are interested in the Spring Boot setup, look in `io.pivotal.cf.Application`. It's as simple as that!

4. Let's make this application cloud aware!

You need to modify the `io.pivotal.cf.ConfigCloud` to return Cloud related information (TODO 03-06). This class should define the same beans `io.pivotal.cf.ConfigDefault` but use the `Cloud` instance to get the data - invoke `cloud()` to get it.

5. Now look at the `io.pivotal.cf.web.InfoController` to see how this data is being used. (TODO 07). Thanks to the use of profiles, it has no knowledge of Cloud or otherwise.
6. Rerun the application locally as a Spring Boot application and check that it still works. If you have got your profile setup wrong you might get null-pointer errors because `cloud()` returns null.
7. If it works, deploy to Cloud Foundry and open the application URL in the browser. Make sure you specify a unique sub-domain (use your name, initials, anything). You should get a lot more information now. (TODO-08) Only Services is empty (we haven't bound any).
8. **BONUS:** Bind a service (any service, it won't be used) to the application and redeploy. Refresh the web-page in your browser and you should see some service information too. (TODO-09)
9. A working solution is provided by the `cf-cloud-info-solution` project.

Congratulations, you have finished this exercise.

Chapter 17. Appendix: Treasure Hunt Solution

17.1. Answers

Here are the answers to the questions in the Treasure Hunt (documentation) lab:

1. Java, Node.js, Ruby, Go, PHP, Python.
2. Application name, buildpack, command, domain, instances, memory, host, timeout, anything described in: <http://docs.cloudfoundry.org/devguide/deploy-apps/manifest.html>
3. Avoid local file system, avoid Http session, avoid non-standard ports, avoid pushing unneeded files.
4. Router, Cloud Controller, DEA/Cell and many more ... See: <http://docs.cloudfoundry.org/concepts/architecture>
5. API, STG, DEA, RTR, LGR, APP, all defined here: <http://docs.cloudfoundry.org/devguide/deploy-apps/streaming-logs.html>
6. Orgs, spaces, users, ...
7. Allows you to extend CF to handle other languages and/or frameworks.
8. See: <http://docs.run.pivotal.io/marketplace/services> - cleardb, cloudamqp elephantsql, mongolab, newrelic sendgrid, etc.
9. Detects whether or not a buildpack can be applied to a given application.
10. weeble.cfapps.io