CLOUDFOUNDRY

Pivotal Cloud Foundry Developer

Lab Instructions

Application Deployment using Pivotal Cloud Foundry

Pivotal

Version 1.12.a

Copyright Notice

Version 1.12.a

2018

Table of Contents

Requirements

1. Personal Experience

- Background in Software Development
- At ease with the command-line or "shell"
- Java/Spring experience is optional

2. Pivotal Cloud Foundry Environment

You will need a Pivotal Cloud Foundry environment to execute the labs. You may use a Pivotal Cloud Foundry instance or Pivotal Web Services.

If taking the course via e-learning, Pivotal Web Services is recommended, unless you have access to another Pivotal Cloud Foundry instance (typically already provisioned by your company). It is not recommended that you provision a personal Pivotal Cloud Foundry environment to complete the labs. The setup and installation of Pivotal Cloud Foundry is beyond the scope of this course.

If taking the course via instructor-led training, your instructor will provide guidance on the environment to use.

Note

If taking this course via e-learning, use a Pivotal Cloud Foundry account that your admin has setup for you (choose the `org` and `space` that works for you) or use Pivotal Web Services.

Throughout the labs you will see specific directions for Pivotal Cloud Foundry or Pivotal Web Services. *Make sure to follow the directions for the right environment.*

2.1. Option A: Pivotal Cloud Foundry

Pivotal Cloud Foundry may be used to complete these labs. The following dependencies are required:

- PCF Elastic Runtime version 1.8 or higher, with the following "tiles" added:
  - MySQL for PCF

- PCF Metrics
- Redis for PCF

Tip

If using self-signed certificates, under the `Elastic Runtime Tile` -> `Settings` -> `Networking` page make sure the following options have been checked:

- `Disable SSL certificate verification for this environment`
- `Ignore SSL certificate verification on route services.`

2.2. Option B: Pivotal Web Services

Pivotal Web Services may be used to complete these labs. Simply sign up for an account.

2.3. A Note About the Service Broker Lab

The Service Broker lab requires access to a MongoDB instance. Please review that lab for alternatives on how to meet that requirement. Options include using the AWS MongoDB AMI.

3. Local Machine

Minimum specification

- Windows, MacOS or Linux
- 4GB Memory
- 1.5G free disk space - if running MS Windows, 1.5G on your C: drive is preferred.
- Network access to the PCF environment and internet. If you have a HTTP Proxy Server on your network, see here.

You will need the ability to install software on your local machine. The following should be installed *prior* to the course:

- Java JDK is required for a few of the labs
- Cloud Foundry Command Line Interface (CLI). (or download the latest version from here)
- Apache JMeter is provided for you and is required by the Auto Scaler Lab. (or download the latest version from here)
- A programmer-friendly text editor. Atom or Notepad++ (for Windows only) are popular options

3.1. Optional

- JSON Viewer (browser plugin)
  - JSON Formatter for Chrome
  - JSON Formatter for Firefox - you can download it directly from Mozilla
- Command Line HTTP client
  - Mac and Linux users already have `curl`. Windows Users can find an installer in the lab materials (or download the latest version here).
  - Alternatively, HTTPie is a nicer client if you prefer.
- Windows users may desire a better terminal experience than the *cmd.exe*.
  - Consider using Powershell (open a CMD windows and type "powershell" at the prompt).
  - Or use ConEmu. It provides multiple tabs and search which is helpful during these labs.
- A Java IDE such as the Spring Tool Suite is optional but can be useful for reading the implementation of the service broker and route service that we work with in those respective labs
- The Continuous Delivery Lab provides the option to use a pre-provisioned Jenkins instance or to install everything locally. If you choose the latter option, then you will also need git and Maven

4. Windows VM

An alternative to installing and configuring the software on your local machine is to use our Windows VM.

If using the Windows VM see these requirements.

Approach and Methodology

1. Approach

After completing this course, you will be prepared to deliver applications with Pivotal Cloud Foundry.

This course is designed to challenge you. In this course, you will learn by doing. For this reason, the course is approximately 1/3 slides and 2/3 labs.

The course makes extensive use of the official product documentation to supplement your learning. Pivotal Cloud Foundry is a fast moving project and it is imperative that you learn where to go for help. The expectation is that you read documentation to complete the labs and answer inline questions (in the labs).

Explore! Dive into the cf CLI and all facets of Pivotal Cloud Foundry.

2. Methodology

The flow of each module will move from a presentation (if applicable) to lab then a recap.

During the presentations you will learn about high level concepts, then move into a lab where you can see things happen in action.

The recap will bring it back together.

Chapter 1. Push to the Cloud

Estimated Time: 25 minutes

1.1. Preface

This lab incorporates many preliminary steps. You'll be learning about and accessing the "Apps Manager" for the first time.

If necessary, you'll download and install the *cf* command line tool (the CLI), and use it to login to a Pivotal Cloud Foundry instance. You'll also learn about the concept of the cf *target*.

You'll deploy your first set of simple applications to cloudfoundry with the `cf push` command.

Although this lab is an introduction to Pivotal Cloud Foundry, you'll learn a great deal in a fairly short span of time.

Enjoy.

1.2. Setup

1.2.1. Download the `cf` CLI

If you have already done this, skip to the next section.

Endpoint and credential information for accessing Apps Manager will be provided by your instructor. Apps Manager is a web application that helps you manage your applications, but it is also the place to download the `cf` CLI. The next set of steps walk you through just that.

Note

If you are consuming these materials via e-learning and using Pivotal Web Services you can login to Apps Manager here. If you are using a Pivotal Cloud Foundry instance please see your system administrators for access to Apps Manager.

1. Review the Apps Manager Documentation. Knowing where to go for help is essential.

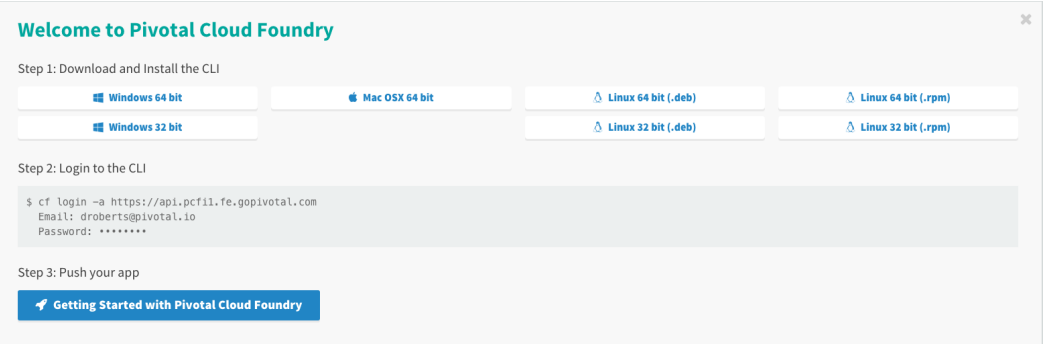
2. Login into Pivotal Cloud Foundry with the Apps Manager. If using Pivotal Web Services (PWS), the url is:

https://console.run.pivotal.io/

Note

It is common to use self signed certs in educational environments, but doing so will result in warnings from your browser. These warnings can safely be ignored (proceed through them).

3. Upon logging into Apps Manager for the first time you will be greeted with a welcome message that will walk you through the installation of the `cf` CLI.



If you do not see this dialog, the same information can be found under `Tools` (Left side).

4. Download and install the `cf` CLI for your platform.

5. Open a terminal window. Explore the `cf` CLI. In the terminal window type `cf`.

```
cf
```

6. Review the commands available with the `cf` CLI. Help on a specific command can be found using the `--help` option (e.g. `cf login --help`).

```
cf login --help
```

1.2.2. Login to Pivotal Cloud Foundry with the `cf` CLI

If you have already done this, skip to the next section.

1. Continue to follow directions for logging in with the `cf` CLI. You can copy and paste the `cf login` command directly from Apps Manager.

```
cf login -a {{api_endpoint}}
```


Note

Are SSL errors preventing you from logging in? Try adding the `--skip-ssl-validation` option.

2. Review your current API endpoint, user, org, and space

```
cf target
```

What Just Happened?

You have logged into Pivotal Cloud Foundry from two different clients (Apps Manager and the `cf` CLI). You have installed the `cf` CLI and targeted a Pivotal Cloud Foundry instance. You are ready to deploy (`push`) applications.

1.3. Exercises

1.3.1. Pushing apps

Next we will push (deploy) several applications.

1. You should have been given a zip file containing most of the lab files needed by this course: `pcf-developer-XXX-labs.zip` (where `XXX` indicates the version of the course).
2. Unzip this file and copy `labs` directory somewhere convenient. We recommend:
   - **Unix, Linux:** Copy it to you home directory: `~/labs/`
   - **Windows:** Copy it to the root of you `C:\` drive: `C:\labs\`

Note

From now on in these notes, we will simply refer to the `…/labs` directory.

3. In `labs` directory you will find a `demo-apps` directory.
   It contains four sub-directories; each directory contains an application developed in a different language.
4. Open a terminal window. First `cd` into the `node-app` directory:


```
cd .../labs/demo-apps/node-app
```

The CloudFoundry `push` command assumes that the artifacts for your application are located in the current working directory.

Important

`node-app` *must* be your current directory, not `demo-apps`. This is a common-mistake.

5. Now push the `node-app` application:

```
cf push node-app --random-route -m 128M
```

6. First, let's find out what our application url is, with the `cf apps` command:

```
cf apps
```

Now, visit the `node-app` application in your browser, or use `curl`:

Example:

```
curl {{application_url}}
```

The output should be a simple `Hello node` message.

7. Repeat steps 3 - 4 for the `php-app`, `python-app`, and `ruby-app` applications.

Important

Make sure you `cd` into each directory before pushing.

What Just Happened?

You just deployed four applications each based on a different language and runtime. Pivotal Cloud Foundry is a polyglot platform, meaning it supports multiple languages and does so in a pluggable way (via buildpacks)!

1.3.1.1. Questions

- What are some common items in the output that occurred when pushing each application?

1.3.2. Explore Apps Manager

1. Review the following views:
   - Org
   - Space
   - App

What Just Happened?

You have interfaced with Pivotal Cloud Foudry from two separate clients (`cf` and Apps Manager). Many of the operations that are available in `cf` CLI are also available in Apps Manager.

1.3.3. Clean up

1. Delete the applications you just pushed.

```
cf delete node-app
```

Warning

This is very important for resource constrained environments. On PWS, all your application instances together may only use 2G of memory, after which you will not be able to push. Stop some previous applications if necessary to reclaim memory.

Repeat to delete the `php-app`, `python-app`, and `ruby-app` applications.

1.4. Spring Music

Check out the Cloud Foundry sample applications.

Spring Music is a favorite.

1. `spring-music.war` is also included in your lab files (under `demo-apps/java-apps`) Push it to Cloud Foundry. To keep the memory usage down specify 768M.

Note

Remember to use `--random-route` or the `-n <hostname>` to ensure a unique URL.

2. Once it is running, view it in the App Manager.

3. Finally stop and delete the application.

Chapter 2. Logging, Scale, and HA

Estimated Time: 30 minutes

2.1. Preface

In this lab, we build on the foundation we've picked up from the first lab. We introduce new commands that are essential for developers working with cloudfoundry. You'll learn how to tail logs, view application events, and scale an application with ease. The last portion of the lab focuses on demonstrating one of the high availability characteristics of the cloud foundry platform: automatic restart of a failed application instance. Enjoy.

2.2. Exercises

2.2.1. Push the `articulate` application

1. The `articulate` application, `articulate-1.0.jar`, should be in your lab files `demo-apps/java-apps` sub-directory. If so, jump to step (3)

2. Push the `articulate` application.

```
Unix, Mac: cd ~/labs/demo-apps/java-apps

Windows:  cd C:\Users\<login-id>\labs\demo-apps\java-apps
```

```
cf push articulate -p articulate-1.0.jar -m 768M --random-route --no-start
```

   Notice how this time, we use the `-p` flag to tell the `cf` command what file to upload.

2.2.2. Access `articulate` logs

1. Review the documentation on application logging.

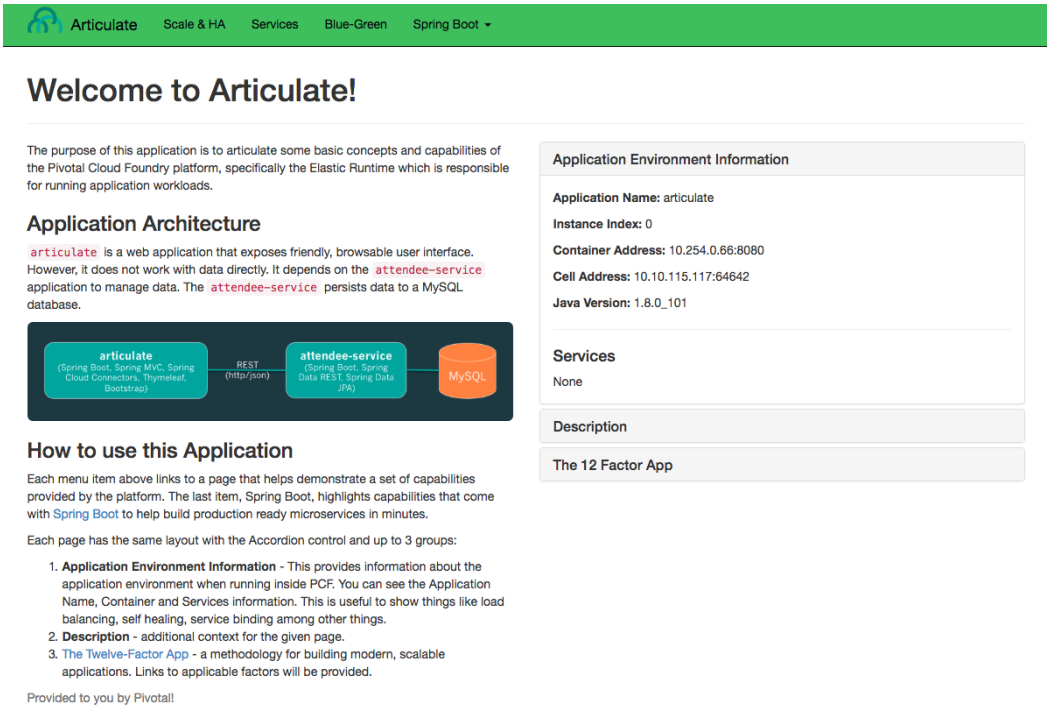
2. Tail the logs of the `articulate` application.

```
cf logs articulate
```

3. Open *another* terminal window and start the `articulate` application. Review the output from both terminal windows.


```
cf start articulate
```

4. Open a browser and view the `articulate` application. Read about our demo application.



5. Observe the log output when the `articulate` web page is refreshed. More logs are added!

6. Stop tailing logs

   a. Go to the terminal tailing the logs

   b. Send an interrupt (kbd:[Ctrl+C])

2.2.2.1. Questions

- Where should your application write logs?

- What are some of the different origin codes seen in the log?
- How does this change how you access logs today? At scale?

2.2.3. Access `articulate` events

Events for the application can also be used to complement the logs in determining what has occurred with an application.

```
cf events articulate
```

2.2.4. Scale `articulate`

2.2.4.1. Scale up

1. Start tailing the logs again.
   mac, linux | windows

```
cf logs articulate | grep "API\|CELL"
```

```
cf logs articulate | findstr "API CELL"
```

   The above statement filters only matching log lines from the Cloud Controller and Cell components.

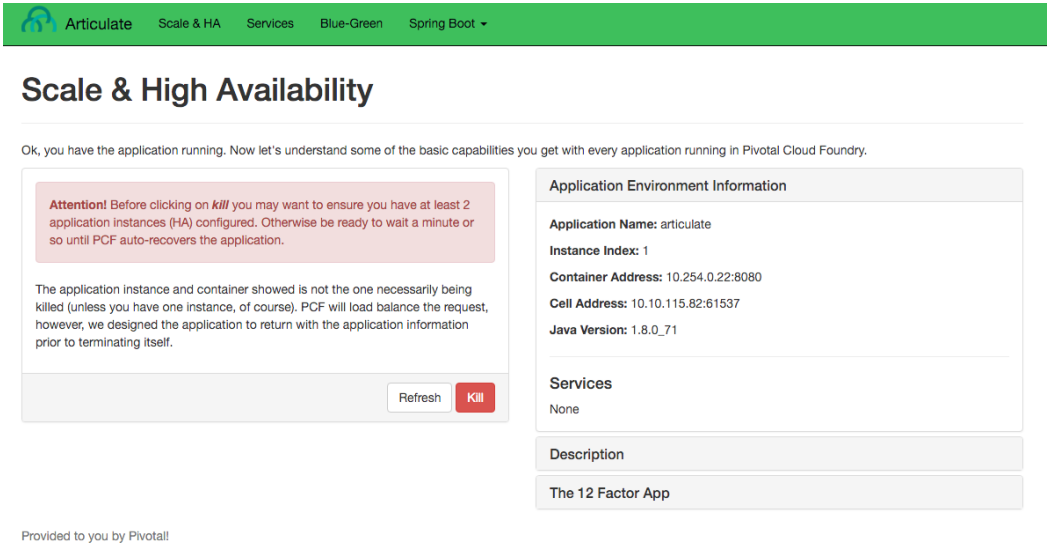
2. In *another* terminal window scale `articulate`.

```
cf scale articulate -m 1G
```

3. Observe log output.

2.2.4.2. Scale out

1. Browse to the `Scale and HA` page of the `articulate` application.



Review the `Application Environment Information`.

2. Press the `Refresh` button multiple times. All requests are going to one application instance.

3. Due to the 2GB memory limitation for PWS trial account, we need to reduce the memory footprint of `articulate` in order to run multiple instances. First reduce the JVM stack size. Remember to restage `articulate` after `set-env`.

```
cf set-env articulate JAVA_OPTS '-Xss228k'
```

4. Scale the `articulate` application with 512M memory and 3 instances.

```
cf scale articulate -m 512M -i 3
```

5. Observe log output. Then stop tailing the logs.

6. Return to `articulate` in a web browser. Press the `Refresh` button several times. Observe the `Addresses` and `Instance Index` changing.

Notice how quickly the new application instances are provisioned and subsequently load balanced!

2.2.4.3. Questions

- What is the difference between scaling out versus scaling up?

2.2.5. High Availability

Pivotal Cloud Foundry has 4 levels of HA (High Availability) that keep your applications and the underlying platform running. In this section, we will demonstrate one of them. Failed application instances will be recovered.

1. At this time you should be running multiple instances of `articulate`. Confirm this with the following command:

```
cf app articulate
```

2. Return to `articulate` in a web browser (`Scale and HA` page). Press the `Refresh` button. Confirm the application is running.

3. Kill the app. Press the `Kill` button!

4. Check the state of the app through the `cf` CLI.

```
cf app articulate
```

   Sample output below (notice the `requested state` vs actual `state`). In this case, Pivotal Cloud Foundry had already detected the failure and is starting a new instance.

```
name:              articulate
requested state:   started
instances:         3/3
usage:             512M x 3 instances
routes:            articulate-jl.cfapps.io
last uploaded:     Fri 17 Nov 15:02:59 +08 2017
stack:             cflinuxfs2
buildpack:         java_buildpack

     state     since                  cpu    memory           disk           details
#0   running   2017-11-17T07:31:29Z   0.1%   195.4M of 512M   156.7M of 1G
#1   running   2017-11-17T07:31:43Z   0.1%   200.9M of 512M   156.7M of 1G
#2   running   2017-11-17T07:31:37Z   0.1%   210.3M of 512M   156.7M of 1G
```

   Repeat this command as necessary until `state` = `running`.

5. In your browser, `Refresh` the `articulate` application.
   The app is back up!
   A new, healthy app instance has been automatically provisioned to replace the failing one.

6. View which instance was killed.

```
cf events articulate
```


7. Scale `articulate` back to our original settings.

```
cf scale articulate -i 1
```

2.2.5.1. Questions

- How do you recover failing application instances today?
- What effect does this have on your application design?
- How could you determine if your application has been crashing?

2.3. Beyond the class

- Try the same exercises, but using Apps Manager instead
- Visit the cloud foundry plugins site at https://plugins.cloudfoundry.org/ and investigate the *Open* plugin.

Chapter 3. Services

Estimated Time: 25 minutes

3.1. Preface

Up until now, we've focused on the deployment of applications. But as we know, applications often use backing services: databases, message brokers, and other applications' services (to name a few).

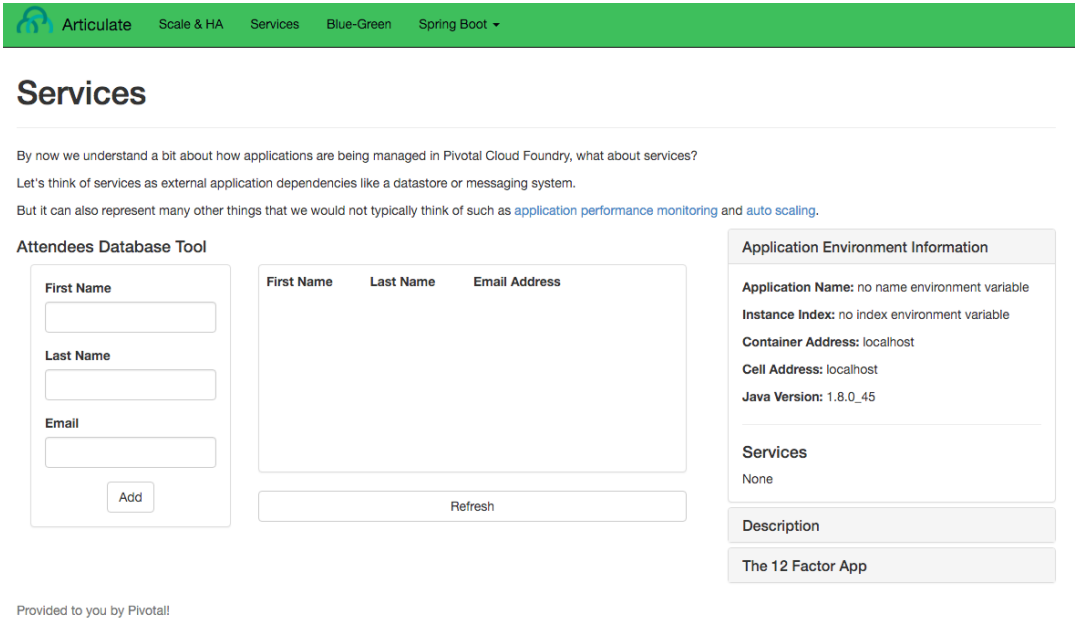
In this lab, you'll have the opportunity to experiment with both managed services and user-provided services: you'll create a mysql backing service to persist information about attendees, and you'll create a user-provided service to allow one application to consume the services of another without hard-coding its route.

In the process, you'll learn new cf commands: `create-service`, `create-user-provided-service`, and `bind-service`. Don't forget: `cf help` is there to help us understand how to invoke each command.

3.2. Exercises

3.2.1. Review `articulate` dependencies

`articulate` exposes functionality to add attendees on the *Services* page. However, `articulate` doesn't do this alone. It makes REST calls to the `attendee` application. To learn more about services, let's provision the `attendee` application.



3.2.2. Push the `attendee` application

1. The `attendee` application, `attendee-1.0.jar`, should be in your lab files `demo-apps/java-apps` sub-directory.

2. Push the `attendee` application.

```
cd ~/labs/demo-apps/java-apps
cf push attendee -p ./attendee-1.0.jar -m 768M --random-route
```

3.2.3. Add a Managed Service

1. Review the documentation on managing service instances

2. Review what services are available in the marketplace.

```
cf marketplace
```


3. There exist two distinct Cloud Foundry managed services that provision MySQL databases:

   a. p-mysql is a Pivotal product, and

   b. cleardb is a company that specializes in cloud-based MySQL instances
   Pivotal Web Services | Pivotal Cloud Foundry
   Pivotal Web Services offers the *cleardb* managed service in its marketplace.

```
cf create-service cleardb spark attendee-mysql
```

   If working in a Pivotal Cloud Foundry environment that is not PWS, you are likely to have the *p-mysql* managed service.

```
cf create-service p-mysql 100mb attendee-mysql
```

4. Now we need to bind the application with the service instance.

```
cf bind-service attendee attendee-mysql
```

Note

You can ignore the *TIP: Use 'cf restage attendee' to ensure your env variable changes take effect* message at this time.

5. Restart the application.

```
cf restart attendee
```

6. View the `attendee` in a browser.
   You should see a response similar to the following (the output in the screenshot is rendered by the JSON Formatter for Chrome):

   The root endpoint of this application is rather uninteresting. Instead, visit the `/attendees` endpoint. An http

GET to the `attendees` endpoint will fetch all attendees in the database and display them in JSON format. Note that in the response, the `totalElements` is currently 0.

This application implements a RESTful API. This means that you should be able to submit http POST to the same `attendees` endpoint with a body containing the JSON representation of the `Attendee` model type to create such a record. This can be done programmatically, or via REST client tools such as Postman or command-line tools such as curl or httpie. Here's an example using httpie:

```
http post {{attendees_app_uri}}/attendees firstName=Eitan lastName=Suez emailAddress=esuez@pivotal.io
```

How does this work?

1. Read about how twelve-factor apps handle backing services and configuration.
2. Read about VCAP_SERVICES.
3. View the environment for `attendee`.

```
cf env attendee
```

4. Different languages/frameworks will have various ways to read environment variables. `attendee` takes advantage of a Java Buildpack feature called Auto-Reconfiguration that will automatically re-write bean definitions to connect with services bound to an application.

3.2.3.1. [Optional] - Verify data in MySQL.

1. To verify that your attendees data are really inserted into MySQL database, install the CF CLI MySQL plugin.

```
cf install-plugin -r CF-Community "mysql-plugin"
```

2. Invoke the MySQL plugin for service instance `attendee-mysql`.

```
cf mysql attendee-mysql
```

3. Observe that the table `attendee` exists, and retrieve your attendees data via SQL.

```
mysql> show tables;

mysql> select * from attendee;
```

3.2.3.2. Questions

- After binding a service to an application why is the application restarted/restaged?
- In this case, why could we `restart` vs `restage`?

3.2.4. Add a User-Provided Service Instance

In the enterprise, not all services will be provisioned by Pivotal Cloud Foundry.

For example, consider your Oracle RAC cluster. How can we connect our applications running on Pivotal Cloud Foundry to these external systems?

Additionally, how can we easily connect applications together running on the platform?

`articulate's` default configuration for the `attendee uri` is `http://localhost:8181/` . The subsequent steps will allow you to override the default configuration with your own.

1. Read about user-provided service instances.
2. Create a user-provided service instance.

```
cf create-user-provided-service attendee-service -p uri
```

This will create an interactive prompt. For the value of `uri`, enter **your `attendee` application**'s base url:

```
uri> https://{{attendees_app_uri}}/
```

3. Bind `articulate` to the `attendee-service` user-provided service.

```
cf bind-service articulate attendee-service
```

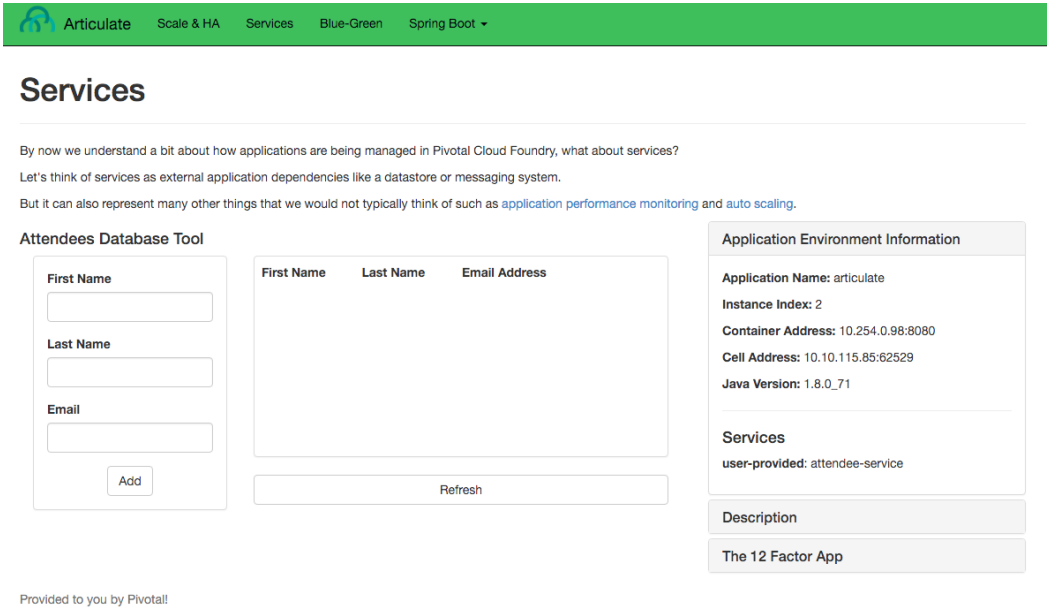
Note

You can ignore the *TIP: Use 'cf restage articulate' to ensure your env variable changes take effect* message at this time.

4. Restart the application.

```
cf restart articulate
```

5. Refresh the `articulate` *Services* page. You can now see the `attendee-service` listed under `Services`.



6. Review the environment.

```
cf env articulate
```

7. Add some attendees.

Note

If you can't add attendees, review the `articulate` logs and the user-provided service instance configuration.

3.2.4.1. Questions

- From an application perspective, are managed services instances different from user-provided service instances?

3.3. Beyond the class

- Use Spring Music and a User Provided Service Instance to connect to a database (MySQL or Oracle) in your environment.

Chapter 4. Manifest

Estimated Time: 10 minutes

4.1. Preface

You can imagine how tedious things would be if we had to repeatedly pass a handful of command arguments to the cf cli each time we push an application. There's a better way: application manifests allow you to consolidate into a single file all of our application's deployment metadata, including the application name, memory, path, hostname, and much more. We can associate services and environment variables to our application in a single manifest file.

Don't forget to take the time to read through the excellent documentation on manifests that's referenced below. There are lots of manifest file attributes you'll want to learn about. If you're new to yaml, take a little time to familiarize yourself with the syntax.

In this lab you'll reverse-engineer an application manifest file from one of the running applications you deployed in the last lab. That's a pretty convenient feature.

4.2. Reverse-Engineer a manifest from a running application

1. Read about application manifests

2. Change directories to the `java-apps`.

   Example:

```
Unix, Mac: cd ~/labs/demo-apps/java-apps

Windows:   cd C:\labs\demo-apps\java-apps
```

3. Generate a manifest.

```
cf create-app-manifest attendee -p ./manifest.yml
```

4. Edit the manifest.

   - Increase the `instances` to `2`.
   - Add a manifest attribute entry to set the path to the application's artifact (`attendee-1.0.jar`).

5. Push `attendee`.

```
cf push
```

6. Confirm the `attendee` is running and the latest push parameters are reflected.

```
cf app attendee
```

4.2.1. Questions

- Assuming you have application running with a bound service (MySQL). You `push` the application again with a manifest, but said service is not specified in the manifest. Is the service still bound to the application?
- What are some `cf` commands that don't have to be explicitly made when using manifests?
- Why did the `create-app-manifest` command not write the `path` attribute for me? In other words, why did I have to specify it manually?

4.2.2. Cleanup

1. Edit the manifest.
   - Decrease the `instances` to `1`
2. Push `attendee`.

```
cf push
```

4.3. Modify a manifest file

Now that we have a manifest file for the `attendee` application, it'd be nice to also have one for `articulate`.

1. This time, edit the manifest by hand. It's good practice, especially if you're not familiar with the yaml format. Remember, yaml files are indented with two spaces, and should not have any tabs.
   Use the manifest documentation as a reference.
   - Review the various manifest property options. Which should you specify?
   - You'll likely need to set the application name, path, what else?

- Also, make sure to specify the service binding directly in the manifest. That will save you the step of having to manually `bind-service` if you ever redeploy the application from scratch, or to a new space.

2. Once you're satisfied with your `manifest.yml`, test it with a `cf push` and see if it properly deploys your application as before.

There's much more to manifest files. You can..

- define multiple applications in a single manifest file
- setup one manifest file per environment, and use the -f flag with the `cf push` command to specify which to use
- there's even a mechanism for manifest files to inherit settings from another file

Congratulations! You've completed this lab.

Chapter 5. Application Security Groups

Estimated Time: 10 minutes (optional)

5.1. Preface

Application Security Groups, often abbreviated as ASGs, are a feature designed for cloudfoundry operators. Nevertheless it's important for developers to learn what they are and how they work, as it might explain why a particular application may in some cases be unable to communicate with a backing service, and how to remedy the issue.

Below is a summary of this feature, and of the cf CLI commands associated with managing application security groups. Be mindful that, as a non-administrative user, you have permission to invoke only a subset of these commands.

5.2. Exercises

5.2.1. Review Application Security Groups

1. Review the documentation on application security groups.

2. List the security groups in your environment.

```
cf security-groups
```

3. Pivotal Cloud Foundry ships with multiple security groups. Pick two security groups from the list and note the differences in the details of those security groups.

```
cf security-group public_networks
```

5.2.1.1. Questions

- Do application security groups use a whitelist or blacklist approach to firewall rules?
- What are the differences in the two application security groups that you looked at?
- What are some reasons why security groups could be used?

- How could security groups affect the staging and running of your application?

5.3. Administration of application security groups

The cf CLI exposes a dozen or so commands for `admin` users to use to create and manage application security groups. The rest of this lab provides an overview.

Note

Unless you're an administrator of the PCF instance you're working with, you won't be able to experiment with these commands. Nevertheless it's important that you be aware of them.

5.3.1. Managing ASGs

When logged in as `admin`, the command `cf security-groups` lists all security groups:

```
cf security-groups
```

The following commands support managing this list:

```
$ cf create-security-group
$ cf update-security-group
$ cf delete-security-group
```

The **create** and **update** commands require that you supply the security group definition as a JSON file that encodes a list of egress rules.

The creation of a security group is only a first step. Additional commands exist to control which security groups are applied in different situations.

5.3.2. Binding ASGs to Running Applications

The command `cf running-security-groups` lists all security groups that apply to all running applications:

```
cf running-security-groups
```

Use these commands to apply (or unapply) a security group to all running applications:

```
$ cf bind-running-security-group public_networks
$ cf unbind-running-security-group public_networks
```


5.3.3. Binding ASGs to Application Staging

Security groups can also control the behavior of scripts that stage an application. To list all security groups that apply to the staging of applications, use the command:

```
cf staging-security-groups
```

Use these commands to apply (or unapply) a security group to the application staging phase:

```
$ cf bind-staging-security-group public_networks
$ cf unbind-staging-security-group public_networks
```

5.3.4. Org and Space specific bindings

The **running-security-groups** binding is coarse-grained; it applies to all running applications in a PCF installation. For finer-grained control of security groups to specific running applications, these commands can scope the binding to a specific org or space:

```
$ cf bind-security-group public_networks my-org my-space
$ cf unbind-security-group public_networks my-org my-space
```

5.3.5. Questions

- What are the differences between the different types (running and staging) of security groups?

Imagine a situation in which an application is bound to a backing database service and deployed to PCF.

- In the absence of ASG bindings, would the application be able to communicate with the backing service?

Let's assume that the backing database is a MySQL instance, which by default accepts connections on port 3306.

- As a Cloud Foundry administrator, what would you have to do in order to allow **any** running application to communicate with this type of backing service?
- What would your security group rules JSON file look like? Should the rules allow all ports over all ip addresses? Can the rules be constrained to allow only port 3306 over a specific range of ip addresses? Try to construct a security group rules file (JSON) that you'd use to create and then bind to running applications. What command would you use to define the security group? What would you name it? What command would you use to bind it?

Chapter 6. Log Drain

Estimated Time: 25 minutes

6.1. Preface

Cloud Foundry's loggregator subsystem routes applications' log streams out of the containers from which they emanate, and makes them accessible via the `cf logs` command.

But what if you needed to maintain a month's worth of logs, and wanted to leverage third party tools or services for log analysis? Cloudfoundry provides the capability to *drain* application logs to some destination, whether it be an internal enterprise system or a third-party service.

In this lab, you'll setup a log drain for your application using a service called Papertail.

6.2. Exercises

6.2.1. Create a log drain

Pivotal Cloud Foundry does not persist logs. For long term storage of logs, drain logs to a third party log management service.

1. Read about log draining.
   In this section, you will configure application log draining into a log analysis tool with a User Provided Service Instance. We will use the free Papertrail service, but any tool that supports tcp syslog draining can be used.

2. Sign up for a free Papertrail account. We've got some instructions to set up Papertrail.
   In a nutshell:

   a. Create a Papertrail account, fill in your email and password

   b. Browse to papertrailapp.com/systems/new

   c. Select "I use Cloud Foundry"

   d. "What should we call it?" give it a name - `log-drain`

   e. Click "Save"

f. Take note of the url (logs2.papertrail.com:<some-port>)

3. Create a User Provided Service Instance that streams logs to Papertrail.

```
cf create-user-provided-service articulate-log-drain -l syslog://{{syslog_drain_url}}
```

4. Bind the `articulate-log-drain` service to the `articulate` application.

```
cf bind-service articulate articulate-log-drain
```

Note

It can take up to ten minutes for logs to start flowing into Papertrail. At the time of writing, this was consistently 10 minutes before seeing logs in Papertrail.

Question: Is `cf restart/restage` required for log draining to start working? Why? Why not?

5. View `articulate` in a browser. Refresh as necessary to generate logs.

6. To view the logs in Papertrail do the following:

   a. Click on "Dashboard"

   b. Click on "log-drain"

7. Experiment. Restart the `articulate` application.

```
cf restart articulate
```

6.3. Questions

- How does this approach differ from how you get your logs today?
- What kinds of alerts or visualizations could you create from your logs data?

6.4. Beyond the Class

- Visualize your logs

Chapter 7. Blue Green

Estimated Time: 30 minutes

7.1. Preface

So you've pushed an app, and now it's time to deploy a new version. Blue-green deployments are a technique for deploying updates with zero downtime.

Cloudfoundry allows developers to manage everything about their application: from deployment to the management of routes to an application. It's thanks to this self-service philosophy that the task of achieving zero-downtime deployments becomes easy.

This lab will walk you through the steps to deploy a new version of an application with zero downtime, and provides a way to visualize how traffic gets routed to the new application.

7.2. Setup

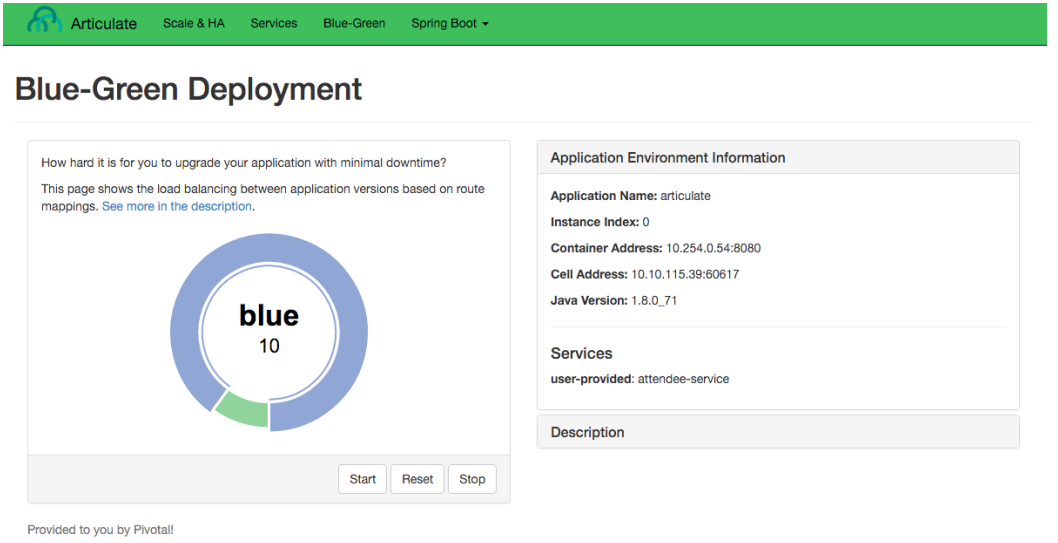
1. To simulate a blue-green deployment, first scale `articulate` to multiple instances.

```
cf scale articulate -i 2
```

7.3. Perform a Blue-Green Deployment

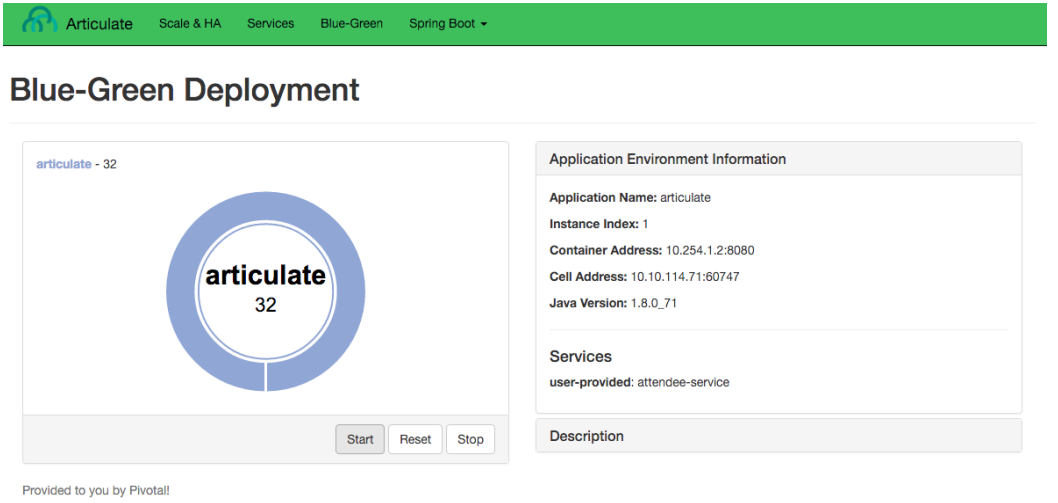
1. Read about using Blue-Green Deployments to reduce downtime and risk.

2. Browse to the `articulate Blue-Green` page.



3. Lets assume that the deployed application is version 1. Let's generate some traffic. Press the `Start` button. *Leave this open as a dedicated tab in your browser. We will come back to this later.*

4. Observe our existing application handling all the web requests.



5. Record the subdomain (`host`) for the `articulate` application.
   This is our production route. *You will use this in the next step.*
   For example:

```
cf routes

Getting routes as droberts@pivotal.io ...

space   host                                    domain                  apps
dev     articulate-heartsickening-elegance      pcfi1.fe.gopivotal.com  articulate
```

6. Now let's `push` the next version of `articulate`.
   However, this time we will specify the subdomain by appending `-temp` to our production route.
   For example:

```
$> cd ../labs/demo-apps/java-apps

$> cf push articulate-v2 -p articulate-1.0.jar -m 768M -n {{articulate_hostname_temp}} --no-start
```

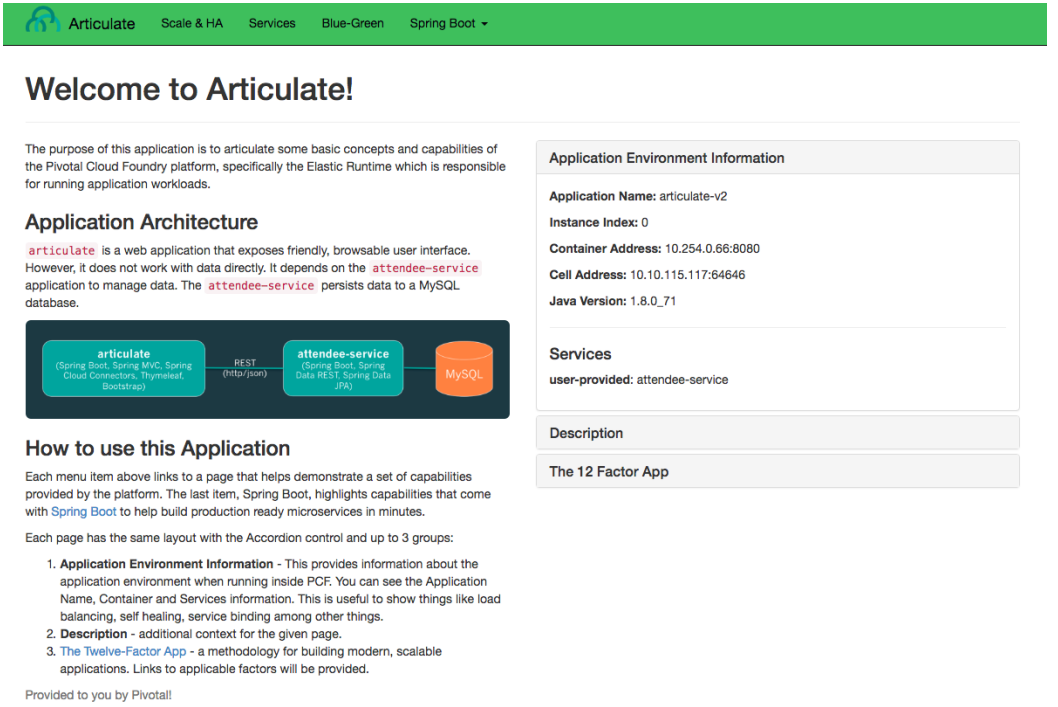
7. Stop the `attendee` app to free up memory in your org.

```
cf stop attendee
```

8. Start the new version of the `articulate-v2` app.

```
cf start articulate-v2
```

9. Now we have two versions of our app deployed.
   Open a new tab and view version 2 of `articulate` in your browser. Take note of the application name.

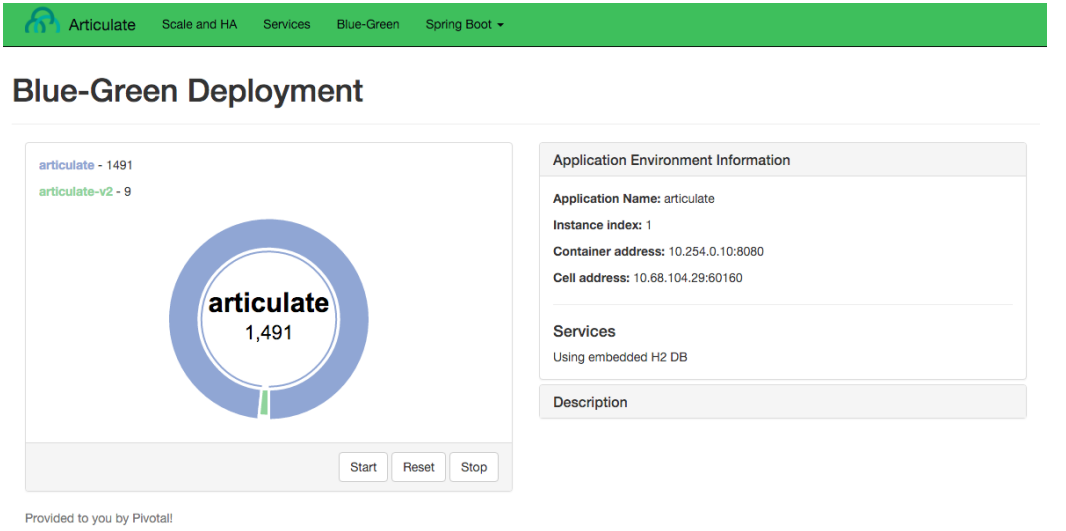


At this point in the deployment process, you could do further testing of the version you are about to release before exposing customers to it.

10. Let's assume we are ready to start directing production traffic to version 2. We need to map our production route to `articulate-v2`.
    For example (your domain and subdomain will be different):

```
cf map-route articulate-v2 {{domain_name}} --hostname {{articulate_hostname}}
```

11. Return to browser tab where you started the load. You should see that it is starting to send requests to version 2.



12.Press the `Reset` button, so we can see how the load get distributed across app instances.

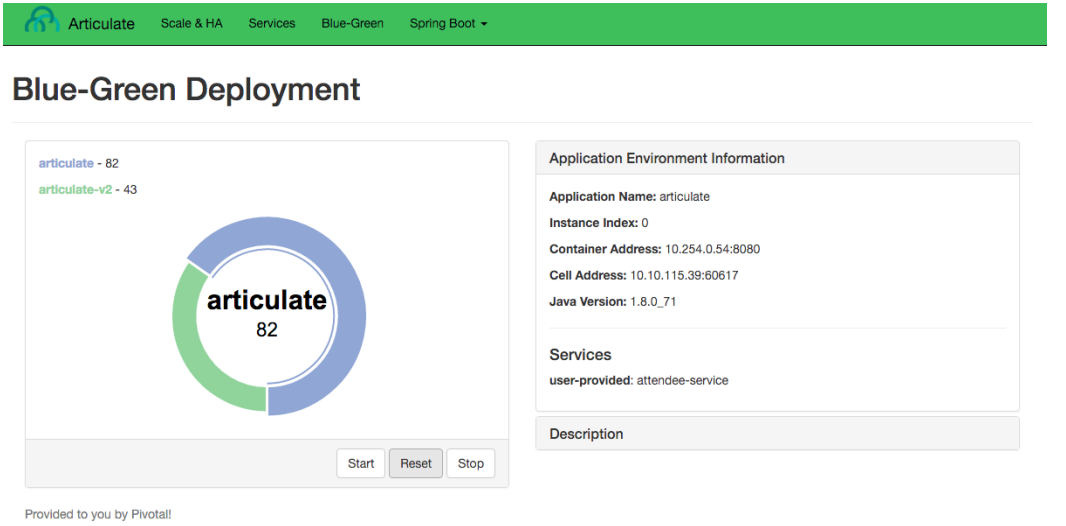
If you are running with a similar configuration to this:

```
cf apps

Getting apps in org dave / space dev as droberts@pivotal.io...
OK

name                requested state   instances   memory   disk   urls
articulate          started           2/2         768M       1G     ...
articulate-v2       started           1/1         768M       1G     ...
```

You should see about a third of the requests going to version 2.



13.Move more traffic to version 2, with:

```
cf scale articulate -i 1
```

..and:

```
cf scale articulate-v2 -i 2
```

If you `Reset` the load generator, you will see 2/3 of the traffic go to `articulate-v2`.

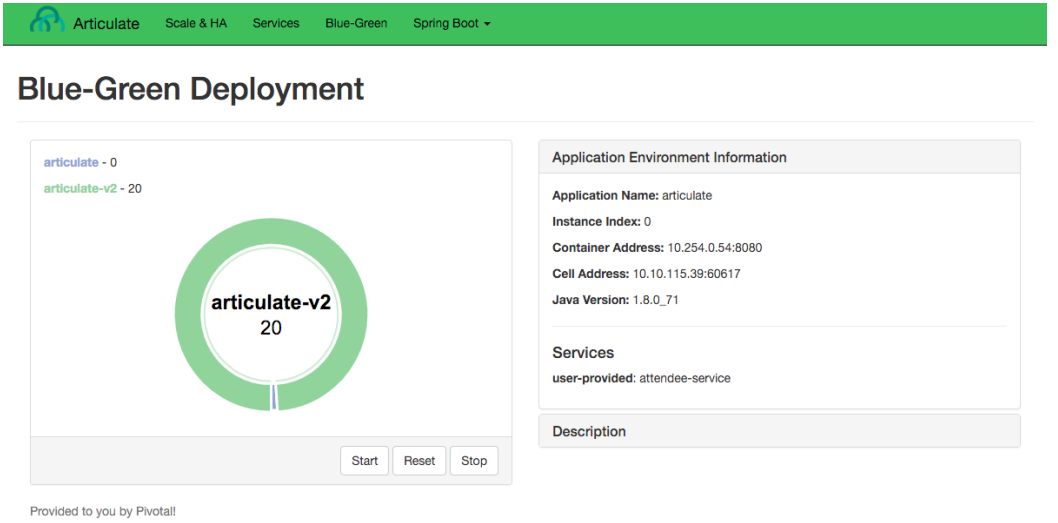
14.Move all traffic to version 2.

Remove the production route from the `articulate` application.

For example (your domain and subdomain will be different):

```
cf unmap-route articulate {{domain_name}} --hostname {{articulate_hostname}}
```

If you `Reset` the load generator, you will see all the traffic goes to `articulate-v2`.



Note

Refreshing the entire page will update the application name.

15.Remove the temp route from the `articulate-v2` application.
For example (your domain and subdomain will be different):

```
cf unmap-route articulate-v2 {{domain_name}} --hostname {{articulate_hostname}}
```

Congratulations! You performed a blue-green deployment.

7.3.1. Questions

- How would a rollback situation be handled using a blue-green deployment?
- What other design implications does running at least two versions at the same time have on your applications?
- Do you do blue-green deployments today? How is this different?

7.4. Cleanup

Let's reset our environment.

1. Delete the `articulate` application.

```
cf delete articulate
```

2. Rename `articulate-v2` to `articulate`.

```
cf rename articulate-v2 articulate
```

3. Restart `articulate`.

```
cf restart articulate
```

4. Scale down.

```
cf scale articulate -i 1
```

7.5. [Optional] Explore Blue-Green Deployment Plugin

Now that we understand the mechanism by which we can perform blue-green deployments, let's explore one of the cf cli plugins that automate some aspects of this deployment process.

7.5.1. Setup

1. Visit https://plugins.cloudfoundry.org/
2. Locate the *blue-green-deploy* plugin and follow instructions to install the plugin
3. Explore the project's github repository README to learn how to use the plugin

7.5.2. Experiment

Let's start again with deploying articulate in a blue-green fashion, but this time using the plugin:

1. Because this plugin will deploy a new application with app and host name `articulate-new`, it will cause route conflict (host is taken) in a shared environment like PWS. In this case, you need to rename your

application to make it unique:

```
cf rename articulate articulate-{{initials}}
```

2. Make sure you have a simple manifest file defined for your articulate application. Here's an example:

   manifest.yml.

```
---
applications:
- name: articulate-{{initials}}
  path: articulate-1.0.jar
  memory: 512M
  buildpack: java_buildpack
  env:
    JAVA_OPTS: -Xss228k
  services:
  - attendee-service
```

3. Instead of using the `push` command, deploy articulate using the blue-green-deploy command:

```
cf blue-green-deploy articulate-{{initials}}
```

Observe what this command does:

1. it deploys articulate using a different application name and host name: `articulate-new`

Once the new version of the app is running..

1. the public route for the application is mapped to the new app
2. the previously deployed application is renamed using the '-old' suffix
3. the '-new' suffix is now dropped from the new application
4. the public route is unmapped from the old version of the application

All this is accomplished via the invocation of a single command!

We can take this a step further: by passing a smoke-test script to the `blue-green-deploy` command, the plugin will run the smoke tests and proceed to upgrade the application only on the condition that the smoke tests passed (returned with an exit code of 0). The plugin passes the fully-qualified domain name of the newly-deployed application as an argument to the smoke-test.

Here's an updated blue-green deployment command that uses a simple health-check test for articulate:

```
cf blue-green-deploy articulate-{{initials}} --smoke-test ./test-health.sh
```


See the included project source code for complete details.

Congratulations! You have completed this lab.

Chapter 8. Application Autoscaler

Estimated Time: 25 minutes

8.1. Preface

So you can deploy your app, you can scale it. But what if we wish to automate scaling an application up and down during periods of higher and subsequently lower traffic. PCF allows us to automate scaling via the marketplace service named the *App Autoscaler*.

In this exercise we'll introduce a load against our application, and watch it automatically scale up to multiple instances, and then automatically scale back down to a specified minimum. Enjoy.

8.2. Setup

1. Ensure the `articulate` application has one application instance running.

```
cf scale articulate -i 1
```

8.3. Provision and Bind an Autoscaler Service Instance

1. Read the documentation about App Autoscaling.

2. Review what's in the marketplace.

```
cf marketplace
```

3. Create a `app-autoscaler` service instance with the free `standard` plan and name it `autoscaler`.

```
cf create-service app-autoscaler standard autoscaler
```

4. Bind the service to `articulate`.

```
cf bind-service articulate autoscaler
```

Note

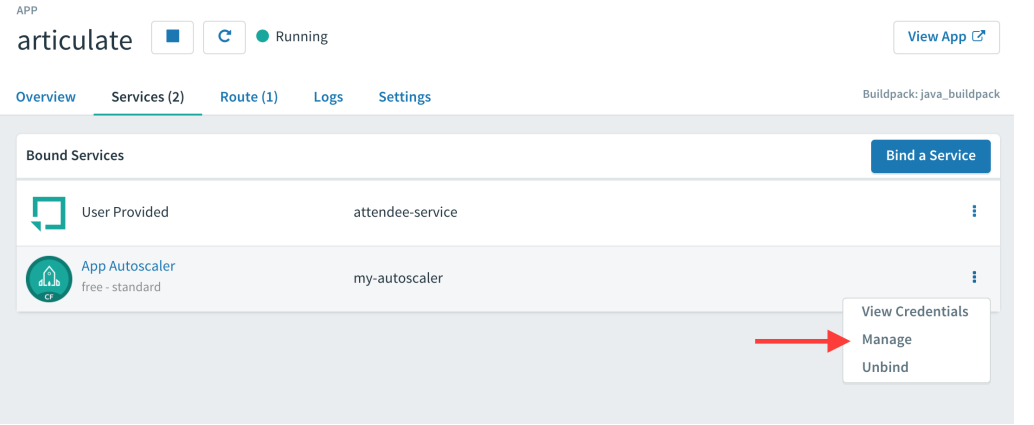
You can ignore the "TIP: Use 'cf restage articulate' to ensure your env variable changes take effect" message at this time.

5. Restart the application.

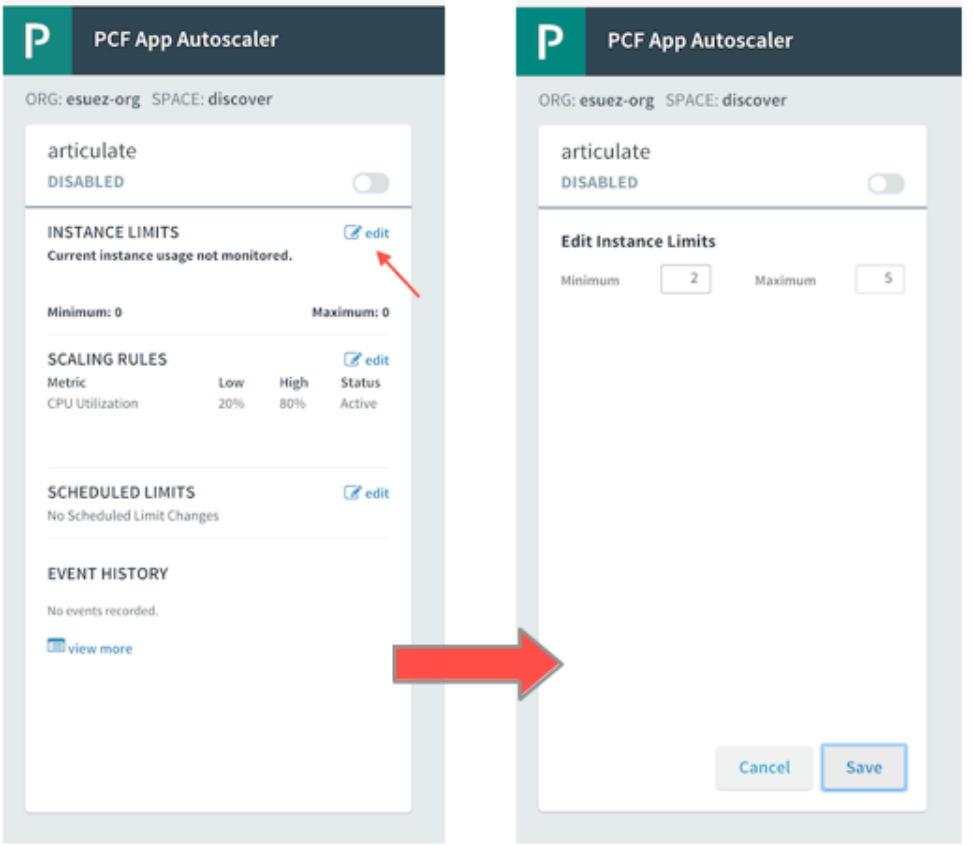
```
cf restart articulate
```

8.4. Configuring Autoscaling Service Instance

1. View `articulate` in Apps Manager.
   a. Navigate to the `Services` tab.
   b. Click the `Manage` link for the autoscaler (this will open a new tab).



2. Edit instance limits
   a. Click 'Edit'
   b. Set the minimum number of instances to 2 and the maximum to 4
   c. Click 'Save'



d. Enable the switch for Autoscaler service.

3. Return to Apps Manager and observe the number of instances increase from `1` to `2`.

8.5. Generate Load and Observe the Results

One popular option for generating a load against a web site is by using the apache load-testing tool JMeter. Instructions for JMeter are outlined below.

Note

Another option is to use a PWS marketplace service named *Load Impact*. One advantage of using Load Impact is that you don't have to download and install any software to your laptop in order to conduct a load test: it's all done via a web interface. Further, the load test itself is run from the cloud: the test specification can specify what geographic location will serve as the origin of the load test. If you feel like trying Load Impact instead of JMeter, head on over to the PWS Marketplace in your Apps Manager, and create an instance of that service, then follow the instructions for creating a test.

8.5.1. Generating a Load with JMeter

1. You should already have `apache-jmeter-XXX.zip` in your `labs/tools` directory, where XXX is the version number. If you prefer, you can also download Apache JMeter. It will be used to generate load. Review the JMeter getting started directions.
   Linux and Mac | Windows

   1. Add execute permissions to the JMeter script: `chmod +x {{jmeter_basedir}}/bin/jmeter.sh`

   2. Start JMeter by executing `{{jmeter_basedir}}/bin/jmeter.sh`

   1. Start JMeter by double clicking the following file in Windows Explorer: `{{jmeter_basedir}}/bin/jmeter.bat`

2. Download the load-gen.jmx file.

3. Open the load-gen.jmx file with the JMeter GUI.

4. Expand the content of the JMeter test plan by:

   a. Selecting the `Load Generator` test plan on the left pane.

   b. Going to the menu: `Options` # `Expand All`.

5. Under `HTTP Request`, edit the test plan as follows:

   a. Set the value of the `Service Name or IP` field to your application's fully-qualified hostname (do not include the `http://`). For example: `jsmith-articulate.cfapps.io`

   b. Set the value of the HTTP Request `Path` field to `/services`

6. Save the test plan: `File` # `Save`.

7. Run the test plan: `Run # Start`.

8. Use Apps Manager and the `cf` CLI to observe your service scale up and back down based on load.

9. Stop the test plan: `Run # Stop`.

10. Review Autoscale history:

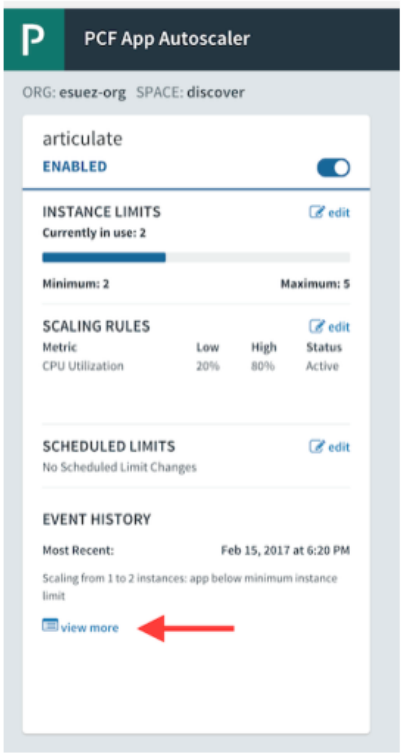


Figure 8.1. History

Note

An even better option for monitoring the state of your application, including your application instances and their CPU usage, is to install the cf cli plugin **cf top**. Head on over to the cloud

foundry plugins catalog, locate the plugin named 'top' and install it. It’s definitely worth your time.

8.6. Clean up

1. Unbind the `autoscaler` service instance.

```
cf unbind-service articulate autoscaler
```

2. Delete the `autoscaler` service instance.

```
cf delete-service autoscaler
```

3. Scale `articulate` back to original settings.

```
cf scale articulate -i 1
```

4. Restart `articulate`.

```
cf restart articulate
```

8.7. Questions

- How do you handle autoscaling today?
- What 12 factor principles are important when it comes to scaling?
- How do you handle scaling at the data layer?

Chapter 9. Application Performance Monitoring

Estimated Time: 25 minutes

9.1. Preface

In the previous labs, we learned how to setup log analysis for our application, and how to autoscale our app. We also must be able to monitor it. Configuring your application for monitoring becomes effortless with integration from monitoring tools such as newrelic and appdynamics. In this lab, we'll use the newrelic managed service to setup application performance monitoring for our app.

9.2. Exercises

9.2.1. Setup

1. APM solutions cause applications to consume more memory. The amount of memory we allocated to `articulate` and `attendee` happens to be sufficient. If not, increase the memory limit for each application using `cf scale`.

9.2.2. Provision a New Relic Service Instance

Read the documentation on New Relic integration with the Java Buildpack.

9.2.2.1. Pivotal Cloud Foundry Instructions

Pivotal Web Services | Pivotal Cloud Foundry

Create a New Relic Service instance.

```
cf create-service newrelic standard newrelic
```

1. Signup for a 60 day free trial license with New Relic.

2. Obtain `license key`.
   This can be found under `Account Settings` from the drop down menu in the upper right corner of the page.

3. Create a user provided service instance. This will create an interactive prompt.

```
cf create-user-provided-service newrelic -p license_key
```

9.2.3. Bind the New Relic Service Instance

1. Bind the `newrelic` service instance to `articulate`.

```
cf bind-service articulate newrelic
```

2. Restage `articulate`.

```
cf restage articulate
```

3. Bind the `newrelic` service instance to `attendee`.

```
cf bind-service attendee newrelic
```

4. Restage `attendee`.

```
cf restage attendee
```

9.2.3.1. Questions

- Why must `articulate` & `attendee` be restaged as oppose to restarted?

9.2.4. Create Some Web Traffic and View in New Relic

1. Refresh the `articulate Service` web page multiple times.
   Optionally use the JMeter script from the Application Autoscaler lab.

2. Observe and explore the performance metrics in the New Relic console under the `APM` tab.

From the New Relic APM dashboard, click on the application you wish to monitor:

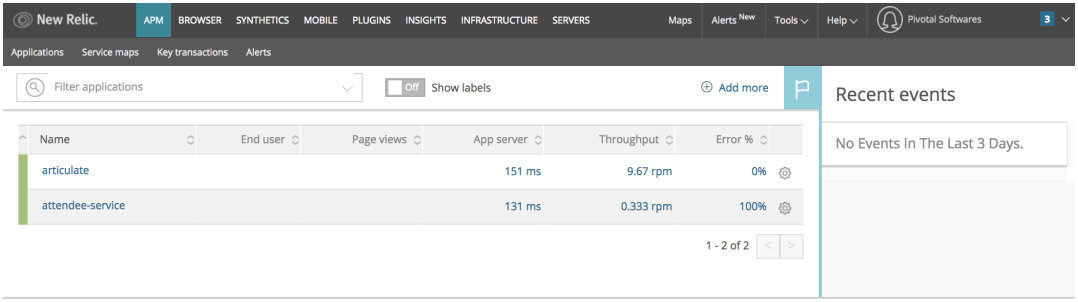


Figure 9.1. New Relic APM Dashboard

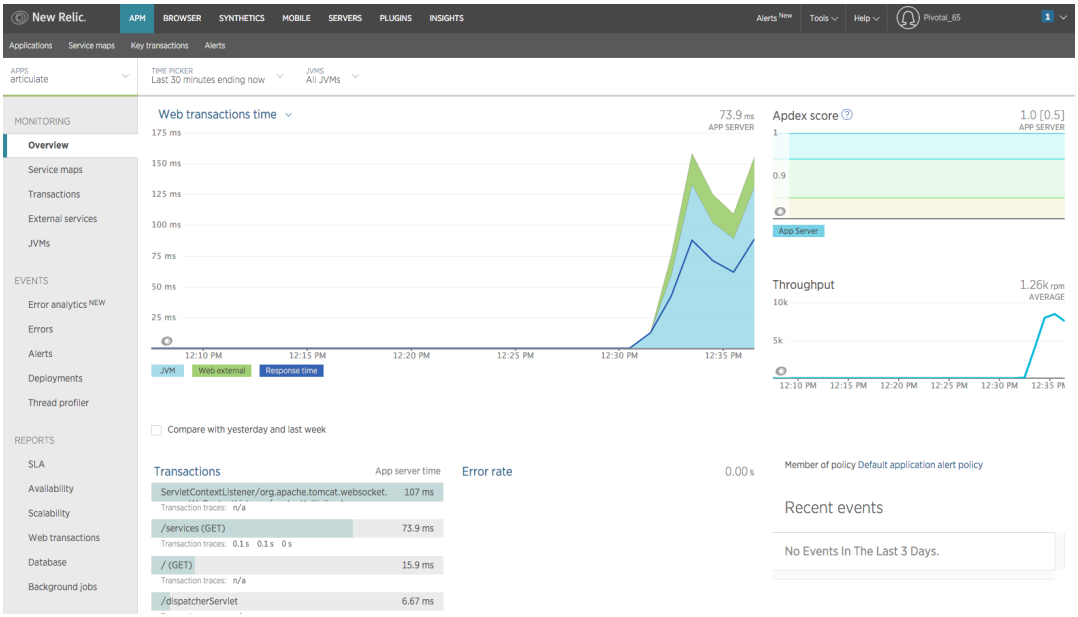


Figure 9.2. `articulate`

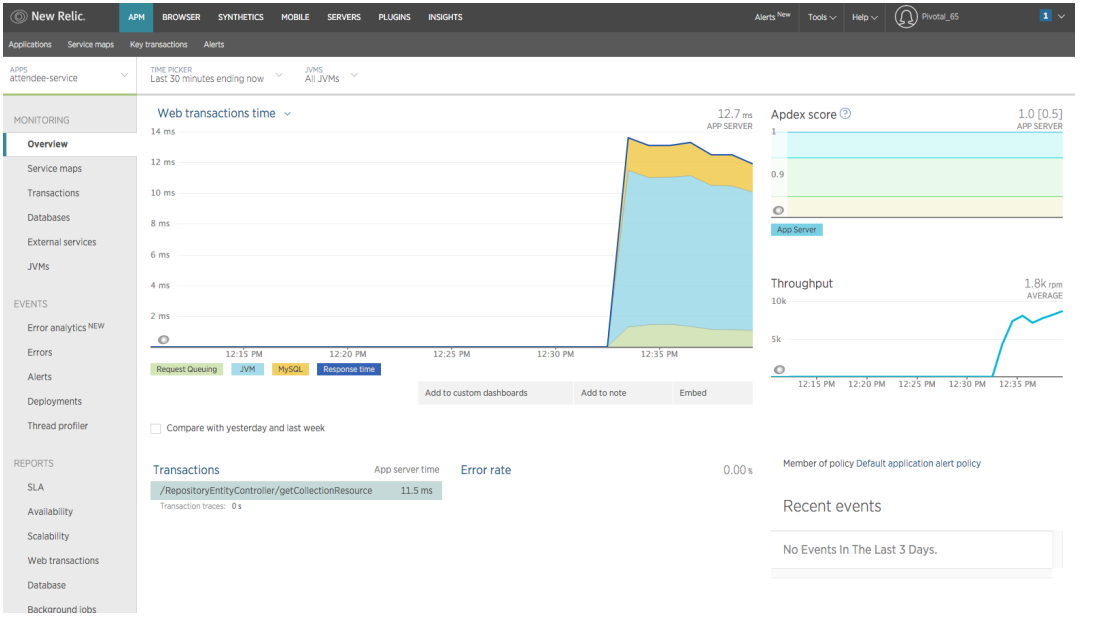


Figure 9.3. `attendee`

9.2.5. Clean up

1. Unbind `newrelic` service instance.

```
cf unbind-service articulate newrelic
```

..and:

```
cf unbind-service attendee newrelic
```

2. Delete `newrelic` service instance.

```
cf delete-service newrelic
```

3. Restage applications.

```
cf restage articulate
```

..and:

```
cf restage attendee
```


4. Scale back down to our original settings.

```
cf scale articulate -m 768M -i 1
```

..and:

```
cf scale attendee -m 768M -i 1
```

9.2.6. Questions

- How do you manage APM tools today? How is this different?

9.3. Beyond the Class

Pivotal Cloud Foundry offers a marketplace experience with both New Relic and App Dynamics. Try it with your Pivotal Cloud Foundry installation.

Chapter 10. Metrics

Estimated Time: 25 minutes

10.1. Preface

PCF Metrics provides application metrics for your application directly inside cloudfoundry. It's an innovative tool that combines application events, metrics (cpu, memory, network), and logs into a powerful view that can help you monitor and gain insight into your application's behavior over time. In this lab you'll take a little time to explore PCF Metrics and its user interface.

10.2. Exercises

10.2.1. PCF Metrics Overview

Pivotal Web Services | Pivotal Cloud Foundry

1. Review the documentation on PCF Metrics.

1. Review the documentation on PCF Metrics.

10.2.2. Explore PCF Metrics

1. Log into PCF Metrics and explore the views for `articulate`.
2. Generate some traffic to `articulate` and observe the effects.

Optionally use the JMeter script from the Application Autoscaler lab to generate some traffic.

10.2.3. Questions

- What might a spike in memory indicate?

Chapter 11. Buildpack

Estimated Time: 25 minutes

11.1. Preface

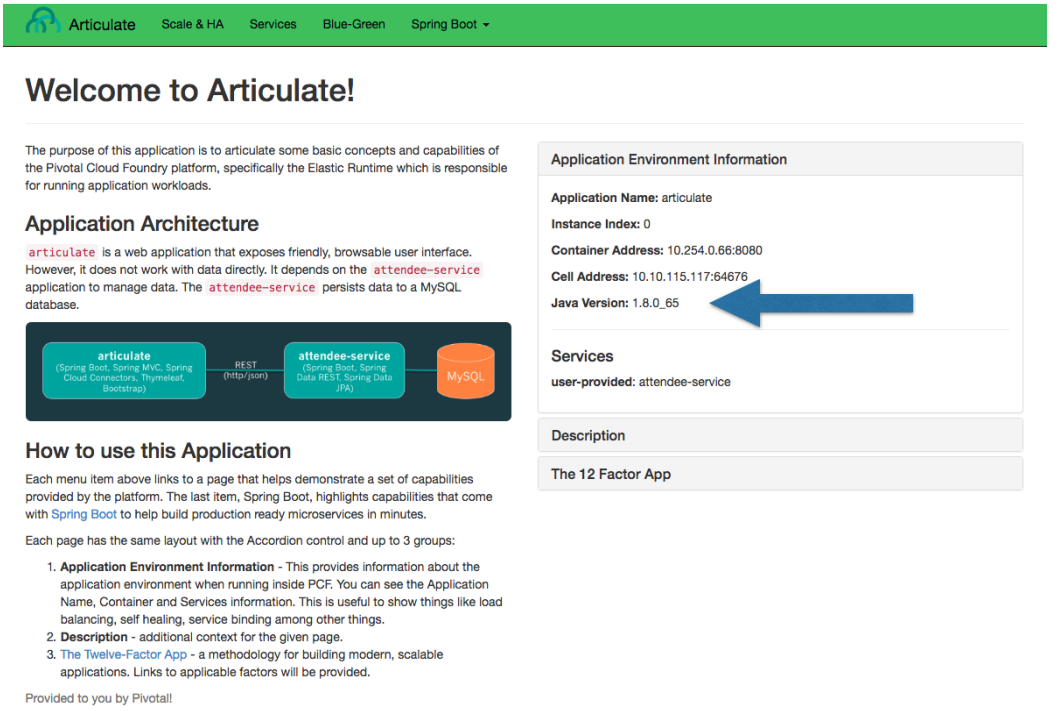
We've learned that buildpacks produce the droplets that are needed to run our applications. But how do we customize their behavior? In this lab, you'll explore ways to configure the version of the java runtime that the java buildpack provisions for your application.

11.2. Exercises

11.2.1. Use a Custom Buildpack

1. Review the documentation on deploying with custom buildpacks and how dependencies are handled with the Java Buildpack online package and the offline package.

2. Review the Java Version reported by `articulate`.



3. Review which buildpack is in use.

```
cf app articulate
```

4. Push `articulate` again, but this time specify a custom buildpack. In this case, we will use the latest version of the Java Buildpack on GitHub.

```
$> cd .../labs/demo-apps/java-apps

$> cf push articulate -p articulate-1.0.jar -b https://github.com/cloudfoundry/java-buildpack.git
```

5. Using your browser, refresh the `articulate` application.

It's likely (but not required) that the `Java Version` changed.

What Just Happened?

We instructed our application to use a custom buildpack (as opposed to a system provided one).

In this case, we used the Java Buildpack source on Github as our custom buildpack. The Java Buildpack source is continuously updated and it is an online package of the buildpack. Meaning it has access to all dependencies via the network (it has access to all JRE versions, etc). Whereas, the system provided Java buidpack is offline, with a limited set of dependencies. For both the online and offline packages, unless the Java version is specified the application is run with the latest version of Java available to the buildpack.

11.2.2. Change the Java version

1. Review the Java Buildpack configuration and extension documentation.
2. Let's assume that we want to run `articulate` on a specific version Java.

```
cf set-env articulate JBP_CONFIG_OPEN_JDK_JRE "{jre: { version: 1.8.0_45 }}"
```

3. Using your browser, refresh the `articulate` application.
   QUESTION: Is the `articulate` running with `1.8.0_45`? Why not?
4. Restage `articulate`.

```
cf restage articulate
```

   QUESTION: Would `cf restart` be sufficient instead of `cf restage`? Why not?

5. Using your browser, refresh the `articulate` application.

11.2.3. Questions

- What other items are easily customized with the Java Buildpack?
- How would you go about customizing the java buildpack to use a different version of the new relic agent jar?
- If you use Java, what items do you think would need customization in your environment?

Chapter 12. Service Broker

Estimated Time: 60 minutes

12.1. Preface

Cloudfoundry is an extensible platform. Managed services can easily be plugged in and made available in the services marketplace. In this lab, we'll use a feature known as *space-scoped service brokers* to register a custom managed service that provisions Mongodb databases for applications.

You'll deploy an application that implements the service broker API, register it as a service broker in Cloudfoundry, and then put it to use by instantiating a mongodb instance (with `create-service`), and attaching that resource (with `bind-service`) to an application that will use it to persist album information.

12.2. Exercises

12.2.1. Setup

1. The code for our service broker is in your lab files in `~/labs/source-code/cloudfoundry-mongodb-service-broker` directory.
2. OPTIONAL STEP - Import applications into your IDE such as Spring Tool Suite (STS). You can either import the application as a maven project or gradle project.

STS Import Help for Gradle

1. Make sure the `Gradle Support` IDE extension is installed.
   a. To install `Gradle Support` select menu:Help[Dashboard]. This will open a new tab to the Dashboard.
   b. Click on "IDE Extensions" then search for "Gradle". If you can't find it or can't select it, it's already installed.
   c. Select `Gradle Support` or `Gradle (STS Legacy) Support` and click the "Install" button.
   d. Complete the installation wizard and restart STS.

2. Import the broker code

   a. Select menu:File[Import…]

   b. Then select Gradle -> Gradle Project.

   c. On the Import Gradle Project page, browse to the `cloudfoundry-mongodb-service-broker` directory.

   d. Then push the "Build Model" button.

   e. Select the project. Click "Finish".

12.2.2. Service Broker API Overview

Note

Review the documentation. Specifically, the sequence diagram. This is what we will implement.

12.2.3. Create a MongoDB Service Broker

12.2.3.1. About this Broker

This broker is implemented with Spring Boot, and leverages the Cloud Foundry Service Broker project. This project reduces the effort required to write a service broker in Java to the implementation of a handful of Spring beans. This blog entry by Scott Frederick originally introduced this project to the Spring Cloud in the Summer of 2016.

12.2.3.2. Implement Catalog Management

1. Review the documentation on implementing catalog management.

2. We need to implement catalog management in our `mongodb-service-broker` application. Fortunately, all the Service Broker API endpoints have been mapped by the Spring Cloud Service Broker project. For instance, the https://github.com/spring-cloud/spring-cloud-cloudfoundry-service-broker/blob/master/src/main/java/org/springframework

3. The Spring Cloud Service Broker can't provide all the implementation. We need to describe our catalog. To to that, all we need to do is provide a `Catalog` bean.

Review the following file:`~/labs/source-code/cloudfoundry-mongodb-service-broker/src/main/java/org/springframework/cloud/serv`

```
@Configuration
public class CatalogConfig {

  @Bean
  public Catalog catalog() {
    return new Catalog(Collections.singletonList(
        new ServiceDefinition(
            getEnvOrDefault("SERVICE_ID","mongodb-service-broker"), //env variable
            getEnvOrDefault("SERVICE_NAME","MongoDB"), //env variable
            "A simple MongoDB service broker implementation",
            true,
            false,
            Collections.singletonList(
                new Plan(getEnvOrDefault("PLAN_ID","mongo-plan"), //env variable
                    "standard",
                    "This is a default mongo plan.  All services are created equally.",
                    getPlanMetadata(),
                    true)),
            Arrays.asList("mongodb", "document"),
            getServiceDefinitionMetadata(),
            null,
            null)));
  }
...
```

4. Push the `mongodb-service-broker` application.

```
cd ~/labs/source-code/cloudfoundry-mongodb-service-broker/
```

and

```
cf push mongodb-service-broker -p build/libs/cloudfoundry-mongodb-service-broker.jar -m 1G
    --random-route --no-start
```

5. Set environment variables.
These environment variables get used by the broker to generate the catalog. These values should be unique across the entire Pivotal Cloud Foundry instance to meet the broker API specifications.
As a convention, append your initials to where specified.

```
$ cf set-env mongodb-service-broker SERVICE_ID mongodb-service-broker-{{add your initials}}
$ cf set-env mongodb-service-broker SERVICE_NAME MongoDB-{{add your initials}}
$ cf set-env mongodb-service-broker PLAN_ID mongo-plan-{{add your initials}}
```

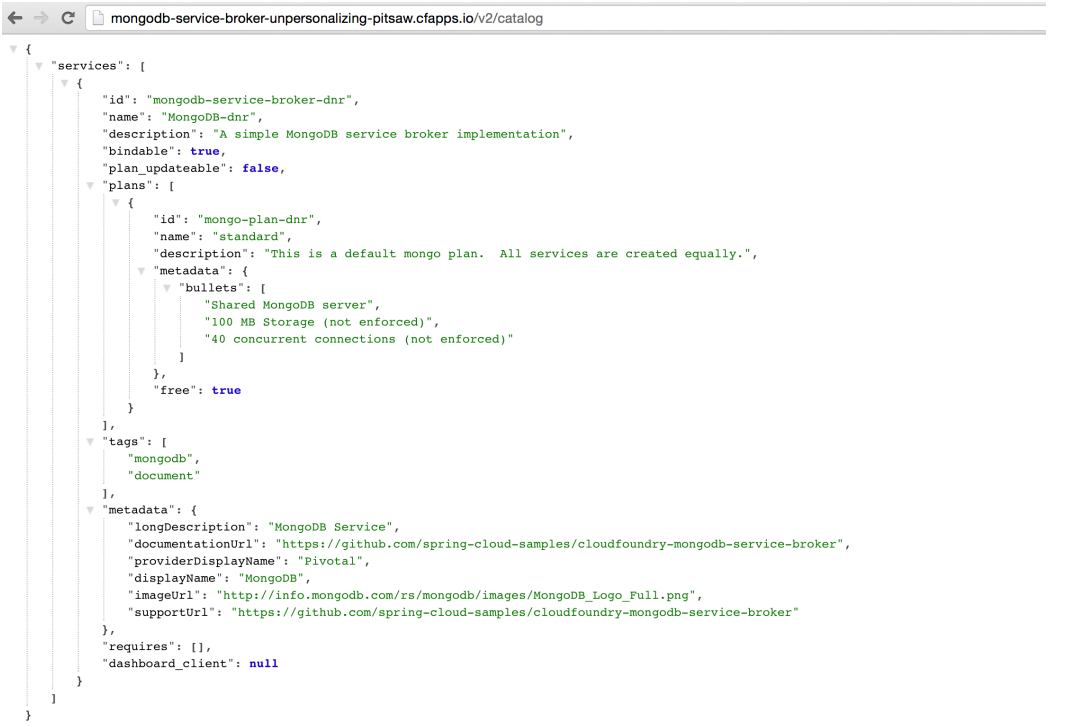
Note

You can safely ignore the "TIP: Use 'cf restage' to ensure your env variable changes take effect" message.

6. Start `mongodb-service-broker`

```
cf start mongodb-service-broker
```


7. Verify your work. Using an http client (e.g. a browser), visit the application's `v2/catalog` endpoint. Because the application is secured with Basic Auth you will need to provide credentials.
   Username: pivotal
   Password: keepitsimple
   You should see response similar to the following (pic is using the JSON Formatter for Chrome):



8. Register your Service Broker.
   We will be creating a Space-Scoped broker. Space-Scoped brokers help you during the development/testing of your service broker, because they are private to a space and don't require an `admin` to enable access (list it in the marketplace, provision service instances, etc).
   A unique broker name is required. Use your initials.

```
cf create-service-broker mongodb-service-broker-{{your initials}} pivotal keepitsimple
    {{service_broker_app_url}} --space-scoped
```

9. View the Service Brokers in your installation. You should see your new Service Broker.

```
cf service-brokers
```


10.Verify that your service is listed in the marketplace.

```
cf marketplace
```

Congratulations, you have implemented and tested the catalog endpoint in your service broker!

12.2.3.2.1. Questions

- Can a service broker support upgrade/downgrade of a service?

12.2.3.3. Implement Provisioning and Deprovisioning

1. Review the documentation on implementing provisioning and deprovisioning.
2. We need to implement provisioning/deprovisioning in our `mongodb-service-broker` application. To do so we just need to implement the ServiceInstanceService interface, because the Spring Cloud Service Broker project has already done the mapping.

Review the following file:

`~/labs/source-code/cloudfoundry-mongodb-service-broker/src/main/java/org/springframework/cloud/servicebro`

Provisioning Code:

```
@Service
public class MongoServiceInstanceService implements ServiceInstanceService {
...

  @Override
public CreateServiceInstanceResponse createServiceInstance(CreateServiceInstanceRequest request) {
    // make sure we haven't provisioned this before (check broker database)
    ServiceInstance instance = repository.findOne(request.getServiceInstanceId());
    if (instance != null) {
throw new ServiceInstanceExistsException(request.getServiceInstanceId(),
    request.getServiceDefinitionId());
    }

    instance = new ServiceInstance(request);

    if (mongo.databaseExists(instance.getServiceInstanceId())) {
      // ensure the instance is empty
      mongo.deleteDatabase(instance.getServiceInstanceId());
    }

    DB db = mongo.createDatabase(instance.getServiceInstanceId());
    if (db == null) {
throw new ServiceBrokerException("Failed to create new DB instance: " +
    instance.getServiceInstanceId());
     }
    //save to broker database for record keeping
    repository.save(instance);

    return new CreateServiceInstanceResponse();
  }
```



```
...
```

What's happening?

The `createServiceInstance` method is where our broker provisions the database. But to do so two things must happen:

1. Record details in the broker database that we are provisioning a service instance (a MongoDB database)
2. Create the database

Deprovisioning Code:

```
@Service
public class MongoServiceInstanceService implements ServiceInstanceService {
...
  @Override
public DeleteServiceInstanceResponse deleteServiceInstance(DeleteServiceInstanceRequest request)
    throws MongoServiceException {
    String instanceId = request.getServiceInstanceId();
    //locate record in broker database
    ServiceInstance instance = repository.findOne(instanceId);
    if (instance == null) {
      throw new ServiceInstanceDoesNotExistException(instanceId);
    }

    // delete mongo database
    mongo.deleteDatabase(instanceId);
    // delete record from broker database
    repository.delete(instanceId);
    return new DeleteServiceInstanceResponse();
  }
}
```

What's happening?

The `deleteServiceInstance` method is where our broker deprovisions the database. But to do so two things must happen:

1. Delete the database
2. Delete the record of the service instance in the broker database

12.2.3.3.1. Questions

- Does provisioning have to be done synchronously?

12.2.3.4. Implement Binding and Unbinding

1. Review the documentation on implementing binding and unbinding.

2. How to implement binding/unbinding in our `mongodb-service-broker` application? Have a look at the following code that implements the ServiceInstanceBindingService interface, because the Spring Cloud Service Broker project has already done the mapping.

Review the following file:

```
~/labs/source-code/cloudfoundry-mongodb-service-broker/src/main/java/org/springframework/cloud/servicebro
```

Binding Code:

```
@Service
public class MongoServiceInstanceBindingService implements ServiceInstanceBindingService {
...
  @Override
public CreateServiceInstanceBindingResponse
    createServiceInstanceBinding(CreateServiceInstanceBindingRequest request) {

    String bindingId = request.getBindingId();
    String serviceInstanceId = request.getServiceInstanceId();

    ServiceInstanceBinding binding = bindingRepository.findOne(bindingId);
    if (binding != null) {
      throw new ServiceInstanceBindingExistsException(serviceInstanceId, bindingId);
    }

    String database = serviceInstanceId;
    String username = bindingId;
    String password = "password";


    mongo.createUser(database, username, password);

    Map<String, Object> credentials =
Collections.singletonMap("uri", (Object) mongo.getConnectionString(database, username, password));

binding = new ServiceInstanceBinding(bindingId, serviceInstanceId, credentials, null,
    request.getBoundAppGuid());
    bindingRepository.save(binding);

    return new CreateServiceInstanceAppBindingResponse().withCredentials(credentials);
  }
...
```

What's happening?

The `createServiceInstanceBinding` method is where our broker binds an application to the provisioned service instance (database). But to do so two things must happen:

1. Create a unique set of credentials for this binding request in MongoDB

2. Create a record of the binding in the broker database

Unbinding Code:

```
@Service
public class MongoServiceInstanceBindingService implements ServiceInstanceBindingService {

  @Override
  public void deleteServiceInstanceBinding(DeleteServiceInstanceBindingRequest request) {
    String bindingId = request.getBindingId();
    ServiceInstanceBinding binding = getServiceInstanceBinding(bindingId);

    if (binding == null) {
      throw new ServiceInstanceBindingDoesNotExistException(bindingId);
    }

    mongo.deleteUser(binding.getServiceInstanceId(), bindingId);
    bindingRepository.delete(bindingId);
  }
}
```

What's happening?

The `deleteServiceInstanceBinding` method is where our broker unbinds an application to the provisioned service instance (database). But to do so two things must happen:

1. Delete the credentials (user) for this binding request in MongoDB
2. Delete the record of the binding in the broker database

Congratulations! You have created a simple service broker.

12.2.3.4.1. Questions

- Do all services have to be bindable?

12.2.4. Use the MongoDB Service Broker

1. Configure the `mongodb-service-broker` application to use a MongoDB instance.

A MongoDB instance can be obtained in the following ways:

a. Your instructor will provision MongoDB and provide connectivity details to you OR

b. Use a local MongoDB instance OR

c. Provision a MongoDB instance on AWS (see the Appendix)

2. You'll need to communicate the mongo db hostname (or ip address) and password that the service broker will use via environment variables:

```
cf set-env mongodb-service-broker MONGODB_HOST {{mongodb_ip}}
```

and

```
cf set-env mongodb-service-broker MONGODB_PASSWORD {{mongodb_password}}
```

3. Restart the application.

```
cf restart mongodb-service-broker
```

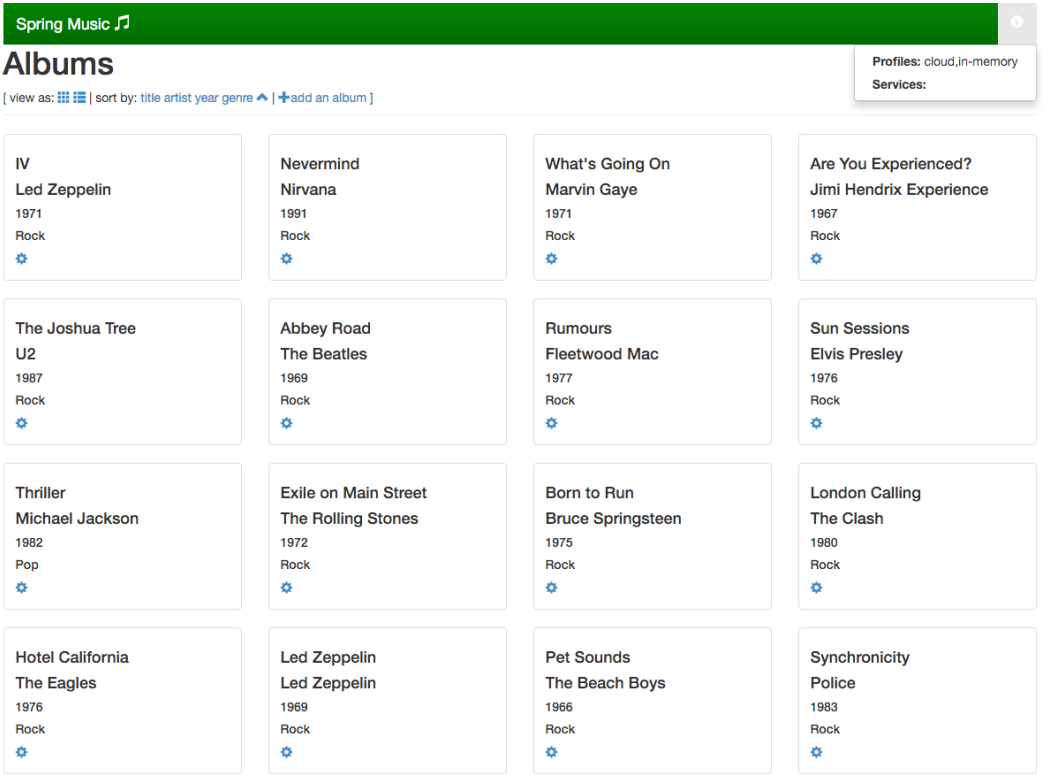
4. You used `spring-music.jar` in the first lab. We are going to use it again.
Source is not required, but you may be curious how it works as you move through the rest of this lab.

5. Push `spring-music`

```
$> cd .../labs/demo-apps/java-apps/

$> cf push spring-music -p ./spring-music.jar -m 768M --random-route
```

6. View `spring-music` in a browser. Click on the button on the top right of the screen. Notice that there are no services attached and `spring-music` is using an embedded (in-memory) database.



7. Create a MongoDB service instance.
   For Example:

```
cf create-service MongoDB-{{initials}} standard mongo-service
```

8. Bind the `spring-music` to `mongo-service`.

```
cf bind-service spring-music mongo-service
```

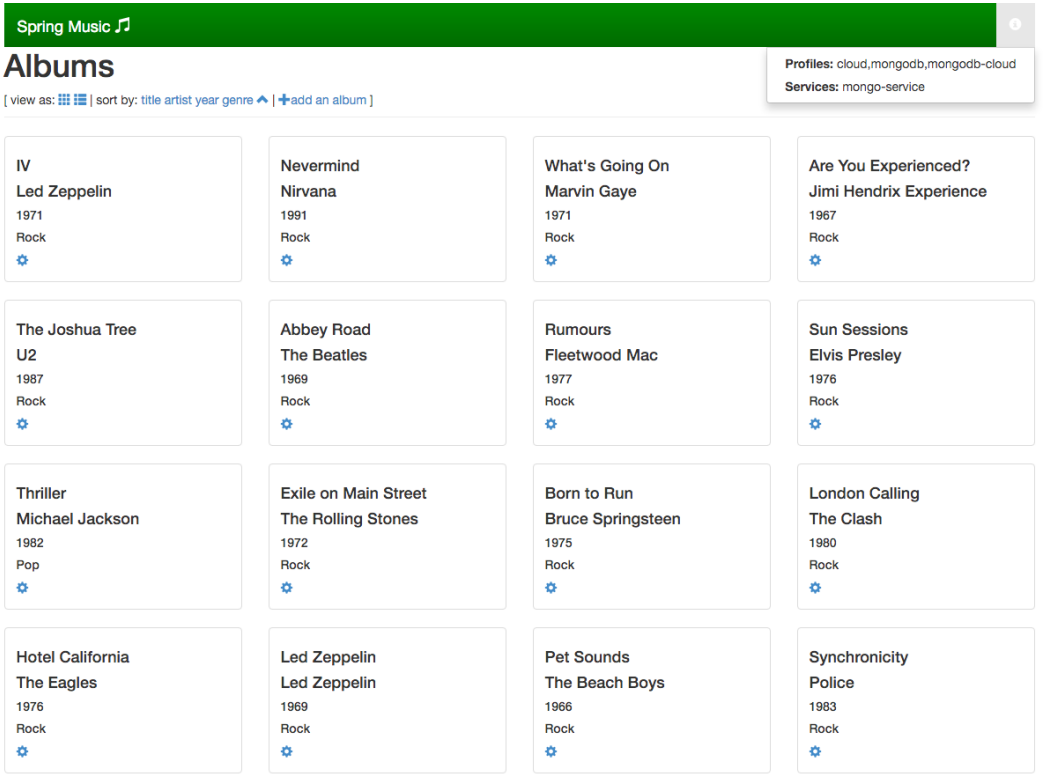
Note

You can safely ignore the "TIP: Use 'cf restage spring-music' to ensure your env variable changes take effect" message.

9. Restart `spring-music`

```
cf restart spring-music
```

10. Refresh `spring-music` in the browser. Click on the `i` button in the top right of the screen. You are now using MongoDB!



11. **Optional Step:** If you have access. View the data in MongoDB.

12.2.5. Clean up

1. Delete `spring-music`.

```
cf delete spring-music
```


2. Delete the `mongo-service` service instance.

```
cf delete-service mongo-service
```

3. Delete the service broker.
   For example:

```
cf delete-service-broker mongodb-service-broker-{{initials}}
```

4. Delete `mongodb-service-broker` application.

```
cf delete mongodb-service-broker
```

5. If provisioned by you, terminate your AWS MongoDB instance by going to your AWS EC2 dashboard, selecting the MongoDB instance, and clicking Actions -> Instance State -> Terminate. The VM should stop, change to "Terminated" status and will eventually be removed from your list of VMs (removal may not happen immediately).

12.3. Beyond the class

Review other sample brokers.

Chapter 13. Continuous Delivery

Estimated Time: 45 minutes

13.1. Preface

There's something special about setting up the automation to produce a new build artifact for an application as soon as a commit is made to version control, and then automatically deploying that application to cloudfoundry. In this lab, you'll do precisely that. You're going to use the Jenkins CI tool for the build job, and CloudFoundry as your target deployment platform. Enjoy.

13.2. Exercises

13.2.1. What is Continuous Delivery?

For a brief explanation, click here.

13.2.2. Setup Github Repository

1. Locate the https://github.com/pivotal-education/pcf-articulate-code on GitHub
2. Use the `Fork` button to make your own private copy of the project in your own GitHub account.

13.2.3. Install Jenkins

Install Jenkins on your local machine or use AWS Jenkins AMI.

13.2.3.1. Local Installation

1. Prior to installing Jenkins the following should have been installed. If not, you will need to do so now.
   - JDK 1.8
   - git
2. A copy of `jenkins.war` should have been provided for you - skip to next step. If not, use this link to

Download `Jenkins` version 1.642.4 war file.

3. Copy the file to folder: `labs/tools/jenkins/`

4. Change the working directory to where you copied the `jenkins.war` file.

```
Unix, Mac: cd ~/labs/tools/jenkins/
Windows:   cd C:\labs\tools\jenkins\
```

5. Run Jenkins. If you have an existing Jenkins installation and want to start afresh, consider moving or deleting "~/.jenkins" folder.

```
java -jar jenkins.war
```

6. Open a browser to http://localhost:8080. Enter the auto-generated admin password when prompted. Close the Getting Started dialog.

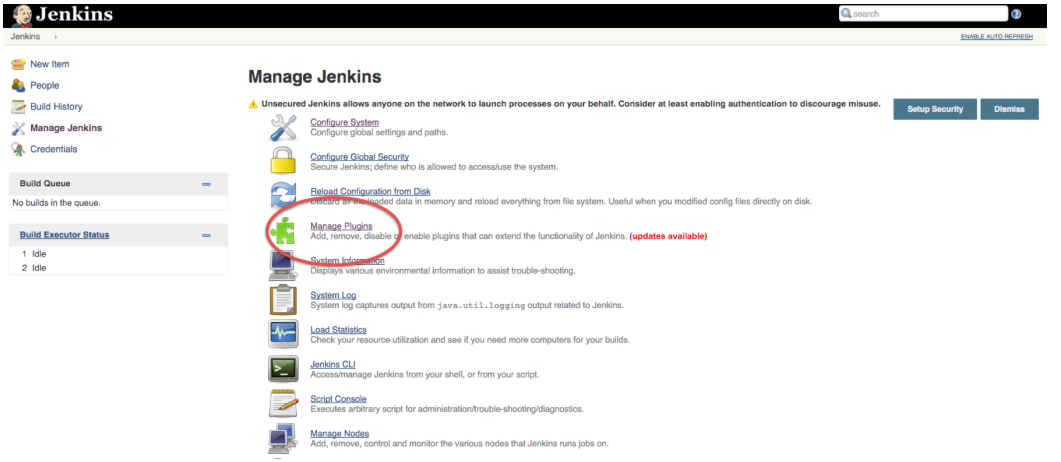
13.2.3.2. AWS Jenkins AMI

Provision a Jenkins instance on AWS.

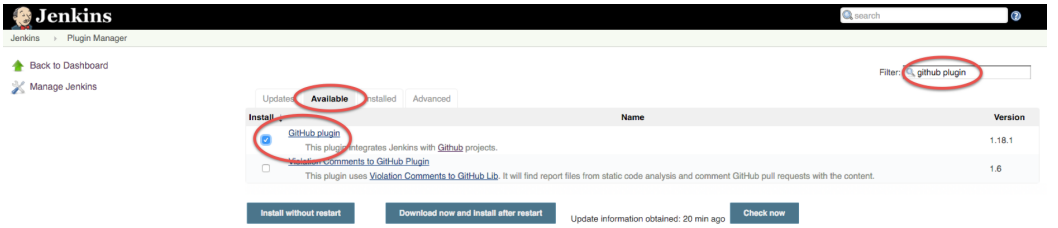
13.2.4. Configure Jenkins

You may skip this section if using the AWS Jenkins AMI.

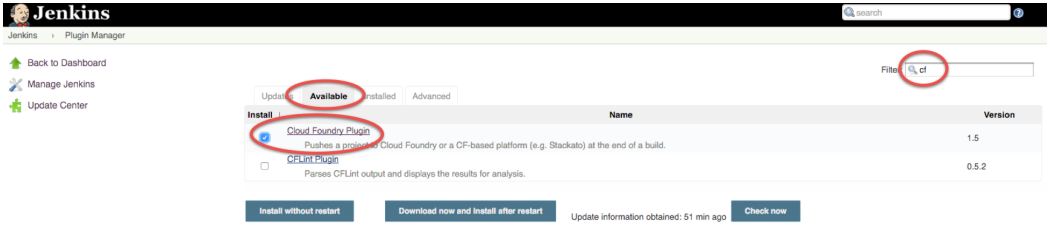
1. Click on the `Manage Jenkins` link, then click the `Manage Plugins` link.



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`.

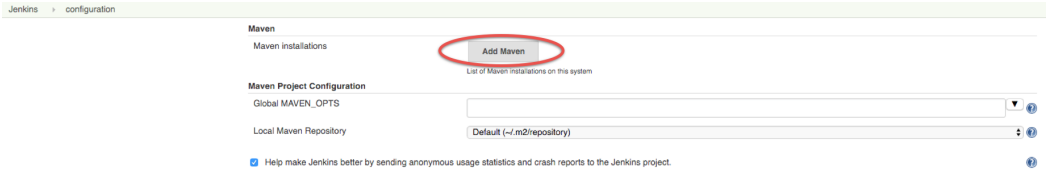


3. Also install the Cloud Foundry Plugin. Select it and click `Install without restart`.

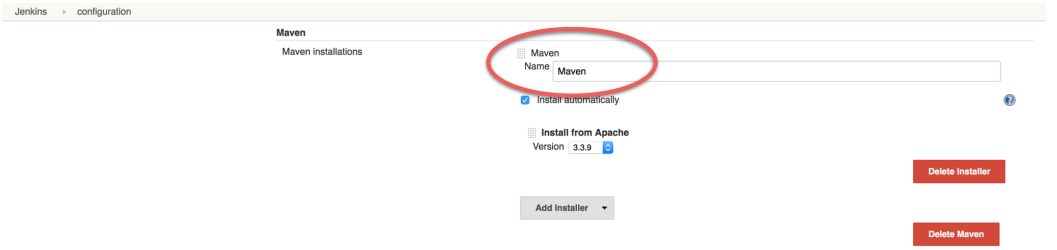


4. Finally install the Maven Integration Plugin. Select it and click `Install without restart`.

5. From the `Manage Jenkins` page, click on `Global Tool Configuration`. Scroll down to the `Maven` section, and click the `Add Maven` button.



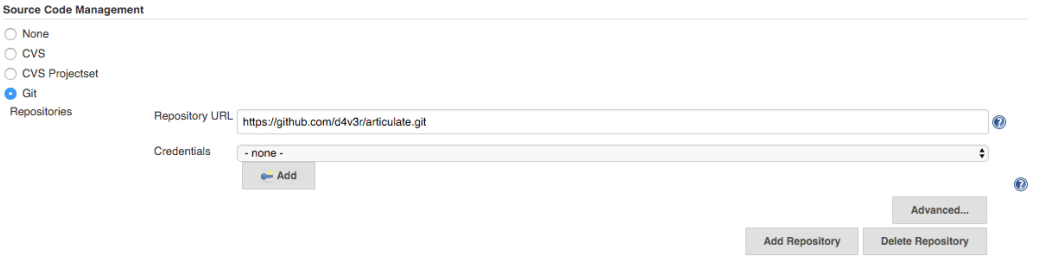
6. Enter a name into the name field then, at the bottom of the page, click `Save`.



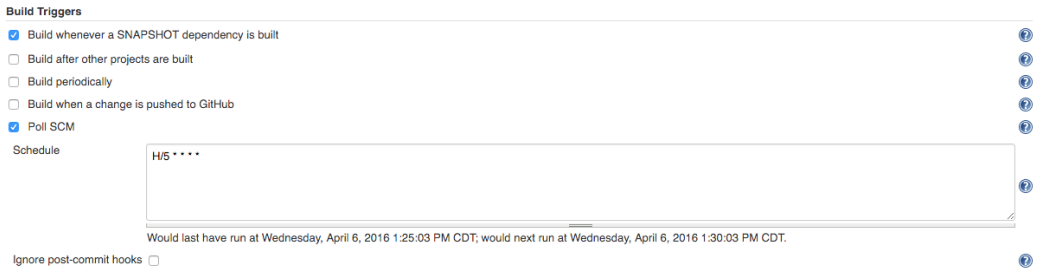
13.2.5. Create Build Job

1. From the Jenkins dashboard, click `New Item` (in top left corner), name it `articulate-{{initials}}` and select `Maven project` Then click `OK`.

2. Under `Source Code Management`, select `Git`, and supply your forked repository URL.



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



4. Under `Build`, enter `pom.xml` for `Root POM`, and `package` for `Goals and options`.

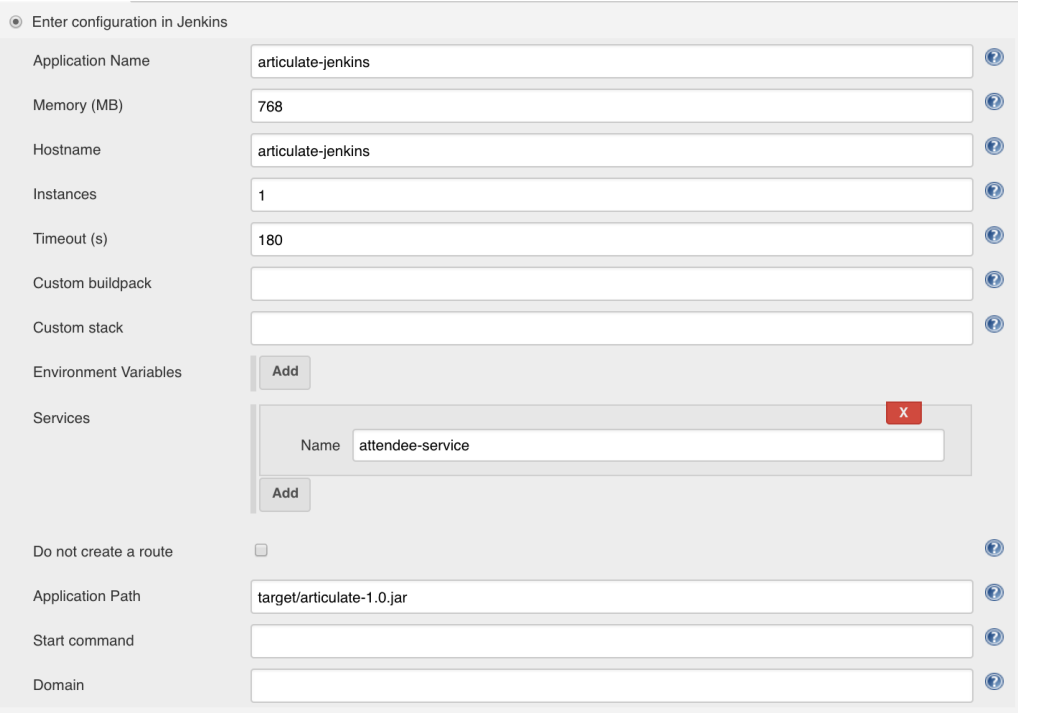
5. Under `Post-build Actions`, click the `Add post-build action`, and select `Push to Cloud Foundry`.

6. 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. Also check the `Reset app if already exists` checkbox. This allows for app bits and configuration to be updated/reset with each deployment; creating a more dependable way to redeploy the application (see the context sensitive help in Jenkins for more details).

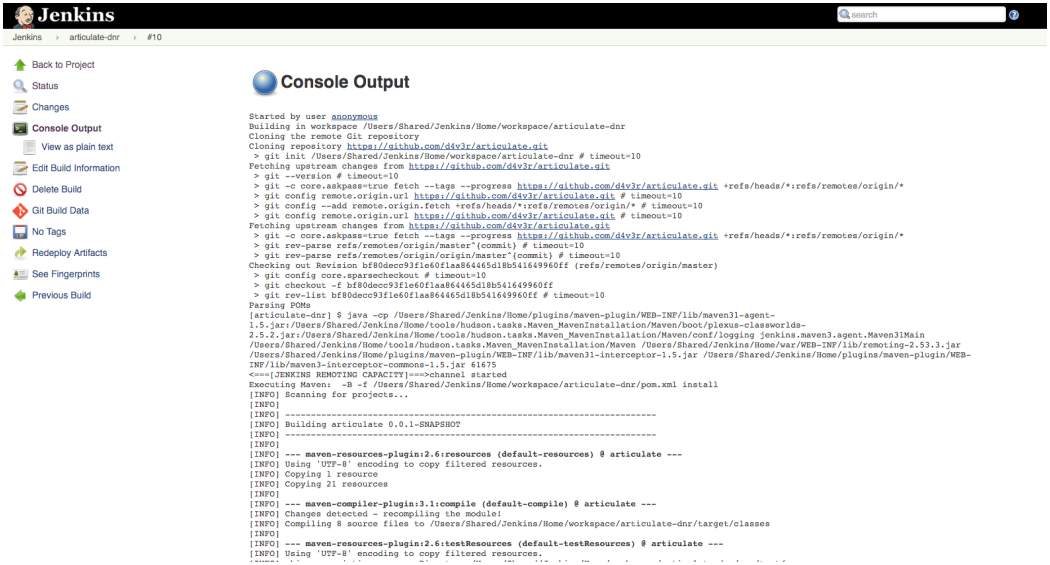
7. Make sure that `Enter configuration in Jenkins` is selected.
   Fill in the following fields:

   - `Application Name` = `articulate-{{initials}}`
   - `Memory` = `768`
   - `Hostname` = come up with something original and unique
   - `Instance` = `1`
   - `Timeout` = `180`

- `Services` = `attendee-service`
  Advanced Settings:
- `Application Path` = `target/articulate-1.0.jar`



8. Save the config and try running the build by clicking `Build Now`. Do not proceed past this step until you have a successful build and deployment to Pivotal Cloud Foundry. Confirm the application is deployed by viewing it in your browser.
   Make sure to view the Build details (Left side of screen -> `Build History` -> `Build #`).
   `Console Output` can be viewed there (for active or completed jobs). This is very useful for debugging failing builds.



9. In your forked repo, edit the Welcome message for Articulate.

   a. Edit the following file (can be done with a browser):https://github.com/{{github_username}}/pcf-articulate-code/blob/master/src/main/resources/templates/index.h

   b. Change the welcome message from `This is Articulate!` to `Welcome to My Articulate!` Commit and push the change to GitHub, wait until the polling detects it, and watch the magic. Verify the build in Jenkins now succeeds. Also verify your change in the deployed application with a browser.

Congratulations, you have finished this exercise!

13.2.6. Questions

- What are some of the benefits of continuous delivery?
- Does continuous delivery mean continuous deployment?

13.2.7. Cleanup

1. Delete the application that the pipeline deployed. For example:

```
cf delete articulate-{{initials}}
```

2. If provisioned, terminate your AWS Jenkins instance by going to your AWS EC2 dashboard, selecting the Jenkins instance, and clicking Actions -> Instance State -> Terminate.

13.3. Beyond the Class

The CD exercise above is very simplistic and should be expanded for real projects.

13.3.1. Artifact Repository

- Ideally, you want to build your artifacts (jars/wars) and publish them to a repository like Artifactory.
- Artifacts should be versioned to match the app deployments on PCF.
- All pushes to PCF should be using the same artifacts. Artifacts should be built once and used throughout the lifecycle.

13.3.2. Code promotion

- Jobs should be established in Jenkins to deploy/promote code to different phases like dev to test to prod.
- Jobs should use the same artifact published to Artifactory.
- Jobs can be triggered automatically or manually but should be fully automated. There should be no manual steps beyond clicking `build now`.

13.3.3. Notification

- Jenkins supports many notification plugins. It is important for code owners to be aware of build status.

13.3.4. Pivotal Cloud Foundry

Try out the CloudBees Jenkins Operations Center tile.

Chapter 14. Route Service

Estimated Time: 25 minutes

14.1. Preface

Route services can be put to use in many ways: to cache responses from back-end services, to control access to an application, to audit how an application is accessed, and more.

In this lab you'll deploy a route service that will allow you to rate-limit access to the `attendee` application. In the process you'll learn yet more cf CLI commands, namely the `create-user-provided-service` command with a new flag (`-r`), and the `bind-route-service` command.

Don't forget to examine the source code to understand how this route service is implemented.

14.2. Exercises

14.2.1. Setup

1. The `~/labs/source-code/route-service` directory included as part of your lab files contains the source code and jar ready for you to deploy (no building necessary).. OPTIONAL STEP - Import applications into your IDE such as Spring Tool Suite (STS).
   STS Import Help:
   Select File#Import. Then select Maven#Existing Maven Projects. On the Import Maven Project page, browse to the route-service directory. Then push the "Next" button. Click "Finish".

14.2.2. Route Service Overview

1. Review the documentation onRoute Services.

14.2.3. Scenario

Route services can be used for a number of things such as logging, transformations, security and rate limiting.

Our `rate-limiter-app` application will do a couple of things. It will log incoming and outgoing requests. It will also impose a rate limit. No more than 3 requests per 15 seconds. Rate limited requests will be returned

with a HTTP status code 429 (too many requests). Rate limiting is very common in the API space. Rate limiting protects your API from being overrun. The `rate-limiter-app` application will keep its state in Redis.

The `attendee` service exposes a RESTful API, so we will front it with our `rate-limiter-app`.

14.2.4. Implementing `rate-limiter-app`

1. Review the following file: `~/labs/source-code/route-service/src/main/java/org/cloudfoundry/example/Controller.java`.

```
@RestController
final class Controller {
    static final String FORWARDED_URL = "X-CF-Forwarded-Url";

    static final String PROXY_METADATA = "X-CF-Proxy-Metadata";

    static final String PROXY_SIGNATURE = "X-CF-Proxy-Signature";

    private final static  Logger logger = LoggerFactory.getLogger(Controller.class);

    private final RestOperations restOperations;

    private RateLimiter rateLimiter;

    @Autowired
    Controller(RestOperations restOperations,  RateLimiter rateLimiter) {
        this.restOperations = restOperations;
        this.rateLimiter = rateLimiter;
    }

    @RequestMapping(headers = {FORWARDED_URL, PROXY_METADATA, PROXY_SIGNATURE})
    ResponseEntity<?> service(RequestEntity<byte[]> incoming) {
        logger.debug("Incoming Request: {}", incoming);
        if(rateLimiter.rateLimitRequest(incoming)){
            logger.debug("Rate Limit imposed");
            return new ResponseEntity<>(HttpStatus.TOO_MANY_REQUESTS);
        };
        RequestEntity<?> outgoing = getOutgoingRequest(incoming);
        logger.debug("Outgoing Request: {}", outgoing);

        return this.restOperations.exchange(outgoing, byte[].class);
    }

    private static RequestEntity<?> getOutgoingRequest(RequestEntity<?> incoming) {
        HttpHeaders headers = new HttpHeaders();
        headers.putAll(incoming.getHeaders());
        URI uri = headers.remove(FORWARDED_URL).stream()
            .findFirst()
            .map(URI::create)
.orElseThrow(() -> new IllegalStateException(String.format("No %s header present", FORWARDED_URL)));

        return new RequestEntity<>(incoming.getBody(), headers, incoming.getMethod(), uri);
    }
}
```

What's happening?

The `service` method is where the `rate-limiter-app` application handles incoming requests.

1. Any request with the `X-CF-Forwarded-Url`, `X-CF-Proxy-Metadata`, and `X-CF-Proxy-Signature` headers gets handled by the `service` method.

2. Log the `incoming` request.

3. Check the `rateLimiter` to see if the number of requests has exceeded the rate limit threshold. If the threshold is exceeded return a HTTP status code 429 (too many requests). If the threshold is not exceeded remove the `FORWARDED_URL` header, log the `outgoing` request, and send the `outgoing` request to the downstream application.

2. Review the following file: `~/labs/source-code/route-service/src/main/java/org/cloudfoundry/example/RateLimiter.java`.

```
@Component
public class RateLimiter {
    private final static  Logger logger = LoggerFactory.getLogger(RateLimiter.class);
    private final String KEY = "host";

    @Autowired
    private StringRedisTemplate redisTemplate;

    @Scheduled(fixedRate = 15000)
    public void resetCounts() {
        redisTemplate.delete(KEY);
        logger.debug("Starting new 15 second interval");
    }

    public boolean rateLimitRequest(RequestEntity<?> incoming)  {
        String forwardUrl = incoming.getHeaders().get(Controller.FORWARDED_URL).get(0);
        URI uri;
        try {
            uri = new URI(forwardUrl);
        } catch (URISyntaxException e) {
            logger.error("error parsing url", e);
            return false;
        }

        String host = uri.getHost();
        String value = (String)redisTemplate.opsForHash().get(KEY, host);
          int requestsPerInterval = 1;

        if (value == null){
            redisTemplate.opsForHash().put(KEY, host, "1");
        }
        else{
            requestsPerInterval = Integer.parseInt(value) + 1;
            redisTemplate.opsForHash().increment(KEY, host, 1);
        }

        if(requestsPerInterval > 3)
            return true;
        else
            return false;
    }
}
```


What's happening?

The `rateLimitRequest` method determines whether a request should be rate limited.

1. Increment the request count by host.
2. Return `true` if request should be rate limited (`requestsPerInterval > 3`).
3. Return `false` if request should not be rate limited (`requestsPerInterval # 3`).

The `resetCounts` method deletes the Redis `KEY` every 15 seconds, which resets the counts by deleting all the state.

Note

This is an example implementation for lab purposes only. A proper rate limiting service would need to uniquely identify the client. That can be accomplished via an API key, the `X-Forwarded-For` header, or other approaches.

14.2.5. Push `rate-limiter-app`

1. Push `rate-limiter-app`:

```
$> cd ~/labs/source-code/route-service/
$> cf push rate-limiter-app -p ./target/route-service-1.0.0.BUILD-SNAPSHOT.jar -m 768M --random-route
    --no-start
```

2. Create a Redis service instance
   Pivotal Web Services | Pivotal Cloud Foundry
   In PWS, the marketplace service for Redis is called "rediscloud".

```
cf create-service rediscloud 30mb redis
```

   Pivotal provides a redis managed service named "p-redis".

```
cf create-service p-redis shared-vm redis
```

3. Bind the service instance.

```
cf bind-service rate-limiter-app redis
```

4. Start the application.


```
cf start rate-limiter-app
```

14.2.6. Create a Route Service and Bind it to a Route

1. Create a user provided service. Let's call it `rate-limiter-service`.

```
cf create-user-provided-service rate-limiter-service -r {{ratelimiter_baseurl}}
```

2. Bind the `rate-limiter-service` to the `attendee` route.

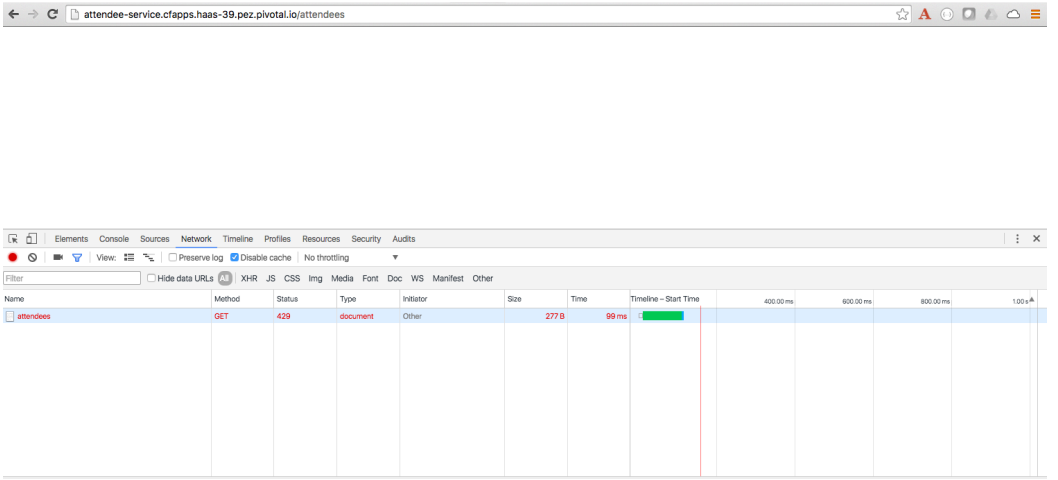
```
cf bind-route-service {{domain_name}} rate-limiter-service --hostname {{attendee_hostname}}
```

14.2.7. Observe the effects of the `rate-limiter-app`

1. Tail the logs of the `rate-limiter-app` application.

```
cf logs rate-limiter-app
```

2. Choose a client of your preference, but one that can show HTTP status code. Hit an `attendee` endpoint (e.g. `/attendees`) several times and see if you can get the rate limit to trigger. Observe the logs.
   Pic below is using Chrome with the Developer Tools.



14.2.8. Questions

- What are the key headers used to implement route services (Service Instance Responsibilities)?
- How would you apply route services in your environment?

14.2.9. Clean up

1. Unbind the route service.

```
cf unbind-route-service {{domain_name}} rate-limiter-service --hostname {{attendee_hostname}}
```

2. Delete `rate-limiter-service` service instance.

```
cf delete-service rate-limiter-service
```

3. Unbind `redis` service instance from the app.

```
cf unbind-service rate-limiter-app redis
```

4. Delete the `redis` service instance.

```
cf delete-service redis
```

5. Delete the `rate-limiter-app` app.

```
cf delete rate-limiter-app
```


Chapter 15. CF Tasks

15.1. Preface

`attendee` is configured to automatically create a database schema on startup. You can verify this by inspecting the source code of the `attendee` project.

Study the file `src/main/resources/application.yml`, and note how it contains the setting `generate-ddl`:

```
spring:
  application:
    name: attendee
  jpa:
    generate-ddl: true
```

This setting ensures that when the application starts, the schema will be auto generated from the JPA domain definition.

In this lab, we'll turn off this feature, and instead learn how to run a "one-off" task that will create the needed schema for us explicitly, as a separate step.

15.2. Setup

1. If you don't have it already, Maven is included in your lab files in `apache-maven-3.5.0-bin.zip`.
   - Unpack it and add …/apache-maven-3.5.0-bin/bin to your `PATH`
   - Close any terminal/command windows and open a new one to pick up the changed \PATH\

Warning

You will need access to Maven Central to download dependencies. If your company mandates its own local Maven repository, it may not contain the right dependencies and you won't be able to do this lab.

2. The `attendee` source code has been included in your lab files.
3. Edit the `application.yml` file and set `generate-ddl` to false. Rebuild the application jar file with:

```
mvn clean package
```


4. Do whatever's necessary to delete your existing `attendee` application.

5. Delete the mysql database you'd originally created for the `attendee` application using the command `cf delete-service`

6. Now, install the `mysql` cf cli plugin from the included lab files (in the `cf-plugins.zip` file) using the following command:

```
cf install-plugin /path/to/cf-mysql-plugin
```

   Alternatively you can install it directly from the cloudfoundry plugins web site. This will add the command 'mysql' to your cf cli.

7. Recreate your attendees database, perhaps like this:

```
cf create-service cleardb spark attendee-db
```

   Validate that the service was created by invoking the `cf services` command.

8. Push `attendee` application once more. Assuming you have a manifest file for `attendee` that looks similar to this:

```
---
applications:
- name: attendee
  path: target/attendee-1.0.jar
  memory: 768M
  random-route: true
  services:
  - attendee-db
```

   Invoke your `cf push` command from within the `attendee` base directory:

```
cf push
```

9. Now, invoke the `cf mysql` command, which should produce output similar to this:

```
MySQL databases bound to an app:

attendee-db
```

10. Let's check that the database has no tables at this time:

```
cf mysql attendee-db
```

   then:

```
mysql> show tables;
```

   And ensure that the output shows that no tables exist at this time.

In fact, let's go further:

1. Tail the logs for `attendee`

```
cf logs attendee
```

2. In a browser, visit the `attendees/` endpoint and verify that you get an error, and verify that the logs contain an error message complaining that the attendees table does not exit.

15.3. Run a "one-off" task to generate the schema

Familiarize yourself with Cloud Foundry tasks by reviewing the documentation.

It turns out that our spring boot application already has the knowledge for creating our schema. That knowledge is embedded in the spring data jpa libraries. The idea is simple:

- we'll run a spring-boot app not as a web service, but as a task by turning off the spring-boot property `spring.main.web-environment`
- next, for this run, we'll turn on the `spring.jpa.generate-ddl` property
- we'll specify a start command for our task that leverages everything that's already inside our droplet: the jdk and our spring boot application

Here's the command we want to run:

```
cf run-task attendee '.java-buildpack/open_jdk_jre/bin/java \
    -Dspring.jpa.generate-ddl=true \
    -Dspring.profiles.active=cloud \
    -Dspring.main.web-environment=false \
    -cp . org.springframework.boot.loader.JarLauncher'
```

You'll see output similar to this:

```
Creating task for app attendee in org eitan-org / space eitan-space as esuez@pivotal.io...
OK

Task has been submitted successfully for execution.
task name:   fb9e9533
task id:     1
```

Run the 'cf tasks' command to check on the status of our task:

```
cf tasks attendee
```

The output should resemble this:


```
id   name       state       start time                      command
1    fb9e9533   SUCCEEDED   Thu, 11 May 2017 02:45:15 UTC   .java-buildpack/open_jdk_jre/bin/java \
    -Dspring.jpa.generate-ddl=true \
    -Dspring.profiles.active=cloud \
    -Dspring.main.web-environment=false \
    -cp . org.springframework.boot.loader.JarLauncher
```

Note that the state is `SUCCEEDED`.

Let's now verify that our table has been created:

```
cf mysql attendee-db
```

and:

```
mysql> show tables;
```

..should show:

```
+-----------------------------+
| Tables_in_ad_12f26b7197bb693 |
+-----------------------------+
| attendee                    |
+-----------------------------+
1 row in set (0.03 sec)
```

Now, attempt to revisit the `attendees` endpoint once more, and you'll see that the call succeeds.

Congratulations, you've just run a one-off task inside cloudfoundry!

Acknowledgements to Liu Dapeng and his published example.

Appendix A. Windows VM

A Windows VM is available to complete the labs. The VM is hosted on Amazon Web Services. You will access it with remote desktop client.

A.1. Who Should Use?

Please consider using this VM if you have any of the following limitations:

- Your system does not meet the technical requirements
- You can't install software on your system
- You have limited network access

A.2. Requirements

- Remote Desktop Client on your system (most operating systems ship with one)
- Network connectivity and access to reach the Windows VM (RDP uses port 3389)

A.3. Provisioning

1. User must have access to an AWS account (can run from any account)
2. User must have an existing key pair in the account or create one
3. The AMI can be found be searching for `WinServer08STS` in the EC2 Dashboard. Must be in the N. Virginia region.
   Launch the AMI named `WinServer08STS`. In AWS, in the EC2 dashboard, using the US East (N. Virginia) region, select (or search for the public image) Images > AMIs > WinServer08STS. Click Launch.

Note

pick the AMI with the greatest number appended to the AMI Name. In this case, `WinServer08STS7` image::ami.png[search for ami]

4. When launching, accept defaults except:

    a. Choose an Instance Type > `t2.medium` (4 GB memory)

    b. Configure Instance Details > Auto-assign Public IP > `Enable`

    c. Tag Instance > Key = `Name`; Value = `<your name here-windows>`

5. Launch the instance. Once the instance is running, select it in list of EC2 instances. Obtain the public IP.

6. Using your Remote Desktop connect to the Windows VM using the the public IP.
   Username: `Administrator`
   Password: `KeepItSimple123`

A.4. Software Installed

The software listed below has been installed and configured on the VM:

- JDK 1.8 - License
- Maven - License
- git - License
- curl - License
- cf - License
- Spring Tool Suite - License
- Cygwin - License
- Chrome - License
- Json Formatter for Chrome

A.5. Best Practices

A.5.1. Work Directory

As part of the VM there is a `repos` directory. It is located at `C:\Users\Administrator\repos`. This is where

you will clone repos from GitHub. This will be referred to as `$REPOS_HOME` throughout the labs.

A.5.2. ConEmu

The Windows console is very limited in the sense that it lacks tab support, search and other features one might like in a console/terminal experience.

Therefore, also installed is an alternative to the command prompt known as ConEmu.

We recommend using ConEmu to execute the labs. You can launch ConEmu from the desktop. It will open to your `$REPOS_HOME` directory.

A.5.2.1. Use Labeled Tabs

As part of the labs you will be starting many processes. Organizing the processes with tabs makes a ton of sense, otherwise you will have windows everywhere making it difficult to manage.

To label the tab execute the following:

```
title - <tab label>
```

For example, creating a tab for the `config-server`. This is where work done with the `config-server` would take place.

A.5.3. Odd Keyboard Behavior?

Sometimes when using Remote Desktop keys (`control`, `command`, `shift`, etc.) may become depressed by accident leading to what seems like very odd behavior on the remote machine. The problem can typically be resolved by toggling the depressed key(s). If needed the **On-Screen Keyboard** can also be brought up.

Open the **On-Screen Keyboard** by clicking the `Start` button, clicking `All Programs`, clicking `Accessories`, clicking `Ease of Access`, and then clicking `On-Screen Keyboard`.

Appendix B. Setting Up Mongo DB VM on AWS

B.1. Context

If you are attending an Instructor-led or Live-Online course, your instructor will do this for you.

If you are taking this course as E-Learning, you will need to do this yourself unless you already have access to a running MongoDB system.

B.2. Setup Mondgo DB VM

We will be using Terraform (from Hashicorp) to create our VM.

You should have a file called `mongo.tf` as part of your lab files in `labs/tools/terraform`.

Go to https://www.terraform.io, download and install the software. Ensure that `terraform` is on your `PATH`.

Login to your AWS account and navigate to the EC2 dashboard. *Make sure you have US East (N. Virginia) set as your active region.*

B.2.1. Keys and Configuration

You need a security key-pair. If you don't have one already:

- In the left-hand panel, click `Key Pairs`.
- Then click `Create Key Pair` and specify your key-pair name
- Create the key-pair and download the PEM file when prompted.
- Copy the PEM file you have downloaded to the directory where you have placed the `mongo.tf` file
- If running on Unix (MacOS/Linux) restrict access using `chmod 400 <your-key>.pem`
- DO NOT lose this file, there is no way to recover it

The VM configuration is specified using the `mongo.tf` file that you should have downloaded previously. It takes a MongoDB AMI (VM template) and wraps security around it.

The `mongo.tf` file expects certain variables to make it work. Create a file called `variable.tf` and copy this text into it:

```
variable "access_key" {
    default = "AWS-ACCESS-KEY"
}

variable "secret_key" {
    default = "AWS-SECRET-KEY"
}

variable "key_name" {
    default = "YOUR-KEY-PAIR-NAME"
}

variable "key_path" {
    default = "/absolute/path/to/my-key-pair.pem"
}

variable "region" {
    default = "us-east-1"
}

variable "username" {
    default = "YOUR_AMS_LOGIN_NAME"
}
```

B.2.2. Creating the VM

For full details of using Terraform, go here: https://www.terraform.io/intro/getting-started/install.html.

Replace each of the defaults with their correct values.

Try and run your plan:

```
terraform mongo.tf plan
```

If it works, run

```
terraform mongo.tf apply
```

You may be prompted to go to AWS and purchase the VM instance you are about to create. Once AWS is happy, come back and run `terraform mongo.tf apply` again.

Once the script has run, it finishes by telling you the static IP address of your new VM. This is the `MONGODB_HOST` they need for the Service Btoker lab.

Return to EC2 dashboard and view your instance. Wait for it to finish starting up.

Select your VM, and in the Actions dropdown -> Instance Settings -> Get System Log. Scan the log for "Setting Bitnami application password to". See here for more details: https://docs.bitnami.com/aws/faq/#how-to-find-application-credentials. This is the `MONGODB_PASSWORD` needed for the lab.

B.2.3. Using the VM

This is all you need to do to finish the Service Broker Lab.

Once you have finished the lab, you can get rid of the VM by going to your AWS EC2 dashboard, selecting the MongoDB instance, and clicking `Actions` -> `Instance State` -> `Terminate`. The VM should stop, change to "Terminated" status and will eventually be removed from your list of VMs (The VM will shutdown and eventually be removed from your list of VMs (removal may not happen immediately)emoval may not happen immediately)

B.3. Accessing Your Mongo VM Remotely

If you wish you can access MongoDB and see what you have setup in the lab.

You can SSH into the VM using your private key, the `bitnami` username and the public IP address or hostname of your VM.

For example:

```
ssh -i my-key-pair.pem bitnami@53.91.185.0
```

For more information refer to the Amazon Reference Documentation

B.3.1. Managing MongoDB

Start MongoDB

```
sudo /opt/bitnami/ctlscript.sh start
```

Stop MongoDB

```
sudo /opt/bitnami/ctlscript.sh stop
```


Restart MongoDB

```
sudo /opt/bitnami/ctlscript.sh restart
```

See here for more information.

B.3.2. The Mongo Shell

The `mongo` shell is also loaded on the VM. It can be accessed in the terminal by doing the following:

```
mongo
```

Once in the shell, you'll have to authenticate. The username is `root`, and the password is the one you previously obtained from the VM system log:

```
use admin
db.auth('root', '<password>')
```

For more information refer to the Mongo Shell Reference

Appendix C. Amazon Web Services Jenkins AMI

A Jenkins VM (Ubuntu 14.04) is available to complete the continuous delivery lab. The VM is hosted on Amazon Web Services.

C.1. Requirements

Network connectivity and access for Jenkins to reach your Pivotal Cloud Foundry environment (push on your behalf).

C.2. Installed Software

The software listed below has been installed and configured on the VM:

- Jenkins - License
- JDK 1.8 - License
- Git - License
- GitHub Plugin - License
- Cloud Foundry Plugin - License

C.3. Provisioning

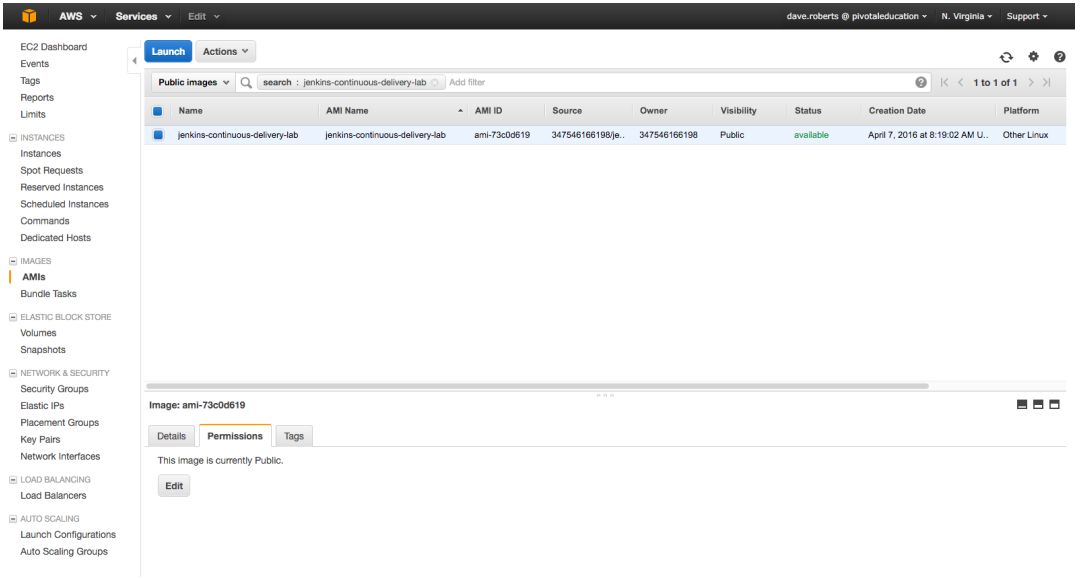
You can use the following `jenkins.tf` terraform script in you lab-files, in `labs/tools/terraform`.

Alternatively, you can provision the VM manually by following these directions:

1. User must have access to an AWS account (can run from any account)
2. User must have an existing key pair in the account or create one

3. The AMI can be found be searching for `jenkins-continuous-delivery-lab` in the EC2 Dashboard. Must be in the N. Virginia region.

In the left navbar of the EC2 dashboard, under `Images`, select `AMIs`. Then, to the left of the search bar, change `Owned by Me` to `Public images`. Search for `jenkins-continuous-delivery-lab`. Click Launch.



C.3.1. Provisioning Wizard

When launching, accept defaults except where specified.

C.3.1.1. Choose an Instance Type

- Use `t2.medium` (4 GB memory)

E-learning Students: **Please note that, though this instance type is not free-tier eligible, it is necessary for completing the continuous delivery lab and the PCF Developer course. Costs associated with its use are negligible. Charges cease as soon as the instance is deprovisioned (the steps for deprovisioning are at the end of the lab).**

C.3.1.2. Configure Instance Details

Auto-assign Public IP = `Enable`

C.3.1.3. Add Storage

Accept defaults

C.3.1.4. Tag Instance

Create the following: Key = `Name`; Value = `{{{yourname}}}-jenkins`

C.3.1.5. Create a Security Group

Create a security group with the following rules:

Rule 1.

```
Type = `ssh`
Protocol = `TCP`
Port Range = `22`
Source = `anywhere` (0.0.0.0/0)
```

Rule 2.

```
Type = `HTTPS`
Protocol = `TCP`
Port Range = `443`
Source = `anywhere` (0.0.0.0/0)
```

C.3.1.6. Review Instance Launch

Select your key-pair an launch the instance.

Once the instance is running, select it in list of EC2 instances.

Obtain the public IP for use in the lab .

C.4. Accessing Your Jenkins VM Remotely

C.4.1. HTTPS

A self signed certificate is used, so browser warnings are expected.

Access Jenkins via the https://{{{jenkins_ip_address}}}/ .

C.4.1.1. Credentials:

Username: `pivotal`

Password: `keepitsimple`

C.4.2. SSH

SSH using your private key, the `ubuntu` user and the public IP address of your VM.

For example:

```
ssh -i my-key-pair.pem ubuntu@{{jenkins_ip_address}}
```

Amazon Reference Documentation

C.5. Managing Jenkins

C.5.1. Start Jenkins

```
sudo service jenkins start
```

C.5.2. Stop Jenkins

```
sudo service jenkins stop
```

C.5.3. Restart Jenkins

```
sudo service jenkins restart
```


Appendix D. Certification

D.1. Introduction

When you are ready to take certification, here is how to register to take the online exam.

D.1.1. Procedure

Pivotal have partnered with PSI to run our Certification exams.

To take certification:

- Sit the relevant course or study up for the exam. Study Guides are available on the certification page for each exam (see below).
- You will need to buy a certification voucher from Pivotal:
  - Go to the Certification page and click on the exam (right hand panel)
  - In the popup panel, click the "Purchase Exam" button to buy a voucher. This popup also has a link to the Study Guide.
- Go to the PSI exams site https://www.examslocal.com/ and click the "*create an account*" button.
- Once you have registered an account, return to https://www.examslocal.com/ and in the search box enter "Spring". Then pick your exam and click `Next`. Make sure to select the right version of the exam to match the course you took.
- Use your voucher to purchase an exam and register a time.

See http://it.psionline.com/exam-faqs/pivotal-faq for more details on scheduling an exam.

Good luck!

