CF Scale and CF Logs

Table of Contents

Push the articulate application

1. Download the application

2. Copy the file to folder: ~/pivotal-cloud-foundry-developer-workshop/sample/ (~ is shorthand for the home directory in Linux, Mac and Unix based operating systems)
3. The given directory must be created in the home directory

Note: If the browser displays a warning about downloading this file, proceed to download it.

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

```
git clone https://github.com/pivotal-education/pcf-attendee-service-code.git

cd .\pcf-attendee-service-code

./mvnw package

cf push attendee-service -p .\target\attendee-1.0.jar -m 752M --random-route

cf bind-service attendee-service attendee-mysql


cf restart attendee-service


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


uri> https://attendee-service-appreciative-dingo.cfapps.io/attendees

$  git clone https://github.com/pivotal-education/pcf-articulate-code.git

$ cd pcf-articulate-code

$ ./mvnw clean package

$ cf push articulate -p .\target\articulate-1.0.jar -m 512M --random-route --no-start
```

Bind articulate to the attendee-service user provided service

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

Tip: Use ‘cf restage articulate’ to ensure env variable changes take effect” message at this time.

Restart the application.

```
$ cf restart articulate
```

Access articulate logs

1) Review the documentation on [application logging]

2) Tail the logs of the articulate application

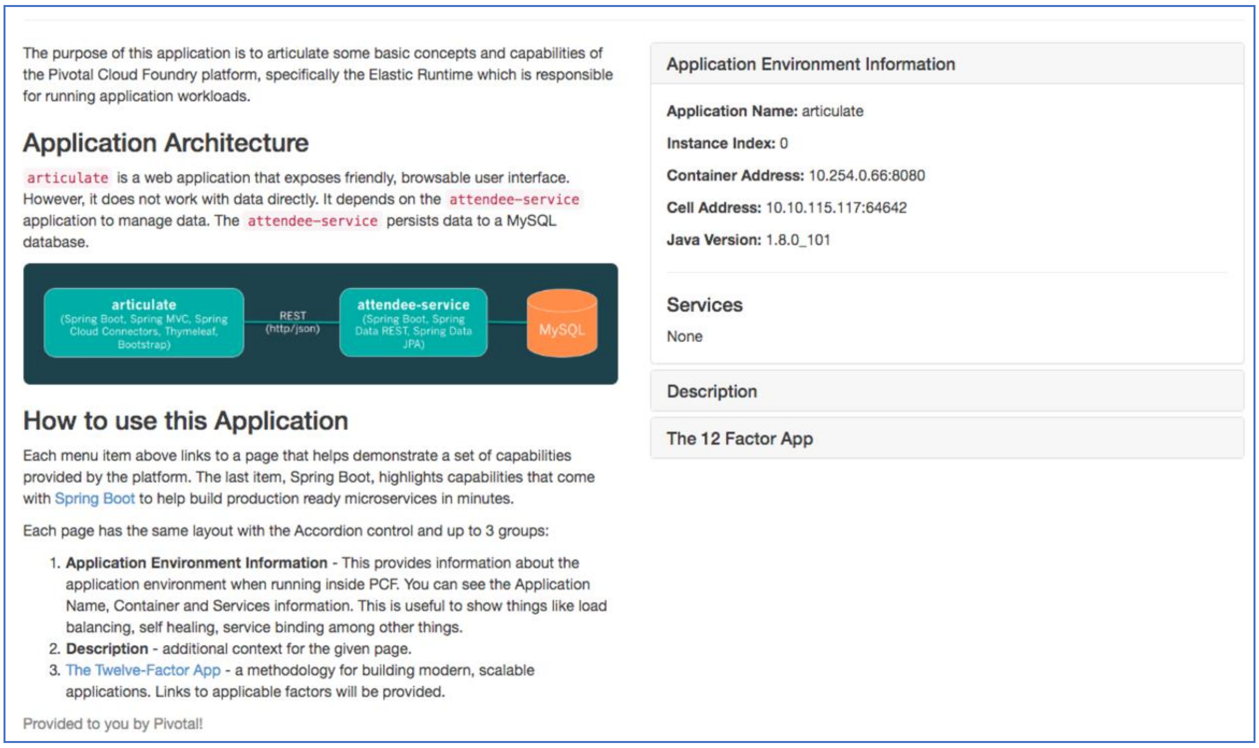
```
$ cf logs articulate
```

3) Open another terminal window and start the articulate application

4) Review the output from both terminal windows

```
$ cf start articulate
```

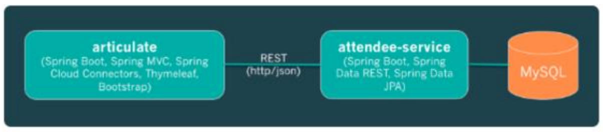
5) Open a browser and view the articulate application and read through the demo application



The purpose of this application is to articulate some basic concepts and capabilities of the Pivotal Cloud Foundry platform, specifically the Elastic Runtime which is responsible for running application workloads.

Application Architecture

`articulate` is a web application that exposes friendly, browsable user interface. However, it does not work with data directly. It depends on the `attendee-service` application to manage data. The `attendee-service` persists data to a MySQL database.


articulate
(Spring Boot, Spring MVC, Spring Cloud Connectors, Thymeleaf, Bootstrap)
REST
(http/json)
attendee-service
(Spring Boot, Spring Data REST, Spring Data JPA)
MySQL

How to use this Application

Each menu item above links to a page that helps demonstrate a set of capabilities provided by the platform. The last item, Spring Boot, highlights capabilities that come with Spring Boot to help build production ready microservices in minutes.

Each page has the same layout with the Accordion control and up to 3 groups:

1. **Application Environment Information** - This provides information about the application environment when running inside PCF. You can see the Application Name, Container and Services information. This is useful to show things like load balancing, self healing, service binding among other things.
2. **Description** - additional context for the given page.
3. The Twelve-Factor App - a methodology for building modern, scalable applications. Links to applicable factors will be provided.

Provided to you by Pivotal!

Application Environment Information

Application Name: articulate

Instance Index: 0

Container Address: 10.254.0.66:8080

Cell Address: 10.10.115.117:64642

Java Version: 1.8.0_101

Services

None

Description

The 12 Factor App

6) Observe the log output when the articulate web page is refreshed (More logs are added)

7) Stop tailing logs

1. Go to the terminal tailing the logs
2. Send an interrupt (Control + c)

Questions

- Where should the application write logs?
- What are some of the different origin codes seen in the log?
- How does this change the way the logs are accessed today, at scale?

Access articulate events

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

```
$ cf events articulate
```

Scale articulate

Scale up

1) Start tailing the logs again

```
[mac, linux]
$ cf logs articulate | grep "API\|CELL"

[windows]
$ cf logs articulate | findstr "API CELL"
```

The above statement filters only match the 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

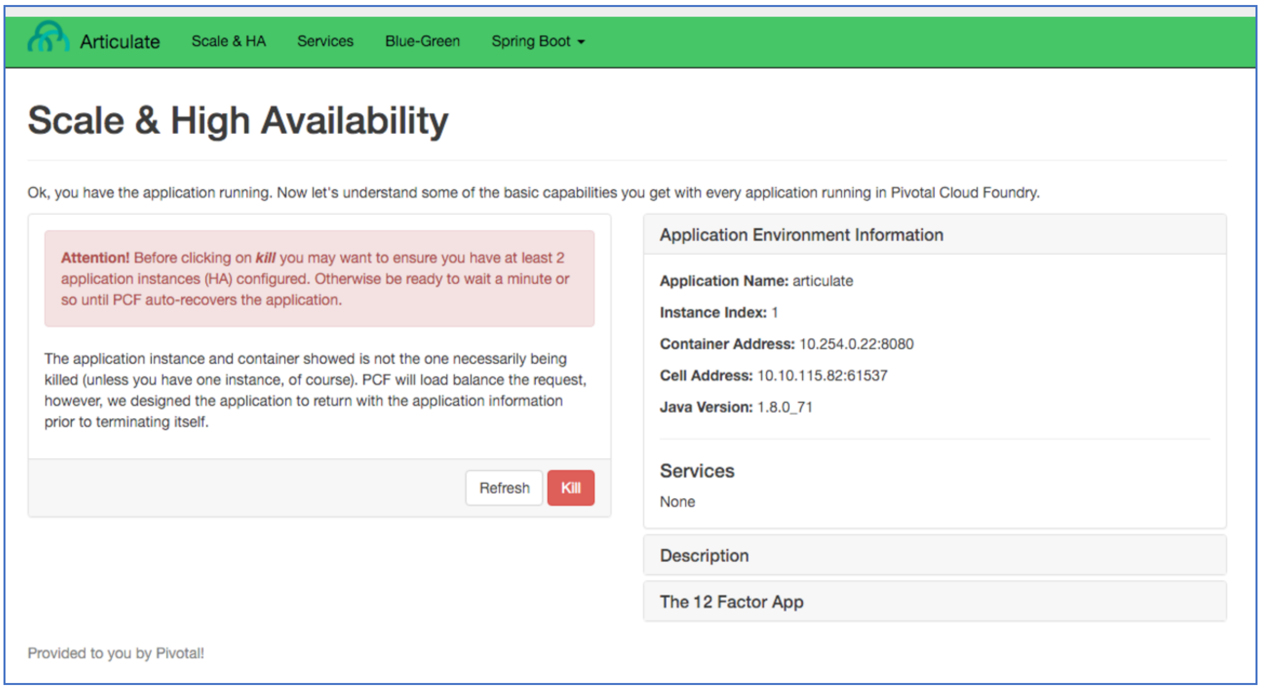
4) Stop tailing the logs

5) Scale articulate back to original settings

```
$ cf scale articulate -m 512M
```

Scale out

1) Browse to the Scale and HA page of the articulate application



Scale & High Availability

Ok, you have the application running. Now let's understand some of the basic capabilities you get with every application running in Pivotal Cloud Foundry.

Attention! Before clicking on ***kill*** you may want to ensure you have at least 2 application instances (HA) configured. Otherwise be ready to wait a minute or so until PCF auto-recovers the application.

The application instance and container showed is not the one necessarily being killed (unless you have one instance, of course). PCF will load balance the request, however, we designed the application to return with the application information prior to terminating itself.

Application Environment Information

Application Name: articulate
Instance Index: 1
Container Address: 10.254.0.22:8080
Cell Address: 10.10.115.82:61537
Java Version: 1.8.0_71

Services
None

Description

The 12 Factor App

Provided to you by Pivotal!

2) Review the Application Environment Information

3) Press the Refresh button multiple times (This sends all requests to one application instance)

4) Start tailing the logs

```
[mac, linux]
$ cf logs articulate | grep 'API\|CELL'


[windows]
```

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

5) In another terminal window, scale the articulate application

```
$ cf scale articulate -i 3
```

6) Observe log output and then stop tailing the logs

7) Return to articulate in a web browser

8) Press the Refresh button several times

9) Observe the Addresses and Instance Index change

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

Questions

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

High Availability

Pivotal Cloud Foundry has four levels of HA (High Availability) that keep your applications and the underlying platform running. Let us now look into how failed application instances can be recovered.

1) At this time multiple instances of articulate should be running

2) Confirm this with the following command:

```
$ cf app articulate
```

3) Return to articulate in a web browser (Scale and HA page)

4) Press the Refresh button and confirm that the application is running

5) Kill the app by pressing the Kill button

6) Check the state of the app through the cf CLI

```
$ cf app articulate
```

Sample output is given 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.

```
requested state: started
instances: 3/3
usage: 512M x 3 instances
urls: articulate.pcfi1.fe.gopivotal.com
last uploaded: Mon Mar 21 20:27:57 UTC 2016
stack: cflinuxfs2
buildpack: java-buildpack=v3.5.1-offline-http://github.com/pivotal-cf/pcf-java-buildpack.git#d6c19f8
java-main open-jdk-like-jre=1.8.0_65 open-jdk-like-memory-calculator=2.0.1_RELEASE spring-auto-
reconfiguration=1.10.0_RELEASE


     state      since                    cpu     memory          disk           details
#0   starting   2016-03-21 04:16:23 PM   0.0%    692K of 512M    93.4M of 1G
#1   running    2016-03-21 03:28:58 PM   0.5%    410.4M of 512M  158.8M of 1G
#2   running    2016-03-21 04:15:57 PM   23.9%   357.8M of 512M  158.8M of 1G
```

7) Repeat this command as necessary until state = running

8) Refresh the articulate application from the browser (This brings the app back up)

9) A new, healthy app instance has been automatically provisioned to replace the failing one

10) View which instance was killed

```
$ cf events articulate
```

11) Scale articulate back to original settings

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

Questions

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