

# Logging, Scale, and HA

## Table of Contents

Requirements

What you will learn

Exercises

- Push the articulate application

- Access articulate logs

  - Questions

- Access articulate events

- Scale articulate

  - Scale up

  - Scale out

  - Questions

- High Availability

  - Questions

Beyond the class

Estimated Time: 30 minutes

## Requirements

---

This lab is part of a workshop available at <https://github.com/Pivotal-Field-Engineering/DevNexus2017>  
General Pre-Requisites are available in the Readme.md

In general the Labs should be done in numerical order since they are interdependent.

# What you will learn

---

- How to access application logs
- How to scale an application
- How to access events
- How Pivotal Cloud Foundry handles failed application instances

## Exercises

---

### Push the articulate application

---

1) Find the `(/sample-apps/articulate/articulate-0.0.1-SNAPSHOT.jar)` application.

This directory is a subdirectory of the "DevNexus2017" git project that you downloaded in Lab 1.

Source (<https://github.com/pivotal-enablement/articulate>) is not required, but you may be curious how it works as you move through the labs.

2) Push the articulate application.

```
$ cd sample-apps/articulate
$ cf push articulate -p ./articulate-0.0.1-SNAPSHOT.jar -m 512M --random-route --no-start
```

# Access articulate logs

---

1) Review the documentation on application logging  
(<http://docs.pivotal.io/pivotalcf/devguide/deploy-apps/streaming-logs.html>).

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.



# Welcome to Articulate!

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.



## 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:** 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

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

6) Stop tailing logs

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

## 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?

## 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 matching log lines from the Cloud Controller (<https://docs.pivotal.io/pivotalcf/concepts/architecture/cloud-controller.html>) and Cell (<https://docs.pivotal.io/pivotalcf/concepts/architecture/#diego-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 our original settings.

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

## Scale out

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

 Articulate Scale & HA Services Blue-Green Spring Boot ▾

## 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.

Refresh Kill

### 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!

### Review the Application Environment Information.

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

### 3) Start tailing the logs.

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

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

4) In another terminal window, scale the `articulate` application.

```
$ cf scale articulate -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!*

## Questions

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

## High Availability

Pivotal Cloud Foundry has 4 levels of HA (<https://blog.pivotal.io/pivotal-cloud-foundry/products/the-four-levels-of-ha-in-pivotal-cf>) (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.

```
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



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
```

## 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?

## Beyond the class

- Try the same exercises, but using Apps Manager instead

[Back to TOP](#)

© Copyright Pivotal. All rights reserved.