# Prerequisites

Please review the requirement below and **complete the following prerequisites before class**.

## Experience

This training is targeted at people with:

* Basic Cloud Foundry experience
* Some experience deploying web-based applications
* Comfortable using the command line and SSH

## System Requirements

* Linux, Windows or OS X (all 64bit)
* 1GB of FREE RAM
* ability to install software
* code editor of choice
* [Pivotal Web Services](https://run.pivotal.io/) account or ability to create one
* [cURL](https://curl.haxx.se/) ([for windows](https://curl.haxx.se/download.html#Win64))

Lab 1 From Monoliths to Microservices

http://cloud-native-workshop.cloudfoundry.org/labs/setup/

# Setup

## Pivotal Web Services

We will use Pivotal Web Services for this class.

* If you don’t have an account, sign up now at: [try.run.pivotal.io/homepage](https://try.run.pivotal.io/homepage)
* You will also need the Cloud Foundry command line interface: [console.run.pivotal.io/tools](https://console.run.pivotal.io/tools)

If you already have the CLI, be sure you have the most recent version.

cf --version

cf version 6.21.1+6fd3c9f-2016-08-10

If your version is not more recent that this one, please install the latest.

### Checking Your Work

Be sure you have correctly installed the cli. From a terminal window/command prompt:

cf

You should see the self documenting help text. This will be very useful as you go through the class.

## CF Help

If you checked your work, you noticed the CF cli is self documenting. You can run cf help at any time to see a list of commands. You can also run cf <SOME\_COMMAND> --help to see the details for a specific command.

### Login & Target

Use cf login to target and login to Pivotal Web Services.

If you are new to PWS, you will notice you are automatically directed to your org and the ‘development’ space.

#### Checking Your Work

You should see output similar to:

API endpoint: https://api.run.pivotal.io (API version: 2.56.0)

User: sgreenberg@pivotal.io

Org: Pivotal-Enablement

Space: development

Alternatively, you can check where you are logged in and targeted at anytime using cf target.

Lab 2 Microservices & 12 Factor Apps

http://cloud-native-workshop.cloudfoundry.org/labs/push/

# Pushing a data service

In this exercise, you will push a restful data microservice.

Note: Currently this microservice does not adhere to all of the 12 factor app principles.

## The App

The app is a very simple Spring Data Rest application. You can download the jar file here: [people.jar](http://cloud-native-workshop.cloudfoundry.org/resources/people.jar).

If you are comfortable with git and gradle, you can also clone the source here [github.com/spgreenberg/people](https://github.com/spgreenberg/people) and build it yourself.

### Push

Use cf push to deploy the application. You should:

* Create 1 instance of the app
* Allocate 750M of memory
* Use the -p flag to point to the jar file
* Use –random-route to ensure no collisions with other students

cf push people --random-route -b java\_buildpack -p ...

#### Checking Your Work

Make sure your app deployed correctly:

cf apps

...

name requested state instances memory disk urls

people started 1/1 750M 1G people-<RANDOM\_ROUTE>.cfapps.io

The app also has an endpoint called /people. You should also be able to curl it:

On Windows, if you don’t have curl you can get it from here: [curl.haxx.se/download.html](https://curl.haxx.se/download.html)

curl people-<RANDOM\_ROUTE>.cfapps.io/people

...

{

"\_embedded" : {

"people" : [ ]

},

"\_links" : {

"self" : {

"href" : "http://people-<RANDOM\_ROUTE>.cfapps.io/people"

},

"profile" : {

"href" : "http://people-<RANDOM\_ROUTE>.cfapps.io/profile/people"

},

"search" : {

"href" : "http://people-<RANDOM\_ROUTE>.cfapps.io/people/search"

}

},

"page" : {

"size" : 20,

"totalElements" : 0,

"totalPages" : 0,

"number" : 0

}

}

## Adding Data

You can add data by issuing a curl request to POST to the people endpoint. Be sure to replace your information and URL below:

curl -X POST -H "Content-Type:application/json" -d '{"firstName":"Steve", "lastName":"Greenberg", "company":"Pivotal"}' http://people-<RANDOM\_ROUTE>.cfapps.io/people

Now, you should see the data:

curl http://people-<RANDOM\_ROUTE>.cfapps.io/people

...

{

"\_embedded" : {

"people" : [ {

"firstName" : "Steve",

"lastName" : "Greenberg",

"company" : "Pivotal",

"\_links" : {

"self" : {

"href" : "http://people-<RANDOM\_ROUTE>.cfapps.io/people/1"

},

"person" : {

"href" : "http://people-<RANDOM\_ROUTE>.cfapps.io/people/1"

}

}

} ]

},

"\_links" : {

"self" : {

"href" : "http://people-<RANDOM\_ROUTE>.cfapps.io/people"

},

"profile" : {

"href" : "http://people-<RANDOM\_ROUTE>.cfapps.io/profile/people"

},

"search" : {

"href" : "http://people-<RANDOM\_ROUTE>.cfapps.io/people/search"

}

},

"page" : {

"size" : 20,

"totalElements" : 1,

"totalPages" : 1,

"number" : 0

}

}

### Restart your App

* Now restart your app with cf restart.
* Curl your app again:

curl http://people-<RANDOM\_ROUTE>.cfapps.io/people

What happened? Why? Not very durable, right? We will correct this in the next exercise.

## Beyond the Class

* What aspects of 12 factor have you already experienced? [12factor.net](http://12factor.net/)

Lab 3 12 Factor Review

http://cloud-native-workshop.cloudfoundry.org/labs/bind/

# Binding and Environment Variables

In the last section, we lost all our data when we restarted our app. In this section, we will fix that.

## Creating a MySQL instance

We will create an instance of mysql and bind it to our app, thereby removing state from memory.

* Use cf marketplace to view the available services and plans.
* Use cf create-service to create a MySQL service instance cleardb and select the free plan spark.

### Checking Your Work

You should be able to see your service instance:

cf services

...

name service plan bound apps last operation

people-mysql cleardb spark create succeeded

## Binding to Your App

You need to bind your service instance to your application so that is can be used.

* Use cf bind-service to bind your service instance to your application.
* Restage your app so that it uses the new service: cf restage.

### Checking Your Work

You should be able to see your service instance bound to your app:

cf services

...

name service plan bound apps last operation

people-mysql cleardb spark people create succeeded

## Testing Statelessness

At this point, you should be able to put data into your service that lands in the external mysql service.

curl -X POST -H "Content-Type:application/json" -d '{"firstName":"Steve", "lastName":"Greenberg", "company":"Pivotal"}' http://people-<RANDOM\_ROUTE>.cfapps.io/people

* Restart your app.
* You should still see the data:

curl http://people-<RANDOM\_ROUTE>.cfapps.io/people

...

{

"\_embedded" : {

"people" : [ {

"firstName" : "Steve",

"lastName" : "Greenberg",

"company" : "Pivotal",

"\_links" : {

"self" : {

"href" : "http://people-<RANDOM\_ROUTE>.cfapps.io/people/2"

},

"person" : {

"href" : "http://people-<RANDOM\_ROUTE>.cfapps.io/people/2"

}

}

} ]

},

"\_links" : {

"self" : {

"href" : "http://people-<RANDOM\_ROUTE>.cfapps.io/people"

},

"profile" : {

"href" : "http://people-<RANDOM\_ROUTE>.cfapps.io/profile/people"

},

"search" : {

"href" : "http://people-<RANDOM\_ROUTE>.cfapps.io/people/search"

}

},

"page" : {

"size" : 20,

"totalElements" : 1,

"totalPages" : 1,

"number" : 0

}

}

Congrats! You now have a stateless app: [12factor.net/processes](http://12factor.net/processes)

## How does it work?

Run the following:

cf env people

This will print the environment variables for your application. Look for a System-Provided variable called VCAP\_SERVICES. You should see the service credentials for your mysql service. Note:

* Cloud Foundry leverage the environment variables: [12factor.net/config](http://12factor.net/config)
* Cloud Foundry treats services as attached resources: [12factor.net/backing-services](http://12factor.net/backing-services)

## Scale Out

By moving the state for your application into an external service, you can now scale out your application horizontally.

* Use cf scale to scale your app to 2 instances.

Notice that you can scale by adding instances: [12factor.net/concurrency](http://12factor.net/concurrency)

### Checking your Work

You should see 2 instances:

cf app people

...

0 running 2016-05-17 09:53:40 AM 0.1% 376.8M of 750M 153.7M of 1G

1 running 2016-05-17 10:01:35 AM 0.0% 232.1M of 750M 153.7M of 1G

## Scale Down

* Use cf scale to reduce your app back to 1 instance.

Notice that you can start quickly and dispose of unneeded instances gracefully: [12factor.net/disposability](http://12factor.net/disposability)

## Beyond the Class

* CF also allows you to manipulate environment variables or create your own: [docs.run.pivotal.io/devguide/deploy-apps/environment-variable.html](https://docs.run.pivotal.io/devguide/deploy-apps/environment-variable.html). Write an app that prints out all environment variables.
* With CF, you can create instances of services that point to existing endpoints with existing credentials: [docs.cloudfoundry.org/devguide/services/user-provided.html](http://docs.cloudfoundry.org/devguide/services/user-provided.html). Create a User Provided Service that points to a DB and bind it to an app.

Lab 4 Service Discovery

<http://cloud-native-workshop.cloudfoundry.org/labs/eureka/>

# Service Discovery with Eureka

In this exercise, you will deploy a [Eureka](http://cloud.spring.io/spring-cloud-netflix/) server. You will also deploy a new app browser that consumes data from our people service.

## About the App

The browser app uses Eureka to identify instances of our people service. It uses the [Ribbon](https://github.com/Netflix/ribbon) support built into [Spring Cloud Netflix](http://cloud.spring.io/spring-cloud-netflix/) to provide client side load balancing. The browser app also leverages [Feign](https://github.com/Netflix/feign) support which makes writing HTTP clients in java simple.

NOTE: PWS trial accounts are limited to 2GB, not enough capacity to scale our people-service instances.

## Deploying Eureka

First you need to deploy the Eureka server. A prebuilt jar is provided here [eureka.jar](http://cloud-native-workshop.cloudfoundry.org/resources/eureka.jar) or you can download and build the source from github [github.com/spgreenberg/eureka](https://github.com/spgreenberg/eureka).

* Push the Eureka server to cloud foundry

cf push eureka -p <path-to-jar> -m 512M --random-route -b java\_buildpack

### Checking Your Work

You should see the eureka app running: cf apps

### Accessing the Eureka Console

Eureka has a built in web UI that shows information on the services registered. You can access it by going to the URL in your browser. At this point, you won’t see any services registered.

NOTE: You can ignore the scary sounding error messages displayed in the UI. This is a single instance deployment of Eureka, not meant for production.

### Creating a User Provided Service for Eureka

We will need to tell our applications where our Eureka server is. There are many ways to do this, but for the purpose of this class, we will create a user provided service instance that can be bound to any client or service apps.

cf cups eureka-service -p '{"uri":"http://<YOUR\_EUREKA>"}'

## Registering the People Service

The people service is written with a Eureka client which is disabled by default. For the purpose of the class, we can use Spring Profiles to activate the Eureka client simply by setting an environment variable.

* Set SPRING\_PROFILES\_ACTIVE=cloud,eureka for your people service.

cf set-env people SPRING\_PROFILES\_ACTIVE cloud,eureka

* Use cf bind-service to bind the eureka-service you created above to your people service app.
* Use cf restage so the people service can pick up the changes.

### Checking Your Work

You should be able to see the environment variable.

cf env people

...

User-Provided:

SPRING\_PROFILES\_ACTIVE: cloud,eureka

You should also see the service instance:

cf services

...

name service plan bound apps last operation

eureka-service user-provided people

people-mysql cleardb spark people create succeeded

### Viewing Your Service in Eureka

Within a few minutes of restaging, you should see your people service registered in the Eureka console.

## Pushing the Browser App

* Now, push the browser app with 512MB of memory. The jar file is located here [browser.jar](http://cloud-native-workshop.cloudfoundry.org/resources/browser.jar) or you can download and build the project yourself: [github.com/spgreenberg/browser](https://github.com/spgreenberg/browser).
* Bind the Eureka service and restart the browser app.

### Checking Your Work

You should see the app.

cf apps

You should see the Eureka service bound to the browser app.

cf services

You should also see the browser app registered in the Eureka console.

### Using the browser app

The Browser app simply logs requests and results to the REST endpoints of the app.

* Open the browser app in your web browser
* In the Explorer text box, enter /people and hit GO

The first request might fail (not gracefully). This is b/c the browser service is still fetching information from Eureka. In the next exercise, we will add resiliency so we can fail gracefully.

### What is happening?

When successful, the browser app is using Eureka to locate the people service instances, then using Ribbon to load balance requests to those instances (b/c of quota limits, we only have 1 instance). Should we add/remove instances of the people service, or should that service move, updates will happen automatically.

Congratulations! You have successfully used service discovery to consume a microservice.

Lab 5 Circuit Breaking

http://cloud-native-workshop.cloudfoundry.org/labs/hystrix/

# Circuit Breakers with Hystrix

Currently, our browser app does not fail gracefully when an issue occurs communicating with the people service. Let’s fix that.

## Simulating Failure

We can simulate failure by stopping the people service app:

cf stop people

Now if you access your browser app and hit Go on the /people endpoint, you should see a 500 error.

{

"timestamp": 1463694694293,

"status": 500,

"error": "Internal Server Error",

"exception": "com.netflix.hystrix.exception.HystrixRuntimeException",

"message": "getPeople failed and no fallback available.",

"path": "/people"

}

## Enabling Hystrix

Our browser app actually has Hystrix built in, but currently disabled. You can enable Hystrix by setting the SPRING\_PROFILES\_ACTIVE environment variable to hystrix.

* cf set-env browser SPRING\_PROFILES\_ACTIVE hystrix
* cf restart browser

**Don’t** write your apps this way. You should always be using Hystrix in your spring code, but we do this for the purposes of teaching.

### Checking Your Work

You should see the environment variable:

cf env browser

### Seeing Hystrix in Action

Now if you access your browser app and hit Go on the /people endpoint, you should see a graceful failure:

Response Headers

200 OK

Date: Thu, 19 May 2016 22:17:46 GMT

Server: Apache-Coyote/1.1

X-B3-Traceid: e301c4c07b2ede89

Content-Type: application/hal+json;charset=UTF-8

X-Vcap-Request-Id: 44f30698-4e9b-4e8a-6e05-e6c1feada10d

X-B3-Spanid: e301c4c07b2ede89

X-B3-Sampled: 1

Connection: keep-alive

Content-Length: 2

X-Application-Context: browser:hystrix,cloud:0

...

Properties

{}

In this case, the fallback method is very simple. It returns an empty array. The fallback could be much more useful as well.

### Restoring…

Now, start your people app. Access the browser a few times again until service is restored.

Lab 6 Distributed Tracing

<http://cloud-native-workshop.cloudfoundry.org/labs/logs/>

# Logs

In this exercise, you will view logs for the browser application as well as instrumentation for distributed tracing in another demo application.

Notice that Cloud Foundry automatically treats your application logs as streams and makes them easily accessible: <http://12factor.net/logs>

## CLI Logs

Cloud Foundry aggregates logs related to your application. These include logs for Cloud Foundry components as well as calls to your application.

To help with tracing of microservice calls, the browser app has been instrumented using [Spring Cloud Sleuth](http://cloud.spring.io/spring-cloud-static/spring-cloud-sleuth/1.0.9.RELEASE/). This adds tracing information to your logs.

### Tailing Logs

* Use cf logs to tail logs for your browser application.
* Issue a few requests through the web interface for the browser application.

You should see logs from the Router component of CF

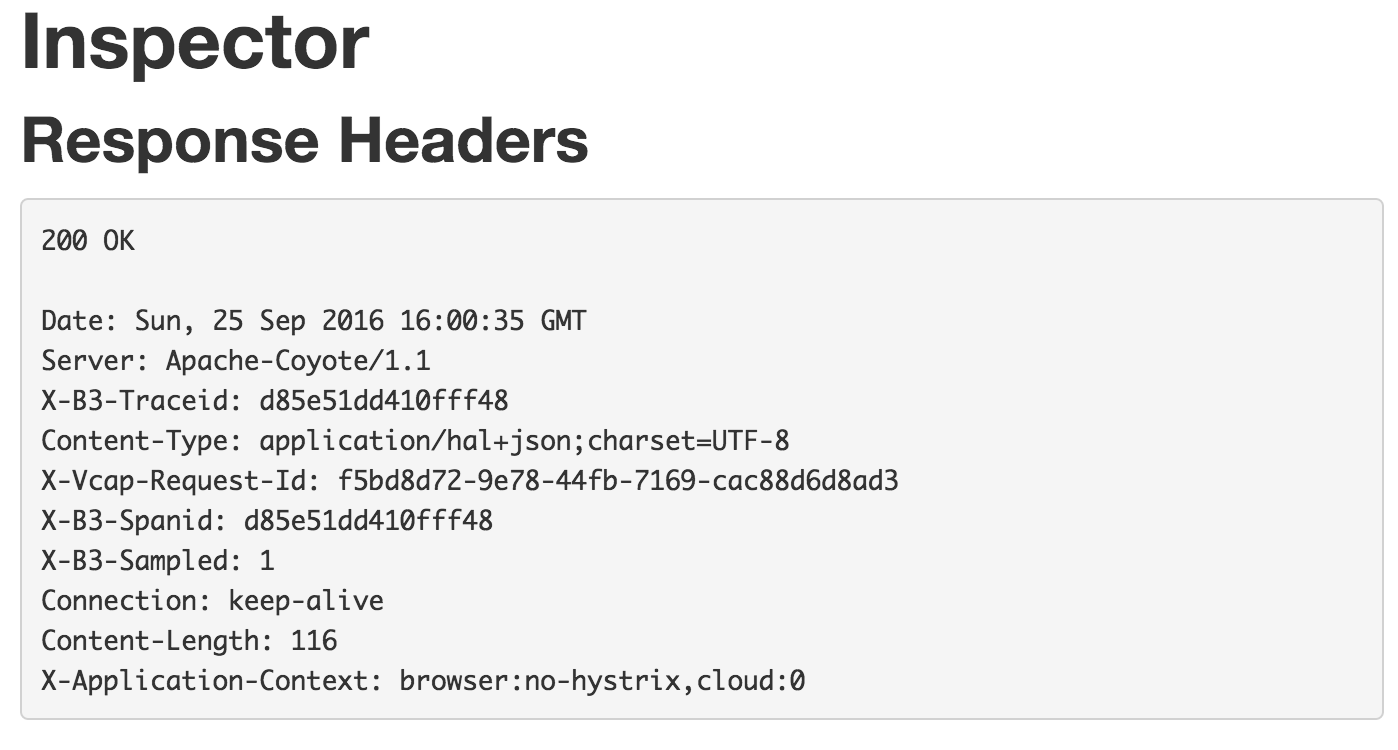
2016-05-20T12:50:59.07-0600 [RTR/5] <-- RTR is the Router.

and also your application`.

2016-05-20T12:52:11.07-0600 [APP/0] OUT 2016-05-20 18:52:11.068 INFO [browser,92ca8a46b9903cdc,92ca8a46b9903cdc,true]

The section [browser,92ca8a46b9903cdc,92ca8a46b9903cdc,true] is added by Spring Cloud Sleuth to display tracing information. This corresponds to [app-name from configuration, spanID, traceID,…].

This info is also available in the browser UI.

If you want to see just the application logs, you can run:

cf logs | grep APP

## Working with Zipkin

We have pushed four microservice applications that use Spring Cloud Slueth with an external [Zipkin](http://zipkin.io/) service.

The source for these applications can be found at: <https://github.com/mikegehard/DistributedTracingDemo_Velocity2016>

The Zipkin UI can be found at <http://zipkin-server-receivable-plasmolysis.cfapps.io/>.

In the previous section you saw span and trace ids added to the application logs. Now lets see how zipkin works with this information to provide visiblity into your microservice application.

If you want to add a new trace, you can curl any one of the microservices using:

curl http://acme-financial-ui-transferrable-conversazione.cfapps.io/start

curl http://acme-financial-account-microservice-parlourish-pumicer.cfapps.io/action

curl http://acme-financial-back-office-microservice-spoutless-nonfreeman.cfapps.io//action

curl http://acme-financial-customer-microservice-overeager-browser.cfapps.io//action

### Understanding the trace hierarchy

In your browser Navigate to the [dependencies](http://zipkin-server-receivable-plasmolysis.cfapps.io/dependency) to view the microservice hierarchy.

* Which microservices get called from the UI service?
* Which microservices are at the end of the call chain?

### Viewing trace timings

In your browser Navigate [here](http://zipkin-server-receivable-plasmolysis.cfapps.io/) to access the find trace form.

Clicking on the Find Traces button will bring up a list of existing traces.

Click on one of the traces and let’s dive into the details.

* How many services are invoved in this trace?
* How long does the total trace take?
* What HTTP endpoints get called for each span?
* What controller methods get called for each span?
* How long is spent in each span for the trace?
* Which leaf service takes the longest?

## Beyond the Class

* Stream Application Logs to [Papertrail](https://papertrailapp.com/" \t "_blank) via [these instructions.](https://docs.cloudfoundry.org/devguide/services/log-management-thirdparty-svc.html)Can you identify the log entries for each span?
* Use PCF Metrics to view detailed information about your app: [docs.run.pivotal.io/metrics/using.html](http://docs.run.pivotal.io/metrics/using.html)
* Deploy the Microservices Dashboard: [github.com/ordina-jworks/microservices-dashboard](https://github.com/ordina-jworks/microservices-dashboard)