**System Requirements**

* Linux, Windows or OS X (all 64bit)
* 1GB of free RAM
* USB port for mounting the USB drive with course material
* code editor of choice
* git client installed with Command Prompt / Terminal support

Lab 1 Sign Up - Pivotal Web Services

**Sign Up**

* Sign up by going to <https://run.pivotal.io/>

Lab 2 CLI Basics

In this section we will cover the basics of installing and using the CF command line interface (cli).

## Installing

* Select and install the appropriate installer for your laptop: [github.com/cloudfoundry/cli](https://github.com/cloudfoundry/cli#downloads)

#### Checking Your Work

Open a terminal/command prompt and type the following:

cf

You should see an output similar to the following:

NAME:

cf - A command line tool to interact with Cloud Foundry

USAGE:

[environment variables] cf [global options] command [arguments...

...

Notice that the CF cli is self documenting. You can type any command with --help to see details. Typing cf only lists all available commands.

cf <some-command> --help

## Logging in

You can use the CLI to log into any cloud foundry you have an account in.

* Use cf help to find out how to login to Pivotal Web Services

The API endpoint you need is api.run.pivotal.io.

cf login --help

#### Checking Your Work

You can see where you are logged in using the following:

cf target

* Are you logged into Pivotal Web Services?
* What org and space are you targeting?

## Targeting Orgs & Spaces

You can change the org and space you are targeting without logging in again.

$ cf target -o ORG -s SPACE

Lab 3 Pushing your first app

In this exercise, you will deploy an app to Cloud Foundry.

## Push my app

Be sure you are logged in and targeting your org/space.

* Push the app in 03-push/web-app to Cloud Foundry

$ cd 03-push/web-app

$ cf push

...

urls: web-app-unpassionate-eighteen.cfapps.io <<< note the route

### Deployment Manifest

This app is configured with a [deployment manifest](https://docs.cloudfoundry.org/devguide/deploy-apps/manifest.html). The manifest tells CF the app name and how many instances to create (among other things). Manifests are optional.

You can see the manifest by opening the file: 03-push/web-app/manifest.yml.

applications:

- name: web-app

random-route: true

### Random Route

The app deploys using random-route. Since the cfapps.io is shared by all [run.pivotal.io](https://run.pivotal.io/) apps, we need an easy way to deploy our app to this shared domain for development.

You can see the details on random-route using cf help:

cf push --help

So where is your app? When you pushed, you should have seen a message:

urls: web-app-unpassionate-eighteen.cfapps.io

Alternatively, you can look up the details on your app (next section).

#### Checking Your Work

* Use cf apps to see what apps are in the currently-targeted org/space

$cf apps

name state instances memory disk urls

web-app started 1/1 32M 256M web-app-unpass...

* Use cf app web-app to see more details of your app

$ cf app web-app

requested state: started

instances: 1/1

usage: 32M x 1 instances

urls: web-app-unpassionate-eighteen.cfapps.io

last uploaded: Mon Nov 2 10:18:05 UTC 2015

stack: cflinuxfs2

buildpack: ruby 1.6.7

state since cpu memory disk

#0 running 2015-11-02 0.0% 25.7M of 32M 95.1M of 256M

## Pushing worker apps

Not all apps need to respond to HTTP requests: instead they might do background work, such as consuming messages from a queue.

* Push the app in the worker-app directory
* What differences are there in the manifest? Why are these needed?

#### Checking Your Work

$ cf app worker-app

requested state: started

instances: 1/1

usage: 16M x 1 instances

urls:

last uploaded: Mon Nov 2 13:56:39 UTC 2015

stack: cflinuxfs2

buildpack: binary\_buildpack

state since cpu memory disk

#0 running 2015-11-02 0.0% 10.7M of 16M 27.3M of 64M

## Viewing Logs

The worker app outputs logs. Use cf help to determine what command to run to see recent logs.

#### Checking Your Work

You should see an output similar to:

2016-08-30T11:53:13.02+0100 [APP/0] OUT Doing some work...

## Make room (for better apps)

You can also delete apps.

* Delete the two apps you deployed (use cf help to find the correct command)

#### Checking Your Work

You should not see your apps when you run:

cf apps

Lab 4 Buildpacks

In this exercise, we will examine a running app to understand what a buildpack provides.

## System Buildpacks

* Use cf help to find out how to take a look at the buildpacks configured in PWS.
* If you don’t specify a buildpack, what is the first one that will be tested for?

## Pushing with the Static Buildpack

An app is included in the 04-buildpacks/static-app directory.

* Push the app in 04-buildpacks/static-app to CF using the provided manifest

#### Checking Your Work

You can view the details of your app:

$ cf app static-app

state since cpu memory disk

#0 running 2015-11-02 0.0% 6.5M of 16M 33.6M of 64M

## Exploring Buildpack Output

* How does the running droplet compare to your app directory?
* Use cf ssh static-app to explore the filesystem of the running application
* Observe what additional dependencies the buildpack made available for your app

## Scaling with Speed

* Use cf scale to scale your static app to 32 instances
* Why is CF able to scale instances so quickly?

#### Checking Your Work

You can view the details of your app:

$ cf app static-app

state since cpu memory disk

#0 running 2015-11-02 0.0% 6.5M of 16M 33.6M of 64M

#1 starting 2015-11-02 0.0% 0 of 16M 0 of 64M

#2 running 2015-11-02 0.0% 6.9M of 16M 33.5M of 64M

...

#30 running 2015-11-02 0.0% 6.8M of 16M 33.5M of 64M

#31 running 2015-11-02 0.0% 7M of 16M 33.6M of 64M

## Picking the Correct Buildpack

Cloud Foundry correctly used the Staticfile Buildpack to deploy your app. It did this because the app includes a file called Staticfile.

Change into 04-buildpacks/mixed-app and see that the directory contains both index.html and index.php.

* Which buildpack do you think will be used to run this app? Staticfile, or PHP?
* Use cf push to deploy mixed-app
* Observe from the CLI output which buildpack was used
* Hit both /index.html and /index.php
* Use cf ssh to see what files the buildpack has added

Instead of letting Cloud Foundry allow each buildpack to detect whether it can run the app, we’re going to specify which buildpack we want to use.

* Push the app again, using a flag to specify this buildpack: https://github.com/cloudfoundry/staticfile-buildpack (use cf push --help to find out which flag to provide)
* Observe from the CLI output which buildpack was used
* Hit both /index.html and /index.php
* What happens this time? Why?
* Use cf ssh to see what files the buildpack has added, and how they differ from the last push

## Cleaning Up

Delete your apps to free up space.

Lab 5 Availability

In this lab, we will use purposefully crash app instances and see how Cloud Foundry works to maintain availability.

## Pushing a Crashable App

What happens when an app crashes?

* Change to the 05-resilience/imperfect-app directory and push the crashable app.
* Note the random URL for your app, and visit it in a browser.
* Click the ‘crash’ link
* Use cf app imperfect-app to see the state of your app.

Can you see it in the “crashed” state before Cloud Foundry restarts it?

## Access Your App Amid Failures

* Scale your app to 3 instances.

#### Checking Your Work

* Use cf apps to ensure you have 3 instances requested.
* Visit your app and click the ‘crash’ link.
* Refresh the page, and Cloud Foundry will send your request to one of the healthy instances.

### Can you crash instances quicker than Cloud Foundry can restart them?

If you have watch available on your system, use it to watch app instances restart.

$ watch cf apps # Watch app instances restart

If not, you can re-run cf app multiple times.

$ cf app imperfect-app

state since cpu memory disk

#0 running 2015-11-02 0.0% 25.3M of 32M 66.9M of 128M

#1 down 2015-11-02 0.0% 0 of 0 0 of 0

#2 down 2015-11-02 0.0% 0 of 0 0 of 0

* What happens if you make a request whilst they are all down?

Lab 6 Debugging

In this lab, you will continue to master debugging techniques for your applications.

## Push a buggy app

A buggy app is included in the 06-debugging/debug-app directory with a manifest.

* Push 06-debugging/debug-app to PWS
* Note the URL

### Access the App

* Open a browser and access the app.
* Observe a 500 error.
* How does this impact app health? How does an error on one page compare to a crashing app from the last lab?

## Check out the Logs

* Use cf logs to access the recent logs and debug the issue.

If you do not specify the --recent flag to cf logs, it will start tailing logs from that point onwards.

#### Checking Your Work

You should see something similar to this in the logs:

... [App/0] ERR ... - RuntimeError - I am a bug, fix me:

### Fix it

This app can be fixed by setting an environment variable and restarting it.

* Use cf help to find out how to set an environment variable for your app called FIXED with a value of true
* Restart your app, and visit it in a browser to check that the bug is fixed

## Debugging with Events

Now the app offers other links that allow you to terminate the app’s process, use up all the app’s RAM, or fill the disk.

* Click “crash”
* Observe the output of cf events and cf logs for your app
* Click “exhaust memory”
* Observe the output of cf events and cf logs for your app
* How do the two compare? What help does Cloud Foundry give you in determining the cause of failure?

#### Checking Your Work

You should see something like the following:

... index: 0, reason: CRASHED, exit\_description: 2 error(s) ...

* Click the ‘exhaust disk’ link, and check cf logs and cf events.
* What happens? Is this what you expected?

## App instrumentation

We will use New Relic as a example. The process involves creating an instance of the New Relic service, binding it to our app, adding a license key, then re-pushing. Services are covered in another section of this course, so don’t worry if you don’t understand these commands.

* Create a New Relic service instance: cf create-service newrelic standard newrelic
* Tell Cloud Foundry that debug-app should use your New Relic service instance: cf bind-service debug-app newrelic
* Find the New Relic license key in the output of cf env debug-app
* In newrelic.yml, replace YOUR-LICENSE-KEY with the value you just found
* Re-push the app

### Viewing the Dashboard

New Relic has a dashboard. You can find the URL by looking at the service details.

* Visit the URL reported by cf service newrelic

It takes some time for the data from your app to be visible in the New Relic dashboard.

## SSH access

Don’t forget that you can use cf ssh to look at the current filesystem of your app.

## Beyond the Class

* Setup [Skylight](https://www.skylight.io/) for app
* Setup [Opbeat](https://opbeat.com/) for app
* Learn about [CF Logging and Metrics](http://www.cfsummit.com/sites/cfs2015/files/pages/files/cfsummit15_king.pdf)
* Send app logs to [Papertrail](https://papertrailapp.com/)

$ cf cups logdrain -l syslog://YOUR-PAPERTRAIL-LOG-DESTINATION

$ cf bind-service debug-app logdrain

# Check your Papertrail Events, no need to restart the app

Lab 7 Stateful Services

In this lab, you will create an instance of a Redis service and use it with an app.

## Creating a Service Instance

First, you need to create an instance of the service.

* Use cf marketplace to view the details of the Redis service
* Use cf service to create an instance of the 30mb plan

#### Checking Your Work

You can view your service instances:

$ cf services

name service plan bound apps last operation

redis rediscloud 30mb create succeeded

## Binding Service Instances

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

* Push 07-shared-state/stateful-app with the --no-start flag
* Use cf bind-service to bind your service instance to your app
* Start your app so that it can pick up the environment variables

#### Checking Your Work

You can view the details of your services, and see which apps your service is bound to:

cf services

If you hit the /env endpoint of your app, or run the command cf env stateful-app, you will see the VCAP\_SERVICES environment variable that Cloud Foundry provides to your app. When a service is bound to your app, the service’s details appear in this variable.

## Demonstrating Persistence

By storing state in a service we can restart apps without losing any data.

* Visit the app in a browser
* Observe the number of requests this app instance has served, along with the overall total number of requests all app instances have served
* Restart the app and visit it in a browser again
* Observe that the total number of requests is still stored in Redis, even though the app was restarted

## Exploring the Service Instance Lifecycle

Service instances can be shared by many apps, and can live longer than the apps that use them.

* Increase the number of instances of your app
* Visit your app to see the difference between different app instances serving requests and the overall hit count in their shared Redis service instance

Now we have many app instances Cloud Foundry is load-balancing between them, but they’re all sharing the same Redis instance.

* What will happen when we unbind the app?
* Stop the app, and use cf unbind-service to unbind the service from the app
* Rebind the app, and start it
* Observe that the Redis instance still holds the same state

Unbinding did not delete data in Redis. It did remove the credentials that our app was using to connect to Redis, but new ones were issued when we bound the app again.

* What will happen when we delete the service instance?
* Can you use *cf delete-service redis*?
* Do whatever is necessary to delete the service instance, and then create it again
* When you start your app and visit it in a browser, you’ll see that this is a new, clean Redis instance with no existing state

Lab 8 Routes

In this exercise, you will use route mapping to perform a zero-downtime upgrade of an app.

## Create a new route

We need to create a route for the app we are going to deploy. This route will be used for all versions of the app, so should stay the same even when the app is updated.

* Use cf create-route to a create a new route for your app, making sure the domain is cfapps.io

### Push v1.0

* Change to the 08-domains-routes/v1.0 directory and push the v1.0 app
* Does the push work? If not, why not? How can you fix this?
* Use cf help to figure out how to map your route above to this app
* Verify your app is accessible by accessing the route you created

#### Checking Your Work

You should be able to access your app on the route created above. You should also be able to see the route/url by looking at the app details:

cf app

### Push v1.1

Imagine that a new version of the app has been developed. We want to push it and check that it works.

* Push version 1.1 of the app from 08-domains-routes/v1.1 and assign it a random route
* Do not map your main route yet
* Validate you can access the app by accessing the random route

If this app was being deployed automatically, this is the point that smoke tests would be run against the new version. If the new version works on the random route, we can proceed to load-balance production traffic to it.

* Use the CF cli to map traffic hitting the main route you created to v1.1

#### Checking Your Work

Check your work by accessing the main route multiple times. You should observe traffic being balanced by Cloud Foundry across both instances. You can also see the routes/urls by looking at the app details:

cf app

### Cutting over

We pushed the new version, smoke tested it on its own route, and then load balanced production traffic to both the new version and the old version. Now it is time to stop traffic going to the old version of the app.

* Use cf help to figure out how to stop sending traffic to v1.0
* Could you leave the v1.0 app running in case you need to roll-back?

#### Checking Your Work

Check your work by accessing the main route multiple times. You should observe traffic only going to v1.1. You can also see the routes/urls by looking at the app details:

cf app

Congratulations! You successfully updated your running application to a new version with no downtime for users.

## Beyond the Class

* Set up a custom [SSL certificate](http://www.selfsignedcertificate.com/)
* Use [feature flags](https://docs.cloudfoundry.org/adminguide/listing-feature-flags.html) instead of ENV vars for new features
* Delete all routes that are no longer used

$ cf delete-orphaned-routes