

LFD232: CLOUD FOUNDRY FOR DEVELOPERS

- V_1.03.2019 -



INSTALLING THE CLI

Installing the CLI

- The cf CLI can be installed by following directions on the cloudfoundry.org docs site: <https://docs.cloudfoundry.org/cf-cli/install-go-cli.html>
- Be sure you have the latest version of the cf CLI before continuing with the course.

Checking your work

Check the version to ensure the cf CLI is installed correctly by running `cf --version`

You should see output similar to:

```
cf version 6.40.1+85d04488a.2018-10-31
```



Localization

The cf CLI also supports localization. If you would like to use it in a language other than English, follow the directions here: <https://docs.cloudfoundry.org/cf-cli/getting-started.html#i18n> .



GETTING STARTED W/ THE CF CLI

In this exercise, you will use the cf CLI to access a Cloud Foundry instance.

Getting Help

The cf CLI is an extensive, self-documenting tool. You can access the CLI by opening a terminal/command prompt. You can access help for the most commonly used commands by typing:

```
cf help
```

You can see help for all available commands with:

```
cf help -a
```

The `cf help -a` output can be a bit overwhelming. On linux based system, piping the output to `grep`, searching for keywords, can be a useful tool. For example, to find

```
$ cf help -a | grep service
restage                Recreate the app's executable
                        artifact using the latest pushed app files and the latest environment (variables, service
                        bindings, buildpack, stack, etc.)
services               List all service instances in
the target space
service               Show service instance info
create-service         Create a service instance
update-service         Update a service instance
delete-service         Delete a service instance
```

Another method is to copy the output of `cf help -a` to a file and search the file for keywords.

If you want help on a specific command, you can always use:

```
cf <command> --help (Example: cf login --help)
```

Throughout the class, it will be up to you to use `cf help` to determine the correct commands to complete the task at hand.

Logging in

The first step to interacting with a Cloud Foundry instance (target) is to log in. The convention is to log into an API endpoint: `api.<cloudfoundry-system-domain>`.

As an example, if you want to log into a Cloud Foundry instance located at `cf.example.com` you would log into `api.cf.example.com`.

- Use `cf login --help` and login to your Cloud Foundry instance. The instructor will provide you with the system domain for this class.

Targeting Orgs and Spaces

When you log in, you will be asked to target an org and space. If you only have access to one, you will automatically be targeted. However, you can also change the org and space you are target to.

- Use `cf orgs` to see orgs you have access to.
- Use `cf spaces` to see the spaces you have access to.

You can use `cf target` to change the org and space you are currently targeting (it is not necessary to do this now) or see your current target.

- Use `cf target` to verify you are targeted to your development space.

Checking Your Work

If you run `cf target` you should see something like:

```
api endpoint:  https://api.***
api version:   2.125.0
user:          you@example.com
org:           cf-training
space:         development
```



CLI to API version

Because each Cloud Foundry target could potentially be running different versions, it is useful to ensure your CLI version works with your current target. For this, we can use a command called `cf curl`.

All Cloud Foundry targets expose an info endpoint. We will use this endpoint to ensure our CLI version is supported by that target.

- Run `cf curl /v2/info`

The output should include something similar to:

```
"min_cli_version": "6.22.0"
```

Be sure your cli version is greater than or equal to the `min_cli_version` of your current target. If not, be sure to update your CLI before proceeding with the course.

Cloud Foundry API

The `cf curl` command above is making REST requests to the Cloud Foundry API: <https://apidocs.cloudfoundry.org>. `cf curl` behaves just like `curl` (a Unix utility) in that they both make REST requests to an HTTP endpoint. The difference is that `cf curl` uses your current target and authentication information (if applicable).

`curl` is not a Cloud Foundry utility, but a generic unix command line utility for making web requests. More information is available here:


<https://www.linux.com/blog/5-basic-curl-command-examples-you-can-run-your-linux-server>

The API exposes REST endpoints. The CLI uses these endpoints to expose cohesive commands. A single command like `cf target` will often result in requests to more than one API endpoint.

The API and the CLI are separate but related projects and therefore have decoupled lifecycles. Features available in the API may not be available in the CLI. Similarly, you won't see CLI commands directly in the API.

Request/Response Diagnostics

The CLI can output request and response information when running any command by appending the `verbose` flag `-v`. Run `cf target -v` and you will see output similar to:



```
REQUEST: [2018-11-13T12:49:33-07:00]
GET /v2/info HTTP/1.1
Host: api.***
Accept: application/json
User-Agent: cf/6.40.1+85d04488a.2018-10-31 (go1.10.3; amd64 darwin)

RESPONSE: [2018-11-13T12:49:33-07:00]
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 623
Content-Type: application/json; charset=utf-8
Date: Tue, 13 Nov 2018 19:49:33 GMT
Server: nginx
X-Content-Type-Options: nosniff
X-Vcap-Request-Id: 8955c2e3-dc42-4616-4f78-3fe5a4d558dd::8a18e961-978a-4e91-944b-89a59a19f141
{
  "api_version": "2.125.0",
  "app_ssh_endpoint": "ssh.***",
  "app_ssh_host_key_fingerprint": "e7:13:4e:32:ee:39:62:xx:54:41:d7:f7:8b:b2:a7:xx",
  "app_ssh_oauth_client": "ssh-proxy",
  "authorization_endpoint": "https://login.***",
  "build": "",
  "doppler_logging_endpoint": "wss://doppler.***",
  "min_cli_version": "6.22.0",
  "min_recommended_cli_version": "latest",
  "name": "",
  "osbapi_version": "2.14",
  "routing_endpoint": "https://api.***",
  "support": "https://support.***",
  "token_endpoint": "https://uaa.***",
  "version": 0
}

...
```

The CLI is outputting information about the requests sent to Cloud Foundry and the responses received from Cloud Foundry. This can be very useful for debugging and implementing automation. Note that the output of this feature will vary by `cf` command as requests and responses differ.

CF_TRACE

Like many things in Cloud Foundry, the cf CLI is configurable as well. Configuration is modified via environment variables or by using `cf config`. We will show you both ways to enable the tracing of requests and responses between the CLI and Cloud Foundry.

CF_TRACE is a particularly useful feature if you suspect communication issues between the CLI and the a Cloud Foundry target or incompatibilities with the cf API.

CF_TRACE with Environment Variables

- Enable CF_TRACE
 - on MacOS/Linux: `export CF_TRACE=true`
 - on Windows: `set CF_TRACE=true`
- Re-run `cf target`. What happens?
The details of the requests and responses to Cloud Foundry are displayed.
- To disable tracing:
 - on MacOS/Linux: `export CF_TRACE=false`
 - on Windows: `set CF_TRACE=`

TIP: If you want to run tracing for a single command, you can prepend `CF_TRACE=true` before any cf command. Example `CF_TRACE=true cf <command>`.

CF_TRACE with `cf config`

You can also set the default tracing using `cf config --trace`. You can use `cf config --help` on how to do this. Please note this sets the `default` value. the CF_TRACE environment variable takes precedence.

Localization

The cf CLI translates terminal output into various languages. You can use cf config to set the output language. A list of available language variants is available here:

<https://docs.cloudfoundry.org/cf-cli/getting-started.html#i18n> .



PUSHING APPS

In this exercise, you will push an app and add some data to it.

As you progress through the course, be sure to follow the directions closely. In most cases, the exercises will build on each other. For this reason, **DO NOT** delete applications unless instructed to do so.

Pushing Your First App

A [rest-data-service app](#) has been provided to you as a jar file. This is a simple Spring Boot app, but the process is the same regardless of the language.

- Download the [rest-data-service app](#) jar file
- Use `cf push --help` to determine how to push your app. Be sure to do the following:
 - Name your app `roster`.
 - Push the jar file by providing the '-p' flag pointing to the jar file on your laptop
 - Create one instance
 - Allocate 750M of memory
 - Use the `java_buildpack` buildpack
 - Use `--random-route` to avoid route collisions

Checking Your Work

If everything is successful, you should see output similar to:

```
Waiting for app to start...
  Uploaded droplet (74.7M)
  Uploading complete
  Cell 2c755f28-7f03-4a12-871c-7b1e60306330 stopping instance 446978dc-
8b91-4269-a2f6-8a824c97617f

name:          roster
requested state: started
routes:        roster-silly-duiker.***
last uploaded: Tue 13 Nov 13:31:42 MST 2018
stack:         cflinuxfs2
```



```

buildpacks:      java_buildpack

type:            web
instances:       1/1
memory usage:    750M
start command:   JAVA_OPTS="-agentpath:$PWD/.java-
buildpack/open_jdk_jre/bin/jvmskill-1.16.0_RELEASE=printHeapHistogram=1 -
Djava.io.tmpdir=$TMPDIR
                -Djava.ext.dirs=$PWD/.java-buildpack/container_security_provider:$PWD/.java-
buildpack/open_jdk_jre/lib/ext
                -Djava.security.properties=$PWD/.java-buildpack/java_security/java.security
$JAVA_OPTS" &&
                CALCULATED_MEMORY=$( $PWD/.java-
buildpack/open_jdk_jre/bin/java-buildpack-memory-calculator-
3.13.0_RELEASE
                -totMemory=$MEMORY_LIMIT -loadedClasses=17460 -
poolType=metaspace -stackThreads=250 -vmOptions="$JAVA_OPTS") && echo JVM
Memory
                Configuration: $CALCULATED_MEMORY &&
JAVA_OPTS="$JAVA_OPTS $CALCULATED_MEMORY" && MALLOC_ARENA_MAX=2
SERVER_PORT=$PORT eval exec
                $PWD/.java-buildpack/open_jdk_jre/bin/java $JAVA_OPTS
-cp $PWD/. org.springframework.boot.loader.JarLauncher
state      since      cpu      memory      disk
details
#0  running  2018-11-13T20:32:13Z  0.0%  83.4M of 750M  156.8M of 1G

```

Notice the `urls` value. Your app should now be available here.

NOTE: `--random-route` is a function of the CLI. It appends two random words to the name of your app as a dev tool to help avoid route collisions. **DO NOT** use this in production.

Add Some Data to Your App

Your app simply stores information about people in key value pairs. It is a REST based application and therefore you can use `curl` or postman to put data in.

NOTE: Both `curl` and postman are general purpose tools not tied to or provided by Cloud Foundry. `curl` is a linux command line utility and may be available in some Windows terminal solutions. Postman is a graphical tool available on any platform.

Using Curl

- Edit the `curl` command below with your information:

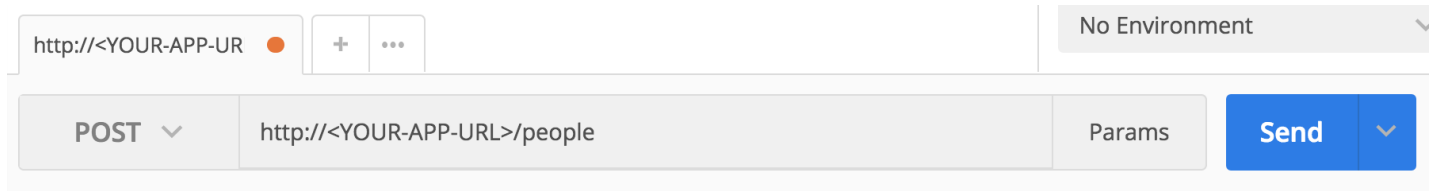
```
curl -H "Content-Type: application/json" -X POST -d '{"firstName":"foo",  
"lastName": "bar"}' http://<YOUR-APP-URL>/people
```

Using Postman

- Be sure you have installed postman: <https://www.getpostman.com/>

On the **Builder** page, you will set up your request.

- Be sure **POST** is selected. This is the REST request type.
- Enter your app's url with `/people` appended in the request URL box.



The image shows the Postman Builder interface. At the top, there's a URL bar with 'http://<YOUR-APP-URL>' and a dropdown menu showing 'No Environment'. Below this, the request method is set to 'POST' and the URL is 'http://<YOUR-APP-URL>/people'. There are buttons for 'Params', 'Send', and a dropdown arrow.

- On the **Headers** tab, set a header called **Content-Type** with a value **application/json**.

Authorization	Headers (1)	Body	Pre-request Script	Tests
Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> Content-Type	application/json			

- On the **Body** tab, select **raw**. In the textbox, add some data. For example:

```
{"firstName":"foo", "lastName": "bar"}
```

● form-data ● x-www-form-urlencoded ● raw ● binary JSON (application/json) ▼

```
1 {"firstName": "foo", "lastName": "bar"} |
```

- Click **Send** to execute the request.

Checking Your Work

At this point, you should see data returned by your app. You can access the **/people** endpoint via a browser, curl or postman (with a **GET** request).

Via curl - `curl http://<YOUR-APP-URL>/people`

You should see something similar to:

```
{
  "_embedded" : {
    "people" : [ {
      "uuid" : "a486b8f5-82be-4999-8beb-1ea59ef82a20",
      "firstName" : "Steve",
      "lastName" : "Greenberg",
      "_links" : {
        "self" : {
          "href" : "http://roster-appreciative-warthog.cfapps.io/people/a486b8f5-82be-4999-8beb-1ea59ef82a20"
        },
        "person" : {
          "href" : "http://roster-appreciative-warthog.cfapps.io/people/a486b8f5-82be-4999-8beb-1ea59ef82a20"
        },
        "personStatuses" : {
          "href" : "http://roster-appreciative-warthog.cfapps.io/people/a486b8f5-82be-4999-8beb-1ea59ef82a20/personStatuses"
        }
      }
    } ]
  },
  "_links" : {
    "self" : {
      "href" : "http://roster-appreciative-warthog.cfapps.io/people"
    },
    "profile" : {
      "href" : "http://roster-appreciative-warthog.cfapps.io/profile/people"
    }
  }
}
```

```
},  
"page" : {  
  "size" : 20,  
  "totalElements" : 1,  
  "totalPages" : 1,  
  "number" : 0  
}  
}
```

NOTE: If you use chrome, you might want to install a JSON formatter like JSONView:

<https://chrome.google.com/webstore/detail/jsonview/chklaanhfefbnpoihckbnefhakgolnmc>



SCALING APPS

In this exercise you will see the difference between horizontal and vertical scale, using the rest-data-service app.

Vertical Scale

To demonstrate vertical scale, you will add memory to your application instance.

- Use `cf scale` to allocate 1G of total memory to your application.
- What happens?
- Use `cf app` to show the status of the instance you have running

Checking Your Work

If everything is successful, you should see output similar to:

```
Showing health and status for app roster in org cloudfoundry-training /
space cff as example_user@example.com...
```

```
name:          roster
requested state: started
routes:        roster-silly-duiker.***
last uploaded: Tue 13 Nov 13:31:42 MST 2018
stack:         cflinuxfs2
buildpacks:    java_buildpack
```

```
type:          web
instances:     1/1
memory usage:  1024M
state         since                cpu    memory    disk
details
#0  running   2018-11-13T22:03:25Z  1.0%   201.3M of 1G  156.8M of 1G
```

- Scale your app back down to 750M of memory

Horizontal Scale

Horizontal scale adds instances.

- Use `cf scale` to increase your instance count to 2.
- What happens?

Checking Your Work

If everything is successful, you should see output similar to:

```
Showing health and status for app roster in org cloudfoundry-training /
space cff as example_user@example.com...

name:          roster
requested state: started
routes:        roster-silly-duiker.***
last uploaded: Tue 13 Nov 13:31:42 MST 2018
stack:         cflinuxfs2
buildpacks:    java_buildpack

type:          web
instances:     2/2
memory usage:  750M
state         since                cpu      memory      disk
details
#0  running   2018-11-13T22:10:15Z  0.4%     200.2M of 750M  156.8M of 1G
#1  running   2018-11-13T22:12:38Z  144.8%   163.1M of 750M  156.8M of 1G
```

Questions

- **One of the scaling methods is very fast. Which one? What do you know about CF that makes this fast?**
Horizontal scaling is fast because CF has already created and cached the droplet. It can simply create a new container and copy the droplet to it.
- **One form of scaling involves downtime. Which one? Why?**
Vertical scaling involves downtime because new containers need to be created with the new resource allocations.
- **What should you do to avoid downtime while still being able to handle variations in load?**
Horizontal scaling should be used in production to avoid downtime. The best practice is to *right*

size applications using vertical scaling in development. Then, in production use horizontal scaling to vary capacity.



APPLICATION LOGS & EVENTS

In this exercise, we find out information about the apps you have already deployed via the `cf` CLI in Cloud Foundry.

Tailing

Cloud Foundry allows developers and operators to stream logs or view recent logs.

- Use `cf logs --help` to determine how to view recent logs for your app
- Use `cf logs` command to stream logs for your app
- Access your app multiple times.

Note: You can exit streaming with `CTRL-C` or you can continue to stream and start a new terminal window.

Questions

- **What is the difference between recent logs and tailing logs?**
Recent shows past logs stored in a temporary buffer. The buffer is bound by the number of messages and time (configured by operators). Buffers are first in first out and old messages are discarded.
Tailing shows the stream of logs in near real time. You will only see messages generated after you initiated the stream.
- **What do you see in the application logs?**
You should see logs for all of your application instances as well as log messages generated by CF components used to fulfill the request. The components generating log messages include routers (messages tagged with `RTR`) and cells (messages tagged with `CELL`).

Information on log formats and contents is available here:

<https://docs.cloudfoundry.org/devguide/deploy-apps/streaming-logs.html#types> .

Events

Cloud Foundry keeps a limited record of application change activity.

- Use `cf help` find the command that lists the event history for your app

Questions

- **What do you see in the event history?**

You should see events in your applications lifecycle. For example, the `audit.app.create` event is fired when you first pushed your application. The `audit.app.update` event is fired when you update your app (for example during scaling).



APPLICATION RESILIENCY

Cloud Foundry contains 4 levels of recoverability. In this exercise, you will demonstrate the application recovery capability of the platform.

Instance crashing

The application contains an endpoint that allows you to programmatically kill an instance. You can access this endpoint in your browser at `https://<YOUR-APP-URL>/kill`.

Obligatory disclaimer: *This endpoint is for demonstration purposes only. You should never implement such an endpoint.*

Before you kill an instance, it is helpful to `see` `cloudfoundry diagnose` and fix the issue.

Witnessing the Carnage

Tailing the logs

- Use `cf` to tail logs for your application in a terminal window

Using `watch`

`watch` is a useful Unix/Bash command which refreshes static commands. *You might have to install this.*

MAC users can install `watch` via homebrew `brew install watch`: <http://brew.sh>

If you have `watch`, open a terminal and run `watch cf app roster` to monitor the state of `roster` app.

Open another terminal and run `watch cf events roster` to monitor the `roster` app events.

Killing an instance

- Ensure you have two instances of your app running
- Once you are tailing the logs and/or using watch, access your app's `kill` endpoint:
`https://<YOUR-APP-URL>/kill`

Because `kill` performs a `System.exit(1)` it kills the app instance you are accessing before it has a chance to respond. Consequently, a `502 Bad Gateway` error is returned, denoting your request was received but you didn't get a response.

Questions

- **What did you see? How long did it take?**

In the logs, you should have seen messages showing the app crashed. You should then see messages showing a new instance starting up.

Cloud Foundry typically takes less than a few seconds to detect a failure and create a new instance. Because of this, you might not see the `crash` using `watch cf app` (remember watch is essentially polling, re-running `cf app` every 2 seconds by default). However, you should see evidence in the logs as well as in `cf events`.

The time to start a new instance is bound by the application itself. Cloud Foundry will begin the process quickly, but as a developer you should also try to ensure your applications have fast start times.

- **What should you ensure about your apps when running in Cloud Foundry?**

You should always ensure you have *at least* two running instances of your app. That way if an app instance crashes, your users can be served by the other instance.



CREATING & BINDING SERVICES

Experiencing Ephemeral Data

In this exercise, we demonstrate the need to move data out of the application tier into an attached service. Currently, our application stores data in memory.

- Scale your application down to 1 instance
- Invoke your application's `/people` endpoint & insert data if necessary
- Restart your application using `cf restart`.

Questions

- **Access your application again. What happened to the data?**
The roster application stored the data in memory because it didn't detect a database. The next section discusses how to attach/bind a database to an application.

Marketplace Services

In this exercise, we will move the state from our application to an external database by creating an instance of the database and binding it to our application.

Marketplace

The marketplace is where you can see services available for provisioning.

- Run `cf marketplace` to see a list of services

Creating Service Instances

Once you know what services are available, you can create an instance according to a plan.

- Use `cf create-service` to create a new instance of a mysql database
 - You can give the service instance any name you wish
 - The service needs to be a mysql-compatible database (i.e. mysql or mariadb)

- Make sure you create the free plan of the mysql service

Checking Your Work

At this point you should have a service instance. Run `cf services` to see service instances.

Binding Services

We need to make our service instance available to our application. We do this by binding it to our app.

- Use `cf help` to determine how to bind your service instance to your application.

Hint: Other than `cf help`, there are two commands you need to run. The output of the correct bind command is going to give you a hint on another command you must run so that your app knows about the service instance.

Checking Your Work

If you re-run `cf services` you should see your instance now bound to your application.

The app has an endpoint that displays some details including the services bound to your app.

- Access your app in a browser and observe the mysql service:
`http://<YOUR-APP>/app-details`

What is Happening?

There is a service broker creating our database instance. This broker is also providing credentials to Cloud Foundry when we bind it to our app. So how does our app know about the credentials?

- Run `cf env roster` and look at the output.

The `VCAP_SERVICES` environment variable contains our application credentials which our application is parsing. Cloud Foundry is satisfying two of the 12 factor principles: *storing config in the environment* and *treating backing sources as attached resources*.

Persistent Data

Use curl or postman to add data again.

- Edit the `curl` command below with your information:

```
curl -H "Content-Type: application/json" -X POST -d '{"firstName":"<some-key>","lastName":"<some-value>"}' http://<YOUR-APP-URL>/people
```



- Restart your app
- Access the data using a browser or curl and verify the data is still there.



ENVIRONMENT VARIABLES & APP MANIFESTS

Setting Environment Variables

In this exercise, you will set some environment variables in your app then check that they are present.

Use the `cf help` command to set 3 environment variables that are named `ROSTER_A`, `ROSTER_B`, `ROSTER_C`.

You can choose any value for each of the variables.

Checking your work

Check that the variables are set by invoking the `/app-details` endpoint of the running app.

Creating an Application Manifest

In this exercise, you will construct an application manifest to allow you to more easily push apps.

Manifest Documentation

Manifests can be used to tell `cf push` what to do with applications. There are many options.

- Review the manifest documentation at:
<https://docs.cloudfoundry.org/devguide/deploy-apps/manifest.html>

Manifests are written in yaml. It is often helpful to be able to validate yaml when building a manifest to identify issues. Yaml can be easily validated with languages like ruby or online with tools like <http://www.yamllint.com/>.

Pre-requisites

Since manifests store the application configuration, we can delete the deployed application from Cloud Foundry and redeploy it from scratch using the manifest configuration

- Locate & run the command to unbind your application from the mysql service

- Locate & run the command to delete your app from Cloud Foundry
- Run `cf apps` to check that your app has gone

Building a manifest

Create a manifest for your roster application that contains the following:

- App name
- Use of random route
- 1 instance of the app
- 750M of memory
- the java buildpack
- the mysql service binding
- the path to the jar file
- the same 3 environment variables as above

Once you have done this, validate you can redeploy your application by simply typing `cf push`.

Checking your work

Use `cf app` to verify the resources of the app match what you put in your manifest. Check that the variables are set by invoking the `/app-details` endpoint of the running app. Try adding data to the `/people` endpoint of the app as you did before and check that it persists across restarts.



LOG DRAINS

In this exercise, you will export logs from Cloud Foundry via log drain.

Log Drains

Due to the ephemeral nature of CF, it is often necessary to drain logs to an external service.

For this exercise, we will leverage the free trial from Papertrail, a log aggregator as a service.

- Sign up for an account here: <https://papertrailapp.com/>
- After sign up, a startup page displays. Select “Add your first system”, which directs you to a page with a URL at the top. Copy this URL including the port (note the port is obscured in the image below):

Setup Logging

Your logs will go to **logs6.papertrailapp.com:** [redacted] and appear in [Events](#).

I'd like to aggregate **system/OS logs** from **Linux/Unix**

1 Run the install script

```
wget -q0 - --header="X-Papertrail-Token: dgwYTHjxt900SSsIZsu" \
https://papertrailapp.com/destinations/5876151/setup.sh | sudo bash
```

This script will make the syslog daemon send logs to Papertrail (and ask for your

- Use `cf help` to create a user provided service for this log drain endpoint. Be sure to use the `log-drain` variant of the UPSI and use the `syslog-tls://` protocol before your papertrail URL.
- Bind your user provided service instance to the roster app.
- You can optionally bind the same upsi to your web-ui.

- After you have made the necessary configuration changes, you will need to access your app to generate some log output and complete the connection to papertrail.

The process of creating a user provided service and draining logs is the same regardless of the log aggregator (ELK, Splunk, etc).

NOTE: `cf cups` creates a User-Provided Service Instance. We came across these earlier in the 'services' slides.

Checking Your Work

Access your app & generate some log output. You should be able to see the application log output in the Papertrail web UI. Verify that this the same as the output from `cf logs`



BLUE/GREEN DEPLOYMENT

In this exercise, we look at updating running applications with blue/green deployments to minimize downtime.

Cloud Foundry allows users to manipulate routes via the cf CLI. We can use this to our advantage to update running applications with zero downtime.

Roster app

DATABASE CONNECTIONS: If you are using a mysql database service with a small number of connections (typically 5 or less), you will want to unbind the database from your roster app before proceeding with this exercise. Be sure to also comment out the binding in your manifest. Otherwise, you will likely exhaust the database connections available to your app in this exercise. This is for TRAINING PURPOSES ONLY!

The roster application allows you to set an environment variable to assist in illustrating a blue/green deployment without changing code.

- Set an environment variable on your deployed app called `APP_VERSION`. Set the value of this environment variable to `blue`. This will represent your v1.0.0 app.
- Access your app in a browser to verify this setting at `http://<YOUR-APP>/app-details`. You should see the `appVersion` field as `blue`.
- Note this URL (denoted `<YOUR-APP>` above). This will be the **main route** we end up moving to the 'green' version of your app.

New version

- Edit the manifest you created earlier:
 - adding a dependency on the syslog drain service created during the previous exercise
 - adding an `APP_VERSION` environment variable with the value 'green'
- Use this manifest to deploy a new version of your app with a different name (hint: command-line arguments override manifest parameters)
- Access this app's `/app-details` endpoint via the new route and verify the new version is displayed in the `appVersion` field.

Mapping the route

- Use `cf` to map the 'green' version of your app to the main route.
- Access the `/app-details` endpoint multiple times in a browser and verify you are being routed to both blue and green.

Un-mapping the route

- Use `cf` to un-map the main route from the 'blue' version of your app.
- Access the `/app-details` endpoint multiple times in a browser and verify you are being only being routed to green.

Congrats! You just updated a running app with zero downtime using Cloud Foundry.

Cleaning up

- Use `cf` to delete the 'blue' version of the app
- Use `cf` to rename the 'green' version of the app to the name the blue version had
- You should now see one app (named roster) with two routes mapped to it
- Identify the route that **isn't** the main route, then un-map it from the app and delete that route.
- Modify the `APP_VERSION` environment variable to `blue`
- Restart the app in order for the running app code to pick up the app name & variable changes

REMEMBER: If you unbound the database from your app before this exercise, rebind it now and uncomment out the binding in the manifest.



SETTING UP APP MONITORING WITH NEW RELIC

In this exercise, we will create a New Relic service and bind it to our Roster app.

New Relic is an application performance monitoring tool.

Prerequisites

Prior to starting this exercise you should have a Roster app deployed from a previous exercise.

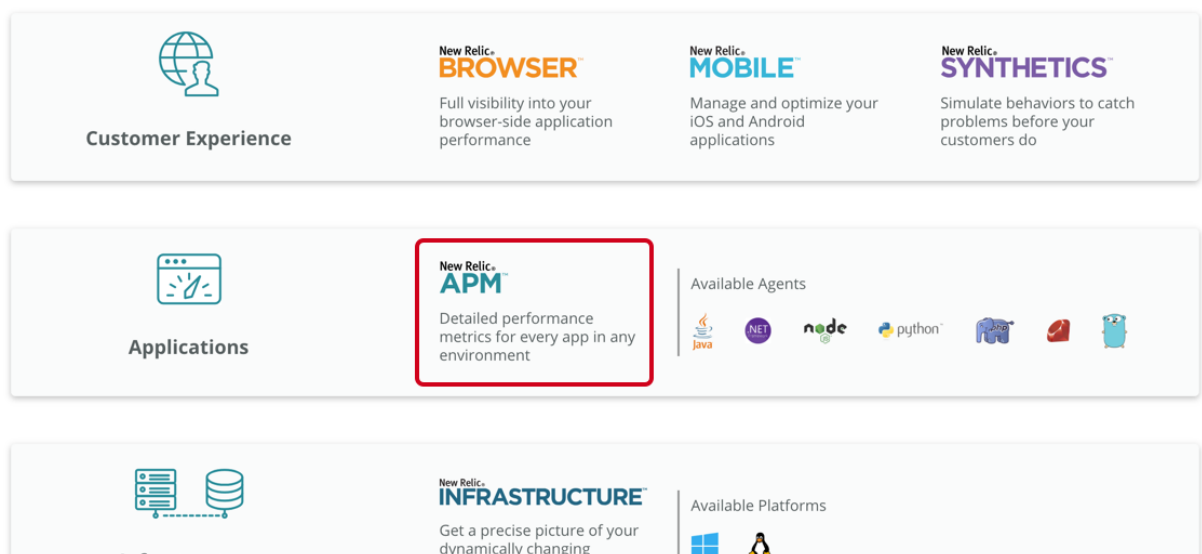
Creating a New Relic account

NOTE: You will not be following the instructions on the New Relic site, but instead following instructions below. Be sure to follow our instructions carefully.

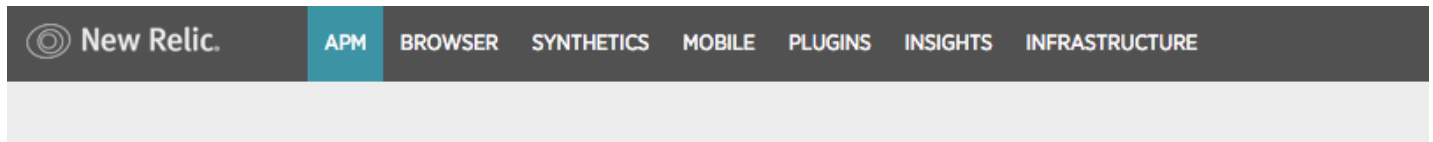
- Go to <https://newrelic.com/signup> and create a free trial account.
- Once logged in choose APM



Choose your product



- Select Java



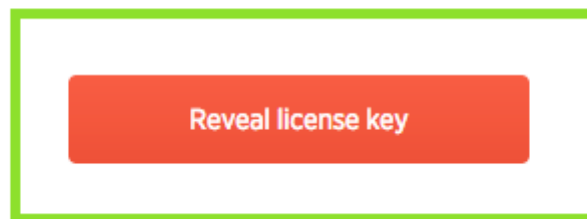
Get started with New Relic

Select a web agent to install.



- Scroll down until you see “Get your license key”. Click “Reveal your license key” and copy the value. You will need it in a minute.

1 Get your license key [\(Java agent docs\)](#)



- DO NOT follow any other instructions on the New Relic site at this time

Creating and binding the New Relic service

Using the CF CLI, create a UPSI with a name ending with `newrelic`. The UPSI should have a single key called `licenseKey` with a value equal to your license key copied above.

Once the service has been created, use the CF CLI to bind it to your Roster app.

Checking app metrics

Now that the service has been bound to your app, go back to New Relic. Scroll down until you see “See data in 5 minutes”. Click `Listen for my application`.

5 See data in 5 minutes

In a few minutes, your application will send data to New Relic and you'll be able to start monitoring your application's performance. You will also be automatically upgraded to New Relic PRO for a limited time.

You won't see any data in your dashboard until restart has completed.

Listen for my application

▶ Deployed and still not seeing your data?

The screen will automatically refresh and you will be taken to the dashboard once data is flowing. This typically takes about 5 minutes but can take up to 20.

- Click on your roster app name. What metrics do you see?



PUSHING THE WEB-UI

We are going to push a Ruby **web user interface** app that will make RESTful calls to the roster app that we have already deployed. IN order to do this, the web app is going to need to know the URL of the roster app.

User-provided service instance

Back in the ‘services’ slides, we met the user-provided service instance (UPSI). The CF CLI refers to these as ‘user-provided services’. Use `cf help` to find the command to create one:

- named `rest-backend`
- with a single credential named `url` for which the value is the route of your roster app

Pushing the Web UI app

A web-ui written in Ruby is available for download as a zip file. The zip contains an application manifest.

- Download and unzip the [web-ui.zip](#)
- Push the web-ui app using the supplied manifest, ensuring that the application **does not start**. That is, after the `cf push` completes, running a `cf apps` shows the state of `web-ui` as “stopped”.

Hint: A Ruby application is pushed as a directory of files, so you’ll need to unzip the application and provide `cf push` the path to the directory that you unzipped it to

Questions

- **Look at the included manifest. Where is it getting the buildpack from?**
Notice the buildpack is pulled from a github repo.

Binding the UPSI to web-ui

Bind the `rest-backend` UPSI to the web-ui app, then start the web-ui app.

Checking your work

Access your web-ui app via its route in a browser and check that it is loading data from the roster app by clicking the `View People` button.

Exploring buildpack behaviour

What was the command that Cloud Foundry used to start your app? How can you find this? What Cloud Foundry component will have set this start command?

Let's take a look at how specifying a different buildpack changes the behaviour of our application.

Create a second running copy of the web-ui app named `web-ui-static` using the manifest, but this time overriding the buildpack with the `staticfile-buildpack`:

<https://github.com/cloudfoundry/staticfile-buildpack> .

Access `web-ui-static` on its route in a browser. If you have deployed correctly the app will clearly state that it has been deployed using the Staticfile buildpack.

We'll explore the ways in which the two apps are different (even though they use the same code) in a later exercise.



CONTAINER SSH

This exercise uses SSH to inspect the differences in the containers of the running apps, based on using two different buildpacks.

Container Comparison

- Use `cf help` to find the command to access the app instance container of the `web-ui` app via SSH. Take a look at the file system and find the application code files. Can you find any information about what command was used to start the app?

The `/var/vcap/staging_info.yml` contains the start command.

- Now use the same technique to connect to the `web-ui-static` app instance and look at the file system. How does the file system differ from that of the `web-ui` container? What does the start command tell you about the running container?

The `app` directory contains the ruby code in `web-ui`, while the `web-ui-static`, the `app` directory contains `nginx` related files. The start command indicates a Ruby application start in the `web-ui` case. In `web-ui-static` the start command shows `nginx` startup.

Cleaning up

Use `cf` to delete the `web-ui-static` version of the app



DEPLOYING A ROUTE SERVICE FOR RATE LIMITING

In this exercise we will deploy a Route Service app that limits access rates by IP address.

Before you start

From the previous exercises ensure you have:

- a running instance of the Web UI app, named `web-ui`, which has a random route
- a running instance of the `roster`, with a service named `rest-backend` bound to the `web-ui` app

Creating the Route Service

- Deploy the provided [rate-limit-route-service.jar](#). At this point in the class you should know how to do this without any guidance.
- Create a user provided service instance for your route service. Be sure you create an instance of type **route service**.
- Bind the route service to the route for your web-ui.

Checking your work

- If you access your web-ui more than 10 times in less than a minute (through a browser or curl), you should see a message about “too many requests”.
- If you wait a minute and try again, you should be allowed to access the app again.

Questions

- **How could you provide rate limiting for your roster app?**
Bind the route service instance to the route for your roster app.
- **Did you have to restart/restage your web-ui? Why/why not?**
You should not have to restart your app as the route service is bound to the route, not the app.

- **Would you have to make any changes to the binding in the event you were blue/green deploying web-ui?**

No. Because the route is not changing, just the mappings, the route service bindings do not change.



PUSHING A DOCKER APP

In this exercise you will push a Docker app to Cloud Foundry.

Pushing the Docker app

A worker-style application has been developed, which uses the RESTful data service `roster` to insert status data for people on the roster. Once running, the application inserts a status & date for one of the people on the roster every 10 seconds.

The application code has been encased in a Docker image, named `engineerbetter/worker-image` and pushed to Docker Hub:

<https://hub.docker.com/r/engineerbetter/worker-image/> .

- Use the `cf` command to push the docker image to Cloud Foundry
 - The running application is a worker app, so it should have *no route*
 - Be sure the app does not try to start
- Bind the newly created application to your existing `rest-backend` UPSI
- Start the app

Checking your work

The Roster application has a `/people_status` endpoint that you can query to check data has been inserted.



SERVICE KEYS

Imagine that a human needed to connect to a service instance created via the Cloud Foundry CLI. How would we get credentials to use when connecting to the service instance? We *could* re-use credentials provided to an app bound to the service instance, but this would be an anti-pattern - we shouldn't pretend to be our app.

Cloud Foundry allows users to generate Service Keys for exactly this use case.

Prerequisites

Before starting this exercise you should have:

- a `roster` app running which is bound to a MySQL (or MariaDB) service
- a Docker `worker` app running

Getting credentials

- Use `cf help` to create a service key for the MySQL service. Give it a descriptive name.
- Next, use `cf help` to extract the credentials out of the service key.
- Take note of the `username`, `password`, `name`, and `hostname` fields.

Connecting to the database

We will use the `mysql` CLI client to connect to our service instance. Not every student will have this client installed on their machine, or even the ability to install new software. Luckily the Docker image for the `worker` app has the MySQL CLI installed on it, so we can `cf ssh` into one of the app instances and use the CLI available there.

- Use `cf ssh` to connect to the Docker app.

From here you can connect to the database using the credentials you noted down earlier.

- Run the following, replacing credentials as appropriate. You will be prompted for the password.

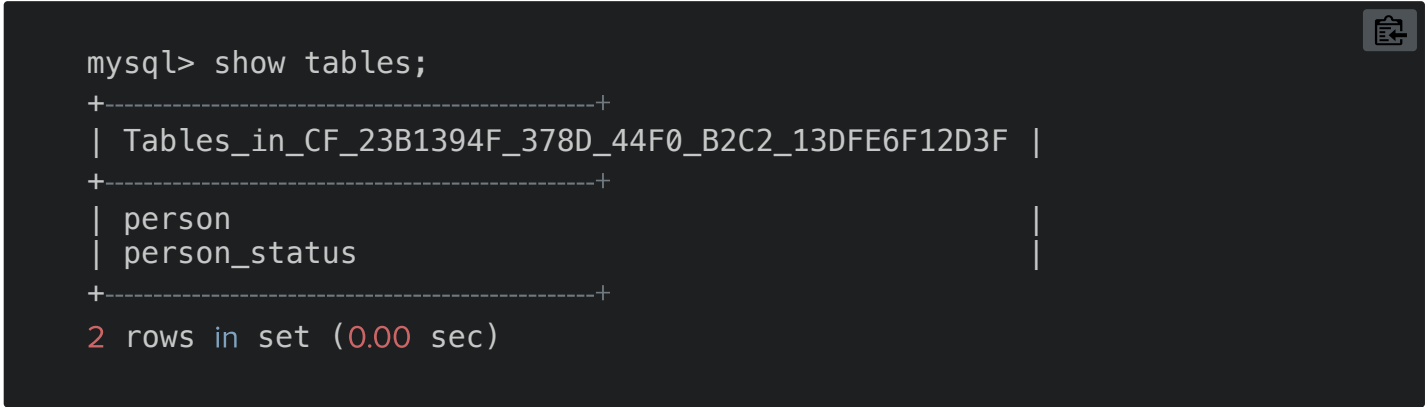
```
mysql -u $username -h $hostname -D $name -p
```

You should now be connected to the database and able to run queries.

- Try listing the tables to start:

```
SHOW TABLES;
```

You should see something like:



```
mysql> show tables;
+-----+
| Tables_in_CF_23B1394F_378D_44F0_B2C2_13DFE6F12D3F |
+-----+
| person                                             |
| person_status                                     |
+-----+
2 rows in set (0.00 sec)
```

Try running some other queries.



DEPLOYING UAA AS A CF APP

In this exercise, we will deploy UAA as a standalone OAuth2 server on Cloud Foundry.

UAA is deployed as a component of CF but is also available to be deployed independently.

Deploying UAA

A [compiled version of the UAA](#), and a [corresponding deployment manifest](#) have been made available.

- Think of a unique hostname for your UAA application such that it will not clash with other users' apps (e.g. `<first-initial><last-name>-uaa`). Use `cf` to create the route. Creating the route prior to pushing the app, ensures the route is available. The command fails if the route is already in use.
- Deploy the UAA war file using the supplied manifest and providing your route using the `--var` flag during the push. The key name is `route` and the value is the route or your uaa application.

Checking your work

Access your running UAA app on its route. If the UAA app is deployed correctly you will find you are redirected to a login page.

Hint: if you test this via `curl` you will require the `-L` argument in order to follow redirects



DEPLOYING A ROUTE SERVICE FOR AUTHENTICATION

The Web UI app has a URL path `/secure` that requires an OAuth2 token for security.

In this exercise we will deploy a Route Service app that initiates the OAuth2 workflow and performs the role of the App Client, so that the Web UI secure endpoint is accessible.

Before you start

From the previous exercises ensure you have:

- a running instance of the UAA application, with a unique name
- a running instance of the Web UI app, named `web-ui`, which has a random route
- a running instance of the `roster`, with a service named `rest-backend` bound to the `web-ui` app

Access the `web-ui` home page & check the page displays `Browsing anonymously`

Questions

- Try accessing the `/secure` endpoint of your `web-ui` app. What response do you get?
The browser displays `Not authorized` message.

Binding Web UI to UAA

Our WebUI app needs to know about UAA so that it can validate a user token. The web ui will authenticate itself to UAA (using `client_name` and `client_secret` below) then validate tokens presented by users.

- Create a UPSI named `uaa-tokens` with the following credentials:
 - `url` = the URL of your UAA application (with https protocol)
 - `client_name` = `oauth_showcase_authorization_code`
 - `client_secret` = `secret`
- Bind the `uaa-tokens` UPSI to your `web-ui` app

Creating the Route Service

- Download and unzip the [uaa-guard-proxy](#)
- Create a route for your `uaa-guard-proxy`. You will need to create a unique hostname (for example: first-initial-lastname-uaa-guard-proxy).
- Deploy your guard proxy using the included manifest. Be sure to supply the `uaa-guard-proxy-route` and `uaa-route` variables using the values for your apps.
- Create a UPSI called `authz` of type **route service**, with the URL of your `uaa-guard-proxy` app.
- Bind the route service to the route of your `web-ui` app.

Checking your work

You should open a new Incognito (or private browsing) window for each test because the `uaa-guard-proxy` app will only cause your browser to go through the OAuth2 workflow once per session.

- In your browser, access the homepage of your `web-ui` app. You should be redirected to the authentication page of your UAA application.
- Enter the username `marissa` and password `koala`, then click Enter.
- You should be shown an Authorization page, click Authorize.
- You should now be shown the homepage of your `web-ui` app.
- Check that the message at the top of the page says 'Logged in as marissa'.

Questions

- Access the `/secure` endpoint again. What do you see this time?
The browser displays `You are secure! Logged in as marissa`



PREVENTING CASCADING FAILURE

In this exercise, you will see an example of graceful degradation in a Cloud Foundry app.

Prerequisite

Before you begin you should have a roster app running with a bound database service. You should also have a web-ui running with a UPSI linking it to the roster app. If you don't have this set up you can follow the instructions for the "Pushing the Web-UI" exercise.

Stopping the backend

Using the CF CLI, stop your roster app. This will simulate an issue with the backend.

Comparing non-graceful and graceful failure

Navigate to `$YOUR-WEB-UI-ROUTE/cascade` in a browser. You will see a page with two buttons.

- First click **View People with failure**. What do you see? Is this a good user experience?
A cryptic error page with information only relevant to the technologist is display. This is a poor user experience since the information is meaningless to a user.
- Now go back to the cascade page and click the other button, **View People without failure**. How is this page better than the previous one?
A user friendly error message displays, describing the problem in common language. A **back** button is also provided for usability.



NON-DEFAULT DOMAINS

In this exercise we investigate non-default domains.

NOTE: This exercise requires some external configuration for the non-default domain to be available. If your instructor has not provided you with a domain then you will not be able to complete this exercise.

Background

Cloud Foundry comes configured with some domains by default. Other domains can be added by admins. Often the default domains are used by developers for development then a non-default domain is used for production apps. For this exercise suppose that you have tested your web-ui fully and a decision has been made to promote it to production.

Normally there would be a different space for production apps but for now we will focus on the domain of the existing app.

Non-default domains

First we need to ensure the non-default domain your instructor has provided you with is configured in Cloud Foundry. Use `cf help` to list the domains available and check that the non-default one is listed.

Now use `cf help` to map a route to your web-ui using the non-default domain. Your app should be accessible via `$app_name.$non_default_domain`.

Check that you can access your app on this different domain and that it still functions as it did on the default domain.