Lab 5 - From Zero to Pushing Your First Application

Setup

1. If you are on a network that requires a proxy, set the proxy as an environment variable:

   ```
   export HTTP_PROXY={HTTP_PROXY}
   ```

2. Set the API target for the CLI:

   ```
   $ cf api {CF-API-URL} --skip-ssl-validation
   ```

3. Your username will be first initial + lastname. Password will be provided in the class.
4. Follow the steps above for login.
5. Create your own space and target that space:

```
$ cf create-space <your_username>
$ cf target -s <your_username>
```

Build and Push!

1. Change to the *Spring Music* sample application directory:

   ```
   $ cd $COURSE_HOME/day_01/session_02/lab_05/spring-music
   ```

2. If you are on a network that is behind an HTTP proxy, you will need to provide some proxy settings to the Gradle build tool so it can download dependencies. In your user home directory, create a directory named `.gradle` and a file named `gradle.properties` in that directory (e.g. `~/.gradle/gradle.properties` on Linux/OSX or `%USERPROFILE%\.gradle\gradle.properties` on Windows). Add lines like these to that file, with the address and credentials for your HTTP proxy:

   ```
   systemProp.http.proxyHost=www.example.com
   systemProp.http.proxyPort=8080
   systemProp.http.proxyUser=userid
   systemProp.http.proxyPassword=password

   systemProp.https.proxyHost=www.example.com
   systemProp.https.proxyPort=8080
   systemProp.https.proxyUser=userid
   systemProp.https.proxyPassword=password
   ```

3. Using the Gradle Wrapper, build and package the application:

   ```
   $ ./gradlew assemble
   ```

   The Gradle Wrapper will automatically download the appropriate version of Gradle for this project along with all of *Spring Music*'s dependencies. This may take a few moments.

4. Push the application!

   ```
   $ cf push
   ```

   You should see output similar to the following listing. Take a look at the listing callouts for a play-by-play of what’s happening:

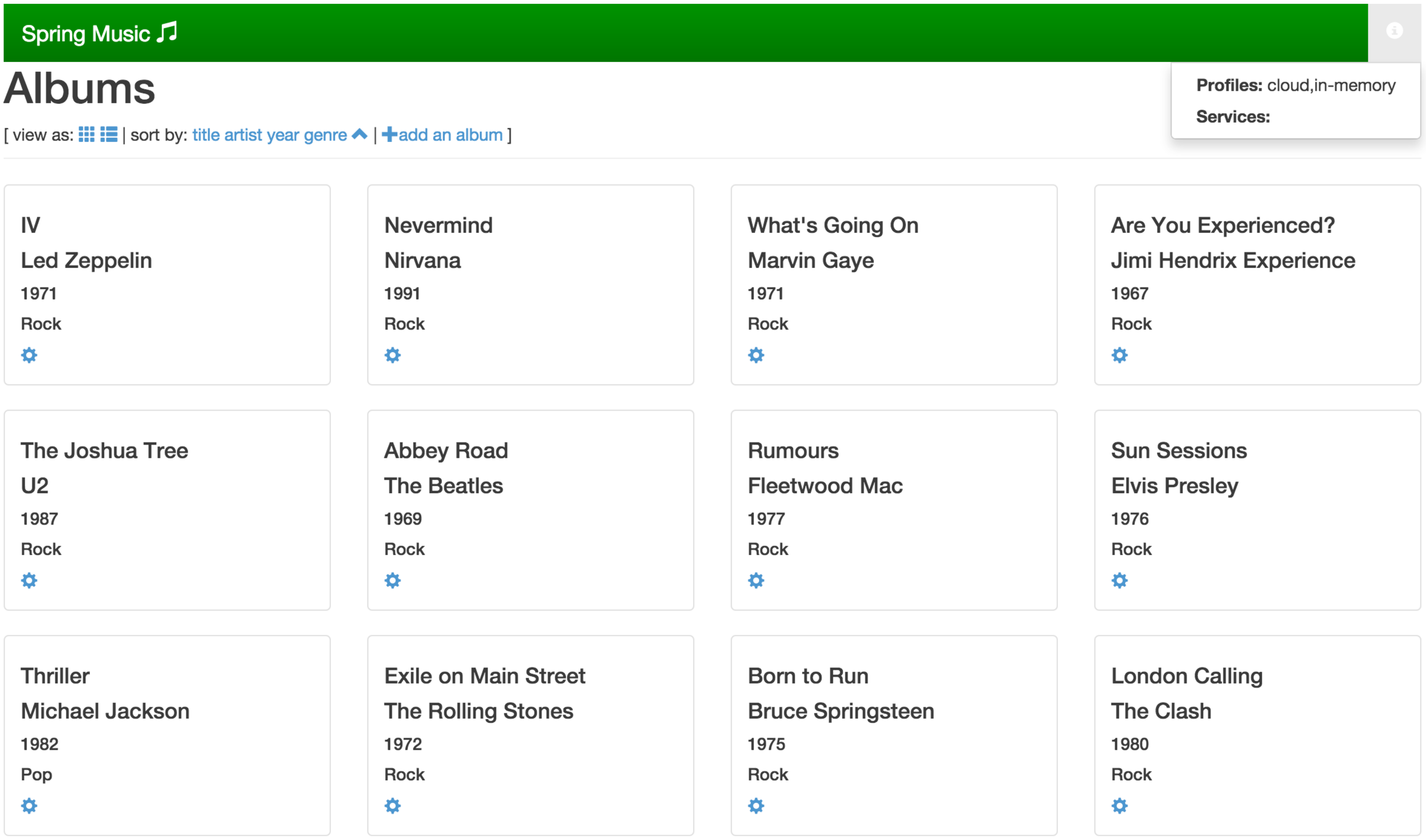
```
Using manifest file
/Users/pivotal/mstine/2015/OReilly/CloudNativeArchitectureClass/day_01/session_02/lab_05/spring-
music/manifest.yml (1)

Creating app spring-music in org oreilly-class / space instructor as mstine@pivotal.io...
OK (2)

Creating route spring-music-hippest-shaman.cfapps.io...
OK (3)

Binding spring-music-hippest-shaman.cfapps.io to spring-music...
OK (4)

Uploading spring-music... (5)
Uploading app files from:
/Users/pivotal/mstine/2015/OReilly/CloudNativeArchitectureClass/day_01/session_02/lab_05/spring-
music/build/libs/spring-music.war
Uploading 569.7K, 90 files
Done uploading
OK

Starting app spring-music in org oreilly-class / space instructor as mstine@pivotal.io... (6)
-----> Downloaded app package (21M)
-----> Java Buildpack Version: v2.6.1 |  https://github.com/cloudfoundry/java-buildpack.git#2d92e70
-----> Downloading Open Jdk JRE 1.8.0_31 from
https://download.run.pivotal.io/openjdk/lucid/x86_64/openjdk-1.8.0_31.tar.gz (1.3s)
       Expanding Open Jdk JRE to .java-buildpack/open_jdk_jre (1.2s) (7)
-----> Downloading Spring Auto Reconfiguration 1.7.0_RELEASE from https://download.run.pivotal.io/auto-
reconfiguration/auto-reconfiguration-1.7.0_RELEASE.jar (0.1s)
-----> Downloading Tomcat Instance 8.0.18 from https://download.run.pivotal.io/tomcat/tomcat-
8.0.18.tar.gz (0.4s)
       Expanding Tomcat to .java-buildpack/tomcat (0.1s) (8)
-----> Downloading Tomcat Lifecycle Support 2.4.0_RELEASE from https://download.run.pivotal.io/tomcat-
lifecycle-support/tomcat-lifecycle-support-2.4.0_RELEASE.jar (0.0s)
-----> Downloading Tomcat Logging Support 2.4.0_RELEASE from https://download.run.pivotal.io/tomcat-
```

```
logging-support/tomcat-logging-support-2.4.0_RELEASE.jar (0.0s)
-----> Downloading Tomcat Access Logging Support 2.4.0_RELEASE from
https://download.run.pivotal.io/tomcat-access-logging-support/tomcat-access-logging-support-
2.4.0_RELEASE.jar (0.0s)
-----> Uploading droplet (66M) (9)

0 of 1 instances running, 1 starting
1 of 1 instances running

App started


OK

App spring-music was started using this command `JAVA_HOME=$PWD/.java-buildpack/open_jdk_jre
JAVA_OPTS="-Djava.io.tmpdir=$TMPDIR -XX:OnOutOfMemoryError=$PWD/.java-
buildpack/open_jdk_jre/bin/killjava.sh -Xmx382293K -Xms382293K -XX:MaxMetaspaceSize=64M -
XX:MetaspaceSize=64M -Xss995K -Daccess.logging.enabled=false -Dhttp.port=$PORT" $PWD/.java-
buildpack/tomcat/bin/catalina.sh run` (10)

Showing health and status for app spring-music in org oreilly-class / space instructor as
mstine@pivotal.io... (11)
OK

requested state: started
instances: 1/1
usage: 512M x 1 instances
urls: spring-music-hippest-shaman.cfapps.io
last uploaded: Fri Feb 13 15:43:08 UTC 2015

     state     since                    cpu    memory          disk
#0   running   2015-02-13 09:43:55 AM   0.0%   394.5M of 512M   131.1M of 1G
```

1. The CLI is using a manifest to provide necessary configuration details such as application name, memory to be allocated, and path to the application artifact. Take a look at `manifest.yml` to see how.

2. In most cases, the CLI indicates each Cloud Foundry API call as it happens. In this case, the CLI has created an application record for *Spring Music* in your assigned space.
3. All HTTP/HTTPS requests to applications will flow through Cloud Foundry's front-end router called (Go)Router (http://docs.cloudfoundry.org/concepts/architecture/router.html). Here the CLI is creating a route with random word tokens inserted (again, see `manifest.yml` for a hint!) to prevent route collisions across the default `cfapps.io` domain.
4. Now the CLI is *binding* the created route to the application. Routes can actually be bound to multiple applications to support techniques such as blue-green deployments (http://www.mattstine.com/2013/07/10/blue-green-deployments-on-cloudfoundry).
5. The CLI finally uploads the application bits to PWS. Notice that it's uploading *90 files*! This is because Cloud Foundry actually explodes a ZIP artifact before uploading it for caching purposes.
6. Now we begin the staging process. The Java Buildpack (https://github.com/cloudfoundry/java-buildpack) is responsible for assembling the runtime components necessary to run the application.
7. Here we see the version of the JRE that has been chosen and installed.
8. And here we see the version of Tomcat that has been chosen and installed.
9. The complete package of your application and all of its necessary runtime components is called a *droplet*. Here the droplet is being uploaded to PWS's internal blobstore so that it can be easily copied to one or more *Droplet Execution Agents (DEA's) (http://docs.cloudfoundry.org/concepts/architecture/execution-agent.html)* for execution.
10. The CLI tells you exactly what command and argument set was used to start your application.
11. Finally the CLI reports the current status of your application's health. You can get the same output at any time by typing `cf app spring-music`.

5. Visit the application in your browser by hitting the route that was generated by the CLI:


Spring Music

Albums

[view as: | sort by: title artist year genre | add an album]

Profiles: cloud,in-memory

Services:

IV
Led Zeppelin
1971
Rock

Nevermind
Nirvana
1991
Rock

What's Going On
Marvin Gaye
1971
Rock

Are You Experienced?
Jimi Hendrix Experience
1967
Rock

The Joshua Tree
U2
1987
Rock

Abbey Road
The Beatles
1969
Rock

Rumours
Fleetwood Mac
1977
Rock

Sun Sessions
Elvis Presley
1976
Rock

Thriller
Michael Jackson
1982
Pop

Exile on Main Street
The Rolling Stones
1972
Rock

Born to Run
Bruce Springsteen
1975
Rock

London Calling
The Clash
1980
Rock

Be sure to click on the ``information icon'' in the top right-hand corner of the UI. This gives you important information about the state of the currently running *Spring Music* instance, including what Spring Profiles are turned on and what Cloud Foundry services are bound. It will become important in the next lab!