PIVOTAL CLOUD FOUNDRY

LAB ASSIGNMENTS

Lab 3:

Logging and Troubleshooting

Steps:

In this section you will push an application to Cloud Foundry that has a fatal flaw. By accessing logs you will attempt to diagnose the issue, repair the application, and re-push.

Warning

If you have trouble viewing logs and events using cf commands, view the logs in the Apps Manager console instead. This is usually due to proxy or firewall restrictions on the websocket port 4443. The typical errors are:

```
Error dialing loggregator server: dial tcp 1.2.3.4:4443: connection timed out.
websocket.Dial wss://loggregator.run.pivotal.io:4443/tail/?... connection timed
out.
```

Debugging Push Problems - IDE Users

IDE (STS /Eclipse) :

1. If you have not done so already, import the cf-spring-mvc-trouble project into your IDE. If you have any other projects open, you might like to close them all so you don't work with the wrong project by mistake.
2. Deploy the cf-spring-mvc-trouble application on Cloud Foundry
    a. Be sure to alter the value of subdomain to produce a unique route.
    b. Watch the console / log output carefully during the deployment. Note that the application appears to be having difficulty starting. Run the following commands to see if you can diagnose the issue (replace "[app]" with your application's name):

        i. cf events [app]

        ii. cf logs [app] –-recent

        iii.cf logs [app]

3. Once you have a general feeling for the problem, return to the IDE and examine this Java class:

gopivotal.cf.workshop.Config - an easy way to do this is to use Ctrl-T or Cmd-T to popup the Open Type dialog and enter Config - pick the Config class in the current project. Find the lines of code marked

with the "BAD CODE" comment. This is a Spring application and the @PostConstruct method is automatically called on startup. Do you see why it is failing?

4. Comment out the code that is causing the problem. (Java comments are lines strating with the "//" characters.) Save your work (File / Save), return to the "Pivotal Cloud Foundry" tab, and use "Update and Restart" to repush and restart the application.

Note

When using an IDE (Eclipse/STS/IntelliJ) there is no need to 'build' the application. Saving causes an automatic build on any modified files.

5. The application should now start successfully. Run the cf events and cf logs commands again and note the differences in the logged output.

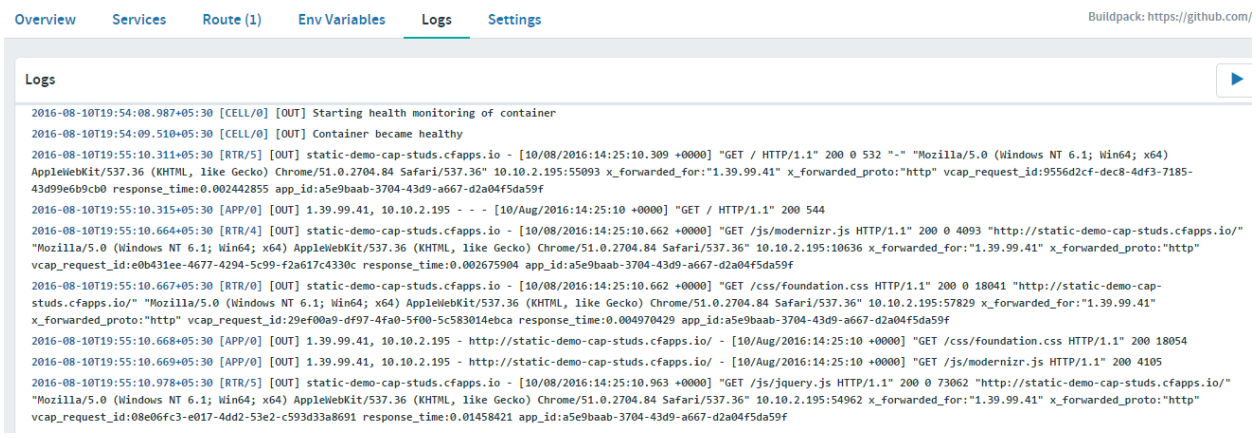
CLI:

Try the following command in CLI

1. cf events <<app-name>>
2. cf logs <<app-name>>
3. cf logs <<app-name>> --recent

Pivotal Web Console:

In pivotal web console you can see the log details under App→logs link as shown below:



Debugging Push Problems - Using cf

In this section we will do the lab using the cf utility.

1. In your terminal or command window, make sure that **cf-spring-mvc-trouble** is the current directory.
2. Deploy the cf-spring-mvc-trouble application on Cloud Foundry. There is no manifest so you will need to specify the -i, -m, -n and -p flags. The application you need to deploy is pre-built/cf-spring-mvc-trouble-0.0.1-SNAPSHOT.war.
    a. Be sure to specify the value of subdomain (-n option) to produce a unique route.
    b. Watch the console / log output carefully during the deployment. Note that the application appears to be having difficulty starting. Run the following commands to see if you can diagnose the issue (replace "[app]" with your application's name):
        i. cf events [app]
        ii. cf logs [app] –-recent
        iii.cf logs [app]

3. Once you have a general feeling for the problem, examine the Java code in the following file: gopivotal.cf.workshop.Config.java. An easy way to do this is to enter
    • more src/main/java/com/gopivotal/cf/workshop/Config.java (Linux/MacOS)
    • more src\main\java\com\gopivotal\cf\workshop\Config.java (Windows)

    in your command/terminal window to view the file. Use the tab key auto-complete feature to avoid typing the path in full. Or feel free to use the editor of your choice.

    Note the lines of code marked with the "BAD CODE" comment:

```
@PostConstruct
public void consumeAllMemory() {
// TODO: LOOK HERE! LOOK HERE! LOOK HERE!
// BAD CODE! BAD CODE! BAD CODE!
// The code you see here will consume all available memory allocated
// to the application instance. Comment out this code and re-push
List<Double> list = new ArrayList<Double>();
while (true) {
list.add(Math.random());
}
// End of bad code.
}
```

    This program has been deliberately setup to fail - the consumeAllMemory method sits in an infinite loop until all memory is consumed and the application fails to start.

4. Let's assume you contacted a developer colleague and got them to remove the offending code. The new application is pre-built/fixed-cf-spring-mvc-trouble-0.0.1-SNAPSHOT.war. Rerun your cf push command but specify this file instead (using the -p option).
5. The application should now start successfully. Run the cf events and cf logs commands again and note the differences in the logged output.

Conclusion:

In this lab we learnt how to view log reports in CLI and web console.

Congratulations, you have completed this exercise!

Prerequisites:

1. A Cloud Foundry account (either on your company's CF installation or on Pivotal Web Services)
2. The CLoud Foundry CLI installed

Lab 4:

Using a Buildpack

Steps:

Pushing a Static HTML Application

In this section you will push an existing static HTML application to Cloud Foundry. CF does not have a

built-in buildpack for static HTML sites, so you will need to specify an external buildpack that utilizes Nginx.

1. Locate the cf-static-demo folder on your file system.
2. Edit the manifest.yml. Give your application a unique host, and specify https://github.com/cloudfoundry-community/staticfile-buildpack.git as the buildpack.
3. Use CLI cf push to push the application to your Cloud Foundry environment.
4. When the push is complete, use a browser to access your running application. If it is not running, attempt to debug using cf logs <app>--recent and other techniques.

Note
Some administrators may choose to disallow external buildpacks, in which case you cannot specify an alternate selection. If this is the case in your environment, you may simply use Pivotal Web Services for this exercise.

Conclusion:

1. Push a simple Java/Spring application to Cloud Foundry
2. Observe and scale a running application.

Congratulations, you have completed this exercise!

Prerequisites:

1. A Cloud Foundry account (either on your company's CF installation or on Pivotal Web Services)
2. Either the CLoud Foundry CLI or an IDE installed (with Cloud Foundry support)
3. ***Most importantly:*** GitHub account. Unlike other labs, a github account is mandatory for this lab.

Lab 5:

Customizing a Buildpack

Fork the Java buildpack to make a minor change.

Steps:

Forking the Java Buildpack

In this section you will fork the Java buildpack, make a minor change to the desired Java version, then use your modified buildpack to push an application to Cloud Foundry, and observe the results.

1. Push the spring-mvc-demo application to Cloud Foundry (if needed)
    a. Use either the CLI or Eclipse/STS, whichever you find most convenient. If you use **Eclipse/STS** it would be best to save a manifest file for the later steps.
    b. Observe the console output during the push process. Note the version of Java being used. Our goal in this lab will be to specify a different Java version.
2. Fork the Java Buildpack.
    a. Using a browser, go to https://github.com/cloudfoundry/java-buildpack (if you are not signed into GitHub, do so now).
    b. Click the "**Fork**" button to create a fork of this repository under your own GitHub account. (You may clone a copy of this repository locally, though it is not necessary for the steps here)
3. Alter the Java version
    a. While still in the browser viewing your forked copy of the Java buildpack, drill-down to **config/open_jdk_jre.yml**. Click the edit button to modify this file in place.
    b. Find the line (near line 20) that indicates the Java version: version: 1.8.0_+
    c. Alter this line to deliberately specify an older Java version: version: 1.8.0_31
    d. Save your work using the 'Commit' button.
4. Re-push the application using the altered Buildpack
    a. This step is performed differently depending on whether you have an existing manifest file or not. Assuming you do:

i. Alter the manifest file. Add a line to indicate you new buildpack (change "yourGitId" and "java-buildpack" as needed to match). Be sure your indentation is correct:
buildpack: https://github.com/[yourGitId]/[java-buildpack]

ii. Re-push the application. If using Eclipse ensure that it is picking up the values you specified in the manifest.

b. If you did not have a manifest file, and you used the CLI, repush using the -b option to specify the buildpack (alter "yourGitId" and "java-buildpack" as needed):
-b https://github.com/[yourGitId]/[java-buildpack]

c. Observe the console output during the push process. Note the version of Java being used. It should be the version you specified above.

Conclusion:

We learnt the simple Customize buildpack creation.

Congratulations, you have completed this exercise!