Instructor Notes

Eitan Suez – esuez@pivotal.io – Version 0.1, 2017

Table of Contents

1. General Overview

This course was designed to be strictly about PCF, and attempts as much as possible not to delve into any coding activities. There are some topics where it's unavoidable, but even there, you'll see that the labs provide the code for the student, often even in pre-compiled fashion, and the extent of 'coding' doesn't go beyond reading and analyzing the source code.

The goal is to learn PCF from the point of view of the developer, who will need to know how to deploy applications to PCF. Administration topics such as installation, upgrades, bosh, etc.. are not not core to this narrative, and not included. The one administrative feature discussed is

Application Security Groups, as in this particular case, it is pertinent for a developer to know about this feature.

2. Duration

This course is designed to be delivered over a period of three days at a leisurely pace. There is plenty of time in the course for both students to read through some of the user guide reference pointed to from the labs, and for an instructor to discuss and teach particular features of PCF which are not specifically covered in the presentation slides.

2.1. Daily Breakdown

2.1.1. Day 1

Suggested topics to cover on day 1:

- An introduction to Pivotal Cloud Foundry
- Logging, Scale and HA
- Services
- Manifest

What stands out about day one is that some time has to be allotted for attendees to bootstrap into the course: there's the general course intro and logistical information, introductions, and then there's setting up the student environment, which primarily entails setting up a cloudfoundry account. The cf cli installation and cf login steps are performed as part of the first lab.

Depending on the experience level of the class, you may finish day 1 a little early.

2.1.2. Day 2

Topics:

- Application Security Groups
- Log Drain
- Blue-Green
- Microservices
- Application Autoscaler
- Application Performance Monitor

- Metrics

I typically finish a little early, perhaps around 4:00 PM on day 2.

2.1.3. Day 3

Topics:

- Buildpack
- Service Broker
- Continuous Delivery
- Route Service

Buildpack and Service Broker labs tend to go together: they both speak to means of extending the platform. There's no particular reason that route services has to be at the end. I personally would prefer to see it group with other service-related topics: services, and log-draining. They all have in common the create-user-provided-service command, which using a different flag: -p, -l, and -r. The continuous delivery topic is a sort of part 2 to the microservices talk. I personally like the fact that the two talks are spread out, each on a separate day.

I often finish day 3 around 4:00 PM as well.

3. Setup

The main concern here is providing students with a PCF account. Technically an instructor could stand up a PCF instance in AWS or GCP or elsewhere, and create accounts for each student. I've done that before. But for the majority of deliveries, I've used Pivotal Web Services. Students setup a free-trial account, and that works fine.

Sometimes we encounter situations where a student has in the past already created such an account, where their free trial has expired. They can still log in, but are prompted for payment information. In these situations, instructors can setup a space for the student in question in the Pivotal-Education-Students organization in PWS. If you do not have access to this organization, contact Pivotal Education.

Here is a shell script that automates the creation of a space for a student. And here's the corresponding cleanup script for cleaning up the space (and removing the user from the org) after class is over.

Additional setup is required for the jenkins and service broker labs. These are explained respectively in each of these modules' notes below.

4. Modules

4.1. Module: An Introduction to Cloud Foundry

The presentation here is designed to be a very high level overview, and attempts to explain the motivation for a PaaS. Somewhere in this presentation I introduce Onsi's famous haiku to try to explain the philosophy of a PaaS.

The lab consists of the necessary bootstrapping:

- downloading the cf CLI
- logging in to the cloudfoundry account

And we follow that with deploying four different 'hello world' style applications, to make the point that PCF is a polyglot platform.

Students learn a lot of the basics very quickly, and start asking themselves all the right questions to begin to gain a deeper understanding of PCF: what happens during a push. It's inevitable that the conversation will lead in this directly, which leads nicely into the next module.

The types of mistakes I've seen students make is to attempt to push an application without first changing to the directory that contains the code in question. It's not a bad idea to let students know before the lab begins: by default what gets uploaded are the entire contents of the current working directory! Another item I stress is that the first argument to cf push is the application na me, not a reference to a file or artifact that is to be uploaded.

During the subsequent review, I like to point out that, for java applications, you must pass in the path (-p) flag to the push command. In the review I also make it a point to cover what happens with respect to the setup of a route for an application, which is an opportunity to explain the random-route flag. I also like to point out that just overriding the host attribute is often sufficient.

TODO: Modify the lab instructions to push a java application too, not just dynamic languages (ruby, node, python, php).

I also like, in a review, to cover other basic cf commands that the students weren't exposed to: apps, app, stop, start, restart, etc..

4.2. Module: Logging, Scale and HA

The presentation here is difficult, because the instructor is needs to explain a great deal about the internal architecture of Cloudfoundry, perhaps a little too early in the course.

TODO: consider moving some aspects of this module to later in the course. The DAwCF course used to do this nicely and delve into the architecture at the end.

I like to try to draw a diagram of the cloudfoundry architecture as I describe each of the components. Sometimes, to explain auctions, I'll demo a few minutes' of Onsi's presentation Re-envisioning the elastic runtime (https://www.youtube.com/watch?v=1OkmVTFhfLY) (minute 14 to 16:30, roughly).

This presentation is long, and will often spill into the afternoon.

When I discuss the cloud controller and the cloudfoundry api, i like to show how one can see the http requests and responses by setting CF_TRACE.

The lab is a refreshing change of pace; it's also a natural progression from the previous lab: learning some new cf commands: logs, events, scale.

It ties into the High Availability portion of the presentation by having the students kill their app and see it restart. I like to use the command line "watch" utility (brew install watch) to auto-refresh the status of an app so I can "see" a container get restarted (`watch cf app <appname>`).

In the review, I look for opportunities to show students things that perhaps were not covered. I like to show them `cf ssh` , for example.

4.3. Module: Services

The presentation deck here is brief and to the point. Explain the idea of services and distinction between managed and user-provided services. I often like to make reference to how one used to manually obtain database credentials, for example.

I also like to state how, although the main purpose for cloudfoundry is to push apps, it doesn't stop there: managed services automate the provisioning of backing resources for developers.

The lab here is rather well done: it provides the context for a student to experiment with both types of services: a user-provided service and a managed service. The crux of the lab is for the student to understand what a create-user-provided-service -p invocation does: that it's interactive,

and specifying the right value for the uri: not to use https (because the client is not setup to make ssl calls, and to adhere to the contract setup by the author of the feature: to specify the uri with the attendees/ suffix.

TODO: personally i'd rather the client add the suffix if it's missing (i.e. allow entry of the base url).

I've seen some oddities in the behavior of articulate here, where sometimes the list of attendees will properly appear on the page, but refreshing the page might result in an empty view. TODO: investigate this.

4.4. Module: Manifest

Once more, this lab is a perfect follow-on to the previous one: now that we've had the exposure to using cf push and learned a bunch of flags, let's tuck them away into a manifest file.

This module has no presentation! The idea was to put the onus on the students to learn the feature by reading the docs. In the past, I've often found attendees resist doing this: they expect you to teach them the material, not for them to have to read it. I can't really disagree with them.

About the lab, I personally wish students were made to write a manifest from scratch. Instead they're instructed to invoke a command that will write it for them, which I think is a shame. I rarely see students write their own manifests in subsequent labs, and I suspect it's because they weren't tasked enough in this lab.

TODO: revise the lab accordingly.

During review, suggest to the students that in subsequent labs, the completion of the lab can be simpler if they opt to write a manifest instead of following the instructions verbatim.

TODO: consider re-writing subsequent labs to require the student to write the manifest, especially where services are involved, to avoid the whole cf push --no-start, bind-service, cf start hoop that they need to jump through.

If at this point I'm at 4:00 PM or later, I typically will choose to stop here for the day.

4.5. Module: Application Security Groups

The major obstacle in this lab is that ASGs cannot be played with unless one is an administrator. I mitigate this problem by demonstrating the lab in a group setting over the projector, by using a separate PCF instance that I stood up for such occasions.

Summary of the demo: - push attendee-service with backing service mysql - ask the class: why does this work? there must be an ASG that allows the app to talk to mysql - task: find the asg in question: cf security-groups and running-security-groups should show the default_security_group - task: remove default_security_group from the running set and restart the app, to demonstrate that it can't connect to the database any longer - task: create a custom asg for mysql (hint: lookup ip and port from VCAP_SERVICES) - task: bind custom asg to the org in question, restart the app, and demonstrate the app is working once more

4.6. Module: Log Drain

The presentation here is very brief, just a review of parts of the presentation from the logging, scale and ha module. Review "treat logs as a stream" from the 12 factors, how the loggregator works. I make analogy to plumbing and pipes. Tapping into doppler to forward application logs to a third party service.

This lab is often completed very quickly: there's not much to do to make this work. And that's partly the point: that it's easy to add capabilities or support to an application that's running in PCF.

Often students will be aware of the potential security issue of sending logs to a third party, and so perhaps point out that this same mechanism can be used to send logs to an internally deployed log analysis tool.

4.7. Module: Blue-Green

The presentation here has two parts: blue-green deployments, and a discussion of other concerns when upgrading an application, such as database migrations, and gotchas having to do with java serialization and class evolution. I try to focus on the former.

The lab is pretty cool, with the visualization of percent of traffic that flows to each of two applications. It's a little contrived in the sense that the same artifact is deployed each time.

TODO: consider how to improve this. TODO: explore using names aritculate-blue and articulate-green instead of articulate and articulate-temp (personal preference).

4.8. Module: Microservices

Presentation only. It's essential to point out that a platform can only go so far. In tandem, we must design our applications to run well in the cloud: to be cloud native.

4.9. Module: Application Autoscaler

This is a fun lab. No presentation here. Using jmeter to exercise a load against an application is always fun, and seeing an app autoscale.

One difficulty here is maintaining a high cpu for over 30 seconds, especially when the endpoint just returns data. I usually can make it work for myself. Increase the number of jmeter threads or lower the cpu threshold, or try to hit a different endpoint. If you encounter such difficulty, consider manually scaling up the application to five instances, then to turn on the autoscaler and watch it scale down your application.

4.10. Module: Application Performance Monitor

Here we use newrelic. Again, there's no presentation. What I like to do with each module that has no presentation is to give a quick overview of the lab, a sort of introduction to the activity they're about to undertake.

Stress that the students should use the managed service in the PWS marketplace (assuming you're using PWS), and that the integration with newrelic transparently takes care of the "account-setup" negotiation between PWS and newrelic (i.e. there's no need to create a newrelic account).

When you click on the "Manage" link for newrelic, you're taken directly to the newrelic dashboard. I've run into an issue where a previous manual login into a pre-existing newrelic account will prevent this from working properly. The remedy is to look for and delete cookies associated with the newrelic.com domain.

Finally, it's important to stress in this lab how the integration with newrelic actually works. Typically one would have to manually bundle the newrelic agent jar file with their app. Here we see the buildpack doing this for us transparently: that's essentially why we need to restage the application for the integration to work: because the buildpack downloads and includes the newrelic agent jar file into the newly-created droplet.

4.11. Module: Metrics

This lab is very short and sparse. Again it puts the onus on the student to take the time to read about pcf metrics and to perform adhoc experimentation with the tool.

TODO: consider ways to expand or re-cast the lab with specific activities and goals.

I've often ended the day early at this point (~ 3:30 PM - 4:00 PM) and reserved the remaining topics for day 3.

4.12. Module: Buildpack

The presentation here is straightforward and to the point. I usually like to complement the existing materials (presentation and lab) with the following discussions and demonstrations:

- discuss the variety of buildpacks that exist: the built-in ones, the community buildpacks. i like to demonstrate the staticfile buildpack as an example.
- demonstrate running the java buildpack's detect, compile, and release scripts locally.
- demonstrate additional features of the java buildpack. I like to point out ben hale's blog entry (https://blog.pivotal.io/pivotal-cloud-foundry/products/new-cloud-foundry-java-buildpack-improves-developer-diagnostic-tools)
  and show how to put a breakpoint in an app and remote-debug a cloudfoundry application by setting up an ssh tunnel.
- show the administrative side of buildpacks: how an admin might ugprade a buildpack with the delete-buildpack and create-buildpack commands (or update-buildpack). i use a personally deployed instance of cf for this, as it requires admin rights.

4.13. Module: Service Broker

The presentation here is straightforward. I like to pull up the service broker api documentation online and show students where they can get to the docs.

For the lab here we provision a vm instance in AWS in advance. Here's the terraform script for it. The output of the script will be the public ip address of the vm, which needs to be communicated to the students (they set an environment variable named MONGODB_HOST).

I like to review the lab in two parts:

1. explain the mechanics of pushing the app, viewing the /v2/catalog endpoint, registering the service broker, and otherwise making things work
2. explain the code that impelments what the service broker does (creating databases, users)

I also like to ask students to create a custom album in spring-music and ssh into the mongo vm, and find that record for them, as a sort of validation that the data is indeed being persisted in their own db.

This also gives me the chance to talk about and explain the --guid flag in many of the cf commands.

4.14. Module: Continuous Delivery

With respect to the lab, students here have two options: they can run a jenkins instance locally on their machine (the hard way), or use a jenkins instance that's already provisioned for them. In my experience, the latter option is a lot simpler and goes a whole lot smoother. So encourage students to use your aws-provisioned instance of jenkins. Here's the terraform script.

For the presentation, I often go beyond what the slides offer.

TODO: retrofit the slide deck to include additional information: pictures of teams working in an agile fashion, radiators on dev floors, a screenshot of a build pipeline, mike cohn's test pyramid, perhaps a photo of mary popendieck to go along with her quote, and the name of the book from which the quote was taken, etc..

4.15. Module: Route Service

This module targets a distinct feature of cloudfoundry and works well. Students often have difficulty with the basics: they often will fail to run a "cf help" on a new command before invoking it, or are confused by the syntax of a command, that one must specify the domain name and hostname separately. They often also struggle with taking an example instruction and revising it correctly so that it uses their application's hostname, for example.

Version 0.1
Last updated 2017-03-21 10:58:42 CDT