Preparation

1. Sign up free account for Pivotal Web Services: http://run.pivotal.io/
2. Install Cloud Foundry CLI: http://docs.pivotal.io/pivotalcf/1-10/cf-cli/install-go-cli.html
3. Push a sample Java app to test the configuration:

```
$ cf login -a api.run.pivotal.io   # Please select Dev
$ git clone https://github.com/cloudfoundry-samples/sp
$ cd spring_music/ ; ./gradlew assemble
$ cf push
```

   Look for `urls: spring-music-XXX-XXX.cfapps.io` in `cf push` command output, then open that URL in your browser.

4. Delete the app to save your money:

```
cf delete -r spring-music
```

5. All done!

PCF workshop reference material

- Open source CF docs can be found at https://docs.cloudfoundry.org/
- If you need to lookup Pivotal's PCF proprietary stuff: http://docs.pivotal.io/pivotalcf/
- Docs for Pivotal's public cloud - Pivotal Web Services (PWS) can be found at: https://docs.run.pivotal.io/
- 12 factor cloud native apps: https://12factor.net/

Workshop #1: From source code to PaaS cloud

Workshop #1 tasks:

1. Login to CF, and deploy `node` and `spring-music` example apps to CF.
   Example apps can be found in `./workshop-material/demo-apps`
2. Open these apps in browser to see if they work

Workshop #1 questions:

1. How does CF support multiple languages? Can you extend it to support more languages/custom runtimes? (Reference: https://docs.cloudfoundry.org/buildpacks/)
2. Can you push your java source code directly to CF, and let CF build & compile & run it?
3. Run `cf target` from your terminal, and explains what each item(`API endpoint` , `User` , `Org` , `Space`) means.

Workshop #1 cleanup:

When you're done with this workshop you can run the following command for all your apps to save some money (PWS price is calculated on a per hour basis):

```
$ cf delete -r YOUR_APP_NAME
```

You can also run `cf apps` to see all your apps in your targeted space.

Workshop #2: Manifest & Logging

Workshop #2 tasks:

1. Now you're familiar with `cf push` to deploy apps, you need to deploy `articulate` application in the `demo-apps` directory, give it `512MB` memory and `random-route`.
   Note: You're required to use `manifest.yml` file instead of CLI to do this.
2. Observe `articulate` application's `logs` and `events`.
3. Don't delete the app yet as we need it for next workshop.

Workshop #2 questions:

1. Where should your application write logs? Hint: see here
2. What are some of the different origin codes seen in the log? (e.g. `API`, `STG`, `CELL`, `APP`, `RTR`). Please explain what each code means. For reference see here
3. How does this change how you access logs today? At scale?

Workshop #3: Scaling & High Availablity

Workshop #3 tasks for scaling:

1. First start tailing the logs and look specifically for logs from Cloud Controller and Cell components:

   ```
   $ cf logs articulate | grep "API\|CELL"
   ```

2. *Vertically scale* `articulate` memory up to `1G`. Observe the log output.
   Hint: you can use either CLI or `manifest.yml` file to achieve this. Feel free to see reference doc here.
3. Scale `articulate` back to origin settings (512MB memory).
4. Horizontally scale `articulate` to 3 instances.
   Notice how quickly the new application instances are provisioned and subsequently load balanced.
   Don't scale back to 1 instance yet.

Workshop #3 tasks for HA:

1. Confirm that `articulate` is running on multiple instances:

```
$ cf app articulate
```

2. Find a way to cause the app to exit, or to crash the app.
3. Observe the app state by running `cf app articulate` again.
4. View which instance was killed by running `cf events articulate`
5. Scale `articulate` back to original settings (1 instance).

Workshop #3 questions:

1. What is the difference between vertically scaling and horizontally scaling? What does "scaling out" and "scaling up" mean?
2. How do you recover failing application instances?
3. What effect does this have on your application design?
4. How could you determine if your application has been crashing?

Hint: read about Disposability in 12 factor apps and Crash-only design

Workshop #4: Services

Workshop #4 tasks for Managed Service:

1. Deploy `attendee-service` from `workshop-material/demo-apps/attendee-service` directory. Make sure it runs correctly, and try visiting its URL in browser.
   Hint:
   i. you can use command: `cf push attendee-service -p ./attendee-service-0.0.1-SNAPSHOT.jar -m 512M --random-route` to push your app
   ii. run `cf marketplace` to find what you can use for providing your application with a *managed database sevice*.
   iii. Doc on CF Services can be found here
   iv. This attendee-service app uses Spring cloud connector to connect to its database. More doc here.

Workshop #4 tasks for User Provided Service Instance:

1. Browse `articulate` application's "Services" page. There is no service currently bound. `articulate` 's default configuration for the `attendee-service` `uri` is http://localhost:8181/attendees. You should override this `uri` parameter to your `attendee-service` in the cloud.
2. Create `attendee-service` as a "user provided service" and bind `articulate` to the `attendee-service` user provided service.
   Hint: Reference docs can be found here
3. Test the setup by going to `articulate` 's Services page and add some attendees.

Reference: articulate source code, attendee-service source code

Workshop #4 questions:

1. How does `attendee-service` find its database credentials and connect to the database?
   Hint:
   i. 12 factor apps have sections on backing services and configuration.
   ii. Run `cf env attendee-service` and read about VCAP_SERVICES
   iii. Different languages/frameworks will have various ways to read environment variables. `attendee-service` takes advantage of a Java Buildpack feature called Auto-Reconfiguration that will automatically re-write bean definitions to connect with services bound to an application.
2. Why could we *restart* `attendee-service` instead of *restage* it?

Workshop #5: Blue-Green Deployment

Workshop #5 tasks:

1. Follow the reference doc and do blue-green deployment on `articulate` app.
   Hint:
   i. Reference doc here
   ii. You need to push 2 apps, one for current version and one for next-release version.
   iii. Use `cf map-route` and `cf unmap-route` commands to manage the traffic to these apps.
   iv. Go to `articulate`'s `Blue-Green` page, hit start to see how requests go to each app based on route mappings.

Workshop #5 questions:

1. Why do we want to do Blue-green deployments?
2. If the new version of application has bug, how do we do a rollback?
3. When you design an app to be blue-green deployed, are there any design constraints?
4. Is there any way for blue-green deployment to be automated?

Workshop #6: Application Security Groups

Workshop #6 tasks:

1. List the security groups in your environment.
   Hint: Reference doc here
2. View the rules detail of `public_networks`, `dns` and `p-mysql` ASGs.

Workshop #6 questions:

1. Is ASG rule a whitelist or blacklist?
2. Run `cf help -a` and explain all the security-group related commands.
3. What are the differences between staging-security-groups and running-security-groups?