Buildpacks

Goal

Explore, configure and update a buildpack.

Note: This lab assumes you are the administrator of the Pivotal Cloud Foundry installation.

Approximate time: 30 minutes

Exercises

Use and explore the staticfile builpack

1. Create a directory named `hellohtml` to hold a simple html application.
2. Add a file named `index.html` to the `hellohtml` directory, containing the following:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Hello world</title>
    </head>
    <body>
          Hello world
    </body>
</html>
```

3. In a terminal window, change to your `hellohtml` directory and **attempt** to push the application. Watch the output during the cf push.

```
cf push myhellohtml -m 128M
```

4. The cf push should not have been successful. Scroll back up through the output. You should see the following:
   - An app and route were created.
   - The files in your `hellohtml` directory were uploaded to Cloud Foundry.
   - A staging container was created.
   - Since no buildpacks were specified, all of the standard/installed buildpacks were downloaded to the container.
   - All of the buildpack `detect` scripts were run, and none of the `detect` scripts were successful.
   - Staging failed.

5. Enter `cf buildpacks` to see a list of system buildpacks. Notice that there is a buildpack called `staticfile_buildpack`. This buildpack is for standard html documents, but the `detect` script wasn't successful. Buildpack source code and documentation are available on GitHub. Most are easy find with a Google search. Do a search for `Cloud Foundry staticfile buildpack` and open the link to the buildpack in GitHub. This should be https://github.com/cloudfoundry/staticfile-buildpack.
6. View the README.md file (on the homepage for the buildpack). Notice that an empty file named `Staticfile` needs to be in the application directory.
7. Here is another way to see that a `Staticfile` is needed. In GitHub, click on the `bin/detect` script. Notice that the script is a bash script, and it succeeds (exits with `0`) if there is a `Staticfile`.

8. Add an empty file named `Staticfile` to your application directory and attempt the push again. This time you should be successful.

```
cf push myhellohtml -m 128M
```

9. Scroll back up through the cf push output. This time you should see the following:
   - The staticfile buildpack `detect` script succeeded. Staging messages from the compile script start with `Staging...` and end with `Staging complete`.
   - A droplet was created and uploaded to the Cloud Controller blobstore.
   - The application was started.
10. In GitHub, click on the `bin/compile` script. Notice that the script is a bash script, and you can see that the echo statements match the staging messages from cf push (they might differ if the installed buildpack version is not the same as the latest GitHub version). One of the major things that this buildpack does is download and configure nginx to be used as the web server in the container.
11. Scroll back up through the cf push output. Notice that there is a message similar to: App myhellohtml was started using this command `sh boot.sh`. In GitHub, click on the `bin/release` script. Notice the `sh boot.sh` in this script. This start command is uploaded to the Cloud Controller database during staging and is used to start new app instances.
12. Scroll back up through the cf push output. Notice that because no buildpack was specified during cf push, all of the installed buildpacks are downloaded to the staging container. Check the logs to see how long it took to do this (you should see that it only takes a few seconds to download the buildpacks and stage the application):

```
cf logs myhellohtml --recent
```

13. cf push the application again, but this time specifying the buildpack name using the -b parameter. Notice that only the staticfile buildpack is downloaded to the staging container. You do not need the `Staticfile` and could remove it if you would like. This is because the `detect` script is not run when specifying the buildpack, and the `compile` script will just assume default settings if no `Staticfile` is found.

Perform a developer buildpack configuration

1. View the configuration information for the staticfile buildpack. This is on the home GitHub page for the buildpack. https://github.com/cloudfoundry/staticfile-buildpack. These are the developer configurations available for the buildpack.
2. Follow the steps under `Basic authentication` to protect your html page with basic authentication. You can remove the protection when you are done.

Use a buildpack hosted on GitHub

1. Instead of using an installed buildpack, you can use buildpacks directly from GitHub. In Ops Manager, verify that this capability is enabled. On the Pivotal Elastic Runtime tile, verify that Cloud Controller > Disable Custom Buildpacks is not checked.
2. Use the latest GitHub staticfile buildpack when pushing your app:

```
cf push myhellohtml -b https://github.com/cloudfoundry/staticfile-buildpack.git
```

3. If you have time and a GitHub account: Fork the staticfile buildpack to your own repository. Add an `echo "helloworld from the compile script"` message to your forked bin/compile script and verify that you see this staging message when pushing your app and specifying your GitHub-hosted buildpack.

Add a custom system buildpack

1. A properly built zip file or directory is needed before you can add or update a system buildpack. http://network.pivotal.io contains downloadable buildpacks for installation. Browse to http://network.pivotal.io, then click on Buildpacks. You can see the latest Pivotal-released backbacks there.
2. Another way to obtain properly built buildpacks is to use the `Release` section of the GitHub repository for the buildpack. On the staticfile GitHub home page, click on `releases` (it is above the Download ZIP button). Download the latest release of the staticfile zip file. You will add this as a new system buildpack.
3. In a terminal window, navigate to the location of your downloaded zip file and add a new system buildpack at position 1 in the buildpack list.

```
cf create-buildpack staticfiles_buildpack<your_initials> <buildpack_zipfile> 1
```

4. Use `cf buildpacks` to verify that you have added a system buildpack. Test your buildpack, specifying it as the -b parameter in cf push.

(If you have time) Build a custom system buildpack

1. To add a custom buildpack to a Cloud Foundry installation, it must be properly built. Build instructions vary by buildpack, and are usually included on the buildpack's GitHub home page. View the build instructions for the staticfile buildpack under To create/upload from source repository (https://github.com/cloudfoundry/staticfile-buildpack#to-createupload-from-source-repository).
2. (If you have a GitHub account, Bundler (http://bundler.io/), and Ruby (http://rvm.io/) installed) Build and upload your forked staticfile buildpack following the staticfile buildpack's instructions. Even though the staticfile buildpack scripts are not written in Ruby, the buildpack-packager is. Verify that you can see your custom hello world message when pushing an application using this buildpack.

Congratulations, you have successfully explored, configured and updated a backpack!

Last updated 2016-01-08 11:18:43 PST