pushing a node app to Cloud Foundry

You won't believe what really happens!

Patrick Mueller - IBMer - **@pmuellr** - **muellerware.org**

http://pmuellr.github.io/slides/2014/04-what-happens-when-you-push-a-node-app

Patrick Mueller

developer advocate at IBM

BlueMix Platform-as-a-Service and node.js

Raleigh, NC

more of my slides: http://pmuellr.github.io/slides

what is BlueMix?

- IBM Platform-as-a-Service product
- Runs node apps! Runs any apps!
- Based on the Cloud Foundry open source project.

For more info: https://bluemix.net

attribution

JS Logo from https://github.com/voodootikigod/logo.js

Slide framework from http://remarkjs.com/

GlyphIcons Free from http://glyphicons.com/

Cloud Foundry icon from somewhere.

overview

1. the process
2. the files




the process

the process

1. determine files to upload
2. upload files to staging machine
3. select buildpack
4. run buildpack on staging machine
5. store buildpack output as a droplet
6. start app with the new droplet

the `cf push` command

- notes on usage
- run **`cf help push`** on the command-line for help
- we will be running the command **`cf push`** in the directory with the node app, with no other command line arguments or options.

determine files to upload

By default **`cf push`** will collect all the files in your current directory, recursively.

Prevent files from being uploaded by specifying them in your **`.cfignore`** file - more on that in later slides.

You will want to make sure you don't upload:

- files that aren't used when the app is running
- files with sensitive data, unless you know what you are doing

upload files to staging machine

The files are placed in a compressed archive, and uploaded to a staging machine for further processing.

select buildpack

Once the files are uploaded to a staging machine, a buildpack is selected.

The existance of a **`package.json`** is what the node buildpack uses to identify these files as a node application.

run buildpack on staging machine

The buildpack **`compile`** step is run, which will:

- download an appropriate version of the node runtime, and npm
- run **`npm install --production`** to ensure the required modules are loaded
- run **`npm rebuild`** to recompile native node modules, if required

store buildpack output as a droplet

The buildpack output will consist of:

- a node runtime
- your application files
- node packages in the node_modules directory, built for the correct platform

This will be placed in a compressed archive as a "droplet"

start app with the new droplet

A Droplet Execution Agent (DEA) is assigned to run the droplet.

The droplet is unpacked, and the start command is used to start the application.

staging optimizations

For node, a significant amount of time is spent on the staging machine to run the **`npm install`** command. And it turns out, it's usually installing the same packages, each time.

To optimize this, the modules installed by the staging machine are saved between **`cf push`** invocations, and reused for subsequent **`cf push`** invocations.

A fresh **`npm install`** will be run if on a subsequent **`cf push`**, the contents of your **`package.json`** file changes.

common problems during staging or application start

- did you create a **`Procfile`** to indicate the start command?
- is npm down? this could cause staging to fail; check on your local machine
- did you check for errors / failures in your startup code?
- is there lots of logging in your startup code?

the files

the files

sample hello world program

```
.gitignore
.cfignore
Procfile
manifest.yml
node_modules/...
package.json
server.js
```

.cfignore

`.cfignore`

- like a **`.gitignore`** file
- indicates files to NOT upload to Cloud Foundry
- **`.git`**, **`.svn`**, and **`.darcs`** files ignored by default

`.cfignore` sample

```
node_modules
bower_components
tmp
```

for more info, see the cf docs

Procfile

Procfile

- patterned off of Heroku's Procfile
- will make your app more compatible with Heroku
- won't need to use the "start command" option
- only ever contains one line

```
web: node [main program here]
```

manifest.yml

rules of pushing apps

rules of pushing apps

1. always use a **`manifest.yml`** file

rules of pushing apps

1. always use a **`manifest.yml`** file
2. **always use a `manifest.yml` file**

`manifest.yml`

- provides initial settings for your app, including
  - app name, as cf knows it
  - host name, for the final URL to the app
  - RAM requirements
  - cf services to bind to
  - etc

`manifest.yml` sample

```
---
applications:
- name:     hello-node
  host:     hello-node-${random-word}
  memory:   128M
```

for more info, see the cf docs

Sample applications use often use **`${random-word}`** in the **`host`** property, so that your app's URL doesn't collide with someone else's.

`manifest.yml` generator

Some YAML-challenged folks, such as myself, might want to use the **`cfmanigen`** web application (running on BlueMix) to generate your **`manifest.yml`** files. This app gives you a fill-in-the-blank form, for entries in the manifest, and then generates the manifest for you in a click of a button.

For more information, see the DeveloperWorks blog post which discusses the **`cfmanigen`** app.

```
node_modules/...
```

`node_modules/...`

The **`node_modules`** directory is where **`npm`** installs packages for your application.

You can choose to upload your node_modules directory, along with your app, or to **NOT** upload them.

`node_modules/...` uploading

pros:

- more control over the modules used in the app
- faster "staging" since modules are already available

cons:

- longer upload for staging process, for each **`cf push`**

`node_modules/...` uploading

recommend:

- not uploading

controlled via:

- **`node_modules`** entry in **`.cfignore`** file

package.json

rules of pushing apps

1. always use a **`manifest.yml`** file
2. **always use a `manifest.yml` file**

rules of pushing apps

1. always use a **`manifest.yml`** file
2. **always use a `manifest.yml` file**
3. always use a **`package.json`** file

rules of pushing apps

1. always use a **`manifest.yml`** file
2. **always use a `manifest.yml` file**
3. always use a **`package.json`** file
4. **always use a `package.json` file**

`package.json`

- nothing special needed for Cloud Foundry!
- file **MUST** exist, or app will not be recognized as a "node" application during staging
- may need/want to tweak for running on Cloud Foundry machines:
  - set specific version of node to use; see the note at Pivotal concerning use of some native modules for node.

server.js

`server.js`

Your application code. For the sample hello world server, that is just this single file. You will likely have other files and directories containing the code for your app.

fin