CF SUMMIT: V7 CF CLI Lab

Originally Prepared and presented by:

- Josh Collins
- Jenna Goldstrich
- Alexander Berezovsky

Pre-requisites

An internet-connected computer running Mac OS, Windows 7+, or a Debian based operating system such as Ubuntu

A Github Account

Prep

1. **Clone the Squeeze app from github** (https://github.com/a-b/cf-summit-2020-lab)
2. **Install the v7 cf CLI**
    - Please note that if you do not uninstall previous v6 cf-cli installations first you may have to use cf7 instead of cf for all of these commands, you can check which version of the cli you're using with cf/cf7 version
    - Binaries (NOT RECOMMENDED) https://github.com/cloudfoundry/cli#installers-and-compressed-binaries-2
    - Windows
        - https://packages.cloudfoundry.org/stable?release=windows64&version=v7&source=github
            - This link is for an installer, if you have cf cli installed previously on windows you might want to remove your previous install first
    - Mac OSX
        - `brew tap cloudfoundry/tap`
        - `brew install cf-cli@7`

    - If you already had the v6 CLI installed run `brew unlink cf-cli@6 && brew link cf-cli@7 --overwrite`

- Fedora
    - `sudo wget -O /etc/yum.repos.d/cloudfoundry-cli.repo https://packages.cloudfoundry.org/fedora/cloudfoundry-cli.repo`
    - `sudo yum install cf7-cli-7.0.0`
- Ubuntu
    - `wget -q -O - https://packages.cloudfoundry.org/debian/cli.cloudfoundry.org.key | sudo apt-key add -`
    - `echo "deb https://packages.cloudfoundry.org/debian stable main" | sudo tee /etc/apt/sources.list.d/cloudfoundry-cli.list`
    - `sudo apt-get update`
    - `sudo apt-get install cf7-cli=7.0.0`

Introduction

The v6 major release line of the CLI (backed by the v2 CC API) has been in production for more than 7 years. It's done an amazing job helping enterprise developers focus on writing and delivering software.

Over the years, the v6 CLI has become increasingly complicated and the CAPI team has implemented many modern features in v3 of the CC API.
These features haven't been available through the v6 CLI (except as experimental v3-prefixed commands).

The v7 CF CLI is backed by the v3 API.
We're excited to be working directly with the API backend going forward in lock-step as we actively develop new features and enhancements to the v7 CLI release line.

Shout out to all the folks who were instrumental in getting here.

In this lab we're going to cover the following V7 features:

- Pushing apps (with and without downtime)
- Scaling apps
- Running Multi-Process apps

- Running apps with Sidecars
- Using Push sub-step commands
- Rolling back an app to the previous droplet with no downtime

Lab Narrative Setup:

We'll organize the lab around a hypothetical "real-life" scenario involving a handful of budding startup entrepreneurs named Ragu, Shelby, and Darrel.

*The three of them founded **Squeeze.***
They aspire to become "The World's Lemonade Stand" within the next two years.

This is a grand aspiration and they're under the gun to get started.

They've decided to separate responsibilities and attack the problem in parallel.
Ragu's responsible for the lemonade stand.
Shelby's responsible for advertisement.
Darrel's responsible for making lemonade.

In the spirit of lean product development and enthusiasm for the massive addressable market for fresh natural juices, they're all on fire as they forge ahead.

Part 1: Squeeze's first push

Ragu's passion for the lemonade business fuels his spirit and he's sprinted out ahead and finished his portion of the app well before his compatriots.
He cannot be contained.
He's ready to push it live.

As Ragu, you'll now execute the initial push using the classic `cf push` command.

Step A) Target your CF API and log in:

- Go to your `cf-summit-2020-lab` directory
- `cf api api.[YOUR-DOMAIN] --skip-ssl-validation`
- `cf login`
- Enter and submit your username/password for the email address prompt

Step B) Push your app!

- `cf push`

Step C) View your app in a browser

- When your app push is complete, open a web browser of your choosing
- Load your unique app URL in your browser window:
  `http://squeeze.[YOUR-DOMAIN]`
  * you can also copy the route value from the cf push command output in your term window

- Once your app is loaded, you should see the following:
    - the lemonade stand graphic
    - an up-time indicator on the top-left region of the screen
- The uptime indicator will change status if any part of the application is unavailable.
- Let's validate the uptime indicator works as expected

Step D) Validate the uptime indicator
The app front-end calls the backend asynchronously to confirm the overall health of the app. If the backend responds with a 404, the uptime indicator will change to a sad face to indicate a "down" state

- `cf push` the app again
- While the app is being updated, monitor the uptime indicator in your browser
- You should see the indicator show the backend is down temporarily while the app starts up (this is sub-optimal for any production app - and we'll use the V7 CLI to avoid this in future app pushes)

Visit our docs for more detailed information on pushing apps:
https://docs.cloudfoundry.org/devguide/deploy-apps/deploy-app.html

Part 2: Scaling multiple processes with the v7 cf CLI

Voila! Ragu is thrilled to see his app up and running.
After a few days with not a single customer, Ragu feels it's time to enhance their product so it's more attractive to the market.
He excitedly calls Shelby and Darrel and learns that Shelby's advertising feature is ready to go.

The advertising portion has been architected as a separate worker process from the Ragu's web process. Running ads via the separate worker process alleviates overhead from the web process allowing it to focus on it's core responsibility of serving lemonade to customers. Additionally, the web and worker processes, while deployed in the same push, can be scaled independently of each other as appropriate (more customers may warrant scaling up the web process while a lack of customers would warrant scaling up the advertising worker process)

(*note - in actuality, the worker process has already been pushed previously and now you'll scale the process to start advertising)

Multi process apps all share the same source, but use different start commands to serve different use cases and scale independently. Some examples where this could be useful are having the front end and back end deployed from the same source with the freedom to scale independently.

Step A) View current details for the app

- `cf app squeeze`
    - Notice there's a worker process section in the app details, and that there are no instances of the worker process running currently
      (*note the presence of a sidecar in the details as well; again, the app you originally pushed had all the components present, just not enabled - you'll enable the feature provided by the sidecar in a later step)

Step B) Scale your worker process

- `cf scale squeeze --process worker -i 1`
    - Monitor your term and browser windows in concert to note the changes in the UI as the worker process spins up asynchronously from the front-end
    - Notice the following:
        - A map is displayed
        - 1 lemonade stand sign is posted on the map
        - And oh my, a customer shows up at some point!
    - Because you know the worker processes are effective, you'll modify your manifest to scale up the processes. This will ensure so you'll always have 8 processes if/when you restart the app in the future.

Step C) Scale your worker processes further by updating the manifest

- Update the value on line #24 of **`manifest.yml`** to instruct the platform to spin up 8 worker processes
    - line #24 of your manifest should be updated from `instances: 0` to `instances: 8`
    - save your changes
- `cf apply-manifest`
- Monitor your term and browser windows as you did previously
- Notice the following:
    - More lemonade stand signs are posted on the map
    - And oh my, more customers show up and are ready to purchase!

Visit our docs for more detailed information on using app manifests and running apps with multiple processes:
https://docs.cloudfoundry.org/devguide/deploy-apps/manifest.html
https://docs.cloudfoundry.org/devguide/deploy-apps/manifest-attributes.html
https://docs.cloudfoundry.org/devguide/multiple-processes.html

Well...this is awesome, we've got a lemonade stand, advertising, and customers. What else could we possibly need?

Part 3: Lemonade! - Sidecars and Zero Downtime Deployments

Rightly so, Ragu and Shelby are giddy with excitement.
Parents have been called. Hugs and libations have been had. Gloryful imaginative basking has begun.

As the dream comes into focus, the Squeeze founders realize the pitchforks will come out unless they can quench their customer's palpable thirst.
Time is of the essence!

Shelby and Ragu's excitement turns into terror when they learn that Darrel has written his portion of the app in a completely different language (python)!

After a panicked huddle, everyone is calmed because Darrel's cool as a cucumber and quickly refactors his lemonade production component so it can run as a sidecar.

Unlike worker processes, sidecars can't be scaled independently from the main app process. However, the sidecar and main app process run in the same container and as such, can share resources such as the file system and can communicate more directly/efficiently with the main app process.

Code that must run alongside other processes to provide shared functionality or access the same resources (disk/network) would be good candidates for running as sidecars. For example: an oauth 2 proxy that can extract auth logic out of a web app.

And because sidecars can be written in a different language than the main app process, code for single sidecar can be deployed alongside many or all apps within an organization in order to meet a common requirement in a reliable, approved, and/or consistent manner (rather than each app dev/team taking the time to code and manage a their own unique solution for said requirement).

As mentioned previously with multiple processes, the sidecar for your app in this lab has already been deployed as part of your previous pushes but its lemonade production feature hasn't been enabled yet.

In the next step, you'll update and push the sidecar code so it starts making lemonade.

Step A) Open `sidecar.py` from the main repo in your text editor of choice

Step B) Enable lemonade production by specifying the recipe in **`sidecar.py`**

- Comment out line 11 `f.write("")` by putting a # in front of it (`#f.write("")`)
- Uncomment line 9 (`#f.write("originalRecipe")`) by removing the `#` sign in front of it (`f.write("originalRecipe")`)
- (TIP: in Python lines of code are commented out with a '#')

```
import os
import time

sidecar_file_name = os.getenv("SIDECAR_FILE_NAME", "/tmp/sidecar.txt")


while True:
    f = open(sidecar_file_name, "w")
    f.write("riginalRecipe")
    #f.write("lemunique")
    f.write("")

    print("Sidecar output: ", f)
    f.close()
    time.sleep(10)
```

Your **`sidecar.py`** should look like the picture above now (make sure you save your changes, and watch your indentation!)

Now that we are this far along in the application's lifecycle, a downtime event could harm our relationship with our users, so we're going to use a new feature in the v7 cf CLI called rolling deployments (or zero down time)

Step C.) `cf push --strategy rolling`

While this push is going, observe the downtime indicator remains in "Available" state throughout the update.

Once the push is completed we finally can see we have our advertisements bringing in customers through multiple processes, and our delicious lemonade being served up via sidecar!

What's happening in the background during the rolling deploy?
During an app update executed with `--strategy rolling`, the web process (and any sidecars running in the same container) are the processes which get rolled one by one.

The worker processes are updated all at once and after all the web processes have been rolled, so there's downtime for those background processes in the current implementation of the feature.

During the rolling deploy:

1. A copy of web process, including any of its changes is created
2. It's scaled to one instance
3. A healthcheck is established
4. Once the web process is healthy…
5. The old web process instance is torn down
6. Steps 1-5 are repeated for as many app instances exist for the app
7. When all the web processes (if they exist) have been rolled, the worker processes are updated all at once

The fact that there will be both the old and new versions of the app web process running simultaneously must be accounted for in how the app is architected such that the changes are forward and backward compatible.

Although similar, the rolling deployment is not the same as a blue-green deployment.
With a blue-green, on top of having to run many different cf commands to execute, a completely separate updated version of the app is deployed and scaled as necessary to match the original, and then the route to the original app is bound to the new version.

Visit our docs for more detailed information on sidecars and rolling deployments:
https://docs.cloudfoundry.org/devguide/sidecars.html
https://docs.cloudfoundry.org/devguide/deploy-apps/rolling-deploy.html

Part 4: Lemon seeds take Squeeze to the next level

The stratospheric success and buzz associated with the Squeeze MVP release has garnered the foundlings a great deal of attention on the interwebs and they've been courted by multiple investors.

The foundlings have just secured a substantial round of Angel Lemon Seed funding and they've decided to invest a significant portion of their capital in customer research.

Five grueling weeks with two Stanford PhD's, a handful of eager interns from CalTech and MIT, and ONE MILLION DOLLARS later, they've discovered a new recipe that should enable them to take a giant leap forward towards being The World's Lemonade Stand. OMG!

Darrel integrates their new recipe, lemunique™ into his production app in a snap (just like always).
Everyone's poised on the edge of their seats with their eyes on their Pithy AnalyticsTM dashboard as they prepare to launch the change.

They can't afford to lose a single customer during the update, so they're going to use the mythical "rolling" deployment strategy to push the update with zero downtime for their customers.

Yeah, so now it's time to do that with a simple change to the python file and a simple push with an additional flag.

Step A) Re-open the **`sidecar.py`** file

Step B) Switch the lemonade recipe in **`sidecar.py`**

- Comment out line 10 (`f.write("originalRecipe")`) by adding a # in front of it (`#f.write("originalRecipe")`)
- Uncomment out line 11 (`#f.write("lemunique")`) by removing the # from in front of it (`f.write("lemunique")`)

```python
import os
import time

sidecar_file_name = os.getenv("SIDECAR_FILE_NAME", "/tmp/sidecar.txt")


while True:
    f = open(sidecar_file_name, "w")
    #f.write("originalRecipe")
    f.write("lemunique")
    f.write("")

    print("Sidecar output: ", f)
    f.close()
    time.sleep(10)
```

Your **`sidecar.py`** should look like the picture above now (Don't forget saving and indentation!)

Step C) `cf push --strategy rolling`

- As the rolling update is executed, keep a close eye on the UI
- What change do you observe?

Visit our docs for more detailed information on sidecars:
https://docs.cloudfoundry.org/devguide/sidecars.html

Part 5: CF push sub-commands save the day

The world was watching with anticipation.
And the world was shocked and disturbed by what they saw.
In an attempted act of heroism, Darrel hurriedly fat-fingered his delivery of Lemunique™. The resulting drink on-offer was a disgusting putrid green.

Darrel needed to fix this, and fix it fast before his world imploded.

The v7 CLI makes it possible to break up cf push into several sub-steps.
Breaking cf push down into sub-steps enables cli users fine grained control over the deployment process and combinations of these sub-commands can be utilized to create custom deployment and app management workflows as required or desired within your organization.

You'll now use a few cf push sub-commands to rollback to the previous stable version of the app.

Step A) View all the droplets associated with your app

- `cf droplets squeeze`
  You'll see all the droplets we've pushed previously
  The one labeled current is the droplet that is currently deployed on Cloud Foundry.
  Now we'll roll back to previous droplet to revert back to our previous, successful, lemonade recipe.

  Copy the previous droplet GUID into your buffer.

Step B) `cf set-droplet squeeze [YOUR COPIED_DROPLET_GUID]`

- For example
  `cf set-droplet 257f3420-9b09-4d2b-9026-5d1b663cd5c6`

Step C) Restart the app to roll back to the previous droplet

`cf restart squeeze --strategy rolling`

- Observe Squeeze reverts to it's previous original recipe with no downtime.

Thankfully, their botched lemunique product offering was only available to the public for a short few minutes rather than hours or days and all three Squeeze founders have collapsed with relief to their beanbags. Finally, there's a moment of respite. Each reflecting upon their whirlwind experience.

You've now utilized many of the key new features available in the v7 cf CLI.
If you've got any questions, suggestions, recommendations, or requests, please feel free to reach out to us via the #cli slack channel or on GitHub!

Thank you and HAPPY CODING!

The cf CLI Team