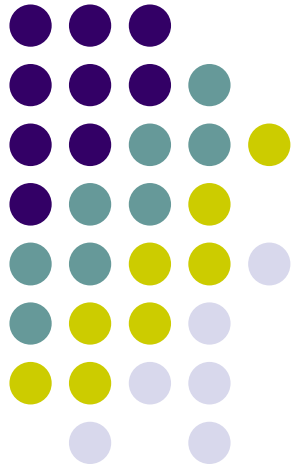
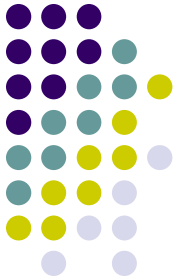
Big Data Application Architecture

November 28, 2020

Mike Spertus

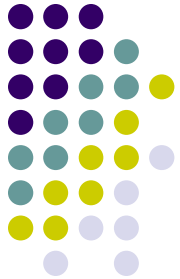
mike@spertus.com





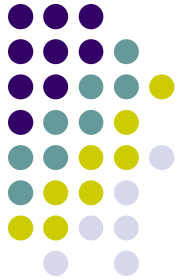
REMOTE DEBUGGING AND DEPLOYMENT OF WEB APPLICATIONS

Remote Development



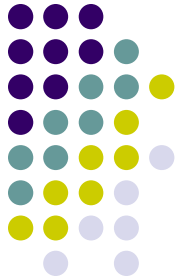
- Our Kafka cluster is not externally visible, so we cannot locally develop and test the parts of our web app that depend on Kafka
- While this is annoying, it is also very typical that Big Data clusters avoid exposing too many services publicly
- So it is probably good to deal with

Single Webserver



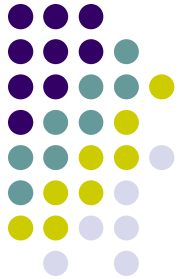
- Our solution to this is to have a development webserver that you can ssh into
  - Note that it would be bad to allow ssh access to our load-balanced production webservers, which should only be managed by a Continuous Deployment tool like CodeDeploy

Using a second port



- You can only run one app on a given port at a given time
- We have a quota of 150 listeners on our web balancer, so each student can have two ports of their own
  - You can use the second port to run and compare both your project and the flight_and_weather app at the same time
  - Alternatively, you can use the second port for working on enhancements to your project while leaving the existing one deployed
- If you need a third port, please let me know

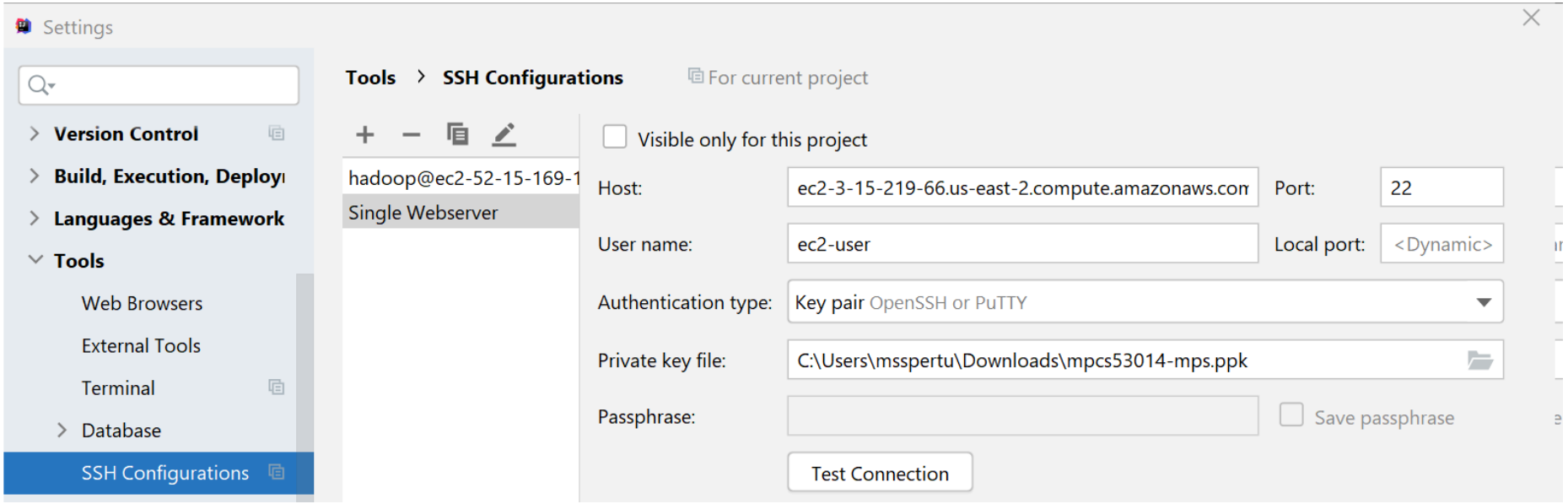
Editing in IntelliJ, Running in Single Webserver



- A good way to develop is to edit your node project in Hive, and then deploy it to Single Webserver
- This has a few differences from our past experiences with IntelliJ deployment, so let’s walk through this step-by-step

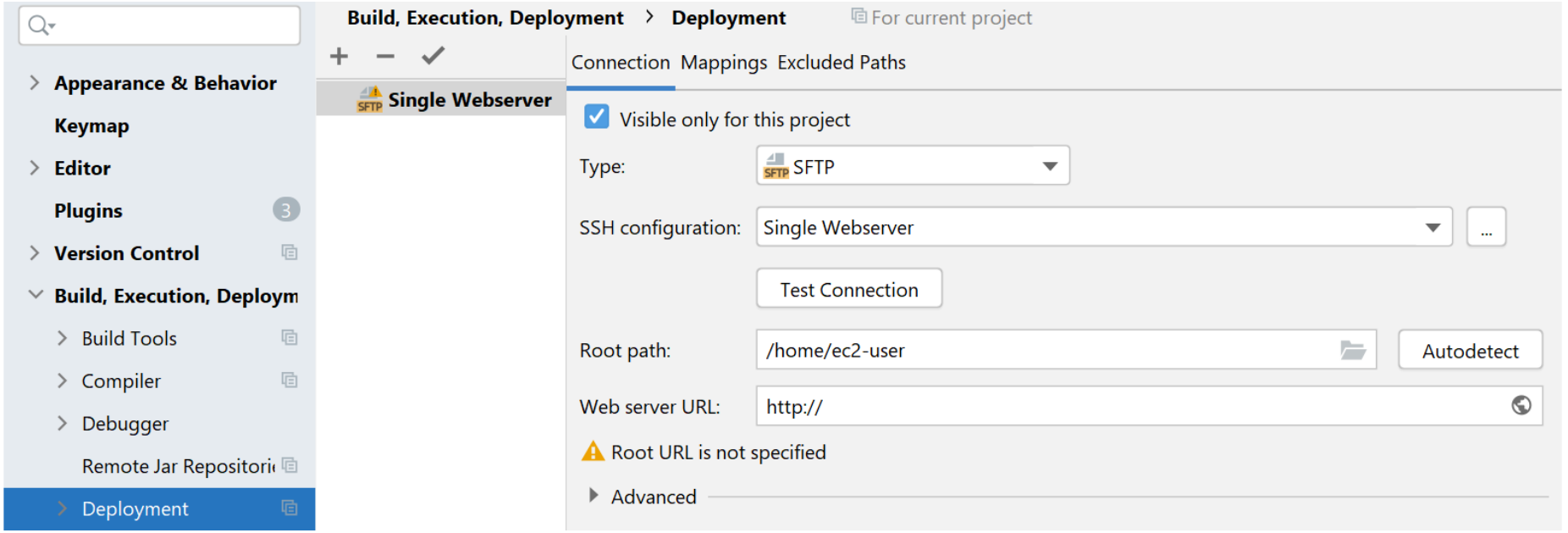
Create an ssh configuration for Single Webserver

- File/Settings/Tools/SSH Configurations
- This is similar to the one you created for our name node, only with a different host and username

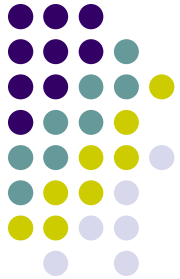


Create a Deployment for your web application

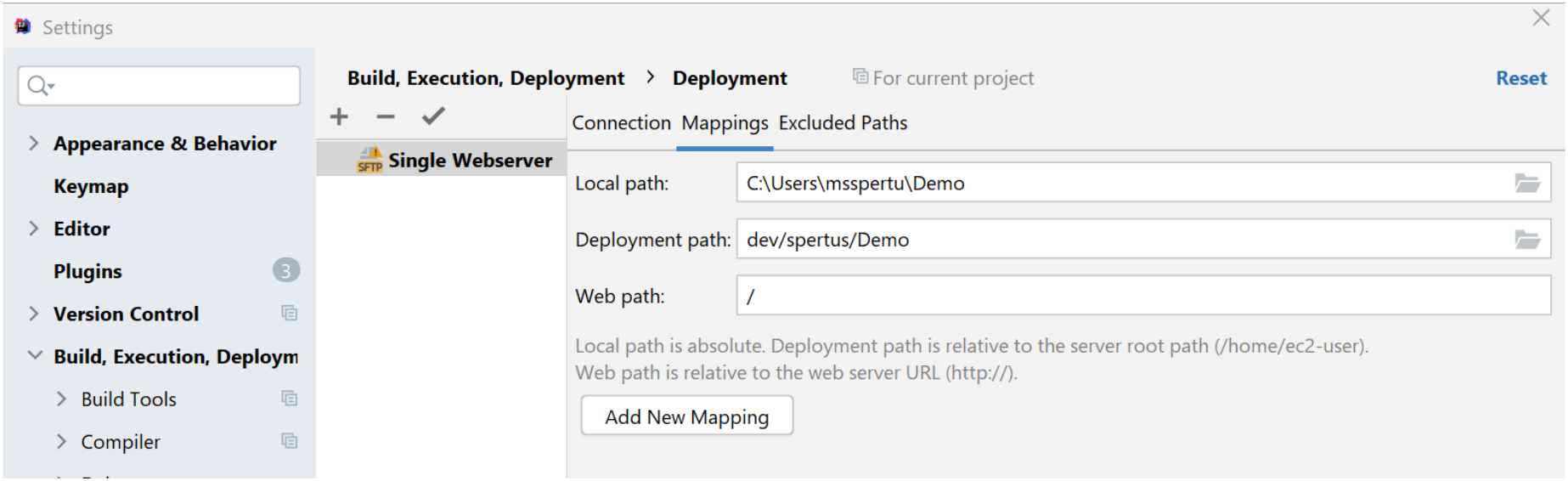
- With your Node project open in IntelliJ, choose File/Settings/Build, Execution, Deployment/Deployment
- Create a new Connection using the SSH Configuration you just created



Setup the mappings for the deployment

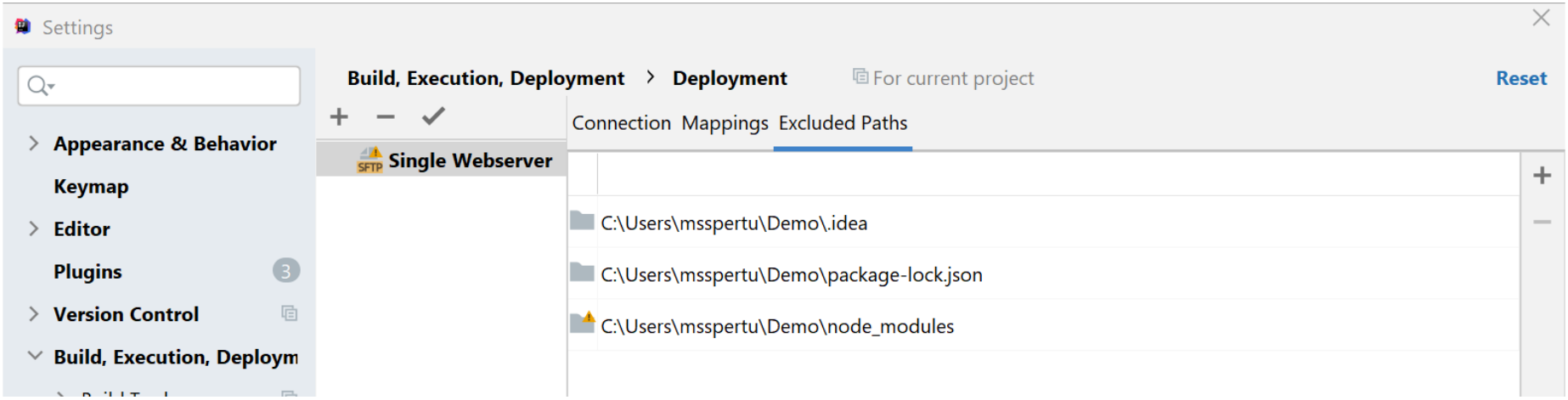


- Since CodeDeploy gave /home/ec2-user/*cnetid* to root, use dev/*cnetid*/*project* in the Deployment's Mapping tab



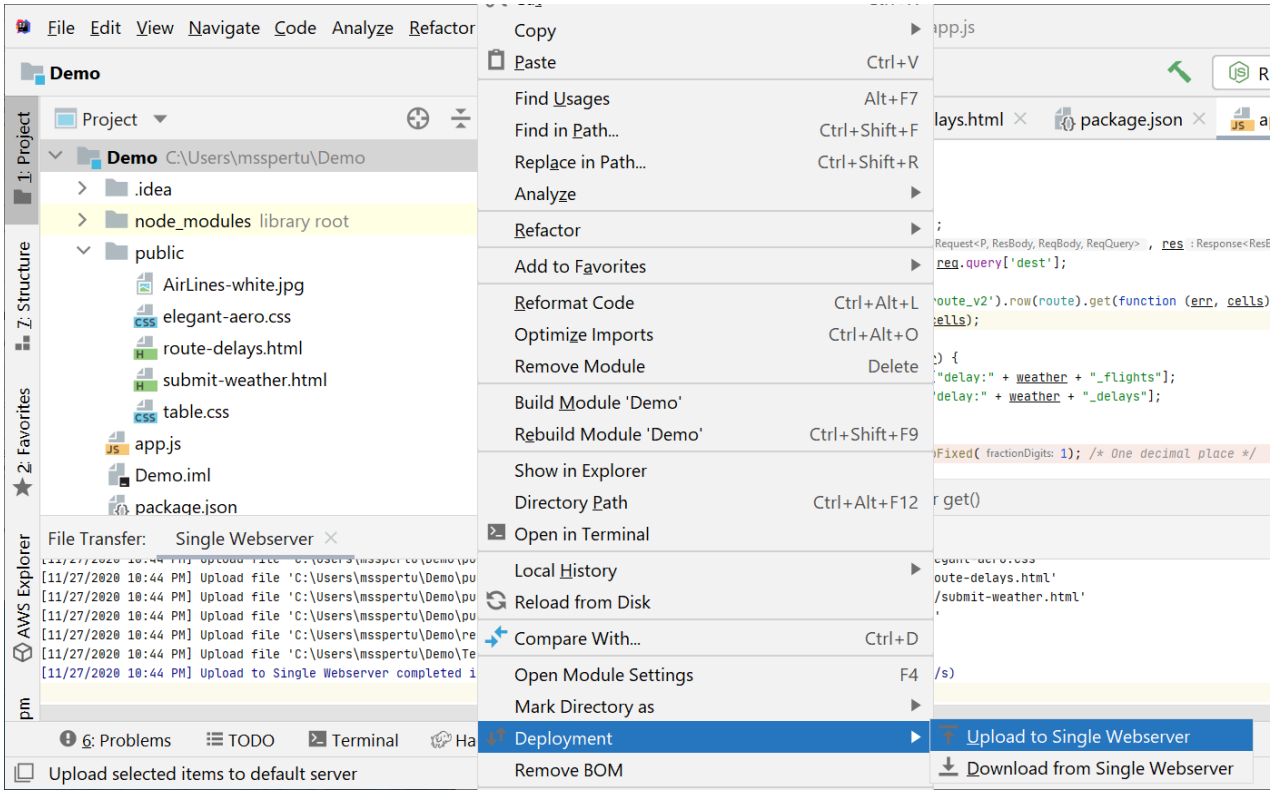
Excluded Paths

- We will not want to upload everything to Single Webserver
  - E.g., not only will your node_modules directory be huge, it may need to be compiled differently for Single Webserver than your laptop
- In the Deployment's Excluded Paths tab, add Local Paths as below



Deploy to Single Webserver

- When you are ready to test out your application, right click on the root folder in IntelliJ and Upload to the Deployment you created



SSH into Single Webserver

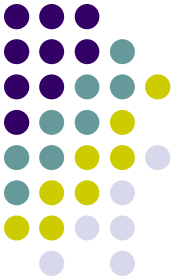
- Log into Single Webserver
  - Either through putty/ssh like with our name node
  - Or by Connecting through the Connect button in the Single Webserver Instance page in the EC2 Console
- Go to the deployment directory and install your dependencies with
  `npm install`

```
ec2-user@ip-172-31-1-18:~/dev/spertus/Demo
Using username "ec2-user".
Authenticating with public key "mpcs53014-mps"
Last login: Sat Nov 28 18:48:58 2020 from ec2-3-16-146-0.us-east-2.compute.amazo
naws.com

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
12 package(s) needed for security, out of 17 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-1-18 ~]$ cd dev/spertus/Demo/
[ec2-user@ip-172-31-1-18 Demo]$ ls
app.js  Demo.iml  package.json  public  result.mustache  Test.iml
[ec2-user@ip-172-31-1-18 Demo]$ npm install
```

Stop any other application running on the same port



- If you have another application running on the same port, you will need to stop it
- If you deployed it w/CodeDeploy, it will be managed by `forever`, so identify and stop as follows

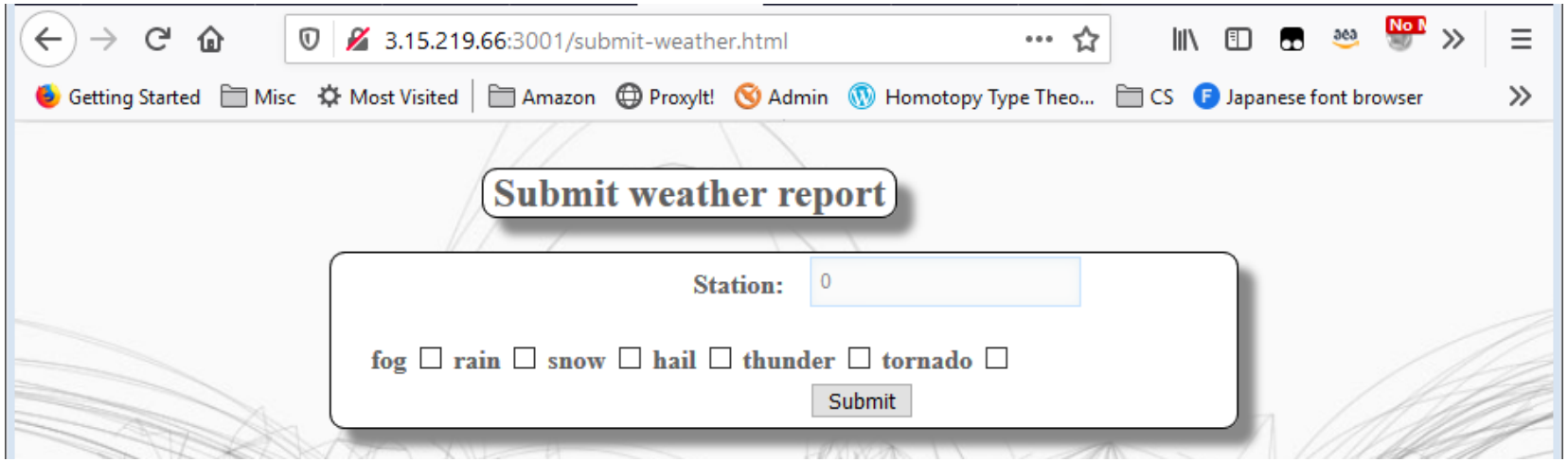
```
ec2-user@ip-172-31-1-18:~/dev/spertus/Demo
[ec2-user@ip-172-31-1-18 Demo]$ # Check for apps by me on port 3001
[ec2-user@ip-172-31-1-18 Demo]$ forever list | grep mspertus | grep 3001
(node:17601) Warning: Accessing non-existent property 'padLevels' of module exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:17601) Warning: Accessing non-existent property 'padLevels' of module exports inside circular dependency
data:    [74] gGhU /usr/bin/node app.js 3001 ip-172-31-11-144.us-east-2.compute.internal 8070 b-2.mpcs53014-kafk
a.fwx2ly.c4.kafka.us-east-2.amazonaws.com:9092,b-1.mpcs53014-kafka.fwx2ly.c4.kafka.us-east-2.amazonaws.com:9092
16171   16178 mspertus_flight_and_weather        /home/ec2-user/.forever/gGhU.log 0:18:7:26.214999999996508
[ec2-user@ip-172-31-1-18 Demo]$ forever stop mspertus_flight_and_weather
(node:17621) Warning: Accessing non-existent property 'padLevels' of module exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:17621) Warning: Accessing non-existent property 'padLevels' of module exports inside circular dependency
info:    Forever stopped process:
    uid  command       script
                                                                                                    forever pi
d   id                          logfile                                uptime
[0] gGhU /usr/bin/node app.js 3001 ip-172-31-11-144.us-east-2.compute.internal 8070 b-2.mpcs53014-kafka.fwx2ly.c
4.kafka.us-east-2.amazonaws.com:9092,b-1.mpcs53014-kafka.fwx2ly.c4.kafka.us-east-2.amazonaws.com:9092 16171   16
178 mspertus_flight_and_weather /home/ec2-user/.forever/gGhU.log 0:18:7:48.908000000003085
[ec2-user@ip-172-31-1-18 Demo]$ # Confirm it's gone
[ec2-user@ip-172-31-1-18 Demo]$ forever list | grep mspertus | grep 3001
(node:17643) Warning: Accessing non-existent property 'padLevels' of module exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:17643) Warning: Accessing non-existent property 'padLevels' of module exports inside circular dependency
[ec2-user@ip-172-31-1-18 Demo]$
```

Now run your app with Node

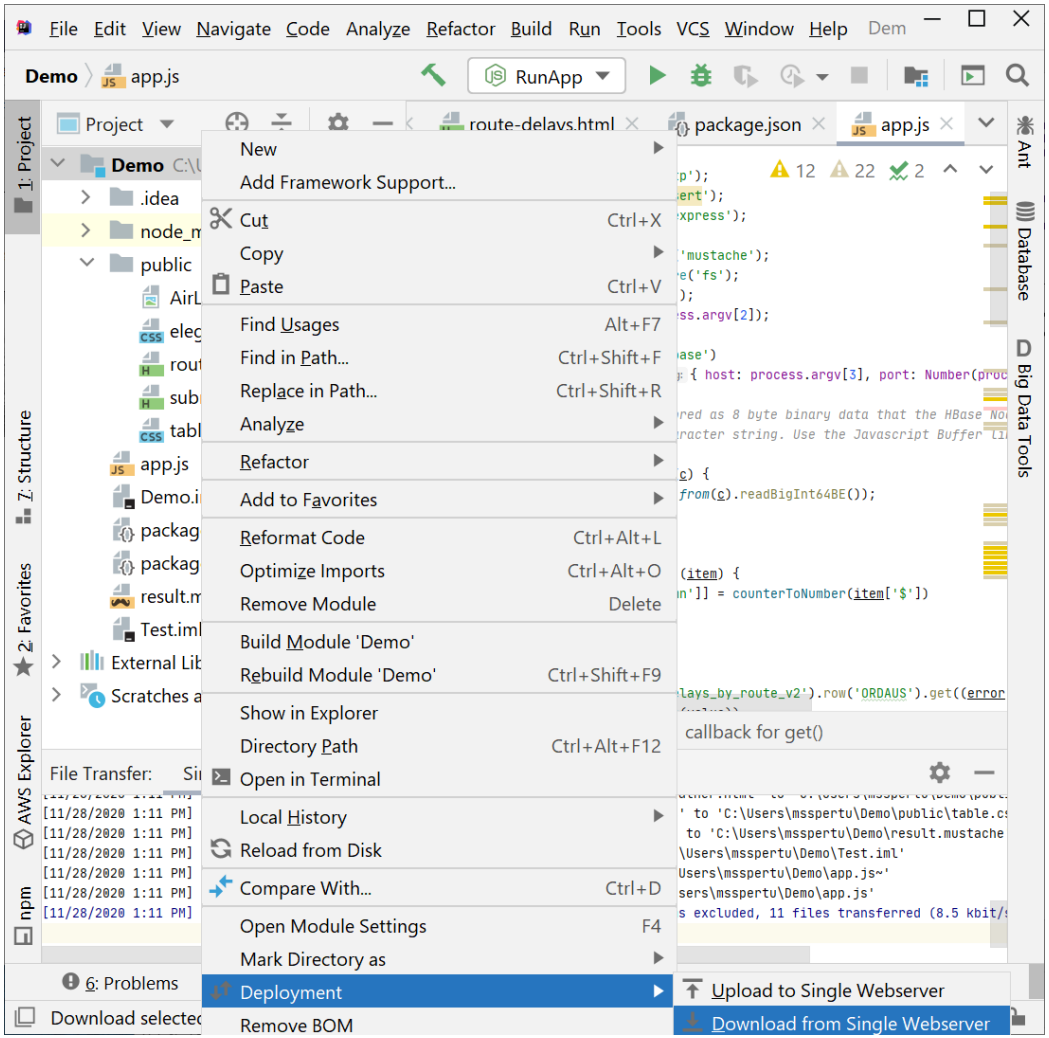
- Run it with node for easy debugging
  - As opposed to a “service” mode like forever

```
[ec2-user@ip-172-31-1-18 Demo]$
[ec2-user@ip-172-31-1-18 Demo]$  node app.js 3001 ip-172-31-11-144.us-east-2.compute.internal 8070 b-2.mpcs53014
-kafka.fwx2ly.c4.kafka.us-east-2.amazonaws.com:9092,b-1.mpcs53014-kafka.fwx2ly.c4.kafka.us-east-2.amazonaws.com:
9092
```

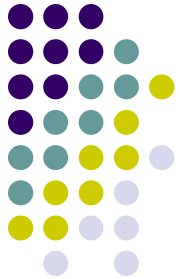
- And test with your app



If you make changes, you can download back into IntelliJ

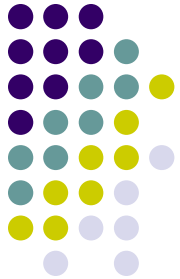


When you're ready to deploy

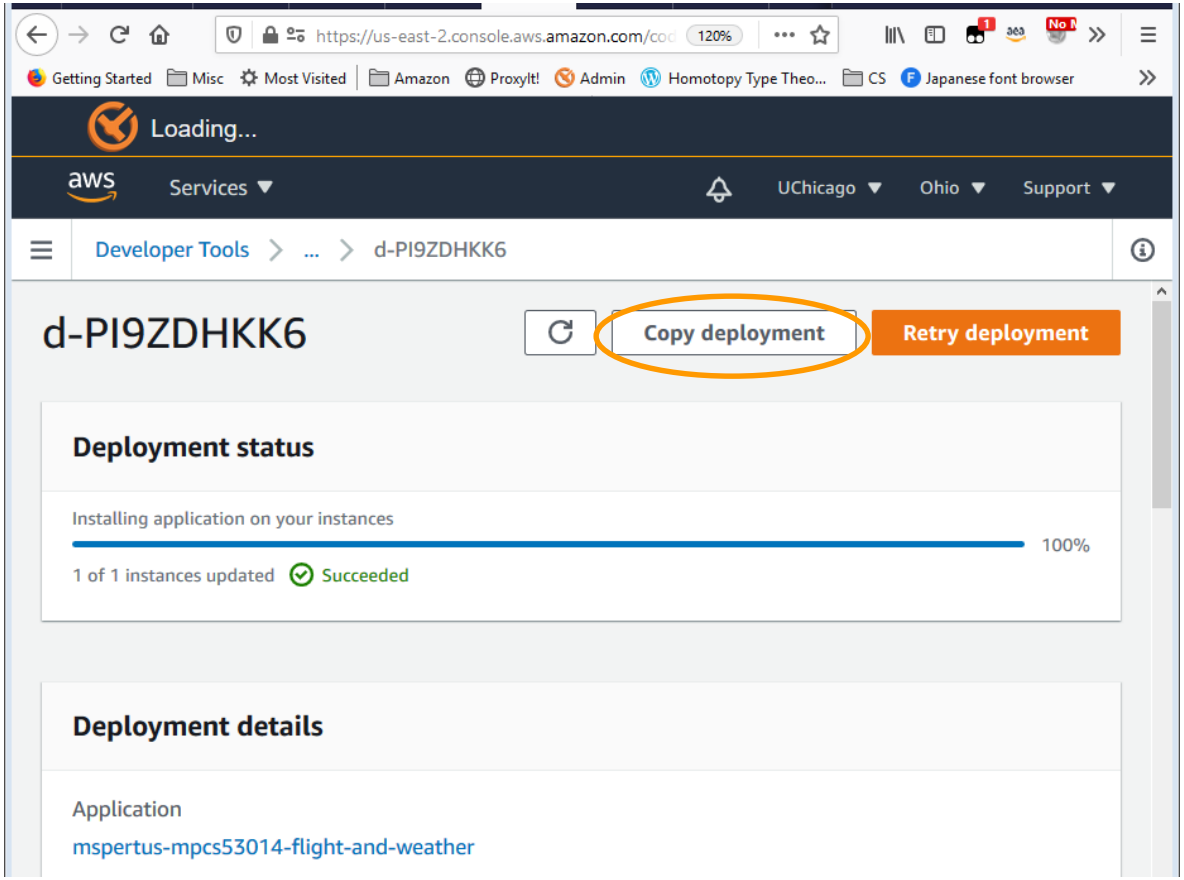


- You will want to test and debug your deployment on Single Webserver
- Stop your node process (ctrl-C) if it's still running in your terminal
- Create your deployment archive and upload to S3 as usual
- Now use CodeDeploy
  - but not quite as usual

CodeDeploy to Single Webserver

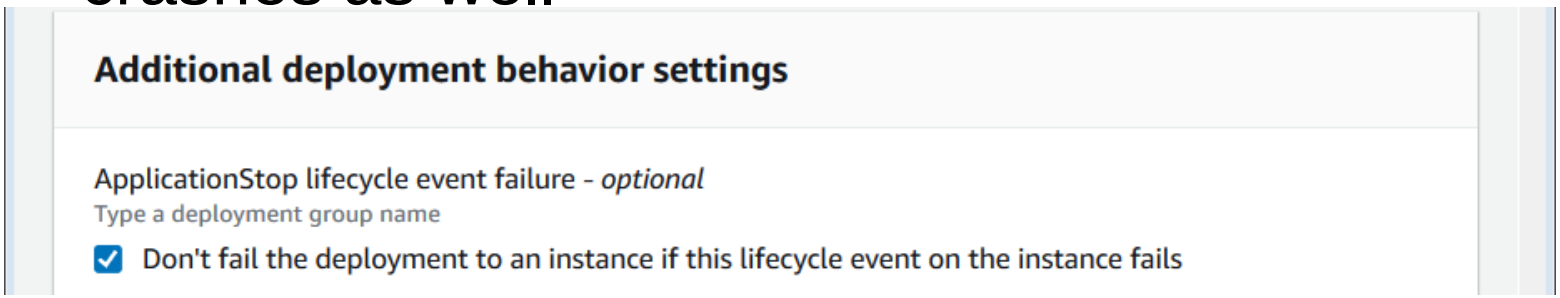


- Copy one of your app's existing deployments



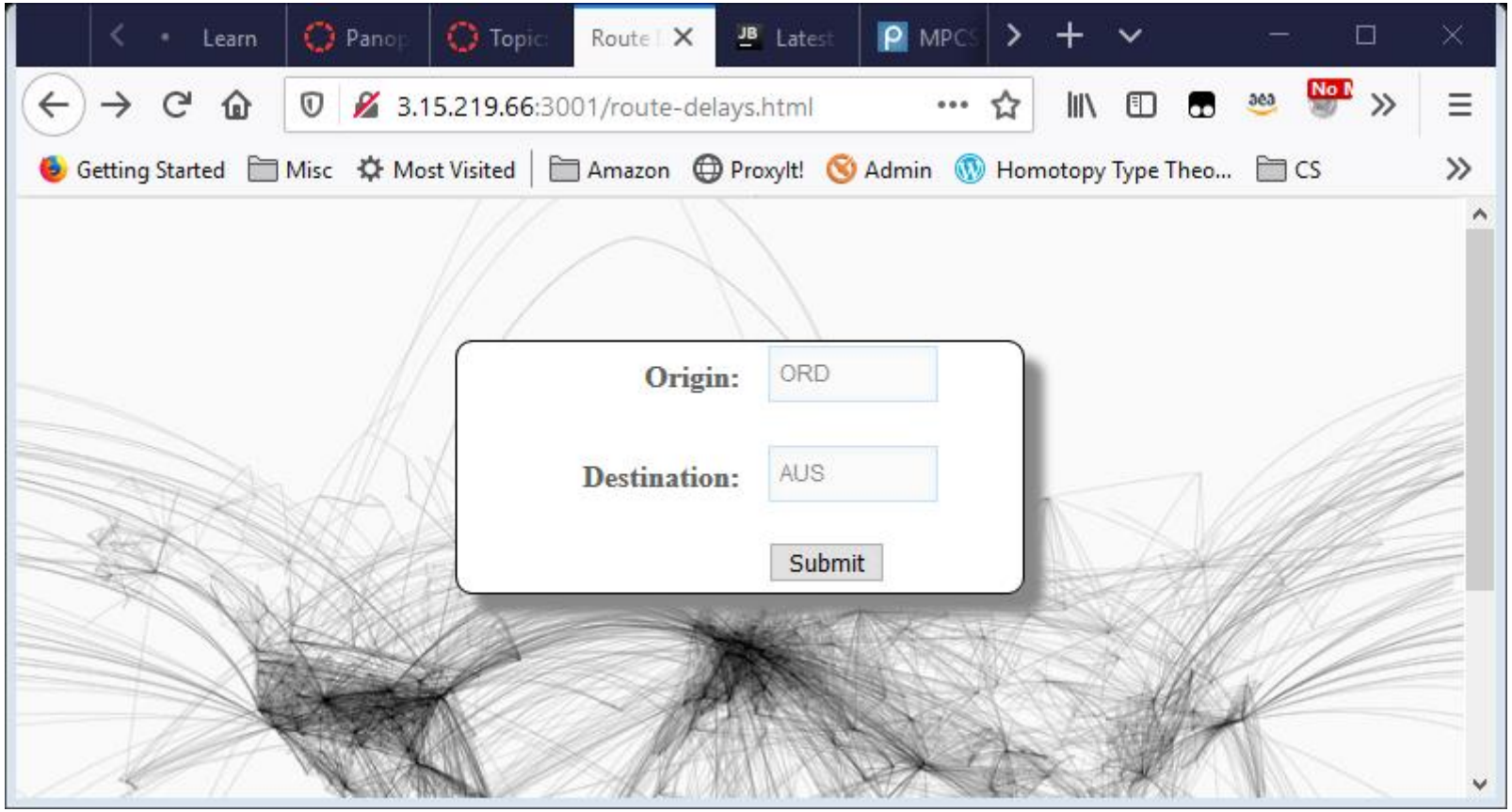
Deployment settings

- Confirm all the usual settings (S3 URL, QuickDeploy
- Since we manually killed the previous deployment earlier, tell it not to worry if it fails to kill the previous deployment
  - This setting often comes in handy if an application crashes as well

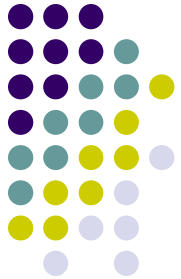


Deploy and test

- Hit Create Deployment and check if it works

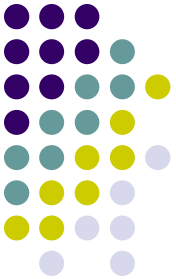


When ready, deploy to our production cluster

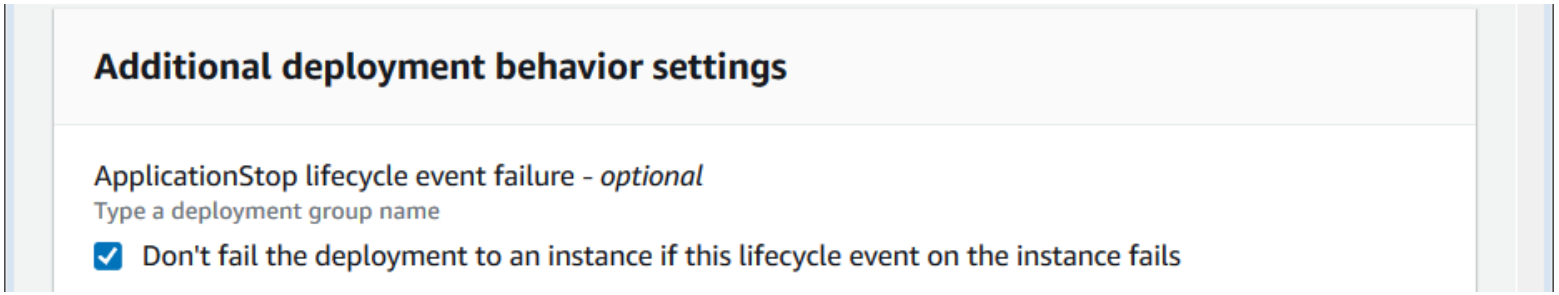


- Once you are ready to deploy, use CodeDeploy to deploy to our production cluster
- You can do this as before, but the following slides show some possible refinements

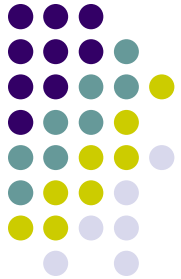
Tip: Deploy even if ApplicationStop fails



- As mentioned above, if there is not a previous app running, you may need to tell CodeDeploy that is OK



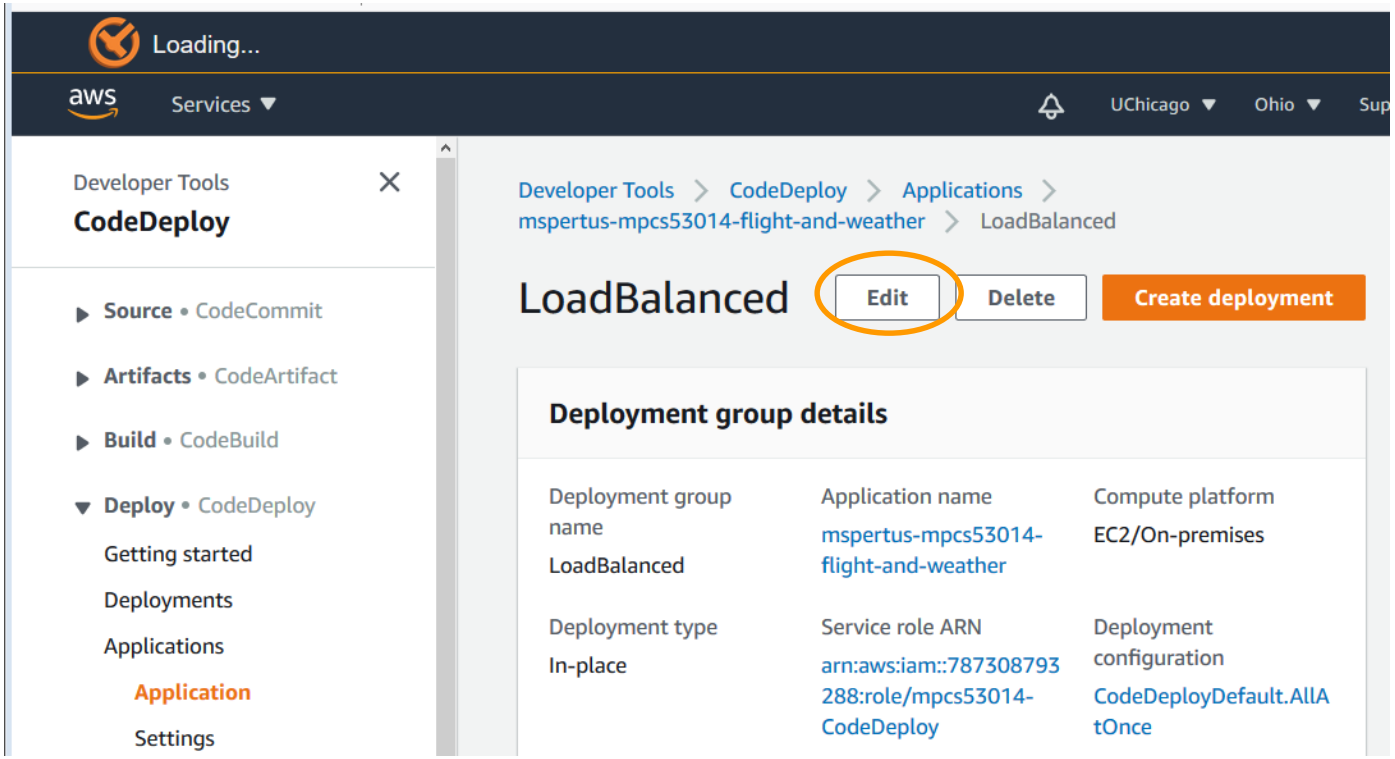
Tip: Don’t tell CodeDeploy about our load balancer



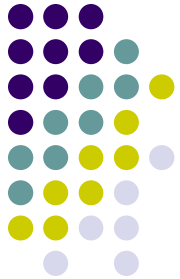
- While our LoadBalanced deployment group uses a Load Balancer, there’s no reason for CodeDeploy to worry about it
  - We leave it running all the time and it’s shared
  - This isn’t a real production app, so we aren’t worrying about letting in process transactions drain
  - Cycling the load balancer can interfere with other students’ apps
  - Draining and restarting the load balance can add ten minutes to the deploy process
  - The constant cycling may contribute to some of the erratic Load Balancer behavior we’ve experienced

Editing your LoadBalanced deployment group

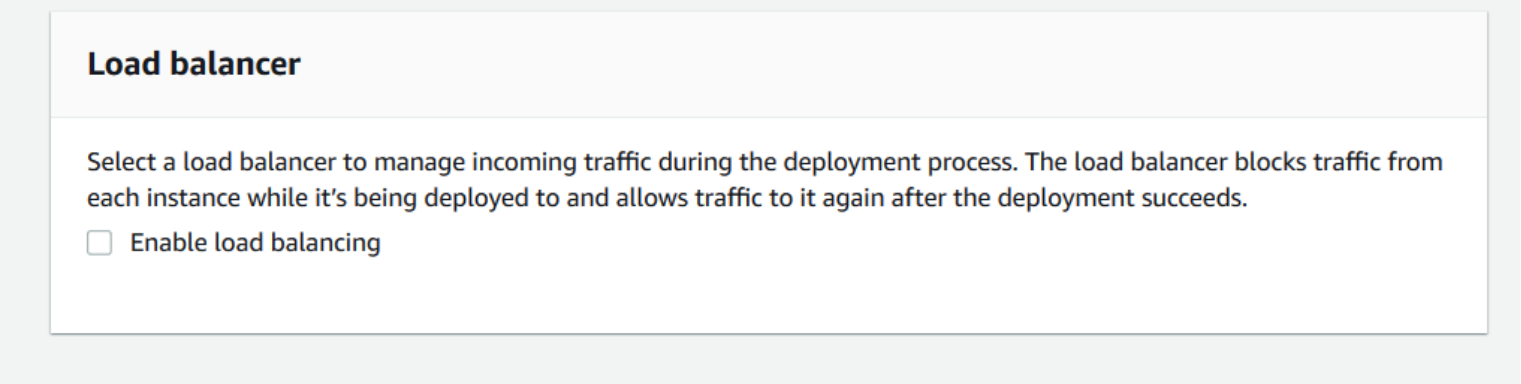
- Based on the above, it may be helpful to edit your application's LoadBalanced Deployment Group



Clear “Enable load balancing”

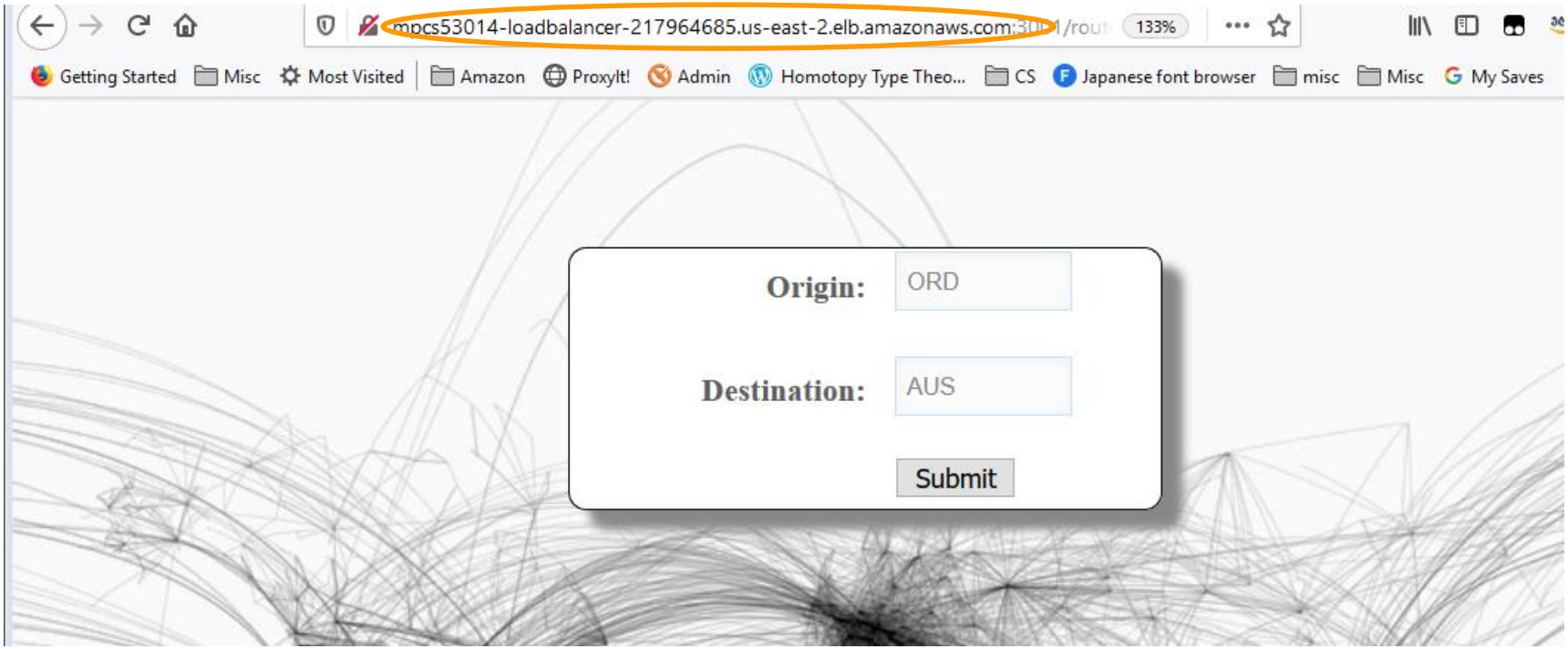


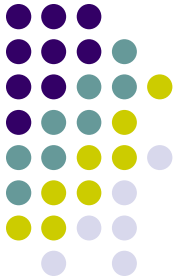
- Note that your deployment group will still be behind a load balancer
  - But we’re keeping that a secret from CodeDeploy!



Deploy and test

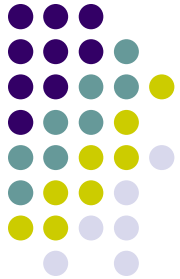
- Your deployment should run quickly now
- If it succeeds, be sure to test





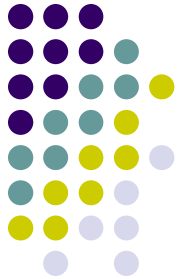
SOME MORE DEPLOYMENT FAQS

My deployment succeeded, but it is still running the old app



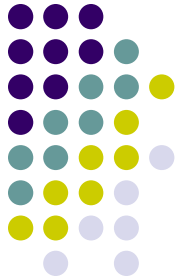
- If you have multiple applications sharing the same port, it is likely that you will have to stop the old application before the new one can bind to the port
- One workaround is to use a different port for each application
  - Remember you can choose two
- Or you can shutdown the previous app
  - Instructions on next slide

How do I stop a deployed application?

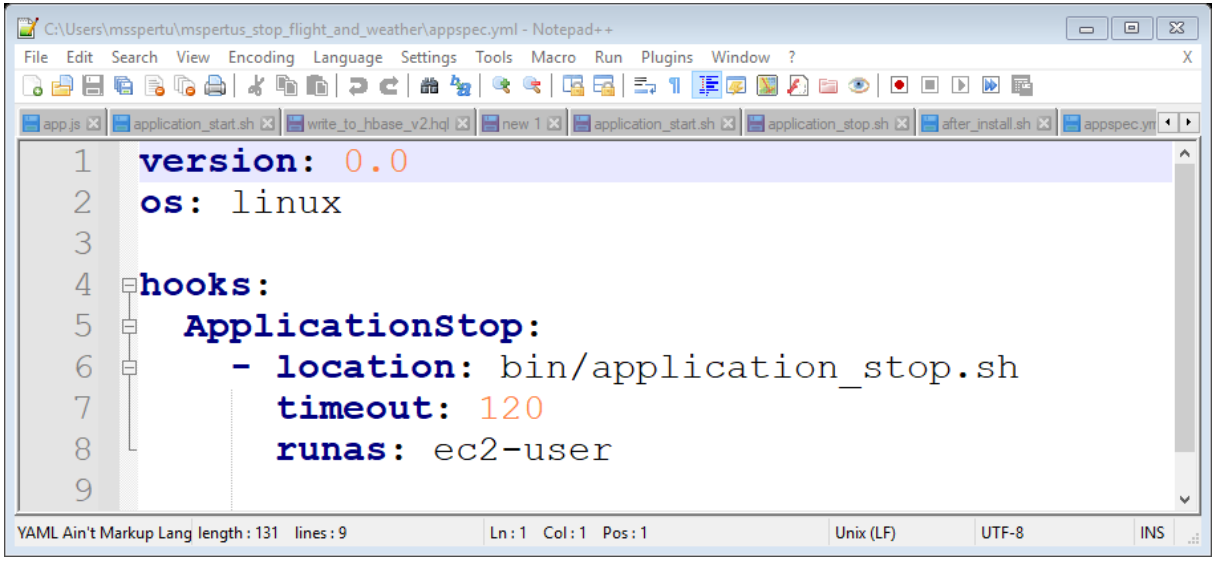


- If it’s just on Single Webserver, you can just do “forever stop” like we did above
- However, if it’s deployed to a fleet of LoadBalancers, you will want to use CodeDeploy

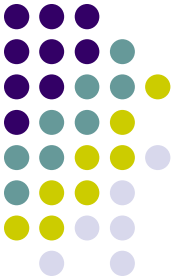
Stopping an application with CodeDeploy



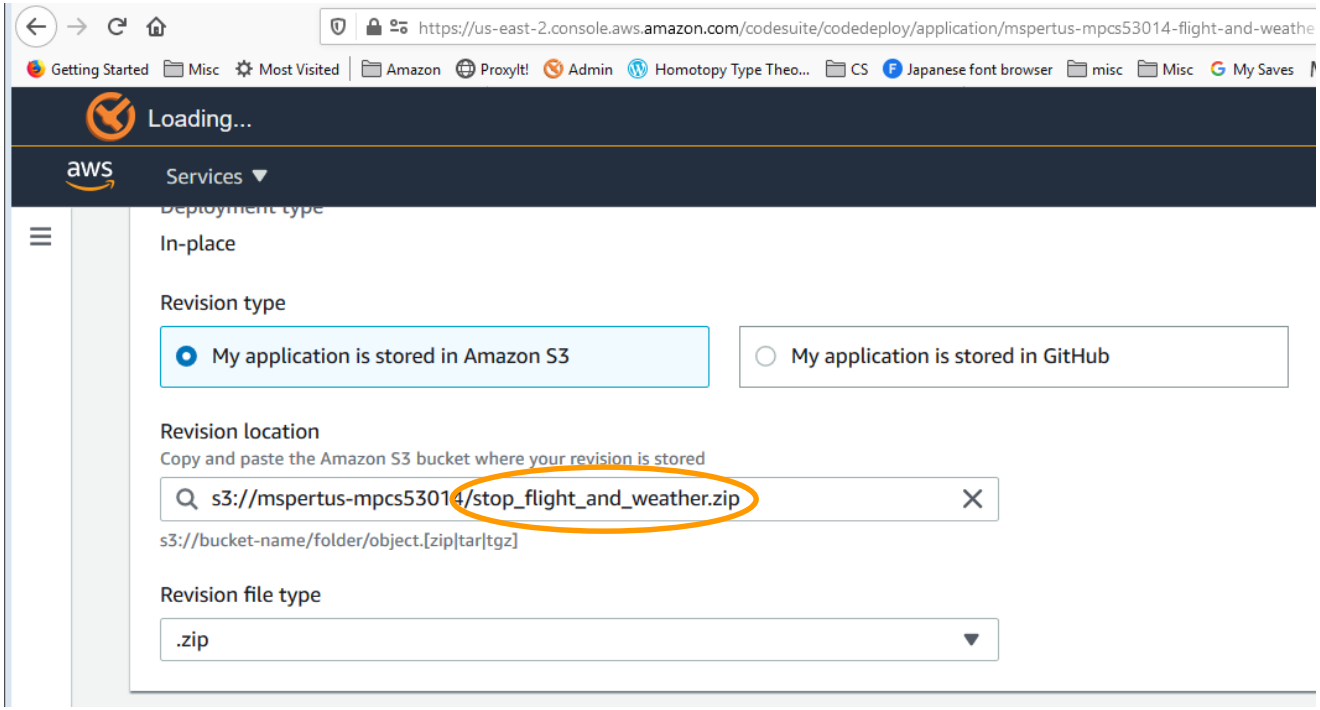
- I create a modified version of my deployment archive named stop_flight_and_weather.zip
- The only change is that the appspec.yml consists only of an ApplicationStop hook



Stopping an application with CodeDeploy (cont)



- Now deploy the “stop application” archive



Now Deploy your new App

- Deploy the new app as normal, but be sure to tell it not to fail if it can’t stop the previous deployment