Automation

What we cover

- Introduction to DevOps
- Key DevOps concepts
- Introduction to popular CI Tools
- Reading for practical experience

NO SCAFFOLDING
NO WORRIES
ifunny.co

Initial Questions

- How many changes do you think you will make to your product?
- How will you make sure your changes have not broken the product?
- Who will deploy and test?
- Is it fun/best use of your time?

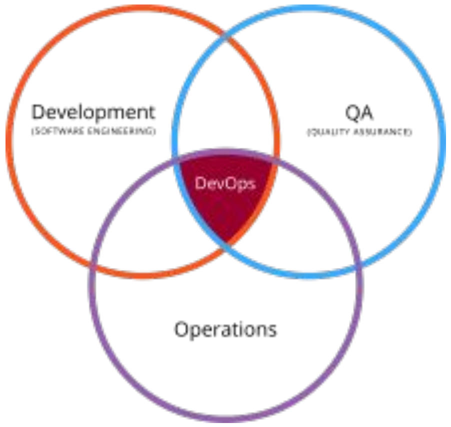
Late 1990s to Early 2000s

- **Testing**
  - Mostly manual
  - Performed by a separate QA team
- **Deployment & Operations**
  - Ship CD/DVD to customer, or (later) provide download link
  - Customer owns production

Late 2000s to Early 2010s

- **Testing**
  - Partly automated (esp. unit tests)
  - QA team for manual parts
- **Deployment & Operations**
  - Software as a Service
  - Deploy to in-house hardware, or third party hosted hardware
  - Vendor's ops team owns prod

Late 2010s & 2020s (Mature Cloud)



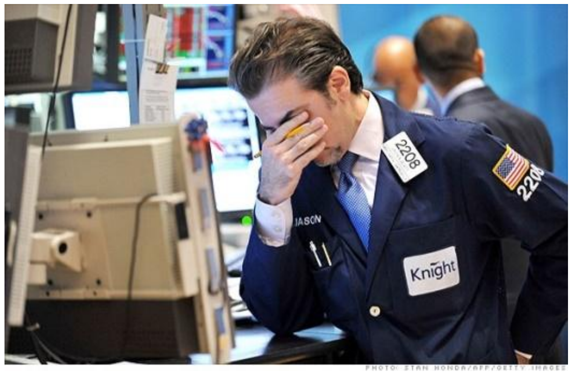
- **Testing**
  - Fully automated
  - By dev team and/or SDETs
- **Deployment & Operations**
  - Hardware is virtual, like software
  - Infrastructure as Code (IaC)
  - Development+Operations = DevOps

1999: Mars Climate Orbiter

- Bad software can **destroy hardware**:
  - **Purpose of probe**:
    - Study the climate, atmosphere and surface changes on Mars
  - **Cost**:
    - $125 million
  - **Problem**:
    - Ground system sent commands in imperial (lbf*s)
    - Orbiter expected metric (N*s)
  - **Result**:
    - Failed orbital insertion: probe destroyed
  - **Cause:**
    - Poor integration testing

2012: Knight Capital

- Bad software can **destroy financial assets**:
    - **Knight Capital's business:**
        - Equity order routing & algorithmic trading
    - **Problem**:
        - New equity order routing code was deployed to 7 servers
        - Unfortunately, Knight had <u>8</u> production servers
    - **Result**:
        - **Knight lost $445M** and nearly went bankrupt
    - **Cause:**
        - errors during <u>manual</u> deployment process



THE PRESENT DAY

User Expectations

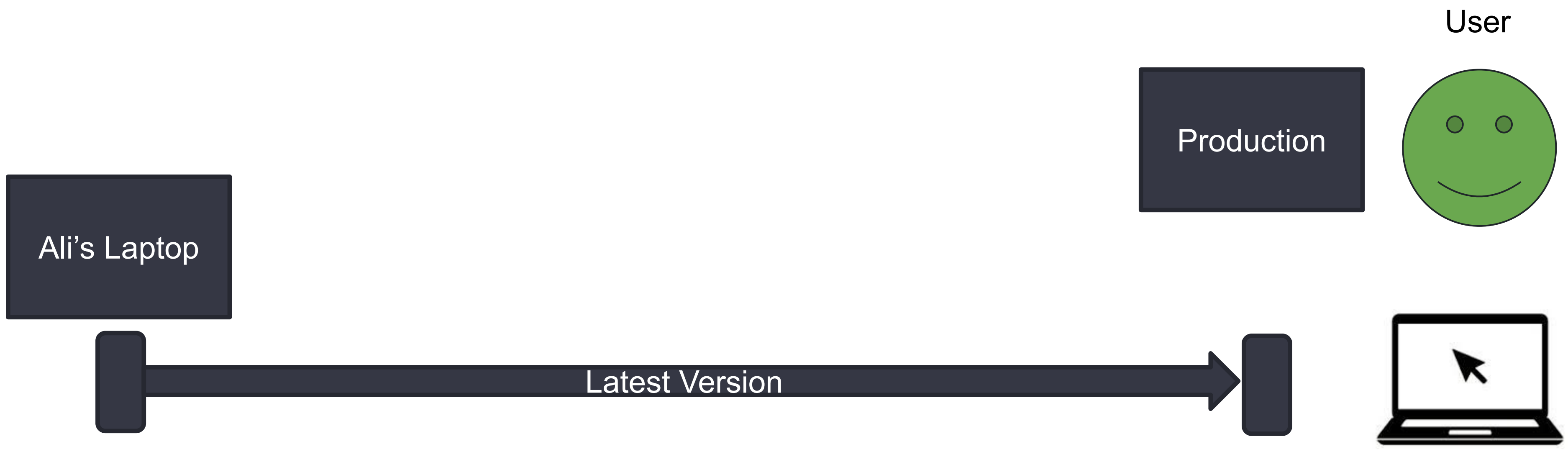
- Users expect:
  - No installation
  - No responsibility for production operations
    - And no production downtime
  - Rapid feature delivery
  - High quality: few bugs... ideally none

Software Delivery Maturity

- Lead time
- Release frequency
- Downtime

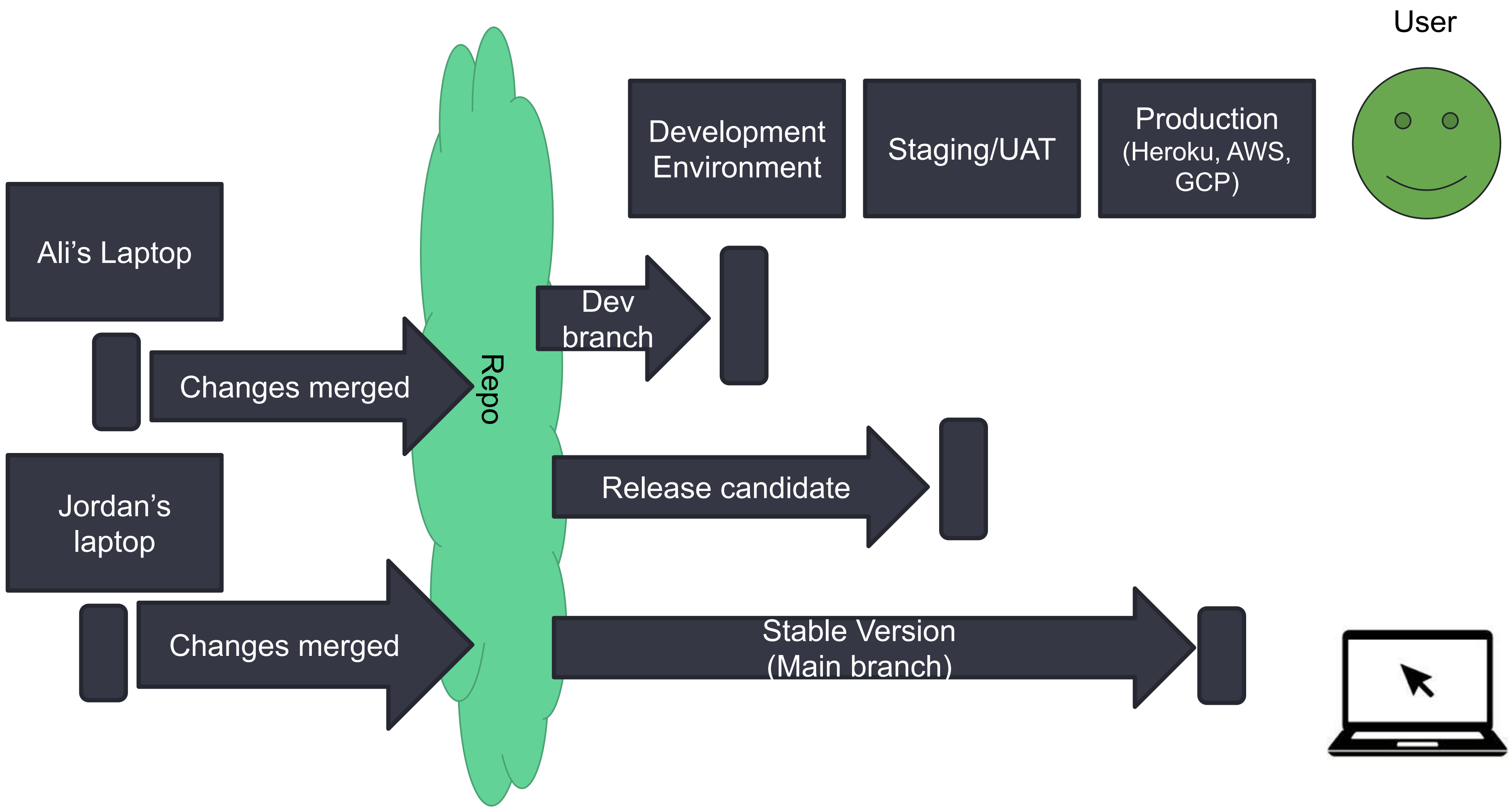
How does your user get the latest version of your application?

Simple model



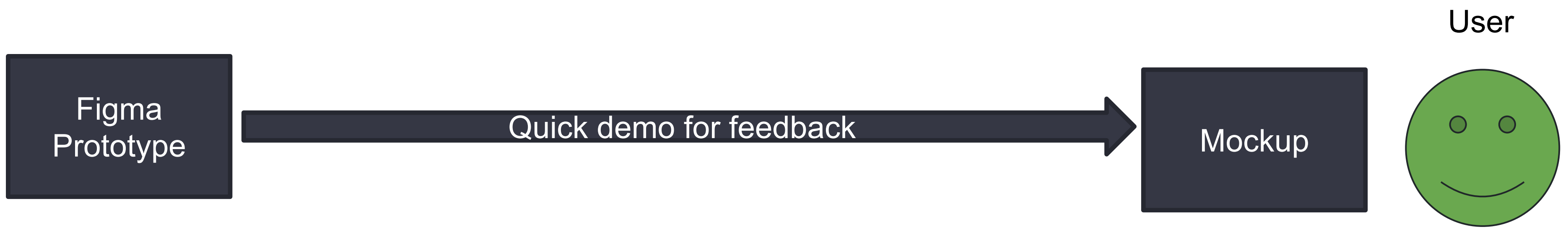
What problems do you see in this model?

Typical production development workflow



What problems do you see in this model?

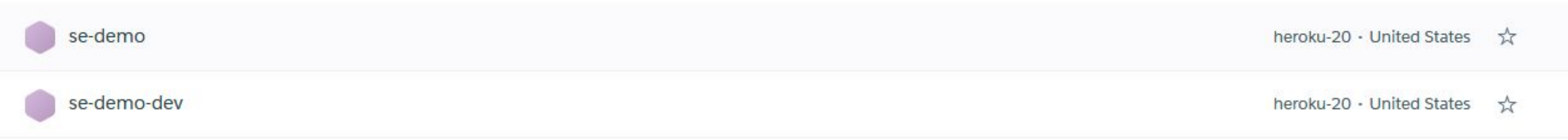
Quick Prototyping



How to make this happen?

1. Setup a dev environment
2. Setup your automation to deploy to the dev environment on merge to dev
3. Setup a staging environment (may be able to skip for csc301 project)
4. Setup your automation to deploy to the staging environment
5. Setup a prod environment (you can share with users)
6. Setup your automation to deploy to prod when merged to main

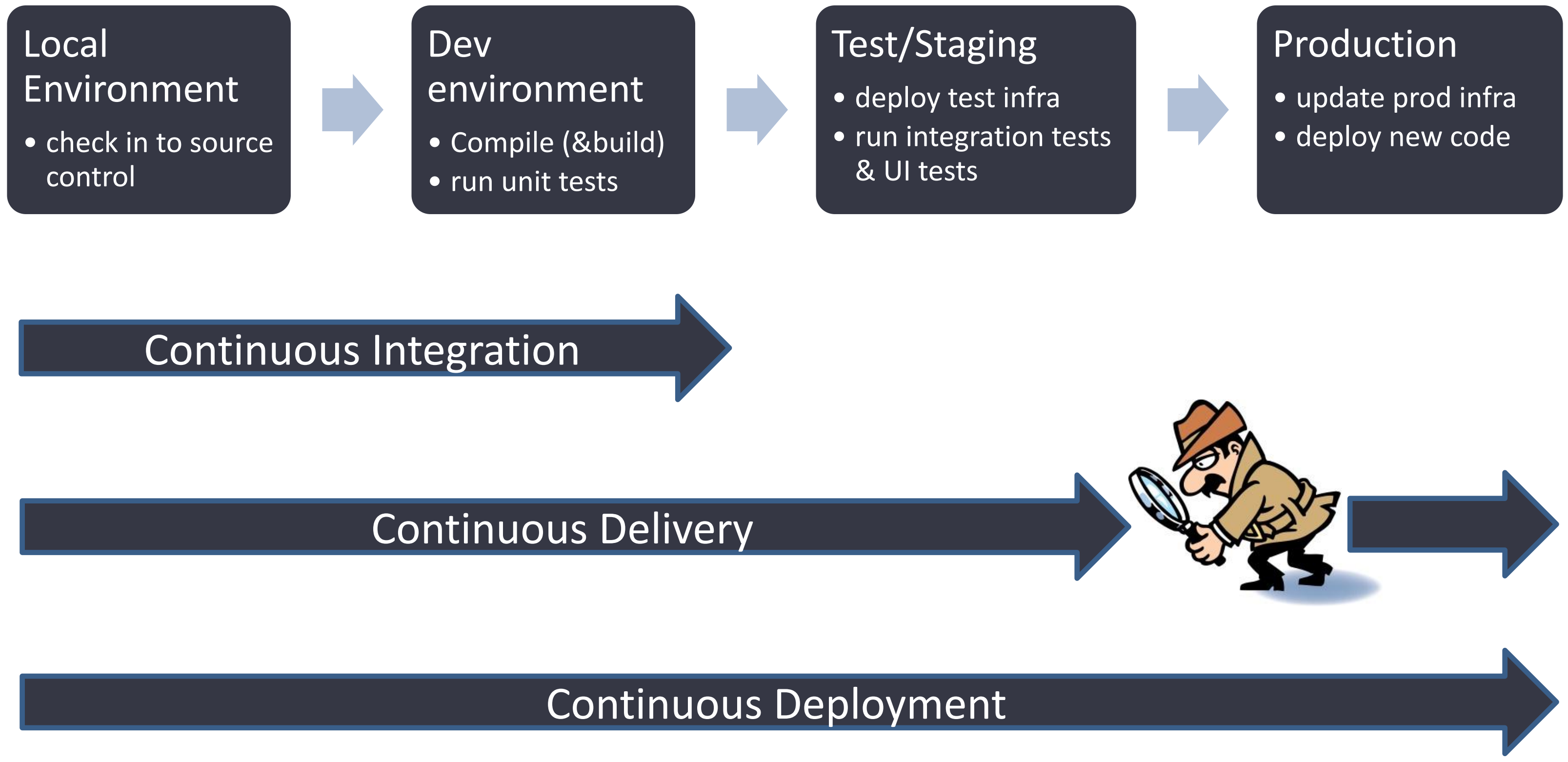
Let’s see an example on Heroku



se-demo heroku-20 · United States

se-demo-dev heroku-20 · United States

How Code Gets to Production

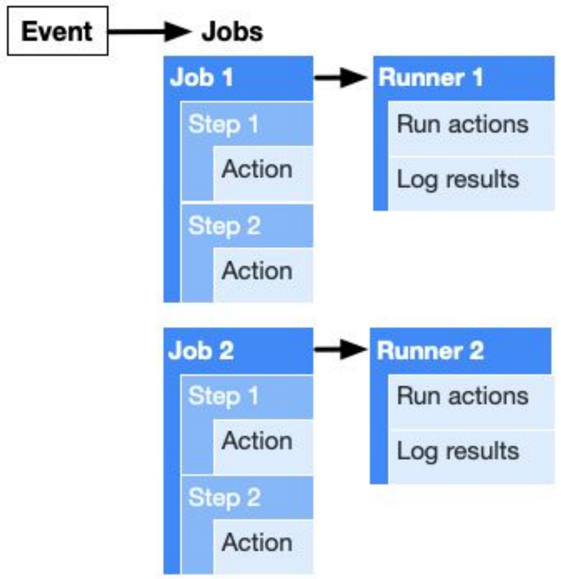


How do we know what gets moved from one environment to another?

Concepts you need to know

- Runners: virtual operating system/processor
- Events: e.g., push, merge, etc.
- Workflows -> Jobs-> Steps-> Actions
  - Instructions specified in YAML files in *.github/workflows*
- `Each workflow contains a job`
  - `Each job runs steps`
    - `Each step have actions`

- `Workflow Syntax for GitHub Actions`



How do we automate?

- The key is to replicate the same software on demand
- Identify dependencies and track your project metadata
  - Packages & libraries are specified in a file
    - Package.json for Node.js
      - Heroku starting example
    - Requirements.txt for Python
      - Heroku starting example
- Use package managers to maintain your packages
  - Pip for Python
  - Npm for Node
- Your CI/CD tool will use the same package managers to build the same environments

AWS CodeDeploy circleci GitHub Actions

How to configure your CI/CD

1. Continuously write tests (we’ll talk more about it)
2. Choose your CI/CD tool and configure
   a. GitHub Actions
      i. QuickStart
   b. CircleCI
3. Choose your server/deployment details. E.g., Heroku, AWS

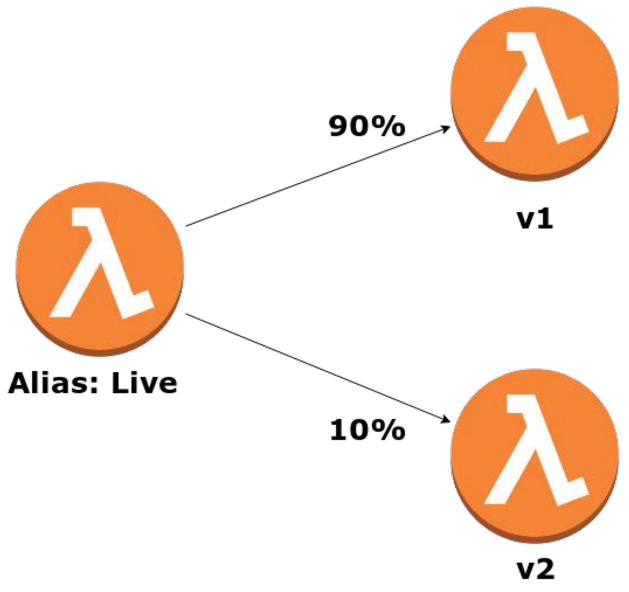
NOTE: If you use GitHub Actions, you must add a time limit to your jobs because CI tools provide limited minutes. You may be penalized if you don’t.

Sample package.json

```
{
  "main": "node_modules/expo/AppEntry.js",
  "scripts": {
        "start": "expo start",
        "android": "expo start --android",
        "ios": "expo start --ios",
        "web": "expo start --web",
        "eject": "expo eject",
        "test": "jest --watchAll"
  },
  "jest": {
        "preset": "jest-expo"
  },
  "dependencies": {
        "@expo/vector-icons": "~10.0.0",
  },
  "devDependencies": {
        "@babel/core": "^7.0.0",
        "babel-preset-expo": "~8.0.0",
        "jest-expo": "~36.0.1"
  },
  "private": true
}
```

Finally... Testing in Production?

- It is important to be able to revert a bad build quickly, cleanly and as automatically as possible
- **Canary Deployments** and/or **Traffic Shifting** are helpful techniques
  - Can prevent major problems early on
- Initially, direct only a <u>subset</u> of users/requests to the new code



Production Monitoring

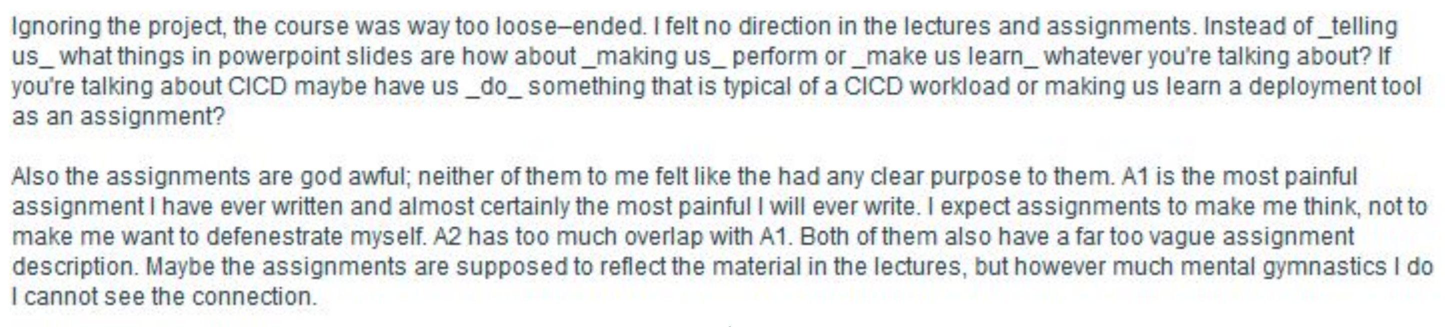
- So, you are the proud owner of a SaaS application in production… now what?
  - Is the application responsive?
  - Is it returning valid responses?
- You need **Monitoring & Logging** software

solarwinds LOGGLY DATADOG New Relic

Summary

- Test thoroughly
- Automate your tests
- Automate your deployment process
- Dev + Test + Ops = DevOps!
- Don’t forget logging – you’ll need it in production!
- Infrastructure is part of the code

You shape the course!


Ignoring the project, the course was way too loose–ended. I felt no direction in the lectures and assignments. Instead of _telling us_ what things in powerpoint slides are how about _making us_ perform or _make us learn_ whatever you're talking about? If you're talking about CICD maybe have us _do_ something that is typical of a CICD workload or making us learn a deployment tool as an assignment?

Also the assignments are god awful; neither of them to me felt like the had any clear purpose to them. A1 is the most painful assignment I have ever written and almost certainly the most painful I will ever write. I expect assignments to make me think, not to make me want to defenestrate myself. A2 has too much overlap with A1. Both of them also have a far too vague assignment description. Maybe the assignments are supposed to reflect the material in the lectures, but however much mental gymnastics I do I cannot see the connection.

I believe the additional assignments we have to do in this course take away from the main project.

Maybe a smaller scale application being built? Larger teams to make sure each student has less of a workload? This project is seeming very overwhelming at the moment

Actual small example that we can play around with to understand the material

This is your opportunity!