Reliable Software Systems

Week 4: Pre-production

January 31, 2018
Alyssa Pittman
University of Washington Allen School

Motivating example: Healthcare.gov launch

Launched October 1, 2013 - 250k visitors on first day, 6 managed to sign up

Expected use was 50k visitors per day

> 5 days prior to launch, testing showed that only 1k simultaneous users overloaded the site
>
> After that, tried to double hardware capacity before launch

Drop-downs of data weren’t populated

> End-to-end testing wasn’t completed before launch

https://oig.hhs.gov/oei/reports/oei-06-14-00350.pdf

Pre-production practices

Building

Integrating

Testing

Pre-production environments

Pre-production practices

One idea:

Keep all code in a separate branch.

Do some testing locally.

Commit to main when you want to push it to production.

What could go wrong?

Continuous Integration (CI)

Frequently building/integrating code.

Automate!

When a build breaks, you know immediately.

You fix it immediately (or revert your commit).

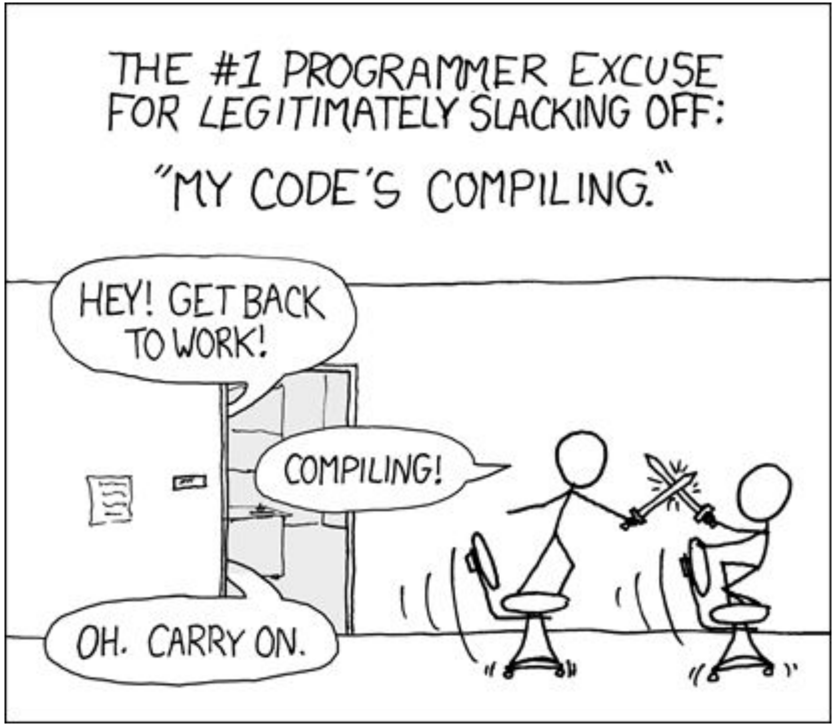
Continuous Integration (CI)

Benefits:

- Problems discovered early.
- Code is fresh in memory.

Drawbacks: (kinda)

- Can take a long time.
- Requires small, independent commits.
- Requires good tests in order to pay off.



https://xkcd.com/303/

Continuous Delivery

Some teams deploy directly to production

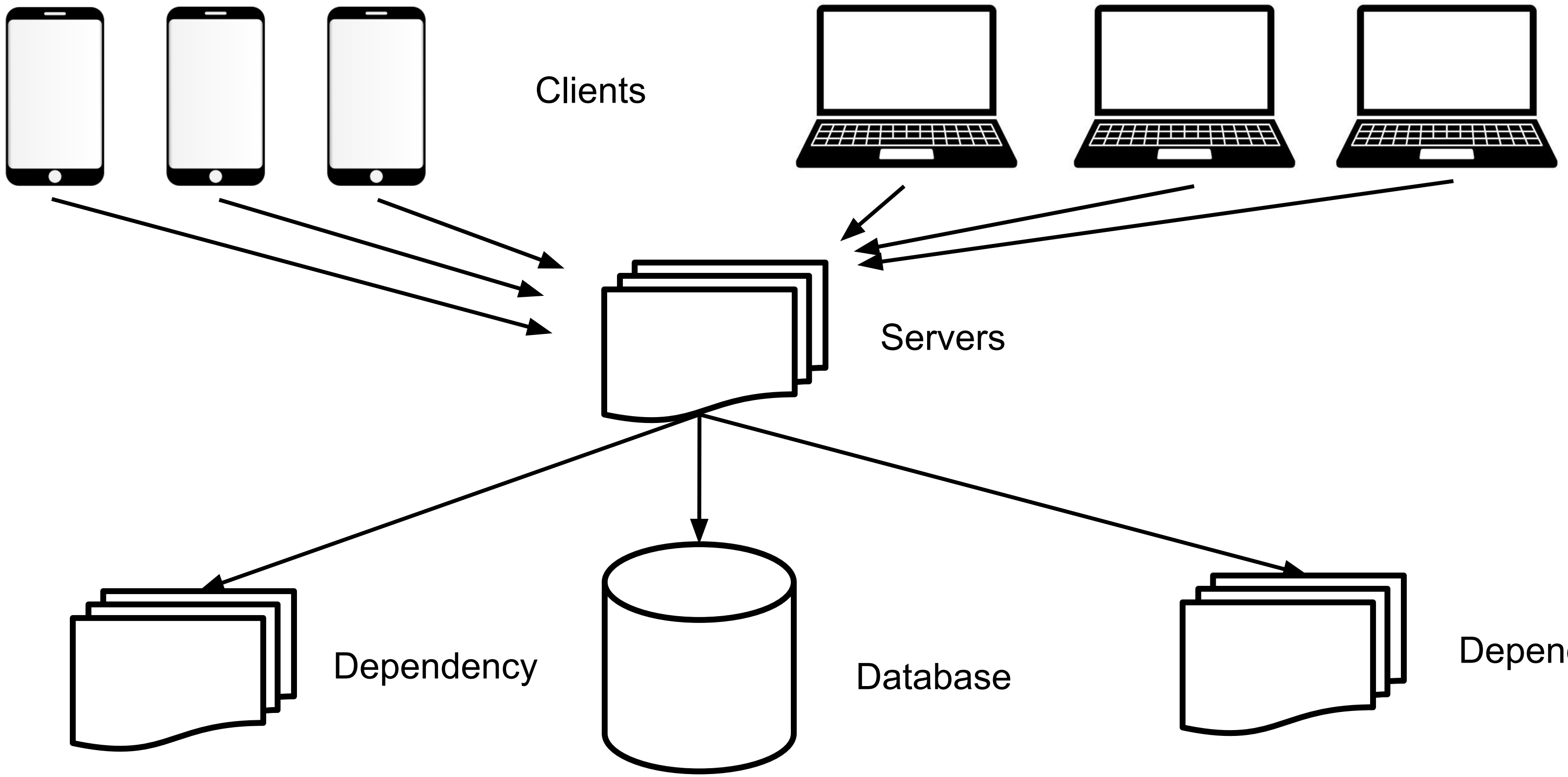
Some teams deliver a package that *could* be deployed

…so far, we know the code will build, how do we know it will work properly?

Environments

Environment	**Dependencies**	**Data**
Local/dev	Local or in memory	Fake
Test/QA	Local or test	Fake
User acceptance testing	UAT or production	Fake
Staging	Staging or production	Fake or production
Production	Production	Production

Systems today



Staging Environment

“Staging is where you validate the known-unknowns of your systems.”

Staging is the closest to production you can get without actually being there.

- Use the same dependencies (services, databases, load balancers…)
- Have the same types of resources
- Fix issues quickly…but it’s ok if they happen!
- Make sure it gets used

Continuous Testing

“Testing shows the presence, not the absence of bugs.” - Edsger Dijkstra

BUT
Test as you go.

Have good unit test coverage.

Have tests that will run quickly.

Integration Testing

Unit tests

- Fast, class-sized tests
- Mock out dependencies

Integration tests

- Make sure that parts of the system work together (end-to-end)
- Verify assumptions about dependencies

Load Testing

Make sure the system works at scale

- Throw as much as you can at it!
- Determine whether it will meet the load you expect

Consider whether to write, read, or both

Use made-up data or real data?

Soak Testing

Make sure the system works for prolonged periods of time at normal load

- Throw just your average load at it!
- Determine whether it will meet the load you expect

Most commonly discovered problems:

- Memory leaks
- Failure to close connections
- Gradual deterioration

END