Reliable Software Systems

Week 5: Production

February 7, 2018
Alyssa Pittman
University of Washington Allen School

Windows 10 October 2018 update

"Windows as a Service" ships fixes and feature updates to customers' PCs.

Early testers noted that this update deleted files from certain directories.

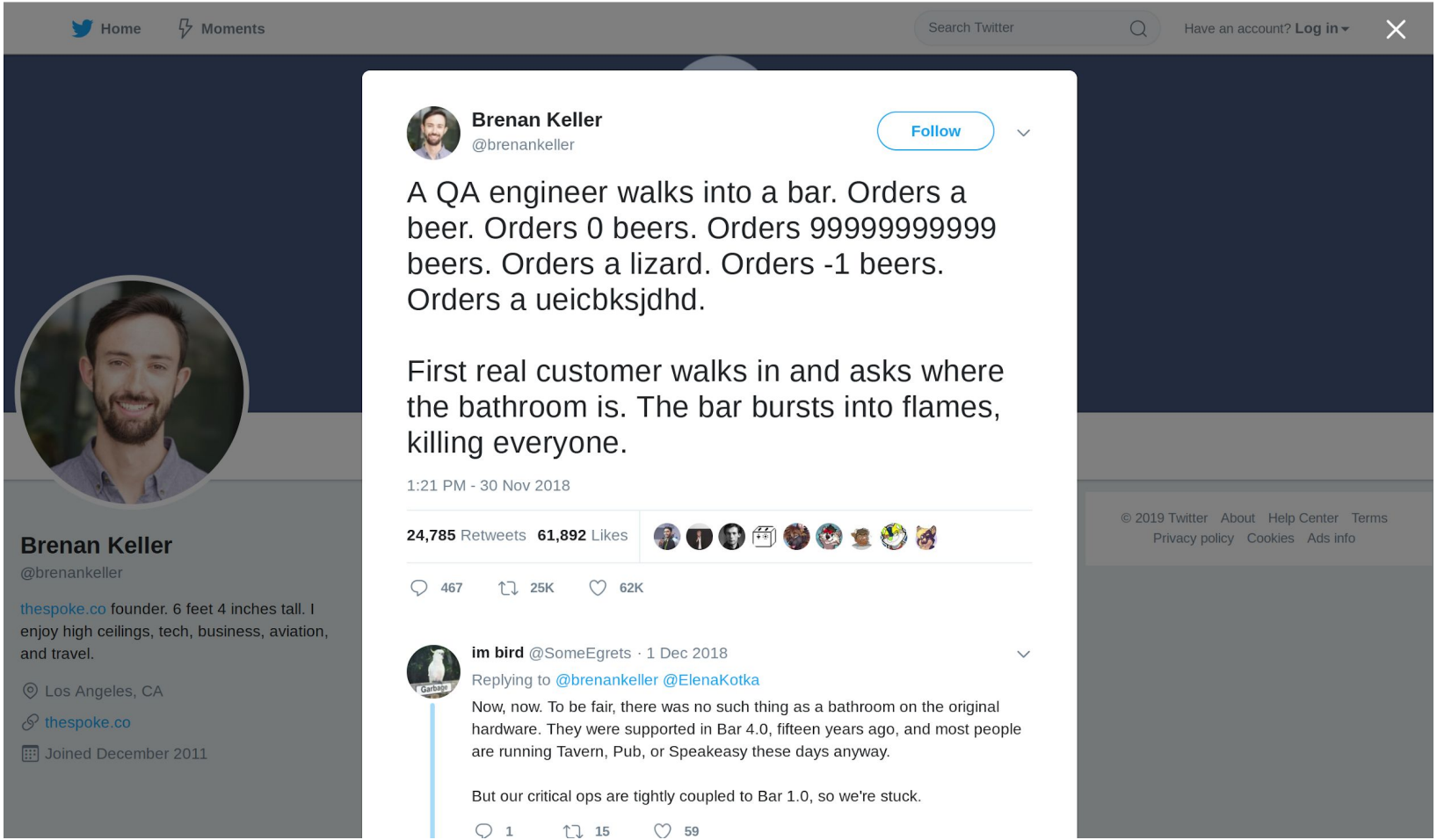
It rolled out anyway, but was halted after more customers reported data loss.

After the fix, they rolled it out more slowly, watched metrics.

Incompatibilities with drivers & some apps caused more delays as some users were restricted from updating.

Eventually (over two months later) all restrictions lifted.

https://www.hindustantimes.com/tech/microsoft-windows-10-october-2018-update-roll-out-resumes-with-bug-fixes/story-gCgxAShzZRCHOGsnkECdgM.htm

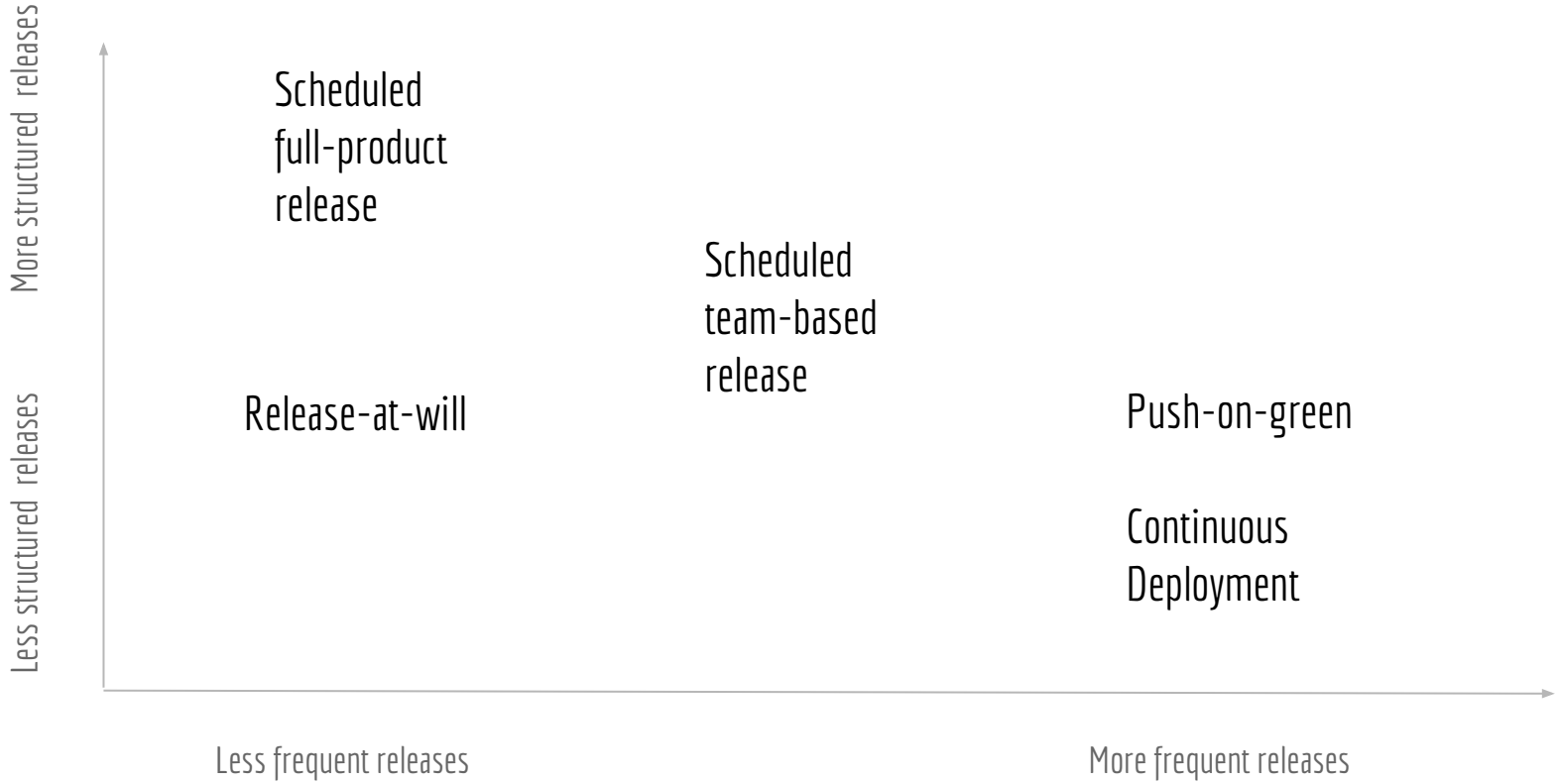


https://twitter.com/brenankeller/status/1068615953989087232

Google's top 8 outage triggers

Binary push	37%
Configuration push	31%
User behavior change	9%
Processing pipeline	6%
Service provider change	5%
Performance decay	5%
Capacity management	5%
Hardware	2%

Deployment models



Mitigating risk

Blue/Green deployments

- Have two separate production environments (“blue” and “green”)
- Deploy to blue
- Move traffic from green to blue
- If problems, move the traffic back to green

Mitigating risk

Canarying

Deploy new code to a small part of your production environment

Send a small amount of traffic to the canary

Monitor to make sure the canary is healthy

If problems, move traffic back to the rest

Otherwise, continue rolling out

Mitigating risk

Gradual release

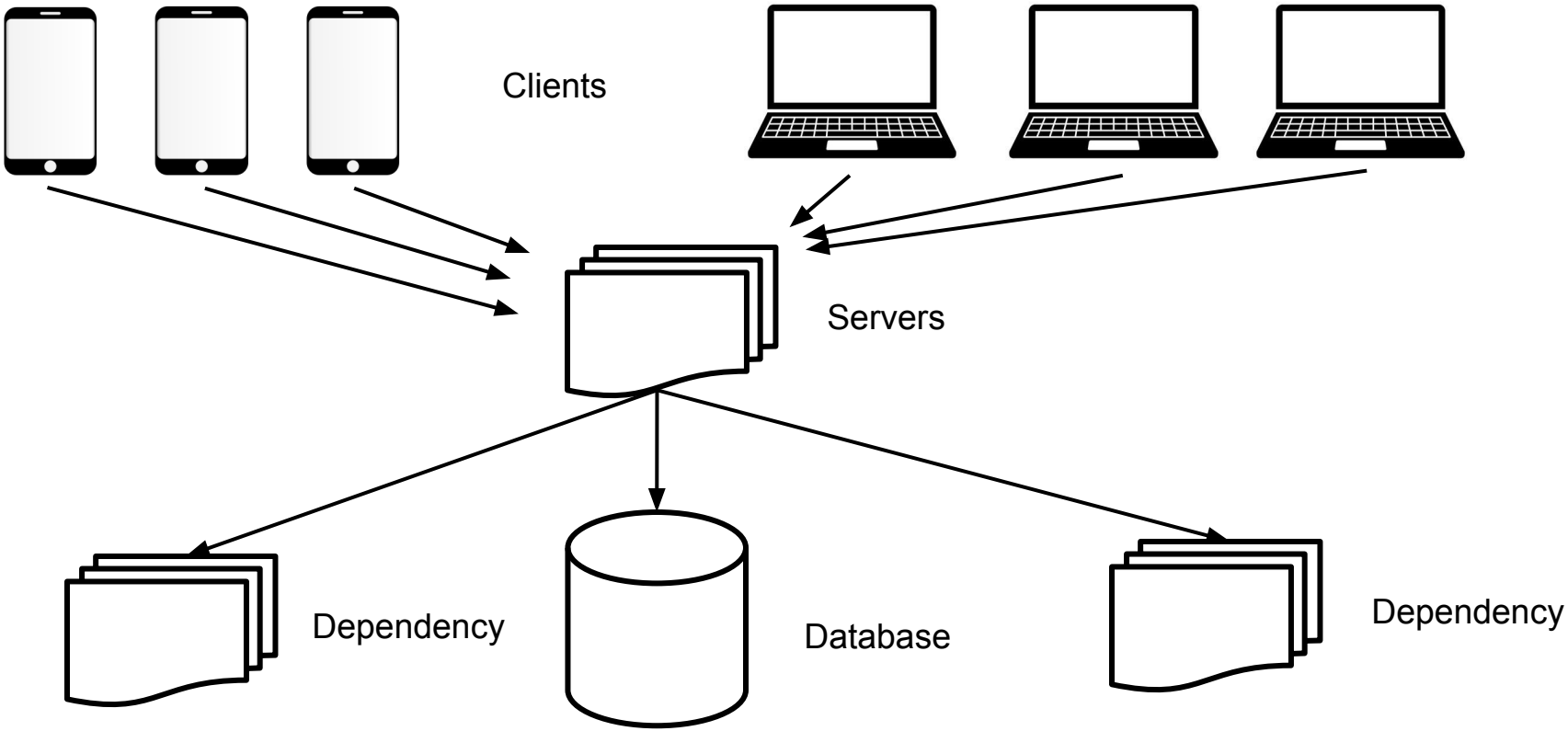
- Deploy to a small part of production
- Send part of your users there
- If good, gradually increase the rollout and the amount of traffic

Mitigating risk

Feature toggles - configuration that turns features on or off

```
if flag(“use_my_super_cool_new_feature”) {
  // do my new awesome stuff
} else {
  // do the boring old stuff that works
}
```

Multiple versions of code running



Multiple versions of code running

Example: to move a field from one database to another

1. Code double writes - write to both the old database and the new.
2. Release.
3. Migrate all old data from the old database to the new format.
4. Code changes to read from the new location.
5. Release.
6. Code removes old write and read path.
7. Release.

Oh no, I broke it!

If still have the old deployment around...

Redirect users to the stable deployment

If there's a feature toggle...

Turn it off!

If rollout is finished...

Prefer rolling back rather than fixing the code

Assorted Best Practices

Automate repeated processes

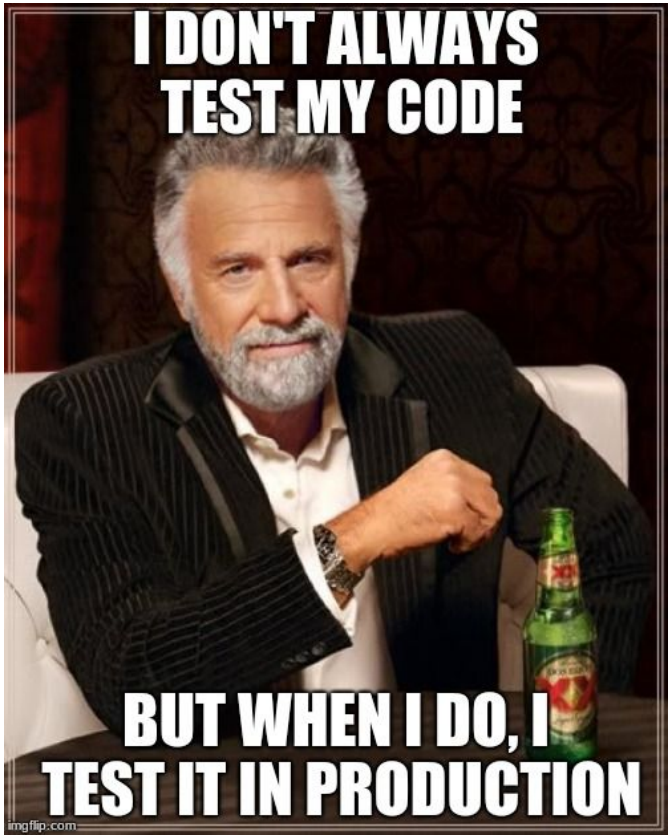
Don’t release on Fridays :)

- Consider a “production freeze”

Make sure there’s an audit trail

- What was released when?
- Show build versions in your monitoring!

Testing in production



Testing in production

There will be failures anyway.

So practice failing* and recovering**!

* requires buy-in from your company

** and careful planning and engineering

Disaster recovery testing

Google annually practices failure by breaking live systems and testing procedures

Large: take down a data center

Small: team takes down a few servers

Real: turn off some internal tools

Procedural: have datacenter operators pretend to buy massive amounts of diesel fuel

Chaos engineering

Started by Netflix in 2010 to vet their move to AWS

- “Chaos Monkey” would pseudo-randomly reboot machines

Later added more types of failure injection

- Latency, error rates…

Also added ways to scope failure and have a feedback loop

END