Lecture 34 — DevOps: Configuration

Patrick Lam & Jeff Zarnett
`patrick.lam@uwaterloo.ca` `jzarnett@uwaterloo.ca`

Department of Electrical and Computer Engineering
University of Waterloo

December 10, 2022

DevOps for P4P

So far, one-off computations: you need to answer a question, so you write code to do that.

But many systems are long-running ("generally available").
$\Rightarrow$ Operations.

Keep it Rolling

Cloud computing: often long-lived systems, but we didn’t talk about how.

Today: many companies fuse development (writes the software) and operations (tends the software).

Disaster Girl Strikes Again

Start Me Up

Startups:

No money to pay for separate
developer and operations teams.

Not that many servers,
just a few demo systems, test systems, etc…
but it spirals out from there.

You’re not really going to ask Sales to manage these servers, are you?
So, there’s DevOps.

DevOps—Good Plan?

Is DevOps a good idea?
Can be used for both good and evil.

Good:

- developers involved across the software lifecycle.
  (can learn a lot doing ops...)
- developers motivated to use correct tools & document processes.

Continuous Integration

Each change or related group of changes is evaluated:

- Pull from version Control
- Build
- Test
- Report results

Social convention to not break the build! Slack/Teams/etc. notifications.

Configuration as Code

Systems have long come with complicated (“flexible”) configuration options.

Sendmail is particularly notorious, but apache and nginx aren’t super easy to configure either.

First principle: treat *configuration as code*.

Configuration as Code

- use version control on your configuration.
- implement code reviews on changes to the configuration.
- test your configurations.
- aim for a suite of modular services that integrate together smoothly.
- refactor configuration files.
- use continuous builds.

Autoconfig

Excellent idea: tools for configuration.

Not enough to write text
    “How to Install AwesomeApp”

e.g. use Terraform—
build, installation, and update automatic & simple.

Complicated means mistakes...people forget steps. They are human.

Terraform

Its whole purpose is to manage your config as codes situation where you want to run your code using a cloud provider (e.g., AWS),

Planning is Essential

Terraform has a “plan” operation: can verify the change it’s about to make.

Verify that we aren’t about to give all our money to Jeff Bezos but also that a small change is actually small.

If you are happy with the change, apply it – but things can change between plan and apply!

Destructive Changes

You can accidentally tell it you want to destroy all Github groups and people DM you thinking that this means they got fired.

You're fired
Your employment is hereby terminated
You're jobn't
You're promoted to customer

Common Infrastructure

Use APIs to access your infrastructure. Examples:

- storage
- naming and discovery
- monitoring

Avoid one-offs—use open-source tools when applicable.
But build your own tools if needed.

Oh, Think Twice...

Is this what we are best at?

Think extra carefully if you plan to do roll your own anything that is security or encryption related.

Remember that platforms like AWS are constantly launching new tools.

Naming

Naming is one of the hard problems in computing.

There are only two hard things in computers:

1. cache invalidation,
2. naming things, and
3. off by one errors.

Naming Suggestions

- use canonical one-word names for servers;
- but, use aliases to specify functions: geography/environment/purpose/serial

There's also the Java package approach of infinite dots: live.application.customer.webdomain.com

Pick something and be consistent.

Billing or Potato?

Debates rage about names should be meaningful or fun.

If the service is called `billing` it may be helpful in determining what it does, more so than if it were called `potato`.

But when you say the word do you mean the service or the team?

What if we need a new billing service? `billing2`

Descriptive Names aren’t Magic

I’ve seen examples where the teams are called (anonymized a bit) “X infrastructure” and “X operations”.

I’d estimate that 35% of queries to each team result in a reply that says that the question should go to the other team.

It gets worse when a team is responsible for a common or shared component (e.g., library).

Naming Solutions

Real solution is like service discovery: tool with directory info.

Something like Opslevel exists; use it!

And don’t forget that fun has a morale impact.

Servers as Cattle, not Pets

Servers means servers, or virtual machines, or containers.

At scale (smaller than you think):
use mass tools for dealing with servers,
rather than doing tasks manually.

At least: cloud-like server initialization without manual intervention;
must be able to spin up a server programmatically.

Another Example

kubernetes

This is used to automate deploying and scaling of applications.

Canarying

Canarying

Deploy new code incrementally in production,
also known as “test in prod”:

- stage for deployment;
- remove canary servers from service;
- upgrade canary servers;
- run automatic tests on upgraded canaries;
- reintroduce canary servers into service;
- see how it goes!

Of course: implement your system with rollback.

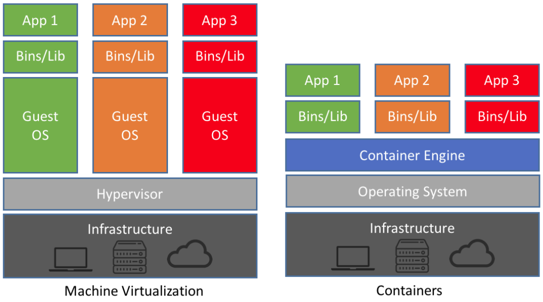
Containerize Me, Captain

How Did We Get Here?

- Manual install
- Package Manager (RPM/JAR/DLL Hell)
- Virtual Machines
- Containerization

Containers

See this diagram from NetApp:


App 1
App 2
App 3
Bins/Lib
Bins/Lib
Bins/Lib
Guest OS
Guest OS
Guest OS
Hypervisor
Infrastructure
Machine Virtualization
App 1
App 2
App 3
Bins/Lib
Bins/Lib
Bins/Lib
Container Engine
Operating System
Infrastructure
Containers