TEST DRIVEN INFRA

How the Heck do you Apply TDD to Infra as Code (IoC)??

Open Source Summit
2018-08-29

> Any decent answer to an interesting question begins, "it depends..."
> @KentBeck - May 2015 [0]

Author: Daniel Pacrami @dansible
SAP, Montreal, QC, Canada
Slides: https://gitlab.com/dansible/oss-2018-tdi
Website: https://dansible.gitlab.io/

INTRO

Plan:

0-3)	Intro	13,14)	Build Pipelines
4)	TL;DR	15,16)	Interfaces
5-8)	Concepts	16,17)	Service Pipelines
9-12)	History	18)	Conclusion

Motivation:

1. Limited IoC literature & online advice. [1]
2. CI/CD for Infrastructure is hard.
3. Need for a CM & IoC development guide.
4. Want to combine Theory & Practice.

INTRO

Premise:
Develop a Platform-as-a-Service, automation and tooling
for development teams.

Background:

Dev	-> Build stuff
Ops	-> Run stuff
DevOps	-> Build & Run stuff :)
SRE/Tools Team (?)	~> Build & Run infra for DevOps?

Environment: [2]

Siloed Dev/Ops	-> "You build it, you run it" [3]
3rd Party Integration	-> Development & Automation.
Waterfall	-> Agile & CI/CD.
"Fix code, not people"	-> Pair/Mob practice.
Reinventing the wheel	-> Baseline standards & tooling.

TL:DR

1. Version everything.
2. Abstract complex infra into components, svcs, and ifcs.
3. Build infrastructure components against executable specs.
4. Define interfaces with clear downstream contracts.
5. Shoot for deterministic, idempotent, & orthogonal svcs.
6. Use linting & style guides.

What is Infrastructure as Code (IoC)?

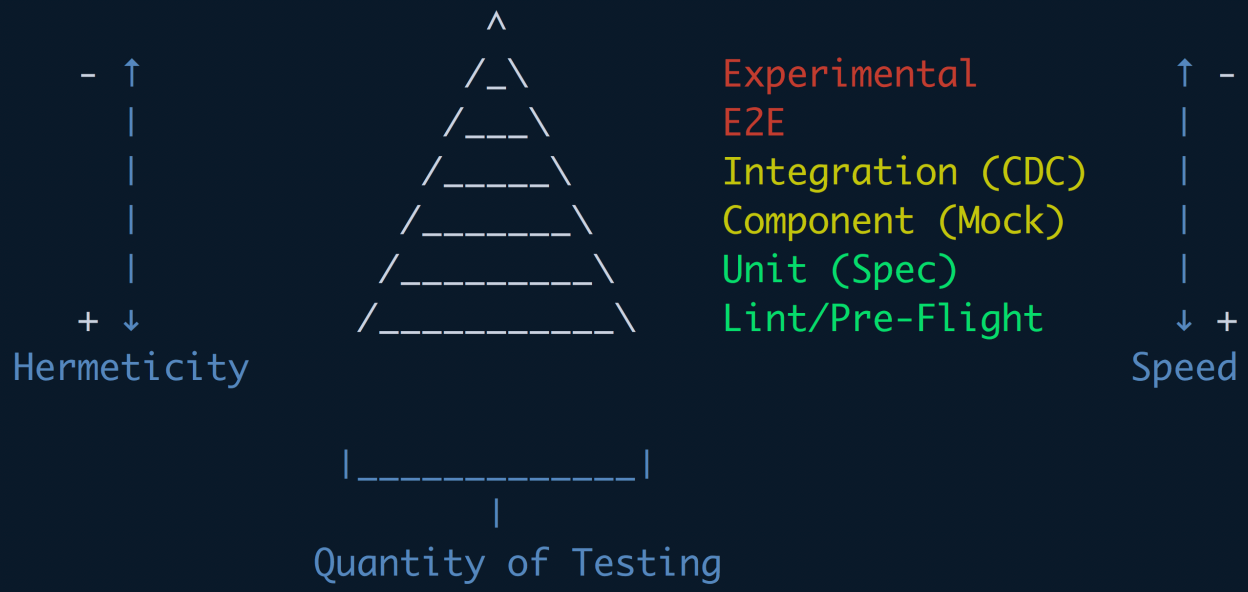
- Defined as an executable document.
- Immutable.
- Scalable.
- Versioned.
- Testable.

+ Applies deterministic Configuration Management (CM),
+ Written in a declarative language,
+ Deployed in an automated Pipeline.

What is Test Driven Development (TDD)?

- Write tests before production code.
- Red Green Refactor :D

The Almighty Test Pyramid:



Why Test-Driven Development?

- Problems manifest and are isolated more quickly.
- Code can be safely modified and refactored.
- Allows [tools | platform | architecture] to change.
~ Tests are more readable documentation than code.
- Test code can drive monitoring & alarming.
- Fosters experimentation.
...

LGTM!

What are the challenges specific to Infrastructure?

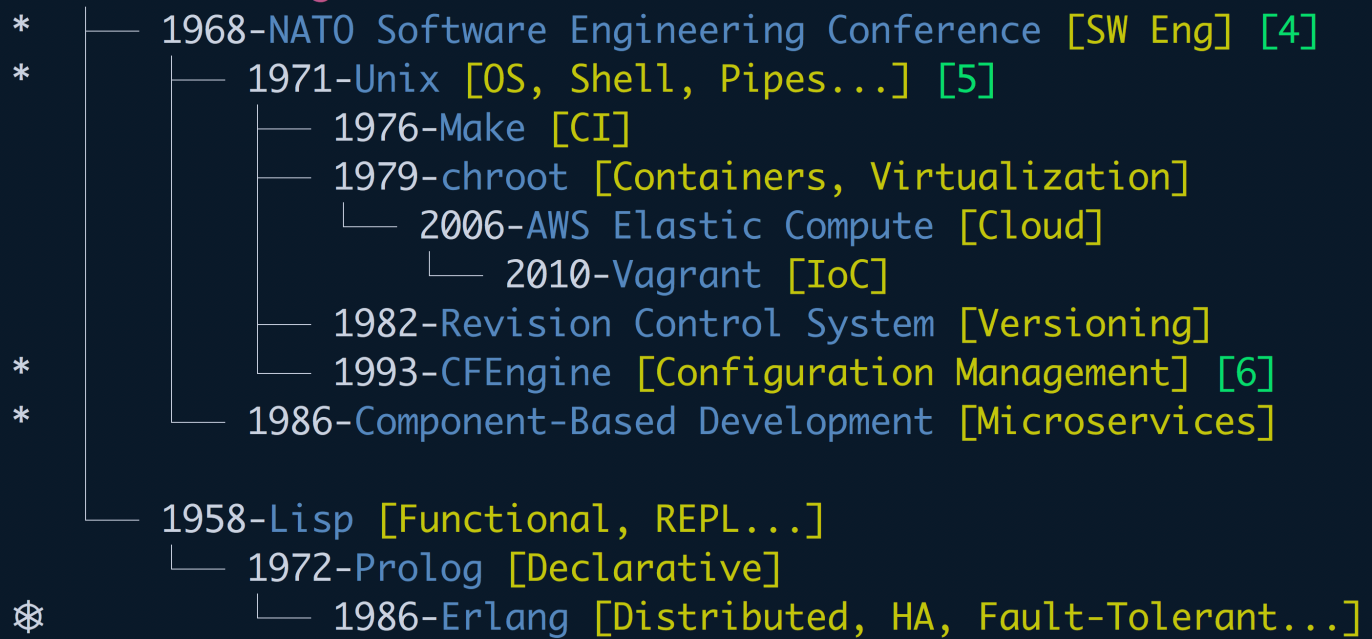
- Static analysis of declarative languages.
- Atomicity of infrastructure codebase and primitives.
- Hermeticity of infrastructure components.
- Integration of 3rd party products and services.
- Management of environment parity.
- Disposability of infrastructure.

> "Can we look back to the history of our industry, theory, methodology, and tooling to find solutions to some of these challenges?"

> "Mass produced software components"
> Douglas McIlroy, NATO Software Engineering Conference, 1968 [4]

HISTORY

How did we get here?



PAAS:

> Use tools in preference to unskilled help, even if you have to detour to build the tools.
>
> Doug McIlroy - Bell System Technical Journal 1978 [5]

Three fundamental system design concepts:

> Modularity helps to isolate functional elements of the system. One module may be debugged, improved, or extended with minimal personnel interaction or system discontinuity.
> Specification: the key to production success of any modular construct is a rigid specification of the interfaces.
> Generality is essential to satisfy the requirement for extensibility.
>
> H.R. Gilette - Nato Software Engineering Conference, 1968 [4]

On Components & Interfaces:

> A piece of software offering (via an interface) a predefined service and which is able to communicate with other components.
> Rainer Niekamp - Software Component Architecture, 2011 [7]

> Software components are used in two different contexts:
> 1. Using components as parts to build a single executable, or
> 2. Each executable is treated as a component in a distributed environment.
>
> Brian Cox - Object-Oriented Programming, 1986 [8]

> We can organize our system as a set of communicating processes. By enumerating all the processes in our system, and defining the message passing channels between the processes we can conveniently partition the system into a number of well-defined sub-components which can be independently implemented, and tested.
> Joe Armstrong (on Erlang), 2003 [9]

The Bezos Mandate:

1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols -- doesn't matter. Bezos doesn't care.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired.

Stevey's Google Platforms Rant [~2006?] [10]

BUILD PIPELINES

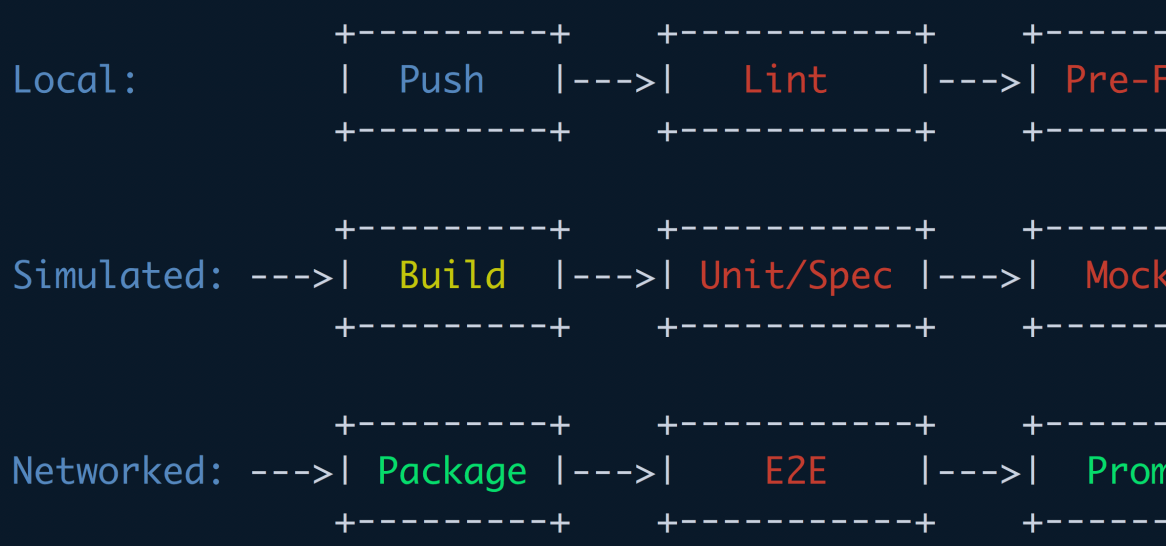
Build Pipeline Definition:
- Produces a versioned infrastructure component.
- Triggered by code change.
- Tested in isolation against specs and mock resources.
- Implemented recursively to all components & interfaces.
- Resembles a traditional CI/CD pipeline structure.

> If the system is simulated at each level of design, errors can be found and the performance checked at an early stage.
> J.W. Graham - Nato Software Engineering Conference, 1968 [4]

Examples:
- Docker Images.
- Terraform Modules.
- Helm Charts.
- Packer VM Images.
- RPM/Deb Packages.
- Homebrew Code.

BUILD PIPELINES

Build Pipeline Example:



INTERFACES

Interface Definition:

- Defines a contract for how a component or svc is consumed.
- Tested in a build pipeline to validate compliance.
- Coupled with a service or component version.
- Abstracts complex implementation details.

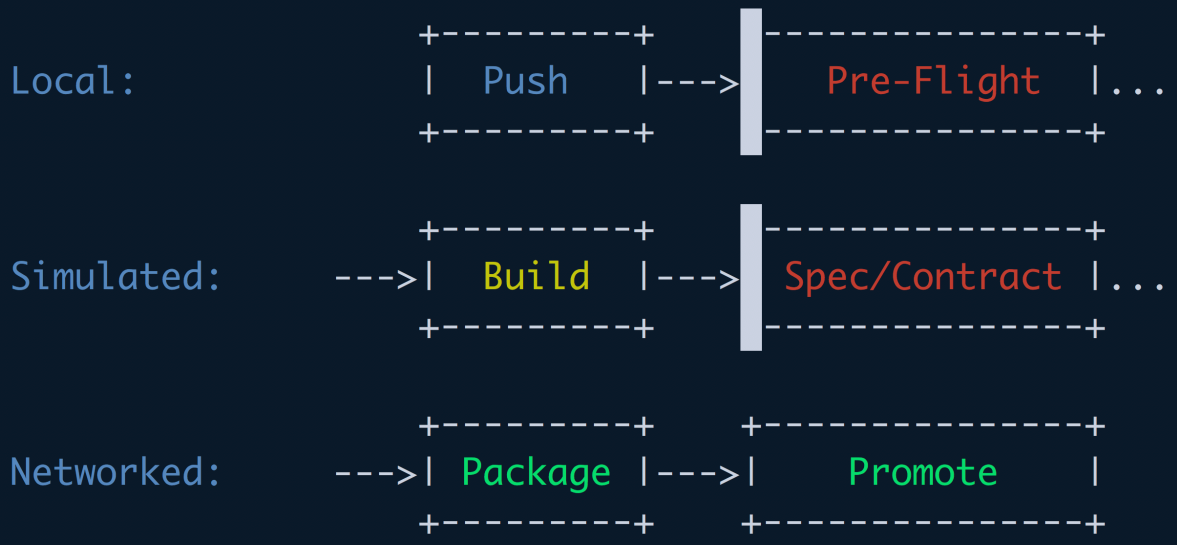
> Whenever some consumer couples to the interface of a component to make use of its behaviour, a contract is formed between them. This contract consists of expectations of I/O data structures, side effects, and performance & concurrency characteristics.
>
> Toby Clemson - Microservices Testing, 2014 [11]

Examples:

- GitHook to validate commit message before pushing code.
- Swagger file defining paths, operations, and I/O of an API.
- BDD "given, when, then" test of expected behavior.
- Sentinel policy to enforce ACLs.

INTERFACES

Build Pipeline (Revisited):



SVC PIPELINES

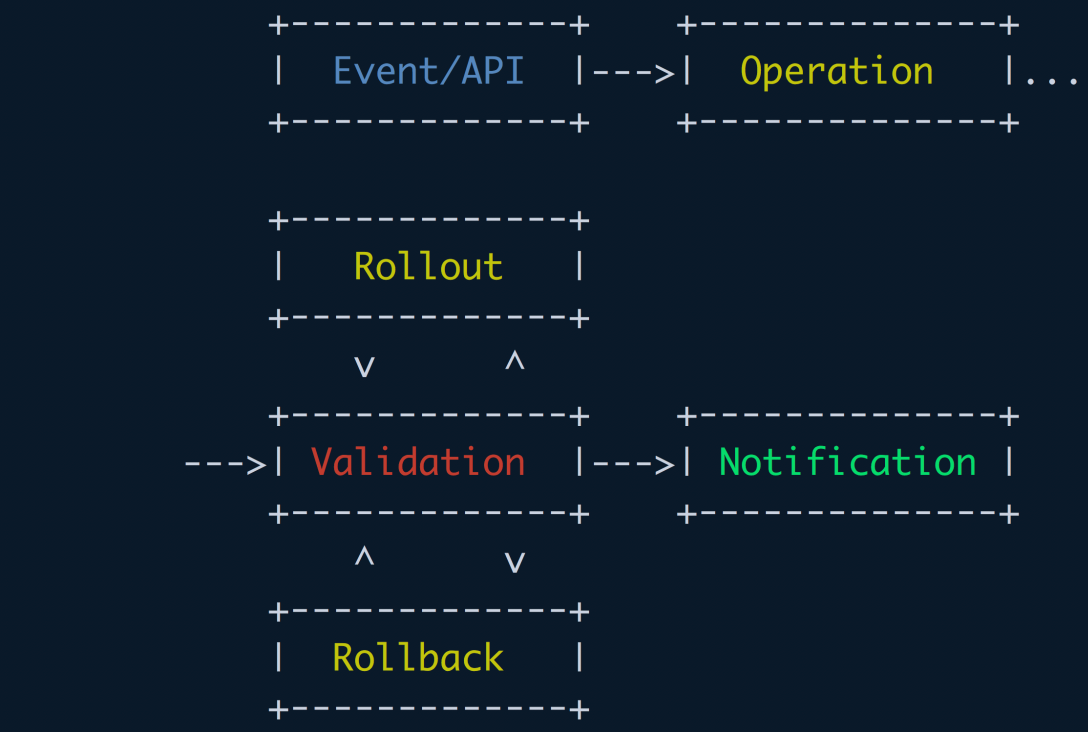
Service Pipeline:
- Modifies state of an existing system.
- Event/API driven.
- Strives for determinism, idempotence, & orthogonality.
- Includes rollout/rollback strategy.

> Orthogonality reduces test and development time, because it's easier to verify code that neither causes side effects nor depends on side effects from other code – there are fewer combinations to test.
> Hunt & Thomas, The Pragmatic Programmer, 1999 [11]

Examples:
- Terraform plan to deploy components.
- Ansible playbook to create users and reset passwords.
- Automated QA, compliance and security scanning.
- Automated Chaos experiments.

SVC PIPELINES

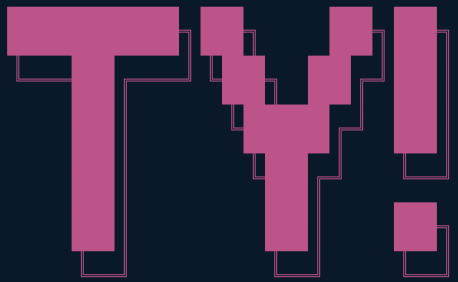
SVC Pipeline Example:



Conclusion:

What are our takeaways?

- TDD best practices are highly applicable but need to take into account peculiarities of infra.
- SRE/DevOps is a new industry but that doesn't mean we can't learn from our past.
- Abstracting infra into components, interfaces and services helps reframe our design and bring our practices in line with software engineering.
- Check out the Unix philosophy's 17 rules.



REFERNCES

[0] - https://twitter.com/kentbeck/status/596007846887628801?lang=en

[1] - Nelson-Smith, Stephen - Test-Driven Infrastructure with Chef:
Bring Behavior-Driven Development to Infrastructure as Code
ISBN-13: 978-1449372200 | ISBN-10: 1449372201

[2] - Linux Foundation - Open Source Guides For The Enterprise
https://www.linuxfoundation.org/resources/open-source-guides/

[3] - Vogels, Werner - "A conversation with Werner Vogels"
in ACM Queue, May 2006
https://queue.acm.org/detail.cfm?id=1142065

[4] - NATO Software Engineering COnference, 1968
http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

[5] - Bell Laboratories - The Bell System Technical Journal, 1978
http://emulator.pdp-11.org.ru/misc/1978.07_-_Bell_System_Technical_Journal.pdf

[6] - Burgess, Mark - Computer Immunology, 1998
https://www.usenix.org/legacy/publications/library/proceedings/lisa98/full_papers/burgess/burgess.pdf

[7] - Rainer Niekamp - Software Component Architecture, 2011
http://congress.cimne.upc.es/cfsi/frontal/doc/ppt/11.pdf

[8] - Brian Cox - Object-Oriented Programming:
An Evolutionary Approach, 1986
ISBN-13: 978-0201548341 | ISBN-10: 0201548348

[9] - Joe Armstrong - Making Reliable Distributed Systems
in the Presence of Software Errors, 2003
http://erlang.org/download/armstrong_thesis_2003.pdf

http://congress.cimne.upc.es/cfsi/frontal/doc/ppt/11.pdf

[8] - Brian Cox - Object-Oriented Programming:
An Evolutionary Approach, 1986
ISBN-13: 978-0201548341 | ISBN-10: 0201548348

[9] - Joe Armstrong - Making Reliable Distributed Systems
in the Presence of Software Errors, 2003
http://erlang.org/download/armstrong_thesis_2003.pdf

[10] - Stevey's Google Platforms Rant ~2006?
https://plus.google.com/+RipRowan/posts/eVeouesvaVX

[11] - Mark Burgess - On the Theory of System Administration, 2003
http://markburgess.org/papers/sysadmtheory3.pdf

[11] - Hunt & Thomas, The Pragmatic Programmer, 1999
https://www.nceclusters.no/globalassets/filer/nce/diverse/the-pragmatic-programmer.pdf