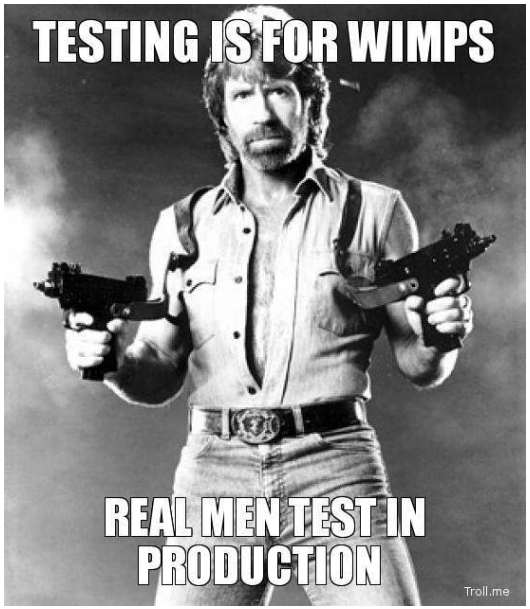
Infra Acceptance As Code

Ankur Kumar

The Common Problem

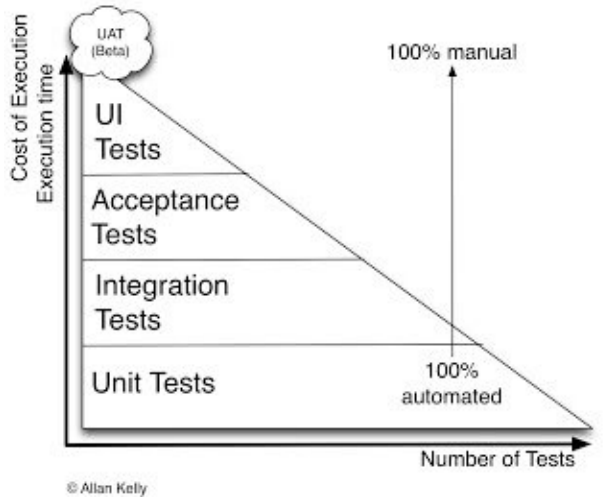
The drift between software versions, configs, processes etc. while taking the software from development to production in-turn becomes a big headache sooner than later

Haaaaaaaaaaaaa ...



- Dev did something, QA did another, other guys did their black magic further to keep on declaring the software successful at their respective stages of delivery. But finally, the Prod busted due to many unknowns and manual tricks
- Who could forget the time wasting painful war room sessions on a regular basis and always wondering and struggling to reason about the weird failures and doing dirty patching?

Why Acceptance Testing for Infra?



- If you can't verify it against the necessary baseline then you can't prove it correct
- The test driven development has been the mantra in the software for the last any years. So why shouldn't it be considered necessary to platform/infra components in the age of IAC?
- Provides a clear and unambiguous **contract** between the producers and consumers
- Becomes part of CI, Diagnostics, Observability and Compliance tooling

The Scope for IAC Artifacts and Solutions

- Machine Images
- Virtual Machines
- Container Images
- Containers
- Cloud Resources
- Structural level e.g. what packages/libs/binaries/configs are required? which vesions and their integrity? the required content for the configs? etc.
- Runtime level e.g. what processes are required to run? on which ports they are suppossed to bind? their endpoints responses? external dependencies reachability? etc.
- Environment level e.g. combining the previous ones along with the Cloud resources

Implementation Phases

- 1st covers expressing the sanity process through relevant DSLs for the respective tools found fit for the respective artifacts
- 2nd covers the orchestration of all the immediate term pieces as workflows to be triggered automatically
- 3rd consists of an unified metadata API built over the orchestration/workflow engine implemented in short term, covering all the use-cases

Tools/Frameworks of The Trade

Desired Properties

- Declarative rather than procedural
- Operationally lightweight with fewer/no dependencies
- Could be used as library
- Easily extendible
- Performant

GOSS: Option A. for Images/VMs/DockerContainers

- **GO** based **ServerSpec** alternative tool for validating a server's provisioning
- Driven by YAML description of the acceptance tests you want to perform against a server
- The same project also provides a shell wrapper known as DGOSS for the docker containers validation
- Also other wrappers DCGOSS/KGOSS to validate the docker containers based upon docker-compose and kubernetes respectively
- Operationally, the goss is a static binary for GNU/Linux which is just a grab, drop in a machine and all ready to validate the Images/VMs/Containers
- Provides an option to expose your test suite as a health endpoint listening on your server on a configurable port
- The community contributions provide some additional addon functionalities/plugins for it including few ansible and a packer plugin

TESTINFRA: Option B. for Images/VMs/DockerContainers

- With **testinfra** you can write tests in Python to test actual state of your servers
- A serverspec equivalent in python and is written as a plugin to the powerful Pytest test engine
- Comes with several connections backends for remote command execution, **local** is the default backend
- If you have a lot of tests, you can use the pytest-xdist plugin to run tests using multiple processes

AAC for Docker Images/Containers

- Pre built: Dockerfile linting
- Post built: DockerImage verification
- Post build: DockerContainer verification unit testing only possible if "degrade but don't crash"

HADOLINT: Option C. for Dockerfiles

- A smarter Dockerfile linter that helps building best practices Docker images
- Standing on the shoulders of ShellCheck to lint the Bash code inside RUN instructions
- Supports specifying the ignored rules using a configuration file in yaml format
- Static binary or Docker image to get integrted with CI pipeline and K8s plugin

DOCKERFILE-LINT: Option D. for Dockerfiles

- A rule based 'linter' for Dockerfiles
- The linter rules can be used to check file syntax as well as arbitrary semantic and best practices attributes determined by the rule file writer
- Can also be used to check LABEL rules against docker images

CONTAINER STRUCTURE TESTS: Option E. for DockerImages

- A powerful framework to validate the structure of a container image
- Command tests for output/error of a specific command issued
- File existence tests for making sure a file is or isn't present in the filesystem of the image
- File content tests for making sure files in the filesystem of the image contain or don't specific contents
- Singular metatdata tets for making sure certain metadata is correct in the image
- File existence/content tests don't require a functioning Docker daemon on the host machine

TERRIER: Option F. for DockerImages

- Create wrapper over docker inspect + terrier
- Terrier is a image and container analysis tool
- Scans OCI images and containers to identify and verify the presence of specific files according to their hashes

INSPEC: Option G. VMs/Docker{Images(VeryLimited),Containers(Limited)}/CloudResources

- An open-source framework for testing and auditing apps and infra
- Built on top of rspec and uses it as the underlying foundation to execute tests
- It has 80+ resources including AWS/GCP/Azure cloud ones
- Very rich community eco-system to include tests from Chef Supermarket, Dev-Sec project etc.
- Provides a mechanism for defining custom resources
- Requires more effort
- Ruby runtime is slow
- Lacks out of the box parallelism

Thank you

Ankur Kumar