Infrastructure as Code (anti) Patterns

Kris Buytaert @krisbuytaert

Foss North April 2024

Kris Buytaert

- I used to be a developer
- Then I became an Ops person
- Chief Yak Shaver @ o11y.eu
- Organiser of #devopsdays, #cfgmgmtcamp, #loadays, …
- Cofounder of all of the above
- Everything is a Freaking DNS Problem
- DNS : devops needs sushi
- @krisbuytaert on twitter/github/mastodon/bluesky

O11y.eu

- Inuits.eu Spinoff
- Open Source Observability
- Currently supporting the Prometheus Ecosystem
- Professional Services & Support (now)
- LTS Release (Long Term stable release) (now)
- Prometheus “Distribution” (soon)

Devops origins

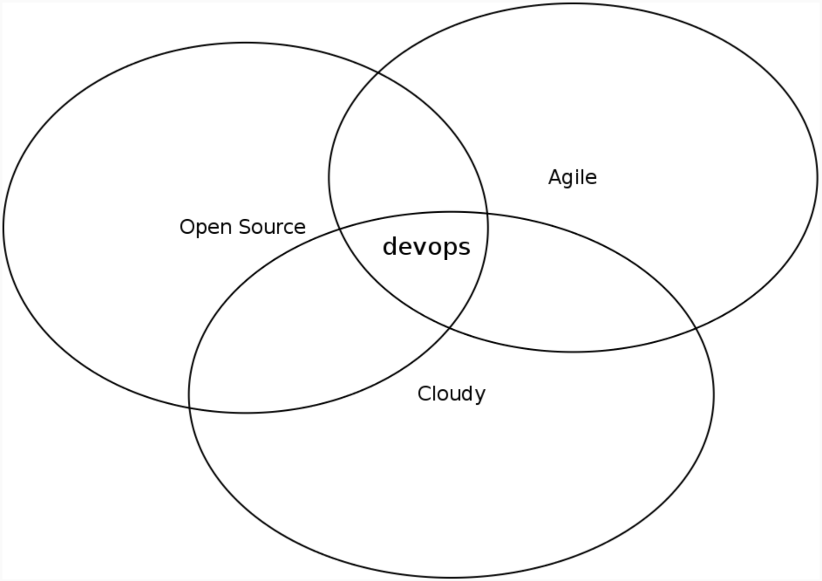


Figure 1: center

Devops definition

A global movement to improve the quality of software delivery leveraging Open Source experiences , started in Gent in 2009

Why we study History

- Because I‘m a grumpy old developer / sysadmin
- Because I‘m opinionated
- Because History Repeats
- Because we need to learn from our mistakes.

We love running in circles

Figure 2: even the devopsdays ams visitor

Deploying an Infrastructure

- 1996 : Manual installation
- 2001 : Mondo Rescue (reproducable single instance)
- 2003 : System Imager , fast reproducable “identical” Infrastructure
  - Image Sprawl
  - “Overrides”
- 2005 : JeOS + Early IAC (CfEngine / Kickstart / FAI)
- 2010 : Desired State infrastructure as code , Puppet, Chef etc
- 2012 : Ansible
- 2013 : Docker
- 2014 : Terraform , K8s
- 2017 : Pulumi
- 2023 : OpenTofu

20+ years of changes

- 1 (Snowflake) server
- Multiple (snowflake) servers (cluster nodes)
- Virtualization
- Cloud
- Containers

Some of The problems

- Manual error
- Differences in nodes due to manual changes
- Not reproducable
- Humans
- Scale

Config Drift

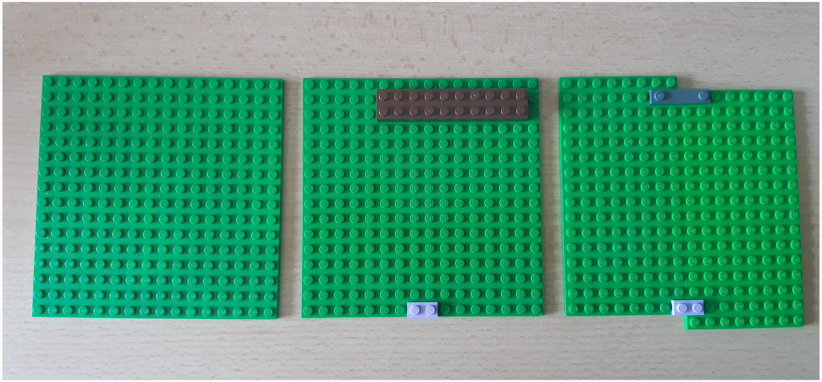


Figure 3: dev vs uat vs prod

Image Sprawling & Golden Images vs JeOS

- Manually Crafted Snowflake
- Copied and modified
- Copied and modified
- Impossible to trace origin
- Impossible to reproduce
- Patching / Security nightmare

What’s wrong with this GW-Basic code ?

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y git
RUN git clone git@github.com:prometheus/prometheus.git
RUN make install
```

What is Infrastructure as Code

- Treat configuration automation as Code
- Model your infrastructure
- Development best practices
    - Model your infrastructure
    - Version Control
    - Pipeline your Code
    - Test your Code

What is infrastructure ?

Everything you need to run your software.

- network
- storage
- compute: vm , container, orchestrator

Infrastructure as Code Goals

- Reproducible Environments
- You never deploy 1x
- Local Test environments
- Development / User Acceptance / Testing / Shadow / Production / DR
- HA <- Keep cluster configuration in sync
- Prevent Configuration drift
- Cheap Disaster recovery

2025 IaC Requirements

3 types of tools needed

- Provisioning
- Desired state
- Orchestration

Provisioning

- Create me an instance of application X
- Container instance
- VM instance
- Service X configuration via API e.g Tofu , Pulumi

Desired state

Ensure that this file present / service is running

- With these permissions
- Always / Verified

e.g Chef, Puppet, Salt, “GitOps”

Orchestration

- Non frequent
- Trigger action X on resource Y based on characteristics A,B and or C
- First do X here then do Y there
- One off actions e.g Ansible, Salt, Puppet Bolt, kubeapply

Idempodency

- In computing, an idempotent operation is one that has no additional effect if it is called more than once with the same input parameters.
- Not idempotent :

echo “root=disabled” » /etc/someconfigfile

IaC & CMDB

Frequent enforcing desired state with reporting = Automated population of a CMDB (Single source of truth) , with up to date data Side Effect : Having a single source from where to generate dynamic configuration eg. reverse proxy , firewall rules, monitoring configuration, backup configuration

Didn’t containers solve this ?

- Where do you deploy them ?
- Do you put everything in containers ?
- How do they know their configuration ?
- export SOMEVARIABLE=“abc” is harmful.

Questions to ask :

- Do you want a reproducible infrastructure ?
- Do you want to reduce manual error ?
- Do you want to achieve desired state ?
- Do you want to have automated configuration for monitoring, metrics, backups, security etc ?
- Do you want to trivially spin up different identical environments (dev/uat/prod/dr)
- Do you want to be able to detect and prevent config drift ?
- Do you want to prevent image sprawl ?

The Patterns

Pattern: It works from my homedir

- Code is not in source control
- Bolt / Ansible / openTofu is triggered from the developer’s local machine
- Works for individual admin
- Doesn’t Scale
- Prevents Collaboration

Pattern: It works from my workstation but its in git

Improvement over $prev, but still doesn’t scale Collaboration has been improved.

Pattern: We only automate the OS

- Just the operating System
- Maybe some basic services
- Application is untouched
- Partial source of maybes
- Not suitable for derived configuration
- Kill your Silos

Pattern: We automate all the things

- Provisioning
- Operating System
- Middle Ware
- Application

Pattern: PCS

- Package
- Config
- Service

Pattern: PCS NFR

- PCS
- Non Functional Requirements
- Monitoring
- Backups
- External dependencies
- Metrics
- ...

Pattern: Reproducable and Scalable Infrastructure

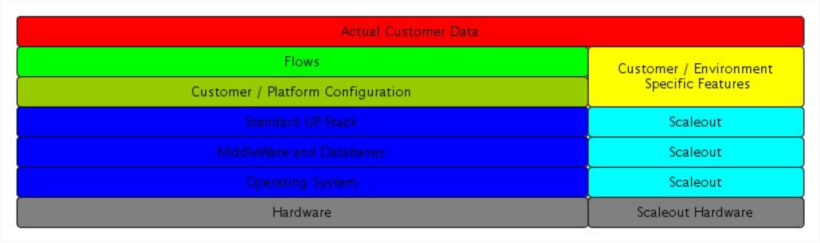


Figure 4: Stacks

Pattern: Multiple tools overwrite changes

- Ops use Chef but Devs use Java templating tool or
- openTofu but Manual changes from UI.

Variants:

- Non centralized code base, people/teams run code from local forks undoing changes made by colleagues
- Templating tool is not idempotent (ruby sort ? / XML)
- Bonus points if the tool restarts a service on a change

Pattern: Centralised apply , from source control

- Code is checked into a git repository
- Code is tested
- The only way to apply is by central user

Pattern: We haven't run it for 6 months

- Tool is run once at deploy time then yolo
- manual changes afterwards are not identified or reversed
- State is not enforced, people are afraid to rerun playbooks
- Now we don't dare to update anymore

Orchestration only mode, no desired state AKA Yolo or Integrator mode

Pattern: It’s broken anyhow .

- Reporting says it’s broken, we are not going to fix it
- Its been disabled for weeks, local modifications break it
- See Config Drift

Pattern: We run it every X minutes

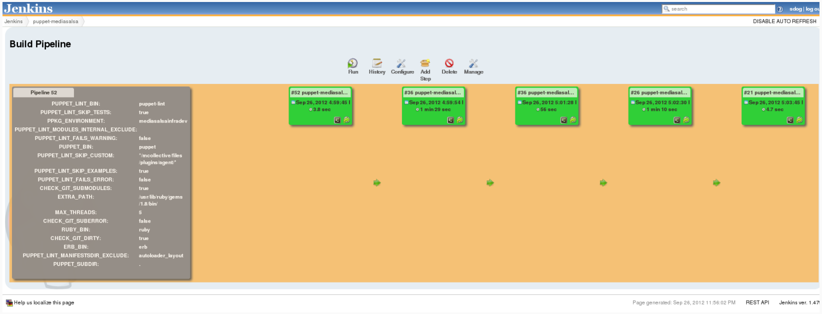
- X < 120
- Periodically we run ensure the state of our Infrastructure
- We prevent config Drift
- We actively monitor broken runs
- We actively monitor change frequency

We have achieved desired state of our infra.

Pattern: We run it in nooop now

- We setup everything, it worked for a while
- Then we broke service A ,
- Then something unrelated broke service B
- Automation got blamed
- We are not allowed to enable it again
- Team X doesn’t trust automation,
- Often Team X = Security
- We manually make the changes now , but we have visibility on drift.
- We have semi desired state

Pattern: We have a pipeline



Build Pipeline

Pipeline 52

PUPPET_LINT_BIN:	puppet-lint
PUPPET_LINT_SKIP_TESTS:	true
PPKG_ENVIRONMENT:	mediasalsainfradev
PUPPET_LINT_MODULES_INTERNAL_EXCLUDE:	
PUPPET_LINT_FAILS_WARNING:	false
PUPPET_BIN:	puppet
PUPPET_LINT_SKIP_CUSTOM:	*/mcollective/files /plugins/agent/*
PUPPET_LINT_SKIP_EXAMPLES:	true
PUPPET_LINT_FAILS_ERROR:	false
CHECK_GIT_SUBMODULES:	true
EXTRA_PATH:	/usr/lib/ruby/gems /1.8/bin/
MAX_THREADS:	5
CHECK_GIT_SUBERROR:	false
RUBY_BIN:	ruby
CHECK_GIT_DIRTY:	true
ERB_BIN:	erb
PUPPET_LINT_MANIFESTSDIR_EXCLUDE:	autoloader_layout
PUPPET_SUBDIR:	.

#52 puppet-mediasal...
Sep 26, 2012 4:59:45
3.8 sec

#36 puppet-mediasal...
Sep 26, 2012 4:59:54
1 min 29 sec

#36 puppet-mediasal...
Sep 26, 2012 5:01:28
56 sec

#26 puppet-mediasal...
Sep 26, 2012 5:02:30
1 min 10 sec

#21 puppet-mediasal...
Sep 26, 2012 5:03:45
4.7 sec

Help us localize this page

Page generated: Sep 26, 2012 11:56:02 PM REST API Jenkins ver. 1.47

Figure 5: A gitops pipeline, from before K8s existed

Environments

What is an Environment , a stack, a platform ?

A logically split of set of servers that together offer a service,
Typically existing in dev/uat/prod flavours

Pattern : Using Environments

- Limiting Blast radius
- Easy to spin up multiple variants
- Feature flagged code introduction
- And promotion of features
- Clear line of ownership (team bound)
- Same code , different config per environment
- Easier release management

Pattern : Not using environments

- Huge blast radius
- Potential Scaling issues
- Difficult to “promote” changes
- Upgrade everthing or write host based if then case spaghetti
- Code duplication : role_app_dev, role_app_uat, role_app_prod

Pattern: Config is hardcoded in the code

- Typically with no environments
- Environment specific config is hardcoded
- See Config drift

Pattern: Config Data defines config per environment

- The same codebase is used
- Only the config data differs
- Multiple Environments possible

Pattern: We don’t test

- Code is run
- We will see the errors when they happen
- Code is not tested

Pattern: Secrets are in code

- Our code / data is full of credentials
- Human readable
- Checked out on everybody’s laptop.
- They are placed in clear text on the servers
- They almost never change

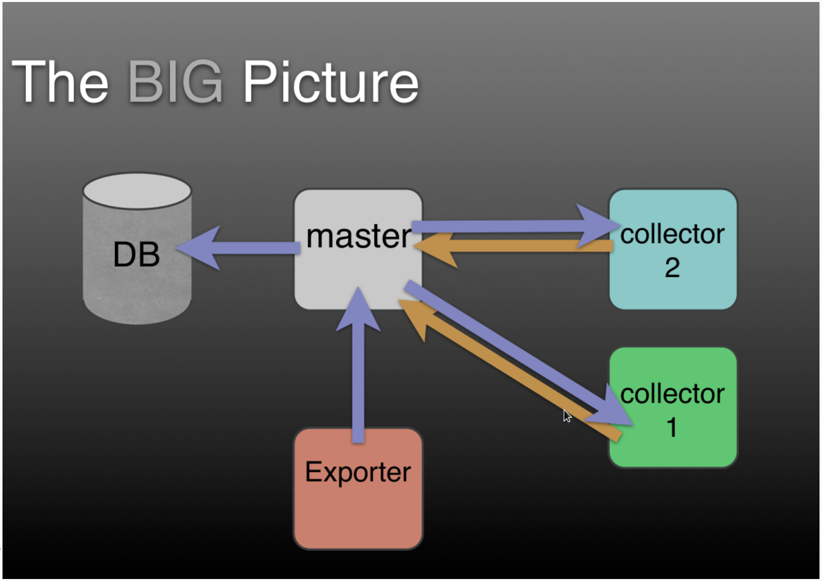
Pattern: Secrets are in a secure Store

- Our code looks up credentials from a central services
- Humans can’t read those credentials
- They get rotated automatically
- They are almost never stored on disk

Pattern: Single source of truth

- Every config run
- Collect information of the run,
- Store that information
- Be able to query a central source with live, correctly, machine generated data
- Reconfigure services based on changes in that data
- Full lifecycle
- Decomission instance => Automatically Clean out the data

SSOT : e.g Puppet Exported Resources



Pattern: No single source of truth

- Our Ansible playbook does a git commit to a config repo
- That repo gets checked out on the monitoring server
- Humans also modify this config
- Information is scattered over different tools

You can’t leverage the real power of automation , no dynamic reconfiguration of resources therefore configuration is often done manually , but through a tool

Pattern: Provisioning from a UI

“Blue printing”

Building a UI on top of a provisioning framework is typically an intermediate step before people realize they either need to build in all AAA functionality and buy a bloated tool.
Or realize git + openTofu solves these problems in a better way.

Typically a pattern like this is enforced by management.

Pattern: we ignore the existing (community) code

- We have written all code in house ?
- What do you mean there are modules on github to manage apache ?
- We spent about 3 months building our own
- Upstream Code Quality is an indicator

Pattern: We love Branching

- Aka We do NOT understand Continuous Integration
- We have mapped our environments to git branches
- Obviously they are long running
- We are cherry picking changes from uat to prod
- We don’t test those combinations
- We are in merge hell now

We have recreated config drift by design.

(Puppet r10k is an antipattern.) So is running your gitops from multiple branches.

About Open Source

- Most tools start out as Open Source
- Communities contribute modules /roles / recipes …
- Sometimes Communities get Betrayed

Lessons Learned

- All of these people say they are doing infrastructure as Code
- Some with 0 real benefit
- Find the patterns you like / dislike
- Implement or remove the ones you please
- My preferences might not fit your use case

Homework

Please stop clicking around and automate

Please think about how your application can be configured

Contact

Kris Buytaert

olly

kris@o11y.eu

https://o11y.eu

info@o11y.eu