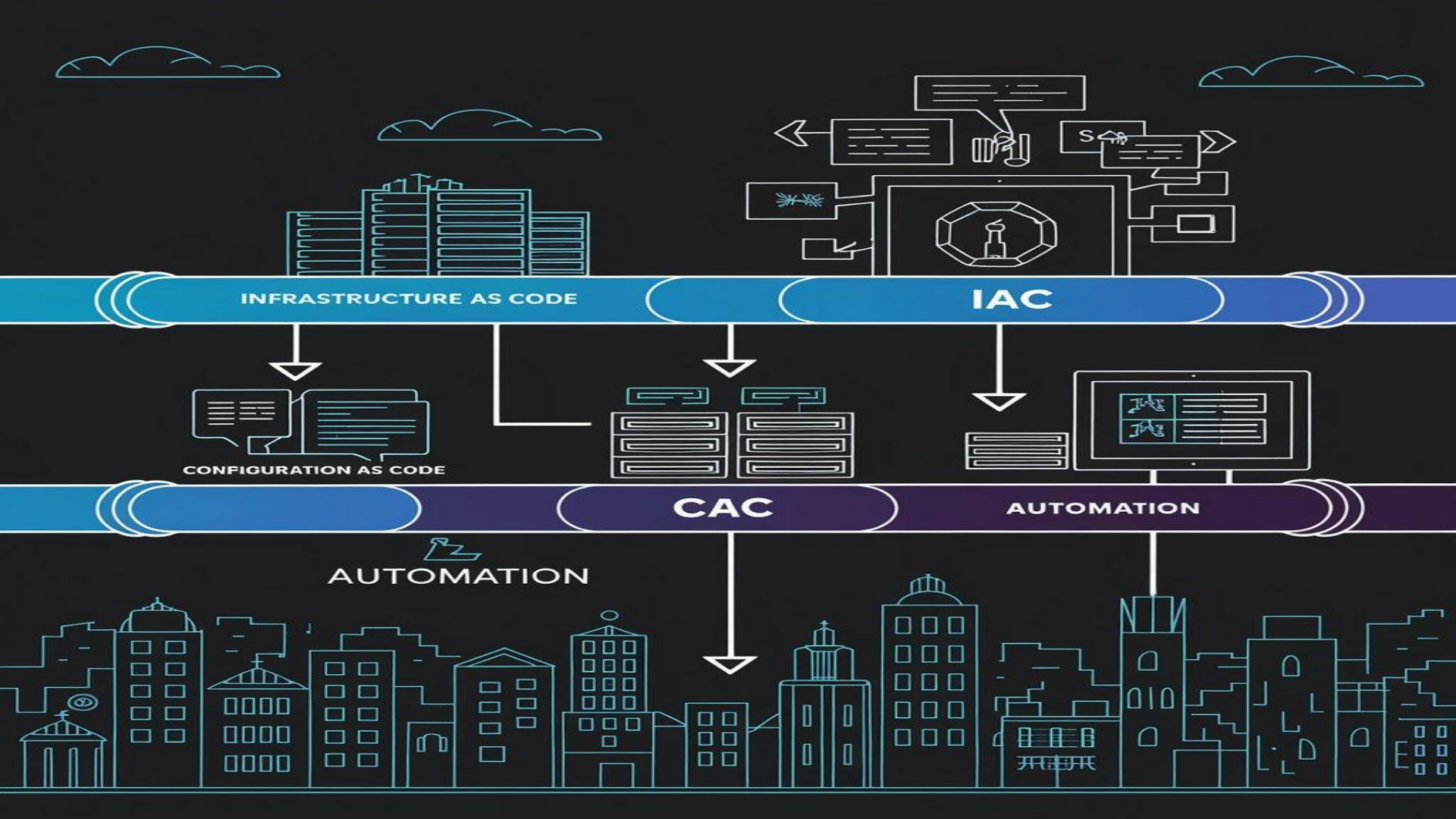

INFRASTRUCTURE AS CODE
IAC
CONFIGURATION AS CODE
CAC
AUTOMATION
AUTOMATION

About me

- Master Security & Network Engineering (OS3) @ UvA
- Worked on IT in different industries (~13 years);
  - Aviation, Marine, Transport, Defense, Retail
  - Topics; Architecture, Cloud, Infra, DevOps, Networking, Security
- Currently working for Sue;
  - Currently consulting at a defense company
  - Team lead for a group of 23 engineers
  - Focus lead on Cloud, Security & Observability

We are Sue

We are Navigators
Executed over **60 cloud migrations**

We are Pioneers
27 years of experience in digital transformation.

Certified Cloud & Automation Experts
870 expert certifications
350 cloud specialized certifications

We are Recognized
The **AWS Star Partner of the Year 2023**

We are Collaborators - Partners

aws
Google Cloud Platform
Microsoft Azure

We are Trusted
Over **200 customers**

https://sue.nl/

Adopting IaC and CaC

From scratch

Customer

- Customer environment consisted of **critical infrastructure** required for **transportation of goods** inside and outside the country.
- System is **linked to Customs** to facilitate the **checking/approving of imported and exported goods**.
- System should be **running 24/7**.
- Customers had **not adopted** any **DevOps** or **Cloud** practices.
- (Only) **1** Infra **engineer** and **2 developers**, due to reasons.....

The Environment

- Combination of baremetal and virtualized systems (on-premise).
  - Approximately **150 VMs** spread over production and acceptance.
  - About **20-25 Bare metal systems**.
  - Split over **2 Datacenters** (DC), but only **one DC** could run **production**.

- All **systems** where **deployed manually**.

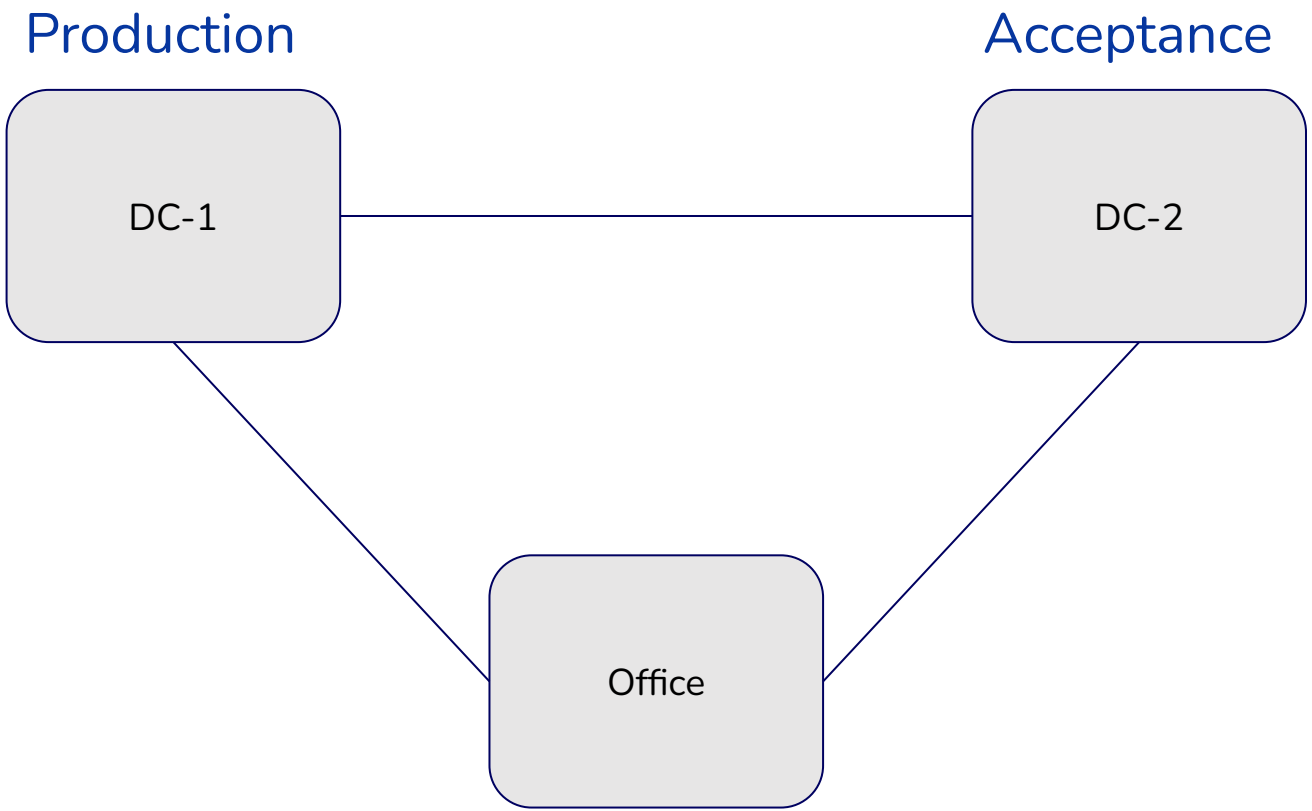
- **Applications where deployed by developers** manually on acceptance and production systems.

- Soft- and hardware was very outdated.
  - Oldest software around 199x ~ newest 2016/7
  - Oldest hardware 2001 ~ newest 2015

The Environment Cont'd

- Outdated and unsupported:
  - Networking Equipment (Cisco IOS 11/12/15)
  - Virtualization hard- and software (vSphere 4)
  - Database hard- and software (Oracle 9, 10 and 11)
  - Storage hardware (EMC, multiple major version behind)
  - Server hardware (HP and Cisco/Dell which where EOL)

- Application where running on:
  - Python 2
  - Java 8 (Tomcat 6, Springboot and Wildfly 11)


Production
Acceptance
DC-1
DC-2
Office

Q. Based on the information given, which problems and risks can you identify?

Problems and Risks

- Deploying new systems and software is (very) time consuming
  - Systems become/are ‘Pets’ (very quickly)
  - Cannot quickly deploy new systems in case of failures
    - This can affect problem/disaster recovery and uptime

- No proper lifecycle management of hard- and software
  - Security risks - e.g. vulnerable soft- & hardware
  - Compliance risks - e.g. GDPR or ISO27001
  - Maintenance risks - e.g. maintenance not routine / error prone

- Documentation becomes non-existent and/or outdated very fast
  - Onboarding people becomes lengthy and time consuming
  - Troubleshooting becomes difficult, will likely require expert/domain knowledge (of the system/software)

Problems and Risks Cont'd

- Old hard- and software makes adopting new technologies difficult.
  - Impossible to apply IaC (Unsupported or no API's)
  - Not possible to apply automation (easily) due to outdated/unsupported software/OS (e.g. python2)

- Positive feedback loop.
  - Due no automation/CM, documentation and DevOps practices
    - Maintaining old soft- and hardware will require more and more time.
    - Harder to find people who have the knowledge and skills to support and maintain the environment.
    - Less time to invest into updating and upgrading.
    - Less time on improving your environment and practices.

My impression when I started

Extracted from: https://www.thesun.co.uk/news/16643711/britains-most-disgusting-home-filthy-rubbish/

How did we get here?

- Multiple upgrade projects over several years that promised to fix everything.....
- But did not deliver...
- The current upgrade project (that would fix everything) was already 2 years behind, had not yet delivered any new services and where focusing on a **big bang** migration.....
- **Key takeaway:** *Big bangs are possible but you need to understand the environment really, **really** well. Upgrades using DevOps practices (e.g. continuous integration) provide value faster, risks are smaller/identified faster, etc. (and they are more fun)*

Q. Based on the given information where would you start?

Q. Based on the given information where would you start?

Talking to your stakeholders!

Talk to the stakeholders

Me: When will the upgrade project be finished?

Stakeholder: We don't know they haven't given a date.

Me: Can you ask me a question?

Stakeholder: Sure, which question?

Me: What is the current state of the system and can it still run for 3 years or maybe longer with acceptable risks?

Stakeholder: What is the current state of the system and can it still run for 3 years or maybe longer with acceptable risks?

Me: Bad and no.

Goals

- **Inventory** (and cleanup) of the environment
- Creating a (high level) **design/architecture** of the environment
- **Upgrade** hardware
- **Simplify**, **automate**, **shorten** and document **deployments**
- **Upgrade** (OS & Software) and **automate** configuration of **VMs**
- **Harden** systems
- **Adopt Cloud** services (where possible)

And now what?

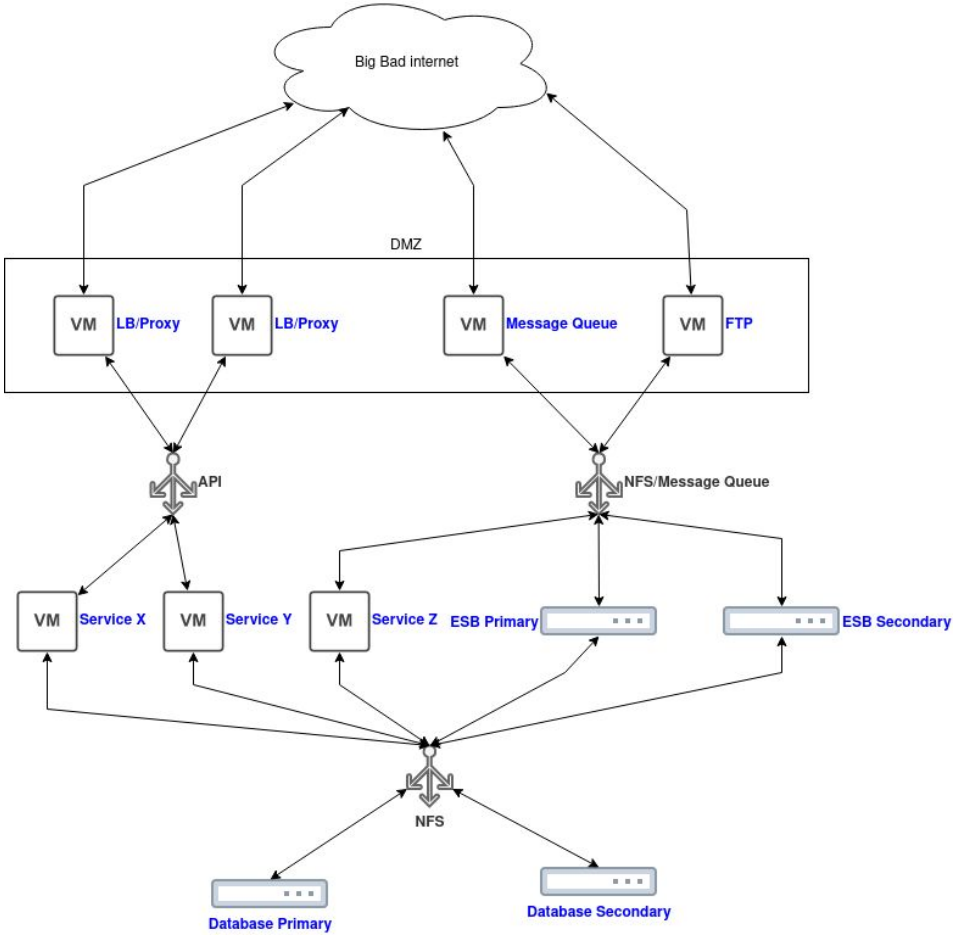
The thing (almost) everyone dreads……

Cleaning up (after other people)

Inventory of everything, everywhere, all at once

- Which systems do we have and what do they do?
- Which systems are still actually used?
  - Removed ~35 unused VMs
  - Removed ~10 unused Bare Metal systems
- (Re)build design (and architecture) based on the environment
- Identify the most critical items and items that block the adoption of DevOps practices
- Removed systems using the ‘piep’ model

Architecture



Q. What type of architecture did you see?

a. Microservices Architecture
b. Service Oriented Architecture
c. N-Tier Architecture
d. Event-Driven Architecture

Q. What type of architecture did you see?

a. Microservices Architecture

b. Service Oriented Architecture

c. N-Tier Architecture 

d. Event-Driven Architecture

The big five (infra perspective)

1. **Virtualization (cloud) stack**; EOL hard- & software, blocks adoption of IaC and Automation practices

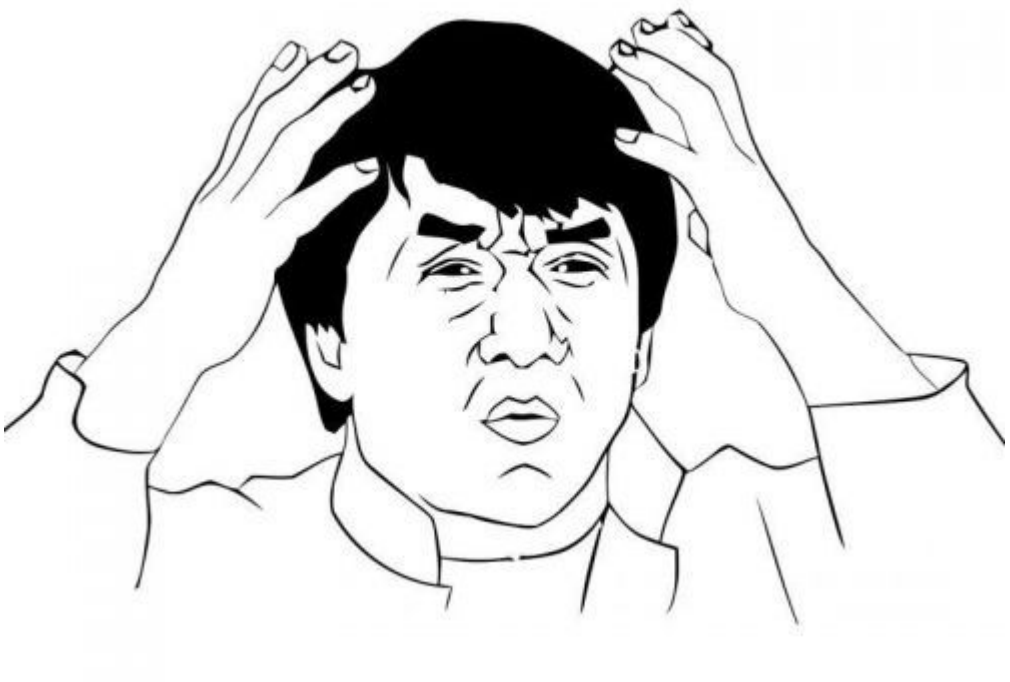
2. **Network Stack**; EOL hardware, no redundancy, most critical item to address for uptime

3. **Database(s)**: Outdated/EOL Most critical system, contains all data and a lot of business logic.

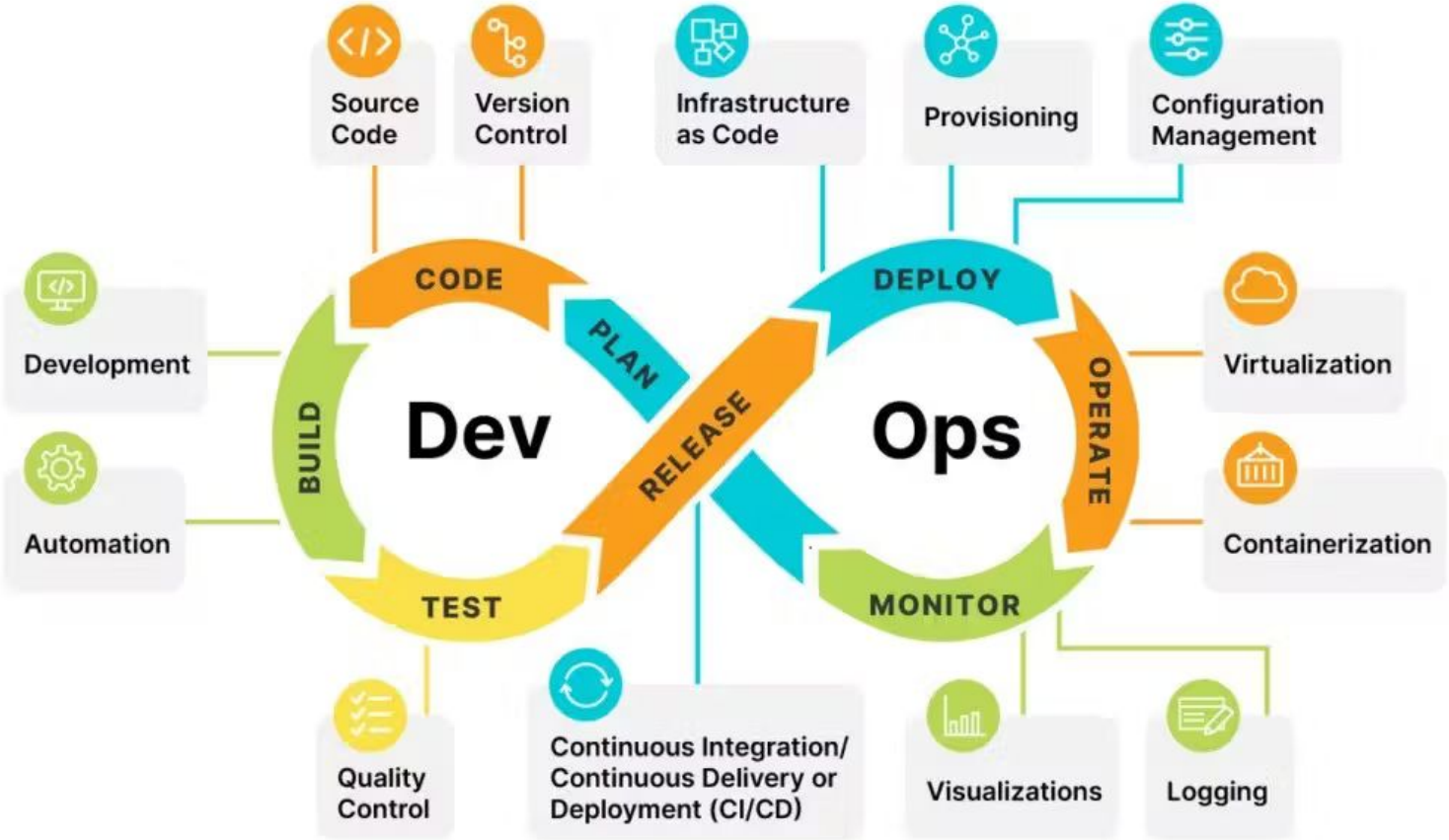
4. **Middleware/System Bus**; Outdated but specialized software suite with lots of custom scripting, 3 upgrade attempts have been done in the past, all failed....

5. **OS and Software**; Outdated/EOL, blocks adoption of Automation practices and blocks developers from using new software/tool(stacks)

No good choices (!?)



Retrieved from: https://imgflip.com/s/meme/Jackie-Chan-WTF.jpg


Source Code
Version Control
Infrastructure as Code
Provisioning
Configuration Management
CODE
PLAN
BUILD
Dev
RELEASE
Ops
DEPLOY
OPERATE
TEST
MONITOR
Development
Automation
Virtualization
Containerization
Quality Control
Continuous Integration/ Continuous Delivery or Deployment (CI/CD)
Visualizations
Logging

DevOps, IaC & Automation/CaC

- How, what and which DevOps practices do we adopt (first)?
  - All code must be stored in a VCS. **= Bitbucket**
  - All infrastructure must be build using code. **= Terraform**
    - **State is required**
  - Systems must be configured automatically. **= Ansible**
  - Deployments will be done using pipelines. **= Bitbucket pipelines**
  - The result of the pipeline shall always be a fully functional system/VM. **= Automate everything**

DevOps... how do we do that again?

- How, what and which DevOps practices do we adopt (first)?
  - No plain-text secrets in code. **= Hashicorp Vault/VCS Secrets**
  - Every deployment (from a pipeline) should yield the same result (per pipeline config). **= No pets, only cattle**
  - **Adopt DTAP**

Q. What is meant by ‘no pets, only cattle’?

a. *Systems are allowed to be unique*
b. *Rebuilding a node should always yield the same result*
c. *Systems need to be reproducibly build using code*
d. *Please don’t shoot my pet*

Q. What is meant by ‘no pets, only cattle’?

a. *Systems are allowed to be unique*

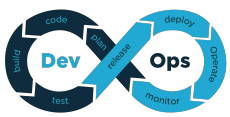
b. ***Rebuilding/replacing a node should always yield the same result***

c. *Systems need to be build using code*

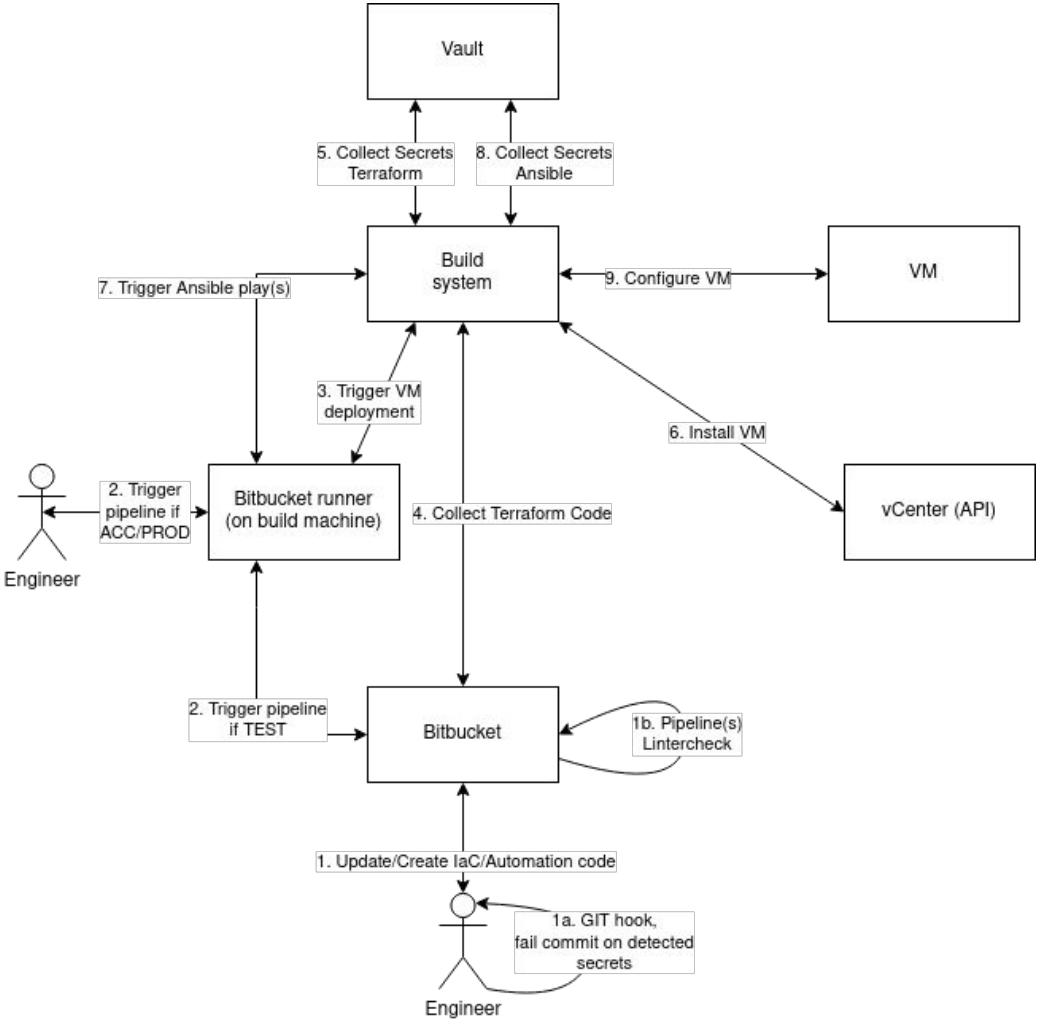
d. *Please don’t shoot my pet*

Upgrade & Automate first

- **Define an approach to automate deployment and configuration**
  - Minimize our deployment overhead and speed it up
- **Upgrade the virtualization stack**
  - This allows the adoption IaC
- **Automate the deployment and configuration (of all VMs)**
  - and upgrade the OS while you do this
  - and also fix redundancy where possible
  - Harden systems (using CIS) by default
- **Please do all this with minimal overhead (you have only 2 engineers actively working on it)**
  - And don’t break any production systems!


code
build
Dev
test
plan
release
deploy
Ops
operate
monitor

PLAN


Vault
5. Collect Secrets Terraform
8. Collect Secrets Ansible
Build system
7. Trigger Ansible play(s)
9. Configure VM
VM
3. Trigger VM deployment
6. Install VM
Engineer
2. Trigger pipeline if ACC/PROD
Bitbucket runner (on build machine)
4. Collect Terraform Code
vCenter (API)
2. Trigger pipeline if TEST
Bitbucket
1b. Pipeline(s) Lintercheck
1. Update/Create IaC/Automation code
1a. GIT hook, fail commit on detected secrets
Engineer

Almost forgot agile…

Sprints of 3 weeks

- Jira → Planning / estimating / tracking
- KISS is key → Focus on (extending) MVP
- Design and build in iterations → Spike first (on new functionality)

Sprint 1; Spike: how can we integrate Terraform & Ansible (automatically)

- Answer use a minimal cloud-init setup to configure SSH keys during creation of a VM

Sprint 2; **Build & design**

- Update design with cloud-init
- Engineer updates terraform code to deploy VMs
  - KISS: Just deploy a VM with a static SSH key
  - Test, review, release and tag code
- Other engineer updates automation (Ansible)/build process

Q. What is a Spike?

Q. What is a Spike?

A special type of user story that helps fill a gap in domain knowledge or knowledge for adopting a new technology.

This helps refining and estimating subsequent user stories!

VCS and Pipelines

- **Automate** the **creation** of your (**golden) images & build systems** (in case you require on-premise build systems)
  - We used Packer to create golden images!

- Use **variables in** your **pipelines**
  - Define layering in your variables if possible
    - E.g. workspace vs repository variables (Bitbucket)
    - E.g. Groups vs repository variables (Gitlab)

- **Linter Checking** & **Secret scanning** (pre-commit hook & Gitleaks)

- Secrets are **always** stored in VCS/pipeline secured variables
  - e.g. to access the vCenter API or Azure
  - This is not secret management!

Q. Which of the items mentioned on the last slide could be considered as part of ‘DevSecOps’?

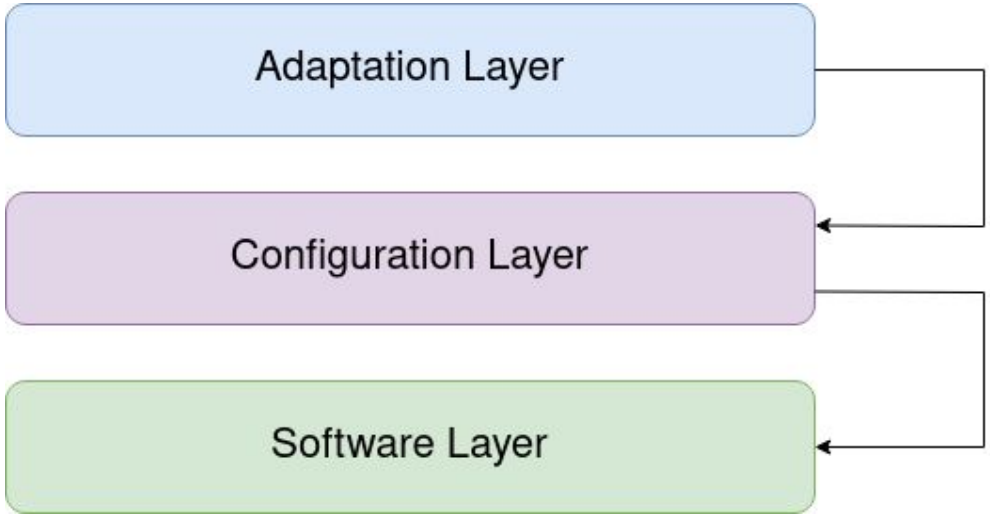
Q. Which of the items mentioned on the last slide could be considered as part of ‘DevSecOps’?

Secret scanning (pre-commit hook & Gitleaks), this make sure that you do not accidentally commit secrets to a (public) VCS repository.
Once a secret is committed it can take minutes for them to be found and exploited![1]

Store you secrets in ‘secure’ variables or even better use a secret manager like AWS KMS or Hashicorp Vault.

[1] https://tweakers.net/nieuws/218706/bmw-maakt-azure-server-met-private-keys-per-ongeluk-openbaar.html

Layered structure

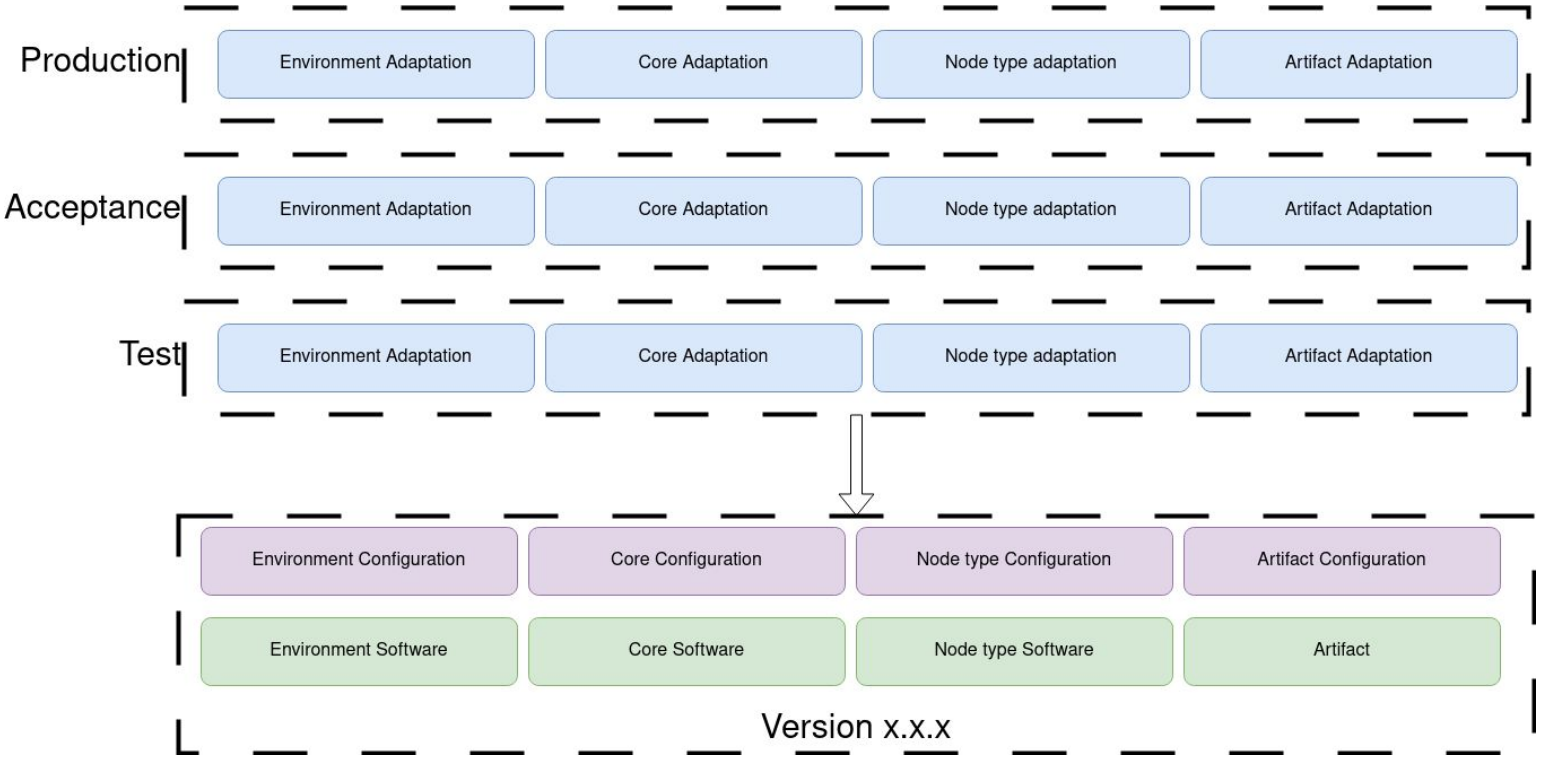


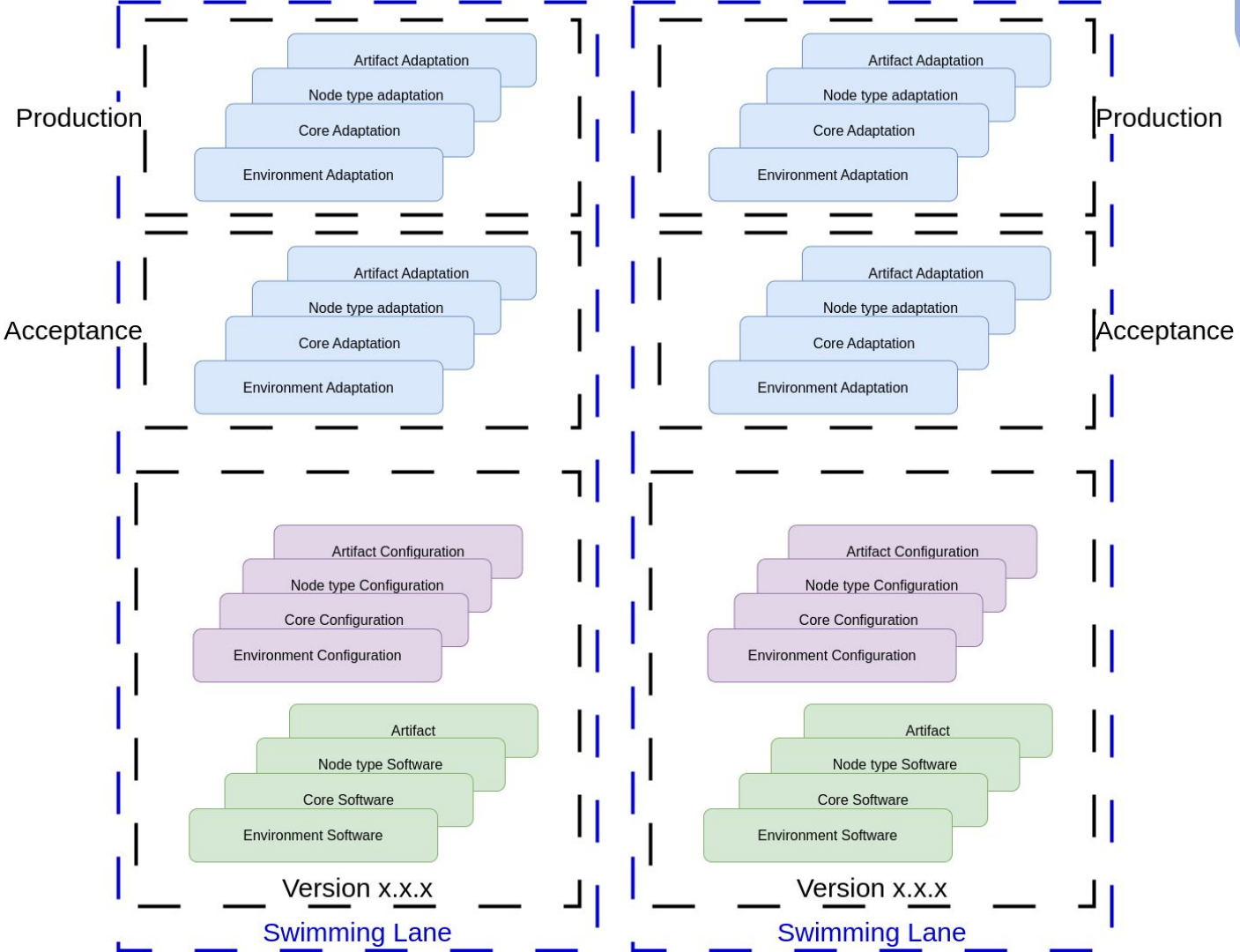
Typically a bunch of (nested) key:value pairs that detail how the configuration should be applied

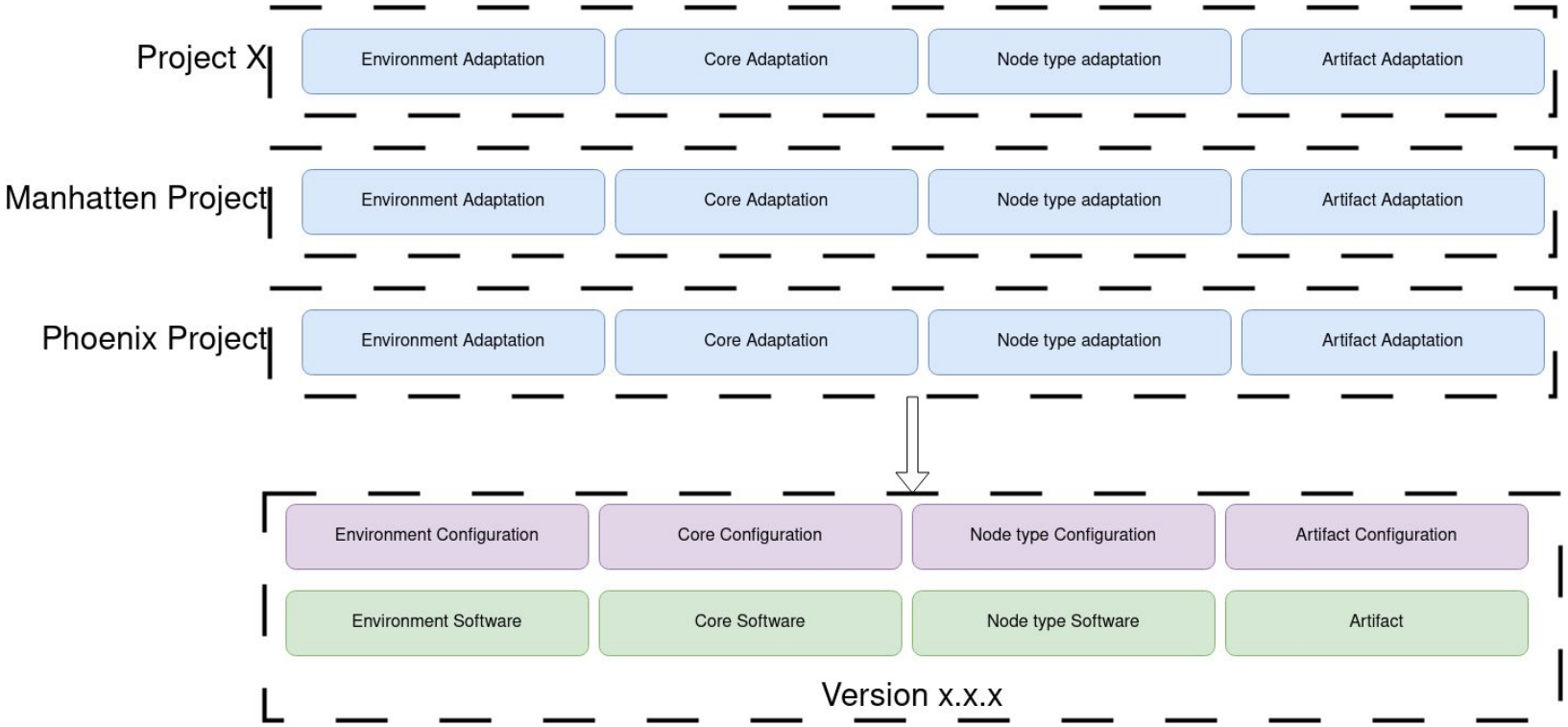
Configuration logic

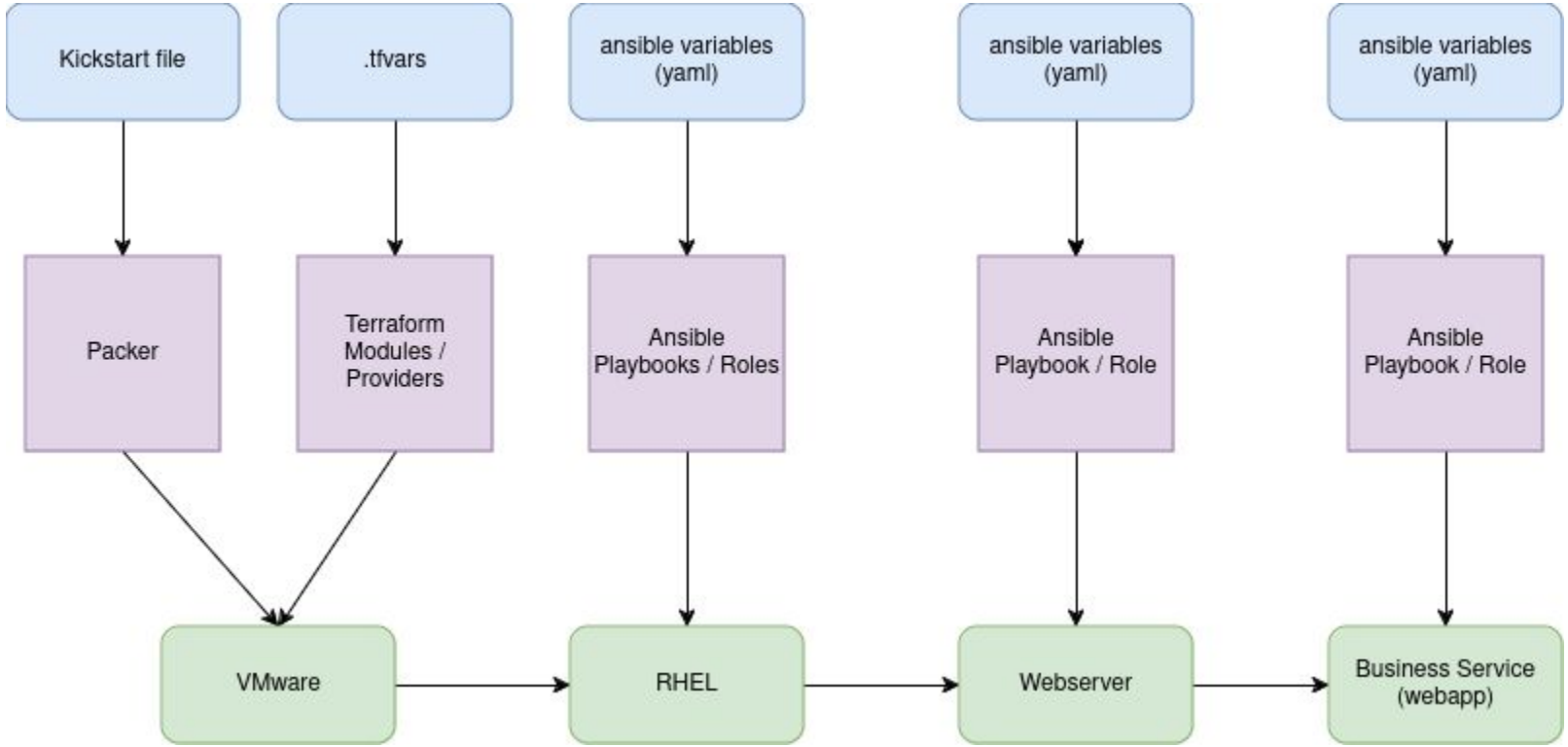
Software is configured and/or deployed based on detected environment in configuration layer & versions (in adaptation layer)

Layers and versioning




Production
Artifact Adaptation
Node type adaptation
Core Adaptation
Environment Adaptation
Acceptance
Artifact Adaptation
Node type adaptation
Core Adaptation
Environment Adaptation
Artifact Configuration
Node type Configuration
Core Configuration
Environment Configuration
Artifact
Node type Software
Core Software
Environment Software
Version x.x.x
Swimming Lane
Production
Artifact Adaptation
Node type adaptation
Core Adaptation
Environment Adaptation
Acceptance
Artifact Adaptation
Node type adaptation
Core Adaptation
Environment Adaptation
Artifact Configuration
Node type Configuration
Core Configuration
Environment Configuration
Artifact
Node type Software
Core Software
Environment Software
Version x.x.x
Swimming Lane


Project X
Environment Adaptation
Core Adaptation
Node type adaptation
Artifact Adaptation
Manhatten Project
Environment Adaptation
Core Adaptation
Node type adaptation
Artifact Adaptation
Phoenix Project
Environment Adaptation
Core Adaptation
Node type adaptation
Artifact Adaptation
Environment Configuration
Core Configuration
Node type Configuration
Artifact Configuration
Environment Software
Core Software
Node type Software
Artifact
Version x.x.x


Kickstart file
.tfvars
ansible variables (yaml)
ansible variables (yaml)
ansible variables (yaml)
Packer
Terraform Modules / Providers
Ansible Playbooks / Roles
Ansible Playbook / Role
Ansible Playbook / Role
VMware
RHEL
Webserver
Business Service (webapp)

Examples

Environment	Type	Example Delivery method
Software	Cloud/Virtualization/BM	Terraform Providers / Golden Image / (i)PXE
Configuration	Terraform / Foreman / MaaS / Cobbler / (etc.)	Terraform (code/module) / Cloud Init / Preseed / Kickstart
Adaptation	k:v pairs	(Global/Location) vars/.tfvars

Core	Type	Example Delivery method
Software	SSH package	RPM / DEB / MSI
Configuration	SSH config files	Ansible role/playbook
Adaptation	k:v pairs	Groupvars/all (+ Hostvars) / pillars

Q. What are providers when using Terraform?

Q. What are providers when using Terraform?

A provider in Terraform is a plugin that enables interaction with an API

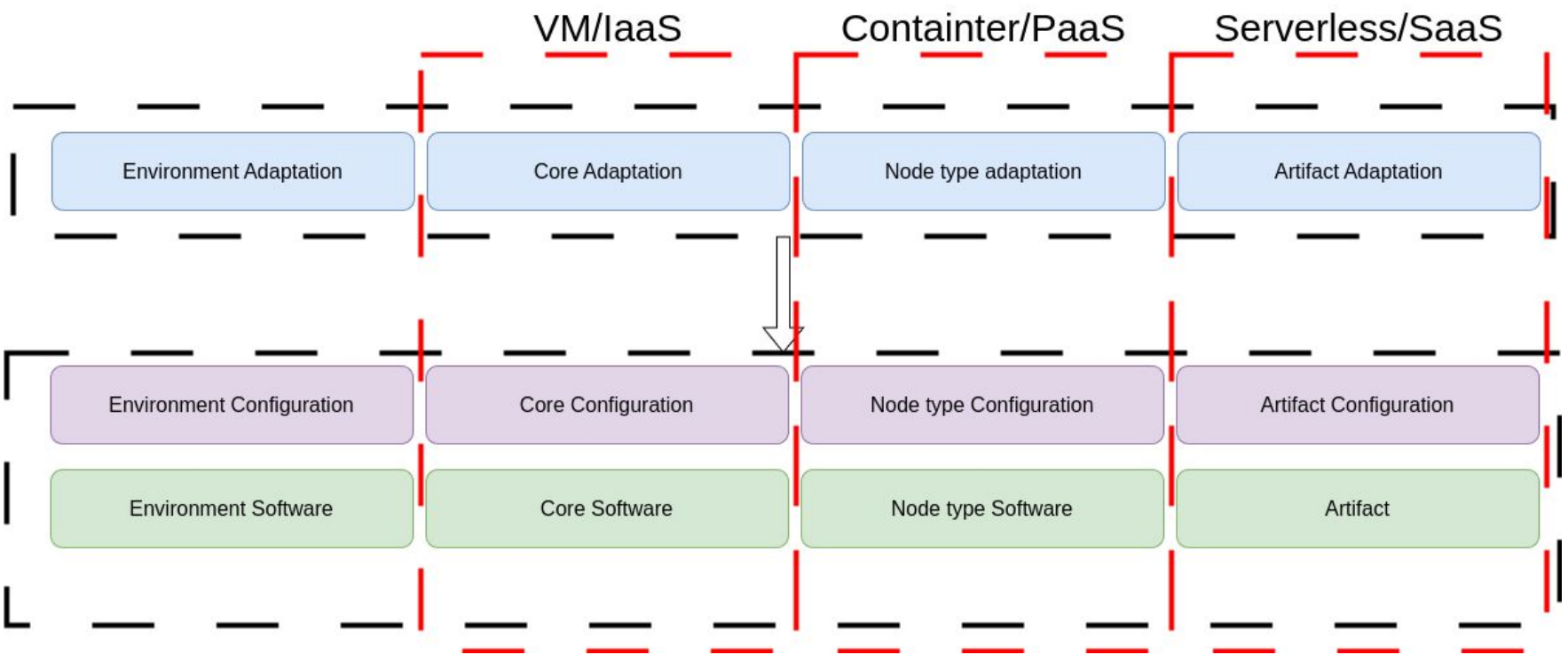
Examples Cont'd

Node type/Container	Type	Example Delivery method
Software	Apache package	RPM / DEB / MSI
Configuration	Apache config files	Ansible role/playbook
Adaptation	k:v pairs	Groupvars / Hostvars / pillars

Artifact/Serverless	Type	Example Delivery method
Software	Packaged Artifact	WAR / JAR / TAR
Configuration	Artifact config files	Ansible role/playbook
Adaptation	k:v pairs	Groupvars / Hostvars / pillars

Abstraction of Layers

- Layers can be abstracted away
- Core/OS configuration is abstracted away using **‘Managed services’** (for virtualization/containerization)
- Core/OS configuration and Node type configuration are abstracted away using **‘serverless’**
- Abstraction = reduced/transferred complexity to someone/something (you don't manage)
  - less flexibility
  - you tend to pay more (OPEX) for abstracting away complexity

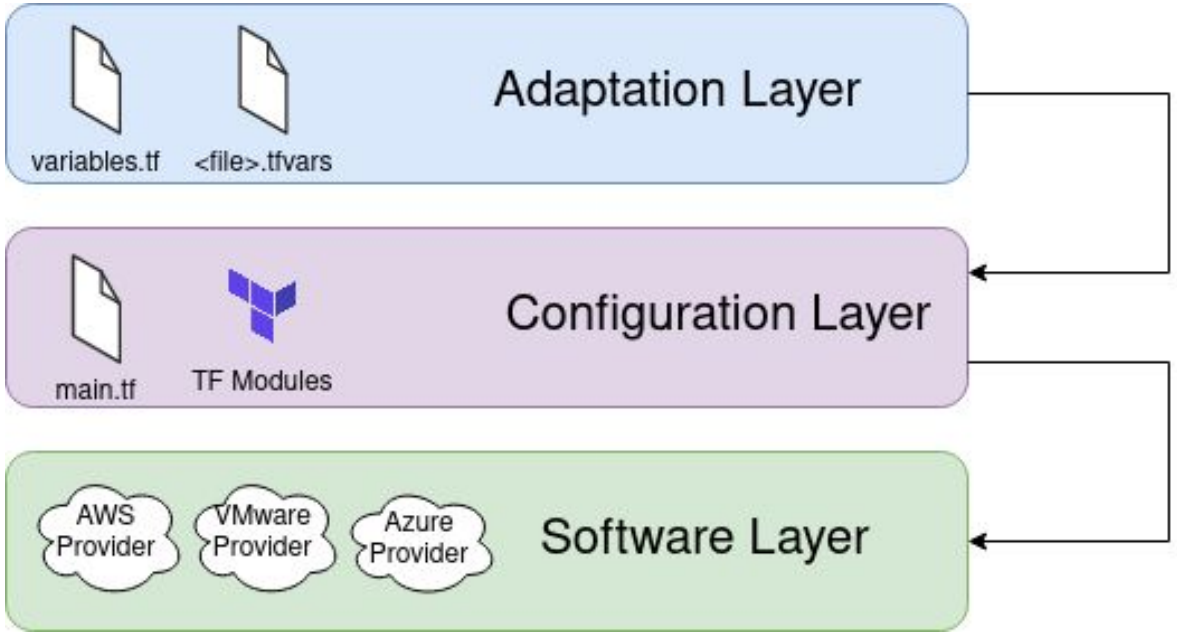

VM/IaaS
Containter/PaaS
Serverless/SaaS
Environment Adaptation
Core Adaptation
Node type adaptation
Artifact Adaptation
Environment Configuration
Core Configuration
Node type Configuration
Artifact Configuration
Environment Software
Core Software
Node type Software
Artifact

Q. Considering that abstraction of complexity is more expensive, why can cloud providers do this at (relatively) low cost?

Q. Considering that abstraction of complexity is more expensive, why can cloud providers do this at (relatively) low cost?

Due to scale, big cloud providers can optimize their resources in a way impossible to achieve for small- to medium sized companies.

IaC approach



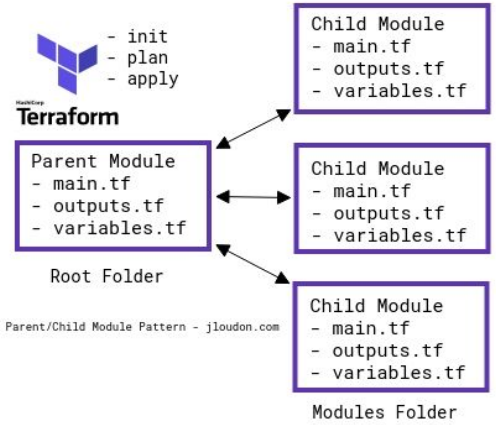
Create modules!

- Focus on creating **reusable modules**

- **Easier to maintain changes**
  - Link external versions to internal tags.
  - Create tags after changes (per module)
  - Only use ‘released’ versions/tags for production.

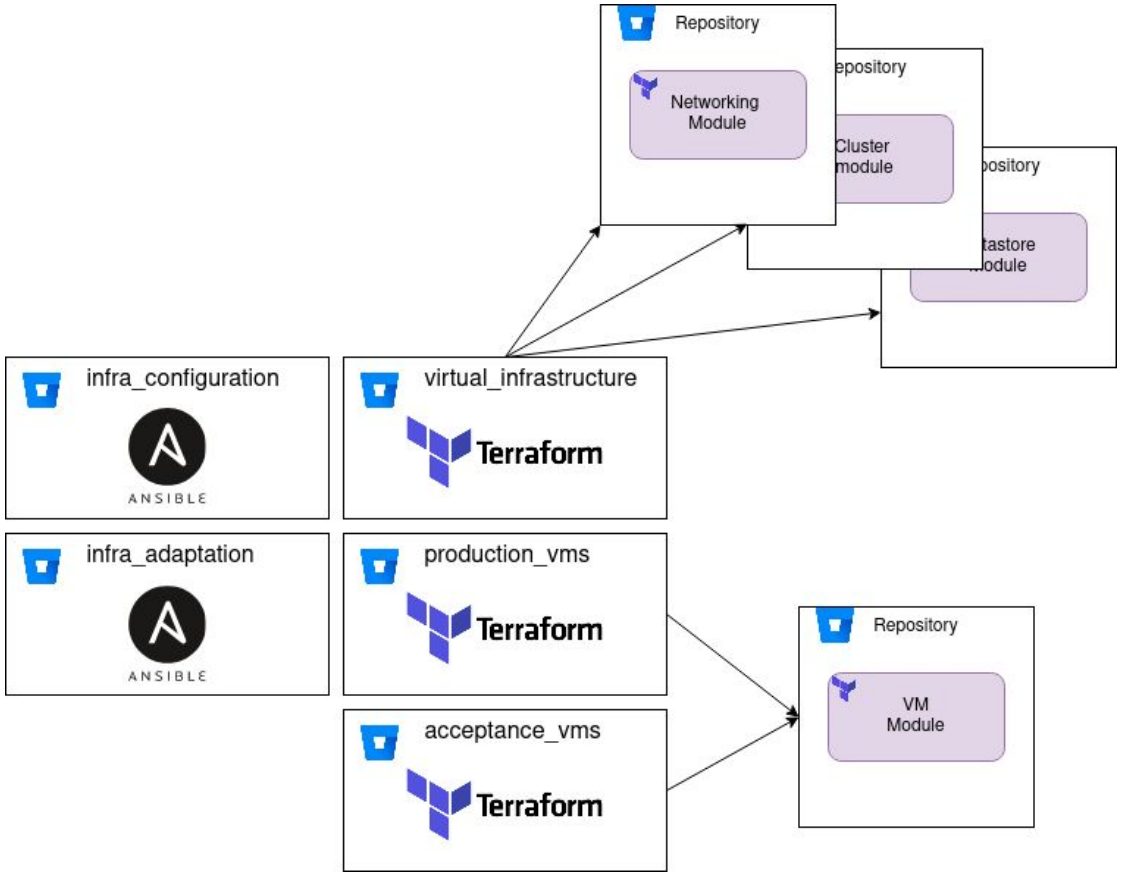
- **Easier to document**
  - Describe modules/logic apart from adaptation
  - Generate README per module (e.g. terraform-docs)

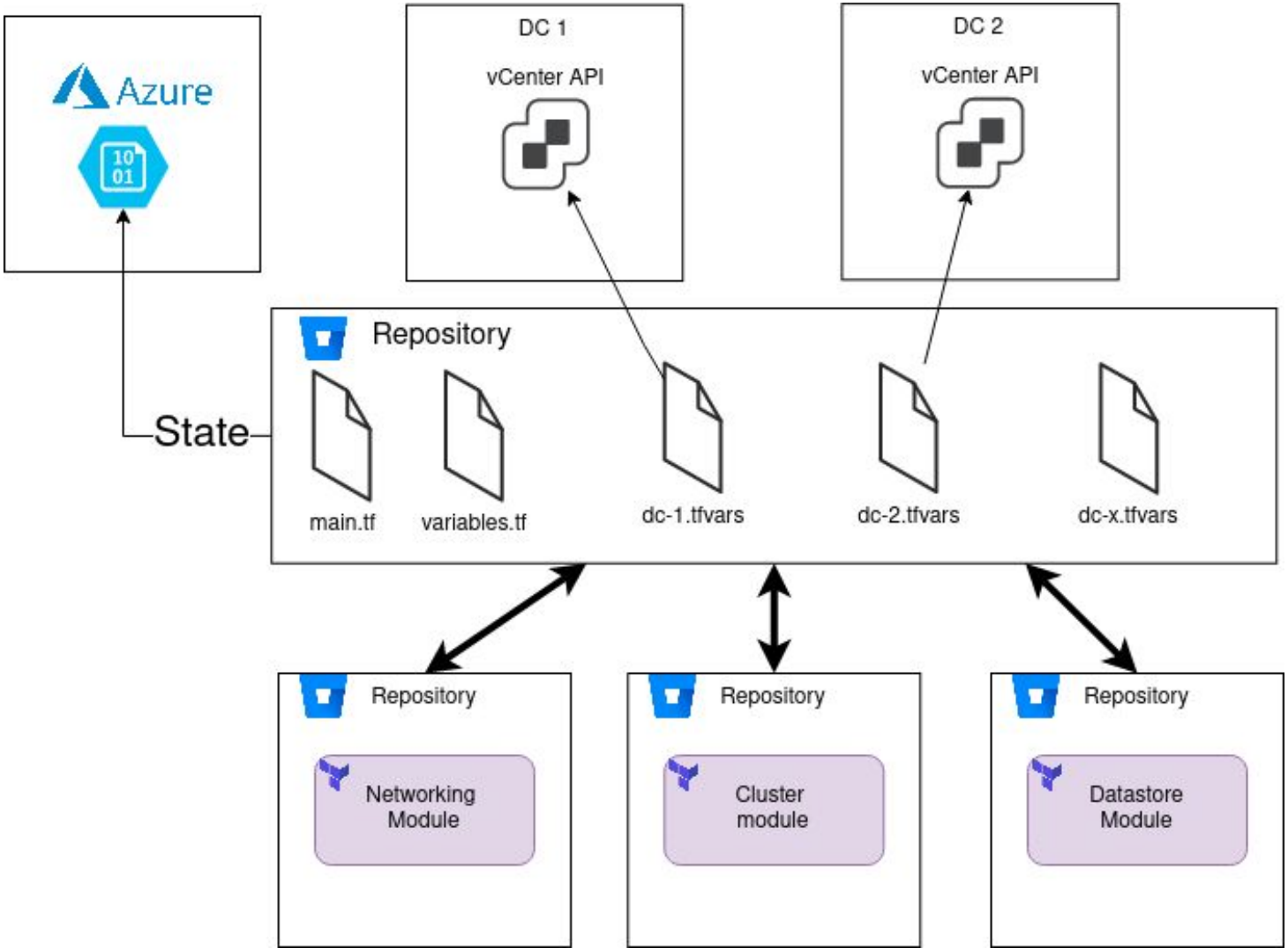
- **Easier to combine/reuse (adaptation)**
  - Variables could be used for different (Cloud) environments



Retrieved from:
https://miro.medium.com/v2/resize:fit:850/1*k057rHk8cvJjA2vuRaq_HQ.png

VCS Structure



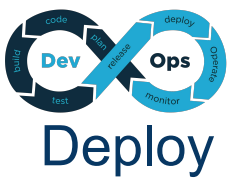

Azure
DC 1
vCenter API
DC 2
vCenter API
Repository
State
main.tf
variables.tf
dc-1.tfvars
dc-2.tfvars
dc-x.tfvars
Repository
Networking Module
Repository
Cluster module
Repository
Datastore Module

Pipeline: Virtual Infrastructure

- One pipeline to test/deploy/destroy all modules for virtual infrastructure

- Build mock resources using terraform code
  - Terraform init
  - Terraform fmt
  - Terraform plan
  - Terraform apply
  - Terraform destroy

- Create (git) tag on the updated modules after pipeline succeeds
  - The tag should identify a stable/working code base for a module



Deploy

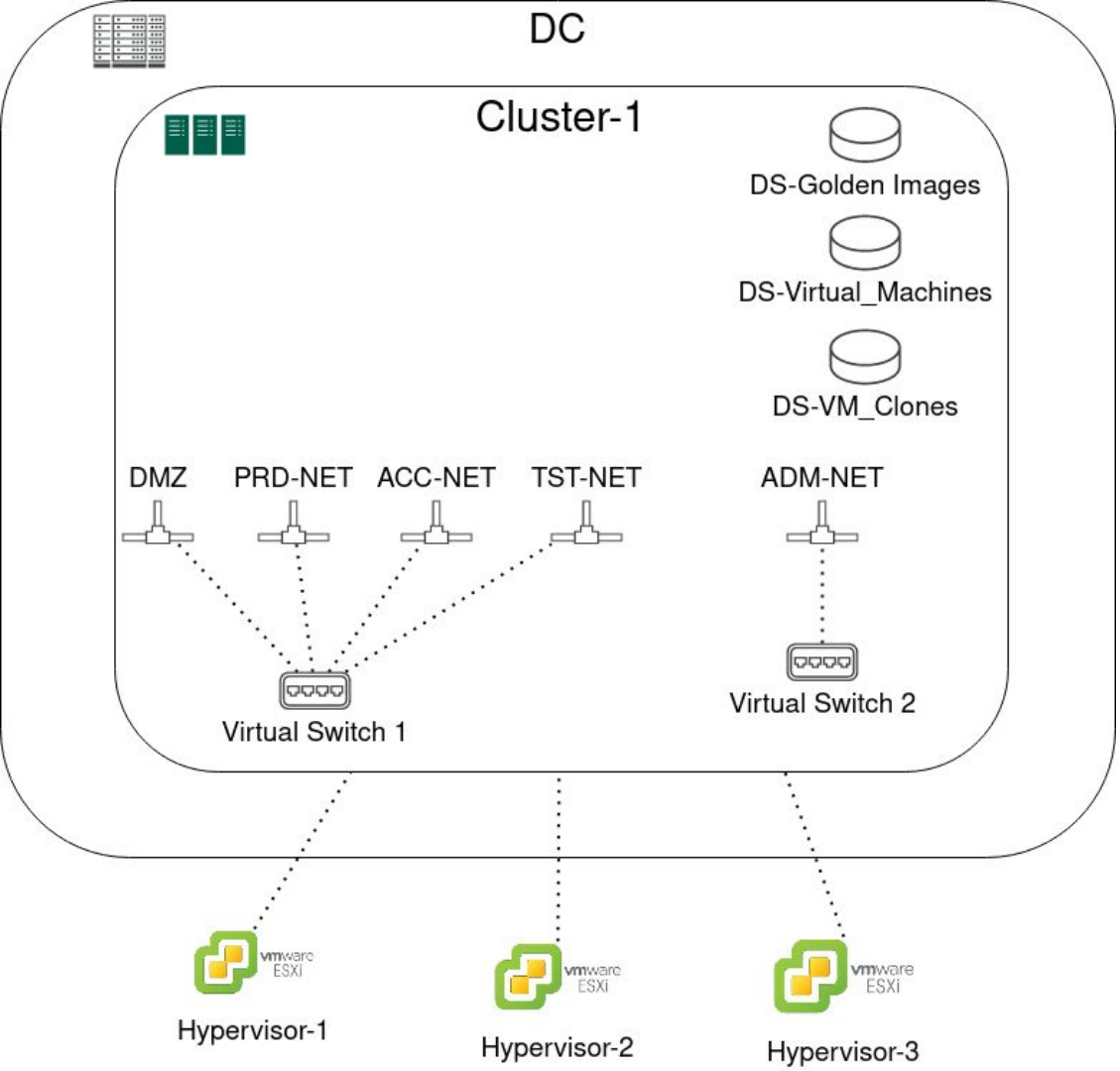
Pipeline: Virtual Infrastructure

- One pipeline to deploy/destroy the virtual infrastructure in a cluster located at a datacenter based on the selected .tfvars file
  - One .tfvars per cluster/region
  - Terraform **‘taint’** functionality to redeploy a virtual resource

- Boolean mechanism to force virtual infrastructure to be created/corrected on detected changes.
  - For production do manual verification first

- Virtual infrastructure tends to have a lot of dependencies between/to other resources.
  - Beware of (forcefully) destroying resources, this can break your environment!
  - Always check this for Acceptance and Production environments!

Before the pipeline

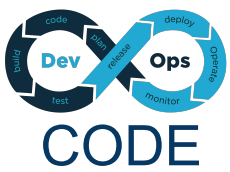

DC
Cluster-1
DS-Golden Images
DS-Virtual_Machines
DS-VM_Clones
DMZ
PRD-NET
ACC-NET
TST-NET
ADM-NET
Virtual Switch 1
Virtual Switch 2
Hypervisor-1
Hypervisor-2
Hypervisor-3

Pipeline: Virtual Infrastructure

- Periodically run the pipeline detect if manual changes were made to the virtual infrastructure
  - Most pipeline tools allow you to schedule a pipeline to periodically run

- Terraform keeps track of state, so you can check if changes were made to an environment

```
Note: Objects have changed outside of Terraform

Terraform detected the following changes made outside of Terraform since the last "terraform apply" which may have affected
this plan:

  # azurerm_resource_group.tf-plan has changed
  ~ resource "azurerm_resource_group" "tf-plan" {
        id       = "/subscriptions/c94b8366-e93b-45d7-9f43-e6afeb271625/resourceGroups/rg-tf-plan-example-centralus"
        name     = "rg-tf-plan-example-centralus"
      + tags     = {}
        # (1 unchanged attribute hidden)
    }


This is a refresh-only plan, so Terraform will not take any actions to undo these. If you were expecting these changes then
you can apply this plan to record the updated values in the Terraform state without changing any remote objects.
```



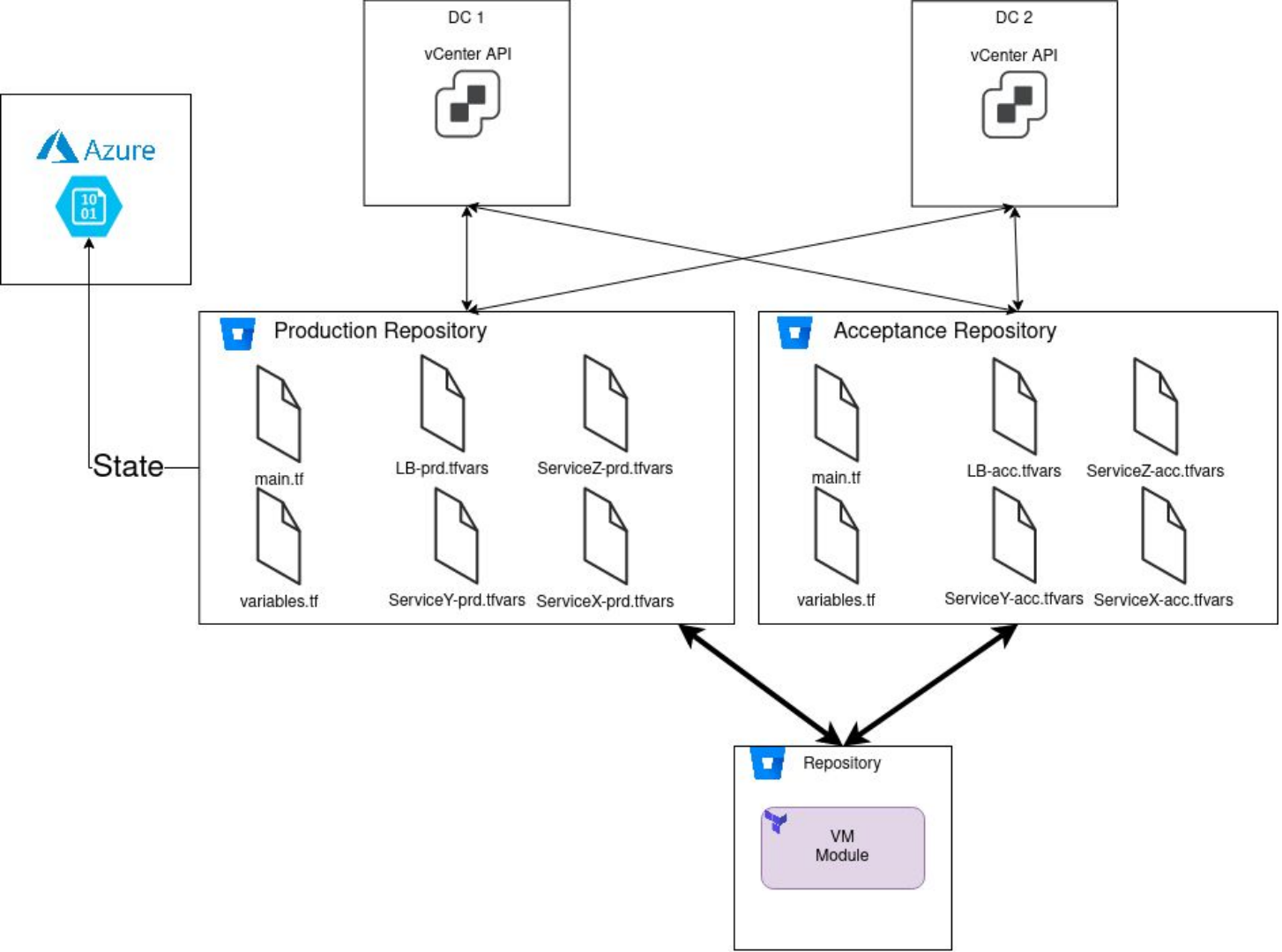
Creating VMs

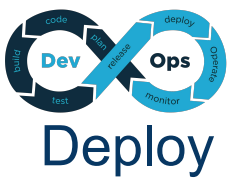
- Separate deployment on a per service basis, per environment
  - One **.tfvars** file per (clustered) service, per environment
  - Keeps deployments small and simple
  - Deploy (and configure) services separately
    - Simplifies upgrade and migration
    - No big bang, but constant improvements

- The same Terraform module is used for all deployments

- Fix order of the deployments with orchestration
  - e.g. A pipeline that can call other pipelines
  - Deploy backend before frontend

- State helps here!


DC 1
vCenter API
DC 2
vCenter API
Azure
State
Production Repository
main.tf
LB-prd.tfvars
ServiceZ-prd.tfvars
variables.tf
ServiceY-prd.tfvars
ServiceX-prd.tfvars
Acceptance Repository
main.tf
LB-acc.tfvars
ServiceZ-acc.tfvars
variables.tf
ServiceY-acc.tfvars
ServiceX-acc.tfvars
Repository
VM
Module



Pipeline (fase 1): build VM

- The name of the tfvars file matches a state file in a Azure blob store using a ***pipeline variable***

- The name can be selected via a dropdown menu when you run the pipeline.
  - e.g. serviceX = serviceX.tfvars & serviceX.tfstate

- Some initialization values during the creation using cloud-init.
  - e.g. set a unique (strong) password for root account (for debugging)
  - add SSH keys of build machines (for Ansible)

One file to deploy them all

One file to find them

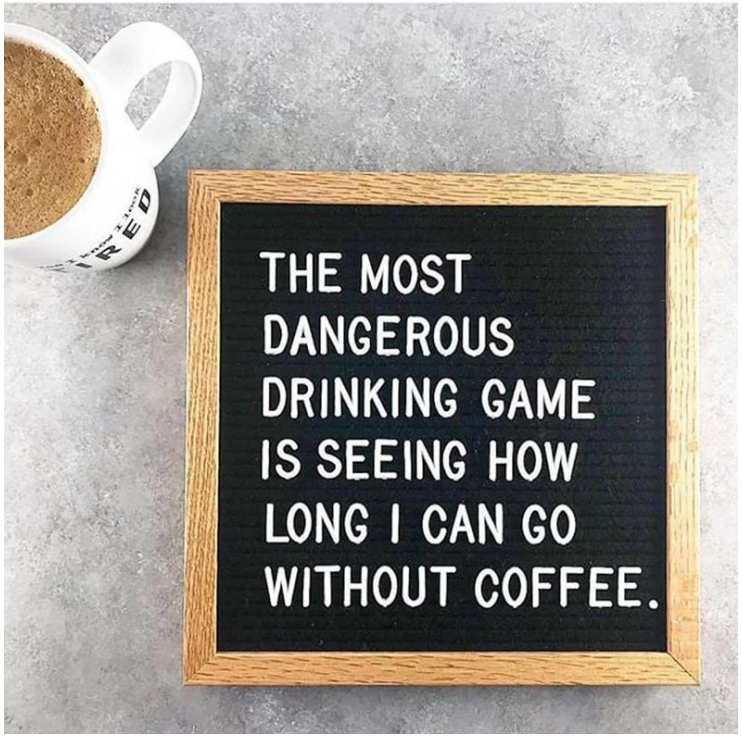
One file to bring them all

and in **deployment** bind them

Example of main.tf

```
module "tf_module_vsphere_virtual_machines_template" {
  source = "git::ssh://git@bitbucket.org/tf_module_vsphere_virtual_machines_template?ref=1.9"

  vsphere_user       = var.vsphere_user
  vsphere_password   = var.vsphere_password
  vsphere_server     = var.vsphere_server
  vsphere_datacenter = var.vsphere_datacenter

  vm_settings = var.vm_settings

}

module "tf_module_vsphere_compute_cluster_groups" {
  source = "git::ssh://git@bitbucket.org/tf_module_vsphere_compute_cluster_groups?ref=1.2"

  vsphere_cluster_id = module.tf_module_vsphere_virtual_machines_template.vsphere_compute_cluster_id
  vsphere_host_id = module.tf_module_vsphere_virtual_machines_template.vsphere_host_id
  vsphere_virtual_machines = module.tf_module_vsphere_virtual_machines_template.vsphere_virtual_machines

  depends_on = [ module.tf_module_vsphere_virtual_machines_template ]
}
```

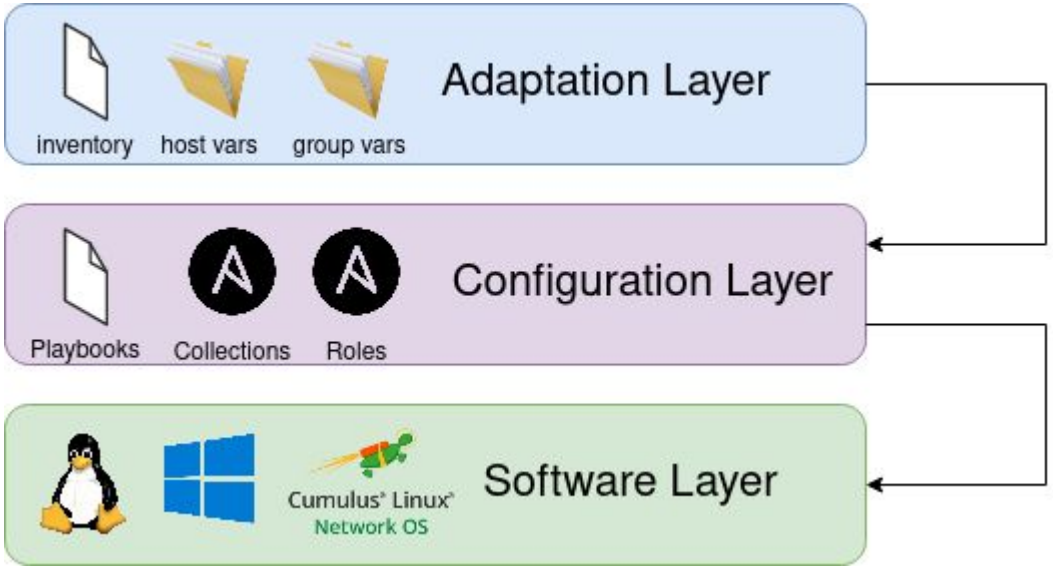
Coffee break (for 455 seconds)



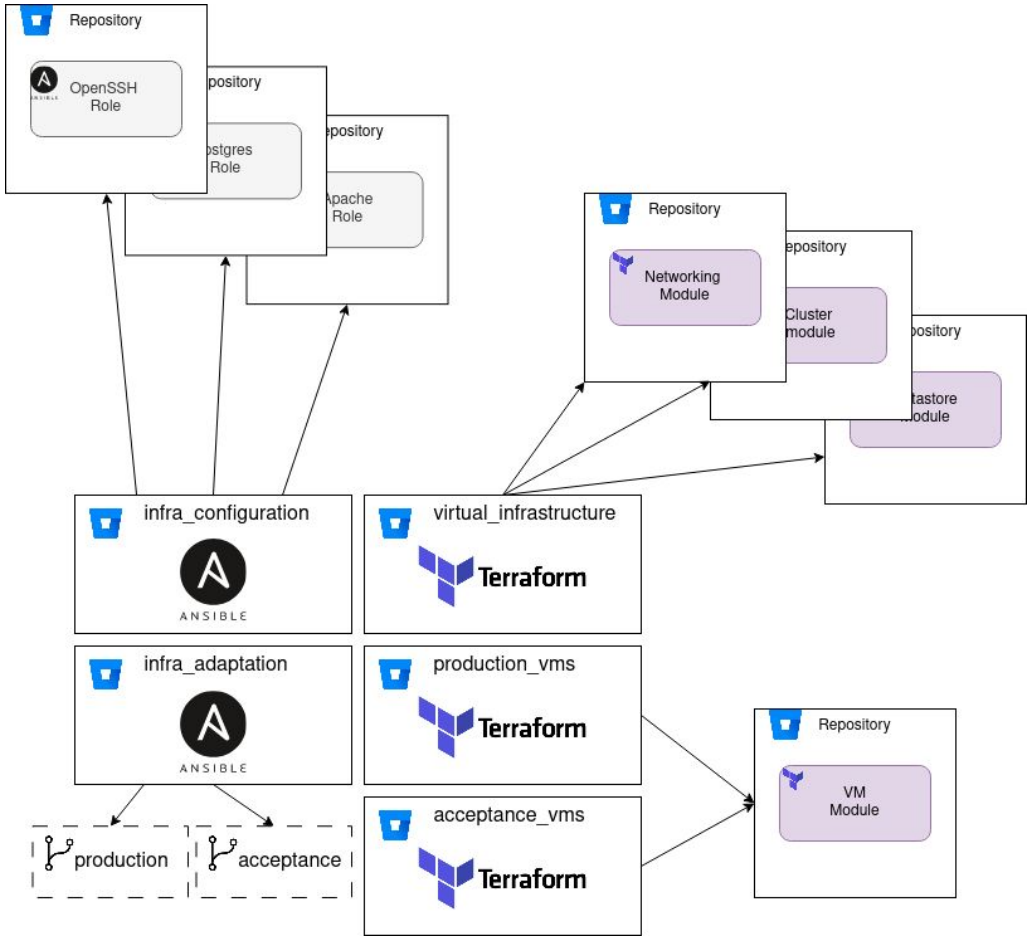
Retrieved from:
https://media1.popsugar-assets.com/files/thumbor/8FCM1quVrcD3B7lvLSzMJv0P7Xk=/fit-in/792x792/filters:format_auto():upscale()/2018/07/27/697/n/1922729/2cb349075b5b3defafdd45.25339740_.jpg

And now.....
We have a clean/baseline VM

Automation/Configuration Management



VCS Structure



Ansible layout

- Storing the inventory (adaptation) apart from your configuration allows for more flexibility
  - Configuration logic can be versioned separately.

- Once playbooks, roles, etc (configuration logic) have been created, they tend to change less often
  - Usually only with Major release changes e.g. migrating from postgres 10 to 14

- Adaptation; e.g. I want database user X and Y to be created with access to db/table Z to be created
  - These are stored in the inventory as variables.

Handling Ansible dependencies

- Internal vs external roles (Ansible Galaxy/Public VCS)
  - Don’t write yourself what you can get online
  - **Advantage**: shorter development time
  - **Disadvantage**: build up knowledge/dependency of/on external code
  - **Disadvantage**: security risk! (basically it’s unprotected seks with the internet)

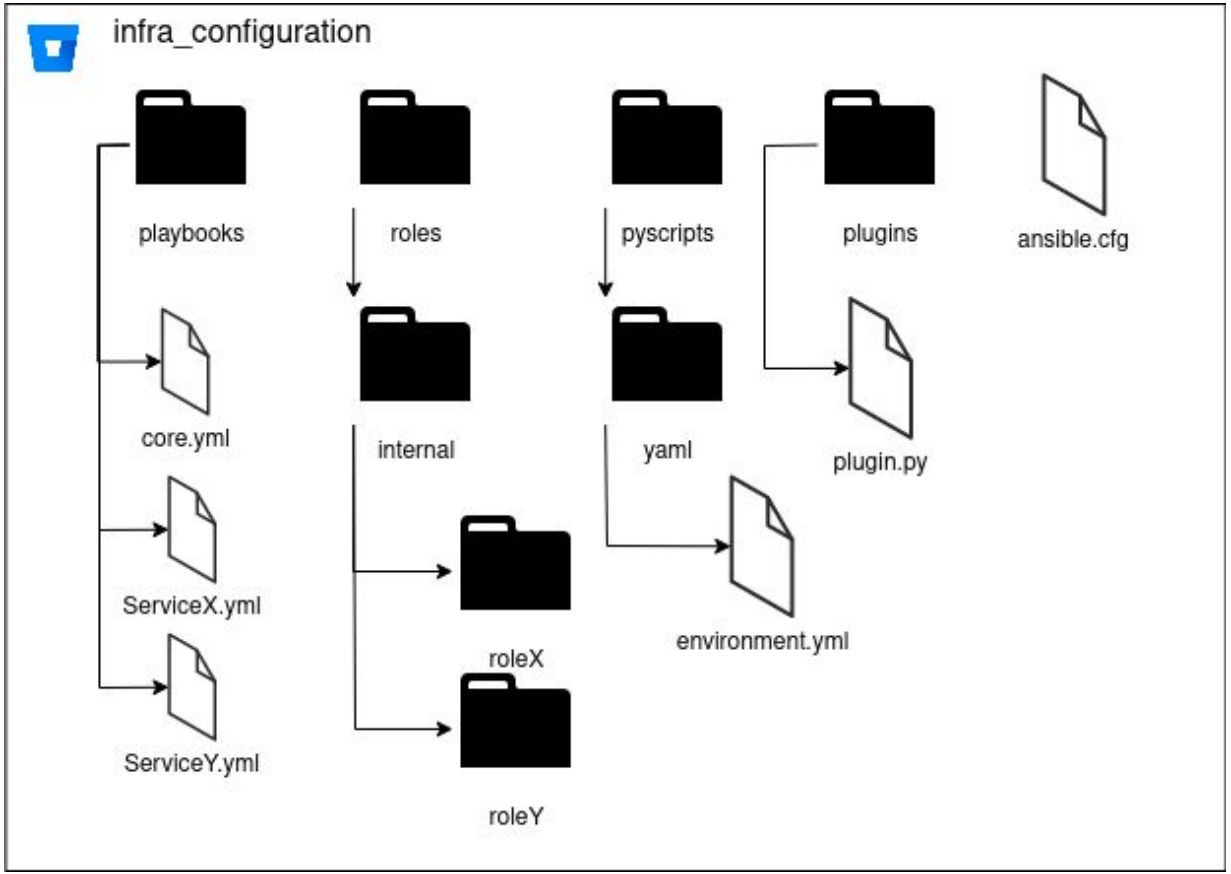
- Storing external roles in our own VCS, always verify if code is safe
  - Create a tag of the original code
  - In case changes are required tag again
  - Ensures you **always** have a working code base
    - Public repo’s/roles are sometimes removed!
  - Easy to track changes you made to external code
  - OpenSSH_<original_version>_<unique_tag>_<our_version>

```
epel_version: original_4.0.1_cst_1.0
openssl_version: original_2.2.7_cst_1.0
python_pip_version: original_4.3.3_cst_1.0
openssh_version: original_4.2.2_cst_1.0.0
snmpd_version: original_1.1.5
selinux_version: original_3.1.6
umask_version: original_2.1.4
httpd_version: original_7.3.5_cst_1.0
tomcat_version: original_5.3.6_cst_1.14
service_version: original_3.1.4
autofs_version: original_2.1.5_cst_1.1
bootstrap_version: original_6.1.3_cst_1.0
buildtools_version: original_3.1.11_cst_1.0
drbd_version: original_2021.04.23_cst_1.3
freeipa_version: original_1.9.2_cst_1.0
geerlingguy_java_version: original_2.3.0_cst_1.0
```

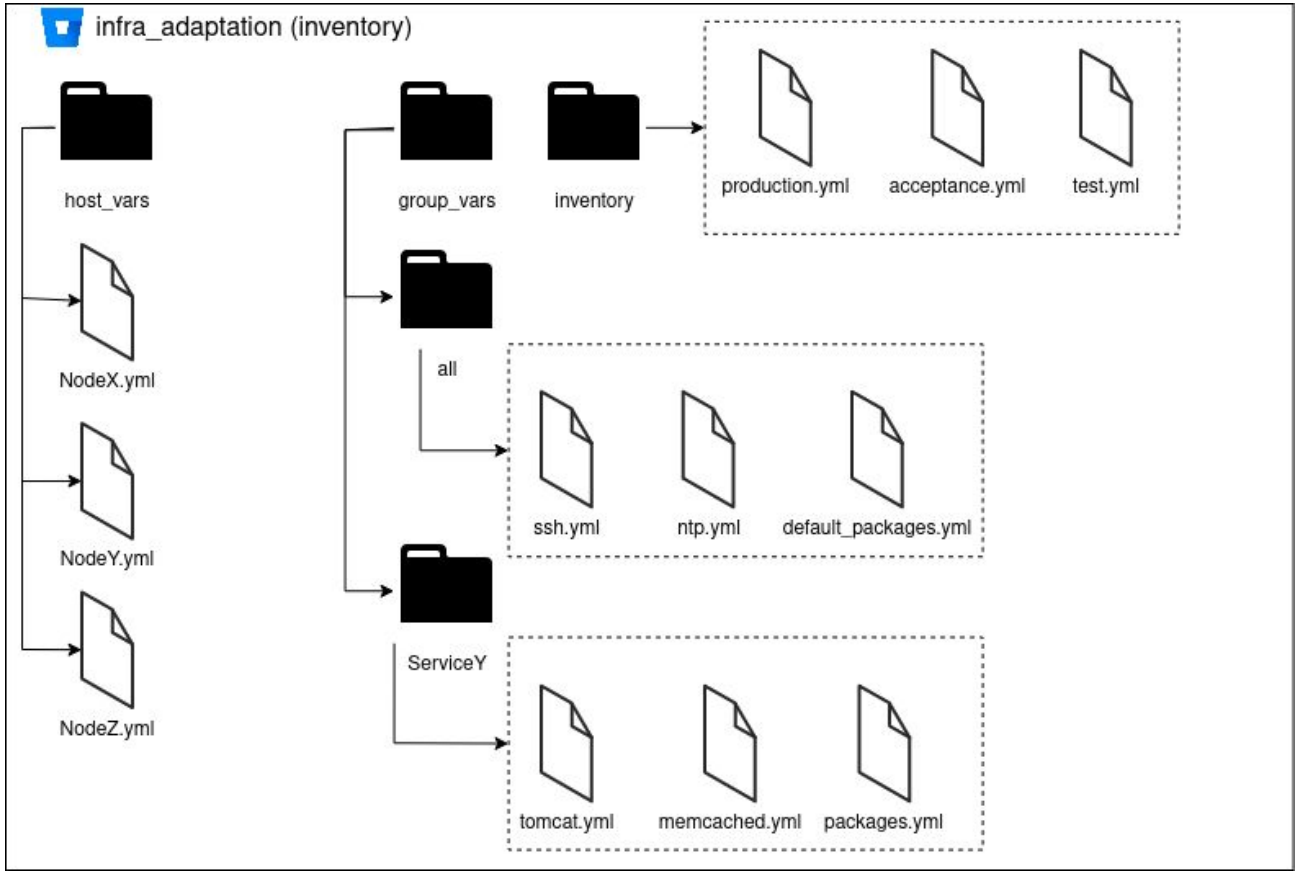
1

2

VCS Structure: Configuration



VCS Structure: Adaptation



Q. What does the ‘all’ directory do?

Q. What does the ‘all’ directory do?

The all directory is a ‘special’ directory that can be added to the group vars. Files that contain variables in this directory can be used for ***all*** *resources specified in the inventory. Its a good way to provide defaults for all resources, these can be overridden in other group var or host var files.*

https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html#default-groups

Q. How would you setup your Ansible inventory?

Ansible Inventory strategies

- Create a inventory per environment
  - Acceptance, Production, Test
  - PropjectX, ProjectY, ProjectZ

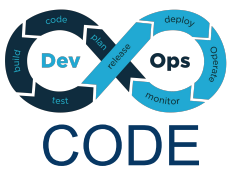
- In case there are a lot of differences between the environment
  - Use separate repositories

- In case environments that are (almost) identical
  - Use branches

https://docs.ansible.com/ansible/latest/tips_tricks/sample_setup.html#sample-setup

https://ansible-tips-and-tricks.readthedocs.io/en/latest/ansible/inventory/

https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html#how-variables-are-merged



Configuring VMs

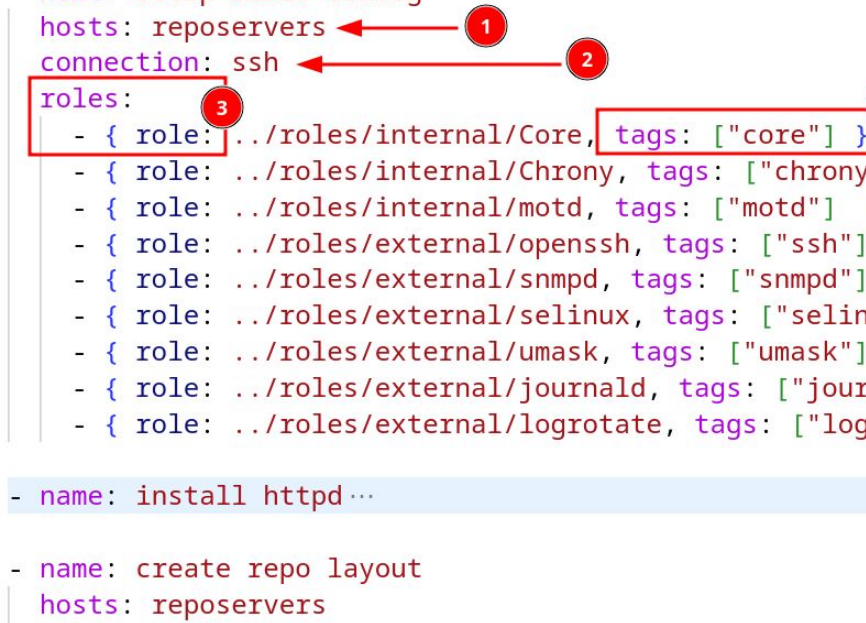
- Create a internal (set of) roles that handles core configuration.
  - Use ‘external’ roles when possible
  - Create new plays/roles when needed

- Create a playbook per ‘service’
  - Keeps configurations small and simple
  - Deploy (and configure) services separately
    - Simplifies upgrade and migration
    - No big bang, but constant improvements

- Provide sane defaults for **required** variables, **omit optional** variables

https://github.com/Mandorath/ansible_workshop_101_sue

```yaml
- name: setup basic config
  hosts: reposervers
  connection: ssh
  roles:
    - { role: ../roles/internal/Core, tags: ["core"] }
    - { role: ../roles/internal/Chrony, tags: ["chrony"] }
    - { role: ../roles/internal/motd, tags: ["motd"]  }
    - { role: ../roles/external/openssh, tags: ["ssh"]  }
    - { role: ../roles/external/snmpd, tags: ["snmpd"]  }
    - { role: ../roles/external/selinux, tags: ["selinux"]  }
    - { role: ../roles/external/umask, tags: ["umask"]  }
    - { role: ../roles/external/journald, tags: ["journald"]  }
    - { role: ../roles/external/logrotate, tags: ["logrotate"]  }

- name: install httpd ···

- name: create repo layout
  hosts: reposervers
  connection: ssh
  vars:
    file_operations: "{{ repo_structure }}"
  tasks:
    - name: Creating repo dir structure
      ansible.builtin.include:
        file: ../roles/internal/Core/tasks/file.yml
    - name: Allow apache to serve in /repo
      community.general.sefcontext:
        target: '/repo(/.*)?'
        setype: httpd_sys_content_t
        state: present
```



```
## This role allows you to define a set of variables to create groups.
## It accepts all values normally available in the group module.
## In case a value is not specified the module will omit the key to preven
## The variable will combine multiple variables when specified, these are:
## default_group <--- specified in the group_var/all
## group <--- specified in the group
## host_group  <---- specified per host
# https://docs.ansible.com/ansible/latest/collections/ansible/builtin/grou

- name: Combine default and group defined users. ···

- name: Combine group and host var users ···

- name: testing variables ···

- name: create group
  ansible.builtin.group:
    name: "{{ item.name }}"
    gid: "{{ item.gid | default(omit) }}"
    local: "{{ item.local | default(omit) }}"
    state: "{{ item.state | default('present') }}"
    non_unique: "{{ item.non_unique | default(omit) }}"
    system: "{{ item.system | default(no) }}"
  loop: "{{ default_group }}"
  loop_control:
    label: "{{ item.name }}"
  when: default_group is defined
```

Test/Release

Pipeline (fase 2): Test Ansible

- Testing of ansible code could be done using ‘test’ VMs
  - Use fase 1 to deploy test VMs
- Run pipeline fase 2 during development.
  - Once testing is finished
- Ansible roles are tested using Molecule
  - We created technical debt here, we did not start with adding these tests to all roles
  - When writing your own modules you can test modules using ansible-test units
- Generate Documentation

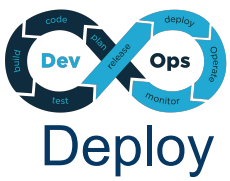
https://docs.ansible.com/ansible/latest/dev_guide/testing_units.html

Q. What is technical debt?

Q. What is technical debt?

Technical debt is the accumulation of suboptimal or incomplete solutions in a software project that can slow down the development process and compromise the quality of the product.

Retrieved from: https://asana.com/resources/technical-debt



Deploy

Pipeline (fase 2): configure VM

- The VM deployment pipeline calls a different pipeline (that runs Ansible)

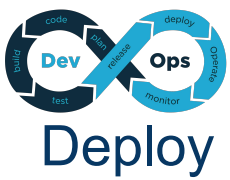
- Infra_configuration repository is clone to the build machine and triggers a python script that:
  - Setup a python virtual environment
    - requirements.txt contains required python versions
  - Installs Ansible, installs external roles and clones/pulls Ansible inventory

- Run core configuration
  - Install (default) packages
  - Create (default) users & groups
  - Configure ssh daemon and client
  - Etc.

Collecting Ansible Dependencies

- Make the collection process of external dependencies flexible!
- Ansible-galaxy command allows you to pull roles directly from GIT repositories
- Automate the installation of ansible dependencies
  - Always specify a version when deploying external dependencies
  - Tokens are collected from Hashicorp Vault

```
repositories:
  - name: epel
    src: "https://x-token-auth:{{ epel_token }}@bitbucket.org/epel.git"
    scm: git
    version: "{{ epel_version }}"
```



Pipeline (fase 3): deploy Service

- If **Infrastructure** service:
  - Run Ansible playbook (e.g. deploy and/or configure DNS)

- If **Business** service:
  - Run Ansible playbook (e.g. install tomcat)
  - Call service/artifact pipeline (maintained by developers)
    - Deploy production artifact(s)
    - Apply configuration on said artifacts

Handling secrets

- All secrets are stored in Hashicorp Vault
  - Easy to rotate secrets, these can be (automatically) updated in Hashicorp Vault using a CLI, API or GUI
- Access to secrets based on principle of least privilege!
- Cluster of 3 Hashicorp Vault servers, setup with Ansible and Terraform
- Backup is created every 12 hours and stored in a Azure blob store
  - Only readable with 3 of the 5 “unseal keys”
  - Backups are removed after 4 weeks

Handling secrets in code

- Ansible/Terraform secrets are only allowed to be used in templates if they are stored in a hashed/encrypted form (*note the password_hash*)

```
default_user:
  - name: automator
    group: automator
    state: present
    shell: /bin/bash
    system: false
    password: "{{ lookup('community.hashi_vault.vault_kv2_get', inventory_hostname )['secret']['automator'] | password_hash('sha512') }}"
```

- Map secret names to inventory_hostname allows for a generic collection string throughout the entire Ansible code base

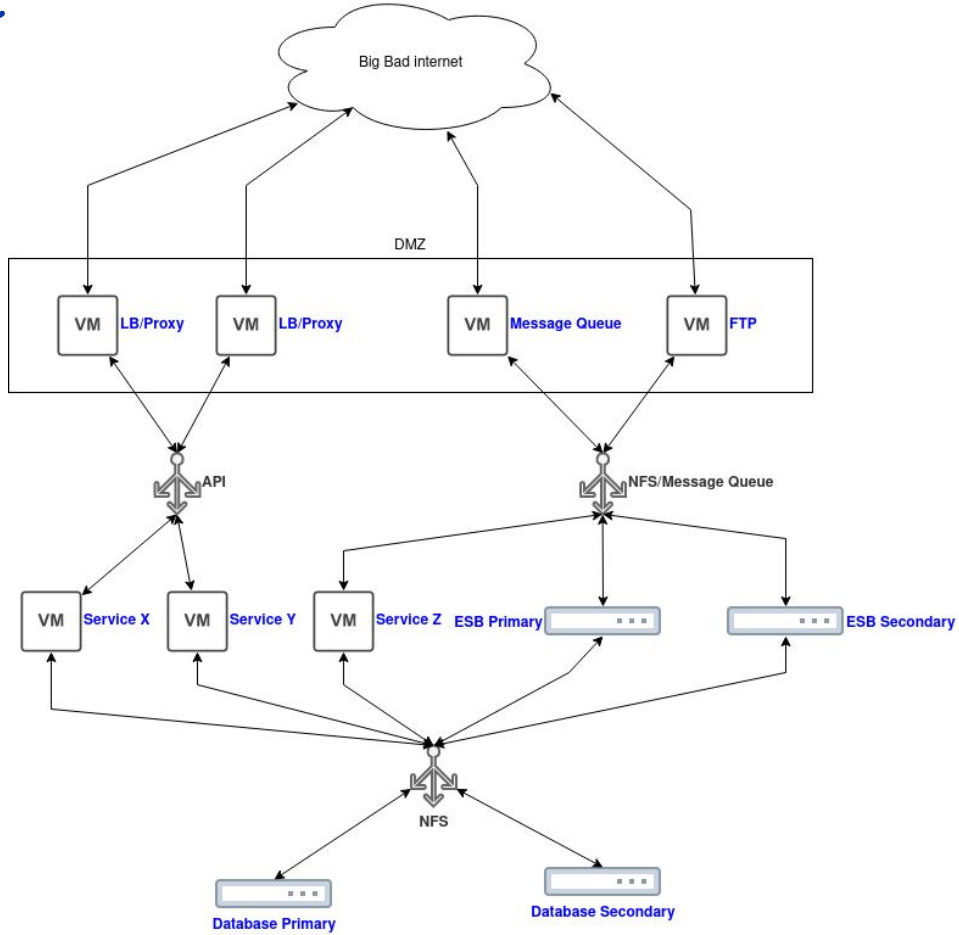
Handling secrets in pipelines

- Pipelines have a token generating token

- These tokens are only allowed to generate tokens with restricted **read** access to a limited set of secrets within Hashicorp Vault
  - Tokens where only valid for several minutes (duration of pipeline)

- Token generating token is stored is within a secure variable in bitbucket
  - All tokens must expire!
  - Exception is the token generating token
    - If the pipeline is not run within 30 days the token generating token expires!
  - Develop and Operation use different tokens
  - DTAP uses different tokens per environment

Architecture



Goals

- Inventory (and cleanup) of the environment
  - Registered all systems (in netbox), cleanup unused VMs and bare metal systems

- Creating a high level design/architecture of the environment

- Upgrade (virtualization) hardware
  - Virtualization hardware has been upgraded
  - (In progress: Databases & Network)

- Simplify, automate, shorten and document deployments
  - Deployment automated using Terraform and Bitbucket pipelines

Goals cont'd

- Upgrade (OS & Software) and automate configuration of VMs
  - Configuration is automated using Ansible and Bitbucket pipelines
  - Call developer pipelines for deploying/configuring business artifacts

- Harden systems
  - Adopted Center of Internet Security (CIS) hardening guidelines in deployment and configuration setup

- Adopt Cloud services (where possible)
  - e.g. Blob storage for backups and state
  - e.g. Monitoring and logging to Splunk

Lessons learned & key takeaways

- Keep informing stakeholders!
  - After 1 sprint we already started using our setup to deploy VMs that were not scoped to be deployed with the new setup (it was quicker than doing it by hand).
  - After our 3 sprint stakeholders wondered if we were ever going to use the tools we were building.....

- Use spikes for gaps in knowledge
  - Allows you to first investigate if a solution/approach fits your needs
  - Allows you to identify potential risks and integration issues
  - Helps with refining an estimating future stories

Lessons learned & key takeaways

- **Do not** store everything in **one pipeline**/terraform file or ansible playbook
  - Design your solution/structure, setup a proper VCS structure
  - Keep things small, e.g. playbook per service or terraform modules
  - Use (external) roles and collections!
    - Verify they are safe
    - Ensure you can always collect the version you require

- **Avoid** taking (to much) **shortcuts**, limit/avoid **technical debt**
  - Higher quality of code
  - Identify/limit (security) risks
  - Prevent your progress/velocity from coming to a grinding halt at a later stage!

Questions or Feedback

https://forms.gle/8yksRxEm5kzHvgUTA