Create an automated cloud factory with IaC

Infrastructure as Code

Julian Pelizäus
Bosch Architect

Moritz Schönwetter
Red Hat Consultant

Eric Lavarde
Red Hat Architect

AnsibleFest

Agenda

- Introduction
- Project & architecture
- Standardization
- Development approach & Git workflow
- CI/CD Landscape
- Learnings

Introduction

Who are we?

Who are we?

Julian Pelizäus
Robert Bosch GmbH

IT Architect for CI/CD tooling and automation

Moritz Schönwetter
Red Hat Consultant

Infrastructure automation and IaC

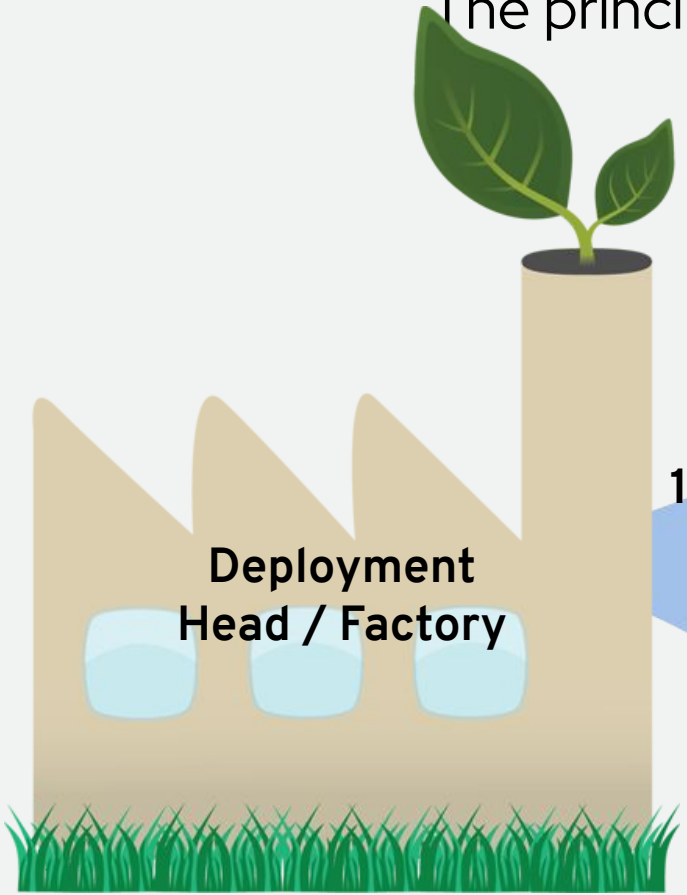
Eric Lavarde
Red Hat Consulting

Principal Architect, specialized in cloud automation and management

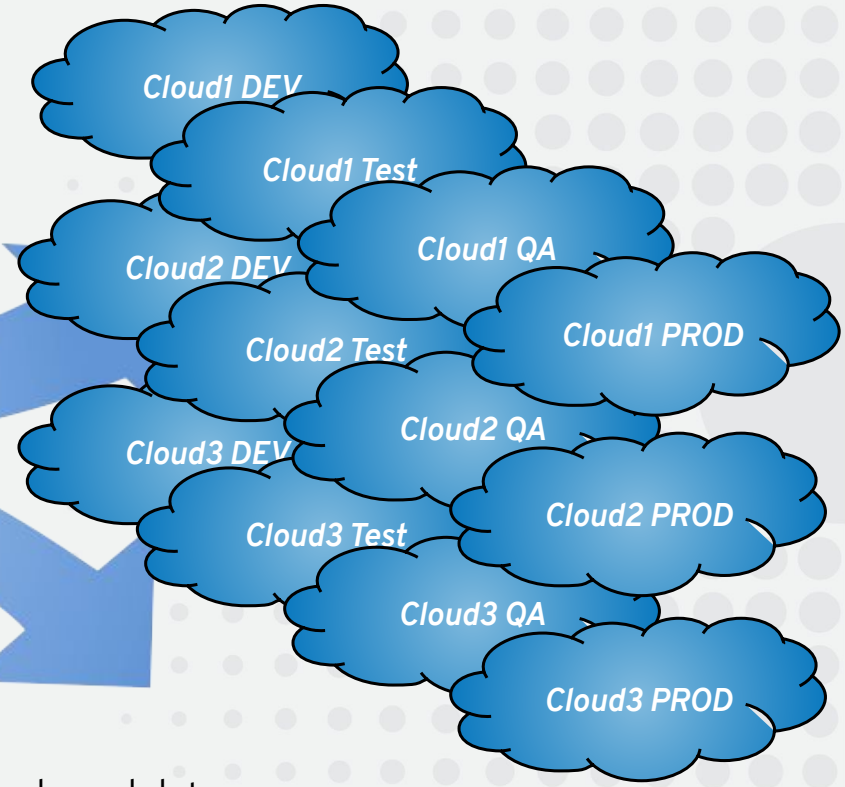
Project & Architecture

An overview

The principles of Software Defined Data Center



1 Automation
+
N flexible inventories



Separation of code and data

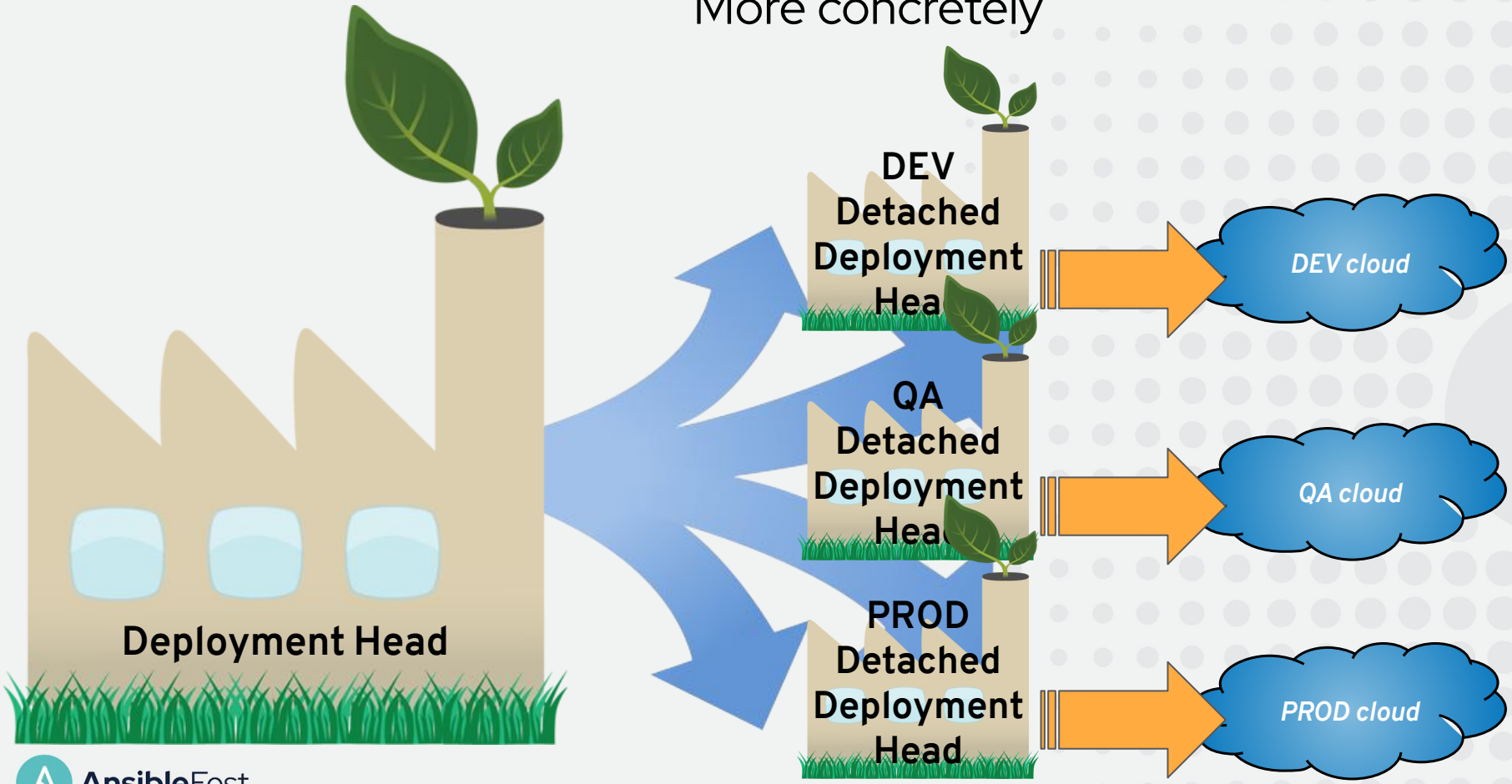
AnsibleFest

Why a factory approach?

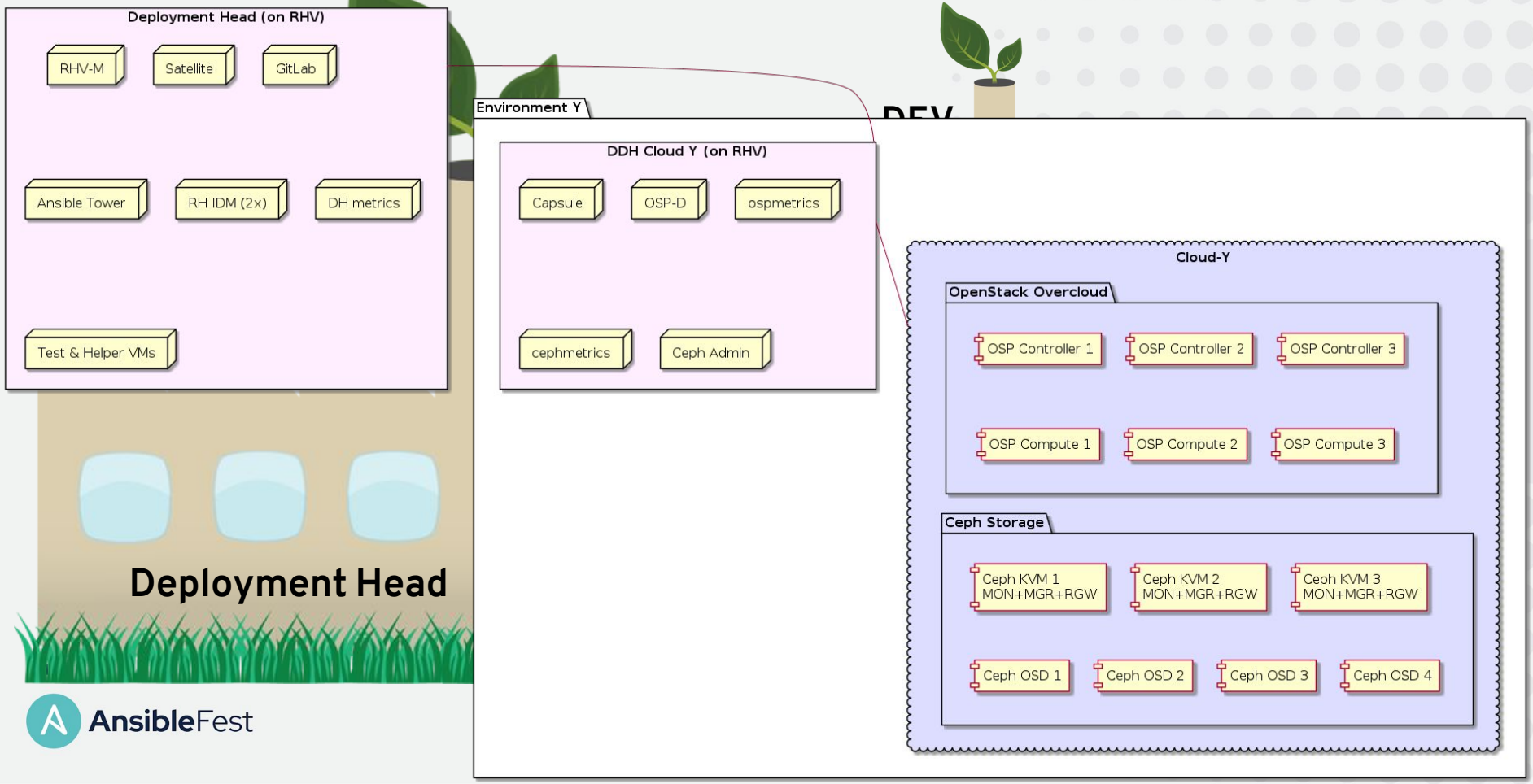
Main reasons

- **Efficiency** - do changes once, apply everywhere
- **Security / Stability** - done once, *tested* once, applied everywhere
- **Flexibility** - multiple smaller clouds fit for purpose vs. one size fits all

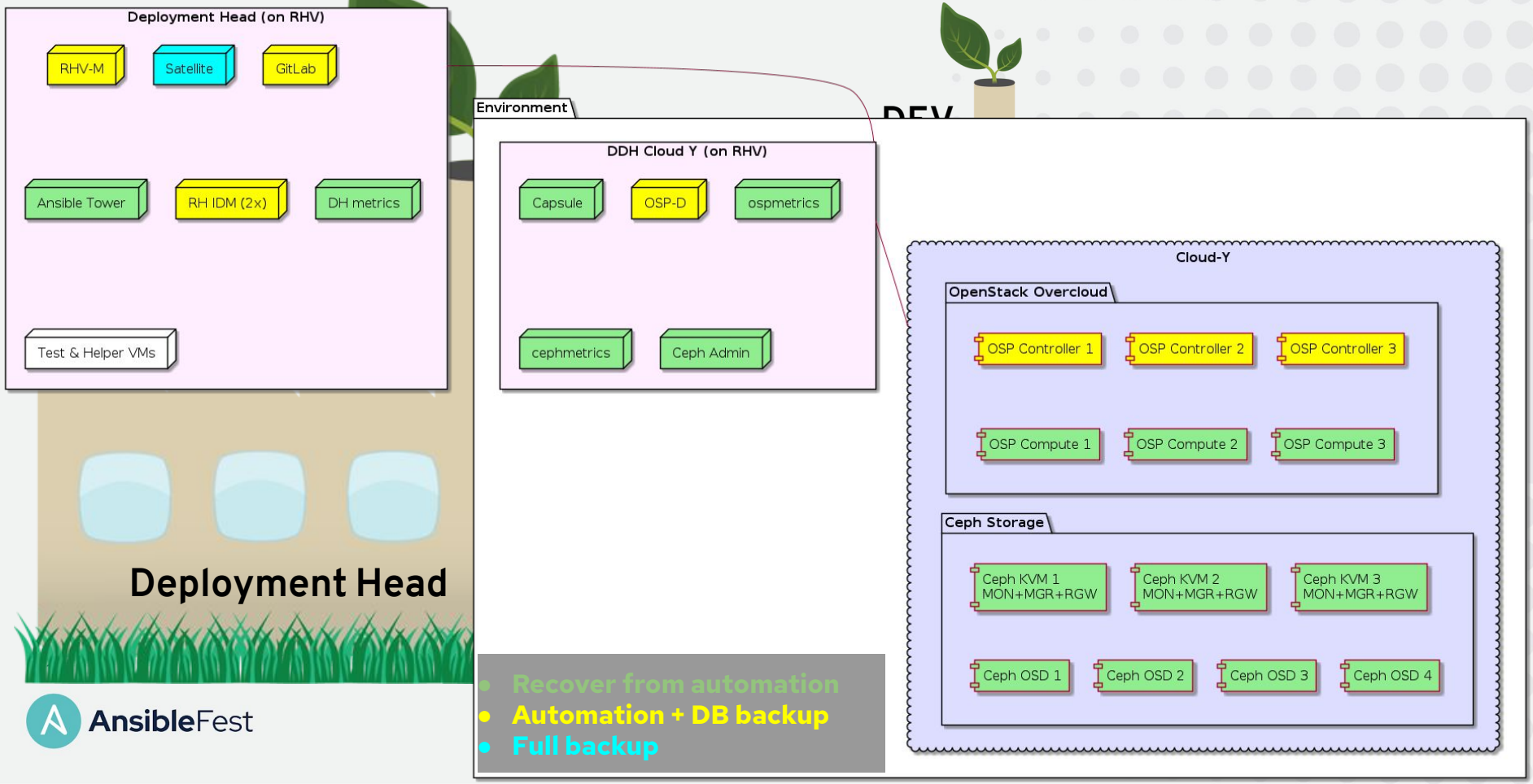
More concretely



The technologies behind (and automated)



Level of automation – based on recovery strategy

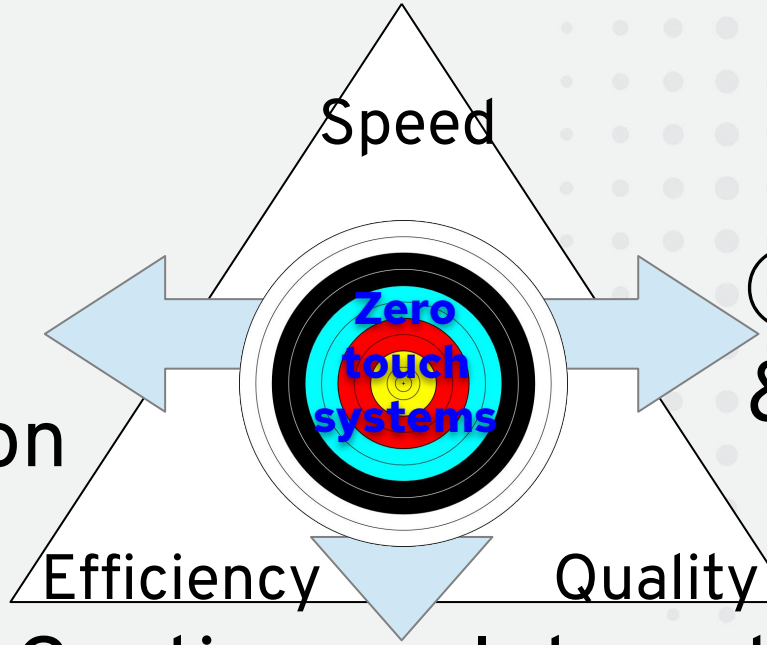


AnsibleFest

Standardization

Successful automation relies on 3 pillars

And modernization (as enabler)



① Standardization

② Automation & IaC, culture

③ Continuous Integration

Standardization

why and what

- **Reduce** the complexity of the environment
- Define **commonly** agreed
  - life cycle management process
  - security and other standards
  - versioned set of certified packages and patches
  - one single structured set of code
  - parametrized through one inventory & credentials per environment (data)
- *Required* **prerequisite** for automation and testing

Anonymous quote:
"who automates chaos, automatically gets chaos"

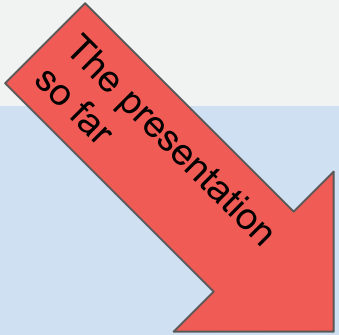
Automation structure

How to standardize code and data

Landscape	Type	Function	Component
One cloud environment = 1 Tower Workflow or Playbook of Playbooks	OSP-Director server = 1 Playbook resp. Job Template	managed VM + managed OS + OSP-D function = 1 Ansible role	User management = 1 task-file in a function role or 1 dependent role
	Each server has 1 and only 1 type	*A matter of re-usability*	*A matter of readability*

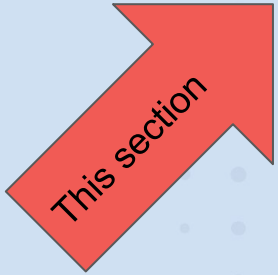
Turning Admins into Programmers

Infrastructure as Code



Infrastructure as Code

Infrastructure as **Code**



Prerequisites

In your team, you need to...

- 🔨 Agree on tools
- 🤓 Learn/teach basic Git
- 😾 Agree on cattle vs. pets
- 🐢 **Accept that it is a slow process**

Agree on Rules

- Coding style, naming schemes, editor settings
- Merge request format
- Acceptance criteria
- External code and licensing

Enforce the Rules

...and adapt them if needed

- Automated compliance checks*
- Code reviews:

  6-eyes principle, notifications to chat
- (Re-)evaluate and **react**

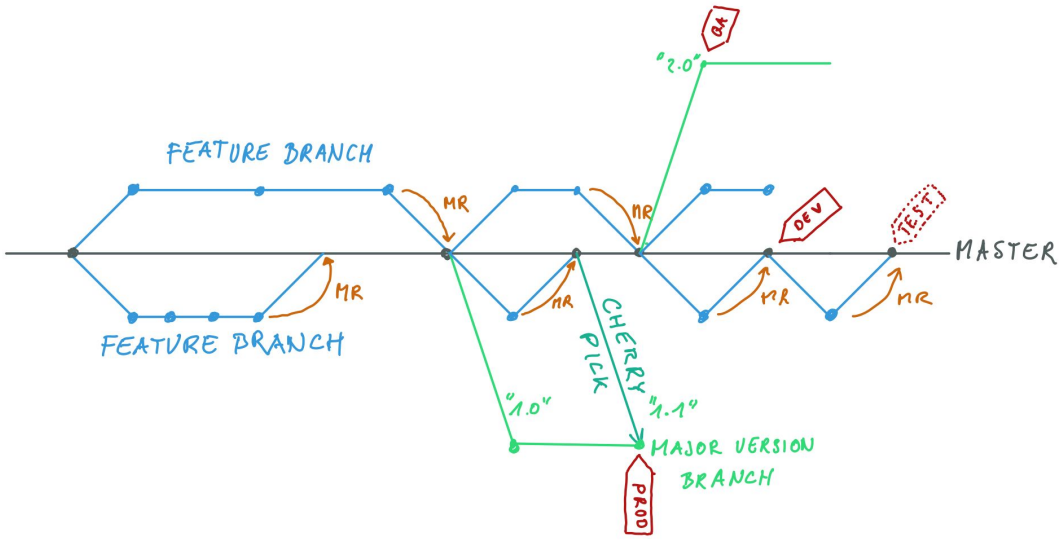
* see CI/CD part of this talk

Git-based Collaboration

Branching, reviewing, tagging, and more

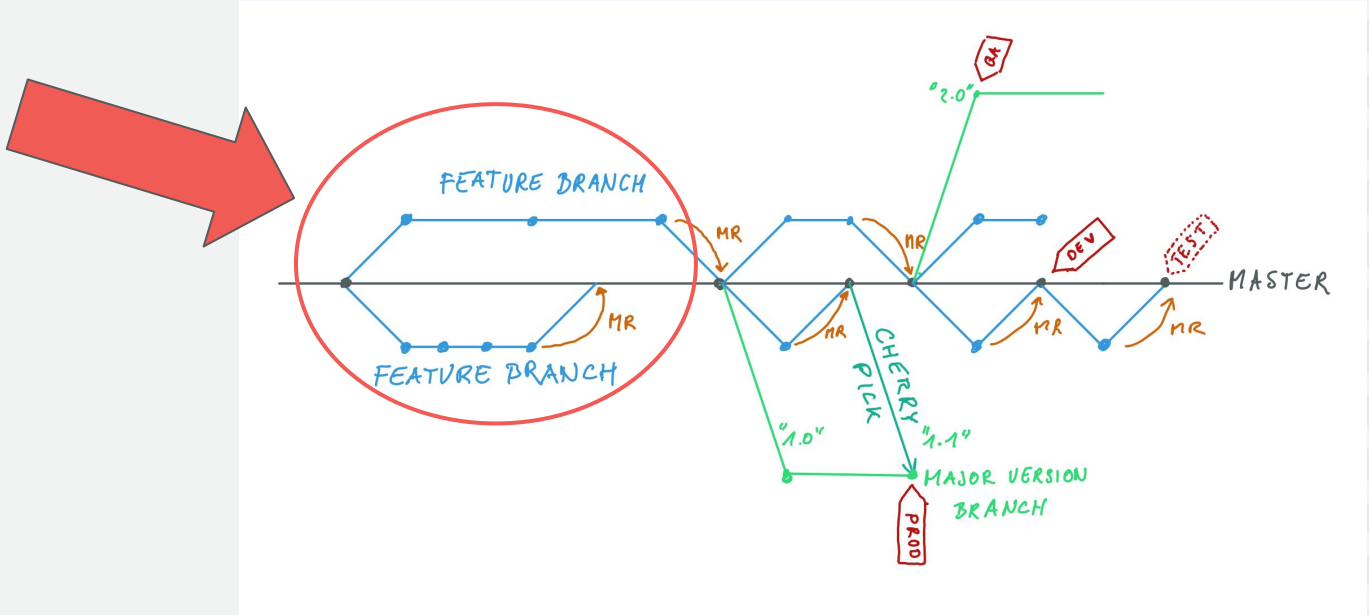
Git workflow

Overview



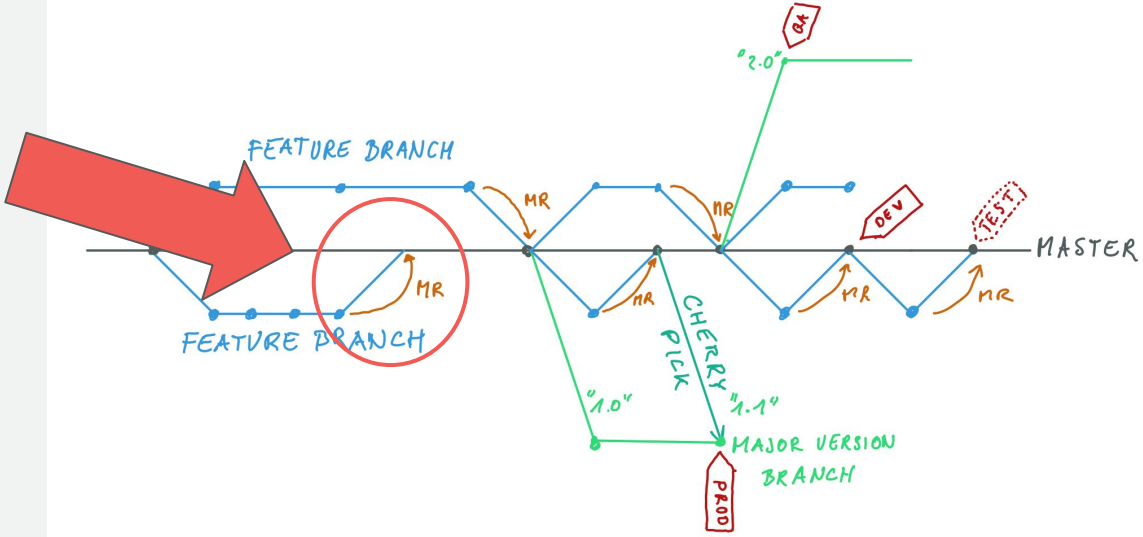
Git workflow

Feature branches



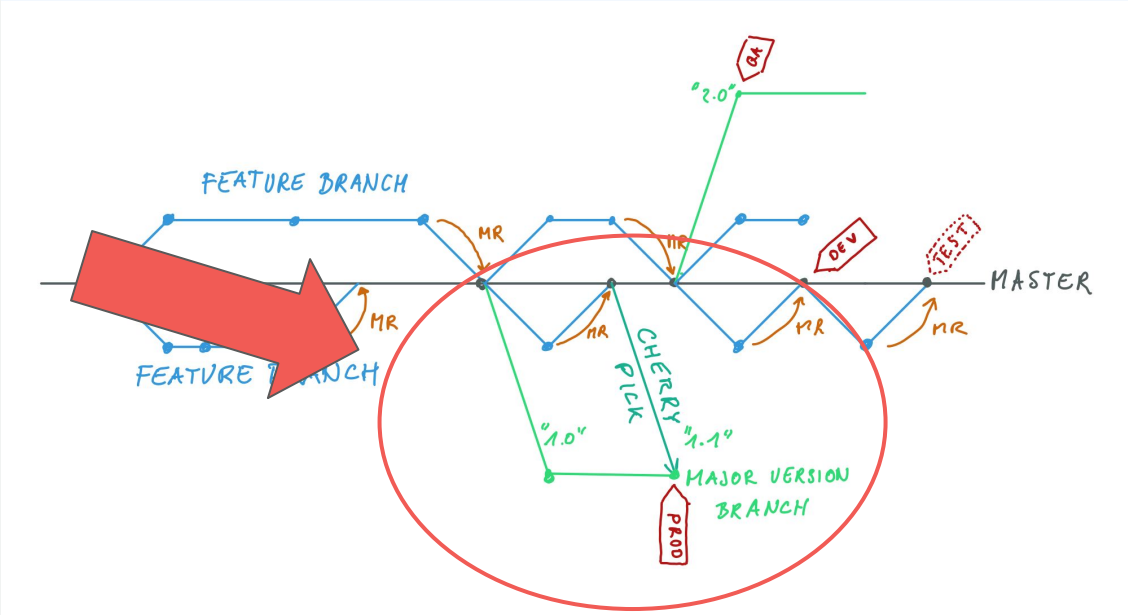
Git workflow

Merge requests and reviews



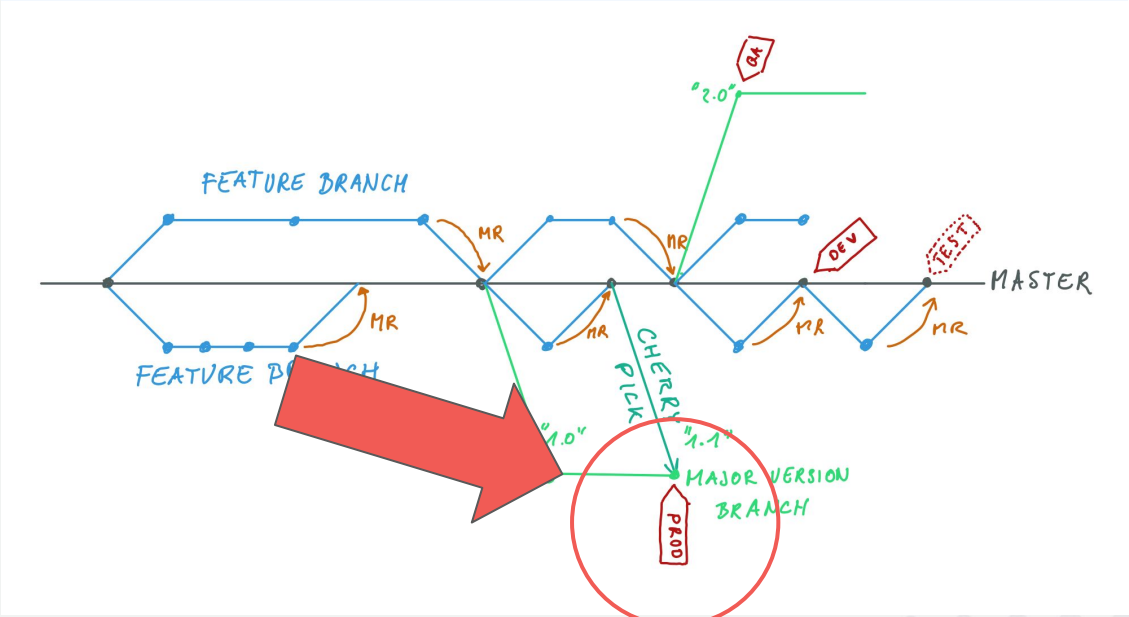
Git workflow

Release branches



Git workflow

Tags



CI/CD Landscape

Our approach to Continuous Integration, Delivery and Deployment

GitLab Service

Central platform for development and operation purposes

Single source of truth for the environment

- The entire codebase is placed in Git repositories
- Changes are transparent and auditable

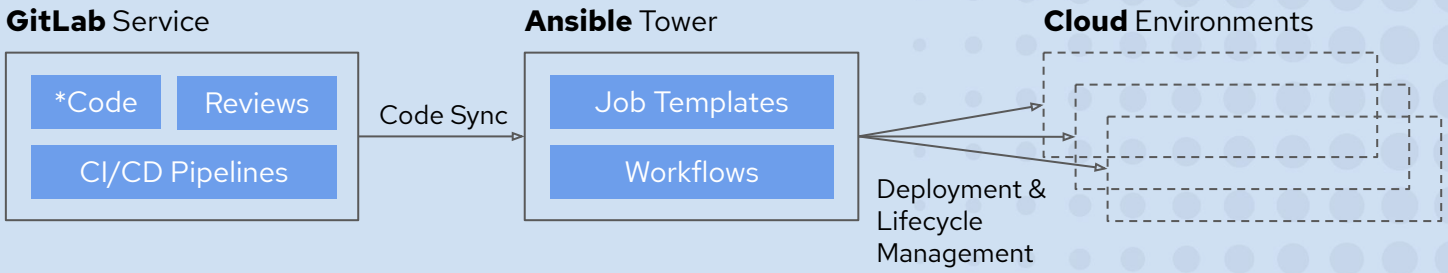
GitOps ready

- One place for development and operation duties
- Pipelines pick up jobs and make them visible to the entire team

Collaboration platform

Project Toolchain

From generic code to fully functioning cloud environments



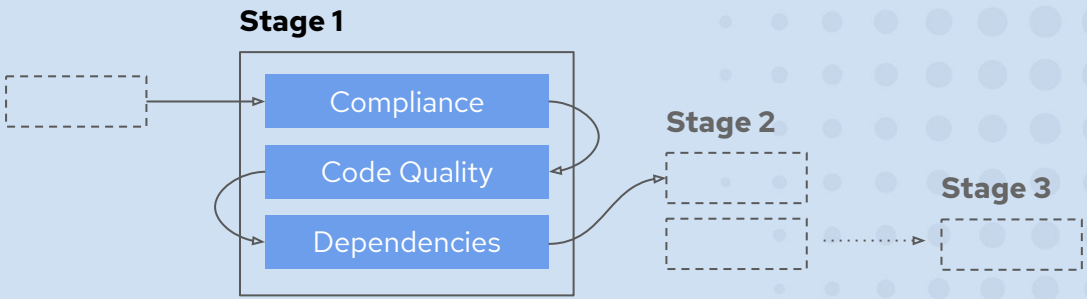
* Ansible code and variables in Git

Continuous Integration (1)

Code Quality & Compliance tests

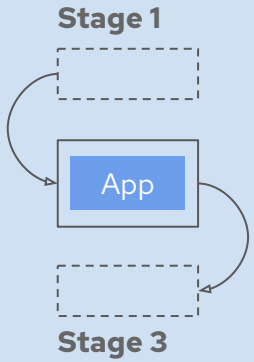
Automated checks on every code change

- Handled via pipelines
- Instant feedback for the developer



Continuous Integration (2)

Standalone application tests



Deploy various **applications** as standalone

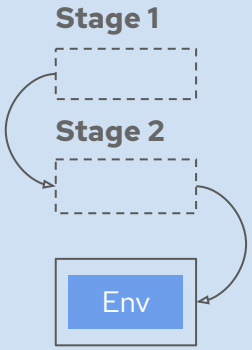
- Keep short time frames
- Ensure a successful deployment of the entire landscape

Reduce complexity for the developers

- Leverage the toolchain and offer deployment pipelines
- Reserve spare capacity for manual test deployments and debugging

Continuous Integration (3)

Extensive system tests



Deploy the entire **landscape**

- Tear down and spawn up everything
- Use predefined time slots and trigger regularly

Ansible as **test case** language

- The playbooks itself already cover quite a lot
- Add dedicated testing playbooks for more granular checks

Continuous Delivery & Deployment

Release often with ease

Ensure release availability at any point in time

- Automate the Git workflow*
- Frequently incorporate new features

Reuse the stack to deploy into production

- Preceding landscape deployments guarantee for error free rollouts
- Traceable by the team

* see *Git-based Collaboration* part of this talk

Learnings

What would we do differently next time?

What are our main learnings?

- Treat the infrastructure like code, but hardware remains hardware
- Build a test and reference environment *ASAP*
- Put a structure and common rules but stay flexible, don’t overload the team

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

youtube.com/user/RedHatVideos

linkedin.com/company/Red-Hat

facebook.com/ansibleautomation

twitter.com/ansible