metasite

Exclude Human – Continuous Deployment and OpenShift

by Valdas Mažrimas

A few words about me

My name is Valdas Mazrimas, I am full stack javascript engineer @ Metasite Business Solutions.

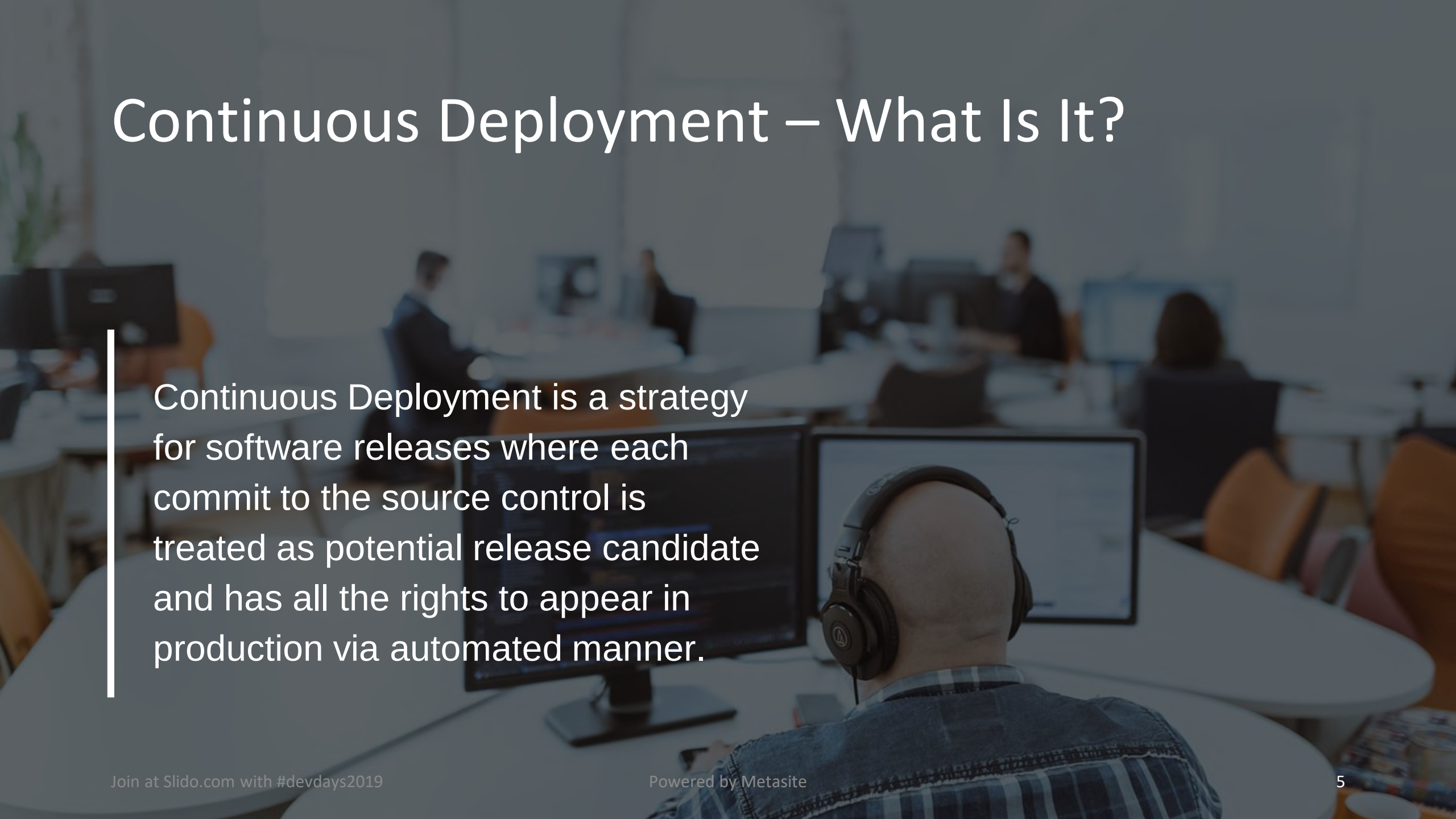
metasite

What we'll talk about today

- Continuous Deployment – What Is It Really?
- Why Continuous Deployment
- Instrumentation as a Key Factor for Continuous Deployment
- Git Strategy that Fits Continuous Deployment
- How We Organise Stateful Set Deployments
- How We Organise Secrets
- Pipelines and Stages

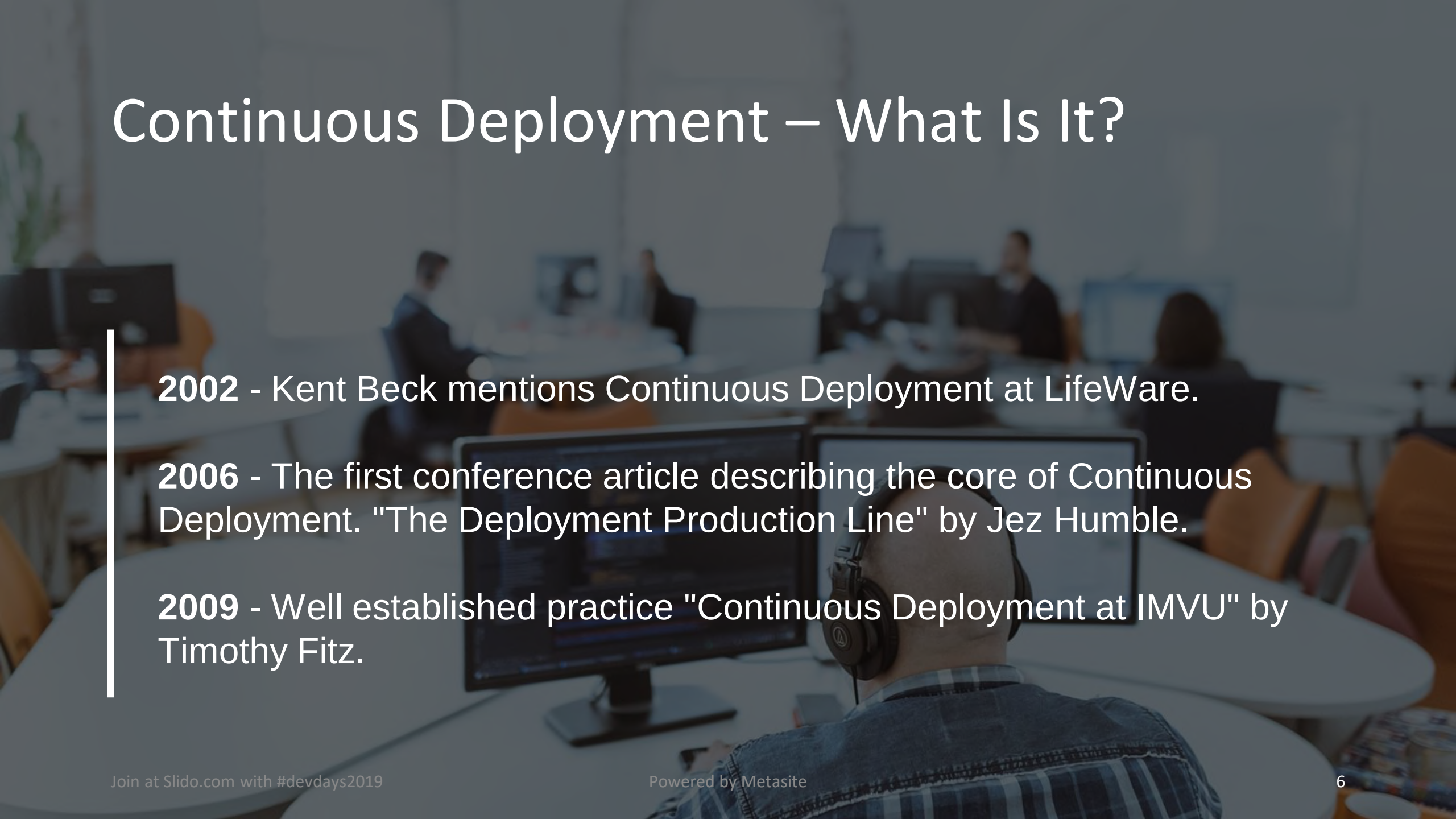
Continuous Deployment: What Is It?

Continuous Deployment – What Is It?



Continuous Deployment is a strategy for software releases where each commit to the source control is treated as potential release candidate and has all the rights to appear in production via automated manner.

Continuous Deployment – What Is It?

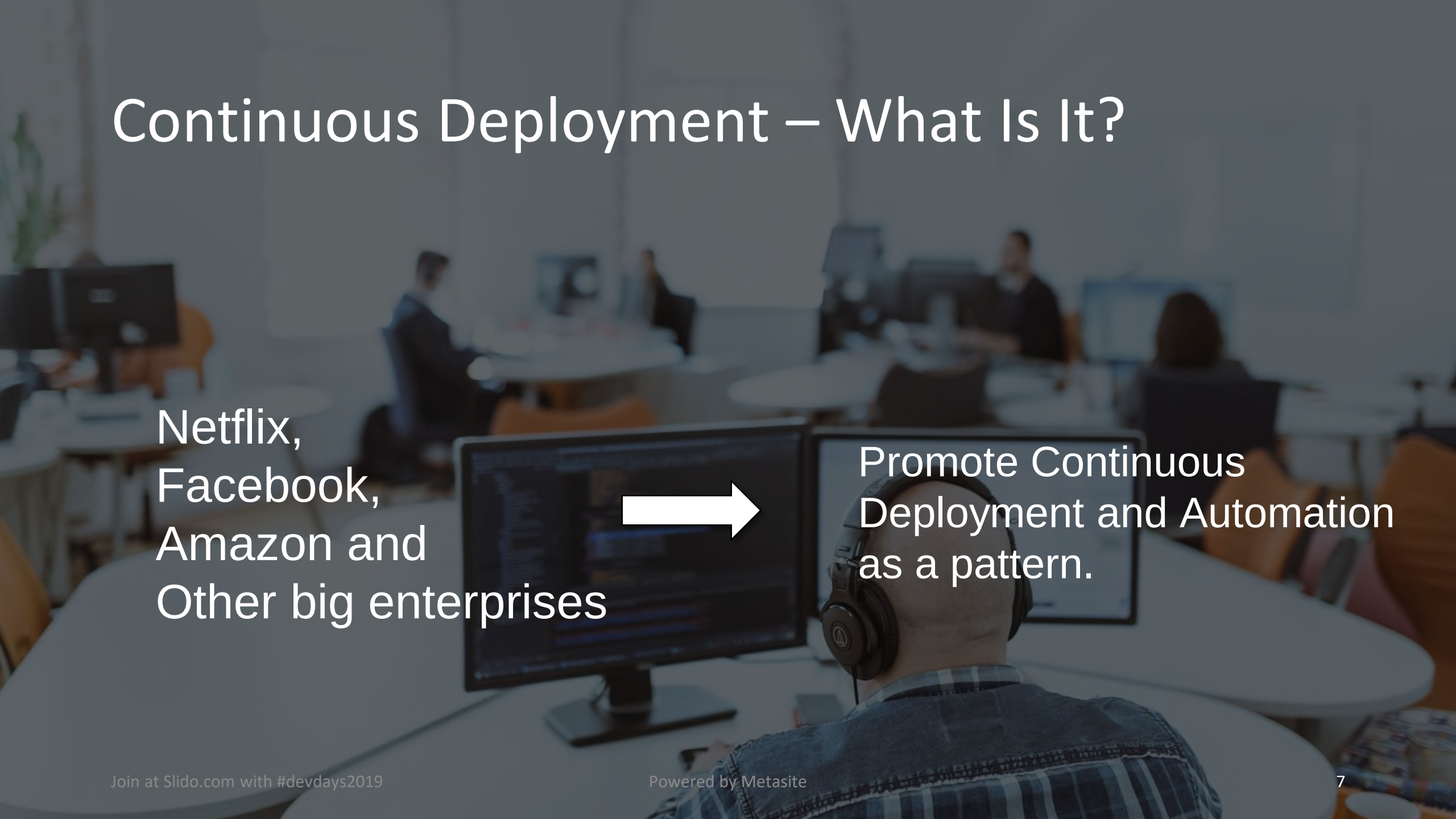


2002 - Kent Beck mentions Continuous Deployment at LifeWare.

2006 - The first conference article describing the core of Continuous Deployment. "The Deployment Production Line" by Jez Humble.

2009 - Well established practice "Continuous Deployment at IMVU" by Timothy Fitz.

Continuous Deployment – What Is It?

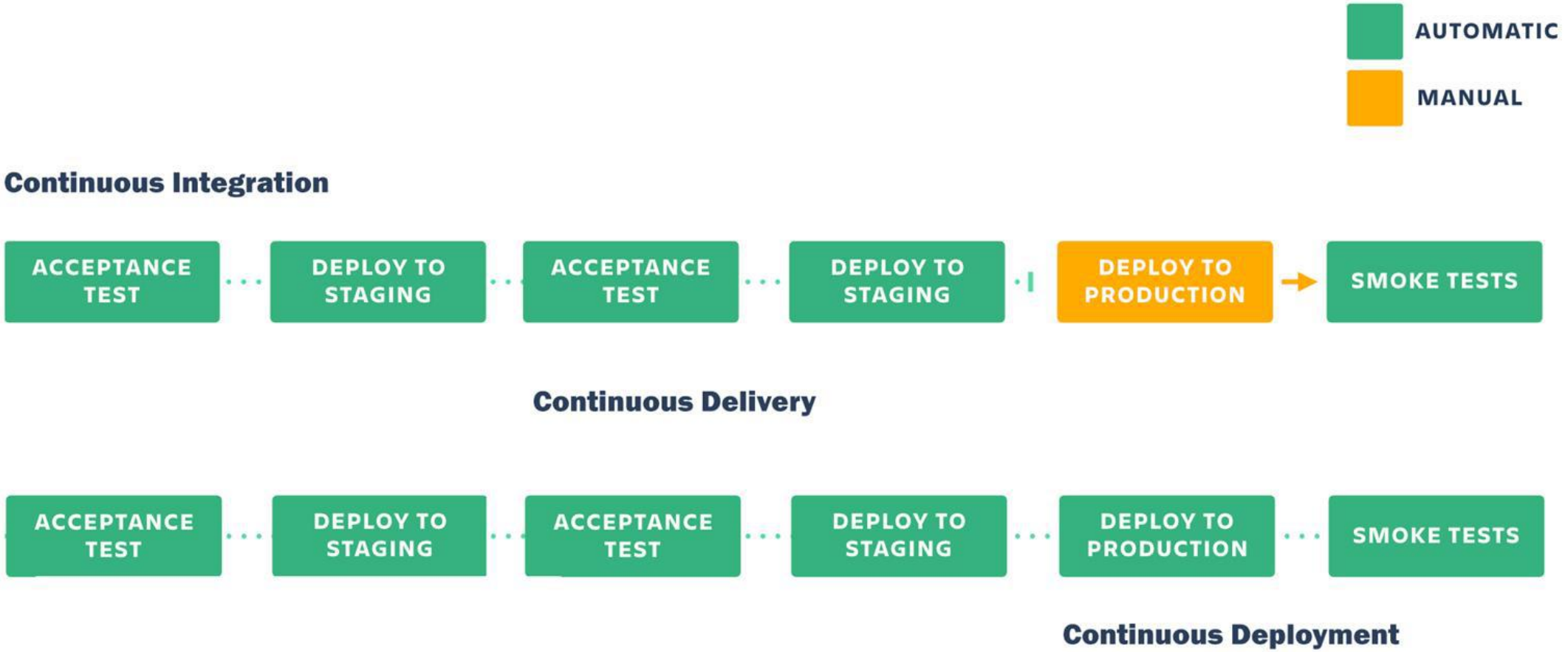


Netflix,
Facebook,
Amazon and
Other big enterprises

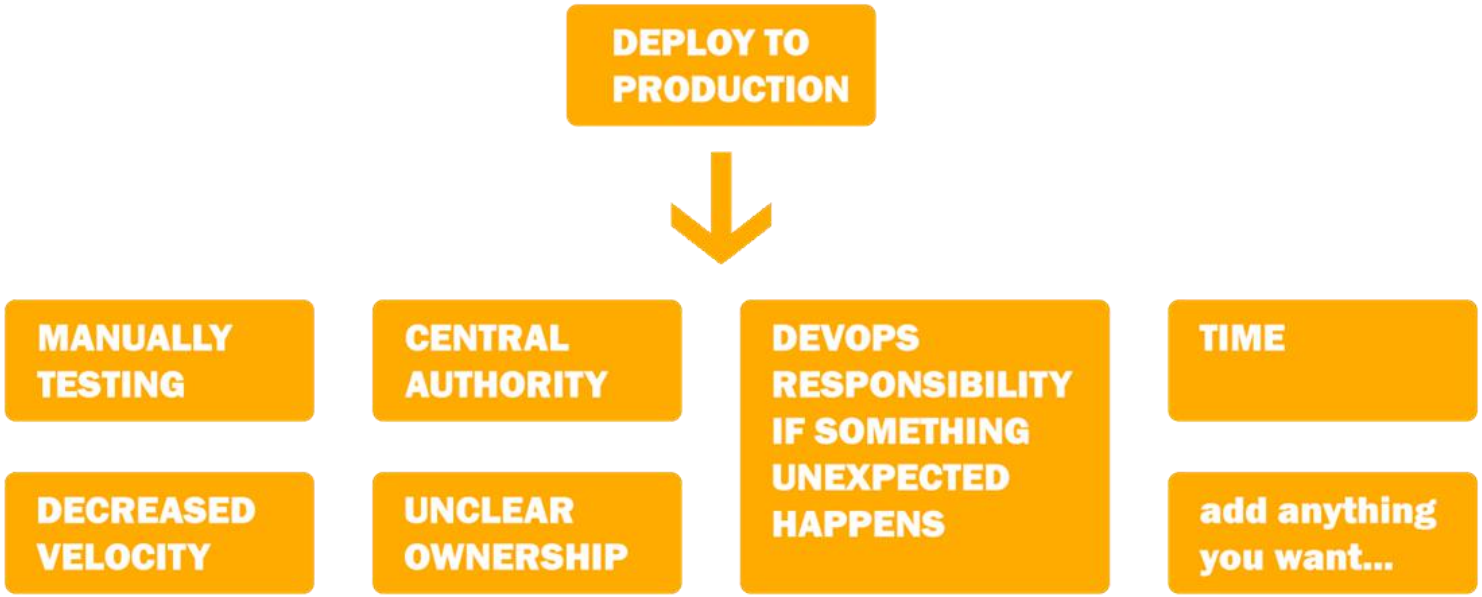
→

Promote Continuous Deployment and Automation as a pattern.

Theoretical Model of CI/CD



The ‘Not Aiming to Continuous Deployment’ Problem



Why Continuous Deployment

Reasons to do Continuous Deployment

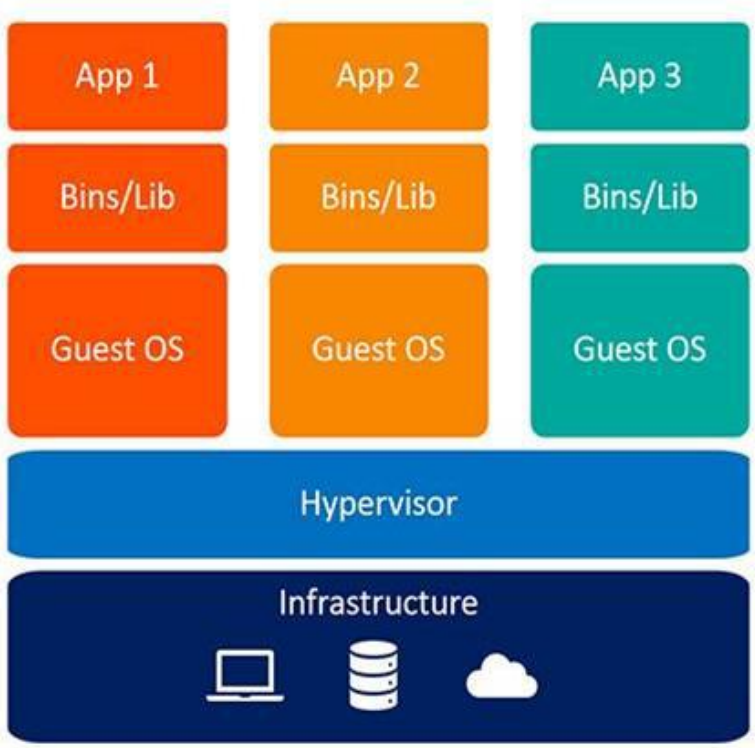
- Unclear ownership of a project codebases
- Humans are bad at doing repetitive tasks
- Teams have different CICD practices now way to unify them
- Every team and team member should be able to understand a release process without a Central Authority
- Bad culture habits are growing
- We are not as productive as we could be

Technical Challenges to implement CD

- Multiple languages and frameworks, hard to unify builds
- Lack of instrumentation, traditional hypervisor infrastructure is not dynamic and can not scale
- Non-functional tests not possible as infrastructure is not self healing
- Rollback from new to previous environment is time consuming
- Can not achieve 0 downtime deployments

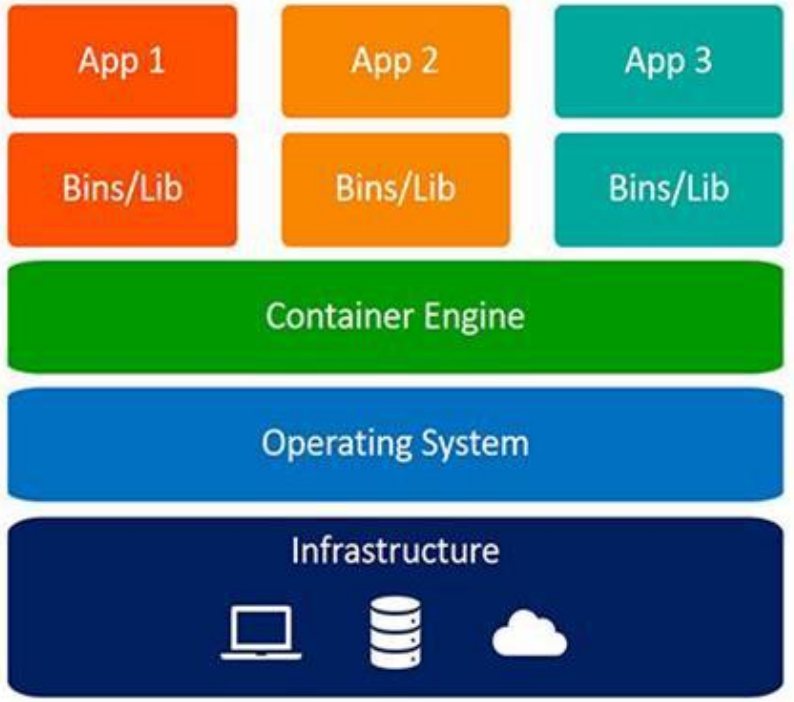
Instrumentation as a Key Factor for Continuous Deployment

Infrastructure change



Traditional Hypervisor

To



Containers

Instrumentation that enables CD

+

+

+

Chaos Tools

+

Why we choose OpenShift over other Kubernetes distributions

- OpenShift builds security around containers
- We like Routers concept in OpenShift
- ImageStreams allow deployment config enchantment
- We have multiple clients and multiple projects, OpenShift focuses more on segregation between projects

Why we build around Jenkins

- Everyone already knows Jenkins
- Jenkins is very nicely integrated in OpenShift
- Unlimited flexibility with plugins
- We can easily share complex pipelines for other projects via shared libraries

Jenkins – Caution (!)

- We tend to overuse Jenkins, build, deploy, orchestrate, now we just orchestrate
- We did not try to make Pipelines fast, now use parallel stages if possible and prepared agents for tasks
- We tend to put all kinds of secrets, passwords, certificates into Jenkins, now using Vault
- We do not allow webhooks from internet, now we put Webhook Payload Proxy in between

Git Strategy that Fits Continuous Deployment

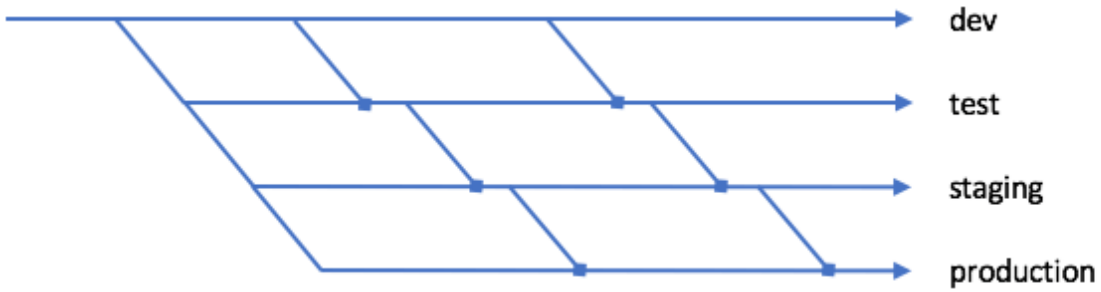

We borrowed something from GitOps
EVERYTHING AS CODE

Everything as code

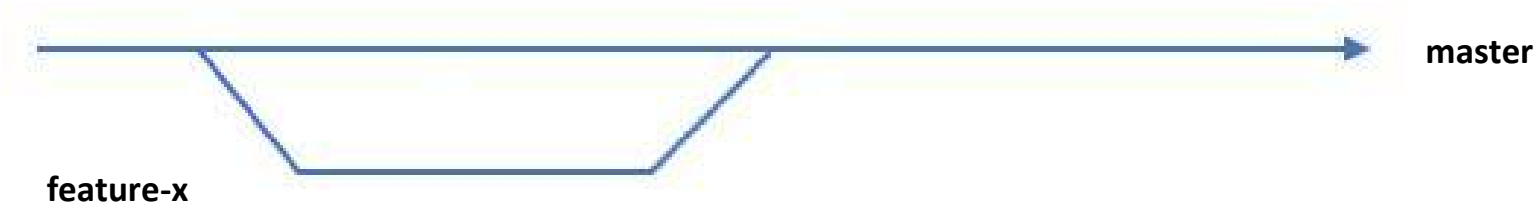
- Infrastructure configuration - **In the Git**
- Application builds, deployments and other configs - **In the Git**
- CI/CD Pipelines – **In the Git**
- Secrets – **In Vault**
- All kinds of tests - **Git**
- Schema migrations – **Straight in Git**
- Everything else - **That’s right, Git**

Git Strategy change

From **Environment branches**



To **xFlow**



xFlow rules

- **Mono Repo**
- One mainline. **Master**
- On PR **- my-app-preview-my-feature-x1234** created
- **Branch Matching** for dependent PR’s
- Git Tags **latest** and **x.y.z** for each release

How We Organise Stateful Set Deployments

Stateful containers - databases, message brokers

- We use OpenEBS for syncing the data sets between B/G Deployments
- OpenEBS Hight Availability Storage Driver enables one click rollout and rollback Application Deployments

When developing, we focus on

- Automatic up and down schema migrations
- Prepare seed data
- One microservice one database schema
- Unit testing data entities

How We Organise Secrets

Secrets #$U*(@&@#!

We all tried using Environment Variables, Secret Config as mounted files in containers...

We all felt bad about it...

Selection - Ansible or Hashicorp

- You do trust humans who configure encryption
- You do not need secrets management

If both True choose Ansible Vault, otherwise Hashicorp Vault.

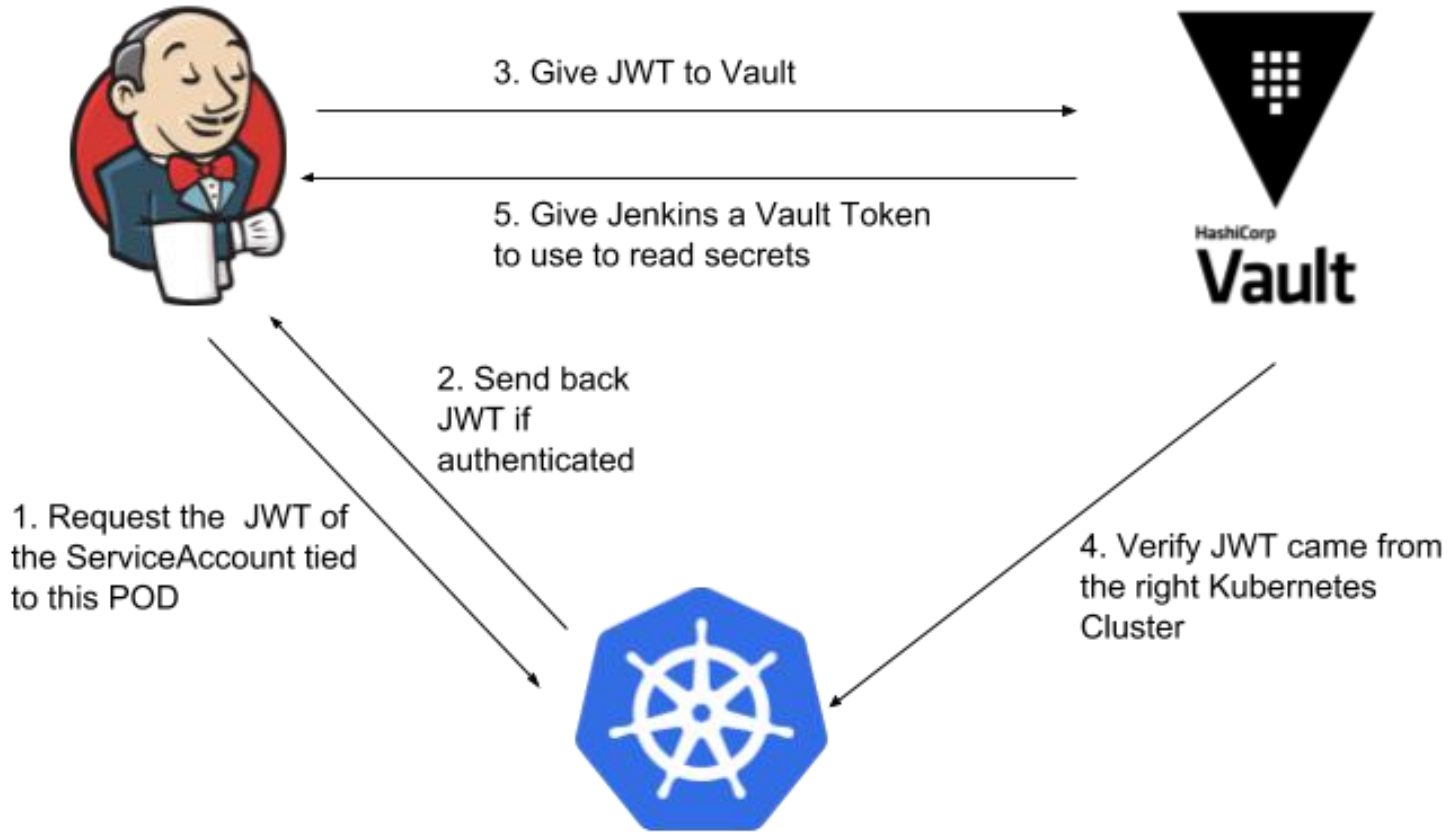
Hashicorp Vault features that we like

- Shamir Shards algorithm for Master Key encryption
- OpenGPG Sharded Keys for Master Key Shards encryption
- Built in sealing and unsealing functionality in The Vault

Hashicorp Vault usage scenarios

- Sidecar containers as Token Issuers to get secrets at REST and use Leases for token renewal
- Jenkins authenticates to Vault via AppRole mechanism and uses secrets in wrapped build stages

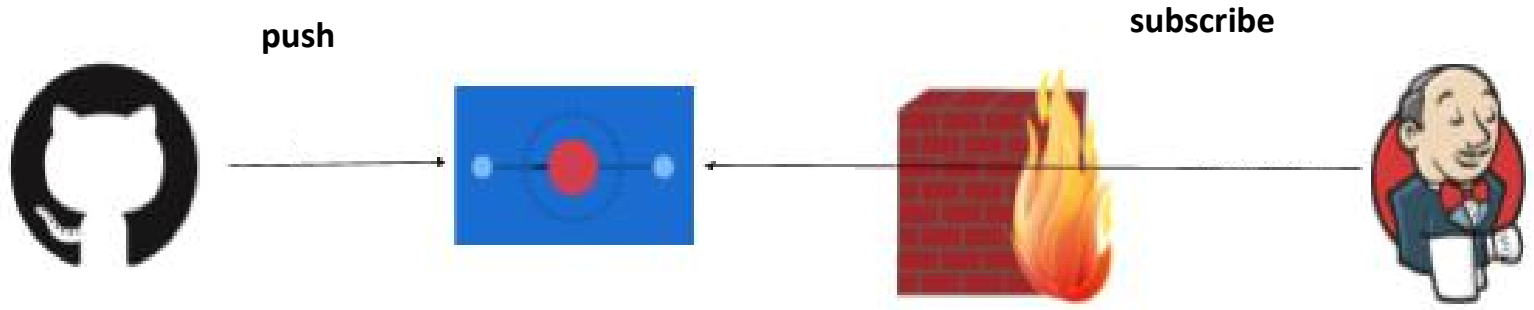
Jenkins integration with Vault



Jenkins Logo: https://jenkins.io

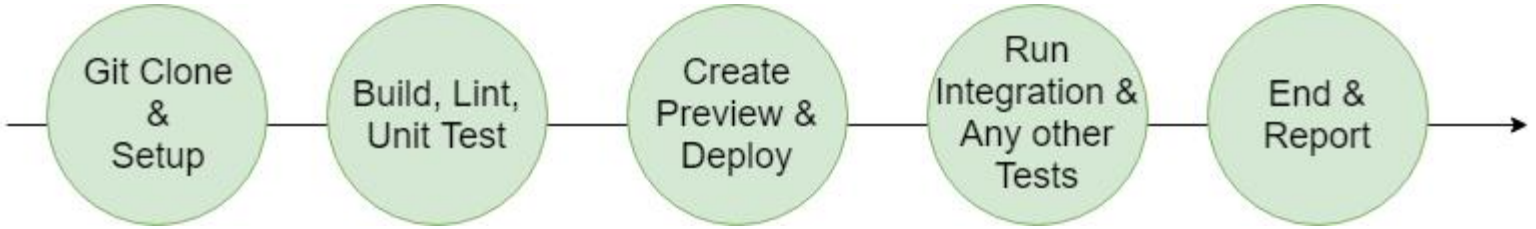
Jenkins perimeter security

GitHub pushes through secure webhook payload proxy service to deliver notifications to Jenkins



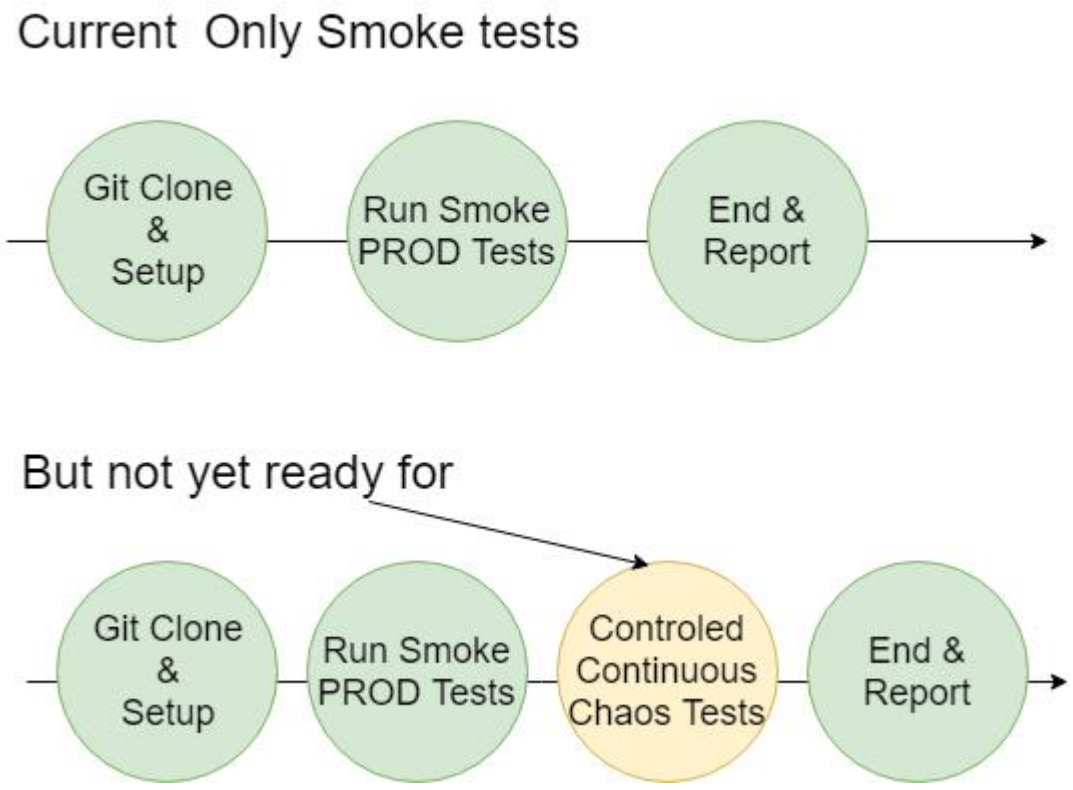
Pipelines and Stages

Pull Request pipeline



Main pipeline

Scheduled production pipeline

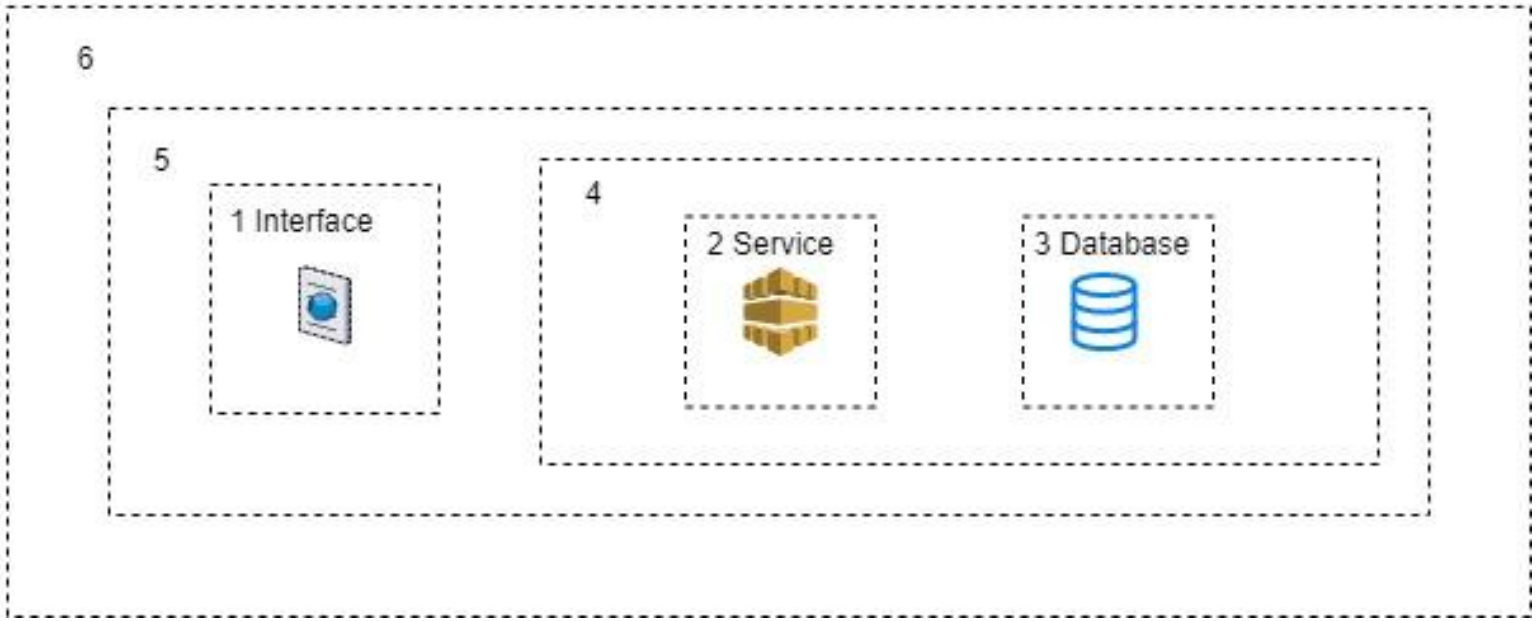


Deployment patterns

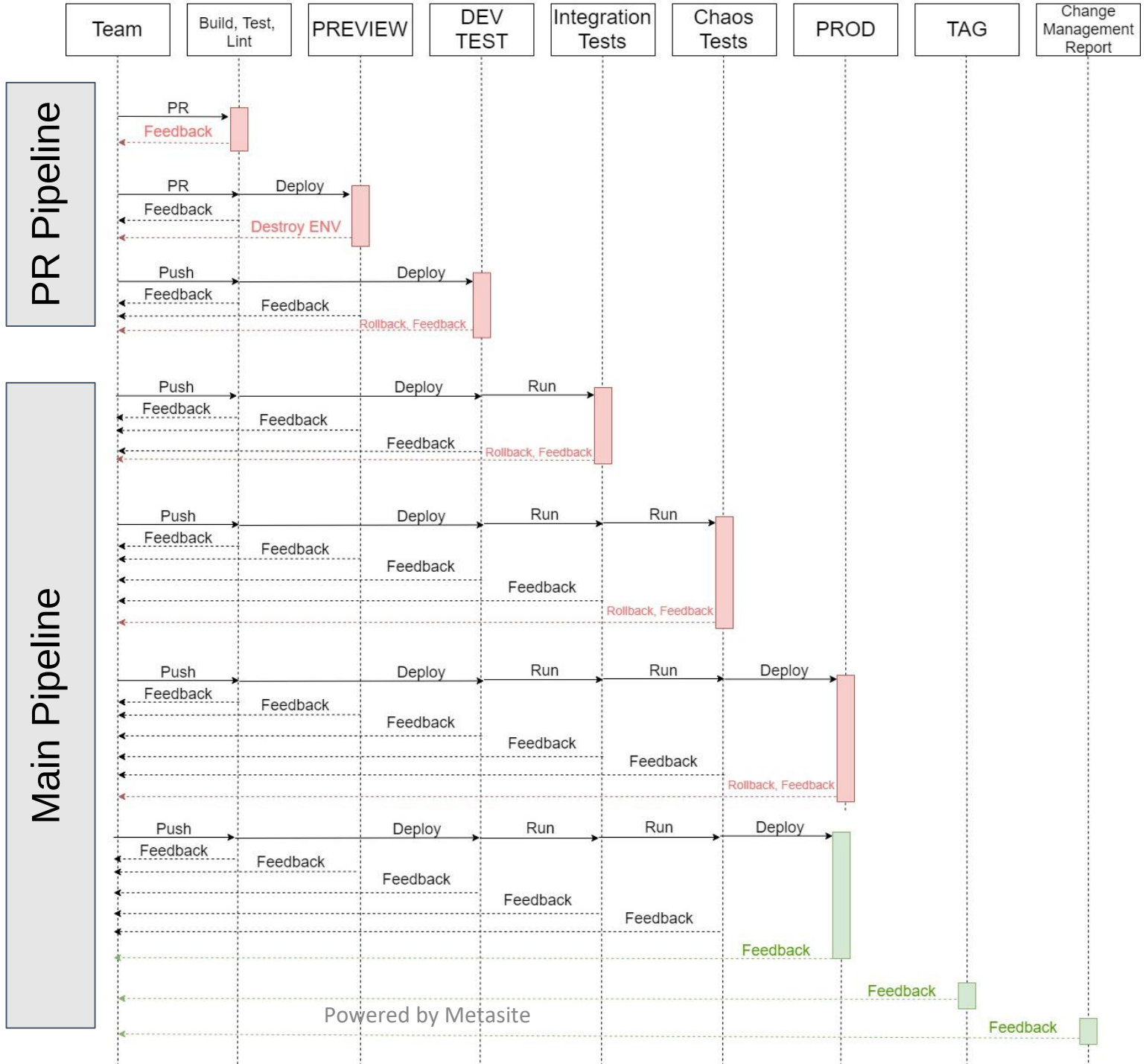
Isolated Deployments
1. User Interfaces
2. Service Only
3. Database Only

Composite Deployments
4. Service & Database
5. Interface & Service & database

Special Deployments
6. Full App & Everything Else



Feedback loops



Thanks, let’s stay in touch

linkedin.com/in/valdestron

github.com/valdestron

Join me at the Ask Me Anything Corner
near the registration zone.