Getting started with CI/CD for cloud infrastructure

Michael Lihs

/thoughtworks

Michael Lihs

Infrastructure Consultant
@Thoughtworks

father of 👶 and 👧

🏡 Tübingen 🇩🇪

🚴 cycling 🪚 woodworking

linkedin.com/in/michael-lihs/

How can we deliver infrastructure faster and safer…

Tools like Terraform are well established

Infrastructure changes take ages until applied to production

… by making changes in small batches

Pipelines for infrastructure automation

Often more of a “prey-and-hope” approach

… by making the process dead simple & easily reproducible

… by providing a proper audit trail

Three principles of Continuous Delivery for infrastructure

There's more to it than writing infrastructure code!

1. Everything as code and in version control

- Infrastructure code
- Configuration
- Tests
- Compliance
- Pipeline
- Automation scripts

2. Continuously test and deliver all work in progress

- Build quality in
- Test as you work
- "Integrate" at least daily
- Reproducibility
- Provide an audit trail

3. Small, simple pieces that you can change independently

- Reduce complexity
- Shorten feedback cycles
- Reduce blast radius
- Apply proper permission boundaries

How?

thoughtworks

Everything as Code

```
.
├── main.tf
├── pipeline.yaml
├── config.yaml
├── tests/
├── policies/
├── README.md
├── backend.tf
├── terraform-dev.tfvars
├── main.tf
└── ...
```

Traceability made easy
Every change is done via a commit to the repository.

Reproducibility
Tooling and infrastructure code are versioned together.

Branching strategy
Branches and IaC don't go well together.
If you use branches merge at least daily!

Three principles of Continuous Delivery for Infrastructure

1. Everything as code and in version control

- Infrastructure code
- Configuration
- Tests
- Compliance
- Pipeline
- Automation scripts

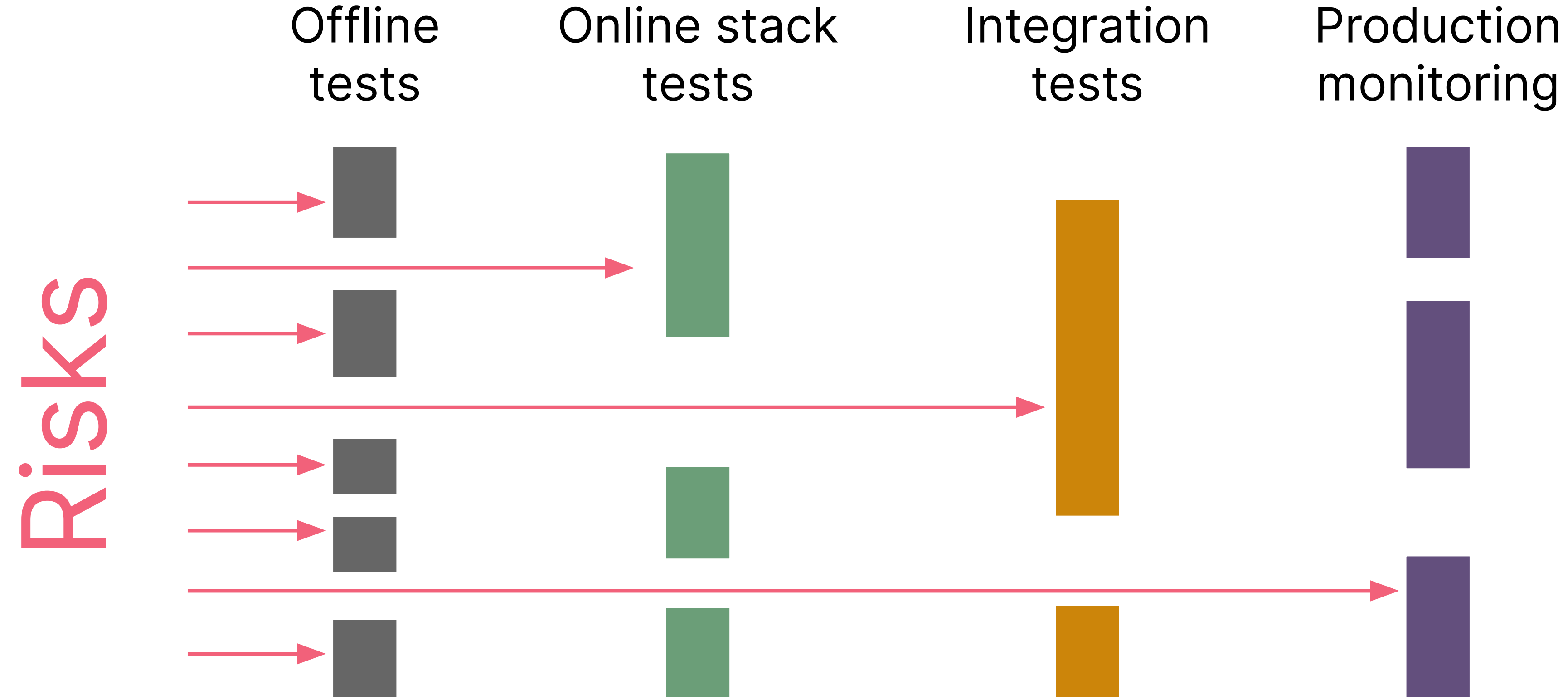
2. Continuously test and deliver all work in progress

- Build quality in
- Test as you work
- Integrate at least daily
- Reproducibility
- Proper audit trail

3. Small, simple pieces that you can change independently

- Reduce complexity
- Shorten feedback cycles
- Reduce blast radius
- Apply proper permission boundaries

Swiss cheese testing model



Offline testing

```
.
├── backend.tf
├── config.dev.tfbackend
├── config.test.tfbackend
├── config.prod.tfbackend
├── main.tf
├── provider.tf
├── terraform.tfvars
├── variables.tf
├── terraform-dev.tfvars
├── terraform-test.tfvars
├── terraform-prod.tfvars
├── policies
├── .tflint.hcl
└── ...
```

```
# dev environment

terraform init -backend=false
terraform validate
tflint
trivy config --policy ./policies .
```

Shift left on
… quality
… security
… compliance

Online testing

```
.
├── backend.tf
├── config.dev.tfbackend
├── config.test.tfbackend
├── config.prod.tfbackend
├── main.tf
├── provider.tf
├── terraform.tfvars
├── variables.tf
├── terraform-dev.tfvars
├── terraform-test.tfvars
├── terraform-prod.tfvars
├── tests
│   └── stack-test.go
└── ...
```

```
cd test
go test -v .
```

Don’t test the framework, but the behaviour

https://terratest.gruntwork.io/docs/getting-started/quick-start/

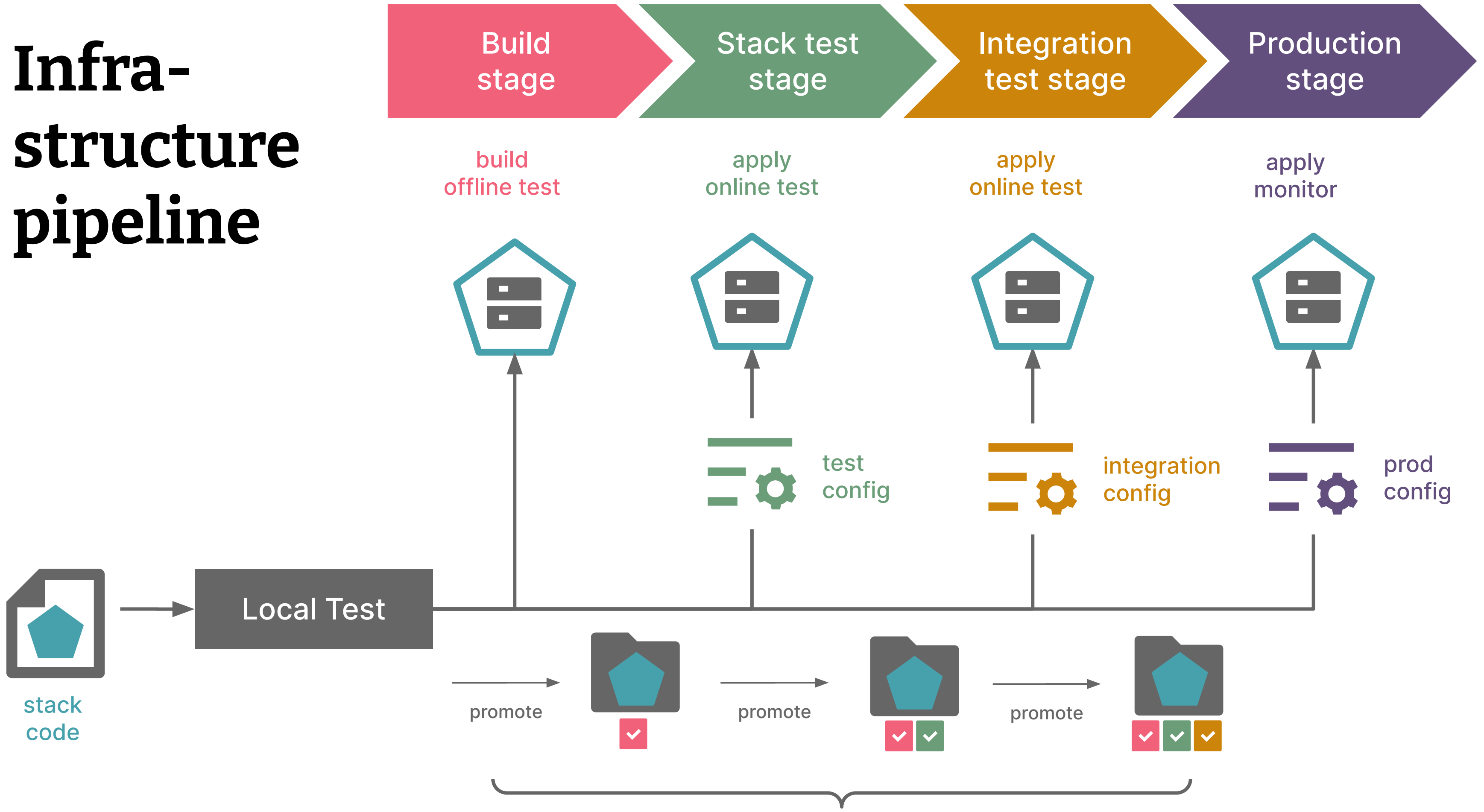
Reuse tested code across all environments

```
.
├── backend.tf
├── config.dev.tfbackend
├── config.test.tfbackend
├── config.prod.tfbackend
├── main.tf
├── provider.tf
├── terraform.tfvars
├── variables.tf
├── terraform-dev.tfvars
├── terraform-test.tfvars
├── terraform-prod.tfvars
└── ...
```

Avoid untested snowflake envs by factoring out configuration

```
# dev environment

terraform init -backend-config= config.dev.tfbackend
terraform plan -var-file= terraform-dev.tfvars
terraform apply -var-file= terraform-dev.tfvars

# test environment

terraform init -backend-config= config.test.tfbackend
terraform plan -var-file= terraform-test.tfvars
terraform apply -var-file= terraform-test.tfvars

# prod environment
terraform init -backend-config= config.prod.tfbackend
...
```

Infra-structure pipeline



promote stack code across instances without code changes

Familiar workflow

1.

Write (infrastructure) code

2.

Commit your changes

3.

Push

Three principles of Continuous Delivery for Infrastructure

✓ **1. Everything as code and in version control**

- Infrastructure code
- Configuration
- Tests
- Compliance
- Pipeline
- Automation scripts

✓ **2. Continuously test and deliver all work in progress**

- Build quality in
- Test as you work
- Integrate at least daily
- Reproducibility
- Proper audit trail

3. Small, simple pieces that you can change independently

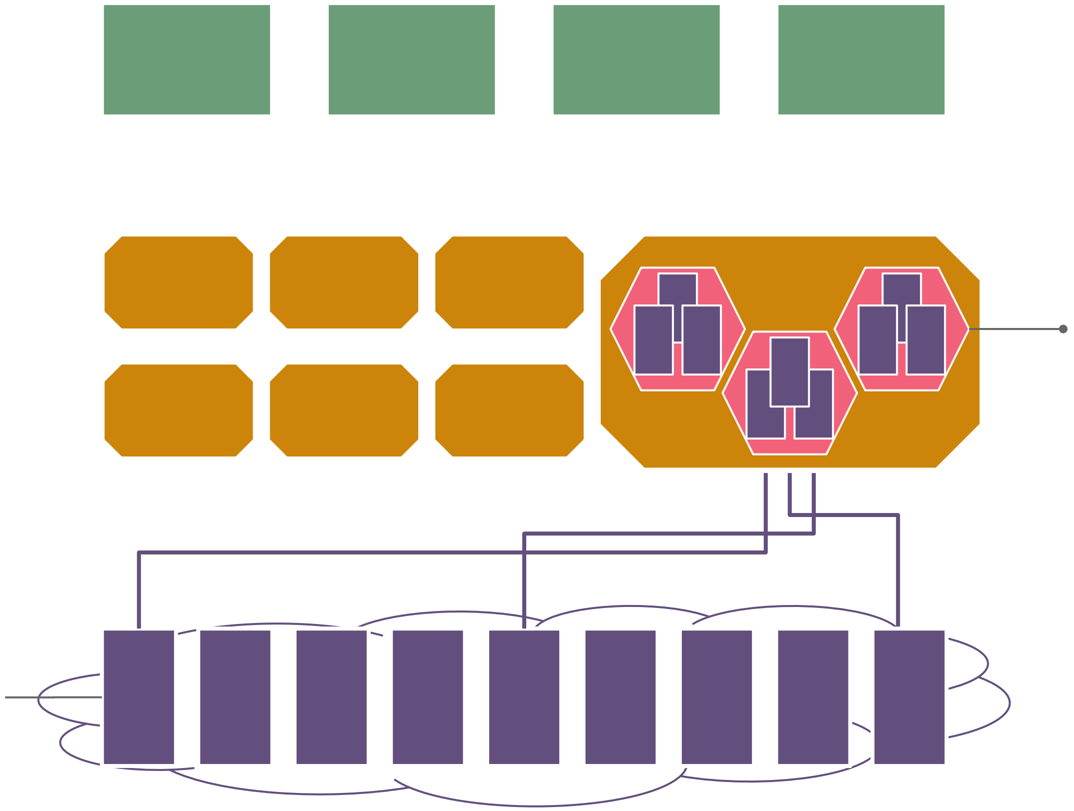
- Reduce complexity
- Shorten feedback cycles
- Reduce blast radius
- Apply proper permission boundaries

Key units of infrastructure architecture

Business capabilities
Products and applications

Technology capabilities
Eg. offered as an internal developer platform

Infrastructure resources
The services that the cloud providers offer



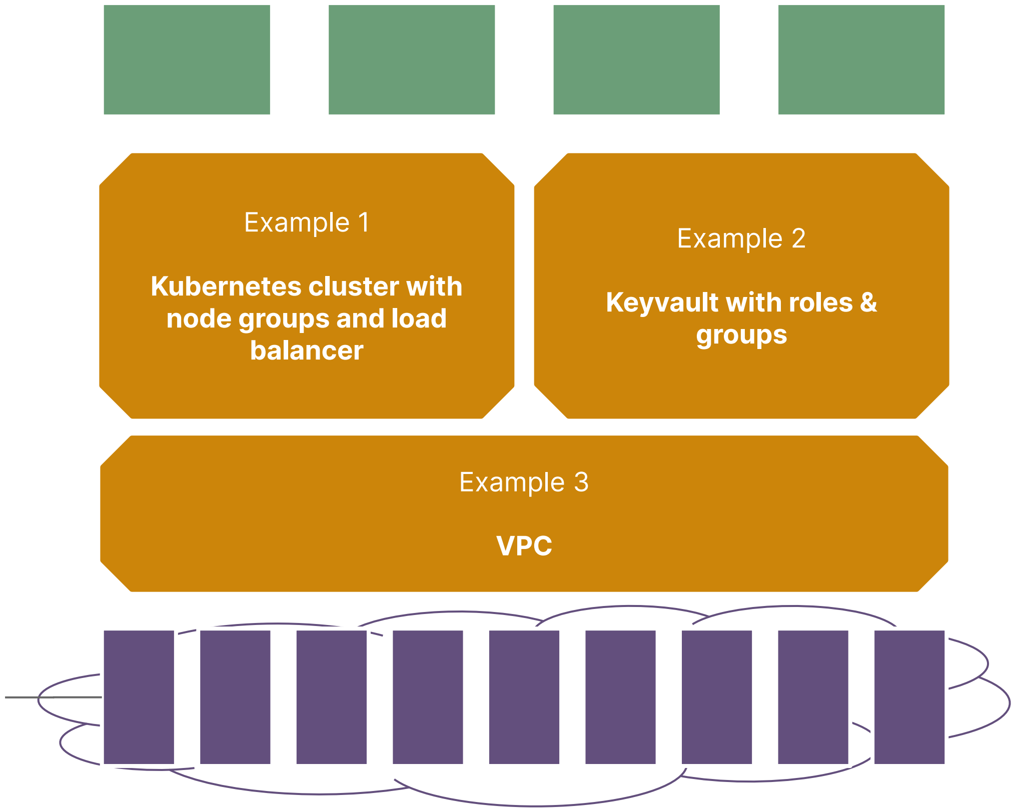
Infrastructure stacks
An infrastructure stack is a collection of cloud infrastructure resources, managed as a group

Key units of infrastructure architecture

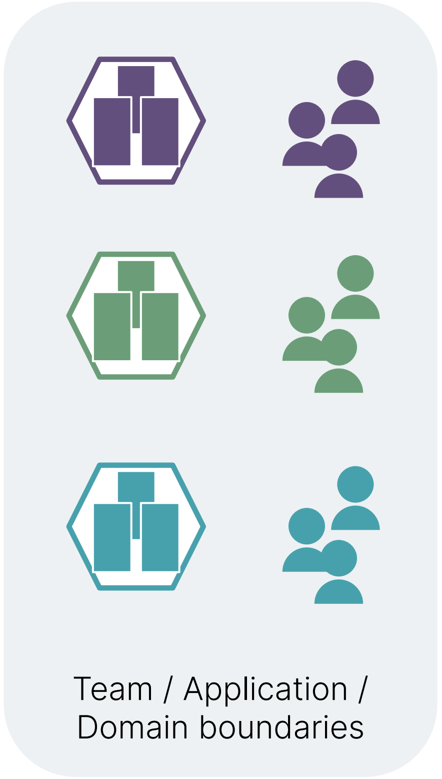
Business capabilities
Products and applications

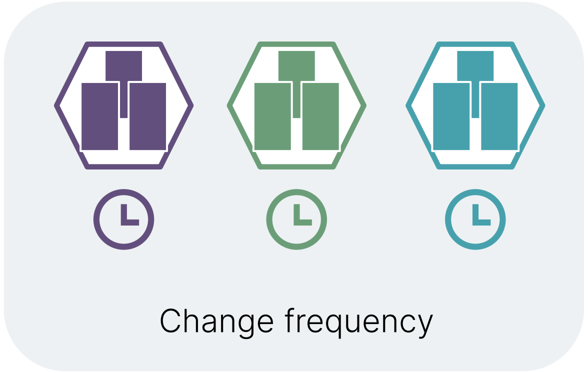
Technology capabilities
Eg. offered as an internal developer platform

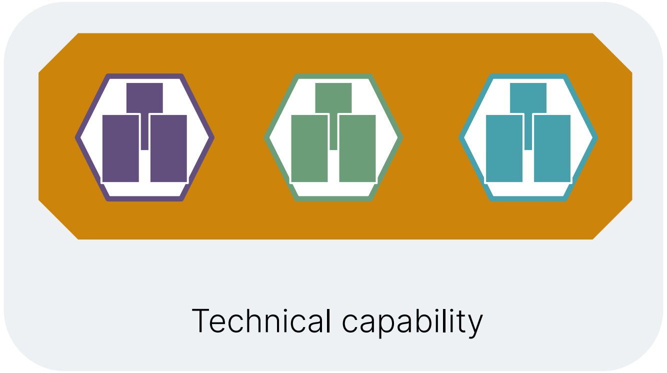
Infrastructure resources
The services that the cloud providers offer

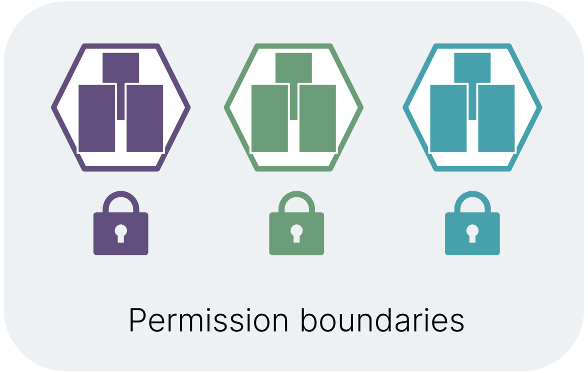

Example 1
Kubernetes cluster with node groups and load balancer
Example 2
Keyvault with roles & groups
Example 3
VPC

Criteria for slicing infrastructure stacks


Team / Application / Domain boundaries


Change frequency


Technical capability


Permission boundaries

Careful with "re-use" - can easily become a bottleneck (DRY vs autonomy)

Three principles of Continuous Delivery for Infrastructure

1. Everything as code and in version control

- Infrastructure code
- Configuration
- Tests
- Compliance
- Pipeline
- Automation scripts

2. Continuously test and deliver all work in progress

- Build quality in
- Test as you work
- Integrate at least daily
- Reproducibility
- Proper audit trail

3. Small, simple pieces that you can change independently

- Reduce complexity
- Shorten feedback cycles
- Reduce blast radius
- Apply proper permission boundaries

Getting started

/thoughtworks

The Walking Skeleton

- Have infra deployment built into earliest iterations of your software product
- Start simple
- Lower the barrier of getting involved
- Practice deployments early & often
- Don't leave the path to production to the last minute

Image by GraphicMama-team from Pixabay

Challenges

thoughtworks

Blast Radius

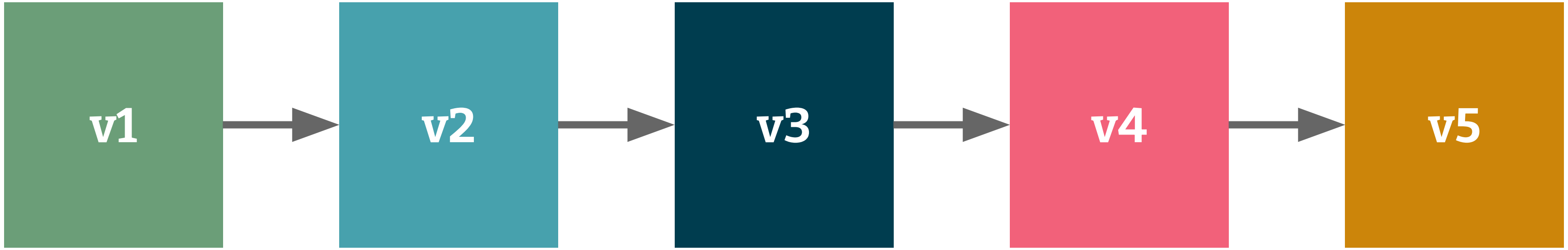
The term *blast radius* describes the potential damage a given change could make to a system. It's usually based on the elements of the system you're changing, what other elements depend on them, and what elements are shared.

Kief Morris, Infrastructure as Code 2nd Edition

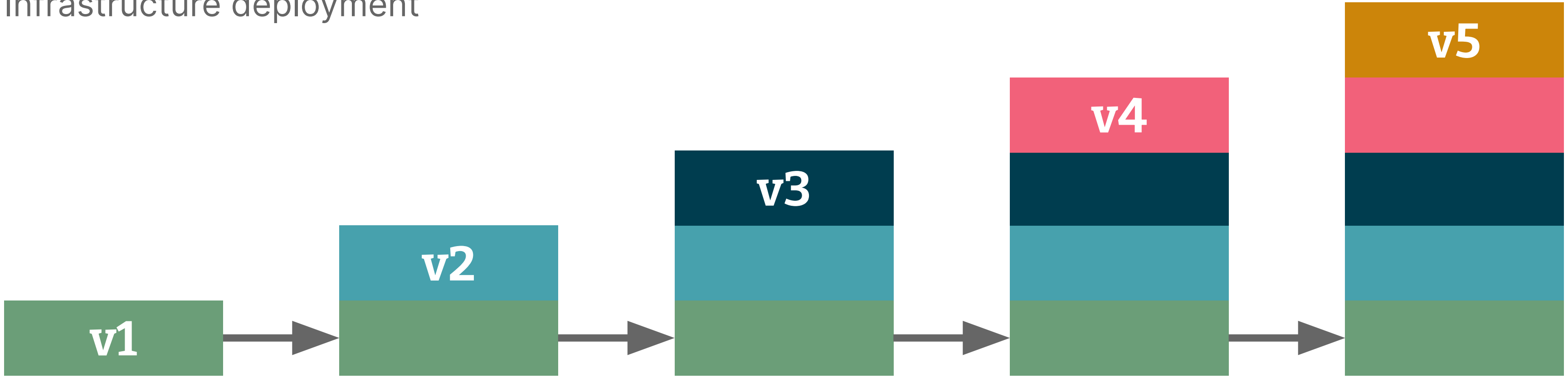
Image Source: commons.wikimedia.org

(Im)mutable deployment

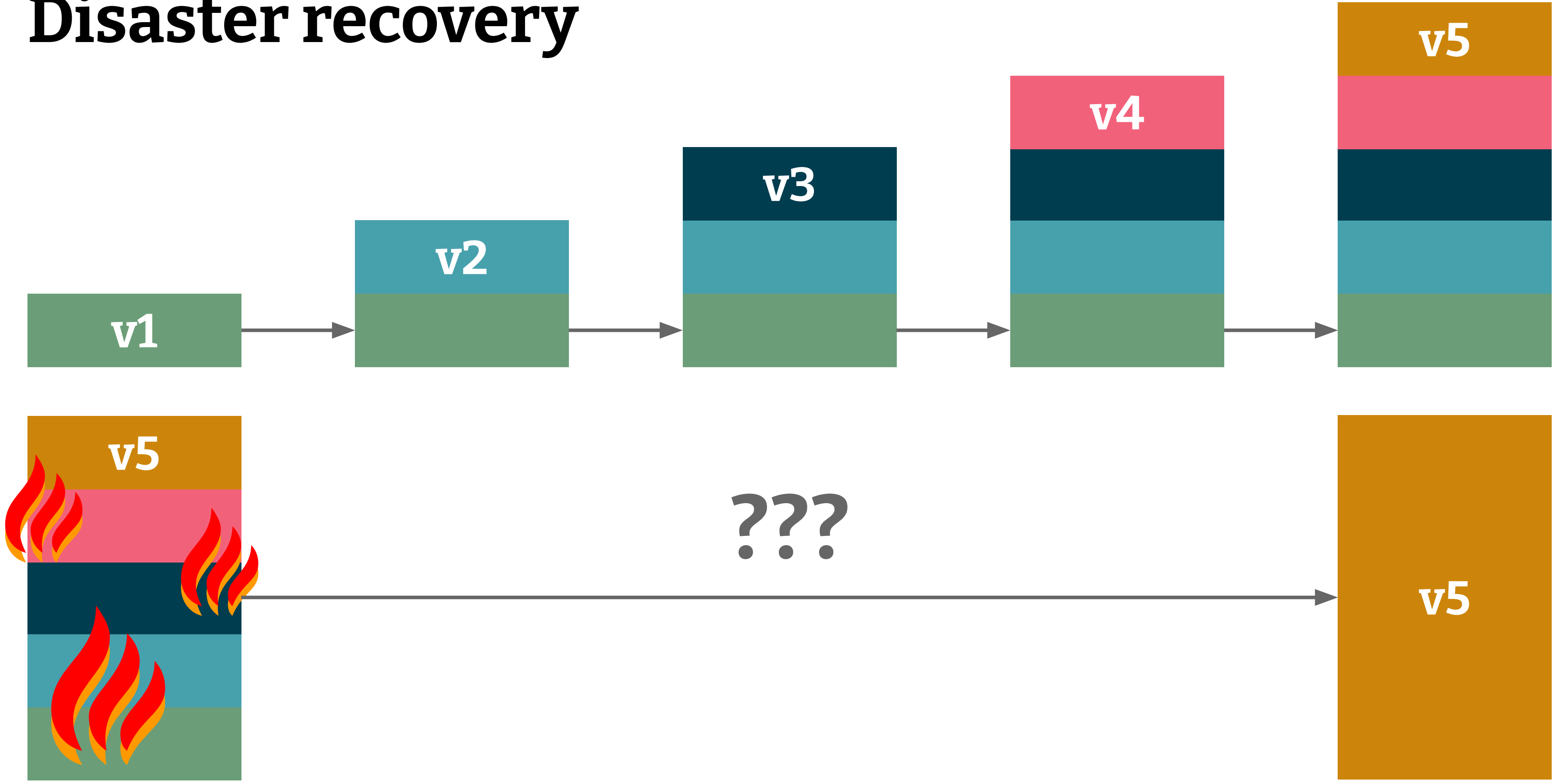
(Modern) application deployment



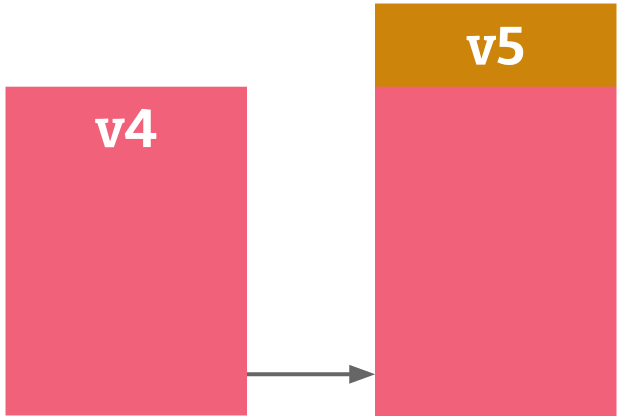
Infrastructure deployment



Disaster recovery



Ideal pipeline run



Build current state of production

Apply latest version

Summary

thoughtworks

Three principles of Continuous Delivery for Infrastructure

There's more to it than writing Infrastructure code!

1. Everything as code and in version control

- Infrastructure code
- Configuration
- Tests
- Pipeline
- Automation scripts

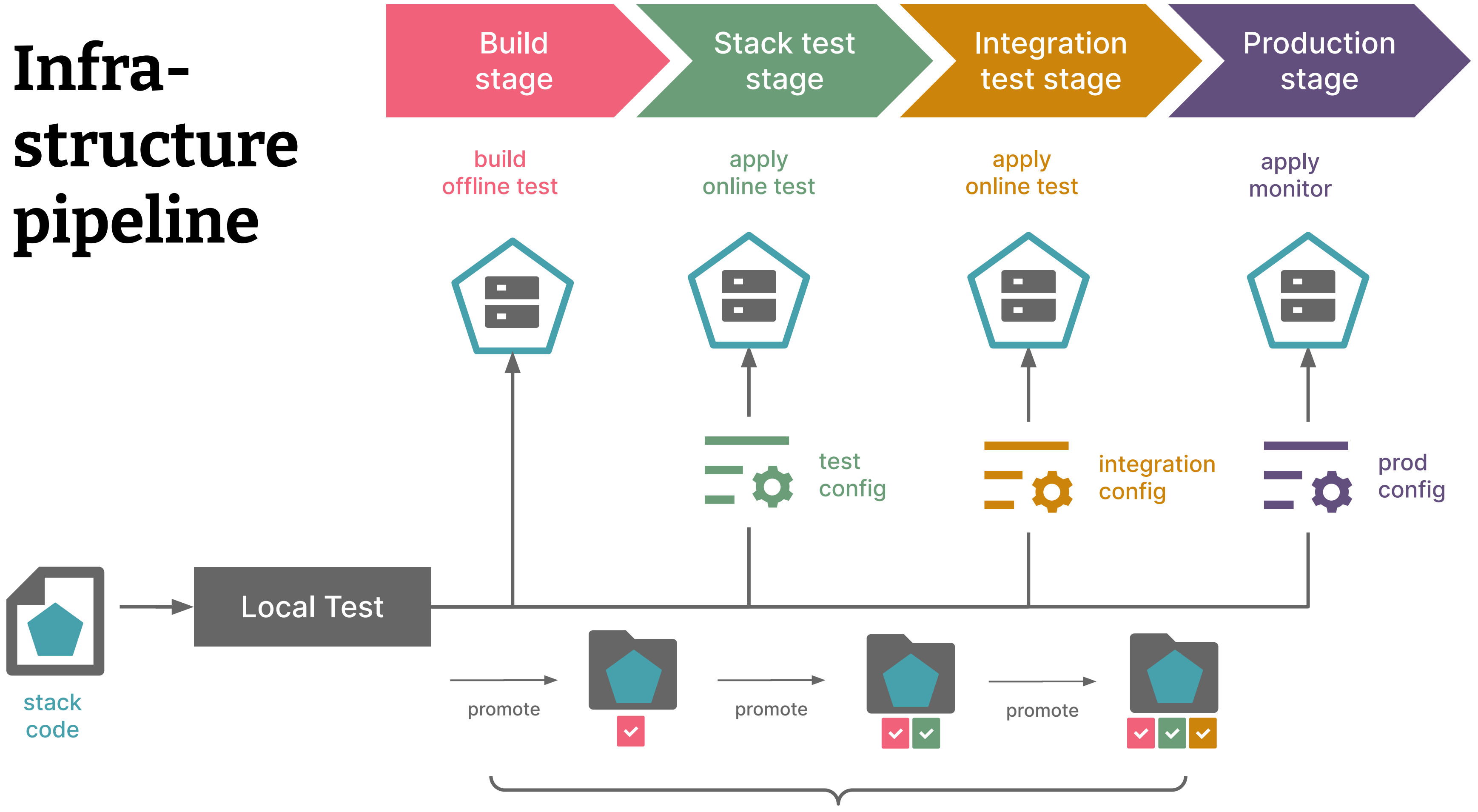
2. Continuously test and deliver all work in progress

- Build quality in
- Test as you work
- Integrate at least daily
- Reproducibility
- Proper audit trail

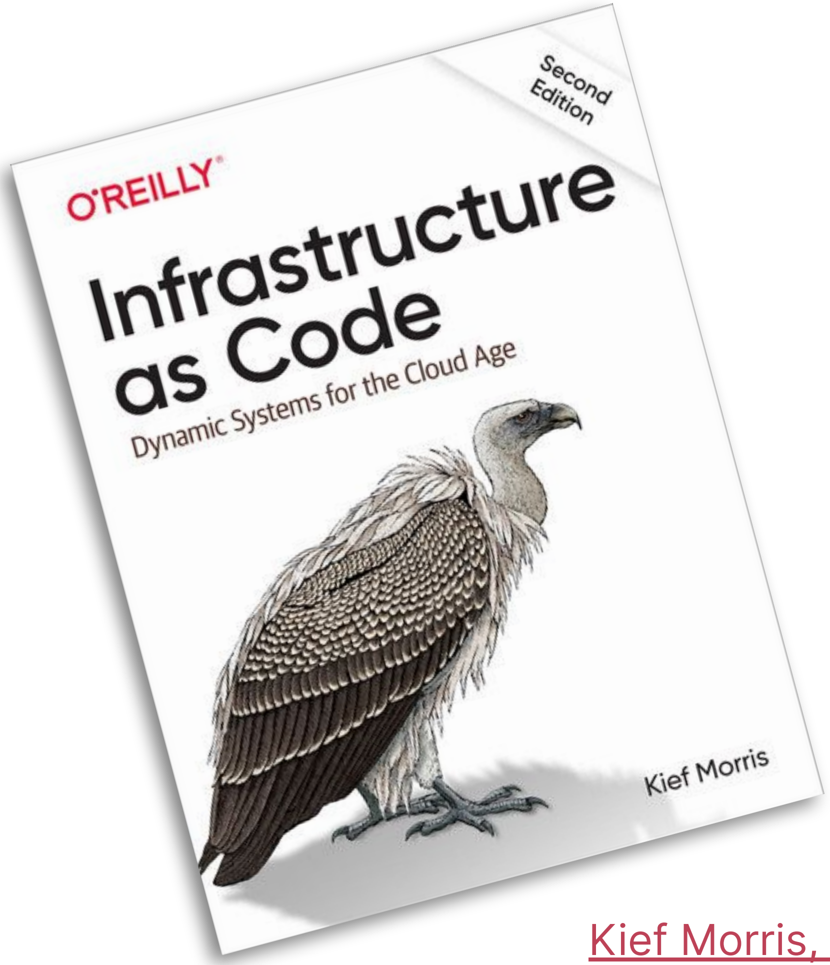
3. Small, simple pieces that you can change independently

- Reduce complexity
- Shorten feedback cycles
- Reduce blast radius
- Apply proper permission boundaries

Infra-structure pipeline



promote stack code across instances without code changes



[Kief Morris, Infrastructure as Code - 2nd Edition]

Thank you for your attention

Michael Lihs
Infrastructure Consultant
michael.lihs@thoughtworks.com

linkedin.com/in/michael-lihs/

/thoughtworks

References

- Alaa Mansour & Michael Lihs, Infrastructure Pipelines
- Structuring Hashicorp Terraform Configuration for Production
- Running Terraform in Automation
- Test-Driven Development for Infrastructure
- Demo Repository: Handling Environment Variables
- What if Infrastructure-as-Code never existed