Getting started with CI/CD for cloud infrastructure

Waldemar Kindler
Michael Lihs

/thoughtworks

Poll

Tools like Terraform are well established

Pipelines for infrastructure automation

Infrastructure changes take ages until applied to production

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

How can we deliver infrastructure faster and safer...

... by making changes in small batches

... by making the process dead simple & easily reproducible

... 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
- Compliance
- 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

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.

Branching strategy
If you us branches merge at least daily!

Reproducibility
Tooling and infrastructure are versioned together.

Three principles of Continuous Delivery for Infrastructure

1. Everything as code and in version control

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

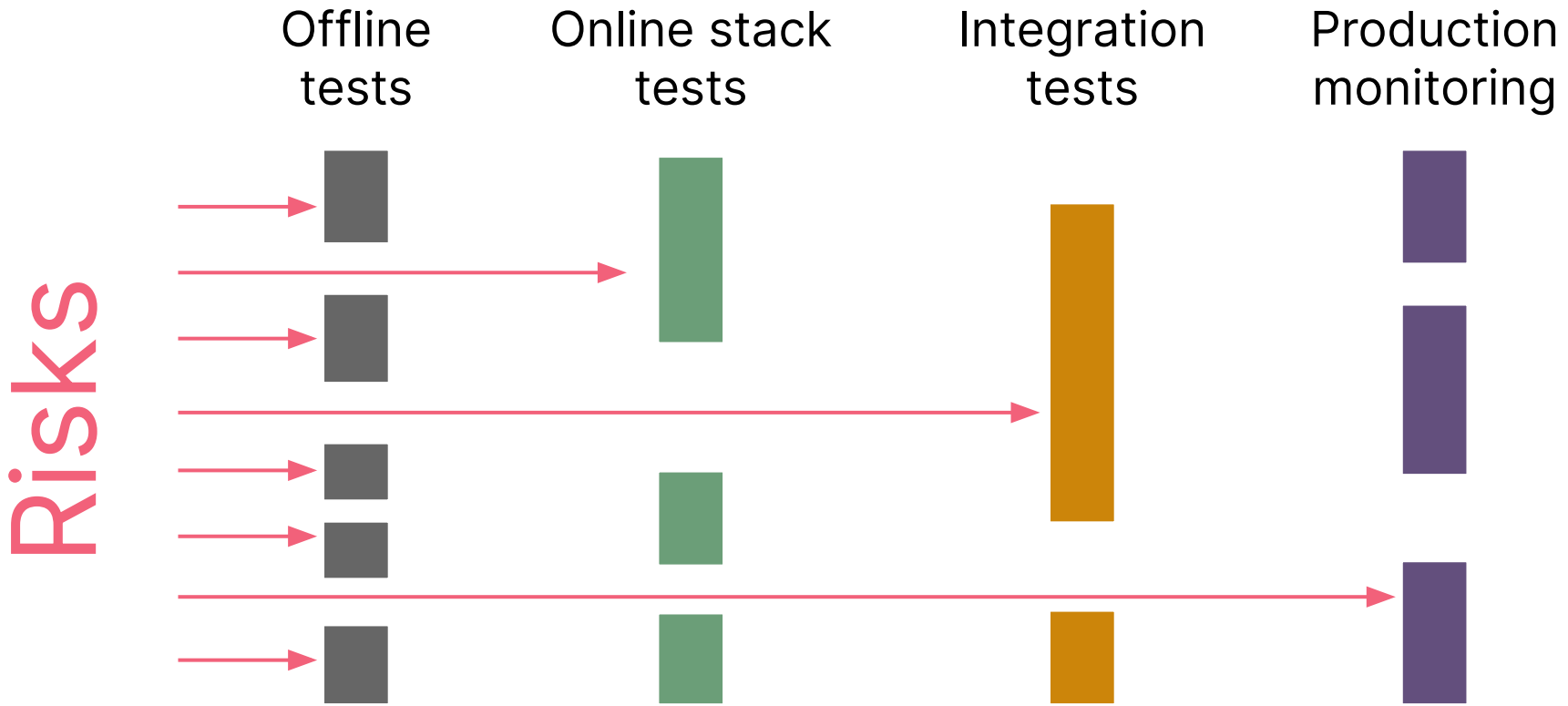
2. Continuously test and deliver all work in progress

- 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

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

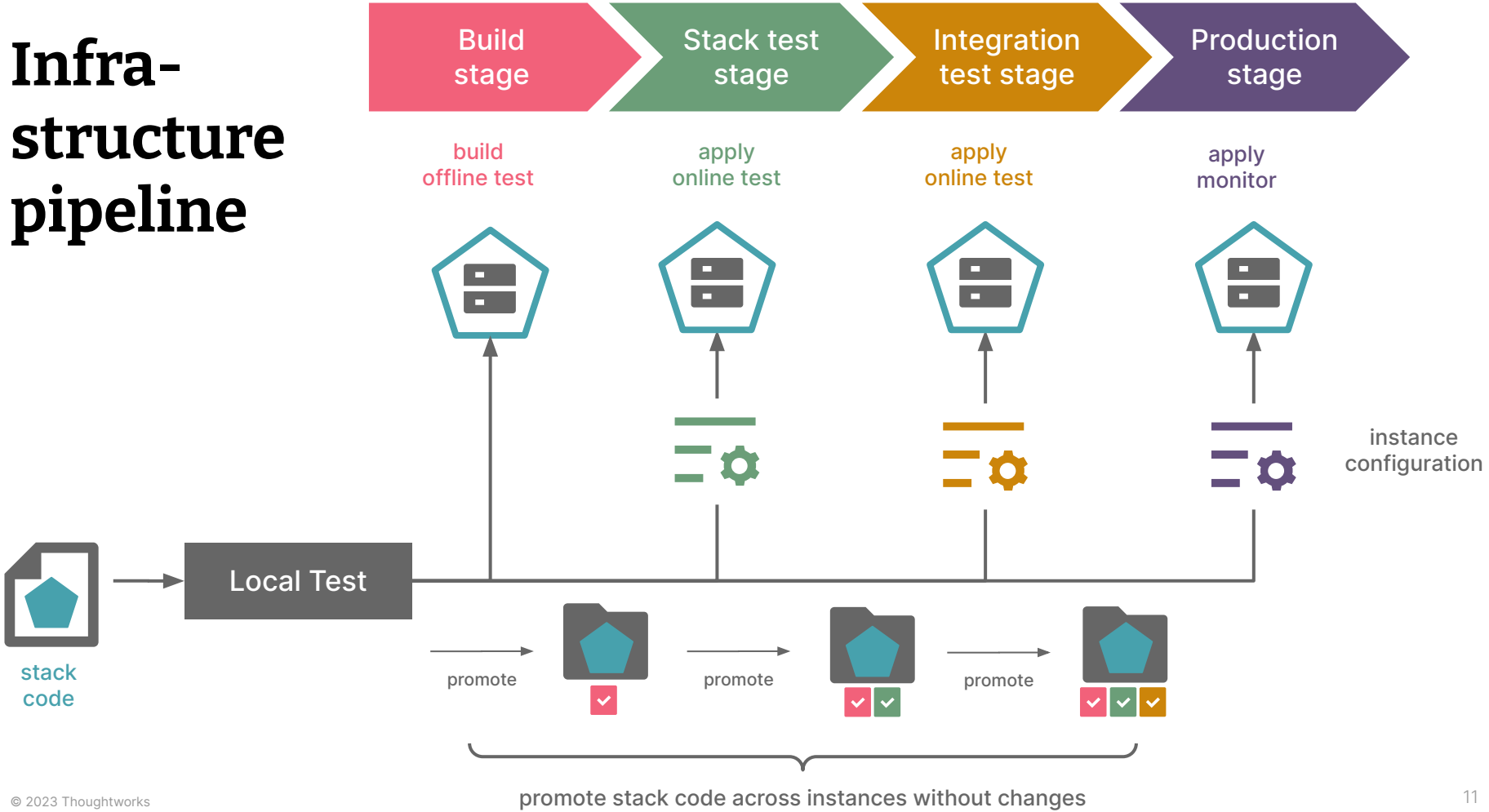
https://terratest.gruntwork.io/docs/getting-started/quick-start/

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
└── ...
```


```
# 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
 ...
```

Infra-structure pipeline



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
- Pipeline
- Automation scripts

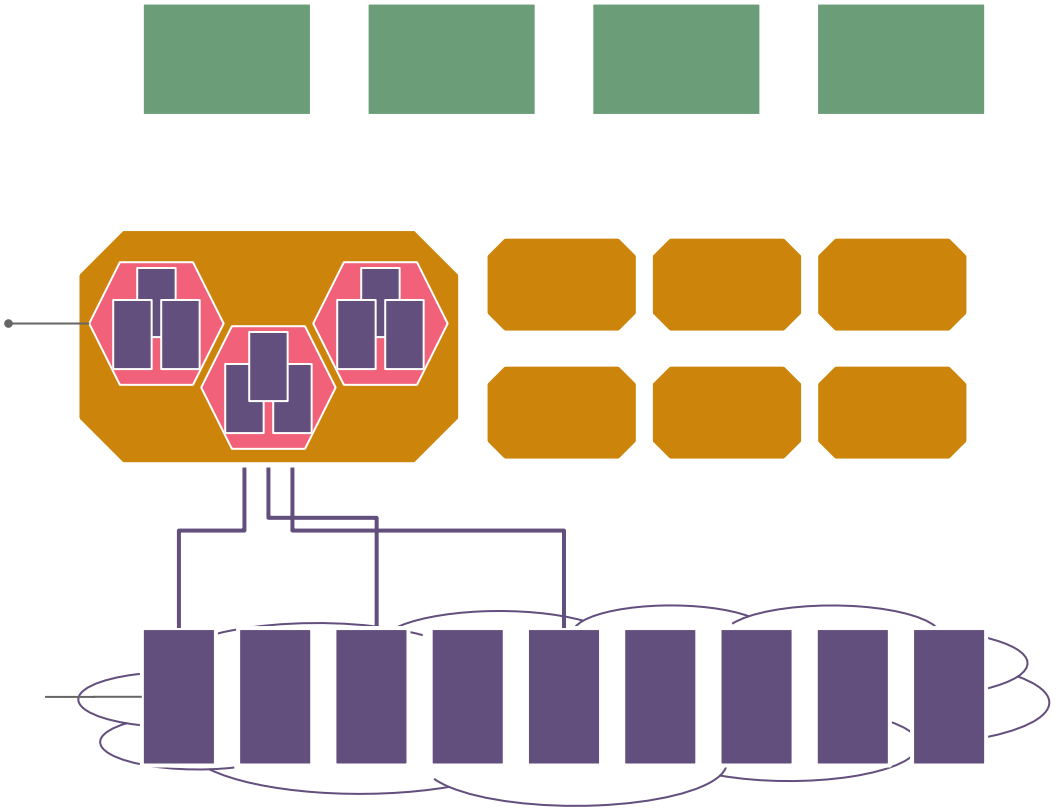
2. Continuously test and deliver all work in progress

- 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

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

Technology capabilities
Eg. offered as an engineering platform

Infrastructure resources
The services that the cloud providers offer

Criteria for slicing infrastructure stacks

Team / Application / Domain boundaries

Change frequency

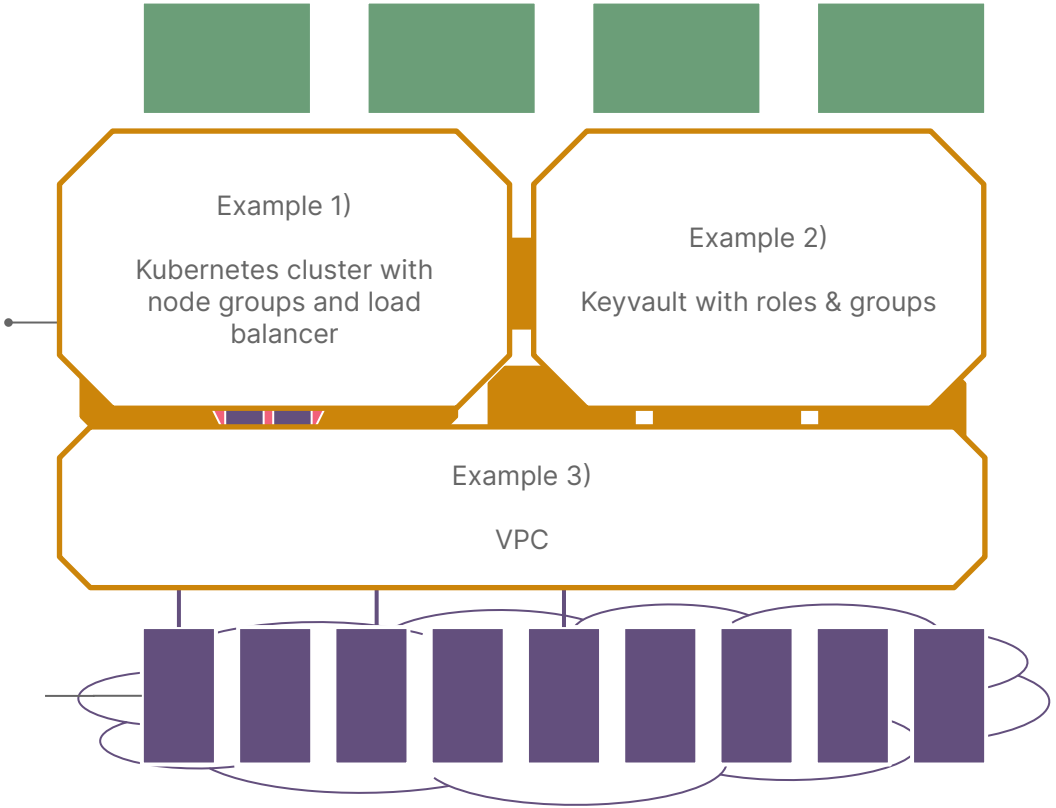
Technical capability

Permission boundaries

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

Key units of infrastructure architecture

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



Business capabilities
Products and applications

Technology capabilities
Eg. offered as an engineering platform

Infrastructure resources
The services that the cloud providers offer

Three principles of Continuous Delivery for Infrastructure

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

Challenges

thoughtworks

Blast Radius

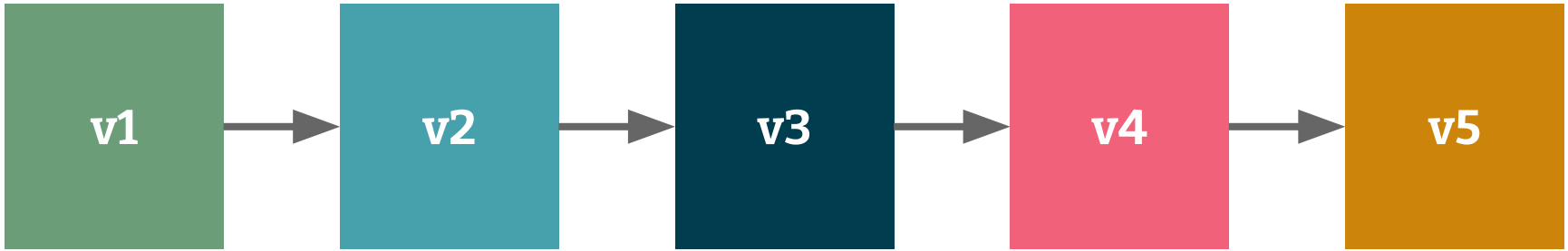
The term *blast radius* describes the potential damage a given change could make to a system. It's usually based on the elements of the system you're changing, what other elements depend on them, and what elements are shared.

Kief Morris, Infrastructure as Code 2nd Edition

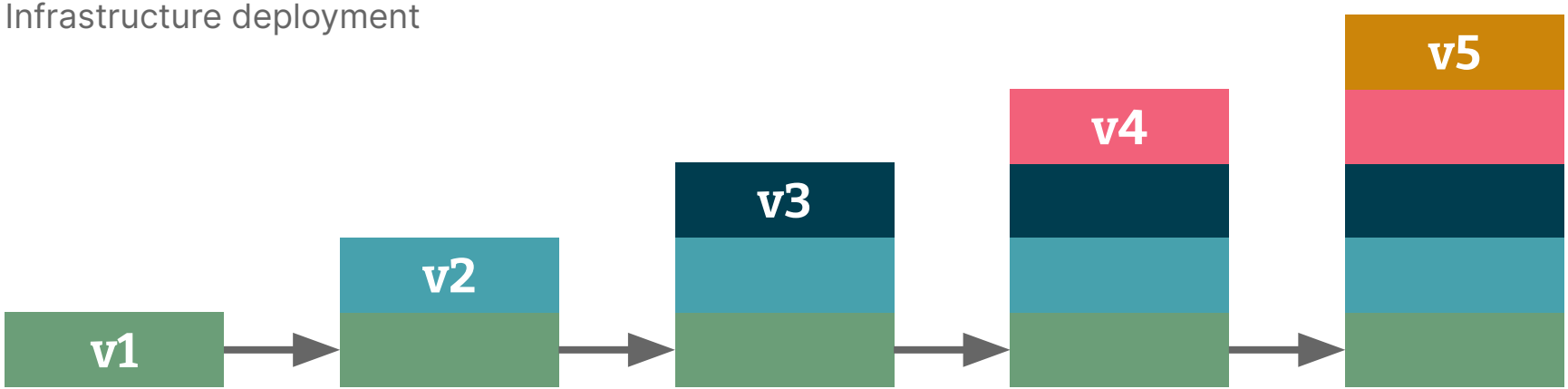
Image Source: commons.wikimedia.org

(Im)mutable deployment

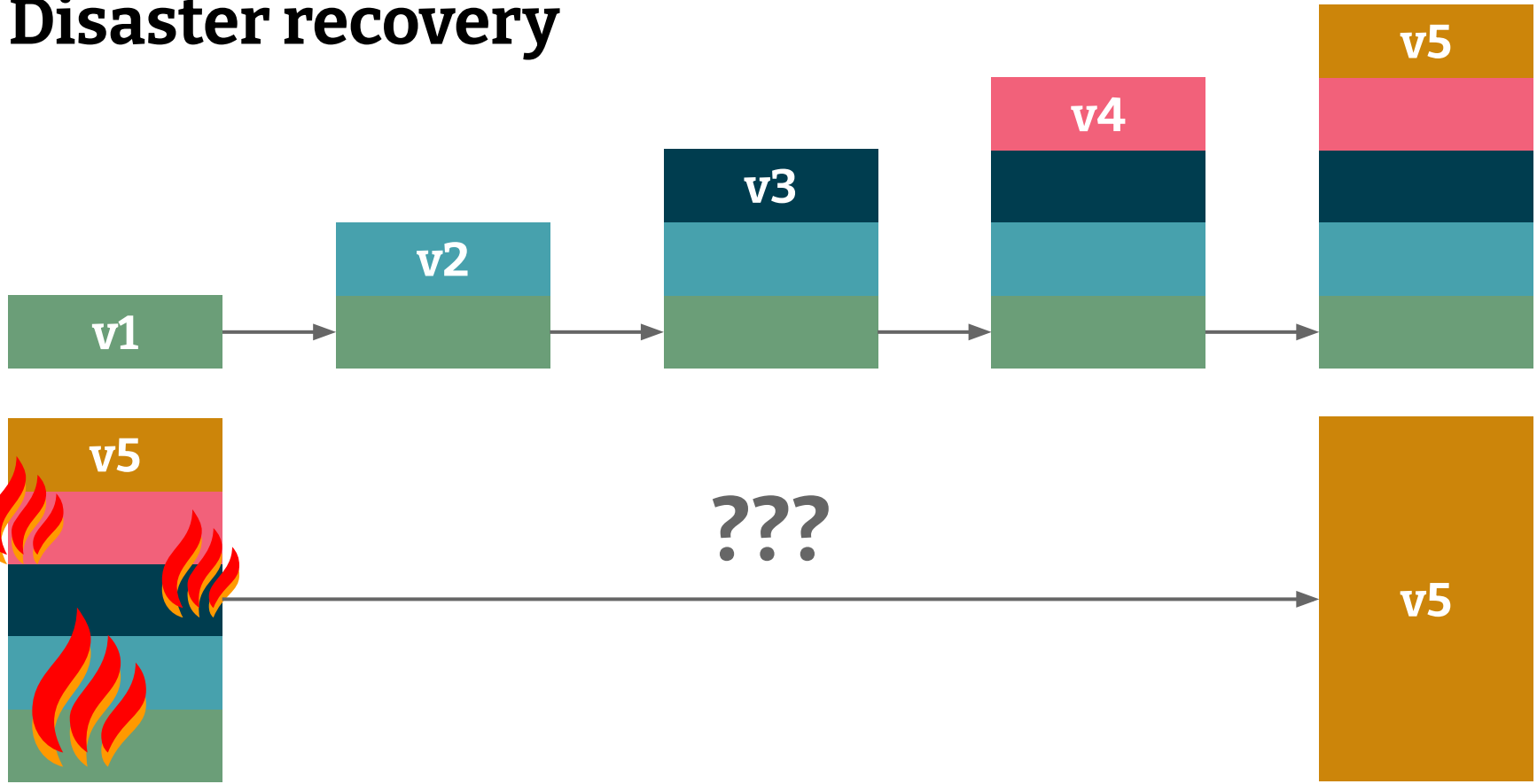
(Modern) application deployment



Infrastructure deployment



Disaster recovery



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

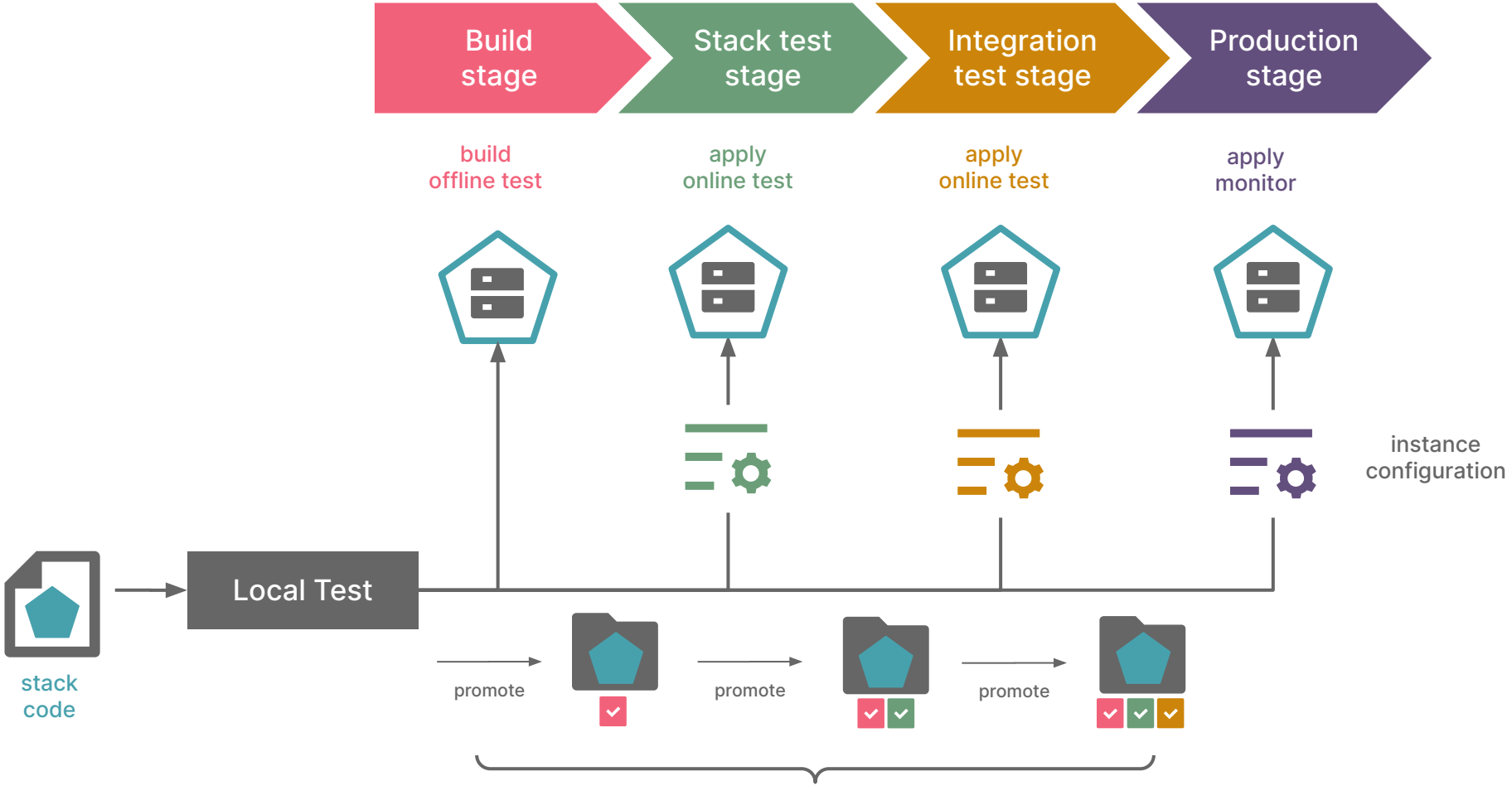
- Infrastructure code
- Configuration
- Tests
- Pipeline
- Automation scripts

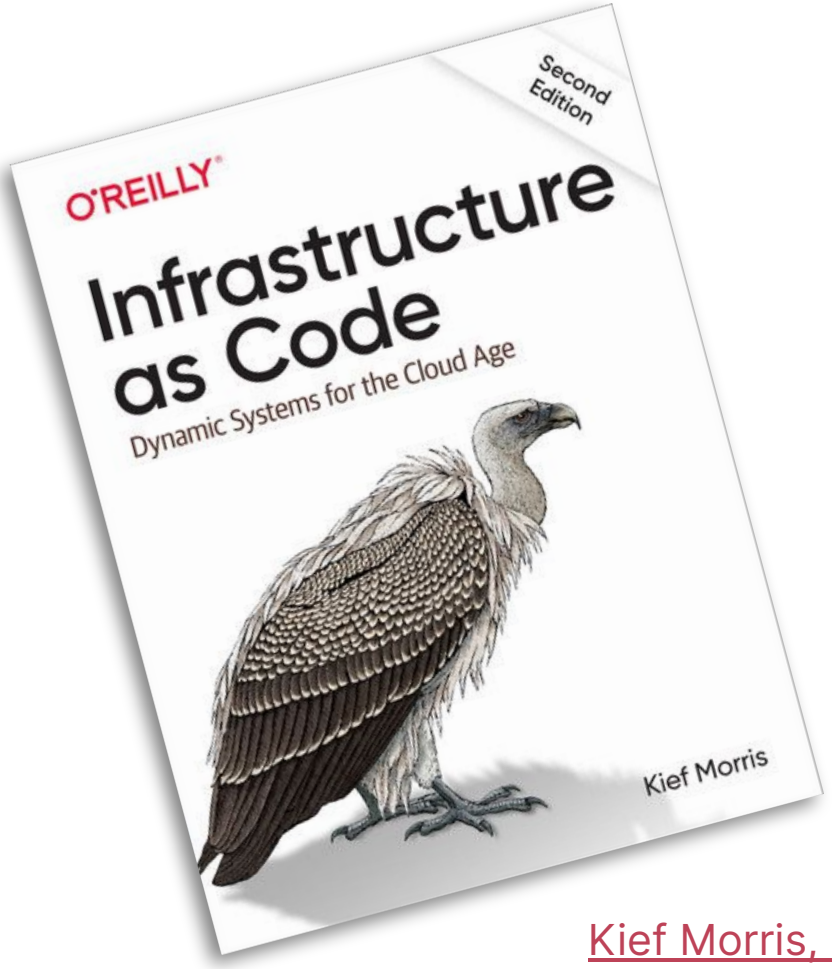
2. Continuously test and deliver all work in progress

- 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





Kief Morris, Infrastructure as Code - 2nd Edition

Thank you for your attention

Waldemar Kindler
Infrastructure Consultant
waldemar.kindler@thoughtworks.com

linkedin.com/in/waldemar-kindler/

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