Getting started with CI/CD pipelines for cloud infrastructure

Michael Lihs

/thoughtworks

Michael Lihs

Infrastructure Consultant
@Thoughtworks

father of 👶 and 👧

🏘️ Tübingen

🚴 cycling 🪵 woodworking

@kaktusmimi

What's on the menu today

1. Quick recap on CI/CD and IaC

What was CI/CD again?

2. The infrastructure stack

One codebase to rule them all

3. Testing infrastructure code

Pyramids and swiss cheese

4. Design your Infrastructure Pipeline

Build, Tests, Staging & Promotion

5. Challenges

What's so special about infrastructure code?

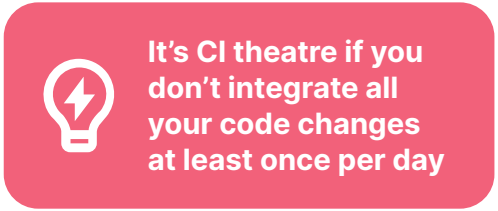
6. Summary

What you should take away from this talk

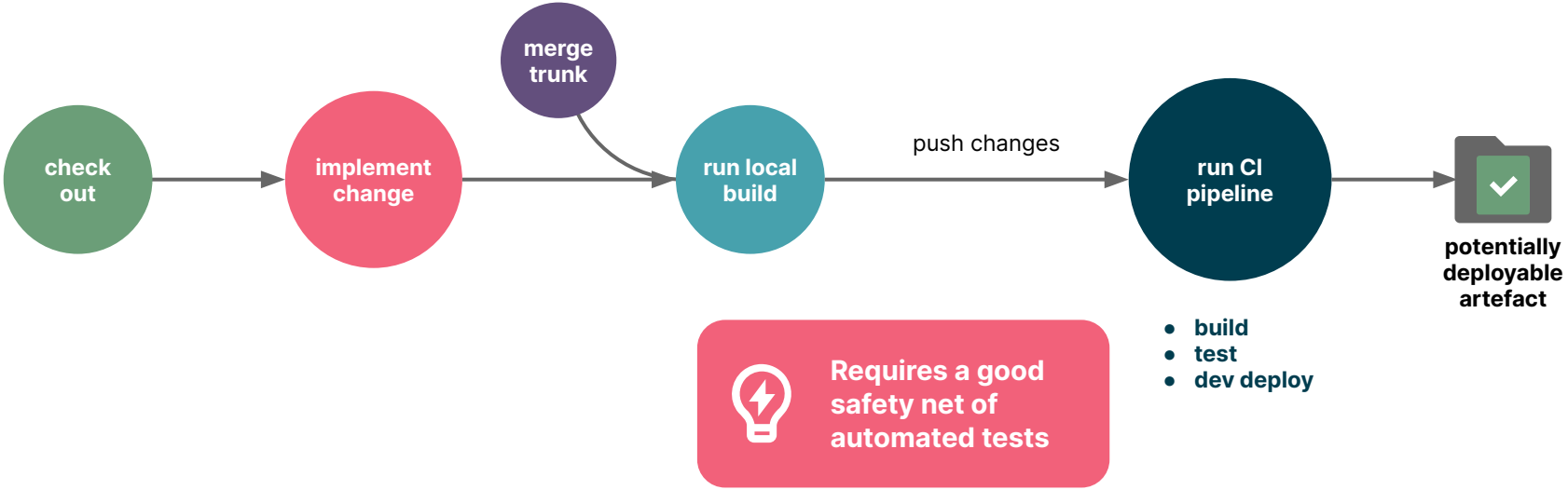
Quick recap
Continuous Integration & Delivery

thoughtworks

Continuous Integration

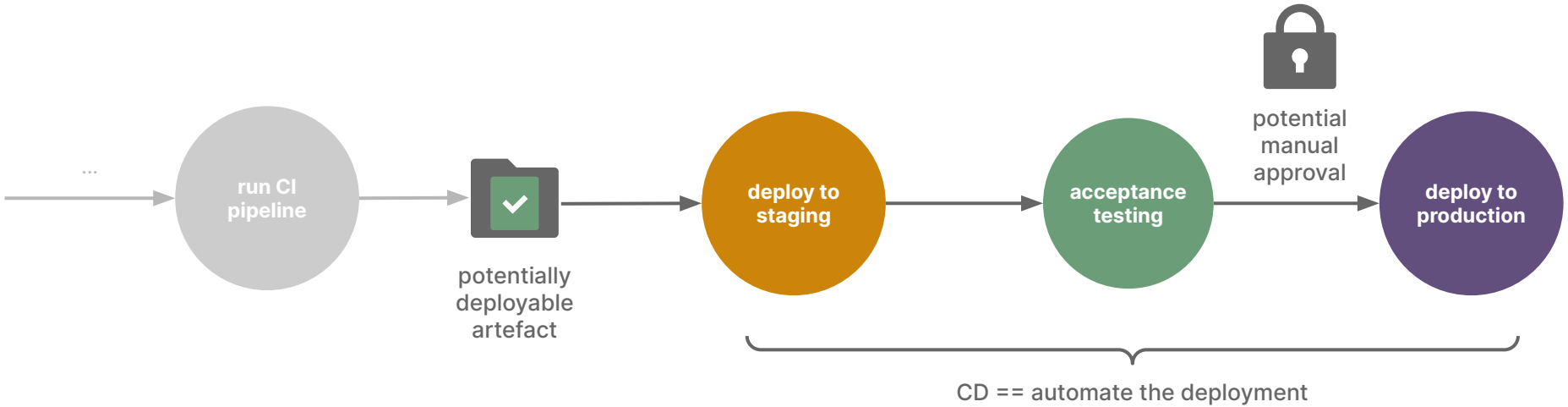


It's CI theatre if you don't integrate all your code changes at least once per day



visualisation of https://martinfowler.com/articles/continuousIntegration.html

Continuous Delivery



Why CI/CD for infrastructure?

We have a situation: one of the team members applied **terraform locally** with a version of the terraform binary older than what was used in the pipeline. Due to that we ran into a **terraform state conflict** that resulted in terraform trying to **re-create all resources**

A fellow Thoughtworker

/thoughtworks

Three core principles of Infrastructure as Code

Infrastructure as Code is an approach to building infrastructure that embraces continuous change for high reliability and quality.

There's more to it than writing Terraform code!

1. Everything as Code

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

2. Continuously test and deliver all work in progress

- Build quality in
- Test as you work
- Integrate at least daily

3. Small, simple pieces that you can change independently

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

Further motivation

Avoid snowflake environments

Have your environments as configuration, use the same codebase for all environments.

Avoid configuration drift

There is one way and one way only to apply changes to your infrastructure

Audit log

Since there is only one way to apply changes, we can easily get an audit log

Familiar workflow

1.

Write (infrastructure) code

2.

Commit your changes

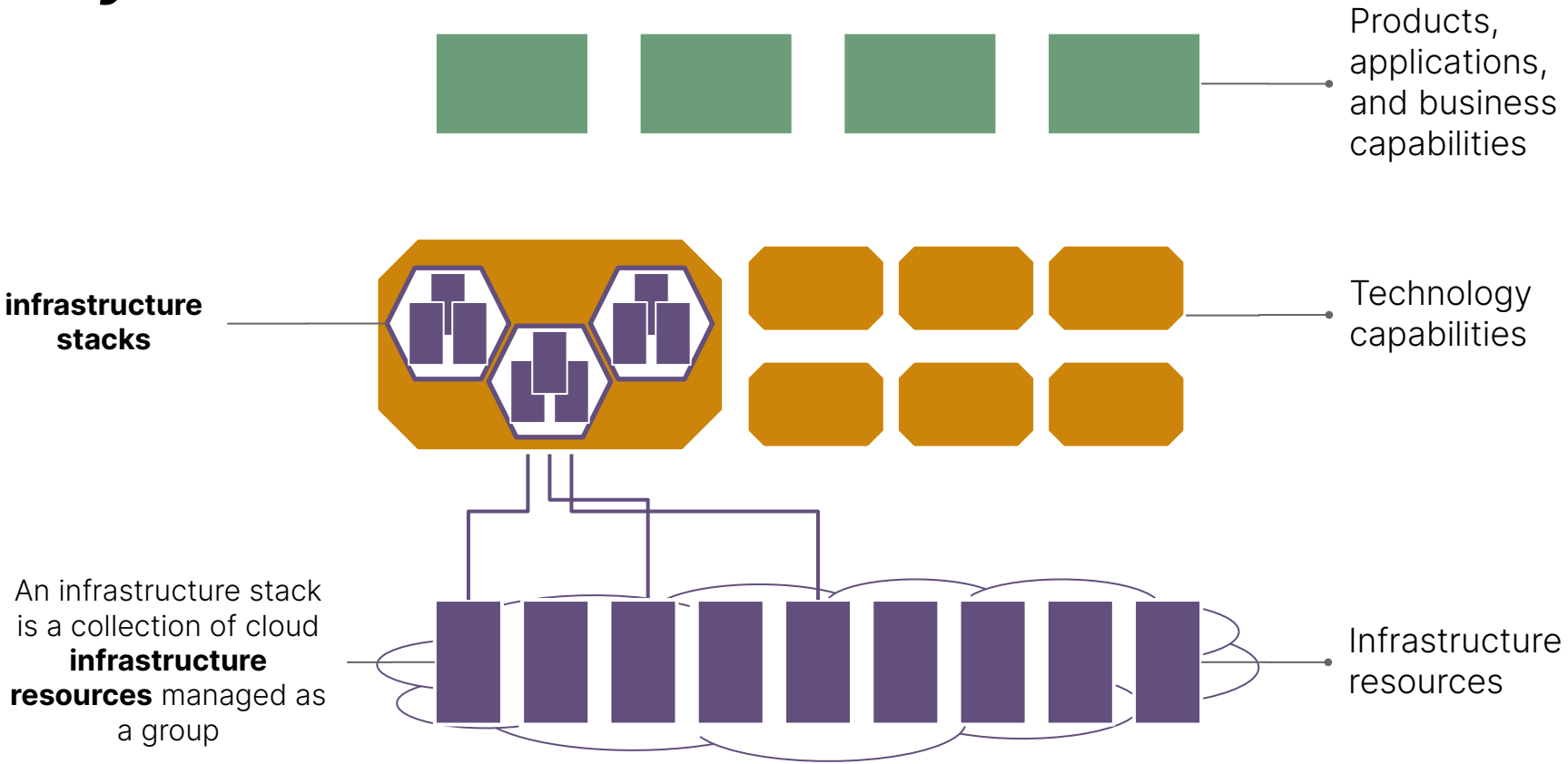
3.

Push

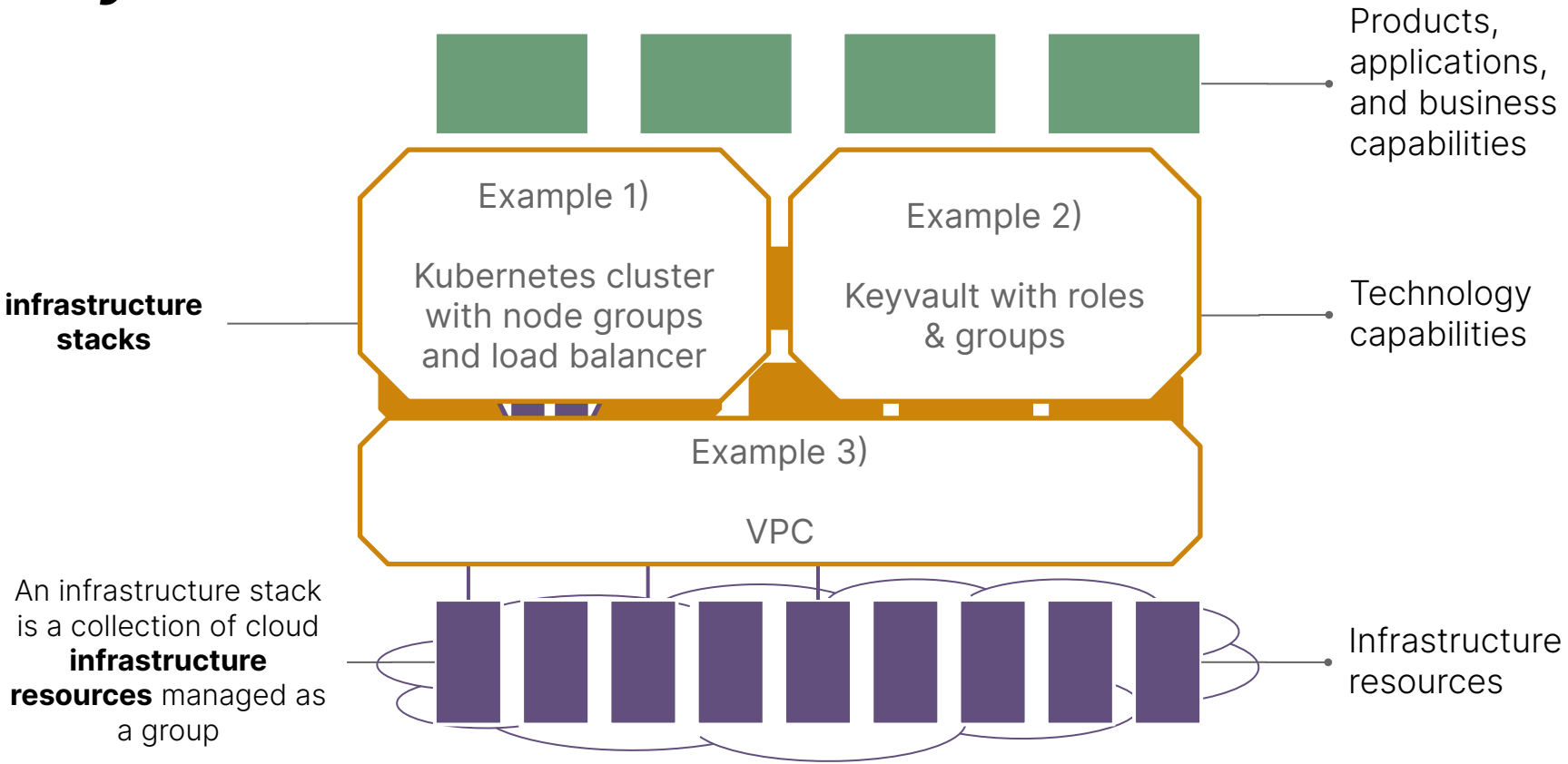
The infrastructure stack

thoughtworks

Key units of infrastructure architecture



Key units of infrastructure architecture



Testing infrastructure code

thoughtworks

What does this test tell us?

Code:

```
subnet:
  name: private_A
  address_range: 192.168.0.0/16
```

Test:

Given:

An AWS account

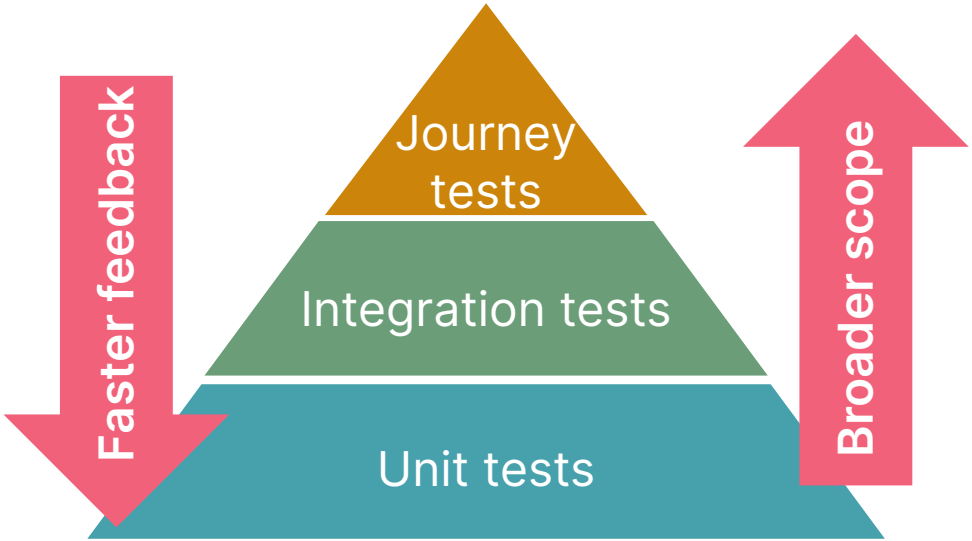
When:

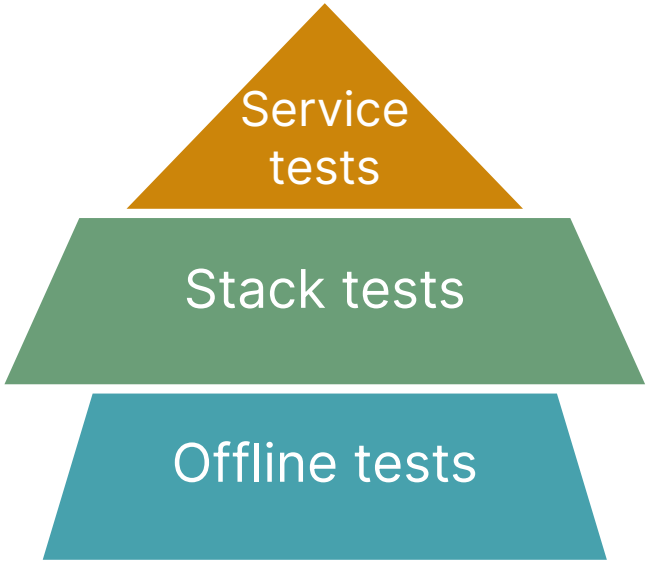
A subnet is created

Then:

the subnet exists and has address block "192.168.0.0/16"

The testing pyramid revisited





Swiss cheese testing model

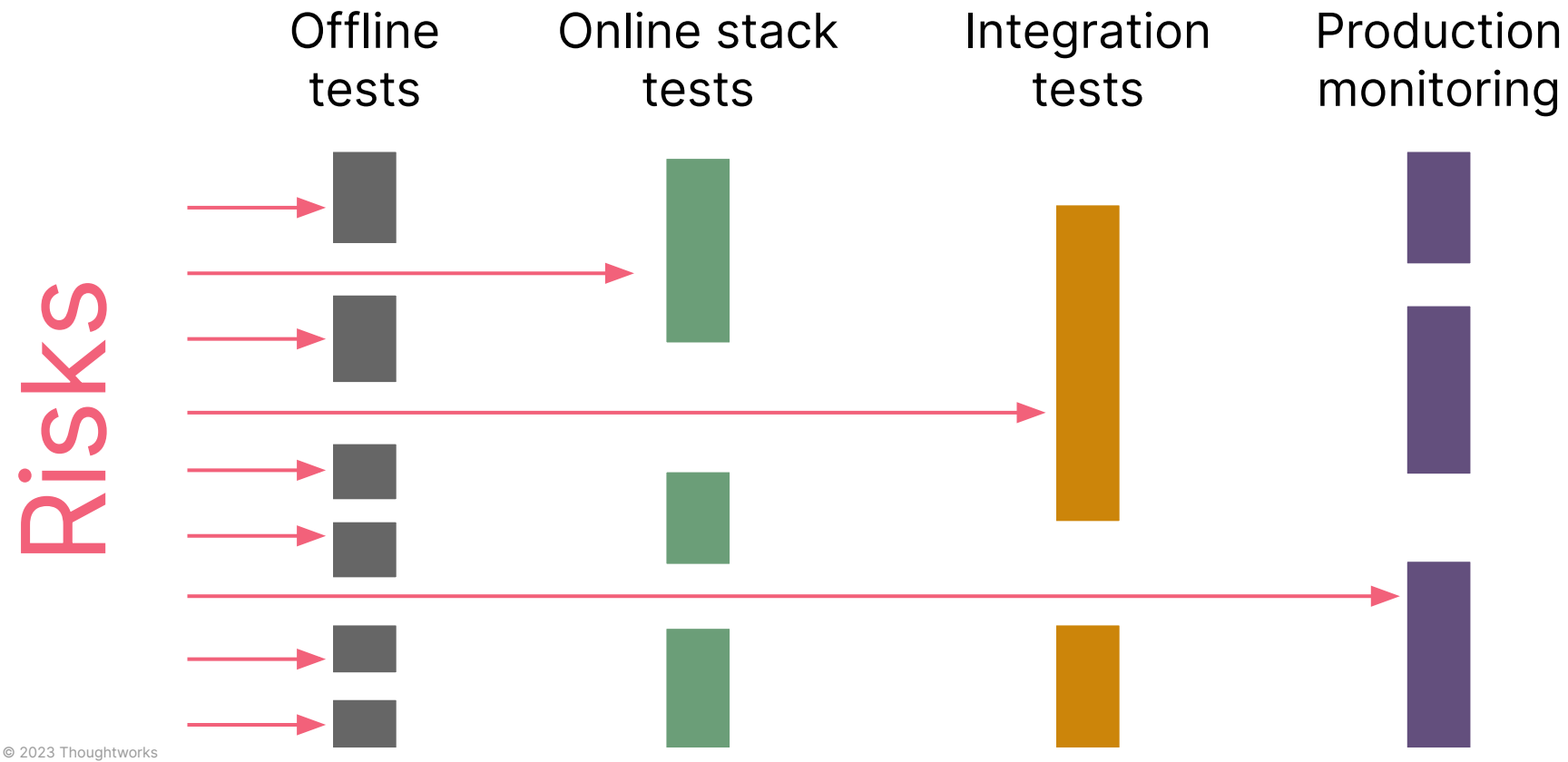
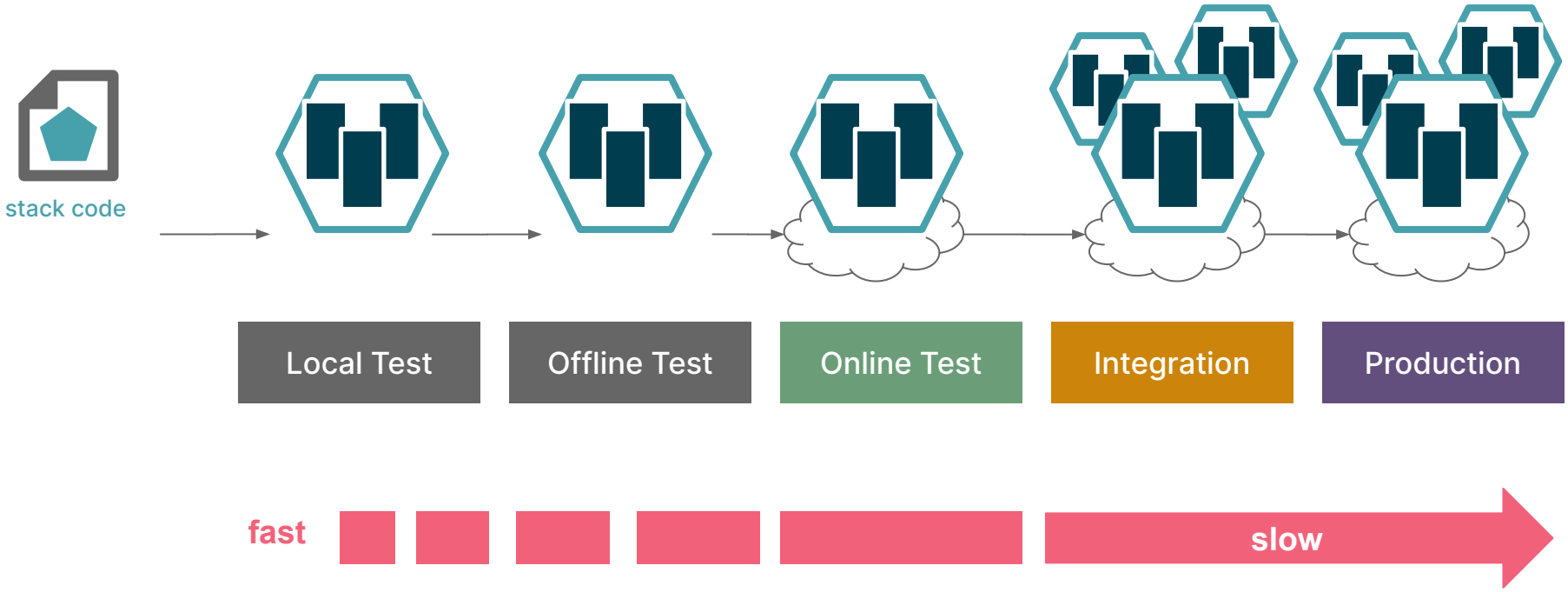


Image taken from "Infrastructure as Code - 2nd edition" by Kief Morris

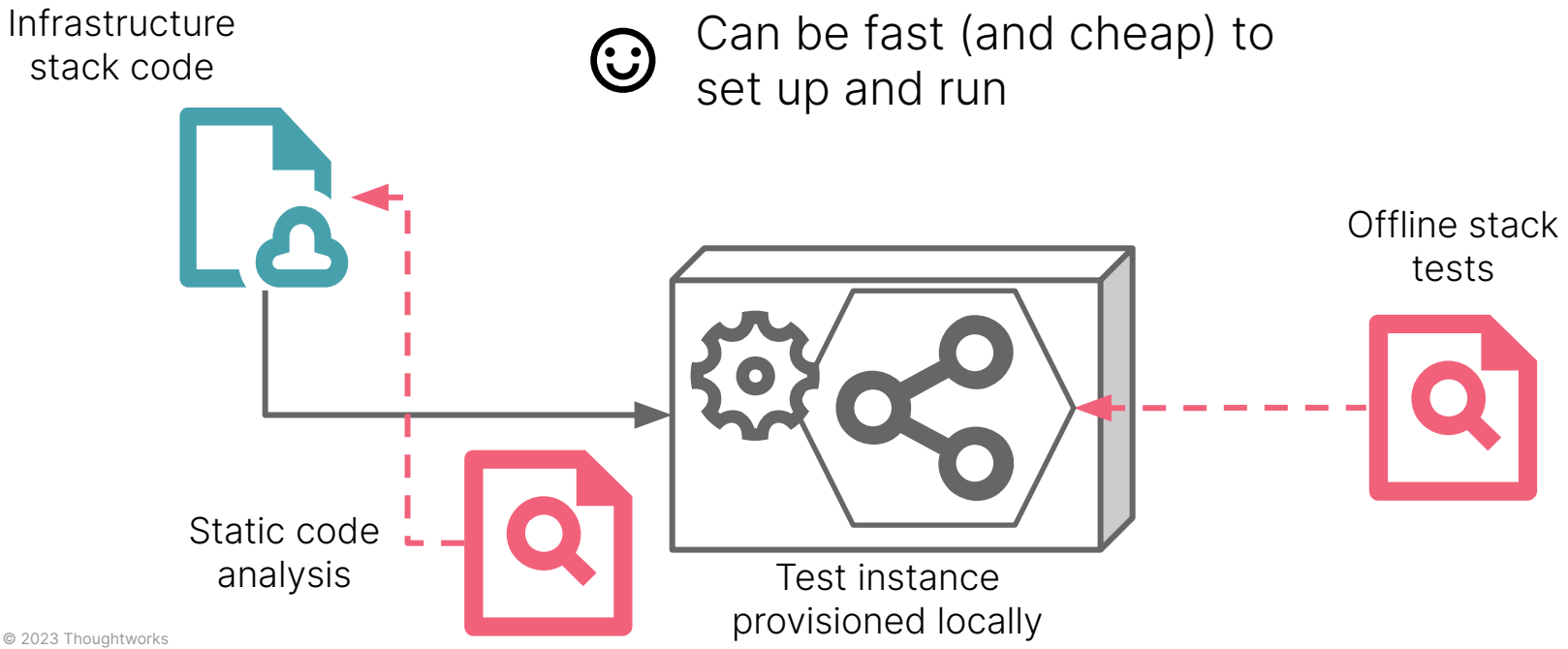
Stack testing



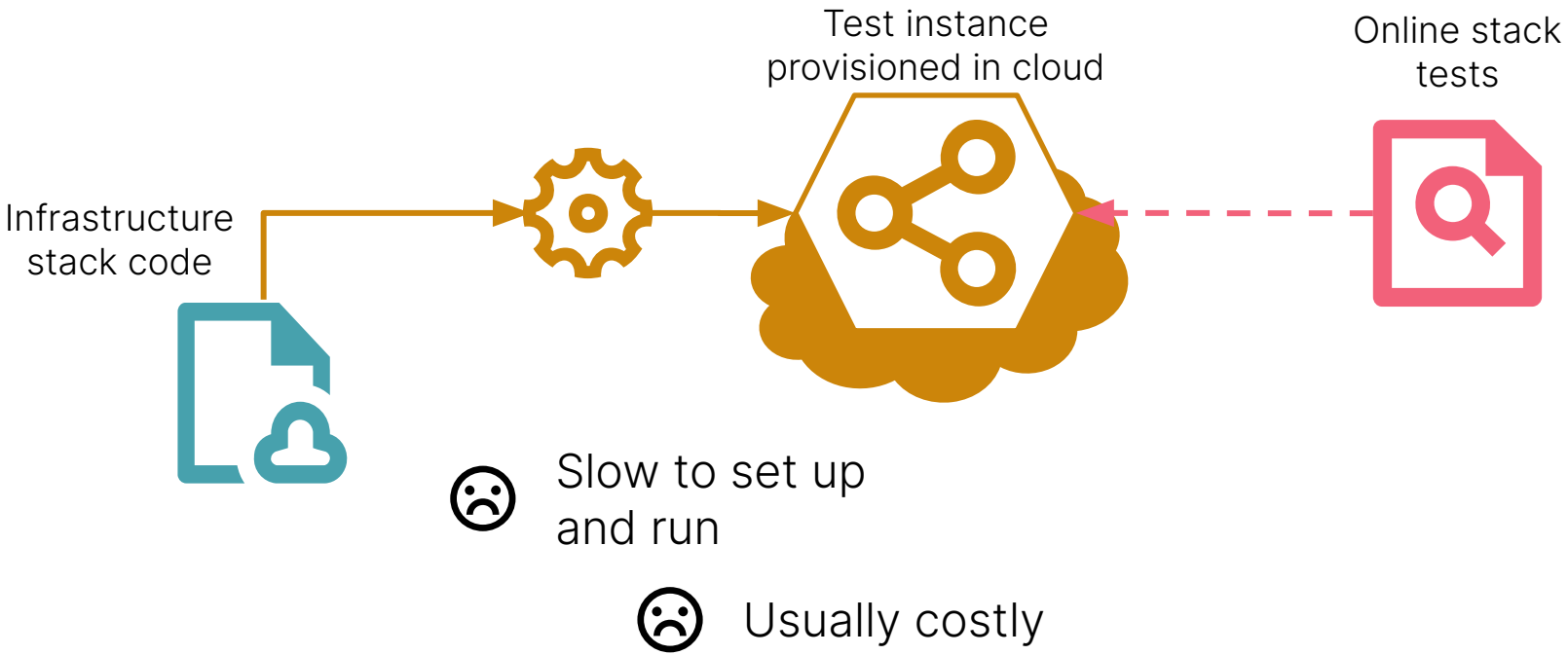
Offline testing

Can run both, as pre-commit hook and in the pipeline

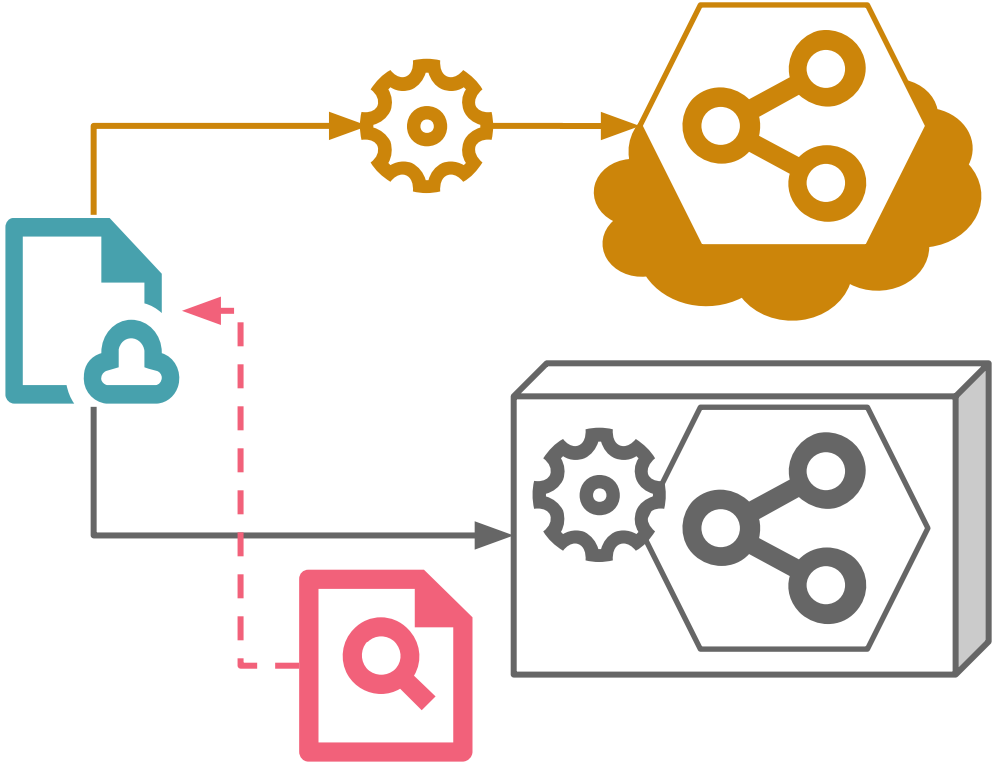
Can be fast (and cheap) to set up and run



Online testing



The best way to optimize feedback loops

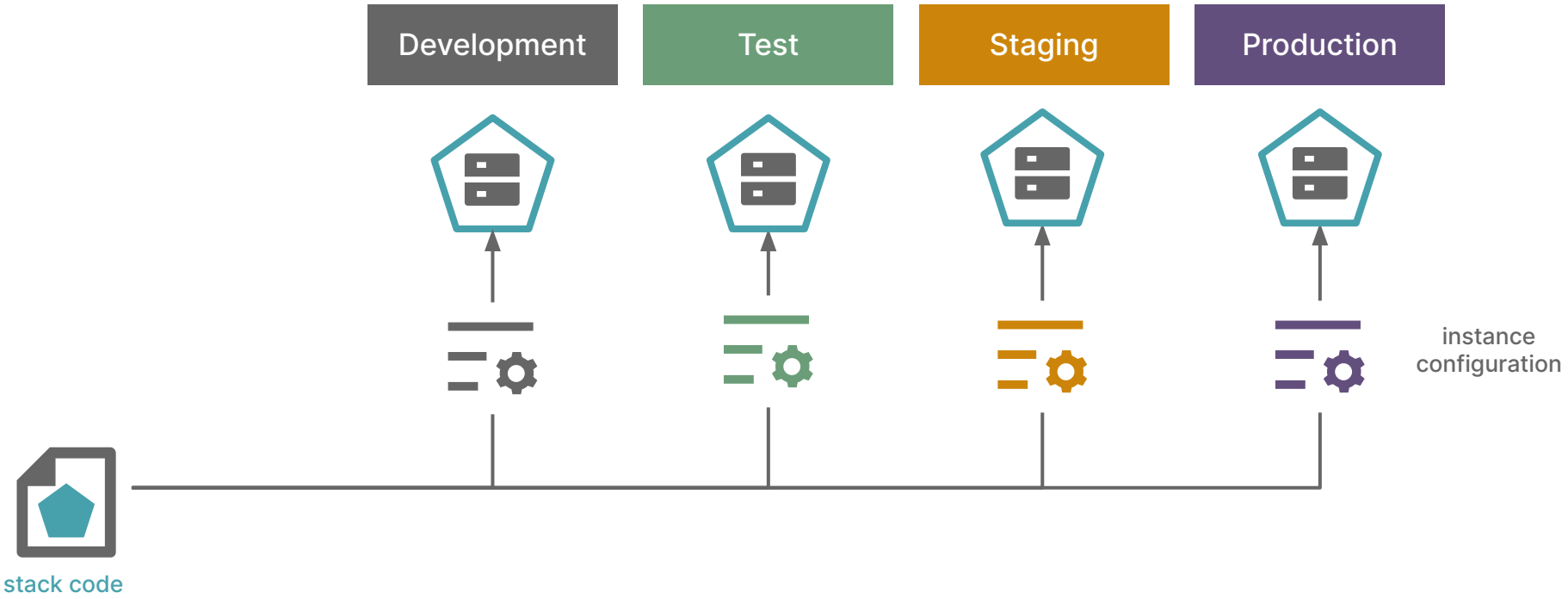


Smaller stacks are faster and easier to test (and fix!)

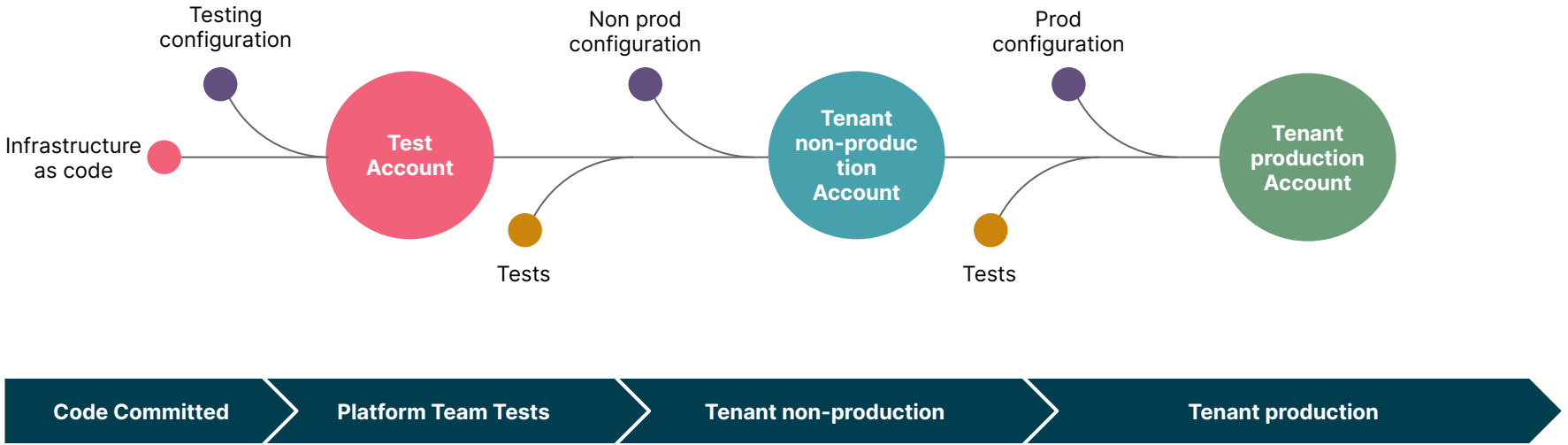
Designing your infrastructure delivery pipeline

/thoughtworks

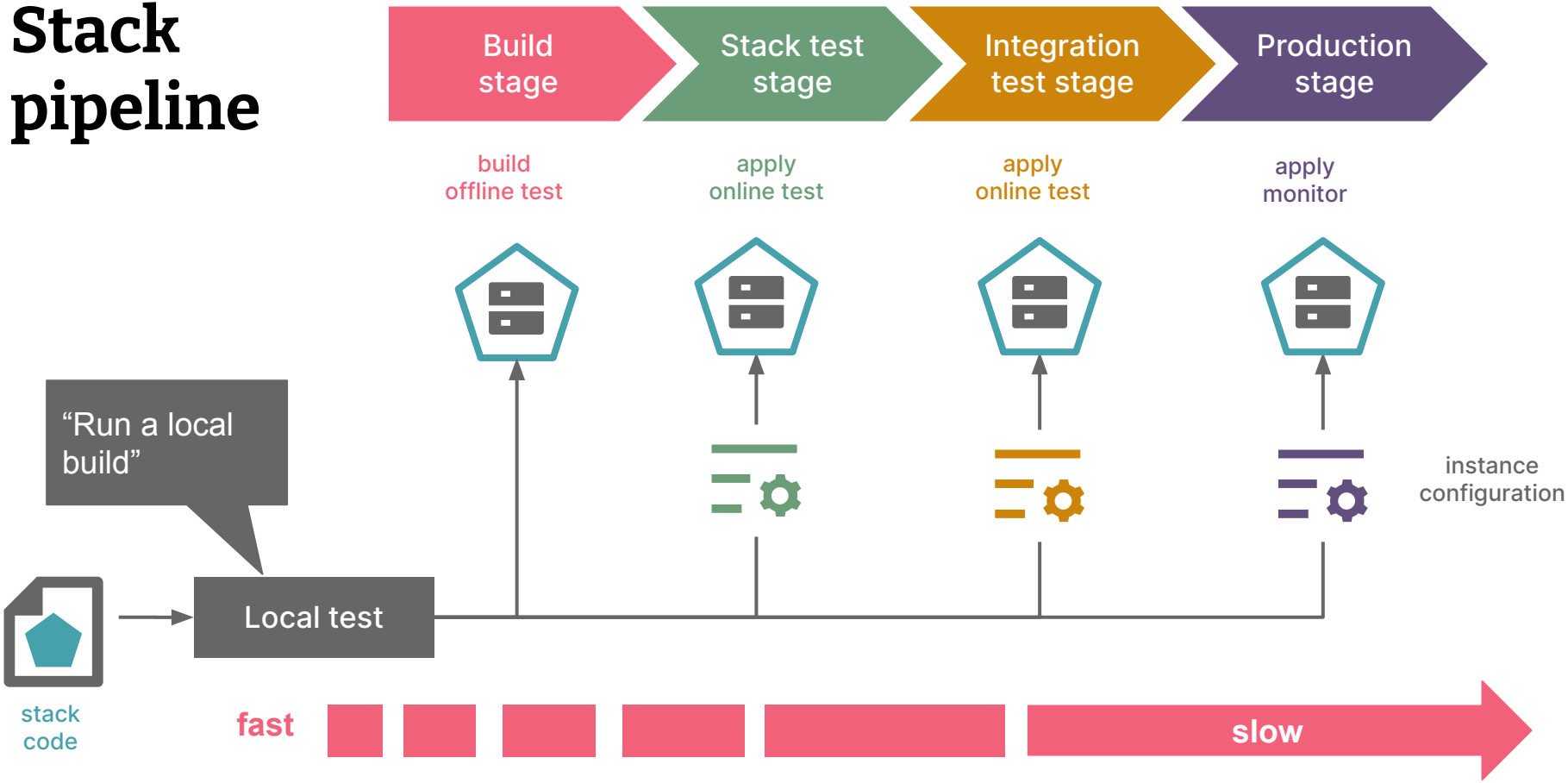
One stack - multiple deployments



Platform Path to Production



Stack pipeline



Local test

Keep your pipeline green

As a quality gate before we send our code off to the pipeline, we want to have a **pre-commit hook** that filters faulty commits. Make it more likely to **keep the pipeline green**.
In this stage we want to **validate** our infrastructure code.

aqua trivy

TFLint

Open Policy Agent

Download Dependencies

Check syntactical correctness

Run linters, formatters & security and compliance checks

Build stage

Validate & package your code

In this stage we want to **validate** our infrastructure code. The outcome of the stage is a **package** that contains all the artifacts we need for applying our infrastructure changes, e.g. validated infrastructure code.

aqua trivy

TFLint

ORAS

Open Policy Agent

Download Dependencies

Check syntactical correctness

Run linters, formatters & security and compliance checks

Create a promotable artifact

Stack test stage

Apply & validate the stack in isolation

In this stage we want to apply our Infrastructure code **run online tests**. This stage gives us the confidence that our code produces cloud resources that fulfill our requirements.

LocalStack

Terratest by Gruntwork.io

Open Policy Agent

Plan changes

Check output of the plan

Apply changes

Test applied cloud resources against expected behaviour

Integration test stage

Integrate multiple stacks

On this stage the stack is deployed into a pre-production staging **environment**. If you are integrating multiple stacks you can **validate end-to-end user journeys** here.

Terratest
by Gruntwork.io

Plan and apply changes

Validate user journey

Validate dynamically generated stacks

Promote release artefact

Production stage

Apply, test & promote your package

We **repeat the exact same steps** that we ran in the previous stage - but in our production environment - with the added safety of having run them in pre-production.

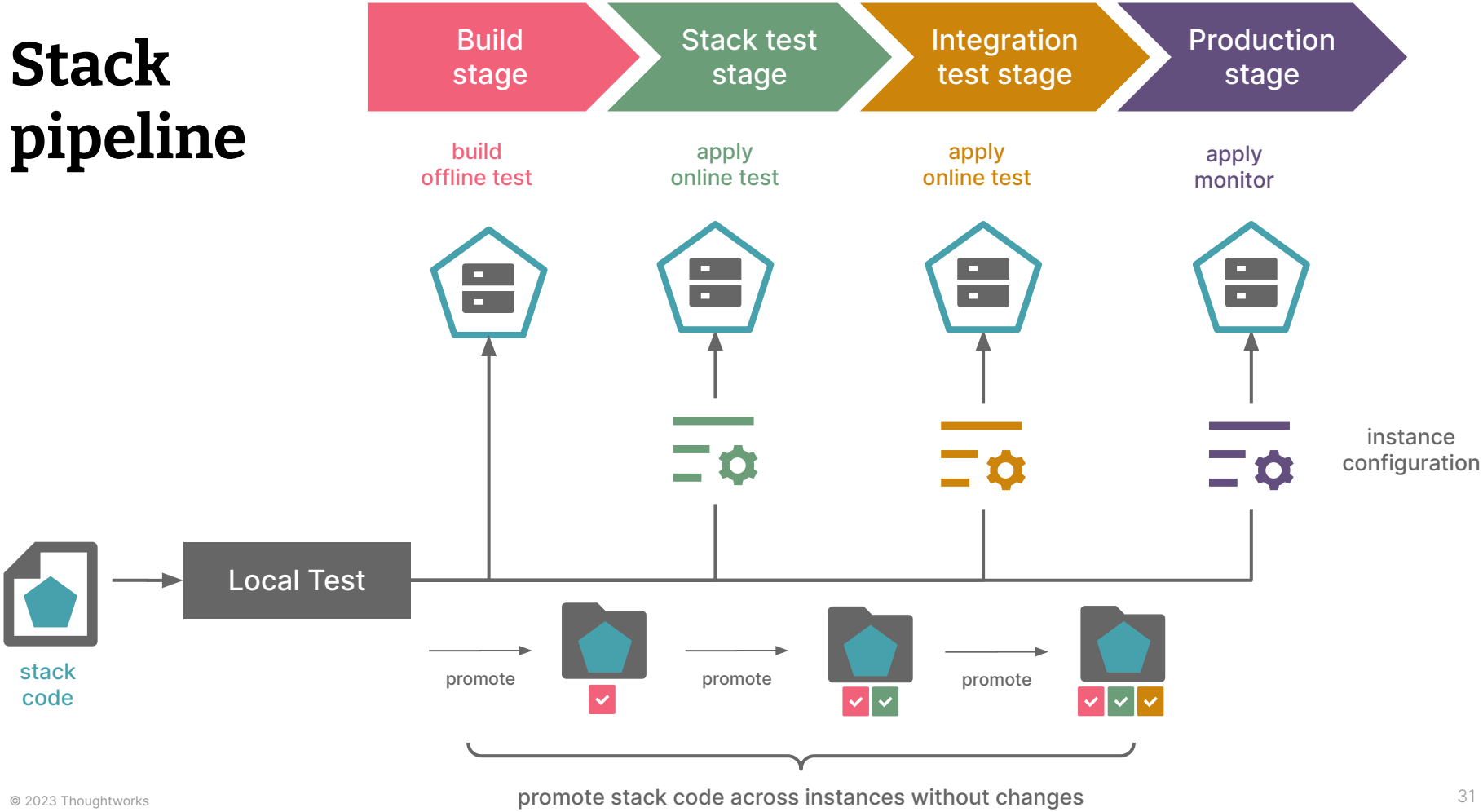
Plan changes

(potentially manual) Approval

Apply changes

Run (smoke) tests and synthetic monitoring

Stack pipeline

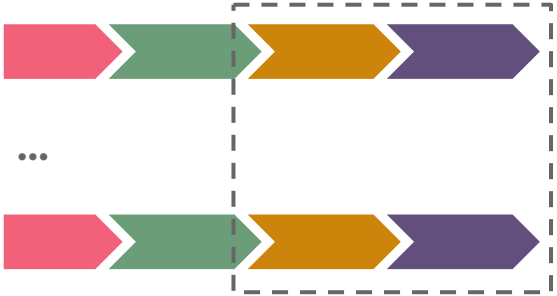


Pipeline topologies

How to handle multiple stacks creating an environment

Single stack

Multiple stacks



Wrapper stack

Challenges

thoughtworks

Hi Team,
In Terraform, we are facing more memory consumption issue while running the plan command,
it's fails the execution in between with below error.

The plugin.(*GRPCProvider).UpgradeResourceState request was cancelled.
[Container] Command did not exit successfully terraform plan -no-color -out=/tmp/changes exit status 1

In Code we have more than 55 provider blocks to communicate with client accounts, In Total its handling more than 2500 resources.

Blast Radius

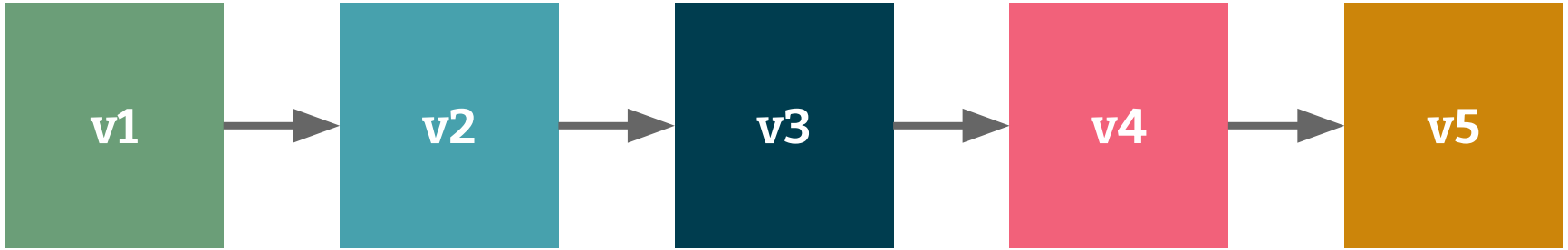
The term *blast radius* describes the potential damage a given change could make to a system. It's usually based on the elements of the system you're changing, what other elements depend on them, and what elements are shared.

Kief Morris, Infrastructure as Code 2nd Edition

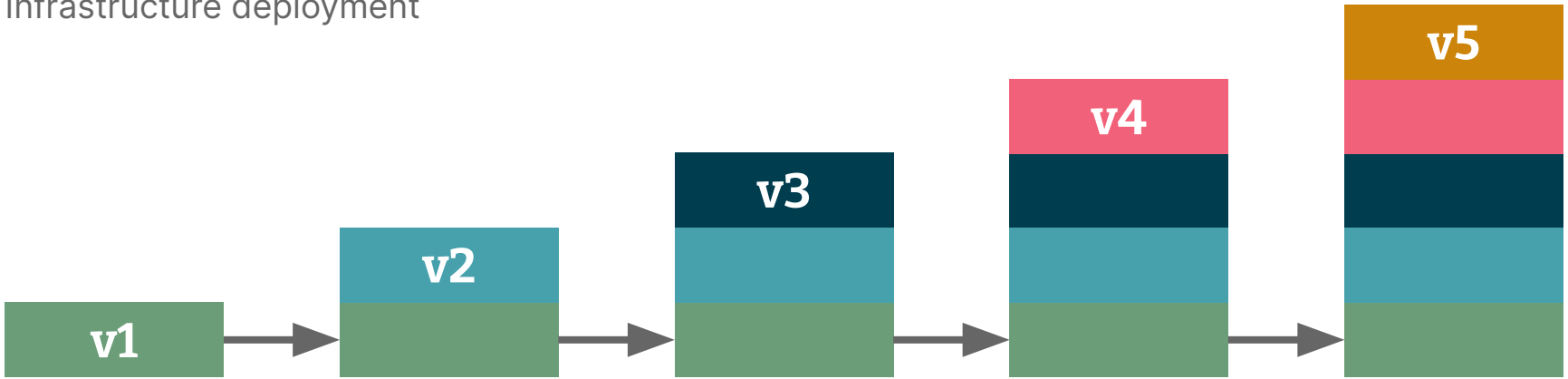
Image Source: commons.wikimedia.org

(Im)mutable deployment

(Modern) application deployment

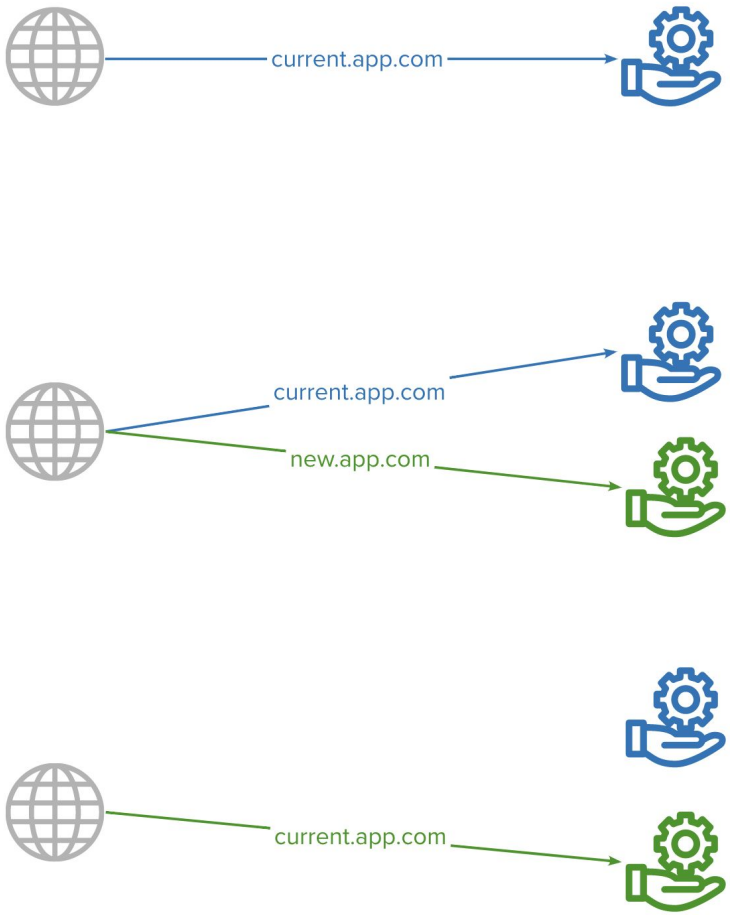


Infrastructure deployment



Roll-backs

With infrastructure code, there is **no easy roll-back** of changes. Having infrastructure as code allows for **re-creating every revision** of your setup - but it doesn't prevent you from **potentially losing state**.



Feedback cycles

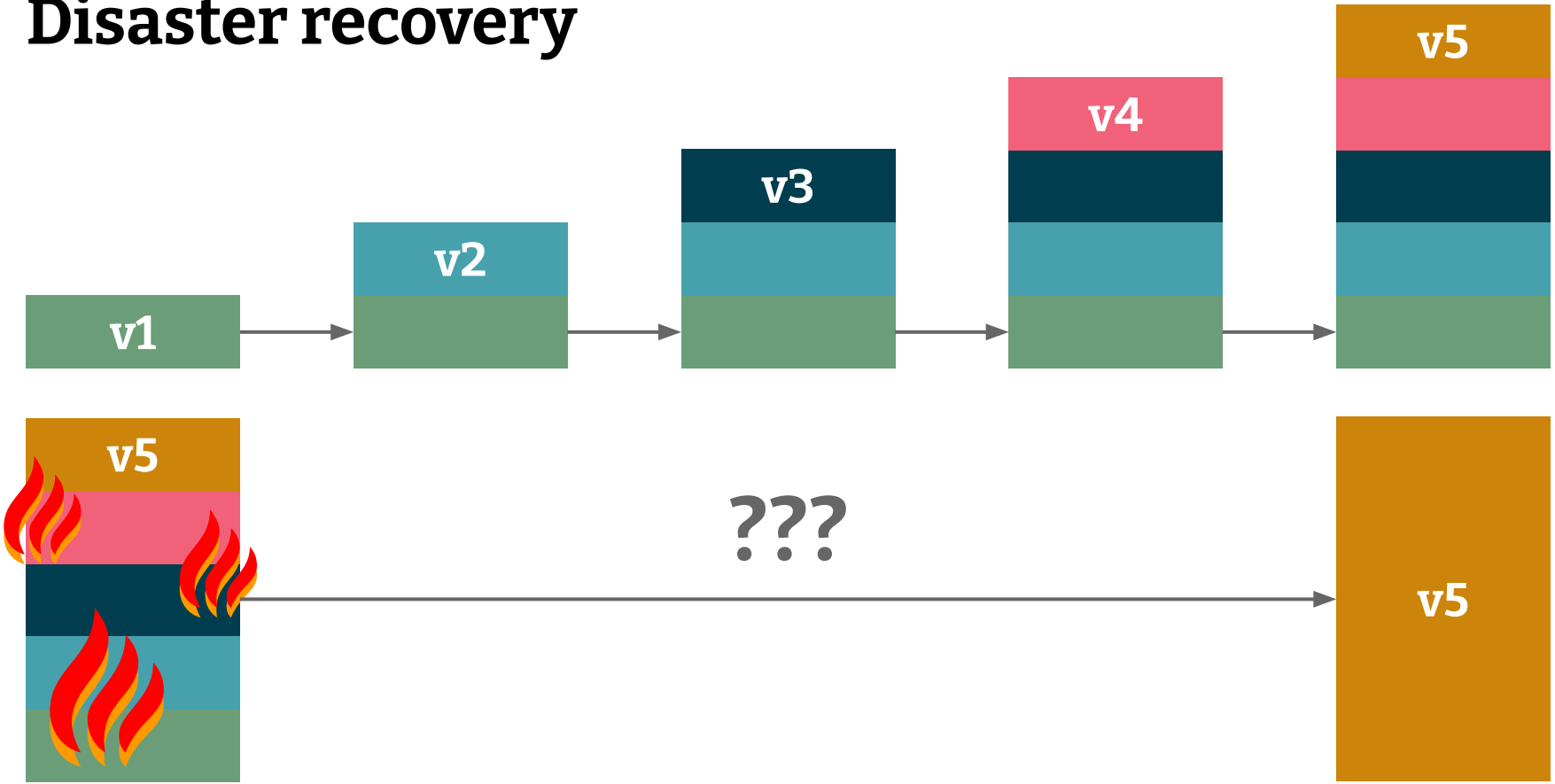
With infrastructure pipelines we usually face **long feedback loops**. This easily leads to developers working around using the pipelines and can bring you into trouble if you “quickly need to fix something in production”.

Stages	
✓ - ✓	17m 58s
✓ - ✓	18m 5s
✓ - ✓	18m 56s
✓ - ✓	18m 25s
✗	<1s
✓ - ✓	17m 28s
✓ - ✓	18m 18s

One more thing...

thoughtworks

Disaster recovery

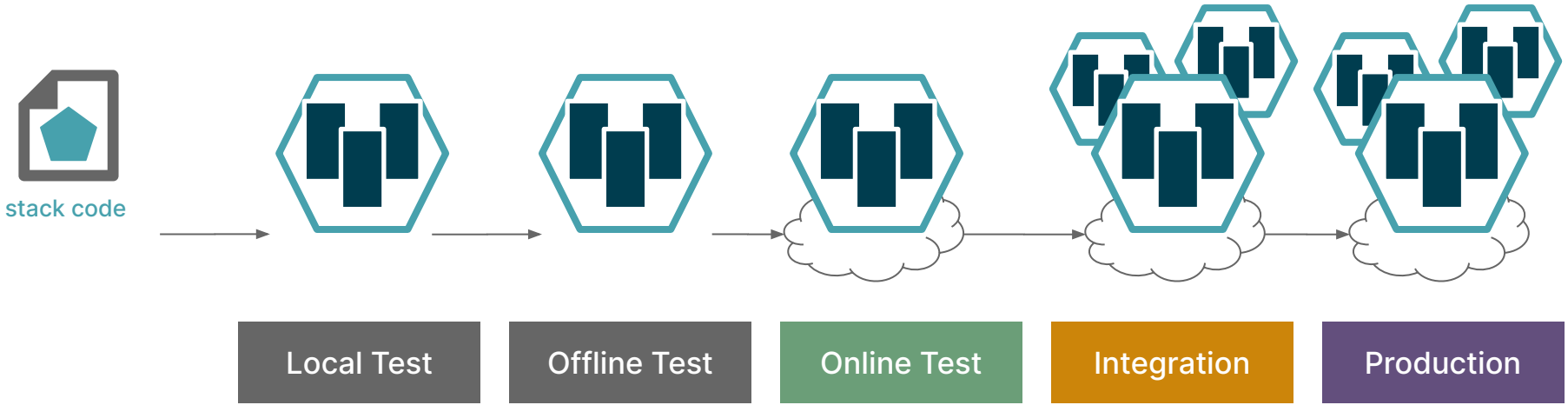


Summary

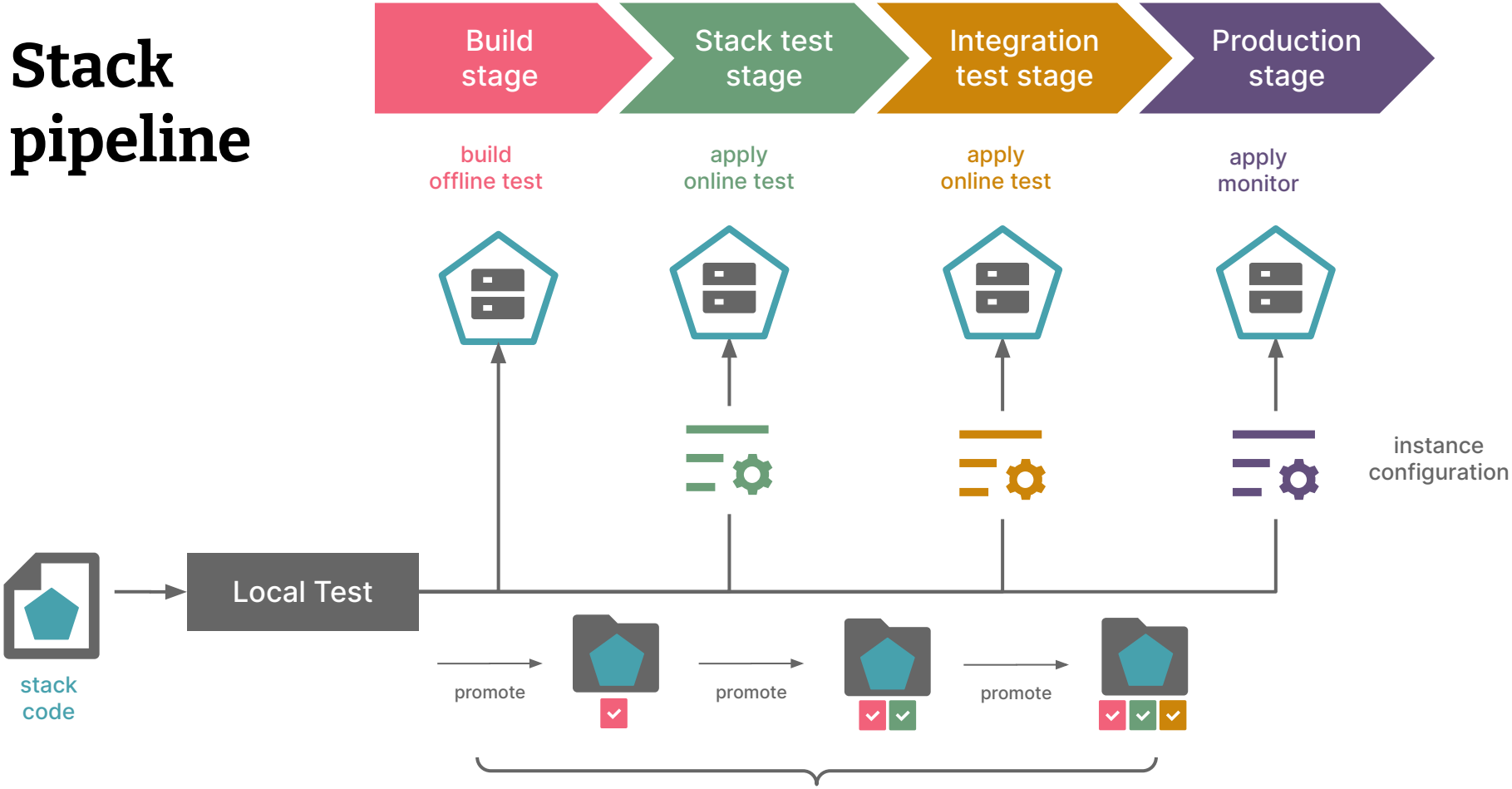
thoughtworks

Stack testing

Smaller stacks are faster and easier to test (and fix!)



Stack pipeline



Thank you for your attention 👍

I'll be around if you have any questions!

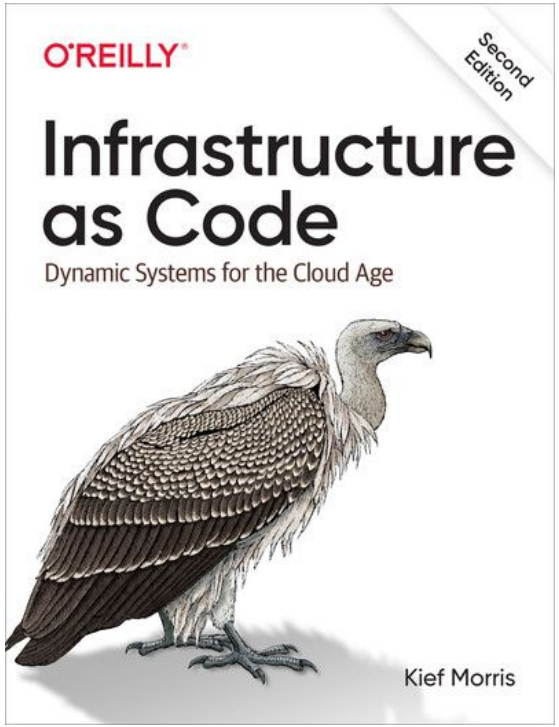
Michael Lihs
Infrastructure Consultant
michael.lihs@thoughtworks.com

@kaktusmimi

linkedin.com/in/michael-lihs/

/thoughtworks

References



Kief Morris, Infrastructure as Code - 2nd Edition

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