# How to grow and handle distributed systems

Max Körbächer - Co-Founder @ Liquid Reply

# Max Körbächer - Co-Founder @ Liquid Reply

My work is all about
**Kubernetes Consultancy & Cloud Native Advisory**

- Former Enterprise Architect, yet design and build hyper converged infrastructures and cloud agnostic solutions
- CNCF TAG Environmental Sustainability Co-Chair, Contributed 3y to the Kubernetes release team and being part of the Advisory Board for keptn & DevNetwork

**maxkoerbaecher**

**mkoerbi**

REPLY
LIQUID

# MEET THE COMMUNITY

**Kubernetes Meetup Munich**

>2.400 Group Members, looking for interested parties to get the group started again


**Kubernetes Community Days Munich**

First time KCD in Munich, 200 participants, 2 days, 2 tracks & BOWLING!

**CNCF TAG Environmental Sustainability**

Technical Advisory Group that brings the green thumb to the cloud native community

# Our 3 course menu for today

## Part 1

Gall's Law & distributed systems

## Part 2

5 steps how you will successfully fail

## Part 3

How to grow your systems without failing on Gall

# Part 1

What Gall says!

**A complex system that works is invariably found to have evolved from a simple system that worked.**

- John Gall

**Baremetal**

Monolithic Software

Installed “by hand” or scripts

Works

**VMs**

Monolithic Software

Installed “by hand” or scripts

Works

**Container**

Monolithic or Micro Service

Installed by scripts or tools

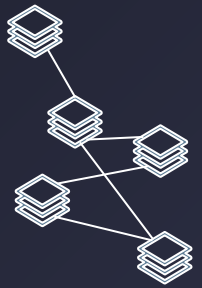
Works

**Container Orchestration**

Monolithic or Micro Service

Installed by scripts or tools

Works



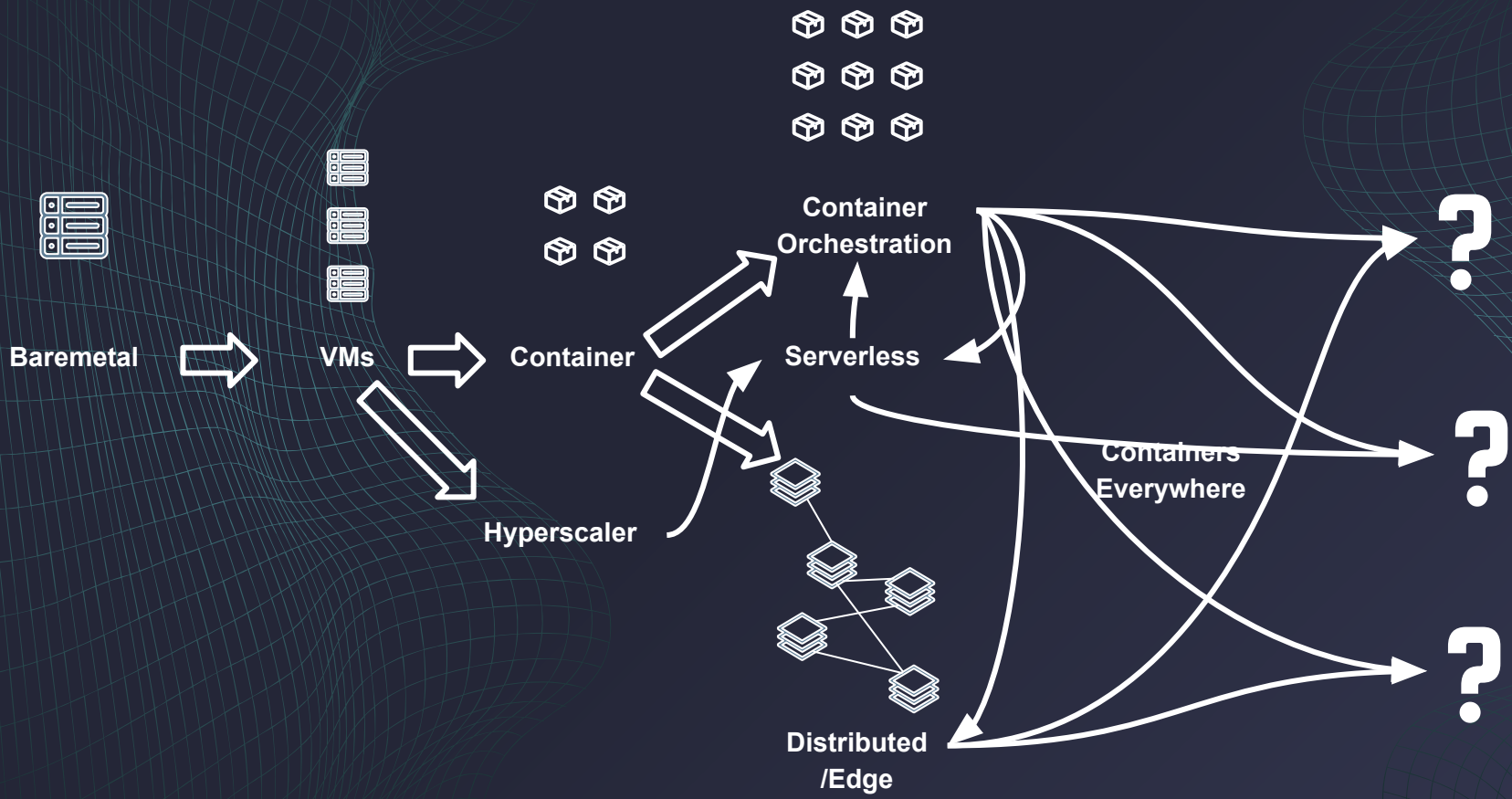
**Distributed/ Edge**

Micro Services or Functions

Installed by tools

Works

?


Baremetal
VMs
Container
Hyperscaler
Container Orchestration
Serverless
Distributed /Edge
Containers Everywhere
?
?
?

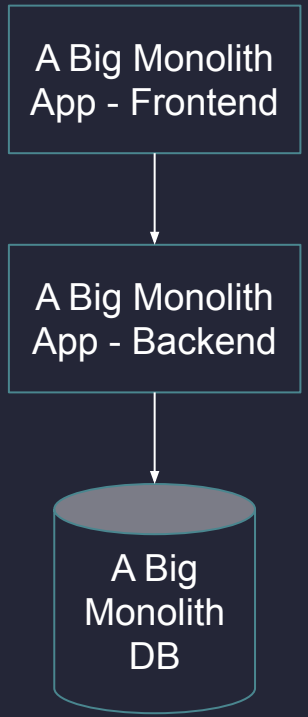
**A complex system that works is invariably found to have evolved from a simple system that worked.**

- John Gall

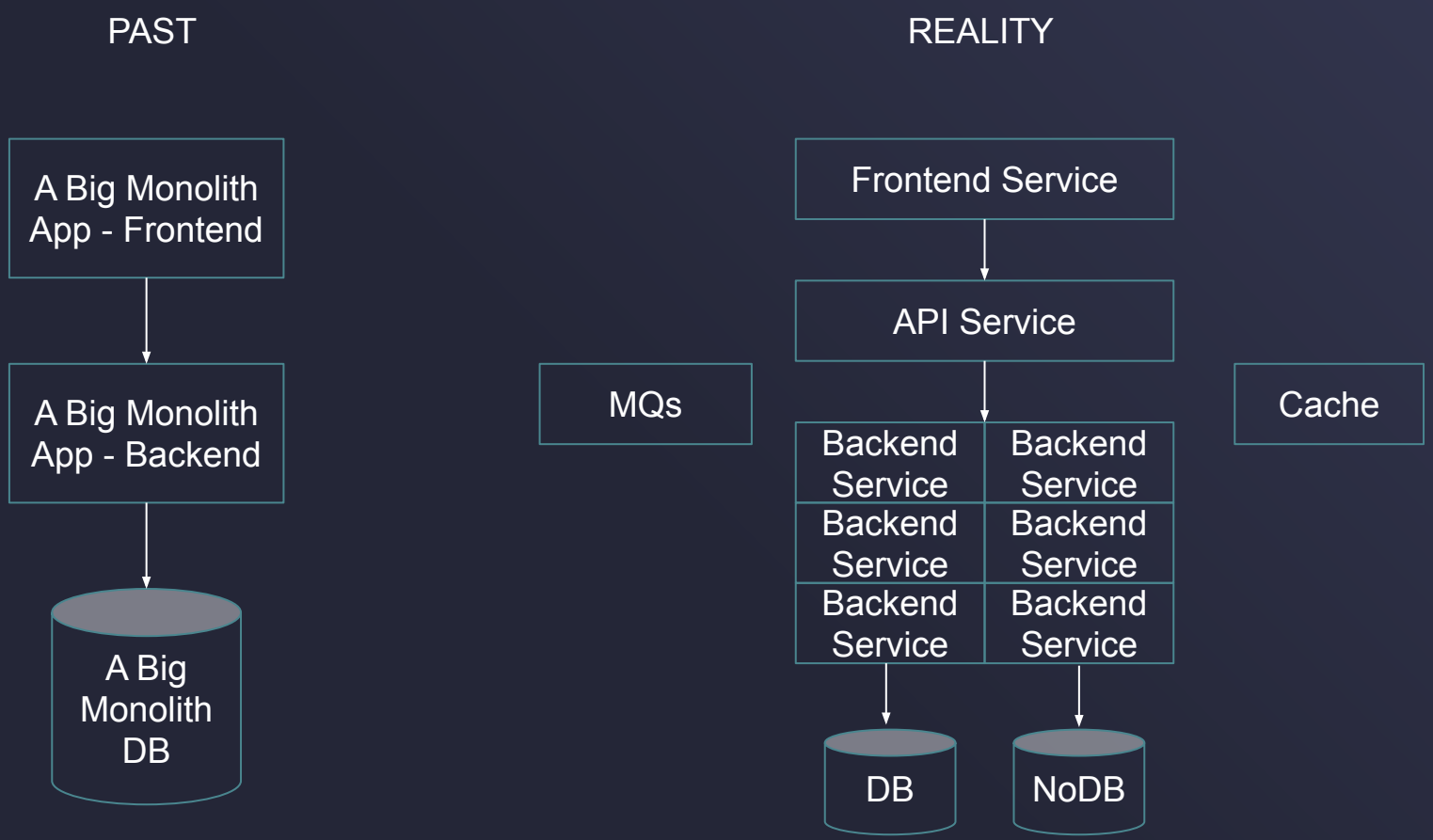
**A simple system may work or not work.**

- John Gall

PAST


A Big Monolith App - Frontend
A Big Monolith App - Backend
A Big Monolith DB

FUTURE


PAST
A Big Monolith App - Frontend
A Big Monolith App - Backend
A Big Monolith DB
REALITY
Frontend Service
API Service
MQs
Cache
Backend Service
Backend Service
Backend Service
Backend Service
Backend Service
Backend Service
DB
NoDB

**A complex system designed from scratch never works and cannot be made to work.**

- John Gall

# Distributed cloud native systems come with their very own specific way of doing things.

Following this approaches all in grants you high flexibility, transparency, scalability and reliability.

# Part 2

Do this to fail!

**Disclaimer!**
**There is some sarcasm in the room**

# 5 Steps for a bloody nose

1. Skip the basics
2. Build everything by yourself
3. Manage your network as it is on prem
4. We know best what we (internal customer) need
5. Trust your security department

# Skip the basics

A 1 day training is enough

Get your team a 10$ Udemy course for 8h and you will have well skilled professional engineers.

OR

Implement a culture, time and budget for continuous learning and experimenting.

# Don’t do it yourself

Enterprises and companies are obsessed with do it yourself. (Maybe a PTBS of decades of utilizing commercial tools)

Build your own tools, processes and integrations is a never ending story.

1. Use IaC for the basic ressource definition
2. A cluster deployer for the K8s and its configuration
3. A GitOps tool for your application management


Photo by Kevin Kandlbinder on Unsplash

# Networking is by far the most over-engineered part

- You don’t need to go through five firewalls if they do the same thing
- Overcomplicated networks prevent the usage of cloud resources from an application development perspective
- Avoid hundreds of accounts, multi-cloud and to many hub & spoke architectures

Utilize simple setups and extend them through in cluster tools like cert-manager, external dns, native integrations to LBs & Gateways

# “Platforms” are not build for themself

Support various entry points or even manage the application delivery path!

A platform is a target for the dev/product teams

Apply best practices on:

- container
- deployment configs
- security
- integrity
- integration between cluster wide solutions and the apps

But with your internal customers in mind!

# Our Security Team knows what to do

We don’t have to be the security experts, others are

“We will follow the recommendations of the Security team”

“If we have firewalls and access control it is enough”

“Our SOC uses the same 10 queries we used before”

“It will be pentested by our team in the end”

“Security isn’t right now our priority, we will fix that later”

# Security makes fun!

- ➔ Cloud Security and best practices are given on hand
- ➔ Security on Kubernetes is 80% done by a good RBAC, Network Policies and a Policy Engine
- ➔ Security beneath Kubernetes is more difficult - the hardening must be done right otherwise the capabilities of the cloud and K8s are inhibited
- ➔ Application and Network Security can be shifted towards your platforms - for apps counts = “Just don’t do stupid things”

# Part 3

How to grow distributed systems without failing Gall’s Law

**No one wakes up like:**

I want to have now a hyperconverged distributed infrastructure that spreads across everything and is technically brilliant, so that no one understand it anymore.

# Follow Gall’s Law - Start Simple

# Define the Baseline

**What are your basic requirements?**

Internal or public apps, high throughput, long time storage, over the day requests

**Which quality criterias are relevant?**

Scalability, Security, Cost, Reliability, Reusability

**Which levels are handled by you?**

Cloud Account, Infrastructure, Network, Application Deployment, Security, Monitoring

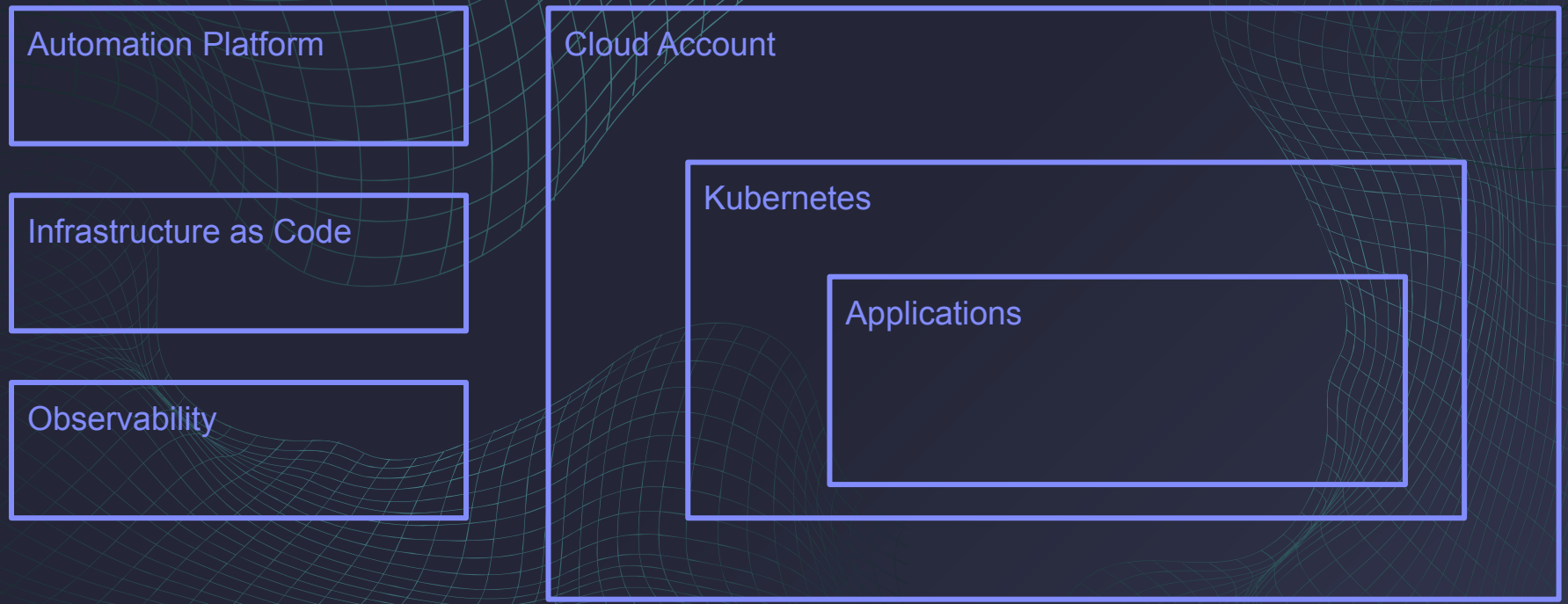
## Design a target that is non blocking

Speaking of architectures, this means to have evolution in mind

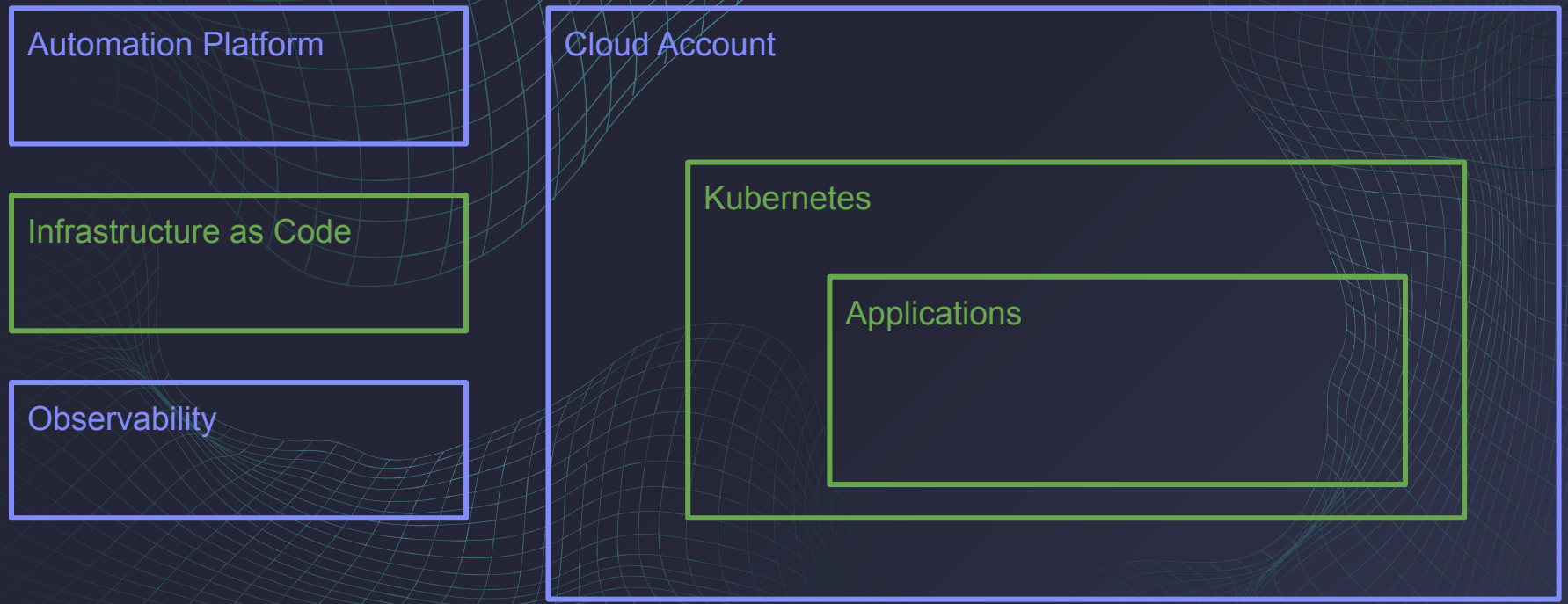
-

An Evolutionary Architecture

# Every domain has simple and complex solutions



# Most of those domains can be independent



# Infrastructure as Code

With distributed platforms we often see only the extreme on the IaC side. Either everything in a single repository managed by self written pipelines or each product gets its own clone of a blueprint and has to self manage those.

Blue prints to get started
BUT don't distribute them

Good for your team and if you manage that for 2-3 projects

Get them into your automation, supporting some variables

Good for your team and if you manage that for 2-3 projects

Get yourself a IaC platform and not only a Git repository

Almost no one goes here, but suffers from the middle step and its overhead

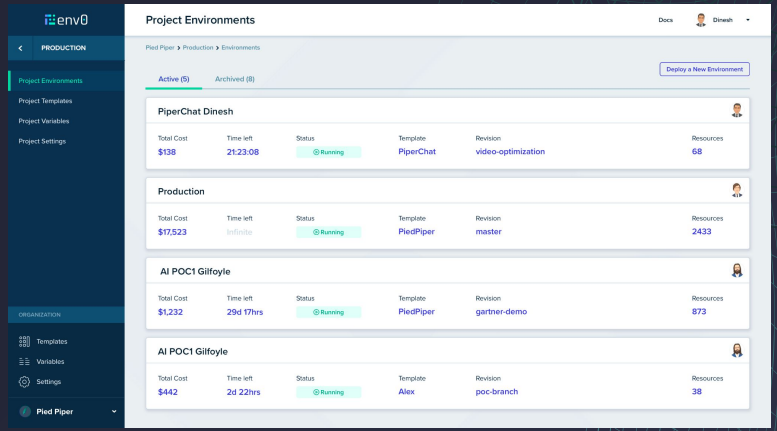
# Infrastructure as Code

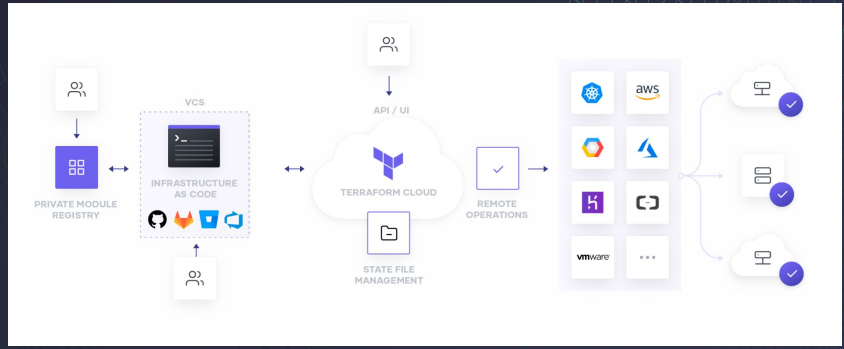
A grown IaC environment can’t be handled centrally from a small team nor it can be distributed to all your teams.

SORRY - NO SALES HERE

But

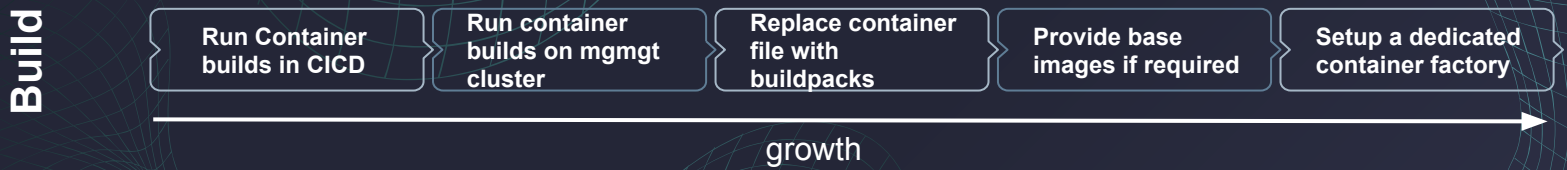
- Team costs
- Technical debts
- Outdated infrastructure
- No central compliance enforcement
- …



# Application Management

Avoid the burden of building enterprise wide the containers for everyone.



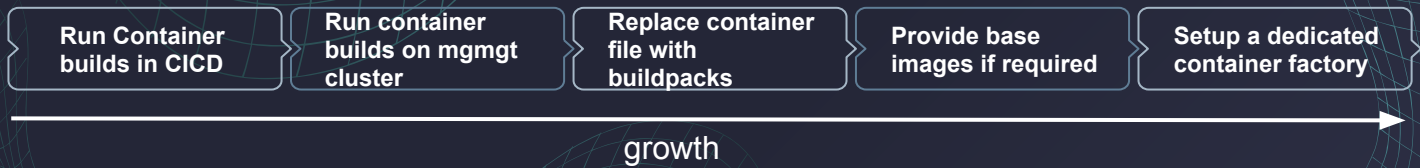

Successful projects offer an option on an opinionated build, but don’t force it.

What they provide is an entry point where the container needs to be delivered at.

# Application Management

Avoid the burden of building enterprise wide the containers for everyone.

Build




**Successful platforms provide an entry point where the container needs to be delivered at.**

# Application Management

Avoid to couple the application deployment with your IaC. Both have different life cycles.

Deploy

## Stage 1

Continuous Deployment: Running bare charts or helm charts in a CD pipeline

Works for everything
Tend to be to complex

## Stage 2

CD: GitOps - still having charts, but managed by a dedicated solution

Yet another solution, but scales distributed and centralized

## Stage 3

Enrich: Progressive Delivery and SRE methodologies

More tools! But now they provide you feedback and insights

# Kubernetes - Provisioning

Part of your infrastructure, but also not. Kubernetes is more than a provisioning machine for mini VMs.

Yet, getting K8s up and running is the easy part.

## Starting Point

Use whatever makes you progressing:

- Pre-existing IaC
- OSS deployment solutions as CAPI
- Opinionated OSS solutions as Rancher

Any of those tools are build cloud native and allow you to grow with you.

## Scaled Environments

I wouldn't go anymore with IaC for scaled environments.

Utilize CAPI, Crossplane or more commercial solutions like Rancher or Spectro Cloud.

Those help you also to get everything on top of Kubernetes done right.

# Multicluster vs Single

There is no right and wrong

Whether you do one or many clusters, important is you are good with:

- automation of provisioning
- less scripts more declarative
- ease of use
- build on usefulness not on fanciness

A management cluster is almost always a must have.

Photo by Alora Griffiths on Unsplash

# Growing systems that work

Mean also adding other systems that work

- The bigger the solutions gets, the more other systems we have to add
- Those are there to help you not to burden to
- Evolution means also the adoption of new circumstances, and they change nowadays within 1-2y frequency

**The key to success: have a dedicated Platform Engineering team, not just a project to introduce a solution.**

The simplest solutions is usually the best one.

# Have a great day!

maxkoerbaecher

mkoerbi