



CI/CD Pipelines for Microservices

Best Practices

DAN GARFIELD

Dan Garfield

Chief Technology Evangelist

 **codefresh**

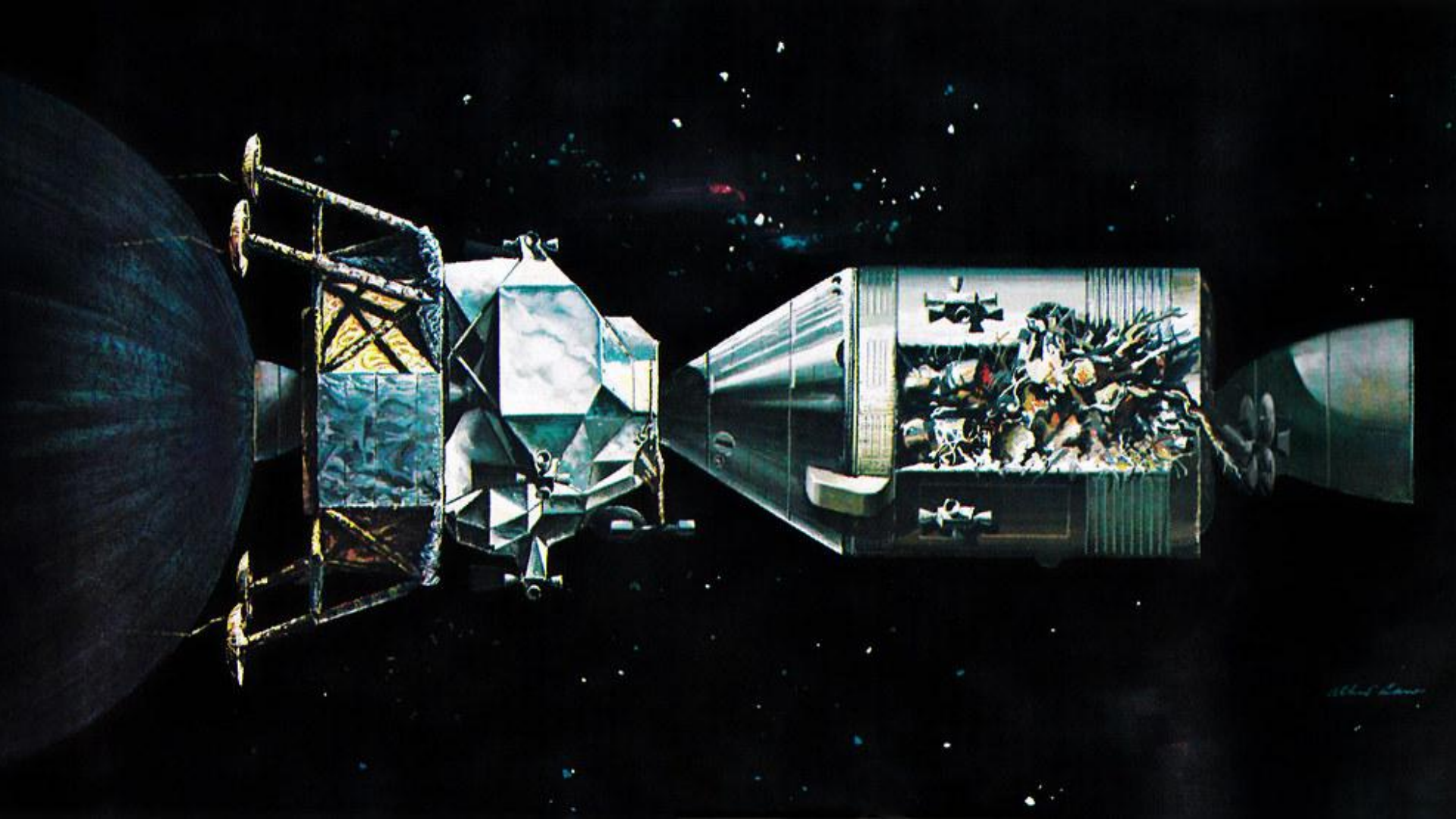
@todaywasawesome

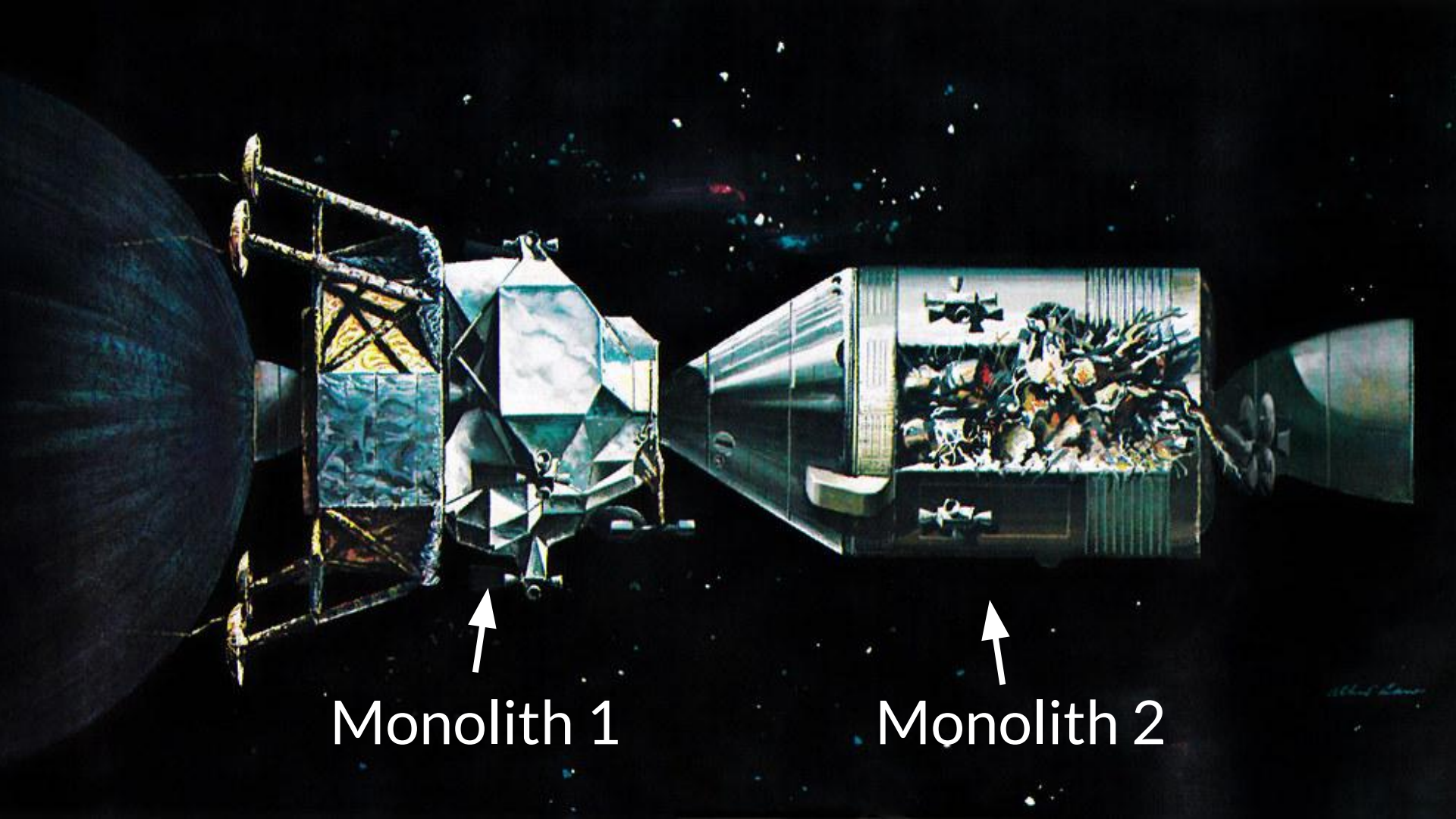


Agenda

1. Why microservices
2. How Expedia approached microservice CI/CD
3. How we do it at Codefresh

Why Microservices?





Monolith 1

Monolith 2

WE NEED TO
MAKE THIS...



...FIT INTO A
HOLE MADE
FOR THIS...



...USING ONLY THIS!



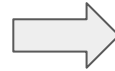
Expedia corporate travel re-architecture

Multiple monoliths with diff. UI&UX

Cars (ECT)

Cars
(Egencia)

Cars
(Travelforce)



Multiple services with single UI&UX

Unified cars UI

Phone apps.

Car search

Car sort

Booking service

Moving to Microservices at Expedia

Approach

- Consolidate code bases and
- Build shared libraries for global platform. Ex:
 - Logging service, monitoring service
- Rely on manual integration testing
- Standardize CI/CD pipelines
- Use Maven for modularity
- Migrate to cloud from on-prem

Issues faced

- Geographically distributed
- Tools consolidation was hard
- Too many pipelines as microservices grew.
(100 pipelines → 1000+)
- Pipelines not modular or re-usable
- Jenkins master-slave issues
- Copy-pasta causing bad patterns
- Central team could not keep up
- Plugin upgrade was a nightmare

Moving to Microservices at Expedia

Lessons learned and Recommendations

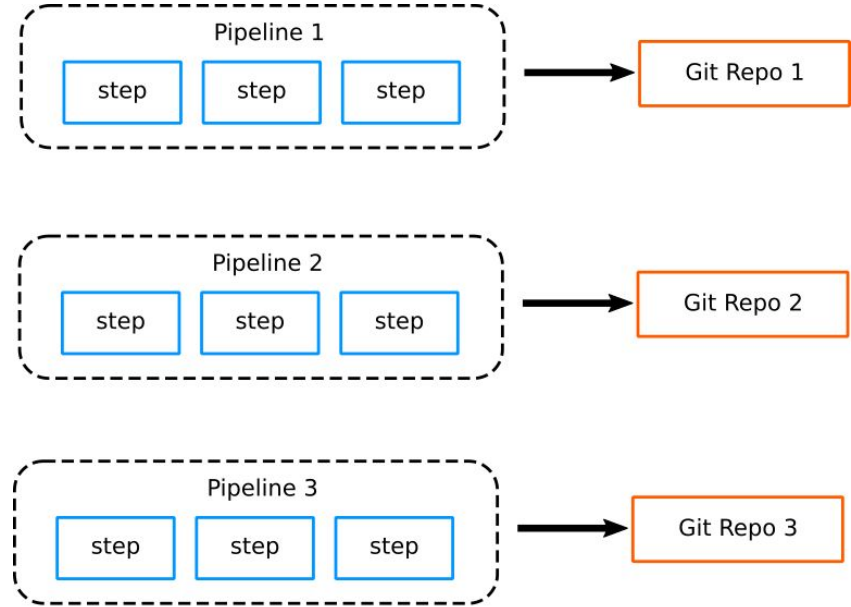
- CI/CD templates should have been prioritized higher than “business needs”
- Bootstrapping new projects should have been externalized from the microservice and adding a new microservice should have full pipeline setup once a repo is created
- A modular pipeline approach would ease the pain caused by different versions
- Reusability in CI/CD platform is critical

Organizing pipelines for monolithic applications

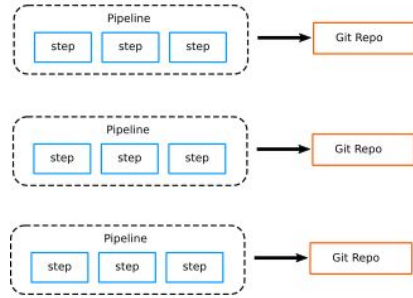
Single pipeline per project

Can be complex/difficult to be maintain

Usually led by a single team (anti-devops)

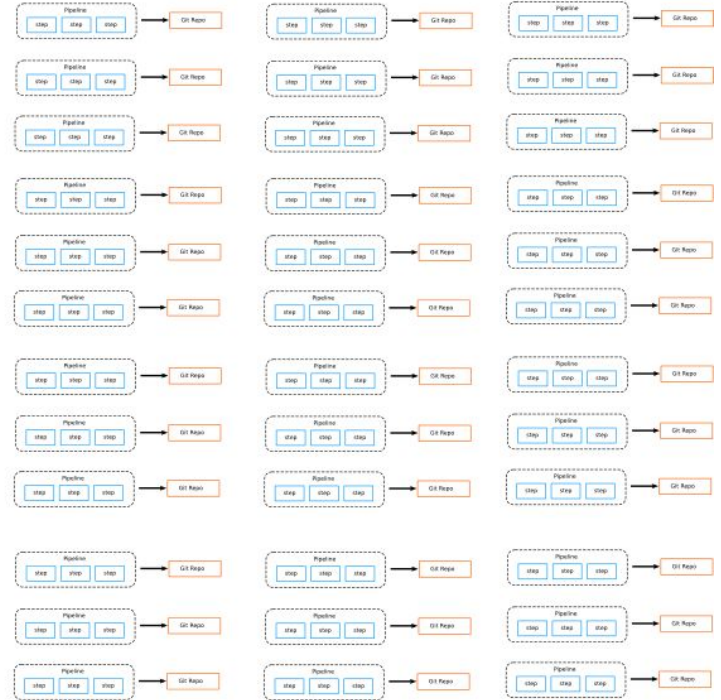


Scalability issues with microservice pipelines



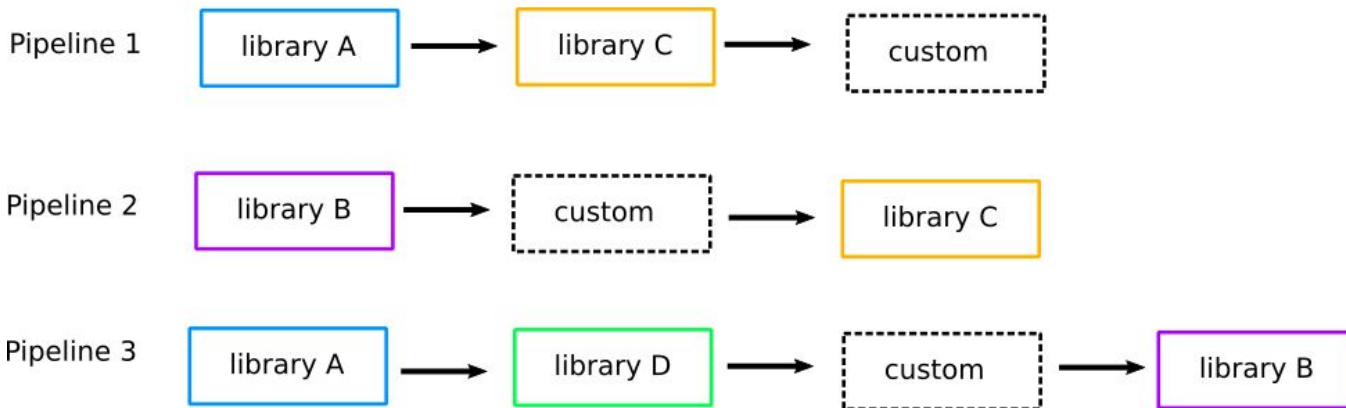
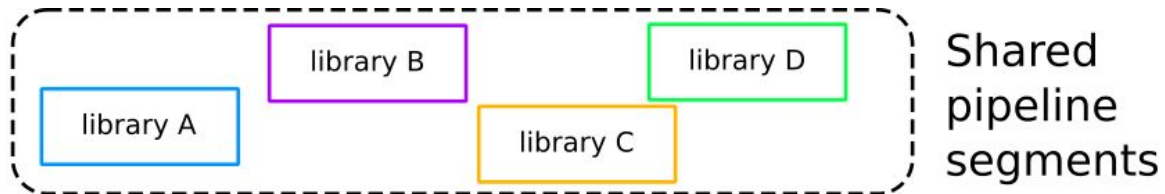
3 monolithic applications

Does this look like a plan??



Each application split to 4 microservices

Shared libraries are not the solution.



Organizing pipelines for monolithic applications

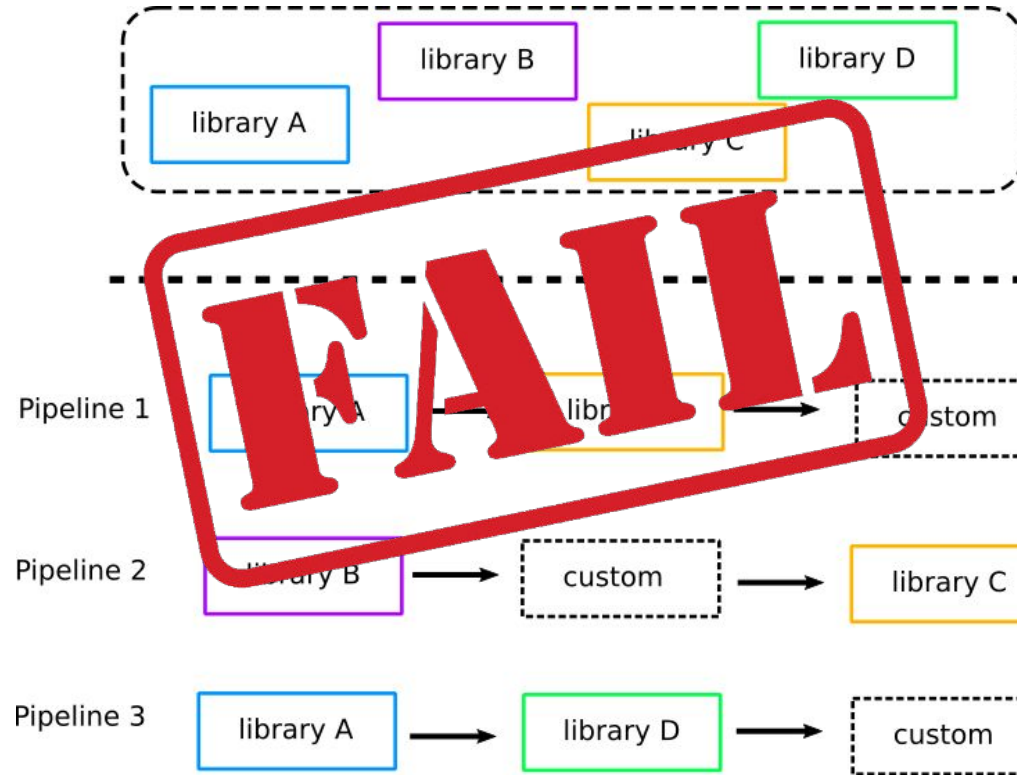
Requires everyone to use same version of library

Libraries often rely on each other in complex ways

Changes have to go to admins

Leads to big stability problems

Relies on proprietary API

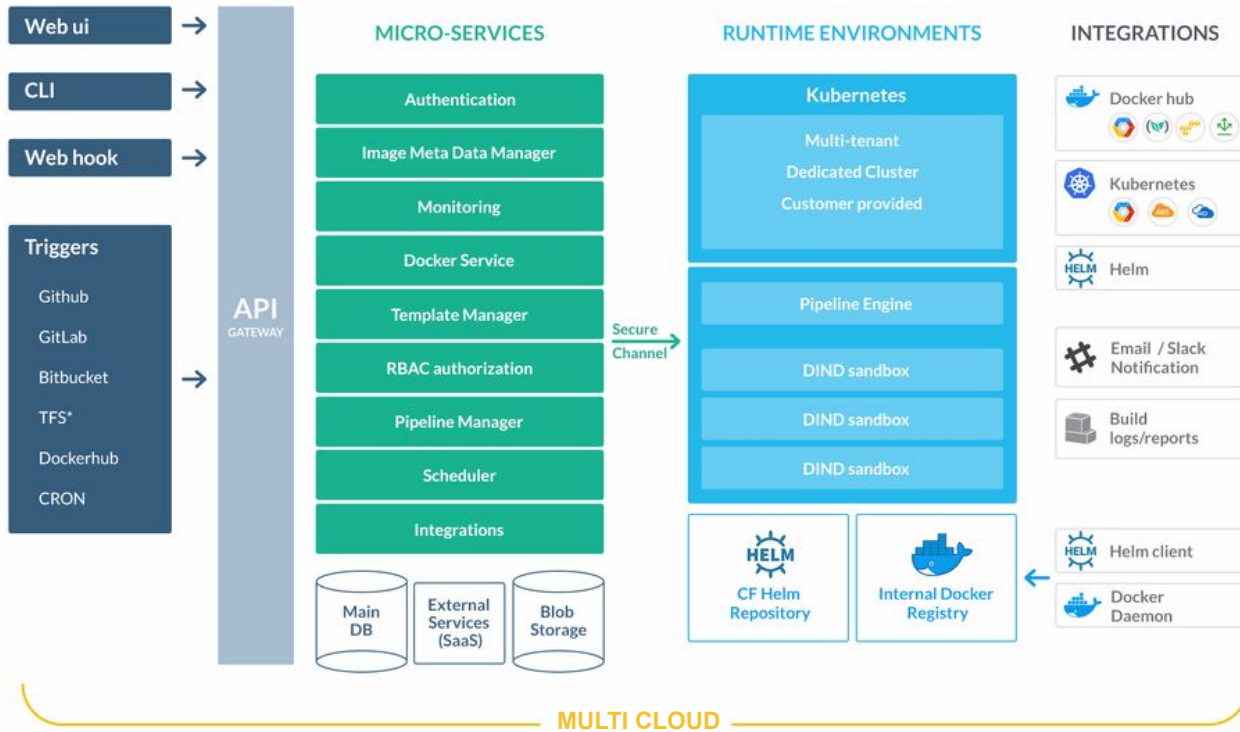


How Codefresh does CI/CD for Microservices

1. Container-based pipelines
2. Shared pipelines
3. Deployment testing



CODEFRESH ARCHITECTURE DIAGRAM



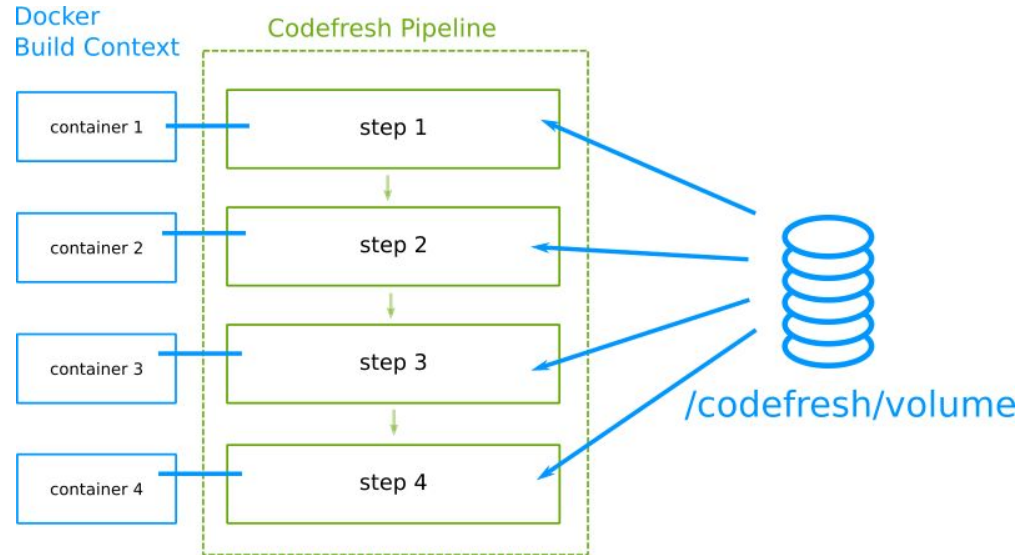
Container-based pipelines

Each task is built into a Docker image.

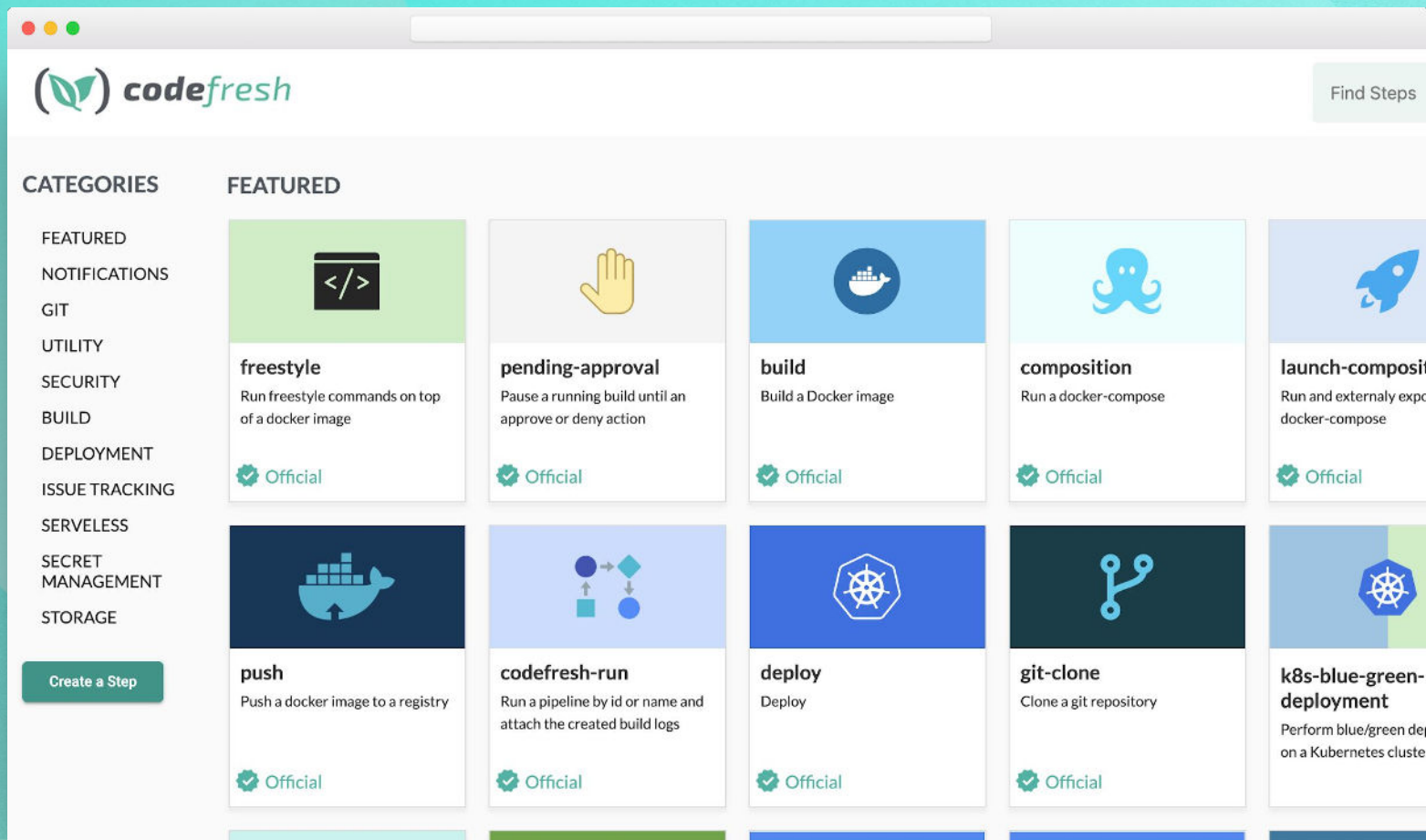
Users can self-serve these images.

Images DO NOT rely on each other.

Containers can be anything: go/node/c++



Huge open source library at steps.codefresh.io



The screenshot shows the Codefresh Steps library interface. At the top left is the Codefresh logo, and at the top right is a search bar labeled "Find Steps". On the left side, there is a vertical navigation menu with categories: FEATURED, NOTIFICATIONS, GIT, UTILITY, SECURITY, BUILD, DEPLOYMENT, ISSUE TRACKING, SERVELESS, SECRET MANAGEMENT, and STORAGE. Below the menu is a "Create a Step" button. The main content area is titled "FEATURED" and displays a grid of step cards. Each card includes an icon, a title, a description, and an "Official" badge.

codefresh











Find Steps

CATEGORIES

- FEATURED
- NOTIFICATIONS
- GIT
- UTILITY
- SECURITY
- BUILD
- DEPLOYMENT
- ISSUE TRACKING
- SERVELESS
- SECRET MANAGEMENT
- STORAGE

Create a Step

FEATURED

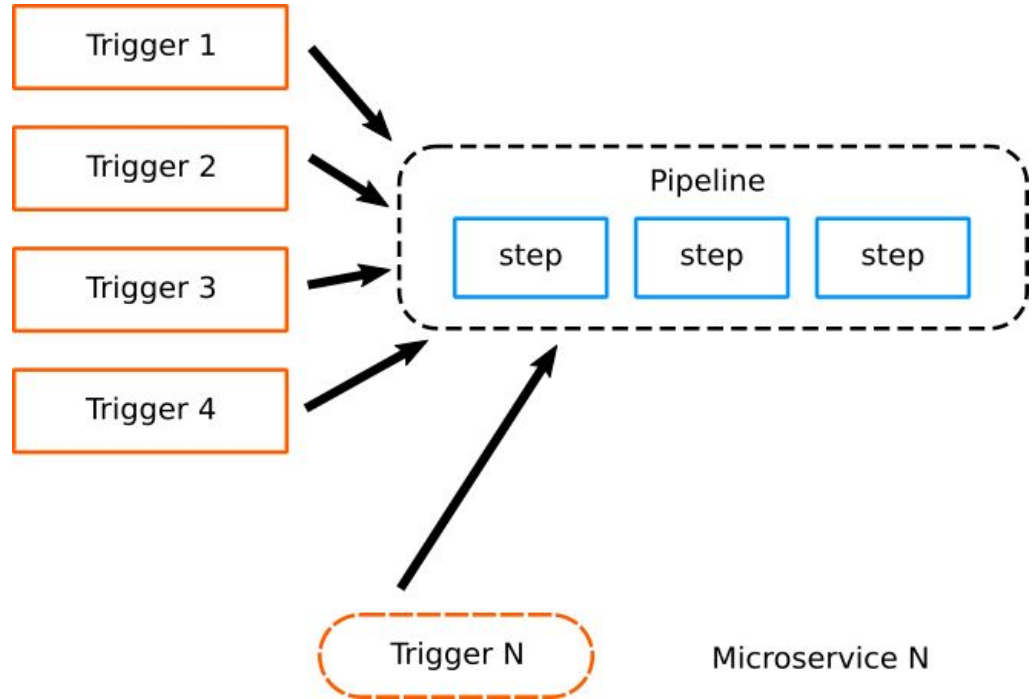
 freestyle Run freestyle commands on top of a docker image Official	 pending-approval Pause a running build until an approve or deny action Official	 build Build a Docker image Official	 composition Run a docker-compose Official	 launch-composit Run and externally expo docker-compose Official
 push Push a docker image to a registry Official	 codefresh-run Run a pipeline by id or name and attach the created build logs Official	 deploy Deploy Official	 git-clone Clone a git repository Official	 k8s-blue-green-deployment Perform blue/green dep on a Kubernetes cluster Official

Use a single pipeline that operates *with context*

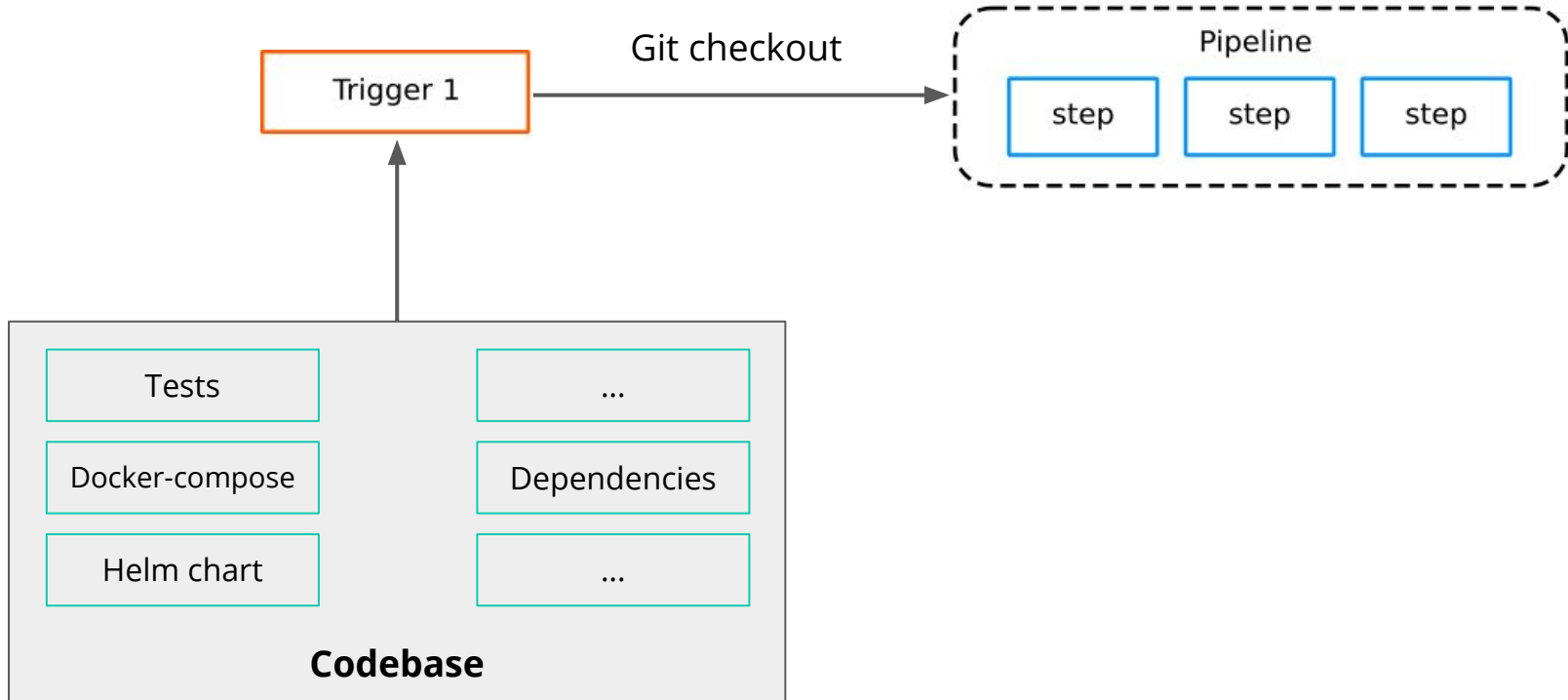
Maintain a single pipeline

Make microservices uniform

Change behavior based on context



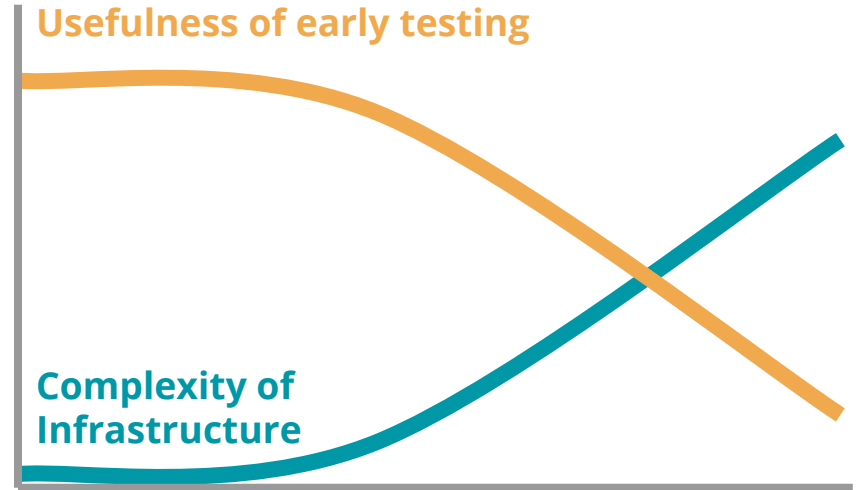
Triggers carry their context



Demo time!

Why Canary?

Testing early
becomes less useful
as infrastructure
complexity rises



<https://codefresh.io/events/canary-deployment-helm-istio-codefresh/>

Summary

Shared pipelines > libraries

Reusable Docker images > Copypasta

Deployment validation with canary

Read the blog post at

<https://codefresh.io/continuous-deployment/ci-cd-pipelines-microservices/>





Questions?

Want to try it yourself?
Open a **FREE** account
today at
[Codefresh.io](https://codefresh.io)



Dan Garfield
@todaywasawesome



Kostis Kapelonis
@codepipes

