codefresh

Webinar

Kubernetes Anti-patterns

Kostis Kapelonis

codefresh

An expert is a person who has made all the mistakes that can be made in a very narrow field

Niels Bohr

Background, Problems and Solutions

codefresh

Kostis Kapelonis

Now Developer advocate at Codefresh
Interests: Kubernetes, CI, CD, GitOps

Ex Java dev (10+ years)
Ex Release manager (5+ years)
Manning author (Java testing with Spock)

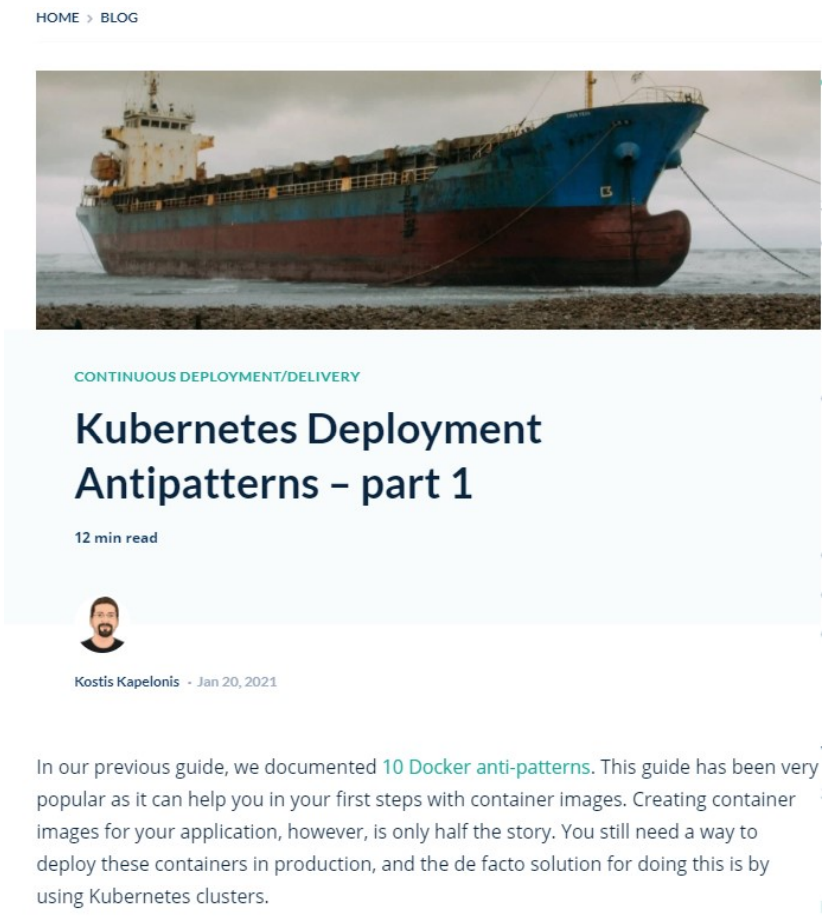
codefresh

Original blog

A collection of all “questionable” practices I have seen companies using without understanding the alternatives

- https://codefresh.io/kubernetes-tutorial/kubernetes-antipatterns-1/
- https://codefresh.io/kubernetes-tutorial/kubernetes-antipatterns-2/
- https://codefresh.io/kubernetes-tutorial/kubernetes-antipatterns-3/

Also published in Medium and dev.to



CLOUD NATIVE COMPUTING FOUNDATION

About ▾ Projects ▾ Certification ▾ Co

The editorial

Articles, announcements, and more that give you a high-level overview of challenges and features.

Datadog and the Container Report, with Michael Gerstenhaber
Craig Box, Kubernetes Podcast from Google

Kubernetes Deployment Antipatterns — part 1
Kostis Kapelonis

Kubernetes Pods Advanced Concepts Explained
Regis Wilson, Release

Discover and invoke services across clusters w
Emeka Nwafor, Product Manager, and Jeremy
Cloud

Kubernetes @kubernetesio · 21m

@codepipes shares part 3 in a series on #Kubernetes deployment antipatterns ⬇️

Kubernetes Deployment Antipatterns—part 3
This is the third and last part in our Kubernetes Anti-patterns series. See also part 1 and part 2 for the previous anti-patterns.
medium.com

7 13

codefresh

JANUARY 2021

Kubernetes Deployment Antipatterns—part 3 4.2K
10 min read · In Container Hub · View story · Details

Kubernetes Deployment Antipatterns—part 2 2.9K
10 min read · In Container Hub · View story · Details

Kubernetes Deployment Antipatterns—part 1 7.2K
12 min read · In Container Hub · View story · Details

DECEMBER 2020

Using Helm to Deploy a Kubernetes Applica... 1.1K
9 min read · In Container Hub · View story · Details

codefresh

Disclaimer!

codefresh

What this talk is about

The Kubernetes cluster is already there (and setup correctly)

All advice is for application deployment and not cluster deployment

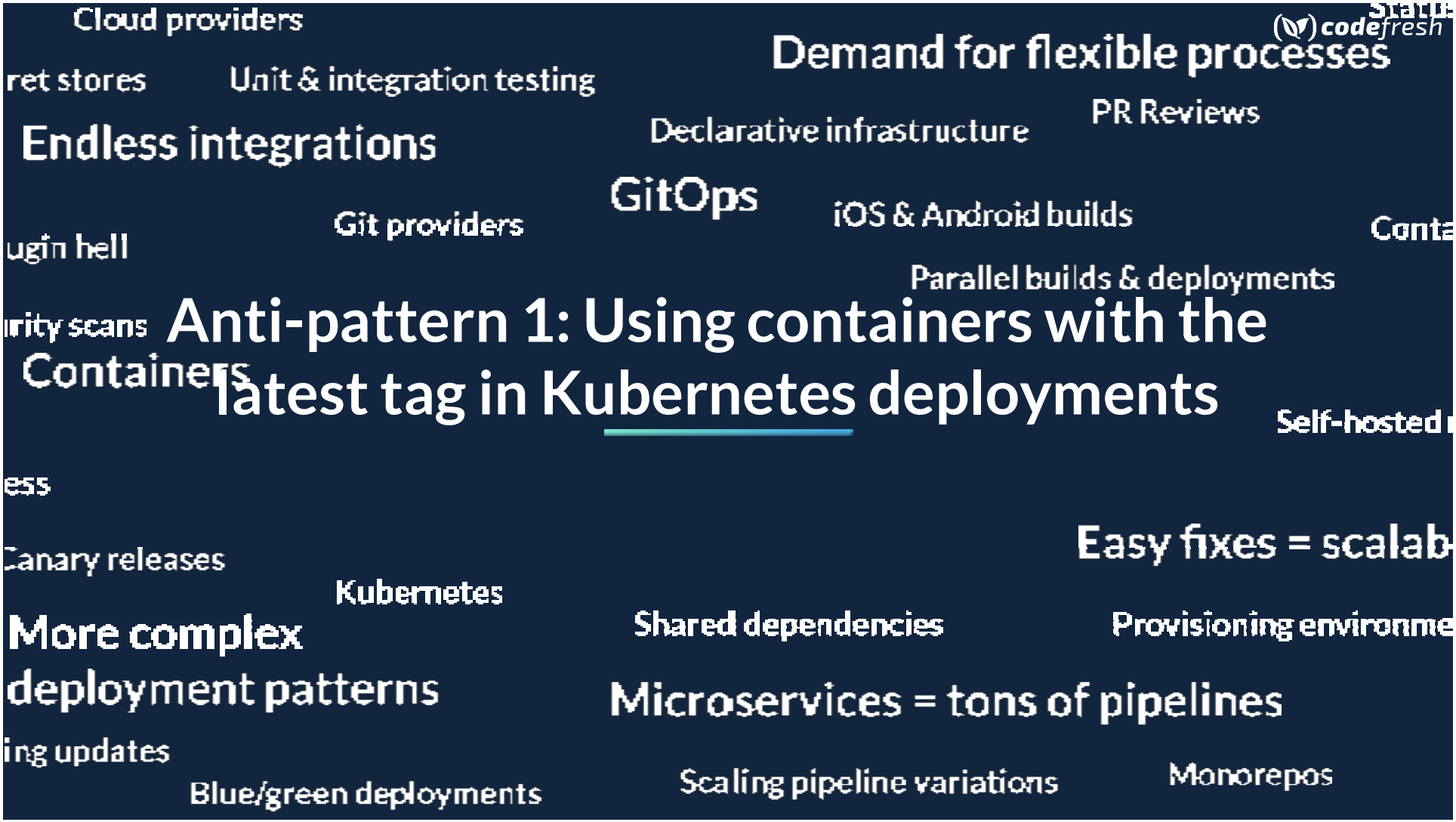
We are interested in applications and not cluster infrastructure

There are different anti-patterns for how to deploy the cluster itself

codefresh

Anti-pattern list

1. Using containers with the latest tag in Kubernetes deployments
2. Baking the configuration inside container images
3. Coupling applications with Kubernetes features/services for no reason
4. Mixing application deployment with infrastructure deployment (e.g. having Terraform deploying apps with the Helm provider)
5. Performing ad-hoc deployments with kubectl edit/patch by hand
6. Using Kubectl as a debugging tool
7. Misunderstanding Kubernetes network concepts
8. Using permanent staging environments instead of dynamic environments
9. Mixing production and non-production clusters
10. Deploying without memory and CPU limits
11. Misusing health probes
12. Not using Helm (and not understanding what Helm brings to the table)
13. Not having deployment metrics to understand what the application is doing
14. Not having a secret strategy/treating secrets in an ad-hoc manner
15. Attempting to go all in Kubernetes (even with databases and stateful loads)


Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
PR Reviews
Endless integrations
Declarative infrastructure
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
rity scans
Anti-pattern 1: Using containers with the latest tag in Kubernetes deployments
Containers
ess
Canary releases
Easy fixes = scalab
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos

Don’t use latest tag

codefresh

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-bad-deployment
spec:
  template:
    metadata:
      labels:
        app: my-badly-deployed-app
    spec:
      containers:
      - name: dont-do-this
        image: docker.io/myusername/my-app:latest
```

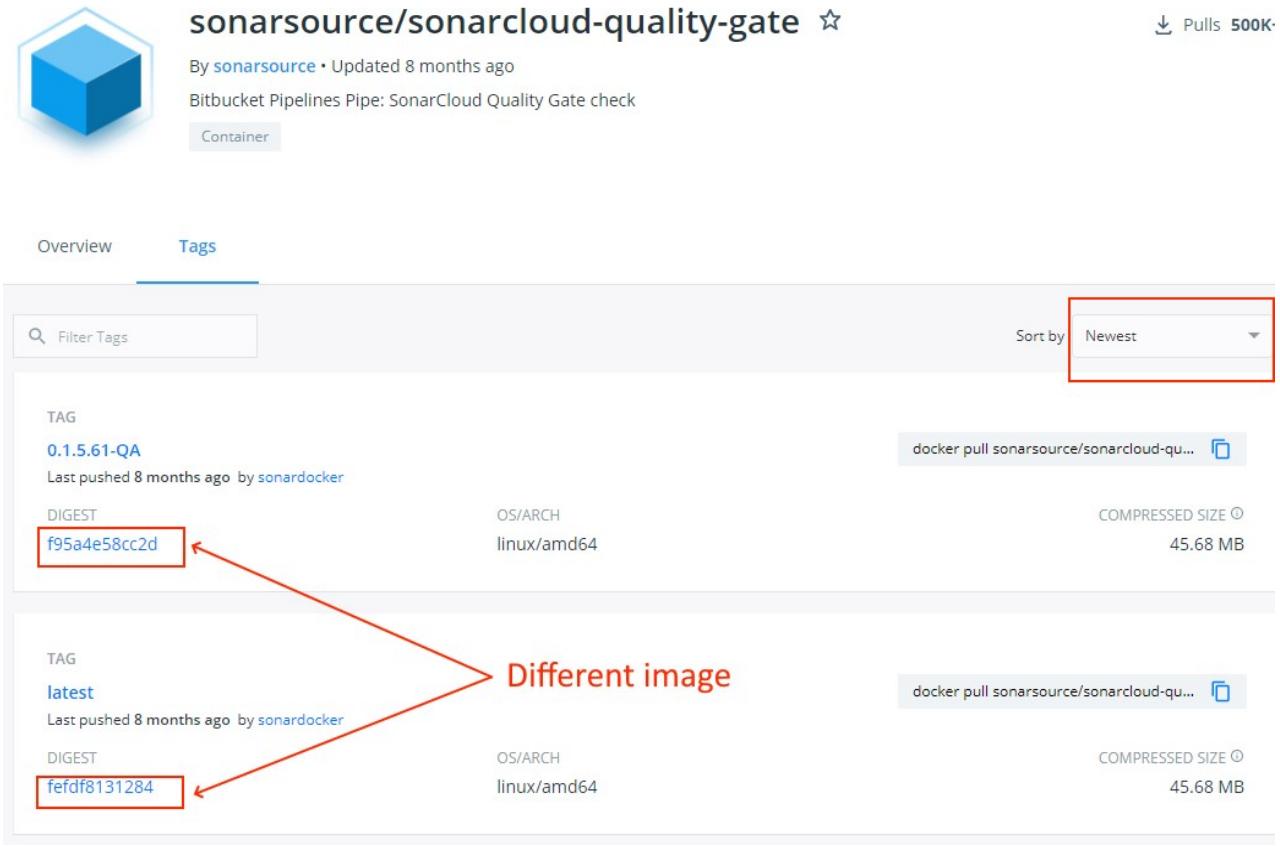
Latest tag does NOT mean the most recent or the last one built

codefresh

Latest is not a special tag in Docker (or Kubernetes).

It is just the default tag used if you don't specify a tag yourself

codefresh



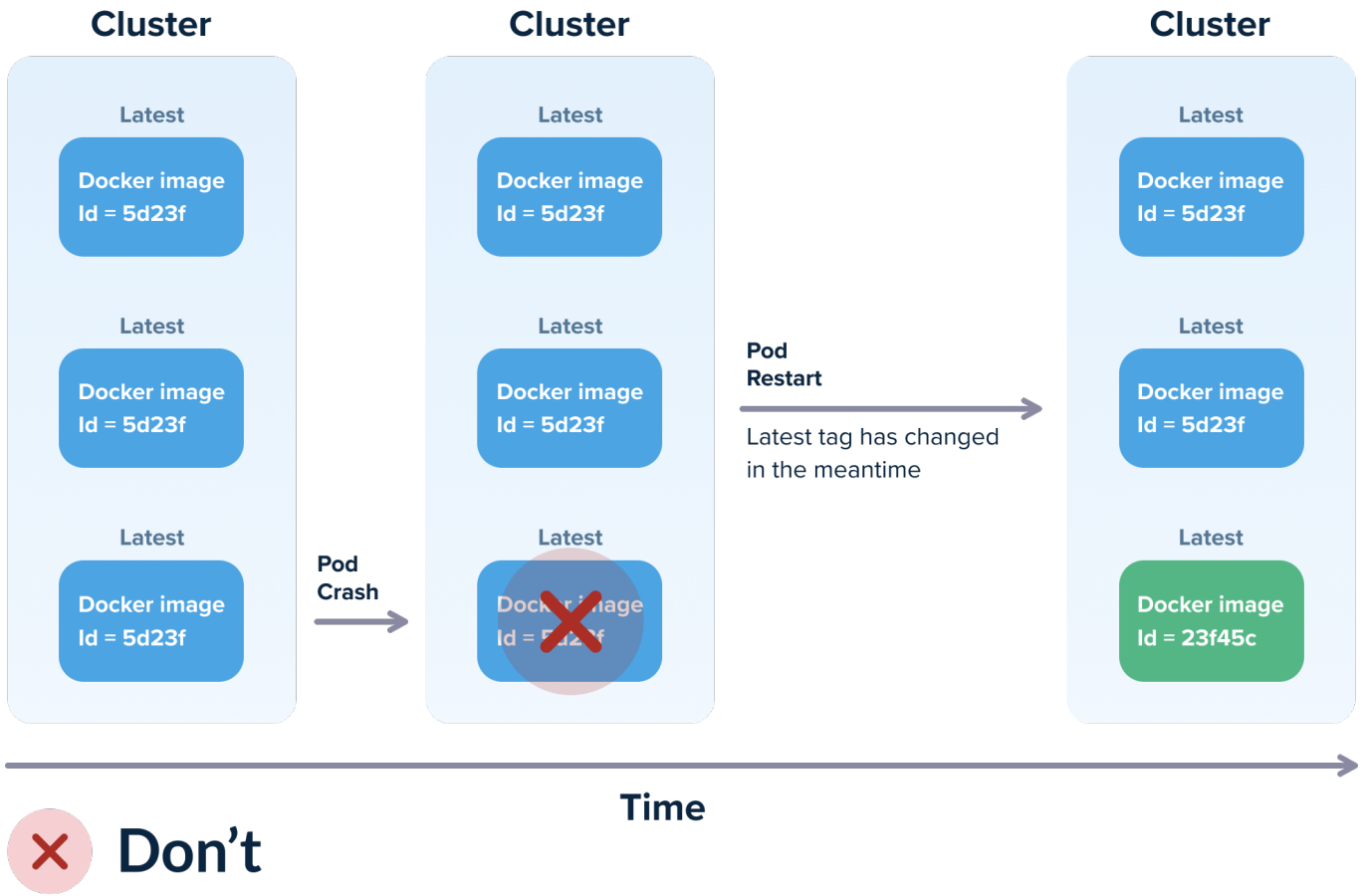
REJECTED

How to detect this anti-pattern

Latest is a transient tag

1. It can be any version of your app
2. You don't really know which application version was deployed
3. Worst case scenario: latest definition changes in the middle of a deployment

codefresh



Don't

codefresh

Solution

Use specific tags in Deployments

Strategy 1 = use the Git hash as a tag
- myapp: ccdd07d
- myapp: a70bfe1
- myapp: 95be785

Strategy 2 = use application version (semver)
- myapp: 0.1
- myapp: 0.2
- myapp: 0.3

Strategy3 : Use date/build number
- myapp: 8789
- myapp: 8790
- myapp: 8791

codefresh

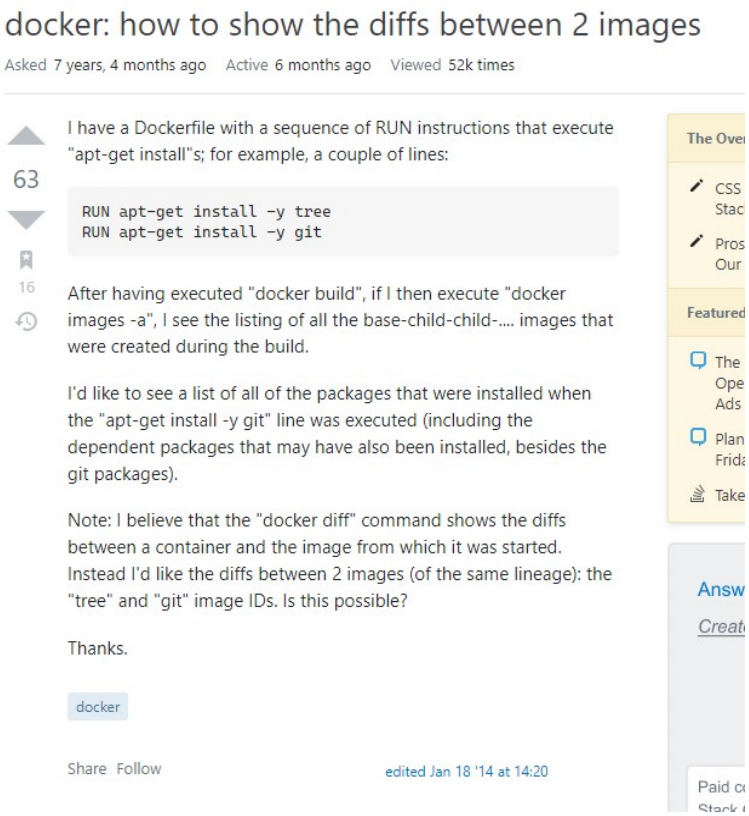
https://unsplash.com/photos/ABNhXfQFtdU

codefresh

Big gotcha!

All Docker tags are mutable (!!!)

Tags can be overwritten. So version 0.1 that John has might be different then version 0.1 that Mary has



docker: how to show the diffs between 2 images

Asked 7 years, 4 months ago Active 6 months ago Viewed 52k times

63

16

I have a Dockerfile with a sequence of RUN instructions that execute "apt-get install"s; for example, a couple of lines:

```
RUN apt-get install -y tree
RUN apt-get install -y git
```

After having executed "docker build", if I then execute "docker images -a", I see the listing of all the base-child-child-.... images that were created during the build.

I'd like to see a list of all of the packages that were installed when the "apt-get install -y git" line was executed (including the dependent packages that may have also been installed, besides the git packages).

Note: I believe that the "docker diff" command shows the diffs between a container and the image from which it was started. Instead I'd like the diffs between 2 images (of the same lineage): the "tree" and "git" image IDs. Is this possible?

Thanks.

docker

Share Follow edited Jan 18 '14 at 14:20

codefresh

An all too common scenario

The problems of mutable tags

1. Mary(dev) deploys image with tag 3.7 on QA Kubernetes cluster
2. Alex (QA) tests image with tag 3.7 and finds a bug
3. John (dev) deploys another image with same tag 3.7 **(oops)**
4. Mary can no longer find the bug as image is different than what Alex tested

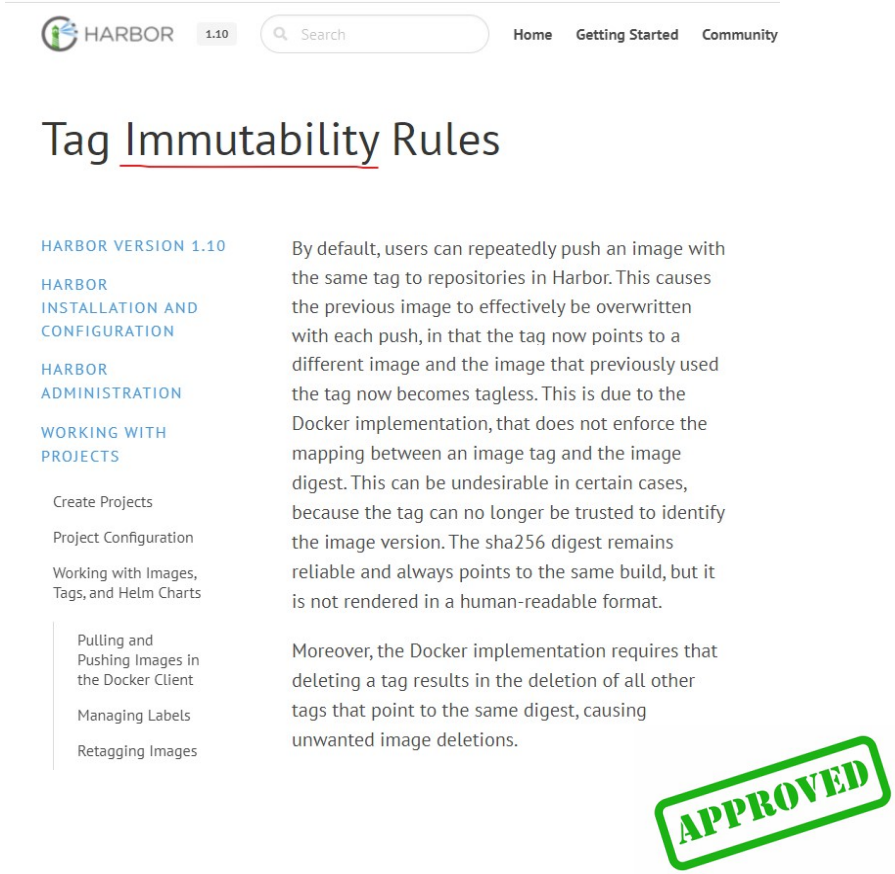
codefresh

Solution to gotcha

Use immutable tags

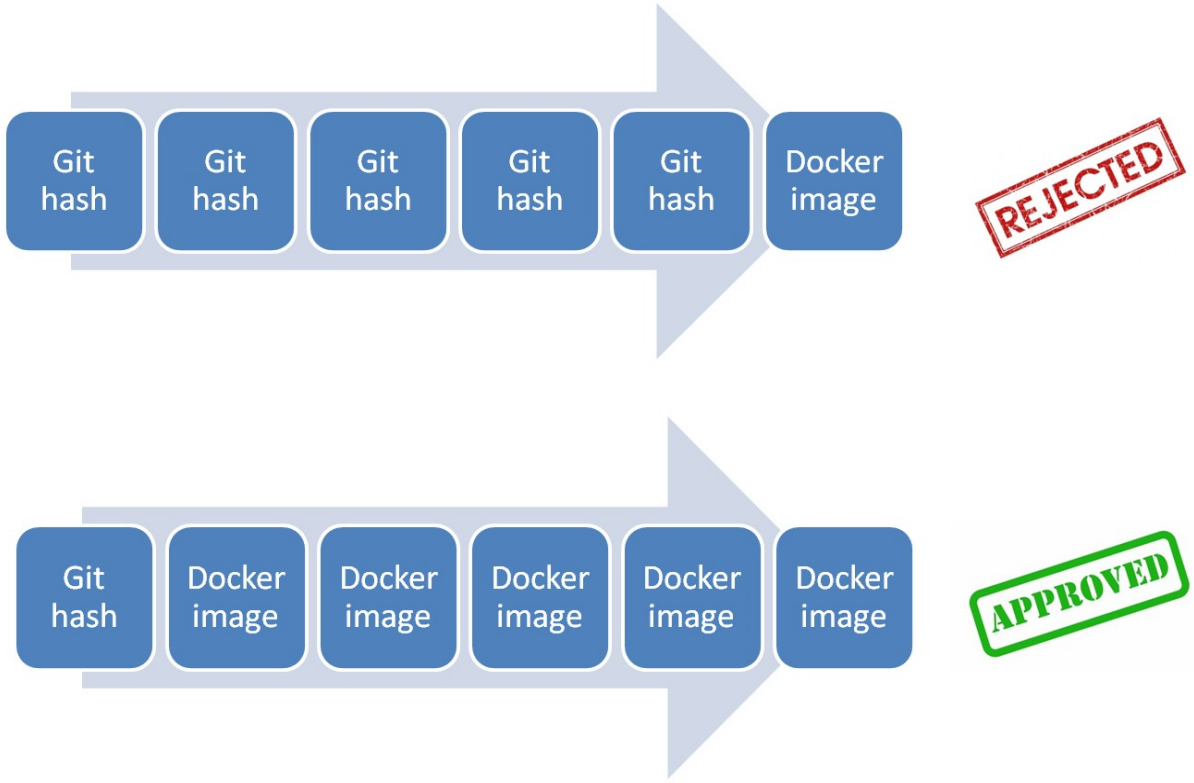
Only push container tags ONCE. This way you know exactly what is in each container image

Check your Registry documentation



Build your image once in CI

codefresh

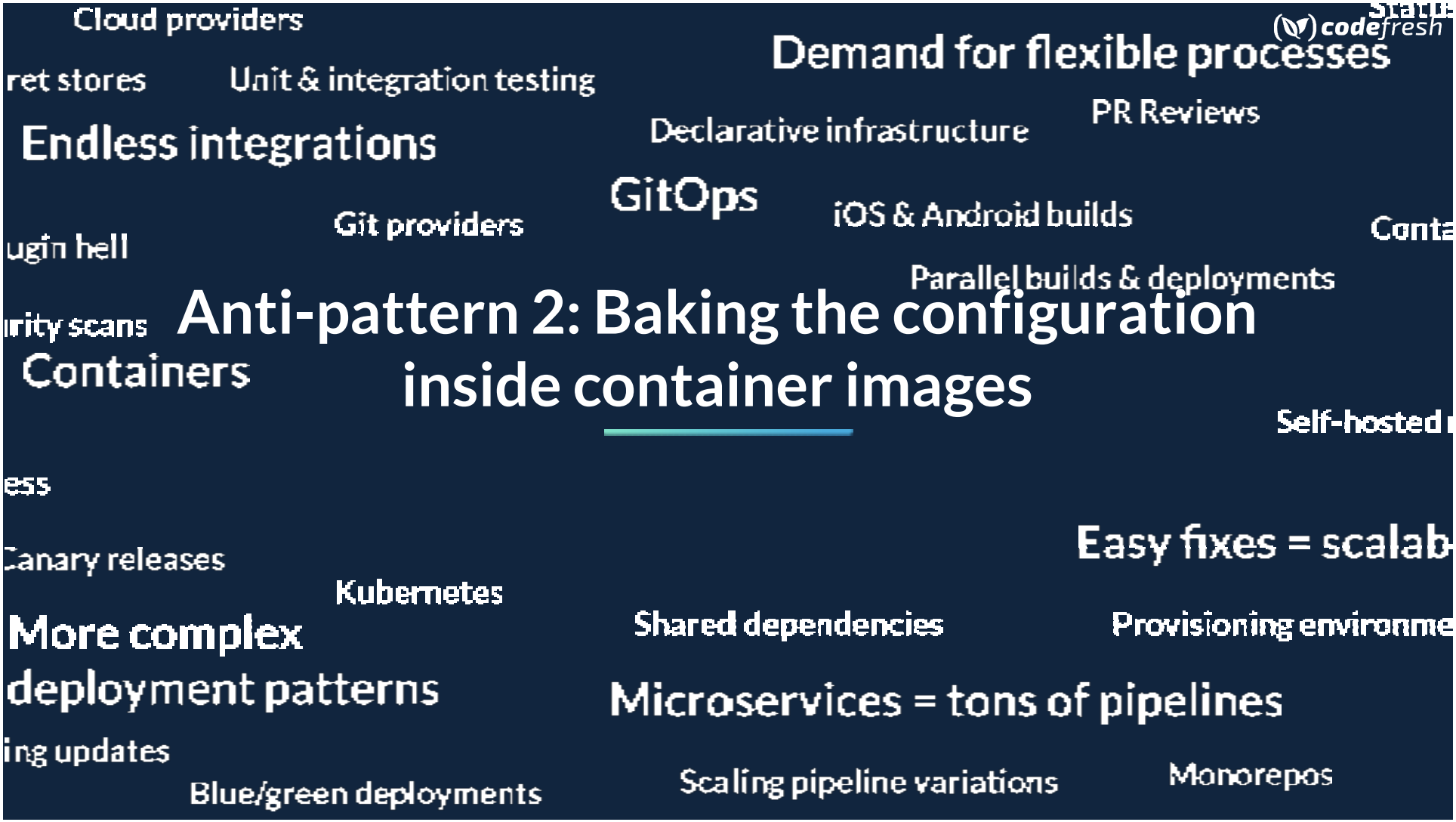


Use specific container tags in deployments. We suggest the application version strategy (semver)

codefresh

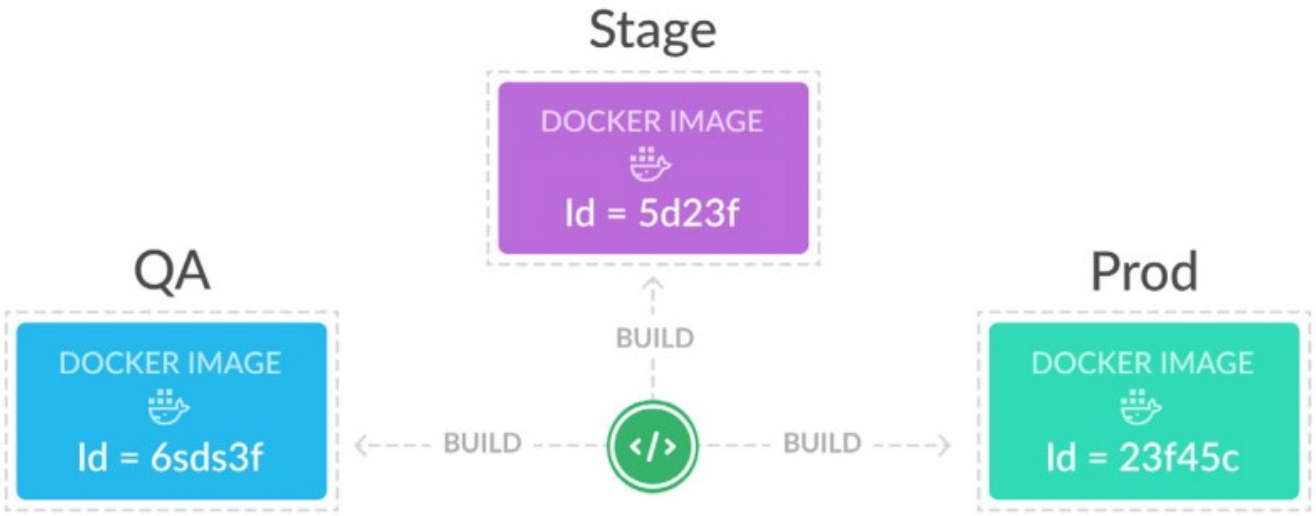
Treat Docker tags as immutable.

Force immutable tags on the Registry level.

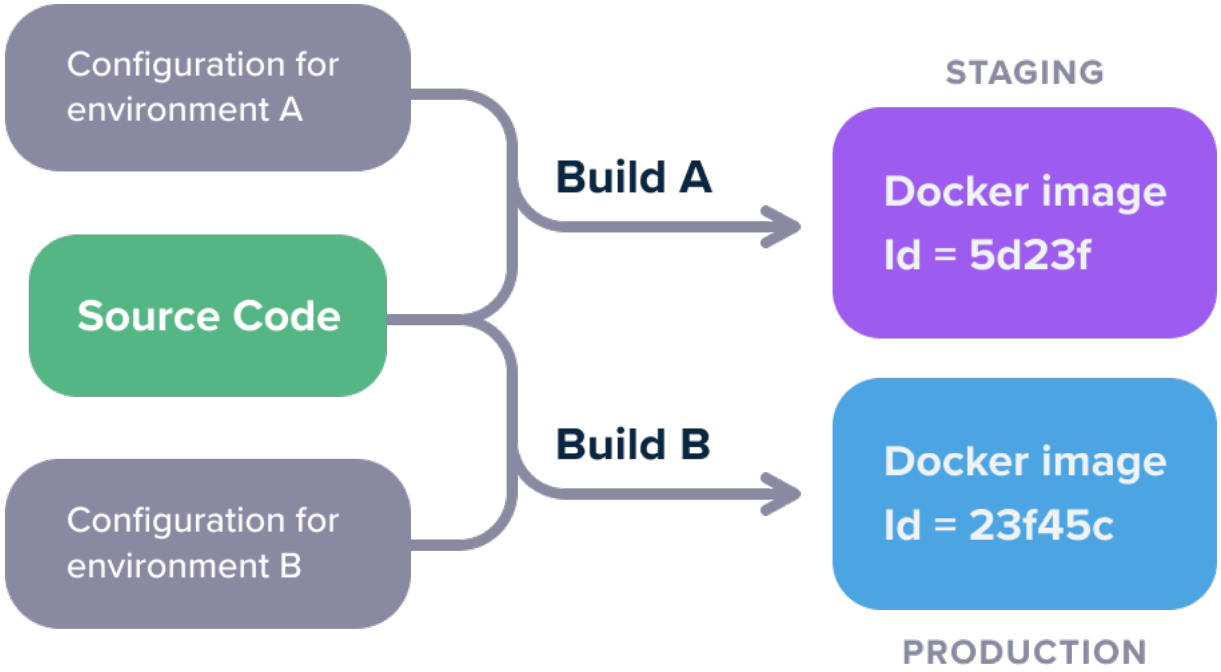

Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
PR Reviews
Endless integrations
Declarative infrastructure
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
Anti-pattern 2: Baking the configuration inside container images
rity scans
Containers
Self-hosted
ess
Easy fixes = scalab
Canary releases
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos

codefresh

Different images per cluster



codefresh



Don't

codefresh

How to detect this anti-pattern

Hardcoded configuration smells

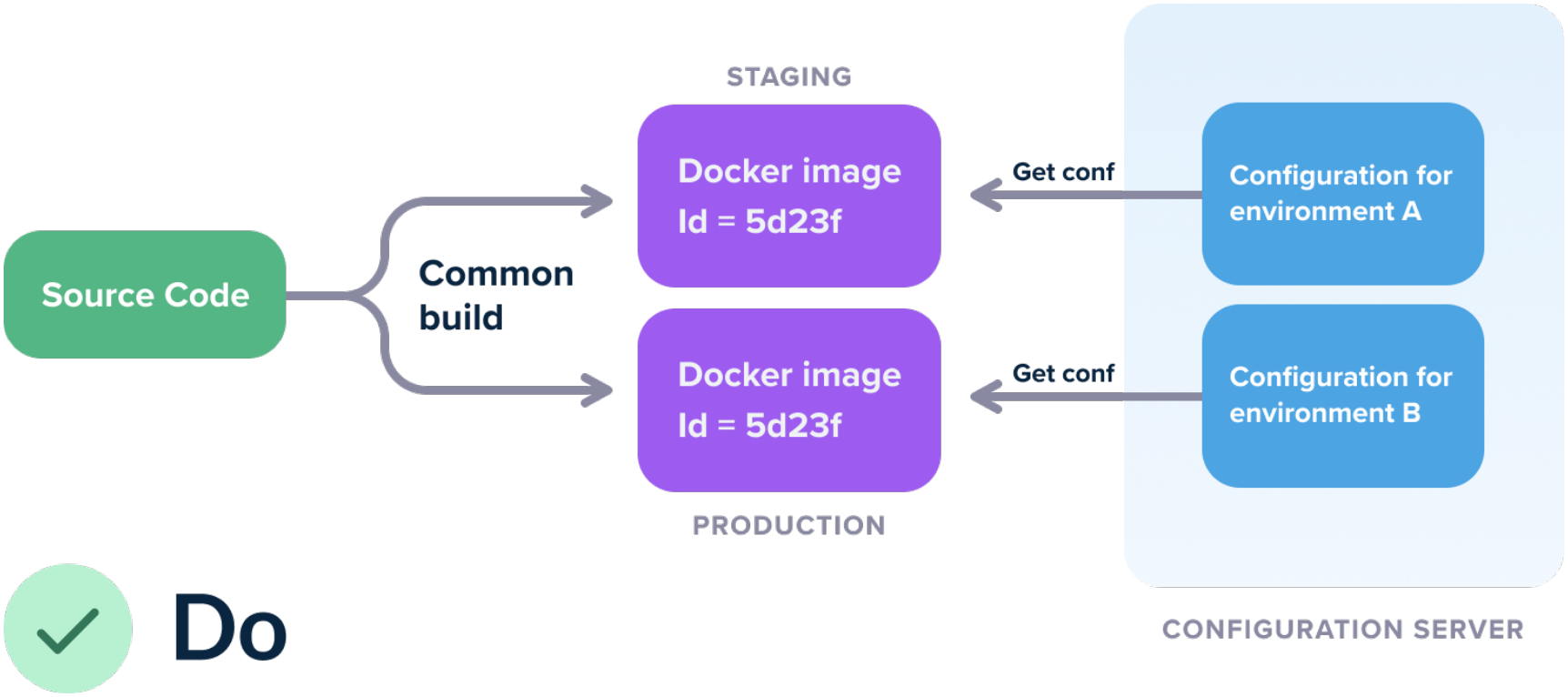
1. Tags myapp:staging, myapp:qa, myapp:prod
2. Git branches staging, production, qa
3. Config folder in Git with prod, qa, staging subfolders in application source code

A single Docker image should be deployed to all clusters (QA/Staging/Prod)

codefresh

Configuration is loaded externally and never hardcoded in the container

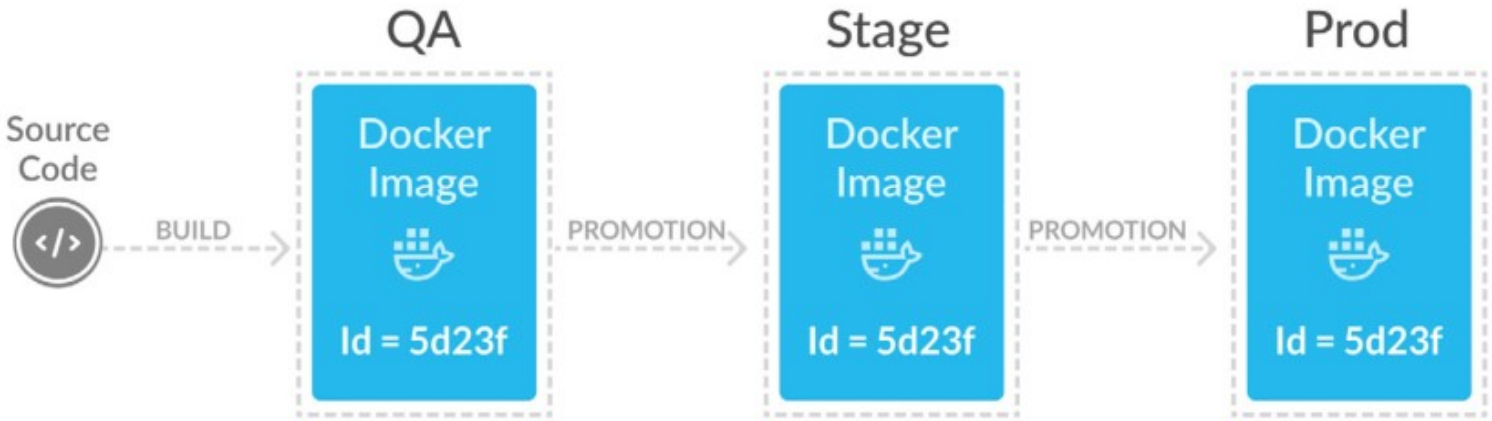
codefresh



Do

codefresh

Promote the same image

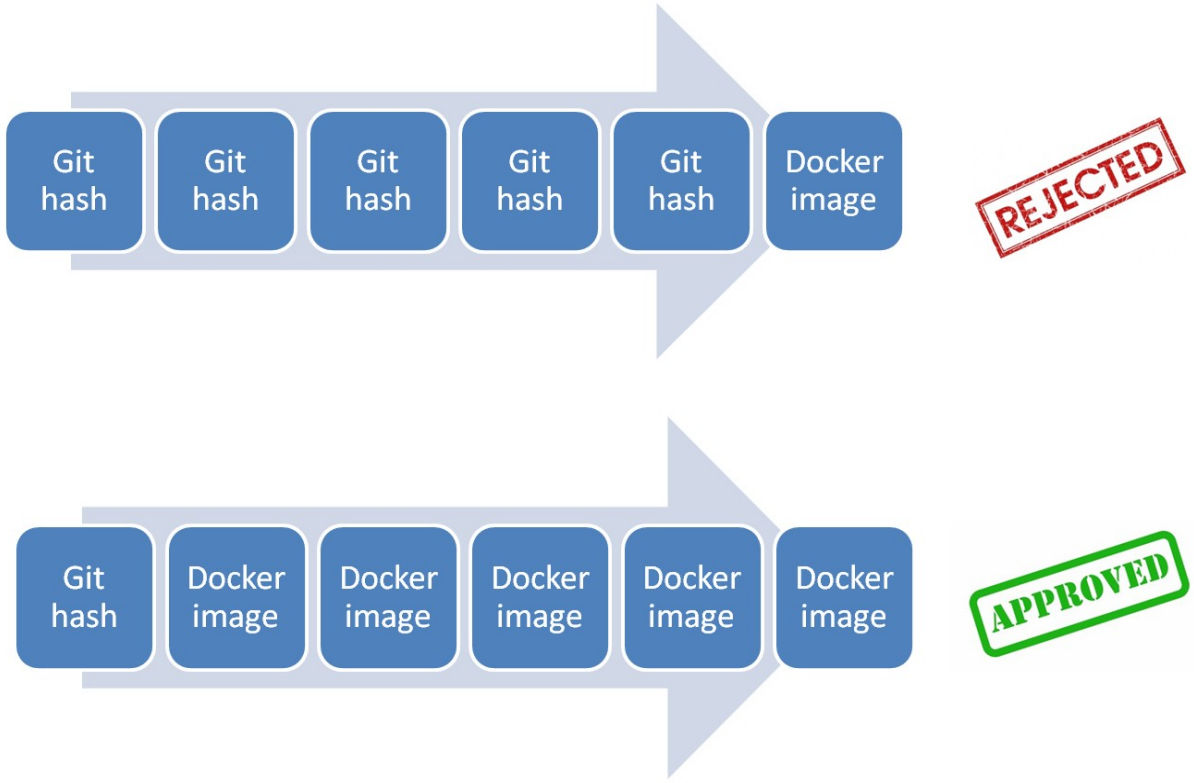


codefresh

https://unsplash.com/photos/QMjCzOGeglA

Build your image once in CI

codefresh



codefresh

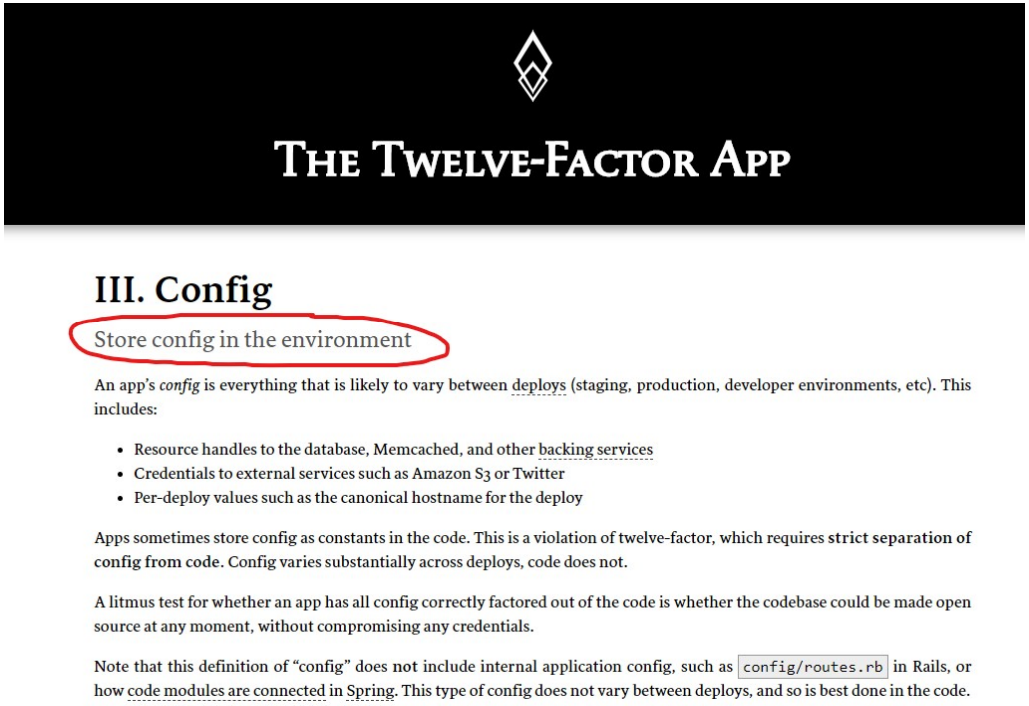
Solution to hardcoded configuration

Decouple configuration

1. Kubernetes configmaps
2. Consul
3. etcd
4. Zookeeper
5. Bitnami Sealed secrets/ Mozilla Sops
6. Hashicorp vault

codefresh

This was good advice even before k8s



THE TWELVE-FACTOR APP

III. Config

Store config in the environment

An app's *config* is everything that is likely to vary between deploys (staging, production, developer environments, etc). This includes:

- Resource handles to the database, Memcached, and other backing services
- Credentials to external services such as Amazon S3 or Twitter
- Per-deploy values such as the canonical hostname for the deploy

Apps sometimes store config as constants in the code. This is a violation of twelve-factor, which requires **strict separation of config from code**. Config varies substantially across deploys, code does not.

A litmus test for whether an app has all config correctly factored out of the code is whether the codebase could be made open source at any moment, without compromising any credentials.

Note that this definition of "config" does **not** include internal application config, such as `config/routes.rb` in Rails, or how code modules are connected in Spring. This type of config does not vary between deploys, and so is best done in the code.

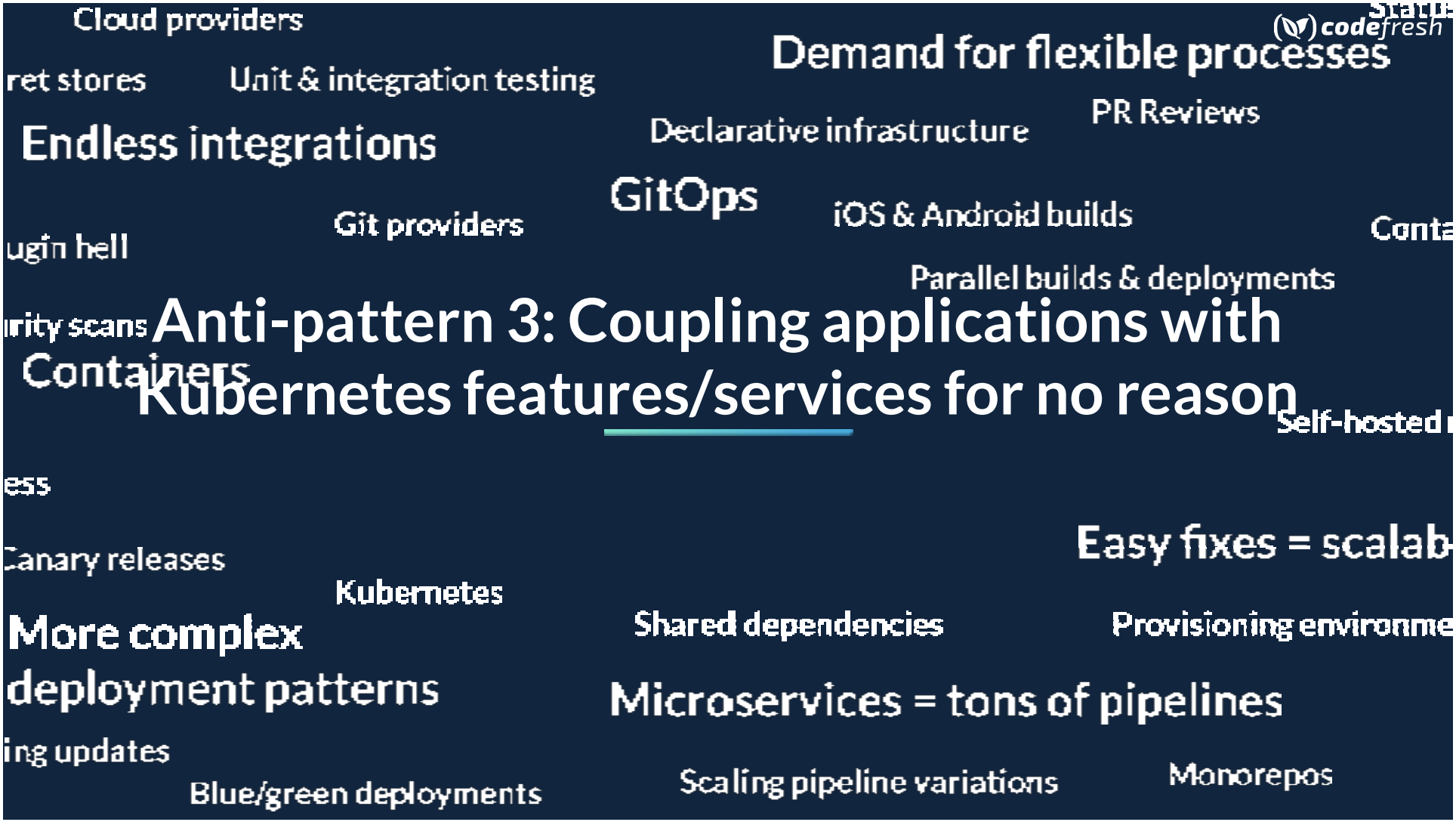
https://12factor.net/config

All clusters get a single image.

Test the same image developers created

codefresh

Each cluster has different runtime configuration


Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
PR Reviews
Endless integrations
Declarative infrastructure
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
Anti-pattern 3: Coupling applications with
Containers
Kubernetes features/services for no reason
ess
Canary releases
Easy fixes = scalab
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos

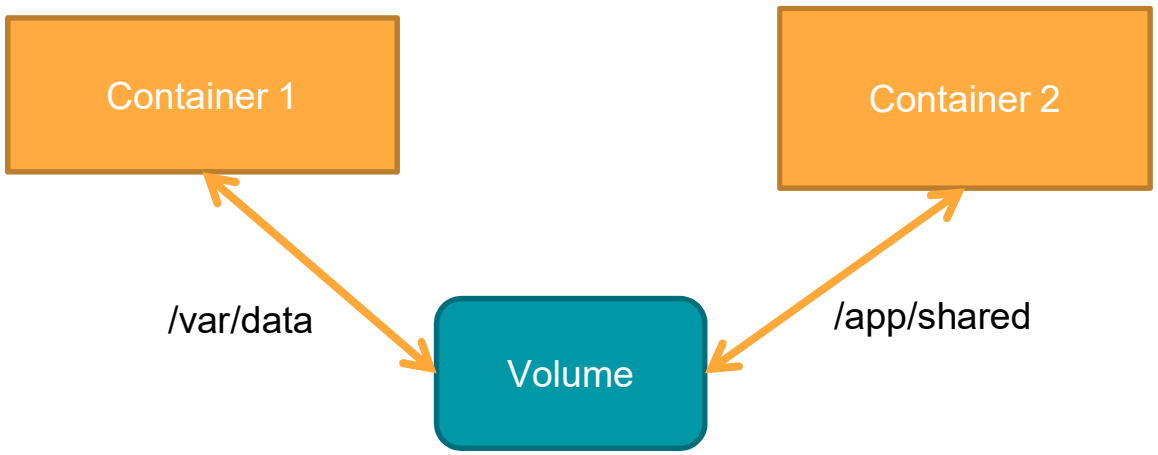
codefresh

Assuming a prod namespace

```
func main() {

        resp, err := http.Get("my-backend.prod.svc.cluster.local")
        if err != nil {
                panic(err)
        }
        defer resp.Body.Close()

        // Print the HTTP response status.
        fmt.Println("Response status:", resp.Status)
}
```

codefresh

Poor man's message queue



codefresh

Common mistakes

Coupling to Kubernetes

1. Expect a certain volume configuration
2. Expect a certain naming of services/DNS
3. Read information directly from labels and annotations
4. Query the pod itself (e.g. for the IP address)
5. Need a sidecar or init (even in local development)
6. Call other services directly with their API (e.g. vault)

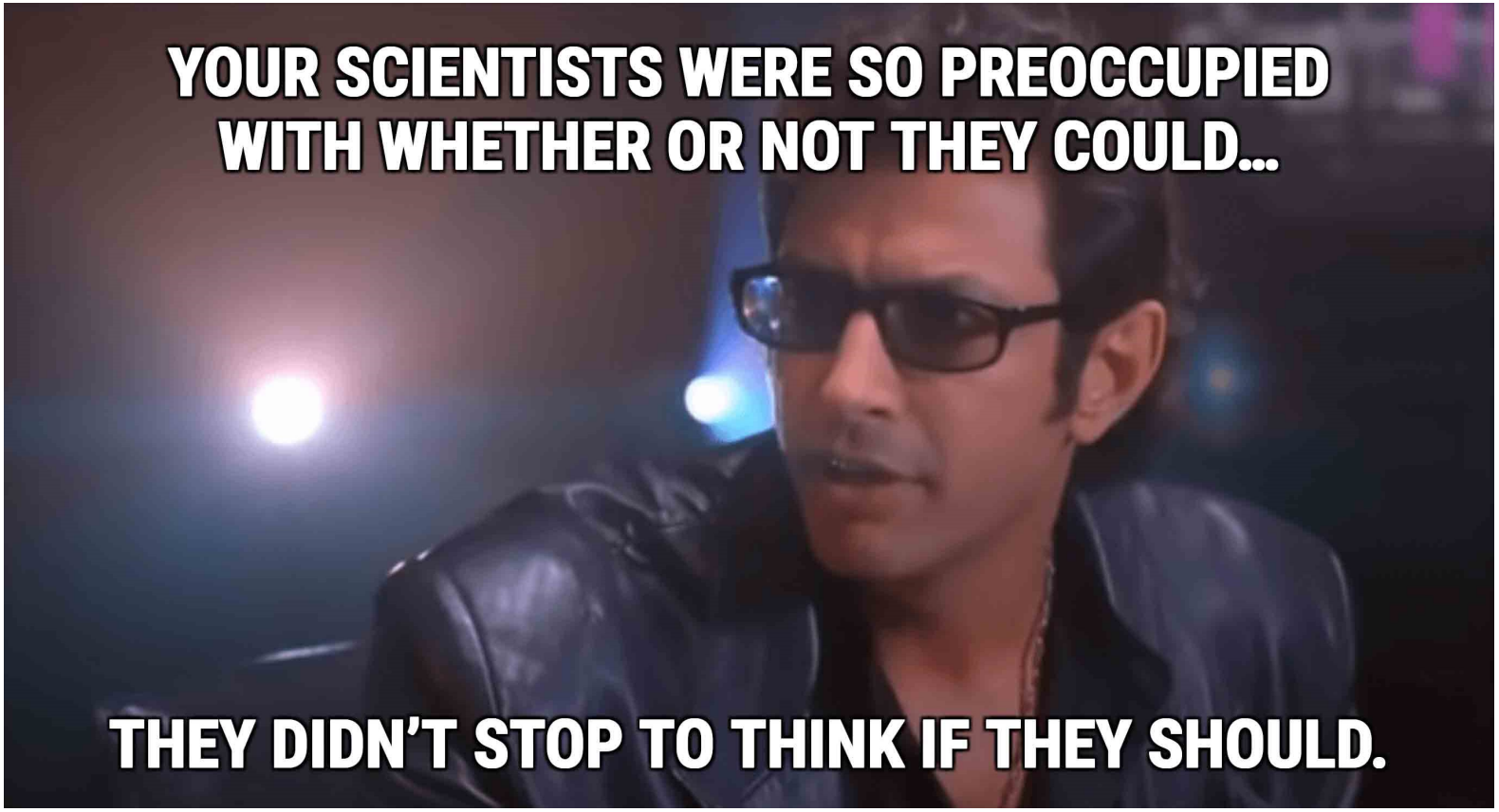
codefresh

Getting secrets from vault

```
final Map<String, String> secrets = new HashMap<String, String>();
secrets.put("value", "world");
secrets.put("other_value", "You can store multiple name/value pairs under a sing

// Write operation
final LogicalResponse writeResponse = vault.logical()
                                      .write("secret/hello", secrets);

...

// Read operation
final String value = vault.logical()
                      .read("secret/hello")
                      .getData().get("value");
```

```
public class Example {                                                    COPY

  // inject the actual template
  @Autowired
  private VaultOperations operations;

  public void writeSecrets(String userId, String password) {

    Map<String, String> data = new HashMap<String, String>();
    data.put("password", password);

    operations.write(userId, data);
  }

  public Person readSecrets(String userId) {

    VaultResponseSupport<Person> response = operations.read(userId, Person.class);
    return response.getBody();
    }
}
```

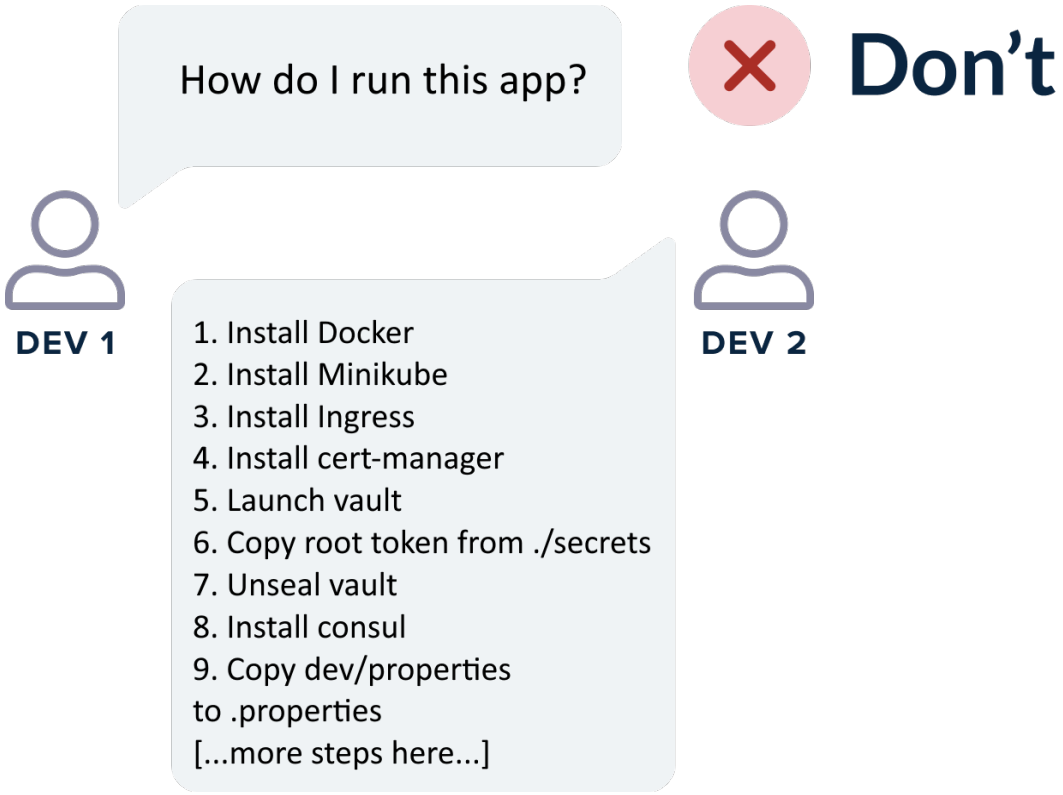
codefresh


YOUR SCIENTISTS WERE SO PREOCCUPIED
WITH WHETHER OR NOT THEY COULD...
THEY DIDN'T STOP TO THINK IF THEY SHOULD.

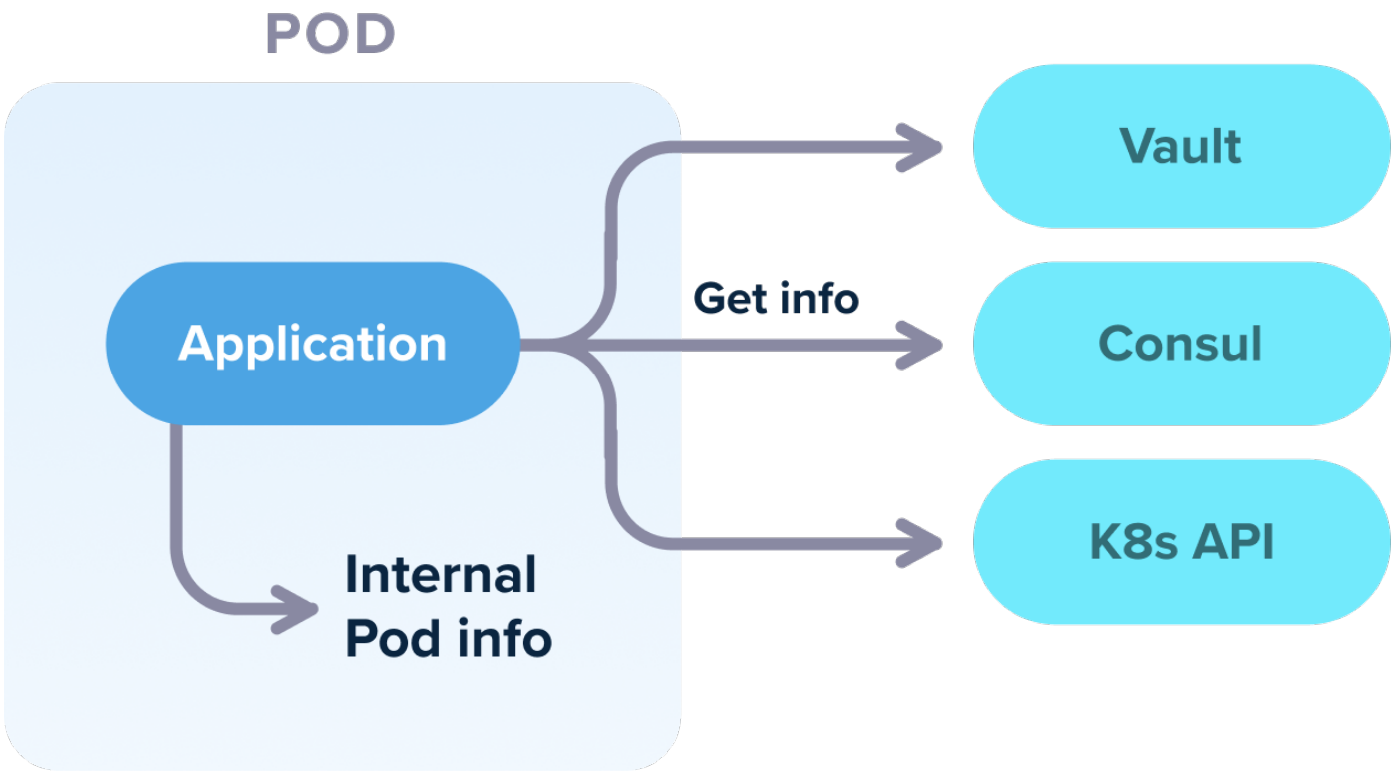
codefresh

Making your life hard

- Developers have a hard time running the app.
- CI pipelines are super complex
- Integration testing is a mess
- There are too many moving parts

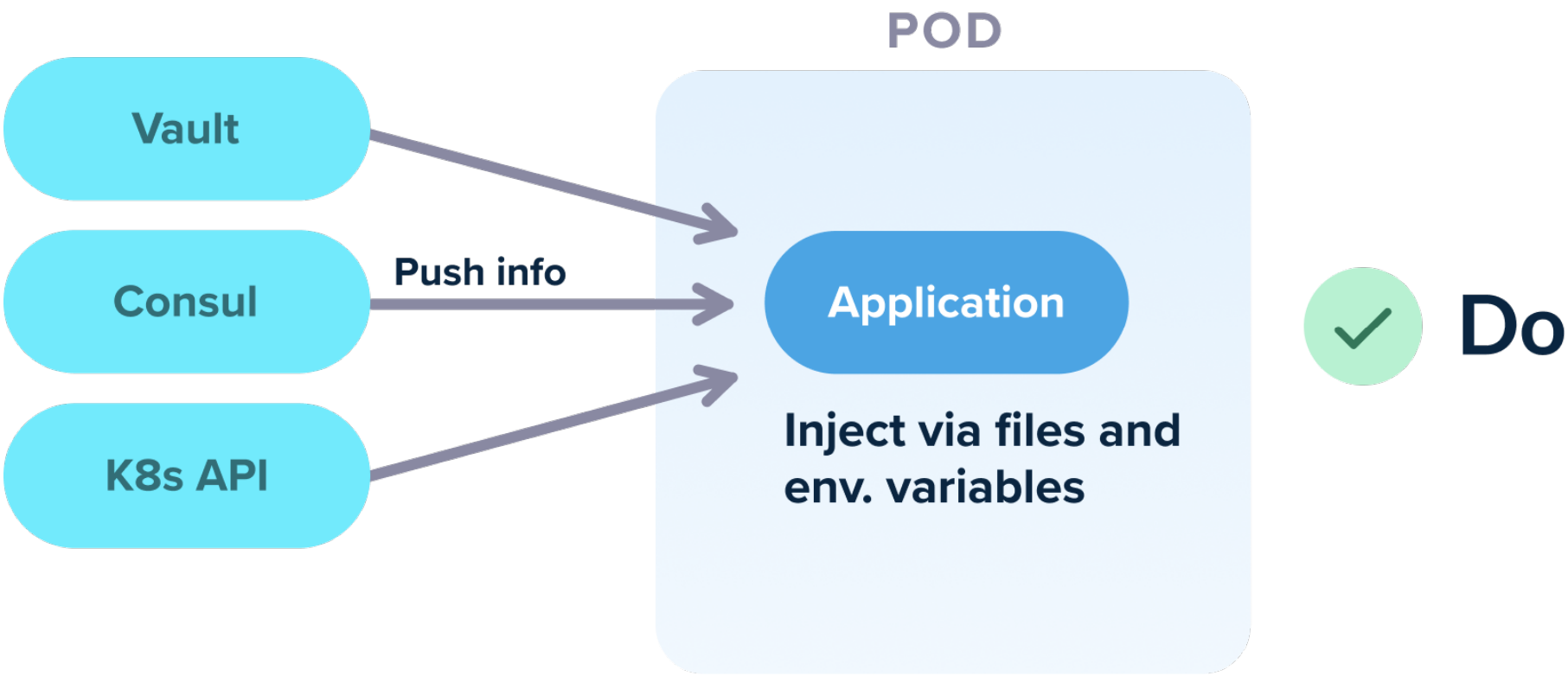


codefresh



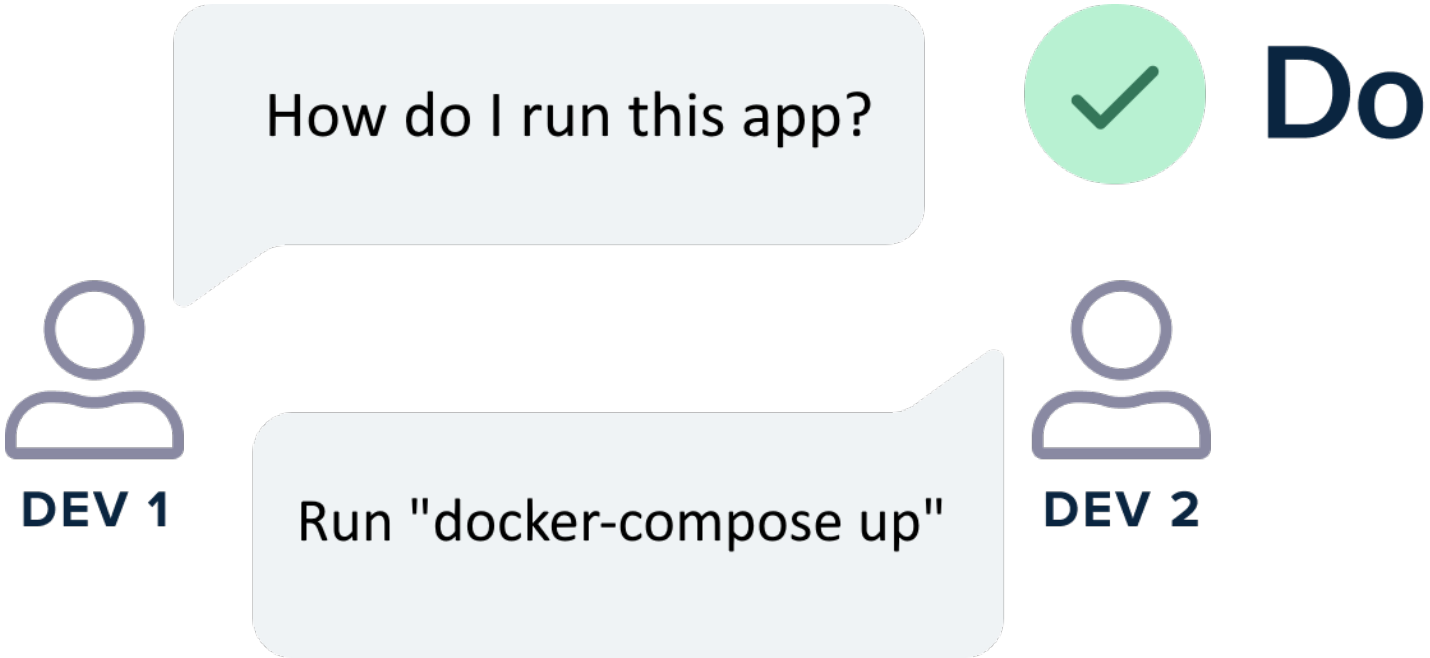
✕ Don't

codefresh



Do

codefresh



codefresh

Use dedicated solutions

Kubernetes local development tools

okteto

TILT

garden

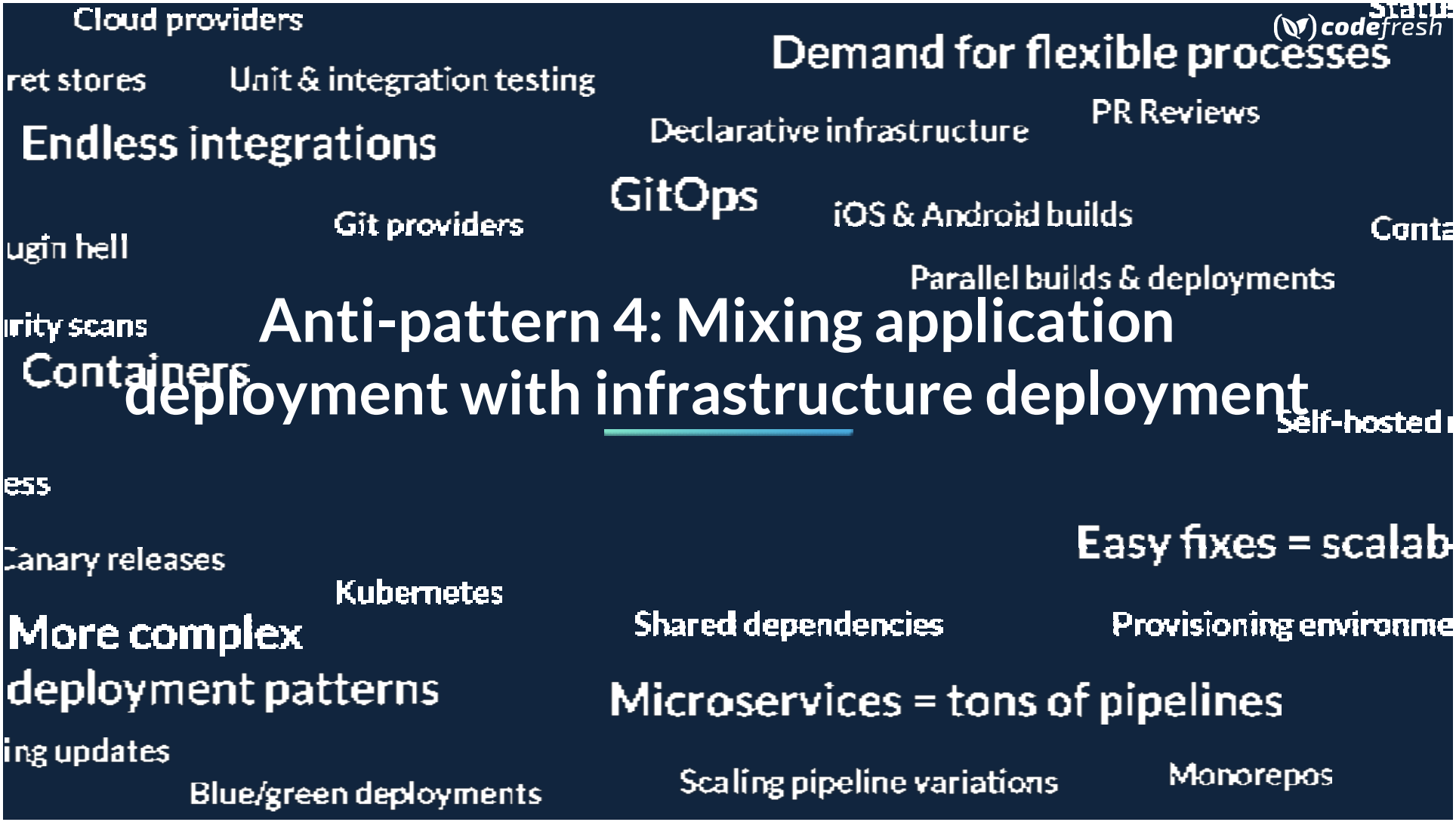
- https://codefresh.io/kubernetes-tutorial/telepresence-2-local-development/
- https://codefresh.io/kubernetes-tutorial/okteto/
- https://codefresh.io/kubernetes-tutorial/local-kubernetes-development-tilt-dev/
- https://codefresh.io/howtos/local-k8s-draft-skaffold-garden/

Don’t use special Kubernetes services/APIs

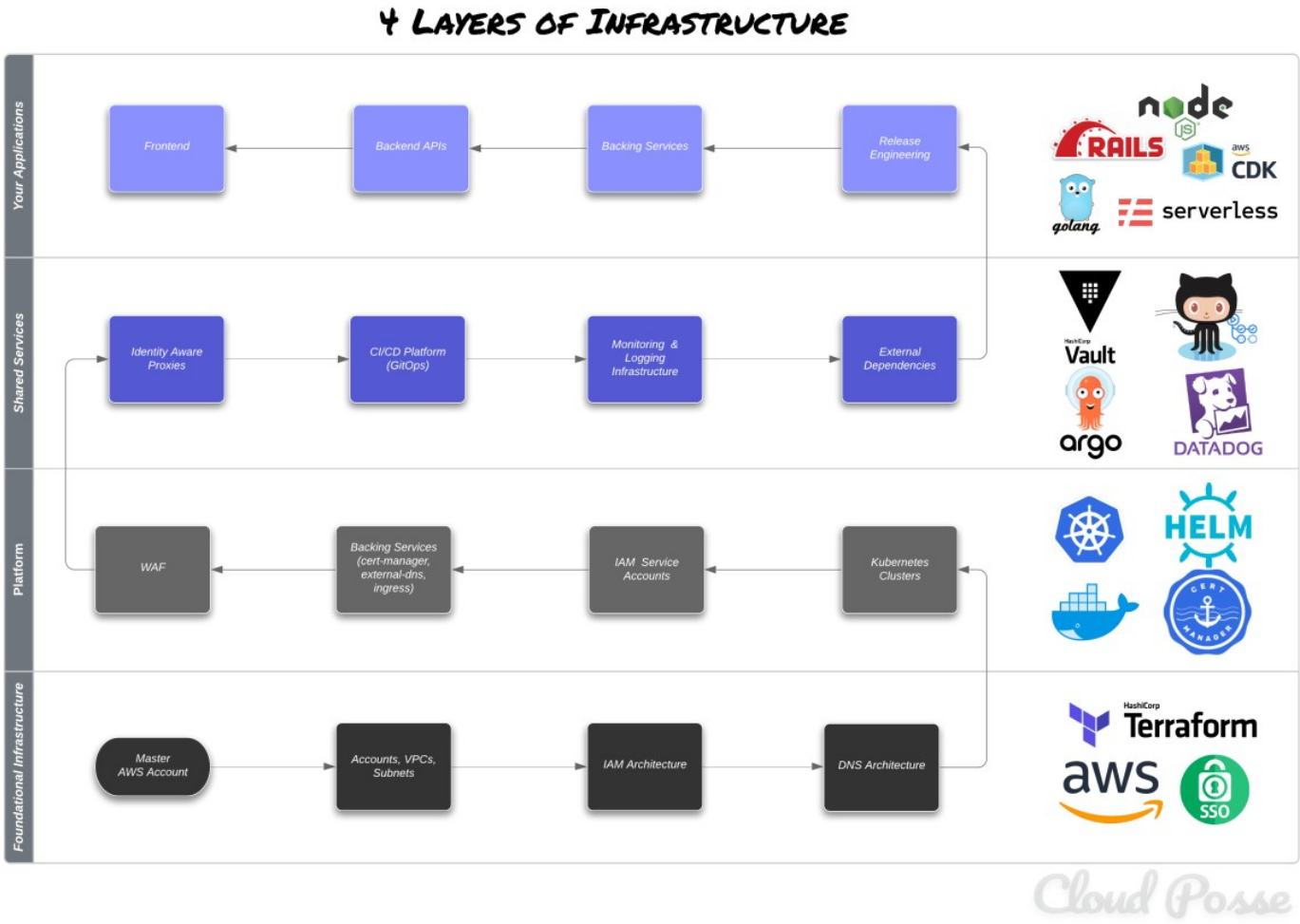
Look at special tools for local dev

codefresh

Your application shouldn’t even know that it is running inside Kubernetes


Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
PR Reviews
Endless integrations
Declarative infrastructure
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
rity scans
Containers
Anti-pattern 4: Mixing application deployment with infrastructure deployment
ess
Canary releases
Easy fixes = scalab
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos

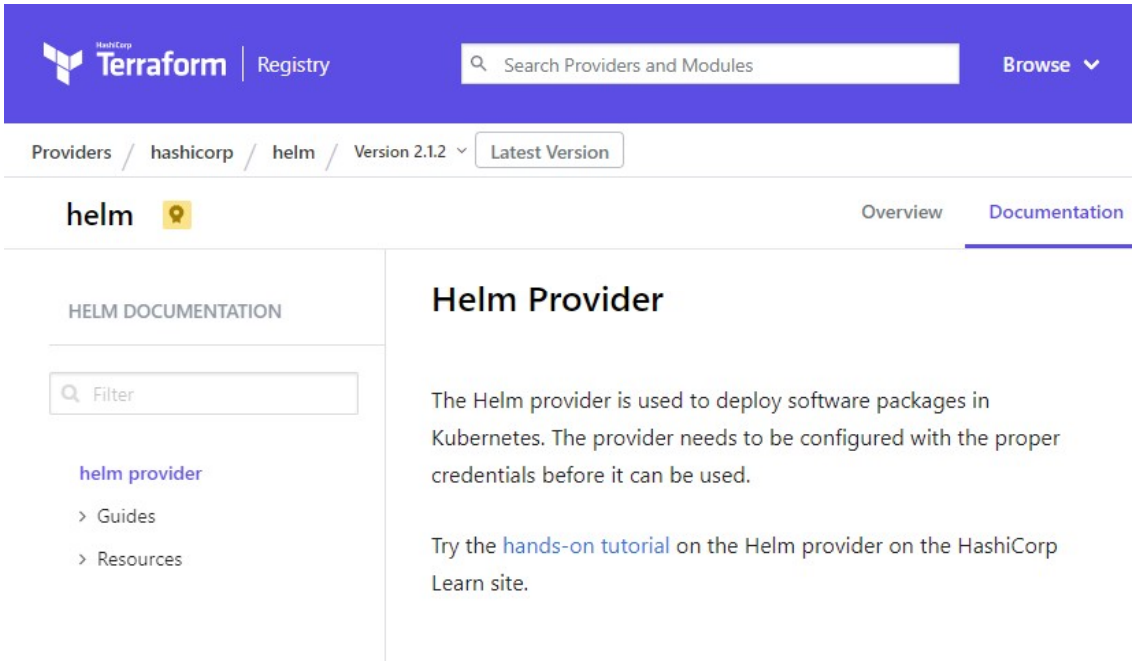
codefresh



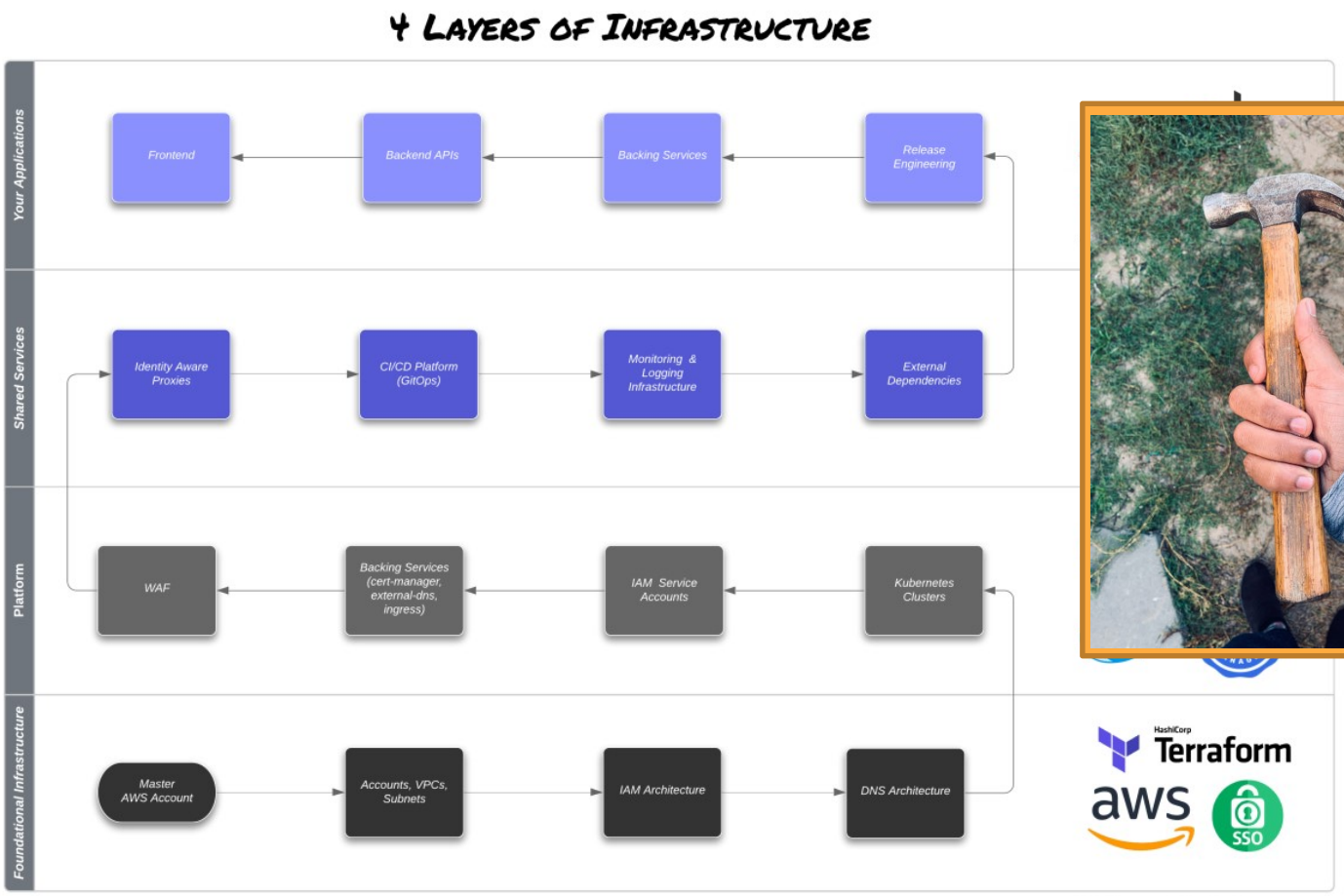
https://cloudposse.com/big-picture/

codefresh

Terraform Kubernetes provider



codefresh



Terraform

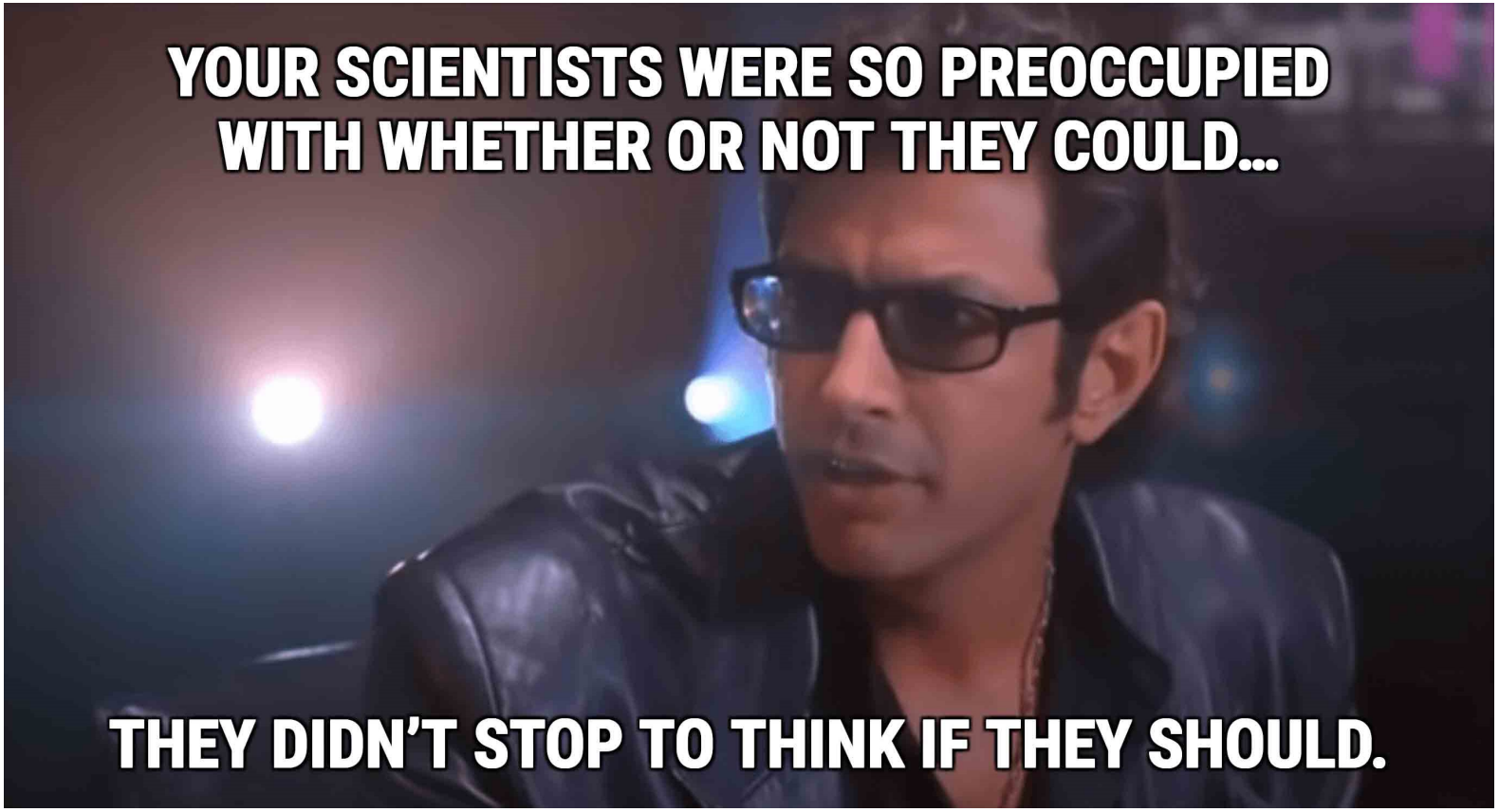
HashiCorp Terraform

aws SSO

REJECTED

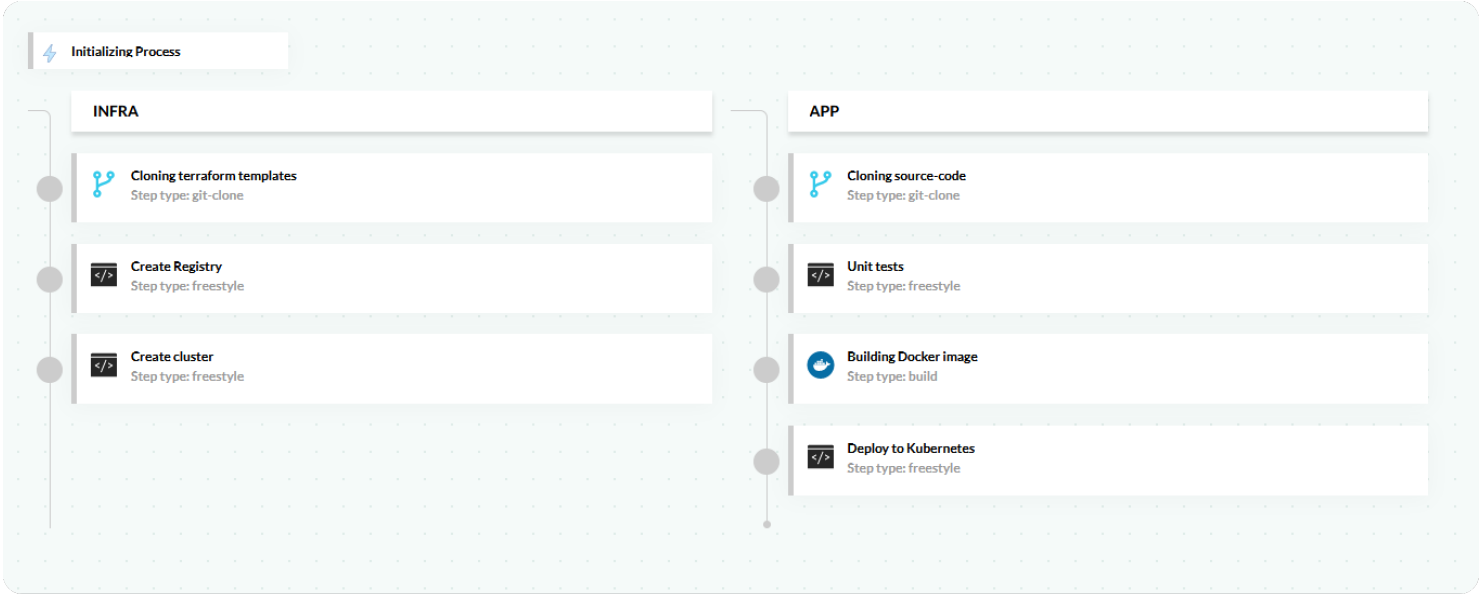
https://unsplash.com/photos/B4YHKz6lLrQ

codefresh


YOUR SCIENTISTS WERE SO PREOCCUPIED
WITH WHETHER OR NOT THEY COULD...
THEY DIDN'T STOP TO THINK IF THEY SHOULD.

codefresh

Single pipeline for Infra and app

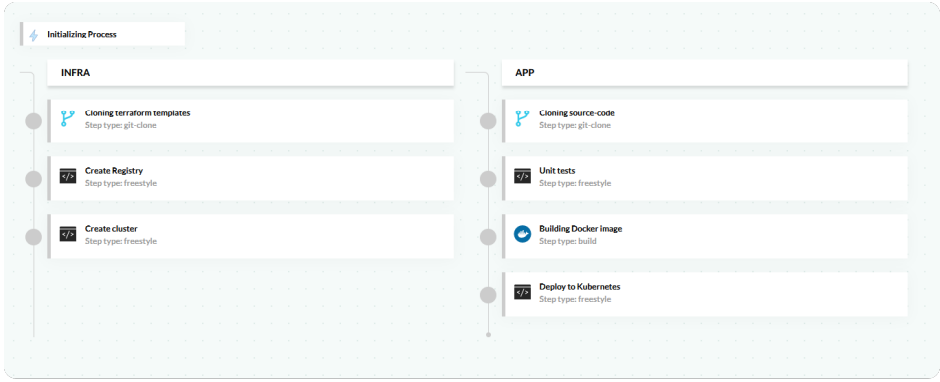


Don't

codefresh

Mixing infrastructure and application deployment

1. You are wasting time for everybody (dev/ops)
2. You are making life difficult for developers
3. Your deployments are very complex
4. Who should look at a broken pipeline? Dev or ops?



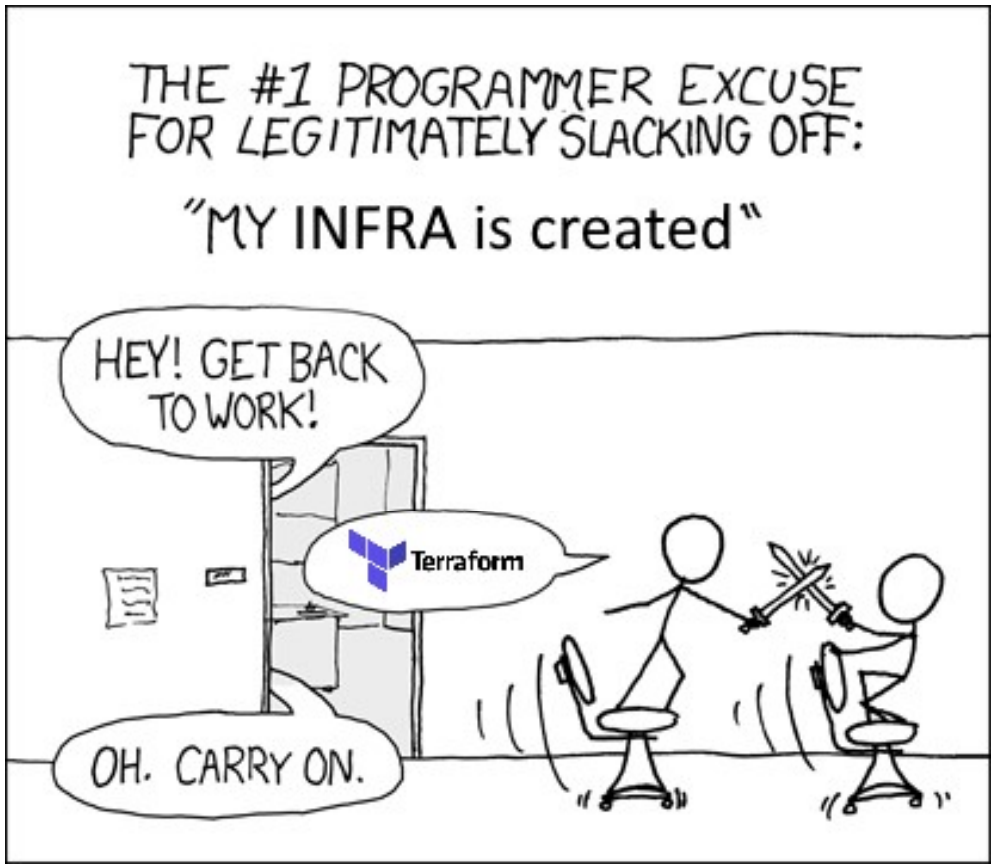
✕ Don't

codefresh

Infrastructure and applications have a different change frequency

1. In most cases applications change 2x-10x more often than infrastructure
2. Deployment of infrastructure/app might take 30 minutes
3. Deployment of application might take 5 minutes
4. For each app deployment you WASTE 25 minutes

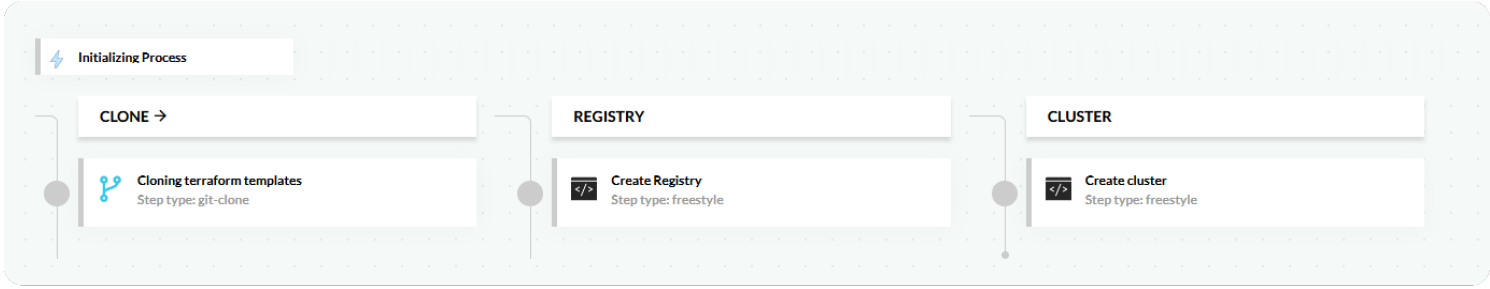
codefresh



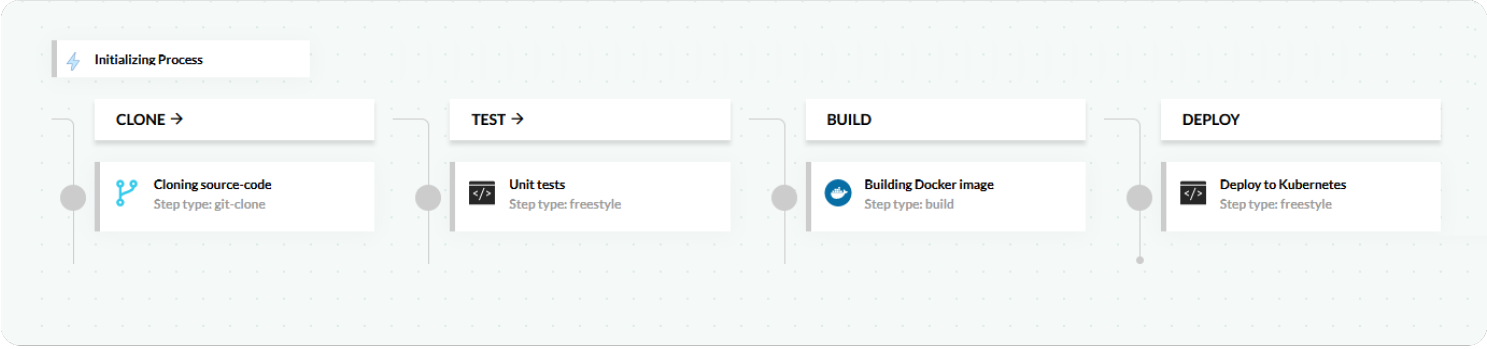
https://xkcd.com/303/

codefresh

Do Infrastructure pipeline, takes 25 minutes runs 3 times a day



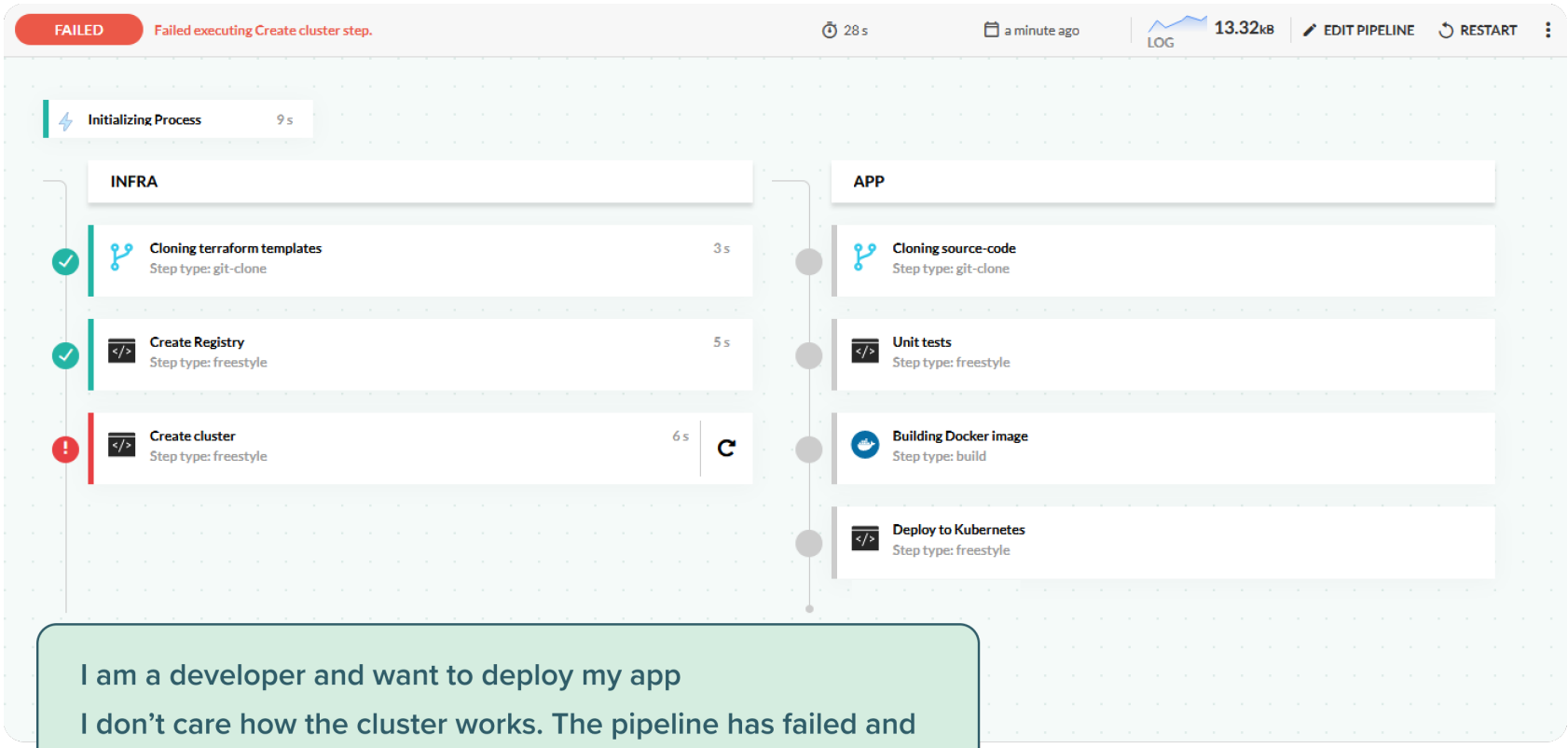
Do Application pipeline, takes 5 minutes and runs 20 times a day



Developers don’t care about infrastructure (and they shouldn’t have to care)

codefresh

Provide Developers with actionable errors in pipelines



I am a developer and want to deploy my app

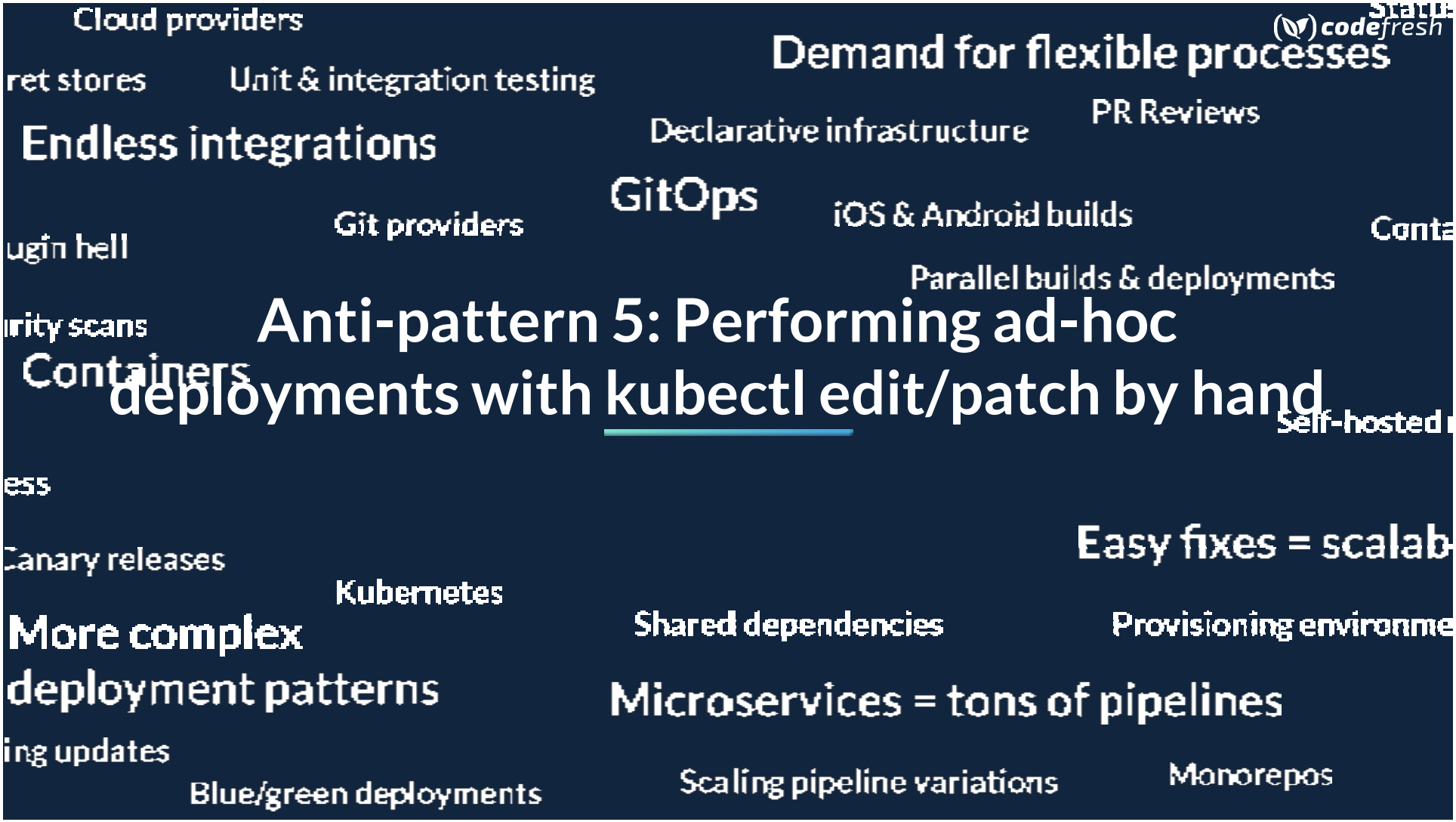
I don't care how the cluster works. The pipeline has failed and my application is not deployed. What do I do?

Applications should be deployed on their own

Infrastructure deployment should be separate

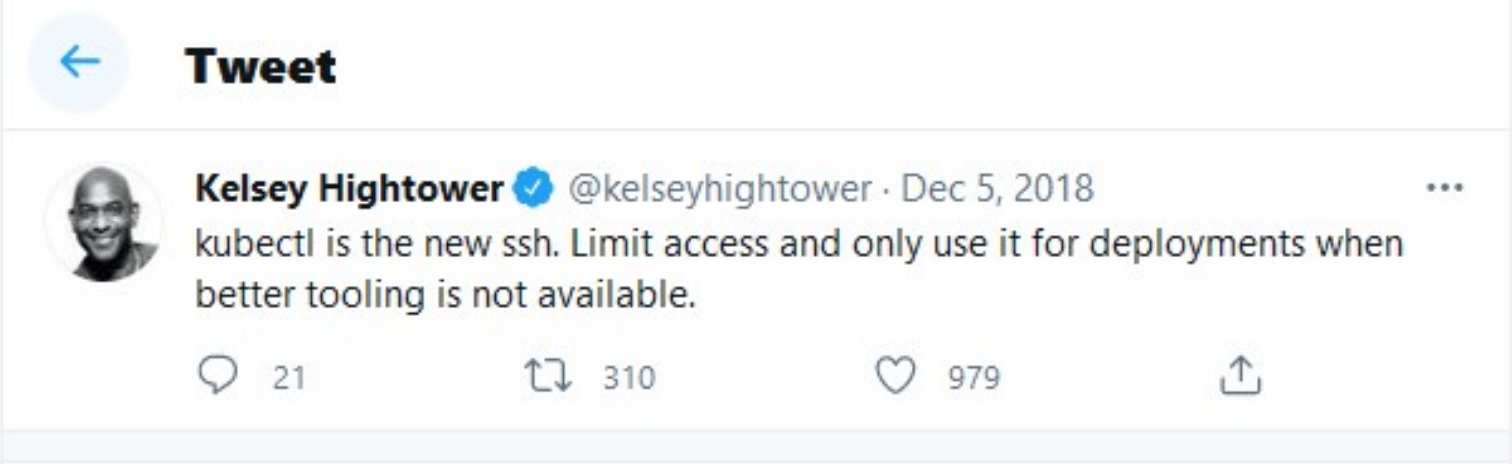
codefresh

Don’t abuse Terraform for application deployments

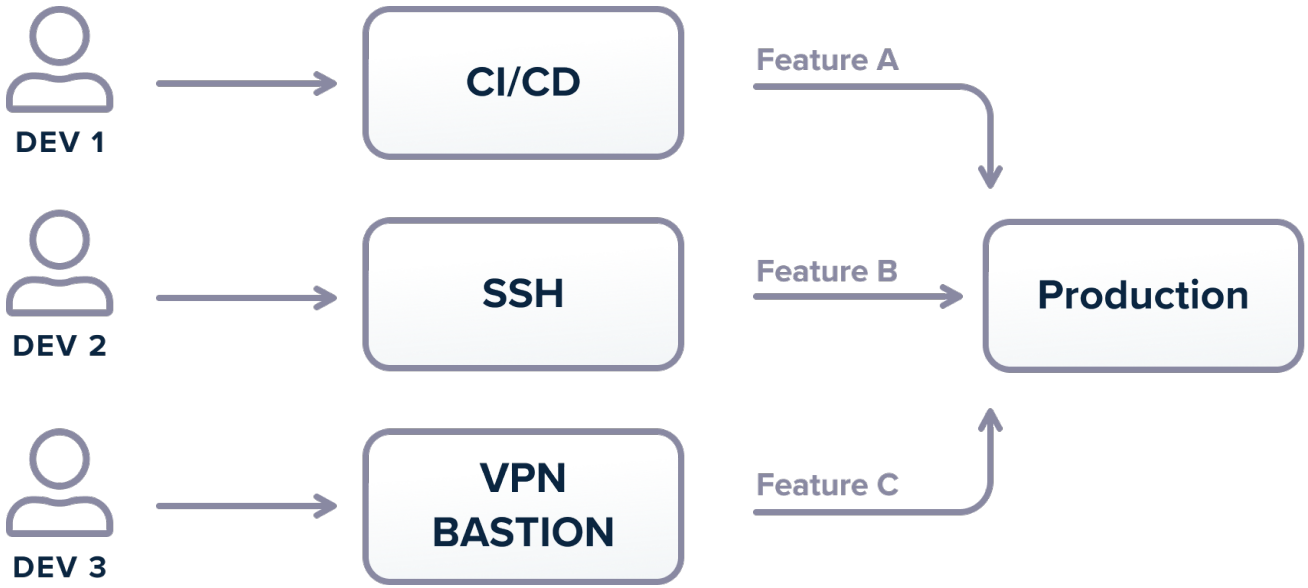

Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
PR Reviews
Endless integrations
Declarative infrastructure
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
Anti-pattern 5: Performing ad-hoc deployments with kubectl edit/patch by hand
rity scans
Containers
ess
Canary releases
Easy fixes = scalab
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos

codefresh

Kubectl is the new SSH

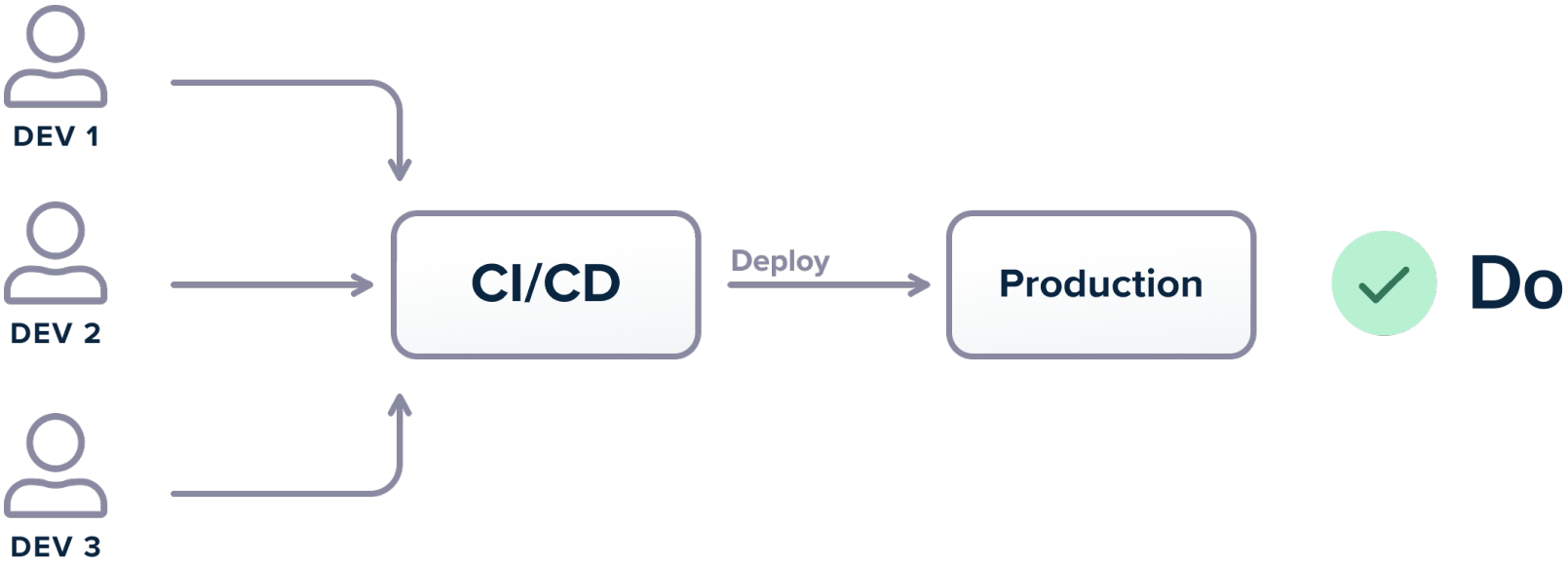


codefresh



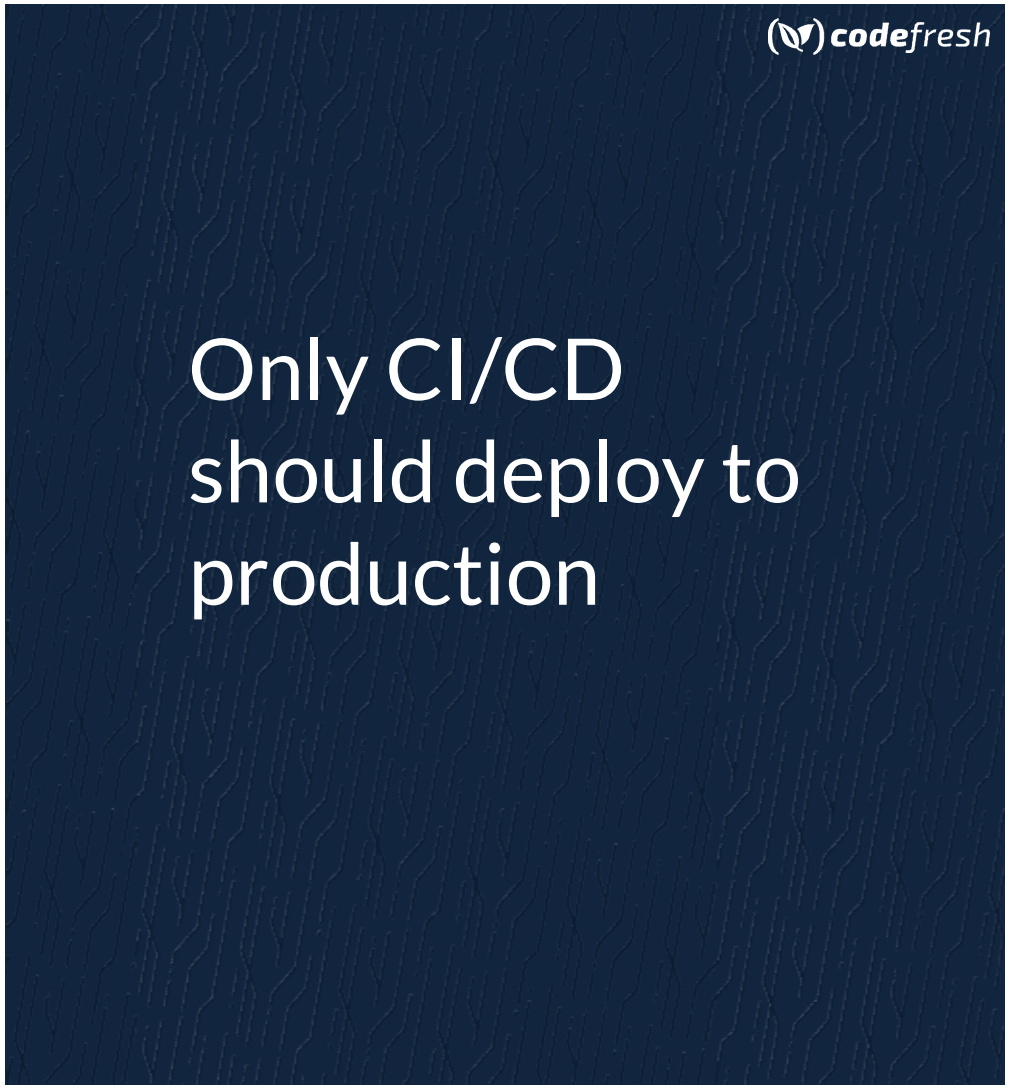
Don't

codefresh



Deploying via SSH was never a good practice

This was true even with VMs



codefresh

3. Access The Argo CD API Server

By default, the Argo CD API server is not exposed with an external IP. To access the API server, choose one of the following techniques to expose the Argo CD API server:

Service Type Load Balancer

Change the argocd-server service type to `LoadBalancer`:

```
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
```

Create Grafana Enterprise configuration

Create a Grafana configuration file with the name `grafana.ini`. Then paste the content below.

> **Note:** You will have to update the `root_url` field to the url associated with the license you were given.

```
[enterprise]
license_path = /etc/grafana/license/license.jwt
[server]
root_url =/your/license/root/url
```

Create Configmap for Grafana Enterprise Config

Create a Kubernetes Configmap from your `grafana.ini` file with the following command:

```
kubectl create configmap ge-config --from-file=/path/to/your/config.ini
```

Edit the file with the command:

```
kubectl edit cm prometheus-server
```

And add this new job:

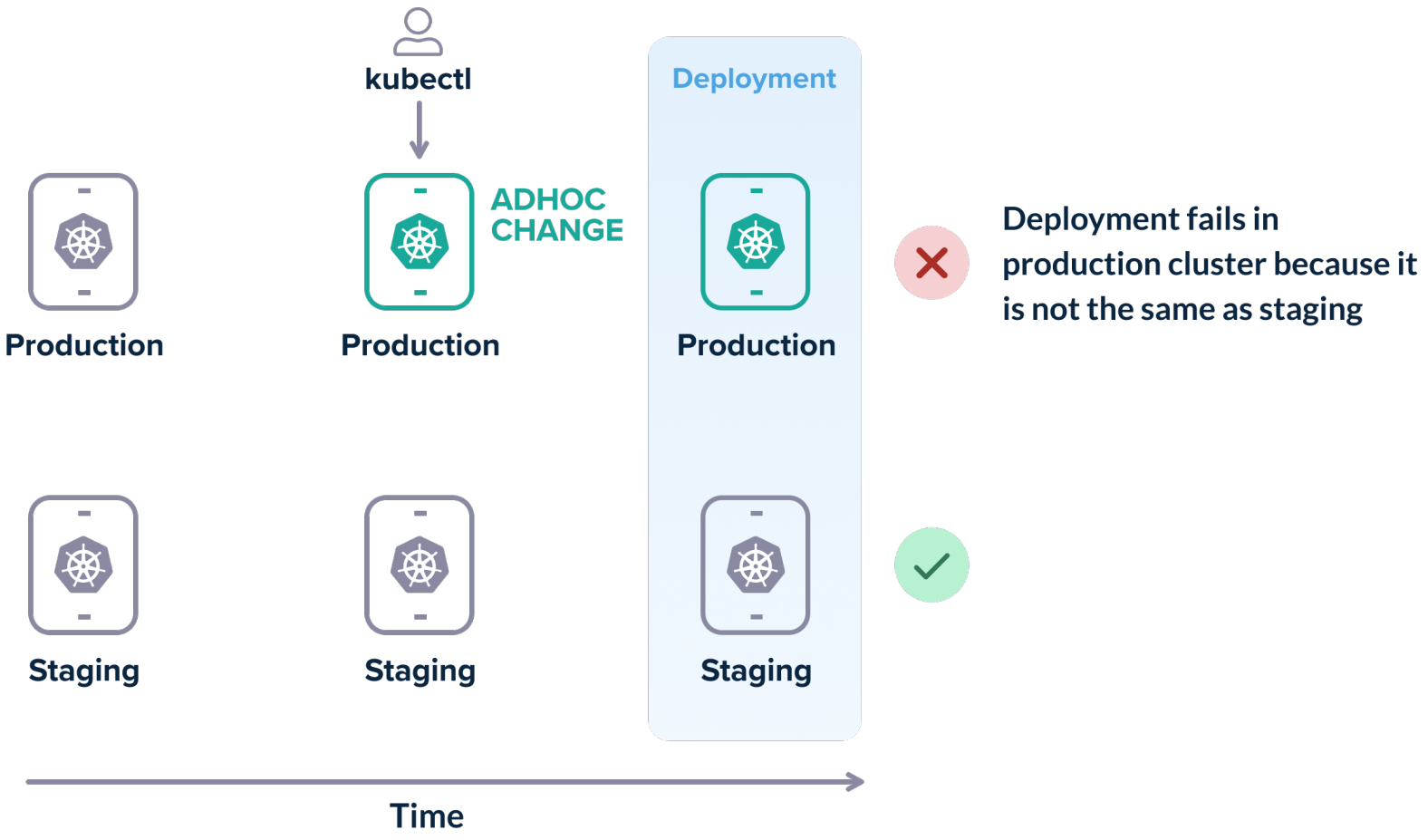
```
- job_name: 'traefik'
  static_configs:
  - targets: ['traefik-prometheus:9100]
```

codefresh

Don’t deploy to production with manual kubectl commands

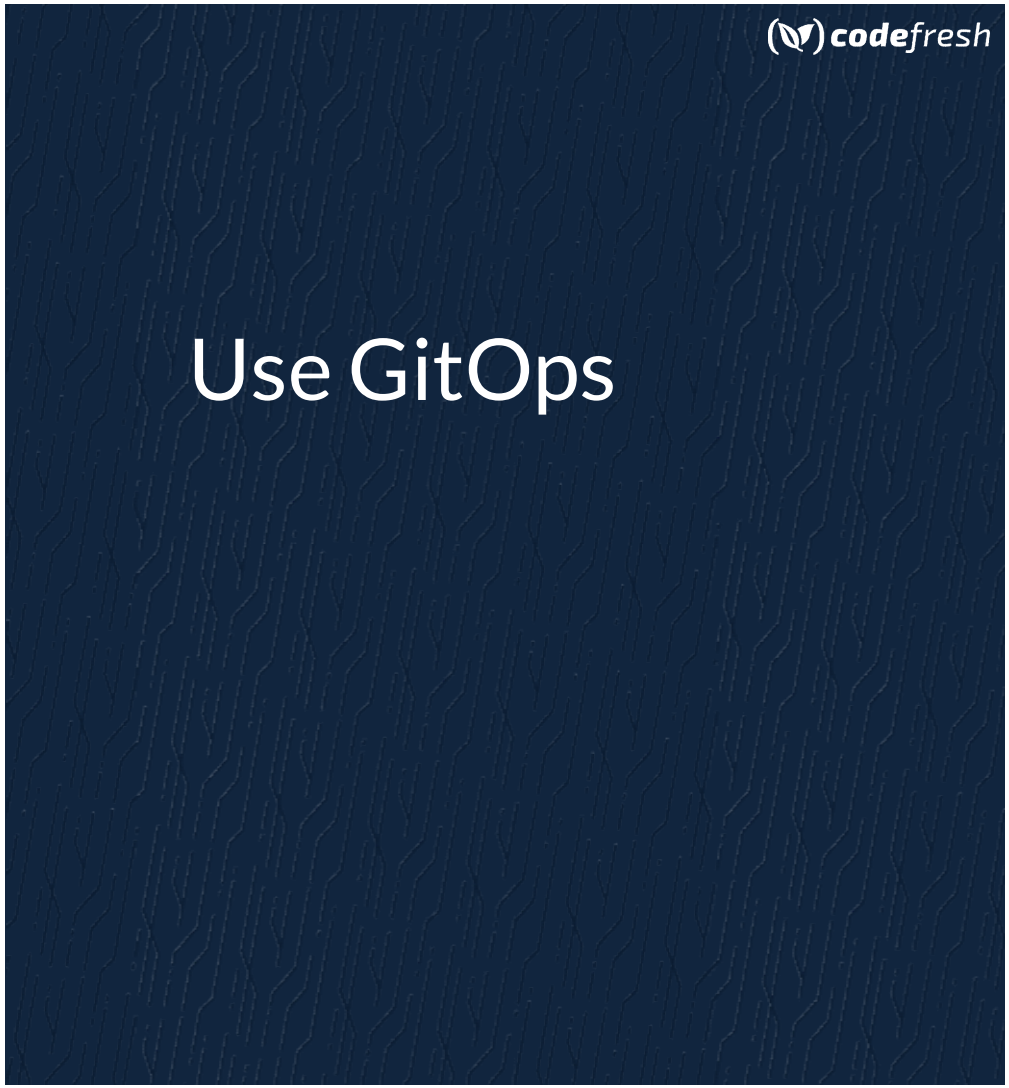
1. Kubectl apply/edit/patch are only for demos and POCs
2. Never change live resources on a cluster
3. You never know what is installed in your cluster
4. Perfect recipe for disaster (configuration drift)

codefresh



Deployment fails in production cluster because it is not the same as staging

Git is the single source of truth. All changes should pass from Git. Change resources by git commit/push



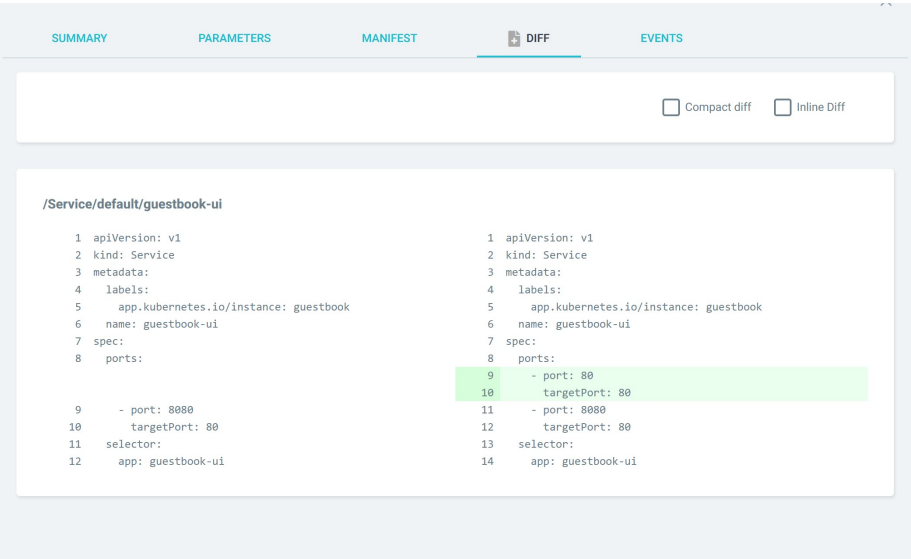
codefresh

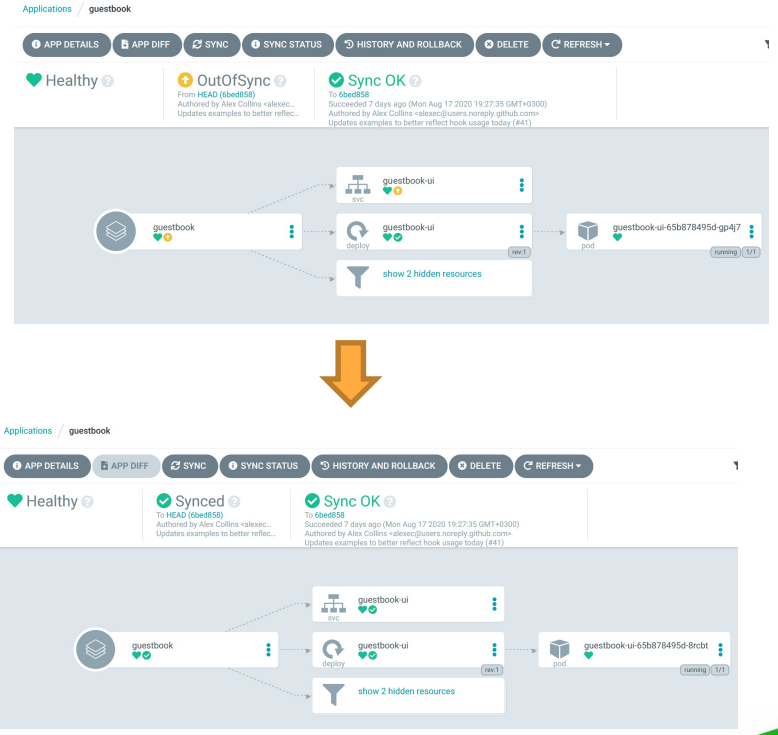
Deploy with a Git commit

1. You know exactly what is in the cluster
2. You have a complete history of what/when/by whom
3. You can create/clone your cluster in minutes
4. Roll back by simply going to a previous commit

codefresh

Avoid configuration drift with GitOps





APPROVED

Avoid manual deployments with SSH

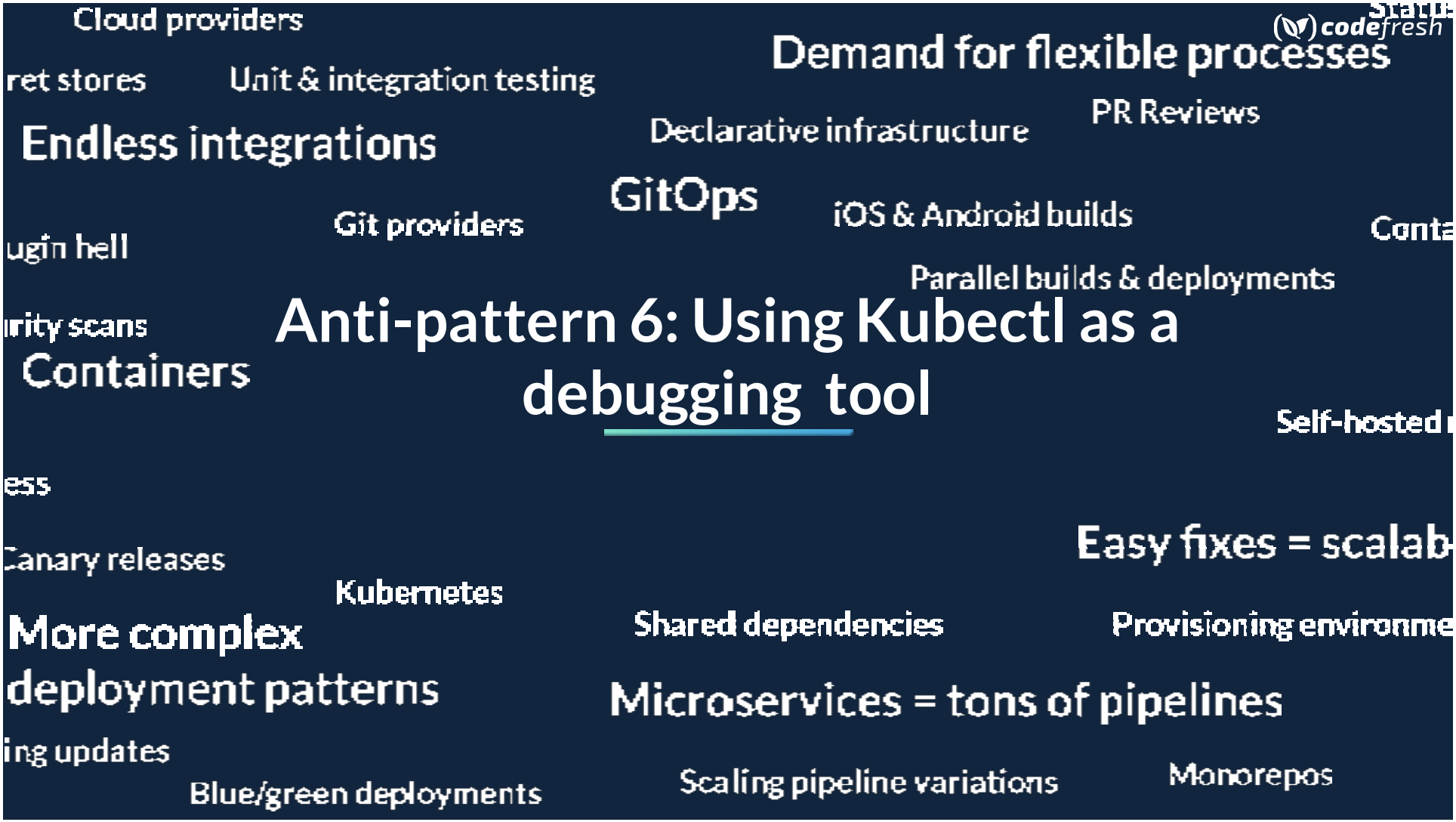
Avoid manual deployments with kubectl

Recap

Top 5 – anti-patterns

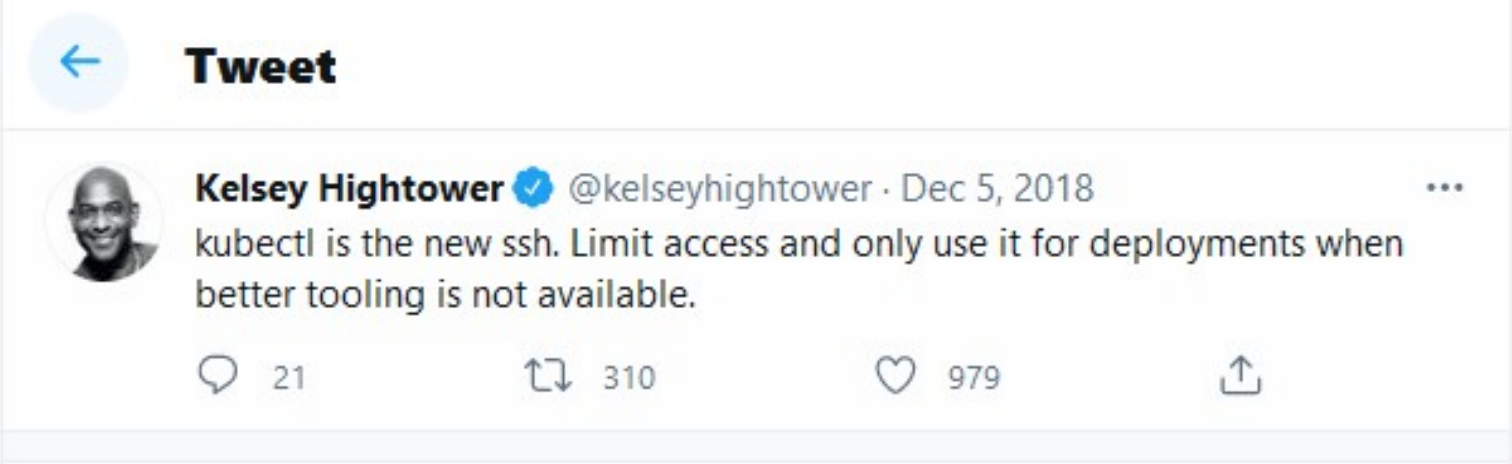
1. Don’t use latest tag. Treat tags as immutable
2. Don’t create different images per environment
3. Don’t couple the application to K8s (or Vault)
4. Don’t mix infrastructure with application deployment
5. Use kubectl apply/patch/edit only for demos/POVs


codefresh
The modern approach to
DevOps automation
Open a FREE account today at codefresh.io


Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
PR Reviews
Endless integrations
Declarative infrastructure
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
Anti-pattern 6: Using Kubectl as a debugging tool
rity scans
Containers
Self-hosted
ess
Easy fixes = scalab
Canary releases
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos

codefresh

Kubectl is the new SSH



You shouldn’t use SSH for debugging VM applications

codefresh

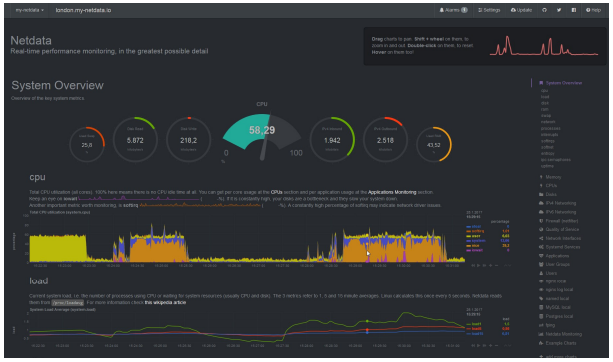
It is 3am. You are getting paged for your “sales” app

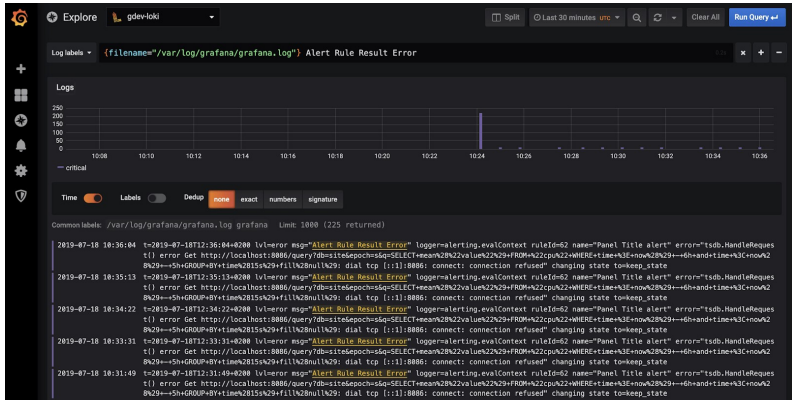
1. Open terminal
2. kubectl get ns
3. kubectl get pods –n sales
4. kubectl describe pod prod-app-123 –n sales
5. kubectl svc –n sales
6. kubectl describe …
7. (more kubectl commands…)

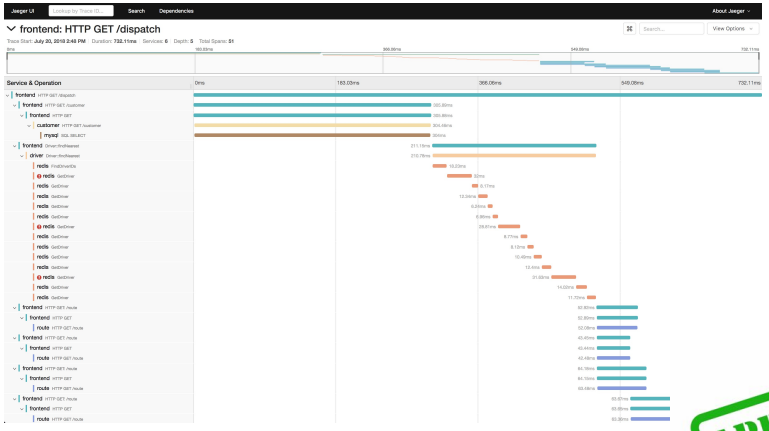
If you need kubectl to inspect something you have a gap in your observability tools

codefresh

Trinity of metrics

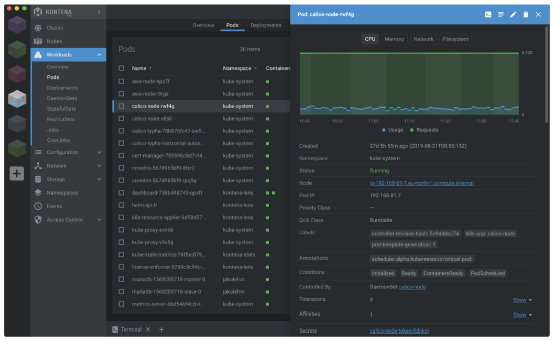


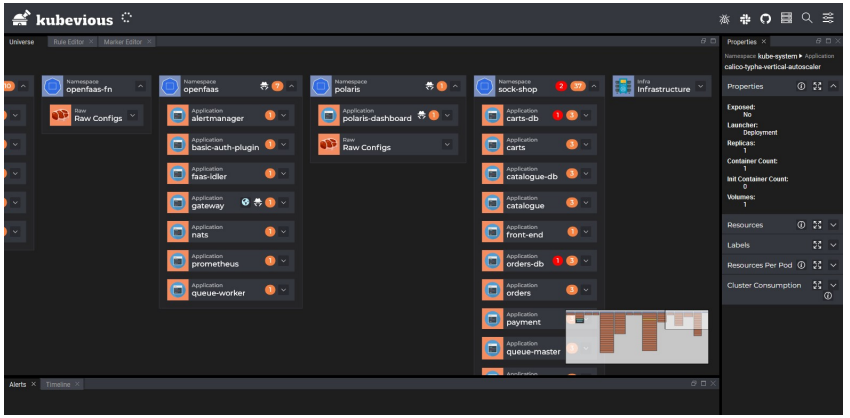


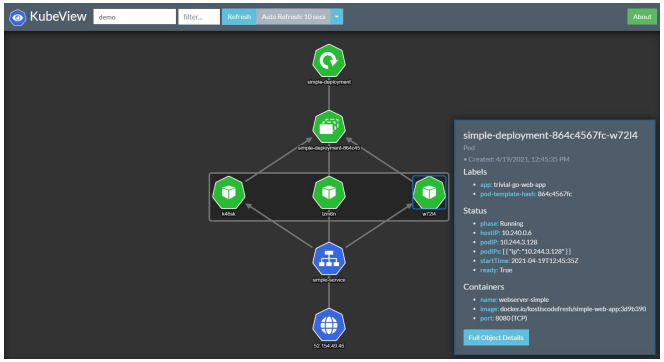


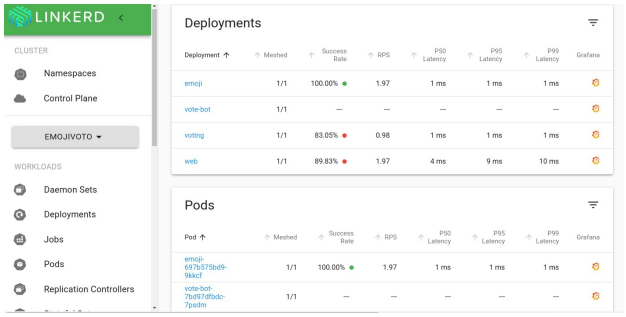
codefresh

General purpose dashboards





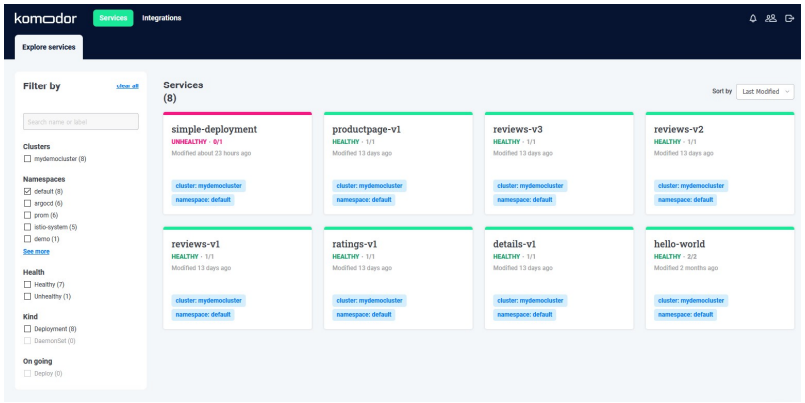


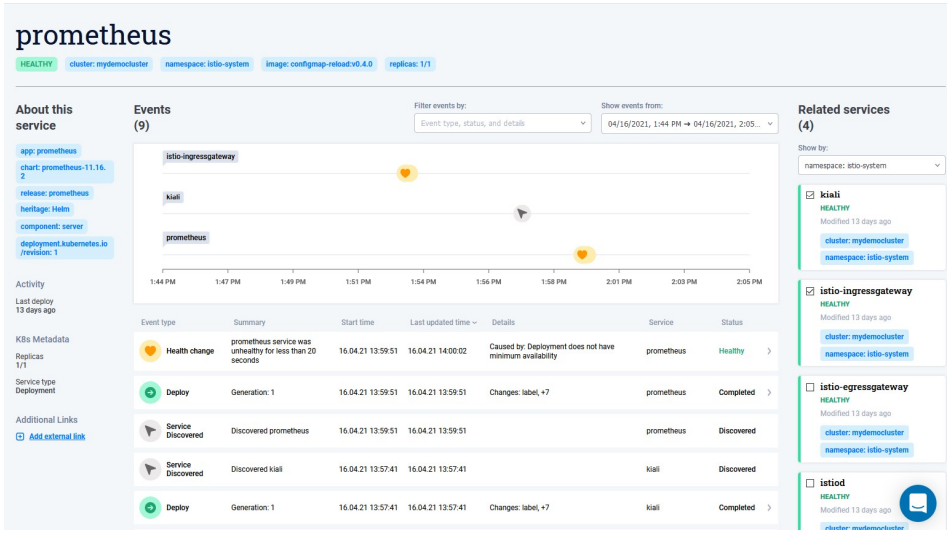


https://codefresh.io/kubernetes-tutorial/kubevious-kubernetes-dashboard/

codefresh

Komodor - Kubernetes troubleshooting





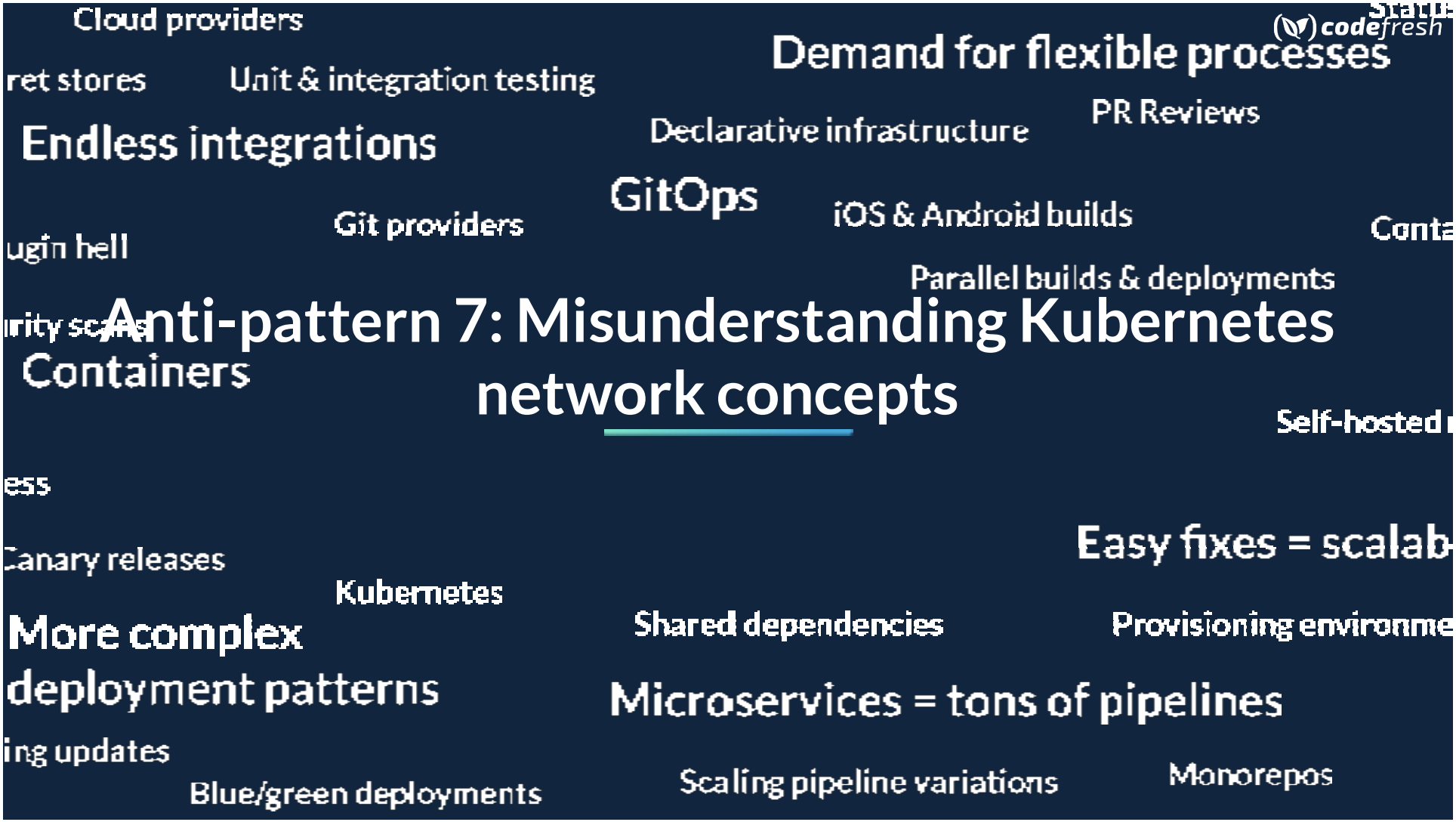
https://codefresh.io/devops/troubleshooting-kubernetes-with-komodor/

Setup metrics and dashboards. Create runbooks

Predict incidents instead of putting out fires

codefresh

Use kubectl as a last resort. After the incident add new metric to your dashboard


Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
Endless integrations
Declarative infrastructure
PR Reviews
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
Containers
Anti-pattern 7: Misunderstanding Kubernetes network concepts
ess
Canary releases
Easy fixes = scalab
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos

VMs:
LoadBalancer
Reverse Proxy

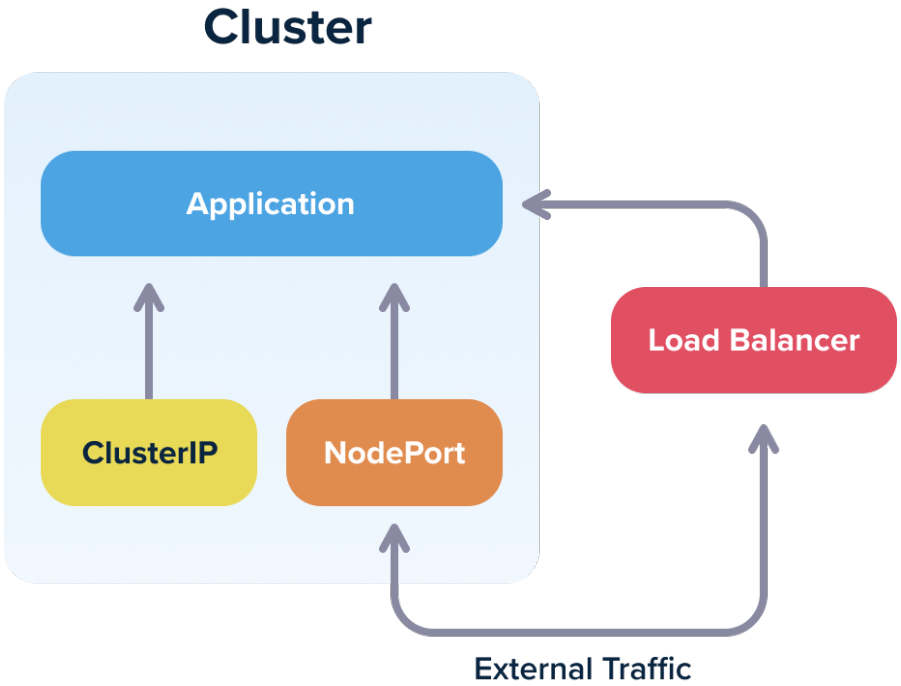
codefresh

Kubernetes:
Service
Load balancer
Ingress
ClusterIP
NodePort
Service Mesh
Endpoint

codefresh

Learn the basics

1. ClusterIP is internal traffic
2. Nodeport is internal/external
3. Loadbalancer is external and also affects billing in cloud installations



codefresh

Learn the network topology

1. Loadbalancer per service (easy but expensive)
2. Single Ingress (cheap but inflexible)
3. Multiple Ingresses (powerful but complex)
4. With or without service mesh
5. With or without API gateway

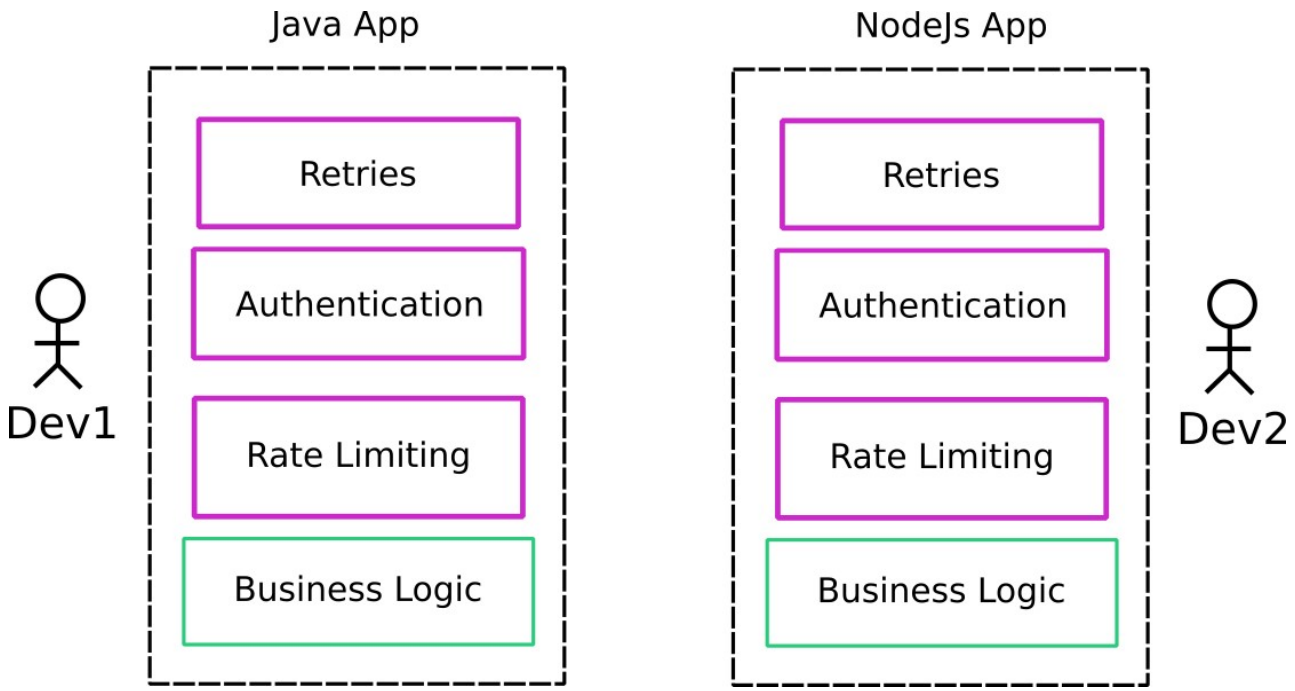
If you are a developer and each microservice has 100ms latency

codefresh

5 hops inside the cluster is 0.5 seconds. Are your customers ready for that?

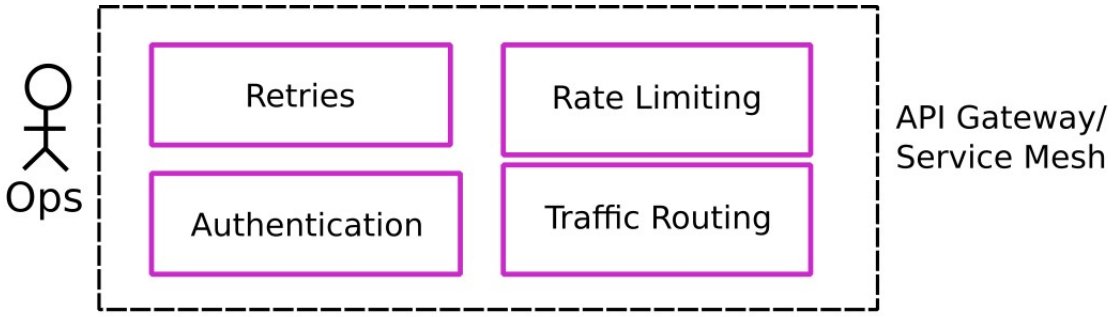
codefresh

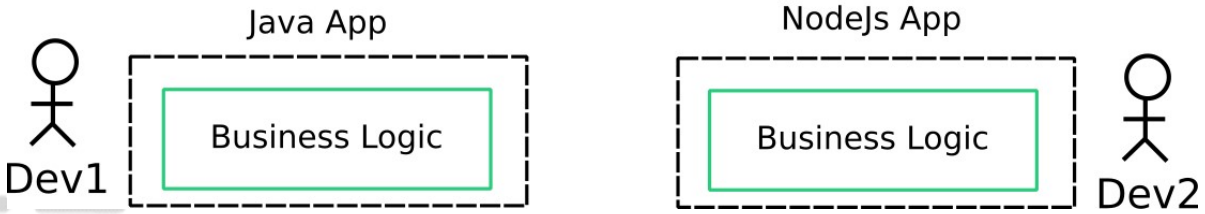
Before Service Mesh/Gateway



codefresh

After Service Mesh/Gateway

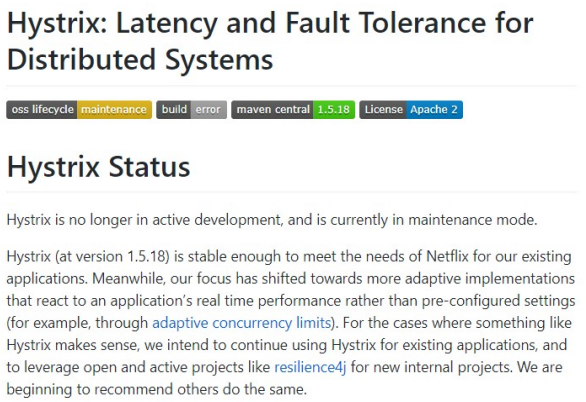




codefresh

Obsolete Programming libraries

1. Service discovery
2. Custom Load balancing
3. Authentication (e.g. oAuth)
4. Rate limiting
5. Retries/timeouts
6. Circuit breakers
7. Utilization metrics
8. Encryption, certificates



Hystrix: Latency and Fault Tolerance for Distributed Systems

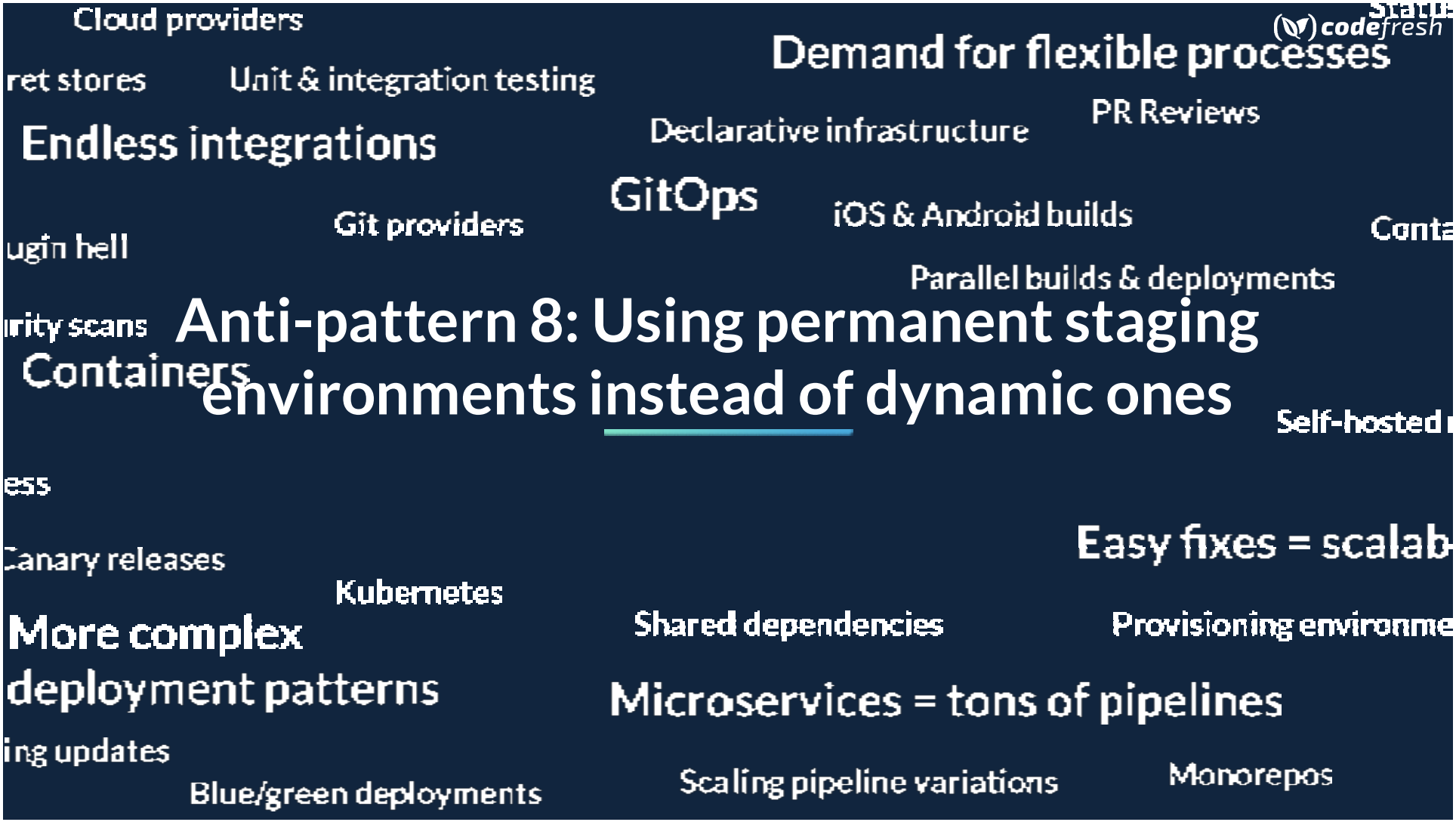
oss lifecycle maintenance | build error | maven central 1.5.18 | License Apache 2

Hystrix Status

Hystrix is no longer in active development, and is currently in maintenance mode.

Hystrix (at version 1.5.18) is stable enough to meet the needs of Netflix for our existing applications. Meanwhile, our focus has shifted towards more adaptive implementations that react to an application's real time performance rather than pre-configured settings (for example, through adaptive concurrency limits). For the cases where something like Hystrix makes sense, we intend to continue using Hystrix for existing applications, and to leverage open and active projects like resilience4j for new internal projects. We are beginning to recommend others do the same.

Understand how traffic reaches your application


Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
Endless integrations
Declarative infrastructure
PR Reviews
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
Anti-pattern 8: Using permanent staging environments instead of dynamic ones
rity scans
Containers
Self-hosted
ess
Easy fixes = scalab
Canary releases
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos

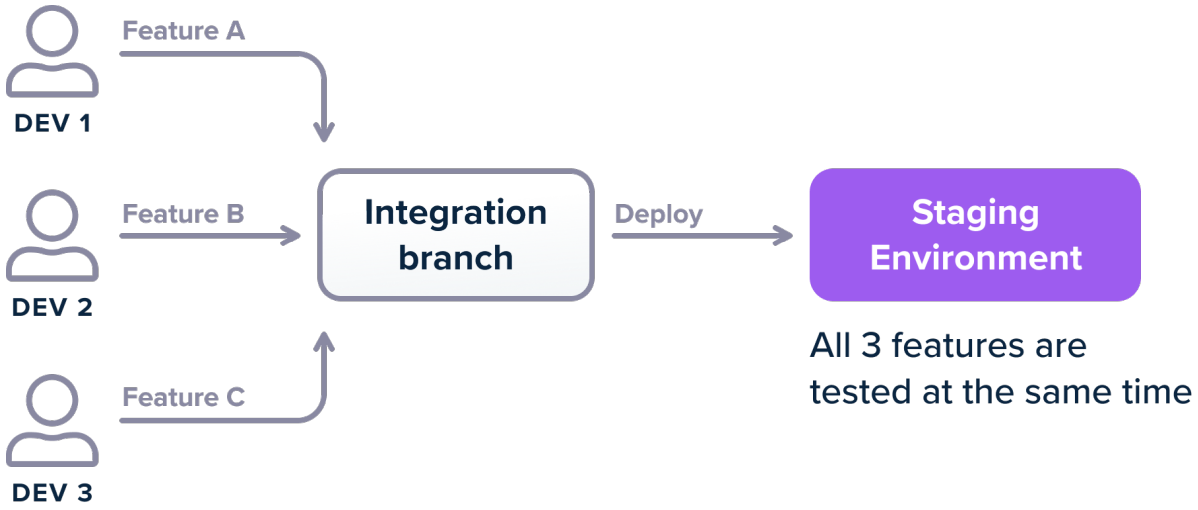
Most companies are still stuck with static environments

codefresh

Adopting Kubernetes impacts testing enviroments like never before

codefresh

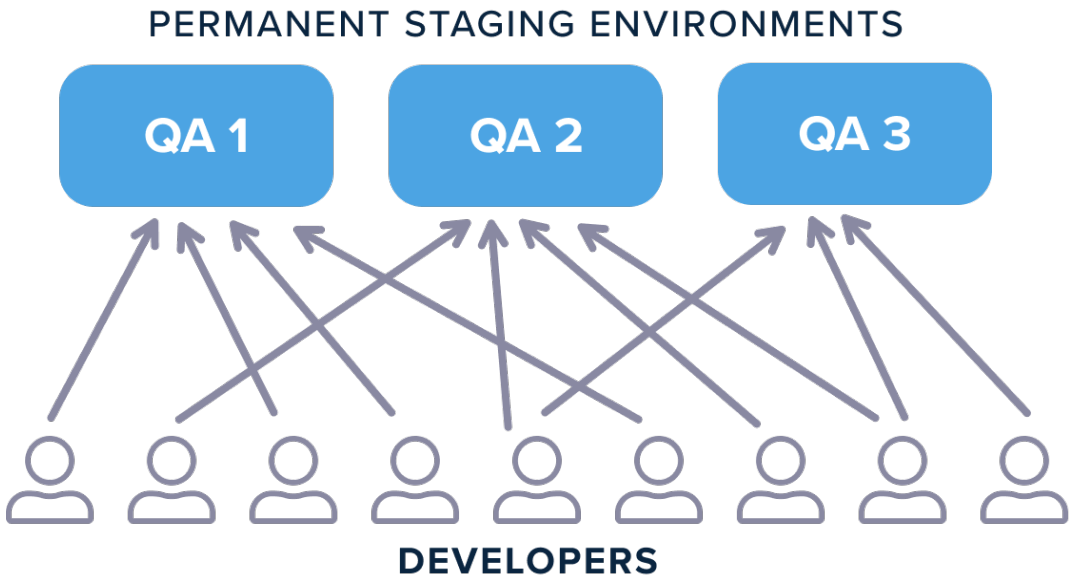
Single staging environment



✕ Don't

codefresh

Multiple staging environments

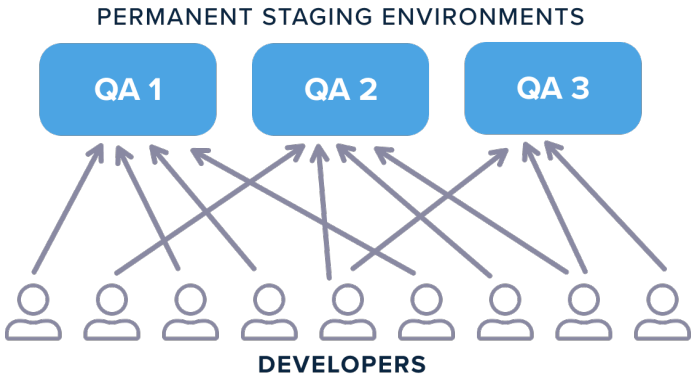


✕ Don't

codefresh

Multiple staging environments

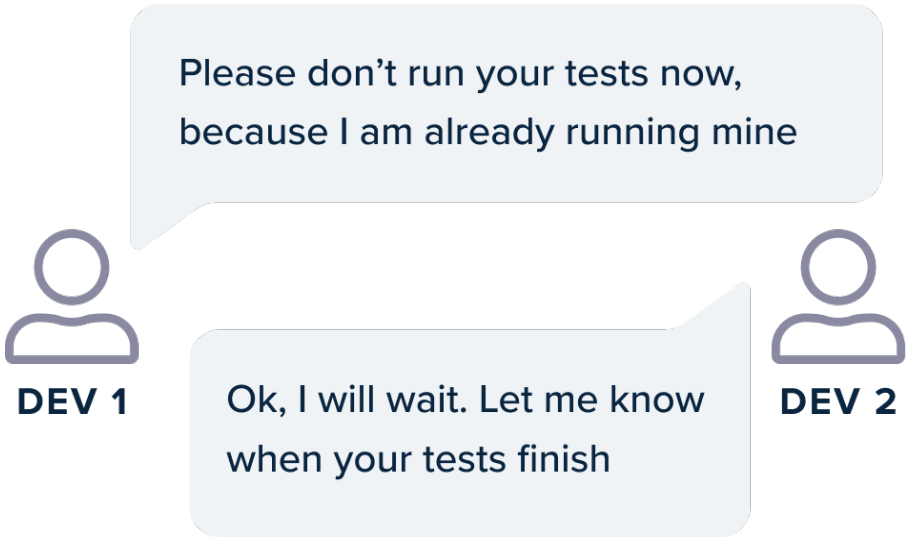
1. Feature conflicts
2. Decreased team velocity
3. Complex clean/setup
4. Wasted resources



✕ Don't

codefresh

Decreased Velocity



Don’t

You pay for resources even when environments are not used

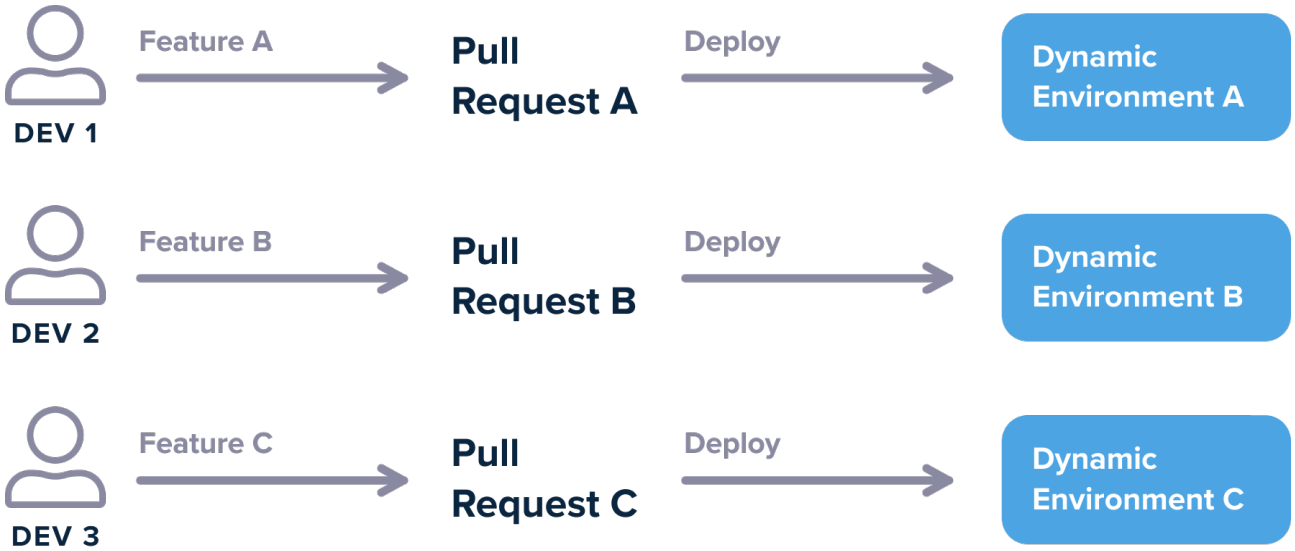
codefresh

Complex cleanup/reset process

Bugs manifest if wrong configuration is present

codefresh

Dynamic environments

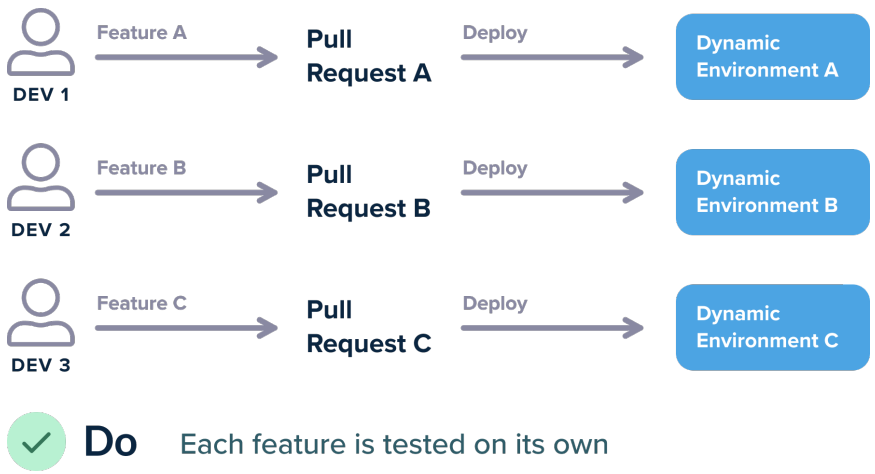


Do Each feature is tested on its own

codefresh

Dynamic environments

1. Feature isolation
2. Better resource utilization
3. Easy cleanup
4. Adapt to any Git Flow

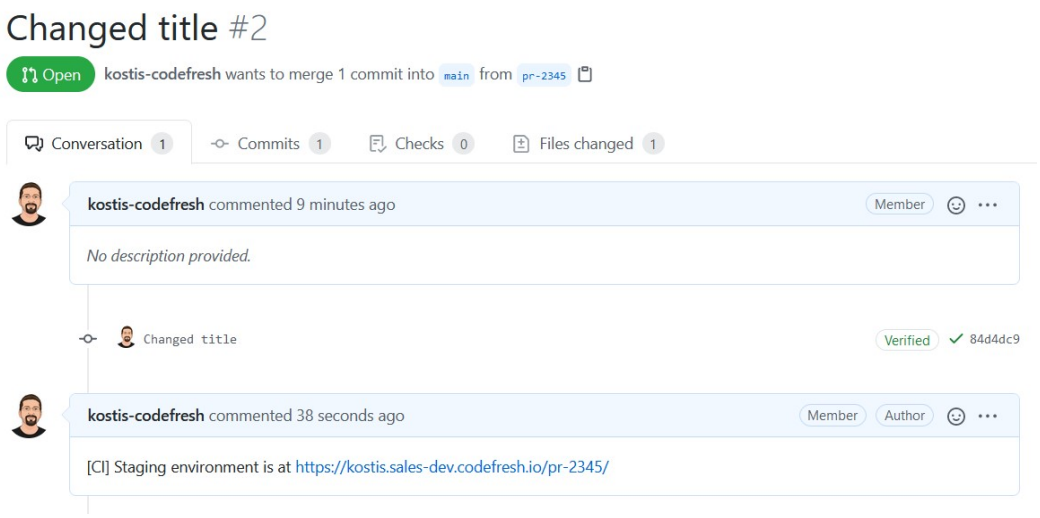


codefresh

Naming patterns (host/path)

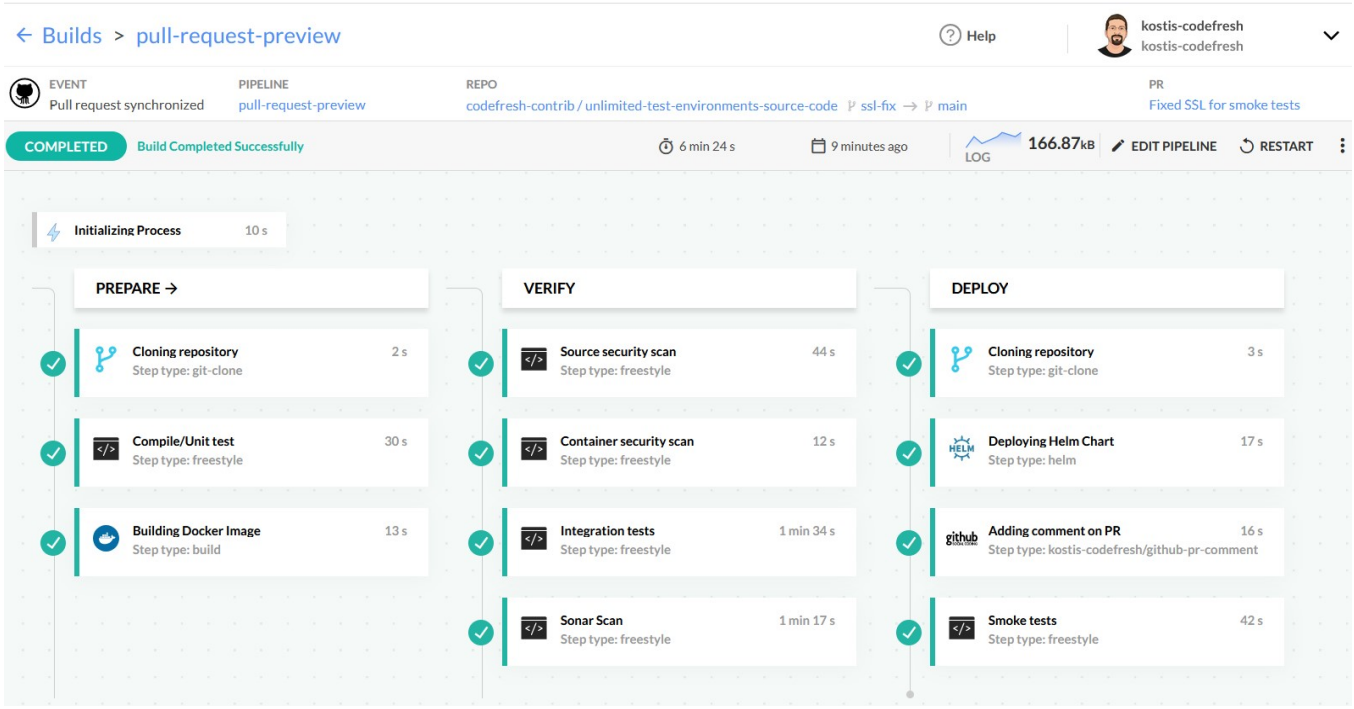
- Pr23 -> pr23.staging.com
- Pr45 -> pr45.staging.com
- Pr39 -> pr39.staging.com

- Pr23 -> staging.com/pr23
- Pr45 -> staging.com/pr45
- Pr39 -> staging.com/pr39



codefresh

Quality gates and smoke tests



https://codefresh.io/docs/docs/ci-cd-guides/preview-environments/

codefresh

Fully automated for devs

1. git checkout master
2. git checkout –b feature-a-b-together
3. git merge feature-a
4. git merge feature-b
5. git push origin feature-a-b-together
6. (open PR in Github)

After some minutes http://staging.example.com/feature-a-b-together is up

Use dynamic environments instead of static ones


Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
PR Reviews
Endless integrations
Declarative infrastructure
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
Anti-pattern 9: Mixing production and non-production clusters
Containers
Self-hosted
ess
Easy fixes = scalab
Canary releases
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos

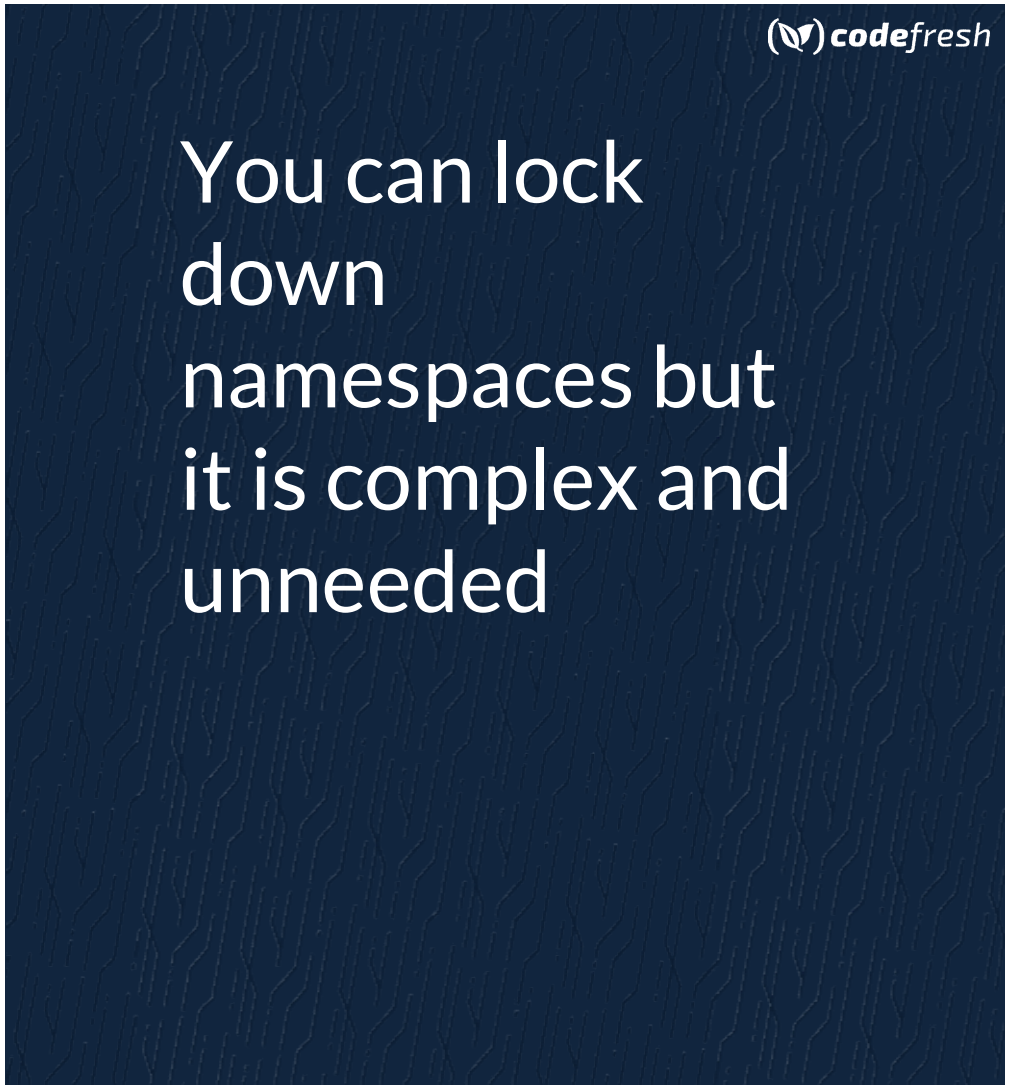
codefresh

Production should be separate

```
→  Kostis kubectl get ns
NAME                   STATUS   AGE
argo-rollouts          Active   91d
argocd                 Active   211d
canary                 Active   34d
dashboard              Active   39d
default                Active   426d
demo                   Active   50d
example                Active   12d
istio-system           Active   53d
komodor                Active   75d
kube-node-lease        Active   426d
kube-public            Active   426d
kube-system            Active   426d
kubenav                Active   39d
kubernetes-dashboard   Active   39d
kubeview               Active   166d
production             Active   10s
prom                   Active   76d
qa                     Active   2s
staging                Active   5s
→  Kostis
```

1. Many tutorials use production/staging as different namespaces
2. Use only for demos/POVs
3. Don’t do this in real projects

Every pod in every namespace can access every other pod in every other namespace



codefresh

Don’t namespace production

1. Resource starvation
2. Cannot easily upgrade cluster
3. Mistakes will happen

```
→  Kostis kubectl get ns
NAME                   STATUS   AGE
argo-rollouts          Active   91d
argocd                 Active   211d
canary                 Active   34d
dashboard              Active   39d
default                Active   426d
demo                   Active   50d
example                Active   12d
istio-system           Active   53d
komodor                Active   75d
kube-node-lease        Active   426d
kube-public            Active   426d
kube-system            Active   426d
kubenav                Active   39d
kubernetes-dashboard   Active   39d
kubeview               Active   166d
production             Active   10s
prom                   Active   76d
qa                     Active   2s
staging                Active   5s
→  Kostis
```

codefresh

Don’t namespace production

1. Developer creates a namespace
2. They deploy feature code and run tests
3. Integrations write dummy data or clean DB
4. A production URL was forgotten inside the code
5. Production DB is destroyed (!!!)

https://unsplash.com/photos/7x18e4cF-nk

codefresh

Suggested clusters

1. Production
2. Shadow/Clone of production but with less resources
3. Developer cluster for feature testing
4. Specialized cluster for load/security testing
5. Cluster for internal tools (e.g. monitoring)
6. Test Cluster for SREs/sys admins

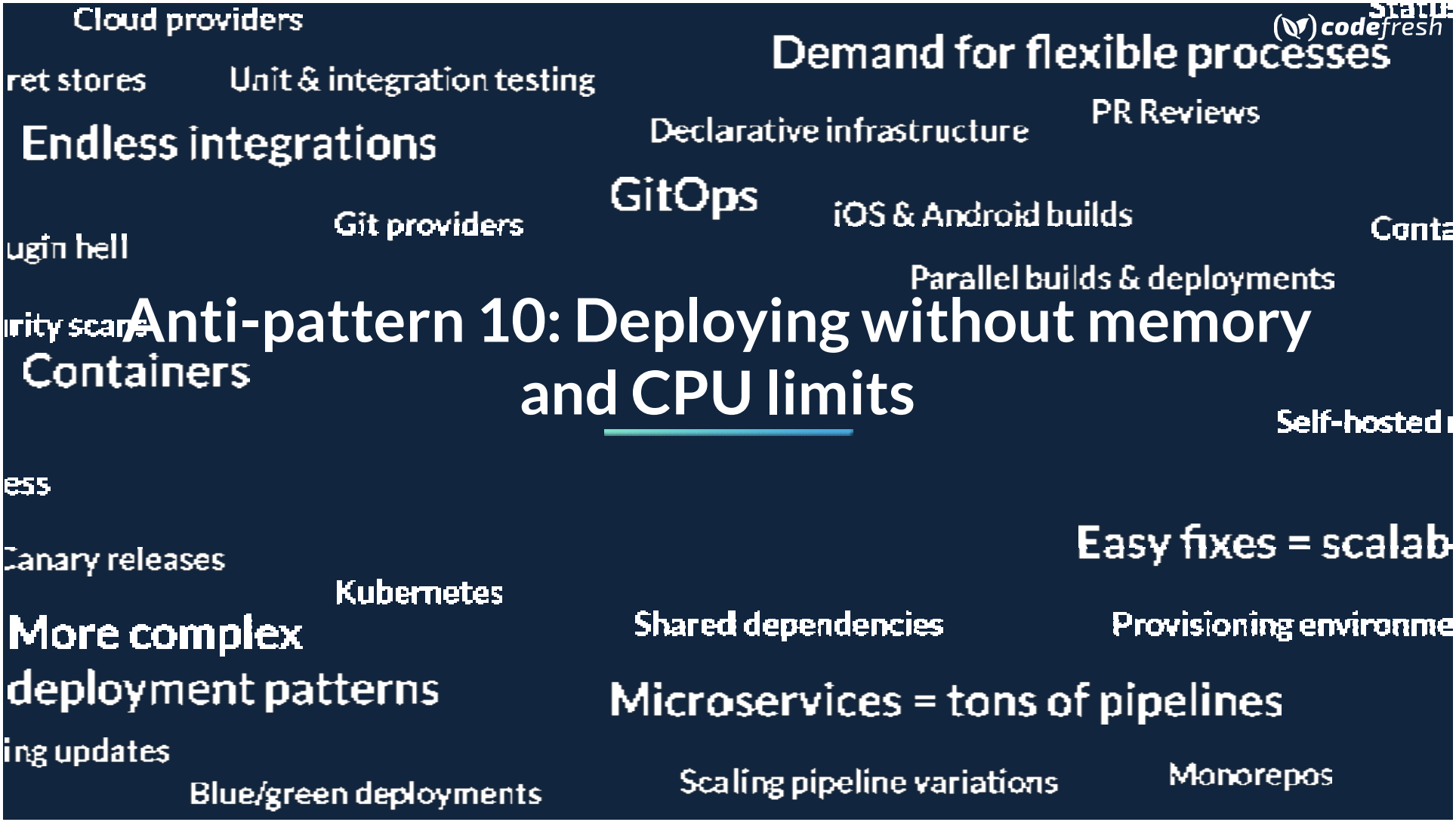
Treat namespaces as soft partitions in the cluster.

Production should run on its own cluster

codefresh

Treating namespaces as a security measure is a recipe for disaster

Use at least 2 clusters (one is prod)


Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
PR Reviews
Endless integrations
Declarative infrastructure
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
Anti-pattern 10: Deploying without memory and CPU limits
Containers
Self-hosted
ess
Easy fixes = scalab
Canary releases
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos

By default an application deployed on Kubernetes has no resource limits

codefresh

This means that a single rogue application can overwhelm the whole cluster

As a developer you need to give some hints to the Kubernetes admin for resource consumption

codefresh

As an operator you need to make sure that all applications have limits (and monitor them)

codefresh

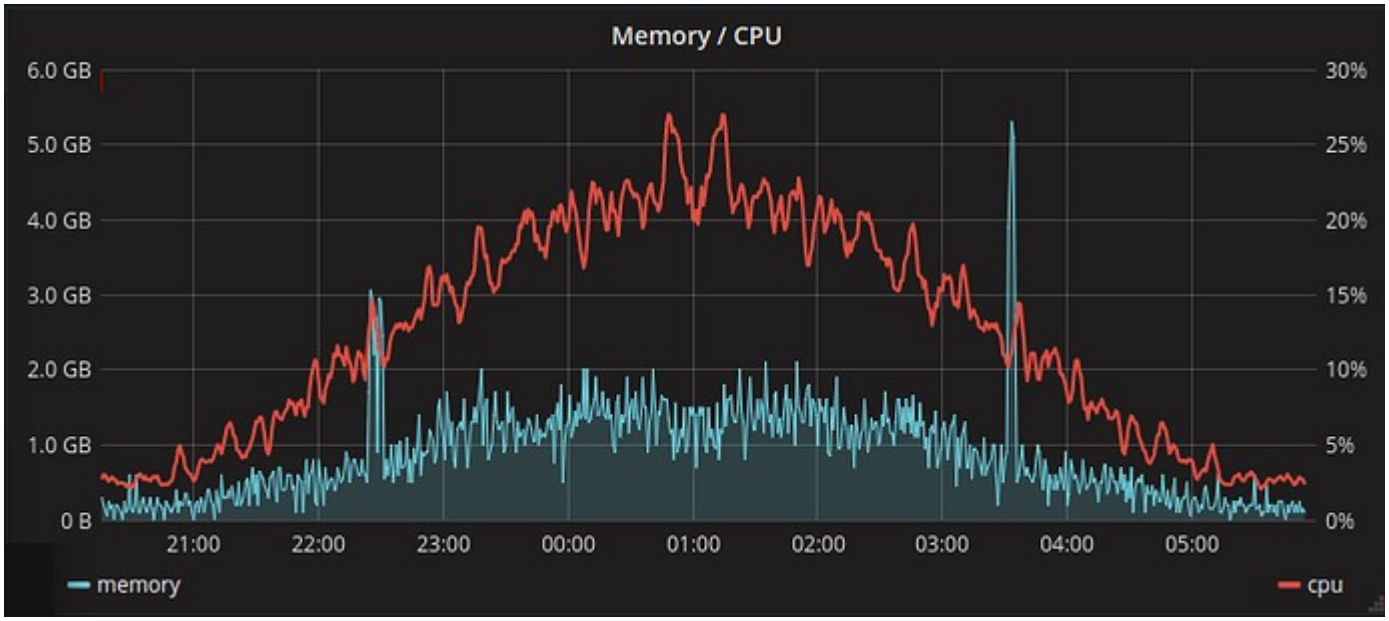
Setting resource limits

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

codefresh

Don’t use the average



codefresh

How to define correct limits

1. Average consumption is “just average”
2. Take into account traffic bursts
3. Perform load testing
4. Learn about minimum/average/maximum
5. Fix your memory leaks ☺
6. Start with a guess and iterate on it (using metrics)
7. Check your programming language documentation

If you put large values you are wasting resources and increase your bills

codefresh

If you put small values, your application performance will suffer (and the cluster will possibly kill your app)

codefresh

Use your metrics

codefresh

Take it to the next level

https://unsplash.com/photos/pqHRNS8Mojc

codefresh

Cloud advantages

Embrace autoscaling

Cluster autoscaling
(increase your nodes)

Horizontal autoscaling
(increase your pods)

Vertical autoscaling
(increase your resource limits)

codefresh

Just watch your apps auto-scale

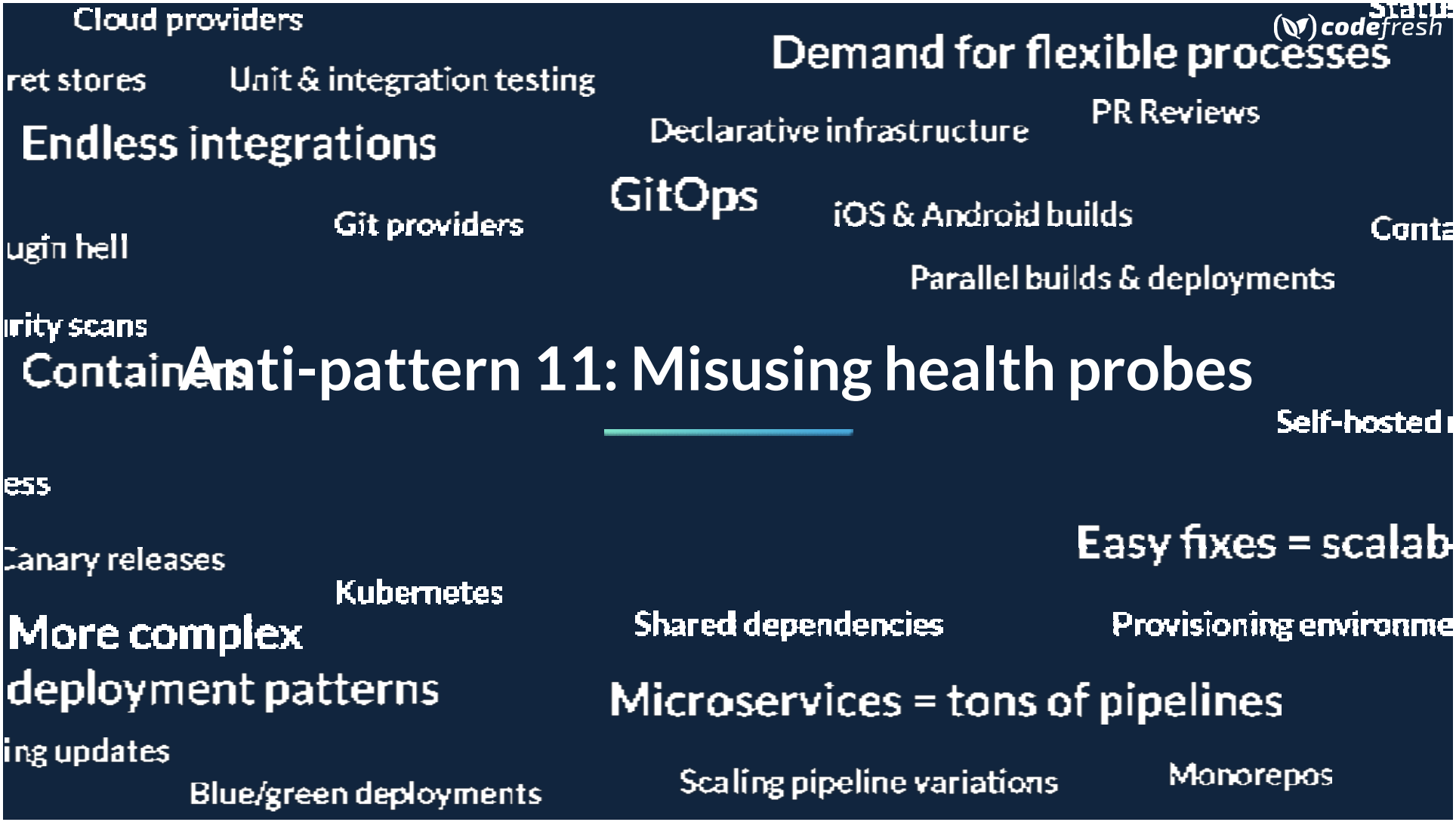
https://unsplash.com/photos/vvLBPW3uS4Q

All applications should have resource limits (even non-prod clusters)

codefresh

Make use of autoscaling facilities

Let your cluster work for you


Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
PR Reviews
Endless integrations
Declarative infrastructure
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
rity scans
Anti-pattern 11: Misusing health probes
Containers
Self-hosted
ess
Easy fixes = scalab
Canary releases
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos

All applications should have resource limits (even non-prod clusters)

codefresh

All applications should have health probes

(some coding required)

codefresh

Health endpoints

Kubernetes queries your app

Startup probe.

Readiness probe.

Liveness probe

codefresh

Setting probe endpoints

- Startup probes
- Readiness probe
- Liveness probe

- Custom command
- Http endpoints
- Tcp port check

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/

codefresh

Learn what the probes do

https://unsplash.com/photos/kBVreEYUzp8

codefresh

Startup probe

1. Runs only once
2. Checks the initial boot of your application
3. Kubernetes will not send traffic to your app
4. Used in combination with liveness probe
5. Mostly for legacy applications

codefresh

Readiness probe

1. Runs all the time
2. Checks if your application can respond to traffic
3. If it fails Kubernetes will stop sending traffic (and try again later)
4. Used when your application needs time to process requests
5. Could also check for external dependencies
6. Should be separate than liveness probe

codefresh

Liveness probe

1. Runs all the time
2. Checks if your application is working (and not deadlocked)
3. If it fails Kubernetes will restart the app
4. Watchdog for stuck/deadlocked applications
5. Should NOT check external depedencies
6. Should be separate than readiness probe

codefresh

Implement the HTTP endpoints

https://unsplash.com/photos/tG36rvCeqng

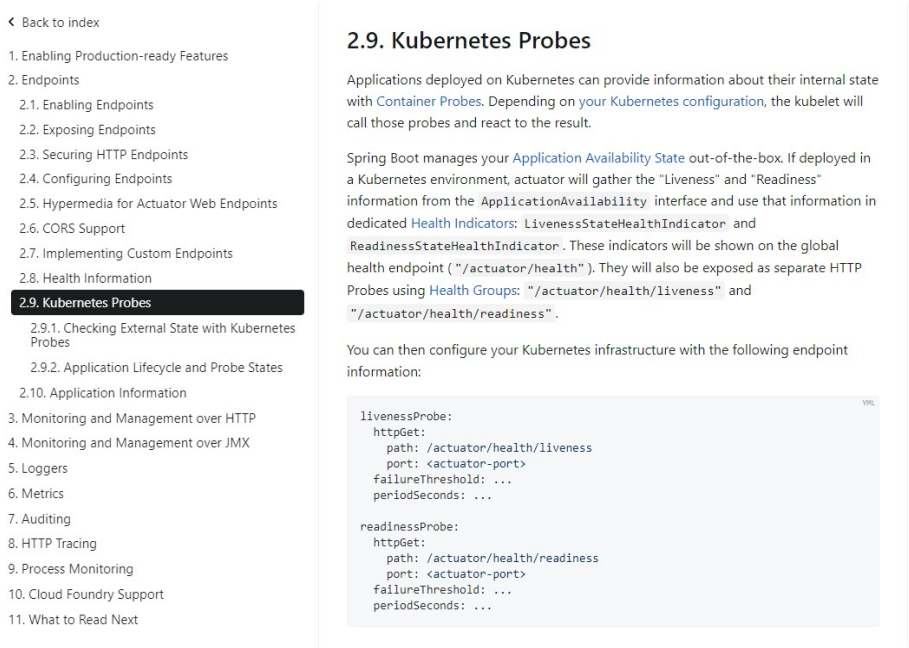
codefresh

Common mistakes

- Not accounting for external services (e.g. DB) in the readiness probe
- Using the same endpoint for readiness and liveness
- Using the existing health endpoint that was created for a Virtual machine
- Not using the Health facilities of your framework
- Creating too complex healthchecks that cause denials of service
- Creating cascading failures (external services in liveness probe)

codefresh

Check your programming framework



Back to index

1. Enabling Production-ready Features
2. Endpoints
   - 2.1. Enabling Endpoints
   - 2.2. Exposing Endpoints
   - 2.3. Securing HTTP Endpoints
   - 2.4. Configuring Endpoints
   - 2.5. Hypermedia for Actuator Web Endpoints
   - 2.6. CORS Support
   - 2.7. Implementing Custom Endpoints
   - 2.8. Health Information
   - **2.9. Kubernetes Probes**
     - 2.9.1. Checking External State with Kubernetes Probes
     - 2.9.2. Application Lifecycle and Probe States
   - 2.10. Application Information
3. Monitoring and Management over HTTP
4. Monitoring and Management over JMX
5. Loggers
6. Metrics
7. Auditing
8. HTTP Tracing
9. Process Monitoring
10. Cloud Foundry Support
11. What to Read Next

2.9. Kubernetes Probes

Applications deployed on Kubernetes can provide information about their internal state with Container Probes. Depending on your Kubernetes configuration, the kubelet will call those probes and react to the result.

Spring Boot manages your Application Availability State out-of-the-box. If deployed in a Kubernetes environment, actuator will gather the "Liveness" and "Readiness" information from the `ApplicationAvailability` interface and use that information in dedicated Health Indicators: `LivenessStateHealthIndicator` and `ReadinessStateHealthIndicator`. These indicators will be shown on the global health endpoint (`"/actuator/health"`). They will also be exposed as separate HTTP Probes using Health Groups: `"/actuator/health/liveness"` and `"/actuator/health/readiness"`.

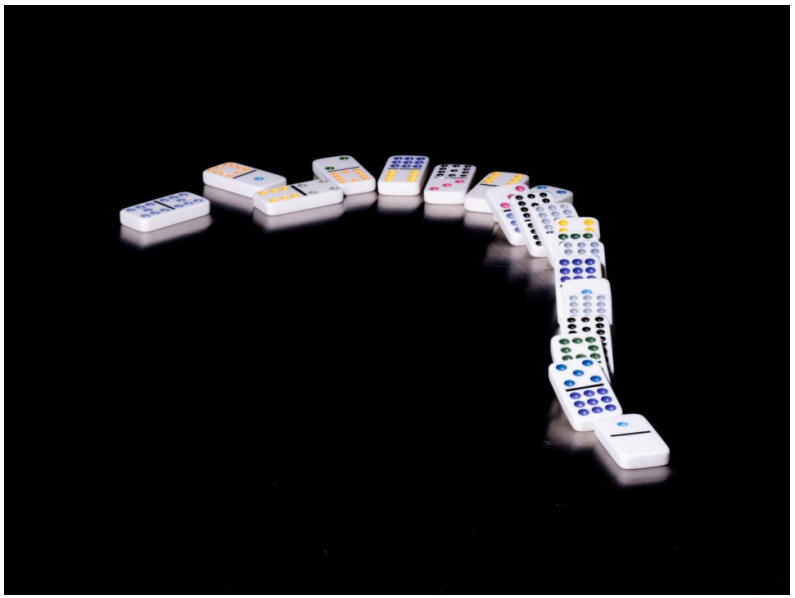
You can then configure your Kubernetes infrastructure with the following endpoint information:

```yaml
livenessProbe:
  httpGet:
    path: /actuator/health/liveness
    port: <actuator-port>
  failureThreshold: ...
  periodSeconds: ...

readinessProbe:
  httpGet:
    path: /actuator/health/readiness
    port: <actuator-port>
  failureThreshold: ...
  periodSeconds: ...
```

https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html

codefresh

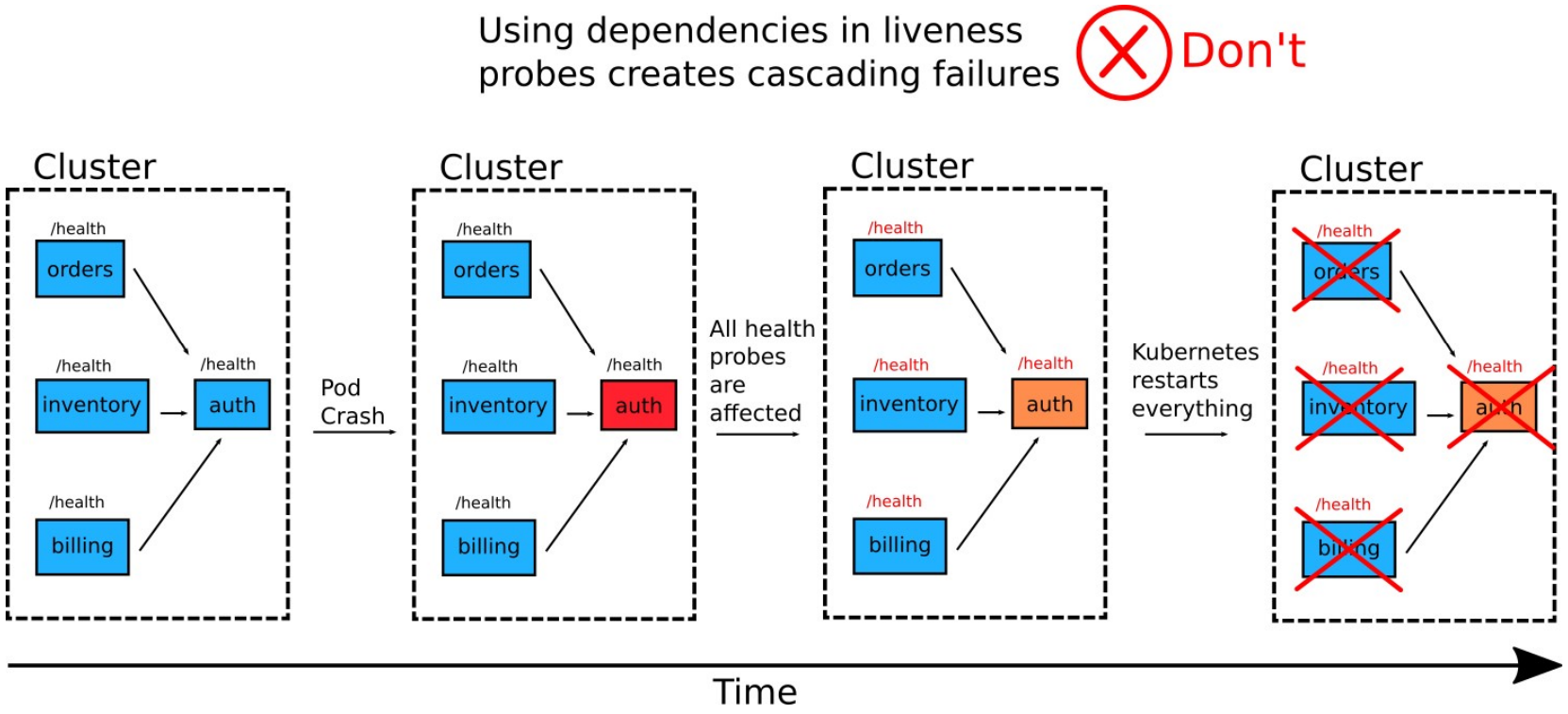
Cascading failures

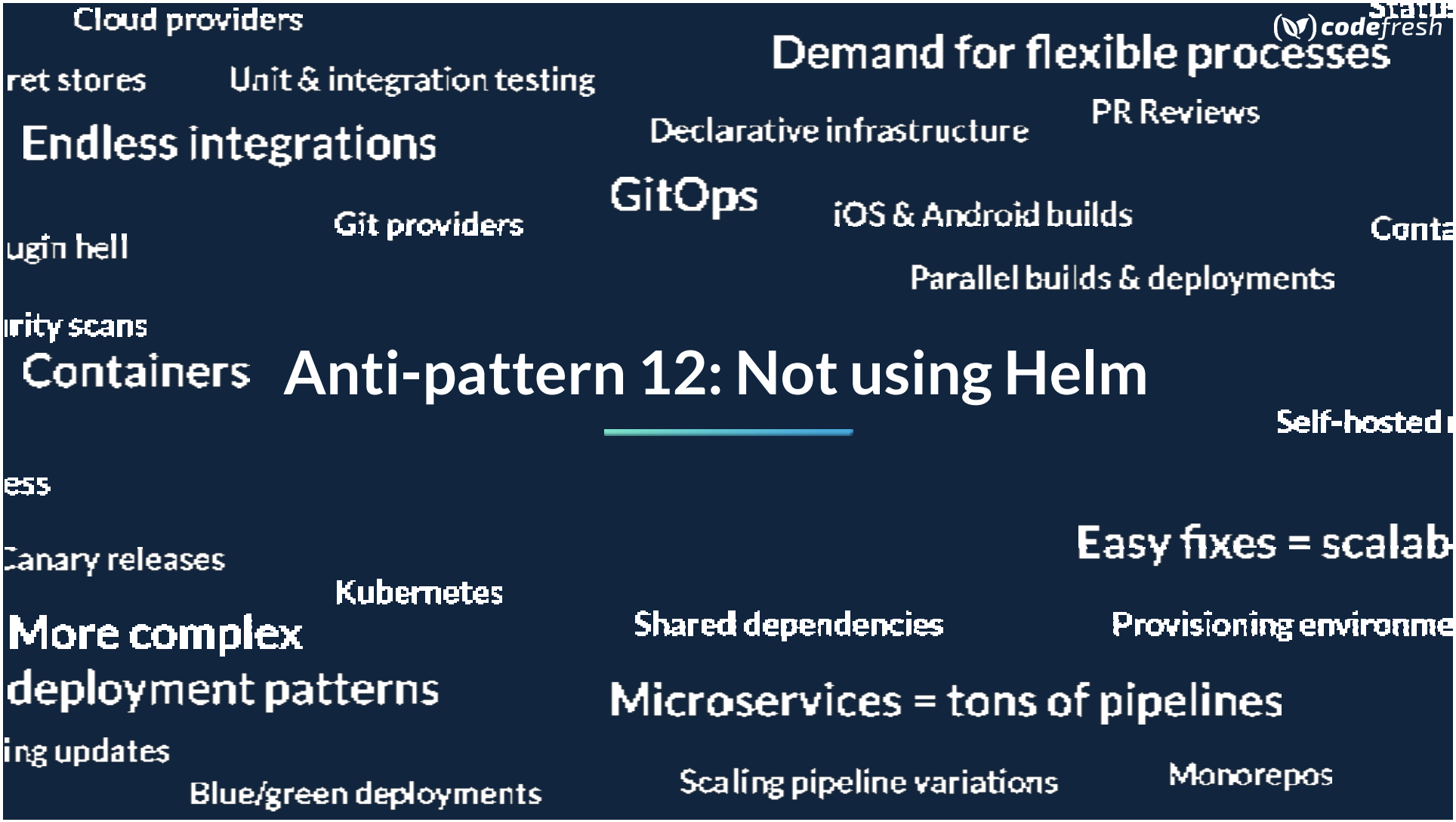


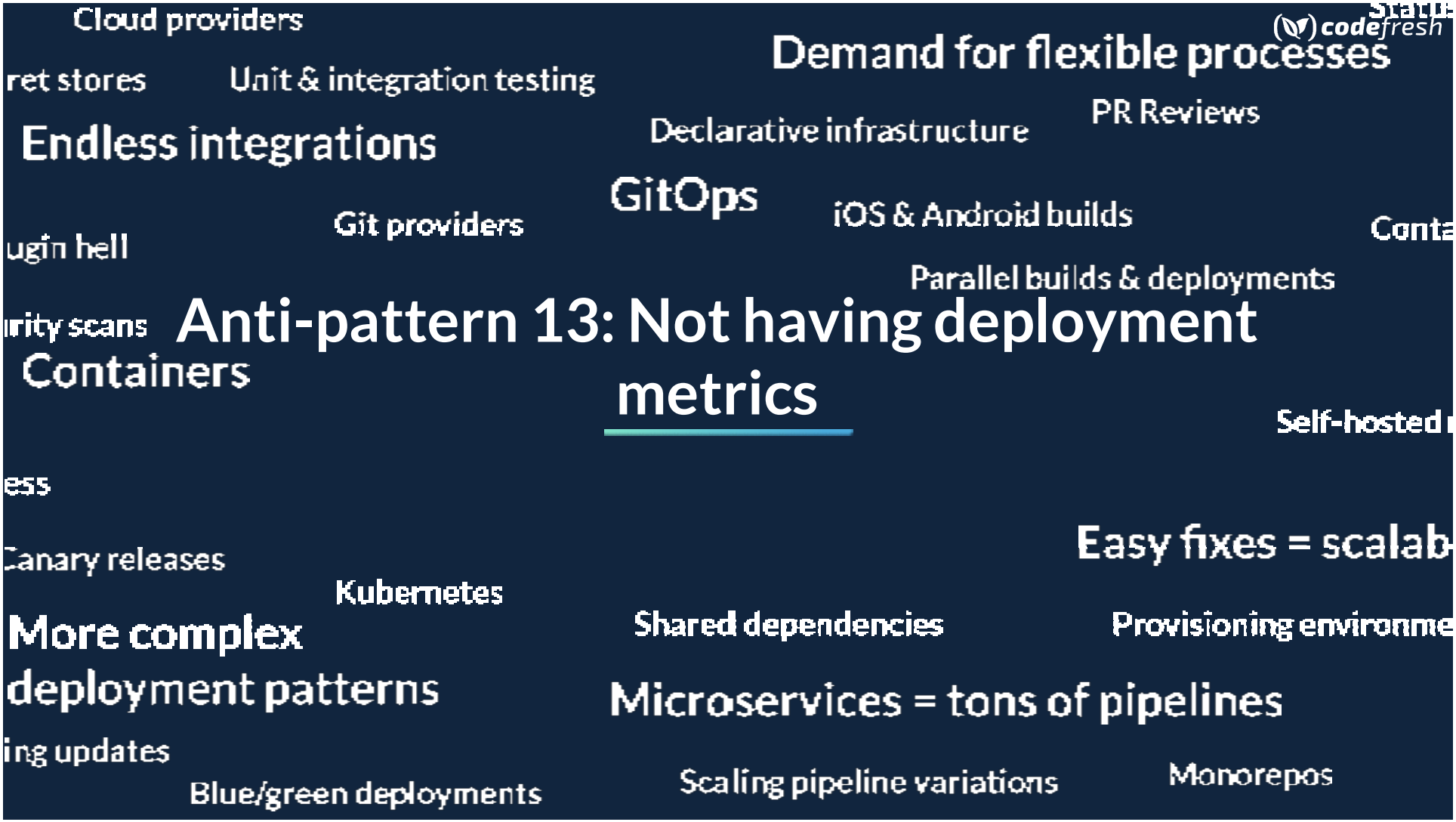
https://unsplash.com/photos/Em2hPK55o8g

codefresh

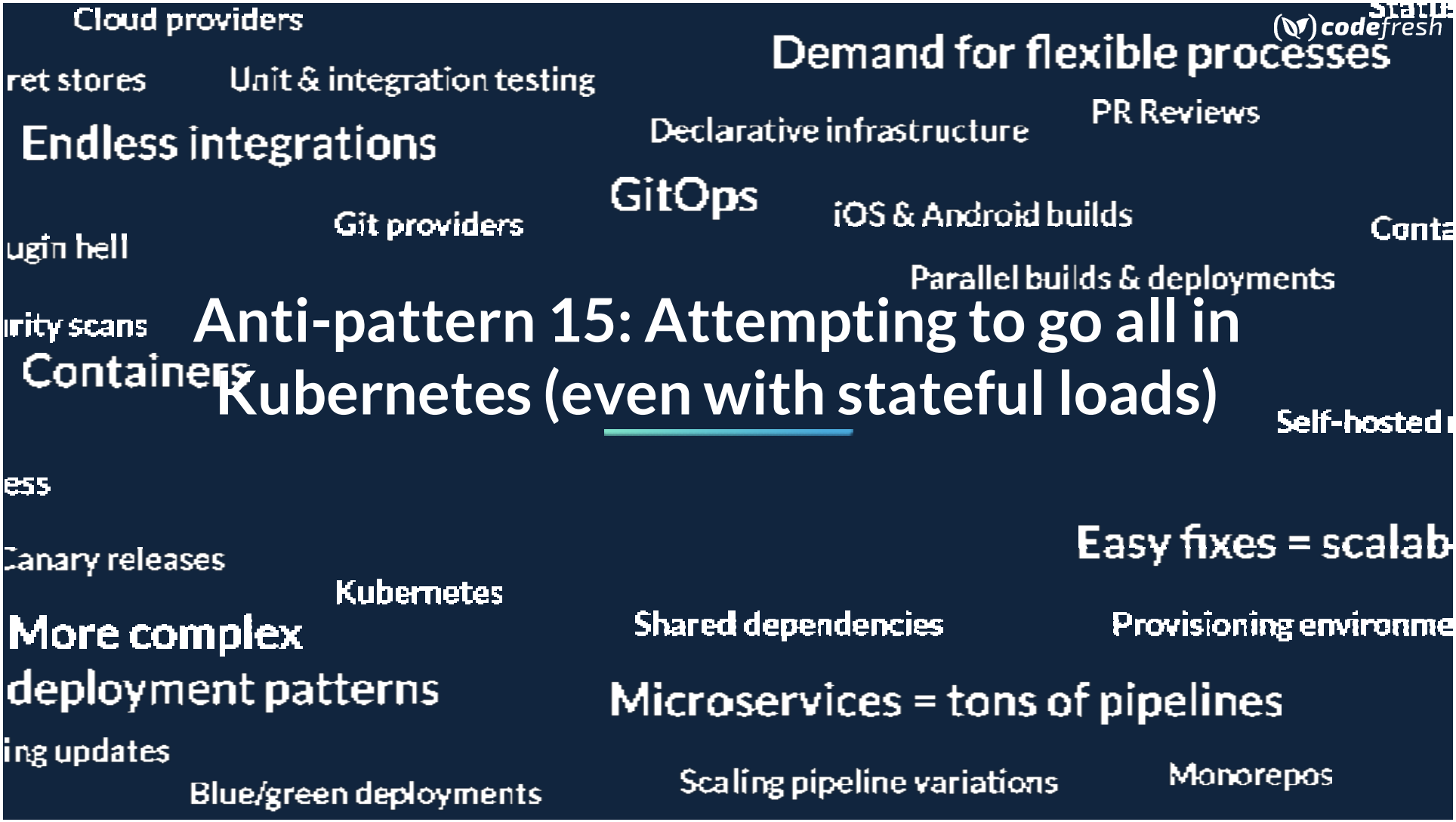
Using dependencies in liveness probes creates cascading failures




Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
PR Reviews
Endless integrations
Declarative infrastructure
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
rity scans
Containers
Anti-pattern 12: Not using Helm
Self-hosted
ess
Easy fixes = scalab
Canary releases
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos


Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
PR Reviews
Endless integrations
Declarative infrastructure
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
rity scans
Anti-pattern 13: Not having deployment metrics
Containers
Self-hosted
ess
Easy fixes = scalab
Canary releases
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos


Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
PR Reviews
Endless integrations
Declarative infrastructure
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
Anti-pattern 14: Not having a strategy for secrets
rity scans
Containers
Self-hosted
ess
Easy fixes = scalab
Canary releases
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos


Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
Endless integrations
Declarative infrastructure
PR Reviews
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
Anti-pattern 15: Attempting to go all in
Containers
Kubernetes (even with stateful loads)
ess
Canary releases
Easy fixes = scalab
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos

codefresh

d

f

d

f

f

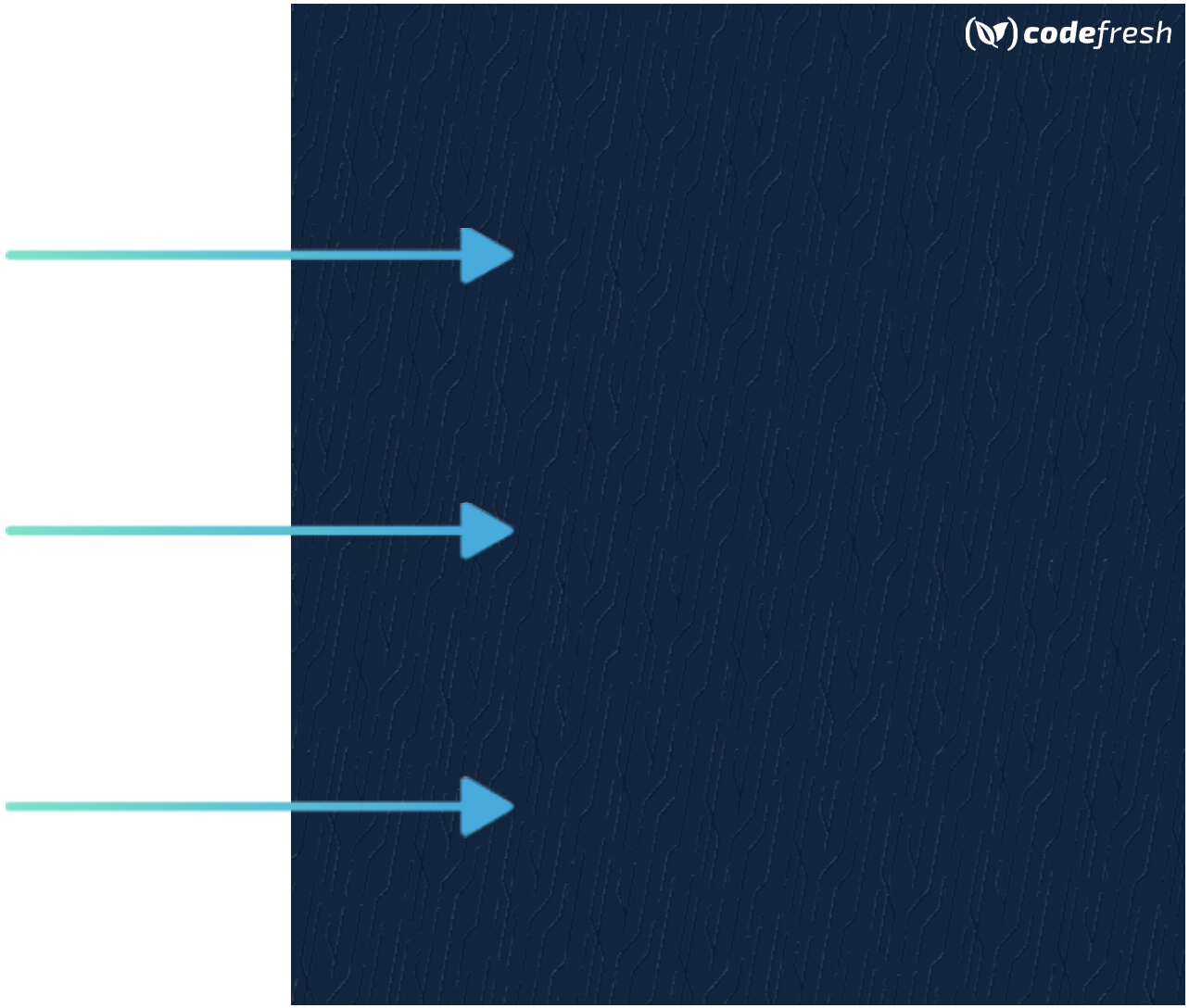
Sample Agenda

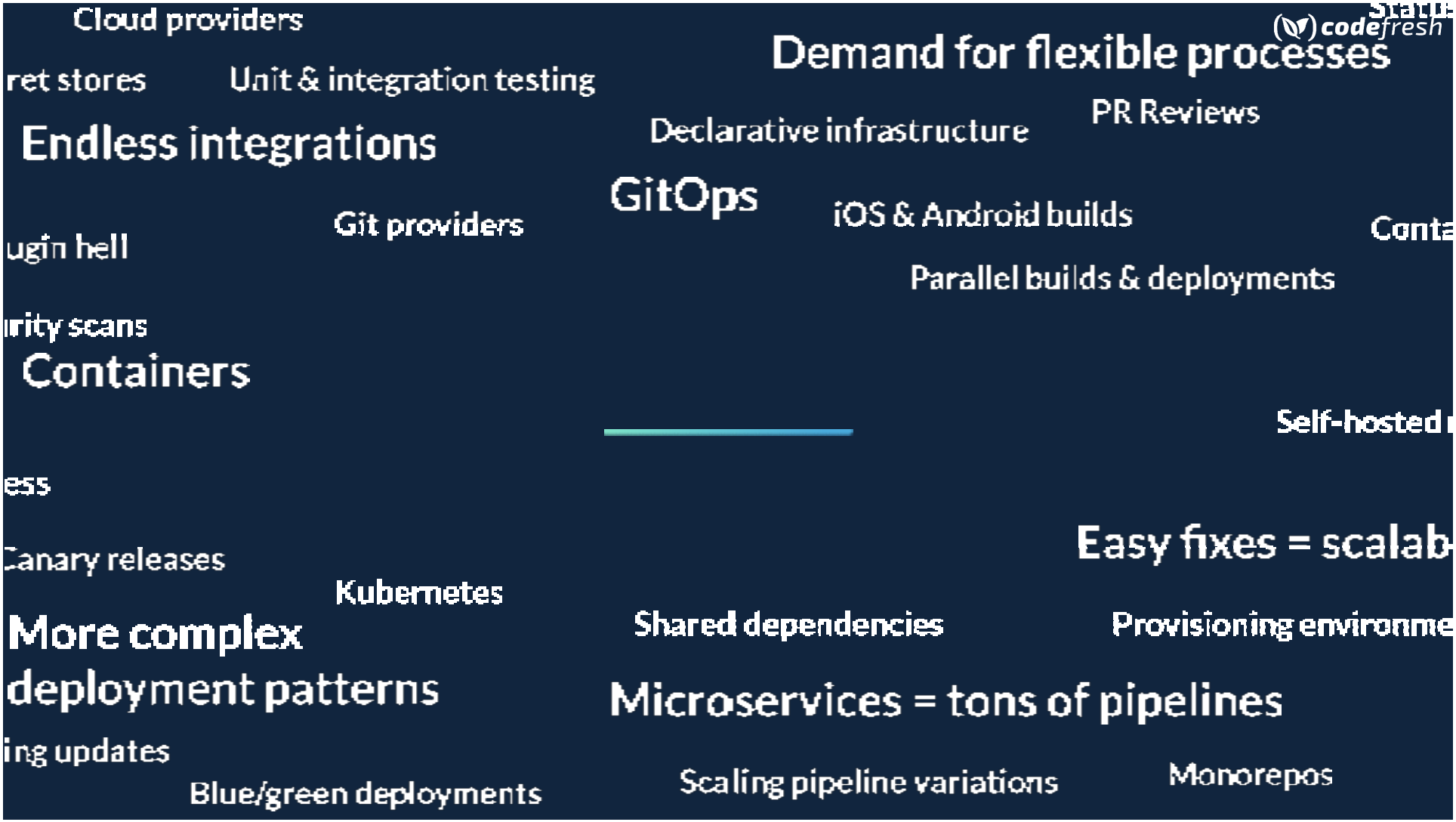
1. 10 MINUTES
   Introduction
2. 20 MINUTES
   Section 2
3. 15 MINUTES
   Live Demo
4. 10 MINUTES
   Q/A

codefresh

codefresh


codefresh


codefresh


Cloud providers
codefresh
Demand for flexible processes
ret stores
Unit & integration testing
Endless integrations
Declarative infrastructure
PR Reviews
GitOps
Git providers
iOS & Android builds
ugin hell
Parallel builds & deployments
rity scans
Containers
Self-hosted
ess
Easy fixes = scalab
Canary releases
Kubernetes
More complex
Shared dependencies
Provisioning environme
deployment patterns
Microservices = tons of pipelines
ing updates
Blue/green deployments
Scaling pipeline variations
Monorepos

codefresh

Meetup title

Kubernetes Anti-patterns

Kostis Kapelonis


codefresh

codefresh

codefresh

codefresh

codefresh

codefresh

codefresh

codefresh

codefresh

codefresh


codefresh

codefresh

The #1 GitOps automation platform for Kubernetes apps

codefresh


codefresh

codefresh

The modern approach to
DevOps automation

Open a FREE account today at codefresh.io

et stores

Unit & integration testing

Endless integrations

Declarative infrastructure

PR Reviews

GitOps

Git providers

iOS & Android builds

gin hell

Parallel builds & deploymen

ity scans

Containers

ICONS TO USE

Se

ss

nary releases

Easy fixes =

Kubernetes

More complex

Shared dependencies

Provisioning

eployment patterns

Microservices = tons of pipelines

g updates

Blue/green deployments

Scaling pipeline variations

Monorep