



Using Virtual Clusters for Development and CI/CD Workflows

Kostis Kapelonis



Your host today: Kostis

Developer Advocate at Codefresh.io
(CI/CD/GitOps) - Enterprise Argo CD

- Argo Rollouts maintainer
- VCluster Zealot
- All about Kubernetes deployments and Continuous Delivery

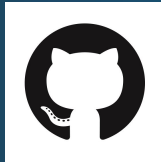


Kostis Kapelonis

DevRel Codefresh CI/CD

 @codepipes

Agenda



Intro to Virtual Clusters/Loft

Demo 1

Terraform/Loft

Demo 2

Terraform/Vcluster/Codefresh

Demo 3

Real World example

Demo 4



**There are no stupid questions
(only stupid answers)**



The running theme: people

Virtual Clusters for Everybody



Developer

Focus is on writing code and shipping features



Operator

Focus is on running the systems and giving infrastructure to developers



DevEx

Focus is on day-2-day productivity, onboarding and removing obstacles

Developer

Wants to ship features (move fast)

- Doesn't really want to learn Kubernetes
- Wants to follow commit from Git to production
- Wants easy way to create envs, deploy dbs, perform prototypes fast and easy (clicking buttons)



SRE/Operator

Wants healthy systems (move slow)

- Helps developers by offering infra and services
- Hates open tickets and lengthy procedures
- Prefers automation and CLIs instead of clicking buttons



DevEx/Lead/Other

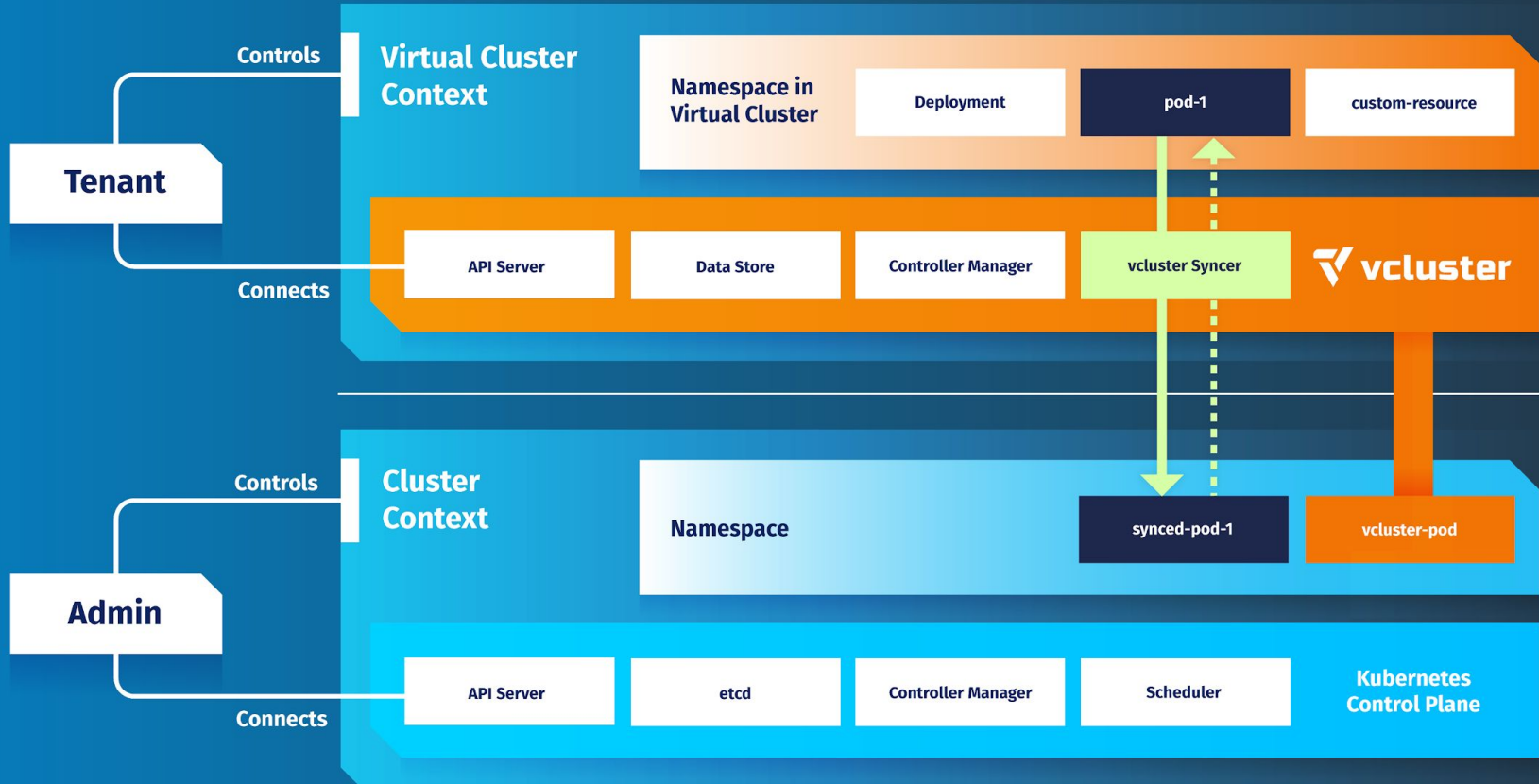
Cares about deployment velocity

- Helps with developer onboarding
- Aims to offer self-service capabilities to everybody
- Responsible for day-to-day operations and overall productivity

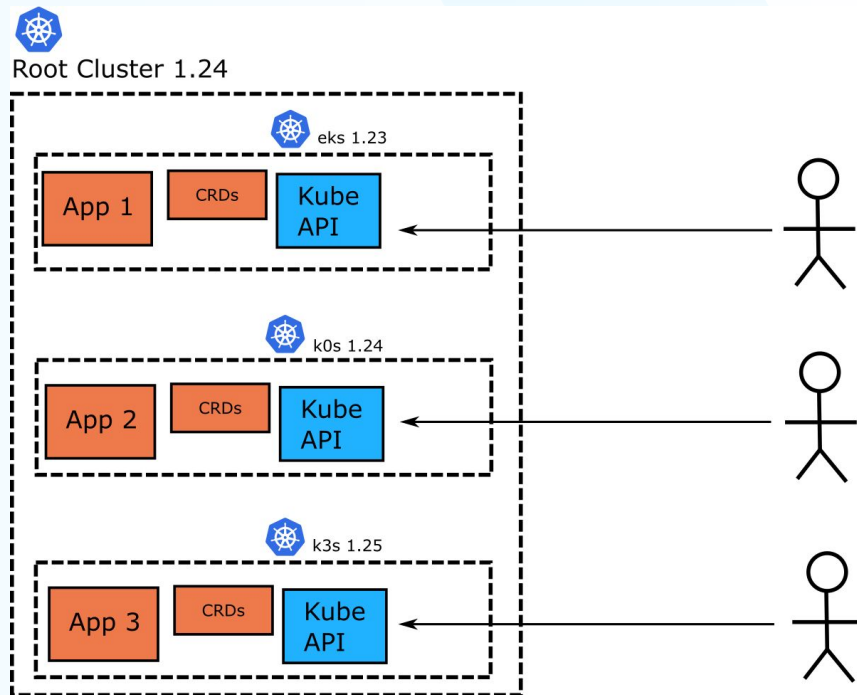
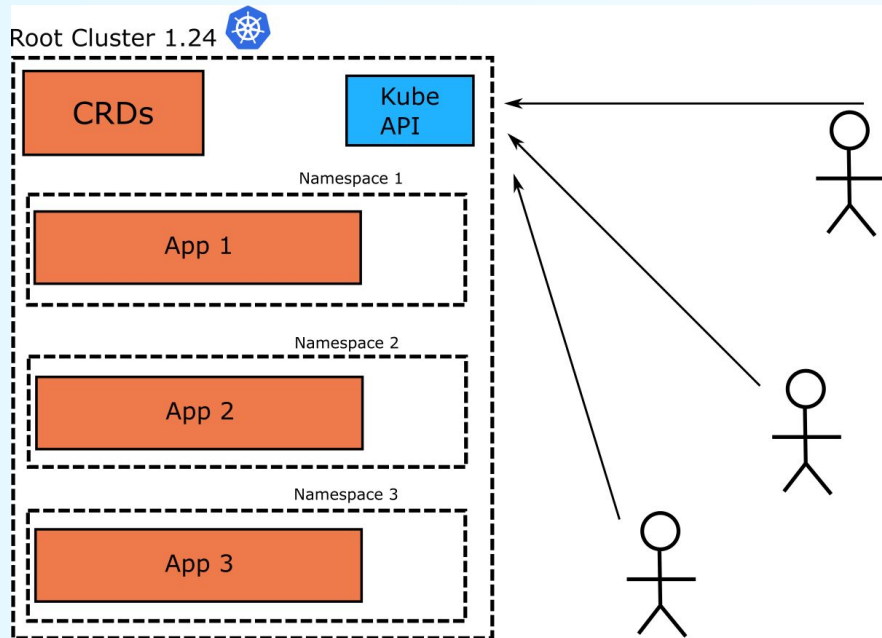


Virtual Clusters (why/what/how)

Virtual Cluster Architecture



Vcluster architecture (mine)



What is Vcluster



Use Cases

01

Dev Environments

Create full clusters on the spot. No more CRD conflicts. Preview each Pull request on its own cluster

02

Multi-tenancy

Give each customer a cluster with full isolation. Take advantage of standard Kubernetes autoscaling. Manage Kubernetes with Kubernetes

03

Cluster tests

Test different versions of Kubernetes (even newer than the host cluster)

04

Cost Control

Pay for a control plane only once. Automatically scale down clusters when not used

Use cases

There are already several sessions (including Kubecon) about different uses of Virtual clusters.

<https://www.youtube.com/@cncf>



How We Securely Scaled Multi-Tenancy with VCluster, Crossplane...
Ilia Medvedev & Kostis Kapelonis



How Adobe Planned For Scale With Argo CD, Cluster API, And
VCluster - Joseph Sandoval & Dan Garfield



Hundreds of Clusters Sitting in a Tree with Argo CD - Mike Tougeron,
Adobe



Unlocking Argo CD's Hidden Tools for Chaos Engineering -
Featuring... - D Garfield & B Phillips

Loft: Friendly dashboard for virtual clusters

Developer needs



01

Deploy to multiple namespaces

Deploy complex applications without affecting other teams

02

Test different Kubernetes versions

You have virtual clusters with different versions than the parent cluster

03

Create Dev environments in seconds instead of minutes

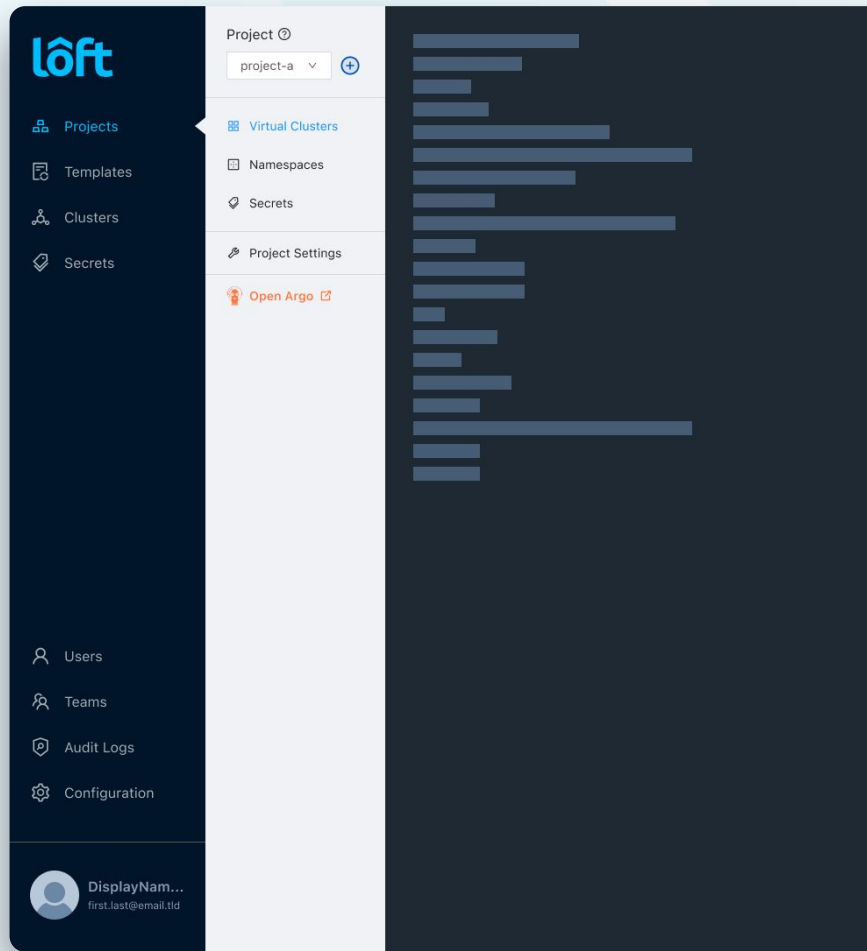
And also manage Kubernetes with Kubernetes

Demo 1 : Dev experience

Why Loft

Create clusters with a button click

- Apply access controls and lock down privileges for different teams
- Auto-scale down clusters when they are not used
- Deploy applications with templates and connect Virtual clusters to Argo CD
- Also check `devspace.sh` and `devpod.sh`



Create Virtual clusters without the UI

Automate everything

The Loft UI is optional. You can still create virtual clusters from the CLI and even from CI/CD pipelines

01

Use the vcluster cli

Just run “vcluster create”

02

Use the vcluster Helm chart

Just run “helm install”

03

Use the vcluster Terraform provider

Just run “terraform apply”

Terraform with Loft

Available at the Terraform registry

<https://registry.terraform.io/providers/loft-sh/loft/latest/docs>

01

Use your existing Terraform infra

Offers native resources and data sources

02

Most constructs from Loft

Projects, spaces, and clusters

03

Use the vcluster instance resource

Virtual cluster (loft 2x,) Virtual cluster instance (Loft 3.x)

Terraform resource

Works with all existing terraform tools

Create a Virtual Cluster Instance using terraform

```
terraform {
  required_providers {
    loft = {
      source = "registry.terraform.io/loft-sh/loft"
    }
  }
}

provider "loft" {}

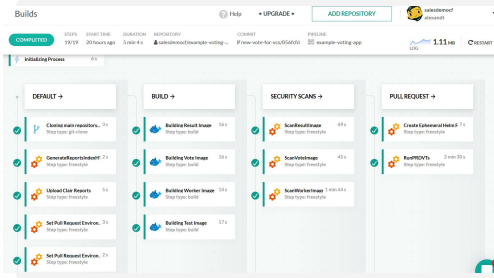
resource "loft_virtual_cluster_instance" "example-vcluster"
  metadata {
    namespace = "loft-p-example-project"
    name      = "example-vcluster"
  }
  spec {
    owner {
      user = "admin"
    }
    template_ref {
      name = "isolated-vcluster"
    }
  }
}
```

Demo 2: Programmatic creation of virtual Cluster

<https://github.com/kostis-codefresh/terraform-loft/tree/main/multiple-clusters>

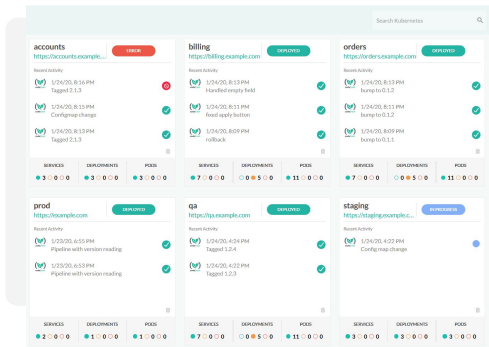
Brief intro to Codefresh

What Codefresh offers



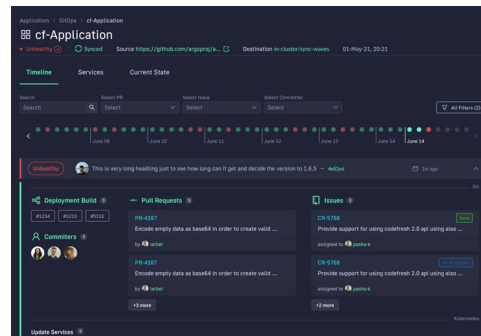
Pipelines

Implement Continuous
Integration



Deployments

Implement Continuous Delivery



GitOps

Enterprise Argo CD

Codefresh pipelines

- Docker based pipelines
- Kubernetes integration
- Docker registry integration
- Argo CD integration

The screenshot displays a Codefresh pipeline execution interface for a build named 'pull-request-preview'. The pipeline is in a 'COMPLETED' state, having finished successfully in 6 minutes and 24 seconds. The build size is 166.87 kB. The pipeline is organized into three stages: PREPARE, VERIFY, and DEPLOY. Each stage contains several steps, all of which are marked as successful with green checkmarks.

Stage	Step Name	Step Type	Duration
PREPARE →	Cloning repository	git-clone	2 s
	Compile/Unit test	freestyle	30 s
	Building Docker Image	build	13 s
VERIFY	Source security scan	freestyle	44 s
	Container security scan	freestyle	12 s
	Integration tests	freestyle	1 min 34 s
	Sonar Scan	freestyle	1 min 17 s
DEPLOY	Cloning repository	git-clone	3 s
	Deploying Helm Chart	helm	17 s
	Adding comment on PR	kostis-codfresh/github-pr-comment	16 s
	Smoke tests	freestyle	42 s

Building/pushing an image

GitHub Actions

```
- name: Log in to Docker Hub
  uses: docker/login-action@f4ef78c080cd8ba55a85445d5b36e214a81df20a
  with:
    username: ${ secrets.DOCKER_USERNAME }}
    password: ${ secrets.DOCKER_PASSWORD }}

- name: Extract metadata (tags, labels) for Docker
  id: meta
  uses: docker/metadata-action@9ec57ed1fcd8f14dcef7dfbe97b2010124a938b7
  with:
    images: my-docker-hub-namespace/my-docker-hub-repository

- name: Build and push Docker image
  uses: docker/build-push-action@3b5e8027fcad23fda98b2e3ac259d8d67585f671
  with:
    context: .
    file: ./Dockerfile
    push: true
    tags: ${ steps.meta.outputs.tags }}
    labels: ${ steps.meta.outputs.labels }}
```

Codefresh

```
BuildMyImage:
  title: Building My Docker image
  type: build
  image_name: my-app-image
  registry: dockerhub
  working_directory: '${ main_clone }'
  tag: 1.0.1
```

Deploying to Kubernetes

GitHub Actions

```
- name: Configure AWS credentials
  uses: aws-actions/configure-aws-credentials@v1
  with:
    aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
    aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
    aws-region: us-east-2

- name: Login to Amazon ECR
  id: login-ecr
  uses: aws-actions/amazon-ecr-login@v1

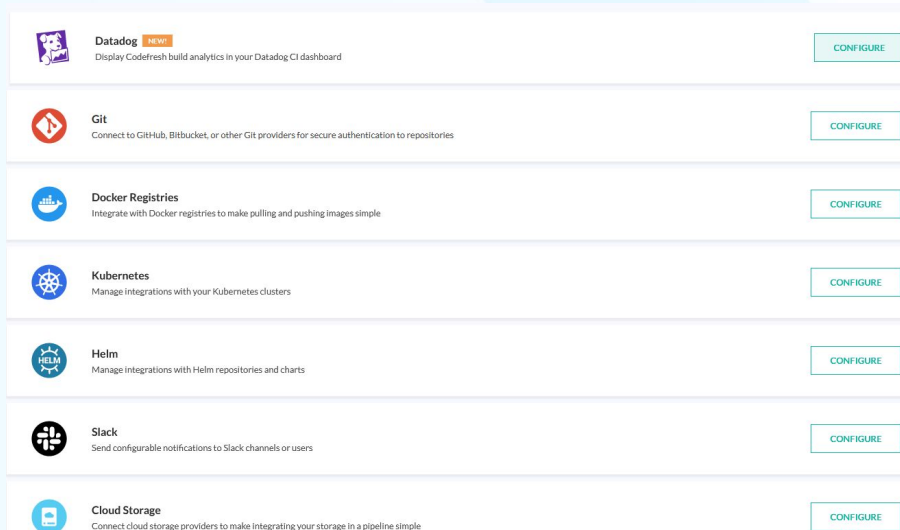
- name: deploy to cluster
  uses: kodermax/kubect1-aws-eks@master
  env:
    KUBE_CONFIG_DATA: ${{ secrets.KUBE_CONFIG_DATA_STAGING }}
    ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
    ECR_REPOSITORY: my-app
    IMAGE_TAG: ${{ github.sha }}
  with:
    args: set image deployment/$ECR_REPOSITORY $ECR_REPOSITORY=$ECR_F
```

Codefresh

```
deploy_to_k8:
  title: deploying to cluster
  type: deploy
  kind: kubernetes
  cluster: myDemoAKSCluster
  namespace: demo
  service: my-python-app
```

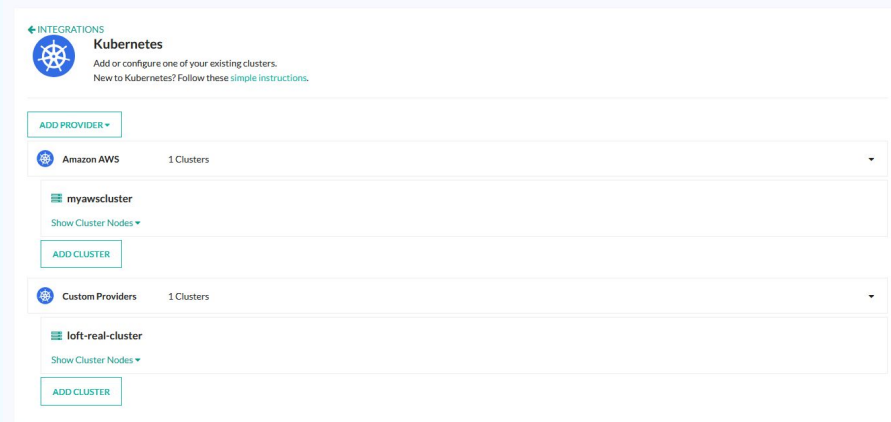
Codefresh Settings

- Attach Registries once
- Attach Clusters once
- Attach Helm repos once
- Other integrations



The screenshot displays the 'Integrations' section of the Codefresh settings. It features a list of seven integration categories, each with a unique icon, a title, a brief description, and a 'CONFIGURE' button on the right side.

Integration	Description
Datadog <small>NEW</small>	Display Codefresh build analytics in your Datadog CI dashboard
Git	Connect to GitHub, Bitbucket, or other Git providers for secure authentication to repositories
Docker Registries	Integrate with Docker registries to make pulling and pushing images simple
Kubernetes	Manage integrations with your Kubernetes clusters
Helm	Manage integrations with Helm repositories and charts
Slack	Send configurable notifications to Slack channels or users
Cloud Storage	Connect cloud storage providers to make integrating your storage in a pipeline simple



The screenshot shows the configuration page for the 'Kubernetes' integration. It includes a header with the Kubernetes icon and title, followed by instructions to add or configure clusters. Below this, there are two sections for existing clusters, each with an 'ADD PROVIDER' button and a list of clusters.

Kubernetes
Add or configure one of your existing clusters.
New to Kubernetes? Follow these [simple instructions](#).

ADD PROVIDER

- Amazon AWS** 1 Clusters
 - myawscluster
 - Show Cluster Nodes

ADD CLUSTER

Custom Providers 1 Clusters

- loft-real-cluster
 - Show Cluster Nodes

ADD CLUSTER

Demo 3: SRE/DevOps Experience

<https://github.com/kostis-codefresh/terraform-loft/tree/main/multiple-clusters-remote>

Terraform pipelines

The screenshot displays a web interface for configuring a pipeline. The main area is divided into two panes: a code editor on the left and a variables panel on the right.

Code Editor (Left Pane):

- Tab: Inline YAML
- Buttons: STEP MARKETPLACE, download, refresh, search, copy, paste
- Code content:

```
6 + stages:
7   - "clone"
8   - "deploy"
9
10 + steps:
11   clone:
12     title: "Cloning repository"
13     type: "git-clone"
14     repo: "kostis-codefresh/manage-cf-with-tf"
15     # CF_BRANCH value is auto set when pipeline is triggered
16     # Learn more at codefresh.io/docs/docs/codefresh-yaml/variables/
17     revision: "${CF_BRANCH}"
18     git: "github-1"
19     stage: "clone"
20
21   DeployWithTerraform:
22     image: hashicorp/terraform:1.3.7
23     title: Deploying Terraform plan
24     working_directory: "${clone}"
25     stage: deploy
26     commands:
27       - terraform init
28       - terraform apply -auto-approve
29
30
```
- Status: 0 Errors, 0 Warnings

Variables Panel (Right Pane):

- Section: VARIABLES
- Search: Search
- Section: Pipeline variables 2 [x] [copy]
- Variables list:
 - TF_VAR_cf_token = [redacted] [lock]
 - TF_VAR_gh_token = [redacted] [lock]
- Button: + ADD VARIABLE

Right Sidebar:

- STEPS
- TRIGGERS
- VARIABLES (Active)
- HELP

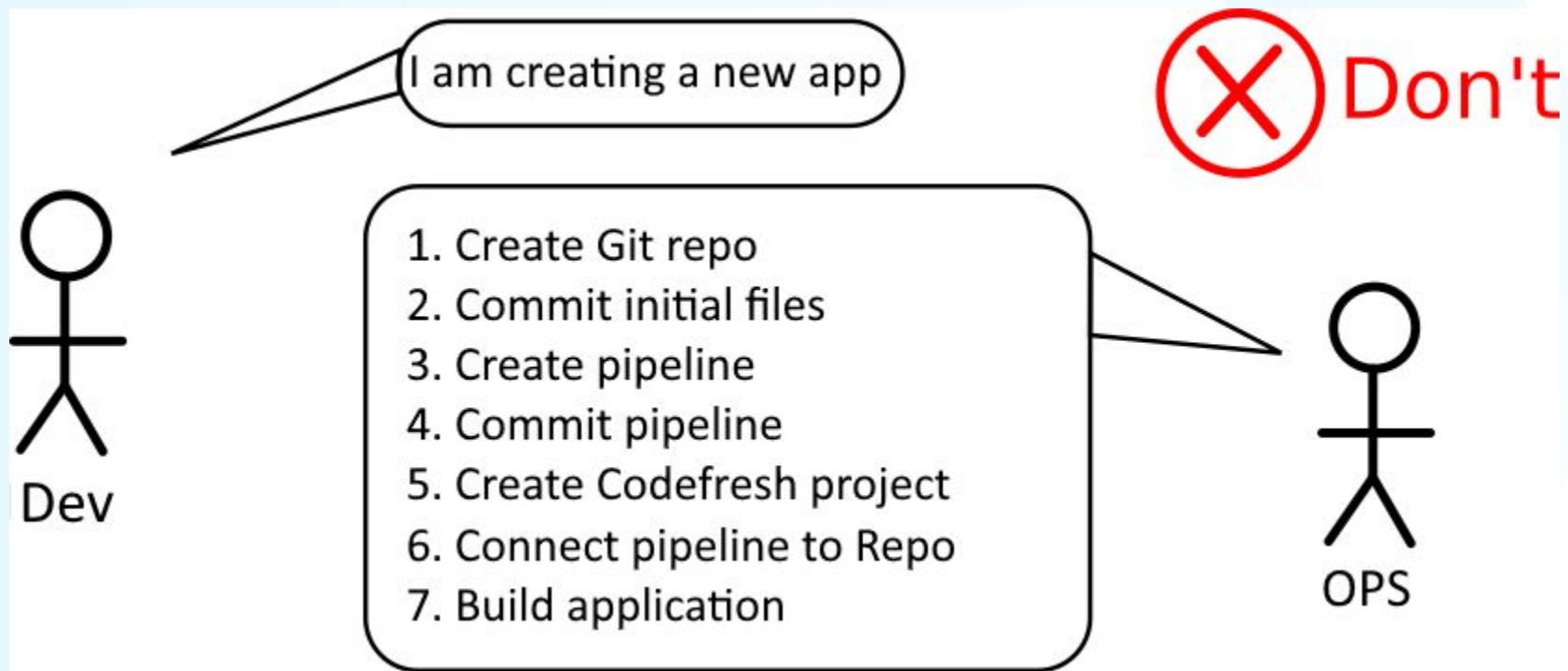
Real World scenario: New service/onboarding



**Developer - Hey, we are starting a
brand new service/application**



What usually happens



What should happen



Building blocks

01

Git repo

Create a starter Git repo for the new application

02

Clusters

Create 1 or more clusters (e.g. QA/staging/Prod) for the new application

03

Deployment pipeline

Create pipeline for building source code and deploying to cluster

04

Bootstrap Process

Declarative management for a process that handles steps 1, 2 and 3

Terraform provider for GitHub

Example Usage

```
resource "github_repository" "example" {
  name          = "example"
  description   = "My awesome codebase"

  visibility = "public"

  template {
    owner          = "github"
    repository     = "terraform-template-module"
    include_all_branches = true
  }
}
```

Terraform provider for Codefresh

Example Usage

```
resource "codefresh_project" "test" {
  name = "myproject"

  tags = [
    "production",
    "docker",
  ]

  variables = {
    go_version = "1.13"
  }
}
```

Master pipeline

Create **Everything** a developer needs in a single step

01

Create Git Repo

Happens via GitHub
terraform provider

02

Create Clusters

Happens via Loft provider
and/or Vcluster chart

03

Create Pipelines

Happens via Codefresh
terraform provider

04

Connect Clusters

Happens via Codefresh

05

Ready!

Developer can start
working

Master Pipeline

EVENT Manual **PIPELINE** create-everything **INITIATOR** kostis-codefresh

COMPLETED Build Completed Successfully 3 min 1 s 3 minutes ago Small LOG 61.93kb EDIT PIPELINE RESTART

Initializing Process 13 s

- PREPARE →**
 - ✓ Cloning repository 2 s (Step type: git-clone)
 - ✓ Choosing Real cluster 8 s (Step type: freestyle)
- INFRA →**
 - ✓ Creating virtual clusters 1 min 31 s (Step type: freestyle)
 - ✓ Listing virtual clusters 8 s (Step type: freestyle)
- CONFIGURE**
 - ✓ Getting Virtual Contexts 8 s (Step type: freestyle)
 - ✓ Creating Service accounts 7 s (Step type: freestyle)
 - ✓ Attaching VClusters to Codefr 22 s (Step type: freestyle)
- PIPELINES**
 - ✓ Creating pipelines and repo 15 s (Step type: freestyle)

Demo 4: Brand new application

<https://github.com/kostis-codefresh/terraform-loft/tree/main/give-me-an-app>

Questions?

Book a demo



<https://loft.sh/demo/>



<https://codefresh.io/request-a-demo/>

Resources

- <https://github.com/kostis-codefresh/terraform-loft/>
- <https://www.vcluster.com/>
- <https://loft.sh/docs/getting-started/install>

- <https://registry.terraform.io/providers/loft-sh/loft/latest/docs>
- <https://registry.terraform.io/providers/integrations/github/latest/docs>
- <https://registry.terraform.io/providers/codefresh-io/codefresh/latest>