

# Release It Like a Pro!



# devfest Alps

2023

*powered by*



**SUB-ZERO TEAM**

# Here we are!



## Simone Pulcini

Software Architect, Cloud lover, GDG Rome member

*"Quality lies in the details, so I make it a daily quest to uncover them"*. Simone is a passion-driven senior software architect, constantly seeking innovation and automation. He loves new dev languages, methodologies, and tools capable to inflame his willingness to learn. Family, friends, books and films are his personal pillars.



## Luigi Giuseppe Corollo

DevOps Architect, GDG Rome member

Luigi is a DevOps Architect with several years of experience in the IT battlefields, he loves using a methodological approach to problem solving driven by a passion for IT and doing things right. *"My good intention is to keep learning, experimenting, challenging myself and having new experiences."* He is also a proud junior father of two lively daughters.



# Agenda & Legenda

- **Release Management**

- It all starts from a version TAG....
- ...to a “Semantic” git commit message
- ...automated with Semantic Release
- Semantic Release: Plugin Lifecycle
- Repository management: Pre-Release strategy

- **Release Management & Automation tools**

- Automation Tools Overview
- Tekton
- ArgoCD

- **Demo**

- Solution Architecture & Demo



**What:** Best Practise section

**How To:** Best Open Source  
Tools choses in our use-case

**Example:** A running  
end2end demo

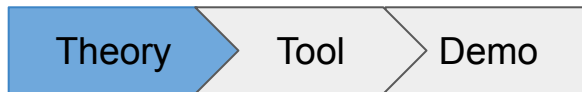




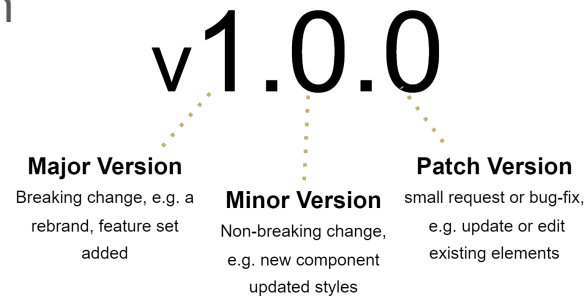
In a world dominated by  
chaos....



# It all starts from a version TAG....



- it's a **GIT tag** with a specific format
- **vX.Y.Z<label>** where X,Y and Z are numbers with an optional label
- **X** represents a “**Major Version**” (e.g. a breaking change)
- **Y** represents a “**Minor Version**” (e.g. a new feature)
- **Z** represent a “**Patch Version**” (e.g. a bug-fix, a performance tuning etc)



We should have a set of rules to be applied to calculate this tag each time there's a change to our code-base.

**Let's define this set of rules!**



# ...to a “Semantic” git commit message

Theory

Tool

Demo

```
<type>[optional scope]: <description>
```

```
[optional body]
```

```
[optional footer(s)]
```

## ***Based on “Conventional Commits” specification***

Type : {feat:, fix:, build:, chore:, ci:, docs:, style:, refactor:, perf:, test:}

Body: {\*.}

Footer: {\*|BREAKING CHANGE:}







# ...automated with Semantic Release

Theory

Tool

Demo

- it's a tool that runs in a CI
- it's written in JS using 
- it defines a lifecycle that covers the whole release management process
- it uses a  architecture (to enable support for different codebases using most of the package managers tools, registries and platforms)
- it's open source with good support and a great community
- your configuration resides in a single yaml/json descriptor file



# Semantic Release: Plugin Lifecycle

Theory

Tool

Demo



Each **plugin** will implement **one or more lifecycle** steps.

Semantic release engine will execute the lifecycle, one step at a time, running the step implementation for **each declared plugin**

**Plugin execution order** = plugin array declaration (descriptor)

- [@semantic-release/commit-analyzer](#)
  - `analyzeCommits`: Determine the type of release by analyzing commits with [conventional-changelog](#)
- [@semantic-release/release-notes-generator](#)
  - `generateNotes`: Generate release notes for the commits added since the last release with [conventional-changelog](#)
- [@semantic-release/github](#)
  - `verifyConditions`: Verify the presence and the validity of the GitHub authentication and release configuration
  - `publish`: Publish a [GitHub release](#)
  - `success`: Add a comment to GitHub issues and pull requests resolved in the release
  - `fail`: Open a GitHub issue when a release fails





# Definitions

Theory

Tool

Demo

- **Protected branch:** a long-lived branch, created at repo startup. When it receives a merge, a release will be done (e.g. pre/beta, main....)
- **Pre-release branch:** a branch that produces pre-release versions.
- **Release branch:** a branch that produces release version
- **Pre-release:** a kind of release for “internal-use” only. One of these will be eligible to become a release. (e.g. pre/beta). Version number is fixed and label number will increment; v1.0.2-beta1→1.0.2-beta2 1.0.2-beta3....
- **Release:** a pre-release that becomes a release (merge pre/beta into main): v1.0.2 (a release for the masses :-)
- **Channel:** a label for the distribution channel name adopted for a particular branch (like @next or @latest)

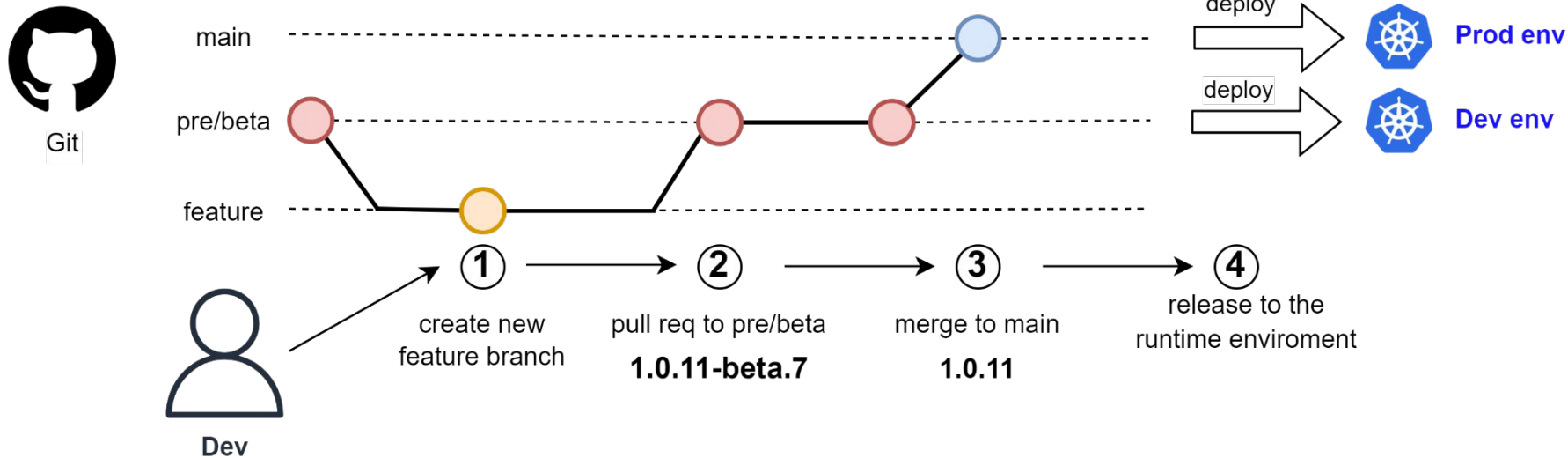


# Repository management: Pre-Release strategy

Theory

Tool

Demo



# Anatomy of a .releaserc.yml file

Theory

Tool

Demo

```
plugins:
  - "@semantic-release/commit-analyzer"
  - "@semantic-release/release-notes-generator"
  - "@semantic-release/changelog"
  - "@semantic-release/npm"
  ...
  - "@semantic-release/github"
  - "@semantic-release/git"
  - assets:
    - package.json
    - package-lock.json
    - CHANGELOG.md
    message: |-
      ci(release): ${nextRelease.version} [skip ci]
      ${nextRelease.notes}
  }
branches:
  #child branches coming from tagged version for bugfix (1.1.x) or new features (1.x)
  #maintenance branch
  - name: "+([0-9])?([.]+([0-9]),x).x"
    channel: "latest"
  #release a production version when merging towards main
  - name: "main"
    channel: "latest"
  #prerelease branch
  - name: "pre/beta"
    channel: "next"
    prerelease: "beta"
  }
debug: true
```

Plugins  
configurations



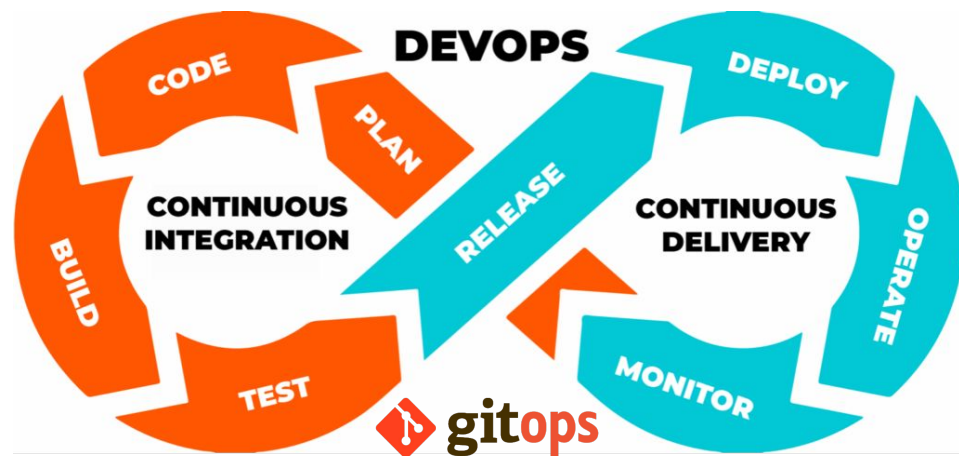
Branches  
configurations

# Release Management & Automation tools

Theory

Tool

Demo



Our approach:

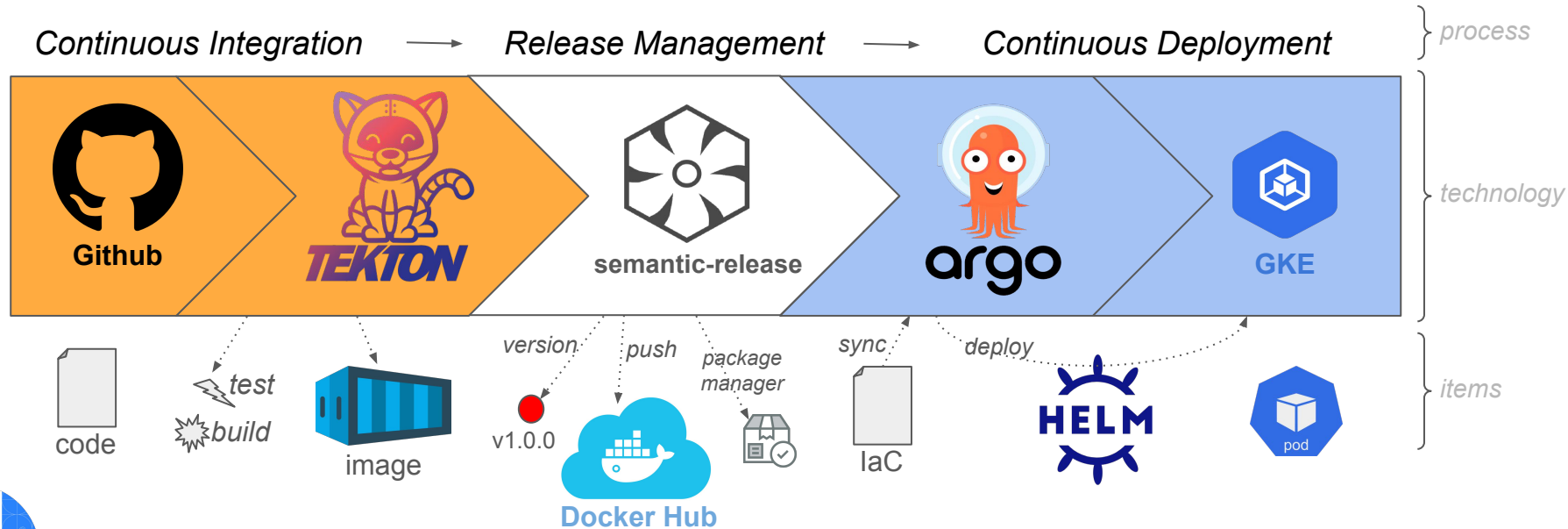
- **Vendor Agnostic** solutions
- **Cloud Native (k8s)** implementations
- Widely used **open source** projects
- Adoption of **consolidated strategies** (DevOps) and new **best practices** (GitOps)

# Automation: Tools Overview

Theory

Tool

Demo





# Tekton: What is?

Theory

Tool

Demo

What is Tekton ?

- Open-source **CI** platform implementation running natively in **Kubernetes**.
- Enables the definition of **pipeline as code**
- Open Source Ecosystem with a **large catalog of reusable components**
- Continuous Delivery Foundation (CDF)

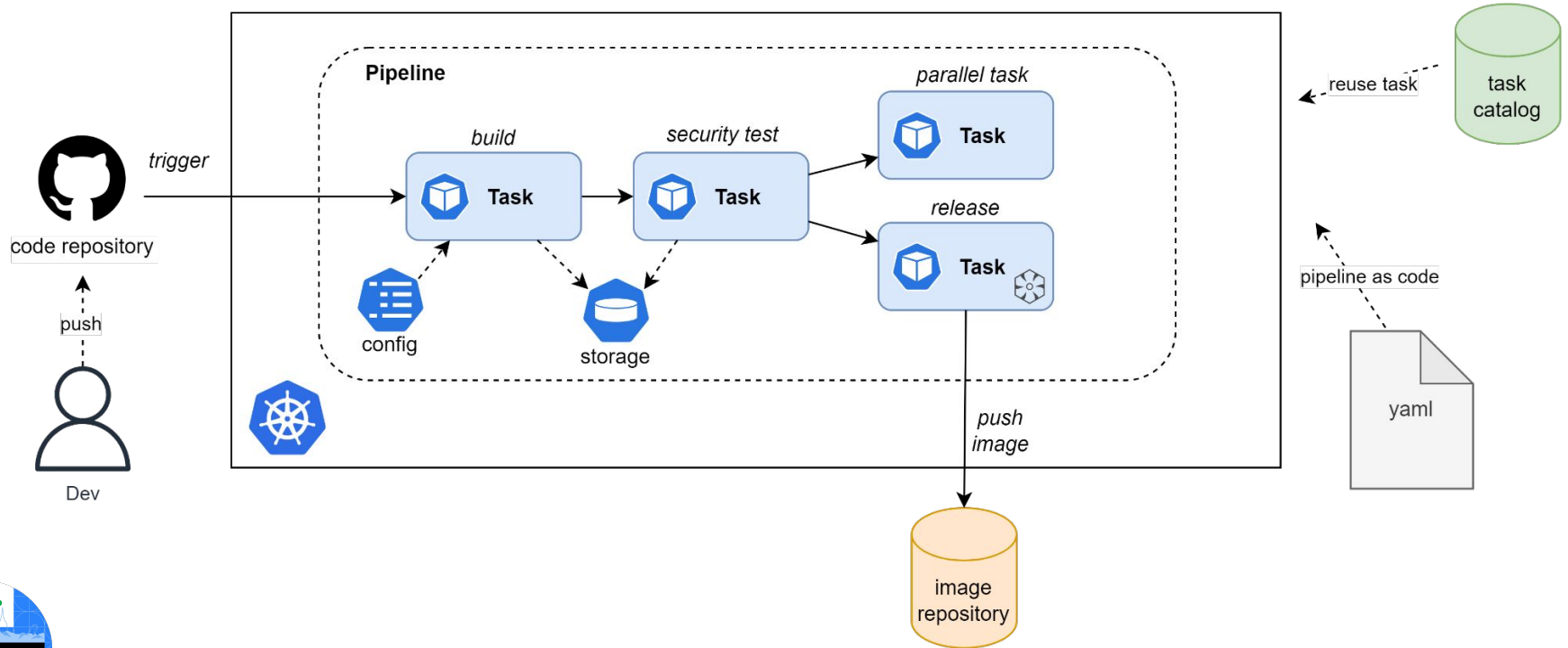


# Tekton: How it works?

Theory

Tool

Demo



# Continuous Delivery with ArgoCD

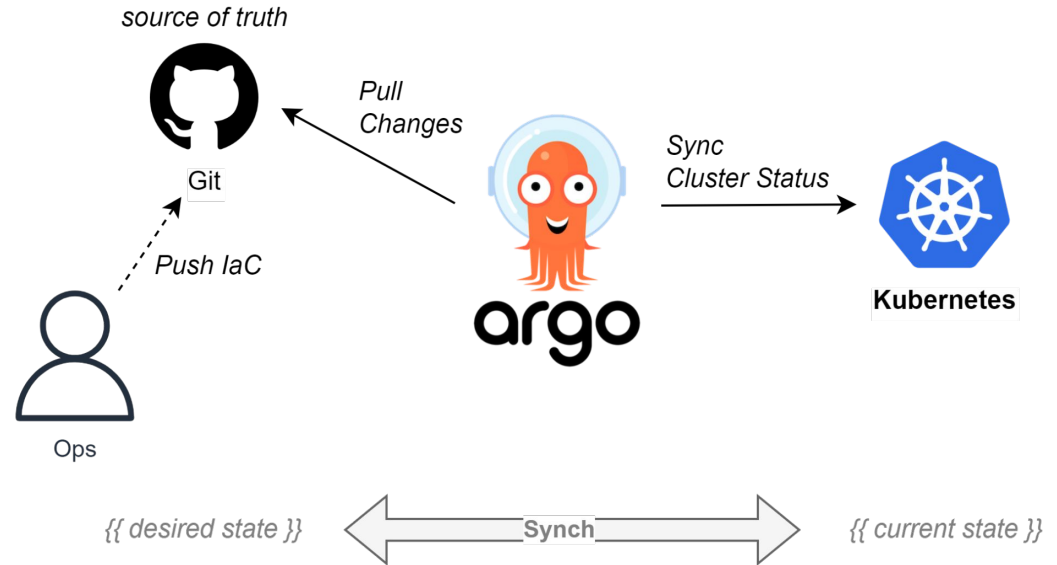
Theory

Tool

Demo

## What is ArgoCD?

- Open-source declarative **GitOps** continuous delivery tool, **native k8s**.
- Automated **Sync** Deployment & **Drift Detection**
- Integrated **Notification Engine**
- **CNCF** Graduated Project



# ArgoCD: Application Example

Theory

Tool

Demo

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: demo-app-dev
  annotations:
    notifications.argoproj.io/subscribe.on-deployed.slack: demo-app-deploy
    notifications.argoproj.io/subscribe.on-sync-failed.slack: demo-app-deploy
    notifications.argoproj.io/subscribe.on-sync-succeeded.slack: demo-app-deploy
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  destination:
    namespace: demo-dev
    server: https://kubernetes.default.svc
  project: default
  source:
    helm:
      valueFiles:
        - env/dev.values.yaml
      path: .
      repoURL: https://github.com/subzero-team/demo-app-iac
      targetRevision: main
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
```

Argo Spec. Kind

Notification Channels



Destination Configuration



Source Configuration



Sync Configuration

demo-app-dev.yaml

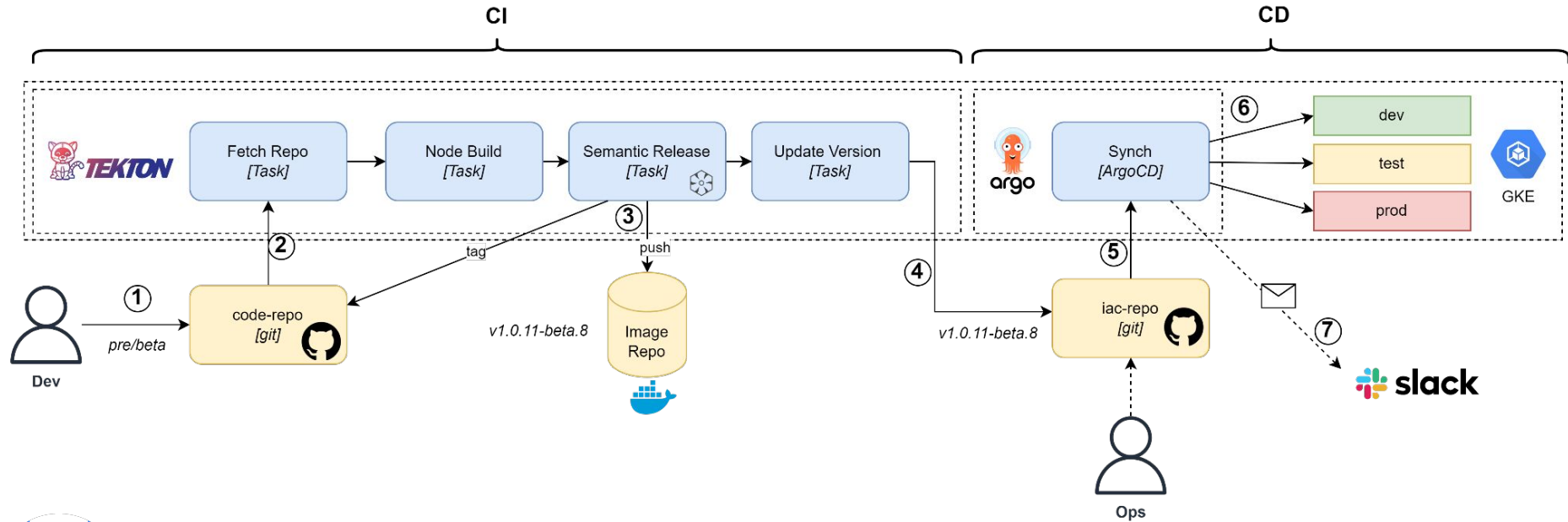


# Demo's Solution Architecture

Theory

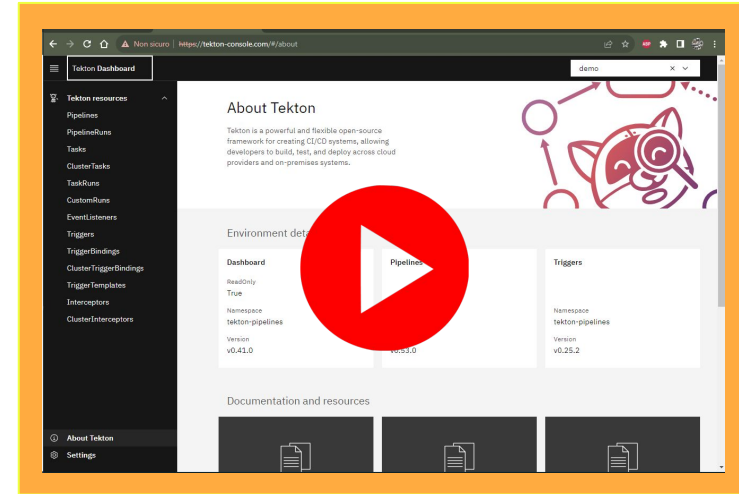
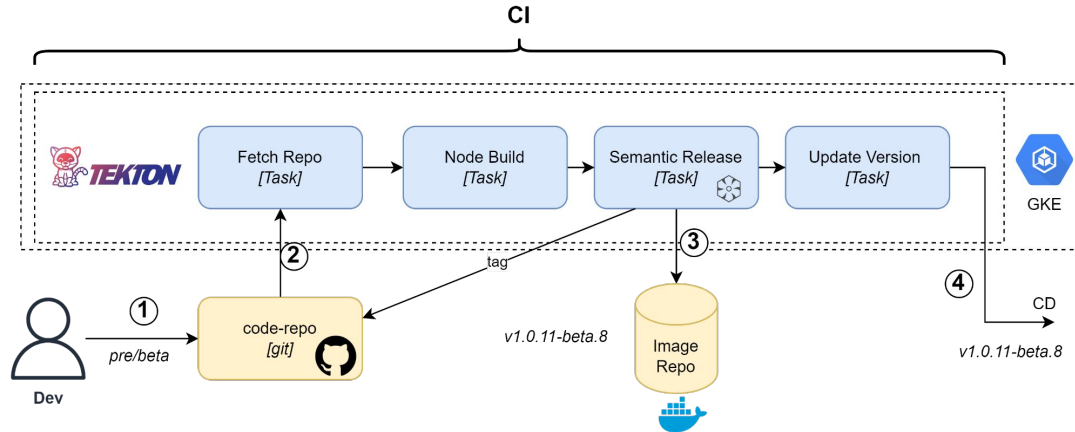
Tool

Demo





# Demo - Pipeline CI

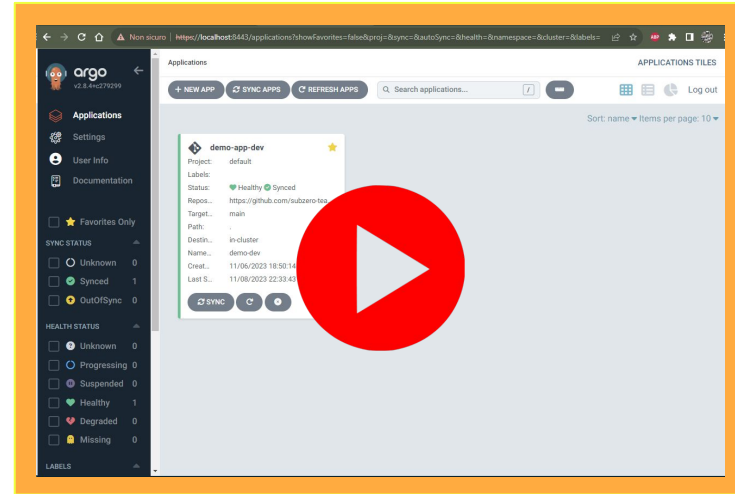
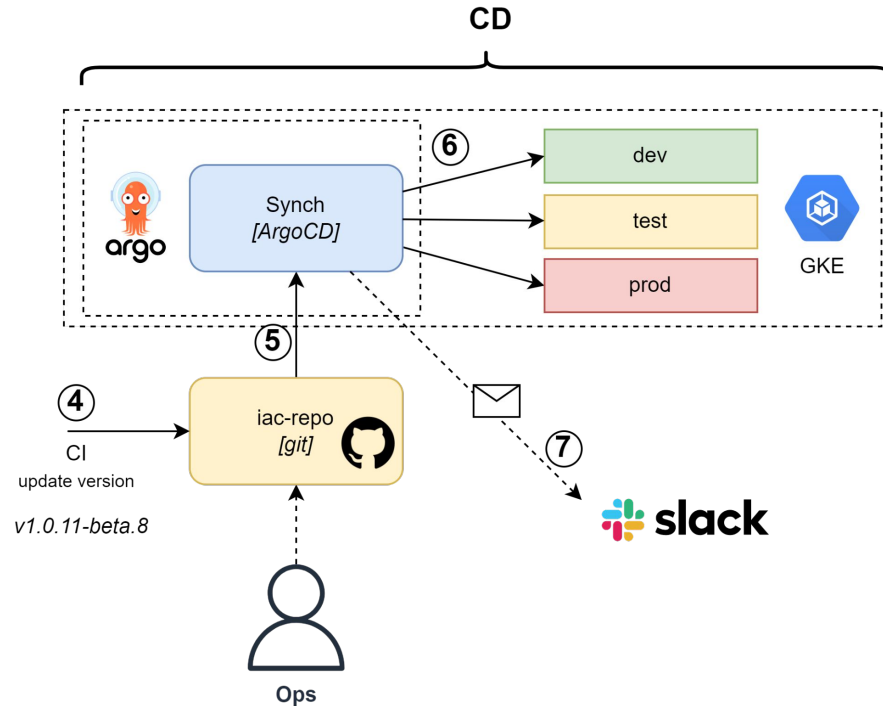


# Demo - Pipeline CD

Theory

Tool

Demo

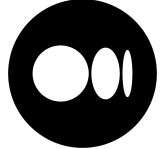




# Thanks!



- <https://github.com/subzero-team/>



- <https://bit.ly/Release-It-Like-a-Pro-Part-1>
- <https://bit.ly/Release-It-Like-a-Pro-Part-2>
- <https://bit.ly/Release-It-Like-a-Pro-Part-3>



- <https://www.linkedin.com/in/simonepulcini/>
- <https://www.linkedin.com/in/luigicorollo/>

