Involving a CI/CD Workflow

GitOps & Event-Sourcing automation at beGroup & Teko

Ageda

- Why do we care?
- What did we try to do?
- What problems did we run into?
- What do we do right now?

speakers/self

- Thinh Tran - Anh Xe Ôm
- **@duythinht**
- Software Engineer at Teko, ex-be-er
- We are hiring
    - duythinht@gmail.com

Why do we care?

- There are a plethora of DevOps tools for provisioning, configuration management, continuous integration, deployments and so on
- Separation of concerns
  - Build
  - Ship
  - Deploy
- Security
- Reproducibility

Our challenge

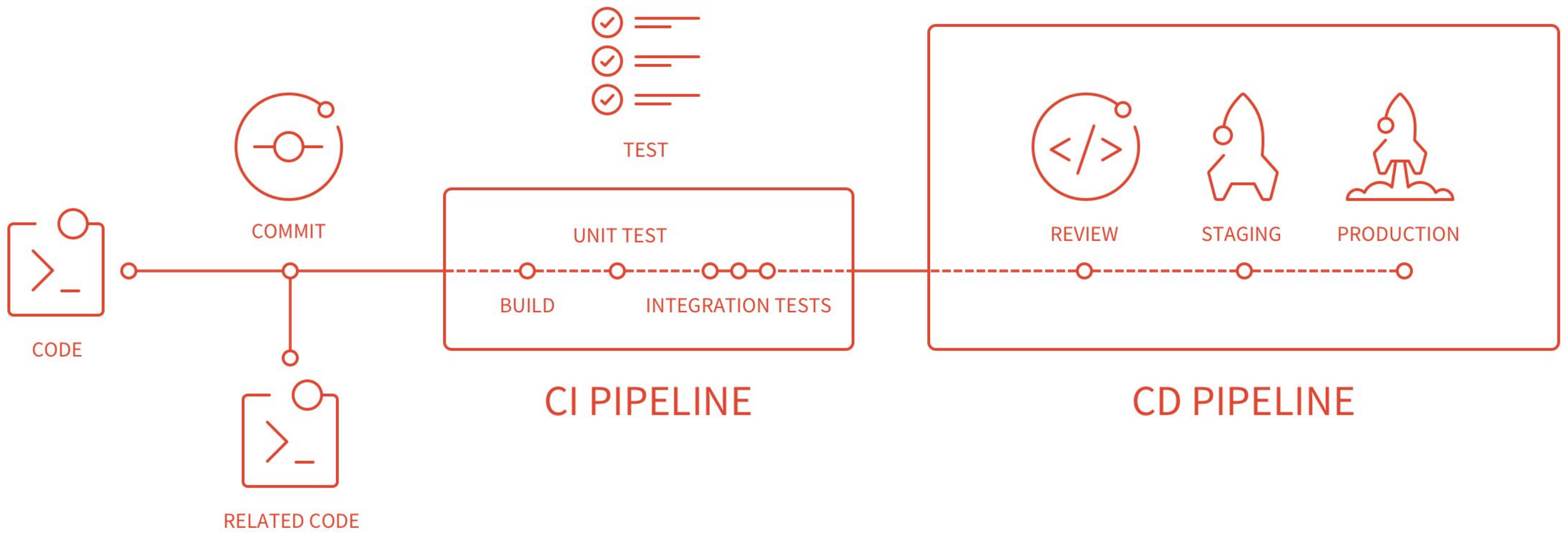
- 100+ developers and complexity of products
- High level of autonomy
- Duplication of efforts
- Varying continuous delivery maturity level
- Central Product & Tech org to drive convergence

What did we try to do?

Promote CI/CD system and build infrastructure to support it

- Deploy delta as small as possible
- Quick bootstrapping
- Cheap maintenance
- Standardised infrastructure footprint
- Transparency

A basis CI/CD workflow



Back to Concepts

- **Continuous Integration (CI)** is the practice of merging all developers working copies to a shared mainline several times a day. For every code change, build, unit test, and package your application. You can also push your package to a PaaS/IaaS or artifact repository.
- **Continuous Delivery (CD)** is a software development practice where code changes are automatically prepared for a release to production. A pillar of modern application development, continuous delivery expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage. When properly implemented, developers will always have a deployment-ready build artifact that has passed through a standardized test process.

What does “continuous” mean?

- **Frequent releases:** The goal behind continuous practices is to enable delivery of quality software at frequent intervals. Frequency here is variable and can be defined by the team or company
- **Automated processes**: A key part of enabling this frequency is having automated processes to handle nearly all aspects of software production. This includes building, testing, analysis, versioning, and, in some cases, deployment.
- **Repeatable**: If we are using automated processes that always have the same behavior given the same inputs, then processing should be repeatable.
- **Fast processing**: "Fast" is a relative term here, but regardless of the frequency of software updates/releases, continuous processes are expected to process changes from source code to deliverables in an efficient manner

Why “continuous”?

- Automate the Software Release Process
- Improve Developer Productivity
- Find and Address Bugs Quicker
- Deliver Updates Faster

What are current CI/CD Problems?

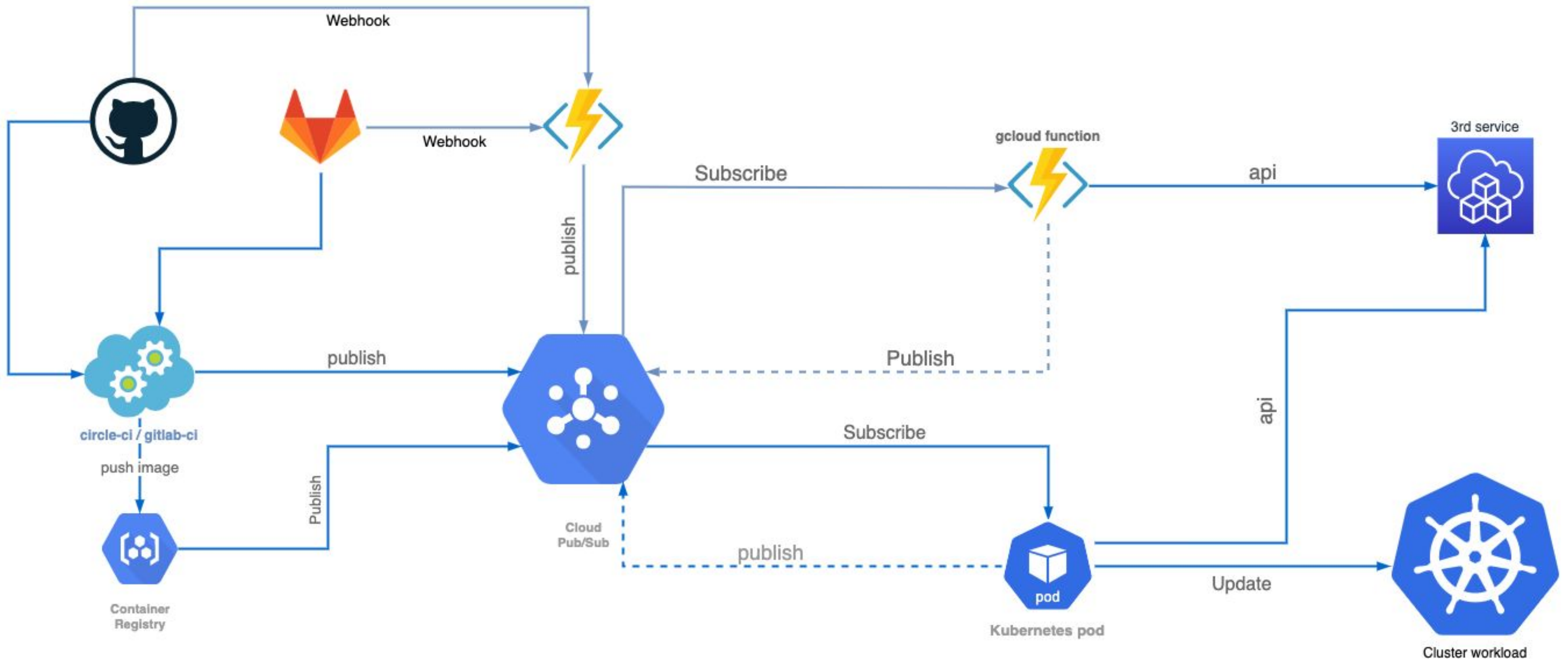
- Lots of tool, framework and opinions but totally unsuitable for run at scale
- Too complex, too verbose & inefficient for configuration and customize workflow
- Too limited, hard to extend logic & explain for other engineer.

We offer “pipeline-as-code”!!!

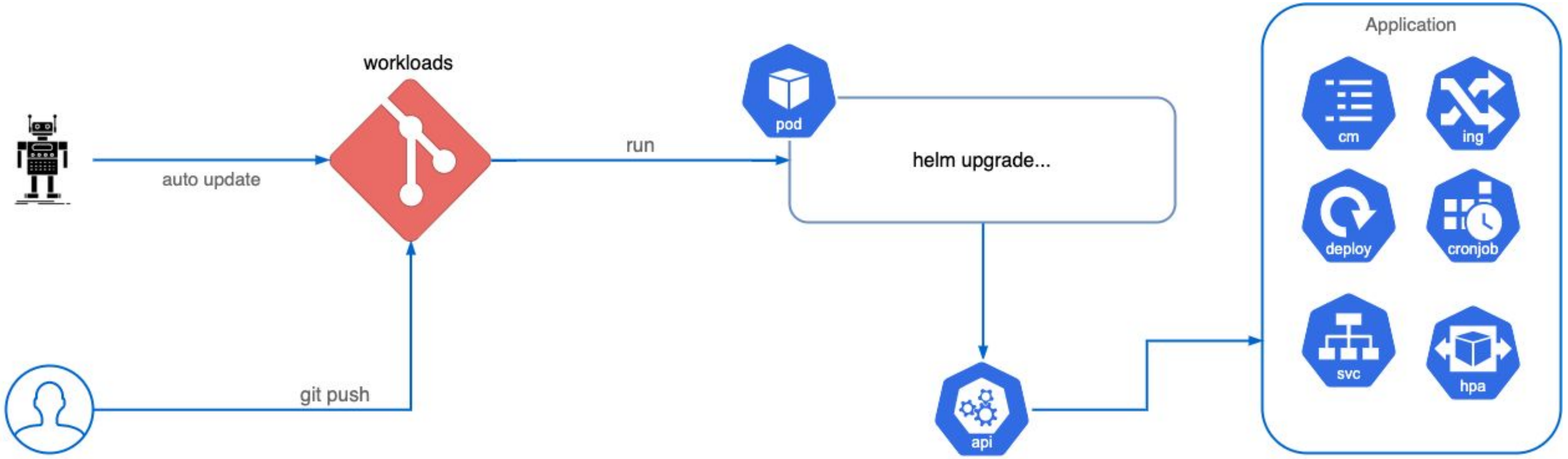
Pipeline-as-code is a general term for creating pipeline jobs/tasks via programming code, just as developers work with source code for products. The goal is to have the pipeline implementation expressed as code so it can be stored with the code, reviewed, tracked over time, and easily spun up again if there is a problem and the pipeline must be stopped

In the Big Picture
We have a pipeline

Event-Driven CI/CD



Workload management



Benefits

- **Simplibility**: Each CI Block is small, easy to produce, configuration and can be programmatic
- **Reliability**: Each block is triggered by an event, could be retry and ack.
- **Scalability**: Extensibile, you can add and remove CI Block due to business, system, and operation requirement
- **Agnostic:** Use any tools, programming language for your block

How we deploy an application on Kubernetes

- Using helm
- Make an “umbrella chart” cover all of application
- Using manifest to manage application (helmfile), but with small tweak to apply chart quick and instantly
- Each cluster has it own authority (using gitlab-ci)

Umbrella Chart

- One chart to rules them all
  - Deployments
  - Jobs
  - Cronjobs
  - Services
  - Ingress
  - Configmap
  - Sealed Secret
- Update & maintain by SRE team
- Developers configuration have freedom like Communism

```
# teko-backend-v2

image:
  repository: duythinht/whereami
  tag: v1
  pullPolicy: IfNotPresent

imagePullSecrets: []


//List of deployment
deploys:
  - name: svc-1
    env:
      HI: hello world
      TEST: this is 1

    # mount secret into /etc/hello
    mountSecrets:
    - name: test
      mountPath: /etc/hello

    # mount configmap into /etc/cfg
    mountConfigMaps:
    - name: test
      mountPath: /etc/cfg

    # this deployment is expose as service
    service:
      name: override-service-name # or this will be $release-svc-1
      type: ClusterIP
      containerPort: 3000
      health:
        live: /
        ready: /
```

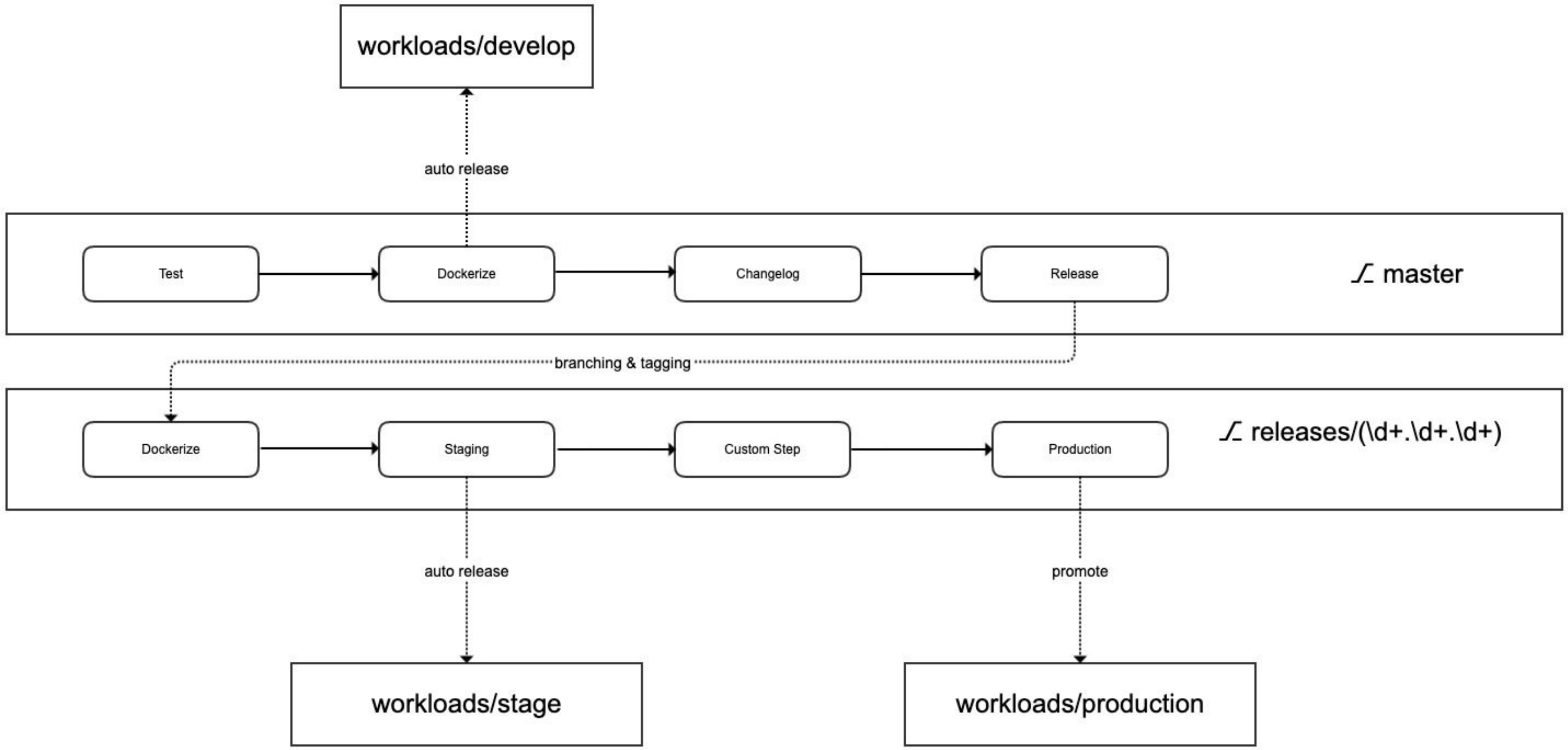
Cluster Manifest (helmfile)

- Keep a directory of chart value files and maintain changes in version control.
- Apply CI/CD to configuration changes.
- Periodically sync to avoid skew in environments.
- Check Point for Disaster Recovery

Just deploy

- New deployments by PRs
- Bot automate update manifest to keep track version
- Update config trigger sync state from git revision to cluster
- Rollback git also rollback (sync state) to production

Trunk based development



Make better tools

- Integrate with current CI system (gitlab ci, circle ci, jenkins...)
- Trying "do one thing and do it well"
- Each operation should be automatic, chain of automatics will be automation
- Capture everything
    - Keep a changelog
    - Labeling commit
    - Semantic version
- Standardization

Demonstration

Thank You!

Don’t forget to continuous deploy your CV