From application code to deployment: Automated building, Gitops and beyond

Michael Trip

Velp / The Netherlands / 2024

AT COMPUTING
ATYPICAL OPEN SOURCE GURUS

M.Trip@atcomputing.nl

who am i?

Name: Michael Trip

From: Apeldoorn / The Netherlands

- Open Source consultant @ AT Computing
- Has worked with Windows before
- Linux Geek (started in 2004)
- Kubernetes addict
- Redhat Certified System Administrator
- Redhat Certified Engineer
- Redhat expert in Openshift Administration
- Certified Kubernetes Application developer
- Certified Kubernetes Administrator
- Certified Kubernetes Security Specialist

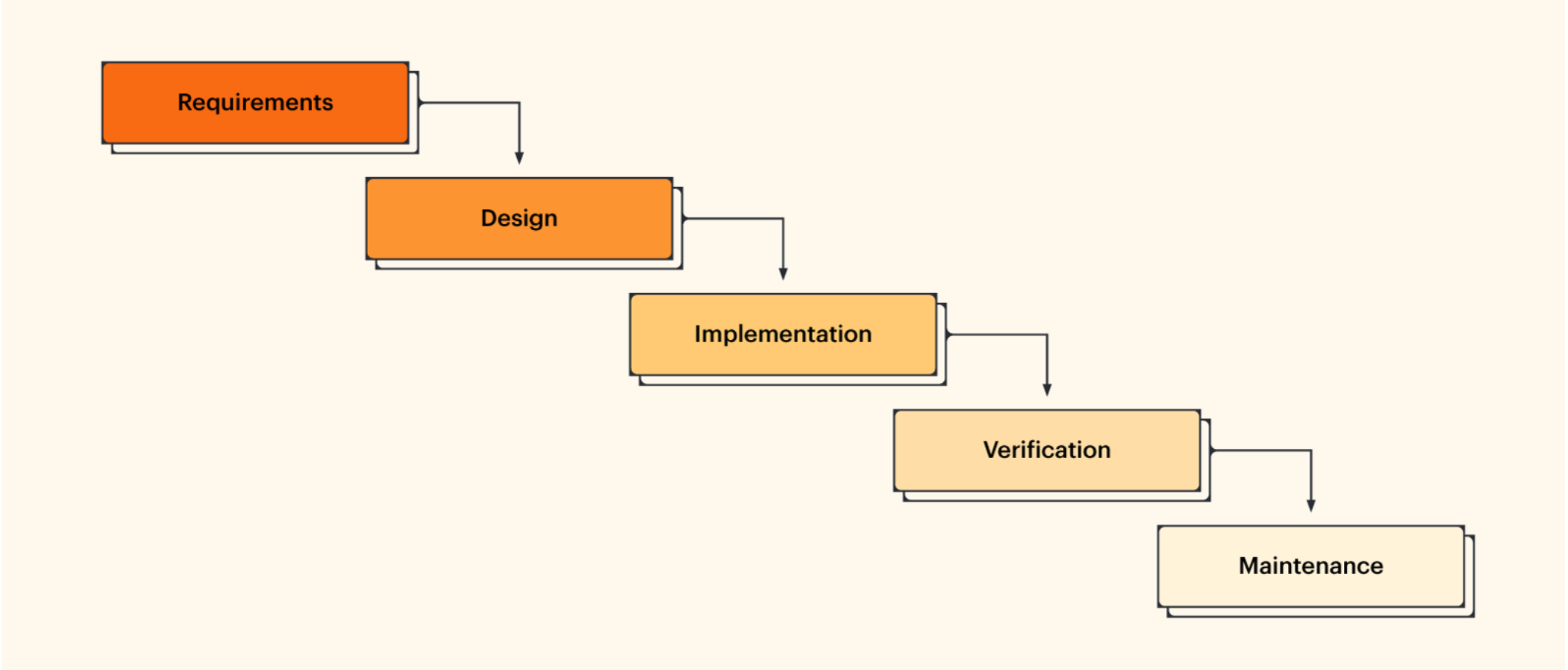
who am i?

Introduction

- This presentation will be about software releasing and deploying:
  - The old way (Waterfall)
  - A possible new way to do this
  - Tools on how to achieve this
  - Tips and tricks
  - A demo

The old way of building and releasing (1)

The “Waterfall method”



Source: https://www.lucidchart.com/blog/nl/watervalmethode

The old way of building and releasing (2)

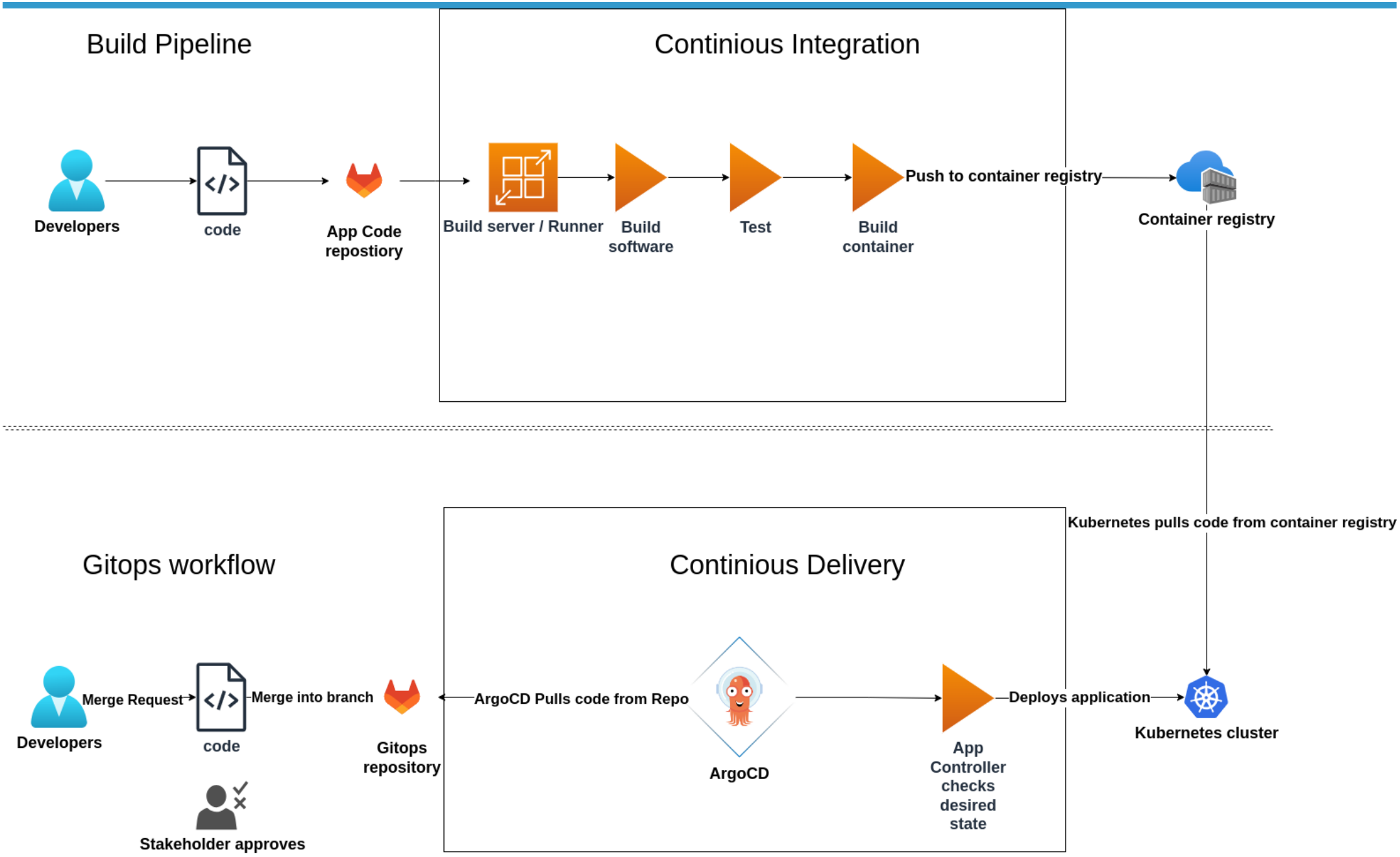
The “Waterfall method”

- Pros:
  - Easy to understand
  - Easy to manage
  - Predictable

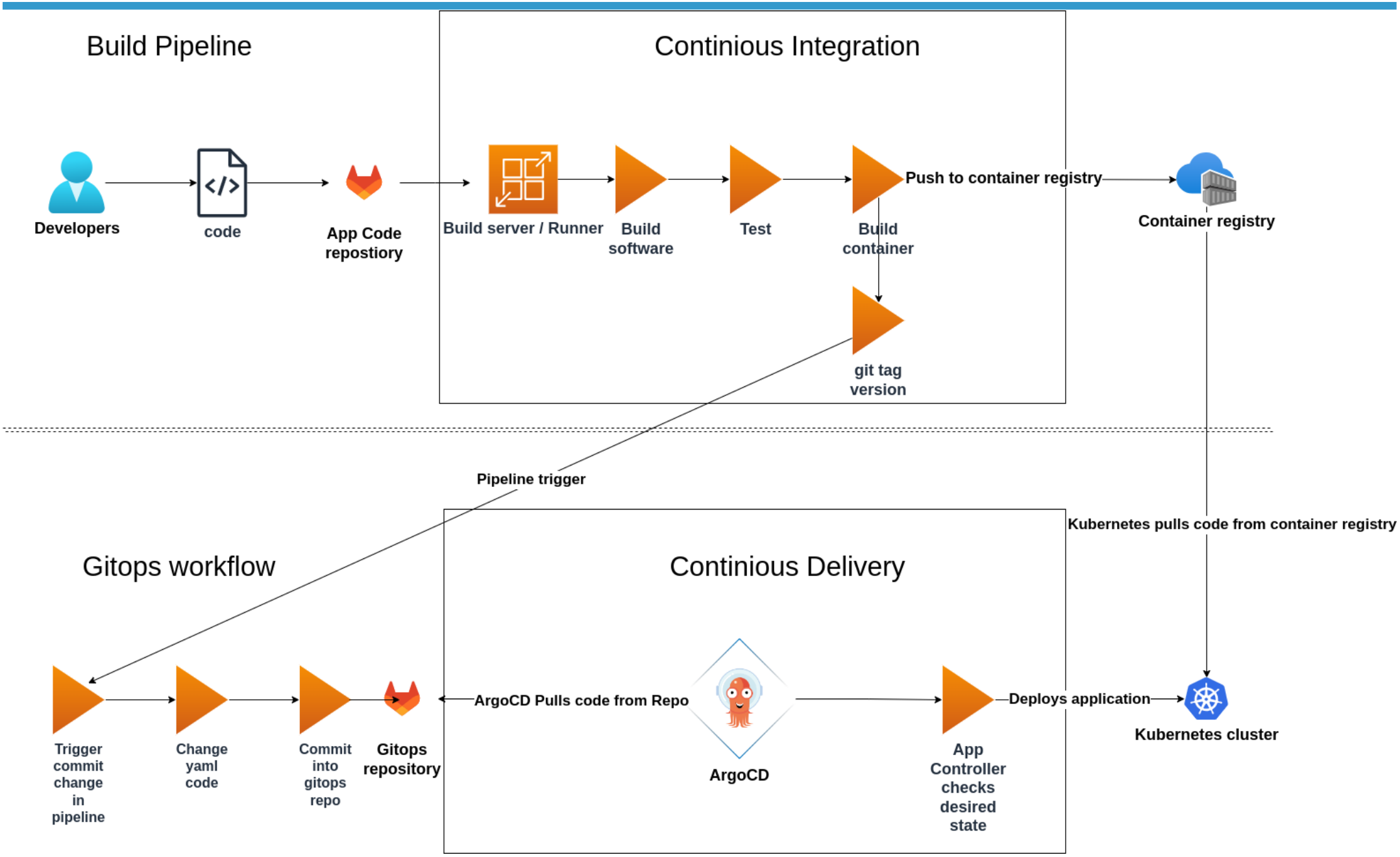
- Cons:
  - Not flexible
  - Slow feedback cycle
  - Slow in delivering

The new way

The possible solution – the manual way



The possible solution – the semi automated way



What tools are we going to use ?

kubernetes

+

GitLab

+

Kubernetes ?

Gitlab ?

- Version control system
- But… with some more tricks up it’s sleeve:
  - **Pipeline support**
  - **CI/CD support**
  - **Container registry**
  - **Kubernetes agent support**
  - **Nice RBAC system**

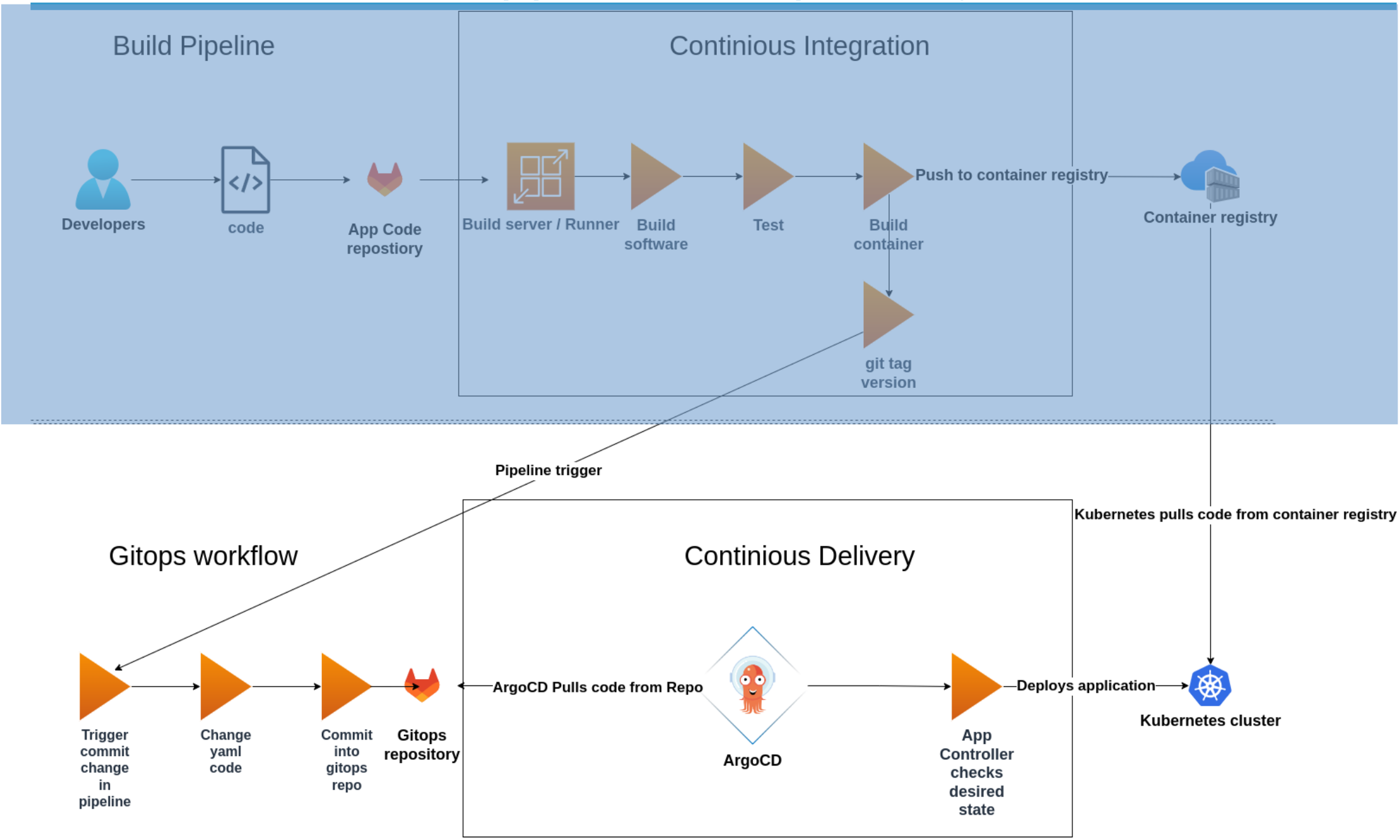
ArgoCD ?

- CD controller running in Kubernetes
- Enforces the state of application in Kubernetes (when configured)
- Integrated health checks for applications
- Can be triggered by a GIT solution such as Gitlab
- When triggered, pulls git repo
  - Looks for changes
  - Applies that changes

Gitops?

- A declarative way to manage your environment
- Git == Single source of Truth
- ArgoCD == CD Part
- Gitlab == CI part
- Gitlab Agent to use Kubernetes from within the pipeline.

The application repository (1)



The application repository (2)

- Compiles the application code
- Creates a container image
- Tests that image
- Generates dynamic manifests
  - `Envsubst` is awesome!
- Starts the application in Kubernetes with that manifest files
  - Uses the Kubernetes agent
- Starts “tests”
- Leaves test env running for dev
- Merge request to develop branch
- Trigger pipeline to refresh gitops repository and redeploy the test environment
- Variables passed through to gitops pipeline

The application pipeline

Variables:

```
variables:
  APPLICATION_PORT: "8080"
  KUBECTL_IMAGE: "gitlab.example.com:5050/kubernetes/gitops/gitops"
  KUBECTL_CONTEXT_NAME: >
    "kubernetes/gitops:kubernetes-agent"
  DEVNAMESPACE: >
    "${GITLAB_USER_LOGIN}-${CI_COMMIT_REF_SLUG}-${CI_PROJECT_NAME}-ns"
  DEVSVC: >
    "${GITLAB_USER_LOGIN}-${CI_COMMIT_REF_SLUG}-${CI_PROJECT_NAME}-
svc"
  DEVDEPLOYMENT: >
     "${GITLAB_USER_LOGIN}-${CI_COMMIT_REF_SLUG}-${CI_PROJECT_NAME}"
  DEVDOMAIN: "example.com"
  DEVINGRESS: >
    "${GITLAB_USER_LOGIN}-${CI_COMMIT_REF_SLUG}-${CI_PROJECT_NAME}.$
{DEVDOMAIN}"
  DEVINGRESSNAME: "${GITLAB_USER_LOGIN}-${CI_COMMIT_REF_SLUG}-$
{CI_PROJECT_NAME}-ing"
```

The application pipeline

- generate_manifests_pipeline stage

```
generate_manifests:
  stage: generate_manifests_pipeline
  image: $KUBECTL_IMAGE
  script:
    - cd $CI_PROJECT_DIR
    - mkdir manifests
    - envsubst < templates/namespace.yml > manifests/1-namespace.yml
    - envsubst < templates/deployment.yml > manifests/2-deployment.yml
    - envsubst < templates/service.yml > manifests/3-service.yml
    - envsubst < templates/ingress.yml > manifests/4-ingress.yml
  artifacts:
    untracked: true
    expire_in: "1 hour"
    paths:
      - "manifests/"
  rules:
    - if: >
      '$CI_COMMIT_BRANCH   != "develop" &&
       $CI_COMMIT_BRANCH   != "main"    &&
       $CI_PIPELINE_SOURCE != "merge_request_event" &&
       $CI_COMMIT_TAG      == null'
```

The application pipeline

Deployment manifest template

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ${DEVDEPLOYMENT}
spec:
  selector:
    matchLabels:
      app: ${DEVDEPLOYMENT}
  replicas: 1
  template:
    metadata:
      labels:
        app: ${DEVDEPLOYMENT}
    spec:
      containers:
      - name: ${DEVDEPLOYMENT}
        image: ${CI_REGISTRY_IMAGE}:${CI_COMMIT_REF_SLUG}
```

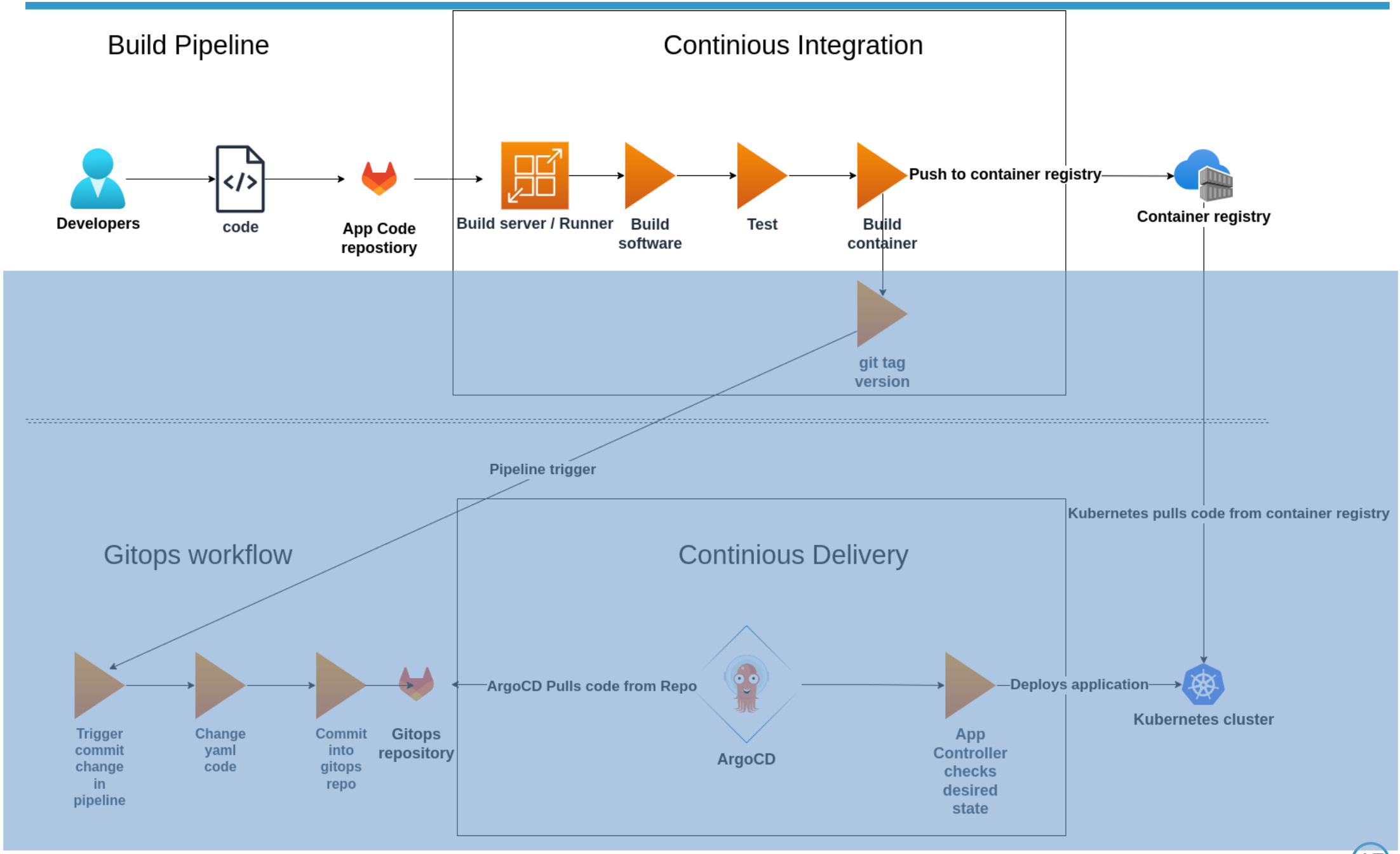
The application pipeline

Deployment manifest output

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: testapp
spec:
  selector:
    matchLabels:
      app: testapp
  replicas: 1
  template:
    metadata:
      labels:
        app: testapp
    spec:
      containers:
      - name: testapp
        image: >
gitlab.hz.alcatrash.cloud/kubernetes/testapp:latest
```

The application pipeline - demo

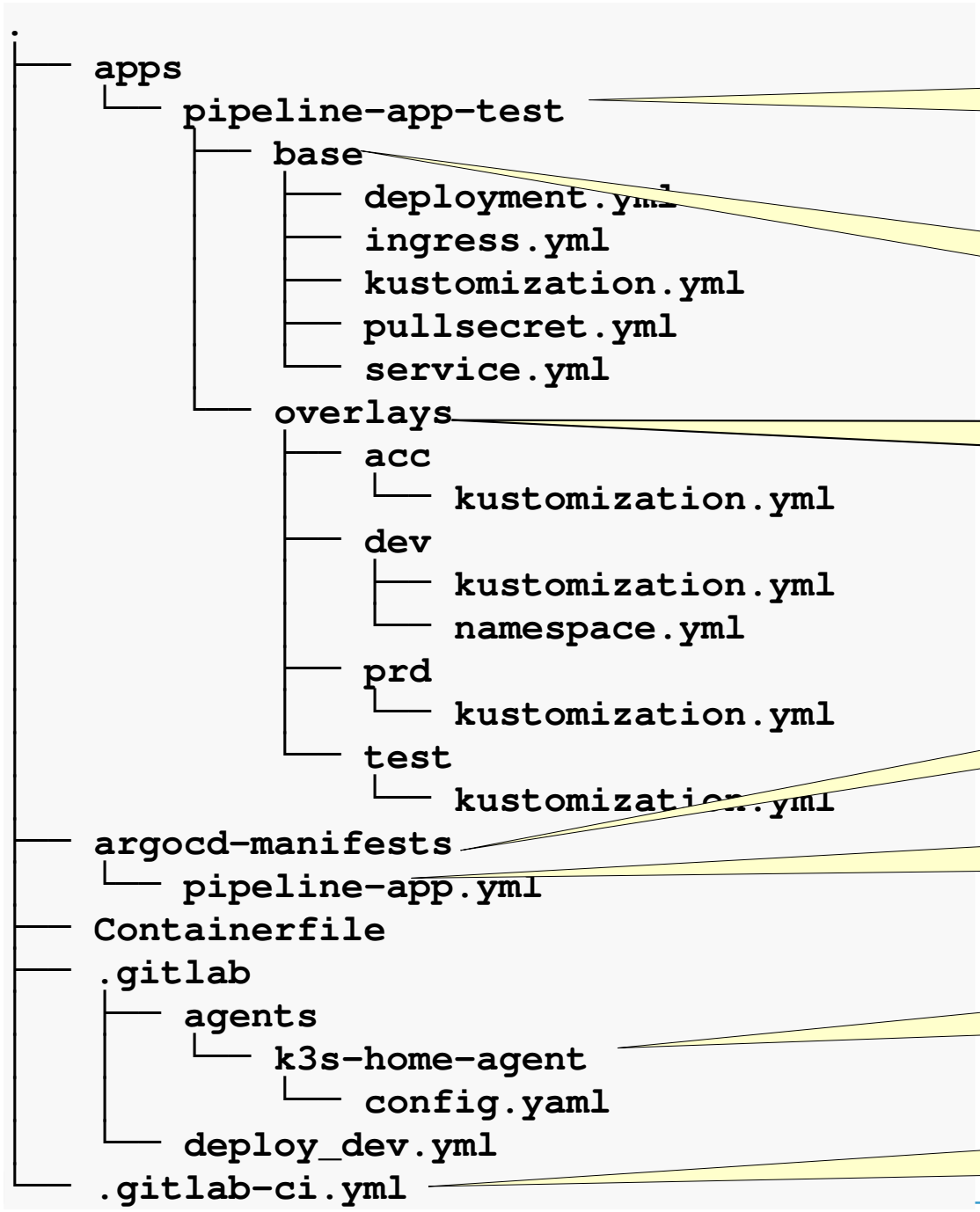
The Gitops repository (1)



The gitops repository (2)

- Holds the gitops code
- Also has a pipeline
  - Receives triggers from the application pipeline
- Pipeline does some things:
  - Redeploys the test environment with the “latest” tag completely
    - Uses the Kubernetes agent for this
  - Triggers a git commit and push for the staging env
  - Triggers a git commit and push for the prod env
- This repository contains a webhook to ArgoCD to refresh it’s state

The Gitops pipeline – Directory layout



App manifest directory

Kustomize base

Kustomize overlay

ArgoCD App manifest

Gitops containerfile

Gitlab Agent config

Pipeline file

The Gitops pipeline

Variables used:

```
variables:
  KUBECTL_IMAGE: >
"gitlab.example.com:5050/kubernetes/gitops/gitops"
  GITOPS_CLONE_URL: >
"git@gitlab.example.com:kubernetes/gitops.git"
  GITOPS_COMMIT_EMAIL: "gitlab@example.com"
  GITOPS_COMMIT_NAME: "GitLab CI/CD"
  GITOPS_KUBECTL_CONTEXT_TST:
"kubernetes/gitops:kubernetes-agent"
```

The Gitops pipeline

deploy_tst stage

```
deploy_tst:
  stage: deploy_tst
  image: $KUBECTL_IMAGE
  script:
    - kubectl config use-context \
      ${GITOPS_KUBECTL_CONTEXT_TST}
    - cd apps/${APPLICATION}/overlays/${ENVIRONMENT}
    - kubectl apply -k .
  rules:
    - if: >
          '$CI_PIPELINE_SOURCE == "pipeline" && \
          $ENVIRONMENT == "tst"'
```

The Gitops pipeline

Triggered from deploy_acc_environment stage in app repo

```
deploy_acc_environment:
  stage: deploy_acc_environment
  trigger: "kubernetes/gitops"
  variables:
    APPLICATION: "$CI_PROJECT_NAME"
    TAG: "$CI_COMMIT_TAG"
    IMAGE: "$CI_REGISTRY_IMAGE"
    ENVIRONMENT: "acc”
  allow_failure: false
  when: manual
  rules:
    - if: $CI_COMMIT_TAG != null
```

The Gitops pipeline

refresh_acc stage

```
refresh_acc:
  stage: refresh_acc
  image: $KUBECTL_IMAGE
  before_script:
    - mkdir -p ~/.ssh
    - echo ${SSH_PRIVATE_KEY_BASE64} | base64 -d > ~/.ssh/id_ed25519
    - git remote set-url origin ${GITOPS_CLONE_URL}
    - git config --global user.email "${GITOPS_COMMIT_EMAIL}"
    - git config --global user.name "${GITOPS_COMMIT_NAME}"
  script:
    - git checkout -B main
    - git pull origin main
    - cd apps/${APPLICATION}/overlays/${ENVIRONMENT}
    - kustomize edit set image $IMAGE:$TAG
    - git commit -am "Automated change by Gitlab CI - Promoted $
{APPLICATION} to version ${TAG} in ${ENVIRONMENT}"
    - git push origin main
  rules:
    - if: '$CI_PIPELINE_SOURCE == "pipeline" && $ENVIRONMENT == "acc"
```

The Gitops pipeline - demo

What can be improved?

- Using Gitlab CI templates
- Create a standard for testing in pipelines
- Explore possibilities to fully automate the development cycle

Lessons learned in the field

- Documentation
- Create a test environment / sandbox environment
- Ask developers what they want
- Describe what the end goal is
- Set proper ACL rights on your project
- This “solution” can be implemented for business that aren´t yet ready for a fully automatic release cycle.

Questions?

Ways to find me:

- M.Trip@ATComputing.nl
- michael@alcatrash.org

- https://michaeltrip.nl
- https://github.com/michaeltrip
- https://linkedin.com/in/michaeltrip/
- https://atcomputing.nl

https://github.com/michaeltrip/cfgmgmtcamp2024-sources/