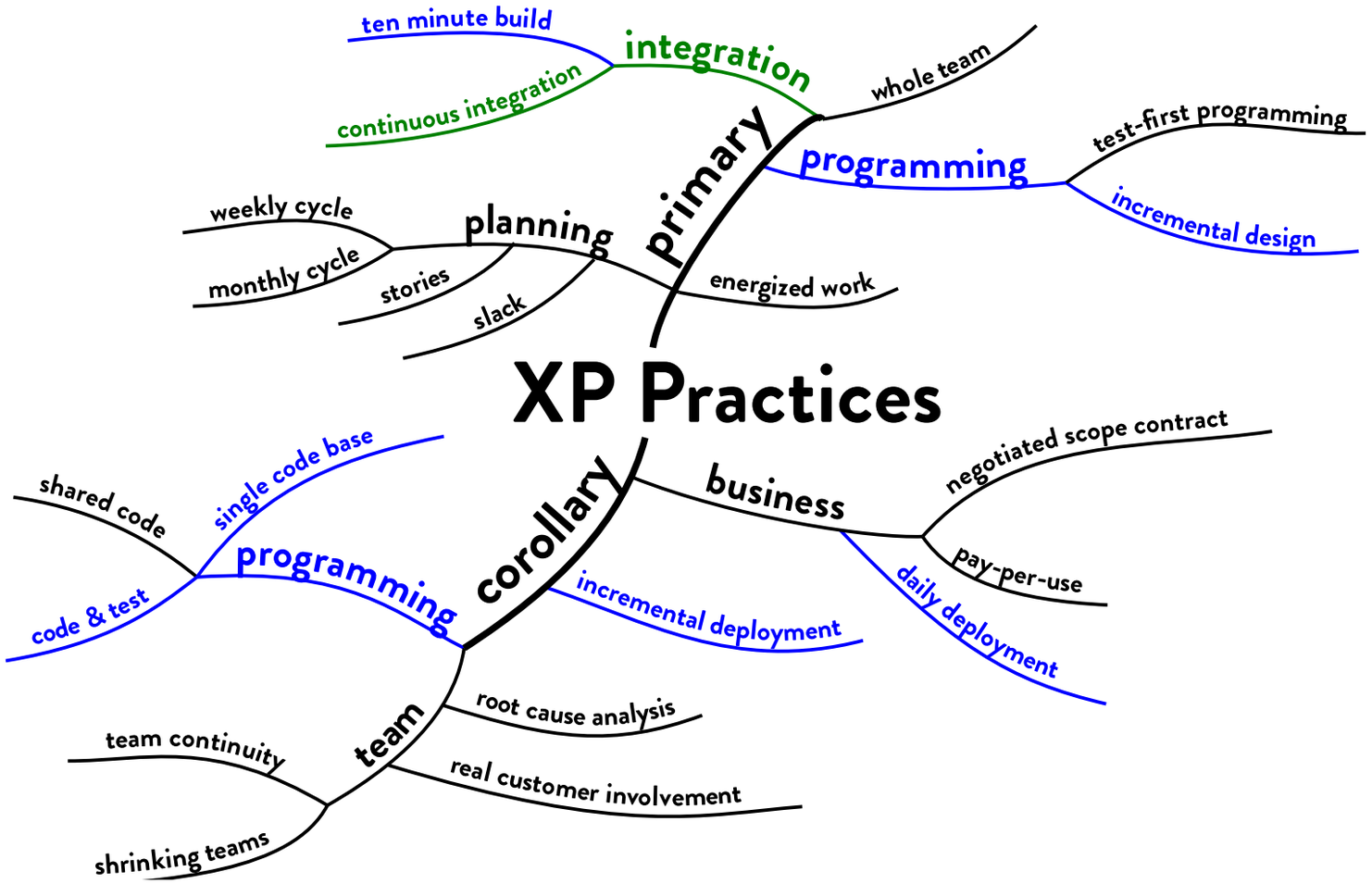
TURNKEY CONTINUOUS DELIVERY

Gianni Bombelli

bombelli@intre.it
https://github.com/bombix
https://www.linkedin.com/in/gianni-bombelli

#IAD2018 - Brescia, 9 Novembre 2018

EXTREME PROGRAMMING



CONTINUOUS INTEGRATION

Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.

CONTINUOUS ...

Kofy Senaya

Director of Product @ Clearbridge Mobile

Continuous Delivery is similar to Continuous Integration. You are building a product that can be released to production at any time. Continuous Delivery requires building testing, and releasing faster and more frequently. Continuous Integration happens before you build as you are testing code. Delivery means you can release something to the staging environment or the pre-production environment.

Continuous Delivery is when your code is always ready to be released but isn't pushed to production unless you make the decision to do so. It is a manual step. With Continuous Deployment, any updated working version of the app is automatically pushed to production.

TOOLS

GitLab

ANSIBLE

WHAT DO WE DO NOW?

- bootstrap our Continuous Delivery and test environment
- configure the tools necessary to manage and build our software
- create our ~~build~~ Continuous Delivery pipeline
- develop a simple micro-service
  - REST server
  - written in ES6
  - executed with Node.js
  - released as a docker image

REPOSITORIES

Build and Release Infrastructure

https://github.com/bombix/workshop-turnkey

Example Project

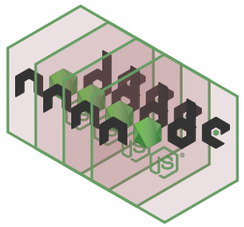
https://github.com/intresrl/microservice-example-hex2hsl

- Remove the remote named origin
- Checkout the tag start
- Reset master branch to the commit with the start tag

INFRASTRUCTURE

Turnkey CD

Testing

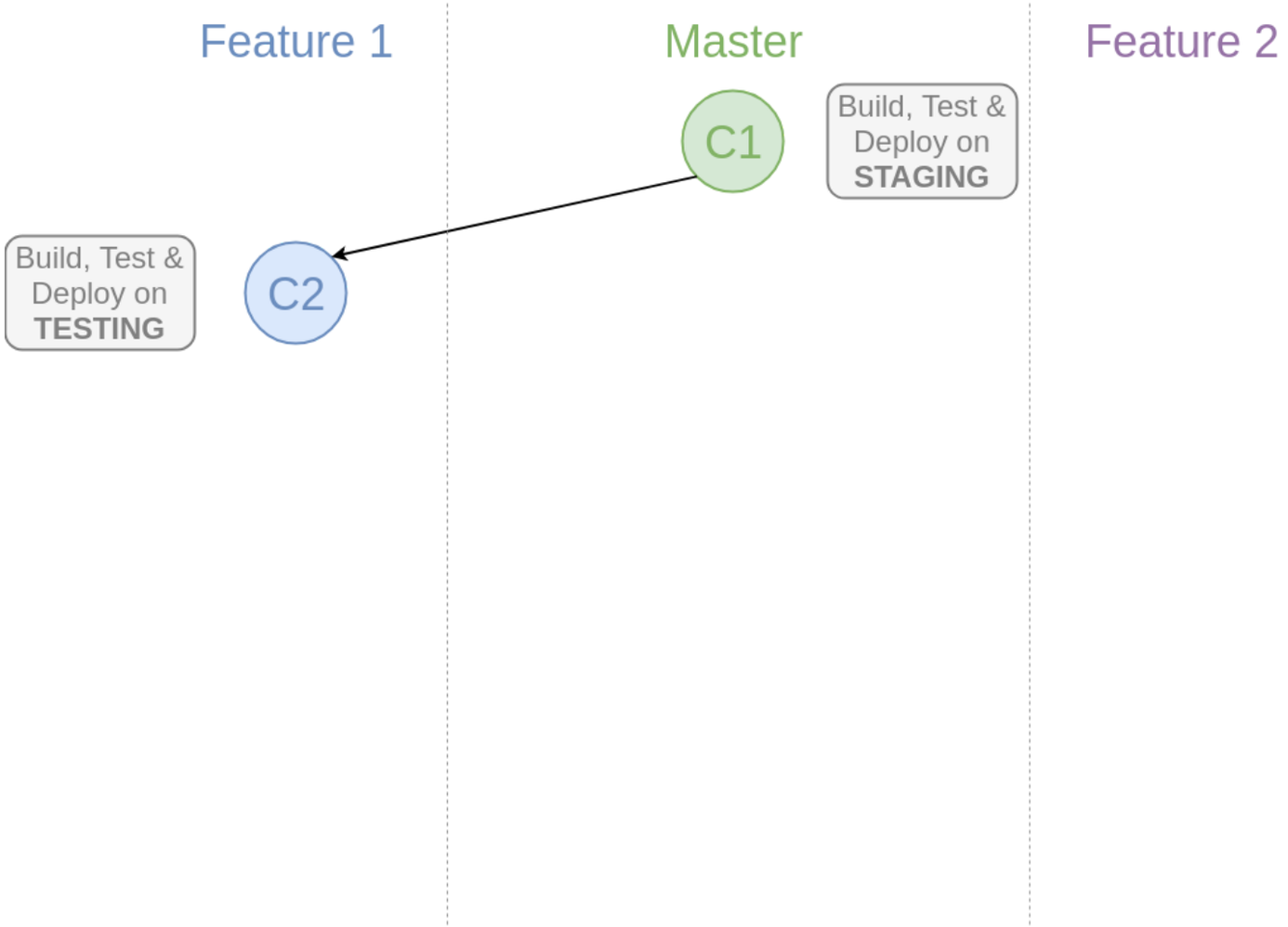


Staging

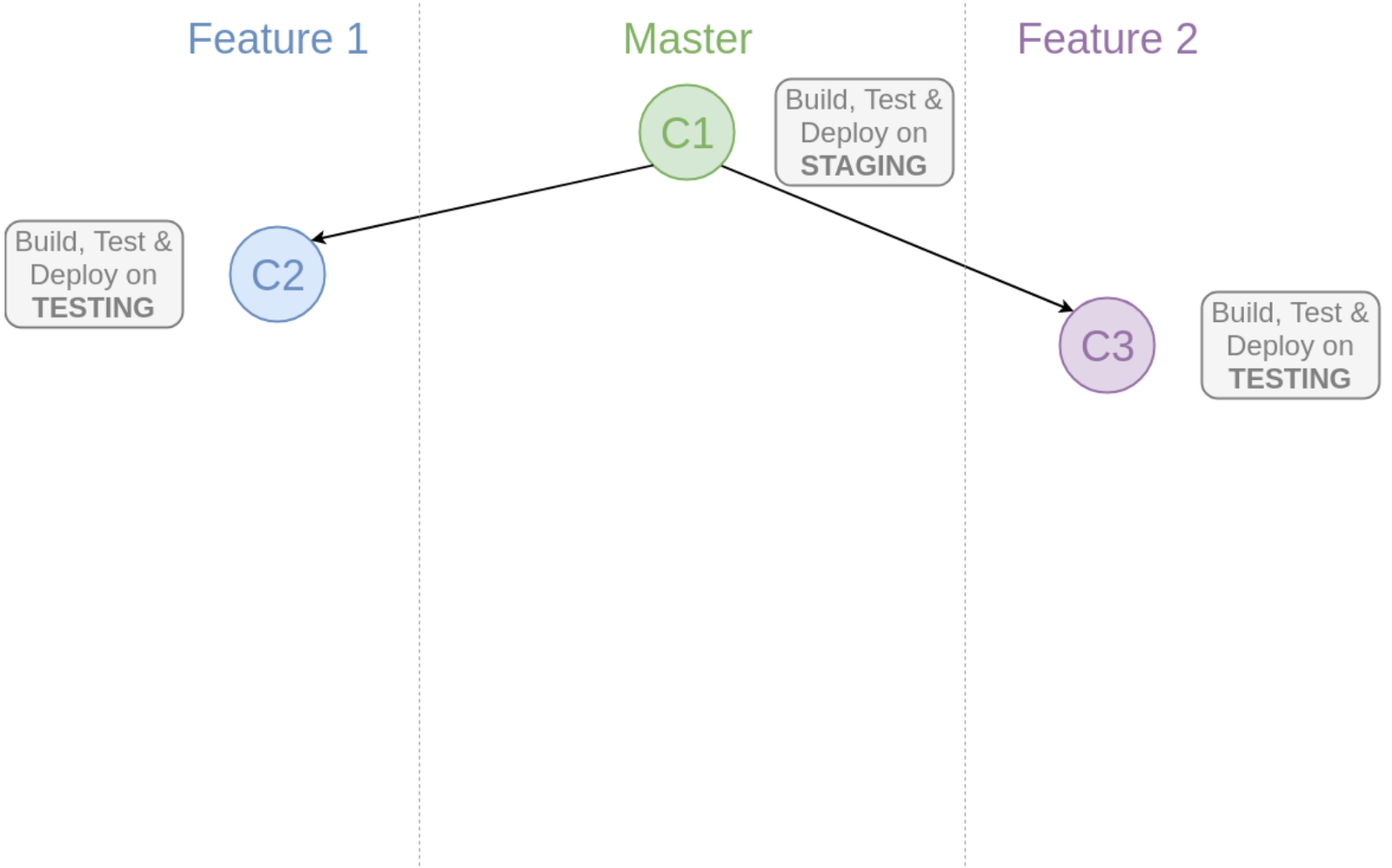
Production

INFRASTRUCTURE AS CODE

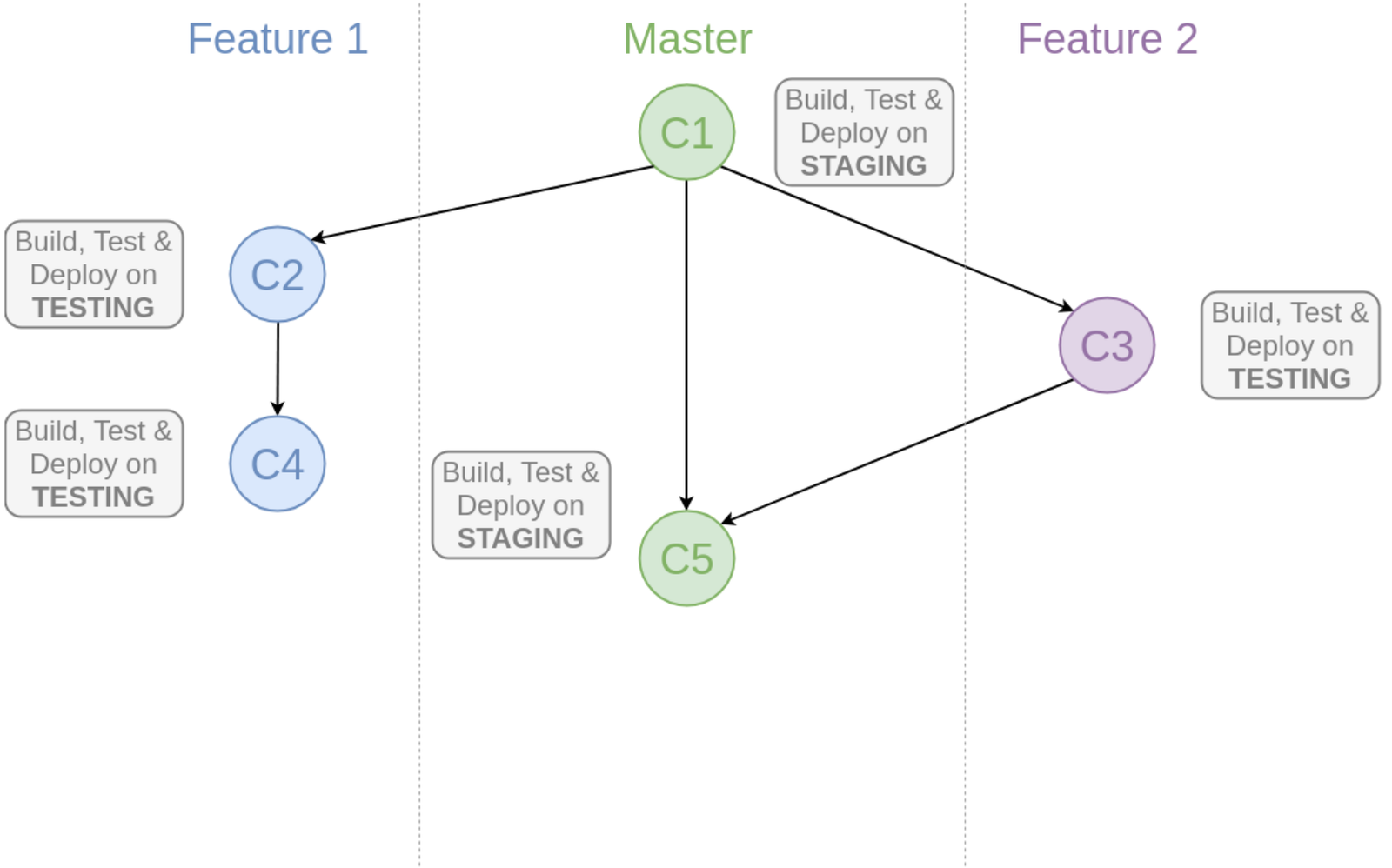
WORKFLOW - F.1


Feature 1
Master
Feature 2
C1
Build, Test & Deploy on **STAGING**
Build, Test & Deploy on **TESTING**
C2

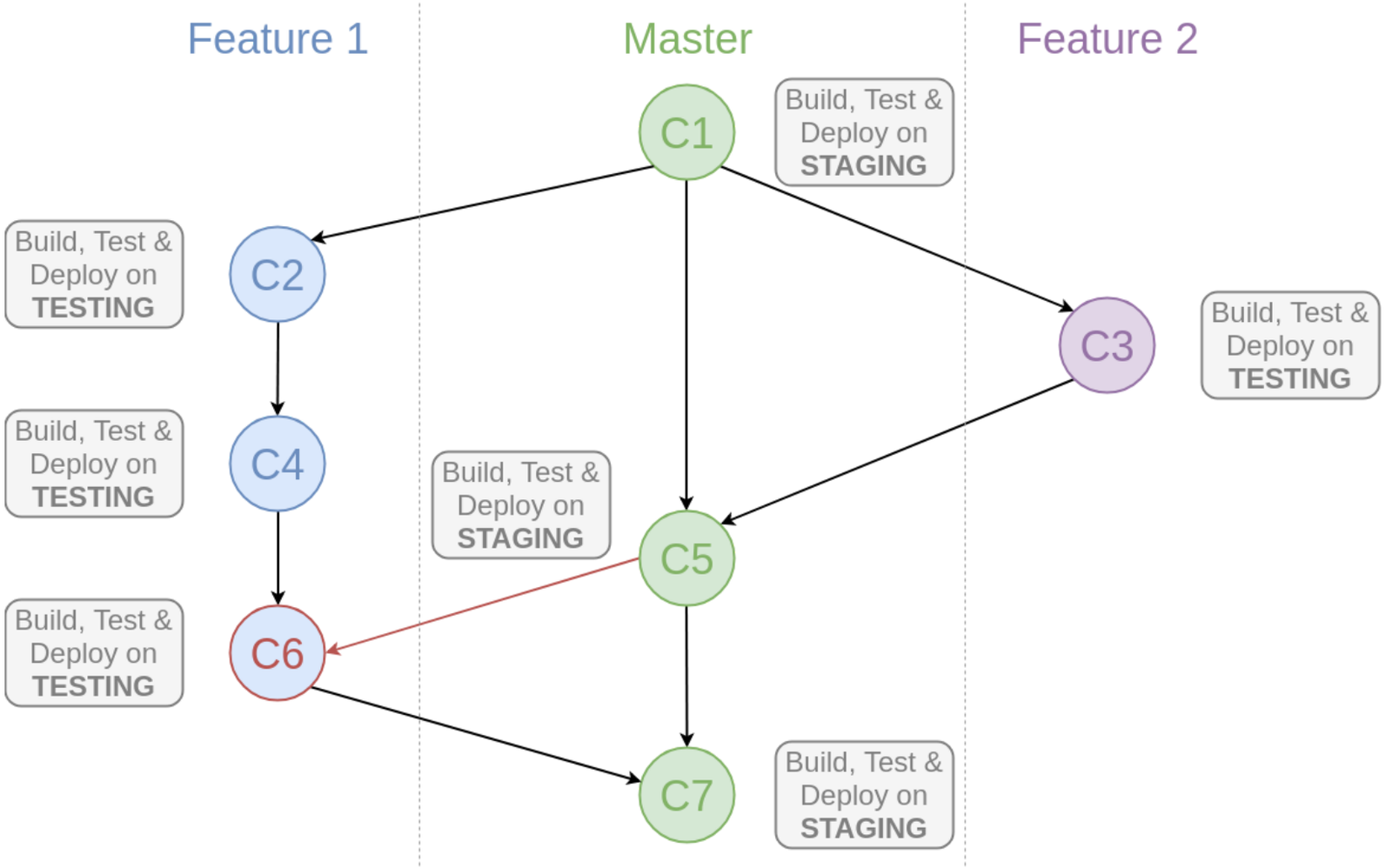
WORKFLOW - F.2


Feature 1
Master
Feature 2
C1
Build, Test & Deploy on **STAGING**
Build, Test & Deploy on **TESTING**
C2
C3
Build, Test & Deploy on **TESTING**

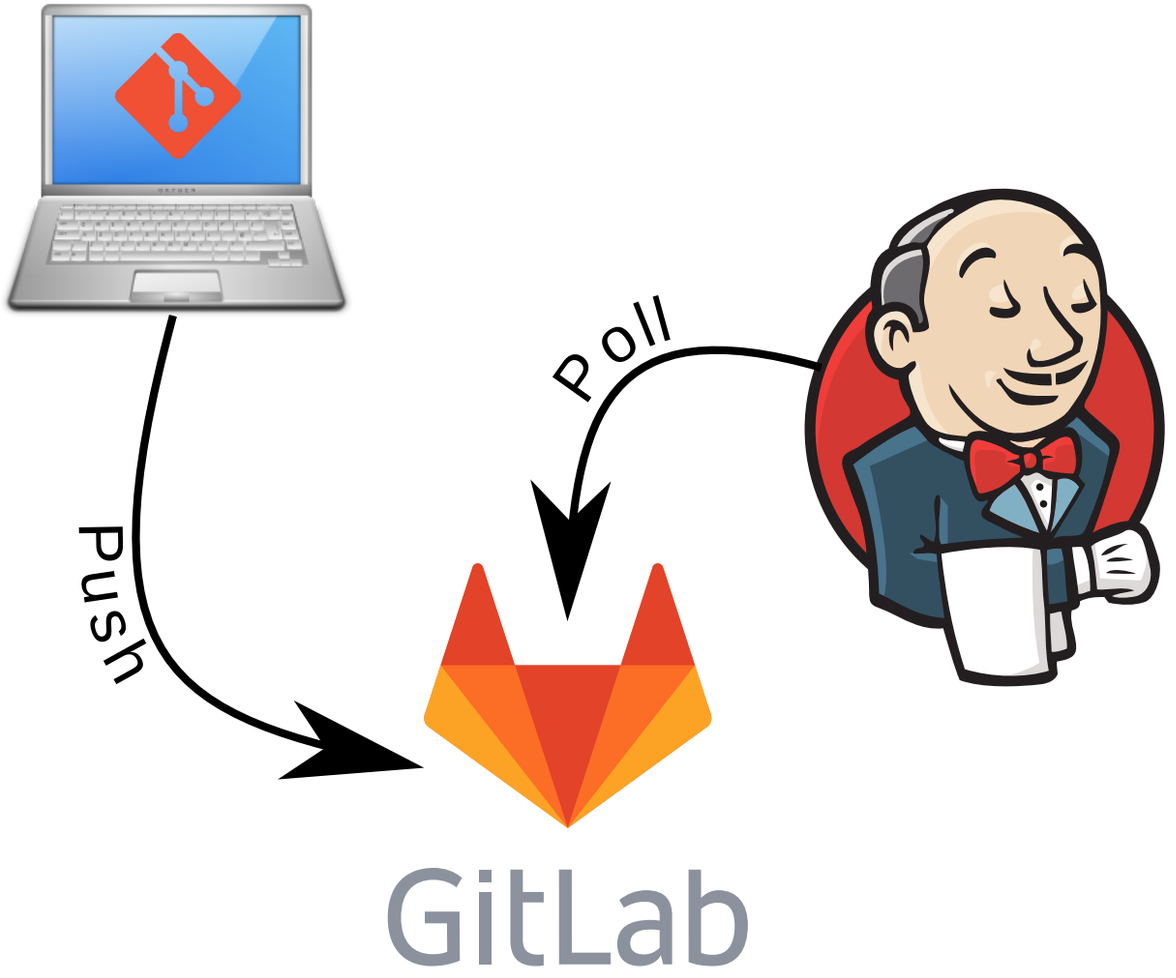
WORKFLOW - F.2 MERGE



WORKFLOW - F.1 MERGE



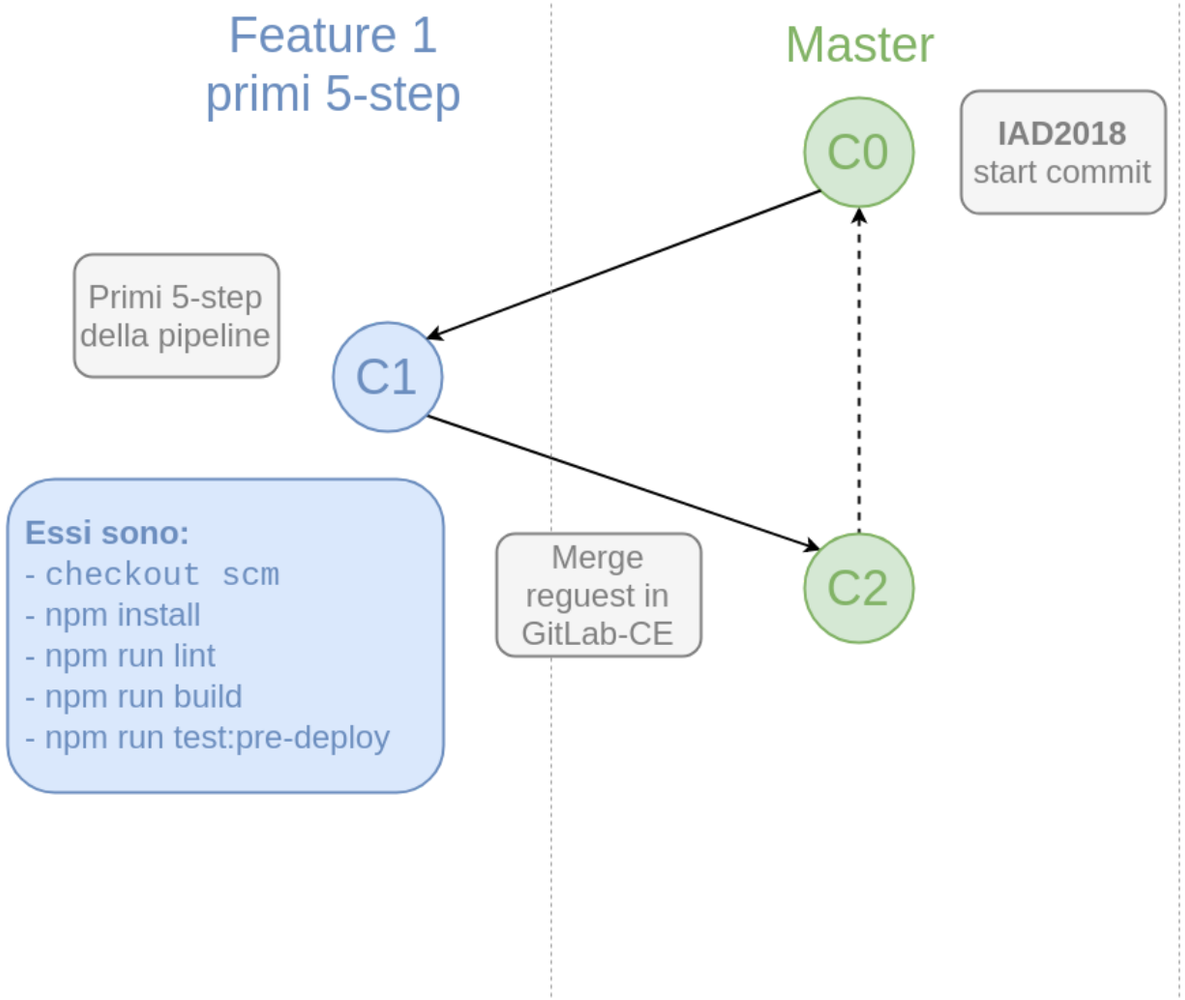
BUILD AUTOMATION



Checkout Source Code
Static Code Analysis
Build
Test pre-deploy
Build Artifact
Push Artifact to Repo
Deploy & Run
Test post-deploy
Cleanup

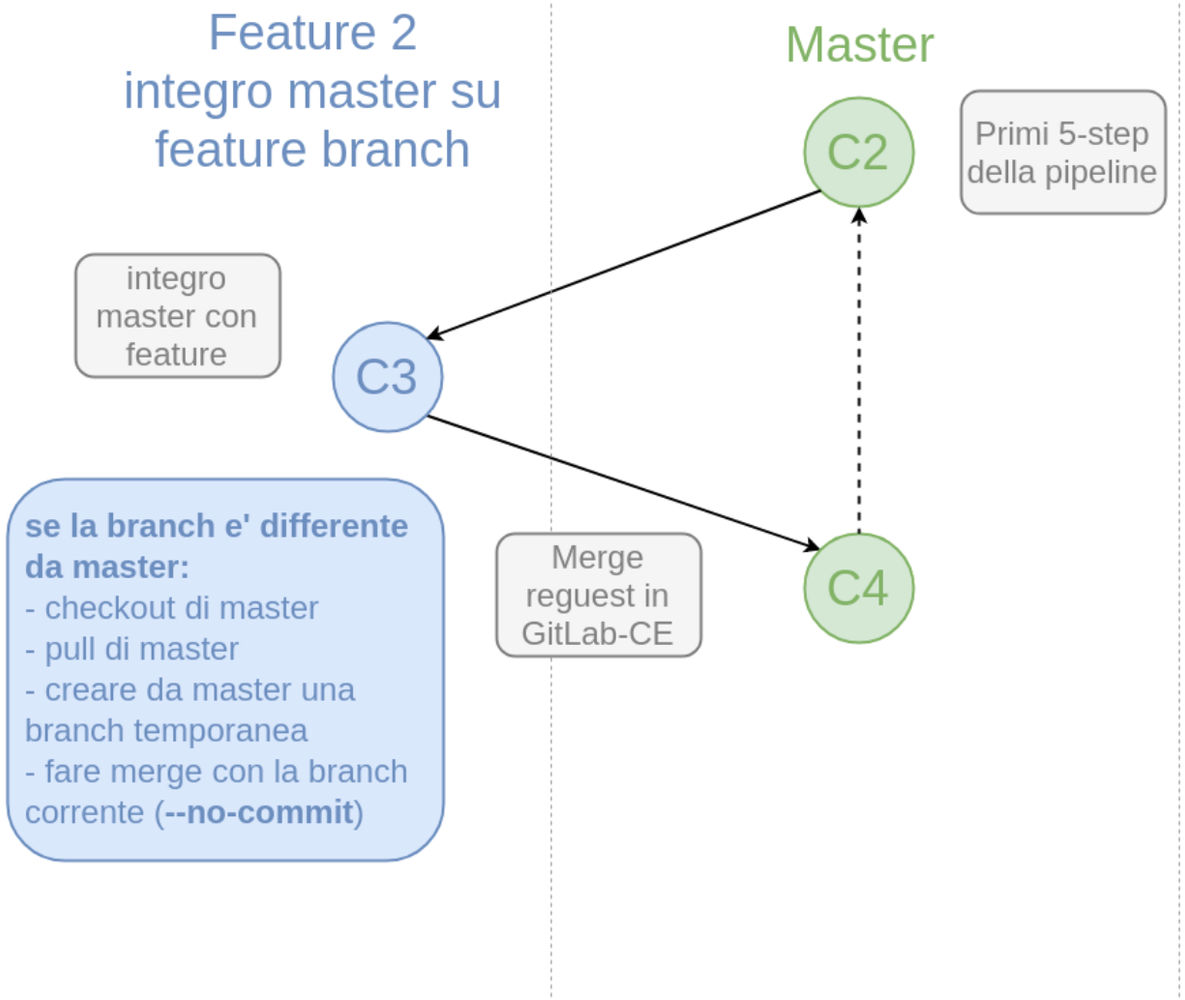
ESERCIZIO 0

ESERCIZIO 0


Feature 1
primi 5-step
Master
C0
IAD2018
start commit
Primi 5-step
della pipeline
C1
Essi sono:
- checkout scm
- npm install
- npm run lint
- npm run build
- npm run test:pre-deploy
Merge
reguest in
GitLab-CE
C2

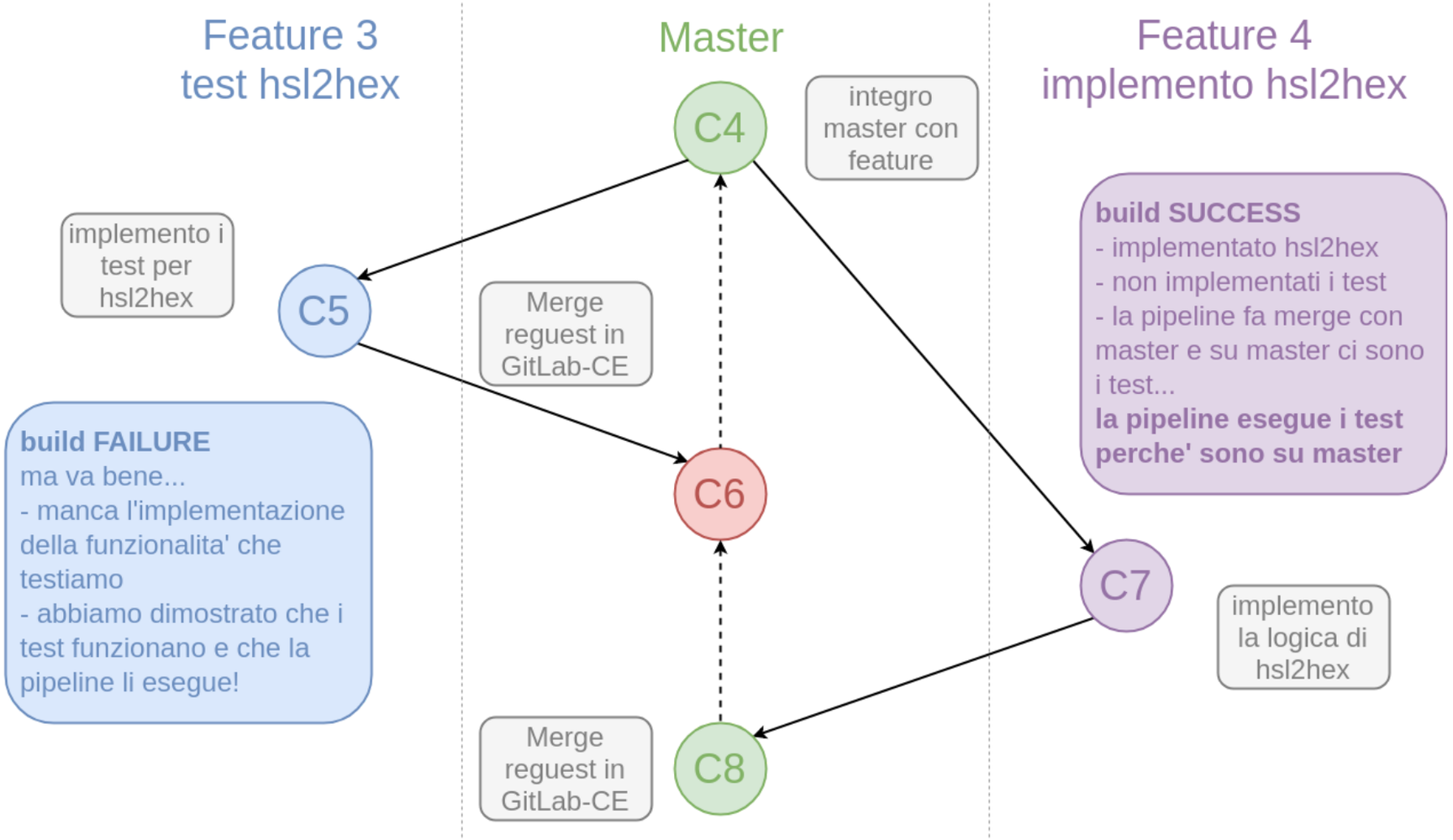
ESERCIZIO 1

ESERCIZIO 1

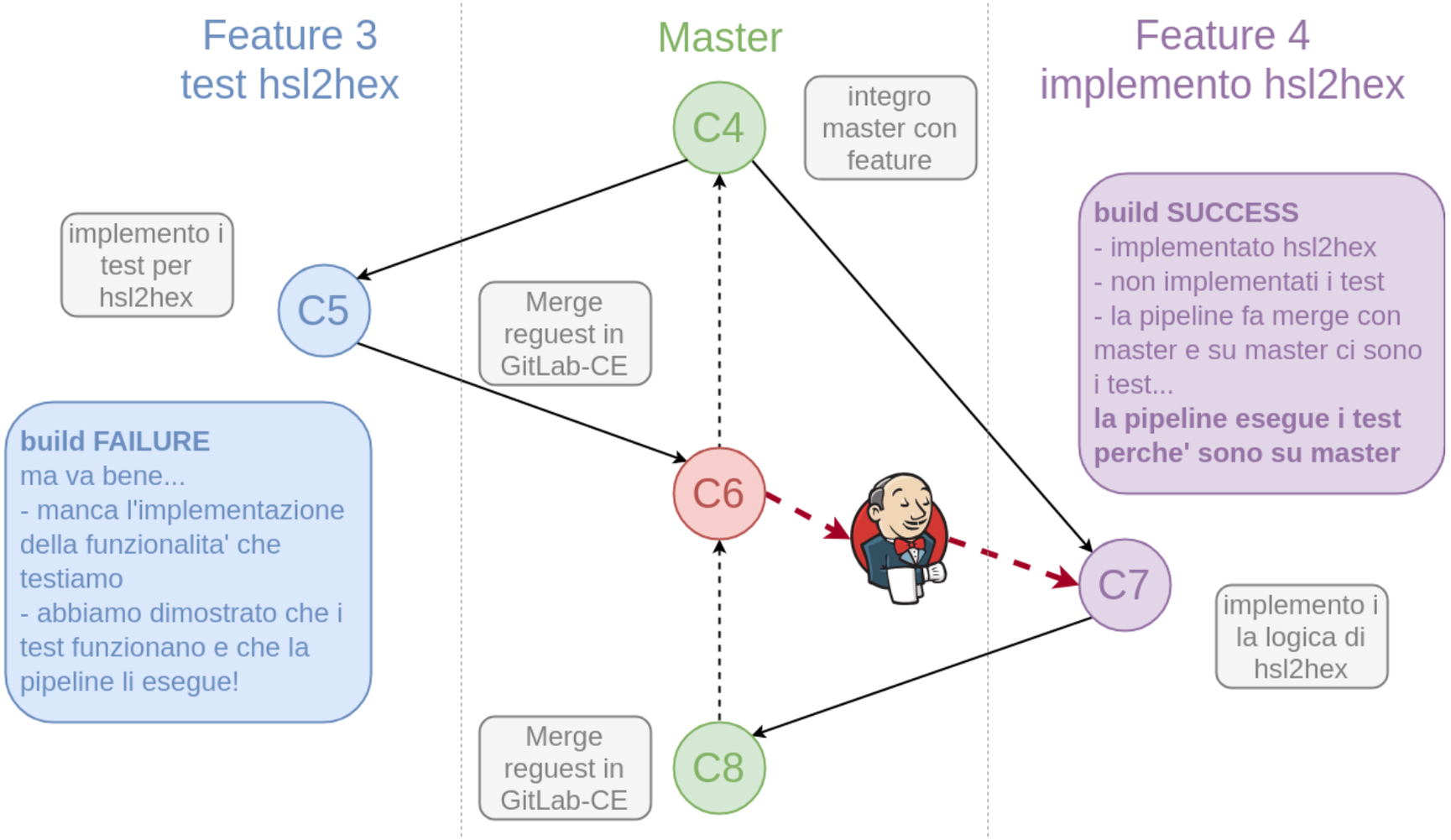

Feature 2
integro master su
feature branch
Master
C2
Primi 5-step
della pipeline
integro
master con
feature
C3
C4
Merge
reguest in
GitLab-CE
se la branch e' differente
da master:
- checkout di master
- pull di master
- creare da master una
branch temporanea
- fare merge con la branch
corrente (--no-commit)

ESERCIZIO 2

ESERCIZIO 2

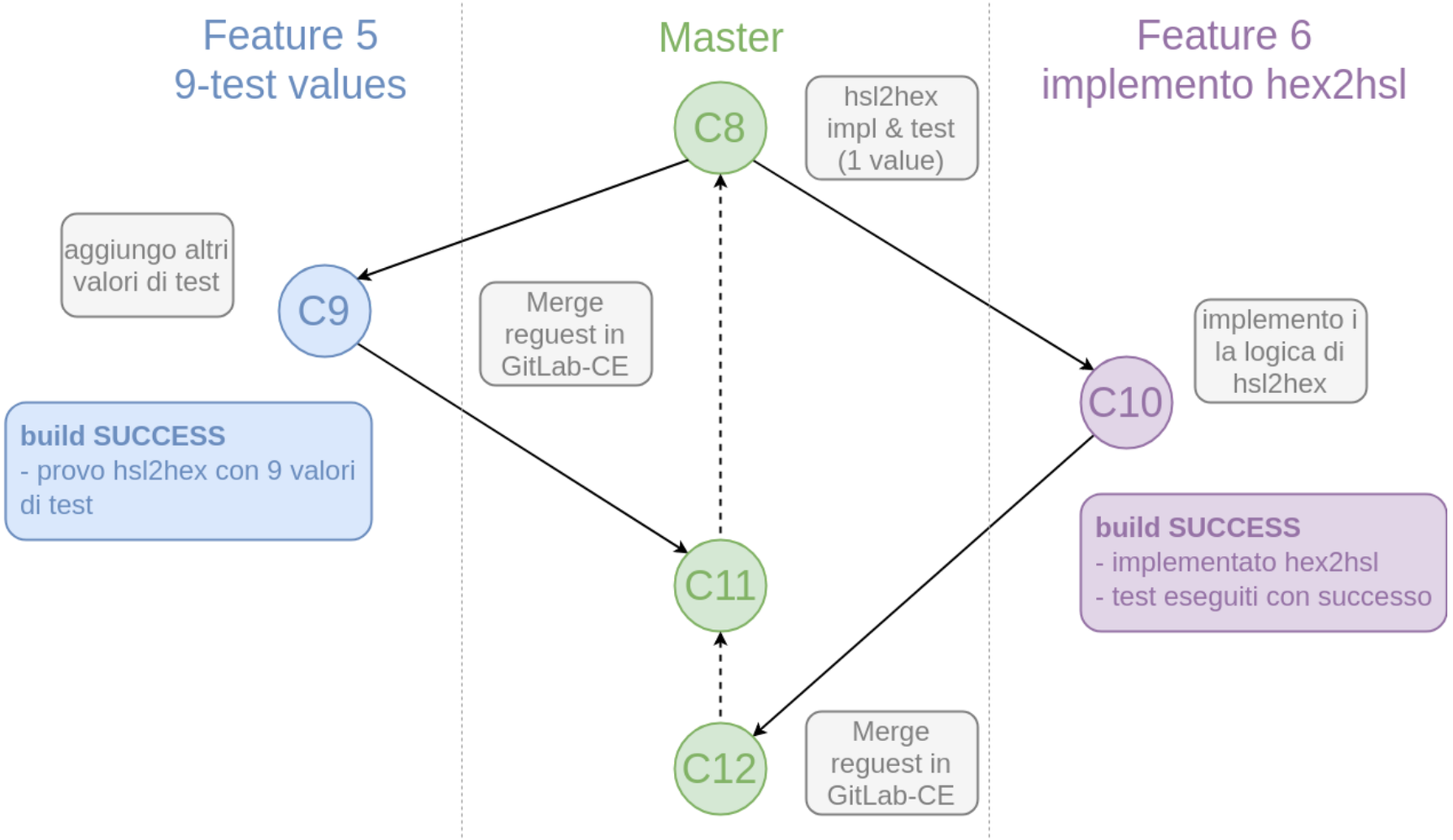

Feature 3
test hsl2hex
Master
Feature 4
implemento hsl2hex
C4
integro master con feature
implemento i test per hsl2hex
C5
Merge reguest in GitLab-CE
build SUCCESS
- implementato hsl2hex
- non implementati i test
- la pipeline fa merge con master e su master ci sono i test...
la pipeline esegue i test perche' sono su master
build FAILURE
ma va bene...
- manca l'implementazione della funzionalita' che testiamo
- abbiamo dimostrato che i test funzionano e che la pipeline li esegue!
C6
C7
implemento la logica di hsl2hex
Merge reguest in GitLab-CE
C8

ESERCIZIO 2

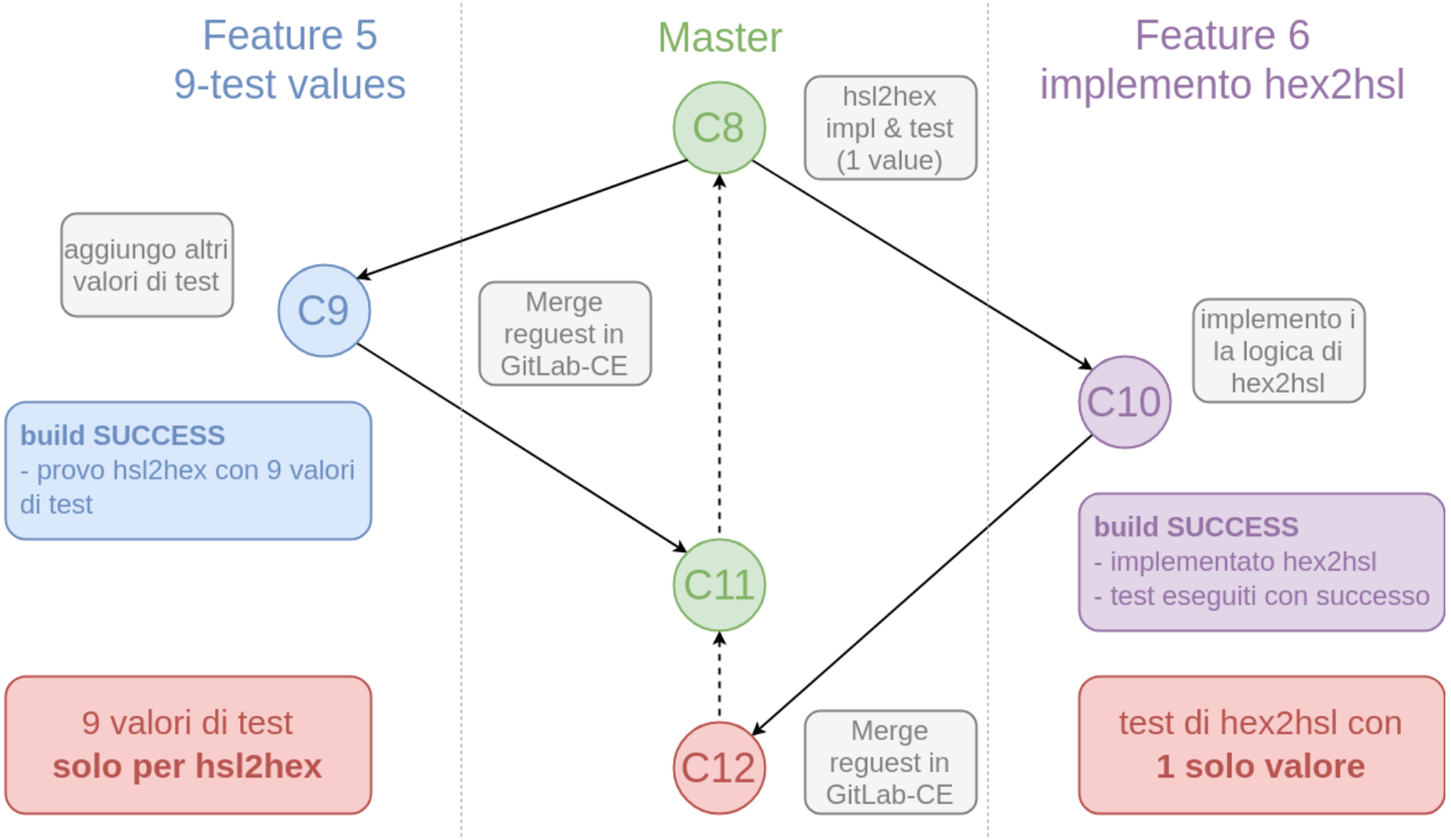

Feature 3
test hsl2hex
Master
Feature 4
implemento hsl2hex
C4
integro master con feature
implemento i test per hsl2hex
C5
Merge reguest in GitLab-CE
build SUCCESS
- implementato hsl2hex
- non implementati i test
- la pipeline fa merge con master e su master ci sono i test...
la pipeline esegue i test perche' sono su master
build FAILURE
ma va bene...
- manca l'implementazione della funzionalita' che testiamo
- abbiamo dimostrato che i test funzionano e che la pipeline li esegue!
C6
C7
implemento i la logica di hsl2hex
Merge reguest in GitLab-CE
C8

ESERCIZIO 3

ESERCIZIO 3

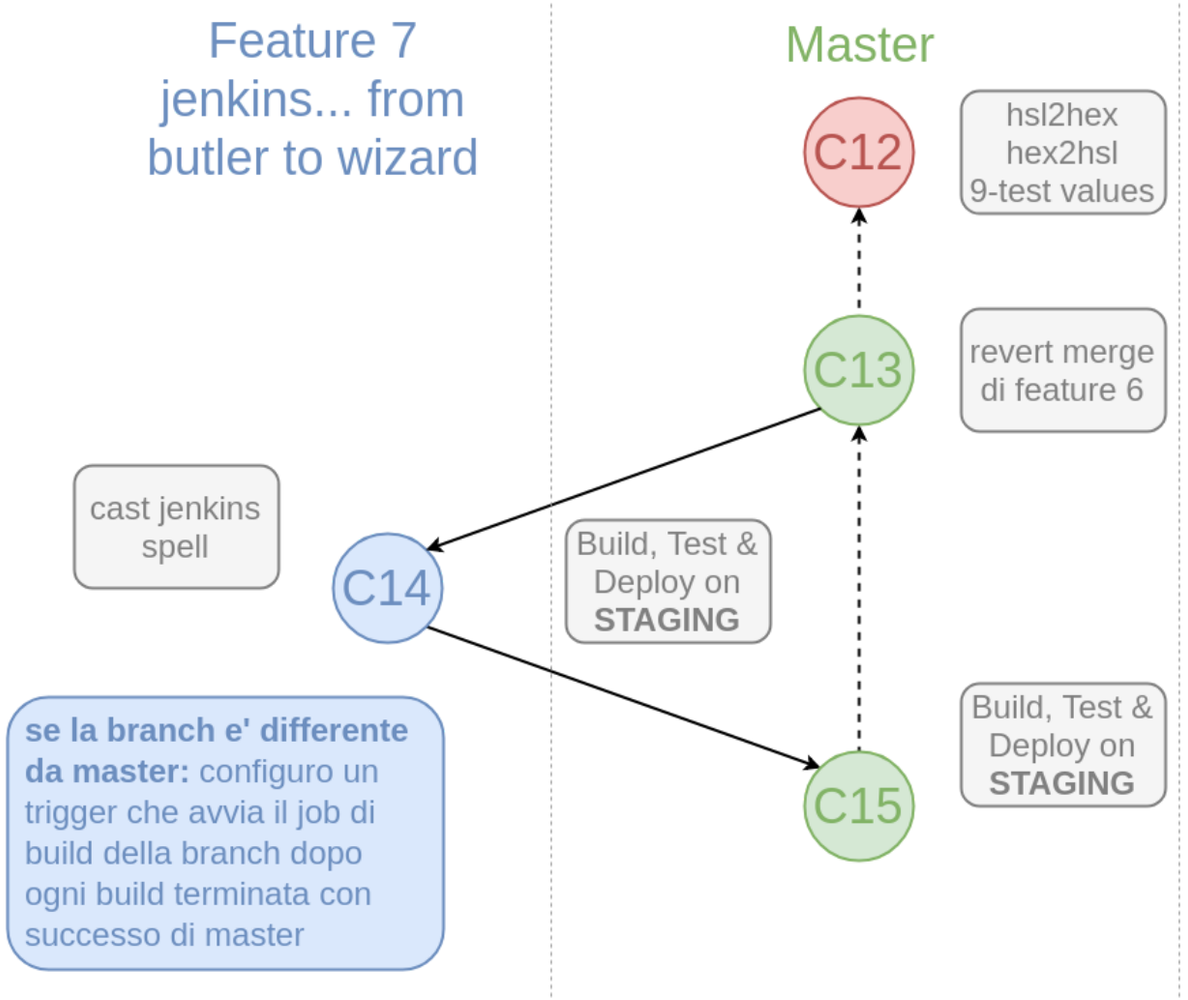

Feature 5
9-test values
Master
Feature 6
implemento hex2hsl
C8
hsl2hex
impl & test
(1 value)
aggiungo altri
valori di test
C9
Merge
reguest in
GitLab-CE
C10
implemento i
la logica di
hsl2hex
build SUCCESS
- provo hsl2hex con 9 valori
di test
C11
build SUCCESS
- implementato hex2hsl
- test eseguiti con successo
C12
Merge
reguest in
GitLab-CE

ESERCIZIO 3


Feature 5
9-test values
Master
Feature 6
implemento hex2hsl
C8
hsl2hex
impl & test
(1 value)
aggiungo altri
valori di test
C9
Merge
reguest in
GitLab-CE
C10
implemento i
la logica di
hex2hsl
build SUCCESS
- provo hsl2hex con 9 valori
di test
C11
build SUCCESS
- implementato hex2hsl
- test eseguiti con successo
9 valori di test
solo per hsl2hex
C12
Merge
reguest in
GitLab-CE
test di hex2hsl con
1 solo valore

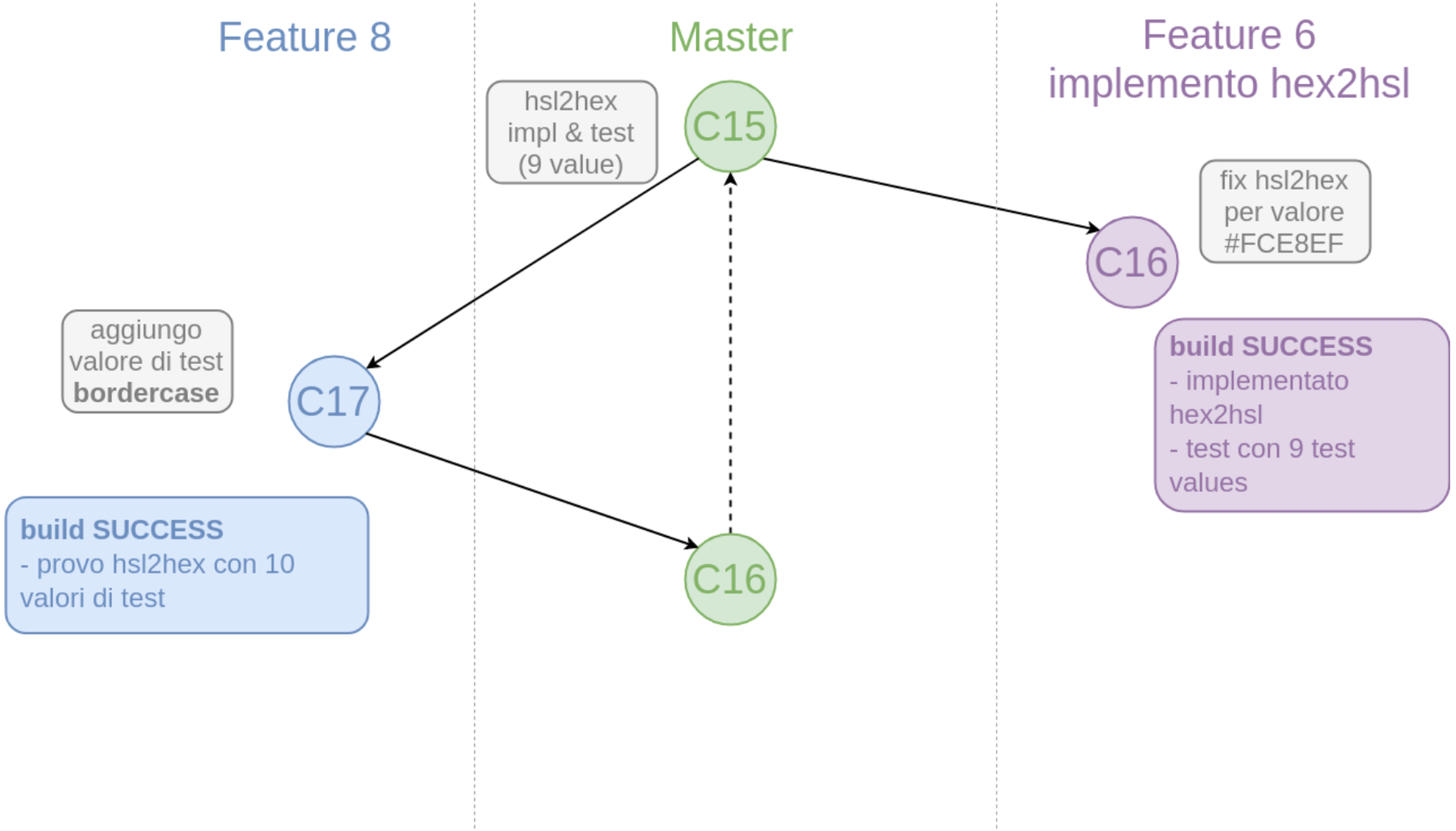
ESERCIZIO 4

ESERCIZIO 4

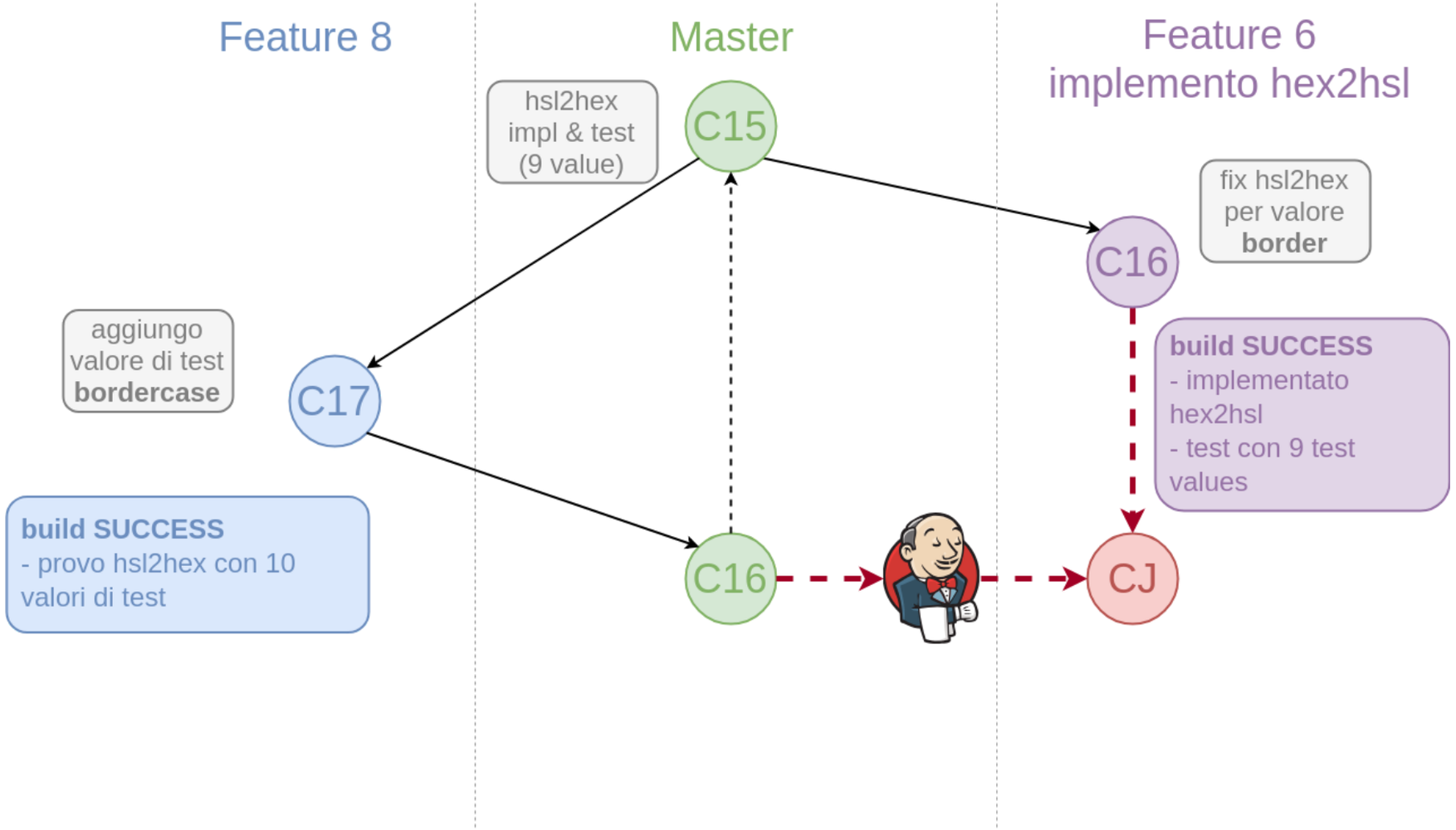

Feature 7
jenkins... from
butler to wizard
Master
C12
hsl2hex
hex2hsl
9-test values
C13
revert merge
di feature 6
cast jenkins
spell
C14
Build, Test &
Deploy on
STAGING
C15
Build, Test &
Deploy on
STAGING
se la branch e' differente da master: configuro un trigger che avvia il job di build della branch dopo ogni build terminata con successo di master

ESERCIZIO 5

ESERCIZIO 5


Feature 8
Master
Feature 6
implemento hex2hsl
hsl2hex
impl & test
(9 value)
C15
fix hsl2hex
per valore
#FCE8EF
C16
aggiungo
valore di test
bordercase
C17
build SUCCESS
- implementato
hex2hsl
- test con 9 test
values
build SUCCESS
- provo hsl2hex con 10
valori di test
C16

ESERCIZIO 5


Feature 8
Master
Feature 6
implemento hex2hsl
hsl2hex
impl & test
(9 value)
C15
fix hsl2hex
per valore
border
C16
aggiungo
valore di test
bordercase
C17
build SUCCESS
- implementato
hex2hsl
- test con 9 test
values
build SUCCESS
- provo hsl2hex con 10
valori di test
C16
CJ

IT'S ENOUGH ?

- single shared code-base
- automated build & merge different branches
- self-testing build
- integrate every commit
- ten-minutes build & fail fast!
- everyone can see what's happening
- end-to-end test
- automated deployment
- test the deployment
- test in a production-like environment

NEXT STEPS ...

- Build Artifact
- Push to Artifact Repository
- Deploy on a test or staging server
- Run it
- Execute the post-deployment tests

BUILD ARTIFACT AND PUSH

```
stage('Build Docker Image') {
    def container-name = ${pipelineName}-${env.BRANCH_NAME}
    sh "docker -H tcp://192.168.50.91:2375 build \
        -t 192.168.50.91:5000/${container-name}:${env.BUILD_ID} \
        -t 192.168.50.91:5000/${container-name}:latest ."
}

stage('Push to Docker Registry') {
    def container-name = ${pipelineName}-${env.BRANCH_NAME}
    sh "docker -H tcp://192.168.50.91:2375 \
        push 192.168.50.91:5000/${container-name}:${env.BUILD_ID}"
    sh "docker -H tcp://192.168.50.91:2375 \
        push 192.168.50.91:5000/${container-name}:latest"
}
```

DEPLOY & RUN

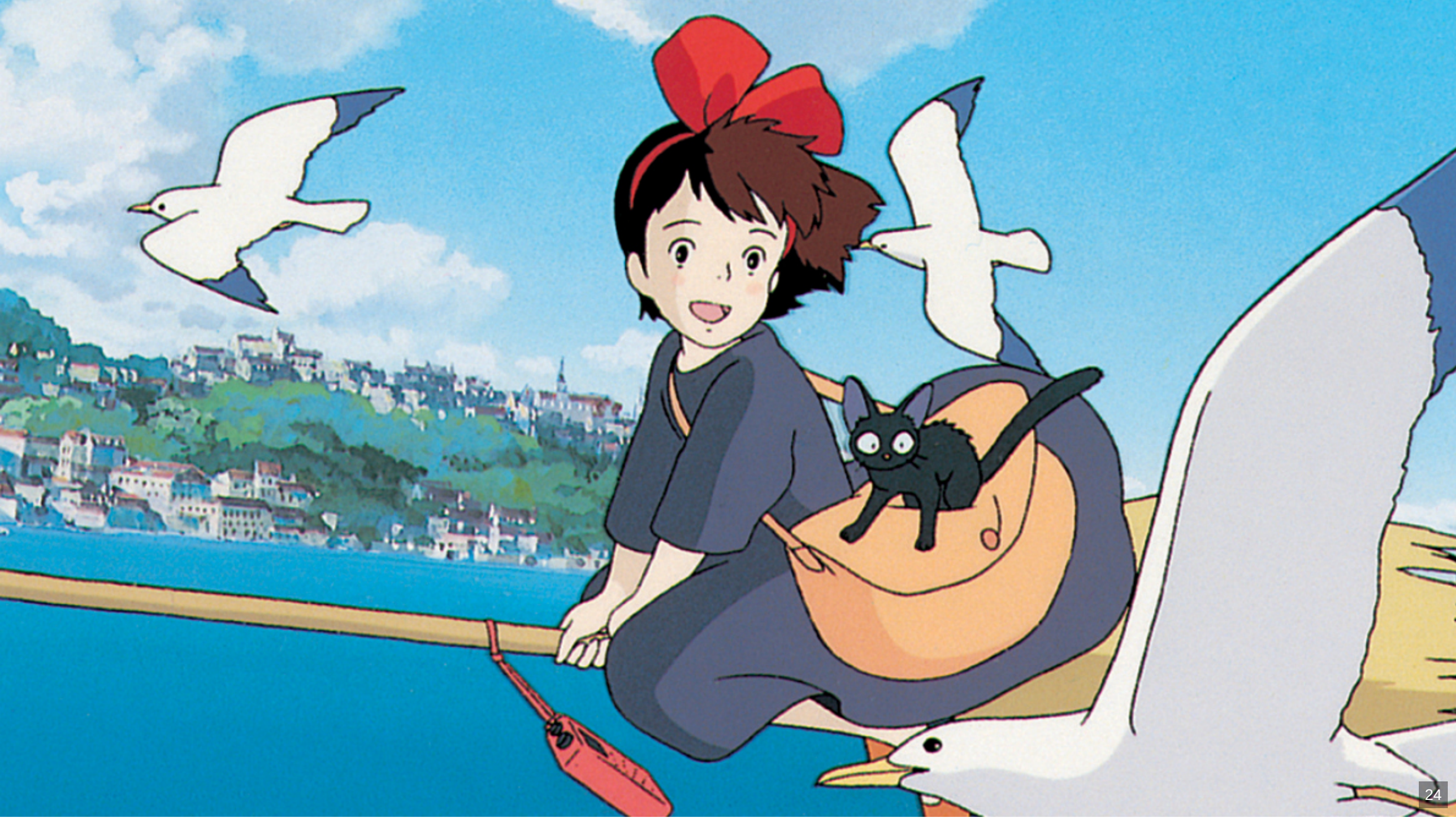
```
stage ('Deploy and Run') {
    if (env.BRANCH_NAME == 'master') {
        print "Deploy docker container to : staging environmet"
        sh "ansible-playbook -i /ansible/inventory/hosts.yml \
            /ansible/hex2hsl.yml --limit staging"
    } else {
        print "Deploy docker container to : testing environmet"
        sh "ansible-playbook -i /ansible/inventory/hosts.yml \
            --extra-vars 'hex2hsl_branch=${env.BRANCH_NAME} hex2hsl_port=${test_
            /ansible/hex2hsl.yml --limit testing"
    }
}
```

TEST POST-DEPLOY

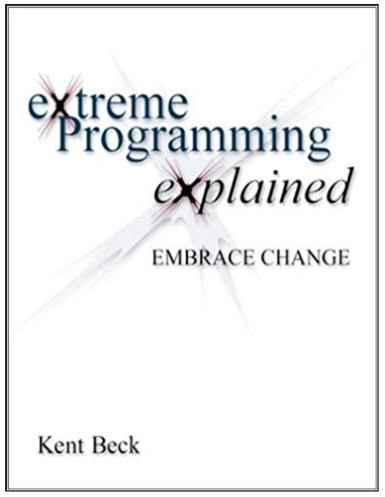
```
stage('Test post-deploy') {
    def test_url = ''
    if (env.BRANCH_NAME == 'master') {
        test_url = 'http://192.168.50.93:3100'
    } else {
        test_url = 'http://192.168.50.92:' + test_port
    }
    sh "npm run test:post-deploy -- --test_url=${test_url}"

    step([$class: 'XUnitBuilder',
        thresholds: [[$class: 'FailedThreshold', unstableThreshold: '1']],
        tools: [[$class: 'JUnitType',
            pattern: 'test-report/test-post-deploy-report.xml']]])
    }
}
```

AND NOW, IT'S ENOUGH ?

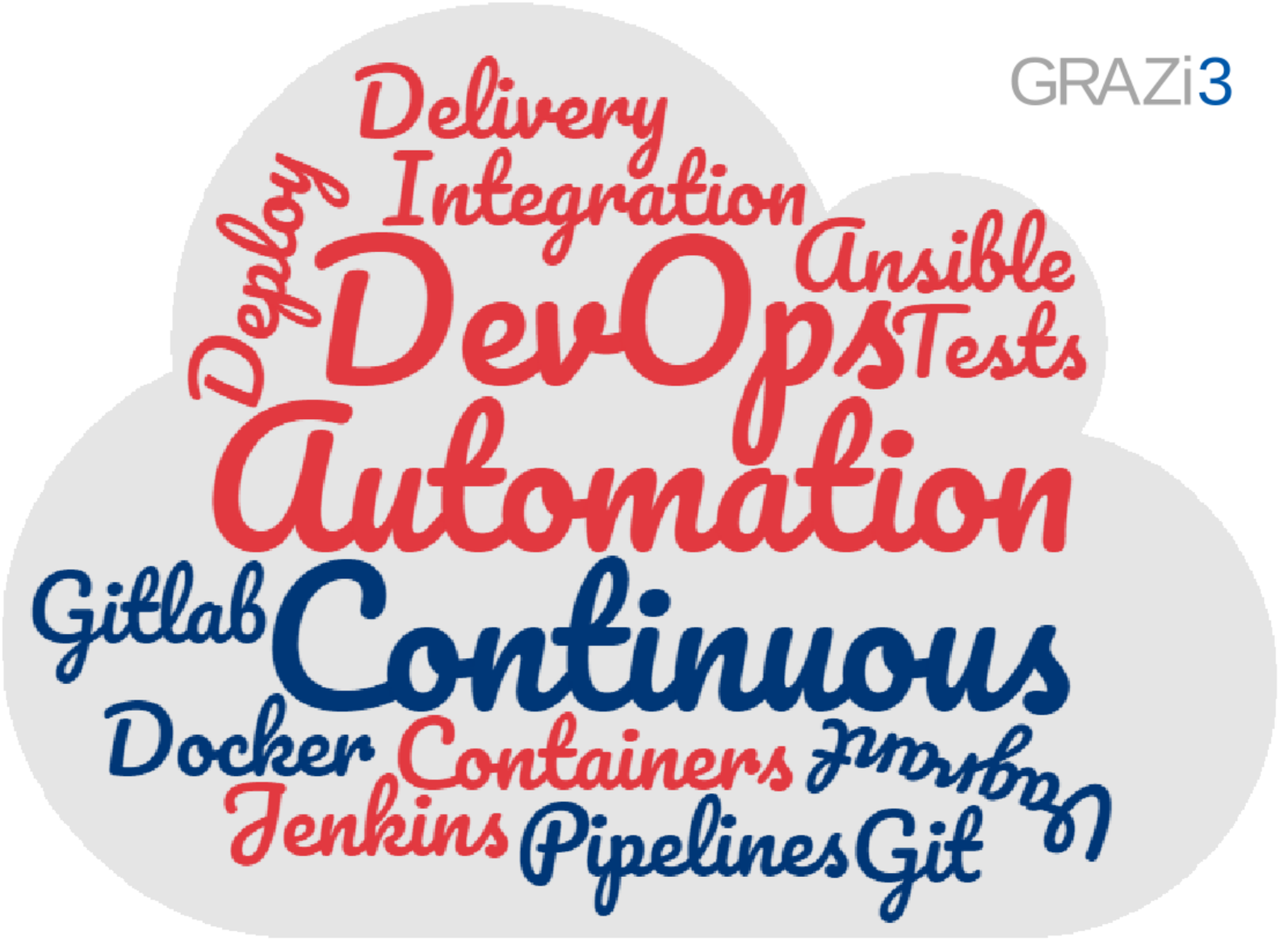
- Regression tests: we always test everything
- e2e tests: yes, post-deploy tests runs on staging
- Security scans: only dependency audit performed by npm
- Performance test: nope!
- Deployment test: yes, deploy on staging environment



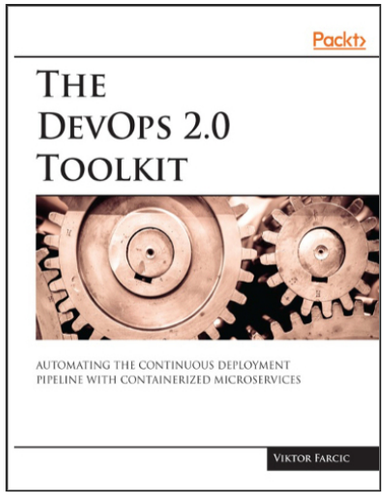

Special thanks to
Ferdinando Santacroce
@jesuswasrasta

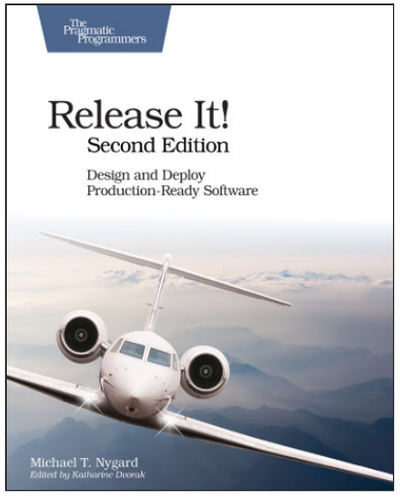

extreme
Programming
explained
EMBRACE CHANGE
Kent Beck


MARTINFOWLER.COM
Continuous Integration
Continuous integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly. This article is a quick overview of Continuous Integration summarizing the technique and its current usage.
01 May 2006
Martin Fowler
Contents


Delivery
Integration
Deploy
DevOps
Ansible
Tests
Automation
Gitlab
Continuous
Docker
Containers
Jenkins
Pipelines
Git

GRAZi3


Packt
THE
DEVOPS 2.0
TOOLKIT
AUTOMATING THE CONTINUOUS DEPLOYMENT
PIPELINE WITH CONTAINERIZED MICROSERVICES
VIKTOR FARCIC


The Pragmatic Programmers
Release It!
Second Edition
Design and Deploy
Production-Ready Software
Michael T. Nygard
Edited by Katharine Dvorak