

Progettazione & Implementazione CI/CD



Restful Booker Platform

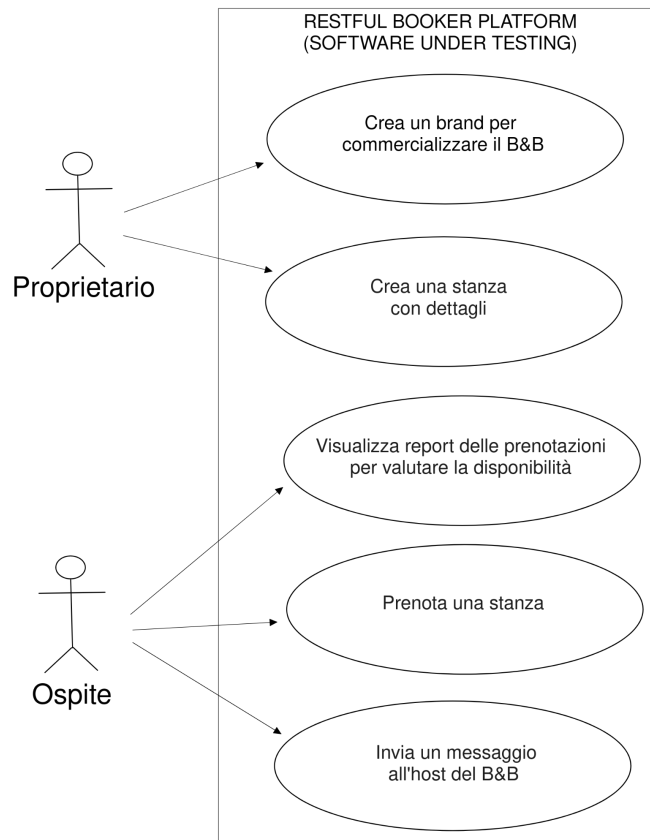
Giuliano Aiello
N97000439

Vincenzo Tramo
N97000433

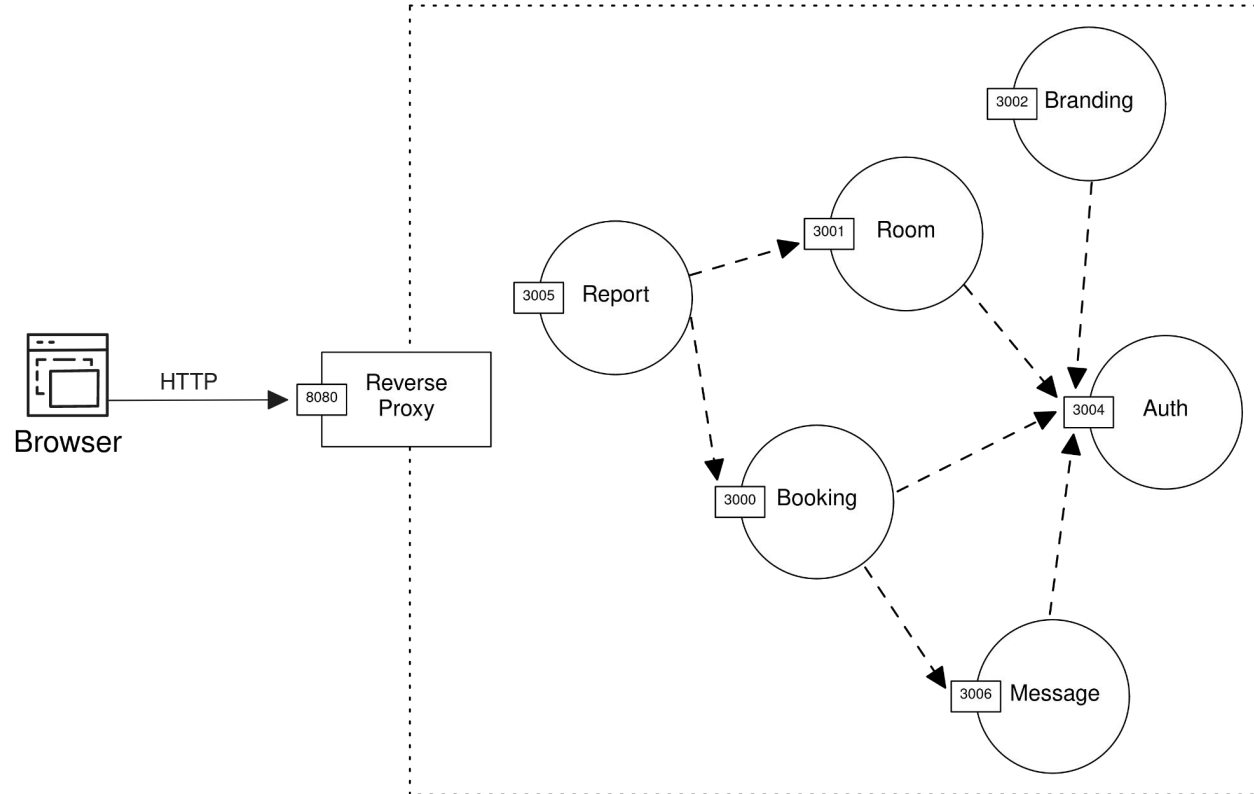
Gennaio, 2024
Università di Napoli, Federico II

Restful Booker Platform

- Sistema di prenotazione per Bed & Breakfast
- Progetto non realizzato da noi studenti
- Program Comprehension: comprendere il sistema



Restful Booker Platform

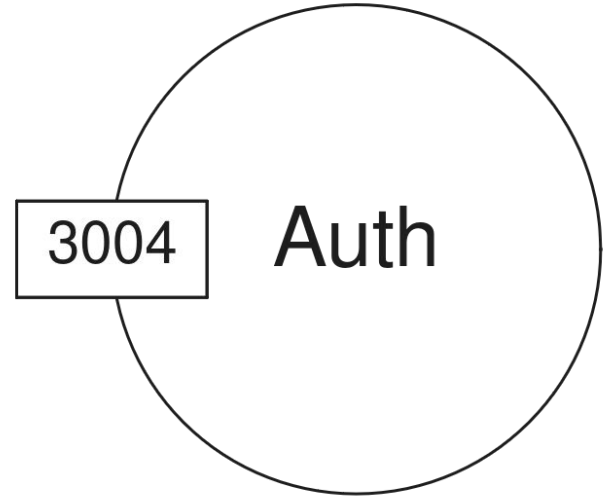


Restful Booker Platform

POST /auth/Login

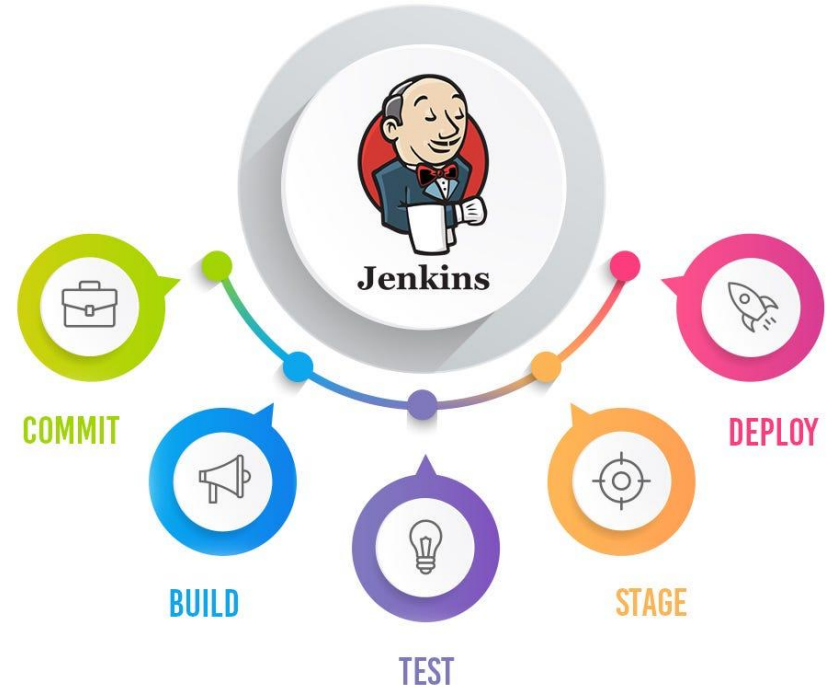
POST /auth/validate

POST /auth/Logout



Jenkins

- Automation Server open source
- Progettato per automatizzare attività legate al:
 - building
 - testing
 - delivering
 - deploying del software

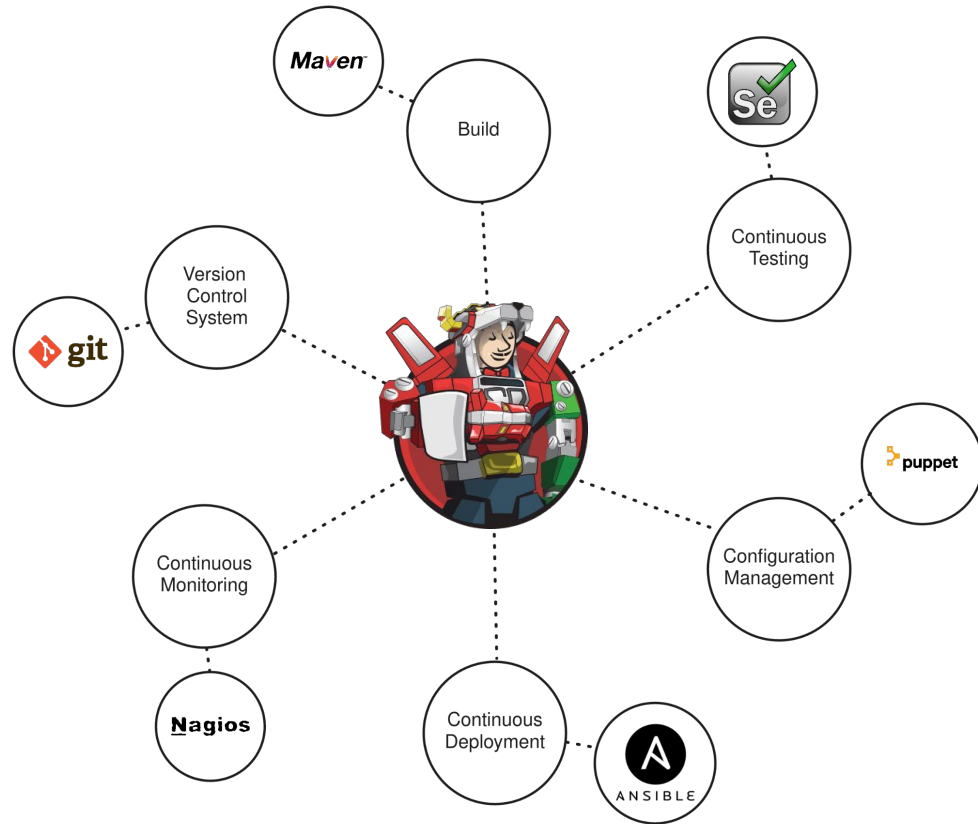


Jenkins - Pipeline as Code (PaC)

- Definizione pipeline scritta in un file chiamato *Jenkinsfile*
- Vantaggio principale: l'intera configurazione della pipeline “vive” insieme al codice sorgente
- *Declarative Pipeline Syntax* come Domain Specific Language (DSL) di facile lettura e scrittura

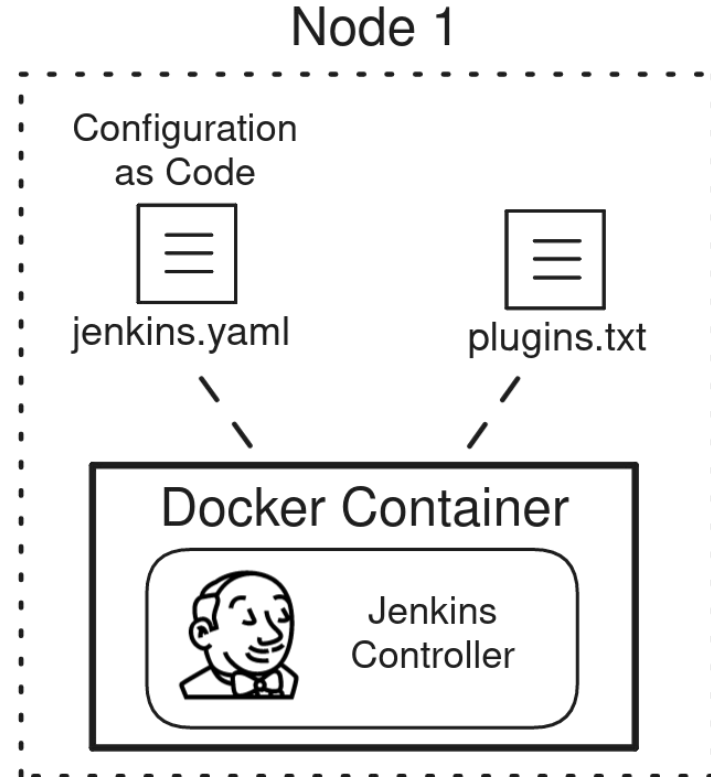
```
pipeline {  
    stage('Build') {  
        ...  
    }  
  
    stage('Test') {  
        ...  
    }  
  
    stage('Deploy') {  
        ...  
    }  
}
```

Jenkins - Plugins



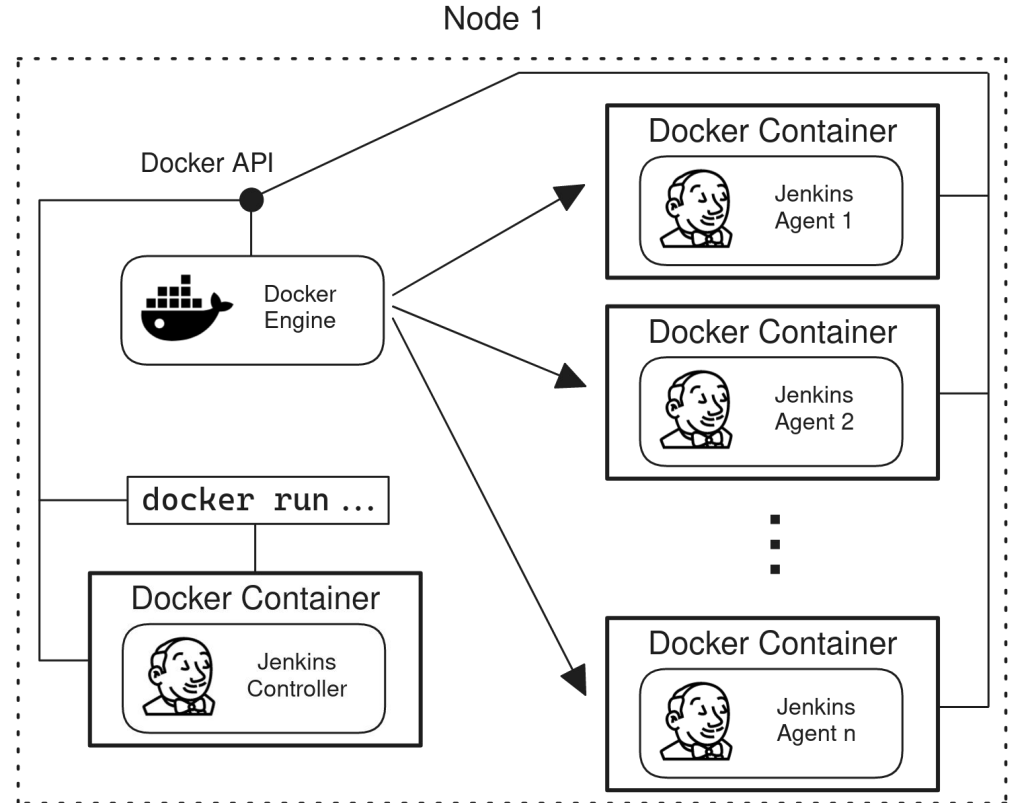
Jenkins & Docker

- Esecuzione di Jenkins in un Docker Container
- *Configuration as Code*
- Iniettare i plugins installati
- Replicabilità e Versioning

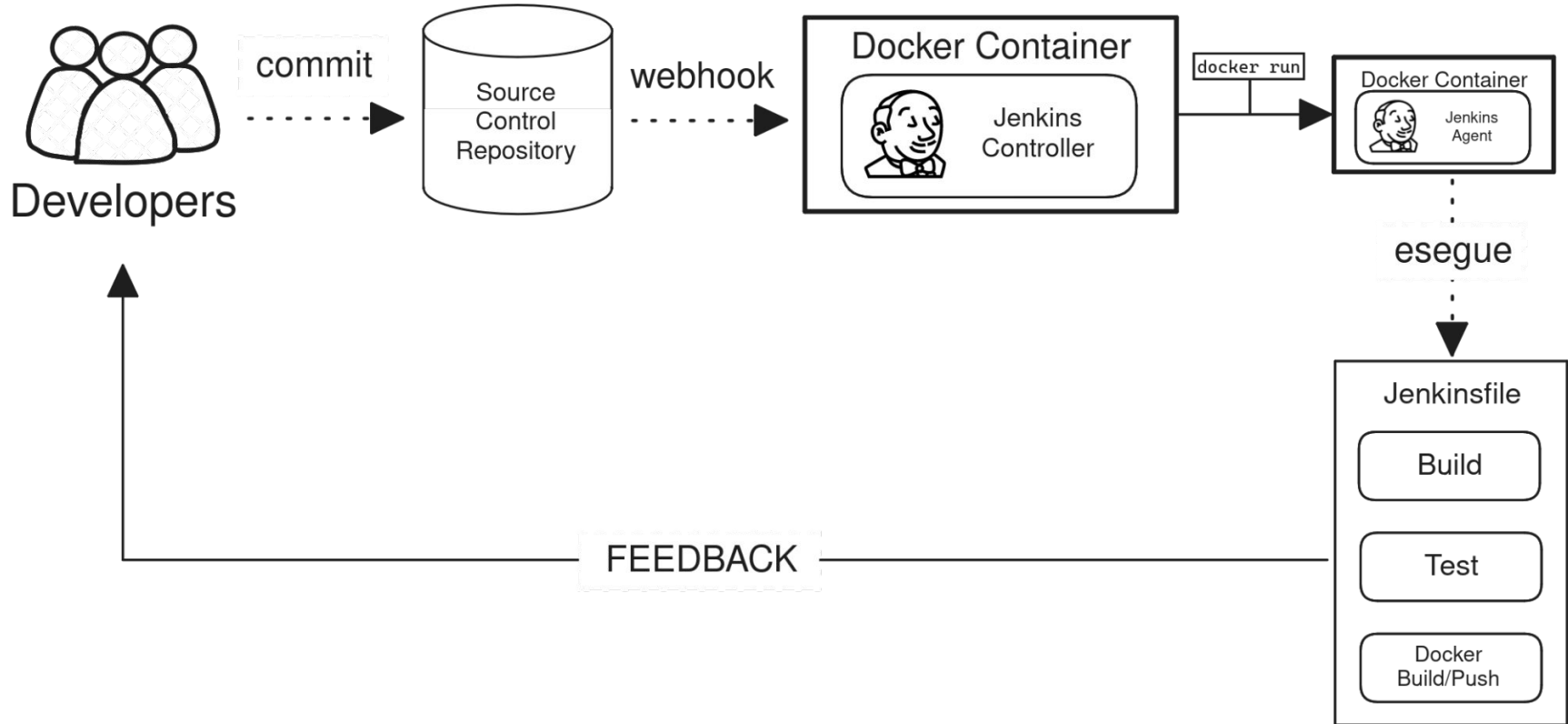


Jenkins & Docker

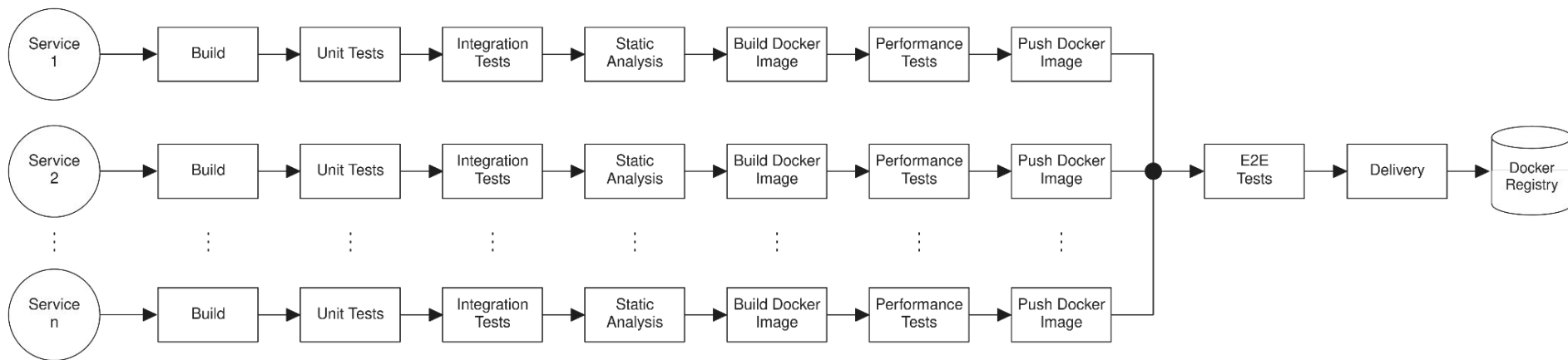
- Master-Slave Architecture
- Jenkins Controller
- Jenkins Agent
- Docker Engine condiviso
- Jenkins Cloud Agent
- Docker in Docker (DinD)



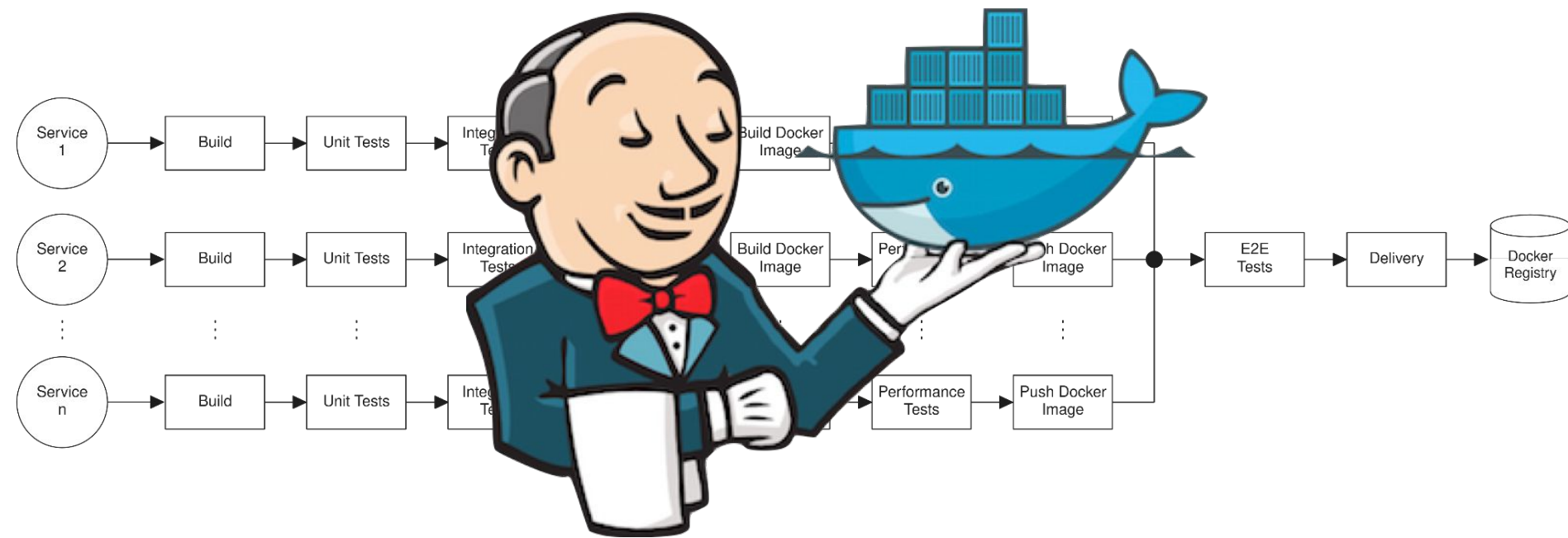
Jenkins & Source Control Server



Progettazione CI/CD

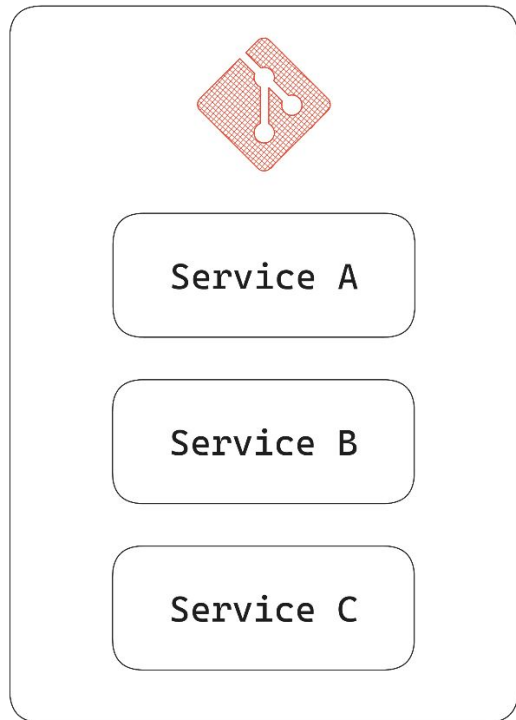


Progettazione CI/CD



Implementazione CI/CD - Monorepo vs Multirepo

Monorepo

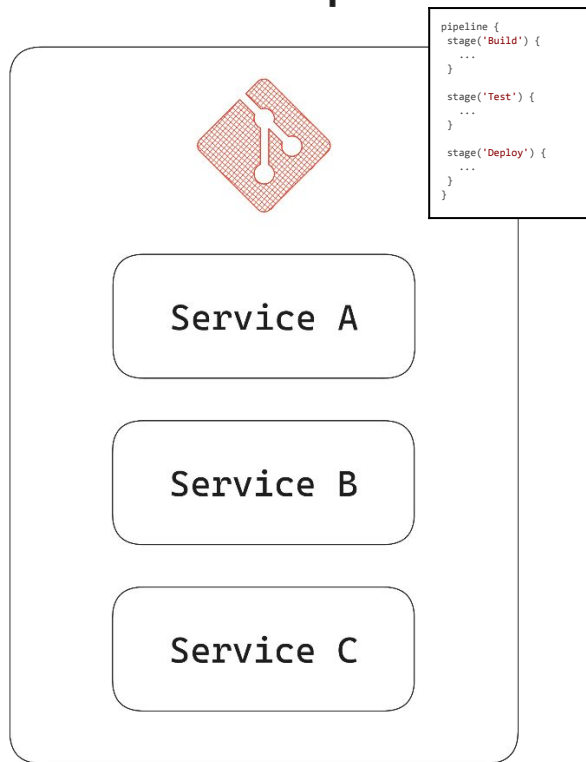


Multirepo

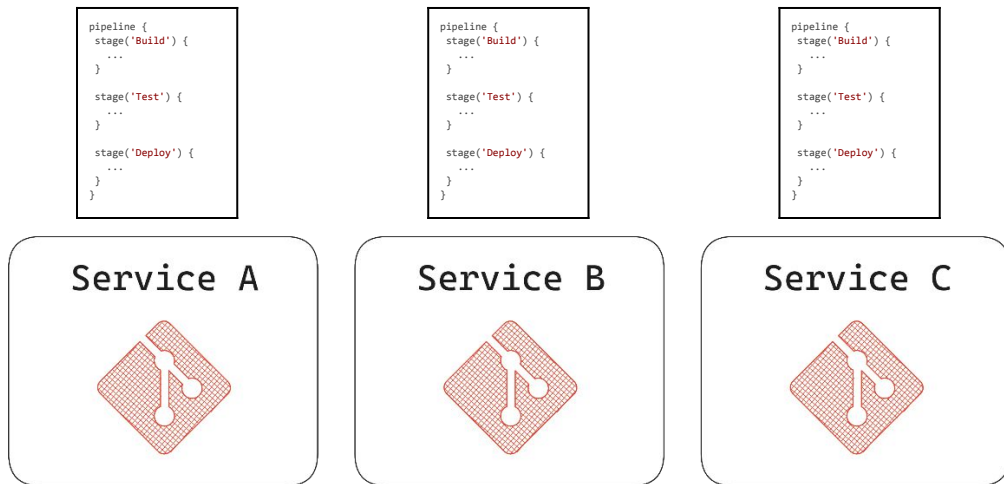


Implementazione CI/CD - Monorepo vs Multirepo

Monorepo



Multirepo



Implementazione CI/CD - Struttura del progetto

```
.  
|-- assets/  
|-- auth/  
|-- booking/  
|-- branding/  
|-- message/  
|-- proxy/  
|-- report/  
|-- room/  
|-- test-pilot/  
|-- ci/  
|-- Jenkinsfile  
|-- docs/  
|-- docker-compose.yml  
|-- pom.xml  
|-- build_locally.cmd  
|-- build_locally.sh  
|-- run_locally.cmd  
|-- run_locally.sh  
|-- README.md
```

- Multi-module Maven Project
- 9 moduli:
 - 6 microservizi
 - 1 dedicato alla front-end
 - 1 modulo funge da proxy
 - 1 modulo è dedicato al testing

Implementazione CI/CD - Struttura del progetto

```
.  
|-- assets/  
|-- auth/  
|-- booking/  
|-- branding/  
|-- message/  
|-- proxy/  
|-- report/  
|-- room/  
|-- test-pilot/  
|-- ci/  
|-- Jenkinsfile  
|-- docs/  
|-- docker-compose.yml  
|-- pom.xml  
|-- build_locally.cmd  
|-- build_locally.sh  
|-- run_locally.cmd  
|-- run_locally.sh  
|-- README.md
```

- 6 **microservizi**:
 - auth
 - booking
 - branding
 - message
 - report
 - room

Implementazione CI/CD - Struttura del progetto

```
.  
|-- assets/  
|-- auth/  
|-- booking/  
|-- branding/  
|-- message/  
|-- proxy/  
|-- report/  
|-- room/  
|-- test-pilot/  
|-- ci/  
|-- Jenkinsfile  
|-- docs/  
|-- docker-compose.yml  
|-- pom.xml  
|-- build_locally.cmd  
|-- build_locally.sh  
|-- run_locally.cmd  
|-- run_locally.sh  
|-- README.md
```

- Modulo **test-pilot** dedicato a test sofisticati:
 - test end-to-end
 - Selenium
 - eseguiti con Cucumber

Implementazione CI/CD - Struttura del progetto

```
.
|-- assets/
|-- auth/
|-- booking/
|-- branding/
|-- message/
|-- proxy/
|-- report/
|-- room/
|-- test-pilot/
|-- ci/
|-- Jenkinsfile
|-- docs/
|-- docker-compose.yml
|-- pom.xml
|-- build_locally.cmd
|-- build_locally.sh
|-- run_locally.cmd
|-- run_locally.sh
|-- README.md
```

```
pipeline {
    /* Esegui pipeline per ogni servizio
       coinvolto nella modifica */

    /* Alla fine, dopo che tutte le pipeline
       sono state eseguite
       con successo, esegui i test e2e */

    /* Consegna le nuove immagini */
}
```

Implementazione CI/CD - Struttura del progetto

```
.
|-- assets/
|-- auth/
|-- booking/
|-- branding/
|-- message/
|-- proxy/
|-- report/
|-- room/
|-- test-pilot/
|-- ci/
|-- Jenkinsfile
|-- docs/
|-- docker-compose.yml
|-- pom.xml
|-- build_locally.cmd
|-- build_locally.sh
|-- run_locally.cmd
|-- run_locally.sh
|-- README.md
```

- La cartella ci contiene scripts eseguiti dalla pipeline durante la sua esecuzione

Implementazione CI/CD - Direttiva agent

- La direttiva agent viene usata per specificare *dove* eseguire gli step o gli stage all'interno della pipeline
- Utilizziamo l'agent etichettato 'build-agent': è una immagine Docker contenente tutto il necessario per affrontare gli stage successivi

```
pipeline {  
    agent {  
        label 'build-agent'  
    }  
    ...  
}
```

Implementazione CI/CD - Direttiva options

```
pipeline {
```

```
...
```

```
options {  
  disableConcurrentBuilds()  
  buildDiscarder logRotator(artifactDaysToKeepStr: '',  
                             artifactNumToKeepStr: '100',  
                             daysToKeepStr: '',  
                             numToKeepStr: '100')  
  
  timestamps()  
  skipStagesAfterUnstable()  
  parallelsAlwaysFailFast()  
  timeout(time: 7, unit: 'MINUTES')  
}
```

```
...
```

```
}
```

Implementazione CI/CD - Direttiva options

```
pipeline {
```

```
...
```

```
options {  
  disableConcurrentBuilds()  
  buildDiscarder logRotator(artifactDaysToKeepStr: '',  
                             artifactNumToKeepStr: '100',  
                             daysToKeepStr: '',  
                             numToKeepStr: '100')  
  timestamps()  
  skipStagesAfterUnstable()  
  parallelsAlwaysFailFast()  
  timeout(time: 7, unit: 'MINUTES')  
}
```

```
...
```

```
}
```

Implementazione CI/CD - Direttiva options

```
pipeline {
```

```
...
```

```
options {  
  disableConcurrentBuilds()  
  buildDiscarder logRotator(artifactDaysToKeepStr: '',  
                             artifactNumToKeepStr: '100',  
                             daysToKeepStr: '',  
                             numToKeepStr: '100')  
  
  timestamps()  
  skipStagesAfterUnstable()  
  parallelsAlwaysFailFast()  
  timeout(time: 7, unit: 'MINUTES')  
}
```

```
...
```

```
}
```

Implementazione CI/CD - Direttiva options

```
pipeline {
```

```
...
```

```
options {  
    disableConcurrentBuilds()  
    buildDiscarder logRotator(artifactDaysToKeepStr: '',  
                               artifactNumToKeepStr: '100',  
                               daysToKeepStr: '',  
                               numToKeepStr: '100')  
  
    timestamps()  
    skipStagesAfterUnstable()  
    parallelsAlwaysFailFast()  
    timeout(time: 7, unit: 'MINUTES')  
}
```

```
...
```

```
}
```


Implementazione CI/CD - Direttiva options

```
pipeline {
```

```
...
```

```
options {  
    disableConcurrentBuilds()  
    buildDiscarder logRotator(artifactDaysToKeepStr: '',  
                               artifactNumToKeepStr: '100',  
                               daysToKeepStr: '',  
                               numToKeepStr: '100')  
  
    timestamps()  
    skipStagesAfterUnstable()  
    parallelsAlwaysFailFast()  
    timeout(time: 7, unit: 'MINUTES')  
}
```

```
...
```

```
}
```

Implementazione CI/CD - Direttiva options

```
pipeline {
```

```
...
```

```
options {  
    disableConcurrentBuilds()  
    buildDiscarder logRotator(artifactDaysToKeepStr: '',  
                               artifactNumToKeepStr: '100',  
                               daysToKeepStr: '',  
                               numToKeepStr: '100')  
  
    timestamps()  
    skipStagesAfterUnstable()  
    parallelsAlwaysFailFast()  
    timeout(time: 7, unit: 'MINUTES')  
}
```

```
...
```

```
}
```

Implementazione CI/CD - Blocco stages

- Il blocco stages esegue sequenzialmente (se non specificato esplicitamente una esecuzione parallela) gli stage in esso contenuto.

...

```
stages {  
    stage('Get git commit info') {  
        ...  
    }  
    ...  
}
```

...

Implementazione CI/CD - Lo stage Get git commit info

...

```
stage('Get git commit info') {  
    steps {  
        script {  
            env.GIT_COMMIT = sh (script: 'git rev-parse HEAD',  
                                returnStdout: true)  
            env.GIT_SHORT_COMMIT = "${env.GIT_COMMIT[0..7]}"  
            env.GIT_PREVIOUS_SUCCESSFUL_SHORT_COMMIT = env.GIT_PREVIOUS_SUCCESSFUL_COMMIT[0..7]  
            env.GIT_COMMITTER_NAME = sh (script: "git show -s --format='%an' ${env.GIT_COMMIT}",  
                                         returnStdout: true)  
            env.GIT_COMMITTER_EMAIL =  
                sh (script: "git show -s --format='%ae' ${env.GIT_COMMIT}",  
                   returnStdout: true)  
            env.GIT_COMMIT_MSG =  
                sh (script: "git log --format=%B -n 1 ${env.GIT_COMMIT}",  
                   returnStdout: true)  
        }  
    }  
}
```

...

Implementazione CI/CD - Lo stage Get git commit info

...

```
stage('Get git commit info') {
    steps {
        script {
            env.GIT_COMMIT = sh (script: 'git rev-parse HEAD',
                                returnStdout: true)
            env.GIT_SHORT_COMMIT = "${env.GIT_COMMIT[0..7]}"
            env.GIT_PREVIOUS_SUCCESSFUL_SHORT_COMMIT = env.GIT_PREVIOUS_SUCCESSFUL_COMMIT[0..7]
            env.GIT_COMMITTER_NAME = sh (script: "git show -s --format='%an' ${env.GIT_COMMIT}",
                                         returnStdout: true)
            env.GIT_COMMITTER_EMAIL =
                sh (script: "git show -s --format='%ae' ${env.GIT_COMMIT}",
                  returnStdout: true)
            env.GIT_COMMIT_MSG =
                sh (script: "git log --format=%B -n 1 ${env.GIT_COMMIT}",
                  returnStdout: true)
        }
    }
}
```

...

Implementazione CI/CD - Lo stage Get git commit info

...

```
stage('Get git commit info') {
    steps {
        script {
            env.GIT_COMMIT = sh (script: 'git rev-parse HEAD',
                                returnStdout: true)
            env.GIT_SHORT_COMMIT = "${env.GIT_COMMIT[0..7]}"
            env.GIT_PREVIOUS_SUCCESSFUL_SHORT_COMMIT = env.GIT_PREVIOUS_SUCCESSFUL_COMMIT[0..7]
            env.GIT_COMMITTER_NAME = sh (script: "git show -s --format='%an' ${env.GIT_COMMIT}",
                                         returnStdout: true)
            env.GIT_COMMITTER_EMAIL =
                sh (script: "git show -s --format='%ae' ${env.GIT_COMMIT}",
                  returnStdout: true)
            env.GIT_COMMIT_MSG =
                sh (script: "git log --format=%B -n 1 ${env.GIT_COMMIT}",
                  returnStdout: true)
        }
    }
}
```

...

Implementazione CI/CD - Lo stage Get git commit info

...

```
stage('Get git commit info') {  
    steps {  
        script {  
            env.GIT_COMMIT = sh (script: 'git rev-parse HEAD',  
                                returnStdout: true)  
            env.GIT_SHORT_COMMIT = "${env.GIT_COMMIT[0..7]}"  
            env.GIT_PREVIOUS_SUCCESSFUL_SHORT_COMMIT = env.GIT_PREVIOUS_SUCCESSFUL_COMMIT[0..7]  
            env.GIT_COMMITTER_NAME = sh (script: "git show -s --format='%an' ${env.GIT_COMMIT}",  
                                         returnStdout: true)  
            env.GIT_COMMITTER_EMAIL =  
                sh (script: "git show -s --format='%ae' ${env.GIT_COMMIT}",  
                   returnStdout: true)  
            env.GIT_COMMIT_MSG =  
                sh (script: "git log --format=%B -n 1 ${env.GIT_COMMIT}",  
                   returnStdout: true)  
        }  
    }  
}
```

...

Implementazione CI/CD - Lo stage Get git commit info

...

```
stage('Get git commit info') {  
    steps {  
        script {  
            env.GIT_COMMIT = sh (script: 'git rev-parse HEAD',  
                                returnStdout: true)  
            env.GIT_SHORT_COMMIT = "${env.GIT_COMMIT[0..7]}"  
            env.GIT_PREVIOUS_SUCCESSFUL_SHORT_COMMIT = env.GIT_PREVIOUS_SUCCESSFUL_COMMIT[0..7]  
            env.GIT_COMMITTER_NAME = sh (script: "git show -s --format='%an' ${env.GIT_COMMIT}",  
                                         returnStdout: true)  
            env.GIT_COMMITTER_EMAIL =  
                sh (script: "git show -s --format='%ae' ${env.GIT_COMMIT}",  
                    returnStdout: true)  
            env.GIT_COMMIT_MSG =  
                sh (script: "git log --format=%B -n 1 ${env.GIT_COMMIT}",  
                    returnStdout: true)  
        }  
    }  
}
```

...

Implementazione CI/CD - Lo stage Get git commit info

...

```
stage('Get git commit info') {  
    steps {  
        script {  
            env.GIT_COMMIT = sh (script: 'git rev-parse HEAD',  
                                returnStdout: true)  
            env.GIT_SHORT_COMMIT = "${env.GIT_COMMIT[0..7]}"  
            env.GIT_PREVIOUS_SUCCESSFUL_SHORT_COMMIT = env.GIT_PREVIOUS_SUCCESSFUL_COMMIT[0..7]  
            env.GIT_COMMITTER_NAME = sh (script: "git show -s --format='%an' ${env.GIT_COMMIT}",  
                                         returnStdout: true)  
            env.GIT_COMMITTER_EMAIL =  
                sh (script: "git show -s --format='%ae' ${env.GIT_COMMIT}",  
                   returnStdout: true)  
            env.GIT_COMMIT_MSG =  
                sh (script: "git log --format=%B -n 1 ${env.GIT_COMMIT}",  
                   returnStdout: true)  
        }  
    }  
}
```

...

Implementazione CI/CD - Lo stage Mine Repository

- Plugin *Git Forensics*:

- Git Blame
- File Statistics
- Commit tracking
- Commit statistics

```
...  
  
stage('Mine repository') {  
    steps {  
        mineRepository()  
    }  
}  
  
...
```

Implementazione CI/CD - Lo stage Mine Repository

- Plugin *Git Forensics*:

...

```
stage('Mine repository') {
```

Details							
Show 10 entries		Search: <input type="text"/>					
File	#Authors	#Commits	Last Commit	Added	#LoC	Code Churn	
AuthRequests.java	1	2	1 month ago	2 months ago	46	52	
mvnw.cmd	1	1	2 months ago	2 months ago	205	205	
application-dev.properties	1	3	5 years ago	5 years ago	1	7	
mvnw	1	1	2 months ago	2 months ago	308	308	
nav-test.js.snap	1	6	1 year ago	4 years ago	112	344	
README.md	1	2	2 months ago	2 months ago	9	9	
application-dev.properties	1	2	5 years ago	5 years ago	0	2	
bookings.feature	1	1	2 months ago	2 months ago	8	8	
loading.gif	1	1	1 year ago	1 year ago	0	0	
report-test.js	1	14	11 months ago	5 years ago	41	667	
Showing 131 to 140 of 347 entries							1 ... 13 14 15 ... 35

Implementazione CI/CD - Lo stage Stash Git Repo

...

```
stage('Stash git repository') {  
    steps {  
        stash includes: '**', name: 'rbp', useDefaultExcludes: false  
    }  
}
```

...

- I Jenkins agent (in questo progetto) sono Docker container che non condividono il workspace
- Lo step stash evita di clonare di nuovo la repository
- Usando unstash 'rbp' velocizziamo particolarmente la pipeline quando nuovi Jenkins agent vengono avviati
- I file con stash vengono archiviati in un file TAR compresso

Implementazione CI/CD - Esecuzione parallela

- Lo stage '**Services Pipeline**' esegue le pipeline singole di ogni servizio
- Le pipeline individuali vengono eseguite in parallelo usando il blocco `parallel`
- Ogni stage può essere eseguito in parallelo perché indipendenti tra di loro
- Ogni stage esegue in un nuovo Jenkins agent (un nuovo Docker container)
 - questa scelta non è una conseguenza automatica dell'uso del blocco `parallel`, ma piuttosto un dettaglio implementativo che forziamo noi esplicitamente

...

```
stage('Services Pipeline') {
```

```
  parallel {
```

```
    // esegui stage 'Auth Service'
```

```
    // esegui stage 'Room Service'
```

```
    // esegui stage 'Booking Service'
```

```
    // esegui stage 'Branding Service'
```

```
    // esegui stage 'Message Service'
```

```
    // esegui stage 'Report Service'
```

```
  }
```

```
}
```

...

Implementazione CI/CD - Evitare pipeline inutili

- **Obiettivo:** quando una modifica ha influenzato soltanto un servizio dovrebbe essere eseguita *solo e soltanto* la pipeline di quel servizio
- **Soluzione:** uso del blocco when
 - la pipeline viene avviata solo quando la modifica influenza:
 - Dockerfile
 - codice sorgente
 - codice di test

...

```
stage('Auth Service') {
```

```
  when {  
    anyOf {  
      changeset "auth/Dockerfile"  
      changeset "auth/src/main/java/**/*.java"  
      changeset "auth/src/test/java/**/*.java"  
    }  
  }
```

```
  // Esegui pipeline per il servizio auth
```

```
}
```

...

Implementazione CI/CD - Shared Library

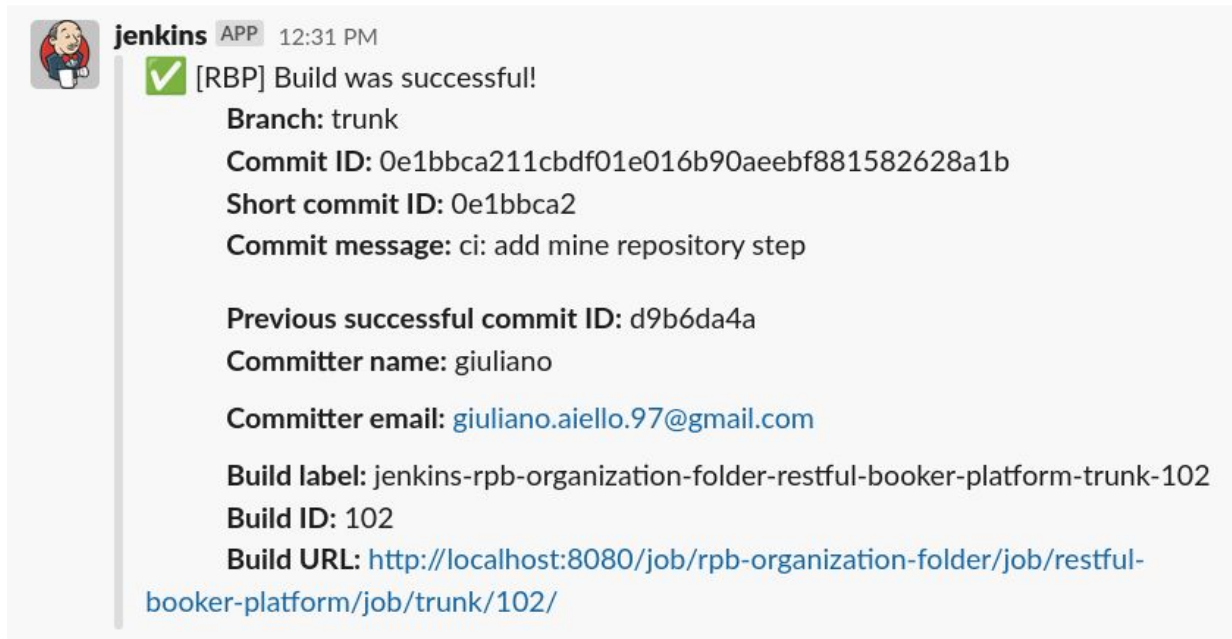
- **Problema:** ogni pipeline individuale presenta la stessa identica struttura
- **Obiettivo:** diminuire la duplicazione di codice (DRY - Don't Repeat Yourself)
- **Soluzione:** definire una libreria (Shared Library) che definisce una pipeline comune per tutti i servizi opportunamente parametrizzata

```
...  
stage('Room Service') {  
    ...  
    steps {  
        rbpServicePipeline(  
            serviceName: 'room',  
            nodeLabel: 'build-agent',  
            rbpServiceHostname: 'rbp-room',  
            rbpServicePort: '3001',  
            skipPerformanceTests: true  
        )  
    }  
}  
...
```

- Creazione di step personalizzati
- Lo step custom `rbpServicePipeline(...)` definisce un “pipeline template”
- La libreria ha una determinata struttura ed è ospitata in una repository GitHub
- Possiamo importare la libreria e usarla nel Jenkinsfile usando l'annotazione `@Library('rbp-shared-library')`

Implementazione CI/CD - Shared Library

- Il custom step `rbpSendSlackNotification()` invia una notifica custom al team usando Slack
- La notifica viene sempre inviata alla fine della build indipendentemente dallo stato di terminazione:
 - SUCCESS
 - FAILED
 - ABORTED
 - UNSTABLE



Implementazione CI/CD - Shared Library



<https://github.com/vtramo/rbp-jenkins-shared-library>

```
.
|-- src/
|   |--BuildStatus.groovy
|   |--BuildUtilities.groovy
|   |--Service.groovy
|   |--SlackUtilities.groovy
|--vars/
|   |--rbpSendNotification.groovy
|   |--rbpServicePipeline.groovy
|-- README.md
```

Qui vengono definiti gli step custom che possono essere utilizzati nel Jenkinsfile dopo aver importato la libreria con `@Library('rbp-shared-library')`

Implementazione CI/CD - Lo stage Build

- *Maven* come Software Project Management and Comprehension Tool
- Lo stage di 'Build' come primo stage della pipeline individuale
- Compila e genera un artefatto .jar senza eseguire nessun test
 - -DskipTests

```
...  
stage("[${serviceName}] Build")  
    timeout(time: 2, unit: 'MINUTES') {  
        dir("${rbpServiceMainDir}") {  
            sh 'mvn clean package -DskipTests'  
        }  
    }  
}  
...
```

Implementazione CI/CD - Gli stage di test

...

```
stage("[${serviceName}] Unit Tests") {  
  timeout(time: 20, unit: 'SECONDS') {  
    dir("${rbpServiceMainDir}") {  
      sh 'mvn test'  
    }  
  }  
}
```

Esegue i test di unità utilizzando il plugin surefire. Il plugin è stato configurato per eseguire tutte le test suite con il suffisso UT (Unit Test).

```
stage("[${serviceName}] Integration Tests") {  
  timeout(time: 40, unit: 'SECONDS') {  
    dir("${rbpServiceMainDir}") {  
      sh 'mvn verify -Dskip.surefire.tests=true'  
    }  
  }  
}
```

Esegue i test di integrazione utilizzando il plugin failsafe. Il plugin è stato configurato per eseguire tutte le test suite con il suffisso CT (Component Test) e IT (Integration Test). Non esegue i test di unità di nuovo grazie a una opzione custom -Dskip.surefire.tests=true.

...

Implementazione CI/CD - Gli stage di analisi statica

...

```
stage("[${serviceName}] SonarQube Scan") {  
  timeout(time: 1, unit: 'MINUTES') {  
    dir("${rbpServiceMainDir}") {  
      withSonarQubeEnv(installationName: 'sonarqube') {  
        sh """  
          mvn sonar:sonar \  
            -Dsonar.projectKey=restful-booker-platform-${serviceName} \  
            -Dsonar.projectName=restful-booker-platform-${serviceName} \  
          """  
        }  
      }  
    }  
  }  
}  
  
stage("[${serviceName}] Quality Gates") {  
  timeout(time: 1, unit: 'MINUTES') {  
    waitForQualityGate abortPipeline: true  
  }  
}
```



Implementazione CI/CD - Gli stage di analisi statica

...

```
stage("[${serviceName}] SonarQube Scan") {  
  timeout(time: 1, unit: 'MINUTES') {  
    dir("${rbpServiceMainDir}") {  
      withSonarQubeEnv(installationName: 'sonarqube') {  
        sh """  
          mvn sonar:sonar \  
            -Dsonar.projectKey=restful-booker-platform-${serviceName} \  
            -Dsonar.projectName=restful-booker-platform-${serviceName} \  
          """  
        }  
      }  
    }  
  }  
}
```

```
stage("[${serviceName}] Quality Gates") {  
  timeout(time: 1, unit: 'MINUTES') {  
    waitForQualityGate abortPipeline: true  
  }  
}
```

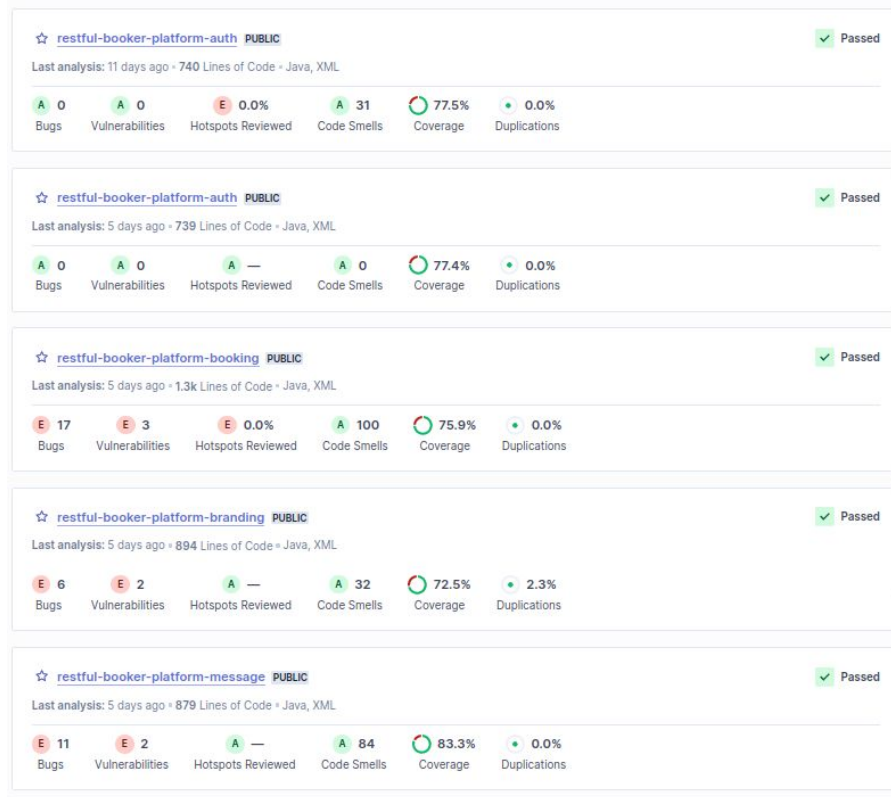
Questo stage attende il risultato dei Quality Gates di SonarQube



Implementazione CI/CD - Gli stage di analisi statica

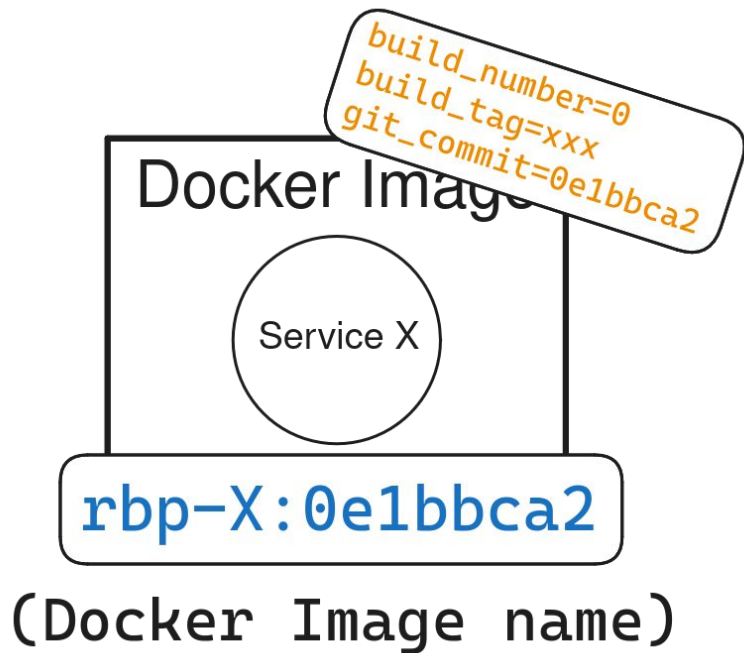
Quality Gate *Clean as You Code*:

- 0 issues (bug, code smell, ...)
- almeno 80% di code coverage
- al più 3% di duplicazione codice



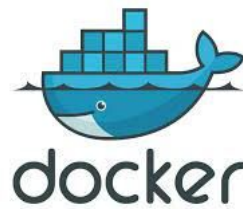
Implementazione CI/CD - Lo stage Build Image

```
stage("[${serviceName}] Build Image") {  
  ...  
  
  steps {  
    sh """"  
      docker build  
        --build-arg BUILD_NUMBER=${BUILD_NUMBER} \  
        --build-arg BUILD_TAG=${BUILD_TAG} \  
        --build-arg GIT_COMMIT=${GIT_COMMIT} \  
        -t ${DOCKER_REGISTRY_URL} \  
        rbp-${serviceName}:${GIT_SHORT_COMMIT} .  
    """"  
  
    ...  
  }  
}
```



Implementazione CI/CD - Lo stage Performance Tests

- Ogni servizio deve avere uno script *JMeter* (estensione `.jmx`) per eseguire dei test di carico
- **Obiettivo:** valutare le performance del servizio simulando il quanto più possibile il comportamento di più utenti concorrenti
- I test delle performance vengono avviati usando un tool chiamato *Taurus*
- Ogni servizio ha una cartella ci che contiene:
 - un file `performance-test.yaml`
 - uno script `performance-test-jmeter.jmx`
 - un `docker-compose-test.yaml` file per eseguire il servizio



Implementazione CI/CD - Lo stage Performance Tests

- Tramite Taurus possiamo definire degli obiettivi (KPI) da raggiungere usando il modulo **passfail**
- Per il servizio auth, abbiamo fissato i seguenti obiettivi:
 - l'endpoint **POST** /auth/login deve avere un tempo di risposta medio minore di 100ms
 - l'endpoint **POST** /auth/validate deve avere un tempo di risposta medio minore di 80ms
 - l'endpoint **POST** /auth/logout deve avere un tempo di risposta medio minore di 100ms
 - tutti gli endpoint di auth devono avere un tempo di risposta medio minore di 100ms
 - nessuna richiesta HTTP durante l'esecuzione del test deve ritornare una HTTP Status Code negativo

- module: **passfail**

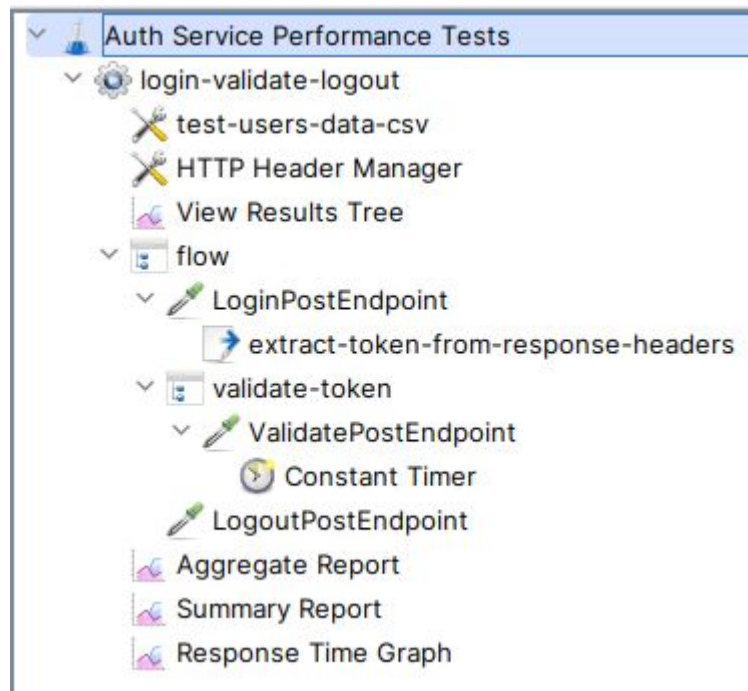
criteria:

- avg-rt of LoginPostEndpoint>=100ms for 5s, stop as failed
- avg-rt of ValidatePostEndpoint>=80ms for 5s, stop as failed
- avg-rt of LogoutPostEndpoint>=100ms for 5s, stop as failed
- avg-rt >=100ms for 5s, stop as failed
- fail >0 for 1s, stop as failed

...

Implementazione CI/CD - Lo stage Performance Tests

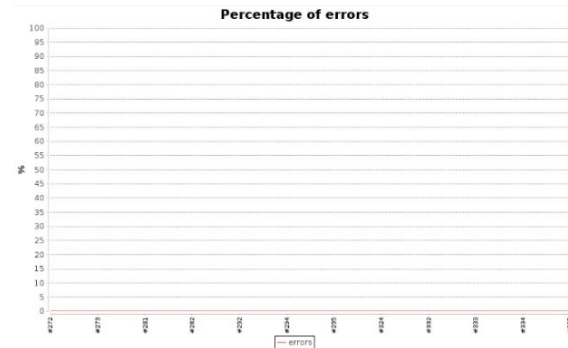
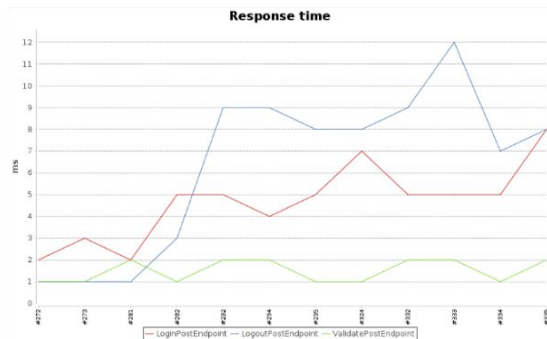
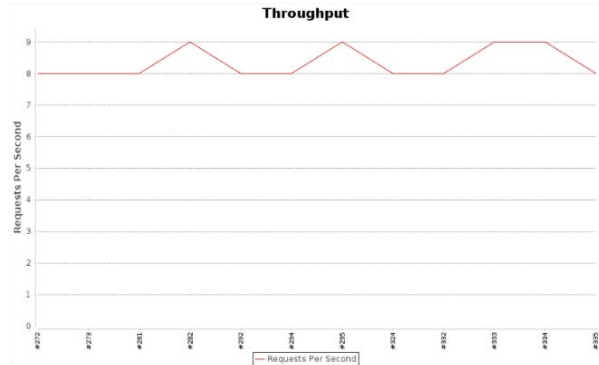
- Lo script JMeter simula un utente in questo modo:
 - l'utente esegue il login una volta;
 - l'utente valida il token 3 volte con una pausa di 2-4 secondi tra una chiamata e l'altra;
 - l'utente esegue il logout.
- Vengono eseguiti 20 utenti concorrenti con un ramp-up di 5 secondi
- Lo script JMeter ripete la stessa procedura per 3 volte prima di terminare



Implementazione CI/CD - Lo stage Performance Tests

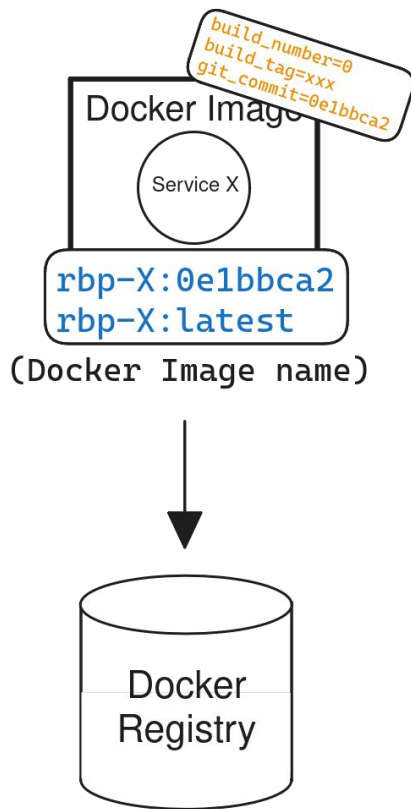
Comparison with previous build

URI	Samples	Average (ms)	Min(ms)	Median(ms)	Line 90.0(ms)	Line 95.0(ms)	Max(ms)	Http Code	Errors (%)	Average (KB)	Total (KB)
LoginPostEndpoint	60 ⁺⁰	8 ⁺³	1 ⁰	2 ⁰	28 ⁺¹³	33 ⁺¹⁷	56 ⁺¹¹		0.0 % 0.0 %	0.0 0.0	0.0 0.0
LogoutPostEndpoint	60 ⁺⁰	8 ⁺¹	1 ⁰	7 ⁺⁵	15 ⁻⁵	20 ⁻¹³	62 ⁺⁹		0.0 % 0.0 %	0.0 0.0	0.0 0.0
ValidatePostEndpoint	180 ⁺⁰	2 ⁺¹	1 ⁰	2 ⁰	3 ⁺¹	4 ⁺¹	23 ⁻¹³		0.0 % 0.0 %	0.0 0.0	0.0 0.0
All URIs	300 ⁺⁰	4 ⁺¹	1 ⁰	2 ⁰	12 ⁺²	19 ⁺⁴	62 ⁺⁹		0.0 % 0.0 %	131.8	39540.0



Implementazione CI/CD - Lo stage Push Image

```
stage("[${serviceName}] Push Image") {  
  if (currentBuild.result != null && currentBuild.result != 'SUCCESS') {  
    echo "[${serviceName}] Pipeline not in a  
      successful state, skipping Push Image stage..."  
  } else {  
    def dockerImageBaseTag =  
      "${DOCKER_REGISTRY_URL}/rbp-${serviceName}"  
    def commitDockerImage =  
      dockerImageBaseTag + ":${GIT_SHORT_COMMIT}"  
    def latestDockerImage =  
      dockerImageBaseTag + ":latest"  
  
    timeout(time: 1, unit: 'MINUTES') {  
      sh """  
        docker tag ${commitDockerImage} ${latestDockerImage} && \  
        docker push ${commitDockerImage} && \  
        docker push ${latestDockerImage}  
      """  
    }  
  }  
}
```



Implementazione CI/CD - Raccolta dei report

```
dir("${rbpServiceMainDir}") {  
    ...  
  
    junit(  
        testResults: 'target/surefire-reports/**/*.xml,target/failsafe-reports/**/*.xml',  
        allowEmptyResults: true  
    )  
  
    jacoco(  
        execPattern: 'target/**/*.exec',  
        classPattern: 'target/classes/com/rbp',  
        sourcePattern: 'src/main/java/com/rbp'  
    )  
  
    ...  
}
```

Implementazione CI/CD - Raccolta dei report

Test Result

0 Failures (±0)

237 tests (+73)

Took 1 min 35 sec.

[Add description](#)

name	instruction	branch	complexity	line	method	class
com.rbp.model.booking	M: 41 C: 127 76% 	M: 0 C: 0 100%	M: 5 C: 28 85% 	M: 8 C: 59 88% 	M: 5 C: 28 85% 	M: 0 C: 5 100% 

Performance Breakdown by URI: aggregate-results.xml


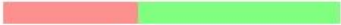
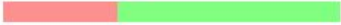



Response time trends for build: "test-pipeline #335"

Comparison with previous build

URI	Samples	Average (ms)	Min(ms)	Median(ms)	Line 90.0(ms)	Line 95.0(ms)	Max(ms)	Http Code	Errors (%)	Average (KB)	Total (KB)
LoginPostEndpoint	60 ⁺⁰	8 ⁺³	1 ⁰	2 ⁰	28 ⁺¹³	33 ⁺¹⁷	56 ⁺¹¹		0.0 % 0.0 %	0.0 ^{0.0}	0.0 ^{0.0}
LogoutPostEndpoint	60 ⁺⁰	8 ⁺¹	1 ⁰	7 ⁺⁵	15 ⁻⁵	20 ⁻¹³	62 ⁺⁹		0.0 % 0.0 %	0.0 ^{0.0}	0.0 ^{0.0}
ValidatePostEndpoint	180 ⁺⁰	2 ⁺¹	1 ⁰	2 ⁰	3 ⁺¹	4 ⁺¹	23 ⁻¹³		0.0 % 0.0 %	0.0 ^{0.0}	0.0 ^{0.0}
All URIs	300 ⁺⁰	4 ⁺¹	1 ⁰	2 ⁰	12 ⁺⁵	19 ⁺⁴	62 ⁺⁹		0.0 % 0.0 %	131.8	39540.0



Jacoco - Overall Coverage Summary

INSTRUCTION	70%	
BRANCH	60%	
COMPLEXITY	66%	
LINE	70%	
METHOD	70%	
CLASS	86%	

Implementazione CI/CD - Memorizzazione artefatti

```
dir("${rbpServiceMainDir}") {  
    . . .  
  
    archiveArtifacts artifacts: 'target/*.jar', fingerprint: true  
  
    . . .  
}
```

Recorded Fingerprints

File	Original owner	Age
target/restful-booker-platform-auth-1.6-SNAPSHOT-exec.jar	this build	5 days 23 hr old
target/restful-booker-platform-auth-1.6-SNAPSHOT.jar	this build	5 days 23 hr old
target/restful-booker-platform-booking-1.6-SNAPSHOT-exec.jar	this build	5 days 23 hr old
target/restful-booker-platform-booking-1.6-SNAPSHOT.jar	this build	5 days 23 hr old
target/restful-booker-platform-branding-1.6-SNAPSHOT-exec.jar	this build	5 days 23 hr old
target/restful-booker-platform-branding-1.6-SNAPSHOT.jar	this build	5 days 23 hr old
target/restful-booker-platform-message-1.6-SNAPSHOT-exec.jar	this build	5 days 23 hr old
target/restful-booker-platform-message-1.6-SNAPSHOT.jar	this build	5 days 23 hr old
target/restful-booker-platform-report-1.6-SNAPSHOT-exec.jar	this build	5 days 23 hr old
target/restful-booker-platform-report-1.6-SNAPSHOT.jar	this build	5 days 23 hr old
target/restful-booker-platform-room-1.6-SNAPSHOT-exec.jar	this build	5 days 23 hr old
target/restful-booker-platform-room-1.6-SNAPSHOT.jar	this build	5 days 23 hr old

Implementazione CI/CD - Lo stage E2E Tests

- I test E2E vengono eseguiti dal modulo test-pilot
- Esecuzione parallela dei test E2E su differenti browser:
 - Chrome
 - Firefox
 - Edge
- Ogni stage esegue in un nuovo Jenkins agent (Docker container)
 - sono necessarie delle configurazioni extra perché *tutti puntano allo stesso Docker engine*

...

```
stage('E2E Tests') {  
    environment {  
        RBP_TEST_PILOT_MAIN_DIR = 'test-pilot'  
        RBP_TEST_PILOT_DOCKER_DIR =  
            "${RBP_TEST_PILOT_MAIN_DIR}/src/test/resources/docker"  
        DISABLE_TESTCONTAINERS = 'true'  
        RBP_PROXY_URL = 'http://rbp-proxy:8080'  
    }  
  
    options {  
        timeout(time: 3, unit: 'MINUTES')  
    }  
  
    parallel {  
        // Stage 'Chrome': esegui test E2E su Chrome  
        // Stage 'Firefox': esegui test E2E su Firefox  
        // Stage 'Edge': esegui test E2E su Edge  
    }  
}
```

...

Implementazione CI/CD - Lo stage E2E Tests



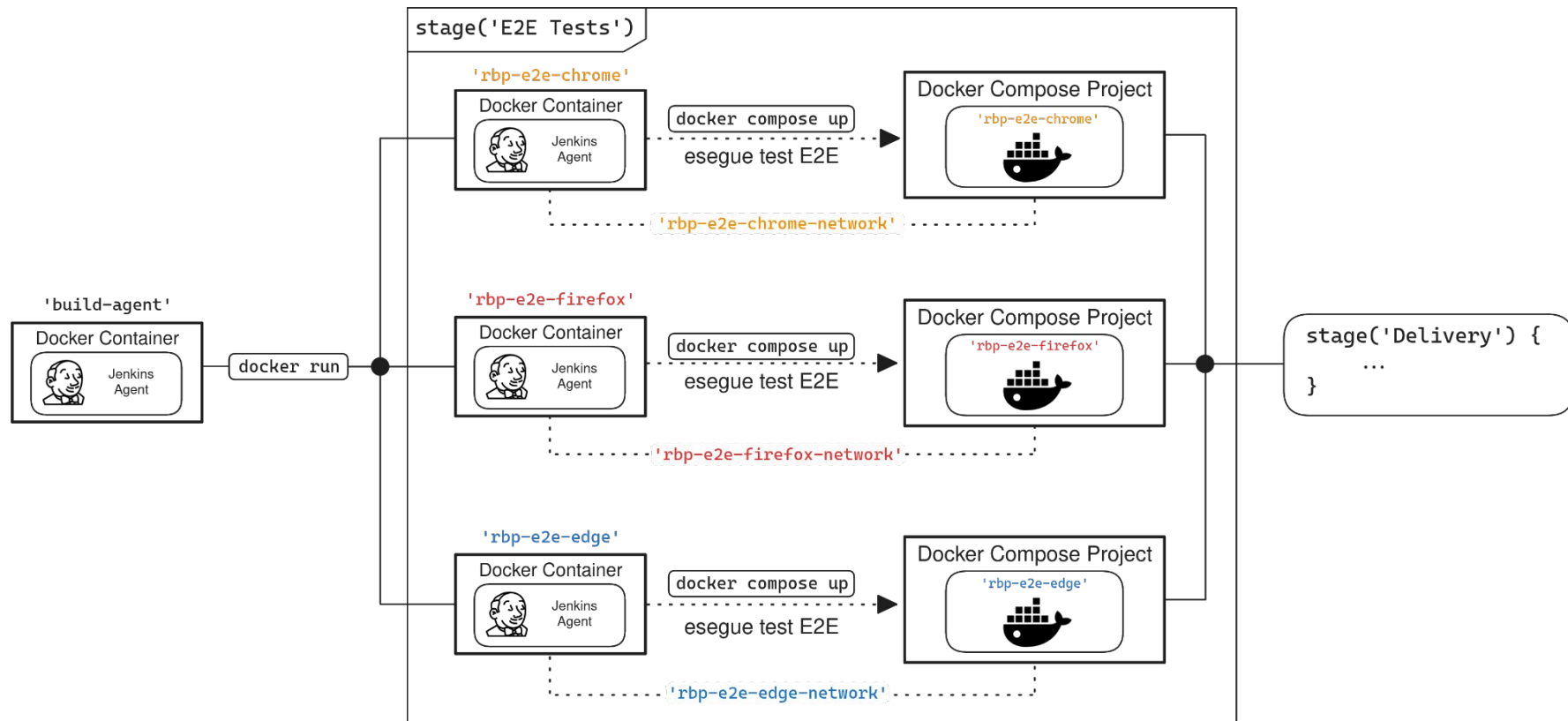
```
stage('Chrome') {  
  environment {  
    WEB_DRIVER = 'chrome'  
    RBP_E2E_DOCKER_NETWORK = 'rbp-e2e-chrome'  
    RBP_E2E_DOCKER_PROJECT_NAME = 'rbp-e2e-chrome'  
  }  
}
```

```
agent {  
  label 'rbp-e2e-chrome'  
}
```

*

```
*steps {  
  unstash 'rbp'  
  
  dir("${RBP_TEST_PILOT_DOCKER_DIR}") {  
    sh """  
        docker compose -f docker-compose-test.yaml \  
        -p ${RBP_E2E_DOCKER_PROJECT_NAME} \  
        up -d  
      """  
  }  
  
  dir("${RBP_TEST_PILOT_MAIN_DIR}") {  
    sh 'mvn clean test -Dcucumber.features=src/test/resources'  
  }  
}
```

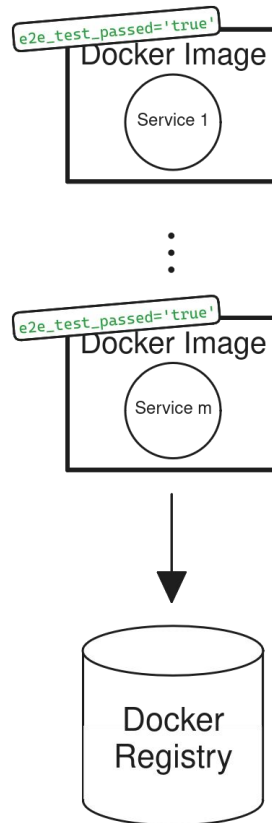
Implementazione CI/CD - Lo stage E2E Tests



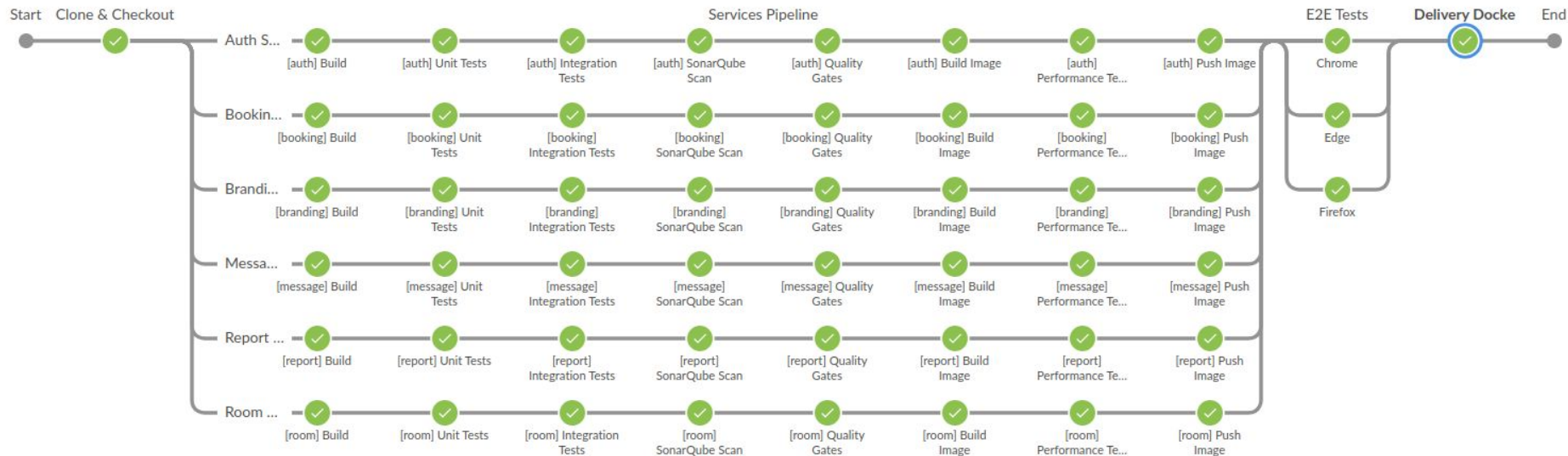
Implementazione CI/CD - Lo stage Delivery

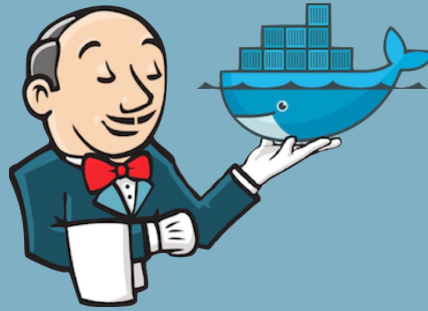
- Push delle immagini dei servizi coinvolti dalla modifica in un registro privato Docker
- Prima di pushare, viene eseguita una “pratica di certificazione”:
 - viene assegnata una label a ogni immagine docker:
 - `e2e_test_passed="true"`
 - usando lo script `ci/image_delivery.sh`

```
stage('Delivery') {  
  environment {  
    CI_DIR = 'ci'  
  }  
  
  steps {  
    dir("${CI_DIR}") {  
      sh './image_delivery.sh'  
    }  
  }  
}
```

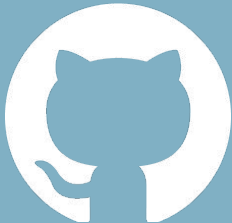


Implementazione CI/CD - Esecuzione pipeline





Grazie per l'attenzione



<https://github.com/vtramo/restful-booker-platform>

<https://github.com/vtramo/rbp-jenkins-shared-library>