Konrad Gomulka

Luca Salvigni

Manuele Pasini

Schema presentazione

- **Introduzione**
  - Cos’è Checkers
- **Requisiti**
  - Funzionali
  - Non funzionali
- **Design**
  - Architettura
  - Struttura
  - Comportamento
  - Interazione
- **Implementazione**
- **DevOps**
  - Testing
  - CI
  - Delivery & Deployment
- **Lavori futuri**
- **Demo dell’applicazione**

CHECKERS

Introduzione

Cos’è Checkers

Checkers rappresenta una versione multiplayer online del gioco da tavolo dama.

Le regole di gioco sono quelle della dama internazionale in cui, a differenza di quella italiana le pedine semplici possono mangiare anche le Dame.

I pezzi si muovono diagonalmente solo su caselle non occupate da altri pezzi ed hanno la possibilità di prendere (o “mangiare”) quelli avversari, scavalcandoli.

I pezzi catturati vengono rimossi dalla damiera ed esclusi dal gioco.

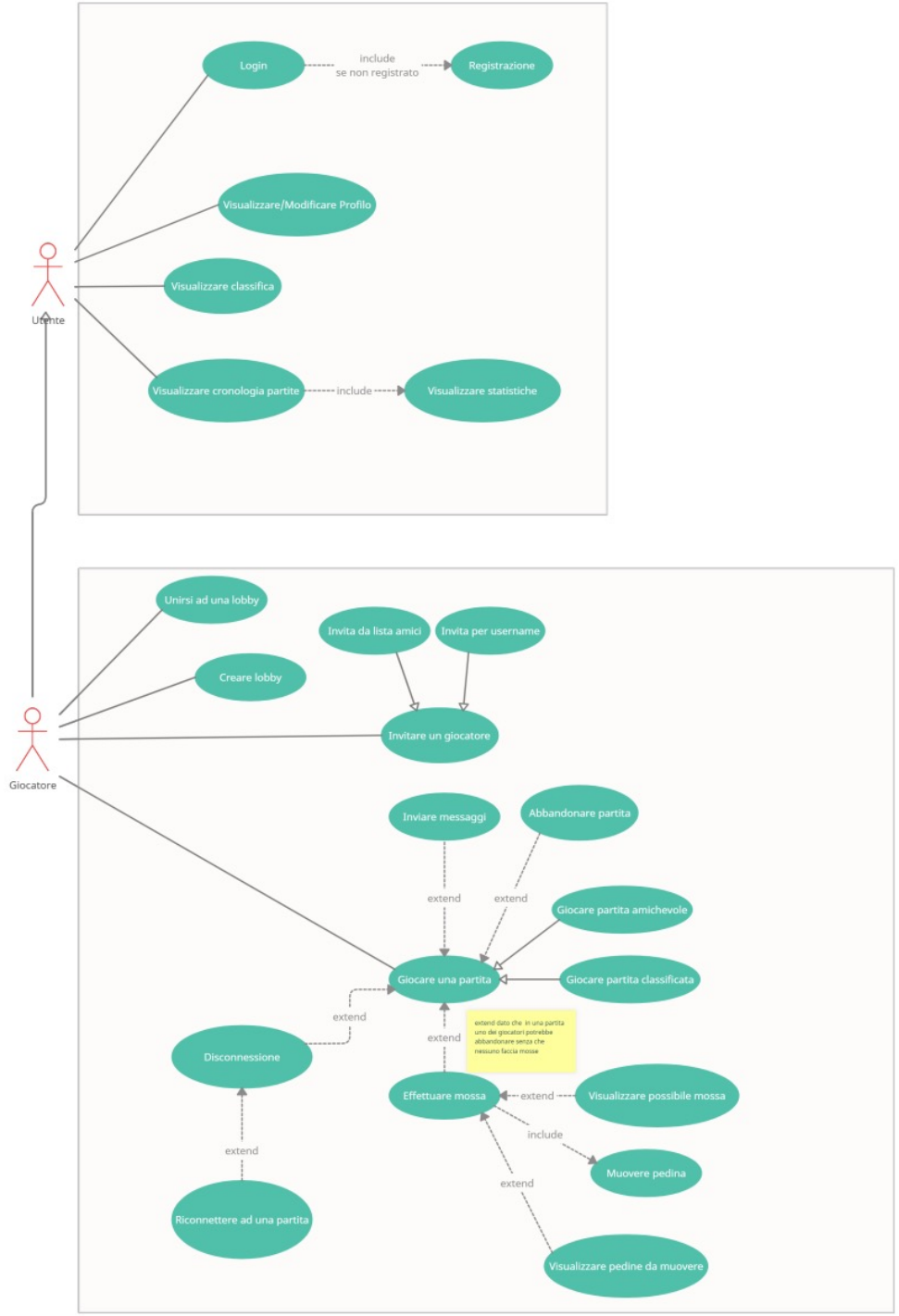
Il giocatore a cui vengono catturati tutti i pezzi o che, durante il suo turno è impossibilitato a muovere, ha perso.

Se, dopo aver effettuato una presa e nella casella di arrivo si presentano le condizioni per la pedina di una nuova presa è obbligatoria la presa multipla, che va effettuata nello stesso turno di gioco.

Requisiti

Requisiti Funzionali

- Meccanismo di autenticazione e registrazione
- Personalizzazione profilo utente
- Creazione lobby
- Visualizzazione di lobby aperte
- Possibilità di entrare in una lobby e partecipare ad una partita
- Meccanismo ad inviti
- Visualizzazione storico partite
- Visualizzazione classifica globale
- Meccanismo di competizione a punti

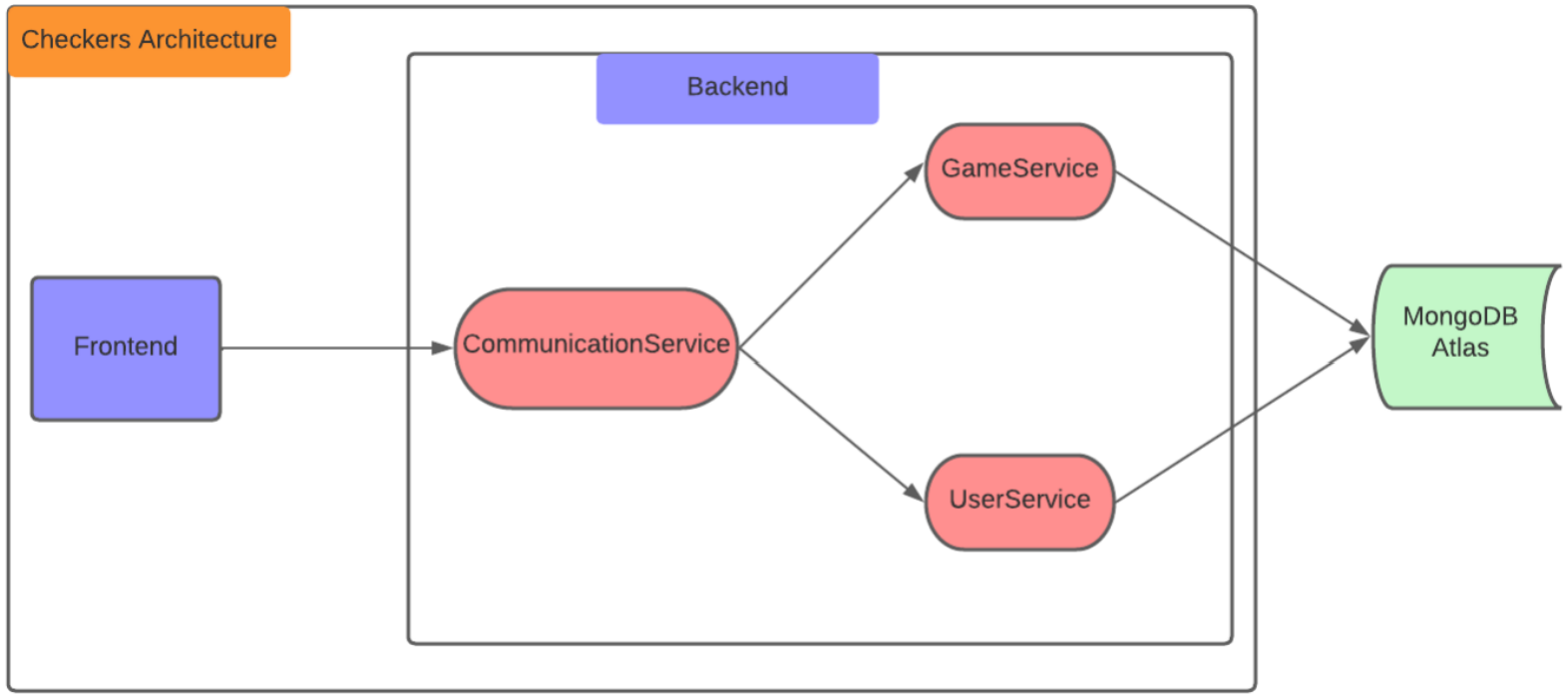

Login
include
se non registrato
Registrazione
Visualizzare/Modificare Profilo
Visualizzare classifica
Visualizzare cronologia partite
include
Visualizzare statistiche
Utente
Unirsi ad una lobby
Invita da lista amici
Invita per username
Creare lobby
Invitare un giocatore
Giocatore
Inviare messaggi
Abbandonare partita
extend
extend
Giocare partita amichevole
Giocare una partita
Giocare partita classificata
extend
extend
extend dato che in una partita uno dei giocatori potrebbe abbandonare senza che nessuno faccia mosse
Disconnessione
Effettuare mossa
extend
Visualizzare possibile mossa
include
Muovere pedina
extend
extend
Riconnettere ad una partita
Visualizzare pedine da muovere

Requisiti Non Funzionali

- Scalabilità
- Disponibilità
- Disaccoppiamento
- Resilienza
- Manutenibilità
- Testabilità
- Qualità del codice
- Facilità d’implementazione

Design

Architettura



- Stack MEVN

- MongoDB
- Express
- Vue
- Node.js

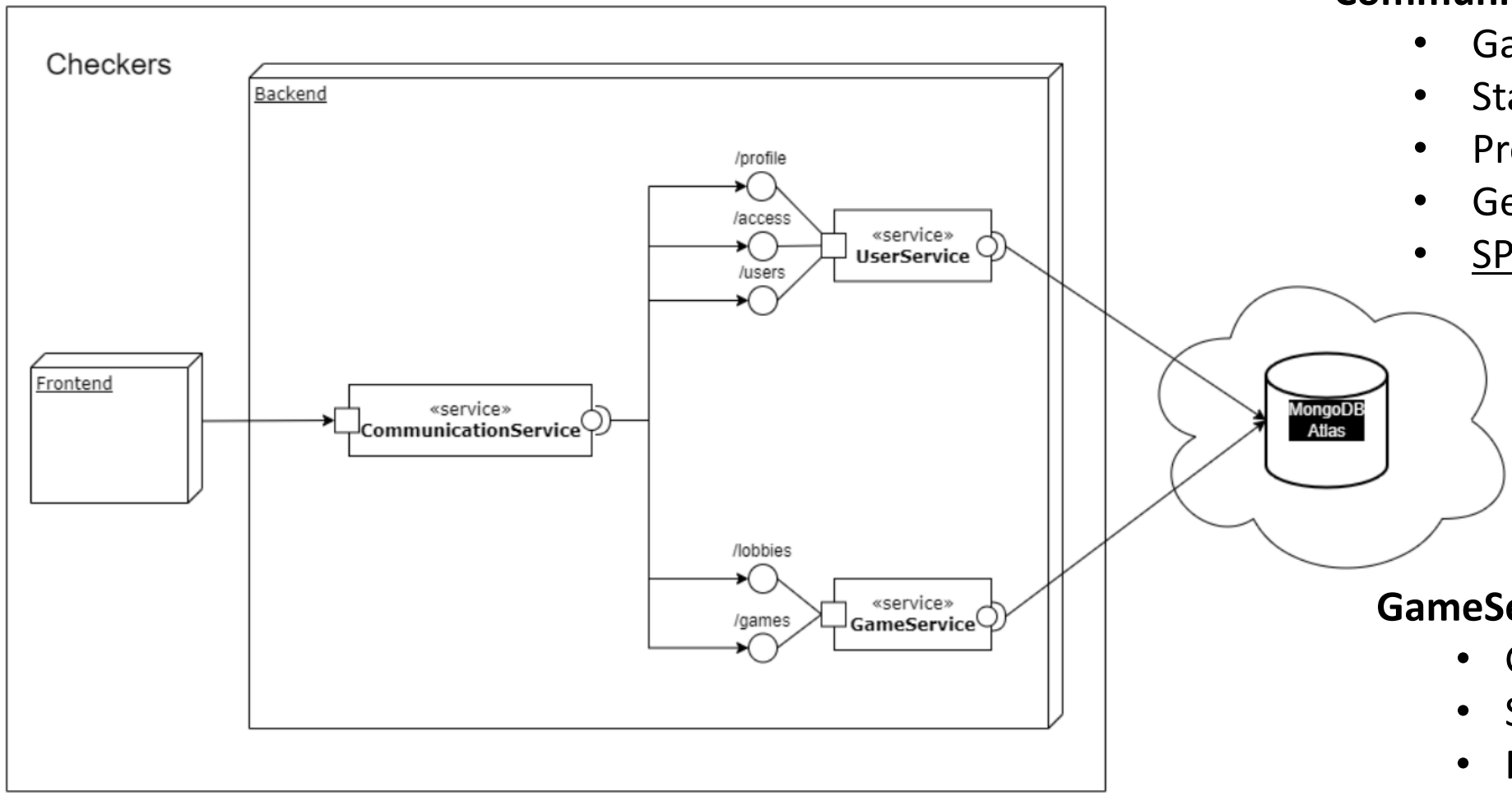
- Architettura a microservizi

- Scalabilità
- Disponibilità
- Manutenibilità
- Modularità

- Cloud Storage

- Persistenza
- Outsourcing
- Gestione indiretta tramite API

Struttura



CommunicationService

- Gateway del sistema
- Stateful
- Pro-attivo
- Gestisce logica applicativa
- SPOF

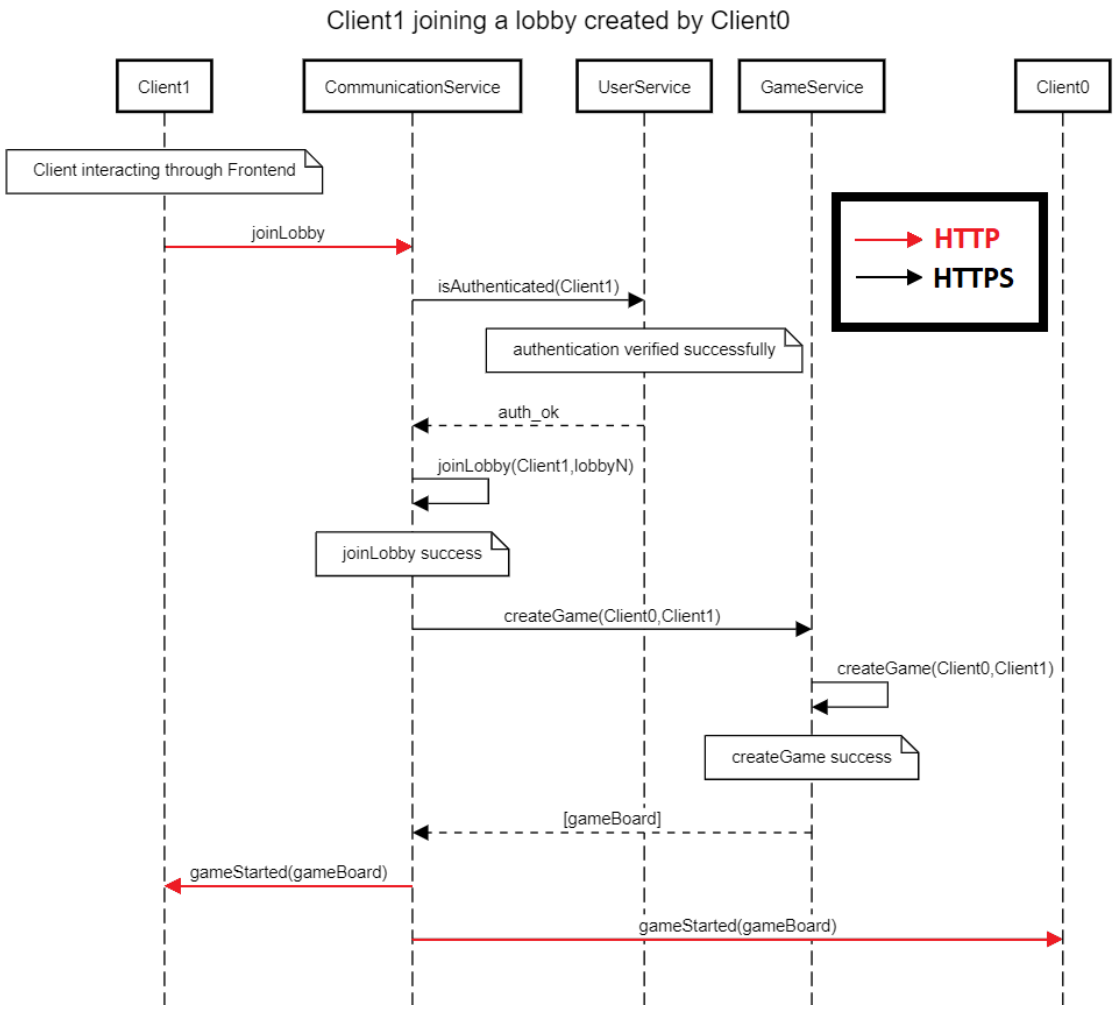
UserService

- Gestione utenti
- Stateless
- Passivo

GameService

- Gestione partite
- Stateless
- Passivo

Gestione interazione e sicurezza



- Due flussi di comunicazione

- *Client – CommunicationService*: HTTP
- *CommunicationService – Game/UserService*: HTTPS

- Access Control

- Autenticazione tramite credenziali (SHA-512) e memorizzate tramite MongoDB Atlas
- Verifica integrità client attraverso JWT

- Vulnerabilities

- CommunicationService è un SPOF, la sua interruzione comporta la perdita dello stato del sistema
- Modellare i due servizi stateless come tali comporta un flusso informativo intenso verso il DB

Implementazione

Tecnologie Utilizzate

Backend:

- Node, Axios, Express, Gulp, Socket-IO, Mongoose, JWT

Frontend:

- Vue3, DaisyUI, TailwindCSS, Vuex, VueRouter, Web-Socket(Vue-socket.io e Socket.io-client)

Testing:

- Mocha-Chai, Socket-IO-client, Vitest, Vue Test Utils

Deploy:

- Docker

DevOps

Testing

- Gulp, task runner JavaScript per eseguire I controlli ad ogni commit
- Per il testing sono state utilizzate le librerie mocha e chai, oltre ad axios per effettuare richieste alle API esposte dai microservizi
- Lato Frontend sono stati utilizzati il framework Vitest ed il plugin messo a disposizione da Vue Test Utils
- La coverage dei test è stata calcolata utilizzando la libreria Istanbul
- ESLint

```
function qualityAssurance(cb) {
  log('checking code quality');
  return exec('npm run lint', (err, stdout, stderr) => {
    log(stdout);
    log(stderr);
    cb(err);
  });
}

function doTest(cb) {
  log('checking all tests');
  return exec('npm test', (err, stdout, stderr) => {
    log(stdout);
    log(stderr);
    cb(err);
  });
}

function testQualityAssurance(cb) {
  log('test coverage');
  return exec('npm run coverage', (err, stdout, stderr) => {
    log(stdout);
    log(stderr);
    cb(err);
  });
}
```

CI

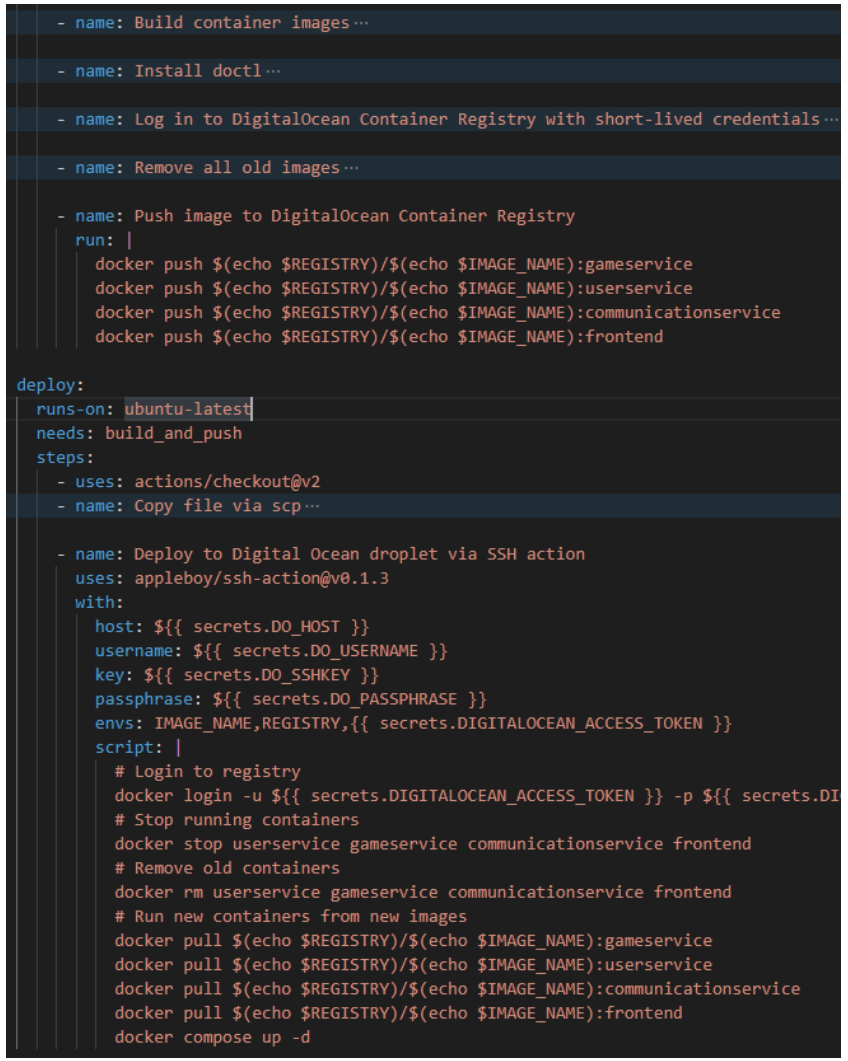
- GHA
- Build Matrix
- On Path for each Service
- Secrets
- Testing
- Dependabot

```yaml
name: gameservice

on:
  push:
    paths:
      - 'Backend/Checkers-GameService/**'
  workflow_dispatch:
  workflow_call:

env:
  GameService_PORT: 3032
  DB_PSW: ${{ secrets.DB_PSW }}
  GAME_KEY: ${{ secrets.GAME_KEY }}
  GAME_CERT: ${{ secrets.GAME_CERT }}
  CERTIFICATE: ${{ secrets.CERTIFICATE }}

jobs:
  build:
    strategy:
      fail-fast: false
      matrix:
        os: [ubuntu-latest, windows-latest, macos-latest]
        node: [ 14, 16, 18]
    runs-on: ${{ matrix.os }}
    steps:
      - uses: actions/setup-node@v3
        with:
          node-version: '16'
      - uses: actions/checkout@v2
      - name: Install dependencies
        run: |
          cd Backend/Checkers-GameService
          npm ci
      - name: Test
        run: |
          cd Backend/Checkers-GameService
          npm run gulp test
```

Delivery & Deployment

- Deployment in caso di push su main
- DigitalOcean
- Push delle Docker Images

1. Vengono azionati i workflow di test relativi a ciascun microservizio (workflow_call)
2. Viene calcolata la Semantic Version della release
3. Mediante il comando docker-compose vengono buildati i file e create le immagini
4. Viene effettuato il login sulla piattaforma mediante un plugin apposito
5. Le vecchie immagini vengono rimosse dal Container Registry di DigitalOcean
6. Le nuove immagini vengono trasferite nel Container Registry di DigitalOcean direttamente mediante Docker CLI
7. Per evitare ridondanza viene trasferito anche il file docker-compose.yml
8. I vecchi container vengono fermati ed eliminati per azionare quelli nuovi mediante comando docker-compose
9. Utilizzando la Semantic Version calcolata ed il commit message più recente viene creata la release su Github (come draft)



```
    - name: Build container images···

    - name: Install doctl···

    - name: Log in to DigitalOcean Container Registry with short-lived credentials···

    - name: Remove all old images···

    - name: Push image to DigitalOcean Container Registry
      run: |
        docker push $(echo $REGISTRY)/$(echo $IMAGE_NAME):gameservice
        docker push $(echo $REGISTRY)/$(echo $IMAGE_NAME):userservice
        docker push $(echo $REGISTRY)/$(echo $IMAGE_NAME):communicationservice
        docker push $(echo $REGISTRY)/$(echo $IMAGE_NAME):frontend

deploy:
  runs-on: ubuntu-latest
  needs: build_and_push
  steps:
    - uses: actions/checkout@v2
    - name: Copy file via scp···

    - name: Deploy to Digital Ocean droplet via SSH action
      uses: appleboy/ssh-action@v0.1.3
      with:
        host: ${{ secrets.DO_HOST }}
        username: ${{ secrets.DO_USERNAME }}
        key: ${{ secrets.DO_SSHKEY }}
        passphrase: ${{ secrets.DO_PASSPHRASE }}
        envs: IMAGE_NAME,REGISTRY,{{ secrets.DIGITALOCEAN_ACCESS_TOKEN }}
        script: |
          # Login to registry
          docker login -u ${{ secrets.DIGITALOCEAN_ACCESS_TOKEN }} -p ${{ secrets.DI
          # Stop running containers
          docker stop userservice gameservice communicationservice frontend
          # Remove old containers
          docker rm userservice gameservice communicationservice frontend
          # Run new containers from new images
          docker pull $(echo $REGISTRY)/$(echo $IMAGE_NAME):gameservice
          docker pull $(echo $REGISTRY)/$(echo $IMAGE_NAME):userservice
          docker pull $(echo $REGISTRY)/$(echo $IMAGE_NAME):communicationservice
          docker pull $(echo $REGISTRY)/$(echo $IMAGE_NAME):frontend
          docker compose up -d
```

Lavori Futuri

Lavori futuri

- Riconnessione di un giocatore ad una partita entro un certo tempo limite
- Recupero partite in corso durante un'eventuale interruzione del servizio(partite già salvate su DB)
- Meccanismo di fault-tolerance(es: heartbeat Hadoop)
- Maggiore personalizzazione per l'utente(es: cambio colore pedina, cambio colore board)
- Pubblicazione del gioco
- Estensioni del gioco(es: Trapdoor Checkers)

Demo dell’applicazione