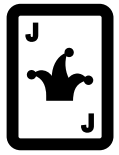
Dixit

Sistemi Distribuiti –

Daniele Di Lillo

Descrizione del progetto



Dixit

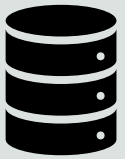
Gioco di carte basato su immagini fantasiose e titoli, il cui scopo è indovinare la carta del narratore

Client-Server

Come approccio di comunicazione distribuita tra nodi

Attori

Paradigma ad attori utilizzato per gestire i match e le comunicazioni tra player di uno stesso match



Database

Database di tipo MySql e container Docker utilizzati per virtualizzare servizi necessari alla gestione dei dati (DBMS e database)

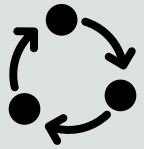
Il gioco

Narratore

Il ruolo di narratore permette al giocatore di scegliere una carta dalla sua mano e un titolo fantasioso per quell'immagine

Giocatori

I giocatori, una volta saputo il titolo scelto, dovranno proporre una loro carta.



Turni

Mischiate le varie scelte insieme alla carta del narratore, dovranno poi indovinare quella effettivamente scelta (tranne la propria)
I giocatori, a turno, diventano narratori

Tecnologie usate

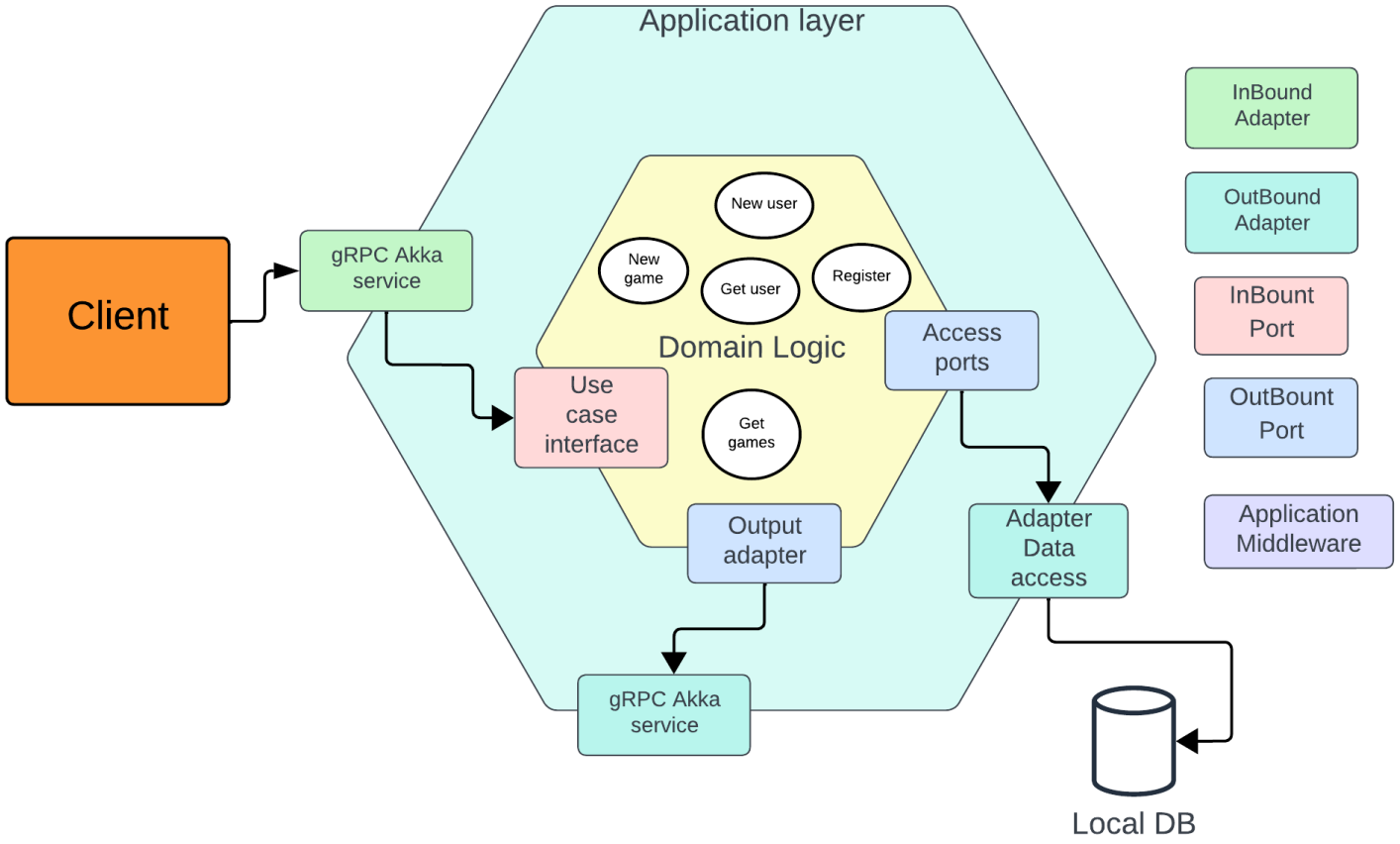
SVILUPPO

- Scala: linguaggio di programmazione funzionale
- Akka Typed Actor: framework per programmazione distribuita ad attori
- Akka gRPC: framework per creazione di streaming gRPC tra client – server

DEPLOY E TESTING

- Docker e TestContainer
- MySQL
- Akka Asynchronous Testing

Client – Server design



Architettura Esagonale

- Business Logic layer
- Utilizzo di porte e adapter

Pregi:

- Componenti intercambiabili e debolmente accoppiati
- Forte indipendenza della business logic dal livello applicativo/infrastrutturale

Interazioni C-S

Casi d'uso

- Login
- Registrazione
- Punteggio
- Lista partite aperte

gRPC

I casi d'uso vengono tradotti in operazioni tra Client e Server sfruttando gRPC

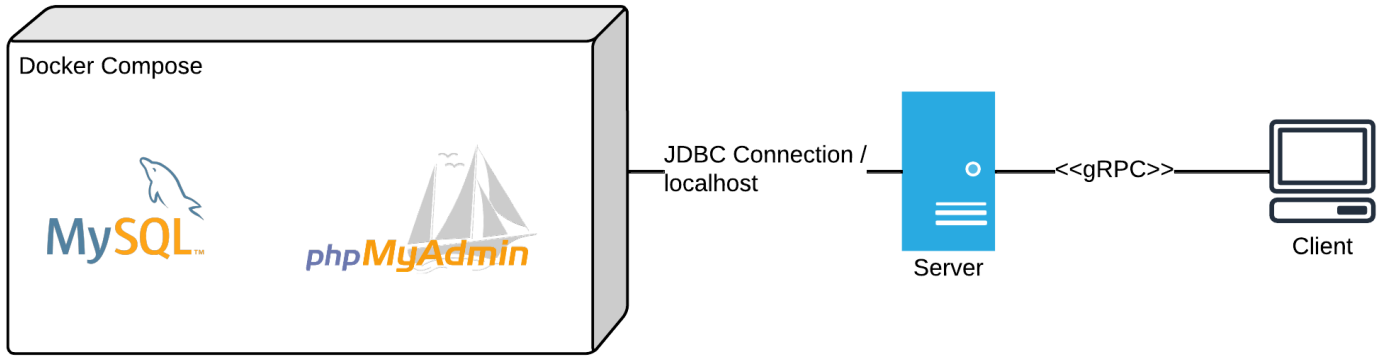
JDBC

Connessioni JDBC per gestire il DB lato server

Database

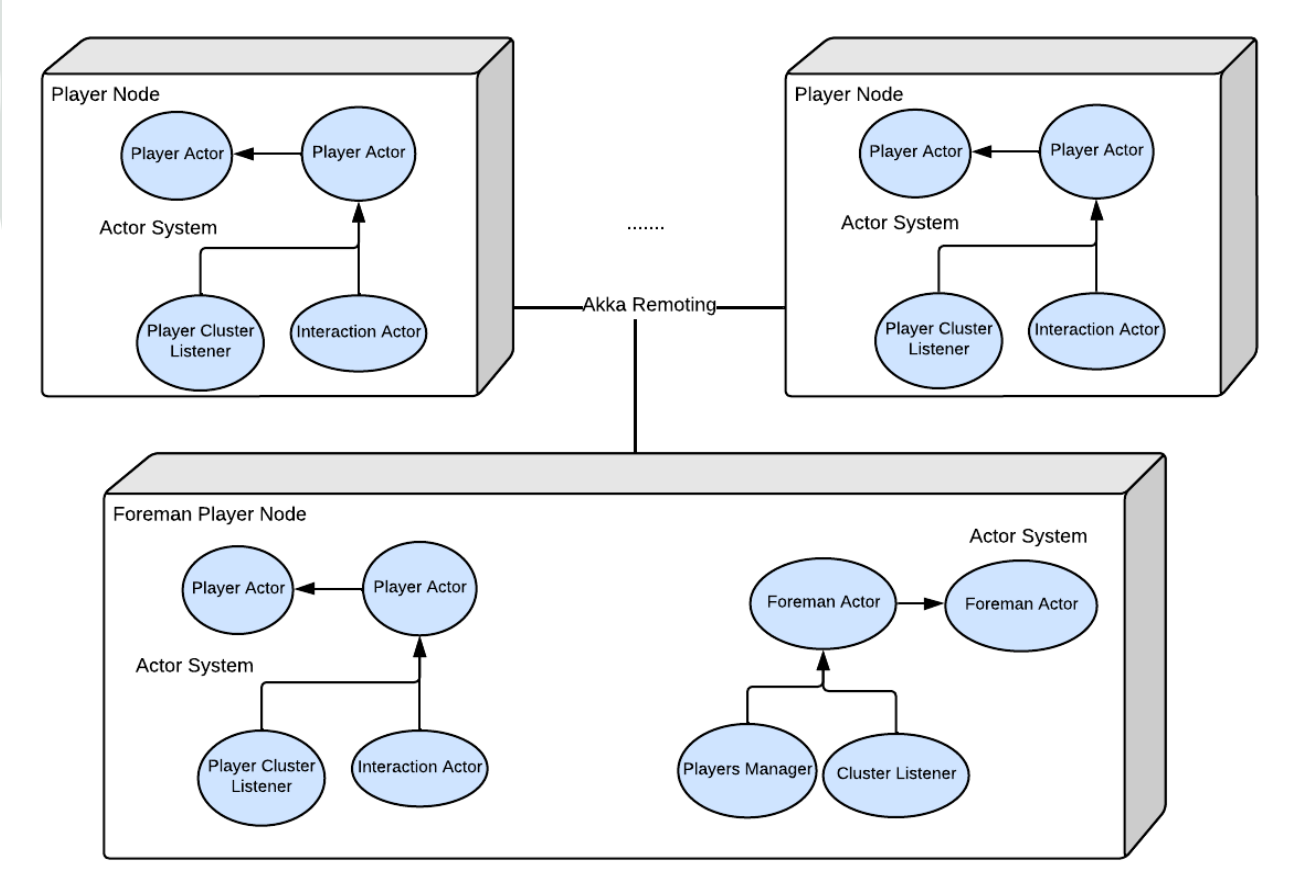
Istanza docker:

- MySQL
- phpMyAdmin



Actor System e Cluster design (1)

- Match gestito mediante un Cluster di Actor System
- Ogni Actor System è composto da più attori
- Svariati tipi di attori:
  - User guardian (entry point per ogni AS)
  - Player
  - Capopartita (Foreman)
  - Event listener
  - Gestore dei giocatori
  - Receptionist



Actor System e Cluster design (2)

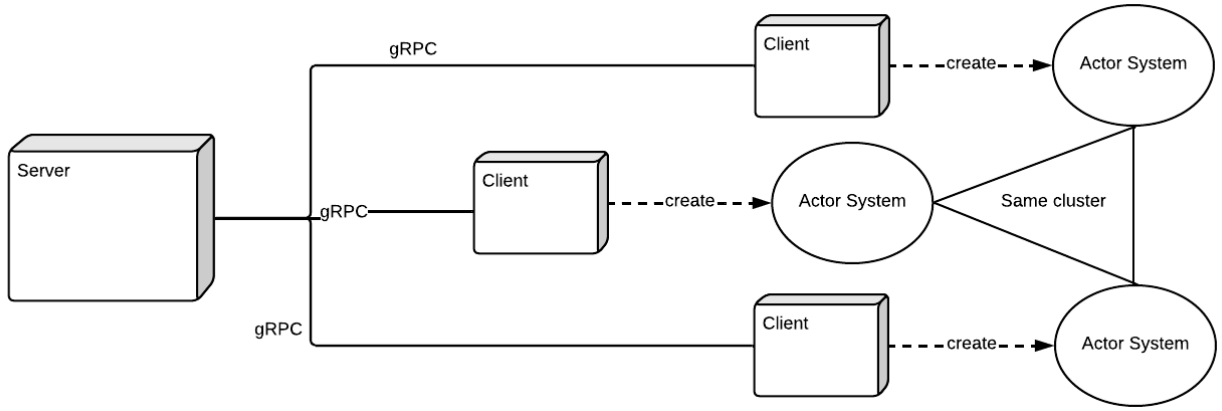
Everything is an actor

- Entità che incapsulano uno stato e un comportamento
- Scambio di messaggi tramite mailbox
- Lifecycle dell'attore
- Child actors

Cluster Membership Service

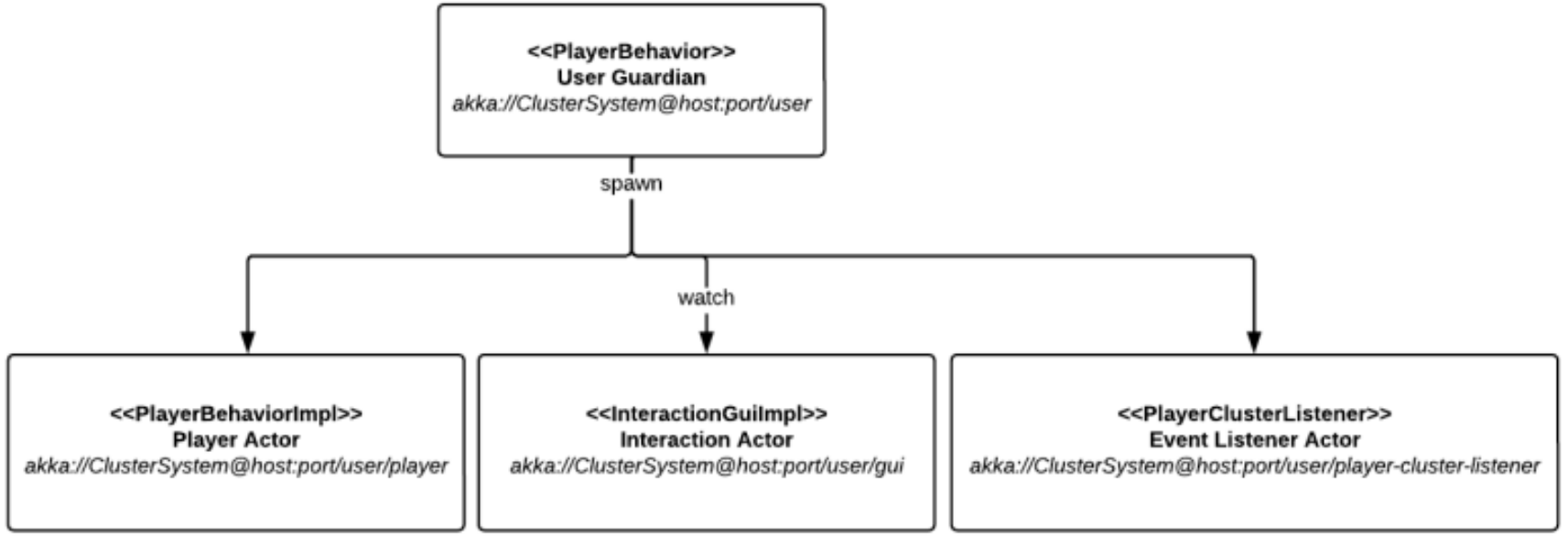
- Gossip Protocol
- Failure Detector
- Members Events

Interazioni tra attori e AS (1)



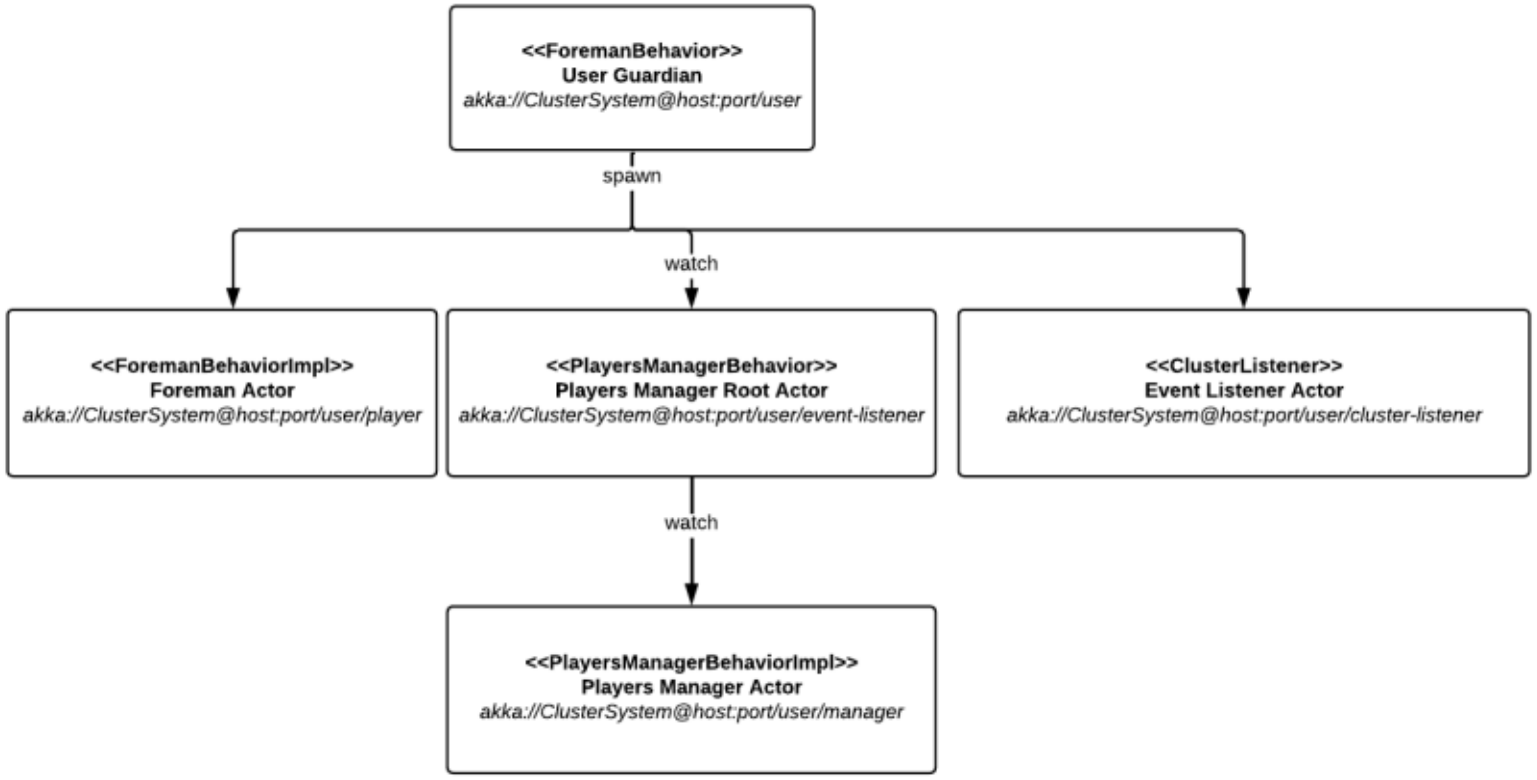
Architettura finale:

- Unione di architettura Client – Server e paradigma ad attori
- Miglior gestione delle responsabilità
- Scalabilità ottimale
- Gestione failures ad hoc mediante supervisione degli attori
  - Uscita di uno o più giocatori
  - Unreachability di uno o più giocatori

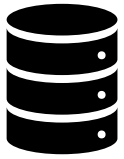


AS Player

AS Capopartita



Interazioni tra attori e AS (2)



Discovery

Servizio di Discovery dinamico degli attori

Message Delivery

Message Delivery di tipo Best Effort e at-most-once

Uscita del capopartita

Se il capopartita esce il match viene annullato e chiuso

Uscita di un giocatore con e senza re-join

Gestione del rientro di un giocatore uscito entro un determinato intervallo di tempo

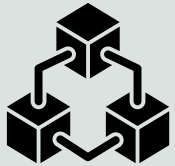
Testing

BDD Style testing

Testing effettuato sfruttando uno stile molto simile al Behavior Driven Development

Actor Test Kit

Simulazione e deploy di un cluster mediante Async Test Kit



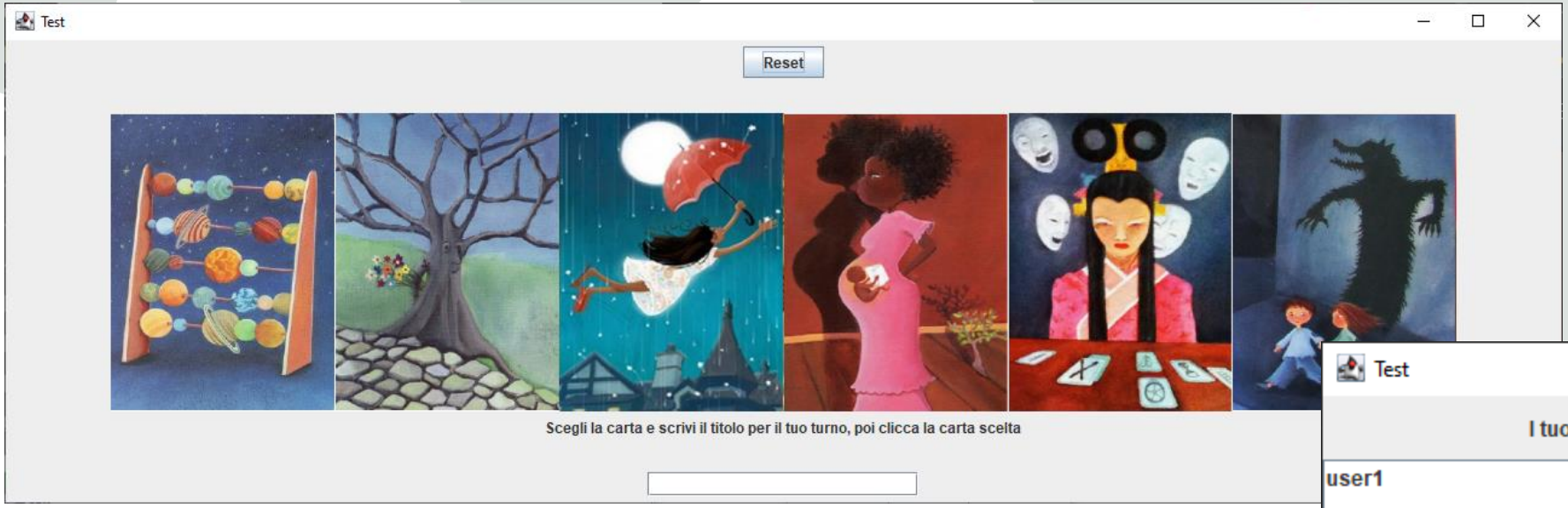
Scala Test

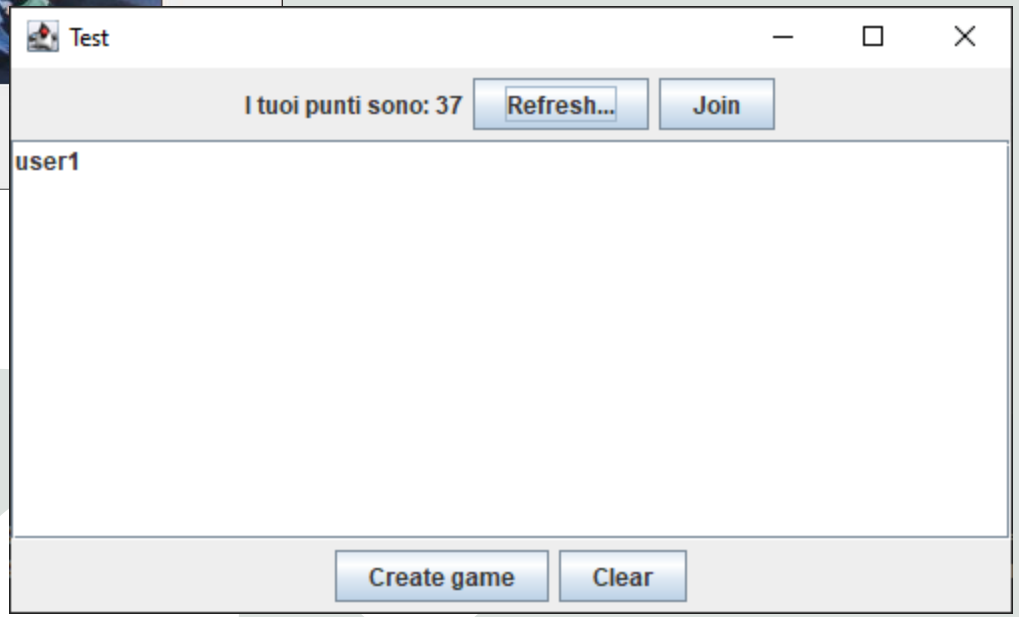
Framework di ScalaTest per effettuare Unit Test e test guidati dal behavior

Test Container

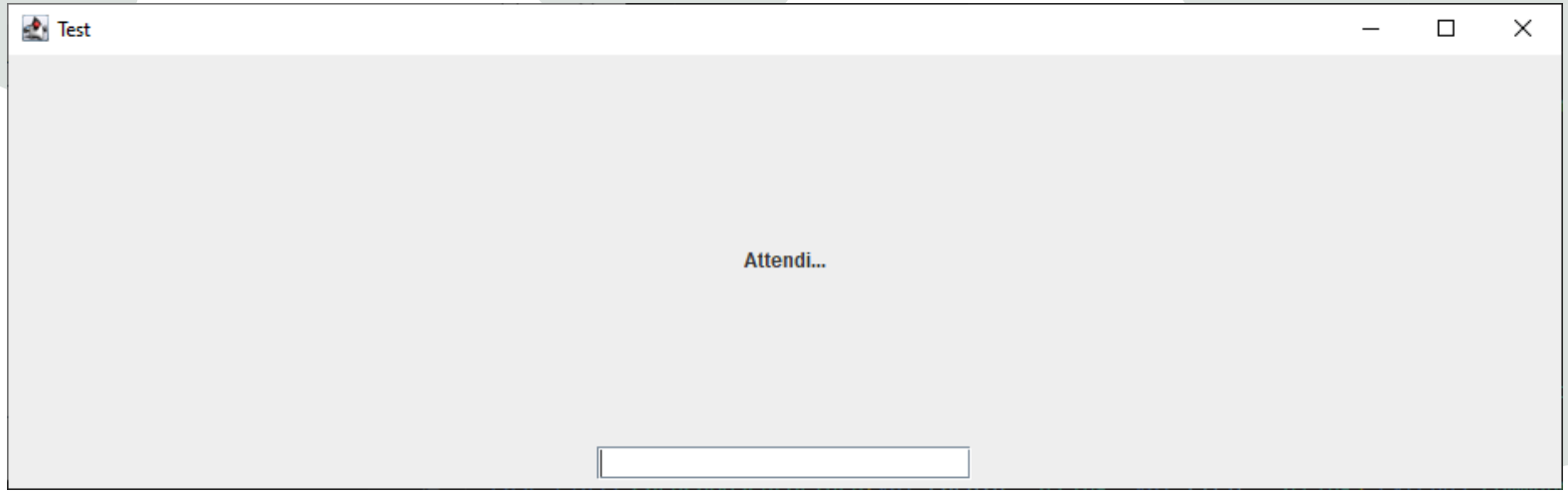
Framework open source per testare ed effettuare deploy di container docker

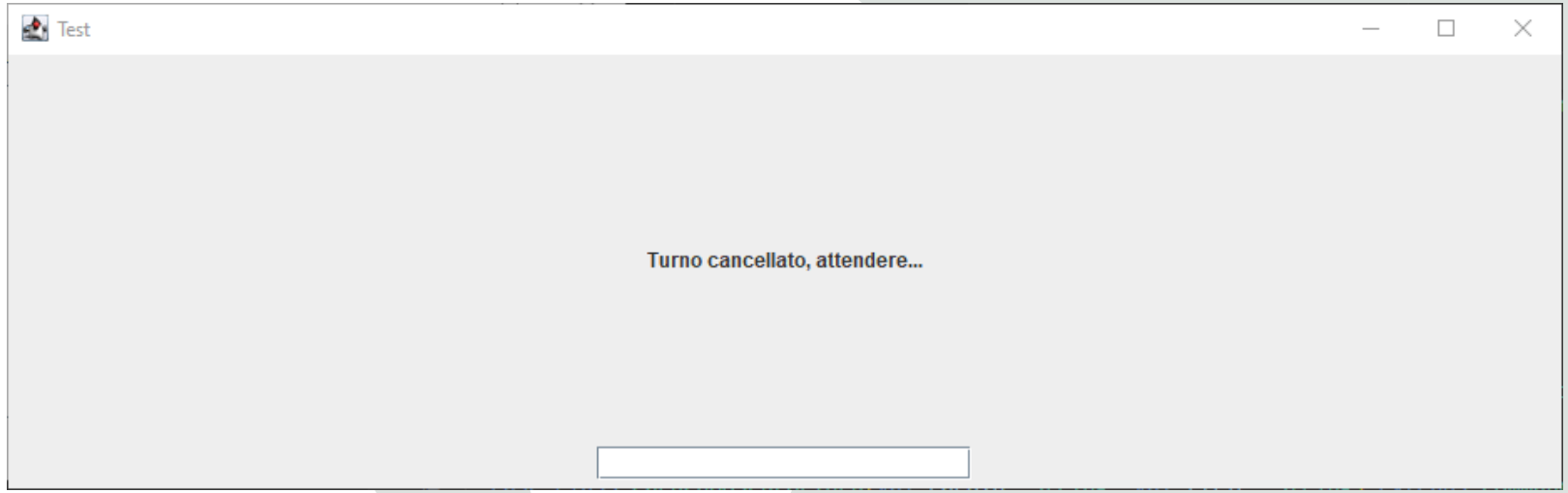
Esempio grafico (1)





Esempio grafico (2)





Esempio grafico (3)