Distributed Team Chess

Un sistema di scacchi cooperativo che dimostra i principi fondamentali dei sistemi distribuiti attraverso il gioco di squadra e il consenso distribuito.

SISTEMI DISTRIBUITI

CONCETTO

Una Nuova Dimensione degli Scacchi

A differenza degli scacchi tradizionali, questo sistema introduce una meccanica basata su **Role Sharding** e **Consenso Distribuito**.

I giocatori sono organizzati in squadre dove ognuno ha permessi esclusivi su un sottoinsieme specifico di pezzi.

Per eseguire una mossa, il team deve raggiungere il consenso attraverso un protocollo di voto, trasformando il gioco in una sfida di coordinamento distribuito.

È un nuovo modo di giocare a scacchi, decisamente meno competitivo e concentrato sul divertimento con amici.

REQUISITI

Requisiti Funzionali Chiave

Il sistema è progettato per supportare un'esperienza di gioco degli scacchi unica, focalizzata sulla collaborazione e sui principi dei sistemi distribuiti.

Modalità di Gioco

Classica (1v1) o **Team Consensus** (e.g., 2v2) con controllo dei pezzi partizionato tra i compagni di squadra.

Interazione Real-Time

Comunicazione a bassa latenza per sincronizzare mosse, proposte e voti tra i giocatori distribuiti.

Interfaccia Web

Accesso tramite moderni browser web con supporto WebSockets e feedback visivo per ruoli e votazioni.

Archiviazione Dati

Stato del gioco e proposte memorizzati in un **Redis in-memory store** per alta disponibilità e velocità.

Architettura del Sistema

Broker-Based Hybrid P2P

Backend .NET scalabile che orchestra la comunicazione via SignalR e persiste lo stato su Redis.

Client Angular (N istanze)

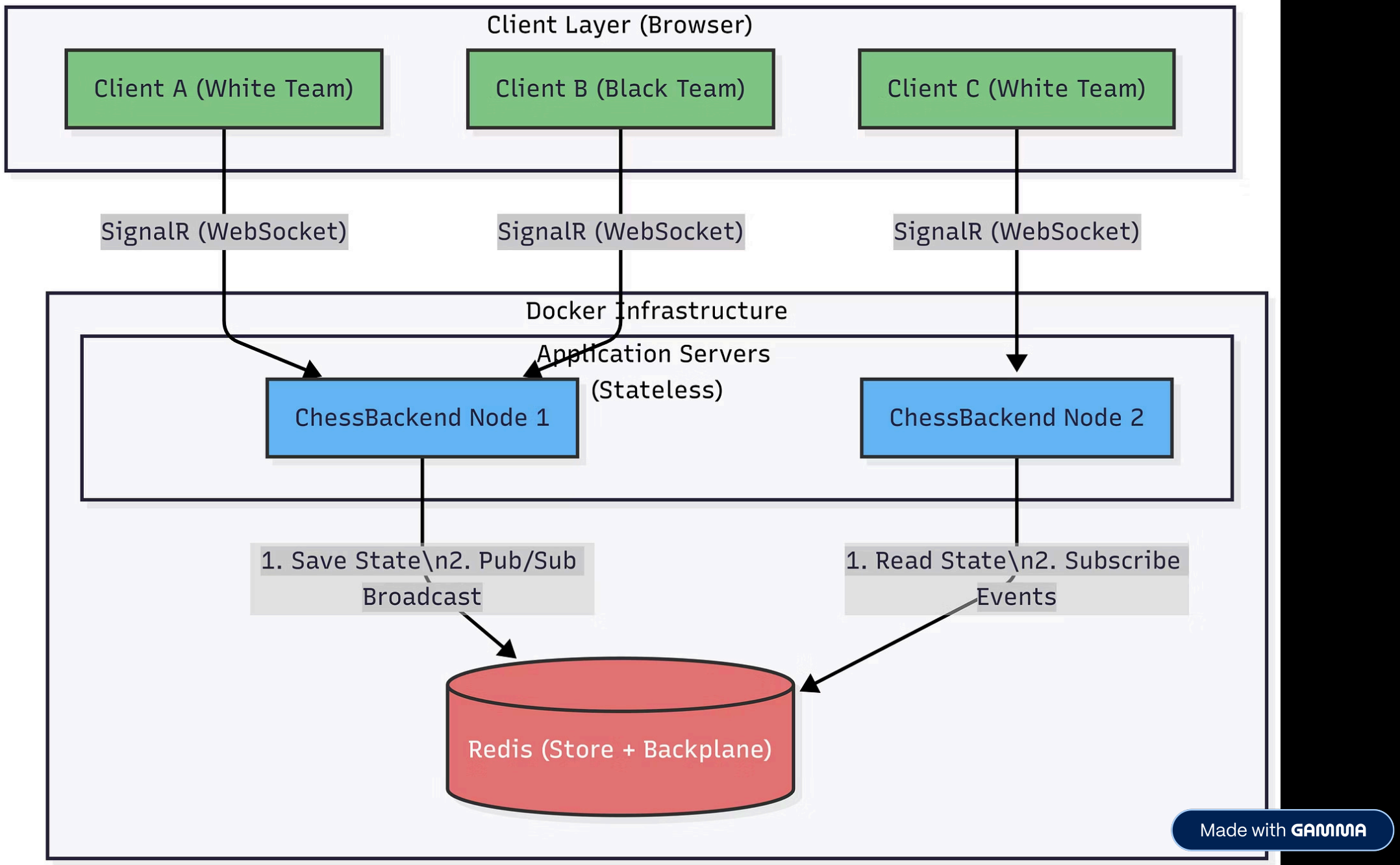
Interfacce utente con SignalR

Backplane Redis

Sincronizza stato e messaggi

Backend .NET (M istanze)

Elaborazione, controlli e logica



IMPLEMENTAZIONE

Stato del Sistema

Lo stato globale del sistema, persistito in Redis, è essenziale per la consistenza e la sincronizzazione del gioco.

Configurazione Sessione

Metadati statici: `GameId`, `Mode` (1v1 o Team Consensus), `Capacity`.

Partecipazione e Ruoli

Definisce i giocatori connessi (`Players`) e l'assegnazione ai team (`Teams`).

Livello di Gioco

Configurazione scacchiera (`FEN`) e permessi sui pezzi (`Sharding Map`).

Livello di Coordinamento

Proposte attive (`ActiveProposals`) e timestamp di sincronizzazione (`LastMoveAt`).

IMPLEMENTAZIONE

Analisi CAP: Consistenza e Tolleranza alle Partizioni

Il nostro sistema di scacchi cooperativo è stato progettato con attenzione ai principi del Teorema CAP, privilegiando la coerenza dei dati in un ambiente distribuito.

Sistema CP

Il design del sistema prioritizza la **Consistenza (C)** e la **Tolleranza alle Partizioni (P)**, garantendo che tutti i client vedano sempre lo stesso stato del gioco, anche in presenza di interruzioni di rete.

Sacrificio dell'Availability

Per prevenire scenari di "Split-Brain", abbiamo scelto di sacrificare l'Availability completa. Se il database Redis dovesse cadere, il sistema si arresta per preservare la **Strong Consistency**.

Gestione dello Stato

Lo stato globale del gioco è interamente persistito in Redis come oggetti JSON. I server applicativi .NET operano in modalità stateless, senza stato in RAM, facilitando scalabilità e resilienza.

IMPLEMENTAZIONE

Sharding e Sicurezza

Il cuore del sistema risiede nella gestione distribuita dei permessi e nella robustezza della validazione, garantendo un controllo preciso su ogni pezzo e ogni mossa.

Dynamic Sharding

All'avvio, i permessi sui pezzi vengono assegnati dinamicamente ai giocatori, ripartendo il controllo sulla scacchiera in base ai ruoli definiti.

Attribute-Based Access Control

Ogni mossa è autorizzata solo se il giocatore soddisfa attributi specifici: deve essere il turno corretto, possedere il pezzo e appartenere al team appropriato.

Il Protocollo di Consenso per le mosse

Proposta

Giocatore A invia mossa.

Il server valida solo i permessi, non la logica della mossa

Propagazione

Invio via backend ai peer

Validazione

Peer eseguono controlli distribuiti

Raccolta

Aggregazione dei voti dei peer

Commit

Applica aggiornamento FEN finale

IMPLEMENTAZIONE

Dettagli Tecnologici: Lo Stack

La soluzione sfrutta un ecosistema full-stack moderno, progettato per offrire alte prestazioni, manutenibilità e scalabilità in un ambiente distribuito.

Backend: .NET 8 Ecosystem

ASP.NET Core per web server Kestrel, SignalR Core per comunicazione real-time (WebSocket) e Hosted Services per la logica di gioco asincrona.

Frontend: Angular 20

Utilizza RxJS per la gestione reattiva degli eventi e Angular Signals per un rendering efficiente della UI con aggiornamenti fini e performanti.

Infrastruttura: Docker & Redis

Containerizzazione Docker per coerenza ambientale e simulazione cluster. Redis per persistenza in-memory a bassa latenza.

Librerie di Dominio

chess.js (TypeScript) per validazione lato client e ChessDotNetCore (C#) come validazione autorevole per le regole lato server.

CONCLUSIONI

Fault Tolerance e Testing

1

Consenso e Sharding

Match 2vs2 con 4 browser: verifica che lo **sharding** blocchi mosse non autorizzate e che il voto di maggioranza committi la mossa su tutti gli schermi.

2

Node Crash e Failover

Chiusura improvvisa del browser: il sistema rileva la disconnessione e trasferisce automaticamente i permessi ai compagni sopravvissuti.

3

Liveness e Timeout

Team che non vota: allo scadere del timer (120s), il **ProposalTimeoutService** forza una risoluzione per prevenire deadlock.

4

State Recovery

Refresh della pagina (F5): il client si riconnette, richiede lo stato da Redis e riprende il gioco senza perdita di dati.

5

Attacco Hacker

Modifica JavaScript per proporre mossa illegale: i nodi peer rifiutano la proposta, e il backend la blocca definitivamente via validazione autoritativa.

CONCLUSIONI

Demo

YouTube

DistributedChess Demo