1. Titolo (0 minuti)
2. Cos'è WASI (2.5 minuti)

L’argomento centrale di questa presentazione è WASI, acronimo di WebAssembly System Interface. WebAssembly è un set di istruzioni per una macchina virtuale stack-based. Con WebAssembly siamo in grado di eseguire codice scritto in diversi linguaggi di programmazione all'interno del browser web, ottenendo prestazioni molto elevate rispetto ad altri linguaggi interpretati come JavaScript. Una delle limitazioni di WebAssembly è l'accesso alle risorse del sistema ospitante, come il file system o la rete, per motivi di sicurezza.

È qui che entra in gioco WASI: una specifica che definisce un'interfaccia standard per l'accesso alle risorse di sistema da parte di WebAssembly.

WASI offre un'interfaccia standardizzata e portabile per l'esecuzione di codice WebAssembly in un ambiente sandboxed, isolato dal sistema operativo sottostante. Questo approccio garantisce un alto livello di sicurezza, in quanto i moduli WASM sono limitati nella loro capacità di accedere alle risorse del sistema e di interagire con l'esterno. In questo modo, le applicazioni WASI sono in grado di eseguire operazioni critiche in modo sicuro e isolato dal resto del sistema.

Inoltre, WASI è altamente portabile, poiché è stato progettato per funzionare su qualsiasi piattaforma che supporta WebAssembly, compresi browser, dispositivi embedded e server. Questa portabilità lo rende un'opzione ideale per lo sviluppo di applicazioni multipiattaforma altamente sicure e affidabili.

La sicurezza è basata su un sistema di capabilities, il quale fornisce un ulteriore strato di sicurezza, in cui per ogni modulo Wasm si possono specificare le capacità e le risorse di sistema alle quali può accedere. Ciò garantisce una maggiore protezione contro eventuali attacchi esterni. Una parentesi importante riguarda le socket, le quali non sono componenti gerarchiche e non possono essere create dal sistema stesso. Questo significa che il processo riceve una capability che gli consente di utilizzare la socket, ma non può creare o distruggere direttamente la socket stessa. Essa deve essere passata dall'esterno.

1. Performance (2 minuti)

Per misurare le prestazioni di WASI, sono stati condotti dei test di benchmark confrontando l'esecuzione di un'applicazione C e il corrispondente codice Javascript per il calcolo del determinante di una matrice tramite decomposizione LU. La versione C è stata compilata in WASM e lanciata con WASI, mentre la versione Javascript è stata lanciata con Node. Ho confrontato il tempo di esecuzione tra l'eseguibile nel formato ELF, la versione WASI e la versione Javascript.

Si può osservare che per task brevi, si introduce una latenza non trascurabile a causa dell’inizializzazione della sandbox nel caso di WASI e della compilazione just-in-time del codice JS da parte del motore V8 di Node.

All’aumentare della complessità, si ha una stabilizzazione dell’andamento, che va ad evidenziare la convenienza dell’utilizzo di WASI rispetto a linguaggi interpretati come Javascript, in quanto si introduce un leggero decremento delle performance rispetto al corrispettivo eseguibile nativo, ma un consistente incremento rispetto a Node.

1. Proof of concept (1 minuto)

A scopo dimostrativo, è stata progettata un’applicazione che permettesse di integrare WASI con alcune delle tecnologie web di tendenza.

Come ambito si è scelto il processamento e la modifica delle immagini, per sfruttare a pieno le potenzialità di WASI a livello di performance. L’utente è in grado di caricare un’immagine e manipolarla attraverso una palette di comandi, tra questi la regolazione di luminosità e contrasto, effetti di sfocatura, aumento della nitidezza, ritaglio, conversione di formato e filtri.

1. Scelte tecnologiche (1 minuto)

Lo stack di tecnologie utilizzato per questo progetto è costituito da diverse componenti. Sul lato del backend, si è scelto di utilizzare Node.js ed Express per la creazione di un'API RESTful. Per implementare le funzionalità di sandboxing e portabilità, si sono integrati WASI e STB, una libreria scritta in C basata su file singoli. Sul lato del frontend, si è utilizzato React per creare un'interfaccia utente responsive con time to market limitato.

Nel diagramma a fianco si può osservare la struttura gerarchica del backend. Nello strato superiore si pongono Node ed Express per la gestione delle richieste e la comunicazione col client in senso generale.

Ogni azione sul frontend è collegata a un diverso endpoint, il quale gestisce le richieste attraverso un’istanza di WASI. Questa verrà creata dopo la verifica della richiesta e gestirà l’esecuzione di un modulo WASM compilato a partire da un singolo file C, adibito a una funzione specifica.

1. Conclusioni (1 minuto)

In questa presentazione abbiamo visto come WASI può essere utilizzato per creare applicazioni web sicure e portabili.

I test di benchmark hanno dimostrato che l'utilizzo di WASI per eseguire codice WASM offre prestazioni simili all'esecuzione di codice nativo, con una maggiore portabilità e un compromesso sul tempo di esecuzione accettabile.

Il sistema delle capabilities di WASI offre un controllo fine sulle risorse a cui le applicazioni hanno accesso, aggiungendo un ulteriore livello alla sicurezza delle applicazioni web.

Inoltre, bisogna considerare che WASI è una tecnologia in evoluzione; perciò, allo stato attuale, non ha ancora la possibilità di esprimere tutto il suo potenziale. Ciononostante, possiamo affermare che WASI sia una tecnologia promettente per la creazione di applicazioni web sicure e portabili, con prestazioni paragonabili all'esecuzione di codice nativo di basso livello.

1. Grazie per l’attenzione (0 minuti)