MFA Core – Studio di Fattibilità

Si vuole sviluppare la componente server di un microservizio per l'autenticazione a più fattori (Multi-Factor Authentication).
In questo caso il fattore di autenticazione aggiuntivo sarà un dispositivo, con una nostra applicazione, precedentemente associato all'utente.
Il flusso completo necessario per un'approvazione sarà il seguente:
- il software utilizzatore richiederà al nostro microservizio l'approvazione di una operazione da parte di uno specifico utente
- l'utente, che avrà precedentemente registrato uno o più dispositivi come fattori di autenticazione, riceverà su ognuno di essi una notifica. Potrà quindi, attraverso uno di essi, decidere se accettare o meno l'operazione
- a quel punto l'esito verrà comunicato dal nostro microservizio al software utilizzatore

Per far sì che il dispositivo possa essere considerato un fattore di autenticazione per un determinato utente, ogni volta che richiederemo l'approvazione di un'azione dovremo verificare due cose:
- il dispositivo con cui stiamo comunicando è quello che è stato associato dall'utente
- il proprietario del dispositivo è effettivamente colui che può approvare l'azione
Per ottenere la prima sicurezza ci avvaleremo di una coppia di chiavi pubblica e privata.
Per ottenere la seconda useremo il protocollo OAuth2.0.

Il server da sviluppare dovrà presentare una serie di interfacce API attraverso le quali comunicherà con i dispositivi e con il software utilizzatore. Le API dovranno permettere le seguenti attività:
- aggiunta, rimozione, modifica e ottenimento informazioni da parte di un utente riguardo ai propri dispositivi
- ottenimento storico delle attività dei dispositivi
- ricezione di approvazioni da parte dei dispositivi
- ricezione di richieste di approvazione da parte del software utilizzatore
- controllo dello stato di un'approvazione da parte del software utilizzatore
- ottenimento informazioni riguardo al server di autenticazione. Questo punto è chiarito meglio nel paragrafo OAuth2.0.

Il server dovrà essere sviluppato in Node.js tramite il framework Express.

Oltre alle forme di comunicazione descritte sopra sarà necessario un sistema di notifica dal server verso i dispositivi, per richiedere l'approvazione di un'operazione. Per questo si utilizzerà Firebase Cloud Messaging.

Il server avrà bisogno di memorizzare i dati dei dispositivi associati e lo storico delle loro attività. Per questo verrà utilizzato MongoDB. Inoltre sarà necessario memorizzare alcune informazioni temporanee utili durante l'approvazione di un'operazione. Per questo verrà utilizzato Redis.

Tutto il server dovrà essere racchiuso in tre container Docker, con la suddivisione seguente:
- componente Node.js
- componente MongoDB
- componente Redis
Di seguito viene presentata un'introduzione ad ogni tecnologia precedentemente nominata.

Crittogafia asimmetrica

Quando si parla di crittografia a chiavi pubblica e privata, o asimmetrica, ci si riferisce all'utilizzo di una coppia di valori con due determinate proprietà crittografiche:
- se la chiave privata viene utilizzata per criptare dei dati, solo la chiave pubblica corrispondente potrà essere utilizzata per decrittarli.
- se la chiave pubblica viene utilizzata per criptare dei dati, solo la chiave privata corrispondente potrà essere utilizzata per decrittarli.

Per usufruire di queste proprietà la chiave privata va mantenuta effettivamente privata, ovvero non deve essere conosciuta da nessuno tranne che dal proprietario originale. La chiave pubblica invece può essere distribuita a chiunque.
Dalle proprietà crittografiche descritte derivano due funzioni, che rendono queste coppie utili:
- se A vuole inviare un dato a B con la sicurezza che solo B lo potrà leggere, sarà sufficiente che cripti il dato con la chiave pubblica di B e glielo invii
- se A vuole reinviare a B un documento firmato da lui, ovvero con una sorta di approvazione, basterà che cripti il documento con la propria chiave privata. B, decriptando il documento con la chiave pubblica di A, otterrà il documento originale e questo gli indicherà l'autenticità della firma.

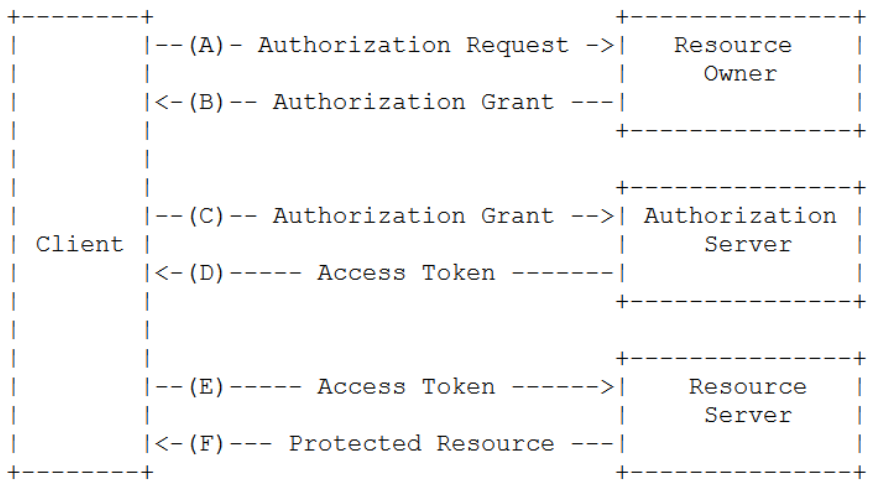
Per il nostro sistema di autenticazione faremo uso della seconda funzione descritta. Più precisamente, quando un utente vorrà registrare un dispositivo, dovrà generare una coppia di chiavi pubblica e privata e condividere con il nostro server la prima. Quando, poi, sarà necessaria l'approvazione di un'operazione da parte del dispositivo, il server gli invierà un codice specifico e si aspetterà di riceverlo firmato, ovvero criptato, con la chiave privata. Per verificarlo dovrà decrittare quanto ricevuto con la chiave pubblica del dispositivo e confrontare ciò che otterrà col codice che aveva inviato.

OAuth2.0

OAuth2.0 è un protocollo utilizzato per fornire a un software una delega per l'accesso a una risorsa offerta da un altro software. Questo sistema è più sicuro della distribuzione diretta delle proprie credenziali, perché il codice di delega, chiamato access token, può avere una scadenza e può fornire l'accesso a una porzione limitata delle risorse.
Il funzionamento di questo protocollo si basa su quattro attori:
- il resource owner, ovvero la persona proprietaria delle risorse
- il client, ovvero il software che ha bisogno della delega
- l'authorization server, ovvero il server che può autenticare il resource owner
- il resource server, ovvero il server che offre le risorse

Nel nostro caso i componenti sono così mappati:
- resource owner: utente
- client: applicazione installata sul dispositivo
- authorization server: server di Athesys o altro server che si occupa dell'autenticazione
- resource server: server Node.js da sviluppare

Il funzionamento ad alto livello di questo protocollo è descritto nell'immagine seguente.



Il procedimento per l'accesso a una risorsa da parte del client segue quindi i seguenti passi:
- il client ottiene dal resource owner un'autorizzazione ad accedere alle risorse di cui ha bisogno (authorization grant). Questa autorizzazione può essere ottenuta in diversi modi ma quello preferibile è attraverso il reindirizzamento verso l'interfaccia web del servizio di autenticazione, la quale, dopo aver verificato le credenziali inserite dall'utente, restituirà l'autorizzazione richiesta
- il client invia all'authorization server l'autorizzazione ottenuta e ottiene un access token
- il client invia al resource server l'access token
- il resource server deve verificare l'access token prima di permettere l'accesso alle risorse. Per questo invia l'access token all'authorization server e richiede che venga verificato
- se l'access token è valido e contiene i permessi necessari per ottenere ciò che il client richiede, il resource server fornisce le risorse

I passi descritti possono subire delle variazioni a seconda del caso d'uso. Per esempio in alcuni casi il client non ottiene un'autorizzazione ma direttamente l'access token. Nel nostro caso tuttavia i passi descritti sono quelli più adatti.
Il microservizio che si vuole sviluppare è pensato per essere un prodotto commercializzabile. I clienti interessati al suo utilizzo potrebbero affidarsi già a un sistema di autenticazione proprio o di terze parti. Per permettere anche a questi utenti di utilizzare il nostro servizio, esso è configurabile e permette di impostare come authorization server un server qualsiasi.
Un client che voglia comunicare con il server Node quindi deve prima conoscere l'authorization server a cui riferirsi. Per questo il server Node deve esporre una API che non necessita di protezione con OAuth2.0 e che fornisce i metadati dell'authorization server.

OAuth2.0 è chiaramente definito nella specifica RFC6749.

Node.js

Node.js è un runtime open-source che permette l'esecuzione di applicazioni Javascript al di fuori di un browser. Grazie alla sua natura asincrona ed orientata agli eventi, questo runtime è particolarmente adatto per lo sviluppo di applicazioni che comunicano in rete, come per esempio un server. La sua architettura lo rende inoltre adatto per software scalabili. Node.js viene distribuito insieme al suo package manager NPM.

La documentazione di Node.js è disponibile a questa pagina.

Express

Express è il framework più utilizzato per sviluppare applicazioni web in Node.js. Le funzionalità principali che offre sono:
- definizione di route handlers, ovvero funzioni che gestiscono le chiamate ad uno specifico URL e con uno specifico tipo (GET, POST, ecc.)
- definizione e utilizzo di middleware, ovvero funzioni che eseguono operazioni intermedie sulle chiamate, per esempio per verificarne l'autenticazione o per gestire i cookie
- integrazione con software di rendering della view per la compilazione di template della pagina

Il team di sviluppo di Express fornisce, tramite NPM, anche una serie di middleware già pronti per alcune funzionalità comuni, come la gestione dei cookie e l'esposizione di file statici.

La documentazione di Express è fornita a questa pagina.

Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) è un servizio cloud per l'invio di notifiche da un server a un client e, purché si utilizzi il protocollo XMPP, anche viceversa.
Le notifiche da parte del server possono essere indirizzare in tre modi diversi:
- verso un singolo dispositivo o un insieme di dispositivi definito solo per quella notifica
- verso un gruppo di dispositivi definito in modo persistente
- verso tutti i dispositi registrati ad un topic

Gli ultimi due metodi si differenziano in quanto nel primo il gruppo viene definito, modificato ed eliminato da parte del server, mentre nel secondo il server definisce i topic ma ogni dispositivo può registrarvisi autonomamente. Il primo metodo è infatti spesso utilizzato per inviare notifiche a tutti i dispositivi di un utente.
Le componenti necessarie ad instaurare un canale di invio di notifiche dal server ai client sono tre:
- un server che genera i messaggi e ne richiede l'invio tramite l'utilizzo diretto del protocollo di comunicazione scelto oppure del Firebase Admin SDK. La documentazione di Firebase consiglia il secondo approccio, che presenta implementazioni per diversi linguaggi tra cui Node.js
- il backend di FCM che riceve le notifiche dal server del punto precedente e le inoltra ai client
- uno o più client che ricevono le notifiche dal backend tramite l'integrazione del SDK di Firebase. Questo SDK è offerto per diverse piattaforme tra cui Android.

I componenti per l'invio di una notifica da un client al server sono gli stessi.
FCM fornisce un'interfaccia web, chiamata Firebase Console, per gestire i progetti che utilizzano i suoi servizi di notifica. Tramite la Console è anche possibile inviare delle notifiche di test per verificare il corretto funzionamento dei client.

Firebase Cloud Messaging è fornito gratuitamente da Google ed è accompagnato da una chiara documentazione.

MongoDB

MongoDB è un database NoSQL orientato ai documenti. La particolarità dei database document-oriented è il fatto che i dati non sono salvati in righe di tabelle strutturate, ma come oggetti, chiamati documenti, di un insieme, chiamato collezione in MongoDB. I documenti di una stessa collezione sono pensati come oggetti diversi di una stessa classe, in riferimento alla programmazione orientata agli oggetti. Quindi non necessariamente dovranno presentare la stessa struttura, allo stesso modo in cui più oggetti di una stessa classe non avranno necessariamente tutti o gli stessi attributi, per esempio grazie al polimorfismo. Questo sistema rende l'interazione tra il programma e il database più diretta e flessibile perché non necessita di uno strato di trasformazione tra la rappresentazione ad oggetti e quella relazionale.
Tra le caratteristiche di MongoDB ci sono anche la facilità nello scalare il database orizzontalmente attraverso lo sharding, ovvero la suddivisione di partizioni dei dati tra server diversi, e nel mantenerne delle repliche su più nodi.

MongoDB fornisce la propria documentazione a questa pagina.

Redis

Redis è un database NoSQL, a coppie chiave-valore, basato sul salvataggio dei dati nella memoria principale (in-memory) invece che nel disco fisso. Questo lo rende più veloce di un database tradizionale, come MySQL o MongoDB, sia nella scrittura che nella lettura.
Redis offre anche la possibilità di salvare i dati su disco fisso in modo da poterli recuperare in caso di arresto inatteso della macchina. Il modo in cui viene generata questa persistenza è abbastanza efficiente da non causare un collo di bottiglia nell'uso normale del database.
La velocità nel lavorare su dati volatili rende Redis un database molto utile per esempio come cache.
Questo database permette di salvare, per ogni chiave, valori in molti formati oltre alla semplice stringa. Tra questi formati ci sono le liste, gli insiemi e gli hash. Quest'ultimo è un'ulteriore mappa chiave-valore, particolarmente utile per salvare oggetti in formato JSON.
Un'ulteriore funzionalità, che risulta interessante per il nostro progetto, è la possibilità di assegnare un tempo di vita ad una chiave del database, superato il quale la chiave e il rispettivo valore vengono eliminati. Redis, e in particolare questa funzionalità, verranno utilizzati per gestire i dati temporanei legati all'approvazione di un'operazione.

La documentazione di Redis è disponibile a questa pagina.

Docker

Docker è un software per la gestione e l'esecuzione di container. Il container è un contenitore che racchiude una o più applicazioni con tutte le loro dipendenze. L'esecuzione dei container, tramite il Docker Engine, può avvenire su qualsiasi sistema operativo. Questo permette una separazione totale tra il software da una parte e il sistema operativo e l'hardware dall'altra. E' pertanto una tecnologia ampiamente utilizzata nell'ambito cloud. Prima della diffusione dei container, per la virtualizzazione venivano utilizzate le macchine virtuali. La differenza con i container sta nel fatto che ogni macchina virtuale contiene anche un sistema operativo, il che le rende più complesse, ingombranti e lente

nell’esecuzione.
La piattaforma Docker fornisce due interfacce con cui è possibile gestire i container, una visuale ma semplice e una a riga di comando più ricca di funzionalità.
Il sistema Docker presenta anche un repository di container chiamato Docker Hub. Chiunque può scaricare i container pubblicati e può caricare i propri.
Alcune dipendenze sono comuni tra molti software, per esempio spesso si ha bisogno di alcune librerie di Linux o di Windows oppure è necessario un server come Apache che esegua l’applicazione da containerizzare. In questi casi è utile creare un container a partire da un altro container esistente e presente su Docker Hub, con le funzionalità di base di cui si necessita.

La documentazione di Docker è fornita a questa pagina.