Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

TITOLO1
TITOLO2
TITOLO3

Relatore:
Prof. Ozalp Babaoglu

Presentata da:
Angelo Galavotti

Correlatore:
Prof. Francesco Giacomini

Sessione II
Anno Accademico 2021/2022

Dedica qui,
Dedica qui 2.

Sommario

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent sit amet lacus nulla. Proin quis nibh vel risus rutrum cursus quis ac dui. Integer nec mollis diam. Sed eget nisl sit amet sem maximus faucibus vel a sem. In hac habitasse platea dictumst. Vivamus a commodo ante. Aliquam erat volutpat. Suspendisse eu porttitor sem, nec porta mi. Etiam et risus euismod, gravida justo nec, pulvinar nulla. Pellentesque finibus tristique mauris, quis molestie nibh consequat nec.

Praesent pretium finibus turpis, sit amet sodales sem facilisis non. Nullam nisi eros, convallis in diam in, faucibus maximus justo. Mauris eu auctor erat, et tristique libero. Praesent pretium gravida magna, a faucibus neque posuere sit amet. Etiam tellus lacus, viverra sit amet dignissim in, eleifend vel diam. In hac habitasse platea dictumst. Nam ornare est nunc, ac interdum turpis rhoncus vitae. Aenean id neque diam. Nunc ac accumsan lectus. Ut vehicula luctus risus eget dignissim. Quisque sed justo non erat vestibulum consectetur vel a diam.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

[cos'è la sicurezza? cos'è il controllo degli accessi?]

1.1 Autorizzazione e autenticazione

[aggiungeer roba sulla storia?] I concetti di *"autorizzazione"* e *"autenticazione"* rappresentano due nuclei fondamentali nell'ambito della sicurezza informatica. Con *autorizzazione* si intende la funzione che specifica i privilegi di accesso a determinate risorse o servizi. L'*autenticazione*, invece, rappresenta l'atto di confermare la verità dell'identità di un utente. Essenzialmente, il prcoesso di autorizzazione fa uso dei dati provenienti dal processo di autenticazione, e in base a questi stabilisce i permessi che un utente possiede. [cos'è un IAM? come si collega con il progetto? chi è stato?] [com'è sviluppato attualmente un tipico sistema di autorizzazione?]

1.2 Contesto dell'operato

[Migliorare coesione dei concetti] Open Policy Agent è un motore di policy open-source, che si occupa principalmente di definire e imporre le politiche di autorizzazione attraverso lo stack una applicazione. Grazie a questo sistema, il processo di decision making nella fase di autorizzazione viene spostato

dall'applicazione ad una compotente esterna.
Le politiche vengono speicificate attraverso un linguaggio dichiarativo di alto livello chiamato *Rego*. [cos'è l'INFN?] [come mai questo progetto? tipi di autenticazione?] [obiettivo della tesi?]

1.3 Struttura del documento

In questa tesi verrà esposta una leggera introduzione su alcuni concetti chiave sul contesto e sulle tecnologie in uso. Dopodiché, sarà mostrerà l'implementazione vera e propria dell'intera infrastruttura, aumentando il numero di feautures andando avanti con i capitoli. Infine, verrà illustrata un'analisi delle performance e verranno fatte le ultime conclusioni.

Capitolo 2

Linee guida architetturali

Durante la progettazione di un sistema di autorizzazione, è necessario considerare alcuni parametri fondamentali che ci permettono di valutarlo nel contesto in cui ci si trova.
Quelli che hanno influito maggiormente sulle scelte progettuali sono stati:

- la versatilità, cioé la capacità di interfacciarsi con sistemi di funzionalità e struttura diversa.
- la manutenibilità, ovvero la facilità di apportare modifiche a sistema realizzato.

La prima è garantita attraverso la divisione in più componenti, così che modificare le regole d'accesso non richiede un'alterazione del codice del complesso. La seconda è ottenuta tramite la separabilità totale dal servizio che si intende proteggere, facilitandone la compatibilità con altre tecnologie.

2.1 Componenti principali

L'architettura dell'infrastruttura presenta uno scheletro formato da tre componenti di principali:

- il *reverse proxy*

- il *sistema di politiche*

- il *service server*

Ciascuna di esse veste un ruolo importante nella gestione della richiesta, tuttavia solo le prime due influenzano il processo di autorizzazione.

2.1.1 Reverse proxy

Il reverse proxy rappresenta l'entry-point dell'intero sistema di autorizzazione. Il compito di questo componente è di mediare la comunicazione tra client e service server: ogni richiesta che viene fatta da un utente è obbligata ad attraversare il reverse proxy prima di giungere al server. L'attraversamento del proxy è trasparente: al client apparirà di comunicare direttamente con il server, nonostante le richieste siano dirette al proxy e da esso propriamente gestite.
La mediazione fornita da questo elemento incorpora un aspetto decisionale, attraverso il quale viene effettivamente espresso il concetto di autorizzazione. Infatti, il reverse proxy inoltra o ignora le richieste al service server, basandosi sull'output del sistema di politiche.

2.1.2 Sistema di politiche

Il sistema di politiche è la parte che determina come la richiesta dovrà essere gestita.
Per adempiere al suo ruolo, sfrutta le informazioni di identità dell'utente contenute nella richiesta, che devono essere propriamente formattate per essere comprese correttamente. Dopodiché, le confronta con dati quali politiche e permessi, ai quali ha accesso diretto. Infine, restituisce in output una decisione, che stabilisce se l'accesso è consentito.

2.1.3 Service server

Con service server si intende il servizio che sfrutta il sistema di autorizzazione. Come tale, il suo ciclo d'esecuzione consiste nel gestire le richieste provenienti dagli utenti tramite il reverse proxy e formulare una risposta. Questa viene indirezionata al mittente iniziale, senza che sia filtrata o analizzata, ma attraversando comunque il proxy.

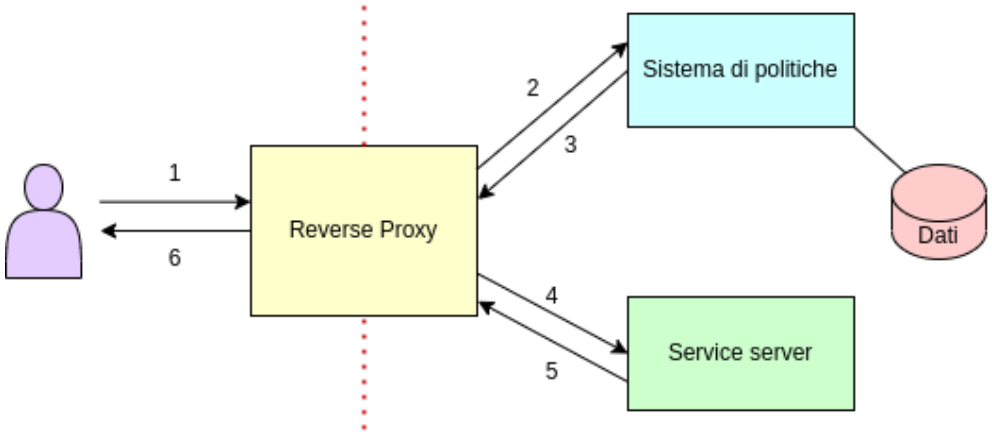

Figura 2.1: Rappresentazione concettuale del sistema

2.2 Topologia finale

In base a quanto è stato esposto nei paragrafi precendenti, si ottiene una struttura topologica simile a quanto mostrato nella Figura 2.1. I numeri su ogni freccia dei flussi dati indicano la sequenza temporale delle trasmissioni di dati tra gli elementi.
Ciascun componente è interpretabile come un processo separato in esecuzione sullo stesso server o su server diversi.

Capitolo 3

Implementazione

La fase di implementazione delle basi del sistema rimane sicuramente la parte più corposa dello sviluppo del progetto.
In questo capitolo verranno discussi in dettaglio gli strumenti e le tecniche che hanno permesso la realizzazione dell'infrastruttura. Particolare attenzione sarà posta nella descrizione nella descrizione della parte operativa, attraverso l'illustrazione di frammenti di codice particolarmente significativi.

3.1 Software stack

Con *software stack* si intende l'insieme di sottosistemi software che permettono di realizzare una piattaforma completa. Questi lavorano assieme per consentire la corretta esecuzione di un'applicazione oppure, come nel caso in questione, il funzionamento di un sistema. Lo stack dell'infrastruttura è composto dai seguenti moduli:

- OpenPolicyAgent
- NGINX
- Docker

che saranno descritti nei paragrafi successivi.
Ogni componente dell'infrastruttura sfrutta lo stack in modo diverso per svolgere il proprio compito.

3.1.1 OpenPolicyAgent

OpenPolicyAgent (abbrev. in OPA) è un software open source che permette di separare il processo di decision-making durante l'autorizzazione dall'applicazione rigorosa delle politiche.
Consideriamo un servizio che richiede di stabilire se un utente possiede dei determinati diritti d'accesso. OPA permette di gestire il problema decisionale analizzando le informazioni su di esso, ricevute attraverso un'interrogazione da parte del servizio. Il carico computazionale della risoluzione del problema viene dunque spostato ad un altro componente, che potrà essere interrogato da più agenti, senza la necessità di codice ridondante.

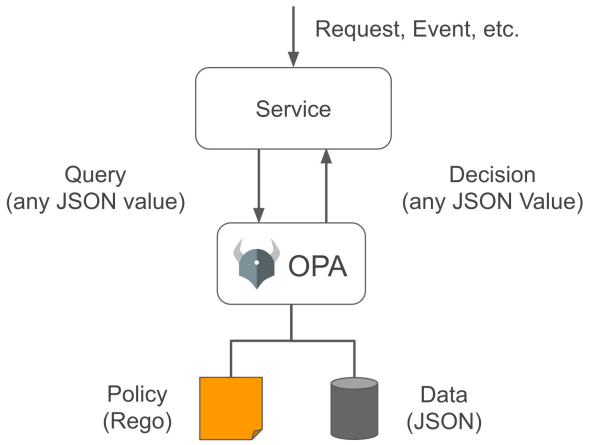

Figura 3.1: Meccanismo di operazione di OpenPolicyAgent

La Figura 3.1 mostra in modo schematico la gestione tipica di una richiesta: un servizio invia un'interrogazione e OPA valuta i dati a sua disposizione per formulare una decisione e restituirla. Le informazioni della richiesta sono definiti in formato JSON, tuttavia sono strutturati in modo totalmente arbitrario.

Le politiche devono essere rappresentate attraverso il linguaggio dichiarativo *Rego*, la cui sintassi si ispira al linguaggio di interrogazione *Datalog*. Attraverso Rego, le regole si esprimono come delle asserzione sui dati mantenuti da OPA. Il codice dedicato alle politiche diventa così molto conciso rispetto ad un loro equivalente in linguaggio imperativo.
Un'altra feature molto interessante di questo software sta nella funzionalità dell'API della pol. In questo modo, una politica può essere modificata o inserita in qualsiasi momento, tramite una richiesta all'API. Solamente chi ha accesso alla rete interna può usufruire di questa funzione.
OpenPolicyAgent è il software che gestisce interamente il sistema di politiche. Oltra a formulare una decisione, infatti, coordina la ricezione delle informazioni e la trasmissione del risultato.

3.1.2 NGINX

NGINX è un web server che incorpora numerose funzionalità, che facilitano l'impostazione del reverse proxy e del service server. È conosciuto per la sua stabilità, semplicità di configurazione, estensibilità e consumo di risorse molto basso, infatti:

- Contrariamente ai server tradizionali, non usa moltecipli thread per gestire le richieste, ma si limita ad un'architettura asincrona a eventi. In questo modo, riesce a gestire un numero alto di connessioni senza che vi siano enormi ripercussioni sulle prestazioni.

- Configurare il software per ottenere il comportamento desiderato diventa un processo relativamente semplice e intuitivo, e si limita alla creazione di un file con estensione ".config" con le clausole necessarie.

- NGINX permette di estendere le proprie funzionalità tramite la creazione di moduli esterni attraverso il linguaggio *NJS*, che consiste in una versione di JavaScript modificata.

Per queste ragione Nginx è stato preferito ad altri sowtware similari.

3.1.3 Docker

Docker è una piattaforma che sfrutta la virtualizzazione a livello del sistema operativo per incorporare software in unità isolate chiamate *container*, includendo anche le sue dipendenze. L'obiettivo di un container è quello di facilitare la distribuzione di un applicativo software, permettendone l'esecuzione a prescindere dal sistema operativo in uso dall'utente.
Docker fornisce lo strumento *Compose*, che permette di definire delle applicazioni singole a più container, che comunicano fra loro su una rete privata. In questo modo è possibile creare facilmente un ambiente di sviluppo avente un funzionamento e struttura analoghe al sistema finale. Le caratteristiche dei container e i collegamenti fra questi vengono definite tramite un file di configurazione in YAML.

3.2 Implementazione delle componenti

L'intera struttura del sistema è definita attraverso il file di configurazione di Docker Compose. Ciò implica che ogni elemento è stato virtualizzato attraverso un container, che svolge il suo compito come un processo separato. La topologia delle relazioni rispecchia quanto mostrato nella Figura 2.1.

3.2.1 Sistema di politiche

Il sistema di autorizzazione usa un approccio *Role-Based Access Control* (abbrev. in RBAC), in cui il controllo degli accessi si basa sui diritti associati al ruolo dell'utente. Altre tecniche di controllo degli accessi possono essere implementate cambiando le politiche. Nonostante ciò, questo approccio risulta quello più consono alla dimostrazione delle capacità di OpenPolicyAgent. La descrizione dei ruoli e dei permessi associati ad essi è mantenuta in file in formato JSON, descritti come nel Codice 3.1. Ad ogni ruolo sono associate le operazioni che l'utente può svolgere.

```
1     "roles" : {
```

```
2         "/dev" : [
3             "retrieve",
4             "submit"
5         ],
6         "/analyst" : [ "retrieve" ],
7         "/admin" : [
8             "retrieve",
9             "submit",
10            "report",
11            "image_request"
12        ],
13        "/moderator" : [ "report" ],
14        "/banned" : []
15    }
```

Codice 3.1: Descrizione dei ruoli

Come già citato in precedenza, le politiche in OpenPolicyAgent sono definite attraverso il linguaggio Rego. L'unità base di una policy scritta in questo linguaggio è data dalla regola, che viene definita tramite una testa e un corpo contenuto in delle parentesi graffe.

```
1     import input.http_method as http_method
2     default allow = false #abilita il default deny
3
4     allow {
5         check_permission
6     }
7
8     check_permission {
9         all_permissions := data.roles[input.role][_]
10        all_permissions == input.operation
11    }
```

Codice 3.2: Policy in linguaggio Rego

Il corpo di una regola contiene delle assegnazioni, operazioni logiche, oppure riferimenti ad altre regole. Una regola assume valore "true" solo se tutte le clausole interne sono eseguibili e risultano vere, altrimenti le viene assegnata

il valore "undefined". Regole aventi stessa testa ma corpo diverso presentano come valore finale il risultato dato dall'OR logico fra i risultati di ciascuna di queste, con "undefined" che assume la stessa semantica di "false". Le regole possono essere annidate facendo sì che il significato di una regola dipenda dal valore di un'altre, in modo da formare una politica strutturata in un'organizzazione gerarchia.
La semplice politica mostrata nel Codice 3.2 è sufficiente ad implementare un controllo degli accessi basato su RBAC. La riga 2 permette di simulare un comportamento "default deny", facendo in modo che il valore di `allow` sia uguale a "false" nella situazione in cui le regole con tale testa siano "undefined".
I confronti con collezioni di dati sono esprimibili in maniera concisa, come avviene nelle righe 9 e 10 del Codice 3.2. La variabile `all_permission` contiene tutti i permessi del ruolo specificato nella query in input. Se l'operazione svolta dall'utente risulta in questa collezione, allora il valore di `check_permission` sarà "true".
La risposta restituita alla componente che ha interrogato OPA consiste in un file in formato JSON contenente il valore delle regole alle radici dell'albero gerarchio, oppure di tutte le regole che hanno portato le radici ad avere un valore undefined.

3.2.2 Reverse proxy

La scelta di inserire NGINX nel software stack è stata decisamente influenzata dagli strumenti che possiede per la realizzazione del reverse proxy. Tra questi, NGINX fornisce la clausola `auth_request` per fare appello a un sistema di decision-making in modo da valutare i diritti d'accesso.

```
1     # operazione protetta
2     location /operation {
3         # appello al server di autorizzazione
4         auth_request /authz;
5         proxy_pass http://service_server_web;
6     }
```

```
7
8     # verifica l'autorizzazione
9     location /authz {
10        internal;
11        proxy_set_header Authorization $http_authorization;
12        proxy_set_header Content-Length "";
13
14        # passiamo gli header settati
15        proxy_pass_request_headers on;
16
17        # sposta la gestione al modulo NJS
18        js_content auth_engine.authorize_operation;
19    }
20
21    # locazione usata per l'invio dei dati a OPA
22    location /_opa {
23        internal;
24        proxy_pass_request_headers on;
25        proxy_pass http://opa:8181/;
26    }
```

Codice 3.3: Frammento di codice del reverse proxy

Una qualsiasi richiesta che accede ad un endpoint avente prefisso `/operation` induce il reverse proxy a controllare se l'utente possiede effettivamente l'autorizzazione ad accedere a tale servizio. Possiamo interpretare la locazione `/operation` come quella associata alle risorse protette, che richiedono un controllo dell'accesso.

Attraverso il comando `auth_request /authz`, il flusso d'esecuzione passa immediatamente al codice associato all'endpoint `/authz`. In questa fase, la richiesta HTTP viene condotta nel modulo NJS dedicato alla creazione della query e alla trasmissione di essa verso il sistema di politiche. La riga 18 del Codice 3.3 invoca questa funzionalità.

Il modulo NJS contiene il metodo `authorize_operation`, che raccoglie le informazioni dagli header e le trascrive in formato JSON. I dati sono quindi inseriti in un pacchetto HTTP, che verrà inviato all'endpoint `/_opa`, associato

alla comunicazione con il sistema di politiche.

```
1   function authorize_operation(r) {
2
3       // dati da inviare a OPA
4       let opa_data = {
5           "operation" : r.headersIn["X-Operation"],
6           "role" : "/" + r.headersIn["X-Role"]
7       }
8
9       // pacchetto HTTP da inviare ad OPA
10      var opts = {
11          method: "POST",
12          body: JSON.stringify(opa_data)
13      };
14
15      // gestisce la risposta di OPA
16      r.subrequest("/_opa", opts, function(opa_res) {
17
18          var body = JSON.parse(opa_res.responseText);
19
20          // se body non c'e' o allow e'
21          // uguale a false, ritorna forbidden (403)
22          if (!body || !body.allow) {
23              r.return(403);
24              return;
25          }
26
27          // altrimenti, ritorna il codice dato da OPA
28          // (che e' 200 in caso non vi siano errori)
29          r.return(opa_res.status);
30      });
31  }
```

Codice 3.4: Frammento di codice del modulo NJS

In base alla risposta del sistema di politche, il reverse proxy inoltrerà il pacchetto originale al service server con la keyword `proxy_pass`, oppure terminerà la gestione della richiesta. La parola chiave `internal` alle righe 10

e 23 del Codice 3.3 garantisce che le locazioni associate all'interrogazione di OpenPolicyAgent possano essere accedute esclusivamente da sottorichieste interne al reverse proxy.
L'uso di un modulo esterno è necessario siccome la clausola `auth_request` scarta il campo body della richiesta HTTP. Ciò implica che non è possibile inviare una richiesta contenente i dati della query nel suo corpo. Dunque, è essenziale l'aggiunta di un passo aggiuntivo che converta gli header HTTP in dati in formato JSON comprensibili da OPA.

3.2.3 Service server

Come descritto nella Sezione 2.1.3, con service server si intende un servizio qualsiasi che sfrutta l'infrastruttura di autorizzazione. Siccome questa fase del progetto si focalizza sullo sviluppo corretto delle componenti dedicate al filtraggio delle richieste, il service server è dato da una API relativamente elementare.
L'API in questione ritorna una risposta generica con Status Code 200 se si accede correttamente all'endpoint `/operation`, mentre una richiesta a `/operation/image_request` permette lo scaricamento di un'immagine. Siccome queste locazioni presentano il prefisso "operation", il loro accesso comporta l'uso del sistema di autorizzazione.

La modalità web server di NGINX permette di implementare quanto appena descritto.

```
1     # invia 200 se si esegue l'operazione con successo
2     location /operation {
3         return 200;
4     }
5
6     # manda un'immagine al client
7     location /operation/image_request {
8         try_files $uri /test.jpg;
9     }
```

Codice 3.5: Frammento di codice del service server

La cluasola `return` invia al client uno specifico HTTP Status Code, Questa scelta ha facilitato lo sviluppo, permettendo di definire in poche righe di codice una API funzionante e adatta ai requisiti, come si può vedere nel Codice 3.5.

3.3 Cifratura SSL/TLS

Spesso un sistema di autorizzaione richiede delle informazioni sensibili riguardo un utente per attuare il controllo degli accessi. Per proteggere il flusso dati da possibili intercettazioni è opportuno che la comunicazione fra il servizio e l'utente venga cifrata.
Il protocollo più popolare che mette in sicurezza un collegamento
è *SSL/TLS*, che si basa sulla cifratura a chiave asimmettrica. Il suo funzionamento necessita dei certificati X.509, e in questo modo garantiscono che la chiave pubblica di un host appartenga effettivamente ad esso.
Nel caso del progetto di questa tesi, solamente la connesione fra il reverse proxy e l'utente deve essere protetta, siccome nessun agente esterno all'infrastruttura possiede accesso diretto alle componenti. NGINX supporta le connessioni SSL/TLS attraverso alcune configurazioni esplicite.

```
1    server {
2        listen 443 ssl;
3        server_name servicecnaf.test.example;
4
5        ssl_certificate     certs/star.test.example.cert.pem;
6        ssl_certificate_key certs/star.test.example.key.pem;
7        ...
```

Codice 3.6: Configurazione di SSL

L'impostazione del canale protetto attraverso NGINX avviene inserendo il parametro `ssl` nella definizione dei socket in ascolto del server. Il percorso del certificato pubblico e della chiave privata devono essere specificati rispettivamente nelle direttive `ssl_certificate` e `ssl_certificate_key`.
Il Codice 3.6 mostra il setup della protezione SSL/TLS all'interno del reverse

proxy del sistema di autorizzazione. Allo stato attuale del progetto, i certificati sono dedicati esclusivamente al testing, quindi non sono validi. Va da sé che la sostituzione dei certificati stessi con una controparte ufficialmente verificata è un processo banale.

3.4 Testing

L'intera infrastruttura prende vita avviando i container di Docker Compose attraverso il comando `docker-compose up`.
Per testare il sistema, è possibile inviare dei pacchetti HTTP al reverse proxy contenti nel proprio header le informazioni necessarie per formulare una query.
cURL (o curl) è il software principale usanto in ambito di testing, siccome permette di inviare dati attraverso numerosi protocolli, tra cui anche HTTP.
Questo metodologia è stata applicata in continuità durante lo sviluppo dell'intero progetto.

Capitolo 4

Gestione dell'autenticazione

Una delle motivazioni principale che ha portato alla creazione del progetto descritto in questa tesi sta nella creazione un sistema che supportasse un nuova tecnica di autenticazione e che fosse retrocompatibile con il metodo usato in precedenza dall'INFN CNAF. Queste due tecniche consistono rispettivamente nell'autenticazione basata su token e su certificati.
In questo capitolo si analizzeranno i due metodi diversi di autenticazione e si implementerà la gestione dei dati provenienti da essi.

4.1 JSON Web Token

I *JSON Web Token* (abbrev. in JWT) rappresentano uno standard per la definizione di un metodo compatto e sicuro per la trasmissione di dati in formato JSON. La sicurezza di questo metodo di scambio delle informazioni deriva dal fatto che i token possono essere firmati digitalmente, assicurando così integrità e confidenzialità dei dati.
Nella loro forma più diffusa, i JWT sono costituiti, in successione, da tre parti:

- L'*header*, che contiene le informazioni riguardo il tipo di token (ovvero JWT) e l'algoritmo usato per la firma dei dati.

- Il *payload*, dove risiedono le informazioni che si vogliono scambiare. Tipicamente, una parte di questo campo è è dedicata a delle dichiarazioni predefinite riguardo il client e la validità del token.

- La *signature*, che si ottiene firmando l'header e il payload con la chiave pubblica del destinatario.

Queste tre campi sono separati da dei punti, quindi un tipico JWT è definito nella forma:

`xxxxx.yyyyy.zzzzz`

L'uso più comune dei JSON Web Tokens consiste nel loro impiego come metodo di autenticazione. Generalmente, dopo che un utente si autentica a un servizio tramite le proprie credenziali, riceve un token. Questo assicura l'utente possiede dell'identità dell'utente, e definisce [da cambiare]. In questo modo, ogni operazione successiva all'autenticazione che richiede un controllo dell'accesso e segue l'autenticazione invierà una richiesta che contiene tale token, senza che sia necessario - OpenPolicyAgent supporta delle funzioni built-in per la decodifica e la verifica dei JWT. con le regole usate per l'implementazione del RBAC nel Codice 3.2. [Cos'è un JWT? Com'è fatto?] [Quali sono le modifiche fatte al sistema?]

4.2 Certificati VOMS Proxy

Nell'ambito del calcolo distribuito, le organizzazioni virtuali (abbrev. VO) rappresentano una collezione di gruppi di invidui sparsi nel mondo che definite da una serie di regole riguardo la condivisione delle risorse.
Il Virtual Organization Membership Service (abbrev. VOMS), è un servizio per il controllo degli accessi nei servizi distribuiti, che implementa un database per ogni organizzazione virtuale (di fatto per ogni esperimento) all'interno del quale sono mantenuti i dati relativi ad ogni singolo membro/utente

Capitolo 5

Interfacciamento con Storm-Tape API

Durante le fasi di sviluppo iniziali, il servizio collegato al sistema di autenticazione era dato da una API elementare. Quest'ultima non presentava delle funzionalità significative: la sua creazione era totalmente a sostegno del sistema di autorizzazione, dunque il carattere del servizio fornito e la sua complessità non erano di estrema rilevanza.
Tuttavia, per quanto un'impostazione simile possa essere utile ai fini dell'implementazione del sistema, non permette di valutare correttamente le caratteristiche del Proof-of-Concept.
In questa sezione verrà mostrato il processo di interfacciamento del sistema con un servizio realistico e complesso, analizzando le modifiche necessarie per abilitare un controllo degli accesso nei suoi endpoint.

5.1 Storm-Tape API

STOrage Resource Manager (o StoRM) è un servizio dedicato alla gestione della memoria di massa nei centri di computazione distribuita.
In relazione a questo, *Storm-Tape API* è un progetto sviluppato attivamente all'INFN CNAF, che nella data di scrittura di questa tesi è in fase di Proof-

Funzionalità	Descrizione
`STAGE`	Richiede che i file mantenuti su un determinato siano resi disponibili con latenza di un disco rigido.
`RELEASE`	Indica che i file precedentemente resi disponibili non sono più richiesti.
`ARCHIVEINFO`	Richiede informazioni riguardo il progresso della scrittura dei file su nastro.

Tabella 5.1: Insieme delle operazioni fornite dalla specifica WLCG Tape REST API

of-Concept. [correggimi se sto sbagliando qui giaco] Consiste in un'implementazione della specifica WLCG Tape REST API, che offre un'interfaccia comune per permettere ai client di gestire la residenza dei dati mantenuti negli storage dati dei GRID di tutto il mondo.
Il nome Storm-Tape deriva dal fatto che, nei centri di calcolo distribuiti, le informazioni vengono mantenute in dischi basati su nastro, in quanto permettono di memorizzare grandi moli di dati con un costo relativamente basso.
Il più grande difetto deCome si può intuire, questo servizio è dedicato esclusivamente ai sistemi che fanno uso di memoria basata su nastro. Nei sistemi GRID con memoria di massa basata su nastro, Le funzionalità che fornisce questa API sono illustrate nella Tabella 5.1. [da inserire maggiori info riguardo l'API, es. endpoint]

5.2 Interfacciare il sistema con il servizio

5.3 Risultati

[Perché questo?] [Cos'è e come funziona storm-tape API?] [Quali sono le modifiche necessarie per far funzionare il sistema?] [Che conclusioni hai

tratto?] [È facile cambiare? Come avviene in generale il cambiamento? Cosa se ne trae?]

Capitolo 6

Conclusioni

Nei capitoli precedenti è stato illustrato l'intero processo di realizzazione del sistema, partendo dalla progettazione architetturale e evolvendolo gradualmente tramite l'aggiunta di nuove feature, che includono il supporto a differenti tecnologie di autenticazione e alla comunicazione crittografata.
Il Capitolo 6 ha mostrato la facilità con cui questo sistema riesce ad integrarsi con un servizio già esistente, senza richiedere una riprogrammazione totale delle componenti, ma modificando le policy esistenti e il modulo dedicato alla formattazione dei dati.
Nel presente capitolo verranno esposte alcune proposte di sviluppi futuri emerse durante l'implementazione, assieme a una riflessione sui risultati a progetto realizzato.

6.1 Sviluppi futuri

Il progetto esposto in questa tesi fornisce una buona base per la creazione di un sistema di autorizzazione in cui il carico del processo di decision-making è affidato a una componente esterna.
Nonostante ciò, sono molte le possibilità per cui quanto è stato creato può essere esteso e migliorato.

6.1.1 Modifica delle policy dall'esterno

OpenPolicyAgent è uno strumento molto affascinante e con questo progetto abbiamo appena scalfito la superficie delle possibilità che offre. Uno degli aspetti particolarmente interessanti sarebbe lo sviluppo di un'estensione del sistema per consentire la modifica delle policy connettendosi da una rete esterna all'infrastruttura. Come è stato illustrato nel Capitolo 3, infatti, OPA fornisce un'API che espone degli endpoint CRUD, in modo da permettere la modifica, l'inserimento e l'eliminazione delle regole "on the fly". Un'estensione del genere abiliterebbe un qualsiasi utente privilegiato a modificare le regole con estrema convenienza.
Seguendo questo pricipio, una possibile implementazone potrebbe impiegare l'uso di policy di OPA per applicare un controllo degli accessi riguardo la modifica delle policy stesse, il che rappresenterebbe un caso di studio decisamente curioso.

6.1.2 Migliorie nella gestione dei VOMS Proxy

Nel Capitolo 4 si è trattato di come il sistema gestisce i certificati VOMS Proxy. Come citato, il modulo al momento disponibile per la validazione dei VOMS Proxy necessita dell'uso di OpenResty, ossia una versione modificata da terze parti che include il supporto a estensioni programmate nel linguaggio Lua. Pertanto, non è supportata direttamente dagli sviluppatori di NGINX. [perché è negativo questo?]
Una possibile miglioramento dell'infrastruttura si potrebbe trovare nella reimplementazione dei moduli VOMS, traducendoli in programmi equivalenti scritti in NJS. Infatti, quest'ultimo è il linguaggio supportato ufficialmente per l'estensione di NGINX con funzionalità personalizzate.
In questo modo il reverse proxy risulterebbe totalmente affidato all'implementazione standard NGINX.

6.2 Osservazioni finali

L'obiettivo fondamentale di un Proof-of-Concept consiste nel dimostrare la realizzabilità di un sistema, mi ritengo soddisfatto del risultato finale ottenuto. L'infrastruttura ricalca l'obiettivo posto da questa tesi, realizzando un controllo efficace degli accessi e interfacciandosi con poche modifiche ad altri servizi.
Vorrei infine dedicare queste ultime righe per esporre alcuni momenti importanti della mia esperienza personale nella concretizzazione del progetto.
Ho avuto il piacere di avere un piccolo spazio "mio" in un centro di ricerca, e di ricevere consigli e supporto da persone eccellenti per know-how, che mi hanno fatto sentire "collega".
Grazie a questa tesi, sono entrato in contatto con il mondo della computazione distribuita, che prima mi era totalmente estraneo. È stato molto interessante osservare i meccanismi che coinvolgono l'esecuzione di batch job, e non dimenticherò mai la quantità di rumore prodotta dai pod.
Infine, questa tesi mi ha dato l'opportunità di mettere alla prova quanto avevo imparato durante questi anni di corso e trasformarli in competenze "sul campo": ritengo che questa esperienza sia stata fondamentale per il mio percorso da studente di Informatica.

Bibliografia

[1] Nginx, Inc., *NGINX*, https://www.nginx.com/.

[2] Nginx, Inc., *NGINX Documentation*, https://nginx.org/en/docs/.

[3] Andrea Ceccanti, Francesco Giacomini, Enrico Vianello, *VOMS*, https://italiangrid.github.io/voms/.

[4] IETF, *RFC 7519*, https://www.rfc-editor.org/rfc/rfc7519.

[5] OpenPolicyAgent, *OpenPolicyAgent*, https://www.openpolicyagent.org/.

[6] OpenPolicyAgent, *OpenPolicyAgent Documentation*, https://www.openpolicyagent.org/docs/latest/.

[7] OpenPolicyAgent, *Policy Language*, https://www.openpolicyagent.org/docs/latest/policy-language/.

[8] auth0, *Introduction to JSON Web Tokens*, https://jwt.io/introduction.

[9] NGINX, *njs scripting language*, https://nginx.org/en/docs/njs/.

[da rivedere]

[10] INFN CNAF, *WLCg Tape API*, https://www.openpolicyagent.org/docs/latest/policy-language/

Ringraziamenti

bho non so ancora chi ringraziare onestamente