Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Analisi e implementazione di un sistema di autorizzazione compatibile con token OAuth 2.0 e certificati VOMS

Relatore:
Prof. Ozalp Babaoglu

Presentata da:
Angelo Galavotti

Correlatore:
Prof. Francesco Giacomini

Sessione II
Anno Accademico 2021/2022

Alla migliore madre del mondo,
al miglior padre del mondo.

Sommario

Il Worldwide LHC Computing Grid (WLCG) è una collaborazione internazionale costituita da decine di centri di calcolo distribuiti globalmente, la cui missione consiste nell'elaborazione delle grandi quantità di dati prodotti dai maggiori esperimenti di Fisica delle Alte Energie, in particolare quelli al CERN di Ginevra.

Uno di questi centri di calcolo è ospitato presso il CNAF dell'Istituto Nazionale di Fisica Nucleare a Bologna, che contribuisce anche allo sviluppo di middleware per la gestione dell'infrastruttura.

Molti componenti di tale middleware, che hanno funzionalità e scopi diversi, richiedono un servizio di autorizzazione versatile e compatibile con i meccanismi di autenticazione attualmente in uso, basati su token OAuth 2.0 e su certificati VOMS Proxy.

In questa tesi si analizzerà l'architettura e l'implementazione di un proof-of-concept di un sistema di autorizzazione che soddisfi queste necessità, enfatizzando l'integrazione delle tecniche di autenticazione citate. Per dimostrare la sua versatilità, verrà illustrato il processo di interfacciamento con un componente middleware attualmente in sviluppo al CNAF.

Il risultato finale ottenuto è un sistema che rispetta i vincoli richiesti e si integra facilmente con servizi eterogenei.

Indice

Elenco delle figure

Elenco dei frammenti di codice

Capitolo 1

Introduzione

Autenticazione e *autorizzazione* rappresentano due concetti fondamentali nell'ambito della sicurezza informatica.
Con autenticazione si intende l'atto di confermare l'identità di un utente. L'autorizzazione, invece, indica la funzione che specifica i privilegi di accesso a determinate risorse o servizi.
I due processi sono interamente interconnessi: l'autorizzazione fa uso dei dati provenienti dall'autenticazione e in base a questi stabilisce i permessi che un'entità possiede.
In informatica, l'autorizzazione si basa sulle *politiche di accesso* (o policy), che vengono utilizzate per determinare se una richiesta di accesso a una risorsa da parte di un utente autenticato deve essere approvata o rifiutata.
Il processo del controllo degli accessi è composto da due stadi principali: una fase decisionale, in cui si valutano i permessi dell'utente formulando una decisione in base alle politiche di accesso del sistema, e una fase esecutiva, in cui si applica effettivamente quanto è stato deciso.

1.1 Worldwide LHC Computing Grid

Nell'ambito del calcolo distribuito, il *grid computing* consiste nell'uso di una rete di risorse computazionali distribuite geograficamente che collabora-

no per raggiungere un determinato obiettivo.
Il *Worldwide LHC Computing Grid* (WLCG) [1] è un progetto dato dalla collaborazione delle infrastrutture grid sparse in tutto il mondo, con l'obiettivo di analizzare le grandi quantità di dati generati dal *Large Hadron Collider* (LHC) [2], l'acceleratore di particelle situato nel CERN di Ginevra.

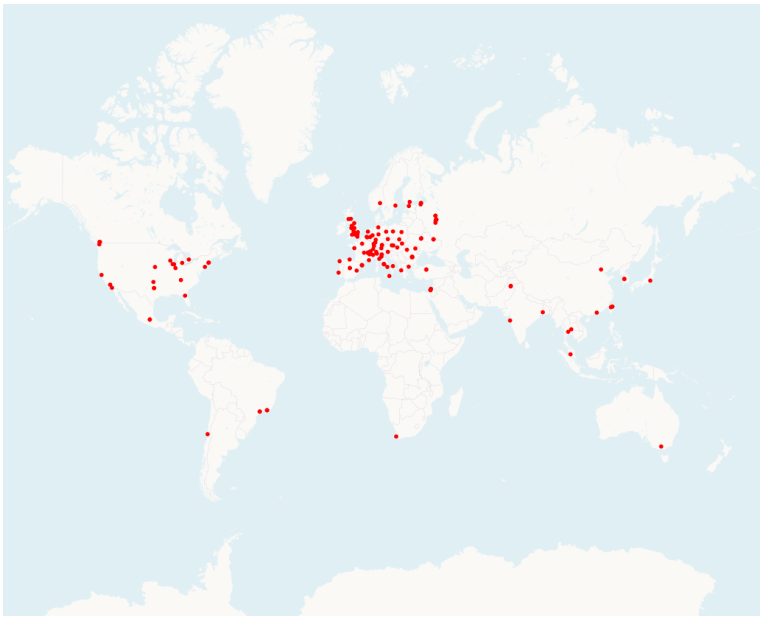

Figura 1.1: Locazione geografica dei centri di calcolo del Worldwide LHC Computing Grid.

Ad oggi, il WLCG incorpora circa 170 centri di calcolo sparsi in 42 paesi. Uno di questi risiede nel centro *CNAF* dell'Istituto Nazionale di Fisica Nucleare (INFN) [3] a Bologna, e rappresenta uno dei principali centri di calcolo di questa associazione. Come tale, il CNAF contribuisce alla gestione e allo sviluppo del middleware associato all'infrastruttura grid del WLCG.

1.2 Scopo del progetto

Molti dei sistemi software sviluppati al CNAF richiedono un sistema di autorizzazione. Siccome si tratta di progetti aventi scopi e funzionalità diverse, implementare il controllo degli accessi direttamente in essi risulterebbe

ridondante e potrebbe portare a problemi di ambiguità se le politiche fossero implementate in modo diverso per ogni componente.
Il progetto sviluppato in questa tesi consiste nell'esplorazione di una soluzione a questo problema, ovvero un proof-of-concept di un sistema di autorizzazione general-purpose e versatile. Siccome si interfaccerà potenzialmente con progetti in relazione al WLCG, il sistema deve essere compatibile con le tecniche di autenticazione adottate dai suoi servizi.
Attualmente, il WLCG si trova in una fase di transizione dall'autenticazione basata su certificati proxy in favore all'uso dei token OAuth 2.0. Dunque, è necessario che il sistema sia compatibile con entrambi i metodi.
Un altro vincolo riguarda l'implementazione: i componenti del sistema devono basarsi su strumenti software di industria prontamente disponibili, in modo da assicurarne la stabilità e l'affidabilità.

1.3 Struttura del documento

Questa tesi si focalizzerà su un'analisi dettagliata del sistema creato, aiutandosi con schemi e frammenti di codice a supporto della spiegazione.
Il Capitolo 2 è dedicato alla descrizione architetturale, analizzandone i componenti.
Il Capitolo 3 mostra e motiva i dettagli implementativi del sistema e i software sul quale si basa, con un accenno alle tecniche di testing.
Il Capitolo 4 spiega le tecniche di autenticazione supportate dall'infrastruttura e come è stata gestita la loro integrazione.
Il Capitolo 5 descrive il processo di interfacciamento con un servizio esistente.
Infine, il Capitolo 6 tratterà dei risultati di quanto è stato creato, proponendo dei possibili miglioramenti e aggiungendo una riflessione personale sul lavoro svolto.

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 contribuito maggiormente sulle scelte progettuali sono stati:

- la versatilità, cioè 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, cosicché modificare le regole di accesso non richieda 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.
Un'altra considerazione che ha influenzato l'architettura del sistema è stata la scelta del software per la gestione della fase decisionale dell'autorizzazione, che richiede una gestione delle due fasi dell'autorizzazione tramite l'uso di componenti separati.

2.1 Componenti principali

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

- il *sistema delle politiche di accesso*
- il *reverse proxy*
- il *servizio applicativo*

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

2.1.1 Sistema delle politiche di accesso

Il sistema delle politiche di accesso 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, applica le regole di accesso e confronta i dati ricevuti con un database dei permessi, al quale ha accesso diretto.
Infine, restituisce in output una decisione, che stabilisce se l'accesso è consentito o meno.

2.1.2 Reverse proxy

Il reverse proxy rappresenta l'unico entry-point dell'intero sistema di autorizzazione. Il compito di questo componente è di mediare la comunicazione tra client e il servizio applicativo: 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 servizio, 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 servizio applicativo, basandosi sull'output del sistema delle politiche di accesso.
Il termine "reverse" permette di distinguerlo dai normali proxy d'inoltro, che interfacciano un client con una rete senza necessariamente esaminare il pacchetto in uscita o in entrata. Un reverse proxy, invece, viene posto nella comunicazione fra un server e i client che cercano di accedervi, applicando un controllo degli accessi.

2.1.3 Servizio applicativo

Con servizio applicativo si intende il servizio che usufruisce del sistema di autorizzazione. Nonostante ciò, l'intero sistema gli è totalmente trasparente e non necessita di modifiche per interfacciarsi con esso.
Il suo ciclo di esecuzione consiste nel gestire le richieste provenienti dagli utenti tramite il reverse proxy e formulare una risposta. Questa viene inoltrata al mittente iniziale, senza che sia filtrata o analizzata, ma attraversando comunque il proxy.

2.2 Topologia finale

In base a quanto è stato esposto nei paragrafi precedenti, 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 delle informazioni tra gli elementi.
Ciascun componente è interpretabile come un processo separato in esecuzione sullo stesso server o su server diversi.

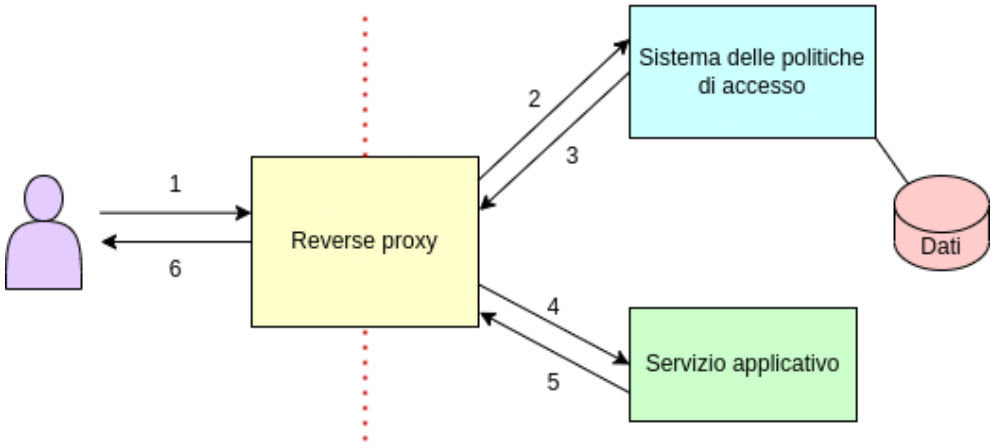

Figura 2.1: Rappresentazione concettuale dell'infrastruttura. La richiesta di un utente viene prima processata dal reverse proxy, che la inoltrerà o meno al servizio applicativo basandosi sulla decisione formulata dal sistema delle politiche di accesso.

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 della parte operativa, attraverso l'illustrazione di frammenti di codice significativi nel contesto dell'implementazione.

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 software:

- 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 (OPA) [8] è un software open source che permette di separare, durante l'autorizzazione, il processo di decision-making dall'applicazione rigorosa delle politiche.
Ad esempio, consideriamo un servizio che richiede di stabilire se un utente possiede dei determinati diritti di accesso. OPA permette di gestire il problema decisionale del controllo dell'accesso analizzando le informazioni su di esso, ricevute attraverso un'interrogazione da parte del servizio. In questo modo, l'applicazione delle politiche di accesso viene dunque delegata a un componente singolo, che potrà essere interrogato da più agenti.
Il principale vantaggio di un'organizzazione di questo tipo consiste nell'eliminazione di eventuali ambiguità causate dall'implementazione del controllo degli accessi in ciascun elemento del sistema. La Figura 3.1 mostra in modo

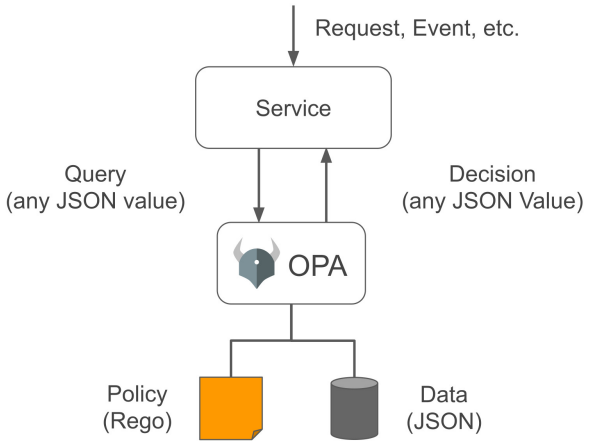

Figura 3.1: Meccanismo di operazione di OpenPolicyAgent. Si analizza la query proveniente da un agente, e in base alle politiche e ai dati a sua disposizione, formula una decisione da inviare al mittente.

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 strutturate in modo totalmente arbitrario.
Le politiche (o policy) devono essere rappresentate attraverso il linguaggio dichiarativo *Rego* [11], 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 a un loro equivalente in linguaggio imperativo.
Un'altra funzionalità molto interessante di questo software sta nella REST API CRUD fornita da OpenPolicyAgent per la gestione delle politiche [24]. In questo modo, una politica può essere modificata o inserita in qualsiasi momento tramite una richiesta all'API. Nel sistema di autorizzazione implementato in questa tesi, il servizio API non è stato abilitato.
OpenPolicyAgent è il software che gestisce interamente il sistema delle politiche di accesso. Oltre a formulare una decisione, infatti, coordina la ricezione delle informazioni e la trasmissione del risultato.

3.1.2 NGINX

NGINX [4] è un web server che incorpora numerose funzionalità, che facilitano l'impostazione del reverse proxy e del servizio applicativo. È conosciuto per la sua stabilità, semplicità di configurazione, estensibilità e consumo di risorse molto basso, infatti:

- Contrariamente ai server tradizionali, non usa molteplici thread per gestire le richieste, ma si limita a un'architettura asincrona a eventi. In questo modo, riesce a gestire un numero alto di connessioni senza inficiare le prestazioni [5].

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

- NGINX permette di estendere le proprie funzionalità tramite la creazione di moduli attraverso il linguaggio *NJS* [12], che consiste in una versione ridotta di JavaScript.

Per queste ragione NGINX è stato preferito ad altri software similari.

3.1.3 Docker

Docker [9] 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* [10], che permette di orchestrare dei servizi singoli a più container, che comunicano fra loro su una rete privata. In questo modo è possibile creare facilmente una configurazione per lo schieramento dei sistemi in fase di produzione, nonché un ambiente di sviluppo avente un funzionamento e struttura analoghi al sistema finale. Le caratteristiche dei container e i collegamenti fra questi vengono definite tramite un file di configurazione in formato YAML.

3.2 Implementazione dei 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 delle politiche di accesso

Il sistema di autorizzazione scelto in questo lavoro usa un approccio *Role-Based Access Control* (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 a 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 e dei relativi permessi in formato JSON.

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 simbolo ":=" indica un assegnamento di valore, mentre "==" permette di confrontare l'uguaglianza fra i valori delle due variabili. L'indice "_" permette di considerare tutti i valori di una dimensione di un vettore.

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'altra, in modo da formare una politica strutturata in un'organizzazione gerarchia.

La semplice politica mostrata nel Codice 3.2 è sufficiente a 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 al componente che ha interrogato OPA consiste in un file in formato JSON contenente il valore delle regole alle radici dell'albero

gerarchico, 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 di 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        ...
12        # passiamo gli header settati
13        proxy_pass_request_headers on;
14
15        # sposta la gestione al modulo NJS
16        js_content auth_engine.authorize_operation;
17    }
```

Codice 3.3: Frammento di codice principale del reverse proxy. La locazione ***`/authz`*** *gestisce e invia la richiesta a OPA, e in base alla sua risposta inoltra la richiesta o meno attraverso la keyword* ***`proxy_pass`****.*

Una qualsiasi richiesta che accede a 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 di 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 delle 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 delle 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 a 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 dedicato all'estrazione dei dati e all'invio di questi al sistema delle politiche di accesso. In questo stesso codice viene inoltre gestita la ricezione della risposta e si valuta la decisione contenuta in essa.

In base alla risposta del sistema di politiche, il reverse proxy inoltrerà il pacchetto originale al servizio applicativo 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 perché 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 livello di indirezione che converta gli header HTTP in dati in formato JSON comprensibili da OPA.

3.2.3 Servizio applicativo

Come descritto nella Sezione 2.1.3, con servizio applicativo si intende un servizio qualsiasi che usufruire in modo trasparente dell'infrastruttura di autorizzazione. Siccome questa fase del progetto si focalizza sullo sviluppo corretto dei componenti dedicati al filtraggio delle richieste, il servizio applicativo 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 servizio applicativo che mostra le funzionalità associate agli endpoint.

La clausola `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 autorizzazione 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 asimmetrica. Il suo funzio-

namento necessita dei certificati X.509 [20], che garantiscono che la chiave pubblica di un host appartenga effettivamente a esso.
Nel caso del progetto di questa tesi, solamente la connessione fra il reverse proxy e l'utente deve essere protetta, siccome nessun agente esterno all'infrastruttura possiede accesso diretto ai 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        ...
8    }
```

Codice 3.6: Configurazione della comunicazione tramite SSL/TLS nel reverse proxy.

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 rilasciati da una Certificate Authority valida. 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) [17] è il software principale usato 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

Uno tra i principali obiettivi che ha portato alla creazione del progetto descritto in questa tesi risiede nella creazione un sistema che supporti l'autorizzazione tramite token OAuth 2.0 e che sia compatibile con il metodo di autenticazione basato su certificati proxy VOMS attualmente in uso. Quest'ultimo consiste nell'uso di certificati che permettono di eseguire delle operazioni facendo le veci di un'altra entità.
In questo capitolo si analizzeranno i due metodi diversi di autenticazione e il processo di implementazione della gestione dei dati provenienti da essi.

4.1 JSON Web Token

I *JSON Web Token* (JWT) [7] 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 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 private del mittente.

Queste tre campi sono separati da dei punti, quindi un tipico JWT si presenta nella seguente forma:

`xxxxx.yyyyy.zzzzz`

L'uso più comune dei JSON Web Tokens consiste nel loro impiego come metodo di autorizzazione.

4.1.1 OAuth 2.0 e OpenID Connect

OAuth 2.0 [19] è un protocollo per l'autorizzazione nelle applicazioni software. Definisce uno standard per la gestione del controllo degli accessi in un qualsiasi servizio HTTP, stabilendo il modo in cui un client possa delegare a un servizio la propria identità attraverso l'uso di un token di accesso, in sostituzione all'uso delle proprie credenziali.
OpenID Connect (OIDC) [16] è un layer di autenticazione costruito su OAuth 2.0, che permette a un'applicazione di verificare l'identità di un utente e di ottenere delle semplici informazioni su di esso tramite un ID Token. Questo protocollo abilita il *Single Sign-On* (SSO) [23], che consiste nella possibilità di autenticare la propria identità su più servizi non necessariamente correlati senza dover immettere le proprie credenziali di volta in volta.
OIDC sfrutta i JSON Web Token, che sono ottenibili in conformità con i metodi specificati da OAuth 2.0. Per questo motivo, spesso si usa il gergo "token OAuth 2.0" per fare riferimento ai JWT usati nel contesto di OIDC. OIDC e OAuth 2.0 di fatto definiscono uno standard per l'accesso basato su token, che rimane un canone per molti servizi su Internet.

4.1.2 Generazione dei tokens

Durante la fase di sviluppo del sistema, i token sono stati generati con il software *OIDC Agent* [18], che fornisce JWT conformi alla specifica di OpenID Connect. Questo software usa un'interfaccia a linea di comando e crea dei token attraverso una opportuna configurazione che stabilisce l'issuer e alcune caratteristiche del token.
Nel contesto del progetto, i token generati sono associati al WLCG e contengono il claim `wlcg.groups` [21], che rappresenta una lista dei gruppi a cui l'utente appartiene.

4.1.3 Integrazione dei JWT

L'integrazione dei JWT come metodo di autenticazione del sistema di autorizzazione viene decisamente semplificata dalla scelta di inserire OpenPolicyAgent nel software stack. Questo software infatti fornisce delle funzioni built-in per la decodifica e la verifica dei JWT.
Un passaggio chiave per abilitare ufficialmente il controllo degli accessi tramite i dati presenti nei token OAuth consiste nella modifica del modulo del reverse proxy dedicato allo scambio dei dati con il sistema delle politiche. In particolare, l'alterazione consiste nell'aggiunta di un campo contenente il token di autorizzazione nella richiesta dedicata a OpenPolicyAgent.

```
1     # controllo dei permessi dell'utente
2     check_permission_jwt {
3         data.roles[payload["wlcg.groups"][_]][_]
4                 == input.operation
5     }
6
7     # validazione del tempo del token
8     check_time_validity {
9         payload.exp <= time.now_ns()
10    } else { # caso di token senza scadenza
11        payload.iss == payload.exp
```

```
12          payload.exp == payload.nbf
13      }
14
15      # formattazione e decodifica del token
16      payload := p {
17          [_, p, _] := io.jwt.decode(input.token)
18      }
```

Codice 4.1: Regole per la gestione dei token OAuth 2.0. Dopo che il token viene decodificato, si estrae il payload e se ne controlla la validità temporale. Infine, se i permessi associati al ruolo corrispondono all'operazione che l'utente sta tentando di eseguire, l'accesso è consentito.

Come mostrato nel Codice 4.1, `io.jwt.decode` è uno dei metodi che OPA offre per l'estrazione dei dati da un token. Grazie al risultato di questa operazione, è possibile verificare se i gruppi associati all'utente possano effettivamente usufruire del servizio al quale si sta tentando di accedere.
Le nuove regole inserite verranno richiamate nella clausola `allow`, in modo da influenzare la decisione finale riguardo l'autorizzazione.
OpenPolicyAgent fornisce anche altre funzioni per la gestione dei JWT, che includono la verifica della firma del token, permettendo di progettare delle politiche complesse in grado di soddisfare molti casi d'uso.

4.2 Certificati VOMS Proxy

Nell'ambito del calcolo distribuito, le *organizzazioni virtuali* (VO) rappresentano delle collezioni di gruppi di individui che partecipano a un esperimento e sono organizzati in gruppi e sottogruppi.
Il *Virtual Organization Membership Service* (VOMS) [22] è un servizio per il controllo degli accessi nei servizi distribuiti, che implementa un database per ogni organizzazione virtuale, all'interno del quale sono mantenuti i dati relativi a ogni singolo membro. Le funzionalità del servizio VOMS sono utilizzate quotidianamente da migliaia di scienziati nel mondo per autorizzare

l'accesso alle risorse dei sistemi distribuiti, come a esempio lo storage o la loro potenza di calcolo.
In questo contesto, le informazioni e gli attributi di un utente che sono gestite da VOMS vengono inserite in un certificato *VOMS Proxy*. Pertanto, quando un membro di una VO necessita di eseguire un'operazione che richiede autorizzazione (i.e. la sottomissione di un problema in relazione a un esperimento), dovrà includere il certificato nella sua richiesta di accesso.
I certificati VOMS Proxy, come indicato dal nome stesso, vengono usati per autenticarsi attraverso le informazioni fornite da un'altra entità, in questo caso una VO, ed eseguire le operazioni per conto di questa. I certificati possiedono lo stesso formato di un certificato X.509, con l'eccezione dell'aggiunta di un campo chiamato *attribute certificate* (AC), contenente un altro certificato associato alla VO.
Altre informazioni provenienti da questi certificati sono:

- I *Fully Qualified Attribute Names* (FQAN), ovvero delle stringhe di testo che indicano il ruolo di ogni utente nel gruppo e nei suoi sottogruppi.

- Informazioni riguardo il certificato dell'utente e dell'ac, tra cui nazione e stato di origine, organizzazione e common name.

L'esteso set disponibile di informazioni che è possibile associare a ciascun utente permette di fornire una notevole flessibilità e specificità nella progettazione delle regole del controllo degli accessi.
Il sistema VOMS e i certificati VOMS Proxy sono impiegati principalmente per la comunicazione attraverso *Grid Security Infrastructure* (GSI), ovvero il sistema di autorizzazione utilizzato nei grid computazionali.

4.2.1 Generazione dei certificati VOMS Proxy

Il processo di generazione di un VOMS Proxy richiede l'applicazione *VOMS Clients*, che contatta un'organizzazione virtuale per ricevere un attribute certificate, e lo inserisce in una copia della chiave pubblica dell'utente.

Tutto ciò avviene attraverso l'uso del comando `voms-proxy-init`.
Il servizio VOMS fornisce una VO associata al testing dei servizi, chiamata test.vo. I certificati usati durante lo sviluppo dell'integrazione a VOMS Proxy presentano un AC associato a questa VO e presentano un tempo di validità di 24 ore.

4.2.2 Integrazione dei VOMS Proxy

L'integrazione del supporto all'autorizzazione tramite certificati VOMS Proxy comporta la riconfigurazione del reverse proxy.
La sezione di Software Development dell'INFN CNAF ha sviluppato una patch di NGINX [14] per introdurre un modulo che estrae automaticamente le informazioni dai VOMS Proxy. Il modulo viene chiamato ogni volta che il flusso di esecuzione del reverse proxy incontra una variabile di VOMS.
Grazie a questa estensione, lo sviluppo del supporto ai certificati VOMS consiste principalmente nell'inserimento da parte del reverse proxy delle informazioni estratte dal certificato in variabili interne a NGINX.

```
1    location /operation/ {
2        ...
3        set $generic_var $voms_user;
4        set $generic_var $voms_vo;
5        ...
6    }
```

Codice 4.2: Assegnazione di valore alle variabili in NGINX, tramite la keyword `set`.

In questo modo, il modulo NJS dedicato all'invio dei dati al sistema delle politiche accede alle informazioni contenute all'interno delle variabili di NGINX, e le inserisce nel pacchetto destinato a OpenPolicyAgent.
Un esempio di assegnazione di valori in variabili interne a NGINX è visibile nel Codice 4.2.

Per permettere che le informazioni contenute nei VOMS influenzino il processo di decision-making del controllo degli accessi, è necessario l'inserimento nelle policy di regole aggiuntive. La loro semantica e sintassi sarà simile a quanto illustrato nel Codice 3.2, tuttavia si farà riferimento a una VO rispetto a un ruolo o a un gruppo.

Il sistema di autorizzazione così formulato, grazie all'aggiunta del supporto all'autorizzazione tramite token OAuth e certificati VOMS, riesce a ottenere informazioni da due tecniche di autenticazione molto diverse e a formulare delle decisioni legate al controllo dell'accesso, senza che queste siano mutualmente esclusive.

Capitolo 5

Caso d'uso: interfacciamento con StoRM-Tape

Durante le fasi di sviluppo iniziali, il servizio collegato al sistema di autenticazione era dato da una API elementare e senza funzionalità significative. Tuttavia, per quanto un'impostazione simile possa essere utile ai fini dell'implementazione del sistema, non permette di valutare correttamente le caratteristiche di un 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 il controllo degli accessi nei suoi endpoint.

5.1 StoRM-Tape

STOrage Resource Manager (StoRM) [15] è un insieme di servizi dedicati alla gestione della memoria di massa nei centri di computazione distribuita. Oltre a sistemi basati su dischi, gli storage di alcuni nodi della grid computazionale offrono anche un servizio di storage su nastro, che sono costituiti da una libreria di nastri etichettati con un codice a barre. Per accedere o modificare i dati di un file, un braccio robotico individua il relativo nastro e lo inserisce all'interno di un lettore.

Funzionalità	Descrizione
`STAGE`	Richiede che i file mantenuti su un determinato nastro siano resi disponibili sul 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.

In relazione a questo, *StoRM-Tape* è una componente middleware sviluppata attivamente dalla sezione di Software Development dell'INFN CNAF, che nella data di scrittura di questa tesi è in fase di proof-of-concept. Consiste in un'implementazione della specifica WLCG Tape REST API [13], che offre un'interfaccia comune per permettere ai client di gestire la residenza dei dati mantenuti negli storage dati basati sui nastri dei grid.
Le funzionalità principali fornite dall'API sono illustrate nella Tabella 5.1. Ai comandi `STAGE` e `ARCHIVEINFO` è associato un endpoint, che si comporta in modo diverso in base al metodo HTTP della richiesta inviata a esso. Ad esempio, la funzionalità `RELEASE` si ottiene inviando una richiesta con metodo `DELETE` all'endpoint `/stage`.

5.2 Interfacciare il sistema con il servizio

L'interfacciamento del sistema con il servizio richiede delle alterazioni alla configurazione del reverse proxy. In particolare, il modulo NJS adibito all'invio delle informazioni a OpenPolicyAgent deve analizzare l'endpoint e il metodo della richiesta per determinare il tipo di operazione che l'utente sta svolgendo.

La prima modifica consiste nel cambiamento della location "protetta", che porta a un controllo dell'accesso, facendo sì che rifletta la locazione associata all'API di StoRM-Tape.
Il metodo HTTP e l'URI della richiesta sono reperibili attraverso delle variabili built-in di NGINX e assegnabili a delle variabili generiche, in modo simile a quanto svolto nel Codice 4.2 per l'estrazione delle informazioni dai VOMS Proxy. In questo modo, è possibile accedere ai due dati dal modulo NJS, e possono essere utilizzati per stabilire il carattere dell'operazione richiesta dall'utente. Il processo di creazione della query e il suo invio al sistema delle politiche rimane inalterato dal modulo originale.
Le regole delle politiche di OPA non richiedono modifiche, purché si voglia mantenere un controllo degli accessi basato su RBAC. Ciò implica che è possibile usare JWT e VOMS Proxy per autorizzare il client, senza l'aggiunta di ulteriori funzioni.
Nonostante ciò, il database contenente le informazioni sui ruoli deve essere riscritto in modo da essere coerente con le funzionalità che fornisce StoRM-Tape.

5.2.1 Risultati

Le modifiche richieste per proteggere il servizio tramite il sistema di autorizzazione sono poche e non influenzano in modo significativo le politiche. Questo fatto dimostra la versatilità del sistema, siccome richiede poche alterazioni per essere compatibile con un servizio di carattere molto diverso da quello iniziale.

Capitolo 6

Conclusioni

Nei capitoli precedenti è stato illustrato l'intero processo di realizzazione del sistema, partendo dalla progettazione architetturale ed evolvendolo gradualmente tramite l'aggiunta di nuove funzionalità, che includono il supporto alla comunicazione crittografata e a differenti tecnologie di autenticazione.
Il Capitolo 5 ha mostrato la facilità con cui questo sistema riesce a integrarsi con un servizio già esistente, senza richiedere una riprogrammazione totale dei componenti, ma modificando 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 un componente esterno.
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 potrebbe essere lo sviluppo di un'estensione del sistema per consentire la modifica delle policy connettendosi all'infrastruttura. Come è stato illustrato nel Capitolo 3, infatti, OPA fornisce una 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 principio, una possibile implementazione 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 interessante.

6.1.2 Controllo degli accessi basato su capability

In un approccio al controllo degli accessi *capability-based*, un utente possiede una lista di oggetti o risorse con una descrizione dei permessi associati a ciascuno di essi, chiamata capability. Quando un client vuole accedere a una determinato servizio mostra a esso la rispettiva capability, e il servizio consentirà al client di eseguire le operazioni che rispettano i permessi elencati in questa. Un esempio classico è l'associazione metaforica fra una capability e una chiave che apre una porta, dove la porta è la risorsa alla quale si vuole accede.
Più volte in questa tesi si è fatto riferimento al fatto che il sistema, per come è stato impostato, implementa una tecnica di controllo degli accessi RBAC, ossia basata sui ruoli.
Nonostante ciò, le politiche possono essere riscritte per ottenere un sistema di autorizzazione con un approccio capability-based.
Questa estensione sarebbe inoltre agevolata dal fatto che i JWT forniti

da WLCG supportano un sistema del genere. Il claim `scope` [21], infatti, è riservato al mantenimento delle capability dell'utente a cui il token è associato.

6.1.3 Accesso diretto a OpenPolicyAgent da parte del servizio applicativo

Un possibile miglioramento alla topologia del sistema consiste nella riconfigurazione del servizio applicativo per permettere di accedere al sistema delle politiche quando è necessario un controllo dell'accesso da parte dell'applicazione.
In questo modo, l'eventuale processo di autorizzazione interno al servizio sarebbe anch'esso gestito da OpenPolicyAgent, che incapsulerebbe tutte le politiche e le regole di accesso dell'infrastruttura.

6.2 Osservazioni finali

L'obiettivo fondamentale di un proof-of-concept consiste nel dimostrare la realizzabilità di un sistema, e da questo punto di vista 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: 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] WLCG, *WLCG*, https://wlcg.web.cern.ch/.

[2] CERN, *The Large Hadron Collider*, https://home.cern/science/accelerators/large-hadron-collider.

[3] INFN CNAF, *Calcolo - INFN-CNAF*, https://www.cnaf.infn.it/calcolo/.

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

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

[6] INFN CNAF SD, *VOMS Client Guide*, https://italiangrid.github.io/voms/documentation/voms-clients-guide/3.0.3/.

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

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

[9] Docker, *Docker*, https://www.docker.com/.

[10] Docker, *Compose*, hhttps://docs.docker.com/compose/.

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

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

[13] WLCG, *WLCG Tape API*, hhttps://indico.cern.ch/event/1026385/contributions/4309594/attachments/2301962/3915769/WLCG%20Tape%20REST%20API%20reference%20document.pdf.

[14] INFN CNAF Software Development, *NGINX HTTP VOMS Module*, https://baltig.infn.it/storm2/ngx_http_voms_module/.

[15] Riccardo Zappi, *Understanding StoRM*, https://agenda.cnaf.infn.it/getFile.py/access?contribId=2&resId=1&materialId=slides&confId=305.

[16] OpenID, *OpenID Connect*, https://openid.net/connect/.

[17] Daniel Stenberg, *cURL*, https://curl.se/.

[18] Indigo-DC, *oidc-agent*, https://github.com/indigo-dc/oidc-agent.

[19] IETF, *RFC 6749*, https://www.rfc-editor.org/rfc/rfc6749.

[20] IETF, *RFC 3820*, https://www.rfc-editor.org/rfc/rfc3820.html.

[21] Mine Altunay, Brian Bockelman, Andrea Ceccanti, et al., *WLCG Common JWT Profiles*, 2019, https://doi.org/10.5281/zenodo.3460258.

[22] CERN, *A quick introduction to VOMS*, https://twiki.cern.ch/twiki/pub/LCG/LhcbPage/A_quick_introduction_to_VOMS.pdf.

[23] auth0, *Single Sign On*, https://auth0.com/docs/authenticate/single-sign-on.

[24] OpenPolicyAgent, *OPA REST API* https://www.openpolicyagent.org/docs/latest/rest-api/.

Ringraziamenti

Ringrazio il professor Ozalp Babaoglu per la sua disponibilità come relatore, nonostante fosse prossimo alla pensione. Sono molto grato di essere uno dei suoi ultimi tesisti della sua carriera. I suoi contributi nell'ambito dei sistemi operativi sono inestimabili.

Ringrazio il professor Francesco Giacomini, per avermi dato l'accesso a una delle esperienze formative più importanti della mia vita. Lo ringrazio inoltre per assistermi attentamente nello sviluppo del progetto e nella scrittura di questa tesi. Grazie anche per la tua pazienza nei miei confronti.

Ringrazio tutto il personale dell'INFN CNAF per aver reso l'esperienza ancora più gradevole e per avermi fatto visitare il centro di calcolo, trattandomi sempre come se fossi un vostro collega.

Un ringraziamento speciale va alla mia famiglia, per avermi sempre dato la spinta di andare avanti e per credere in me, senza negarmi mai nulla.

Infine, ringrazio tutti gli amici che ho conosciuto nel corso durante questi tre anni, e Leti, per essermi stata accanto nei momenti più bui e avermi dato tutto l'affetto che esiste in questo mondo, anche quando non me lo meritavo.