ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA
Campus di Cesena

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso Di Laurea in Ingegneria e Scienze informatiche

Cluster Hadoop su un cluster Swarm: un approccio automatizzato

Elaborato in:
Virtualizzazione e Integrazione di Sistemi

Relatore:
Prof. Vittorio Ghini

Presentata da:
Daniel Benazzi

Anno Accademico 2022/2023

Indice

Introduzione

Grazie allo sviluppo tecnologico la potenza di calcolo offerta dai computer continua ad aumentare, ma contemporaneamente cresce anche la sua domanda.
In certi casi un unico computer non basta, servono gruppi di computer che collaborano per risolvere i problemi assegnati, questi gruppi sono detti “cluster”.

All’interno dell’università, per fini di ricerca, sono diventati necessari dei cluster, sistemi di calcolo distribuito, tuttavia in molti casi sono complessi da configurare e mantenere in funzione.
Lo scopo del progetto è quello di fornire ad un gruppo di ricercatori un cluster su cui eseguire i programmi necessari.
Si vuole creare un sistema facilmente avviabile e manutenibile, perché i tecnici dell’università, che si occuperanno della sua gestione, possano mantenerlo in funzione in modo relativamente semplice.

Il progetto sarà diviso in due parti.
La mia tesi si concentrerà sulla prima, dovrà creare l’infrastruttura del sistema, e fornire gli strumenti per avviare i programmi di calcolo distribuito.
La seconda parte del progetto sarà dedicata alla configurazione di tali programmi da fornire agli utenti, ovvero i ricercatori dell’università; i programmi richiesti riguardano la suite di **Apache Hadoop**.

Il progetto ha alcuni requisiti.
La preparazione dei computer e dei programmi dovrà essere il più possibile automatica, deve essere svolta da dei software su richiesta dell’utente, non manualmente da delle persone.
Verranno usati due software per l’automazione: **MaaS** e **Ansible**.
I programmi dovranno essere virtualizzati a livello del sistema operativo, grazie all’uso di container.
I container saranno creati con **Docker** e i computer del sistema coopereranno grazie alla sua funzionalità **Swarm**.

Verrà quindi creato un cluster (Hadoop) su un cluster (Swarm), i due cluster dovranno essere avviati autonomamente dal sistema; per questo il titolo “un approccio automatizzato”.

I primi due capitoli sono dedicati alla spiegazione di Hadoop e Docker, si passa poi a spiegare, nei capitoli quattro e cinque, il contesto del progetto e la sua struttura generale.
Il sesto capitolo spiega il funzionamento degli strumenti usati.
I capitoli sette e otto sono dedicati rispettivamente a: l’implementazione del sistema e la valutazione del risultato ottenuto.

Il progetto è svolto in collaborazione.
della seconda parte di occuperà un collega del corso di studi che si concentrerà sulla containerizzazione di Hadoop e il suo avvio.

Capitolo 1
L'ecosistema di Hadoop

In questo capitolo viene data una panoramica della struttura e dei software di Hadoop, essendo la creazione di un sistema distribuito e containerizzato, sul quale eseguire Hadoop, il fine del progetto.

Vengono prima spiegati: scopo e vantaggi di Hadoop; segue brevemente la sua storia, per poi parlare della struttura e dei componenti parte del sistema; prima si parla dei componenti fondamentali poi di alcuni dei programmi esterni che possono cooperare con Hadoop.

Non sono esposti dettagli di basso livello su funzionamento ed uso dei componenti di Hadoop.
Questo Poiché sarà il collega (Riccardo Quercetti), con il quale la tesi è svolta in collaborazione, ad occuparsi della configurazione e dell'instanziazione del cluster di calcolo.

1.1 Il Framework Hadoop

Con il nome "Hadoop" [1], [2] si identificano un **gruppo di programmi** e strumenti *open source* [1] nati per permettere l'analisi di grandi quantità di dati ed eseguirla su gruppi di computer detti ***cluster***[2].

Il numero di macchine appartenenti ad un cluster può andare da poche unità (anche una singola) fino centinaia e Hadoop permette il loro coordinamento, distribuendo i compiti da svolgere ad ognuna.
L'analisi dei dati avviene quindi con tecniche di **calcolo distribuito** (*distributed computing*): i computer eseguono parallelamente gli stessi calcoli, o calcoli simili, su una porzione dei dati in ingresso, così da ottenere i risultati nel tempo più breve possibile.

I dati usati dai nodi del cluster sono a loro volta gestiti da Hadoop attraverso un ***File System* distribuito**.
Le porzioni dei file vengono: suddivise e distribuite tra le macchine appartenenti al cluster; in questo modo i computer forniscono "potenza di calcolo" e spazio di archiviazione.
Un approccio di questo tipo permette diversi vantaggi, tra i quali: la possibilità di usare tecniche per la ridondanza dei file, così che non vengano persi se una macchina ha problemi, e la possibilità di assegnare compiti di calcolo ad una macchina in modo che questa lavori su dati presenti sulla macchina stessa, così che siano accessibili immediatamente e non debbano essere letti attraverso la rete.

I principali vantaggi di Hadoop sono:

1. Open source: programmi con codice accessibile pubblicamente al quale chiunque può contribuire
2. Cluster: traduzione letterale: grappolo. Gruppo di computer connessi che cooperano.

- **Scalabilità e flessibilità**: si adatta a cluster molto diversi tra loro, per numero di macchine, ma anche per *hardware*, l'architettura delle macchine, e *software*, il sistema operativo.
- **open source**
- **Resilienza**: nasce e viene sviluppato considerando che le macchine possono avere problemi, quindi i programmi devono riuscire a adattarsi autonomamente ai cambiamenti, non perdendo dati e continuando il lavoro.

Queste caratteristiche rendono Hadoop molto versatile ed adatto a diverse situazioni; sono molte infatti le aziende informatiche che da anni lo utilizzano nei loro sistemi, in ambiti come: *data analysis* e *machine learning*, ad esempio per l'indicizzazione di documenti o pagine web, per analisi di mercato o per generare raccomandazioni personalizzate ai singoli utenti.
Tra queste sono presenti nomi importanti come: Yahoo!, Spotify, IBM e molte altre [3].

1.1.1 Storia di Hadoop

Hadoop nasce nel 2004 grazie a Mike Cafarella e Doug Cutting, con il nome "Nutch", dopo aver letto due pubblicazioni di Google: "The Google File System" e "MapReduce: Simplified Data Processing on Large Clusters" [4], [5].
Cafarella e Cutting volevano creare un software per l'indicizzazione di pagine internet.
La prima versione venne rilasciata nell'aprile 2006 e venne sviluppata in collaborazione con Yahoo!.
Due anni dopo Yahoo! donò il software alla Apache Foundation che lo mantiene da allora; nel dicembre 2007 venne rilasciata la versione 3.0, attiva oggi.

Il nome "Hadoop" venne scelto da Cutting, era il nome dell'elefante pupazzo del figlio, elefante simbolo di Hadoop.

1.2 struttura di Hadoop

Le tecnologie principali di Hadoop sono state sviluppate in **Java**, linguaggio usato anche per la definizione dei programmi da eseguire sui nodi, negli anni, tuttavia, sono stati creati strumenti alternativi che non ne richiedono l'uso.
Gli strumenti alternativi non sono necessari al funzionamento di Hadoop ma sono installabili separatamente; oltre a questi molti altri strumenti sono installabili.
Hadoop ha infatti una struttura organizzata a moduli, si compone di pochi componenti fondamentali (spiegati sotto) che forniscono funzionalità ad altri, opzionali.
Alcuni degli strumenti aggiuntivi sono nati per essere usati con Hadoop, una parte di questi

sono programmi indipendenti, talvolta creati sempre da Apache, che possono essere configurati per cooperare con Hadoop.
Si parla per ciò di "**ecosistema Hadoop**"

1.2.1 componenti principali

HDFS (Hadoop Distributed File System)

Questo componente si occupa della distribuzione e salvataggio dei dati sui nodi del cluster configurati per fornire archiviazione.
HDFS distribuisce i file replicandoli perché non vengano persi in caso di problemi ad un nodo (**resilienza**).
La sua coordinazione con YARN permette alle macchine, quando possibile, di eseguire calcoli sui dati salvati nella memoria locale, migliorando il **throughput** (rendimento).
A differenza dei F.S. (*File System*) tradizionali HDFS è stato ottimizzato per la memorizzazione di file di grandi dimensioni, sotto una certa soglia non vengono divisi e distribuiti ma sono salvati in un singolo nodo master.

Solitamente l'alternativa ad un sistema come HDFS, nell'ambito di utilizzo di Hadoop, è un RDBMS[3].
Questi sistemi tuttavia richiedono la definizione di una struttura dei dati fissa, (relazioni) prima che vengano memorizzati e, di conseguenza, una fase di "pre processamento" per rispettare le relazioni.
Hadoop invece non ha restrizioni su struttura e tipo dei file, ma, a differenza di un RDBMS, non permette la modifica dei dati una volta memorizzati, fatta eccezione per le modifiche in *append*.

HDFS fornisce accesso ai dati tramite una sua linea di comando o attraverso un'API[4] web.

YARN (Yet Another Resource Negotiator)

Componente per l'esecuzione dello *scheduling* delle operazioni all'interno del cluster.
Le risorse, come: memoria volatile e CPU, vengono assegnate ai programmi dell'utente.

Introdotto nella versione 2.0 di Hadoop, YARN permette di definire diverse regole di distribuzione dei calcoli alle macchine, tra queste MapReduce (spiegato sotto).

Grazie al coordinamento tra YARN e HDFS i nodi possono lavorare su dati locali.

3. RDBMS (Relational Database Management System): programma per l'accesso e la gestione di una base di dati relazionale; permette l'accesso ai dati con query SQL.
4. API (Application Programming Interface): interfaccia di un applicazione che permette la comunicazione tra due o più programmi. Una API definisce quali messaggi e richieste può accettare il programma.

MapReduce

Modello di programmazione distribuita, adatto all'elaborazione di grandi quantità di dati, Nasce dalla pubblicazione di Google su MapReduce ed è stato programmato in Java.

MapReduce usa la tecnica del *divide-et-impera* e lavora in due fasi: prima i calcoli sono suddivisi tra i nodi del cluster (fase di map), poi i risultati vengono raccolti (ridotti) e raggruppati in un unico risultato finale.
Per fare questo MapReduce sfrutta la **programmazione funzionale**[5]: l'utente definisce funzioni pure[6], contenenti i calcoli del compito da svolgere, e MapReduce si occupa dell'esecuzione distribuita.
L'uso di funzioni pure, definite come classi Java, permette di evitare l'accesso concorrente ai dati: tutti i nodi eseguono la stessa funzione pura ma su una porzioni dei dati diverse. La porzione viene scelta per ogni nodo da MapReduce.

A partire dalla versione 2.0 di Hadoop l'algoritmo MapReduce viene re-implementato con YARN; ora, oltre a MapReduce, sono disponibili altri modelli di calcolo distribuito.

Hadoop Common

Insieme di librerie e moduli condivisi.
Forniscono supporto agli altri moduli di Hadoop e ai componenti aggiuntivi installabili separatamente.

1.2.2 componenti aggiuntivi

Pig [6]

Strumento di alto livello per la definizione di programmi distribuiti da eseguire su Hadoop. Usa un proprio linguaggio, chiamato Pig Latin e simile ad SQL, che viene compilato in una serie di programmi MapReduce.
Pig Latin può essere esteso dagli utenti con altri linguaggi, ad esempio con: Ruby, Python, Java, ...

Hive [7]

"Sovrastruttura" di Hadoop; permette l'accesso ai dati salvati nel HDFS con *query* SQL. Le ricerche sono tradotte in programmi per MapReduce ed ottimizzate.

È stato creato da Meta (Facebook), prima di essere donato ad Apache, ma lo hanno esteso altre aziende, tra cui: Netflix e Amazon.

Hive usa una versione non standard di SQL chiamata "HiveQL" ottimizzata per il compito.

5. Programmazione funzionale: stile di programmazione incentrato sull'uso di funzioni pure.
6. Funzioni pure: elemento fondamentale della programmazione funzionale; il risultato di una funzione pure dipende esclusivamente dai parametri in ingresso (non usa dati esterni).

Spark [8]

Motore per il calcolo distribuito dei dati, non necessita di Hadoop per eseguire ma può interfacciarsi al HDFS, per caricare i dati.
A differenza di YARN, Spark lavora caricando i dati nelle memorie RAM dei nodi, invece di leggerli direttamente da disco; per questo motivo si presta particolarmente all'addestramento di algoritmi di *machine learning*.

È stato creato all'università di Berkeley e poi donato ad Apache.

Kafka [9]

Con Kafka possono essere letti e gestiti flussi di dati da fonti come: sensori di dispositivi IoT, flussi video o log di programmi (*log aggregation*).
Kafka riceve i dati, li memorizza e analizza e permette una reazione; le reazioni sono definite da eventi: azioni da svolgere dipendenti dalle caratteristiche dei flussi di dati.

I vantaggi di Kafka sono: scalabilità, latenza bassa e memoria persistente.
Può essere integrato con Hadoop usando il F.S. e usando Hadoop come fonte di dati.

Viene creato da LinkedIn come semplice broker di messaggi[7], ma una volta donato ad Apache è stato modificato per dati in tempo reale.

7. Broker di messaggi: programma che fa da intermediario per la ricezione e distribuzione di messaggi.

Capitolo 2
Virtualizzazione e Docker

in questo capitolo vengono spiegati i concetti alla base del funzionamento di Docker e Docker Swarm; partendo dalla definizione di virtualizzazione ci si sposta poi sulla containerizzazione, il confronto tra queste e i relativi vantaggi.
Si passa poi a parlare del funzionamento dei container, di Docker, strumento fondamentale del progetto, ed infine della funzione Swarm di Docker.

2.1 Virtualizzazione e container

2.1.1 virtualizzazione

Con la **virtualizzazione** vengono create risorse virtuali, queste sono assegnate ad un sistema operativo, il sistema operativo esegue quindi su hardware simulato.
Con virtualizzazione si intende il processo di **creazione di macchine simulate** [10]

L'insieme di hardware simulato e sistema operativo prende il nome di **macchina virtuale** (M.V.), o macchina ospite; il computer che fornisce le risorse viene detto macchina ospitante, o macchina fisica.

Le risorse virtualizzabili possono essere diverse: dischi di memoria, processori, reti, ...
Queste possono essere spartite tra più macchine virtuali, così da fare un uso più efficiente delle risorse fisiche di un host, infatti un computer, nella maggioranza dei casi, non usa sempre a pieno l'hardware, può quindi essere distribuito a più macchine.
Il sistema che fornisce e gestisce le risorse virtuali si chiama **hypervisor**, ne esistono diverse tipologie, nella figura 2.1 si possono vedere due esempi di esecuzione di M.V..

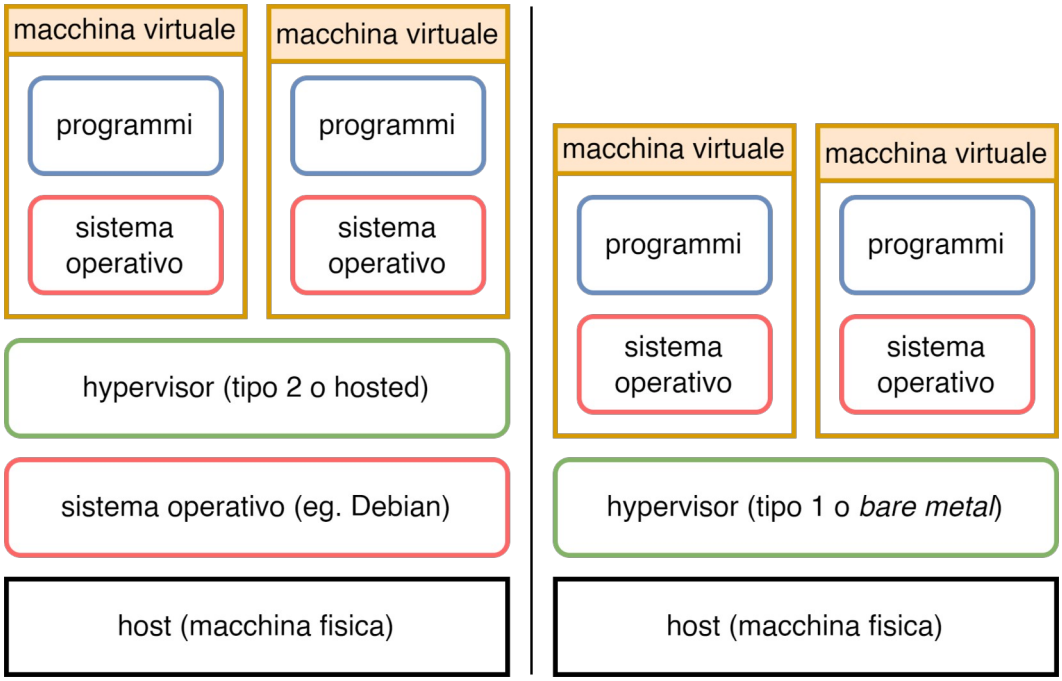

Figura 2.1: Architettura virtualizzazione e confronto tra tipologie di hypervisor

2.1.2 containerizzazione

Similmente alla virtualizzazione, anche la **containerizzazione** (o virtualizzazione a livello del S.O.) permette un uso più efficiente delle risorse.
Con containerizzazione si indica la creazione di pacchetti di codice contenenti il programma da eseguire e le sue dipendenze: file di configurazione e librerie [11]; gli altri strumenti vengono forniti dal sistema operativo su cui esegue il pacchetto.
Un approccio di questo tipo permette un **comportamento consistente** dei software tra più ambienti di esecuzione, rendendoli **portabili**.

La virtualizzazione simula un computer, su cui esegue un S.O. completo, su cui vengono eseguiti i programmi necessari; replicare il processo per molte macchine porta alla duplicazione superflua di codice.
La containerizzazione, invece, condividendo il kernel della macchina ospite tra più container, porta molti degli stessi vantaggi con costi minori.

I pacchetti sono chiamati immagini, le immagini in esecuzione container; i container, proprio come un'applicazione normale, possono essere avviati e fermati.
I container sono gestiti da un *container runtime*.
Un container runtime riceve i metadati delle immagini con cui crea i container.
I container runtime sono programmi di "basso livello", espongono API e si interfacciano al

kernel del S.O., raramente sono usati direttamente dall'utente.
Il programma che riceve gli input dell'utente è chiamato *container engine*, si occupa di creare, scaricare e modificare le immagini che i container runtime avviano in container; il container engine usa le API del container runtime.

Standard OCI (Open Container Initiative)

Ad oggi il formato delle immagini e le API fornite dai principali container runtime sono **standardizzate**; di questo se ne occupa OCI (Open Container Initiative) [12], un progetto creato da *Linux Foundation* per uniformare la containerizzazione a livello del S.O..
ad OCI fanno parte molti dei colossi informatici, tra cui: Microsoft, RedHat, Intel, Fujitsu.

OCI definisce due standard: "image specification" e "runtime specification", rispettivamente la definizione: dell'immagine da cui vengono creati i container e delle operazioni di configurazione ed esecuzione di un container.

OCI fornisce un'implementazione open source della runtime specification chiamata *runc*; il container runtime su cui si "appoggiano" molti dei container engine in ambito Linux.
Sono i container engine ad aderire alla image specification.

Nell’immagine 2.2 si può vedere la struttura del sistema di OCI.

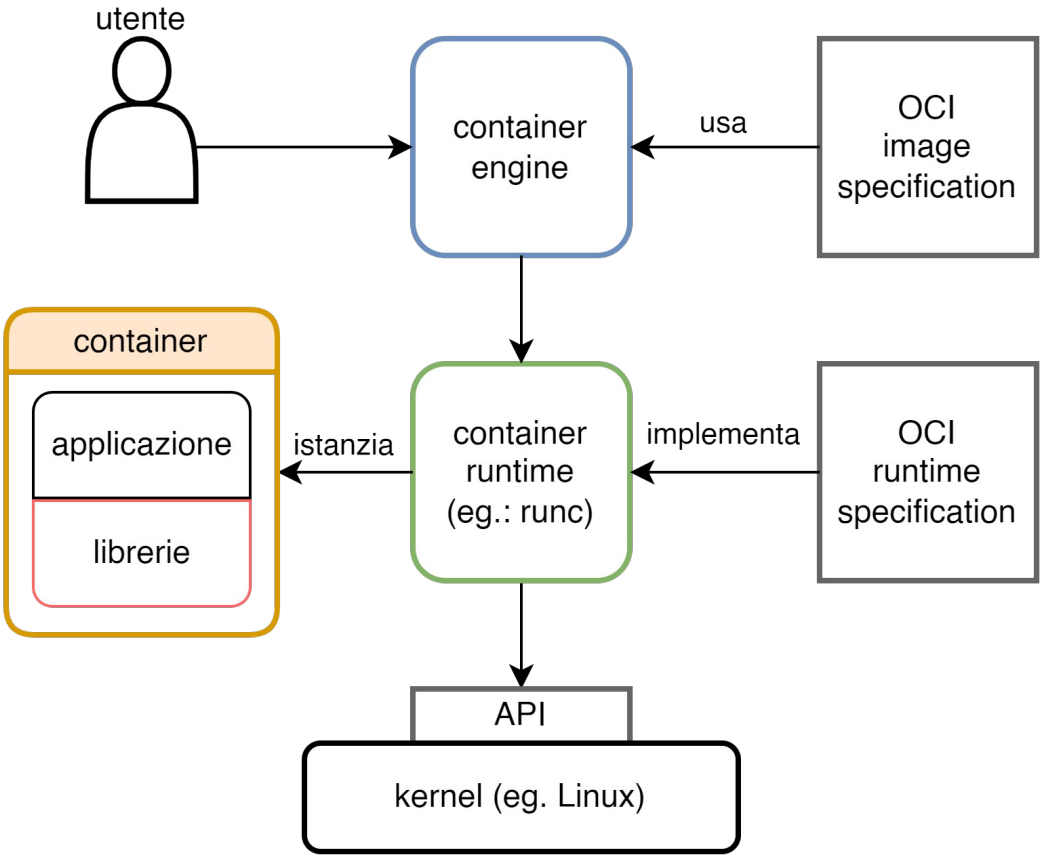

Figura 2.2: Sequenza di operazioni e componenti per la creazione di un container

Funzionamento e vantaggi della containerrizzazione

Per isolare i processi dei container dal sistema, runc usa funzionalità del kernel Linux chiamate **Namespace**: i namespace sono risorse virtuali che possono essere assegnate ai processi.
I processi possono "vedere" solo i gruppi di risorse a loro assegnati alla creazione.
Alcuni dei namespace principali forniti dal Kernel sono: PID (Process ID), network e mount namespace; questi permettono: isolamento dei processi, della rete e del filesystem.
Grazie ai namespace i container, eseguono come gli altri processi del sistema ma, "credono" di avere un kernel a loro dedicato.

Oltre alla migliore efficienza rispetto alle M.V., i container portano altri vantaggi, tra questi: la **portabilità** e l'**isolamento**.
Un container che segue gli standard, grazie alla pacchettizzazione delle dipendenze, può essere avviato su macchine diverse e agire in modo consistente.
L'isolamento dagli altri processi aumenta la sicurezza e facilita la gestione delle

applicazioni; in caso di problemi ad un container non interferiscono con altri programmi in esecuzione.

In ambito Linux si possono trovare diversi container runtime: Linux ne ha uno proprio chiamato "*LXC*" , per progetti grandi può essere usato Kubernetes, che permette coordinamento e monitoraggio di un grosso numero di container; ma il container runtime più usato è **Docker**, relativamente immediato e ricco di funzioni.

2.2 Docker

Docker [13] nasce nel 2013 ed ha guidato l'evoluzione della containerizzazione, infatti: runc, il container runtime di OCI, venne creato dagli sviluppatori di Docker e sempre da questi sono nate le prime versioni degli standard per immagini e runtime; molti dei loro progetti sono open source.

Docker espone all'utente un'interfaccia relativamente intuitiva, per la creazione e l'avvio di container, chiamata **Docker engine**.
Il programma da linea di comando, detto Docker client, invia richieste al demone[1] Docker, chiamato "dockerd", dockerd a sua volta si interfaccia con runc.

Con Docker, oltre ad avviare e fermare container, si possono creare e modificare: immagini, volumi e reti; volumi e reti possono essere connessi ai container.

- Le **immagini** sono "stampi" in sola lettura da cui si creano e avviano container.
  Un'immagine contiene: metadati, configurazione del container e i suoi dati, ovvero i file con cui viene avviato il container.
  La memoria di un'immagine è organizzata a **livelli** detti *layers: i*l primo layer contiene file "completi", i layer successivi contengono cambiamenti, differenze, rispetto al livello precedente.
  Questo approccio modulare permette il riuso dei layer in comune tra immagini diverse.
  Le immagini sono salvate in archivi chiamati registry; i registry permettono di: cercare, scaricare o caricare immagini.
  Docker fornisce un registry gratuito, accessibile pubblicamente , chiamato DockerHub [14] ma possono essere creati registry privati.
- Normalmente i dati scritti in un container sono temporanei, durante l'esecuzione dei programmi sono salvati nella RAM del sistema, per questo esistono metodi per rendere i dati persistenti; tra questi ci sono i **volumi**.
  I volumi sono spazi di memoria che possono essere "montati" nei filesystem dei

1. Demone: programma che esegue come processo in secondo piano.

container durante il loro avvio.
Un container vede i volumi come delle normali directory, in realtà sono file, gestiti da Docker, salvati in una zona protetta della macchina host, così che siano inaccessibili ad altri processi.

- Docker permette la gestione di **reti virtuali** da fornire ai container.
  Durante l'installazione di Docker viene creata una rete predefinita che si comporta come una LAN virtuale; se non specificato i container sono connessi automaticamente a questa.
  Dal punto di vista dei container le reti sono reali: i container hanno un indirizzo IP legato ad un'interfaccia ed un gateway (se la rete non è isolata); I container nella stessa rete possono comunicare tra di loro direttamente.
  Sono possibili diversi tipi di rete, tra queste: bridge, il tipo della rete preimpostata, e overlay, una rete che permette di connettere più demoni Docker, e quindi container, su macchine diverse.

Le reti, i volumi e i container possono essere creati da linea di comando ma in caso di applicazioni complesse questo approccio può essere impratico; Per semplificare l'avvio di un'applicazione può essere usato **docker-compose** [15].
Compose si serve di file YAML, chiamati "compose file", per definire i container e le strutture accessorie di un'applicazione (reti e volumi).
In un compose file i container sono organizzati in **servizi**; un servizio corrisponde ad un'immagine unita eventualmente a: variabili, configurazioni, reti e volumi.
Le reti e i volumi possono a loro volta essere definiti nel compose file.

Un'immagine permette il riuso di container su macchine diverse in modo consistente; docker-compose permette l'avvio di applicazioni su macchine diverse in modo consistente.

2.2.1 Docker Swarm

Talvolta per i container di un'applicazione non basta un'unica macchina; che sia per la necessità di molte risorse, o per garantire l'esecuzione senza interruzioni, in caso una delle macchine abbia problemi.

Una soluzione può essere **Docker Swarm** [16] una funzione di Docker con la quale si possono coordinare più demoni Docker per eseguire container, formando un cluster.
I sistemi di questo tipo sono detti orchestratori di container.
Il cluster di demoni Docker prende il nome di "Swarm"; una macchina può partecipare allo swarm come **nodo manager**, **nodo worker** o entrambi.
I manager coordinano l'esecuzione dei container, i worker eseguono i container.

Lavorando con Swarm si definiscono **servizi** (services), come si definiscono per Docker compose, che contengono immagini e configurazioni.
I nodi manager assegnano per ogni servizio delle unità di lavoro ai nodi worker, le unità di lavoro sono detti **task**, un task corrisponde ad un container.
A differenza dei servizi di compose, quelli di swarm possono essere replicati, cioè possono corrispondere a diversi task quindi più container dello stesso tipo su una o più macchine.

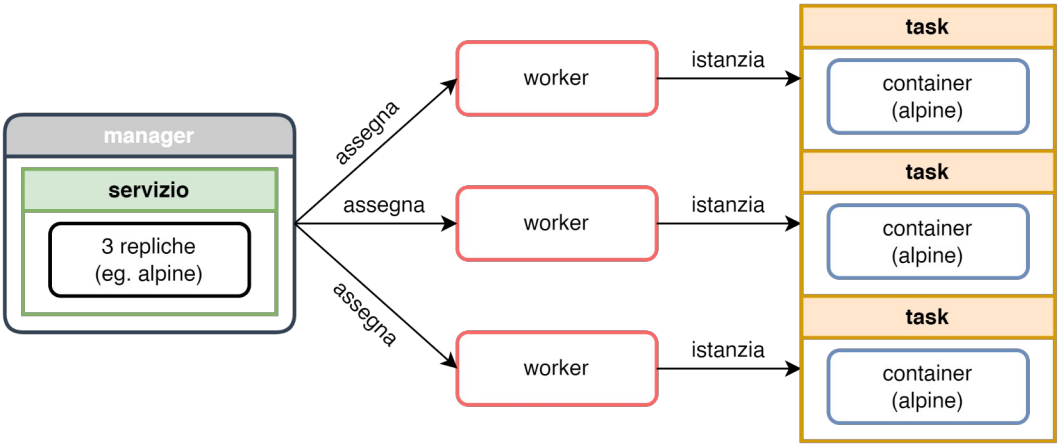

Figura 2.3: Esempio di assegnamento task, dal nodo manager ai nodi worker, in un cluster Swarm

Specificando servizi e non direttamente i container da eseguire, specifico uno **stato desiderato** del programma; i nodi manager si occupano di distribuire i task per raggiungere lo stato.
Questo porta il vantaggio di poter modificare la definizione dei servizi durante l'esecuzione e lo swarm può aggiornare i container senza fermare del tutto il programma; infatti sono disponibili metodi per un rilascio graduale dei nuovi task.
Nella Figura 2.3 viene mostrato un esempio di assegnamento dei task.

Load balancing

Docker swarm ha anche una funzione integrata di ***load balancing***.
Quando un servizio espone una porta accessibile all'esterno dello swarm, i client possono connettersi a quella porta su un nodo qualunque dello swarm, che il nodo stia eseguendo o meno un task del servizio.
I pacchetti di rete destinati ad un servizio sono reindirizzati ad uno dei nodi che esegue il task.
Il re-indirizzamento avviene mediante una rete interna allo swarm di tipo overlay.
Allo stesso modo, quando ci sono più task legati ad un servizio accessibile esternamente, i nodi manager decidono verso quali task instradare i pacchetti.

La decisione viene presa considerando il numero di connessioni verso un task, distribuendo i collegamenti equamente; questo metodo si chiama **ingress load balancing**.

Capitolo 3
Contesto e requisiti del progetto

Vengono ora spiegate le motivazioni del progetto ed il contesto: perché serve e a chi è destinato.
Si passa poi a proporre una soluzione e i suoi vantaggi, per definire i requisiti che questa deve rispettare, sia dal punto di vista dell'uso sia dal punto di vista dell'amministrazione.

Le scelte progettuali presentate nei capitoli successivi, che riguardano strumenti e architettura, dipendono dai requisiti esposti in questo capitolo.

3.1 Contesto

Un gruppo di ricercatori interno all'università necessita di un cluster di macchine su cui eseguire calcoli.
La ricerca del gruppo si concentra sull'addestramento di algoritmi di apprendimento automatico (*machine learning*) e analisi di *Big Data*.
Per questo motivo i programmi usati e richiesti sulle macchine sono quelli del framework di Hadoop esposti nei capitoli precedenti.

L'università aveva già reso disponibile un gruppo di macchine ma non sono più fruibili semplicemente, per due ragioni.
La prima: il programma con cui erano configurati e coordinati i computer non può più essere usato; il software, prima gratuito e open source, ora richiede una licenza a partire da un certo numero di nodi.
La seconda riguarda la natura del sistema, che dimostrava rigidità e difficoltà di gestione: essendo i software installati direttamente sul S.O., le macchine erano dedicate esclusivamente all'uso di Hadoop, non potevano essere usate agevolmente per altri programmi.
Inoltre l'aggiornamento dei software richiedeva di riavviare e riconfigurare i nodi manualmente, un processo lungo che oltre a portare un tempo di inattività delle macchine, rende impraticabile la sperimentazione con più versioni dei software.

Sono queste le motivazioni del progetto: si vuole creare un sistema per controllare le macchine del cluster e dispiegare i programmi.

3.1.1 soluzione proposta

I programmi non verranno installati direttamente sui nodi.
Invece si vuole creare un **cluster Docker Swarm**, sul quale eseguiranno i container contenenti: la suite di Hadoop e dei sistemi di monitoraggio.

Dovranno essere definiti dei **servizi** di swarm, per ogni programma da “containerizzare”, ma sarà poi il cluster a gestirne la distribuzione.

Il sistema dovrà essere avviato nel modo meno manuale possibile, implementando **meccanismi** di **automazione**, validi anche per l’eventuale aggiunta di nuovi nodi in un secondo momento.
Così come l’avvio si vuole rendere immediate anche configurazione e sincronizzazione dei software nei container.

Grazie a questo approccio si avranno diversi vantaggi:

- Flessibilità: avendo a disposizione delle macchine preparate per la containerizzazione si possono eseguire più programmi contemporaneamente, anche diversi tra loro, in modo isolato e sicuro
- Manutenzione e monitoraggio: individuare e risolvere eventuali problemi richiederà poco tempo.
  Si prevede l’installazione di strumenti di controllo e strumenti per la configurazione delle macchine.
- Avvio automatico: anche partendo da uno stato iniziale, vale a dire con macchine nuove, il tempo per raggiungere uno stato utilizzabile sarà relativamente breve.
  Questo può essere vantaggioso per la prima configurazione, ma anche a regime, per aggiungere altri nodi al cluster

3.2 Requisiti del progetto

Il sistema dovrà garantire alcune qualità, sia agli utenti dei programmi in esecuzione, sia per la manutenzione, che non sarà svolta dagli utenti stessi.

3.2.1 requisiti di utilizzo

I requisiti di utilizzo riguardano le caratteristiche da garantire agli utenti, ovvero chi usa i software da “containerizzare”, e sono:

- Programmi usati: lo scopo finale del progetto, creare un’installazione funzionante dei software necessari, principalmente la suite di Hadoop.
  Questo requisito richiede anche di creare una infrastruttura per installare altri programmi decisi in futuro.
- Fruibilità dei servizi: l’uso di container dovrà influire il meno possibile sull’utilizzo dei programmi.

Ad esempio l'accesso, il collegamento ai servizi, dovrà essere immediato e non influenzato da scelte strutturali.

3.2.2 requisiti di gestione

I requisiti di gestione sono legati al mantenimento in funzione delle macchine e del sistema, in termini di monitoraggio e di ripristino in caso di guasti.
I compiti di gestione non saranno svolti dagli utenti ma dai tecnici dell'università.

- Monitoraggio: in caso avvengano malfunzionamenti, dovranno essere notificati in modo autonomo, o dovranno essere individuabili semplicemente.
- Avvio/installazione: il sistema dopo una prima installazione dovrà essere facile da avviare, anche in caso di cambiamenti di versione dei programmi.
- Manutenzione: dovranno essere presenti sistemi che permettono installazione di aggiornamenti, cambiamento delle configurazioni e risoluzione di problemi.

Capitolo 4
Panoramica della struttura

In questo capitolo viene spiegata l'organizzazione delle macchine di calcolo e i loro programmi.
Nella prima parte del capitolo verrà spiegata l'architettura generale, cioè i ruoli dei computer nel sistema, insieme a dove eseguono i programmi e quali sono gli strumenti parte del progetto.
Dopo aver elencato i programmi in modo generico verranno spiegati più nel dettaglio quelli della parte di cui mi sono occupato.
Per ognuno di questi componenti verranno indicati: le azioni svolte e il perché della scelta anche rispetto ai requisiti elencati al capitolo precedente; la spiegazione dei programmi nella seconda parte avviene in "fasi", corrispondenti all'ordine con cui vengono usati nel progetto.

I Software sono solamente introdotti in questo capitolo, per poter definire la struttura del sistema, verranno descritti nel capitolo seguente.

4.1 Architettura generale

Il sistema, una volta ultimato, eseguirà su circa dieci macchine dedicate, collegate con una rete virtuale separata dal resto della rete di ateneo, così che possano comunicare tra di loro senza ritardi.
Queste macchine sono relativamente potenti e preparate per lavorare in cluster a compiti di calcolo distribuito; parlando di *hardware* ogni nodo avrà le seguenti caratteristiche: processore Intel serie i9, 256GB RAM, 3 dischi di memoria e 2 schede di rete.
Queste macchine saranno i "**nodi di calcolo**".

I nodi di calcolo eseguiranno i container e i programmi per gli utenti ma nel sistema è prevista anche una **macchina di controllo**.
A differenza delle prime questa sarà una M.V., non una macchina fisica, ma sarà comunque connessa alla rete virtuale del cluster per comunicare direttamente.
Il nodo di controllo eseguirà alcuni programmi per la **configurazione** e il **monitoraggio** delle altre macchine.
La scelta di un nodo virtuale permette, scegliendo un server affidabile, di limitare al minimo la possibilità di guasti.
Anche in caso avvengano, i sistemi di virtualizzazione permettono di spostare la macchina temporaneamente su un altro server, così da avere sempre controllo sul cluster.

Per via della natura di Docker Swarm, le macchine del cluster verranno distinte in: **nodi manager** (o master) e **nodi "worker"**.

I nodi manager saranno più di uno, ma pochi in numero rispetto ai nodi worker, che verranno controllati dai nodi manager.

I test e lo sviluppo del progetto verranno svolti con una combinazione di macchine virtuali e fisiche, non con le macchine "finali", ma simulando la struttura e le caratteristiche dell'ambiente di esecuzione effettivo.
Infatti il tipo di macchina non influisce sul comportamento dei software.
Questo è possibile grazie all'astrazione dei container, che garantisce un comportamento prevedibile e uniforme, uno dei vantaggi della virtualizzazione.

Componenti "sistemistici"

Con componenti "sistemistici" si intendono quei programmi e servizi, anche di gestione e monitoraggio, che permettono ai programmi per gli utenti di eseguire.
Per ogni componente viene indicato dove esegue; mentre le azioni svolte sono spiegate sotto (sezione 4.2).

- **MaaS (Metal-as-a-Service)[17]** : esegue nel nodo di controllo. Permette primo avvio nodi e loro monitoraggio.
- **Ansible [18]**: installato nel nodo di controllo. Permette l'esecuzione di script negli altri nodi per installazioni e per la gestione dello Swarm.
- **Docker e Swarm**: tutti i nodi fanno parte dello Swarm. Non tutti i nodi hanno gli stessi container Docker in esecuzione.
- **Portainer** [19]: Permette la visualizzazione dello stato dello swarm e dei suoi container. Esegue come più container all'interno dello swarm stesso: Ogni nodo ha un container di monitoraggio che comunica con il container del server; il container del server si trova unicamente in un nodo manager dello swarm.
- **Zabbix** [20]: Permette il monitoraggio delle macchine inviando informazioni ad un server centrale. Esegue come programma utente e si trova su ogni nodo del cluster, ma non sul nodo di controllo.

Componenti "software"

I componenti software sono i programmi e i servizi usati dagli utenti, nella struttura sono avviati, o installati, dai componenti sistemistici ed eseguono in container.

- **Hadoop** (e i componenti del framework): Esegue come container, uno per ogni nodo di calcolo; i container di Hadoop formano un cluster che esegue sul cluster di Swarm.
- **MQTT [21]**: Permette di raccogliere messaggi da dispositivi IoT (fa da broker di messaggi). Esegue come container su un unico nodo del cluster.

Nell'immagine 4.1 si possono vedere i componenti descritti e le loro interazioni, che verranno spiegate nei prossimi paragrafi.

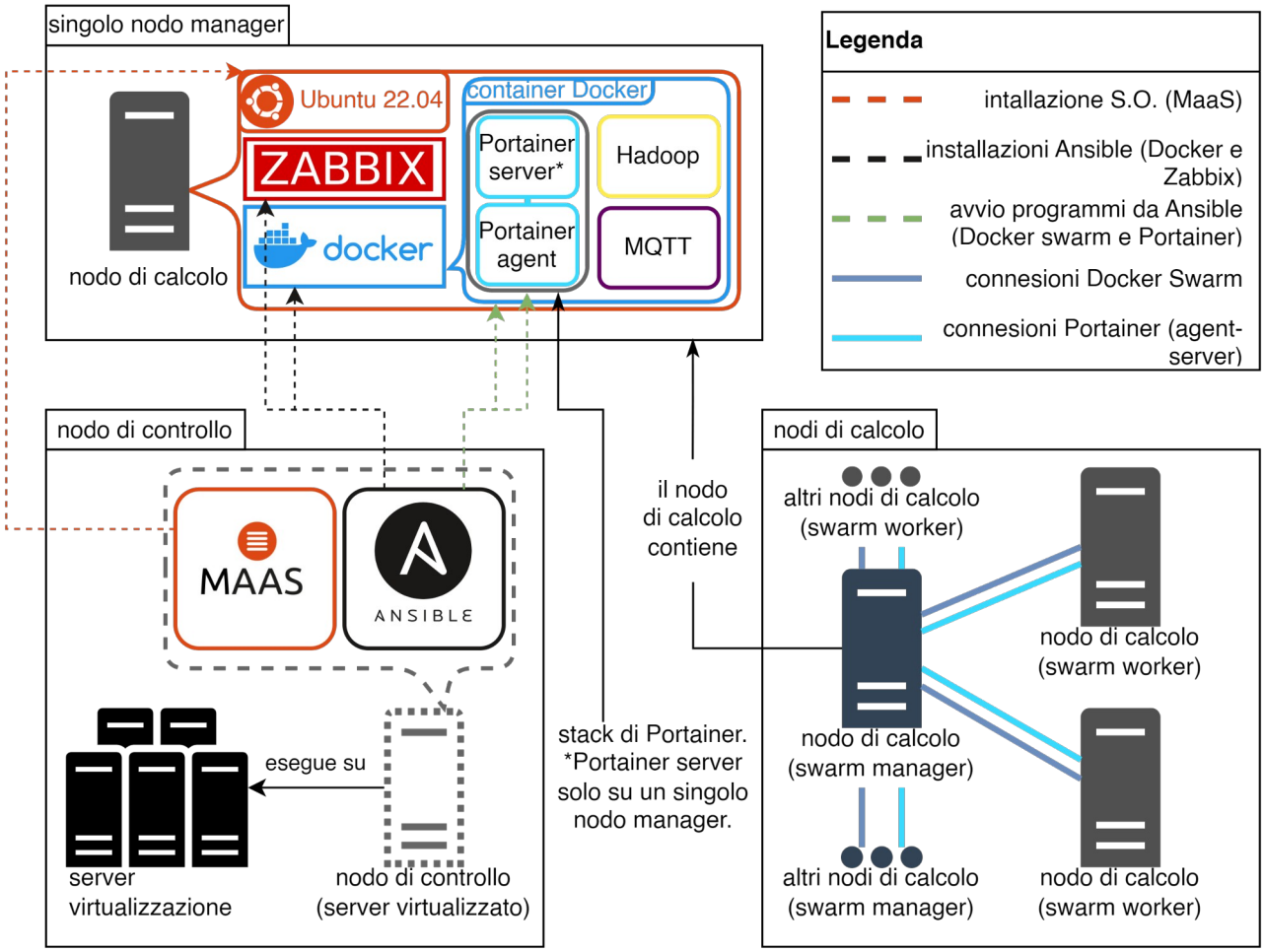

Figura 4.1: Immagine dell'architettura del sistema. Sono rappresentati i programmi in esecuzione sulle macchine e le loro connessioni.

4.2 Funzionamento dettagliato

Segue una spiegazione più dettagliata dei componenti usati nel progetto.
L'ordine di presentazione corrisponde alle "fasi" di avvio dei componenti del sistema; i programmi eseguono in ordine per: fornire servizi oppure installare altri programmi.
A sistema avviato i componenti sono usati separatamente.

Per ogni strumento viene indicato: che compiti svolge e il perché della scelta, anche in relazione ai requisiti indicati nel capitolo precedente, quindi cosa il software permette di fare.

MaaS

Nella prima fase MaaS esegue il "*deploy*" dei nodi di calcolo, ovvero la prima installazione del sistema operativo, in modo autonomo ed attraverso la rete; al momento di scrivere le macchine usano Ubuntu ver. 22.04.
Dopo il primo avvio delle macchine MaaS permette di: controllare se siano in funzione, averle in una lista/inventario, accendere i computer ed eseguire test diagnostici, tutto da interfaccia WEB accessibile sul nodo di controllo; nella figura 4.2 si può vedere l'interfaccia con le macchine di test.
Con MaaS si semplificano le operazioni di gestione delle macchine, permettendo di svolgerle a distanza, sfruttando il **boot da rete**; Quindi risponde ai requisiti di: **installazione** semplice e **manutenzione**.

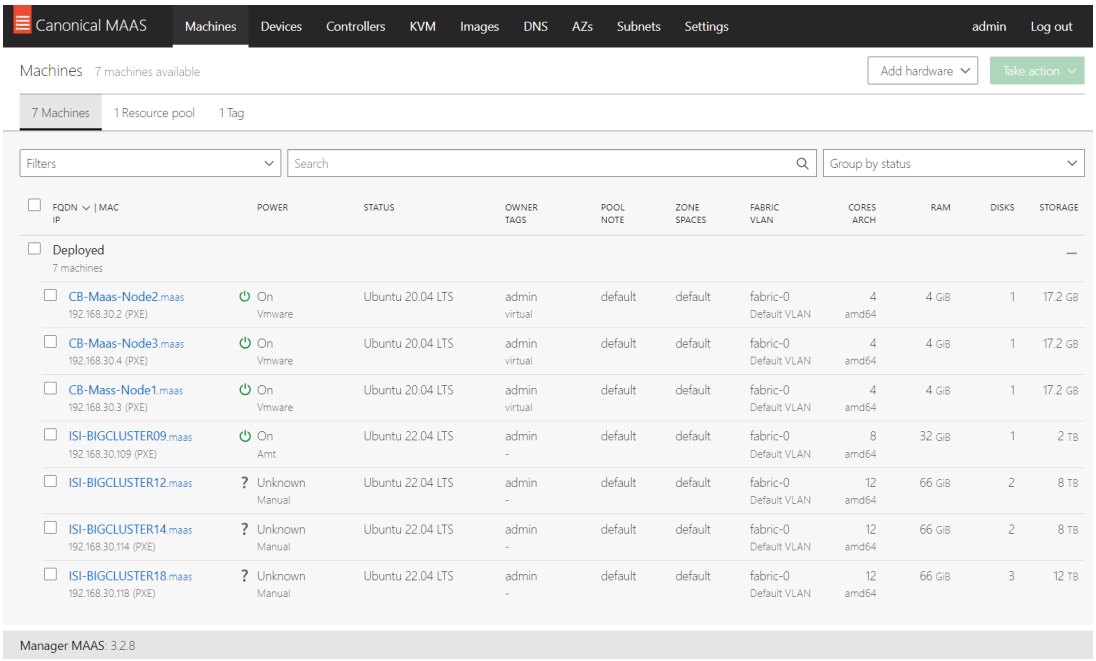

Figura 4.2: esempio di interfaccia web del server MaaS. si possono vedere i nodi usati per i test del progetto.

Ansible

Il progetto nasce per creare un **sistema automatizzato**.
Ansible permette proprio l'automatizzazione per installare e avviare i componenti del sistema.
Ansible esegue sul nodo di controllo, si connette tramite SSH alle macchine, e avvia degli script contenuti in file YAML con una sintassi propria.
A differenza di un semplice script di una shell, Ansible ha due vantaggi: una proprietà detta

idempotenza[1] e l'astrazione degli script.
Durante l'esecuzione Ansible controlla lo stato della macchina, e esegue solo le operazioni che porterebbero ad un cambiamento, quindi, lanciando più volte gli stessi script, si ottiene sempre lo stesso risultato.
L'astrazione permette di scrivere script in modo agnostico alle macchine su cui dovranno eseguire.

Con Ansible viene: installato Docker e installato Zabbix, questi solo a seguito del primo avvio del S.O.; Ansible viene poi usato per avviare lo stack Swarm di Portainer.
Anche MaaS nel nodo di controllo viene configurato e installato con uno script Ansible.

Programmi installati da Ansible

Gli script Ansible installano Docker e Zabbix.

Grazie ai container di Docker vengono forniti i programmi usati dagli utenti.
L'uso di container facilita l'aggiornamento ed il cambiamento di versione dei programmi e, usando la funzione Swarm, si possono dispiegare velocemente immagini sui diversi nodi del cluster.

Il requisito di **monitoraggio** del sistema viene soddisfatto da Zabbix; lo script Ansible installa lo "Zabbix Agent" sui nodi del sistema.
L'agent raccoglie informazioni sullo stato di esecuzione delle macchine, e dei componenti, poi le invia ad un server di Zabbix configurato dai tecnici dell'università.
Il server raccoglie i dati, li elabora e ne mostra lo storico; Il server può anche avviare operazioni in risposta ai dati, ad esempio: l'invio di notifiche per il superamento di una soglia, come la percentuale di uso del processore o dei dischi di memoria.

Programmi avviati da Ansible

Sempre con Ansible sono avviati Swarm e Portainer.

Uno script configura i demoni Docker dei nodi di calcolo per lavorare in modalità Swarm; nello stesso script vengono scelti i nodi manager e worker dello swarm.

Dopo il cluster Docker può essere avviato Portainer.
Con Portainer non serve più accedere ad una linea di comando per controllare lo Swarm.
I programmi successivi possono essere avviati in container dall'interfaccia WEB.
Dalla stessa interfaccia Portainer mostra lo stato dei nodi e dei container.
Portainer quindi risponde ai requisiti di: **facilità di gestione** e **monitoraggio** dei programmi destinati agli utenti.

1. Idempotenza: proprietà matematica di una funzione, per la quale applicando più volte la funzione al proprio risultato, si ottiene lo stesso ogni volta.

Capitolo 5
Strumenti usati

Durante lo sviluppo del progetto sono stati usati strumenti e programmi di natura diversa tra loro.
Questo capitolo definisce e spiega le caratteristiche dei componenti del sistema.

Non tutti gli strumenti esplorati e ricercati durante lo sviluppo fanno parte dell'architettura, in quanto sono stati scartati in favore di altri, o perché non adatti.
L'ultima parte del capitolo presenterà alcune di queste possibili scelte alternative, il confronto con i programmi del progetto, e il motivo per il quale non sono state scelte.

La descrizione dei programmi si limita al necessario per poterne parlare durante la spiegazione dell'implementazione.
Per approfondimenti si lasciano i collegamenti alle fonti e alle documentazioni.

5.1 Strumenti e programmi

Segue una descrizione dei programmi presenti nel sistema.
i programmi principali sono: MaaS, Ansible e Portainer, poi si parla di Zabbix e NFS.
Non si parla di Docker o Docker Swarm, anche essi parte centrale del progetto, perché già spiegati nel capitolo dedicato alla containerizzazione.

5.1.1 MaaS (Metal as a Service) [17]

MaaS fornisce un metodo di automazione per la gestione di macchine server, permette la creazione di un sistema di tipo IaaS on-premise[1], cioè con hardware gestito in modo autonomo.

I sistemi **IaaS** (Infrastracture as a Service) seguono un modello per la fornitura di servizi da parte di aziende verso i propri utenti.
I servizi offerti sono di tipo hardware, le aziende forniscono ad esempio: server fisici, dischi per spazio di archiviazione o apparati di rete.
La fornitura di servizi in un modello IaaS avviene solitamente *in cloud*, tramite internet, al momento di una richiesta degli utenti.
I servizi vengono poi "liberati" quando non servono più all'utente, diventano disponibili da assegnare al prossimo utente.

MaaS mantiene una lista delle macchine disponibili in un dato contesto.
Sulle macchine della lista MaaS può installare, automaticamente attraverso la rete, un sistema operativo e configurarlo; la configurazione viene specificata prima dell'installazione

1. On-premise: letteralmente "sul posto". Si dice di hardware o software gestito direttamente da un ente.

e può comprendere ad esempio: utenti da creare, programmi da installare o chiavi ssh preimpostate.
Il S.O. viene installato a seguito della richiesta di un utente, prima di questo i computer sono in un stato “di attesa”, spenti e senza S.O..
In questo senso i computer (la “M” di MaaS vuol dire “metal”) sono visti come un servizio: vengono scelti e forniti a diversi utenti nel momento della richiesta, quando all’utente non serve più la macchina viene riportata allo stato iniziale “in attesa”, pronta per la prossima richiesta di servizio.

Nel contesto di questo progetto MaaS verrà usato più come uno strumento di monitoraggio. I nodi saranno preparati idealmente solo una volta ma, la possibilità di avviare una macchina in poco tempo, può essere utile in caso di problemi, così da riavviare il sistema.

Le macchine gestite dal sistema seguono un “ciclo di vita” composto da diversi stadi, in ogni stadio MaaS esegue operazioni diverse.
Una macchina individuabile nella rete può essere aggiunta alla lista controllata da MaaS. Per essere aggiunta MaaS esegue un’operazione detta “*commissioning*”, viene avviato un S.O. temporaneo che raccoglie informazioni sull’hardware della macchina; i dati sono salvati in inventario.
Dopo il *commissioning* i nodi sono in stato “*ready*”, pronti per essere allocati ad un utente con l’operazione di “*deploy*” del S.O.; dopo aver finito di usare la macchina si esegue il “*release*” e il nodo torna in stato *ready.*

Per avviare installazioni e sistemi operativi MaaS usa **PXE** (Preboot eXecution environment), parte dello standard UEFI [22], un insieme di protocolli che permettono ai computer in una rete di essere avviati da un server.
MaaS deve avviare e spegnere le macchine da rete per poterle controllare autonomamente; l’impostazione si chiama “power type”, in base alle caratteristiche della macchina ne esistono diverse tipologie, tra cui: AMT per interfacce di rete Intel e “vmware” per le macchine virtuali usate nel progetto.

MaaS nel sistema viene usato come **punto di controllo** del cluster, vi si accede tramite l’interfaccia web, esposta dal nodo di controllo.

5.1.2 Ansible [18]

Ansible permette l'**automazione in ambito IT**.
Comprende un insieme di strumenti per configurare sistemi, installare software e altro, e segue la filosofia "IaC".

Nell'IaC (Infrastructure as Code) le impostazioni di programmi e macchine sono salvate sotto forma di file, spesso come script o forme di programmazione dichiarativa, evitando

file di configurazione o processi manuali.
Questo approccio permette negli ambienti di produzione di velocizzare i compiti e di salvare la cronologia dei file in un sistema di *versioning*.

Ansible, riassunto in modo essenziale, permette l'esecuzione su delle macchine di una serie di comandi, simili a script.
Ansible viene installato su un nodo di controllo e si connette alle macchine remote attraverso la rete per mezzo di SSH.
I comandi sono chiamati "**task**", liste di task sono raggruppati in "**play**" e i file contenenti le *play* sono chiamati "**playbook**".
Le macchine remote, su cui sono eseguiti i playbook, si chiamano "**host**", la lista degli host viene mantenuta in file di inventario ("**inventory**")**,** organizzati per gruppi di host.
I file con cui Ansible lavora sono in formato YAML [23].

Nel progetto Ansible viene usato come lo strumento cardine per l'automazione.
Permette di definire in modo relativamente immediato compiti di routine; così da inizializzare il sistema e avviarlo, attraverso la rete, su tutte le macchine.

I playbook vengono avviati con un comando da una shell del nodo di controllo, sugli host presenti in inventario.
Normalmente i task di un playbook procedono parallelamente: il nodo di controllo attende che tutte gli host abbiano completato un task, prima di passare al successivo.
Durante l'esecuzione Ansible raccoglie dati e valori di ritorno dei task; se un task termina con un errore, l'host del task viene escluso dall'esecuzione dei task seguenti, fino al termine della "*play*".

Nei file di inventario possono essere definite delle variabili, sono coppie chiave-valore che possono essere assegnate a gruppi o singoli host.
Le variabili possono essere usate nei task, ad esempio per esecuzioni condizionali.
Alcune variabili vengono create da Ansible, raccolgono informazioni sullo stato di esecuzione o sulle caratteristiche dell'host, e sono chiamate "special variables".
Certe *special variables* possono essere usate per cambiare il metodo di connessione verso gli host, sono dette "connection variables".

I comandi dei task sono raggruppati in moduli.
Molti dei moduli, prima di eseguire un task, controllano che lo stato dell'host corrisponda a quello finale desiderato; in caso i due corrispondano il task non viene eseguito.
Eseguire più volte il task non influenza la macchina; questa caratteristica viene detta **idempotenza** ed è uno dei vantaggi di Ansible rispetto uno script con SSH.

5.1.3 Portainer [19]

Portainer permette il controllo di diversi sistemi di containerizzazione, situati su macchine locali o servizi *cloud*, da un'unica interfaccia web.

Nel contesto del progetto viene usato per avviare e monitorare stack di servizi swarm, in modo fruibile a qualunque utente, e evitando la linea di comando su una macchina specifica.

Portainer gestisce servizi containerizzati ed esegue a sua volta in container.
I componenti dell'architettura sono due: un "Portainer server" e più "Portainer agent"; entrambi sono dei container, il server espone l'interfaccia web e comunica con gli agenti, per raccogliere informazioni o inviare comandi.
Gli agenti si connettono ad un singolo server e sono presenti su tutti i computer di un cluster Swarm.

L'installazione di Portainer su cluster Swarm avviene con un compose file fornito dagli sviluppatori.
Dopo l'avvio dello stack si hanno due servizi: uno per il server, in esecuzione su un unico nodo "swarm manager", uno per gli agent, in esecuzione su ogni worker.
Il server salva i propri dati in un volume Docker, creato sul nodo manager, così da garantire persistenza.

Portainer non si limita alla gestione di container Docker ma può controllare ad esempio anche cluster Kubernetes.
Ogni istanza di containerizzazione controllabile da Portainer si chiama "ambiente" (environment).

5.1.4 Altri strumenti

Zabbix (Agent) [24]

Lo Zabbix agent fa parte dei programmi del sistema di Zabbix.
Zabbix permette di raccogliere informazioni sullo stato di Programmi, macchine, dispositivi di rete e altro, per poi controllare, memorizzare e reagire ai dati raccolti; le reazioni possono essere ad esempio notifiche o serie di azioni.
Grazie allo zabbix Agent si possono controllare un insieme molto vasto di metriche.

L'agent di Zabbix viene installato su un sistema, in questo progetto sui computer del cluster, per il monitoraggio dello stato di risorse locali e programmi.
Alcuni dei dati che possono essere raccolti sono: spazio di archiviazione disponibile, memoria centrale usata, uso dei processori o numero di programmi in esecuzione.
I dati raccolti vengono inviati ad un server di Zabbix.

Grazie a Zabbix i tecnici dell'università possono essere notificati non appena si verifica un problema o controllare che i nodi parte del sistema siano in buona "salute".
In caso di guasti può essere utile controllare lo storico dei dati raccolti per risalire all'origine del problema.

NFS (Network File System) [25]

NFS è un protocollo per filesystem di tipo distribuito; un client si connette ad un server, il server contiene dei file, il client accede ai file attraverso la rete.
Una volta connesso il client al server, i file sono visti come fossero locali, nel caso di Linux la connessione viene gestita dal kernel.

Nel progetto è stato scelto per poter memorizzare i dati del Server di Portainer in un unico luogo.
Il volume contenente i dati, che viene connesso al container del Portainer server, si trova nel nodo di controllo; viene condiviso con NFS, non viene creato come volume Docker.
In questo modo dati e impostazioni sono sempre gli stessi; anche se il Portainer server dovesse essere avviato su un nodo swarm manager diverso dal primo.

5.2 Scelte alternative e confronti

Nello sviluppo del progetto sono state considerate soluzioni o provati alcuni strumenti alternativi a quelli che fanno parte del sistema.
In questa sezione vengono elencati e confrontati alcuni dei sistemi, per poi motivare perché sono stati scartati.

I confronti riguardano: Kubernetes, come alternativa a Docker Swarm, alcune alternative a Portainer e alcuni sistemi in sostituzione ad NFS, tra gli ultimi anche Ceph.

Docker Swarm e Kubernetes [26]

Sia Docker Swarm che Kubernetes sono strumenti di orchestrazione di container.

Docker Swarm lavora con container Docker e viene installato inseme alla CLI di Docker.
Kubernetes permette l'orchestrazione di diverse tipologie di container, tra cui: container Windows, container Docker (tramite dockerd) o container nativi Kubernetes (con CRI-O).

Uno dei vantaggi di Kubernetes, rispetto a Docker Swarm, è lo *scaling* automatico dei servizi: il numero di container di un servizio viene aumentato o diminuito dinamicamente sulla base del loro utilizzo di risorse.
In Swarm lo scaling dei servizi va gestito manualmente o tramite strumenti esterni.
Tuttavia nel progetto ogni nodo dovrà eseguire un unico container Hadoop quindi non verrebbe comunque usato uno scaling dinamico.

Docker Swarm crea automaticamente una rete di ingresso verso i propri servizi.
Questa ha un load balancing delle connessioni che, per quanto sia poco configurabile, permette di accedere facilmente ai container in esecuzione.
Infatti l'accesso ad un nodo qualunque dello Swarm indirizza sempre al servizio giusto.
Diversamente con Kubernetes sarebbe necessario avviare un container dedicato al load balancing che svolgerebbe anche il compito di "*reverse proxy*".

In Sintesi, Docker Swarm offre una buona semplicità di utilizzo, mentre Kubernetes mette a disposizione funzionalità evolute.
Il maggiore livello di astrazione di Kubernetes richiede un processo di configurazione complesso.

Nel contesto di questo progetto le funzioni aggiuntive di Kubernetes non sono necessarie.
Al contrario la sua struttura renderebbe il dispiegamento sulle macchine e la gestione delle stesse più difficile; per questo la scelta ricade su Docker Swarm.

Alternative a Portainer

Le alternative a Portainer possono essere distinte in due categorie: sistemi di gestione di cluster, ovvero sistemi complessi che astraggono dalle singole implementazioni, e i programmi che forniscono una GUI dalla quale controllare i servizi di un cluster.

Nel primo gruppo molte delle alternative a Portainer permettono unicamente il controllo di cluster Kubernetes, non sono quindi adatti a questo progetto, ma se ne lasciano i collegamenti; i sistemi ricercati sono Rancher [27] e Openshift [28].
Questi, anche in caso fossero stati funzionali al progetto, sono strumenti sovradimensionati rispetto ai requisiti del progetto.

Nel secondo gruppo di programmi simili a Portainer, che espongono un'interfaccia utente, sono pochi quelli disponibili: molti non sono più sviluppati attivamente, quindi non possono essere considerati alternative valide.
Una delle poche alternative si chiama Swarmpit [29].

Swarmpit, Come Portainer, esegue in container Docker direttamente sul cluster, ma a sua differenza non permette di controllare singoli demoni Docker, solo cluster Docker Swarm.
Swarmpit fornisce tuttavia un'interfaccia nuova e ricercata, che segue un design "*mobile friendly*".

In questo progetto si sceglie Portainer per la sua maggiore affidabilità e per la facilità di installazione.
Riguardo all'affidabilità: Swarmpit non ha un'azienda che lo sviluppa ma è un progetto *open source* di sviluppatori indipendenti.
Riguardo all'installazione: Swarmpit richiede un Database da installare separatamente,

mentre Portainer esegue completamente all'interno di container, va fatta un unica eccezione per il volume dei dati.

Alternative a NFS

Riguardo alla scelta sugli strumenti per la sincronizzazione dei volumi di Portainer, non vi erano requisiti stringenti, quindi sono state ricercate diverse alternative.
Inizialmente si era voluto creare un sistema di memoria distribuita tra i nodi del cluster, il programma scelto per questo sistema era Ceph, successivamente ci siamo orientati nella ricerca di sistemi per la sincronizzazione di file.

Ceph [30] permette la creazione di uno spazio di archiviazione, i dati sono distribuiti su uno o più nodi di un cluster, sono accessibili tramite rete e ridondanti.
Ceph nel progetto sarebbe stato installato sui nodi master del cluster, ma richiede che ogni macchina abbia un intero disco fisico da usare per l'installazione; essendo in numero limitato sulle macchine, sarebbe rimasto poco spazio da dedicare ad Hadoop, per questo motivo abbiamo scelto altri strumenti.
Oltre al problema dei dischi disponibili Ceph sarebbe stato "sovradimensionato" rispetto alle necessità del sistema.
Siccome la sincronizzazione riguarda solo i file di un volume docker, abbiamo ricercato sistemi più leggeri, tra questi lsyncd e csync2.

Sia lsyncd che csync2 sono strumenti per la sincronizzazione asincrona di file ma funzionano in modo diverso.

Csync2 [31] nasce come programma per la sincronizzazione di file tra i computer di un cluster.
Controlla i cambiamenti di una o più directory e, in caso ci siano modifiche ai file, diffonde gli aggiornamenti agli altri nodi.
Le informazioni sullo stato dei file sono mantenute in un database leggero, non vengono confrontati i file remoti come con rsync, per questo lavora in modo efficiente.
Le caratteristiche del programma sarebbero state perfette per il progetto ma sfortunatamente non viene più aggiornato.

Lsyncd [32] esegue monitorando i cambiamenti di una directory, tramite gli eventi del F.S. linux, per poi avviare uno o più processi per la sincronizzazione dei file.
I processi di sincronizzazione sono programmi esterni e di default viene usato rsync.
Lsyncd può essere usato per sincronizzare solo due macchine, nel progetto sono almeno tre i nodi di cui sincronizzare i file, quindi sarebbe stato necessario una sincronizzazione "a stella"; ogni macchina avrebbe dovuto comunicare individualmente con altre due.

Altri strumenti simili avrebbero portato al problema della sincronizzazione a stella, quindi abbiamo preferito usare NFS, creando un volume in un punto esterno al cluster.

Capitolo 6
Implementazione del sistema

Nel sistema mi sono occupato della fase di “inizializzazione”, ovvero i primi passaggi per arrivare ad un sistema pronto ad essere usato, sul quale il collega che ha lavorato al progetto e i tecnici dell’università possono avviare i software voluti.
In questo capitolo si spiega come vengono usati i programmi spiegati, insieme a alcuni dettagli di basso livello, problemi riscontrati e come sono stati affrontati.

Le operazioni svolte hanno un ordine che può essere diviso in “fasi di installazione”, le sezioni seguono lo stesso ordine, molte sono necessarie alle successive.
Le fasi di installazione seguono in parte quelle del Capitolo 4 e sono:

1. Installazione del sistema operativo tramite MaaS
2. Installazione di Docker e Zabbix tramite Ansible
3. Avvio dei programmi tramite Ansible
4. Modifiche finali per l’esecuzione ottimale del sistema

6.1 Installazione Sistema Operativo

Come primo passaggio, le macchine hanno bisogno di un sistema operativo, che viene installato grazie a MaaS.
La macchina deve seguire il “ciclo di vita” di MaaS: per l’installazione viene prima aggiunta alla lista dei nodi: *commissioning*, installazione S.O. e utilizzo.
tutte queste operazioni vengono svolte da GUI web.

Se la macchina da preparare cerca un server PXE per il Boot, questa viene aggiunta automaticamente nella lista di MaaS, altrimenti può essere specificato il suo indirizzo MAC con il form nell’immagine 6.1.
L’unica condizione necessaria è che la macchina sia sulla stessa rete della macchina MaaS.

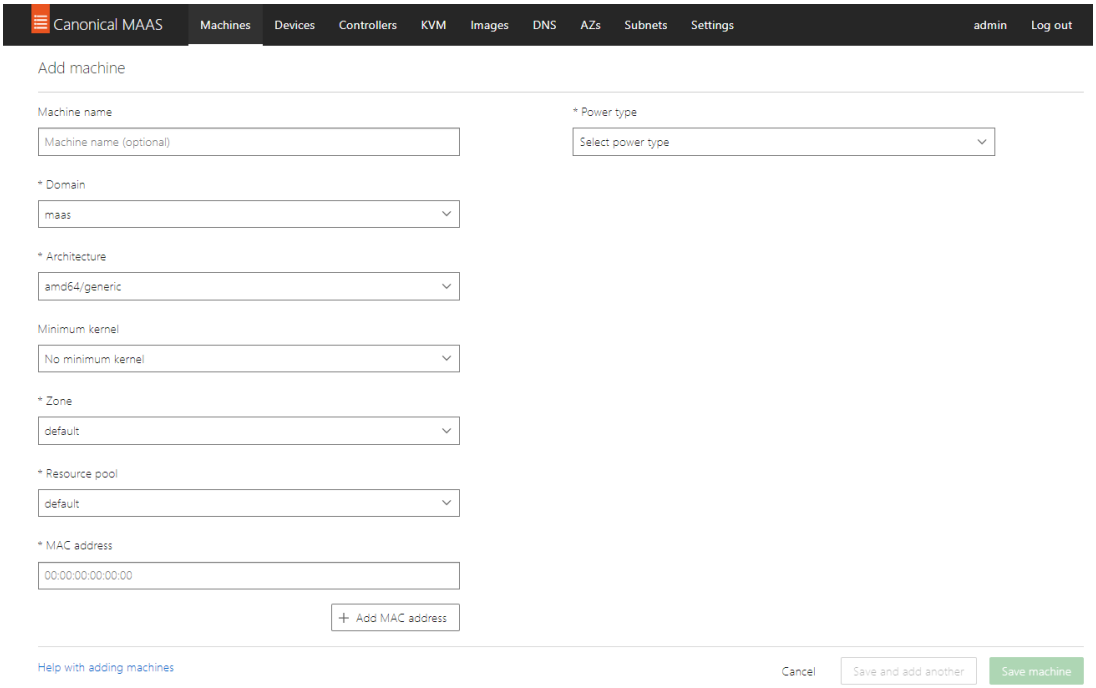

Figura 6.1: Schermata della GUI di MaaS per l'aggiunta di una macchina.

Individuata la macchina viene aggiunta alla lista, ora MaaS esegue il *commissioning*, ovvero la raccolta delle informazioni sulla Macchina; dopo questa fase i dati sono visibili dalla GUI.
Nell'immagine 4.2, del capitolo precedente, viene mostrata la lista delle macchine di test.

La procedura prosegue con l'assegnazione della macchina ad un utente di MaaS, e poi con l'installazione del sistema operativo.
L'utente verrà ritrovato come utente nel sistema operativo installato; In questo progetto abbiamo usato Ubuntu versione 22.04 e utente "ubuntu".
Nell'immagine 6.2 può essere visto un esempio di macchina configurata tra quelle usate per i test.

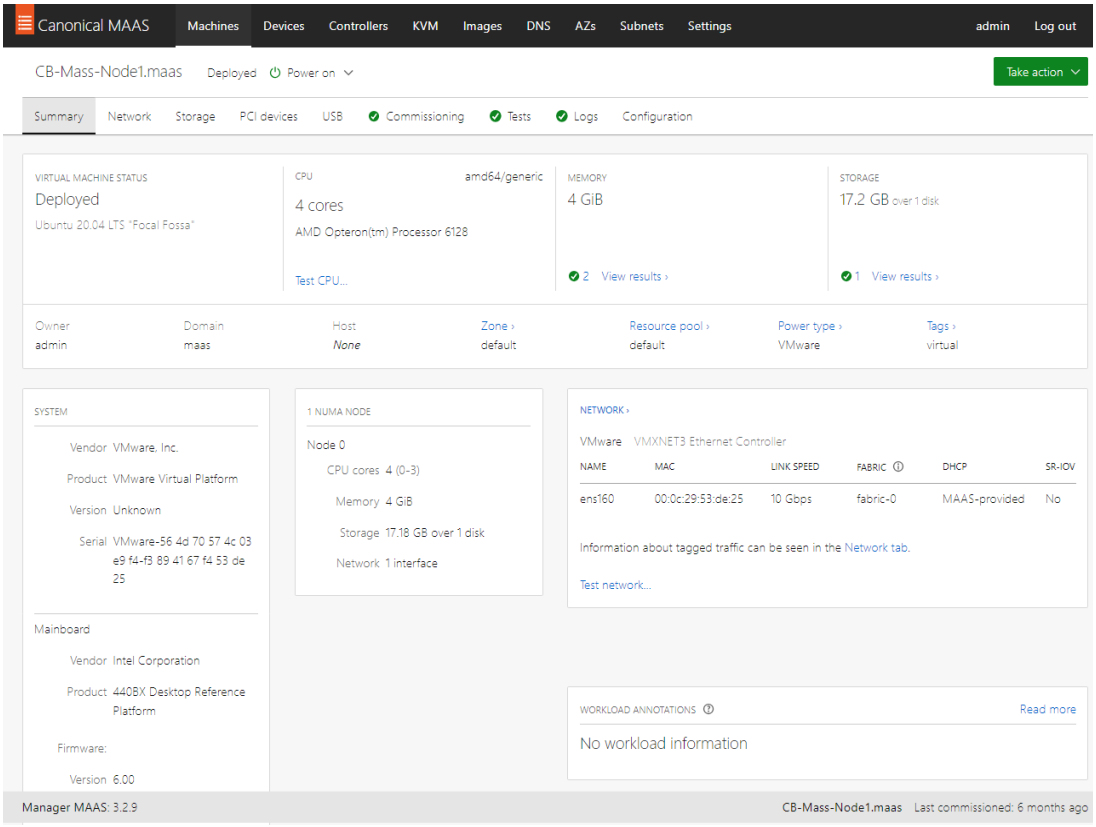

Figure 6.2: Schermata della GUI di MaaS, viene mostrato l'esempio di una macchina configurata, con il sistema operativo installato.

L'installazione del sistema operativo viene eseguita in autonomia da MaaS, ma per riuscire ha bisogno accendere e spegnere le macchine tramite rete, così da poterle riavviare durante l'installazione.
Per i nodi nella lista di MaaS può essere impostato un "powertype", un metodo per l'accensione tramite rete, può essere di diversi tipi; nell'immagine 6.3 viene mostrato un esempio della configurazione per una macchina del progetto, con powertype AMT, su una scheda di rete Intel.

Figura 6.3: Schermata della GUI di MaaS, Esempio di configurazione per accessione di una macchina tramite rete, attraverso la tecnologia AMT

Il sistema operativo scelto può essere personalizzato prima che venga installato, con alcune

impostazioni da GUI, o con degli script.
Nel progetto la personalizzazione viene usata per caricare delle chiavi SSH pubbliche legate all'utente, così che possano essere usate per le connessioni; Ansible, per connettersi ai nodi, usa una chiave privata legata a alla chiave pubblica caricata con questo metodo.

Integrazione Ansible e MaaS

La lista dei computer in MaaS potrebbe cambiare nel tempo.
Per rendere il sistema automatico, Ansible ha bisogno di caricare dinamicamente quali macchine siano attive al momento dell'esecuzione dei playbook.
Solitamente nei *repository* di Ansible sono disponibili molti *Plug-in* per l'integrazione con sistemi esterni, ma sfortunatamente questo non era il caso.

È stato necessario creare un playbook per aggiungere la lista di host all'inventario prima dell'esecuzione; il playbook può essere visto nella figura 6.4 e si chiama "`load_maas_machines.yml`".
L'inventario Ansible usato nel progetto prevede quattro gruppi di host, il playbook popola dinamicamente i gruppi di host nell'inventario, che sono:

- **Administrator:** gruppo con il nodo su cui esegue Ansible. Contiene solo questa macchina e non viene popolato ulteriormente dal playbook per le macchine di MaaS.
- **Master:** gruppo con un unico nodo, viene aggiunto dal playbook, e viene scelto tra le macchine di MaaS. Il nodo in questo gruppo verrà usato nei playbook successivi per avviare lo swarm e lo stack di Portainer.
- **Manager:** gruppo con i nodi manager dello swarm, escluso il nodo nel gruppo "master".
- **Worker:** gruppo con i nodi worker dello swarm.

I gruppi master, manager e worker appartengono al gruppo "swarm"; questi gruppi sono inizialmente vuoti, non contengono host, prima dell'esecuzione di Ansible.

Il playbook viene richiamato all'inizio dell'esecuzione di tutti gli altri playbook.
Il caricamento delle macchine da MaaS avviene in più fasi: si esegue un comando per ottenere una lista json dei nodi da MaaS, i nodi vengono selezionati con una json query [33], poi viene scelto in quali gruppi assegnare i nodi.
I nodi Manager dello swarm devono sempre essere in numero dispari, per evitare problemi in caso di disconnessioni, l'assegnazione avviene in questo playbook; questa necessità viene spiegata nel capitolo 7.2.

```yaml
---
- name: Load hosts from MaaS
  hosts: maas
  vars:
    maas_query: "[?contains(fqdn, 'ISI')].{name: fqdn, ip: ip_addresses[0]} | []"
    length_query: "[?contains(fqdn, 'ISI')] | length(@)"
    manager_node_every: 5 # add a manager machine every n machines in the list (not counting master)

  tasks:
    - name: load JSON file containing hosts info from MaaS
      command: "maas admin machines read"
      register: maas_out

    - name: Debug line for maas output
      debug:
        msg: "{{ maas_out.stdout | from_json | community.general.json_query(maas_query) }}"
        verbosity: 2

    # variables explanation:
    # maas_machines: list of hashes containing info on maas machines
    # machines_number: number of machines in the list
    - name: Filter JSON data from MaaS output
      set_fact:
        maas_machines: "{{ maas_out.stdout | from_json | community.general.json_query(maas_query) }}"
        machines_number: "{{ maas_out.stdout | from_json | community.general.json_query(length_query) | int }}"
        cacheable: true

    - name: Set host as the swarm's master node
      add_host:
        name: "{{ maas_machines[0].ip }}"
        groups: master

    # number of manager nodes
    - name: Select manager nodes
      set_fact:
        manager_nodes: "{{ ((machines_number | int) / manager_node_every) | round | int }}"
        cacheable: true

    - name: Set swarm's manager nodes
      add_host:
        name: "{{ item.ip }}"
        groups: manager
      loop: "{{ maas_machines[1: (manager_nodes | int) +1]}}" # select manager nodes from top of the list
      when: manager_nodes | int > 1

    - name: Set swarm's worker nodes
      add_host:
        name: "{{ item.ip }}"
        groups: worker
      loop: "{{ maas_machines[(manager_nodes | int):] }}" # skip master and manager nodes (if present)

```

Figura 6.4: File "load_maas_machines.yml" per il caricamento dinamico dell'inventario Ansible

6.2 Installazione Docker e Zabbix

Dopo l'installazione del S.O. si procede a rendere i nodi funzionali, con l'installazione di Docker, e controllabili, con l'installazione di Zabbix.

Installazione Docker

Le macchine hanno Ubuntu installato, l'installazione di Docker e le dipendenze avviene con apt, il gestore di pacchetti delle distribuzioni Debian.

Il file del playbook si chiama “`docker_install.yml`” e può essere visto nelle figure 6.5 e 6.6.
I task presenti svolgono altri compiti legati all’uso di Docker, oltre alla sua installazione, in ordine:

1. Installare le dipendenze di Docker e i pacchetti di Python perché Ansible possa connettersi a Docker
2. Aggiungere il repository di Docker a quelli controllati da apt; per mantere Docker all’ultima versione.
3. Installare Docker e configurarlo; Cambiando il metodo di salvataggio dei log dei container, con uno più efficiente, e controllando che l’utente “ubuntu” faccia parte del gruppo “docker”, per lanciare comandi senza diritti di root.

Il playbook viene lanciato con il comando “`ansible-playbook -i hosts.yml docker_install.yml`”.

```yaml
---
- import_playbook: "/ansible/swarm/load_maas_machines.yml"
- name: Ansible script to install Docker (and configure apt)
  hosts: swarm # execute only on swarm nodes
  strategy: free
  become: true

  tasks:
    - name: Install docker dependencies
      apt:
        pkg:
          - ca-certificates
          - curl
          - gnupg
          - lsb-release
        state: latest
        cache_valid_time: 86400

    - name: Install python and pip
      apt:
        pkg:
          - python3
          - python3-pip
        state: latest

    - name: Add Docker GPG apt Key
      apt_key:
        url: https://download.docker.com/linux/ubuntu/gpg
        state: present

    - name: Add Docker Repository
      apt_repository:
        repo: deb https://download.docker.com/linux/ubuntu {{ ansible_lsb.codename }} stable
        state: present

    - name: Stop Docker service
      service:
        name: "docker"
        state: "stopped"
      ignore_errors: true
      notify: Restart Docker service
```

Figura 6.5: Prima parte del file "docker_install.yml" per l'installazione e la configurazione di Docker

```
43    - name: Install docker-ce
44      apt:
45        pkg:
46          - docker-ce
47          - docker-ce-cli
48          - containerd.io
49          - docker-buildx-plugin
50          - docker-compose-plugin
51        state: latest
52        autoclean: true
53        update_cache: true
54
55    - name: Set default docker loggin driver to "local"
56      copy:
57        dest: "/etc/docker/daemon.json"
58        force: false
59        content: |
60          {
61            "log-driver": "local",
62            "log-opts": {
63              "max-size": "10m"
64            }
65          }
66      notify:
67        - Restart Docker service
68
69    - name: Install Docker Module for Python
70      become: false
71      pip:
72        name:
73          - docker
74          - jsondiff
75        extra_args: --user
76
77    - name: Ensure group "docker" is present
78      group:
79        name: docker
80        state: present
81
82    - name: Add user to docker group (in order to use docker without root access)
83      user:
84        name: "{{ ansible_user }}"
85        groups: docker
86        append: yes
87
88  handlers:
89    - name: Restart Docker service
90      service:
91        name: "docker"
92        state: "restarted"
93
```

Figura 6.6: seconda parte del file "docker_install.yml" per l'installazione e la configurazione di Docker

Installazione Zabbix

L'installazione dell'agent di Zabbix richiede solo tre passaggi: l'aggiunta del repository di Zabbix ad apt, l'installazione dell'agente tramite apt e la sua impostazione per avviarsi all'avvio del sistema; i task si trovano nel file "`zabbix_agent_install.yml`".

6.3 Avvio di Docker Swarm e Portainer

Completate le installazioni vengono avviati: lo swarm, connettendo tra loro i nodi del cluster, e Portainer, usando il compose file fornito dagli sviluppatori stessi.

Avvio Docker Swarm

Il playbook per l'avvio di swarm si trova nel file "`swarm_init.yml`".
I nodi con cui lavora, cioè quelli del cluster, sono caricati dinamicamente dalla lista dei nodi di MaaS, e ordinati nei gruppi dedicati, come spiegato sopra.

Il playbook comprende due "*play*", due fasi: l'**inizializzazione** dello swarm e la connessione (**join**) delle altre macchine.
La prima play può essere vista nella figura 6.7 mentre la seconda nella figura 6.8.
L'inizializzazione avviene su una sola macchina, quella aggiunta al gruppo "master", questa sarà uno dei manager dello swarm.
Dopo l'inizializzazione vengono creati due codici ("*token*") per l'accesso degli altri nodi allo swarm; servono per accedere come manager o come worker.
I codici vengono salvati in una variabile globale di Ansible e accessi nella seconda fase.
Nella seconda fase le macchine appartenenti al grupppo "manager" dell'inventario si collegano allo swarm come manager, le altre come worker, usando i *token* giusti.

il playbook viene lanciato con il comando "`ansible-playbook -i hosts.yml swarm_init.yml`".

```yaml
---
- import_playbook: "/ansible/swarm/load_maas_machines.yml"
# play 1: start swarm on master node
- name: Init docker swarm on host controller
  hosts: master
  tags: init

  tasks:
    - name: debug line for docker daemon address
      debug:
        msg:
          - "{{ inventory_hostname }}"
          # - "{{ docker_host }}"
        verbosity: 2

    - name: Start Docker daemon
      service:
        name: "docker"
        state: "started"
      become: truea

    - name: Initialise swarm in master node
      community.docker.docker_swarm:
        state: present
      register: swarm_init_rt

    - name: Set join token for worker nodes as Ansible fact
      set_fact: worker_token="{{ swarm_init_rt.swarm_facts.JoinTokens.Worker }}"

    - name: Set join token for manager nodes as Ansible fact
      set_fact: manager_token="{{ swarm_init_rt.swarm_facts.JoinTokens.Manager }}"

    - name: Debug line for join tokens
      debug:
        msg:
          - "manager node join token: {{ manager_token }}"
          - "worker node join token: {{ worker_token }}"
        verbosity: 2
```

Figura 6.7: Prima play del file "swarm_init.yml" per l'avvio dello Swarm Docker.

```
40 # play 2: connect managers and workers to the swarm
41 - name: Connect nodes to the swarm
42   hosts: swarm,!master # on all nodes of the swarm except first manager
43   tags: join
44
45   tasks:
46     - name: Debug line for join tokens (from other nodes)
47       debug:
48         msg:
49           - "manager node join token: {{ hostvars[groups['master'][0]]['manager_token'] }}"
50           - "worker node join token: {{ hostvars[groups['master'][0]]['worker_token'] }}"
51           - "master node advertise address: {{ hostvars[groups['master'][0]]
   ['inventory_hostname'] }}"
52         verbosity: 2
53
54     - name: Connect manager nodes to the swarm
55       community.docker.docker_swarm:
56         state: join
57         join_token: "{{ hostvars[groups['master'][0]]['manager_token'] }}"
58         remote_addrs: "{{ hostvars[groups['master'][0]]['inventory_hostname'] }}"
59       when: "'manager' in group_names"
60
61     - name: Connect worker nodes to the swarm
62       community.docker.docker_swarm:
63         state: join
64         join_token: "{{ hostvars[groups['master'][0]]['worker_token'] }}"
65         remote_addrs: "{{ hostvars[groups['master'][0]]['inventory_hostname'] }}"
66       when: "'worker' in group_names"
67
```

Figura 6.8: Seconda play del file "swarm_init.yml" per la connessione dei nodi allo Swarm Docker.

Avvio Portainer

A questo punto dell'esecuzione i nodi sono connessi allo swarm; quindi basta avviare uno stack Docker per far si che i container vengano avviati sui nodi.

I file per la gestione di Portainer sono: "`portainer_stack_deploy.yml`" e "`portainer_stack_purge.yml`".
Nel primo viene: scaricato il compose file di Docker dal sito di Portainer e poi viene usato per avviare i suoi servizi.
Nel secondo viene fermato lo stack creato nel file precedente.

Dopo l'avvio il server di Portainer può essere accesso sulla porta 9443 da un browser web, connettendosi a uno qualunque dei nodi, grazie alla *routing mesh* di swarm che reindirizza le richieste.
Dalla GUI di Portainer verranno avviati gli altri programmi, come Hadoop, ma della preparazione dello stack si occuperà il collega citato.

6.4 Modifiche finali

Vengono ora spiegati alcuni cambiamenti e configurazioni aggiuntive per migliorare il funzionamento del sistema.
Queste sono state svolte un unica volta dopo l'avvio del sistema.

Assegnamento label ai nodi

Docker permette di assegnare delle "label" ai nodi di uno swarm.
Le label sono etichette, coppie chiave-valore, che permettono di selezionare i nodi e quindi distinguere le macchine le une dalle altre.
Alcune label sono create da Docker engine stesso, indicano alcune caratteristiche delle macchine, come ad esempio il numero di CPU o il sistema operativo.

Le label possono essere referenziate nei file compose di uno stack Docker Swarm tramite le keyword: "`placement`" e "`constraints`"; se una label rispetta il valore specificato nel file compose, allora i container di un certo servizio possono essere avviati sul nodo legato alla label.

Nel progetto le label vengono assegnate ai nodi tramite la GUI di Portainer; sono state usate per: distinguere i nodi fisici da quelli virtuali durante lo sviluppo del progetto, e conoscere quali nodi hanno disponibile un disco fisico su cui salvare i dati.
Con questi accorgimenti si può regolare su quali macchine vengono avviati i singoli task.

Uso di NFS

Nel progetto abbiamo riscontrato il problema di mantenere una memoria persistente, accessibile a più nodi, per programmi o servizi esterni ad Hadoop, che invece usa il suo file system distribuito.
Questo problema nasce dal fatto che i nodi del cluster non possono essere considerati persistenti o sicuri: potrebbero essere spenti, riaccesi o aggiornati per sperimentare diverse configurazioni.

La soluzione ci ha portato alla creazione di una share NFS sul nodo di controllo.
Essendo il nodo di controllo una M.V., difficilmente avrà problemi di stabilità, quindi i dati salvati nella share saranno (virtualmente) sempre raggiungibili.

Lavorando con container possiamo sfruttare le funzioni dei volumi Docker.
Queste permettono di connettere dei volumi non locali alla macchina, ma accessibili attraverso la rete, e per farlo viene usato un driver di Docker per NFS.

I volumi collegati ad NFS sono stati usati per il container di Mosquitto.
In questo modo, se il container dovesse essere riavviato su una macchina diversa dalla prima, i dati trovati dal nuovo container saranno gli stessi scritti dal primo.

Virtual server per accesso ai servizi

I nodi del cluster sono connessi tra loro in una rete privata, l'accesso alla rete può essere fatto attraverso una VPN, usando le credenziali di ateneo; nella figura 6.9 può essere vista l'architettura della rete.

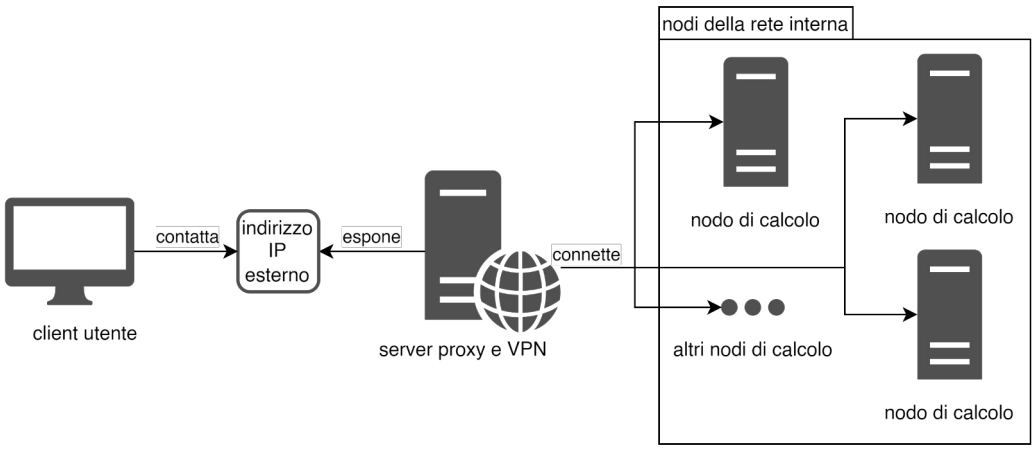

Figura 6.9: Struttura della connessione ai nodi della rete del cluster. un server esterno fa da Proxy per i nodi interni.

Gli utenti dei servizi in esecuzione sul cluster devono poter accedere ai servizi senza utilizzare la VPN.
La soluzione è fornire un indirizzo IP, raggiungibile dall'esterno, tramite cui connettersi ai servizi.
I servizi in questione sono ad esempio la GUI di Portainer o il cluster di Hadoop.
Fortunatamente la *routing mesh* di Swarm permette di comunicare con un servizio tramite uno qualunque dei nodi, indipendentemente dal nodo su cui sta eseguendo; basta riuscire a contattare un nodo del cluster Swarm dall'esterno per poter accedere ai programmi.

Il firewall che controlla la VPN, per accedere alla rete dello stack, permette di creare quello che definisce un "**virtual server**" (server virtuale).
Il firewall espone un indirizzo IP esterno e ascolta le connessioni in ingresso all'indirizzo.
Può essere impostato per inoltrare le connessioni ad un indirizzo IP interno alla rete in base alla porta contattata esternamente.
In questo modo il firewall fa da "proxy", da tramite, per l'accesso ai servizi e fa corrispondere ad una certa porta una coppia indirizzo IP – porta nella rete interna.

Esternamente alla rete gli utenti vedono un unico indirizzo ma le chiamate vengono reindirizzate ai nodi dello swarm.
Accedendo esternamente alla porta “9443”, all’indirizzo del proxy, si accede in realtà a Portainer.

Capitolo 7
Valutazione risultato e prove svolte

In questo capitolo vengono analizzati i risultati ottenuti e elencati alcuni dei test svolti per controllare la stabilità del sistema.

L'analisi dei risultati procede considerando i requisiti del sistema espressi nei capitoli precedenti.
Per ogni requisito viene detto in che modo, con quali programmi, si è cercato di rispettarlo.

Alcuni dei test svolti sono elencati nella seconda parte del capitolo.
I test hanno richiesto la ricerca di alcune soluzioni ai problemi riscontrati, e talvolta alcune modifiche all'architettura del progetto.

7.1 Valutazione risultato ottenuto

Segue la valutazione del risultato ottenuto, nella creazione del progetto, verificando in che grado i requisiti, imposti nel Capitolo 3, sono stati rispettati; Sono stati divisi in requisiti: "di utilizzo" e "di gestione".

7.1.1 Requisiti di utilizzo

Sono stati definiti "di utilizzo" i requisiti da garantire agli utenti del sistema, sono due, cioè: uso dei programmi e fruibilità dei servizi.
Si ritiene di aver raggiunto un buon livello di accessibilità per gli utenti del sistema per i motivi seguenti.

Uso dei programmi

Il requisito richiede di preparare il sistema all'installazione dei software necessari.
Grazie all'infrastruttura fornita da Docker Swarm possono essere avviati i programmi desiderati; si richiede solo la scrittura di Compose file per definire stack di servizi Docker, che verranno lanciati attraverso la GUI di Portainer.

L'uso di Docker Swarm non limita gli utenti a soli software distribuiti, in esecuzione su più macchine, ma lascia la possibilità di usare programmi sui singoli nodi.
Inoltre selezionando le label, aggiunte alle macchine tramite Portainer, si può scegliere su quali nodi avviare i programmi.

L'uso di container consente di variare facilmente la versione dei programmi in esecuzione.
Può essere utile sia per aggiornare i servizi, sia per sperimentare con diverse versioni degli stessi.

Fruibilità

Si può accedere ai programmi, e a eventuali programmi che verranno aggiunti in futuro, agilmente grazie al server proxy.
Dal punto di vista degli utenti basta collegarsi ad un unico indirizzo IP.
Il server che espone l'indirizzo verrà impostato per reindirizzare i messaggi al programma giusto tra quelli che saranno in esecuzione sul cluster.
La gestione avviene in modo in modo trasparente agli utenti, una volta che i tecnici avranno impostato il server proxy.

7.1.2 Requisiti di gestione

I requisiti "di gestione" riguardano le caratteristiche da garantire ai tecnici dell'università per facilitare la gestione del sistema; sono tre, e sono i requisiti di: monitoraggio, avvio e installazione, manutenzione.
I risultati da valutare riguardano il primo avvio e il mantenimento del sistema in funzione.

Per controllare che i requisiti di gestione siano stati rispettati, vengono elencati i tratti principali del sistema; sono anche stati eseguiti alcuni test sui tempi richiesti per l'esecuzione.

Avvio e installazione

Il requisito riguarda il punto cardine del progetto: l'avvio per lo più automatico del sistema e l'installazione non manuale dei programmi.

I due strumenti che hanno permesso di raggiungere un buon grado di automazione sono MaaS e Ansible; MaaS per la preparazione delle macchine, Ansible per l'installazione dei programmi e l'avvio del sistema.
MaaS, una volta installato, si rivela facile da usare e gli script di Ansible sono molto versatili per diversi compiti.

I punti principali durante il primo avvio del sistema sono l'installazione del sistema operativo e di Docker sui nodi; di queste due fasi sono stati provati i **tempi di esecuzione**.
Gli altri passaggi dell'installazione sono trascurabili dal punto di vista del tempo richiesto.
I test sono stati svolti con una delle macchine virtuali usate in fase di progettazione quindi, rispetto alle macchine dedicate su cui eseguiranno i programmi, un sistema poco potente.
Il processo di installazione del S.O. ha richiesto circa **6 minuti**; prima di questo serve aggiungere la macchina all'inventario ed eseguire il commissioning, per raccoglierne le informazioni.
L'installazione di Docker e delle dipendenze, su macchina appena preparata, ha richiesto circa **3 minuti e 30 secondi**.

I tempi misurati dimostrano buoni risultati.
Presi singolarmente l'installazione dei sistemi operativi e l'esecuzione dei playbook eseguono in parallelo; quindi il primo avvio può essere svolto in tempo molto breve.
Per dare una stima dei tempi: in circa un'ora si potrebbe avere l'intero sistema funzionante.

Monitoraggio

Il requisito di monitoraggio richiede che i malfunzionamenti siano notificati in modo automatico o che siano rilevabili semplicemente.

Per quanto riguarda il controllo dello stato dei nodi i due strumenti che lo permettono sono Zabbix e MaaS.
Zabbix, inviando notifiche in caso di necessità, non richiede un controllo frequente delle macchine da parte dei tecnici.
La GUI di MaaS, quando necessario, aiuta a verificare lo stato generale dei nodi.

Il monitoraggio dei container in esecuzione sul cluster può essere svolto con Portainer.
Dalla sua interfaccia web si possono visualizzare velocemente lo stato dei servizi e dei demoni Docker presenti sui nodi.

Manutenzione

Il requisito di manutenzione richiede di fornire degli strumenti ai tecnici dell'università che possano, qual ora si verificassero malfunzionamenti, aiutare a ripristinare l'esecuzione del sistema.

In situazione estrema, cioè in caso di un "disastro" per cui le macchine non possono più funzionare correttamente, MaaS garantisce la possibilità di un'installazione pulita del sistema in tempo breve.

Gli stessi playbook di Ansible usati per le installazioni nella fase iniziale, possono essere usati per ripristinare lo stato di esecuzione del sistema in caso di piccoli malfunzionamenti, o per aggiornare i programmi.
I playbook, grazie alla proprietà di idempotenza, se avviati più volte svolgono sui nodi solo le operazioni necessarie.

7.2 Prove svolte

Durante lo sviluppo del progetto sono stati svolti alcuni test; utili a verificare la stabilità del sistema e a controllare le conseguenze di possibili problemi.
I risultati hanno portato alcune modifiche a parti della struttura del progetto.

Di seguito sono riportati i test e i problemi riscontrati, le conseguenze e le soluzioni.
I test riguardano: la scelta del numero di nodi manager nel cluster Swarm, cosa succede se il Server di Portainer viene disconnesso e cosa succede a seguito di un errore di MaaS

Numero nodi Manager Swarm

In questo test abbiamo verificato cosa succede se uno dei nodi uniti allo swarm come manager dovesse disconnettersi senza preavviso.
abbiamo svolto la prova con le macchine virtuali di test, scollegando la rete di una di queste, tra quelle unite allo Swarm come nodo manager.

Se alla disconnessione, nello Swarm Docker sono presenti solo uno o due nodi manager, il cluster va in stallo.
Nel primo caso nessun nodo gestiste più gli altri, mentre nel secondo il nodo rimasto cerca di contattare l'altro ma non riesce, rimanendo in attesa indefinitamente.

Normalmente per slegare un nodo dal cluster Swarm va lanciato un comando che notifichi gli altri nodi della disconnessione.
Se questo non avviene gli altri nodi Manager non "sanno" che il nodo disconnesso non può più essere raggiunto; i nodi cercheranno comunque di comunicare con la macchina disconnessa ma non verranno ricevute risposte.

In un cluster Swarm le decisioni del cluster vengono prese per "votazione maggioritaria". Queste scelte possono riguardare ad esempio: l'espulsione di un nodo o lo spostamento di uno container da un nodo worker ad un altro.
La disconnessione di un nodo Manager può diventare un problema per le scelte di gestione del cluster: nella votazione va raggiunto un numero di voti pari alla metà del numero dei manager più uno.
Per questo si consiglia di avere, se possibile, almeno tre nodi manager nel cluster.

Se, per esempio, in un cluster Swarm sono presenti due nodi manager, e uno dei due si disconnette senza preavviso, per poter poter rimuovere il nodo irraggiungibile servono i voti di entrambi; essendo solo due, il secondo nodo disconnesso non risponde, di coseguenza il cluster va in stallo.

Per evitare problemi in un cluster Docker Swarm si consiglia di avere un **numero dispari** di nodi manager.
La scelta dei nodi nel progetto viene fatta all'interno del playbook "`load_maas_machines.yml`", che "elegge" un numero dispari di Manager.

Disconnessione nodo Portainer

In questa prova abbiamo controllato cosa succede se il nodo su cui esegue il server di Portainer si disconnette dal cluster Docker Swarm.

La prova è stata svolta spegnendo il computer, unito allo swarm come nodo manager, su cui stava eseguendo il container.

Grazie al compose file di Portainer il server riparte su un altro dei nodi manager presenti nello Swarm, quindi la GUI rimane accessibile.
Tuttavia, non essendo il server di Portainer progettato per essere spostato dinamicamente, il riavvio porta a due problemi: la perdita dei dati del Portainer server e problemi di connessione tra i Portainer agent e il server.

Il Server di Portainer salva i dati in un volume Docker, il volume si trova in una directory locale alla macchina su cui esegue.
Quando, a seguito della disconnessione, il container viene spostato su un nodo diverso, i dati non sono più accessibili, quindi Portainer andrebbe riconfigurato come dopo al primo avvio.

I Portainer agent in esecuzione sui nodi del cluster sono impostati per comunicare con un unico Portainer server.
Quando il server di Portainer viene spostato, gli agent non accettano più le comunicazioni del server; dalla GUI non si riescono a controllare i nodi dello Swarm Docker o a vederne lo stato.

Per risolvere il problema della perdita dei dati, il volume Docker può essere spostato sulla share NFS del nodo di controllo, così che ovunque esegue Portainer accede agli stessi dati.
Per risolvere il problema degli agent di Portainer, vanno riavviati i container; va lanciato un comando da una shell di un nodo Manager dello Swarm: “`docker service update --image portainer/agent:'versione' --force Portainer_agent`”.
Con “`versione`” si intende il codice della versione installata di Portainer.
Purtroppo non sono stati trovati metodi automatici per risolvere il problema.

Errore di MaaS

In questa prova abbiamo verificato cosa succede se MaaS dovesse essere installato da capo.
I dati sul disco del nodo di controllo sono stati riportati a prima dell’installazione di MaaS.
Essendo il nodo di controllo una macchina virtuale, abbiamo usato uno “*snapshot*”, un salvataggio dello stato del disco, per tornare allo stato precedente.

Dopo aver reinstallato e impostato MaaS le macchine del cluster vanno aggiunte nuovamente alla lista, questa fase di commissioning non causa problemi, richiede solo un riavvio dei nodi.
Una volta eseguito il commissioning le macchine fanno parte della lista, ma MaaS non può sapere che su queste era già stato installato il S.O., per cui rimangono in stato “ready” invece che “deployed”.

La soluzione trovata prevede di eseguire una query di aggiornamento sul database che MaaS usa per mantenere i dati: “`UPDATE maasserver_node SET status='6' WHERE hostname = 'nome macchina'`”.
La query seleziona una macchina per volta e ne aggiorna lo stato; “`nome macchina`” va appunto sostituito con il nome del computer nell’inventario di MaaS.

La soluzione purtroppo richiede di eseguire alcuni passaggi manuali ma permette comunque di tornare ad uno stato funzionale del sistema.
L’abbiamo presa come una soluzione accettabile considerando che MaaS esegue su una macchina virtuale su un server fidato; rendendo un suo errore uno scenario poco probabile.

Capitolo 8
Conclusione

Il progetto nasceva per creare un cluster Docker Swarm, su cui eseguire un cluster Hadoop, dedicato al calcolo distribuito.
Si può dire che la tesi, dedicata alla prima parte del progetto, ha raggiunto, nel complesso, uno stato soddisfacente.

Siamo riusciti a creare un sistema per l'avvio automatico del cluster, che lavora attraverso la rete locale dal nodo di controllo, grazie a MaaS e Ansible; Ansible ha permesso di creare uno Swarm Docker sul quale avviare i servizi.
Sono poi stati resi disponibili programmi e strumenti, grazie ai quali i colleghi potranno sviluppare la seconda parte di progetto: Portainer, come interfaccia comoda per gestire lo Swarm e i container, Zabbix, per il monitoraggio da parte dei tecnici, e NFS, come punto di salvataggio dei.

Il sistema è stato creato insieme al dott. Ciro Barbone, solo grazie ai continui confronti sullo sviluppo, nonché ai suoi consigli.
Un ringraziamento speciale va al Prof. Vittorio Ghini, che ha proposto il progetto, per la sua disponibilità; Ogni studente, che lo abbia conosciuto, può confermare la grande dedizione che mette nella sua professione.

La tesi ha permesso di sperimentare aspetti dell'informatica diversi da quelli visti durante le lezioni; aspetti più vicini al "mondo" sistemistico rispetto alla programmazione, come l'uso di macchine virtuali e la configurazione dei programmi.

Bibliografia

[1].....«Apache Hadoop». https://hadoop.apache.org/ (consultato 12 luglio 2023).
[2].....«What is Hadoop and What is it Used For?», *Google Cloud*. https://cloud.google.com/learn/what-is-hadoop (consultato 12 luglio 2023).
[3].....«PoweredBy - HADOOP2 - Apache Software Foundation», *cwiki.apache.org*. https://cwiki.apache.org/confluence/display/hadoop2/PoweredBy (consultato 12 luglio 2023).
[4]....S. Ghemawat, H. Gobioff, e S.-T. Leung, «The Google file system», in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, in SOSP '03. New York, NY, USA: Association for Computing Machinery, ott. 2003, pp. 29–43. doi: 10.1145/945445.945450.
[5].....J. Dean e S. Ghemawat, «MapReduce: simplified data processing on large clusters», *Commun. ACM*, vol. 51, fasc. 1, pp. 107–113, gen. 2008, doi: 10.1145/1327452.1327492.
[6].....«Welcome to Apache Pig!» https://pig.apache.org/ (consultato 12 luglio 2023).
[7].....«Apache Hive». https://hive.apache.org/ (consultato 12 luglio 2023).
[8].....«Apache Spark™ - Unified Engine for large-scale data analytics». https://spark.apache.org/ (consultato 12 luglio 2023).
[9].....«Apache Kafka», *Apache Kafka*. https://kafka.apache.org/ (consultato 12 luglio 2023).
[10] «What is Virtualization? | IBM». https://www.ibm.com/topics/virtualization (consultato 12 luglio 2023).
[11].....«Containerization Explained | IBM». https://www.ibm.com/topics/containerization (consultato 12 luglio 2023).
[12].....«Open Container Initiative», *(OCI) Open Container Initiative*. https://opencontainers.org/ (consultato 12 luglio 2023).
[13]..«Docker Docs: How to build, share, and run applications», *Docker Documentation*, 11 luglio 2023. https://docs.docker.com/ (consultato 12 luglio 2023).
[14]«Docker Hub Container Image Library | App Containerization». https://hub.docker.com/ (consultato 12 luglio 2023).
[15].....«Docker Compose overview», *Docker Documentation*, 11 luglio 2023. https://docs.docker.com/compose/ (consultato 12 luglio 2023).
[16].....«Swarm mode overview», *Docker Documentation*, 11 luglio 2023. https://docs.docker.com/engine/swarm/ (consultato 12 luglio 2023).
[17].....«Metal as a Service», *MAAS*. https://maas.io/ (consultato 26 luglio 2023).
[18].....«Ansible is Simple IT Automation». https://www.ansible.com/ (consultato 26 luglio 2023).
[19]....«Portainer: Docker and Kubernetes Management Platform». https://www.portainer.io (consultato 26 luglio 2023).
[20].....«Zabbix :: The Enterprise-Class Open Source Network Monitoring Solution». https://www.zabbix.com/ (consultato 26 luglio 2023).
[21].....«MQTT - The Standard for IoT Messaging». https://mqtt.org/ (consultato 26 luglio 2023).
[22]«24. Network Protocols — SNP, PXE, BIS and HTTP Boot — UEFI Specification 2.10 documentation».
https://uefi.org/specs/UEFI/2.10/24_Network_Protocols_SNP_PXE_BIS.html#pxe-base-code-protocol (consultato 12 agosto 2023).
[23].....«The Official YAML Web Site». https://yaml.org/ (consultato 13 agosto 2023).

[24].....«Zabbix Agent». https://www.zabbix.com/documentation/current/en/manual/concepts/agent (consultato 21 agosto 2023).
[25].....«Linux NFS - Main Page», 7 giugno 2022. http://linux-nfs.org/wiki/index.php?title=Main_Page&oldid=5834 (consultato 23 agosto 2023).
[26].....«Kubernetes», *Kubernetes*. https://kubernetes.io/ (consultato 21 agosto 2023).
[27].....«Enterprise Kubernetes Management | Rancher», *Rancher Labs*. http://www.rancher.com (consultato 23 agosto 2023).
[28].....«Red Hat OpenShift enterprise Kubernetes container platform». https://www.redhat.com/en/technologies/cloud-computing/openshift (consultato 23 agosto 2023).
[29]…© 2020 TopMonks s.r.o, «Swarmpit». https://swarmpit.io (consultato 23 agosto 2023).
[30].....«Ceph.io — Home». https://ceph.io/en/ (consultato 25 agosto 2023).
[31]. .«file synchronization tool using librsync and current state databases LINBIT/csync2», *GitHub*. https://github.com/LINBIT/csync2 (consultato 25 agosto 2023).
[32]…«Lsyncd (Live Syncing Daemon) synchronizes local directories with remote targets», *GitHub*. https://github.com/lsyncd/lsyncd (consultato 23 agosto 2023).
[33].....«JMESPath Specification — JMESPath». https://jmespath.org/specification.html (consultato 4 settembre 2023).