ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

CEPH, UNA PIATTAFORMA DI ARCHIVIAZIONE DISTRIBUITA PER APPLICAZIONI CONTAINERIZZATE

Elaborato in

Virtualizzazione e Integrazione di Sistemi

Relatore
Vittorio Ghini

Presentata da
Michele Michelotti

Anno Accademico 2021/2022

Indice

1. Sommario

In questa tesi vogliamo analizzare il funzionamento della piattaforma di archiviazione distribuita chiamata Ceph. Faremo una breve introduzione in cui parleremo del sempre più crescente utilizzo dei micro-servizi, soprattutto in ambito cloud, e dell'ormai consolidata tecnologia dei container. Mostreremo le loro funzionalità concentrandoci su Docker, uno dei principali container engine, e di Docker Swarm, orchestrator di applicazioni containerizzate in ambienti Linux, cercando anche di portare alla luce le loro problematiche e necessità. Spiegheremo cosa si intende per Software Defined Storage, un'architettura moderna in cui lo spazio di storage viene astratto e definito via software tralasciando all'utente finale tutta la complessità dei dispostivi hardware sottostanti. Passeremo infine a parlare di Ceph, uno dei principali Software Defined Storage che promette diversi tipi di interfacce di utilizzo, per ogni tipo di applicazione e delle sue esigenze. Guarderemo in dettaglio la struttura del cluster e i vari servizi che lo compongono: Manager, OSD, MDS e MON, analizzando i principali algoritmi che sono alla base del suo funzionamento come Paxos e CRUSH. Il primo viene utilizzato in ambienti distribuiti per mantenere la consistenza dei dati tramite un sistema di accettazione tra i nodi del cluster; il secondo, invece, è un algoritmo pseudo-casuale che si occupa di dividere i dati tra i vari nodi del cluster e della loro ridondanza. Parleremo in dettaglio dei tre tipi di interfacce che Ceph mette a disposizione dei propri utilizzatori: ad oggetti, a blocchi ed a file, cercando di mostrare le loro caratteristiche e le loro differenze. Creeremo una piccola implementazione di Ceph, molto basilare, su cui monteremo dei dischi per creare uno spazio di storage ed un file system CephFS da utilizzare su una macchina client. Infine, verrà fatto un piccolo confronto con altri software SDS sul mercato per mostrare pregi e difetti di Ceph rispetto ai suoi concorrenti.

2. Introduzione

2.1. Un mondo sempre più diretto verso i micro-servizi

Fino ai primi anni 2000, le applicazioni erano quasi tutte monolitiche, cioè erano composte da un singolo pezzo o al massimo due ma comunque indissolubili. Questa struttura, con l'utilizzo sempre più massivo del cloud e della sempre più pervasiva presenza di dispositivi connessi ad esso nella vita di tutti i giorni, non è più in grado di fornirci una soluzione accettabile per le esigenze moderne; si è vista quindi la necessità di creare una nuova architettura, più agile, facilmente mantenibile e scalabile in base alle richieste ricevute: i sistemi a micro-servizi. I micro-servizi sono un'architettura software in cui le applicazioni sono suddivise in piccoli servizi autonomi che comunicano tra loro tramite API. Essi stanno diventando sempre più popolari nel mondo dell'informatica e sono visti come una delle tendenze più importanti degli ultimi anni.

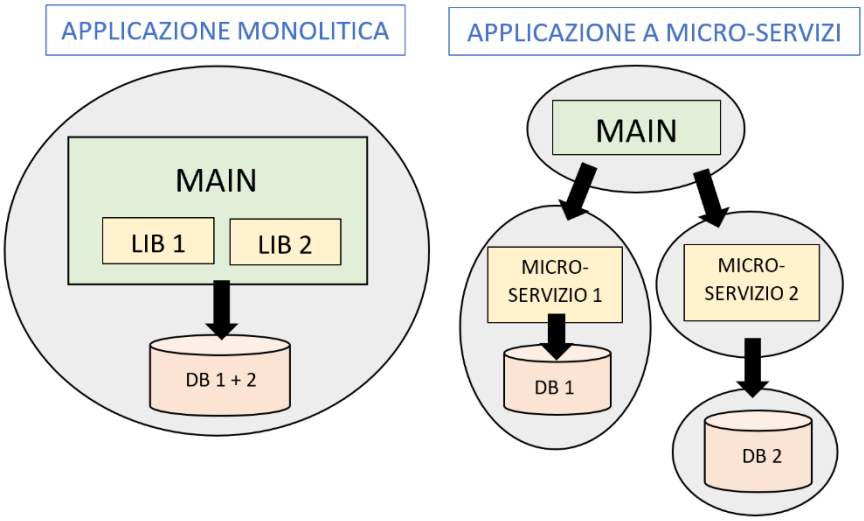

Figura 1 – architettura monolitica vs. a micro-servizi

Come si può notare in Figura 1, il concetto di base di un sistema a micro-servizi è spezzare l'applicativo in tante piccole parti, ognuna delle quali si occuperà di offrire un singolo servizio richiesto dal sistema. Poiché in un'architettura a micro-servizi è indispensabile la comunicazione tra i vari componenti, si rende necessaria l'esposizione

di interfacce software e protocolli di comunicazione diffusi e condivisi, nonché la progettazione delle applicazioni stesse affinché utilizzino solo questi standard. Nonostante però la maggior complessità nelle comunicazioni, questi sistemi offrono molti vantaggi rispetto alla vecchia applicazione monolitica:

- **Maggiore agilità e scalabilità**: ogni servizio è autonomo ed è quindi più facile aggiornare e scalare solo quelli che ne hanno bisogno, senza dover toccare l'intera applicazione;

- **Maggiore gestione e organizzazione**: i micro-servizi consentono di suddividere l'applicazione in componenti più piccoli e gestibili, semplificando la manutenzione e la gestione dell'intero sistema;

- **Maggiore flessibilità tecnologica:** ogni servizio può essere scritto in un linguaggio di programmazione differente o utilizzare un database diverso, in questo modo può essere scelta la tecnologia più adatta al compito che deve essere svolto da quel servizio;

- **Maggiore riusabilità:** i componenti di un applicativo trasformati in micro-servizi possono essere usati anche da altri applicavi;

- **Maggiore tolleranza ai guasti:** i micro-servizi sono progettati per essere resilienti ai guasti, in modo che se un servizio smette di funzionare, l'intera applicazione non viene compromessa. Inoltre, è molto più facile sostituire un micro-servizio in caso di rottura piuttosto che l'intero applicativo;

- **Migliore distribuzione:** essendo un sistema di tanti servizi autonomi tra loro non è obbligatoria la loro distribuzione su una singola macchina ma anzi, possono essere posizionati su macchine differenti.

Per migliorare ancora di più la distribuzione e la replica in base al carico di lavoro viene sempre di più utilizzato, soprattutto in ambito cloud, un particolare tipo di virtualizzazione a livello di Sistema Operativo, in grado di contenere i servizi in ambienti separati senza dover virtualizzare l'intera macchina: i container.

2.2. Container Docker e Docker Swarm

Docker è una piattaforma open source, eseguita su sistemi con kernel Linux, che consente di creare, distribuire e gestire applicazioni in contenitori isolati (chiamati container) semplificando la distribuzione delle applicazioni e la gestione dell'infrastruttura.

I container sono un tipo di virtualizzazione a livello di sistema operativo che consente di eseguire applicazioni in modo isolato e sicuro su un host condiviso. Nella virtualizzazione tradizionale basata su macchine virtuali, ciascuna di esse possiede il proprio sistema operativo, che eventualmente può essere diverso tra le varie macchine ospitate e la macchina host. L'accesso alle risorse hardware fisiche, la protezione e l'isolamento tra le macchine sono gestiti da un'entità chiamata hypervisor. La virtualizzazione a livello di sistema operativo è composta invece da un solo kernel, quello della macchina ospitante, e molteplici istanze isolate di spazi utente che possono essere avviate e spente in maniera indipendente tra loro. Il vantaggio di quest'ultimo tipo di virtualizzazione è l'utilizzo più efficiente delle risorse del sistema (basso overhead sia per il context-switch che di memoria), anche se di contro l'isolamento non potrà mai essere del tutto perfetto e non è possibile ospitare sistemi operativi differenti da quello della macchina host. I container, quindi, creano ambienti separati e isolati in cui possono essere eseguite le applicazioni; ogni container include solo le librerie, le dipendenze e le risorse del sistema necessarie per la loro esecuzione. Una volta costruito e adattato ad un servizio, un container può essere facilmente messo in esecuzione e replicato più volte anche su macchine diverse rendendo estremamente più semplice la gestione, la distribuzione e la scalabilità del micro-servizio anche su cluster di host.

Mentre Docker permette la gestione di container su un singolo host, Docker Swarm è uno strumento di orchestrazione di container che consente di creare un cluster di host Docker che possono essere gestiti come una singola entità. Offre inoltre funzionalità avanzate come il bilanciamento del carico, la gestione degli errori e la scalabilità automatica delle applicazioni in base alla domanda.

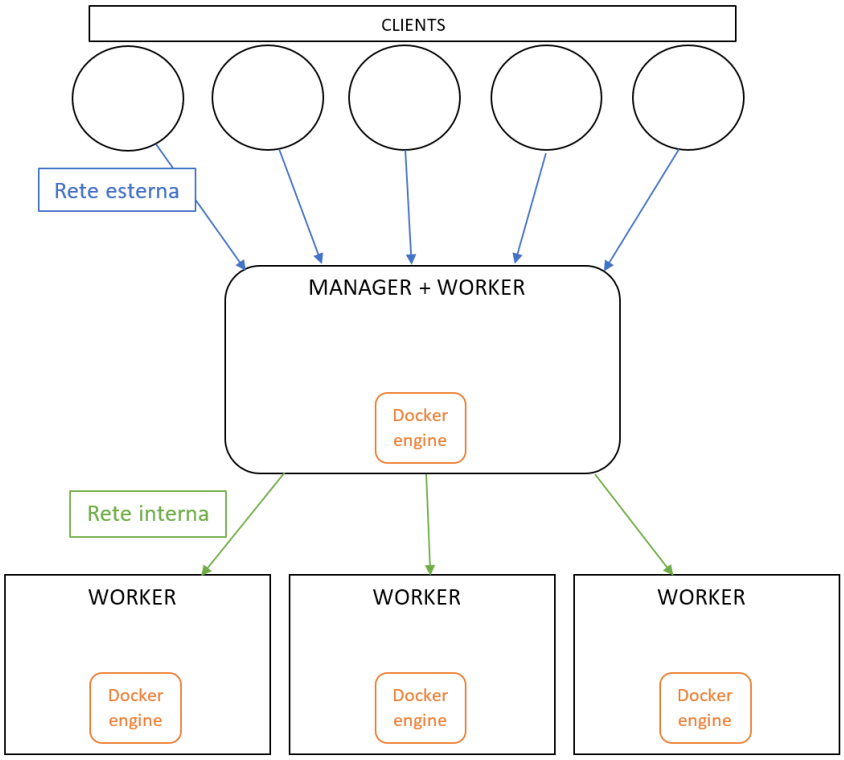

Figura 2 – Struttura di Docker Swarm

Come si può notare in Figura 2, Docker Swarm è composta da uno o più nodi (macchine) che si dividono in:

- **Manager Node:** è il punto di ingresso per gestire un cluster Docker Swarm. Ogni cluster deve avere almeno un nodo manager, che è responsabile della gestione dello stato del cluster e della distribuzione dei servizi sui nodi worker. Il nodo manager esegue il servizio di orchestrazione di Swarm, che mantiene lo stato del cluster e decide dove eseguire i servizi. Questo nodo svolge anche il ruolo di Worker;

- **Worker Node:** è un host che esegue i container Docker. Il numero di nodi worker dipende dalle esigenze dell'applicazione.

È possibile distribuire un servizio su più worker creando multiple istanze di container e ogni worker può eseguire più contenitori di uno o più servizi. Questi ultimi sono costituiti da un gruppo di container

Docker che eseguono lo stesso codice e possono essere scalati aggiungendo o rimuovendo container. Docker Swarm garantisce che il numero desiderato di container sia sempre in esecuzione, indipendentemente dallo stato dei nodi. L'istanza di un container all'interno di un servizio è chiamata Task: esso viene avviato dal nodo manager sui nodi worker disponibili quando il servizio viene eseguito.

Per le loro caratteristiche fondamentali, quali la scalabilità, portabilità e per un fattore di performance, i container necessitano di un sistema di storage esterno che possa avere a sua volta queste caratteristiche, che sappia gestire i dati in maniera sicura anche in caso di guasti, accessibile concorrentemente da diverse entità, distribuito e di facile gestione.

2.3. Software Defined Storage

Il Software Defined Storage (SDS) è un'architettura che consente di separare il software di gestione dei dati dall'hardware di archiviazione sottostante. Ciò significa che le funzionalità di archiviazione, come la creazione di volumi, la gestione delle repliche o la compressione dei dati, possono essere eseguite via software e applicate a qualsiasi hardware di archiviazione compatibile. In questo modo, il Software Defined Storage consente di creare un pool di risorse di archiviazione in grado di scalare in modo flessibile, senza dover dipendere da un singolo fornitore di hardware; ciò rende l'archiviazione più agile riducendone i costi di implementazione. L'obbiettivo principale dell'SDS, quindi, è quello di semplificare la gestione e l'allocazione delle risorse di storage, ridurre i costi, aumentare la flessibilità e migliorare l'efficienza dei data center. Nella figura seguente (Figura 3) viene mostrato un esempio di funzionamento di SDS.

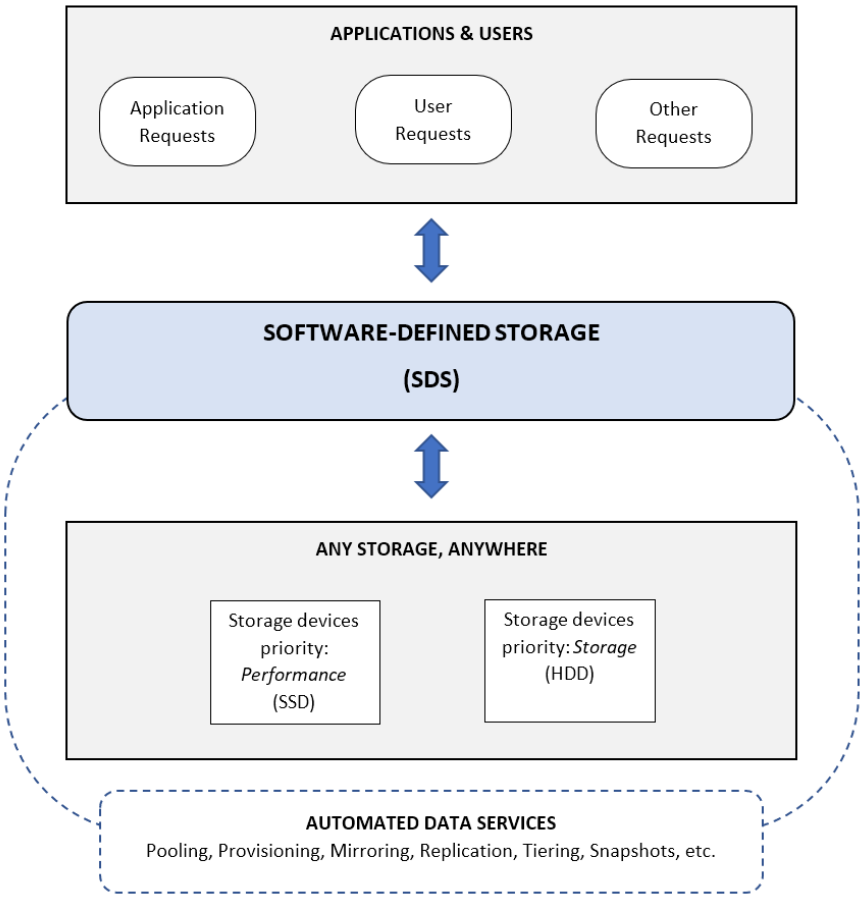

Figura 3 – Software Defined Storage

Nella maggior parte dei casi un sistema SDS deve essere dotato di:

- **Automazione**: il sistema è in grado di gestire in autonomia l'intera struttura, riducendo in questo modo la necessità dell'intervento umano. Vengono quindi automatizzate le operazioni di allocazione delle risorse di storage e del loro ridimensionamento, espandendone o riducendone la capacità. Oltre a ciò, anche il monitoraggio delle prestazioni e la risoluzione di problemi vengono gestiti direttamente dal sistema, agevolando così il contenimento dei costi. Le attività di routine possono essere automatizzate consentendo agli amministratori di concentrarsi su attività più critiche e strategiche. Può, inoltre, migliorare la scalabilità del

sistema, consentendo di gestire grandi quantità di dati e di risorse di storage in modo efficiente.

- **Interfacce standard (API)**: affinché un sistema SDS sia efficace, è importante che sia in grado di interagire con altri componenti del data center in modo uniforme. Per questo motivo, esso deve essere dotato di interfacce standard API, che consentano l'interoperabilità con altri componenti, come server, applicazioni, sistemi di gestione e dispositivi di storage. Le API permettono di semplificare l'integrazione tra questi ultimi e il sistema SDS, eliminando la necessità di sviluppare adattatori personalizzati per ognuno di essi. Ciò dà la possibilità di ridurre i costi e migliorare l'efficienza nella gestione del sistema. Inoltre, l'adozione di interfacce standard consente di migliorare l'interoperabilità tra diversi fornitori di tecnologie, fornendo maggiore flessibilità e scelta agli utenti finali nella selezione dei componenti del proprio data center.

- **Percorso dati virtualizzato**: un percorso dati virtualizzato è un'astrazione del percorso dati sottostante che consente al sistema SDS di creare un'immagine logica unificata di tutti i dispositivi di storage fisici presenti nel data center, come dischi rigidi, dispositivi di archiviazione in rete o memorie flash. Ciò significa che il percorso dati virtuale maschera le complessità del percorso dati fisico sottostante, facendo sì che il sistema SDS sia in grado di gestire dinamicamente l'allocazione delle risorse di storage in base alle esigenze dei carichi di lavoro e alle prestazioni richieste. Ad esempio, il percorso dati virtualizzato dà la possibilità di creare uno spazio di archiviazione unificato a cui accedere da diverse applicazioni o dispositivi, semplificando la gestione dei dati e migliorando l'efficienza del sistema. Esso, inoltre, consente di implementare funzionalità avanzate come la replica, la deduplicazione e la compressione dei dati in modo trasparente per le applicazioni e gli utenti finali.

- **Scalabilità**: dà la possibilità di espandere o ridurre le capacità di storage in modo dinamico e flessibile in base alle esigenze del data center. In particolare, un sistema SDS scalabile consente di aggiungere o rimuovere facilmente risorse di storage, ad esempio

dischi rigidi, dispositivi di archiviazione in rete o memorie flash, senza interrompere l'attività del sistema. Ciò significa che il sistema SDS può adattarsi in modo dinamico alle esigenze dei carichi di lavoro e delle applicazioni, garantendo prestazioni elevate e affidabilità del sistema. Oltre a ciò, un sistema SDS scalabile permette di ridurre i costi e migliorare l'efficienza del data center, in quanto favorisce un migliore utilizzo delle risorse di storage, evitando sprechi o sovradimensionamento del sistema.

- **Trasparenza**: grazie alla presenza di un percorso dati virtualizzato precedentemente descritto, un sistema SDS può definirsi trasparente poiché la complessità del sistema di storage sottostante viene nascosta da un'immagine logica unificata delle risorse. Questo meccanismo consente di semplificare l'utilizzo di queste ultime, perché sia le applicazioni che l'utente finale possono utilizzare il percorso dati virtualizzato senza dover gestire direttamente quello fisico sottostante. Ciò significa che gli utenti finali possono accedere alle risorse di storage in modo semplice e trasparente, indipendentemente dai dettagli tecnici del sistema SDS. Inoltre, la trasparenza consente di semplificare la gestione di quest'ultimo, eliminando la necessità di interventi manuali per l'allocazione delle risorse, la migrazione dei dati o la risoluzione dei problemi.

3. Ceph

3.1. Che Cosa è Ceph

Ceph [1] è un sistema di archiviazione distribuito open source che utilizza un'architettura SDS per fornire archiviazione su larga scala e altamente affidabile per applicazioni cloud e non solo. Gestisce grandi quantità di informazioni distribuendo il carico di lavoro su un diverso numero di nodi, ognuno di essi contenente una parte del dato complessivo del sistema; tutti i nodi lavorano insieme per fornire l'accesso ai dati in modo efficiente e affidabile. La scalabilità orizzontale, inoltre, consente di aggiungere facilmente nuovi nodi al cluster di storage per aumentare la capacità e le prestazioni del sistema senza dover interrompere il servizio. Ceph può essere configurato come uno storage backend per molte piattaforme di containerizzazione come Kubernetes e Docker Swarm, offrendo un'infrastruttura di archiviazione altamente scalabile, resiliente e distribuita.

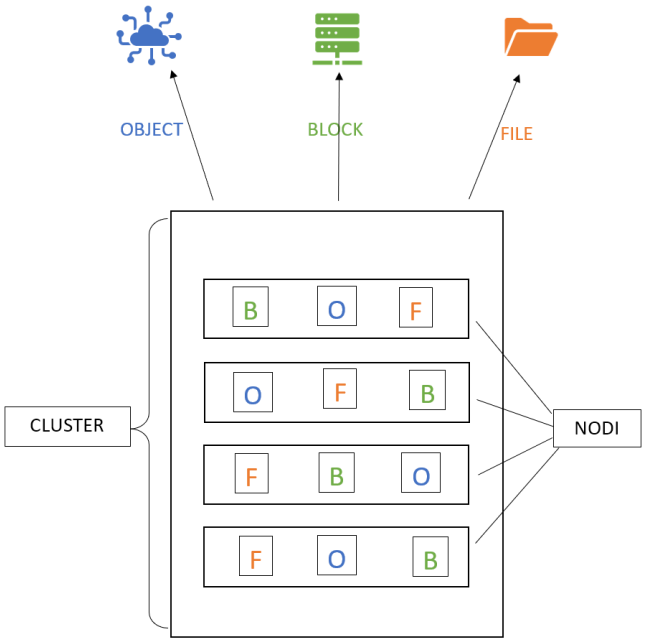

Figura 4 – Struttura di Ceph

In Figura 4 viene mostrato un esempio di struttura di Ceph. Esso può fornire servizi di archiviazione a blocchi, file e oggetti, consentendo di

archiviare informazioni in modo persistente e di condividere dati tra i container in un cluster. Ceph è altamente flessibile e personalizzabile ed è in grado di adattarsi alle esigenze specifiche di archiviazione delle diverse applicazioni containerizzate. Questo è necessario perché i container vengono spesso spostati e riavviati su diverse macchine, rendendo quindi difficile mantenere la persistenza dei dati. Ceph fornisce una soluzione di archiviazione distribuita che consente di accedere ai dati da qualsiasi container e di garantirne la persistenza anche in caso di arresto o avvio di una macchina. Nei capitoli successivi esamineremo in dettaglio la sua struttura e il suo funzionamento per poi analizzare come poterlo utilizzare su Docker Swarm. Il seguente studio si basa sulla documentazione ufficiale rilasciata dagli sviluppatori; essendo un progetto open source è possibile trovare varie versioni di Ceph, personalizzate da terze parti in base alle loro necessità.

3.2. Architettura di Ceph

L'architettura di Ceph può essere suddivisa in tre componenti principali:

1. **RADOS (Reliable Autonomic Distributed Object Store)**: è la componente di archiviazione di basso livello di Ceph. RADOS fornisce uno spazio di archiviazione distribuito, affidabile e scalabile attraverso la creazione di un cluster di server di archiviazione. I dati sono suddivisi in oggetti (object) e replicati su più nodi per garantire affidabilità.

2. **librados**: è la libreria client di basso livello di Ceph, che consente alle applicazioni di accedere al cluster RADOS. Esso offre funzionalità avanzate come la lettura/scrittura parallela di oggetti e la gestione delle repliche.

3. **CephFS**: è un sistema di file distribuito basato su RADOS. CephFS permette di creare un filesystem distribuito scalabile e affidabile.

Oltre a ciò, Ceph utilizza un'architettura a oggetti, dove ogni oggetto è identificato da un ID univoco e può essere indirizzato e recuperato tramite quest’ultimo.

3.2.1. I nodi Ceph

In Ceph, un nodo è un singolo sistema informatico che funge da membro del cluster. Un nodo di Ceph può eseguire uno o più ruoli all'interno del cluster: il ruolo di monitor, che ha il compito di tenere traccia dello stato del cluster e di mantenere la sua configurazione; quello di manager, che gestisce le operazioni del cluster; quello di OSD, responsabile dell'archiviazione e del recupero dei dati, ed infine il ruolo di MDS per la gestione dei metadati associati ai dati archiviati.

Ruolo OSD (Object Storage Daemon)

Un nodo che svolge il ruolo di OSD è responsabile dell'archiviazione dei dati sul disco locale e della gestione dell'accesso ad essi da parte delle applicazioni client. Per garantire la disponibilità e l'affidabilità dei dati, essi vengono replicati automaticamente su più nodi OSD; questo significa che in un cluster possono coesistere più nodi con questo ruolo e che ciascuno di essi ha la responsabilità di archiviare e recuperare una parte specifica dei dati. In questo modo, Ceph garantisce che le informazioni siano sempre disponibili e accessibili, anche in caso di guasti o malfunzionamenti di uno o più nodi. Quando un client richiede delle informazioni il cluster invia la richiesta ai nodi OSD che contengono i dati desiderati. Ciascuno di essi restituisce le informazioni richieste al client, che le utilizza per comporre la risposta completa.

Ruolo MON (Monitor)

In Ceph, un nodo che ha il ruolo di Monitor (MON) è responsabile di mantenere la configurazione e lo stato del cluster, tra cui la posizione dei dati, l'elenco dei nodi OSD disponibili, dei nodi MDS e degli altri nodi Monitor, oltre alle condizioni di salute e disponibilità di tutti i tipi di nodi citati. I Monitor inoltre forniscono al cluster informazioni sulle modifiche allo stato e alla configurazione, in modo da garantire che tutti i nodi del cluster siano sincronizzati. È possibile avere più nodi Monitor contemporaneamente in un cluster Ceph: solitamente si consiglia di averne almeno tre. Avere più nodi Monitor aumenta la tolleranza ai guasti del cluster e riduce il rischio di interruzioni dovute a problemi dei singoli nodi MON. Ognuno di essi mantiene una copia delle informazioni sul cluster, inclusa la sua configurazione e il suo stato, e tali nodi vengono replicati tra loro. Questo garantisce che, in caso di guasto o malfunzionamento di un nodo Monitor, un altro sia in grado di

prenderne il controllo e mantenere il cluster operativo. I nodi Monitor in Ceph comunicano tra loro utilizzando un meccanismo di elezione e consenso basato su Paxos, di cui parleremo nei prossimi capitoli. Questo meccanismo garantisce che tutti i nodi Monitor siano allineati sulla configurazione e sullo stato del cluster e che tutte le modifiche alla configurazione vengano propagate in modo coerente e affidabile a tutti loro. In pratica, questo significa che i nodi Monitor mantengono una copia della stessa mappa di configurazione del cluster e che qualsiasi modifica ad essa viene effettuata tramite un processo di consenso tra tutti i nodi con questo ruolo. Così facendo, viene garantita la consistenza e la disponibilità delle informazioni sul cluster anche in caso di malfunzionamento o interruzione di uno o più nodi Monitor. Un nodo con questo ruolo è anche responsabile di garantire che le richieste effettuate dai client vengano indirizzate ai nodi OSD corretti, e di verificare che la replica dei dati sia effettuata correttamente tra i nodi OSD. In caso di guasto o malfunzionamento di uno di questi ultimi, il Monitor notifica il problema al Manager che fa sì che i dati vengano automaticamente spostati ad un altro nodo. Quando un client richiede dati, il nodo Monitor viene consultato per determinare dove sono attualmente memorizzati. Esso fornisce quindi al client le informazioni necessarie per accedere ai dati e il client può quindi interagire direttamente con i nodi OSD per recuperare ciò che ha richiesto. Inoltre, i nodi Monitor fungono da punto di controllo per la configurazione e le politiche di gestione dello storage. Ad esempio, un amministratore può utilizzare il nodo Monitor per aumentare o diminuire la copia dei dati all'interno del cluster o per modificare la politica di ripartizione dei dati.

Ruolo Manager

Il ruolo di Manager è quello di gestire la configurazione e le operazioni di sistema. Funziona fornendo informazioni sul cluster, creando e coordinando il mapping tra gli oggetti e i placement group, gestendo l'equilibrio dei carichi di lavoro, l'aggiornamento delle informazioni sulle unità di archiviazione, la raccolta e l'utilizzo delle statistiche sulle prestazioni. Il Manager usa i dati ricevuti dai nodi Monitor per prendere decisioni informate sulle operazioni del cluster Ceph. In quest’ultimo possono esistere più nodi Manager contemporaneamente, ma di solito è consigliato averne solo uno attivo in un momento specifico per evitare conflitti e mantenere l'integrità del sistema. Il passaggio del ruolo di

Manager attivo ad un altro nodo Manager può essere gestito automaticamente dal sistema in caso di fallimento di quello attualmente attivo. In Ceph, i nodi Monitor comunicano con il nodo Manager per inviargli informazioni sulle condizioni del cluster e sulla salute dei nodi OSD. I client che accedono al cluster Ceph comunicano con i nodi Monitor per ricevere informazioni sulla posizione degli oggetti richiesti e accedere ai dati. I nodi OSD comunicano con il nodo Manager per ricevere informazioni sulle operazioni di archiviazione e sulle decisioni sul posizionamento degli oggetti. Il nodo Manager comunica con i nodi OSD per fornire informazioni sulle operazioni di archiviazione e per gestire la replicazione dei dati.

Ruolo MDS

Il nodo Ceph con il ruolo MDS (Metadata Server) gestisce i metadati del file system Ceph, inclusi la struttura della directory, l'allocazione dei file e la mappatura di questi ultimi all'object storage. MDS fornisce anche le funzionalità di locking per evitare conflitti tra client che modificano lo stesso file e garantisce la coerenza dei dati. MDS lavora insieme ai nodi Object Storage Daemon (OSD) per fornire un sistema di file completamente distribuito e altamente disponibile. In un cluster, ci possono essere più nodi MDS attivi contemporaneamente, a seconda della configurazione e delle esigenze del carico di lavoro, tuttavia, solo uno di questi è designato come "primario" per ogni file system Ceph, mentre gli altri sono in standby come nodi MDS di backup. Il numero di MDS attivi contemporaneamente dipende dalla dimensione del cluster, dalle prestazioni richieste e dalla tolleranza ai guasti. Un nodo MDS in Ceph comunica con tre tipi di componenti:

- Client Ceph: accedono ai file e alle directory sul file system Ceph tramite MDS. Quest’ultimo fornisce informazioni sul posizionamento dei file sugli OSD e sulle operazioni di locking.
- Monitor: esso richiede agli MDS informazioni sul loro stato e sui metadati.
- Altri nodi MDS: questi nodi comunicano tra loro per sincronizzare la replica dei metadati e per eleggersi come nodo primario o di backup.

3.3. Algoritmo Paxos

Come abbiamo accennato in precedenza, i nodi Monitor contengono tutte le informazioni sui nodi del cluster e il loro stato. Per mantenere la coerenza tra i diversi nodi Monitor, Ceph utilizza una semplificazione dell'algoritmo Paxos [2]; esso aggrega protocolli distribuiti che fanno sì che i vari Monitor si accordino tra loro per gli aggiornamenti delle informazioni. L'implementazione dell'algoritmo Paxos su Ceph è definita dalle seguenti fasi:

1. **Fase di preparazione**: uno dei nodi Monitor, chiamato *Proposer*, cerca di convincere gli altri nodi MON del cluster, chiamati *Acceptor*, a scegliere un valore. Il *Proposer* invia un messaggio di preparazione a questi ultimi, contenente un numero di proposta unico e crescente, utilizzato per garantire l'ordine delle stesse, in modo che ognuna di essa venga accettata una sola volta.

2. **Fase di accettazione**: i nodi *Acceptor* ricevono il messaggio di preparazione dal *Proposer* e rispondono con una promessa, contenente il loro valore più recente accettato e il numero di proposta più grande che hanno visto. Se il nodo che ha effettuato la proposta riceve una maggioranza di promesse, allora invia un messaggio di accettazione contenente il valore proposto e il numero della proposta. Gli altri nodi ricevono il messaggio di accettazione e lo confermano solo se il numero di proposta è il più grande che hanno visto.

3. **Fase di commit**: se il *Proposer* riceve una maggioranza di accettazioni, allora invia un messaggio di commit a tutti i nodi del cluster, indicando che il valore proposto è stato scelto permettendo così l'aggiornamento del metadato corrispondente al valore accettato.

In questo modo, Paxos garantisce che tutti i nodi MON del cluster concordino sullo stesso valore. Se un nodo fallisce o si disconnette temporaneamente, gli altri possono continuare a prendere decisioni sulla base della maggioranza dei nodi rimanenti, facendo sì che il sistema rimanga operativo e che le informazioni sullo stato siano sempre disponibili.

3.4. Comunicazioni

Quando un client fa una richiesta di dati in Ceph, i dati possono essere forniti da più nodi, ovvero gli OSD. Ceph utilizza una tecnologia di storage distribuito che ripartisce i dati su più OSD in modo da garantire la disponibilità, la resilienza e la tolleranza ai guasti. A seguito di una richiesta, il sistema cerca di individuare la copia più recente dei dati e la consegna al client. Se una copia è disponibile su più nodi, i dati possono essere consegnati in parallelo da ognuno di loro per migliorare la velocità e la performance. In questo modo, Ceph garantisce la disponibilità dei dati anche in caso di guasti a uno o più nodi e offre una soluzione scalabile e affidabile per la gestione dei dati. Prima di ogni operazione di lettura e scrittura, il cliente deve contattare un server Monitor per ottenere la mappa aggiornata del cluster.

3.4.1. Richiesta di lettura dati

Quando un client Ceph richiede dei dati, i passaggi sono i seguenti:

1. **Richiesta mappa del cluster**: il client comunica con il Monitor per ottenere una mappa aggiornata del cluster.
2. **Richiesta all'MDS**: il client invia una richiesta di file o di directory al nodo MDS.
3. **Ricerca dei metadati**: MDS accede ai propri dati per trovare le informazioni sul posizionamento dei dati richiesti sugli OSD.
4. **Invio delle informazioni al client**: MDS invia al client le informazioni sui dati richiesti e in quale posizione trovarli.
5. **Richiesta all'OSD**: il client invia una richiesta agli OSD che contengono i dati.
6. **Risposta al client**: l'OSD invia i dati richiesti al client.

Per mostrare al meglio il concetto, in Figura 5 viene riportato il diagramma di sequenza dei passaggi sopra descritti.

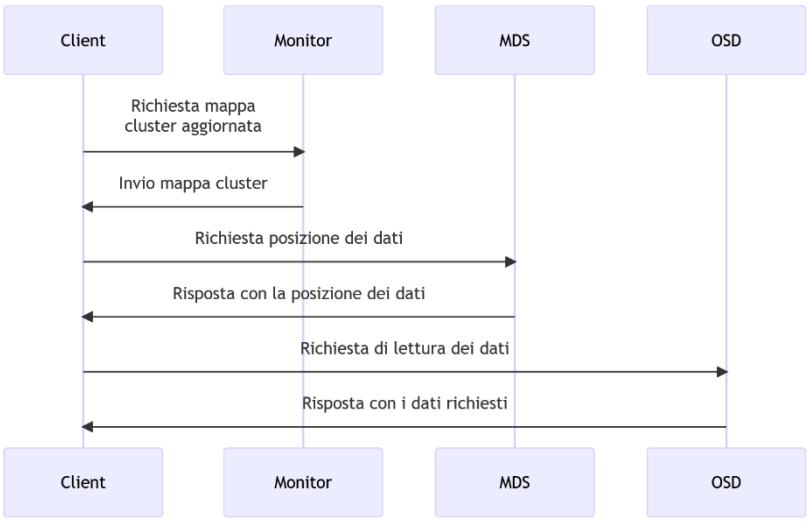

Figura 5 – Diagramma di sequenza per le richieste di lettura

3.4.2. Richiesta di scrittura dati

I client possono inviare, in modo concorrente, richieste di scrittura in parallelo e in modo asincrono; queste vengono inserite in una coda ed elaborate in modo sequenziale all'interno del cluster. Il motivo di ciò è legato alla necessità di mantenere la coerenza dei dati. Se due richieste di scrittura collidono (ad esempio, entrambe cercano di scrivere sullo stesso file nella stessa posizione), allora il sistema deve risolvere il conflitto in modo deterministico, eseguendo la prima arrivata e mettendo la seconda in coda. Per evitare che le scritture rallentino troppo il sistema, Ceph utilizza tecniche di parallelismo per elaborare più richieste di scrittura contemporaneamente, come l'utilizzo di thread e la suddivisione delle richieste in sotto-operazioni più piccole.

Quando un client Ceph deve salvare dei dati, i passaggi sono i seguenti:

1. **Richiesta mappa del cluster**: il client comunica con il Monitor per ottenere una mappa aggiornata del cluster.

2. **Richiesta all'MDS**: il client comunica con l'MDS per ottenere informazioni di accesso.

3. **Locking dei metadati**: MDS acquisisce un lock sul file interessato, per garantire che non ci siano conflitti con altri client che potrebbero voler modificare lo stesso file.

4. **Invio delle informazioni al Client**: MDS invia al client le informazioni su dove trovare il file da modificare.

5. **Scrittura sugli OSD**: il client invia i dati da scrivere agli OSD per la memorizzazione.

6. **Conferma della scrittura**: gli OSD confermano la scrittura dei dati al client.

7. **Modifica Metadati**: il client comunica all'MDS della modifica e quest'ultimo aggiorna i suoi metadati

8. **Rilascio del lock**: MDS rilascia il lock sul file.

Nella figura sottostante (Figura 6) viene riportato il diagramma di sequenza dei passaggi appena descritti.

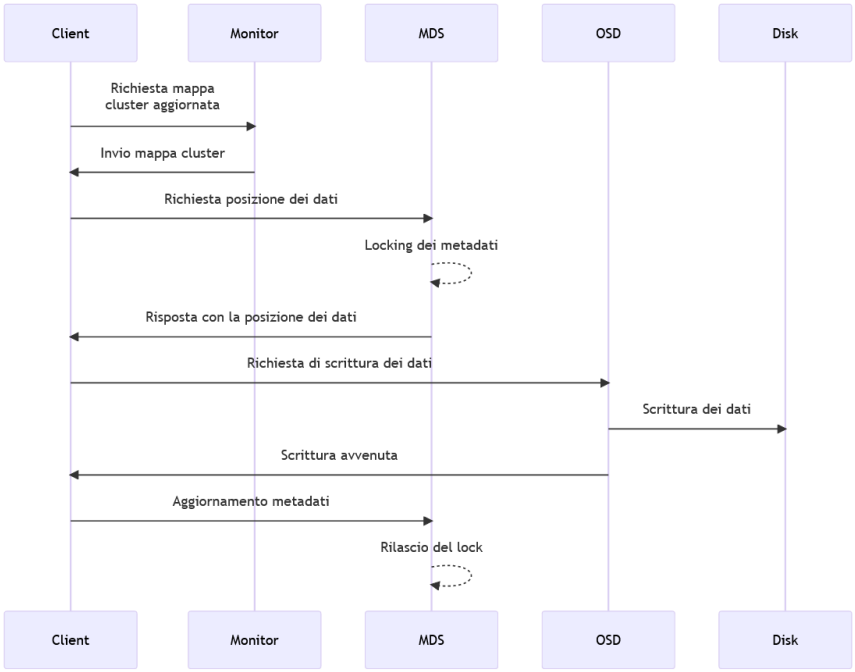

Figura 6 – Diagramma di sequenza per le richieste di scrittura

3.5. Hardware

Per garantire la massima performance e affidabilità, ogni nodo deve essere configurato con hardware dedicato, ad esempio dischi dedicati o unità flash, e sufficiente capacità di elaborazione, memoria e larghezza di banda. In questo modo, Ceph garantisce la disponibilità dei dati anche in caso di guasti a uno o più nodi e offre una soluzione scalabile e affidabile per la gestione dei dati. L'acquisto di hardware dedicato è un investimento a lungo termine che garantisce la qualità e la disponibilità dei dati in un sistema di storage distribuito. Se si unisce hardware più efficiente con hardware meno efficiente in un sistema Ceph, le prestazioni complessive del sistema saranno limitate dal nodo con l'efficienza più bassa. In Ceph, tutti i nodi lavorano insieme per gestire e distribuire i dati, quindi un nodo più lento può rallentare il sistema complessivo e compromettere l'affidabilità del cluster. Per ottenere le migliori prestazioni e affidabilità in un sistema Ceph, è importante utilizzare hardware omogeneo con capacità simili di elaborazione, memoria e archiviazione. In questo modo, tutti i nodi funzioneranno in modo coerente e non vi saranno colli di bottiglia che influenzeranno l'efficienza complessiva del sistema. Ceph è una soluzione di storage distribuito che può essere eseguita su una varietà di hardware, tra cui server dedicati, desktop, laptop, cluster di calcolo e persino dispositivi IoT. L'importante è che il dispositivo soddisfi i requisiti minimi di hardware, come ad esempio la quantità di memoria RAM e di spazio di archiviazione, per supportare l'esecuzione del software Ceph.

3.5.1. Requisiti minimi

I requisiti minimi di hardware per l'esecuzione di Ceph variano in base alla configurazione e all'utilizzo previsto del sistema. Tuttavia, di seguito sono riportati i requisiti di base per l'installazione di un cluster Ceph:

- **Memoria RAM**: almeno 2 GB di memoria RAM per ogni nodo;
- **Archiviazione**: almeno 20 GB di spazio di archiviazione libero su un'unità disco rigido o SSD per ogni nodo;
- **CPU**: almeno un processore a 64 bit con almeno 2 core;

- **Sistema operativo**: Ceph supporta una vasta gamma di sistemi operativi, tra cui Red Hat Enterprise Linux (RHEL), CentOS, Ubuntu e Debian.

È importante notare che questi sono solo i requisiti minimi per l'installazione di un cluster Ceph. Per implementazioni più grandi o per prestazioni migliori, potrebbero essere necessari hardware più potente e più spazio di archiviazione. Il numero minimo di nodi per un cluster Ceph dipende dalle esigenze specifiche del cluster e dalle configurazioni di archiviazione e disponibilità.

Per i nodi Object Storage Daemon (OSD), è possibile iniziare con uno solo di essi, ma per garantire la tolleranza ai guasti e la ridondanza dei dati, è consigliabile averne almeno tre. Per quanto riguarda i nodi Metadata Server (MDS), questo dipende dalla quantità di dati e dall'utilizzo dei file system distribuiti; in generale, è possibile iniziare con un solo nodo MDS, ma potrebbe essere necessario aumentarne il numero per garantire una maggiore capacità di elaborazione e disponibilità dei metadati. È importante considerare che i requisiti minimi per un cluster Ceph possono variare in base alle esigenze specifiche del cluster e che potrebbe essere necessario aumentare il numero di nodi in base alla crescita dei dati e all'utilizzo del cluster.

4. Servizi di Storage di Ceph

Ceph fornisce tre diversi tipi di servizi di storage: a blocchi, a file e ad oggetti; queste sono divisioni software che descrivono come i dati vengono organizzati e gestiti e sono implementati come funzionalità software all'interno di un sistema di archiviazione. La scelta del modello di archiviazione dipende dalle esigenze specifiche dell'applicazione e dalle richieste dell'utente, ma Ceph fa sì che si possano utilizzare tutti e tre i tipi di archiviazione in uno stesso storage.

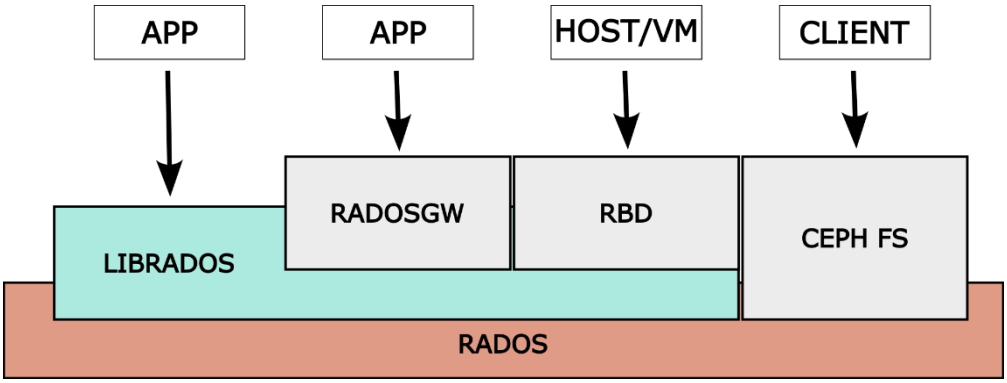

Figura 7 – Interfacce di comunicazione con i diversi tipi di storage

L'immagine in Figura 7 mostra come i diversi tipi di client possono comunicare con il cluster utilizzando specifiche interfacce, le quali, a loro volta, si appoggiano a RADOS.

4.1. RADOS

RADOS (Reliable Automatic Distribuited Object Store) [3] è il protocollo utilizzato da Ceph per la gestione dei dati. Questi ultimi vengono divisi in oggetti, ad ognuno dei quali viene assegnato un ID univoco per la sua identificazione, un peso e una serie di attributi, tra cui il numero di repliche e la loro posizione. Gli oggetti vengono poi assegnati in maniera pseudo casuale ai vari nodi del cluster utilizzando l'algoritmo CRUSH e replicati per garantire alta disponibilità e tolleranza ai guasti. RADOS gestisce la coerenza dei dati attraverso l'utilizzo di un meccanismo di scrittura di tipo "update-in-place", in cui i dati vengono aggiornati direttamente nell'oggetto: quando quest'ultimo viene aggiornato, esso si occupa di propagare la modifica a tutte le sue repliche. L'accesso ai dati in RADOS avviene attraverso

diverse interfacce, tra cui librados, RBD (RADOS Block Device) e RGW (RADOS Gateway). L'interfaccia librados è una libreria client utilizzata dalle applicazioni per accedere ai dati archiviati in RADOS; RBD, invece, consente ai client di accedere ai dati archiviati come blocchi di dati, come se fossero un dispositivo di archiviazione block-level tradizionale. L'interfaccia RGW, infine, permette l'accesso ai dati archiviati tramite protocolli di archiviazione web standard, come Amazon S3 e OpenStack Swift.

4.2. Algoritmo CRUSH

L'algoritmo CRUSH (Controlled Replication Under Scalable Hashing) [4] determina come archiviare e recuperare i dati calcolando le posizioni di archiviazione. Esso è progettato per garantire una distribuzione uniforme dei dati su tutti i nodi di archiviazione del sistema, ridurre il carico di rete, rendere il sistema resiliente ai guasti su singoli nodi e consente ai client di Ceph di comunicare direttamente con gli OSD invece di utilizzare un server o un broker centralizzato; in questo modo viene evitato un singolo punto di errore, un collo di bottiglia delle prestazioni e un limite fisico alla sua scalabilità.

CRUSH utilizza una mappa del cluster, denominata CRUSH map, per mappare in modo pseudo casuale i dati agli OSD, distribuendoli nel cluster in base alla politica di replica configurata. Questa mappa contiene una lista di dispositivi, una gerarchia di bucket, e regole per definire il modo in cui CRUSH replica i dati all'interno dei pool del cluster. I dispositivi sono singoli OSD che memorizzano i dati, in genere uno per ogni unità di archiviazione. Essi sono identificati da un ID (un numero intero non negativo) e un nome *osd.id* dove *id* è l'ID del dispositivo. Inoltre, dalla versione Luminous, ai dispositivi può anche essere assegnata una classe di dispositivo come ad esempio hdd, ssd, nvme. Bucket, invece, è un termine usato per definire i nodi interni della gerarchia. Riflettendo l'organizzazione fisica sottostante dell'installazione, CRUSH è in grado di modellare, e quindi affrontare, i potenziali guasti ai dispositivi. I criteri con cui l'algoritmo decide il posizionamento delle repliche possono essere: armadi rack, vicinanza fisica delle macchine, una fonte di alimentazione condivisa e reti condivise; ad esempio, per affrontare la possibilità di guasti simultanei, potrebbe essere auspicabile che le repliche di dati si trovino su

dispositivi che utilizzino, armadi rack, alimentatori, controller e posizioni fisiche diverse. L'insieme di questi criteri, aggiunti alla mappa di CRUSH, viene denominato dominio di errore.

La posizione di un OSD all'interno della gerarchia della mappa viene definita "posizione CRUSH" e questo indicatore assume la forma di un elenco di coppie chiave/valore.

La mappa CRUSH è costituita da una gerarchia che descrive la topologia fisica del cluster e da un insieme di regole che definiscono i criteri di posizionamento dei dati. Considerando la gerarchia come una struttura ad albero con un nodo radice di tipo root, troviamo nelle foglie i dispositivi, mentre nei nodi interni, ovvero i bucket, sono presenti host, rack, ecc. Le regole, invece, descrivono come vengono posizionate le repliche in termini di tale gerarchia. Nella mappa di CRUSH sono definiti una serie di tipi predefiniti utilizzati per descrivere questi nodi: osd(dispositivo), host, chassis, rack, row, pdu, pod, room, datacenter, zone, region, root. È possibile anche definire, in base alle esigenze, altri tipi personalizzati. Nell'immagine sottostante (Figura 8) viene visualizzato un esempio di gerarchia della mappa CRUSH.

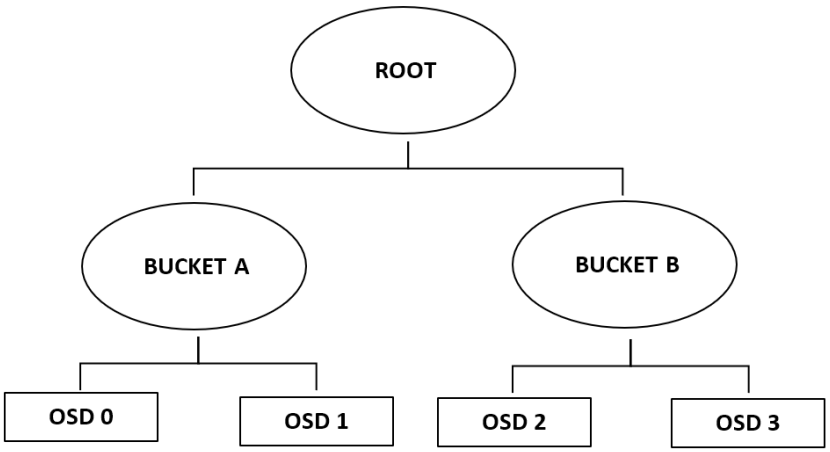

Figura 8 – Esempio di mappa CRUSH

Ogni nodo nella gerarchia ha un peso che indica la proporzione relativa dei dati totali che il dispositivo o la struttura della gerarchia devono archiviare; il peso di una foglia è impostato in base alle dimensioni del dispositivo, mentre quello della radice è la somma dei pesi di tutti i dispositivi che dipendono da essa. I pesi consentono al cluster di

eseguire l’ottimizzazione numerica in base alle specifiche dello stesso per ottenere una distribuzione bilanciata; tuttavia, poiché CRUSH è un processo di posizionamento pseudocasuale probabilistico, c’è sempre qualche variazione rispetto ad una distribuzione ideale. Ci sono due tipi di pesi supportati:

- Compact: viene assegnato un peso per ogni dispositivo o nodo del cluster, non è adatto a correggere tutte le anomalie ma ha il vantaggio di essere retrocompatibile con le versioni di Ceph precedenti alla release Luminous v12.2.z

- Per-pool: consente di ottimizzare il posizionamento dei dati regolando il peso di ogni data-pool, permettendo così di correggere una leggera inclinazione dei dati verso dispositivi con pesi piccoli rispetto ai loro pari provocando problemi di bilanciamento (questo di solito si può presentare in cluster molto grandi)

In caso di utilizzo di tutte e due le tipologie di pesi, il sistema sceglierà il tipo per-pool per il posizionamento dei dati in un determinato pool. Se nessuna delle due viene utilizzata, CRUSH userà i suoi pesi predefiniti.

4.3. Ceph File System

Lo storage a file è un tipo di archiviazione dei dati in cui le informazioni vengono trattate come file, ovvero come unità organizzate di dati con un nome e un percorso specifici. Il sistema di archiviazione a file gestisce questi ultimi come unità complete, permettendo di accedere, modificare ed eliminare i file come se fossero unità autonome. In pratica, il sistema di archiviazione a file mantiene una tabella di catalogo che contiene informazioni sulle proprietà dei file, tra cui il nome, il percorso, la dimensione e la data di modifica. Ogni volta che viene effettuata un'operazione di scrittura, questo sistema di archiviazione sovrascrive i dati esistenti nel file corrispondente, mentre per le operazioni di lettura i dati vengono ricostruiti a partire dai blocchi di storage su cui il file è memorizzato. Lo storage a file è il tipo di archiviazione più adatto per le applicazioni che richiedono un accesso sequenziale ai dati, poiché non è ottimizzato per le operazioni di lettura e scrittura casuali come lo storage a blocchi. Il File System di Ceph, chiamato CephFS, è un file system conforme a POSIX costruito sopra

un archivio di oggetti distribuito chiamato RADOS. L'obbiettivo di CephFS è fornire un archivio di file all'avanguardia, multiuso, disponibile ed efficiente per tutte le applicazioni. Ciò è possibile attraverso l'uso di alcune nuove scelte architettoniche, come la divisione tra i dati e i loro metadati, i quali vengono archiviati in un pool RADOS separato. L'accesso a questo pool avviene tramite un cluster di server di metadati (MDS), che può essere ridimensionato per supportare carichi di lavoro a throughput più elevato. Questa divisione è dovuta al fatto che in un file system le operazioni sui metadati occupano, in genere, più del 50% di tutte le operazioni sullo stesso; dividendoli dai dati si evita di sovraccaricare il cluster RADOS. I client che vogliono utilizzare il file system CephFS hanno accesso diretto a RADOS per leggere e scrivere i file, non essendoci nessun gateway o broker che media l'input/output i carichi di lavoro possono scalare linearmente con la dimensione dell'archivio RADOS sottostante. Nella Figura 9 seguente possiamo notare come i client che vogliono utilizzare il file system CephFS hanno accesso diretto al pool dei dati di RADOS per la lettura e la scrittura dei file.

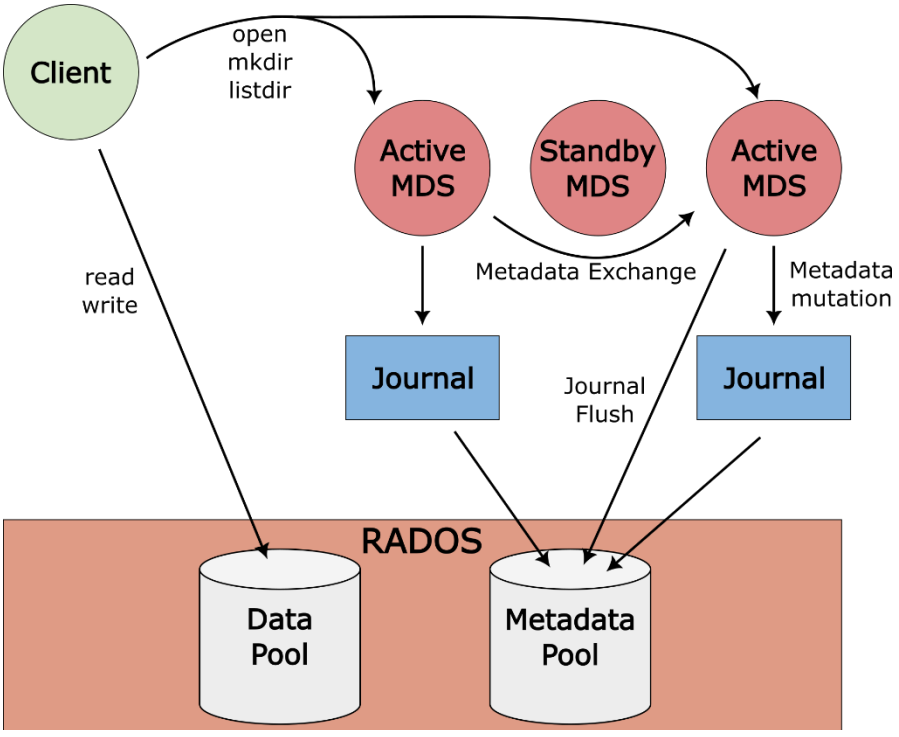

Figura 9 – File system CephFS

Poiché non sono presenti gateway o broker per la mediazione dell'input/output, i carichi di lavoro possono scalare linearmente con la dimensione dell'archivio RADOS sottostante. L'accesso ai dati è coordinato da cluster di MDS che funge da autorità per lo stato della cache dei metadati distribuiti, la quale viene gestita in modo cooperativo tra client e MDS. I cambiamenti ai metadati vengono aggregati da ciascun MDS in una serie di scritture efficienti su una sorta di diario su RADOS, evitando così che lo stato dei metadati venga memorizzato localmente dal MDS. Questo modello consente una collaborazione coerente e rapida tra i client nel contesto di un file system POSIX.

4.3.1. Differenze con POSIX

CephFS cerca di aderire alla semantica POSIX il più possibile; tuttavia, ci sono diversi punti in cui CephFS si discosta da questa per diversi motivi:

- Nel caso di fallimento della scrittura richiesta da un client. Le scritture non sono necessariamente atomiche e utilizzano un buffer per effettuare l'operazione; può quindi capitare che si blocchi e che la modifica sia applicata solo parzialmente. Quasi tutti i file system, anche quelli locali, hanno questo comportamento.

- Nelle situazioni di scrittura simultanea condivisa, poiché se essa attraversa i limiti dell'oggetto non è necessariamente atomica. Questo potrebbe far sì che il risultato finale dell'operazione sia un misto delle operazioni svolte contemporaneamente e quindi sbagliato.

- Se i file sparsi si propagano in modo non corretto nel campo st_block, che indica il numero di blocchi di dimensione fissa che il file occupa sul sistema di archiviazione. CephFS non tiene traccia in modo esplicito di quali parti di un file vengano allocate o scritte e il campo st_block contiene la dimensione del file divisa per la dimensione del blocco. Questo può portare a sopravvalutare lo spazio occupato.

- Quando un file viene mappato in memoria tramite mmap() su più host, le scritture non vengono propagate in modo coerente alle cache degli altri client; questo significa che se una pagina viene

memorizzata nella cache di un host A e poi aggiornata su un host B, la pagina presente in A non viene invalidata.

- I client CephFS presentano una directory nascosta .snap utilizzata per accedere, creare, eliminare e rinominare gli snapshot; qualsiasi client che tenti di creare un file o una cartella con lo stesso nome otterrà un codice di errore.

4.3.2. Diario MDS

Come abbiamo detto precedentemente CephFS utilizza due pool separati tra loro per la gestione dei dati e dei metadati; il pool di metadati contiene tutte le informazioni sui file, inclusa la gerarchia del file system. Inoltre, CephFS mantiene un diario in cui vengono segnate le operazioni effettuate sui metadati che gli MDS inviano a RADOS prima di eseguire un'operazione sul file system.

Questo sistema permette di mantenere la coerenza in caso di rottura di uno o più MDS, in quanto gli eventi segnati sul diario possono essere rieseguiti per raggiungere uno stato coerente del file system. Oltre a ciò, ne traggono vantaggio anche le prestazioni in quanto gli aggiornamenti sul diario sono principalmente sequenziali, quindi veloci, e più aggiornamenti possono essere accorpati in un'unica scrittura, risparmiando così il tempo di ricerca su disco necessario per gli aggiornamenti di divere parti di un file. Ogni server MDS attivo mantiene il proprio diario nel pool dei metadati e quando alcune voci presenti sono considerate vecchie e non più necessarie, vengono eliminate.

Oltre agli aggiornamenti dei metadati, vengono registrati anche altri eventi come, ad esempio, le informazioni sulla sessione del client e lo stato di importazione ed esportazione della directory che vengono utilizzati per le riconnessioni e il ripristino della sessione in caso di riavvio del client.

4.3.3. Distribuited Metadata Cache

Generalmente i client di CephFS richiedono ad un MDS di recuperare o modificare i metadati per proprio conto, ma possono anche farsi rilasciare la capacità di memorizzare nella propria cache ed eventualmente modificare una parte dei dati o dei metadati. Quando un client desidera operare un'inode (abbreviazione di index node, struttura

dati che contiene informazioni sui file e sulle directory presenti nel file system), può richiedere diversi tipi di funzionalità che permettono di lavorarci in vari modi. Le funzionalità a differenza di altri file system di rete, ad esempio NFS o SMB, sono molto granulari e permettono a più client di avere funzionalità diverse sugli stessi inode. Quando un altro cliente ha bisogno di accedere alle stesse informazioni, il server MDS revoca il permesso rilasciato precedentemente ed il primo client restituisce i dati con le eventuali modifiche. I client posso sempre richiedere le funzionalità sugli inode, ma in caso di accesso concorrente o di un utilizzo troppo elevato della memoria sull'MDS, esse possono essere revocate; in tal caso il client è tenuto a restituirle appena possibile altrimenti il sistema lo inserisce in una lista di blocco che gli vieta di comunicare con il cluster. Poiché la cache è distribuita, l'MDS deve prestare attenzione per garantire che nessun client disponga di capabilities che potrebbero entrare in conflitto con quelle di altri client o con operazioni che esegue esso stesso. Su un file system con molteplici server MDS, la cache dei metadati è distribuita anche tra gli MDS del cluster. Per ogni inode, in un dato momento, solo uno dei server è considerato autorevole e le eventuali modifiche possono essere concesse solo da esso (se le richieste vengono inviate ad un server non autorevole le può inoltrare). Inoltre, i server non autorevoli hanno comunque la possibilità di bloccare la scrittura in caso ricevano delle richieste di lettura finché l'operazione non viene ultimata.

4.4. Ceph Block Device

Lo storage a blocchi è un tipo di archiviazione dei dati in cui questi ultimi vengono suddivisi in blocchi di dimensioni predefinite e vengono trattati come unità separate. Ogni blocco viene indirizzato con un identificatore univoco e viene trattato come una singola unità di archiviazione, indipendentemente dal contenuto o dalla dimensione del file di origine. Lo storage a blocchi viene utilizzato principalmente in sistemi di archiviazione di grandi dimensioni, in cui è necessario gestire grandi quantità di dati in modo efficiente. Questo tipo di archiviazione è ottimizzato per le operazioni di lettura e scrittura casuali, poiché è possibile accedere a un singolo blocco senza dover leggere l'intero file. In pratica, il sistema di archiviazione a blocchi crea un mapping tra gli identificatori dei blocchi e la loro posizione sul supporto di archiviazione fisico; ogni volta che viene effettuata un'operazione di

scrittura, il sistema sovrascrive i dati esistenti nei blocchi corrispondenti, mentre per le operazioni di lettura i dati vengono ricostruiti a partire dai blocchi. Le interfacce di archiviazione basate su blocchi sono un modo comune per archiviare i dati su supporti come HDD, SSD, CD, ecc. I dispositivi a blocchi di Ceph sono thin-provisioning, (lo spazio viene effettivamente allocato dinamicamente solo quando il sistema ne fa richiesta, ad esempio quando bisogna salvare nuovi file), ridimensionabili e memorizzano i dati su più OSD usando lo striping, che permette di dividere il dato in piccoli blocchi, ciascuno scritto su un dispositivo di archiviazione diverso. Ceph Block Device utilizza le funzionalità di RADOS tra cui gli snapshot, la replicazione e la consistenza dei dati. I client che vogliono utilizzare questo servizio di storage comunicano con il cluster Ceph tramite i moduli del kernel o la libreria librdb.

4.4.1. Snapshot

Uno snapshot è una copia logica di sola lettura di un'immagine in un determinato momento. Una delle funzionalità avanzate di Ceph Block Device è la possibilità di creare istantanee di immagini del sistema e conservare la cronologia dello stato in quel punto del tempo. Ceph supporta anche la stratificazione delle istantanee, che consente di clonare immagini in modo rapido e semplice. Gli snapshot vengono gestiti utilizzando il comando *rdb* e diverse interfacce di livello superiore, tra cui, QUEMU, libvirt, OpenStack e CloudStack. È anche possibile creare molti cloni copy-on-write (COW) di uno snapshot di un dispositivo a blocchi; in questo caso ogni immagine clonata (child) memorizza un riferimento alla sua immagine padre che le consente di aprirla e leggerla.

4.4.2. Lock esclusivo

I blocchi esclusivi sono dei meccanismi progettati per impedire a più processi di accedere allo stesso RBD in modo non coordinato. Questi sono ampiamente utilizzati nella virtualizzazione (dove impediscono alle macchine virtuali di saturare le reciproche scritture) e nel mirroring RDB. Per mantenere l'accesso multi-client, la funzione di blocco esclusivo implementa transizioni di blocco cooperative automatiche tra i client. Esso garantisce che solo un client possa scrivere su un'immagine RBD in un qualsiasi momento e quindi protegge le

strutture interne dell'immagine come la mappa degli oggetti e il diario da modifiche simultanee. Questa funzionalità è per lo più trasparente per l'utente: ogni volta che un client deve gestire una scrittura su un'immagine RBD, prima deve richiedere il blocco e se quest'ultimo è già detenuto da un altro client viene richiesto di rilasciarlo. Se un client che detiene un blocco riceve una richiesta di rilascio, interrompe la gestione delle scritture, scarica la sua cache e lo rilascia consentendo al client successivo di poterlo acquisire ed iniziare a gestire le scritture. Può capitare che un client che aveva il blocco su un'immagine non termini correttamente o riscontri perdita di connettività con il cluster Ceph per un certo periodo di tempo; in questo caso il rilascio del blocco non avviene in modo corretto e alla richiesta di un nuovo client il nodo Monitor deve aggiungere il vecchio possessore in una blocklist e comunicare agli OSD pertinenti di non servire più le richieste del vecchio processo client; una volta che la mappa del OSD è stata aggiornata si può procedere ad assegnare il blocco al nuovo richiedente.

4.4.3. Mirroring

È possibile eseguire il mirroring asincrono delle immagini RBD tra due cluster Ceph con due modalità:

- Basato sul Journal: questa modalità utilizza la funzionalità dell'aggiornamento del diario RBD per garantire una replica tra i due cluster. Ogni scrittura sull'immagine RBD viene registrata nel diario prima di effettuare effettivamente la modifica; il cluster remoto legge dal diario ed effettua gli aggiornamenti nella sua copia locale. Poiché ogni scrittura sull'immagine comporta due scritture (prima nel diario e poi nello storage), le latenze di scrittura aumentano fino a quasi raddoppiare.

- Basato su Snapshot: questa modalità utilizza gli snapshot pianificati periodicamente o creati manualmente per replicare il cluster.

4.4.4. Live-Migration

Le immagini RBD possono essere migrate in tempo reale tra diversi pool all'interno dello stesso cluster, tra diversi formati e layout o da fonti di dati esterne. Una volta avviata la migrazione, l'immagine di origine verrà copiata in quella di destinazione assieme allo storico degli

snapshot, preservando l'allocazione dei dati ove possibile. Per impostazione predefinita, l'immagine di origine sarà contrassegnata come di sola lettura e tutti i client reindirizzeranno gli I/O alla nuova immagine. Facoltativamente è anche possibile preservare il collegamento tra immagine padre e figlio oppure unificare tutto eliminando la dipendenza. Questo processo di copia può anche essere eseguito in modo sicuro in background mentre la nuova immagine è in uso e si compone di tre fasi:

1. Preparazione della migrazione: viene creata la nuova immagine di destinazione e collegata a quella di origine. Quando quest'ultima non è configurata di sola importazione, anch'essa viene collegata alla nuova e contrassegnata come di sola lettura.

2. Esecuzione della migrazione: operazione in background che esegue la copia completa di tutti i blocchi inizializzati dall'immagine di origine a quella di destinazione. Questo passaggio può essere eseguito mentre i client utilizzano attivamente la nuova immagine.

3. Fine della migrazione: una volta completato il processo di migrazione è possibile eseguire il commit o interrompere la migrazione. Il commit della migrazione rimuoverà i collegamenti tra l'immagine di origine e quella di destinazione e, se non configurata di sola importazione, eliminerà quella di origine.

4.5. Ceph Object Gateway

Lo storage ad oggetti è un tipo di archiviazione dei dati in cui essi vengono trattati come oggetti, ovvero come unità di dati con proprietà e metadati associati. Ogni oggetto viene indirizzato con un identificatore univoco e viene trattato come una singola unità di archiviazione, indipendentemente dalla dimensione del file di origine. Lo storage ad oggetti è progettato per gestire grandi quantità di dati non strutturati, come immagini, video, file audio e documenti. Questo sistema di archiviazione mantiene i metadati associati ad ogni oggetto, che possono includere informazioni sulle sue proprietà, sulla data di creazione e sulla data di modifica. In pratica, esso crea un mapping tra gli identificatori degli oggetti e la loro posizione sul supporto di archiviazione fisico. Ogni volta che viene effettuata un'operazione di scrittura, il sistema di archiviazione ad oggetti sovrascrive i dati

esistenti nell'oggetto corrispondente, mentre per le operazioni di lettura i dati vengono ricostruiti a partire dai blocchi di storage su cui l'oggetto è memorizzato.

Ceph Object Gateway è un'interfaccia di storage di oggetti basata su librados che fornisce un gateway RESTful chiamato RADOS Gateway (radosgw): un server HTTP progettato per interagire con il cluster. Questo tipo di archiviazione supporta due interfacce:

- **Interfaccia S3**: fornisce funzionalità di storage degli oggetti con un'interfaccia compatibile con un ampio sottoinsieme dell'API RESTful di Amazon S3 (Simple Storage Service), un servizio di archiviazione ad oggetti offerto da Amazon Web (AWS)

- **Interfaccia Swift**: fornisce funzionalità di archiviazione degli oggetti con un'interfaccia compatibile con un ampio sottoinsieme dell'API OpenStack Swift, un sistema di archiviazione di oggetti open-source.

Le API S3 e Swift condividono uno spazio dei nomi comuni, permettendo così di poter scrivere i dati con una e poi recuperarli con l'altra. È importante segnalare che Ceph Object Gateway a differenza degli altri due tipi di archiviazione non utilizza gli MDS.

4.5.1. HTTP Frontend

Per le chiamate RESTful Ceph Object Gateway utilizza un servizio frontend HTTP che si basa su due librerie integrate: Boost.Beast e Boost.Asio. Queste fanno parte di una collezione di librerie gratuite e open source che forniscono, per la programmazione C++, funzionalità avanzate in molti ambiti, come quelle per la comunicazione di rete, per la gestione dei thread o dei file e tante altre. Boost.Beast è una libreria per la comunicazione di rete basata su protocolli HTTP/1, HTTP/2 e WebSocket; essa permette la creazione di connessioni persistenti, supportando anche la crittografia TLS/SSL per proteggere le comunicazioni. E', inoltre, in grado di gestire grandi carichi di lavoro attraverso il controllo del buffering e la gestione del timeout. Boost.Asio, invece, fornisce una serie di funzionalità di rete, tra cui: la comunicazione TCP/UDP e dei socket (Livello 4 del modello ISO/OSI), la gestione degli indirizzi IP (Livello 3 del modello ISO/OSI) e una gestione asincrona delle operazioni di I/O. Boost.Asio è stato

progettato anche per integrarsi con Boost.Beast, che lavora ad un livello più alto rispetto alla prima (Livello 5 del modello ISO/OSI).

4.5.2. Gestione utenti Ceph Object Gateway

Il servizio di Object Storage viene amministrato tramite la gestione degli utenti, il controllo degli accessi, il monitoraggio dell'utilizzo ed altre funzionalità. Con utenti si intendono quelli che dovranno utilizzare l'Object Storage, non quelli del cluster. Per permettere l'utilizzo della funzionalità di storage ad oggetti è necessario creare, oltre all'utente, una chiave di accesso e un segreto. Ci sono due tipi di utente: il termine utente si riferisce agli utilizzatori dell'interfaccia S3, mentre, con il termine subutenti quelli che utilizzano Swift; un subutente è sempre connesso ad un utente. È possibile creare, modificare, visualizzare, sospendere e rimuovere sia utenti che subutenti. Ogni utente ha un ID di identificazione ed è possibile aggiungere un nome visualizzato e un indirizzo e-mail. La chiave e il segreto possono essere specificati manualmente oppure generati automaticamente e bisogna tenere presente che ad ogni ID utente corrisponde una chiave S3 mentre ad ogni ID subutente corrisponde una chiave Swift. Queste ultime hanno anche livelli di accesso di lettura, scrittura e lettura e scrittura complete.

4.6 Autenticazione ed Utenti in Ceph

Indipendentemente dal tipo di client e dall'interfaccia utilizzata, Ceph memorizza tutti i dati come oggetti all'interno dei pool. Gli utenti devono avere accesso ai pool per poter leggere e scrivere dati ed avere le autorizzazioni per i comandi amministrativi. Il protocollo per l'autenticazione si chiama Cephx ed opera come Kerberos, senza però un singolo punto di rottura. Un utente in Ceph è un individuo o un attore di sistema come un'applicazione che è identificato da un tipo e un identificativo separati da un punto: TIPO.ID, ad esempio client.admin oppure client.user1. Anche gli OSD e gli MDS utilizzano il protocollo Cephx, quindi anch'essi sono considerati utenti, ma non di tipo client: la presenza del tipo, quindi, aiuta a distinguere tra gli utenti client e gli altri semplificando il controllo degli accessi, il monitoraggio degli utenti e la tracciabilità. Si può inoltre osservare che un utente di Ceph Storage Cluster non coincide né con un utente di Ceph Object Storage né con uno di Ceph File System. Ceph object Gateway utilizza un utente Ceph Storage Cluster per la comunicazione tra il daemon del gateway

e il cluster di archiviazione, ma il gateway dispone di una propria funzionalità di gestione degli utenti per gli utenti finali. Il file system di Ceph, invece, utilizza la semantica POSIX e lo spazio utente associato ad esso non è lo stesso di un utente di Ceph Storage Cluster. Ceph utilizza il termine “capabilities” (caps) per descrivere l’autorizzazione di un utente autenticato a esercitare le funzionalità di monitor, OSD e MDS. Le capabilities possono anche limitare l’accesso ai dati all’interno di un pool o a uno spazio dei nomi al suo interno. È possibile impostare le capabilities di un utente durante la sua creazione o modificandolo successivamente utilizzando i privilegi di amministratore di Ceph.

5. Implementazione di un cluster Ceph

5.1. Descrizione

Mostriamo ora come poter effettuare un'implementazione di base di un cluster Ceph, creando un file system CephFS e agganciandolo ad una macchina client. La macchina host su cui sono state create le nostre macchine virtuali è un windows 11 con processore AMD Ryzen 7 5700x (8 core/16 thread), 16 GB di RAM DDR4 e 1TB SSD M2. Abbiamo utilizzato VirtualBox per creare cinque macchine virtuali su cui è stato installato Ubuntu server 22.04 LTS e abbiamo assegnato loro le seguenti specifiche e nomi:

- **ceph-mon**: server principale su cui è stato installato e creato il nostro cluster Ceph (processori: 2, RAM: 2GB, disco 25GB);
- **node1, node2, node3**: nodi da aggiungere successivamente per estendere il cluster. A queste macchine abbiamo assegnato anche il ruolo di OSD, per cui, oltre a una configurazione hardware simile a quella di ceph-mon, hanno un ulteriore disco da 20 GB ciascuna;
- **clientceph**: la macchina client a cui abbiamo agganciato il file system creato con Ceph.

La struttura risultante può essere osservata nella Figura 10 sottostante.

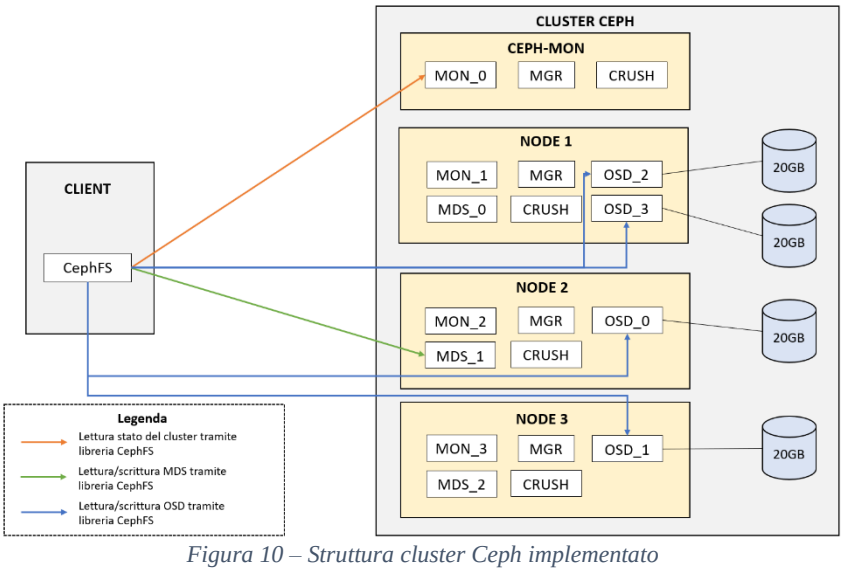

Figura 10 – Struttura cluster Ceph implementato

Date le piccole dimensioni del cluster, per garantire un minimo di ridondanza, il sistema ha replicato in automatico i servizi di Monitor, Manager ed MDS. È stato invece necessario creare manualmente gli OSD che, nel nostro caso, sono quattro. Nonostante ci siano quattro servizi MDS, solo uno è attivo, mentre gli altri sono in stand-by, in attesa di subentrare al primo in caso di guasti. Per implementazioni più avanzate è però possibile attivare più nodi MDS contemporaneamente. Lo stesso discorso può essere fatto per il servizio di Monitor, utilizzato dal client per richiedere la mappa più aggiornata del cluster e per autenticarsi. Sulla macchina client, invece, è stato montato un file system CephFS che si occupa di gestire tutte le comunicazioni con il cluster Ceph tramite le proprie librerie.

5.2. Prerequisiti

I passaggi che andremo a descrivere in questo sottoparagrafo dovranno essere effettuati su ognuna delle macchine virtuali. Dopo aver installato il nostro sistema operativo, se non è già stato fatto in precedenza o durante l'installazione guidata, dobbiamo installare il server OpenSSH per permettere ai nodi di comunicare tramite il protocollo SSH. Il comando da lanciare è il seguente:

```
sudo apt install openssh-server
```

Successivamente, è necessario installare il servizio NTP ed aggiungere come server NTP lo stesso per tutti i nodi in modo da avere lo stesso tempo su tutte le macchine:

```
sudo apt install ntp
```

Il comando successivo, invece, permette di installare curl per leggere il repository di Ceph:

```
sudo apt install curl
```

È poi necessario installare un qualsiasi container engine: in questo caso abbiamo installato Docker seguendo i passaggi della sua guida ufficiale [5], qui riportati.

```
sudo mkdir -m 0755 -p /etc/apt/keyrings
```

```
curl -fsSL
  https://download.docker.com/linux/ubuntu/gpg | sudo
  gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \ "deb [arch=$(dpkg --print-architecture)
  signed-by=/etc/apt/keyrings/docker.gpg]
  https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee
  /etc/apt/sources.list.d/docker.list > /dev/null

sudo chmod a+r /etc/apt/keyrings/docker.gpg

sudo apt update

sudo apt install docker-ce docker-ce-cli
  containerd.io
```

Come ultimo passaggio assegniamo ad ogni server un IP statico; Ubuntu utilizza Netplan come network manager, quindi dobbiamo creare un file di configurazione dentro la cartella /etc/netplan/ in cui inserire i dati desiderati tramite editor di testo. Per fare ciò lanciamo il comando seguente:

```
sudo nano /ect/netplan/00-installer-config.yaml
```

Il file deve essere composto ed indentato come nella Figura 11 sottostante che mostra la configurazione data al nodo ceph-mon.

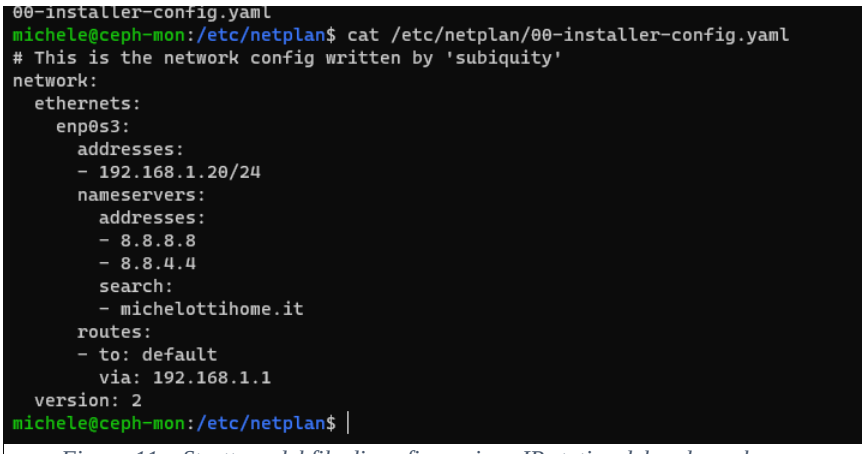

Figura 11 – Struttura del file di configurazione IP statico del nodo ceph-mon

Non è obbligatorio, ma, nel caso non fossero già presenti, è consigliato installare anche le utility LVM2 per semplificare la gestione di più drive, come nel nostro caso per i nodi:

```
sudo apt install lvm2
```

Una volta effettuati questi passaggi siamo pronti per l'installazione vera e propria di Ceph.

5.3. Installazione

Queste operazioni sono da effettuare solo sul nodo principale, nel nostro caso ceph-mon. Per prima cosa usiamo curl per recuperare la versione più recente dello script standalone e rendiamo lo script cephadm eseguibile:

```
curl --silent --remote-name --location
  https://github.com/ceph/ceph/raw/quincy/src/cephadm
  /cephadm

chmod +x cephadm
```

Anche se lo script è sufficiente per inizializzare un cluster, è consigliato installare il comando cephadm sull'host, quindi aggiungiamo il repository dell'ultima versione di Ceph disponibile (nel nostro caso Quincy) e lo installiamo con i prossimi due comandi.

```
./cephadm add-repo --release quincy

./cephadm install
```

Una volta installato cephadm, possiamo eseguire il comando seguente:

```
cephadm bootstrap --mon-ip 192.168.1.20
```

Esso creerà il primo nodo monitor, passandogli come argomento l'indirizzo IP del nostro server che dovrà ricoprire quel ruolo (ceph-mon). Una volta eseguito questo comando:

- viene creato un demone di monitoraggio e gestione per il nuovo cluster sull'host locale;
- viene generata una chiave SSH per il cluster Ceph nel file /root/.ssh/authorized_keys e viene scritta una copia della chiave pubblica in /etc/ceph/ceph.pub;

- vengono scritti un file di configurazione minimale in /etc/ceph/ceph.conf, una copia della chiave segreta di amministratore in etc/ceph/ceph.client.admin.keyring e viene aggiunta l'etichetta _admin all'host che ha eseguito bootstrap.

Figura 12 – Risultato del comando di creazione del nodo monitor

In Figura 12 viene mostrato il risultato dell'esecuzione dell'ultimo comando citato contenente anche i dati per accedere alla dashboard di gestione di Ceph (interfaccia dei nodi monitor). Al primo login verrà subito chiesto di creare una nuova password scelta dall'utente.

Una volta creato il primo nodo monitor dobbiamo condividere la chiave pubblica di Ceph con gli altri server in modo da permettere al sistema di collegarsi via SSH e aggiungerli al cluster. Per prima cosa copiamo la chiave presente nel file /etc/ceph/ceph.pub e la andiamo ad inserire sui vari node1, node2, node3 tramite utente root in /.ssh/authorized_keys. Da impostazioni predefinite, l'utente root su Ubuntu è disattivato, ma per impostarne l'accesso bisogna semplicemente assegnargli una password tramite il comando:

```
sudo passwd root
```

Adesso, tramite la dashboard possiamo aggiungere i nostri nodi al cluster in maniera semplice.

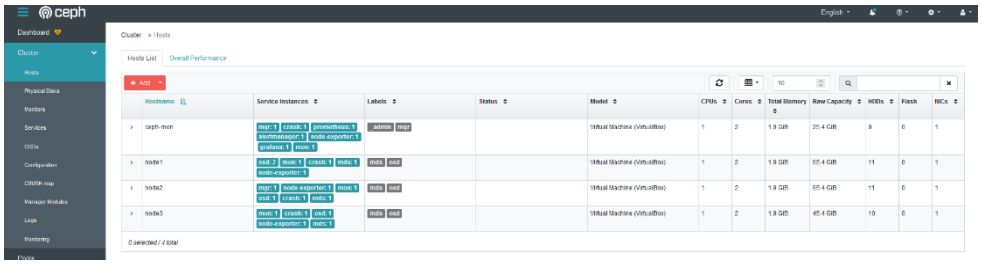

Figura 13 – Scheda degli host all'interno della dashboard di Ceph

Come mostrato in Figura 13, nella scheda host, oltre a visualizzare tutti gli host già aggiunti al cluster, possiamo anche inserirne di nuovi impostando il nome dell'host, il suo indirizzo IP ed eventuali etichette. Una volta che la procedura è andata a buon fine bisogna aspettare che il sistema installi i componenti sul nuovo server e quando sarà pronto verrà visualizzato insieme agli altri. Il sistema si bilancerà in automatico attivando i servizi su tutti i nodi disponibili come ulteriori monitor e manager.

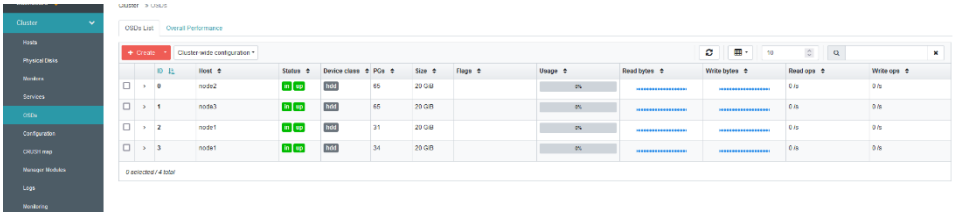

Figura 14 – Scheda di gestione degli OSD

In Figura 14 viene mostrata la scheda per la gestione e l'aggiunta di nuovi OSD (scheda OSDs nel menu principale di sinistra). In tempo reale viene mostrato lo stato di ogni OSD, l'host a cui appartiene, la sua classe, dimensione e l'utilizzo. Ad ogni OSD corrisponde uno storage fisico separato, nel nostro caso sono i dischi virtuali secondari assegnati alle macchine node1, node2, node3. Premendo il tasto in alto "create" è possibile creare ulteriori OSD per aumentare lo spazio del cluster utilizzando nuovi dispositivi di storage aggiunti ai nostri nodi. Nella Figura 15 si può notare la scheda con la mappa CRUSH del cluster, nel nostro caso molto semplice in quanto abbiamo solo 3 host: due di essi

hanno un solo OSD assegnato, mentre l'ultimo ne ha due. Tutti gli OSD presenti hanno classe HDD.

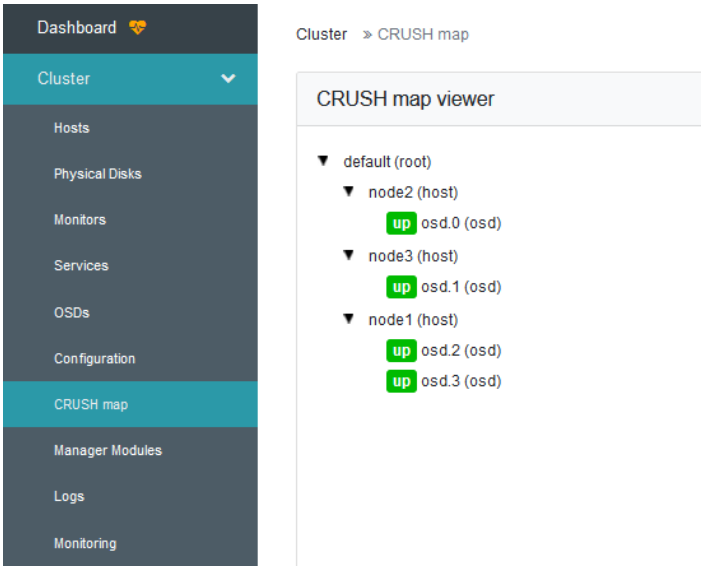

Figura 15 – Scheda con la mappa CRUSH del cluster

In Figura 16 viene mostrata la pagina iniziale della dashboard con tutte le informazioni di base del cluster come il numero di OSD, di monitor attivi ed in standby, dei manager, la capacità dello storage ed altri dati.

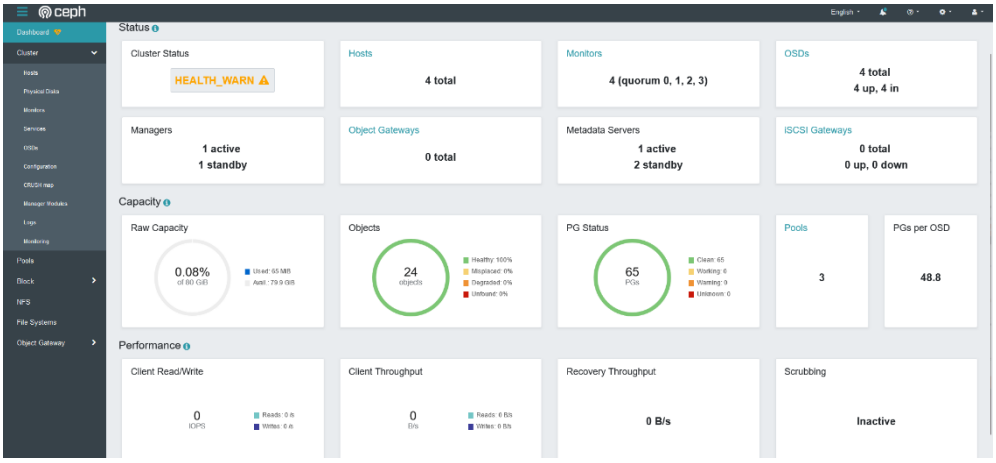

Figura 16 – Dashboard del cluster Ceph

Lo stato di health warning è dovuto al fatto che sul nodo monitor principale è stato montato un disco virtuale thin, che si ingrandisce in base alla quantità di dati che il sistema necessita di scriverci, per il sistema il disco è pieno al 75% e segna il warning anche se in realtà sono occupati solo 7Gb su 25GB.

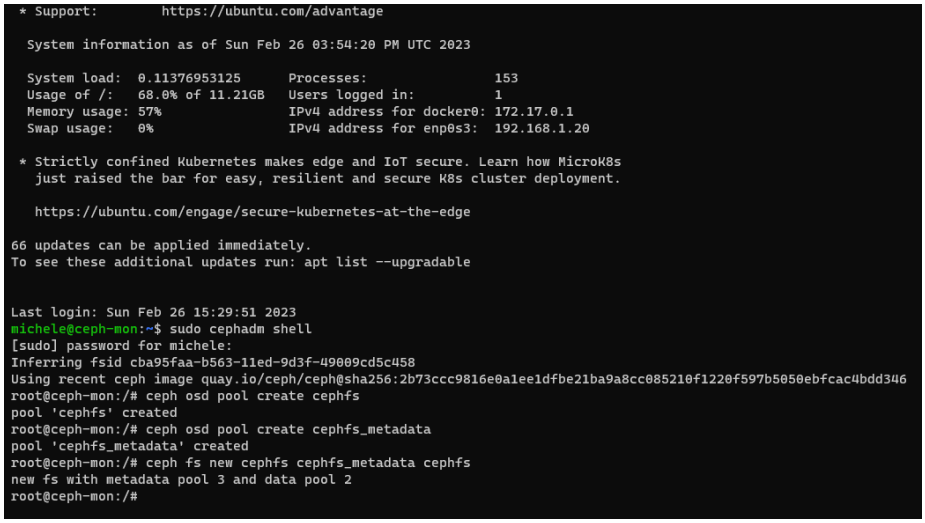

Figura 17 - Creazione del File System CephFS

Nell'immagine riportata sopra (Figura 17) viene mostrato come creare un filesystem CephFS. Collegandoci via SSH al nostro server _admin, ceph-mon, possiamo ora utilizzare la shell cephadm per la creazione del filesystem [6].

In primo luogo, creiamo un nuovo pool per i dati:

```
ceph osd pool create cephfs
```

In seguito, ne creiamo uno anche per i metadati:

```
ceph osd pool create cephfs_metadata
```

Infine, creiamo il filesystem che per i dati si appoggi al pool cephfs e per i metadati al pool cephfs_metadata con il comando:

```
ceph fs new cephfs cephfs_metadata cephfs
```

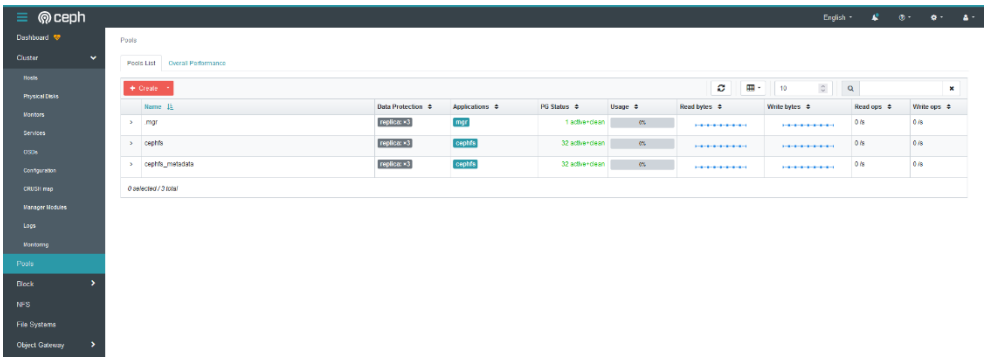

Figura 18 – Scheda dei pool

Tornando ora sulla dashboard, come si può osservare dall'immagine sopra riportata (Figura 18), la scheda dei pool risulta aggiornata con le nuove modifiche; similmente, come riporta l'immagine sottostante (Figura 19), nella scheda dei filesystem sono riportati quelli appena creati.

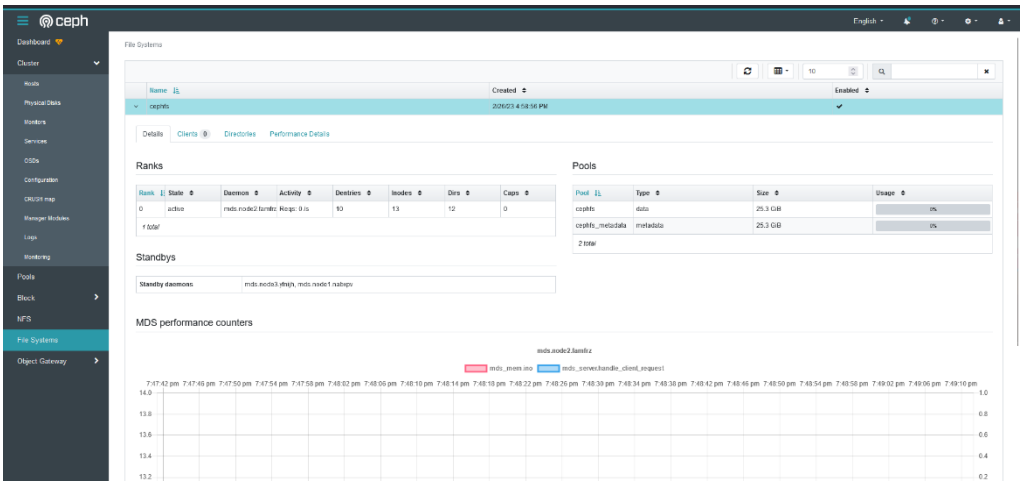

Figura 19 – Scheda dei File System

5.4. Utilizzare CephFS su macchine client

Dopo aver creato il nostro cluster Ceph e un file system, dobbiamo creare un utente ed assegnare i permessi su cosa è in grado di effettuare. Con il comando seguente, viene creato un utente e una chiave da inviare alla macchina o alle macchine che lo dovranno utilizzare:

```
ceph auth get-or-create client.1 mon 'allow r' mds
  'allow rw' osd 'allow rwx pool=cephfs'
```

L'utente così creato è in grado di accedere al pool di dati cephfs e di leggere e scrivere i suoi metadati. Per verificare che la chiave sia creata e i permessi assegnati usiamo il comando come in Figura 20:

```
ceph auth get client.1
```

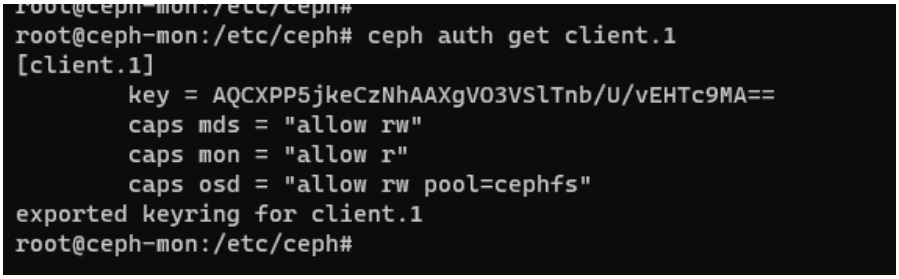

Figura 20 – Chiave per utilizzo del file system

Questa chiave e il file ceph.conf situato nella cartella /etc/ceph/ sono i due elementi necessari per permettere ad un client di utilizzare Ceph come uno storage personale, mentre l'installazione del pacchetto ceph-common permette di lanciare il comando per effettuare l'attacco.

Nella macchina client andiamo a creare una cartella /etc/ceph in cui andremo a copiare il file ceph.conf e a creare uno denominato ceph.client.1.secret, in cui inseriremo la chiave creata poco fa.

In questa implementazione monteremo il file system di Ceph manualmente.

Creiamo una directory in cui montare cephfs:

```
mkdir -p /mnt/cephfs
```

Montiamo ora il file system specificando sia gli indirizzi IP e le porte dei nostri Monitor separati dalla virgola, sia il nostro file segreto contenente la chiave:

```
mount -t ceph 192.168.1.20:6789,192.168.1.21:6789,
  192.168.1.22:6789:/ /mnt/cephfs -o
  name=1,secretfile=/etc/ceph/ceph.client.1.secret
```

In questa fase è necessario aggiungere gli IP delle nostre macchine che svolgono il ruolo di Monitor perché servono al client per avere sempre una mappa aggiornata del cluster per le richieste ai nodi OSD e MDS. Possiamo inserire nel comando anche un solo indirizzo IP di un nodo Monitor ma è sempre conveniente aggiungerne di più, in quanto, se il primo dovesse rompersi, il file system continuerebbe a funzionare utilizzando gli altri. Inoltre, è possibile utilizzare i DNS dei server al posto del loro indirizzo IP, se disponibili. Possiamo ora verificare che il file system sia montato come visualizzato in Figura 21 utilizzando:

```
stat -f /mnt/cephfs
```

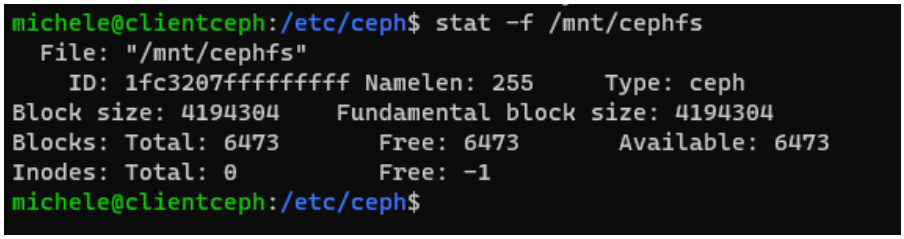

Figura 21 – Verifica che il file system sia montato

Una volta che viene aggiunto, il sistema è in grado di utilizzarlo come se fosse una risorsa locale. Assegnando un file system alle diverse macchine di un cluster Docker Swarm, è possibile assegnare quella stessa risorsa ai container che godranno così di una risorsa persistente e distribuita; sarà Ceph a gestire gli accessi simultanei e le varie problematiche che possono insorgere. In questo contesto abbiamo mostrato come usare CephFS, ma i passaggi per utilizzare invece l'interfaccia a blocchi sono piuttosto simili. È stato scelto di mostrare la prima piuttosto che la seconda in quanto è l'approccio più indicato in un contesto di più container che devono scrivere e leggere file da possibili diverse macchine come è Docker Swarm. Per utilizzare invece l'interfaccia ad oggetti, una volta impostato il gateway, la comunicazione avviene tramite le librerie citate nel capitolo di riferimento (4.5 Ceph Object Gateway da pagina 34).

6. Confronto tra Ceph ed altre soluzioni SDS

6.1. GlusterFS

6.1.1. Introduzione a GlusterFS

GlusterFS [7] è un file system distribuito open source che consente di creare un pool di storage scalabile e ridondante su più server e di metterlo a disposizione ai client, che lo utilizzano tramite connessioni TCP/IP. Le risorse che vengono messe in condivisione possono essere montate sui client attraverso i protocolli CIFS, NFS oppure attraverso il client nativo Gluster. Spieghiamo ora brevemente alcune terminologie fondamentali di GlusterFS:

- **Brick**: rappresenta un'unità di storage di un nodo del cluster ed è l'unità di base per l'archiviazione e la distribuzione dei dati all'interno di un volume.
- **Volume**: identifica la condivisione effettiva che viene messa a disposizione dal cluster. Un volume è costituito da uno o più brick
- **Translator**: librerie che estendono le funzionalità del file system.
- **Server**: sono le macchine (anche virtuali) che compongono il cluster dove risiedono i dati
- **Client**: sono le macchine che devono montare ed usare il volume condiviso.

6.1.2. Tipi di volumi

Il file system di GlusterFS supporta diversi tipi di volumi, alcuni sono utili per migliorare la scalabilità del sistema, alcuni per migliorarne le prestazioni ed altri per entrambi:

- **Distribuited GlusterFS Volume:** questo è il tipo di volume che viene creato per impostazione predefinita se non viene specificato alcun tipo di volume. I file sono distribuiti su vari bricks senza essere replicati, di conseguenza non contiene la ridondanza dei dati. Lo scopo di questo volume è di facilitare il suo ridimensionamento, anche a livello economico, benché ciò significhi che un errore su uno

dei brick comporterà la perdita dei dati e sia necessario quindi fare affidamento sull'hardware sottostante per la loro protezione.

- **Replicated GlusterFS Volume:** In questo volume si pensa principalmente a contrastare il rischio di perdita dei dati mantenendo delle copie esatte di essi su ogni brick. Il numero di repliche può essere deciso dal client durante la creazione del volume ed in base al numero scelto si rendono necessari un top di brick da assegnare ad esso; se vogliamo creare un volume con due repliche dobbiamo obbligatoriamente assegnare 2 brick, per tre repliche tre brick. Questo volume, quindi, viene utilizzato per una migliore affidabilità e ridondanza dei dati, a discapito di un maggior costo di espansione se si necessita di aumentare lo spazio di storage.

- **Distribuited Replicated GlusterFS Volume:** Qui i file vengono distribuiti su un set di brick replicati ed il loro numero deve essere un multiplo del numero delle repliche impostate. Anche l'ordine in cui specifichiamo i brick è importante perché quelli adiacenti diventeranno l'uno le repliche dell'altro. Questo tipo di volume viene utilizzato quando è richiesta un'elevata disponibilità dei dati e un'alta scalabilità del sistema. Se ad esempio, creiamo un volume con otto brick e impostiamo due repliche, i primi due brick saranno repliche uno dell'altro, poi i due successivi e così via e questo volume viene indicato come 4x2.

- **Dispersed GlusterFS Volume:** In un volume di tipo sparso i dati vengono codificati e spezzettati in diversi brick, tenendo conto anche di una certa ridondanza. In questo modo è possibile avere un livello configurabile di affidabilità, impostandola alla creazione del volume, con il minimo spreco di spazio. I brick ridondanti determinano quanti di essi possono essere persi senza interrompere il servizio.

- **Distribuited Dispersed GlusteFS Volume:** Questi volumi sono l'equivalente dei volumi replicati distribuiti, ma utilizzano sotto-volumi dispersi invece di quelli replicati. Il numero di brick deve essere sempre un multiplo del primo sotto-volume. Lo scopo di ciò è quello di ridimensionare facilmente il volume e distribuire il carico su diversi brick.

6.1.3. File system

GlusterFS è un file system in spazio utente, gli sviluppatori hanno optato per questo approccio per evitare la necessità di avere moduli nel kernel linux. Per interagire con il Kernel VFS (Virtual File System), GlusterFS utilizza FUSE (File System in Userspace). FUSE è stato sviluppato per permettere l'implementazione di un file system in spazio utente e supporta l'interazione tra il kernel VFS e le applicazioni utente non privilegiate e dispone di un'API a cui è possibile accedere dallo spazio utente.

6.1.4. DHT (Distributed Hash Table)

DHT è il vero fulcro di come GlusterFS aggrega capacità e prestazioni su più server. La sua responsabilità è di posizionare ogni file esattamente su uno dei suoi brick ed è una funzione di routing. Ad ogni sotto-volume (brick) viene assegnato un intervallo all'interno di uno spazio hash a 32 bit, che copre l'intero intervallo senza buchi o sovrapposizioni. Successivamente a ogni file viene assegnato un valore in quello stesso spazio, eseguendo l'hashing del nome. Esattamente un brick avrà un intervallo assegnato che include il valore dell'hash del file, quindi esso dovrebbe trovarsi lì. Ci sono però dei casi speciali in cui questo può non avvenire, ad esempio, se il brick è quasi pieno, se un set di brick viene cambiato ed il sistema si deve ingegnare con opportuni sistemi avanzati. In generale però quando vogliamo aprire un file, GlusterFS utilizza il suo nome per determinare dove si trova eseguendo l'hashing per trasformare quel nome in un numero.

6.1.5. Differenze con Ceph

Ceph e GlusterFs hanno differenze significative in diversi ambiti, di seguito elenchiamo i punti di forza di Ceph rispetto al suo concorrente:

- **Scalabilità:** Ceph è altamente scalabile e in grado di gestire grandi quantità di dati su un gran numero di nodi, mentre GlusterFS ha limitazioni sulla scalabilità e sulla quantità di server che possono essere uniti al cluster;

- **Affidabilità:** Ceph è un sistema altamente affidabile grazie alla sua architettura ridondante e alla capacità di replicare i dati su più nodi, GlusterFS invece non ha un sistema integrato di ridondanza ma

va impostato durante la creazione del volume come se si creasse un tipo specifico di RAID;

- **Gestione della cache:** GlusterFS utilizza una cache centralizzata per memorizzare i dati, mentre Ceph utilizza una cache distribuita, che consente di migliorare le prestazioni;

- **Protocolli ed interfacce supportate:** Ceph permette ai client di utilizzarlo come uno storage a file, a blocchi e ad oggetti ed inoltre supporta diversi protocolli ormai divenuti standard come S3 e Swift per interfacciarsi con i dati in maniera semplice. GlusterFS invece utilizza principalmente il protocollo NFS ed offre ai client un file system condiviso.

Nonostante questo, GlusterFS ha come suoi punti di forza il file system, un algoritmo di memorizzazione più veloce rispetto a quello di Ceph, non necessita di server per la gestione dei metadati ed è più adatto per l'utilizzo con file strutturati e con accesso sequenziale. Inoltre, GlusterFS è più semplice da gestire ed utilizzare, per cui la scelta tra le due soluzioni dipende molto dalle necessità del client e l'ambiente in cui deve essere utilizzato.

6.2. BeeGFS

6.2.1. Introduzione a BeeGFS

BeeGFS [8] è un file system parallelo POSIX indipendente dall'hardware, noto anche come archiviazione parallela definita dal software; è stato sviluppato con una forte attenzione verso le prestazioni e progettato per la facilità d'uso, d'installazione e di gestione. Esso, nonostante il codice sorgente sia disponibile pubblicamente, offre una Community Edition gratuita ed una Enterprise Edition a pagamento completamente supportata e con funzionalità aggiuntive. BeeGFS è pensato per tutti gli ambienti ad alte prestazioni come HPC (High Performance Computing), AI, Deep Learning, Media e intrattenimento.

6.2.2. Architettura

BeeGFS combina più server di archiviazione per fornire un file system di rete condiviso, scalabile, con lo striping dei file tra i vari nodi. In questo modo consente al client di recuperare il dato, prendendo parallelamente le varie parti di esso da diversi server. BeeGFS separa i

metadati dal contenuto dei file: mentre i server di archiviazione sono responsabili della memorizzazione delle parti di dati, i server di metadati coordinano il loro posizionamento e la loro divisione sui primi. Per mantenere al minimo la latenza di accesso ai metadati, il sistema consente anche di distribuire questi ultimi su più server in modo che ciascuno memorizzi una parte dello spazio dei nomi del file system globale. Lato server, BeeGFS viene eseguito come un gruppo di demoni nello spazio utente senza requisiti speciali sul sistema operativo. Il client, invece, può essere installato su tutti i kernel Linux ed è implementato come un modulo del kernel che fornisce un punto su cui è montato il file system condiviso, per far sì che le applicazioni possano accedervi come se fosse un file system locale. Oltre ai tre ruoli base citati fino ad ora (client, server di archiviazione e server di metadati), ci sono due servizi di sistema aggiuntivi: il servizio di gestione (manager), che funge da registro del sistema, e watchdog per client e server, che funge da monitoraggio.

6.2.3. Differenze con Ceph

Le differenze con Ceph sono le seguenti:

- **Architettura:** BeeGFS è un sistema di file parallelo distribuito che organizza i dati in file e directory e li distribuisce su un cluster di nodi di archiviazione. Ceph invece, è un sistema di archiviazione distribuito ad oggetti, e i dati vengono scritti su nodi di archiviazione in modo ridondante, consentendo un accesso rapido e affidabile.

- **Scalabilità:** entrambi i sistemi sono altamente scalabili, ma Ceph è stato progettato fin dall'inizio per scalare su una vasta gamma di nodi di archiviazione. Esso può essere utilizzato per archiviare grandi quantità di dati, distribuendoli su numerosi nodi di archiviazione in modo ridondante e affidabile. BeeGFS è stato originariamente progettato per un cluster di calcolo ad alte prestazioni, ed è stato ottimizzato per offrire elevate prestazioni di lettura e scrittura su grandi file; anche se può lavorare in ambienti di archiviazione di dati di grandi dimensioni, il suo progetto è stato pensato per lavorare su un numero limitato di nodi.

- **Gestione dei dati:** Ceph offre strumenti per la gestione di storage di file, blocchi ed oggetti; BeeGFS, d'altra parte, è limitato

alla gestione dei dati come file e cartelle, ma questo lo rende più semplice da utilizzare.

- **Sicurezza:** Ceph offre un livello aggiuntivo di sicurezza grazie alla crittografia a livello di oggetto. Esso può crittografare i dati su ogni nodo di archiviazione, offrendo maggiore protezione. BeeGFS non offre la crittografia dei dati a livello di file ma è possibile crittografare i dati sul file system utilizzando strumenti standard come dm-crypt o LUKS.

- **Prestazioni:** BeeGFS è stato progettato per offrire prestazioni elevate su grandi file ed è noto per le sue prestazioni di lettura e scrittura a bassa latenza. Questo lo rende ideale per applicazioni che richiedono un accesso rapido a file di grandi dimensioni. Ceph, invece, è molto flessibile: anche se non raggiunge le prestazioni di BeeGFS in lettura e scrittura, il suo vantaggio è la duttilità con cui si può adattare per diversi tipi e carichi di lavoro.

- **Costo:** BeeGFS ha due tipi di licenze, una community gratuita e una Enterprise per le aziende con un canone da pagare per avere delle funzionalità aggiuntive rispetto alla prima versione. Ceph invece è un progetto open-source.

7. Conclusioni

In questa tesi abbiamo studiato ed analizzato il funzionamento di Ceph, un sistema di archiviazione molto complesso, ma che permette all'utente finale di utilizzare le sue funzionalità in maniera semplice e trasparente. Alla base del suo funzionamento c'è RADOS, un sistema di archiviazione interno ad oggetti che permette una distribuzione dei dati uniforme tra i nodi del cluster tramite l'utilizzo dell'algoritmo CRUSH (Controlled Replication Under Scalable Hashing). Abbiamo visto come dati e metadati vengono gestiti in maniera separata permettendo così di ottimizzarne la gestione per ognuno. I dati vengono salvati come oggetti per poi essere suddivisi in piccole parti distribuite e replicate tra i vari nodi per garantire ridondanza ed affidabilità, i metadati vengono invece gestiti utilizzando un servizio chiamato MDS (Metadata server). Questa divisione permette al sistema di scalare in modo più efficiente e migliorare le prestazioni di lettura/scrittura, in quanto i nodi di archiviazione possono concentrarsi solo sui dati e non sui metadati. Ceph utilizza tre diversi servizi di storage: a blocchi, a file e ad oggetti, e possono coesistere tutti e tre contemporaneamente in un unico cluster Ceph; l'utente che ne utilizza i servizi potrà scegliere quello più indicato alle proprie esigenze. Abbiamo poi implementato un piccolo cluster Ceph con quattro nodi di cui tre anche con il ruolo di OSD per la gestione dello storage fisico. È stato inoltre creato un file system CephFS e montato su una quinta macchina Ubuntu Server per poterlo utilizzare come un'estensione del file system locale. Infine, abbiamo analizzato brevemente due soluzioni simili a Ceph anch'esse famose in ambito cloud: GlusterFS e BeeGFS, ambe due focalizzate su un sistema di file system, di gestione più facile di Ceph, ma con meno opzioni ed interfacce di utilizzo. La grande forza di quest'ultimo è la moltitudine di funzionalità e protocolli che mette a disposizione ai suoi utilizzatori e che lo rendono così una soluzione più che valida per una miriade di scenari di utilizzo.

Ceph è un sistema di archiviazione distribuito, pensato e progettato fin dall'inizio per essere altamente scalabile, affidabile e duttile. Proprio queste sue caratteristiche lo rendono uno strumento indicato per il suo utilizzo sul cloud e con cluster di servizi containerizzati.

Bibliografia

[1] Ceph, «Welcome to Ceph,» [Online]. Available: https://docs.ceph.com/en/latest/.

[2] J. Luis, Interviewee, *Monitors and Paxos, a chat with Joao.* [Intervista]. 10 Settembre 2013.

[3] Admin Magazine, «The RADOS Object Store and Ceph Filesystem,» [Online]. Available: https://www.admin-magazine.com/HPC/Articles/The-RADOS-Object-Store-and-Ceph-Filesystem.

[4] S. A. Weil, S. A. Brandt, E. L. Miller e C. Maltzahn, «CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data,» [Online]. Available: https://ceph.io/assets/pdfs/weil-crush-sc06.pdf.

[5] Docker, «Install Docker Engine on Ubuntu,» [Online]. Available: https://docs.docker.com/engine/install/ubuntu/.

[6] Red Hat, «Red Hat Training. Chapter 3. Deploying Ceph File Systems,» [Online]. Available: https://access.redhat.com/documentation/it-it/red_hat_ceph_storage/3/html/ceph_file_system_guide/deploying-ceph-file-systems.

[7] GlusterFS, «Gluster Docs - Architecture,» [Online]. Available: https://docs.gluster.org/en/main/Quick-Start-Guide/Architecture/.

[8] BeeGFS, «BeeGFS Documentation 7.3.2,» [Online]. Available: https://doc.beegfs.io/latest/index.html.