Alma Mater Studiorum · Università di Bologna

DIPARTIMENTO DI INFORMATICA

Corso di Laurea in Ingegneria e Scienze Informatiche

DEPLOY AUTOMATIZZATO DI UN CLUSTER CEPH COME STRUMENTO DIDATTICO PER LA GESTIONE DEI SISTEMI DISTRIBUITI

Tesi di Laurea in

Virtualizzazione e Integrazione di Sistemi

Relatore:
Prof. Vittorio Ghini

Presentata da:
Giacomo Biagioni

Anno Accademico 2024–2025

Terzo appello di laurea

Parole chiave

Ceph

MicroCeph

Multipass

Kubernetes

Rook

storage distribuito

deploy automatizzato

Abstract

Il lavoro analizza, in un contesto didattico e con risorse limitate, il deploy di un'infrastruttura di storage distribuito basata su Ceph. Sono stati presi in esame tre approcci con livelli crescenti di automazione e astrazione: un'installazione semi-manuale tramite gli strumenti nativi di Ceph (in particolare cephadm), utile per osservare in modo esplicito le fasi di bootstrap, l'aggiunta degli OSD e il controllo dello stato del cluster; una soluzione semi-automatica che combina Multipass, MicroCeph e un orchestratore sviluppato ad hoc in Python, capace di creare in modo ripetibile più VM, inizializzarle con cloud-init e ottenere un cluster Ceph funzionante e accessibile anche da client esterni; un approccio container-native basato su Kubernetes e sull'operatore Rook, analizzato soprattutto dal punto di vista architetturale, che consente di esporre i servizi di storage Ceph come risorse dichiarative del cluster. Il confronto, condotto rispetto a complessità di installazione, livello di automazione, requisiti hardware, visibilità sui meccanismi interni di Ceph e scalabilità, mostra che i tre approcci sono tra loro complementari: quello manuale è il più formativo, quello automatizzato è quello che meglio si presta a essere replicato in laboratorio, quello con Kubernetes/Rook è il più vicino agli scenari cloud-native, ma anche il più esigente in termini di competenze e risorse. Le prove sono state svolte su un unico host fisico e con storage in parte simulato, per cui i risultati vanno letti in chiave principalmente didattica.

Introduzione

Negli ultimi anni l'evoluzione delle infrastrutture informatiche ha portato a una crescente diffusione di sistemi di archiviazione distribuiti, capaci di garantire elevata disponibilità, affidabilità e scalabilità. In tale contesto, il progetto `Ceph` si è affermato come una delle soluzioni open source più complete per la gestione di storage unificato a oggetti, blocchi e file system. La sua architettura distribuita, basata su algoritmi di replica e auto-bilanciamento, consente di ottenere resilienza e buone prestazioni utilizzando hardware di tipo commodity, cioè economico e facilmente reperibile.

All'interno di un contesto didattico e sperimentale, riprodurre un cluster `Ceph` rappresenta un'opportunità significativa per comprendere le dinamiche dei sistemi distribuiti e dei meccanismi di orchestrazione. Tuttavia, la complessità dell'installazione e della configurazione manuale di `Ceph` può costituire un ostacolo per studenti e ricercatori che intendono esplorare la tecnologia in modo pratico. Da questa considerazione nasce l'idea di confrontare diversi approcci alla realizzazione di un'infrastruttura `Ceph`, con l'obiettivo di individuare un metodo efficace, ripetibile e adatto a scopi formativi.

In questo lavoro vengono quindi prese in esame tre strategie di implementazione di un cluster `Ceph`, ma con un diverso grado di verifica sperimentale. La prima è un approccio semi-manuale basato su `cephadm`, che è stato effettivamente realizzato e testato in ambiente di laboratorio, così da osservare concretamente le fasi di *bootstrap*, l'aggiunta dei dispositivi e la verifica dello stato del cluster. La seconda è un approccio semi-automatico basato su `MicroCeph`, `Multipass` e un orchestratore sviluppato ad hoc, anch'esso

effettivamente implementato e provato, con lo scopo di valutare quanto l'automazione e la virtualizzazione leggera possano facilitare la riproducibilità del cluster. La terza è l'approccio containerizzato con `Rook` su `Kubernetes`, che in questa tesi viene analizzato principalmente in termini teorici e architetturali, senza una realizzazione completa nel medesimo ambiente di test, allo scopo di comprendere come uno storage `Ceph` possa essere integrato in un cluster `Kubernetes` e reso disponibile tramite le primitive native della piattaforma.

L'obiettivo principale del lavoro è quindi descrivere, confrontare e valutare queste metodologie sia dal punto di vista tecnico, sia in termini di efficacia didattica. L'analisi si concentra su aspetti quali la semplicità di deploy, la riproducibilità dell'ambiente, la flessibilità nella configurazione e la comprensione dei meccanismi interni di `Ceph`, tenendo conto dei limiti sperimentali derivanti dall'hardware disponibile e dalla natura teorica del terzo approccio.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Obiettivi della tesi

1.1 Obiettivo generale

Il presente lavoro si propone l'obiettivo generale di analizzare tre differenti approcci per il deployment di un'infrastruttura di storage distribuito basata su Ceph, focalizzandosi in particolare su ambienti didattici e di sperimentazione con risorse hardware limitate.

1.2 Obiettivi specifici

Per conseguire l'obiettivo generale, sono stati definiti i seguenti obiettivi specifici:

- Analizzare le tecnologie e i concetti teorici alla base di Ceph, della virtualizzazione e dell'orchestrazione containerizzata.
- Realizzare un approccio completamente manuale per l'installazione e la configurazione di un cluster Ceph, al fine di acquisire una comprensione dettagliata del funzionamento interno del sistema.
- Sviluppare un framework semi-automatico, basato su MicroCeph e Multipass, in grado di automatizzare la creazione e il deploy del cluster su macchine virtuali leggere.

- Analizzare l'integrazione dello storage distribuito nell'ecosistema Kubernetes tramite l'operatore Rook, valutando la fattibilità teorica del deploy container-native del cluster Ceph.
- Confrontare i tre approcci in termini di semplicità d'uso, ripetibilità, scalabilità e risorse richieste, evidenziando vantaggi, svantaggi e casi d'uso raccomandati.

1.3 Ambito e limiti

L'ambito della tesi è circoscritto a scenari di laboratorio e didattici, eseguiti su hardware non dedicato e con risorse limitate; non si affrontano deployment in ambienti di produzione su larga scala. Le verifiche sono orientate alla fattibilità, alla ripetibilità e alla comparazione tra diversi approcci di orchestrazione di Ceph, più che a test prestazionali o di resilienza. In tutti i casi le prove sono state condotte su un singolo host fisico che ha ospitato le componenti necessarie. I vincoli operativi e le conseguenze su scalabilità e misurazioni sono dettagliati nella Sezione 6.4.

Capitolo 2

Fondamenti teorici e tecnologie coinvolte

In questo capitolo vengono presentati i fondamenti teorici e le principali tecnologie impiegate nel progetto. In particolare, verranno descritti gli strumenti utilizzati per la realizzazione dell'infrastruttura distribuita, tra cui **Ceph** e la sua versione leggera **MicroCeph**, il sistema di orchestrazione dei container **Kubernetes** (e le sue varianti), l'interfaccia di integrazione **Rook**, l'ambiente di virtualizzazione **Multipass** e vari strumenti di supporto.

2.1 Cos'è Ceph

Ceph è un sistema di storage definito via software (software-defined storage) open-source, progettato per offrire un'infrastruttura di archiviazione distribuita, unificata e altamente scalabile. Lo scopo principale di Ceph è mettere a disposizione servizi di storage a oggetti, a blocchi e a file all'interno di un unico cluster composto da hardware commodity [1], eliminando i punti singoli di guasto.

[1] *Hardware commodity* — Dispositivi e sistemi costruiti con componenti convenzionali, prodotti in serie, e interoperabili con altri dell'identico tipo; usati per costruire infrastrutture scalabili ed economiche.

2.1.1 Caratteristiche fondamentali

Tra le caratteristiche distintive del sistema Ceph troviamo:

- **Scalabilità orizzontale**: l'architettura consente di espandere il cluster semplicemente aggiungendo nuovi nodi, senza modifiche profonde all'infrastruttura.
- **Tolleranza ai guasti e auto-gestione**: grazie alla distribuzione dei dati e alla replica (o codifica a cancellazione) su più nodi, il sistema è in grado di recuperare automaticamente da guasti di dispositivi o nodi.
- **Unificazione dei modelli di storage**: in uno stesso cluster possono essere forniti servizi di storage ad oggetti, servizi di storage a blocchi (volumi per macchine virtuali o container) e file system distribuiti compatibili POSIX.
- **Uso di hardware commodity**: non è necessario hardware proprietario o controller RAID particolari: il cluster può essere costruito su server standard e dischi/dispositivi generici.

2.1.2 Architettura di base

Il livello fondamentale di Ceph è rappresentato da **RADOS** [2] (Reliable Autonomic Distributed Object Store): in questo modello tutti i dati sono trattati come oggetti e distribuiti tra i vari nodi del cluster in modo autonomo e resiliente. La distribuzione degli oggetti è governata dall'algoritmo **CRUSH** [3] (Controlled Replication Under Scalable Hashing), che determina in modo deterministico su quale nodo o dispositivo un dato oggetto debba essere memorizzato. Questo approccio evita l'utilizzo di tabelle centralizzate di lookup e migliora la scalabilità e il bilanciamento del carico.

Al di sopra di RADOS si colloca **librados**, la libreria client che fornisce le API per leggere e scrivere oggetti nello store distribuito. Tutti i servizi di livello superiore di Ceph (come RBD per i volumi a blocchi, RADOS

Gateway per l'object storage compatibile S3/Swift e CephFS per il filesystem distribuito) non accedono direttamente ai singoli OSD, ma utilizzano librados per interagire con RADOS in modo uniforme. In questo modo la logica di replica, bilanciamento e recupero resta concentrata nello strato di storage, mentre i servizi sopra possono offrire interfacce diverse (a blocchi, a oggetti, a file) senza dover reimplementare i meccanismi di affidabilità.

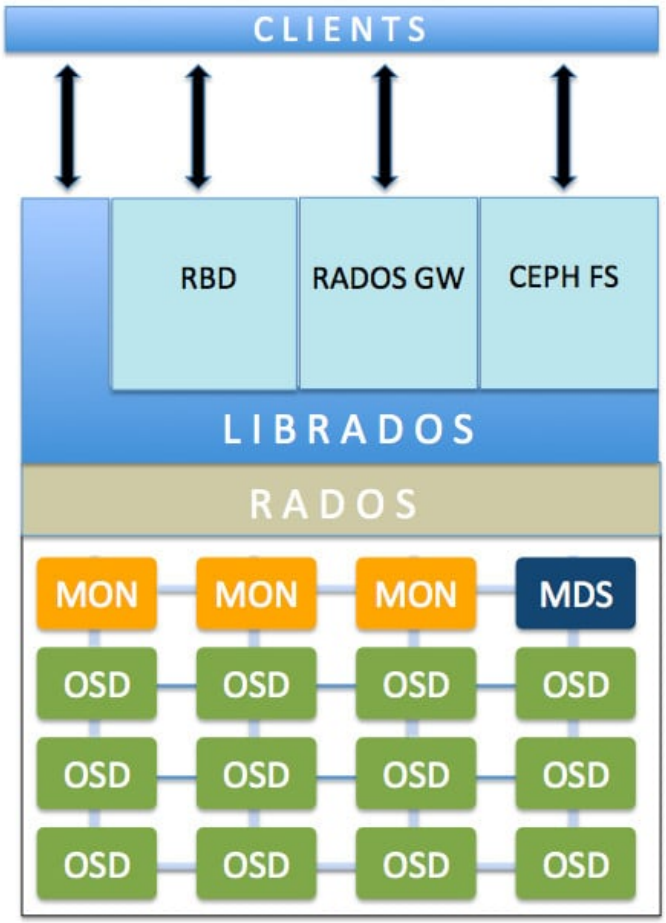

Figura 2.1: Architettura a livelli di Ceph: i client accedono ai servizi di alto livello (RBD, RADOS Gateway, CephFS), che si appoggiano alla libreria `librados` e allo store distribuito RADOS; alla base il cluster è costituito dai demoni MON, OSD e, per CephFS, MDS.

Fonte: adattato da [13].

Il cluster Ceph si compone di diversi demoni principali. I Monitor (MON) hanno il compito di mantenere la mappa dello stato del cluster, la configurazione e il consenso tra i nodi. Gli Object Storage Daemon (OSD) gestiscono direttamente i dispositivi fisici di memorizzazione, memorizzano gli oggetti, partecipano alla replica e al bilanciamento automatico; quando viene utiliz-

zato il filesystem distribuito CephFS, interviene inoltre il Metadata Server (MDS), che ha il compito di gestire i metadati di file e directory. A questi si aggiungono altri demoni come il Manager (MGR) e il Gateway (RGW), che estendono le funzionalità verso il monitoraggio, le interfacce a oggetti e il supporto al file/block storage.

2.1.3 Meccanismi chiave del sistema Ceph

Il corretto funzionamento di Ceph si basa su un insieme di meccanismi distribuiti che assicurano disponibilità dei dati, resilienza ai guasti e bilanciamento del carico. In particolare, tre elementi fondamentali ne caratterizzano l'architettura: la replica dei dati, l'auto-guarigione (self-healing) e la distribuzione controllata degli oggetti all'interno del cluster.

2.1.4 Replica dei dati

La replica rappresenta il meccanismo attraverso cui Ceph garantisce affidabilità e tolleranza ai guasti all'interno del cluster. Ogni oggetto memorizzato nel livello RADOS viene duplicato su più demoni OSD secondo una policy di replica definita già in fase di progettazione del pool (tipicamente tre copie). Il posizionamento e la gestione delle repliche non avviene attraverso un controllo centralizzato, bensì in modo distribuito grazie all'algoritmo CRUSH, che calcola in modo deterministico la collocazione degli oggetti nel cluster evitando colli di bottiglia e favorendo scalabilità e bilanciamento. Di conseguenza, i dati risultano ridondanti e restano disponibili anche in presenza di uno o più guasti su dischi o nodi; la dimensione della replica può essere modificata in base alle esigenze di resilienza o alle risorse disponibili; inoltre, è possibile ricorrere a tecniche avanzate come l'*erasure coding* che, pur riducendo lo spazio occupato, mantiene un livello di protezione dai guasti conforme alla configurazione del pool.

2.1.5 Auto-guarigione

Una delle caratteristiche più innovative di Ceph è la capacità di recuperare in modo autonomo la ridondanza e la coerenza dei dati al verificarsi di un guasto (*solitamente indicata in inglese come self-healing*). Quando un demone OSD diventa irraggiungibile o viene rimosso dal cluster, i monitor aggiornano la mappa dello stato segnando quell'OSD come non disponibile. Gli oggetti che risiedevano su quell'OSD vengono quindi rilevati come sotto-replicati e il sistema avvia automaticamente la replica verso altri OSD ancora attivi, ripristinando il livello di ridondanza previsto. Il processo avviene in modo trasparente per l'utente e senza intervento manuale, minimizzando l'impatto sul servizio. Questo meccanismo è fondamentale per garantire l'alta disponibilità in scenari di produzione.

2.1.6 Distribuzione dei dati

In Ceph, la distribuzione dei dati è governata dall'algoritmo CRUSH (Controlled Replication Under Scalable Hashing) che calcola in modo deterministico dove ogni oggetto debba essere memorizzato, tenendo conto dell'identificativo dell'oggetto, della topologia del cluster (nodi, rack, data-center) e delle regole di posizionamento definite dall'amministratore. Questa architettura elimina la necessità di tabelle centralizzate di lookup, migliorando la scalabilità e le prestazioni del sistema.

La distribuzione presenta le seguenti caratteristiche:

- **Bilanciata**: i dati vengono ripartiti in maniera omogenea tra gli `OSD`, ottimizzando l'utilizzo delle risorse.
- **Deterministica**: qualsiasi client può calcolare autonomamente la posizione di un oggetto, senza bisogno di interrogare un server centrale.
- **Adattiva**: in caso di aggiunta o rimozione di nodi, solo una minima frazione dei dati viene ridistribuita, riducendo i tempi di ribilanciamento.

2.1.7 Pool e Placement Group (PG)

Per capire davvero come Ceph distribuisce e bilancia i dati nel cluster è utile introdurre due concetti chiave: i **pool** e i **placement group** (PG). Questi due livelli logici mediano tra gli oggetti che i client vogliono salvare e i demoni `OSD` che li memorizzano fisicamente.

Pool

Un **pool** è un contenitore logico di dati all'interno del cluster Ceph. Ogni oggetto scritto dal client viene sempre memorizzato dentro un pool, che rappresenta il livello in cui si definiscono le policy di storage. Tra i parametri più importanti impostati a livello di pool troviamo:

- il **tipo di protezione dei dati** (replica tradizionale oppure *erasure coding*);
- il **fattore di replica** (quante copie mantenere di ciascun oggetto);
- il **tipo di applicazione** (ad esempio un pool dedicato a `CephFS`, uno per `RBD`, uno per `RGW`);
- eventuali **regole CRUSH** specifiche per quel pool.

Separare i dati in più pool permette di tenere isolati carichi di lavoro diversi, applicare policy differenti (ad esempio maggiore ridondanza per i dati critici, minore per i dati temporanei) e semplificare la gestione.

Placement Group (PG)

I **placement group** sono il livello intermedio tra il pool e gli `OSD`. Quando un client salva un oggetto:

1. indica il *pool* di destinazione;
2. l'algoritmo CRUSH calcola a quale *PG* di quel pool deve appartenere l'oggetto;

3. lo stesso PG è mappato, sempre tramite CRUSH, a un insieme di `OSD` (uno per la copia primaria e gli altri per le repliche).

In questo modo Ceph non deve gestire la posizione di ogni singolo oggetto, ma solo quella dei PG: ogni PG è come un "secchiello" logico che contiene un certo numero di oggetti e che viene assegnato a un gruppo di `OSD`. Se cambia la composizione del cluster (aggiunta/rimozione di `OSD`), Ceph può ridistribuire i PG in modo controllato, e con essi gli oggetti che contengono.

Scelta del numero di PG

Il numero di PG per ciascun pool è un parametro importante perché influenza:

- il **bilanciamento**: più PG permettono di distribuire i dati in modo più uniforme tra gli `OSD`;
- il **carico di gestione**: troppi PG aumentano l'overhead e il lavoro di monitoraggio del cluster.

La documentazione di Ceph [1] propone una regola empirica per stimare il numero totale di PG del cluster, ad esempio

$$\text{PG\_totali} \approx \frac{\text{numero OSD} \times 100}{\text{fattore di replica}}$$

e poi suddividere questi PG tra i vari pool in base all'uso. Si tratta comunque di una linea guida: in ambienti piccoli conviene restare su valori più contenuti, mentre in cluster più grandi si può aumentare per migliorare la distribuzione.

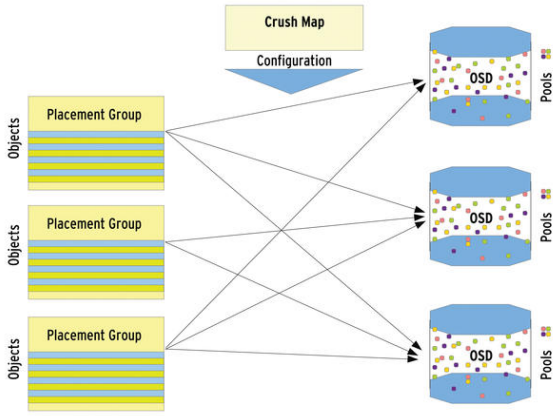

Figura 2.2: Esempio di distribuzione degli oggetti in Ceph: gli oggetti vengono raggruppati in Placement Group (PG) e, sulla base della *CRUSH map* e della relativa configurazione, ogni PG viene mappato in modo deterministico sugli OSD del cluster.

Fonte: adattato da [14].

2.2 MicroCeph

`MicroCeph` è una soluzione leggera sviluppata da Canonical per semplificare il deployment e la gestione di un cluster Ceph. È distribuito come pacchetto **snap** ed è progettato per ridurre la complessità tipica delle installazioni Ceph tradizionali, rendendo possibile l'avvio rapido di cluster anche su hardware minimo o in ambienti sperimentali.

2.2.1 Caratteristiche principali

Tra le funzionalità più rilevanti di MicroCeph si segnalano:

- gestione semplificata dei dischi e del placement dei servizi: MicroCeph automatizza operazioni come l'aggiunta degli OSD, la distribuzione dei demoni MON/MGR/MDS/RGW e l'inizializzazione del cluster

- compatibilità con tutti i protocolli Ceph (RBD, CephFS, RGW), nonché supporto per ambienti di test, home-lab o edge
- forte orientamento verso la riproducibilità: ideale per scenari didattici o sperimentali in cui la rapidità di deploy e la semplicità di gestione sono cruciali.

2.2.2 Sintesi

In sintesi, MicroCeph rappresenta una valida scelta per ambienti didattici, laboratori di ricerca, prototipazione veloce e test di concetti di storage distribuito [5] [6]. Permette di avviare un cluster Ceph in pochi minuti con un overhead minimo, offrendo un buon compromesso tra funzionalità e semplicità. Tuttavia, per scenari di produzione ad alta scala o con requisiti complessi, può essere necessario adottare soluzioni Ceph più tradizionali e personalizzate.

2.3 Kubernetes

Kubernetes (spesso abbreviato in **K8s**) è un sistema *open-source* originariamente sviluppato da Google e ora gestito dalla *Cloud Native Computing Foundation* (CNCF). La sua funzione principale è l'**automazione** del *deployment*, dello *scaling* e della *gestione* delle **applicazioni containerizzate** (tipicamente tramite Docker). Kubernetes agisce come un "sistema operativo" per i cluster di macchine, astraendo l'infrastruttura sottostante e facilitando l'esecuzione coerente dei container su larga scala [7].

Questo strumento è emerso come lo **standard *de facto*** nell'orchestrazione dei container per diversi motivi fondamentali. Tra i principali vi sono la **scalabilità**, che permette l'aumento o la diminuzione automatica del numero di repliche di un'applicazione in base al carico (*Horizontal Pod Autoscaler*); la **resilienza e auto-riparazione (*self-healing*)** che rileva e sostituisce automaticamente i container che falliscono; la **portabilità**, fun-

zionando in ambienti *on-premise*, *cloud pubblici* (es. AWS, Azure, GCP) o *hybrid cloud* senza modifiche significative; e la **gestione dichiarativa**, dove l'utente descrive lo stato desiderato dell'applicazione e Kubernetes si impegna a mantenerlo.

L'architettura di Kubernetes si basa sul modello **Control Plane – Worker Node**. Il **Control Plane**, noto anche come Piano di Controllo o Master, funge da "cervello" del cluster, gestendo lo stato desiderato, pianificando i carichi di lavoro e supervisionando l'intero sistema. Tra i suoi componenti principali vi sono *kube-apiserver*, che funge da interfaccia di comunicazione tra utenti, componenti e nodi, ed *etcd*, l'archivio distribuito che memorizza lo stato del cluster.

I **Worker Nodes**, invece, sono le macchine su cui vengono eseguiti effettivamente i container. Su ciascun nodo operano componenti chiave come *kubelet*, l'agente che applica le istruzioni del Control Plane, e *kube-proxy*, che gestisce la comunicazione di rete tra i pod e i servizi, garantendo il corretto instradamento del traffico all'interno del cluster.

2.3.1 Terminologia

Per facilitare la lettura e la comprensione del testo, di seguito vengono riportate le definizioni dei principali termini tecnici che saranno utilizzati con frequenza nel corso della tesi.

Orchestratore

Un *orchestratore* è uno strumento software che automatizza il deploy, la gestione, il networking e la scalabilità di container o VM in un cluster. Gestisce in modo centralizzato il ciclo di vita delle applicazioni distribuite, assicurando che vengano rispettati lo stato desiderato, le dipendenze e le politiche di bilanciamento del carico.

Container

Un *container* è un'unità leggera e portabile di esecuzione che racchiude un'applicazione insieme a tutte le dipendenze necessarie. A differenza delle macchine virtuali, i container condividono il kernel del sistema operativo host, consentendo avvii rapidi e un utilizzo più efficiente delle risorse.

2.3.2 Varianti di Kubernetes

Oltre alla distribuzione **Kubernetes classica** (spesso definita anche K8s), esistono diverse distribuzioni ottimizzate per specifici casi d'uso, come ambienti *edge*, IoT o sviluppo locale. Le più note sono:

k3s

k3s è una distribuzione altamente **lightweight** e certificata di Kubernetes, sviluppata da Rancher Labs. È progettata per ambienti con risorse limitate o dove la stabilità e la ridotta superficie di attacco sono cruciali (es. *edge computing*, IoT). **k3s** riduce drasticamente i requisiti di memoria e dipendenze eliminando i componenti non essenziali e rimpiazzando *etcd* con un database SQL più leggero (di default SQLite).

k0s

k0s è un'altra distribuzione leggera di Kubernetes, che punta a un'installazione semplice e un'operatività minima senza dipendenze esterne. Include sia il Control Plane che i nodi Worker in un unico pacchetto eseguibile, facilitando l'avvio di cluster temporanei o sperimentali con un overhead ridotto, pur offrendo funzionalità compatibili con l'ecosistema Kubernetes.

	k3s	**k0s**
Filosofia	Leggero ma con pacchetto completo	Minimalista, single binary
Installazione	Richiede alcune dipendenze minime	Nessuna dipendenza esterna obbligatoria
Architettura	Componenti distribuiti ridotti	Control Plane + Worker in un unico binario
Uso tipico	Edge, IoT, laboratori didattici	Laboratori, cluster temporanei, sperimentazioni
Storage dei dati	SQLite di default, etcd opzionale	Etcd opzionale, configurabile

Tabella 2.1: Confronto tra k3s e k0s

2.3.3 Rook

Rook è un operatore open-source progettato per portare i sistemi di storage distribuito, in particolare Ceph, all'interno di un cluster Kubernetes. Attraverso l'uso di Custom Resource Definitions (CRD) e controller dedicati, Rook automatizza la configurazione, il deploy, il dimensionamento, l'aggiornamento e il monitoraggio del cluster Ceph, trasformandolo in un servizio di storage "self-managing" e "self-healing" [8]. In pratica, Rook opera come livello intermedio tra Kubernetes e Ceph: Kubernetes gestisce il ciclo di vita dei container e delle risorse, mentre Rook si occupa di orchestrare le componenti di Ceph (MON, OSD, MGR, MDS, pool, StorageClass) come se fossero servizi nativi di Kubernetes. Grazie a questa integrazione, un'applicazione containerizzata può richiedere volumi persistenti, file system o storage a oggetti direttamente tramite risorse Kubernetes (ad esempio, PersistentVolumeClaim), senza che l'utente debba intervenire manualmente sul cluster Ceph sottostante. L'adozione di Rook risulta particolarmente utile nei contesti in cui lo storage distribuito deve essere fortemente integrato con ambienti containerizzati, offrendo un livello di automazione e scalabilità superiore rispetto ai deployment tradizionali di Ceph.

2.4 Multipass

Multipass è uno strumento sviluppato da Canonical che consente di creare e gestire macchine virtuali leggere basate su Ubuntu in modo rapido e semplificato [9]. Le VM vengono lanciate con un singolo comando, e l'ambiente può essere configurato tramite `cloud-init`, replicando modalità di provisioning [2] tipiche del cloud. Multipass supporta i sistemi host Linux, Windows e macOS, utilizzando rispettivamente QEMU, Hyper-V o altri driver, con l'obiettivo di offrire un'esperienza coerente su diverse piattaforme.

Queste caratteristiche rendono Multipass particolarmente adatto per utilizzi sperimentali, laboratori o ambienti didattici in cui è necessario creare e distruggere VM in modo rapido e controllato.

2.4.1 Precisazioni sull'installazione tramite Snap

Multipass viene distribuito principalmente come pacchetto `Snap`, un sistema di packaging e distribuzione sviluppato da Canonical che consente di installare applicazioni in modo isolato dal sistema host, includendo tutte le dipendenze necessarie. Questa modalità di distribuzione garantisce un'installazione semplificata, aggiornata automaticamente e uniforme su diverse distribuzioni Linux, indipendentemente dal gestore di pacchetti nativo del sistema.

Tale isolamento, però, ha anche alcune conseguenze. Poiché ogni `snap` viene fornito con il proprio insieme di librerie e runtime, può accadere che più applicazioni `snap` installino copie diverse delle stesse dipendenze, con un conseguente aumento dello spazio occupato su disco. Questo fenomeno viene spesso indicato come *isolamento e duplicazione delle dipendenze*: è utile per garantire che ciascun pacchetto funzioni in modo riproducibile e non venga

[2] Nel contesto IT, "provisioning" indica il processo mediante il quale vengono predisposte le risorse infrastrutturali (hardware, rete, storage, macchine virtuali o container) e rese disponibili per l'uso da parte dei sistemi o degli utenti.

rotto da aggiornamenti del sistema host, ma va considerato quando si lavora su macchine con risorse limitate.

Strato di base e content interfaces

Per ridurre il problema della duplicazione, l'ecosistema `Snap` introduce il concetto di *strato di base* (*base snap*), ovvero un'immagine comune (ad esempio `core` o `core22`) che fornisce un ambiente runtime condivisibile tra più `snap`. In questo modo le applicazioni possono appoggiarsi a un set di librerie già presente nel sistema invece di includerle tutte al proprio interno.

In aggiunta, tramite le cosiddette *content interfaces*, uno `snap` può esporre contenuti riutilizzabili da altri `snap`, così da evitare di duplicare asset di grandi dimensioni o dipendenze comuni. Nonostante questi meccanismi di ottimizzazione, è comunque possibile che, rispetto a una installazione tradizionale basata su pacchetti di sistema, l'uso di più `snap` porti a un overhead complessivo maggiore in termini di spazio e di livelli di isolamento.

Confinamento e permessi dei pacchetti Snap

Uno degli aspetti centrali dell'ecosistema `Snap` è il modello di sicurezza basato sul *confinamento* dell'applicazione. In pratica, ogni `snap` viene eseguito in un ambiente controllato, con accesso mediato alle risorse del sistema (file, rete, dispositivi, servizi di sistema). Questo isolamento è implementato tramite meccanismi del kernel come `AppArmor`[3] e `seccomp`[4] , che permettono a `snapd` di applicare una *policy* di sicurezza specifica per ciascuna versione del pacchetto. In questo modo si riduce la superficie d'attacco e si applica il principio del minimo privilegio.

La piattaforma distingue principalmente tre livelli di confinamento:

[3] **AppArmor**: modulo di sicurezza per Linux che applica controlli d'accesso obbligatori tramite profili associati ai singoli programmi, limitandone file, capacità e operazioni consentite.

[4] **seccomp**: funzionalità del kernel Linux (*secure computing mode*) che permette di filtrare o ridurre l'insieme di system call disponibili a un processo.

- **strict**: è l'impostazione predefinita e più sicura; lo `snap` vede solo il suo filesystem privato e può accedere ad altre risorse solo tramite interfacce dichiarate;
- **classic**: lo `snap` gira con accesso molto più ampio al sistema host (simile a un pacchetto tradizionale) ed è riservato a casi d'uso particolari; richiede una revisione specifica nello Snap Store;
- **devmode**: usato in fase di sviluppo/debug, permette di rilassare i controlli per facilitare i test.

Questi livelli sono importanti perché determinano quanto un'applicazione è isolata dal resto del sistema.

Per poter accedere a risorse esterne all'ambiente confinato, gli `snap` dichiarano delle *interfaces* (ad esempio per l'accesso alla home dell'utente, all'audio, alla grafica, a `network`, ecc.). Alcune interfacce vengono collegate automaticamente perché considerate sicure o necessarie al funzionamento dell'applicazione; altre richiedono l'approvazione dell'utente o dell'amministratore di sistema. Questo meccanismo di interfacce e permessi consente di mantenere l'isolamento di base pur permettendo allo `snap` di svolgere i propri compiti.

Il ruolo di `snapd`

Alla base del funzionamento di tutto il sistema c'è `snapd`, il demone che gira in background e che coordina l'intero ecosistema degli `snap`. È `snapd`, infatti, a occuparsi dell'installazione, dell'aggiornamento e dell'eventuale rimozione dei pacchetti, applicando ogni volta le relative policy di sicurezza associate a ciascuna applicazione. Sempre `snapd` gestisce le connessioni tra le interfacce dichiarate dagli `snap` e le risorse reali del sistema, decidendo quali permessi concedere e con quali modalità. Infine, è il componente che mantiene il contatto con lo `Snap Store` per verificare la disponibilità di nuove versioni, di canali di rilascio diversi e per applicare gli aggiornamenti in modo transazionale.

2.4.2 Cos'è una macchina virtuale (VM)

Una *Virtual Machine* (VM) è un ambiente software che emula il comportamento di un computer fisico, permettendo di eseguire un sistema operativo e applicazioni in modo isolato dal sistema host. Ogni VM dispone di risorse virtualizzate — come CPU, memoria, disco e interfacce di rete — fornite e gestite da un *hypervisor*, ossia un livello di astrazione che coordina l'accesso dell'hardware fisico tra più ambienti virtuali.

Questo approccio consente di eseguire più macchine indipendenti sullo stesso hardware fisico, migliorando l'efficienza nell'utilizzo delle risorse e semplificando la sperimentazione di sistemi complessi. Nel contesto di questo lavoro, le VM sono utilizzate per simulare i nodi di un cluster Ceph in modo controllato e riproducibile, senza la necessità di disporre di più server fisici dedicati.

2.5 Strumenti di supporto

Oltre alle tecnologie principali impiegate per la virtualizzazione e l'orchestrazione, in questa tesi vengono utilizzati e citati alcuni strumenti complementari che supportano e semplificano le operazioni di configurazione e gestione dell'ambiente sperimentale.

2.5.1 cephadm

cephadm è uno strumento a riga di comando progettato per facilitare il deployment, la gestione e l'aggiornamento di un cluster Ceph [4]. In particolare, cephadm permette di eseguire il bootstrap di un cluster partendo da un singolo nodo, automatizzare la distribuzione dei demoni Ceph dentro container, aggiungere nuovi nodi al cluster e gestire il ciclo di vita dell'intero sistema.

Il suo punto di forza principale è la riduzione della complessità del deploy rispetto a un'installazione completamente manuale, grazie alla gestione automatica dei container e della configurazione iniziale;

Nell'ambito della sperimentazione con questo studio, cephadm rappresenta l'elemento che consente di armonizzare un approccio "quasi-manuale" con un livello di automazione sufficiente a rendere riproducibili i test, pur mantenendo una buona visibilità sui processi di deploy e configurazione.

2.5.2 cloud-init

`cloud-init` è lo strumento standard per l'inizializzazione automatica delle istanze cloud o virtuali al primo avvio [10]. Sviluppato per ambienti *cloud-native*, è supportato nativamente da Ubuntu e da molte altre distribuzioni Linux. In particolare, cloud-init consente non solo di configurare hostname, utenti e chiavi SSH, ma anche di installare automaticamente pacchetti di base e di eseguire comandi personalizzati al primo avvio della macchina. In questo modo si garantisce che ogni VM parta da una configurazione identica e predefinita, favorendo la riproducibilità dell'ambiente sperimentale.

2.5.3 Netplan

Netplan è l'utilità introdotta da Ubuntu (a partire dalla versione 17.10) per la configurazione dichiarativa della rete tramite file YAML. La sua funzione principale è astrarre la configurazione delle interfacce di rete, delegando a sottosistemi come `systemd-networkd` l'applicazione concreta. Nel contesto della sperimentazione con MicroCeph, Netplan è stato utilizzato per configurare in modo statico gli indirizzi IP di tutti i nodi del cluster.

2.5.4 Samba

`Samba` è un insieme di strumenti open-source che implementano il protocollo `SMB/CIFS`, consentendo l'interoperabilità tra sistemi Linux/Unix e

Windows. Nel nostro ambiente di prova il servizio `Samba` non è stato collocato su una macchina dedicata, ma direttamente sulla *prima VM del cluster Ceph*, che funge quindi sia da nodo del cluster sia da punto di accesso per i client Windows.

La sequenza è la seguente:

- le VM del cluster costituiscono l'infrastruttura `Ceph` (con i servizi `MON`, `MDS`, `OSD`, ...) e mettono a disposizione il filesystem distribuito `CephFS`;
- sulla **prima VM del cluster** viene montato `CephFS` in spazio utente tramite `ceph-fuse` (ad esempio in `/mnt/cephfs`);
- sulla stessa VM viene configurato `Samba` per esportare in rete una o più sottodirectory di `CephFS` (ad esempio `/mnt/cephfs/progetto`);
- i client `Windows`, che sono macchine esterne al cluster, accedono a queste directory tramite il protocollo `SMB` come a normali condivisioni di rete (ad esempio `\\ceph-vm1\cephshare`).

In questo modo i client `Windows` non interagiscono direttamente con `CephFS` né devono conoscere la sua configurazione interna: vedono soltanto una share `Samba` esposta dalla prima VM del cluster, che svolge il ruolo di "traduttore" tra il filesystem distribuito e il protocollo `SMB`.

Nota per ambienti di produzione. Nel setup descritto, `Samba` è stato eseguito direttamente sulla prima VM del cluster `Ceph` per semplificare l'ambiente sperimentale. Tuttavia, in un contesto di produzione è generalmente consigliabile separare i ruoli: i nodi del cluster dovrebbero essere dedicati ai servizi `Ceph`, mentre i servizi di esposizione verso i client (`Samba`, `NFS-Ganesha`, ecc.) dovrebbero risiedere su macchine/front-end distinti che montano `CephFS` e lo riesportano. Questa separazione riduce la contesa delle risorse, migliora la sicurezza e permette di effettuare manutenzione sui servizi di condivisione senza impattare direttamente sul cluster di storage.

Capitolo 3

Installazione semi-manuale

In questo primo approccio il cluster Ceph viene realizzato mediante l'utilizzo dell'utilità `cephadm`, che gestisce il lifecycle del cluster (bootstrap, deploy dei demoni, aggiunta di nodi), anziché la tradizionale configurazione totalmente manuale dei singoli componenti Ceph. Si parte con il comando di bootstrap offerto da `cephadm` su un nodo iniziale, si prosegue con l'aggiunta manuale degli altri host e dispositivi, la creazione dei pool, la gestione di OSD, MON e altri servizi, e l'esecuzione dei test di funzionamento. Questo metodo consente comunque un controllo granulare di molti aspetti del deployment, pur richiedendo un significativo lavoro manuale e la conoscenza approfondita dell'architettura Ceph. La scelta di questo approccio nel contesto della tesi è motivata dal desiderio di comprendere in profondità il funzionamento di Ceph con un minimo livello di automazione — ossia capire come cephadm gestisce molti aspetti, ma lascia comunque all'amministratore gran parte delle decisioni e della configurazione.

3.1 Procedura e implementazione

L'ambiente sperimentale è stato configurato all'interno di `WSL (Windows Subsystem for Linux)`, che consente di eseguire un ambiente Linux nativo su sistema operativo Windows senza ricorrere a una virtualizzazione com-

pleta. Questa scelta ha permesso di ridurre i tempi di configurazione e di eseguire i test direttamente su una macchina host Windows, mantenendo al contempo la compatibilità con gli strumenti Linux necessari al deploy di Ceph.

Il sistema operativo utilizzato all'interno di WSL è `Ubuntu 24.04.1 LTS`.

Dopo aver predisposto l'ambiente di test su `WSL`, si è proceduto con l'installazione di `cephadm`, prelevato direttamente dai repository ufficiali di Ubuntu tramite il gestore di pacchetti `APT`.

3.2 Bootstrap del nodo

Completata l'installazione, si è proceduto con la fase di *bootstrap*, che inizializza il primo nodo del cluster, installa i servizi fondamentali (`MON`, `MGR` e `OSD`) e configura automaticamente il sistema di rete e autenticazione. Il comando utilizzato è stato:

```
sudo cephadm bootstrap --mon-ip <indirizzo_IP_locale>
```

Durante questa fase, `cephadm`:

- crea un demone `MON` (Monitor) e un demone `MGR` (Manager) per il nuovo cluster sul nodo locale;
- crea e distribuisce automaticamente le chiavi SSH e amministrative necessarie, genera un file minimo di configurazione del cluster e colloca tali elementi nella directory di configurazione (`/etc/ceph/`), garantendo che il nodo bootstrap sia pronto per operare come punto iniziale del cluster.
- aggiunge l'etichetta `_admin` all'host bootstrap: per default, qualsiasi host con tale etichetta riceverà anche una copia di `/etc/ceph/ceph.conf` e `/etc/ceph/ceph.client.admin.keyring`.

Una volta terminato il bootstrap, l'installazione Ceph a nodo singolo risulta operativa.
Il corretto avvio dei servizi è verificabile tramite il comando:

```
sudo ceph -s
```

che restituisce un riepilogo dello stato del cluster e dei demoni attivi.

3.3 Aggiunta di dischi

Una volta completato il bootstrap, la fase successiva ha riguardato l'integrazione dello storage. In questo contesto, "aggiunta di dischi" indica la preparazione e il riconoscimento dei dispositivi che ospiteranno le unità di memorizzazione (OSD, Object Storage Daemon).

In particolare, è stato necessario verificare che il dispositivo o il file-loop scelto fosse libero da filesystem, partizioni o configurazioni precedenti, poiché cephadm controlla che il disco soddisfi determinati criteri prima di trasformarlo in OSD. Successivamente, il comando

```
ceph orch apply osd --all-available-devices
```

ha consentito di aggiungere automaticamente tutti i dispositivi idonei al cluster, semplificando l'espansione dello storage. Nel mio caso, trattandosi di un ambiente sperimentale su macchina singola, si è fatto ricorso a file-loop simulati come dispositivi OSD.

3.4 Verifica dello stato

Una volta completato il deploy mediante cephadm e con il nodo iniziale in funzione, è stata effettuata una verifica dello stato del cluster per accertare il corretto avvio dei demoni e la disponibilità dello storage.
L'esecuzione del comando

```
sudo ceph -s
```

ha permesso di ottenere un sommario dello stato operativo del cluster: lo stato del quorum dei monitor, il numero di OSD attivi e in stato "up & in", e la presenza di eventuali errori o warning. Se non vengono rilevati problemi

nel cluster, la voce *health* restituita dal comando risulta "HEALTH_OK", ad indicare che tutti i demoni sono attivi, in quorum e senza avvisi critici.

Cephadm include un'interfaccia grafica di amministrazione, accessibile via browser web e ospitata dal demone `ceph-mgr`. Questo strumento consente la visualizzazione dello stato generale del cluster, delle prestazioni, delle risorse e dei log, offrendo anche funzionalità di gestione come definizione di utenti, ruoli, esportazione di metriche e monitoraggio in tempo reale.

Di seguito è riportato uno schema riassuntivo della struttura ottenuta, utile a visualizzare i servizi effettivamente attivi al termine del deploy:

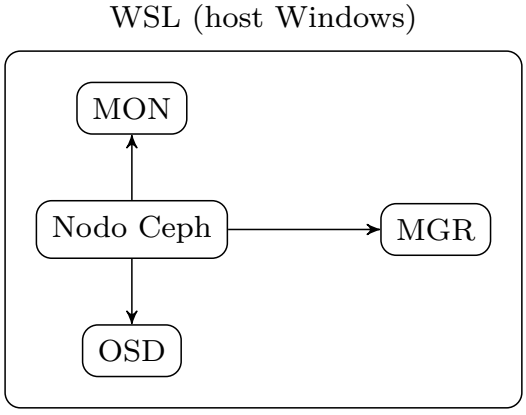

Figura 3.1: Struttura del cluster Ceph a nodo singolo eseguito in WSL, con il nodo di bootstrap che ospita i demoni principali (MON, MGR e OSD).

3.5 Eventuale aggiunta di nodi

In questo progetto non è stata effettuata l'estensione reale del cluster a più host, perché la sperimentazione è stata poi svolta direttamente con Multipass. In uno scenario ideale, l'espansione di un cluster gestito con cephadm prevede l'aggiunta di nuovi host che si uniscono al nodo di bootstrap iniziale: ogni nuovo nodo deve essere preparato con le chiavi SSH del cluster, etichettato correttamente e registrato tramite il comando

```
ceph orch host add <nome_host> <indirizzo_IP>
```

Il comando `ceph-orch` consente di gestire i nodi e i demoni del cluster, ad esempio per aggiungere nuovi host, applicare servizi o verificare lo stato dell'orchestratore.

3.6 Difficoltà incontrate

- **Complessità intrinseca del sistema**: sebbene `cephadm` semplifichi notevolmente il deploy, `Ceph` rimane una piattaforma distribuita articolata con numerosi servizi (come `MON`, `MGR`, `OSD`, `MDS` e `RGW`) che devono operare in modo coordinato. Questo livello di complessità può rappresentare un ostacolo significativo soprattutto per chi si avvicina per la prima volta al sistema.
- **Necessità di competenze sistemistiche Linux**: Cephadm è progettato per ambienti Linux; è quindi necessario avere familiarità con l'amministrazione di sistemi Linux, la gestione dei pacchetti, la configurazione di rete, i permessi e la sicurezza del sistema.

3.7 Risultati e osservazioni

L'adozione di cephadm ha permesso di ridurre drasticamente i tempi di deploy rispetto a un'installazione completamente manuale del cluster Ceph. Su un singolo host, è stato possibile avviare il bootstrap e ottenere un nodo operativo in pochi minuti, evidenziando la rapidità dell'approccio.

Durante l'esperimento, il comando `ceph -s` ha restituito lo stato "HEALTH_OK", confermando che tutti i demoni essenziali (MON, MGR, OSD) risultavano attivi e in quorum. In aggiunta, la dashboard grafica integrata (accessibile via browser) ha fornito una panoramica immediata delle risorse e delle operazioni del cluster, semplificando la verifica visiva dell'infrastruttura.

Tuttavia, l'esperienza ha anche confermato che, nonostante la semplificazione offerta da cephadm, persiste un significativo impatto iniziale in termini

di curva di apprendimento. Per un utente privo di esperienza con Ceph, concetti quali quorum, placement group e bilanciamento automatico richiedono comunque tempo per essere compresi e valutati correttamente.

Un'ulteriore osservazione riguarda l'ambiente di test utilizzato: operando su un singolo host con risorse limitate, è emerso che la fase di provisioning e configurazione delle macchine virtuali rappresenta ancora un collo di bottiglia nelle iterazioni di test. Sebbene cephadm consenta velocità elevate, la disponibilità di risorse hardware rimane un fattore critico nella sperimentazione.

Capitolo 4

Installazione automatizzata con Multipass e MicroCeph

Il secondo approccio analizzato in questa tesi prevede l'utilizzo di **MicroCeph**, una versione semplificata e containerizzata di Ceph sviluppata da Canonical, combinata con l'uso di **Multipass** per la creazione di macchine virtuali leggere e un orchestratore creato appositamente per questi due strumenti. L'obiettivo di questo approccio è ridurre la complessità legata all'installazione manuale del cluster e rendere più semplice la gestione e la sperimentazione in ambienti virtualizzati.

4.1 L'orchestratore sviluppato

Nel contesto della sperimentazione, si è ritenuto opportuno sviluppare un proprio strumento di automazione per la creazione e gestione del cluster Ceph, al fine di ridurre la complessità delle operazioni manuali e migliorare la ripetibilità degli esperimenti.
È stato realizzato quindi `microceph-cluster-orchestrator`, un progetto software in linguaggio `Python` e pubblicato su GitHub[1]. Il programma rappresenta un *prototipo funzionale* con finalità sperimentali e didattiche, volto

[1] `https://github.com/GiacomoBiagioni/microceph-cluster-orchestrator`

a verificare la possibilità di automatizzare la creazione di cluster Ceph in ambienti virtualizzati.

L'orchestratore si occupa di automatizzare l'intero processo di deploy di un cluster basato su `MicroCeph`, sfruttando `Multipass` come piattaforma di virtualizzazione leggera. Attraverso una serie di comandi e script, il programma crea in modo dinamico le macchine virtuali necessarie, assegna configurazioni di rete statiche, installa e inizializza `MicroCeph`, e unisce i nodi in un cluster funzionante con minima interazione manuale.

4.1.1 Struttura

L'orchestratore è organizzato secondo un'architettura modulare, basata su componenti denominati `manager`. Ogni modulo ha la responsabilità di gestire una specifica parte del processo di deploy e di configurazione del cluster, favorendo la separazione delle funzionalità e una maggiore manutenibilità del codice.

In particolare, il modulo `fs_manager.py` si occupa della configurazione del filesystem all'interno delle macchine virtuali, garantendo che ciascun nodo sia predisposto in modo coerente per ospitare i dati di Ceph. Un secondo componente è dedicato all'interazione con `Multipass`, e gestisce la creazione, la rimozione e l'inizializzazione delle VM attraverso comandi eseguiti in modo programmato. Infine, un `cluster_manager` coordina l'insieme dei nodi del cluster, eseguendo operazioni di unione, inizializzazione e controllo dello stato dei servizi Ceph distribuiti.

Questa struttura modulare consente di mantenere il codice facilmente estendibile: ogni manager può essere aggiornato o sostituito senza influire sugli altri componenti, rendendo l'orchestratore adattabile a futuri sviluppi o a configurazioni di cluster differenti.

4.2 Architettura del sistema

L'architettura del sistema sviluppato si basa su una stretta integrazione tra i tre componenti principali.

L'orchestratore rappresenta il livello di controllo e coordinamento dell'intero sistema e attraverso una serie di script, interagisce con l'interfaccia a riga di comando di `Multipass` per creare, configurare e distruggere le macchine virtuali che costituiranno i nodi del cluster. In seguito alla fase di inizializzazione, vengono assegnati indirizzi IP statici, definite le interfacce di rete interne ed esterne e predisposti i file di configurazione necessari per l'avvio del cluster.

Una volta creati i nodi virtuali, l'orchestratore procede all'installazione e alla configurazione dei pacchetti necessari per il corretto funzionamento di MicroCeph.

Le relazioni tra i componenti sono di tipo gerarchico e questa suddivisione a livelli consente una chiara separazione delle responsabilità: `Multipass` gestisce le risorse virtuali, `MicroCeph` configura i servizi di storage distribuito, mentre l'orchestratore coordina e automatizza l'intero processo di provisioning.

4.3 Flusso di deploy

L'orchestratore si esegue con questo comando:

```
python main.py deploy --default
```

Il comando `deploy` crea le VM con i parametri predefiniti, esegue il bootstrap di MicroCeph sul primo nodo, fa entrare gli altri nodi nel cluster e prepara Samba.

Per personalizzare il deploy si possono usare i parametri mostrati in Tabella 4.1.

Parametro	Descrizione
`--nodes <N>`	Numero di nodi del cluster MicroCeph da creare con Multipass (default: `2`).
`--base-name <nome>`	Prefisso usato per nominare le VM generate (default: `ceph-node`).
`--cpus <n>`	Numero di vCPU assegnate a ciascuna VM (default: `2`).
`--ram <dimensione>`	Quantità di RAM per VM, es. `2G`, `4G` (default: `2G`).
`--disk <dimensione>`	Dimensione del disco della VM (default: `10G`).
`--os <release>`	Immagine Ubuntu usata da Multipass (default: `22.04`).
`--with-client`	Crea anche la VM Linux esterna `ceph-client` per i test di montaggio (default: disabilitato).
`--debug`	Abilita l'output esteso per il troubleshooting (default: disabilitato).

Tabella 4.1: Parametri principali per l'esecuzione dell'orchestratore.

Esempi d'uso:

- `python main.py deploy --default --with-client` per avere cluster + VM client;
- `python main.py deploy --nodes 5 --ram 4G --cpus 4 --disk 20G` per un ambiente più grande.

In caso di errori durante una delle fasi il programma interrompe l'esecuzione per evitare di lasciare l'ambiente in uno stato incoerente. Un'eventuale riesecuzione del comando di `deploy` effettua alcuni controlli di base per capire se una parte del setup è già stata completata (ad esempio se le VM esistono già o se il nodo di bootstrap è stato creato), ma al momento non è ancora presente una gestione avanzata degli errori e dei ripristini.

Al termine dell'esecuzione del comando di deploy si ottiene quindi un cluster MicroCeph funzionante a N nodi; se è stata abilitata l'opzione per il client, sarà inoltre presente una VM Linux esterna già collegata alla stessa rete e pronta per i test di accesso al filesystem.

4.4 Flusso operativo dell'orchestratore

In questa sperimentazione è stato realizzato un ambiente composto da:

- **3 VM** create con `Multipass` che costituiscono il cluster `MicroCeph` (node-1, node-2, node-3);
- **1 VM Linux aggiuntiva** (non parte del cluster) usata come client per verificare l'accesso a `CephFS` da un sistema Linux;
- **1 macchina Windows** (il PC dell'operatore) usata per verificare l'accesso tramite `Samba`, esposto dalla prima VM del cluster.

L'orchestratore esegue una serie di passaggi ben definiti, che vengono riportati nello schema seguente e descritti nelle sezioni successive.

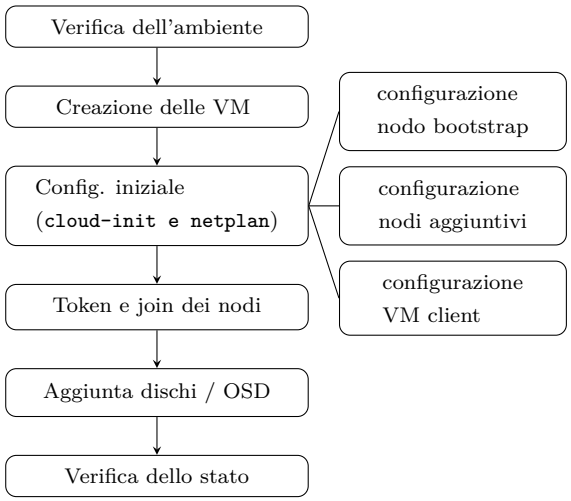

Figura 4.1: Sequenza delle operazioni eseguite dallo script di orchestrazione per la creazione del cluster MicroCeph.

4.4.1 Verifica dell'ambiente

La prima fase controlla che l'host che esegue lo script disponga di un hypervisor funzionante e compatibile con `Multipass`. Il modulo `hypervisor_check.py` esegue quindi una validazione preliminare (presenza di `multipass`, stato del servizio, capacità di avviare istanze). Se questo controllo non va a buon fine, il deploy viene interrotto, perché le VM successive dipendono da `Multipass`.

4.4.2 Creazione delle VM con Multipass

Il modulo `multipass_manager.py` crea, in maniera ripetibile, le 4 VM necessarie:

Nome VM	**Ruolo / Descrizione**
`ceph-node-1`	Nodo primario del cluster MicroCeph.
`ceph-node-2`	Nodo del cluster MicroCeph.
`ceph-node-3`	Nodo del cluster MicroCeph.
`ceph-client`	VM Linux esterna al cluster, usata per i test di accesso al filesystem CephFS.

Tabella 4.2: VM create dallo script di orchestrazione e funzioni svolte nell'ambiente di sperimentazione. (Per nodo primario si intende nodo su cui viene eseguito il comando di bootstrap)

Per ciascuna istanza vengono passati i parametri di default pre-configurati nel programma (numero di vCPU, RAM, disco) e soprattutto, viene associato un file `cloud-init` che si occuperà delle configurazioni iniziali.

Contestualmente viene configurata anche la rete interna: tramite la modifica del file `netplan` all'interno delle VM, lo script assegna indirizzi IP statici alle tre VM del cluster, in modo che i nodi si possano raggiungere sempre con lo stesso indirizzo anche dopo un riavvio.

4.4.3 Configurazione iniziale tramite cloud-init

Al primo avvio di ciascuna VM, gli script `cloud-init` installano i pacchetti di base e predispongono il sistema per l'installazione di MicroCeph. Questa fase serve a portare tutte le VM a uno stato omogeneo così che lo script di orchestrazione possa collegarsi in SSH e lanciare i comandi successivi.

Nel caso della prima VM del cluster, `cloud-init` esegue inoltre i comandi di inizializzazione di MicroCeph, in particolare il `bootstrap` del cluster, così da creare il cluster di partenza direttamente al primo avvio.

Da questo momento esiste un cluster MicroCeph funzionante, ma composto da un solo nodo.

Differenziazione dei file `cloud-init`

Nel progetto sono previsti tre file `cloud-init` distinti, ciascuno pensato per un ruolo preciso all'interno dell'infrastruttura. Tutti eseguono una serie di operazioni comuni (aggiornamento dei pacchetti, installazione dei prerequisiti di base, configurazione dell'utente e dell'accesso SSH) per portare le VM in uno stato noto e raggiungibile dallo script di orchestrazione.

- **Cloud-init per il nodo di bootstrap**: è il file usato dalla prima VM del cluster, quella che deve inizializzare MicroCeph. Oltre alle operazioni comuni, questo script esegue anche il `bootstrap` del cluster MicroCeph, creando cioè il primo cluster funzionante già al primo avvio della macchina. È quindi l'unico dei tre che contiene i comandi di inizializzazione del cluster.

- **Cloud-init per i nodi aggiuntivi del cluster**: questo file prepara la VM a essere aggiunta a un cluster MicroCeph già esistente, senza eseguire il `bootstrap`. Queste VM vengono soltanto portate in uno stato pronto per essere unite al nodo principale: sarà lo script di orchestrazione a collegarsi in SSH e ad eseguire i comandi di `microceph cluster add` o equivalenti.

- **Cloud-init per la VM client**: il terzo file è pensato per una macchina che non fa parte del cluster ma che deve poterlo utilizzare (ad esempio per montare `CephFS` o per testare l'accesso ai servizi esposti). Si limita quindi a installare gli strumenti necessari lato client e a configurare l'accesso verso il cluster, senza installare o inizializzare MicroCeph sulla VM stessa.

Preparazione del servizio Samba

Nella stessa fase di configurazione iniziale lo script predispone anche il servizio `Samba` sulla prima VM, cioè quella su cui è stato eseguito il `bootstrap` del cluster. Dopo aver installato i pacchetti necessari, lo script crea un utente dedicato all'accesso SMB e aggiorna il file `/etc/samba/smb.conf` in modo che la macchina sia in grado di accettare connessioni in ingresso e di associare in seguito una condivisione alla directory di `CephFS` montata. Al termine viene riavviato (o abilitato) il servizio `smbd`, così che la VM sia già pronta a ricevere connessioni senza ulteriori configurazioni manuali.

4.4.4 Generazione dei token e join degli altri nodi

Sempre su `ceph-node-1`, lo script genera i token per permettere agli altri due nodi di entrare nel cluster:

```
sudo microceph cluster add node-2
sudo microceph cluster add node-3
```

I token restituiti da questi comandi vengono poi usati, via SSH, su `ceph-node-2` e `ceph-node-3`:

```
sudo microceph cluster join <token-per-node-2>
sudo microceph cluster join <token-per-node-3>
```

Al termine di questa fase il cluster è effettivamente a 3 nodi e ogni macchina risulta registrata con il proprio indirizzo IP statico.

4.4.5 Aggiunta dei dischi/OSD

Per rendere il cluster utilizzabile viene aggiunto lo storage su ciascun nodo. Lo script esegue, su tutte e tre le VM del cluster:

```
sudo microceph disk add loop,4G,1
```

Questo comando fa sì che MicroCeph crei automaticamente un dispositivo di loop da 4 GB sul nodo corrente e lo utilizzi come disco per un nuovo OSD. L'allocazione del disco virtuale e la relativa registrazione nel cluster avvengono quindi in modo completamente automatico, senza dover predisporre a mano un secondo disco nella VM.

4.4.6 Verifica dello stato

Una volta che i tre nodi si sono uniti e che i dischi sono stati aggiunti, lo script richiama:

```
sudo microceph status
```

e/o

```
sudo ceph status
```

per verificare che tutti i demoni (`mon`, `mgr`, `mds`, `osd`) siano attivi su ciascuno dei tre nodi e che lo stato del cluster sia riportato come "HEALTH_OK". Questo conferma che il flusso di deploy ha portato a un cluster a 3 nodi funzionante.

4.4.7 VM Linux client e accesso da Windows

La quarta VM (`ceph-client`) non entra nel cluster: viene usata per testare l'accesso al filesystem dal punto di vista di un host Linux esterno. Su questa macchina è stato effettuato l'accesso al filesystem tramite il mount `cifs`[2],

[2]Il mount CIFS utilizza il client SMB/CIFS del kernel Linux per accedere a una condivisione esportata via Samba.

verificando che il cluster fosse raggiungibile e che la condivisione risultasse montabile.

Per la verifica da Windows, invece, è stato usato il PC dell'operatore. Su `ceph-node-1` (la prima VM del cluster) è stato installato e configurato `Samba` per riesportare una directory di `CephFS`, in modo che la macchina Windows possa accedere ai dati tramite il protocollo SMB come a una normale condivisione di rete. In questo modo si dimostra che i dati scritti nel filesystem distribuito sono accessibili anche da sistemi eterogenei.

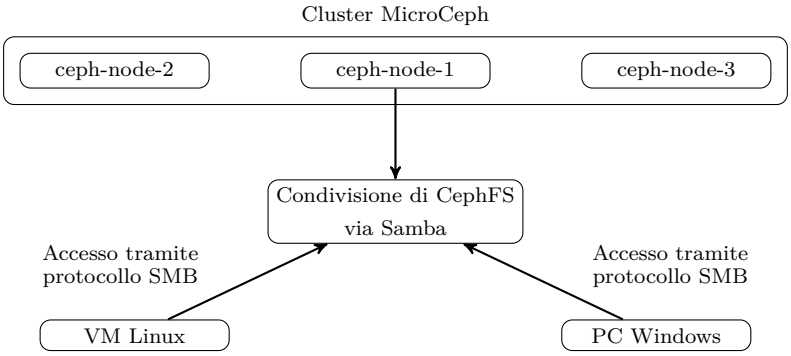

Figura 4.2: Accesso eterogeneo a CephFS tramite condivisione Samba pubblicata dal nodo `ceph-node-1`.

4.5 Difficoltà incontrate

Durante la fase di sviluppo e sperimentazione, sono emerse diverse difficoltà, riconducibili sia ad aspetti software che alle limitazioni dell'ambiente di test. Le problematiche principali possono essere riassunte come segue:

- **Gestione dell'output di Multipass**: poiché l'interazione con l'hypervisor avviene tramite comandi eseguiti da `Python`, i messaggi non standard o di errore prodotti da `Multipass` possono generare incoerenze nell'elaborazione dell'output e interrompere il flusso di esecuzione.
- **Gestione degli errori *runtime***: in alcuni casi, errori durante la fase di deploy richiedevano il reset della macchina virtuale coinvolta

e il riavvio della procedura di test, rallentando il ciclo di sviluppo e verifica.

- **Limitazioni hardware**: le risorse disponibili sulla macchina host non consentivano di gestire cluster di grandi dimensioni, limitando il numero di nodi virtualizzabili e influenzando la scalabilità degli esperimenti.

- **Tempi di provisioning**: la creazione e l'inizializzazione delle VM richiedevano tempi non trascurabili, specialmente in fase di testing iterativo, rallentando la sperimentazione e la raccolta dei risultati.

4.6 Vantaggi e svantaggi

Questo approccio ha mostrato diversi punti di forza, specialmente in un contesto di sperimentazione e apprendimento. La possibilità di automatizzare gran parte del processo di deploy riduce sensibilmente la complessità operativa rispetto all'installazione manuale, rendendo l'esperienza più accessibile anche a chi si avvicina per la prima volta alla gestione di un cluster Ceph. La natura portabile di `Multipass`, compatibile con i principali sistemi operativi, consente inoltre di riprodurre gli esperimenti su piattaforme diverse mantenendo un comportamento coerente, mentre l'integrazione con `MicroCeph` semplifica ulteriormente la configurazione dei nodi e la gestione dei demoni del cluster.

D'altro canto, l'approccio presenta anche alcune limitazioni strutturali. L'utilizzo di macchine virtuali locali, seppur leggere, impone un consumo di risorse che cresce rapidamente con l'aumentare del numero di nodi, limitando così la possibilità di testare configurazioni su larga scala. Le prestazioni risultano inevitabilmente inferiori rispetto a un'installazione su hardware dedicato, e il livello di astrazione introdotto da `MicroCeph` riduce la possibilità di intervenire su parametri avanzati o scenari non previsti. L'orchestratore stesso, essendo un prototipo sperimentale, non dispone ancora di meccanismi

di controllo degli errori e di gestione automatica dei fallimenti comparabili a quelli di strumenti di produzione.

4.7 Risultati e osservazioni

L'esecuzione del processo di deploy ha confermato l'efficacia dell'orchestratore nel creare automaticamente un cluster `MicroCeph` completamente operativo nell'ambiente di prova. Al termine della procedura, il sistema risultava funzionante e accessibile sia da una VM Linux esterna sia da una macchina Windows, a dimostrazione della corretta configurazione dei servizi di rete e del filesystem distribuito. D'altra parte, essendo ancora una soluzione realizzata ad hoc e in una fase iniziale di sviluppo, il codice era più esposto a rischi legati a errori non gestiti, piccole incongruenze tra le fasi del deploy e bug introdotti dalla novità dell'implementazione, non fornendo quindi un approccio immediatamente pronto per la produzione.

Dal lato Linux, l'accesso al filesystem è stato effettuato dalla VM `ceph-client`, cioè la macchina virtuale esterna al cluster, montando la condivisione SMB esposta da `ceph-node-1` tramite il client `cifs`. In questo modo la VM Linux ha potuto raggiungere i dati che risiedono in `CephFS` pur non facendo parte del cluster e senza dover montare direttamente `CephFS`.

Dal lato Windows, l'accesso è avvenuto dal PC dell'operatore verso la stessa condivisione `Samba` configurata sulla prima VM del cluster (quella su cui è stato eseguito il `bootstrap`), rendendo disponibile il contenuto di `CephFS` anche ai client eterogenei tramite protocollo SMB.

Questi risultati evidenziano la stabilità complessiva del sistema e la coerenza delle configurazioni generate automaticamente. Pur trattandosi di un ambiente prototipale, il cluster ha dimostrato di poter riprodurre in modo affidabile il comportamento di un'infrastruttura Ceph reale, confermando la validità dell'approccio e l'efficacia delle soluzioni adottate per l'automazione del deploy.

Capitolo 5

Installazione automatizzata tramite Kubernetes e Rook

Il terzo approccio prevede l'analisi di un cluster Kubernetes assistito da Rook, utilizzato come operatore per orchestrare un cluster Ceph. Rook consente di definire, distribuire e gestire risorse di storage distribuito direttamente all'interno dell'ecosistema Kubernetes, trasformando i demoni Ceph in oggetti containerizzati e gestiti dal *control-plane*[1]. In questa modalità, le applicazioni containerizzate possono utilizzare storage persistente (a blocchi, a file o a oggetti) gestito da Ceph tramite risorse Kubernetes come `StorageClass` e `PersistentVolumeClaim`, senza la necessità di intervenire manualmente sulla configurazione del cluster. L'interesse verso questo approccio nasce dalla crescente diffusione delle architetture basate su container e dalla volontà di valutare la possibilità di integrare lo storage distribuito in modo nativo con i sistemi di orchestrazione applicativa.

[1] Il *control-plane* di Kubernetes è l'insieme dei componenti che gestiscono lo stato globale del cluster, tra cui l'API Server, lo Scheduler, il Controller Manager e `etcd`.

5.1 Architettura del sistema

L'architettura del metodo basato su Kubernetes e Rook si articola in tre livelli principali: il cluster Kubernetes, l'operatore Rook e il cluster Ceph containerizzato.

Il **cluster Kubernetes** comprende nodi con ruoli distinti: il control-plane, responsabile della gestione dello stato globale del cluster attraverso componenti come l'API Server, lo Scheduler, il Controller Manager e `etcd`[2], e i nodi worker, che eseguono le applicazioni containerizzate e i demoni Ceph orchestrati da Rook. Questa separazione permette di centralizzare la gestione dello stato e di distribuire il carico computazionale e di storage sui nodi worker.

Il **Rook Operator** funge da orchestratore per il cluster Ceph. Automatizza il deploy, la configurazione e il monitoraggio dei demoni Ceph trasformandoli in container gestiti dal control-plane di Kubernetes. In questo modo, le risorse di storage distribuito diventano oggetti nativi di Kubernetes, integrandosi perfettamente con le funzionalità del cluster.

Il **cluster Ceph containerizzato** fornisce storage distribuito accessibile dalle applicazioni. I componenti principali includono: **MON, OSD, MDS e RGW**: rispettivamente il monitor del cluster, i daemon di storage, il server dei metadati per CephFS e il gateway per interfacce a oggetti come S3. Questi elementi collaborano per garantire la resilienza, la scalabilità e l'alta disponibilità dei dati. La containerizzazione dei demoni consente di sfruttare la gestione nativa di Kubernetes, riducendo la complessità operativa e semplificando l'integrazione con applicazioni cloud-native.

5.2 Integrazione storage-orchestrazione

L'integrazione tra Kubernetes e Ceph tramite Rook permette alle applicazioni containerizzate di accedere a storage persistente senza la necessità di

[2]Il database chiave-valore distribuito che memorizza lo stato del cluster.

configurazioni manuali sul cluster. Rook espone le risorse di Ceph attraverso oggetti nativi di Kubernetes, come `StorageClass`, `PersistentVolume` e `PersistentVolumeClaim`, consentendo agli sviluppatori di richiedere spazio di storage in maniera dichiarativa e automatica.

Le `StorageClass` definiscono le caratteristiche dello storage, come il tipo di dispositivo, la politica di replica e le funzionalità avanzate di Ceph, mentre le `PersistentVolume` rappresentano le unità effettive di storage disponibili nel cluster. Le applicazioni accedono a queste risorse tramite `PersistentVolumeClaim`, richiedendo spazio persistente senza preoccuparsi della sua allocazione fisica o della gestione dei demoni Ceph.

Questo modello consente di utilizzare diversi tipi di storage, come blocchi, file e oggetti, direttamente all'interno del cluster Kubernetes. In particolare, Ceph RBD (RADOS Block Device) fornisce volumi a blocchi altamente scalabili, CephFS offre filesystem distribuiti condivisi tra più pod, e RADOS Gateway permette l'accesso a oggetti tramite protocolli compatibili con S3 o Swift.

In questo modello la capacità del sistema di storage di esporre un'interfaccia compatibile sia con S3 sia con Swift è fortemente consigliata per due motivi principali: da un lato favorisce l'interoperabilità applicativa, poiché molte applicazioni cloud-native o sistemi legacy sono già sviluppati per interagire con S3 oppure Swift, e offrendo un'interfaccia compatibile lo storage distribuito risulta "plug-and-play" con tali applicazioni senza dover riscrivere codice o cambiare driver; dall'altro migliora la flessibilità architetturale, dal momento che la disponibilità di entrambi i protocolli consente di supportare scenari diversi — ad esempio applicazioni che richiedono l'ecosistema AWS/S3 o ambienti open-source/privati che preferiscono Swift — rendendo lo storage distribuito più versatile.

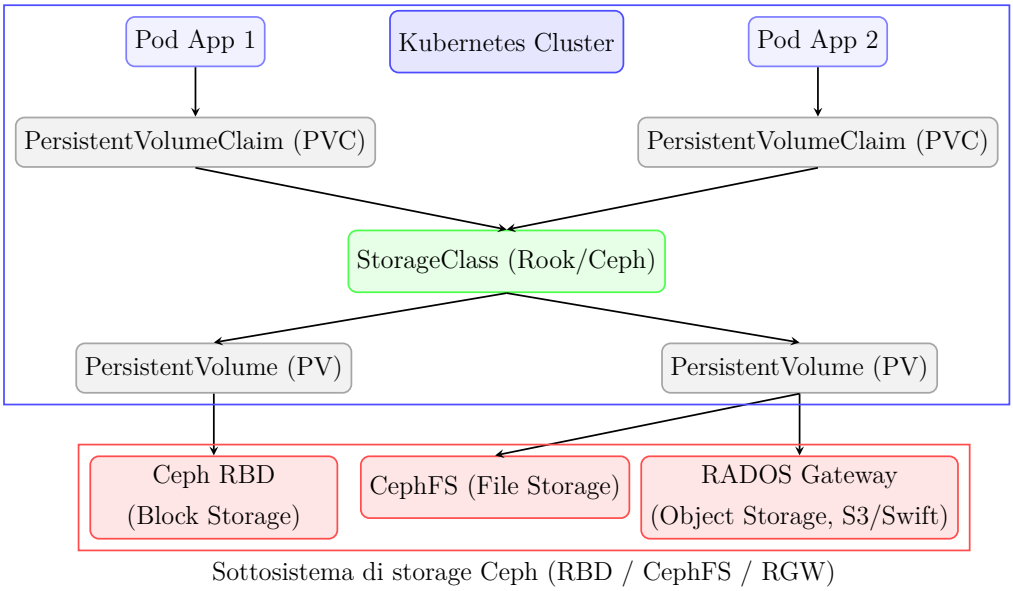

Figura 5.1: Schema dell'integrazione tra Kubernetes e Ceph tramite Rook. I pod delle applicazioni richiedono spazio di storage tramite PersistentVolumeClaim (PVC), che viene gestito automaticamente da Rook attraverso una StorageClass. La StorageClass definisce le caratteristiche dello storage, mentre i PersistentVolume (PV) rappresentano l'allocazione fisica dei dati. Rook traduce queste richieste in operazioni sul cluster Ceph, che può fornire diversi tipi di storage: RBD per volumi a blocchi, CephFS per filesystem distribuiti condivisi e RADOS Gateway per l'accesso a oggetti compatibili con S3 o Swift. Lo schema evidenzia così il flusso dichiarativo dello storage, dalla richiesta dei pod fino ai backend Ceph.

5.2.1 Amazon S3

Amazon Simple Storage Service (S3) è un servizio cloud di storage ad oggetti gestito da Amazon Web Services (AWS) [11]. Esso consente di memorizzare e recuperare in qualsiasi momento quantità praticamente illimitate di dati mediante una API REST basata su operazioni HTTP (GET, PUT, DELETE). Nel modello, gli oggetti vengono organizzati in "bucket" e identificati da una chiave (key) univoca. S3 fornisce inoltre funzionalità avanzate quali versioning, controllo degli accessi, replicazione geografica e classi di storage differenziate. Viene nominato perché molti sistemi di storage distribuito

(come ad esempio Ceph tramite il suo gateway oggetti) offrono un'interfaccia `compatibile con S3`, permettendo alle applicazioni che già lo utilizzano di essere interoperabili senza modifiche significative.

5.2.2 OpenStack Swift

OpenStack Swift è un progetto open source di storage oggetti facente parte della piattaforma OpenStack. Swift è progettato per l'archiviazione distribuita di grandi quantità di dati non strutturati ("blob"/oggetti), con particolare attenzione a scalabilità, disponibilità e resilienza [12]. La sua API REST (Object Storage API) consente la gestione di `account`, `contenitori` (containers) e `oggetti`. Viene nominato perché, analogamente a S3, rappresenta un altro standard per l'accesso a storage ad oggetti; quando il gateway oggetti di Ceph (es. il demone RGW) espone un'interfaccia Swift-compatibile, le applicazioni che già usano Swift possono interfacciarsi senza modifiche.

5.3 Considerazioni teoriche sull'implementazione

5.3.1 Allineamento con cloud-native e DevOps

Questo approccio si inserisce pienamente nel paradigma del cloud-native e DevOps[3]: la containerizzazione, la gestione dichiarativa delle risorse, l'orchestrazione automatica e il provisioning dinamico dello storage ne sono espressione diretta. In un contesto DevOps, l'operatore Rook funge da elemento che integra la gestione dell'infrastruttura di storage nel ciclo di vita delle applicazioni, permettendo agli sviluppatori e agli operatori di trattare lo storage come codice (Infrastructure as Code) e di includerlo nei processi di Continuous Delivery e Continuous Deployment. Il risultato è una maggiore agilità,

[3]DevOps è un insieme di pratiche, strumenti e una filosofia culturale che automatizza e integra i processi tra team di sviluppo software (Dev) e team IT operativi (Ops).

una migliore integrazione tra il livello applicativo e quello infrastrutturale, e una riduzione della barriera tra sviluppo e operazioni.

5.3.2 Criticità operative e competenze richieste

L'adozione di questo modello comporta anche una serie di criticità e punti di attenzione che devono essere valutati attentamente. In primo luogo, il modello introduce una maggiore complessità operativa: la coesistenza di un cluster Ceph containerizzato all'interno di Kubernetes comporta che non solamente le applicazioni, ma anche i demoni storage siano sottoposti a orchestrazione, aggiornamenti e monitoraggio. Questo significa che il team operativo deve possedere competenze sia cloud-native sia di storage distribuito.

5.3.3 Prestazioni, rete e aggiornamenti

In secondo luogo, le prestazioni possono risultare inferiori rispetto a soluzioni di storage dedicate: come evidenziato, in ambienti a bassa latenza o ad altissimo I/O, la sovrapposizione dell'infrastruttura containerizzata e dell'orchestratore può introdurre overhead e ritardi. Ulteriori criticità includono la rete (il corretto funzionamento della comunicazione tra demoni Ceph distribuiti è essenziale) e i rischi durante l'aggiornamento: l'upgrade del cluster Ceph o del Rook operator richiede attenzione al quorum dei monitor, alla compatibilità delle versioni e alla conservazione dei dati durante le transizioni.

5.3.4 Scalabilità e pianificazione delle risorse

Infine, va valutata la scalabilità e l'utilizzo efficiente delle risorse: i demoni storage richiedono CPU, memoria e I/O adeguati, e la pianificazione errata delle risorse può portare a degradazioni o blocchi del servizio.

Capitolo 6

Analisi comparativa e Discussione

Di seguito vengono messi a confronto i tre approcci analizzati nei capitoli precedenti. L'obiettivo è comprendere in che modo ciascuna di queste soluzioni risponda a esigenze differenti in termini di complessità, automazione, scalabilità e facilità di gestione.

6.1 Criteri di confronto

Per rendere confrontabili i tre approcci descritti nei capitoli precedenti sono stati individuati alcuni criteri comuni, applicabili a soluzioni con diverso livello di automazione e astrazione. In particolare sono stati considerati:

- **Semplicità d'installazione e configurazione**: misura quanto è lineare il processo di predisposizione del cluster, quante operazioni manuali richiede e quanto sia esposto a errori di configurazione.
- **Automazione e ripetibilità**: indica in che misura la procedura può essere rieseguita con gli stessi risultati e quanto del processo è delegato a strumenti o script.

- **Efficienza nell'uso delle risorse**: valuta l'overhead complessivo su CPU, RAM e storage a parità di funzionalità.

- **Visibilità interna**: riguarda quanto la soluzione lascia "vedere" i componenti e i flussi del cluster (MON, MGR, OSD, rete), quindi il suo valore didattico.

- **Scalabilità**: considera con quanta facilità l'ambiente può crescere (più nodi, più dischi, più servizi) mantenendo stabilità e prestazioni adeguate.

6.2 Sintesi Comparativa

In questa sezione si riportano, per ciascun approccio, i principali punti emersi dal confronto rispetto ai criteri definiti.

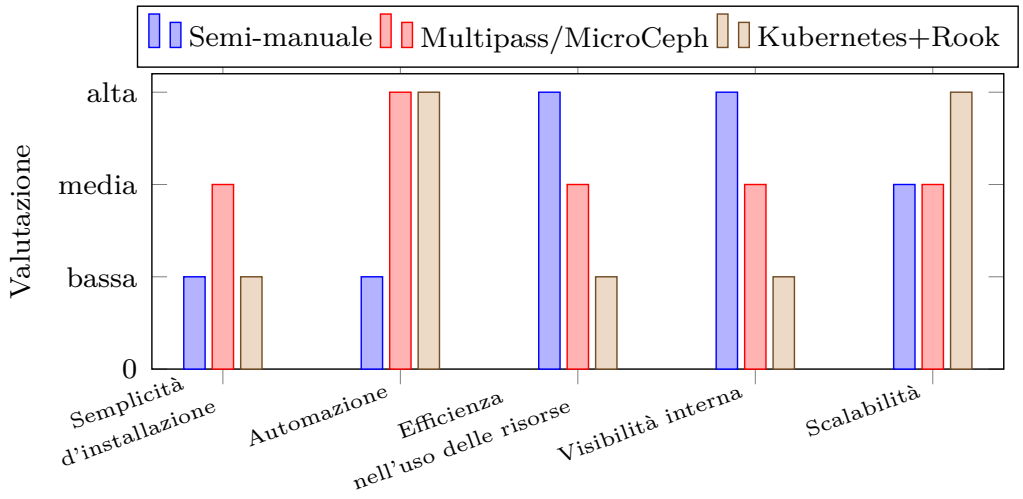

Figura 6.1: I tre approcci sono confrontati sui cinque criteri considerati, usando una scala qualitativa in cui valori più bassi indicano prestazioni migliori.

6.2.1 Installazione semi-manuale con gli strumenti Ceph

- **Semplicità di installazione e configurazione**: bassa; richiede interventi manuali e una buona conoscenza degli strumenti messi a disposizione da Ceph.
- **Automazione**: bassa; la procedura è meno ripetibile e più soggetta a errori umani.
- **Efficienza nell'uso delle risorse**: alta; sono richiesti solo i pacchetti/strumenti Ceph necessari sull'host.
- **Visibilità interna**: alta; tutti i componenti (MON, MGR, OSD) e i comandi sono esposti, quindi è adatto alla didattica.
- **Scalabilità**: media; il cluster può essere esteso, ma richiede ulteriori interventi manuali.

6.2.2 Installazione automatizzata con Multipass e MicroCeph

- **Semplicità di installazione e configurazione**: media; molte operazioni sono automatizzate dallo
  script/orchestratore sviluppato.
- **Automazione**: alta; il deploy può essere rilanciato con gli stessi parametri e risulta più ripetibile.
- **Efficienza nell'uso delle risorse**: media; l'uso di più VM sulla stessa macchina fisica aumenta il consumo di risorse rispetto alla soluzione più manuale.
- **Visibilità interna**: media; alcune scelte vengono nascoste dal livello di automazione (ad es. MicroCeph).

- **Scalabilità**: media; l'estensione è possibile finché l'host può sostenere ulteriori VM.

6.2.3 Integrazione con Kubernetes e Rook

- **Semplicità di installazione e configurazione**: bassa; oltre alla parte Ceph è necessario disporre di un cluster Kubernetes funzionante.
- **Automazione**: alta; una volta definiti i manifest, la creazione delle risorse è dichiarativa.
- **Efficienza nell'uso delle risorse**: bassa; la presenza di Kubernetes e Rook introduce un overhead non trascurabile.
- **Visibilità interna**: tendenzialmente più bassa; l'astrazione di Kubernetes nasconde alcune scelte, resta però possibile ispezionare i componenti tramite gli strumenti messi a disposizione dall'operatore.
- **Scalabilità**: alta dal punto di vista teorico; la soluzione è pensata per crescere in ambienti clusterizzati.

6.3 Discussione e interpretazione

Dal punto di vista della **complessità di installazione e configurazione** e della **visibilità sul funzionamento interno di Ceph**, l'approccio maggiormente "manuale", basato sull'utilizzo diretto degli strumenti messi a disposizione da Ceph (ad esempio `cephadm`), si è rivelato quello più vicino al funzionamento reale del sistema. Esso rende esplicite le dipendenze tra i demoni, le fasi di bootstrap e le condizioni del cluster, favorendo quindi la comprensione dell'architettura interna. Il rovescio della medaglia, in termini di **livello di automazione e ripetibilità**, è una maggiore sensibilità a errori di configurazione e, in generale, una richiesta più alta di competenze sistemistiche, oltre a tempi di predisposizione meno prevedibili.

L'approccio intermedio, che fa uso di un orchestratore sviluppato ad hoc per automatizzare la creazione delle macchine virtuali (ad esempio tramite Multipass) e l'installazione di una soluzione semplificata come MicroCeph, introduce un livello di comfort significativamente maggiore proprio in termini di **automazione e ripetibilità**: la procedura di deploy diventa infatti ripetibile e parametrizzabile, caratteristica utile in un contesto didattico o di test in cui sia necessario ricreare spesso l'ambiente da zero o proporre lo stesso laboratorio a più studenti. Questo vantaggio viene però pagato, rispetto al criterio della **visibilità sul funzionamento interno di Ceph**, con una minore osservabilità delle scelte interne del sistema di storage, poiché alcune configurazioni vengono effettuate automaticamente dallo script di orchestrazione. Inoltre, sotto il profilo dei **requisiti e consumo di risorse**, la dipendenza dalle risorse dell'host fisico diventa più marcata: all'aumentare del numero di VM le prestazioni complessive degradano e il margine di sperimentazione si riduce.

L'ultimo approccio, che integra Ceph in un ambiente orchestrato (ad esempio tramite Kubernetes e Rook), risulta particolarmente interessante se valutato secondo i criteri di **scalabilità e possibilità di estensione**: rappresenta infatti la soluzione più moderna e più vicina ai contesti cloud-native, in cui lo storage distribuito diventa una risorsa del cluster e viene richiesto tramite oggetti dichiarativi. Dal punto di vista concettuale è il metodo più elegante e scalabile, perché consente di uniformare il ciclo di vita delle applicazioni e dello storage. Tuttavia, in relazione alla **complessità di installazione e configurazione** e ai **requisiti**, richiede prerequisiti più elevati (un cluster Kubernetes funzionante e la conoscenza dei relativi oggetti) e introduce un overhead non trascurabile in ambienti con risorse limitate.

Nel complesso, i risultati suggeriscono che i tre approcci non sono da considerarsi alternativi in senso assoluto, ma piuttosto complementari rispetto all'obiettivo e ai criteri individuati:

- l'approccio manuale è più indicato quando la priorità è comprendere l'architettura di Ceph, quindi quando pesano soprattutto i criteri di

visibilità e di **controllo sulla configurazione**;

- l'approccio automatizzato è più indicato quando la priorità è disporre rapidamente di un ambiente di prova riproducibile, quindi quando il criterio principale è il **livello di automazione e ripetibilità**;

- l'approccio integrato in Kubernetes è più indicato quando la priorità è vedere come uno storage distribuito si inserisce in una piattaforma orchestrata, quindi quando si vogliono massimizzare **scalabilità** e **integrazione con ambienti reali**.

6.3.1 Implicazioni per l'uso didattico

I risultati mostrano che gli approcci con maggior esposizione ai componenti di Ceph sono più efficaci nelle fasi iniziali di apprendimento, mentre le soluzioni automatizzate permettono di ridurre i tempi di preparazione del laboratorio e di replicare l'ambiente su più postazioni.

6.4 Limiti sperimentali

Come anticipato nella Sezione 1.3, le prove descritte sono state condotte in un ambiente controllato e non di produzione, e questo impone alcune cautele nell'interpretazione dei risultati.

In primo luogo, l'infrastruttura era basata su un unico host fisico che ospitava sia le macchine virtuali sia gli strumenti di orchestrazione: ciò ha limitato la scalabilità e non ha consentito di verificare configurazioni realmente distribuite su più server fisici, che rappresentano lo scenario tipico di impiego di Ceph.

In secondo luogo, in alcune prove lo storage è stato simulato tramite file di loopback o dischi non dedicati: scelta adeguata allo scopo dimostrativo, ma non sufficiente per trarre conclusioni sulle prestazioni reali del sistema o sul comportamento sotto carichi sostenuti.

Un ulteriore limite riguarda l'assenza di una campagna sistematica di benchmark: l'obiettivo principale era confrontare il livello di automazione e la semplicità dei tre approcci, più che misurare throughput, latenza o tempi di recovery. Per un'analisi prestazionale sarebbero necessari strumenti dedicati (ad es. `rados bench`) e dischi dedicati.

Va inoltre segnalata una certa dipendenza dalla versione degli strumenti utilizzati (Multipass, MicroCeph, componenti Kubernetes): alcune criticità osservate potrebbero non presentarsi in versioni successive o, viceversa, emergere in ambienti software differenti.

Infine, il terzo approccio, basato sull'integrazione con Kubernetes e Rook, è stato considerato prevalentemente dal punto di vista architetturale e concettuale, senza una realizzazione completa dell'ambiente in esecuzione. Le valutazioni riportate per questa soluzione riguardano quindi soprattutto la fattibilità, i prerequisiti e il grado di integrazione ottenibile.

6.4.1 Minacce alla validità dei risultati

I risultati possono essere influenzati da fattori esterni quali la versione del software, la specifica configurazione hardware dell'host e l'assenza di un carico applicativo reale. Queste minacce sono parzialmente mitigate dal fatto che tutti gli approcci sono stati provati nello stesso ambiente.

Capitolo 7

Conclusioni

Il lavoro aveva l'obiettivo di individuare e confrontare modalità diverse per rendere riproducibile, in un contesto didattico e con risorse limitate, il deploy di un'infrastruttura di storage distribuito basata su Ceph. Sono stati quindi analizzati e messi alla prova tre approcci caratterizzati da livelli crescenti di automazione e astrazione: l'installazione semi-manuale tramite gli strumenti nativi di Ceph (`cephadm`), la soluzione automatizzata su macchine virtuali leggere con Multipass e MicroCeph supportata da un orchestratore sviluppato ad hoc, e infine l'integrazione in un ambiente container-native tramite Kubernetes e l'operatore Rook.

Dalle sperimentazioni svolte emerge che i tre approcci non vanno letti come alternativi, ma come soluzioni complementari, ognuna più adatta a un certo obiettivo. L'approccio più manuale espone meglio il funzionamento interno di Ceph (demoni, bootstrap, *health*, aggiunta degli OSD) ed è quindi il più utile quando lo scopo è comprendere il sistema, accettando però tempi di preparazione maggiori e una curva di apprendimento più ripida. L'approccio automatizzato con Multipass e MicroCeph, invece, dimostra che è possibile ottenere rapidamente un cluster funzionante e replicabile su una sola macchina fisica, aspetto particolarmente interessante in ambito laboratoriale o quando la stessa esercitazione deve essere proposta a più studenti. La soluzione basata su Kubernetes e Rook risulta infine la più allineata agli scenari

cloud-native e la più promettente in termini di scalabilità e integrazione, pur richiedendo prerequisiti più elevati ed essendo stata trattata nel lavoro soprattutto dal punto di vista architetturale.

Un'ulteriore conclusione riguarda l'aspetto didattico: quando si vuole insegnare Ceph è utile partire da un livello in cui i componenti sono visibili e configurati manualmente, e solo successivamente introdurre livelli di automazione che velocizzano il deploy ma nascondono alcune scelte interne. In altri termini, esiste un trade-off piuttosto netto tra "quanto si vede di Ceph" e "quanto è veloce e ripetibile il laboratorio", e il confronto tra i tre approcci lo rende esplicito.

I risultati vanno comunque letti alla luce dei limiti dichiarati: tutte le prove sono state eseguite su un unico host fisico, con storage in parte simulato e senza una campagna sistematica di benchmark. Questi vincoli riducono la generalizzabilità dei dati, ma non invalidano il confronto tra i livelli di automazione e di astrazione, che costituiva lo scopo principale del lavoro.

Bibliografia

[1] *Ceph Documentation*, sezione su pool e placement group: «Pools» e «Placement groups». Disponibile rispettivamente all'indirizzo: `https://docs.ceph.com/en/latest/rados/operations/pools/` e `https://docs.ceph.com/en/latest/rados/operations/placement-groups/`

[2] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "CRUSH: Controlled, scalable, decentralized placement of replicated data," in *Proc. ACM/IEEE Supercomputing (SC)*, 2007.

[3] S. A. Weil, S. A. Brandt, E. L. Miller, and D. D. E. Long, *RADOS: A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters*. University of California, Santa Cruz, 2007.

[4] Ceph Authors, "Cephadm documentation" in *Ceph Documentation*. Disponibile su: `https://docs.ceph.com/en/latest/cephadm/`

[5] Canonical, *MicroCeph Documentation*. Disponibile su: `https://canonical-microceph.readthedocs-hosted.com/stable/`

[6] Canonical, "Cloud storage at the edge with MicroCeph," *Canonical Blog*, 2023. Disponibile su: `https://canonical.com/blog/`

[7] The Kubernetes Authors, *Kubernetes Documentation*. Disponibile su: `https://kubernetes.io/docs/home/`

[8] Rook Maintainers, *Rook*. Disponibile su: `https://rook.io/docs/`

[9] Canonical, *Multipass documentation*. Disponibile su: https://documentation.ubuntu.com/multipass/latest/

[10] cloud-init Project, *cloud-init Documentation*. Disponibile su: https://cloudinit.readthedocs.io/en/latest/

[11] Amazon Web Services, *Amazon Simple Storage Service (S3) — Developer Guide*. Disponibile su: https://docs.aws.amazon.com/AmazonS3/latest/dev/

[12] OpenStack Project, *OpenStack Object Storage (Swift) Documentation*. Disponibile su: https://docs.openstack.org/swift/latest/

[13] Packt Publishing, *Ceph – the architectural overview*, in "Ceph Cookbook, Second Edition". Disponibile su: https://subscription.packtpub.com/book/cloud-and-networking/9781788391061/1/ch01lvl1sec05/ceph-the-architectural-overview

[14] ADMIN Magazine, *The RADOS object store and Ceph filesystem: Part 2*. Disponibile su: https://www.admin-magazine.com/HPC/Articles/RADOS-and-Ceph-Part-2

Ringraziamenti

Desidero ringraziare il Prof. Vittorio Ghini, relatore di questa tesi, per la costante disponibilità, la guida attenta e i preziosi suggerimenti che hanno contribuito in modo significativo allo sviluppo del lavoro, permettendo di migliorarne e approfondirne i contenuti.