ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

Titolo dell'elaborato

Comparazione performance di un'applicazione enterprise basata su service mesh

Elaborato in
Reti di telecomunicazioni

Relatore
Esimio Professore
Callegati Franco

Presentata da
Speranza Alex
Matricola n° 0000830072

Correlatori
Borsatti Davide
Maiorano Carlo
Sabbioni Andrea

Anno Accademico 2019/2020

Sommario

Ultimamente si stanno sviluppando tecnologie per rendere più efficiente la virtualizzazione a livello di sistema operativo, tra cui si cita la suite Docker, che permette di gestire processi come se fossero macchine virtuali. Inoltre i meccanismi di clustering, come Kubernetes, permettono di collegare macchine multiple, farle comunicare tra loro e renderle assimilabili ad un server monolitico per l'utente esterno. Il connubio tra virtualizzazione a livello di sistema operativo e clustering permette di costruire server potenti quanto quelli monolitici ma più economici e possono adattarsi meglio alle richieste esterne.

Data l'enorme mole di dati e di potenza di calcolo necessaria per gestire le comunicazioni e le interazioni tra utenti e servizi web, molte imprese non possono permettersi investimenti su un server proprietario e la sua manutenzione, perciò affittano le risorse necessarie che costituiscono il cosiddetto "cloud", cioè l'insieme di server che le aziende mettono a disposizione dei propri clienti.

Il trasferimento dei servizi da macchina fisica a cloud ha modificato la visione che si ha dei servizi stessi, infatti non sono più visti come software monolitici ma come microservizi che interagiscono tra di loro. L'infrastruttura di comunicazione che permette ai microservizi di comunicare è chiamata service mesh e la sua suddivisione richiama la tecnologia SDN.

È stato studiato il comportamento del software di service mesh Istio installato in un cluster Kubernetes. Sono state raccolte metriche su memoria occupata, CPU utilizzata, pacchetti trasmessi ed eventuali errori e infine latenza per confrontarle a quelle ottenute da un cluster su cui non è stato installato Istio.

Lo studio dimostra che, in un cluster rivolto all'uso in produzione, la service mesh offerta da Istio fornisce molti strumenti per il controllo della rete a scapito di una richiesta leggermente più alta di risorse hardware.

Indice

1 Virtualizzazione

1.1 Full virtualization

1.1.1 Macchine virtuali

Una macchina virtuale (MV) è un'emulazione di un sistema informatico ed è basata su un'architettura di un dispositivo e fornisce le funzionalità di un computer fisico[138].

In termini informatici, un'architettura è l'insieme di regole e metodi che descrivono le funzionalità, l'organizzazione e, talvolta, l'implementazione di un sistema informatico[96]. Le più note sono le architetture `x86`, principalmente usata per i personal computer e i server[140], e `ARM`, presente negli smartphone e nei sistemi integrati[88], entrambe disponibili sia a 32 che a 64 bit[95].

Le macchine virtuali forniscono un sostituto di un dispositivo reale, infatti forniscono le funzionalità necessarie per eseguire un intero sistema operativo. Le MV sono gestite da un hypervisor che permette di condividere e amministrare le risorse hardware, consentendo così di avere più ambienti isolati conviventi sulla stessa macchina fisica. L'isolamento è dovuto alle possibili restrizioni applicate riguardanti l'accesso a determinate risorse.

Alcuni software di emulazione, come QEMU, possono emulare architetture di processore diverse da quella della macchina ospitante[138], ma con una perdita in termini di prestazioni in quanto deve essere anche eseguita una traduzione dinamica delle istruzioni macchina dall'architettura emulata all'architettura del sistema ospitante, mentre se l'architettura ospitante comprende le istruzioni dell'architettura emulata, il processore può eseguirle direttamente[91].

Per avere una virtualizzazione completa (full virtualization), il sistema ospitante deve possedere tutte le funzionalità salienti del sistema che si vuole riprodurre: set di istruzioni, operazioni di input/output, segnali di interrupt, accesso alla memoria e qualsiasi altro elemento che è presente nella macchina reale e che deve essere funzionante nella macchina virtuale[105].

1.1.2 Hypervisor

Un hypervisor è un componente software, firmware o hardware che permette di gestire le macchine virtuali. Il dispositivo su cui l'hypervisor esegue una o più MV viene chiamato "host" (padrone, ospitante), mentre ciascuna macchina virtuale viene chiamata "guest" (ospite). Quindi un hypervisor fornisce una piattaforma operativa virtuale al sistema operativo ospitato e ne gestisce l'esecuzione. In questo modo, istanze multiple di vari sistemi operativi possono condividere le stesse risorse hardware virtualizzate, infatti delle istanze Linux, Windows o macOS possono tutte essere eseguite su una sola macchina fisica con architettura x86[107].

Il termine "hypervisor", ipervisore, è una variante di "supervisor", supervisore, che indica il kernel di un sistema operativo: l'hypervisor è il supervisore dei supervisori, in quanto gestisce i kernel dei sistemi operativi installati nelle varie macchine virtuali[107].

Sebbene la classificazione non sia a compartimenti stagni, gli hypervisor sono divisi in due categorie[108]:

- Type-1, nativo o bare-metal: questi hypervisor sono eseguiti direttamente sull'hardware ospitante per controllarlo e gestire le MV, per questo talvolta sono chiamati "bare-metal hypervisor". Sostituiscono il sistema operativo ospitante, perciò possiedono tutte le funzioni vitali di un SO, compreso il kernel[139]. Alcuni esempi di hypervisor di questo tipo sono Microsoft Hyper-V, XCP-ng, VMware ESXi e Xen.

- Type-2 o ospitati: gli hypervisor di questo tipo sono eseguiti come un qualsiasi altro programma all'interno di un sistema operativo, sotto forma di processo. Questi hypervisor astraggono il sistema operativo ospite da quello ospitante. Qualche esempio di questa categoria di hypervisor sono QEMU, VirtualBox e VMware Workstation.

Gli hypervisor moderni usano la funzionalità di virtualizzazione assistita da hardware fornita dal processore della macchina ospitante[138]. Quando disponibile, gli hypervisor possono attingere a funzionalità del processore del sistema ospitante che permettono una virtualizzazione efficiente, chiamata "virtualizzazione assistita da hardware", "virtualizzazione accelerata" o "virtualizzazione nativa"[106].

Gli hypervisor spesso adottano il meccanismo chiamato "overcommitment": consentono di definire più macchine virtuali la cui somma di risorse (tendenzialmente RAM e CPU) è inferiore alle risorse disponibili nel sistema ospitante, ad esempio anche avendo 4 gigabyte di RAM, si possono dichiarare quattro MV, ciascuna con 2 gigabyte di RAM. Questo meccanismo è funzionale dato che normalmente alcune MV hanno un basso carico di lavoro, mentre altre ne hanno uno più alto e i livelli di attività di ciascuna macchina ha fluttuazioni nel tempo. Perciò. gli hypervisor trasferiscono la memoria libera delle macchine inattive alle macchine a cui serve[85].

Un'altra funzionalità messa a disposizione dagli hypervisor moderni è la possibilità di fare "snapshot" di una macchina virtuale. Uno snapshot è il salvataggio dello stato e dei dati di una MV in un determinato istante temporale. Uno snapshot può essere usato per replicare velocemente una macchina virtuale configurata; ripristinare una macchina andata in crash ad uno stato funzionante; spostare una macchina virtuale da un sistema ad un altro[81].

1.1.3 Vantaggi della full virtualization

La virtualizzazione su macchine virtuali offre molti vantaggi[86][82]:

1. Gestione centralizzata: tutte le macchine virtuali possono essere gestite da una sola postazione, cioè il terminale che le ospita, attraverso il software di virtualizzazione.

2. Ottimizzazione dell'uso delle risorse: il software che gestisce le risorse può ottimizzarne l'uso nel caso in cui una parte delle MV abbia un carico di

lavoro ridotto. Questo sistema permette il meccanismo di overcommitment delle risorse, in quanto non sempre tutte le macchine virtuali sfruttano a pieno la potenza di calcolo che gli è stata fornita.

3. Disaster recovery veloce: grazie al meccanismo degli snapshot, è possibile ripristinare una macchina virtuale ad uno stato funzionante in poco tempo. Inoltre gli snapshot permettono di migrare o spostare MV da un sistema ospitante all'altro, anche nell'eventualità che il primo sia la causa del malfunzionamento. Infine anche gli aggiornamenti al sistema sono più sicuri in quanto in caso di errori si può ritornare ad uno stato precedente.

4. Supporto a sistemi operativi obsoleti: grazie alle macchine virtuali è possibile emulare un set hardware compatibile con i vecchi sistemi operativi. In questo modo, ad esempio, è possibile continuare ad offrire supporto per un'applicazione datata. Rimangono comunque presenti i problemi legati alla sicurezza dell'uso di un sistema obsoleto.

5. Esecuzione di sistemi operativi multipli: le macchine virtuali rendono possibile eseguire diversi sistemi operativi su uno stesso dispositivo fisico senza dover riavviare il computer ogni volta che si presenta la necessità di cambiare sistema operativo.

1.1.4 Gli svantaggi della full virtualization

Sebbene la virtualizzazione su macchine virtuali porti molti vantaggi, ci sono anche aspetti negativi[86][82]:

1. Costo iniziale: Il costo iniziale per il dispositivo ospitante sarà più alto rispetto ad un server unico, dato che questo computer dovrà sostenere il carico di lavoro di più macchine virtuali.

2. Complessità: la difficoltà nel configurare il sistema totale è aumentata dalla gestione delle regole di rete che, nel caso sia necessario un certo livello di sicurezza, può essere un compito oneroso e articolato. Inoltre, per evitare di avere un solo punto di fallimento, è possibile aumentare il numero di dispositivi su cui vengono dispiegate le MV, ma ciò aggiunge difficoltà nella gestione delle macchine e nell'eventuale configurazione di rete.

3. Prestazioni ridotte: le macchine virtuali sono meno efficienti delle macchine fisiche in quanto l'accesso alle risorse hardware non è diretto, ma viene effettuata una richiesta tramite software che comporta rallentamenti.

4. Overcommitment eccessivo: nel caso in cui sia stato effettuato l'overcommitment delle risorse per le macchine virtuali e il carico di lavoro di esse richiede tutte le risorse fornite, si riscontra una diminuzione considerevole delle performance.

1.2 Virtualizzazione a livello di sistema operativo

I container fanno parte un meccanismo di virtualizzazione a livello del sistema operativo che permette di creare istanze di spazi utenti isolati, in modo che i programmi che vengono eseguiti non possano distinguere se sono all'interno di un container o di tutto il sistema operativo.

Uno strumento messo a disposizione dal kernel Linux per implementare i container sono i "namespace" (NS), i quali permettono di astrarre una risorsa globale in modo da far apparire ad un processo all'interno del namespace di possederne una propria istanza privata ed isolata. Le modifiche fatte alla risorsa sono visibili a tutti i processi del namespace, mentre non sono visibili ai processi esterni. Le risorse possono esistere in più namespace; alcuni tipi di esse sono: gli ID dei processi, l'hostname, gli ID degli utenti e i nomi dei file. I tipi di namespace sono:

- Mount: controllano i punti di montaggio dei dispositivi. I punti già esistenti sono copiati nel nuovo namespace, mentre quelli creati all'interno di esso non sono propagati agli altri namespace.
- Process ID: fornisce ai processi un insieme di Process ID indipendente. Questo tipo di namespace può essere annidato, perciò ciascun processo avrà un PID per ogni namespace; questo permette al primo namespace di vedere tutti i processi, sebbene con PID diversi rispetto agli altri NS.
- Network: un namespace di rete virtualizza lo stack di rete; alla creazione contiene solo l'interfaccia di loopback. Ogni interfaccia di rete (fisica o virtuale) è presente in un solo NS e può essere spostata tra essi. Ogni namespace ha il proprio set di IP privati, la propria tabella di routing e le altre risorse di rete.
- Interprocess Communication (ipc): Questo namespace isola i processi, impedendo le comunicazioni che utilizzano la memoria condivisa tra processi in namespace ipc diversi.
- UTS: Il namespace UTS (UNIX Time-Sharing) può far apparire un sistema come se avesse più hostname e nomi di dominio.
- User ID: questo namespace fornisce l'isolamento dei privilegi e la segregazione dell'identificazione dell'utente tra insiemi di processi. Questo permette di avere un utente amministratore all'interno del namespace che è un semplice utente all'interno del sistema.
- Control group: nasconde l'identità del gruppo di controllo (cgroup) del quale un processo fa parte.

I sistemi Linux inizialmente hanno un namespace per ogni tipo, usati da tutti i processi, i quali possono crearne di nuovi e unirsi a diversi namespace[116].

Nei sistemi Unix-like è possibile considerare questo meccanismo come un coordinamento tra gli strumenti "chroot" e "cgroups"[124], di cui il primo permette di cambiare la cartella radice visibile al processo e ai suoi figli[63], mentre il secondo consente di gestire e limitare le risorse da cui un processo può attingere[76].

Pertanto, un container comprende il processo che deve essere eseguito e la definizione delle risorse del sistema ospitante a cui può accedere. Il processo può dipendere da librerie esterne: se sono presenti nel sistema ospitante vengono utilizzate quelle, altrimenti vengono scaricate e salvate all'interno dei file del container.

Al contrario, su una macchina virtuale è necessario installare anche un sistema operativo oltre al programma da eseguire e le librerie da cui dipende, anche se sul sistema ospitante sono già presenti.

Perciò, a parità di costo computazionale di un'applicazione, un container risulterà meno oneroso al sistema operativo ospitante, infatti una macchina virtuale utilizza,in aggiunta, le risorse per eseguire il proprio sistema operativo oltre a quelle usate dal programma.

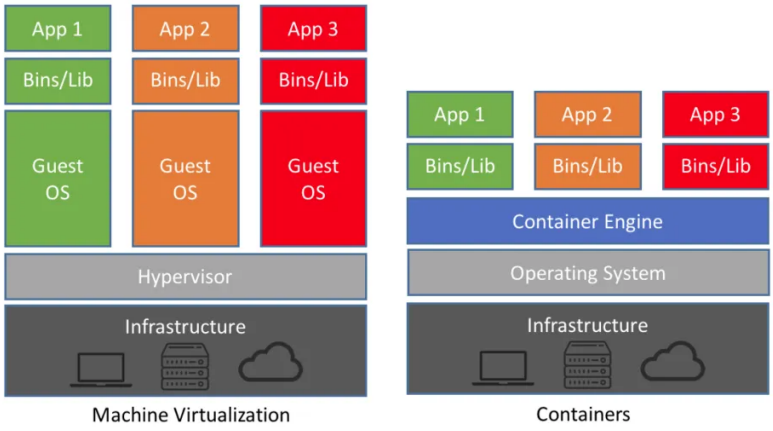

Figura 1: Differenza dello stack applicativo tra container e macchine virtuali
Fonte: https://softwareengineeringdaily.com/2018/10/04/cloud-structures-kubernetes-container-instances-serverless/

L'uso dei container, rispetto ad una virtualizzazione completa con una macchina virtuale, offre alcuni vantaggi[30]:

- Efficienza: un container richiama le funzioni di sistema del sistema operativo su cui è in esecuzione il proprio gestore, mentre una MV utilizza le funzioni del sistema operativo installato, che potrebbe differire da quello su cui è in esecuzione il supervisore della virtualizzazione.
- Dimensioni ridotte: la maggior parte delle implementazioni di gestori di container fornisce meccanismi di copy-on-write, cioè solo nel momento

in cui viene fatta una modifica ad un file viene salvata una copia dello stesso[99]. Questo meccanismo consente di risparmiare spazio su disco anche avendo tanti container, a patto che non modifichino file, in tal caso la copia modificata sarà salvata nel file system del container incriminato.

Come nella virtualizzazione completa, i container mantengono l'isolamento del processo in esecuzione e forniscono ambienti deterministici agli sviluppatori, aumentando così la produttività e diminuendo la comparsa di bug dovuti all'uso di ambienti diversi tra sviluppo e produzione[30].

Nonostante ciò, i container sono meno flessibili delle macchine virtuali in quanto supportano solo le funzioni del sistema ospitante, quindi, ad esempio, non è possibile eseguire un programma che richiede librerie di Windows su un sistema Linux.

I due tipi di virtualizzazione non sono mutualmente esclusivi, pertanto è possibile eseguire dei container all'interno di macchine virtuali.

2 Gestione delle reti internet

I dispositivi che sono usati per instradare i pacchetti al livello di rete in internet sono chiamati router. Al loro interno sono presenti molti componenti coordinati tra loro per eseguire le funzioni richieste. I componenti sono divisi in due tipi: elementi di controllo ed elementi di instradamento. I primi fanno parte del piano di controllo (control plane), che si occupa di gestire i percorsi della rete che sono disponibili e quindi le azioni da svolgere con i pacchetti in entrata; i secondi fanno parte del piano dei dati (data plane o forwarding plane) e processano le azioni definite nell'altro piano[98][100].

La presenza di standard che separano questi due tipi di componenti a livello logico permette, eventualmente, di separarli anche a livello fisico. Ciò permette ai produttori di componenti di specializzarsi su una sola tipologia di essi e non su tutte; inoltre è possibile utilizzare elementi di produttori diversi per produrre un router, con caratteristiche diverse in base alle esigenze[141].

2.1 Indirizzi MAC

L'indirizzo MAC (indirizzo Media Access Control, chiamato anche indirizzo LAN) è un codice univoco di 48 bit (6 byte) assegnato ad ogni scheda di rete, ethernet o wireless, dal proprio produttore. Il codice è normalmente scritto in esadecimale, separando ogni byte con un "-", anche se non è raro trovare la forma che usa ":" come separatore dei byte esadecimali. Gli indirizzi MAC sono utilizzati al livello 2 dello standard ISO/OSI, il livello di collegamento. Per tradurre un indirizzo MAC ad un indirizzo IP viene utilizzato il protocollo ARP[109].

2.1.1 Protocollo ARP

Il protocollo ARP serve per associare un indirizzo MAC ad un indirizzo IP che si trova nella tabella di routing di un host.

Se il pacchetto da inviare è destinato ad un host della stessa sottorete, ARP permette di conoscere il suo indirizzo MAC; se, al contrario, il destinatario appartiene ad un'altra sottorete, ARP viene utilizzato per scoprire l'indirizzo del gateway della rete.

L'host che vuole conoscere l'indirizzo MAC di un altro dispositivo di cui conosce l'indirizzo IP, invia un messaggio "ARP Request" in broadcast a tutti gli host della sottorete in cui allega il proprio indirizzo MAC e l'IP desiderato. Tutti i computer della sottorete ricevono la richiesta e, se il proprio indirizzo IP corrisponde a quello richiesto, viene mandata una risposta, chiamata "ARP Reply", contenente il proprio MAC direttamente all'host richiedente, che lo memorizza in un'apposita cache[87].

2.2 Indirizzi IP

Per comunicare in rete viene usato il protocollo IP: la versione 4 prevede l'uso di indirizzi di 32 bit, quindi 4 byte, per indicare un'interfaccia di un dispositivo connesso in rete, perciò gli indirizzi a disposizione sono 2^{32}. Quindi un dispositivo con una scheda di rete che contiene 4 interfacce avrà un solo indirizzo MAC, ma potrà avere fino a 4 indirizzi IP, in base al numero di interfacce utilizzate al numero di reti a cui si collega.

Gli indirizzi IP possono essere scritti in due notazioni: la prima è la notazione binaria, nella quale viene scritto il valore dei bit, separati da punti ogni 8 bit; la seconda viene definita "dotted notation" (notazione puntata): si scrive l'indirizzo traducendo ciascuno dei 4 byte in un numero decimale separandoli con dei punti.

Gli indirizzi IP sono divisi in bit facenti parte dell'identificatore della rete e in bit facenti parte dell'identificatore dell'host: i primi indicano la rete di appartenenza, mentre i secondi indicano, all'interno della rete specificata, un determinato dispositivo. Gli indirizzi IP, all'interno di una rete, sono univoci.

2.2.1 Classful domain

Dal 1986 al 1993 è stata usata l'architettura di indirizzamento in rete chiamata "Classful domain", che prevede la suddivisione dello spazio degli indirizzi IP in cinque classi predefinite e di dimensione fissata (vedi tabella 1). Le classi D ed E furono destinate a funzionalità specifiche e quindi non furono rese disponibili per i dispositivi connessi, comportando una leggera contrazione nel numero di indirizzi assegnabili.

In questa architettura non è necessario specificare la sottorete di appartenenza di un indirizzo in quanto è definita implicitamente all'interno dello stesso.

Questa architettura contiene nella sua progettazione alcuni problemi: in alcuni casi, una rete di classe C potrebbe non bastare (si pensi ad un'azienda

Classe	Bit iniziali	N° bit per ID di rete	N° bit per ID di host	Indirizzo iniziale	Indirizzo finale
A	`0`	8	24	`0.0.0.0`	`127.255.255.255`
B	`10`	16	16	`128.0.0.0`	`191.255.255.255`
C	`110`	24	8	`192.0.0.0`	`223.255.255.255`
D	`1110`	Non definito	Non definito	`224.0.0.0`	`239.255.255.255`
E	`1111`	Non definito	Non definito	`240.0.0.0`	`255.255.255.255`

Tabella 1: Caratteristiche delle classi di indirizzi IP

Fonte:
https://en.wikipedia.org/wiki/Classful_network#Classful_addressing_definition

con 300 dispositivi connessi, solo 256 sarebbero coperti dagli indirizzi forniti) e una rete di classe B risulterebbe troppo grande (solo 300 indirizzi su 65536 sarebbero occupati); inoltre, con l'aumentare del numero dei dispositivi sempre connessi ad internet, i poco più di quattro miliardi di indirizzi non sarebbero bastati per coprirli tutti[113].

Per far fronte a queste problematiche impellenti, è stato necessario ideare nuove tecnologie per poter ampliare, fisicamente o virtualmente, lo spazio di indirizzi. Per la prima categoria, è stata proposta una nuova versione del protocollo IP con indirizzi a 128 bit (quindi sono disponibili 2^{128} indirizzi) chiamata IPv6; mentre nel secondo caso si è ideato il meccanismo di NAT (Network Address Translation) e il CIDR (Classless Inter-Domain Routing)[113] (entrambe le categorie di tecnologie non sono mutualmente esclusive tra loro ed al loro interno, infatti attualmente viene usata l'architettura CIDR e il NAT e la maggior parte dei dispositivi supporta l'uso di indirizzi IPv6).

2.2.2 Network Address Translation

Il meccanismo di NAT modifica le informazioni sulla rete nei pacchetti IP quando transitano attraverso un router. Inizialmente fu stato ideato per evitare di riassegnare gli indirizzi degli host di una rete spostata o nel caso in cui si fosse cambiato ISP (Internet Service Provider, il fornitore della linea internet); però quando si presentò il vicino esaurimento degli indirizzi IPv4 pubblici, venne usato per nascondere dietro ad un solo router (tendenzialmente necessitante di un solo indirizzo IP pubblico) un'intera rete locale e privata (operazione chiamata "IP masquerading")[120].

Le implementazioni di NAT sono classificate in quattro categorie (vedi figura 2):

1. Full-cone NAT: è un'associazione uno a uno tra una coppia indirizzo-porta interna ed una esterna. Qualsiasi host esterno che voglia comunicare con

il client nascosto dal NAT può inviare i pacchetti alla coppia esterna, che saranno inoltrati verso la rete interna.

2. Address-restricted-cone NAT: richiede un'associazione come in un NAT Full-cone. I pacchetti inviati da un host esterno verso un client interno sono accettati solo se il client ha instaurato una comunicazione con l'indirizzo IP dell'host (in questo caso la porta non è significativa).

3. Port-restricted-cone NAT: funziona allo stesso modo dell'address-restricted-cone NAT, con la differenza che in questo caso la porta dell'host esterna deve essere la stessa con cui il client interno ha precedentemente iniziato la comunicazione.

4. Symmetric NAT: utilizza una coppia indirizzo-porta per ciascuna destinazione raggiunta dal client interno.

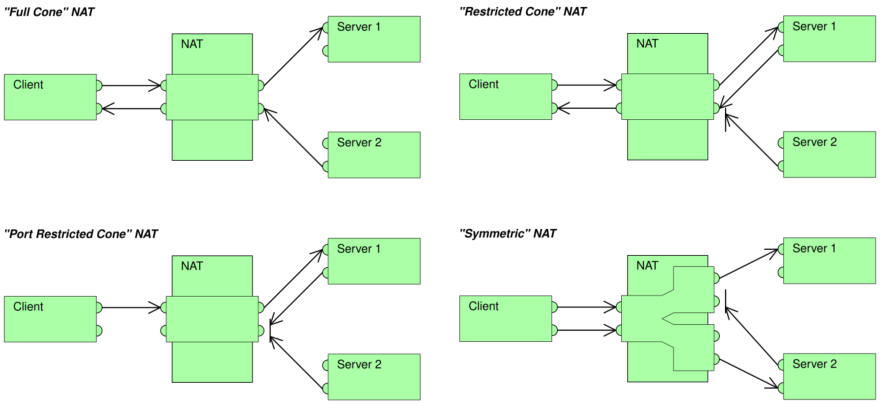

Figura 2: Tipologie di NAT

Fonte: https://en.wikipedia.org/wiki/Network_address_translation #Methods_of_translation

2.2.3 Classless domain

Per fronteggiare meglio l'esaurimento degli indirizzi IP, oltre al NAT, nel 1993 venne adottata l'architettura CIDR. Anche all'interno di questa architettura, gli indirizzi IP sono composti da una serie di bit che identificano la rete e un'altra sequenza di bit che identifica l'interfaccia di un host, ma non sono più di lunghezze predeterminate[93].

In questo caso, pertanto, è necessario aggiungere un'informazione che, dato un indirizzo, ne identifichi la rete di appartenenza. Questa informazione, chiamata "subnet mask", è una sequenza di bit lunga quanto l'indirizzo IP a cui è collegata ed è definita usando la notazione `192.168.1.0/24`, che, in questo caso, indica la rete che ha una netmask in cui i primi 24 bit sono degli `1`[135].

Per convenzione, l'indirizzo che ha l'host ID con tutti `0` indica la rete nella sua totalità, mentre l'indirizzo con tutti i bit dell'host ID a `1` è l'indirizzo di broadcast, quindi un messaggio indirizzato a quell'indirizzo viene inviato a tutti gli host presenti nella rete[114]; perciò gli indirizzi di una rete sono ridotti di 2. Inoltre, tendenzialmente gli indirizzi per le interfacce dei router che fungono da gateway (quindi separano una rete dall'esterno) vengono assegnate a partire dal valore più alto possibile, anche se non è una convenzione sempre adottata. Ad esempio, nella rete `192.168.1.0/24`, all'interfaccia del primo gateway configurato, se si segue la convenzione, sarà assegnato l'indirizzo `192.168.1.254`.

Per trovare la rete di un indirizzo, si deve conoscere la netmask e eseguire un'operazione di AND logico bit a bit con l'indirizzo stesso (vedi tabella 2).

	Notazione binaria	Notazione puntata
Indirizzo IP	`11000000.10101000.10111110.01010101`	`192.168.190.85`
Netmask	`11111111.11111111.11110000.00000000`	`255.255.240.0`
Rete	`11000000.10101000.10110000.00000000`	`192.168.176.0`

Tabella 2: Esempio di applicazione di una subnet mask di 20 bit ad un indirizzo IP

La rete trovata in tabella 2 contiene gli indirizzi assegnabili da `192.168.176.1` a `192.168.191.254`.

Per questo motivo, normalmente le reti "/32" e "/31" non sono utilizzabili in quanto nel primo caso si deve specificare esplicitamente una regola di routing per l'interfaccia (single-host networks), invece nel secondo caso non si avrebbero host disponibili, pertanto ci sono eccezioni per poter costruire una connessione punto-a-punto con soli due host[114].

L'architettura CIDR permette anche l'aggregazione dei prefissi (prefix aggregation): se, ad esempio, sedici reti contigue "/24" hanno i primi 20 bit uguali, possono essere aggregate e rese note ad una rete più grande come una singola voce nella tabella di routing[136]. Questo rende possibile definire reti di qualsiasi dimensione intermedia, permettendo così di assegnare una quantità giusta di indirizzi, lasciandone il meno possibile inutilizzati.

Nonostante la rinnovata disponibilità di indirizzi IP assegnabili, la IETF (Internet Engineering Task Force) ha definito alcuni intervalli di indirizzi riservati per scopi particolari, tra cui le reti `10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16` per reti private e la rete `127.0.0.0/8` per le interfacce di loopback (che fanno tornare alla sorgente i messaggi inviati), tra cui l'interfaccia "localhost" con indirizzo `127.0.0.1`[12].

Tramite il NAT e con l'architettura CIDR, è stato possibile suddividere le reti. Ciò ha giovato molto agli ISP, in quanto possono gestire la loro zona d'influenza come se fosse una rete privata e ciascun utente avrà la propria sottorete (vedi figura 3).

Nell'esempio in figura 3 è presente una rete privata composta da tre host e un router gateway con prefisso `192.168.10.0/24`. Il router, come altri posseduti

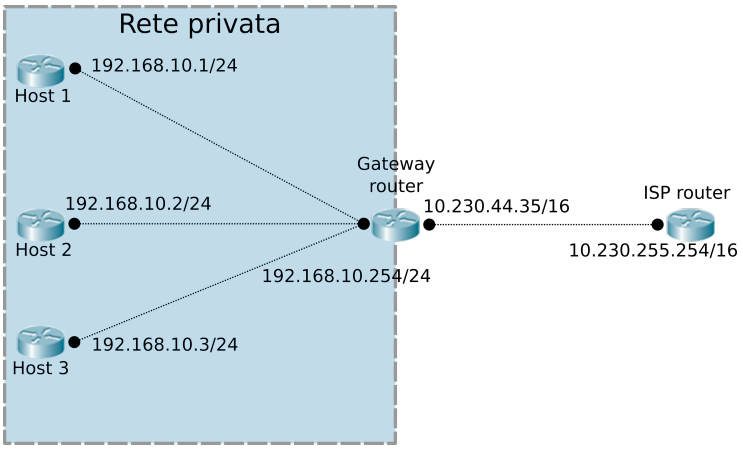

Figura 3: Esempio di sottorete privata con IP masquerading

dai clienti dell'ISP, è collegato ad un router della rete dell'ISP che ha configurato, in questo caso, una rete "/16".

2.3 Porte di rete

Una porta è un costrutto logico che permette di mantenere più connessioni attive su uno stesso host e indirizzare i pacchetti ricevuti al processo corretto. Sono identificate con un numero senza segno di 16 bit ed operano al livello 4 (livello di trasporto) dello stack ISO/OSI. Un numero di porta è sempre associato all'indirizzo dell'host, infatti completa l'origine o la destinazione del pacchetto di dati[126].

L'intervallo dei numeri delle porte è diviso in 3 parti: le porte "well-known" che sono usate principalmente per i processi di sistema che forniscono servizi largamente usati, ad esempio i protocolli Secure Shell (SSH), HTTP e Network Time Protocol (NTP); le porte registrate, che sono assegnate dalla IANA (Internet Assigned Numbers Authority) per servizi particolari sotto richiesta di una specifica organizzazione, come OpenVPN, Network File System (NFS), e il database MySQL; le restanti porte sono libere e cosiddette "effimere", utilizzate per scopi privati e allocazioni automatiche. I rispettivi intervalli per queste tre categorie sono dalla porta 0 alla 10243, dalla 1024 alla 49151, dalla 49152 alla 65535[117].

2.4 Autonomous Systems

Con l'avvento dell'architettura CIDR, la rete internet è stata suddivisa in Autonomous System (AS), cioè, secondo la definizione più recente, un insieme di prefissi IP gestito da uno o più operatori di rete ed avente una singola e ben definita politica di routing[31]. Una politica di routing indica come le decisioni

di instradamento sono prese nella rete internet, quindi l'oggetto delle politiche è lo scambio di informazioni di routing tra AS.

Prendiamo come esempio il seguente schema, in cui gli AS X e Y si scambiano le informazioni di routing:

$$Rete1...ASX <=> ASY...Rete2$$

ASX sa come raggiungere il prefisso Rete1, indifferentemente dal fatto che Rete1 sia all'interno di ASX o di un altro AS che scambia le informazioni con ASX (sia indirettamente che direttamente); assumiamo che ASX sappia instradare i pacchetti verso Rete1. Allo stesso modo, ASY sa come raggiungere Rete2.

Per far procedere il traffico da Rete2 a Rete1, attraverso ASX e ASY, ASX deve annunciare Rete1 a ASY con un protocollo di routing esterno; ciò significa che ASX accetterà il traffico proveniente da ASY verso Rete1. Le politiche di routing (di ASY) sono prese in causa quando ASX decide di annunciare la presenza di Rete1.

Per far fluire il traffico, ASY deve accettare le informazioni di routing di ASX e usarle; altrimenti può decidere di non usarle e non inviare traffico a Rete1 o di utilizzare percorsi che considera più appropriati.

Idealmente, anche se nella pratica raramente, gli annunci e le politiche di accettazione di ASX e ASY sono simmetriche, infatti, per far raggiungere Rete2 dal traffico in arrivo, il suo annuncio e l'accettazione da parte di ASX devono essere portati a termine (specularmente a Rete1). Per la maggior parte delle applicazioni, la connettività unilaterale è inutilizzabile.

In topologie più complesse è probabile che il traffico da Rete1 a Rete2 non usi lo stesso percorso del traffico da Rete2 a Rete1; questa situazione è chiamata routing asimmetrico. Non è sconsigliata in quanto tale, ma spesso può causare problemi ad alcuni protocolli di alto livello (come TCP), perciò dovrebbe essere usato con cautela e solo quando strettamente necessario.

Le politiche non sono definite per ciascun prefisso, ma per gruppi di prefissi, che sono gli AS.

Un AS connesso ad internet deve essere registrato ufficialmente e gli deve essere assegnato un numero univoco globalmente, chiamato Autonomous System Number (ASN). Gli ASN sono usati nei protocolli dei gateway di confine (Border Gateway Protocol, BGP) per indirizzare un pacchetto ad un certo AS o per lo scambio di informazioni di routing tra AS[31]. Correntemente, sono stati assegnati quasi centomila ASN[77].

Riprendendo l'esempio della figura 3, il router gateway può essere associato al router di accesso a Rete1, mentre il router dell'ISP è un router di ASX. La rete dell'AS è una rete "/16" in quanto, nell'esempio, ha necessità di contenere alcune migliaia di interfacce (i propri router e quelli degli utenti).

2.5 Internal Gateway Protocol

Un protocollo IGP è usato per scambiare le informazioni di routing tra i router gateway all'interno di un Autonomous System per poi essere usate per indirizzare i pacchetti di un protocollo di rete come IP. I protocolli IGP si dividono in varie categorie; le principali sono: i protocolli Distance-vector, i protocolli Link-state[111] e i protocolli Path-vector.

2.5.1 Distance-vector protocol

Un protocollo Distance-vector (DV) trova il percorso migliore per un pacchetto basandosi sulla distanza tra il mittente e il destinatario. Solitamente la distanza è calcolata come il numero di router da attraversare per consegnare i dati, ma in alcuni casi sono prese in considerazione anche altre metriche, ad esempio la latenza o altri fattori che influenzano il traffico in un determinato segmento di rete. I router in cui è implementata questa tipologia di protocolli devono scambiarsi diverse informazioni, tra cui la tabella di routing e il numero di hop (salti, sono i router da attraversare) per arrivare alle reti di destinazione (come per esempio le reti private direttamente collegate ad un router). Questi protocolli richiedono che l'invio delle informazioni sia eseguito periodicamente da ciascun router[101].

I router che utilizzano questi protocolli si affidano completamente alle informazioni ricevute dagli altri dispositivi di rete, pertanto non ricostruiscono la topologia di rete.

Gli aggiornamenti alle tabelle di routing sono inviati ai router vicini (cioè direttamente connessi) che sono configurati per usare lo stesso protocollo; appena un router riceve le informazioni da un vicino, rettifica la propria tabella di routing. Questo processo è chiamato "routing by rumour" (routing da voci) in quanto i router utilizzano le informazioni dei vicini e non possono verificare la loro veridicità.

I protocolli DV utilizzano l'algoritmo di Bellman-Ford per ricalcolare le distanze da un nodo verso gli altri, ma questo tipo di protocollo soffre del cosiddetto "count-to-infinity problem" (problema della conta all'infinito): se un router A fornisce un aggiornamento al router B su una destinazione, non c'è modo per B di sapere se lui stesso faccia parte del percorso verso la destinazione, quindi, se ad esempio la destinazione C non è più raggiungibile per A, ma per B è ancora visibile, quando B invia il suo aggiornamento periodico ad A, esso trova un nuovo percorso (attraverso B) per la destinazione C; quando A invia l'aggiornamento periodico, B ha aggiornato la sua tabella e non può più raggiungere C, però vede che lo può raggiungere attraverso A (che in verità poteva raggiungere C attraverso B), quindi modifica la voce della sua tabella di routing impostando la distanza "B->C" come la distanza "A->C" + 1; questo meccanismo si ripete propagandosi per tutta la rete e perciò la distanza viene aumentata fino, idealmente, all'infinito[101].

L'algoritmo di Bellman-Ford permette di calcolare la distanza a partire da un singolo nodo di un grafo rispetto a tutti gli altri. Inizialmente la distanza

dal nodo sorgente viene impostata a 0, mentre per tutti gli altri ad infinito; dopodiché, prendendo in considerazione ogni arco del grafo, si controlla se la distanza dalla destinazione diminuisce, in caso affermativo si aggiorna la distanza dei nodi del grafo; quest'ultimo passaggio si ripete fino ad esaurire gli archi disponibili. Perciò, per ogni iterazione i, l'algoritmo trova tutti i percorsi lunghi i archi di distanza minima dal nodo sorgente. Per un esempio dell'esecuzione dell'algoritmo di Bellman-Ford, vedere la figura 4.

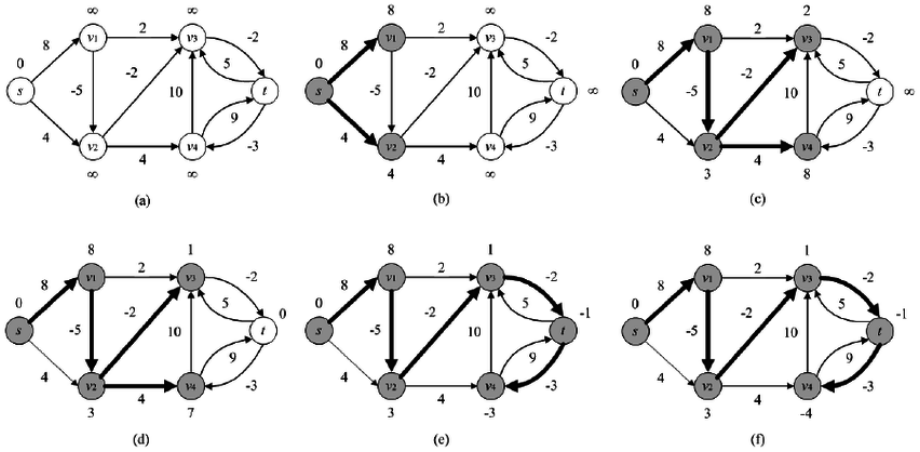

Figura 4: Esempio dell'esecuzione dell'algoritmo di Bellman-Ford

Fonte: https://www.researchgate.net/figure/The-schematic-procedure-of-the-Bellman-Ford-Moore-algorithm-other-related-explanations_fig7_282135528

Il primo protocollo DV è Routing Information Protocol (RIP); la prima versione non è adatta alle reti molto grandi in quanto il numero massimo di hop consentiti è 15, quindi le reti connesse da più di 15 router risulterebbero inaccessibili. Nonostante ciò, RIP usa la tecnica chiamata "split horizon": per evitare il problema della conta all'infinito, il router che riceve un aggiornamento non spedisce al mittente il proprio aggiornamento; esiste una variante chiamata "poison reverse" in cui, invece di non rispondere all'aggiornamento ricevuto, viene inviato un aggiornamento al mittente nel quale le metriche ricevute dallo stesso gli vengono reinviate ma impostate con il valore "infinito" (il più grande possibile)[133][32].

2.5.2 Link-state protocol

I protocolli Link-state (LS) sono un'altra tipologia di protocolli di routing in cui tutti i nodi della rete (i router) si costruiscono una mappa dello stato di connettività della rete. Ogni nodo quindi si calcola indipendentemente il miglior percorso logico da esso a tutte le destinazioni possibili. L'insieme di tutti i percorsi migliori formano la tabella di routing di un router. Le uniche informazioni scambiate tra i router di una rete sono legate alla connettività. Un esempio di protocollo LS è Open Shortest Path First (OSPF), usato generalmente come

IGP nelle grandi reti aziendali, mentre IS-IS (Intermediate System to Intermediate System) è un altro protocollo LS usato principalmente nelle reti dei grandi Service Provider[115].

Distribuzione di una mappa iniziale[103] Come primo passo, si deve fornire una mappa iniziale ai router con cui avviare il protocollo LS. La mappa risultante, se non ci sono errori, è uguale per tutti i router.

1. Determinare i vicini di ciascun nodo: ciascun nodo deve determinare a quali altri router è connesso con collegamenti funzionanti. Per farlo viene utilizzato un protocollo di raggiungibilità periodicamente e separatamente per ciascun router vicino connesso direttamente.

2. Distribuire le informazioni per la mappa: ogni nodo manda regolarmente (e quando la connettività con un vicino è modificata) un messaggio, l'annuncio di link-state, il quale: identifica il nodo che lo produce; identifica gli altri nodi (router o reti) a cui è connesso direttamente; include un numero di sequenza che viene incrementato ogni volta che il mittente di questo annuncio ne invia un aggiornamento. Questo messaggio è inviato ad ogni nodo della rete, i quali devono mantenere in memoria l'ultimo numero di sequenza dei propri vicini. Quando un nodo riceve un nuovo annuncio, controlla il suo numero di sequenza e, se è maggiore del numero salvato e associatogli, lo salva e aggiorna il numero di sequenza, infine viene inoltrata una copia dell'annuncio ai router vicini. In questo modo tutti i nodi della rete ricevono velocemente una copia del nuovo messaggio.

3. Creare la mappa: quando si hanno le informazioni da ciascun nodo della rete, ognuno di essi genera un grafo per la mappa di rete. L'algoritmo scorre la collezione di annunci di link-state e per ciascuno crea i collegamenti nella mappa tra il nodo che ha inviato il messaggio e tutti i suoi vicini. Lo stato di un collegamento è considerato legale solo se entrambi i nodi coinvolti lo riferiscono nell'annuncio, in caso di errori o discordanze il collegamento incriminato non è aggiunto alla mappa.

Calcolare la tabella di routing[92] Questa fase popola le tabelle di routing di ciascun nodo a partire dalla mappa di rete generata precedentemente. Le mappe devono essere uguali tra loro, altrimenti si possono formare dei cicli nei percorsi di routing.

1. Calcolare il percorso minimo: Ciascun nodo esegue un algoritmo sulla mappa per calcolare il percorso più corto tra se stesso e ogni altra destinazione nella rete; generalmente è usata una variante dell'algoritmo di Dijkstra. Un nodo mantiene in memoria due strutture dati: un albero contenente i nodi già visitati e una lista di candidati; entrambe inizialmente sono vuote, all'albero viene aggiunto il nodo stesso. Poi, tutti i nodi collegati direttamente al nodo corrente sono aggiunti all'albero (tranne i nodi

già presenti in una delle due strutture dati), mentre gli altri sono aggiunti alla lista. Successivamente, ciascun nodo nella lista è confrontato con quelli nell'albero e il più vicino viene spostato dalla lista all'albero e connesso al nodo appropriato. I nodi spostati non sono più considerati nelle iterazioni successive dell'algoritmo. Questi passaggi sono ripetuti fino a quando la lista dei candidati contiene almeno un nodo. Il risultato è un albero contenente tutti i nodi della rete che come radice ha il nodo che ha calcolato i percorsi.

2. Riempire la tabella di routing: Per riempire la tabella è necessario attraversare l'albero partendo dalla radice fino al nodo destinazione, ricordandosi anche il primo nodo intermedio del ramo del percorso seguito per ciascuna destinazione.

Motivi di fallimento[104] Nel caso in cui i router di una rete non calcolino la tabella di routing a partire dalla stessa mappa, è possibile che si formino dei cicli di routing infiniti. Ad esempio, due router vicini possono pensare che l'altro nodo faccia parte del percorso più corto, quindi qualsiasi pacchetto che arriverà ad uno dei due rimbalzerà tra i due indefinitamente. Questo può accadere poiché ciascun nodo calcola la propria tabella di routing senza interagire con gli altri dispositivi.

2.5.3 Path vector protocol

I protocolli path vector mantengono informazioni sui percorsi all'interno della rete che vengono aggiornati dinamicamente, in questo modo gli aggiornamenti che ritornano al nodo iniziale vengono scovati e scartati facilmente. L'algoritmo alla base di questi protocolli è spesso usato in congiunzione all'algoritmo di routing di Bellman-Ford per evitare il problema della conta all'infinito. È una tipologia di protocolli differente da DV e LS in quanto ogni voce nella tabella di routing contiene la rete di destinazione, il router successivo (a cui inviare i pacchetti) e il percorso per arrivare alla destinazione.

2.6 Border Gateway Protocol

BGP è un protocollo path vector usato principalmente per connettere i router di confine, sia all'interno dello stesso AS (definito Internal BGP, iBGP) oppure tra AS differenti (chiamato external BGP, eBGP), anche se generalmente è più comune il secondo caso.

2.6.1 Funzionamento

Idealmente, anche se spesso è impraticabile, tutti i possibili nodi dovrebbero essere collegati a ciascun altro. BGP, "il più modulare tra i protocolli di routing"[9], fornisce un espediente e strumenti per verificare quanto efficacemente è stato configurato.

I vicini per BGP, chiamati "peer" (pari), sono stabiliti con una configurazione manuale tra i router e viene creata una sessione TCP. I comunicanti mandano un messaggio "keep-alive" per mantenere attiva la sessione ogni sessanta secondi.

Quando BGP viene utilizzato come protocollo esterno, tutti i percorsi ricevuti da un pari eBGP (un router di confine) sono propagati a tutti i nodi interni ed esterni; mentre i percorsi ricevuti da un pari interno sono inviati nuovamente solo ai router esterni. Questa modalità di propagazione richiede che tutti i nodi interni siano connessi completamente tra di loro.

Il modo in cui i percorsi sono propagati può essere controllato dettagliatamente dal meccanismo della "mappa dei percorsi". Essa consiste in una serie di regole che descrivono, per i percorsi che corrispondono a determinati criteri, quale azione si deve eseguire (ad esempio scartarlo o modificarne alcuni parametri prima di inserirlo nella tabella di routing).

Per prendere delle decisioni che riguardano i pari, un router BGP utilizza una macchina a stati finiti. Per ciascuna sessione viene mantenuta una variabile che traccia lo stato attuale del dispositivo[123].

2.6.2 Adattabilità interna

Dato che idealmente in una rete BGP tutti i nodi dovrebbero essere interconnessi, la quantità di collegamenti aumenterebbe quadraticamente rispetto al numero di dispositivi. Perciò BGP offre delle opzioni per alleviare i costi di connessione: i "route reflector" o le confederazioni BGP.

1. Route Reflector (RR): È un tipo di configurazione per i router di una mesh BGP che li eleva a Route Reflector server e serve per ridurre il numero di connessioni in una rete interna BGP, infatti una parte di router nell'Autonomous System possono essere configurati come pari solo del RR, e quindi possono essere connessi anche solo ad esso (i router connessi con il RR sono definiti "client"), ma non sono vietate connessioni tra client dello stesso RR (vedi figura 5). Questo, ad esempio, in una rete di 10 nodi di cui un RR, invece di dover amministrare 90 voci nelle tabelle di routing (il numero di connessioni in una rete completamente connessa è calcolabile con la formula $< numero\_di\_nodi > * < numero\_di\_nodi - 1 > /2$, in questo caso $10 * 9/2 = 45$, ma si devono aggiungere le voci in entrambi i router coinvolti), si hanno solo 18 voci (una in ogni nodo client del RR e 9 nel router designato). Questo permette di diminuire il numero di connessioni in maniera significativa.

   Il RR e i router connessi solo ad esso formano un Cluster. L'identificatore del cluster è inserito in ciascun percorso annunciato dal RR ai suoi sottoposti o ai suoi pari.

   I percorsi sono propagati dal RR secondo le seguenti regole:

   - Se il percorso è ricevuto da un pari non client, è propagato ai pari client e ai pari di confine.

- Se il percorso è ricevuto da un pari client, viene propagato ai pari non client, ai pari client (tranne il mittente) e ai pari eBGP.

2. Confederazioni BGP: Le confederazioni sono un insieme di AS visibile come un AS solo dalla rete. Ciascun AS facente parte di una confederazione contiene una rete completamente connessa con iBGP ed i router di confine degli AS confederati, sebbene siano pari eBGP, utilizzano iBGP per scambiarsi i percorsi. In questo modo conservano informazioni riguardanti il prossimo router, le metriche e le preferenze locali. Pertanto, all'interno della confederazione solo i router di confine di ciascun AS sono connessi completamente tra loro, mentre i router interni sono "nascosti" all'interno del proprio AS.

I RR e le confederazioni BGP possono essere combinati per creare configurazioni più complesse, ad esempio creando una gerarchia di route reflector[112].

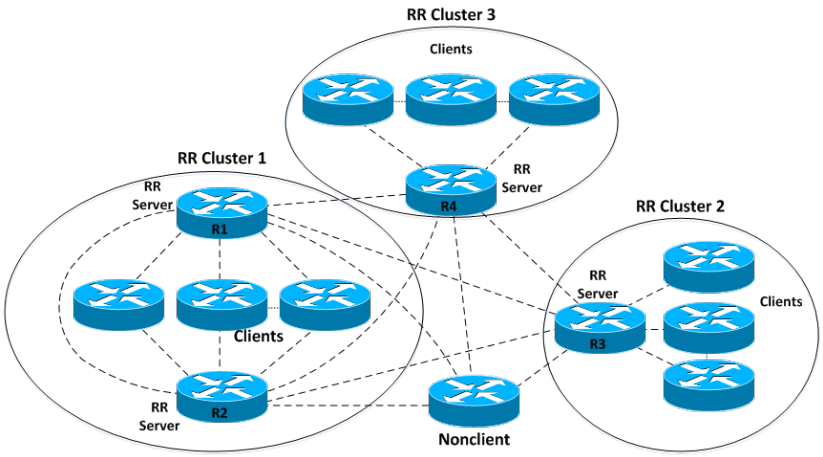

Figura 5: Esempio di mesh BGP con Route Reflector
Fonte: http://netconsultings.com/portfolio-posts/bgp-route-reflectors/

2.6.3 Stabilità

Le tabelle di routing gestite da BGP sono continuamente adattate per riflettere i cambiamenti della rete, come i collegamenti che si interrompono o vengono ripristinati oppure i router che si spengono o si riaccendono. Nel complesso di una rete questi cambiamenti sono molto frequenti, ma per un router o collegamento particolare si suppone che le modifiche siano abbastanza rare. Se un router è configurato male, potrebbe entrare in un ciclo di attività e inattività. Questa ripetizione di ritiri e riannunci del collegamento, chiamata "route flapping" (sfarfallamento del percorso), può causare un eccesso di attività dei router che sono a conoscenza del collegamento non funzionante, in quanto lo stesso percorso è inserito e rimosso continuamente dalla tabella di routing. Dato che BGP

potrebbe interrompere il traffico quando dei percorsi sono aggiornati, il route flapping potrebbe ritardare gli aggiornamenti importanti della rete, quindi è stato sviluppato un meccanismo per ridurre gli effetti del route flapping.

Questa funzionalità è chiamata "route flap damping" (attenuazione del route flapping), che fa decadere le richieste di aggiornamento del router malfunzionante in maniera esponenziale: la prima volta che un percorso diventa non disponibile ed è annunciato nuovamente subito dopo, non accade nulla; a partire dalla seconda ricorrenza di un evento simile, le richieste sono scartate per un certo periodo di tempo; ogni aggiornamento aumenta l'intervallo di tempo in modo esponenziale. Dopo che le anomalie sono cessate di esistere e dopo un arco di tempo ragionevole, è possibile riscrivere le voci riguardanti i percorsi che hanno provocato il damping[134].

2.6.4 Sicurezza

Normalmente, i router che utilizzano BGP accettano i percorsi annunciati da altri router BGP. Questo permette di automatizzare e decentralizzare l'instradamento del traffico in internet, ma lo lascia vulnerabile a interruzioni accidentali o volontarie, chiamate "sabotaggio BGP". Dato che BGP è ormai integrato in profondità nel sistema internet, la correzione di questa vulnerabilità è una sfida sia tecnica che economica[128].

2.7 SDN - Software-defined networking

La tecnologia SDN (software-defined networking, connessione di rete definita in software) è un approccio alla gestione di rete che permette una configurazione dinamica e programmaticamente efficiente per migliorare le prestazioni e il monitoraggio della rete. Questa tecnologia dovrebbe fornire funzionalità più affini alla rete odierna, che necessita di flessibilità e semplicità nel risolvere i problemi, rispetto all'architettura statica delle reti tradizionali, le quali sono decentralizzate e complesse. SDN cerca di centralizzare la gestione delle reti dividendo il piano di controllo dal piano dei dati[132]. Le operazioni di controllo sono centralizzate in almeno un controllore che detta le politiche di rete (vedi figura 6). Per questo motivo, è possibile assimilare concettualmente SDN al cloud computing.

Un'eventuale migrazione ad SDN non comporterebbe cambiamenti nei protocolli e standard di rete (come IP ed Ethernet) in quanto SDN offre la retrocompatibilità ad essi.

Il concetto di SDN ha iniziato ad evolversi fino dalla metà degli anni '90, sebbene progetti come Ethane[84] e OpenFlow[70] hanno pubblicato le prime implementazioni di SDN a partire dal 2006. Il primo, un'architettura, permette di definire una politica estesa a tutta la rete e la impone ad ogni switch collegato ad essa; il secondo è un protocollo che fornisce delle interfacce per amministrare in remoto le tabelle di inoltro dei dispositivi connessi. I router che implementano OpenFlow possiedono una tabella di routing e un'interfaccia applicativa per comunicare con il controllore.

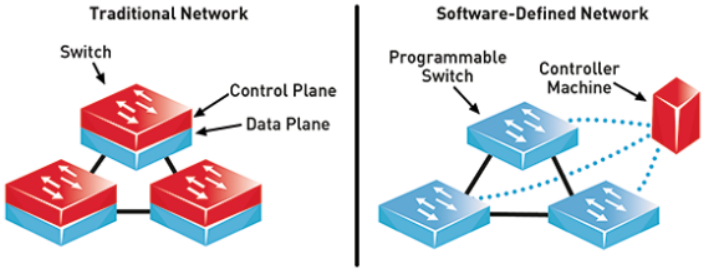

Figura 6: Confronto tra un'architettura di rete tradizionale e una rete SDN
Fonte: https://www.commsbusiness.co.uk/features/software-defined-networking-sdn-explained/

Inoltre, la domanda per i servizi cloud sta crescendo rapidamente; sebbene questi servizi siano centralizzati nei data center, pongono una sfida imponente ai propri fornitori: con la crescita veloce delle richieste dei clienti, l'operatore è tenuto a rispondere adeguatamente considerando server e componenti di rete aggiuntivi, alta qualità del servizio e un'architettura sicura che rispetti gli standard[4].

Le importanti barriere che SDN deve superare sono[4]:

1. Adattabilità: definisce la capacità di SDN, soprattutto nel piano di controllo, di gestire e elaborare un carico di lavoro in aumento. L'adattabilità mira ad accrescere la capacità di SDN implementando meccanismi come delegazione, creazione di cluster e elaborazione veloce.

2. Affidabilità: Un SDN è considerato affidabile quando notifica i fallimenti di consegna di un pacchetto dati in tempo reale. In una rete del genere dovrebbe essere specificata un'affidabilità minima per le consegne di dati importanti. Nelle implementazioni attuali, i controllori SDN devono fornire la notifica in tempo reale per una consegna affidabile e tempestiva.

3. Alta disponibilità: L'alta disponibilità è un aspetto importante dei servizi che devono essere responsivi ogni volta che un utente gli invia una richiesta. La disponibilità è spesso misurata come percentuale di tempo di reperibilità in un dato anno. La non disponibilità di servizi è generalmente causata da crash di sistemi o blackout di rete. I fornitori di rete spesso forniscono servizi di backup per offrire alta disponibilità, dispiegando in modo ridondante server e componenti di rete.

4. Elasticità: è l'abilità di SDN di adattare dinamicamente la sua capacità incrementando o riducendo le risorse disponibili per affrontare le variazioni nel carico di lavoro. Generalmente, l'elasticità è concentrata sul piano di controllo e potrebbe essere assimilata all'adattabilità.

5. Sicurezza: La sicurezza di SDN consiste nel proteggere i dati dai furti o i componenti hardware e software da un danneggiamento e dalle interru-

zioni del servizio. Rendere sicuro SDN comprende la sicurezza fisica dei componenti e le vulnerabilità logiche che provengono dalla rete o dai dati.

6. Prestazioni: Le prestazioni prendono in considerazione la quantità di operazioni eseguite dai componenti SDN in base al tempo o alle risorse usati. Per SDN le metriche importanti sono la larghezza di banda(il numero di frequenze che il segnale può assumere, più è alto maggiore è la quantità di dati trasmissibili nell'unità di tempo), la quantità di dati trasmessi effettivamente (throughput), la latenza (l'intervallo di tempo tra l'invio di un segnale ad un sistema e la risposta del sistema), le variazioni del segnale (jitter, possono variare caratteristiche del segnale come l'ampiezza o la frequenza).

7. Resilienza: è la capacità di mantenere un livello di servizio accettabile anche in caso di fallimento di rete o di un nodo. Quando un elemento SDN è difettoso, la rete dovrebbe fornire un servizio continuo con le stesse performance. Per aumentare la resilienza, i potenziali rischi devono essere trovati e risolti.

8. Dipendenza: indica quanto si può dipendere da SDN, è strettamente legata a disponibilità e affidabilità, infatti mira a prevenire i guasti e ad implementare un meccanismo di tolleranza agli errori per garantire i servizi anche in caso di difetto.

2.7.1 Architettura

L'architettura di SDN è divisa in tre strati:

1. Piano applicativo: definisce le regole di routing ed offre diversi servizi, ad esempio un firewall, il controllo degli accessi ed il monitoraggio della rete. Questo strato è il responsabile dell'astrazione della gestione della rete di SDN.

2. Piano di controllo: è un'astrazione della topologia di rete. Il controllore è il componente principale e si occupa di stabilire le politiche di trattamento dei dati e delle tabelle di instradamento, oltre ad astrarre la complessità di rete, ottenere informazioni sullo stato della rete e mantenere una visione aggiornata della rete. È possibile avere più controller in questo piano che comunicano tra di loro.

3. Piano dei dati: offre i dispositivi di rete (come gli switch, i router e i punti di accesso) sia fisici che virtuali ed è responsabile di tutte le attività svolte sui dati tra cui l'inoltro, la frammentazione e il riassemblamento.

Inoltre, è possibile inserire un piano trasversale ai tre descritti sopra che astrae l'amministratore di rete che si interfaccia col piano applicativo, configura il piano di controllo e installa i dispositivi del piano dei dati[6].

2.7.2 Protocollo di comunicazione

Comunemente, l'interfaccia tra controllori e dispositivi di rete si affida al protocollo OpenFlow, le cui specifiche permettono ai primi di controllare i secondi.

Esistono tre modalità di comunicazione: controllore-a-switch, asincrona, simmetrica.

La comunicazione controllore-a-switch è responsabile della configurazione dell'inizializzazione, degli switch e della tabella di flusso.

La comunicazione asincrona è usata da un dispositivo per informare il proprio controllore attraverso i messaggi di "pacchetto in entrata", stato della porta e flusso rimosso.

La comunicazione simmetrica non ha restrizioni su chi può iniziare la conversazione. È usata, ad esempio, per scovare problemi nel collegamento tra switch e controllore.

Il canale di comunicazione tra switch e controllore può essere criptato usando il protocollo TLS (Transport Layer Security) o non criptato, in base alle esigenze di sicurezza[4].

3 Cloud computing

La locuzione "Cloud Computing" si riferisce al modello che consente di accedere ad una serie di risorse computazionali configurabili (reti, server, spazio di archiviazione, applicazioni e servizi) convenientemente da qualsiasi luogo e da qualsiasi tipo di dispositivo (per una rappresentazione semplificata vedere la figura 7), definito dal National Institute of Standards and Technology (NIST) in un documento del 2011[64].

3.1 Software snelli e software pesanti

Grazie al cloud computing, si possono distribuire software "snelli" (thin client), i quali non eseguono calcoli al loro interno ma si interfacciano ad un server remoto per carpire le informazioni da visualizzare e fornire le funzionalità promesse. La controparte dei software snelli sono i software "pesanti" (fat client), che invece implementa almeno alcune delle proprie funzionalità all'interno del software stesso[80].

Un esempio di software snello è un web browser, infatti per fornire la maggior parte, se non tutte, delle sue funzionalità deve collegarsi ad un server. Dall'altro lato, un esempio di software pesante è un videogioco, il quale, per offrire all'utente un'esperienza ottimale, è elaborato dal sistema locale ed eventualmente si può connettere in rete per motivi sociali[80].

Ciascuno dei due tipi ha i suoi vantaggi e svantaggi (tabella 3), ma in un mondo sempre più interconnesso e connesso perennemente ad una rete internet i software snelli si stanno diffondendo velocemente, sebbene i software pesanti siano ancora molto usati.

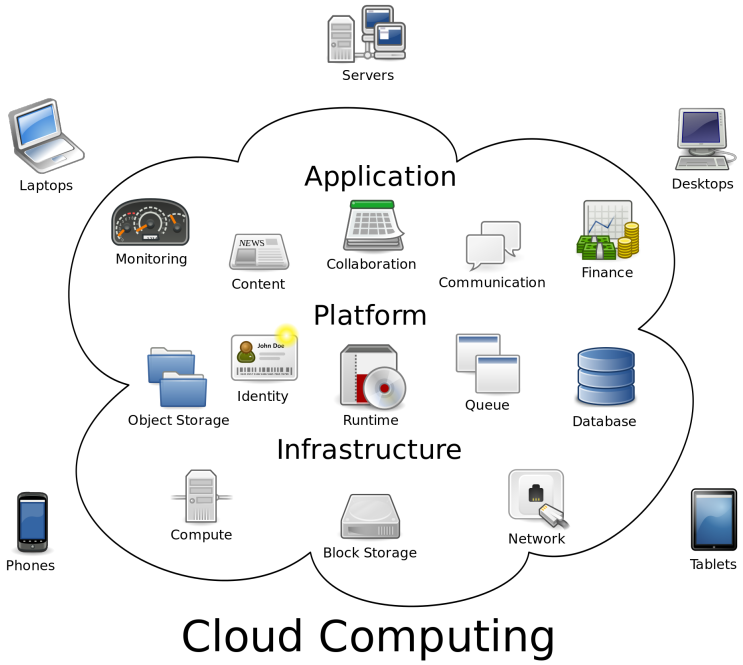

Figura 7: Rappresentazione semplificata del cloud e dei servizi che offre
Fonte: https://8kmprod.s3.amazonaws.com/wp-content/uploads/2016/03/Cloud_computing.png

	Software snello	Software pesante
Esecuzione offline	La maggior parte delle funzionalità non è disponibile	La maggior parte delle funzionalità è disponibile
Risorse locali	Consuma meno risorse come disco, RAM e CPU	Tendenzialmente consuma più risorse locali
Latenza di rete	Potrebbe servire una connessione veloce per usufruire a pieno delle funzionalità	Non è necessaria una connessione veloce per usufruire delle funzionalità
Dati	I dati sono generalmente salvati nei server	I dati potrebbero essere salvati localmente

Tabella 3: Confronto tra software snelli e pesanti
Fonte: https://simplicable.com/new/thin-client-vs-thick-client

3.2 Caratteristiche essenziali dei sistemi cloud

Per il cloud computing, il NIST ha definito cinque caratteristiche essenziali[64].

3.2.1 On-demand self-service

Un consumatore può richiedere unilateralmente risorse computazionali (come disponibilità dei server o spazio di archiviazione) automaticamente quando sono richieste, senza avere interazioni umane con ciascun fornitore di servizi.

3.2.2 Broad network access

Le funzionalità sono disponibili in rete e vi si può accedere attraverso meccanismi standard che promuovono l'uso di dispositivi eterogenei (telefoni cellulari, tablet, computer portatili, workstation).

3.2.3 Resource pooling

Le risorse fornite sono raggruppate per servire più consumatori usando un modello a più occupanti, con risorse fisiche e virtuali assegnate e riassegnate dinamicamente in base alla richiesta del consumatore. C'è un senso di indipendenza della locazione, infatti generalmente il consumatore non ha controllo sulla posizione esatta delle risorse fornite, ma può definire l'ubicazione ad un livello di astrazione più alto, come nazione, territorio o datacenter. Esempi di risorse includono spazio di archiviazione, potenza di calcolo, memoria e banda di rete.

3.2.4 Rapid elasticity

Le risorse sono fornite e rilasciate dinamicamente, talvolta anche automaticamente, per poter adattarsi alla domanda del consumatore. Per esso, le risorse disponibili appaiono illimitate e possono esserne richieste una qualsiasi quantità in qualsiasi istante.

3.2.5 Measured service

I sistemi cloud monitorano e ottimizzano l'uso delle risorse applicando meccanismi di misurazione ad un livello di astrazione appropriato per il tipo di servizio (ad esempio la banda di rete usata, la potenza di calcolo in uso, lo spazio di archiviazione occupato o gli utenti attivi). Il monitoraggio delle risorse è trasparente, quindi sia il fornitore che il consumatore possono controllare e riferire l'uso delle risorse di un determinato servizio.

3.3 Modelli di servizio

Inoltre il NIST ha definito tre modelli di servizi in cloud[64]:

3.3.1 IaaS (Infrastructure as a Service)

Un fornitore di infrastrutture fornisce server e MV (macchine virtuali), risorse di archiviazione, reti di comunicazione. Il fruitore del servizio deve configurare manualmente ciò che ha a disposizione, a partire dall'installazione del sistema operativo in ciascun dispositivo fino alla quantità di spazio di archiviazione

da fornire. L'utente conosce e progetta la struttura del servizio, ma non controlla l'infrastruttura del cloud sottostante, a meno di una limitata scelta di componenti aggiuntivi (ad esempio un firewall)[66].

Spesso IaaS implica l'uso di un orchestratore, come OpenStack o Apache Cloudstack, che permette, tra le altre cose, di amministrare la creazione di una MV, sceglierne l'host fisico, migrarla tra gli host disponibili, gestire i volumi, riportare le informazioni sull'uso.

Infine, molti fornitori IaaS offrono risorse aggiuntive, come un catalogo di immagini (simili a snapshot) per MV, vari tipi di spazi di archiviazione, firewall, bilanciatori di carico, indirizzi IP, reti locali virtuali (VLAN) e suite di software[110].

3.3.2 PaaS (Platform as a Service)

Come nei servizi IaaS, sono forniti server e risorse, inoltre vengono resi disponibili middleware, strumenti per lo sviluppo e servizi di gestione e analisi. Il fruitore dovrà perciò sviluppare le applicazioni che saranno ospitate sulla piattaforma affittata, ma non è necessario che conosca la struttura interna della piattaforma.[67]

Ad esempio è possibile richiedere delle macchine con Ubuntu Server 18.04 installato su cui si andrà poi a sviluppare una propria applicazione, ad esempio un servizio web, senza doversi preoccupare dello spazio di archiviazione o di installare sistemi di monitoraggio.

3.3.3 SaaS (Software as a Service)

Sono forniti software gratuiti o a pagamento che l'utente può usare collegandosi da qualsiasi suo dispositivo, utilizzando un qualsiasi sistema di identificazione per accedere ai file e dati personali. In questo caso, l'utente non viene a conoscenza della struttura sottostante, tant'è che ha la parvenza di comunicare con un solo server monolitico, mentre normalmente l'applicazione e i componenti annessi sono dispiegati in macchine diverse, eventualmente anche lontane geograficamente.[68]
In questa tipologia di servizi, l'utente fruisce direttamente del servizio, senza dover configurare o installare alcuna applicazione.

Col tempo si sono sviluppati però altre tipologie di servizio[89], tra le quali:

3.3.4 CaaS (Container as a Service)

Questo modello è una variante di IaaS che si trova tra IaaS e PaaS, in cui al posto delle macchine virtuali sono forniti degli strumenti per amministrare container. Come già visto, i container offrono prestazioni migliori, eliminano i problemi di sovraccarico o di sottocarico e permettono di applicare tariffe in base all'uso effettivo delle risorse al posto di una quantità fissa predeterminata[34].

I container possono essere raggruppati in unità logiche in base al loro uso, ad esempio si può avere un'unità per la fase di testing di un'applicazione web e un'altra unità per il ramo di produzione (quindi quello esposto al pubblico) e, grazie al load balancing, si possono effettuare test e rendere pubblici gli aggiornamenti in tempo reale e senza spegnere i server.

I servizi CaaS godono del supporto degli orchestratori che automatizzano le principali attività di configurazione e gestione dei container, tra questi si citano Kubernetes e Docker Swarm[34].

I principali fornitori di CaaS sono IBM, Heroku, Amazon Web Services (AWS), e Google[34].

3.3.5 MBaaS (Mobile Backend as a Service)

Spesso le applicazioni web e per dispositivi mobili necessitano di una serie di funzionalità comuni, ad esempio un servizio di notifiche, l'integrazione con i social network e dello spazio di archiviazione in cloud. Tendenzialmente questi aspetti sono gestiti attraverso API separate, il cui studio e configurazione possono risultare onerosi in termini di tempo, perciò un servizio di tipo MBaaS mette a disposizione un'interfaccia unificata e semplificata per poter accedere ad un database e talvolta anche un kit di sviluppo per programmare l'applicazione da distribuire, in modo tale da massimizzare la compatibilità con l'interfaccia[119]. In alcun casi sono anche offerti funzionalità aggiuntive come la distribuzione dell'applicazione o il monitoraggio di alcune metriche[72].

3.3.6 Serverless computing

Il modello Serverless computing è simile a un servizio PaaS, ma l'adattamento delle risorse alle richieste e le operazioni di manutenzione sono eseguiti dal fornitore del servizio[69]. In più al cliente viene addebitato solo il costo delle risorse effettivamente usate e non di una quantità prestabilita di esse come invece accade nei servizi di tipo PaaS, perciò possono risultare più convenienti economicamente (vedi figura 8).

3.3.7 FaaS (Function as a Service)

Questo modello è un tipo particolare di serverless computing, fornisce una piattaforma per sviluppare, eseguire e gestire le funzionalità di un'applicazione, senza dover pensare all'infrastruttura sottostante. All'utente vengono messe a disposizione funzioni e le risorse per eseguirle al presentarsi di un evento, per esempio un click in una pagina o applicazione web. Per questo, al cliente viene addebitato solo il costo dell'uso effettivo delle risorse e non anche quello di inattività, come nel modello serverless. Gli svantaggi di questo modello sono una maggiore complessità nell'incorporare le funzioni FaaS nei test e la difficoltà aggiunta in fase di debugging data dalla gestione delle funzioni da parte di un terzo[10].

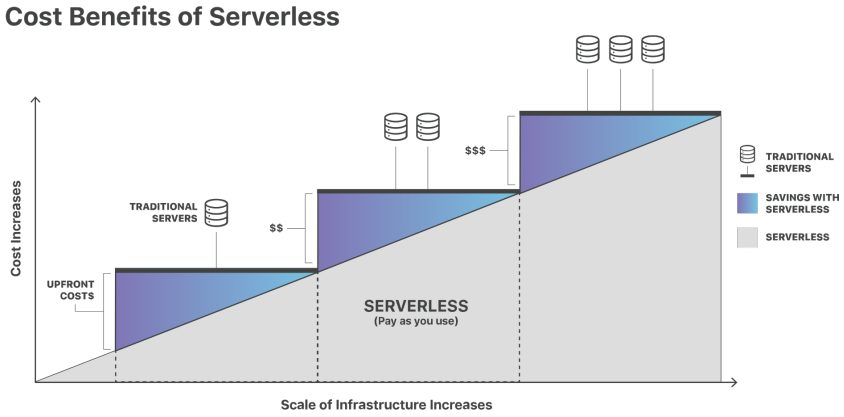

Figura 8: Confronto di costo tra una soluzione PaaS e una serverless computing
Fonte: https://www.cloudflare.com/learning/serverless/what-is-serverless/

3.3.8 NaaS (Network as a Service)

La definizione Network as a Service descrive tutti i servizi che hanno come obiettivo la connettività della rete[121]. Alcuni di questi servizi sono[121]:

- Reti private virtuali (VPN): estende una rete privata e le risorse che gli appartengono attraverso internet. I client di questa rete possono scambiarsi dati utilizzando reti condivise o pubbliche utilizzando funzionalità e politiche di una rete privata comune.
- Bandwidth on Demand (BoD): è un meccanismo utilizzato per assegnare dinamicamente la capacità di rete in base al numero di nodi o di utenti.
- Virtualizzazione della rete mobile: è un modello nel quale gli operatori di rete indipendenti costruiscono la propria rete e ne vendono l'usofrutto a terzi. Un operatore la cui rete è costituita solo da reti virtuali affittate è definito "mobile virtual network operator" (MVNO).

3.4 Modelli di dispiegamento

Infine il NIST ha anche descritto i possibili modelli di dispiegamento[64]:

3.4.1 Cloud privato

L'infrastruttura è fornita per l'uso esclusivo di una singola organizzazione che comprende consumatori multipli. Può essere posseduta, gestita e manovrata dall'organizzazione stessa, da un terzo o da una combinazione dei due. Può essere locata sia nella sede dell'organizzazione che fuori.

3.4.2 Cloud di comunità

In questo caso, l'infrastruttura è fornita ad una specifica comunità di consumatori provenienti da organizzazioni che hanno interessi in comune (ad esempio requisiti di sicurezza o politiche gestionali). Questa tipologia di dispiegamento può essere gestita da una o più organizzazione della comunità, da un terzo o da una combinazione dei due; inoltre può essere dislocato in sede o fuori sede.

3.4.3 Cloud pubblico

L'infrastruttura è adibita all'uso del pubblico generale. Potrebbe essere posseduta e gestita da un'impresa, un'organizzazione accademica o governativa o una combinazione di questi. È sempre dislocato nella sede del fornitore.

3.4.4 Cloud ibrido

È la combinazione di due o più infrastrutture cloud (private, di comunità o pubbliche) che rimangono entità separate, ma sono legate da tecnologie standardizzate o proprietarie che abilitano la portabilità dei dati e delle applicazioni, cioè è possibile usare la stessa applicazione e gli stessi dati nei diversi cloud.

4 Docker

Docker è la piattaforma più usata[7] per creare, eseguire e condividere le applicazioni nei container[15]. Infatti fornisce una suite di strumenti per amministrare i container in un dispositivo, al cui interno si può trovare il demone `dockerd`, che gestisce i container attivi in background, e l'applicativo `docker`, che consente all'utente di eseguire comandi da terminale per interfacciarsi con il suddetto demone.

Utilizzando degli orchestratori, è possibile raggruppare più host Docker in un'entità unica, definita "swarm" (sciame). L'orchestratore fornito da Docker è chiamato `Docker Swarm`.

Inoltre, Docker introduce la nozione di "immagine": essa contiene il codice o il file binario da eseguire, le librerie e le dipendenze che servono al processo, oltre a qualsiasi altro oggetto del file system necessario[16]. Insieme alle immagini, è definito il concetto di layer, cioè l'istantanea di un'immagine basata sui comandi precisati nel file di configurazione, il cosiddetto "Dockerfile", incorporando così un sistema di controllo di versione, similmente a "git", basato sulle differenze tra un layer e il successivo[8] (vedi figura 9).

4.0.1 Costruzione di un'immagine

Un Dockerfile contiene le specifiche per un'immagine: partendo da un'immagine di base, permette di eseguire alcune azioni su di essa. Un esempio basilare è il seguente[13]:

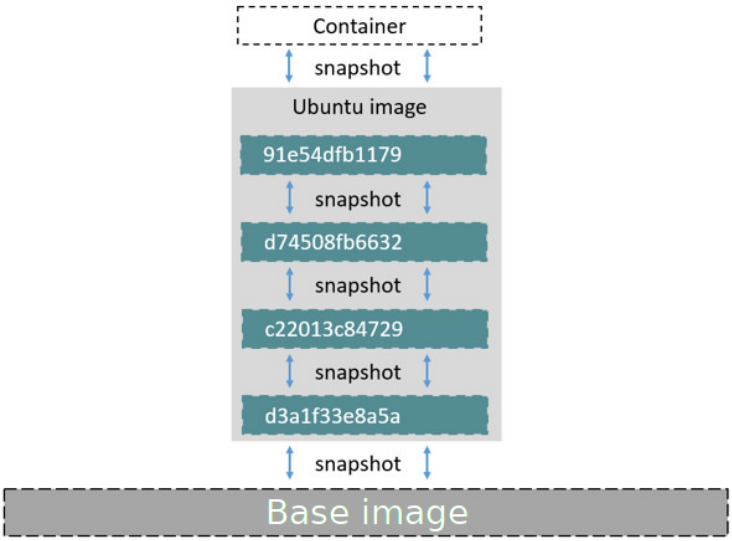

Figura 9: Esempio di immagine Docker con layer
Fonte: *https* : *//blog.csdn.net/weixin*$_2$8738845/*article/details*/81811645

```
1 FROM ubuntu:18.04
2 COPY . /app
3 RUN make /app
4 CMD python /app/app.py
```

Questa immagine è derivata da quella del sistema operativo Ubuntu 18.04 (riga 1), in cui viene copiata, dal file system ospitante a quello del container, una cartella contenente dei file di codice per un'applicazione (riga2); nella terza riga viene eseguito un comando per compilare l'applicazione dal codice sorgente; infine viene eseguita l'applicazione utilizzando l'interprete python. Viene aggiunto un layer all'immagine sia per ciascuna azione specificata che nel momento in cui il container viene avviato, nel quale vengono inserite tutte le modifiche fatte durante l'esecuzione, come la creazione, la modifica o la cancellazione di file (vedi figura 9)[13].

Parallelamente a git, è possibile salvare un'immagine ed i suoi layer come nuova immagine di base localmente utilizzando il comando `docker commit` e inviare le modifiche ad un server remoto usando `docker push`. Il server remoto deve ospitare un registro delle immagini, cosiddetto registro Docker, da cui si attingerà nel caso in cui serva un'immagine, esplicitamente con `docker pull` e implicitamente con `docker run`. Il demone `dockerd` mantiene localmente una cache di immagini che può recuperare nel caso in cui sia richiesto di creare un nuovo container partendo da un'immagine già richiesta in precedenza (figura 10).

L'azienda sviluppatrice di Docker mette a disposizione un registro pubblico (chiamato "Docker Hub"), che è quello configurato all'installazione di Docker,

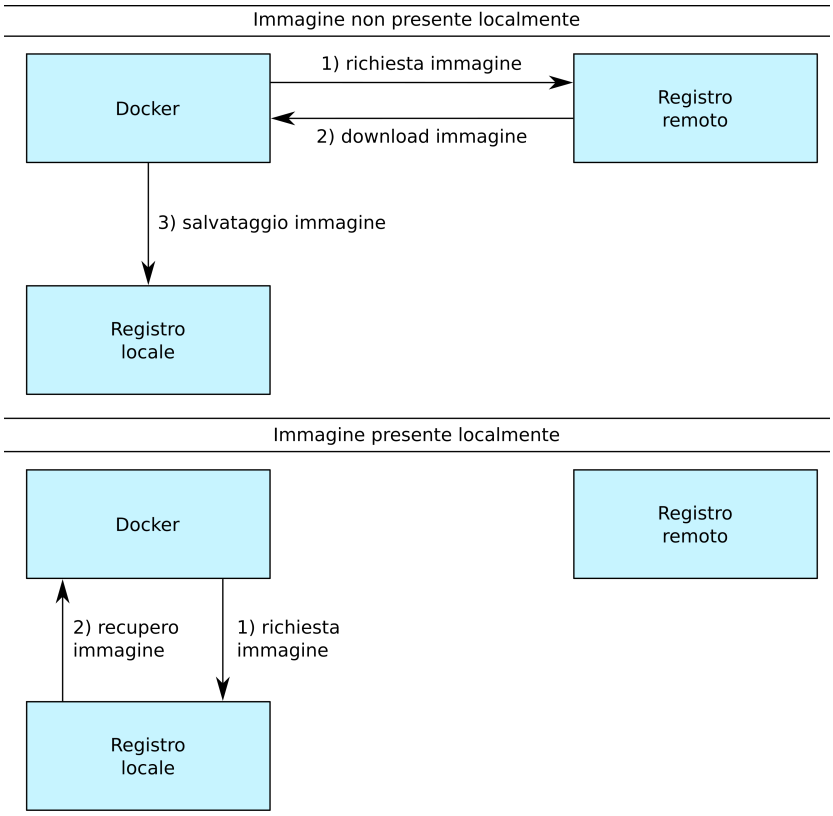

Figura 10: Flusso delle immagini Docker in base alla presenza nel registro locale o meno

a cui è possibile inviare le proprie immagini previa registrazione gratuita e permette di scaricare le immagini base presenti senza costi.
A partire dal Dockerfile è quindi possibile costruire un'immagine personalizzata, che potrà essere usata all'interno dei container, utilizzando il comando `docker build .` da riga di comando nella cartella in cui il file è presente.

4.0.2 Configurazione di rete

Docker adotta lo standard di networking denominato Container Networking Model (CNM) per far comunicare i container tra loro e con l'esterno.
CNM comprende interfacce sia per plugin di gestione degli indirizzi IP (IPAM) che per plugin di rete: i primi permettono di amministrare intervalli di indirizzi e la loro allocazione; i secondi sono usati per creare ed eliminare reti e aggiungere o rimuovere i container da esse.
Docker offre un'implementazione di CNM, anche se l'amministratore di sistema

può decidere di utilizzare dei plugin di terze parti[71].
Il modello CNM è composto da tre componenti[142]:

- Sandbox: è un ambiente operativo di rete isolato che mantiene la configurazione dello stack di rete del container, includendo le interfacce di rete, le regole di indirizzamento dei pacchetti e la configurazione della risoluzione dei nomi. In questo modo, i container possono essere completamente isolati tra loro. Un sandbox può contenere multipli Endpoint da diversi Network.

- Endpoint: rappresenta il dispositivo con cui il container si connette alla rete (si può immaginare una scheda di rete). Un Endpoint appartiene ad un solo container e ad un solo Network.

- Network: delinea una serie di Endpoint che possono comunicare tra loro direttamente, basata sulle tecnologie "bridge" e "vlan" di Linux. Nel sistema ospitante, ciascun Network è uno spazio dei nomi di rete separato.

Docker implementa CNM nella libreria `libnetwork`, fornendo vari driver per la gestione della rete dei container e la possibilità di usarli insieme a plugin esterni[18].

Host Se si usa questo tipo di rete per un container, il suo stack di rete non viene isolato dall'host di Docker (condividono lo spazio dei nomi di rete) e il container non riceve il proprio indirizzo IP. Ad esempio, se un container offre un servizio sulla porta 80 e viene configurato per usare la rete host, sarà possibile accedere all'applicazione del container connettendosi alla porta 80 all'indirizzo IP dell'host.
Questo tipo di rete può essere utile per ottimizzare le prestazioni o nelle situazioni in cui il container deve utilizzare molte porte, ovvero non è necessario il NAT.
È possibile utilizzarla solo sugli host Linux [23].

Macvlan Alcune applicazioni, come quelle di monitoraggio del traffico o quelle obsolete, hanno la necessità di essere connesse alla rete fisica. In queste situazioni si può usare il driver macvlan per assegnare un indirizzo MAC ad ogni interfaccia virtuale di rete del container, facendola così apparire come una interfaccia fisica connessa direttamente alla rete reale. In questo caso, si deve designare un'interfaccia fisica dell'host di Docker, la sotto-rete e il gateway per l'uso di macvlan. È anche possibile isolare diverse reti macvlan usando più interfacce fisiche[24].
Usando questo driver si devono tenere a mente alcuni concetti[24]:

- È molto semplice danneggiare non intenzionalmente la rete a causa dell'esaurimento di indirizzi IP o dell'espansione delle VLAN (VLAN spread, è la situazione in cui si hanno un numero spropositato di indirizzi MAC univoci all'interno di una sola rete).

- I dispositivi di rete fisici devono poter gestire la modalità "promiscua", nella quale ad una interfaccia fisica possono essere assegnati più indirizzi MAC.
- Se l'applicazione può funzionare utilizzando i driver bridge o overlay, forse sono soluzioni migliori nel lungo periodo.

None Questa configurazione di rete disabilita le funzionalità di rete di un container. Infatti non viene creata l'interfaccia (virtuale) responsabile del collegamento via cavo[14].

Bridge È il driver usato di default da Docker se non è specificato altro. Ha due configurazioni: default e user-defined. La prima è dichiarata obsoleta perciò non è raccomandata per l'uso in produzione. Inoltre differiscono nelle funzionalità fornite (vedi tabella 4)[22].

	Default	User-defined
DNS	I container possono comunicare tra loro solo attraverso indirizzi IP a meno di usare l'opzione obsoleta `--link`	I container possono risolvere i nomi e gli alias degli altri container
Isolamento	Tutti i container senza un driver di rete specificato sono inseriti nel bridge di default e possono comunicare tra loro	Solo i container che fanno parte di una determinata rete possono comunicare tra loro
Connessione in tempo reale	Per modificare le impostazioni di rete, si deve fermare il container e ricrearlo con la nuova configurazione	Si possono connettere e disconnettere i container dalla rete user-defined in tempo reale
Granularità della configurazione	Tutti i container della rete usano la stessa configurazione. Inoltre, la configurazione avviene fuori da Docker, quindi si deve riavviare Docker per applicare le modifiche	Si possono configurare le reti in modo diverso e separatamente quando si creano
Condivisione d'ambiente	I container collegati condividono le stesse variabili d'ambiente	Non è possibile condividere variabili d'ambiente in questo modo tra container

Tabella 4: Differenze di funzionalità tra una rete bridge di default e una user-defined

Fonte: https://docs.docker.com/network/bridge/#differences-between-user-defined-bridges-and-the-default-bridge

Con il driver bridge è possibile esporre una porta di un container all'esterno,

specificando l'indirizzo IP o la porta da associare al container[79].
È possibile indicare quali porte esporre nel Dockerfile aggiungendo la linea `EXPOSE <numero_porta>` (se si vuole esporre la porta 80, il comando sarà `EXPOSE 80`) e apponendo al comando `docker run` l'opzione `-P` o `--publish-all=true`. Se si aggiunge a `docker run` la stringa `--expose <numero_porta>`, non è necessario modificare il Dockerfile. Le porte esposte in questi modi sono associate ad una porta dell'host casuale all'interno dell'intervallo di porte effimere (di default comprende le porte dalla 32768 alla 61000, ma è possibile modificarlo)[79].
Se si vuole specificare la porta dell'host da associare , è possibile indicarla apponendo l'opzione `-p <porta_host:porta_container>` (equivalentemente `--publish=<porta_host:porta_container>`): se si vuole associare la porta 8080 del container alla porta 80 dell'host, il comando da eseguire è `docker run -p 80:8080 nginx` o `docker run --publish=80:8080 nginx`[79].
Normalmente Docker espone le porte all'indirizzo `0.0.0.0` (che corrisponde a qualsiasi indirizzo IP del sistema); per modificarlo è necessario inserire l'indirizzo da vincolare prima della porta dell'host: prendendo come indirizzo d'esempio `192.168.1.10`, il comando è `docker run 192.168.1.10:80:8080 nginx`[79].

Overlay Il driver overlay consente di creare una rete distribuita su più host Docker. Appoggiandosi alle reti specifiche degli host, permette ai container che sono collegati di comunicare in maniera sicura, quando la criptazione è abilitata. Pertanto questo driver è usato spesso per configurare una rete di uno swarm Docker, anche se presenta alcune operazioni utili ai singoli container.
Docker gestisce in modo trasparente l'instradamento di ogni pacchetto tra i corretti host e i corretti container[25].
Quando un host Docker crea o si unisce ad uno swarm, vengono create due reti sull'host[25]:

1. una rete overlay chiamata `ingress` che gestisce i piani di controllo e dati dei servizi dello swarm. Quando un servizio dello swarm viene creato e non viene connesso ad una rete overlay definita dall'utente, viene connesso alla rete `ingress` di default.

2. una rete bridge chiamata `docker_gwbridge` che connette il demone Docker singolo agli altri demoni che compongono lo swarm.

4.0.3 Gestione dei dati

Union file system Un union file system è un servizio per Linux, FreeBSD e NetBSD che permette di creare file system (FS) virtuali. Può aggregare file e cartelle di file system diversi (chiamati "rami") sovrapponendoli e formando un singolo FS. Quando i rami vengono "montati" si specifica quello che ha la priorità sugli altri, così, nel caso di file con lo stesso nome, vengono mostrati gli elementi prioritari. I diversi rami possono essere FS in sola lettura o in lettura/scrittura, in questo modo le modifiche al FS virtuale sono direzionate ad un file system reale specifico. Questo permette di far apparire scrivibile un

FS in sola lettura, senza però modificarlo, utilizzando il meccanismo di copy-on-write[137] .

Normalmente, tutti i file creati all'interno di un container sono salvati in un layer. Ciò significa che:

- I dati non persistono quando il container viene eliminato, inoltre può essere difficile estrarre i dati nel caso un altro processo li richieda.
- Il layer è collegato strettamente alla macchina host sulla quale il container sta eseguendo. Non è possibile spostare i dati facilmente in un altro host.
- La scrittura nel layer del container richiede l'uso dei driver per lo spazio di archiviazione. Il driver offre un "union file system" usando le funzionalità del kernel Linux. Questa astrazione riduce le prestazioni in confronto alla scrittura diretta su un volume di dati nel file system dell'host.

Docker ha due opzioni per salvare permanentemente i file dei container nella macchina host, cosicché i file persistano anche dopo la terminazione del container stesso: i volumi e le "bind mount". Se il sistema operativo dell'host è derivante da Linux, sono disponibili anche i "tmpfs" ("temporary file system", FS temporanei)[17].

- Volumi: i volumi sono salvati in una parte del file system gestita da Docker (generalmente al percorso `/var/lib/docker/volumes/`). I processi non appartenenti a Docker non dovrebbero modificare questa parte del file system. Possono essere gestiti attraverso comandi nel terminale. I volumi possono essere creati esplicitamente con il comando `docker volume create` oppure vengono creati autonomamente da Docker quando il container o il servizio che lo usa è creato. Nel caso non venga specificato un nome per il volume, alla creazione Docker gliene assegna uno casuale assicurandosi che sia univoco all'interno dell'host. Un volume può essere montato da più container, inoltre si possono installare dei driver per i volumi in modo da poter montare i volumi di un host remoto o di un cloud provider; infine l'uso di un volume non aumenta la dimensione del container che lo usa, perciò è conveniente usarli al posto di salvare i dati nel layer del container[27].
- Bind mount: Una bind mount monta un file o una cartella del file system dell'host nel container, per questo il container può accedere a file sensibili e di sistema e questo potrebbe intaccare il funzionamento del sistema host. Il file (o la cartella) se non esiste al momento della creazione del container, viene creato quando è richiesto dall'applicazione. Le bind mount sono molto performanti, ma presuppongono che la macchina host disponga di una specifica struttura di cartelle. È possibile condividere una bind mount tra più container, ma non possono essere usati comandi per gestire direttamente le bind mount[21].

- tmpfs: nei sistemi Linux, è possibile creare un file system temporaneo e montarlo all'interno di un container. Il FS risiede nella memoria del sistema host e i file creati al suo interno non persistono nel layer del container. Quando il container viene terminato, il file system temporaneo viene eliminato. I tmpfs sono utili per salvare informazioni che non devono persistere sull'host né nel layer del container. Non è possibile condividere i tmpfs tra più container[26].

5 Computer clusters

Un gruppo di computer che operano congiuntamente e possono essere visti come un unico dispositivo viene definito "cluster"[97].
Nella maggior parte dei casi i computer sono collegati attraverso la rete locale (LAN), sebbene non sia sempre il caso in quanto attraverso un servizio PaaS si potrebbero richiedere alcune macchine da raggruppare in un cluster, ma non essendo gestite dal cliente potrebbero essere dislocate in zone geografiche lontane. L'uso dei cluster sta aumentando, infatti è una soluzione economica rispetto ad un server unico ed è possibile anche avere prestazioni maggiori rispetto alla piattaforma monolitica[3].
Un esempio è il super computer IBM Sequoia, composto da più di novantottomila nodi (computer formanti un cluster), ciascuno avente un processore con 16 core e 16GB di RAM, per un totale, rispettivamente, di quasi un milione e seicento mila core e 1,6 petabyte di memoria volatile (1 petabyte = 1 milione di gigabyte), che forniscono all'agglomerato un posto tra i primi 500 super computer più potenti al mondo[129].
È possibile configurare un cluster per raggiungere un certo obiettivo, di seguito sono riportati i principali [90][83]:

- Bilanciamento del carico: un cluster può essere configurato in modo tale che le richieste siano distribuite ai nodi secondo un algoritmo scelto dall'amministratore. Normalmente viene selezionato il nodo più vicino al richiedente oppure quello con meno carico di lavoro. Altrimenti ci sono casi in cui si utilizza un metodo round-robin (quindi le richieste sono distribuite equamente tra tutti i nodi in maniera circolare).

- Calcolo intensivo: alcuni cluster sono usati per eseguire simulazioni di eventi complessi, come ad esempio un incidente automobilistico, che richiedono una potenza di calcolo enorme. Un esempio è il sopracitato supercomputer IBM Sequoia.

- Alta disponibilità: per alcuni servizi è necessario che non ci siano momenti di non disponibilità, in quanto comporterebbero delle grosse perdite all'azienda che li subisce[2]. Perciò, per poter rispondere alle richieste in modo continuativo, si deve configurare un cluster che possa sostituire un nodo malfunzionante mentre è attivo senza interruzioni.

5.1 Condivisione di dati

5.1.1 Memoria condivisa

Il primo metodo per condividere dati tra processi è l'uso della memoria condivisa: più CPU sono collegate al bus dati, il quale è connesso a sua volta alla memoria e ai dispositivi di input/output. Questo tipo di condivisione è molto efficiente in quanto i componenti sono collegati direttamente tra loro e la velocità della RAM è elevata, ma non è possibile utilizzarlo nei cluster di computer in quanto si tratta di nodi separati e non è possibile connetterli allo stesso bus.

5.1.2 Clustered file system

Un "clustered file system" (CFS) è un file system condiviso e montato simultaneamente da più server. Spesso, in un cluster, ogni nodo ha il proprio spazio di archiviazione privato al posto di avere un file system condiviso. I CFS possono offrire funzionalità aggiuntive, tra cui l'indirizzamento indipendente dalla posizione e la ridondanza dei dati, le quali migliorano l'affidabilità o riducono la complessità di altre parti del cluster. I "file system paralleli" sono una categoria particolare di CFS che sparge i dati su più nodi di archiviazione, sia per migliori prestazioni che per ridondanza dei dati[94].

L'utilizzo di un CFS può creare un punto singolo di fallimento che, in caso di malfunzionamento, può risultare in perdite di dati o non disponibilità di essi. Per rimediare, sono consigliati la replicazione dei dati in dispositivi diversi e l'uso di file system tolleranti agli errori.
I clustered file system hanno costi temporali più alti rispetto ad un FS locale, in quanto, oltre al tempo di accesso al disco e quello di processo della CPU, sono significanti i tempi di consegna della richiesta al server, i tempi di consegna della risposta al client e, per entrambe le parti, il tempo computazionale usato per il protocollo di comunicazione.
Il controllo della concorrenza è un problema importante, infatti potrebbero essere scritte modifiche in parti sovrapposte di un file. Normalmente è fornito dal file system o da un protocollo aggiuntivo installabile[94].

File system a disco condiviso Un file system a disco condiviso utilizza un'area di archiviazione in rete per permettere a computer diversi di ottenere l'accesso ai dischi al livello dei blocchi di byte. Il controllo dell'accesso e la traduzione dal livello dei file che usa l'applicazione al livello dei blocchi devono essere eseguiti nel nodo cliente.
Sono presenti meccanismi di controllo della concorrenza, quindi protezione dalle "data race", cioè le situazioni in cui due thread devono scrivere nello stesso indirizzo di memoria senza essere controllate da sistemi appositi: in tale situazione, il risultato è imprevedibile in quanto non si sa in che ordine verranno eseguite le operazioni dallo scheduler di sistema; come conseguenza, questo tipo di protezione fornisce una visione del file system consistente e trasponibile su disco, evitando la corruzione e la perdita involontaria dei dati quando più client tentano di accedere allo stesso file contemporaneamente.

Questo tipo di file system impiega un meccanismo di isolamento per prevenire la corruzione dei dati nel caso del fallimento di un nodo, in quanto un dispositivo non isolato potrebbe causare l'alterazione dell'integrità dei dati se il cliente perde la connessione al file system mentre sta cercando di accedere ad informazioni che sono richieste anche da altri nodi[131].

File system distribuito Un file system distribuito non fornisce l'accesso al livello dei blocchi, ma utilizza un protocollo di rete. Per questo sono anche conosciuti come "network file system" (file system di rete), anche se non sono gli unici che utilizzano la connessione in rete per inviare dati.
In base a come è progettato il protocollo, è possibile impedire l'accesso al FS in base ad una lista di dispositivi o utenti oppure alle capacità sia dei server che dei client.
Un file system distribuito permette, al contrario degli archivi di dati distribuiti, di accedere ai file con le stesse interfacce e la stessa semantica dei file locali.
I file system distribuiti ambiscono alla trasparenza in molti aspetti: tentano di essere invisibili ai programmi dei client, i quali dovrebbero percepire un sistema simile ad un FS locale. Gli aspetti che le implementazioni di un FS distribuito dovrebbero tenere in considerazione sono[102]:

- Trasparenza d'accesso: i client sono completamente ignari della distribuzione dei file e possono accedervi come in un file system locale.
- Trasparenza di località: il nome di un file non fornisce informazioni riguardo la sua posizione.
- Trasparenza di concorrenza: tutti i client hanno la stessa visione dello stato del file system. Se un processo sta modificando un file, qualsiasi altro processo, locale o remoto, che accedono al file vedranno le modifiche in maniera coerente.
- Trasparenza dei fallimenti: i client e i loro programmi dovrebbero funzionare correttamente dopo un malfunzionamento di un server.
- Eterogeneità: il servizio dovrebbe essere fornito a diversi hardware e diversi sistemi operativi.
- Adattabilità: il file system deve funzionare correttamente sia in ambienti piccoli (da 1 client a una decina) che in ambienti più grandi (da centinaia a decine di migliaia di client).
- Trasparenza di replicazione: i client dovrebbero essere inconsapevoli della replicazione dei file eseguita su più server per supportare l'adattabilità.
- Trasparenza di migrazione: i file possono essere trasferiti tra server differenti senza che i client ne vengano a conoscenza.

Archiviazione connessa alla rete Gli spazi di archiviazione connessi alla rete (Network-attached storage, NAS) forniscono sia spazio di archiviazione che un file system, come un file system a dischi condivisi applicato ad un'area di archiviazione di rete (Storage Area Network, SAN). Normalmente i NAS utilizzano dei protocolli basati sui file, al contrario di SAN che usa i protocolli basati sui blocchi[122].

5.2 Comunicazione

Per far comunicare dei computer tra loro, sono stati usati principalmente due approcci: Parallel Virtual Machine (PVM) e Message Passing Interface (MPI).

5.2.1 Parallel Virtual Machine

PVM è un software che permette di utilizzare un insieme di sistemi eterogenei come una risorsa computazionale flessibile e concorrente (una "macchina virtuale parallela"). I computer connessi possono essere di qualsiasi tipologia ed essere connessi attraverso reti di vario genere.

PVM consta di librerie ed eseguibili per lo scambio dei messaggi, gestione delle risorse e dei compiti e notifiche dei fallimenti. Permette di parallelizzare manualmente un programma esistente o scrivere nuovi applicativi distribuiti.

Il software deve essere installato specificamente su ciascuna macchina che deve far parte della macchina virtuale parallela[125].

5.2.2 Message Passing Interface

MPI è uno protocollo standardizzato "de facto" per la programmazione di computer paralleli. Gli obiettivi principali sono l'alta efficienza, l'adattabilità e la portabilità (cioè la possibilità di essere utilizzato in sistemi con configurazioni diverse senza essere modificato).

Esistono tre versioni principali di MPI; la prima non ha il concetto di memoria condivisa, la seconda ha una definizione limitata di memoria condivisa distribuita, mentre la terza introduce esplicitamente la condivisione della memorie. Nonostante ciò, viene incoraggiato l'uso della memoria locale rispetto a quella condivisa.

MPI gode di portabilità e di ottimizzazione per la maggior parte delle architetture utilizzate, oltre ad essere disponibile per i linguaggi di programmazione più diffusi (C/C++, Java, Python, C# e Fortran).

L'interfaccia di MPI intende fornire le funzionalità essenziali per la topologia virtuale, la sincronizzazione e la comunicazione tra un insieme di processi, infatti un programma MPI lavora con i processi. Tendenzialmente, per avere le prestazioni massime, a ciascuna CPU (o "core" nel caso la macchina abbia una CPU con più "core") verrà assegnato un solo processo. L'assegnamento è eseguito durante l'esecuzione del programma[118].

6 Architetture a microservizi

La comparsa degli strumenti per gestire i container, come Docker, ha indotto i fornitori di servizi a migrare dai server monolitici ad architetture a microservizi e, di conseguenza, adottare i container su larga scala, sia per motivi economici che per la semplicità di replicabilità degli stesso.

Il dover configurare e dispiegare una moltitudine di container per rispondere alle richieste in arrivo ha espresso la necessità di avere dei software che semplificassero questo passaggio, automatizzando la configurazione e coordinando il dispiegamento e l'esecuzione dei container e delle repliche all'interno di più host, come se fossero processi all'interno di un cluster. Questi software prendono il nome di "orchestratori", i più diffusi sono Docker Swarm e Kubernetes.

6.1 Service mesh

Una service mesh (tradotto in "rete dei servizi") è uno strato di infrastruttura dedicata per facilitare la comunicazione tra i microservizi, spesso usando un server proxy (un server intermediario)[130], definito "sidecar", che gestisce tutte le mansioni che non sono legate direttamente ai servizi individuali, come le comunicazioni inter-servizi, il monitoraggio e le questioni di sicurezza. Alcuni compiti di questa rete sono la limitazione delle richieste in entrata, il controllo degli accessi e l'autenticazione end-to-end[39].

L'architettura delle service mesh ricorda una rete SDN, infatti è divisa in piano di controllo e piano dei dati (vedi figura 11).
Nel primo sono contenuti tutti gli strumenti per amministrare la rete, per definire le regole e le politiche sulla modalità di implementazione di determinate funzionalità, per il monitoraggio e per rinforzare la sicurezza[65].
Il piano dati tipicamente consiste in un proxy affiancato ad ogni replica dell'applicativo, configurato per controllare tutto il traffico in entrata e in uscita dal contenitore[65] e collegato agli altri proxy della rete.

6.2 Docker swarm

Docker Swarm è lo strumento di gestione ed orchestrazione di cluster di Docker. È specifico per la gestione dei container Docker. Esso permette di connettere più nodi allo "swarm" (il cluster di host che eseguono il motore Docker)[19].

6.2.1 Nodi

I nodi sono le istanze del motore Docker che partecipano al cluster, perciò è possibile eseguire più processi del motore stesso in un singolo computer fisico, anche se gli swarm di produzione generalmente hanno i nodi distribuiti in più dispositivi.
I nodi devono avere almeno un ruolo: "manager", "lavoratore" (worker) o entrambi. I manager gestiscono lo stato del cluster ed eleggono un leader per la gestione dei "compiti" (task), il quale li invia ai lavoratori che devono svolgerli.

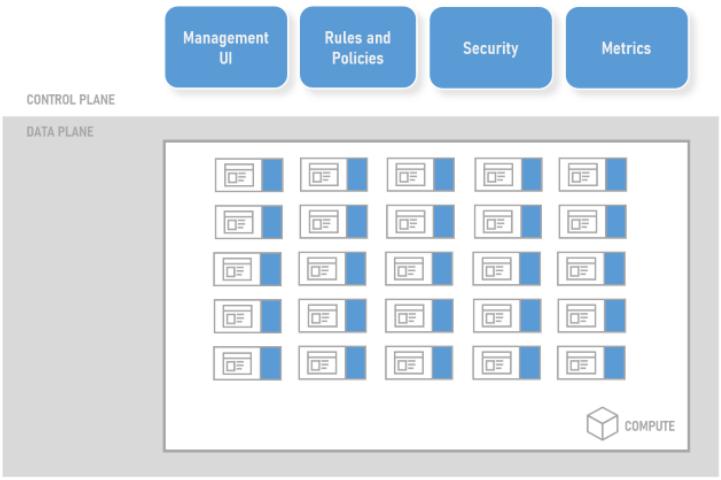

Figura 11: Architettura di una service mesh ordinaria
Fonte: https://docs.microsoft.com/en-us/azure/aks/servicemesh-about

Un "agente" (agent) viene eseguito su ogni nodo lavoratore ed ha il compito di riportare ogni task correntemente assegnati ad esso, in questo modo il nodo manager può conoscere lo stato dei compiti affidati e mantenere lo stato desiderato del cluster[20].

6.2.2 Servizi e compiti

Un "servizio" (service) è la definizione dei compiti da eseguire sui nodi, sia manager che lavoratori. Quando un servizio è creato, deve essere specificato quale immagine di container utilizzare e, eventualmente, quali comandi eseguire all'interno dei container in esecuzione.
I servizi possono essere "replicati", per cui i manager dello swarm distribuiscono un numero specifico di repliche del compito tra i lavoratori in base al numero configurato nello stato desiderato, oppure "globale", per i quali è eseguita un task in ogni nodo disponibile del cluster.
Un compito descrive il container Docker e i comandi da eseguire al suo interno. È l'elemento più piccolo che è possibile programmare. Quando un compito è assegnato ad un nodo, non può essere spostato su altri nodi; perciò può solo eseguirlo o fallire nel farlo.
I nodi manager cercano sempre di rimanere allo stato desiderato, quindi se sono state richieste 10 repliche per un servizio e 2 falliscono durante l'esecuzione, altri 2 task per lo stesso servizio saranno programmate ed eseguiti all'interno di alcuni nodi disponibili.

6.2.3 Bilanciamento di carico

Per pubblicare un servizio è possibile assegnargli una porta da esporre; può essere specificata dall'amministratore o lasciata scegliere dal manager tra quelle dell'intervallo 30000-32767.

Dall'esterno è possibile accedere al servizio connettendosi a qualsiasi nodo, anche a quelli che non ospitano task riguardanti il servizio stesso, in quanto il servizio di bilanciamento di carico intercetta la richiesta e la ridireziona automaticamente ad un nodo che sta eseguendo il compito rilevante.

Ad ogni servizio è anche assegnato un nome nel server DNS interno allo swarm ed è usato per la funzione di bilanciamento del carico.

6.3 Kubernetes

Kubernetes è uno strumento estendibile per gestire container e servizi in maniera automatizzata. Fornisce una struttura per amministrare resilientemente sistemi distribuiti e si occupa di adattare e rimediare agli errori dell'applicazione.

Tra le funzionalità offerte da Kubernetes ci sono:

- Scoperta dei servizi e bilanciamento di carico: Kubernetes può esporre un container attraverso il nome nel DNS o il suo indirizzo IP. Inoltre, se il traffico verso un container è elevato, può distribuirlo per rendere il dispiegamento stabile.

- Gestione dell'archiviazione: Kubernetes permette di montare automaticamente un sistema di archiviazione a scelta, che sia locale o fornito da un cloud provider.

- Avvio e ritiro di servizi automatizzati: potendo descrivere lo stato desiderato del cluster, si può modificare lo stato in maniera controllata. Si può automatizzare Kubernetes per creare nuovi container, rimuovere quelli esistenti e modificare chi ha accesso a determinate risorse.

- Ottimizzazione delle risorse: conoscendo le specifiche dei nodi del cluster, quando si richiede a Kubernetes il dispiegamento di nuovi container, esso provvederà a inserirli nei nodi più adatti.

- Correzione di errori: Kubernetes riavvia i container che falliscono, li sostituisce, li ferma in caso non rispondano ai controlli imposti dall'utente e non li espone fino a quando sono pronti per servire le richieste.

- Amministrazione dei segreti e delle configurazioni: è possibile salvare informazioni sensibili, come password e chiavi crittografiche. Inoltre si possono aggiornare i segreti e le configurazioni delle applicazioni senza dover ricreare l'immagine del container.

Kubernetes utilizza un gestore di container e supporta la maggior parte delle implementazioni, di default però è configurato per servirsi di Docker.

Inoltre, Kubernetes permette di creare multipli cluster virtuali in un unico cluster fisico, chiamati "namespace". I namespace forniscono uno spazio per i nomi delle risorse, i quali devono essere univoci all'interno di un namespace. Non è possibile inserire un namespace dentro l'altro e ogni risorsa di Kubernetes deve essere all'interno di un solo namespace.

6.3.1 Risorse

Kubernetes definisce un insieme di risorse per definire la struttura di un cluster. È possibile aggiungere la definizione di risorse personalizzate (Custom Resource Definition, CRD) creando estensioni per Kubernetes. Di seguito sono descritte le risorse principali.

Pod Un pod è composto da un gruppo di uno o più container, che condividono le risorse di archiviazione e di rete, e una specifica sull'esecuzione dei containers. Un pod modella un host logico specifico per un'applicazione, comprendente tutti i container strettamente legati. Un pod è isolato allo stesso modo di un container Docker, inoltre ai singoli container possono essere applicate ulteriori restrizioni. All'interno di un pod, i container possono comunicare tra loro utilizzando l'indirizzo `localhost`[59].

Deployment Un deployment specifica un insieme di repliche di pod in modo dichiarativo. Si descrive lo stato del cluster desiderato e il controllore del dispiegamento modifica lo stato attuale per rispecchiare ciò che ha definito l'utente: ciò può comportare la creazione di nuove repliche di pod, l'eliminazione di altre e la modifica all'accesso alle risorse per una serie di repliche[44].

Service Un servizio permette di esporre un'applicazione del cluster, sia verso l'esterno che verso l'interno, come servizio in rete. I servizi permettono di astrarre i pod, infatti essi sono effimeri, in quanto un deployment può crearli e distruggerli dinamicamente. Per non far gravare il compito di tracciamento degli indirizzi IP dei pod, che potrebbero cambiare, da coloro che ne dipendono, Kubernetes ha sviluppato i servizi: coloro che necessitano delle funzionalità di un servizio, si interfacciano solo con esso ed è Kubernetes che gestisce associazione dei pod al servizio in modo trasparente. È possibile definire un selettore che comunica i parametri secondo cui Kubernetes deve associare un pod ad un dato servizio [60].

Endpoint Nel caso un servizio non specifichi un selettore, è necessario definire almeno un Endpoint da associare al servizio stesso. È un indirizzo comprendente una porta su cui dirigere il traffico in arrivo; normalmente è Kubernetes che li gestisce autonomamente quando è specificato il selettore del servizio, altrimenti è l'amministratore del cluster che deve indicarli[61].

Volumi e PersistentVolume I file presenti sul disco di un container sono volatili, infatti, quando un container è arrestato o riavviato, i file sono persi; inoltre in un pod è spesso utile scambiare dei file tra i container contenuti in esso. Kubernetes mette a disposizione i "volumi", uno spazio di archiviazione condiviso all'interno del pod che è strettamente legato alla vita di quest'ultimo: tant'è che quando il pod viene distrutto, il volume smette di esistere[62].
Per avere dispositivi di archiviazione virtuali non legati alla vita di un pod, e quindi ottenere la persistenza dei dati, è stato sviluppato il concetto di `PersistentVolume` (PV), da definire separatamente dal o dai pod che lo usano. Un PV astrae un disco locale oppure uno spazio di archiviazione in rete, fornito dall'amministratore del cluster o da un cloud provider[57].

PersistentVolumeClaim Per indicare l'uso di un PV, un pod deve definire una `PersistentVolumeClaim` (PVC) all'interno della propria descrizione. In base alla classe di archiviazione richiesta, è possibile che venga assegnato un PV definito in precedenza o un volume creato dinamicamente, tenendo conto delle caratteristiche imposte dalla PVC, ad esempio quantità di spazio e, se prevista dalla classe di archiviazione, velocità di trasferimento dati o altre[58].

6.3.2 Componenti del piano di controllo

Il piano di controllo amministra i pod e i nodi lavoratori del cluster: per poter portare a termine questo compito, ogni cluster contiene obbligatoriamente alcuni componenti. Questi possono essere eseguiti su qualsiasi nodo e non obbligatoriamente tutti sullo stesso, ma normalmente sono dispiegati tutti sulla stessa macchina, sulla quale non saranno programmati container di applicazioni, chiamata "master" ("padrone")[43].

kube-apiserver Espone l'API (Application Programming Interface, è l'interfaccia che definisce come comunicare con un determinato software) di Kubernetes ed è l'implementazione principale del server API di Kubernetes. È progettato per adattarsi orizzontalmente, cioè si adatta dispiegando altre istanze del software, e il traffico può essere bilanciato tra esse[50].

etcd Database per memorizzare delle coppie chiave-valore che gode delle proprietà di coerenza e alta disponibilità[46]. Permette di ottenere i dati attraverso l'uso di strumenti HTTP e le informazioni sono organizzate gerarchicamente in cartelle, come in un file system standard[28].

kube-scheduler Osserva i pod appena definiti senza un nodo assegnato e ne seleziona uno su cui eseguirli. Utilizza alcun parametri per la scelta, tra cui i requisiti di risorse, la vicinanza ai dati e il carico di lavoro già presente[53].

kube-controller-manager Componente che amministra i processi dei controllori. Logicamente, ogni controllore è un processo separato, ma per ridurre

la complessità sono compilati nello stesso eseguibile ed avviati come processo singolo. I controllori includono: il controllore dei nodi, che osserva e notifica i nodi che si disconnettono; il controllore delle repliche, che mantiene il numero corretto di repliche di pod per ciascun oggetto che le definisce; il controllore degli endpoint, che associa pod e servizi creando degli endpoint dinamicamente[51].

cloud-controller-manager Integra la logica di controllo specifica per il cloud. Permette di collegare il cluster con l'API del cloud provider e divide i componenti che interagiscono con la piattaforma cloud da quelli che invece interagiscono solo con il cluster. Questo componente è presente solo se il cluster è ospitato in una piattaforma nel cloud, se è su macchine locali non è eseguito. Può essere replicato per migliorare le prestazioni o per tollerare i fallimenti. I controllori che possono dipendere da questo componente sono: il controllore dei nodi, che interroga il cloud provider per determinare se un nodo è stato eliminato dopo che ha smesso di rispondere; il controllore dell'instradamento, che configura i percorsi nell'infrastruttura sottostante; il controllore dei servizi, che crea, aggiorna ed elimina i bilanciatori di carico[41].

6.3.3 Componenti dei nodi

Questi processi sono dispiegati in qualsiasi nodo del cluster per amministrare i pod e fornire l'ambiente di esecuzione di Kubernetes.

kubelet È un agente (programma che esegue ed agisce per conto di un utente o di un altro processo) e si assicura che i container eseguano all'interno dei pod. Partendo dalle specifiche di un pod, controlla che i container descritti siano stati avviati e non abbiano incontrato errori. Non gestisce i container che non sono stati creati da Kubernetes[54].

kube-proxy Questo proxy mantiene le regole per la rete del nodo. Permettono comunicazioni verso i pod, sia dall'interno che dall'esterno del cluster. Se disponibile, kube-proxy utilizza il livello di filtraggio dei pacchetti del sistema operativo, altrimenti li inoltra lui stesso[52].

Gestore dei container È il software responsabile di eseguire i container. Kubernetes supporta vari gestori, tra cui Docker, containerd e CRI-O, oltre a tutti quelli che implementano l'interfaccia definita da Kubernetes[42].

6.3.4 Estensioni

Sono disponibili estensioni per un cluster Kubernetes che forniscono funzionalità, ad esempio, per la rete e le politiche di rete, la scoperta dei servizi e gestione del DNS, monitoraggio delle risorse del cluster, amministrazione dell'infrastruttura e virtualizzazione[49].

6.3.5 Operatori

Gli operatori sono un modello di estensione per Kubernetes. Questo modello punta a replicare il comportamento di un operatore umano che gestisce un servizio o un insieme di servizi per poter automatizzare le fasi che si ripetono. Gli operatori, umani e non, devono perciò sapere come il sistema si deve comportare, come dispiegarlo e come reagire in caso di errore. In un cluster Kubernetes, fanno uso delle CRD per amministrare le applicazione ed i loro componenti e ne diventano i controllori.
I compiti automatizzabili con un operatore includono:

- dispiegare un'applicazione su richiesta;
- eseguire e ripristinare i backup dell'applicazione;
- gestire gli aggiornamenti sia dell'applicazione stessa che di altre risorse, come gli schemi dei database o parametri di configurazione aggiuntivi;
- esporre un servizio alle applicazioni che non supportano la funzionalità di scoperta dell'API Kubernetes;
- simulare un fallimento per testare la resilienza del cluster;
- eleggere un leader per le applicazioni distribuite senza la necessità di avere elezioni interne.

Gli operatori sono eseguiti al di fuori del piano di controllo, come se fossero applicazioni per gli utenti[56].

6.3.6 Gestione della rete in un cluster

Ogniqualvolta che un servizio viene creato, gli viene assegnato un nome DNS. Di default, la lista di ricerca DNS di ciascun pod comprende il proprio namespace e il dominio del cluster[45].

Container Network Interface Kubernetes adotta il modello CNI (Container Network Interface) per gestire la connettività tra container. CNI è un'interfaccia che tutte le estensioni di rete implementano per completare la configurazione di rete di un container e si posiziona tra il gestore dei container (come ad esempio Docker) e l'estensione stessa. L'interfaccia consta di soli due comandi: uno per aggiungere un container alla rete ed uno per rimuoverlo da essa. L'estensione è distribuita come eseguibile e richiede di specificare una configurazione quando invocata. Può anche richiamare un'altra estensione per eseguire i passi necessari, per esempio associare un indirizzo IP al container[29].

Ingress e controllore Ingress Un Ingress è un oggetto definito nell'API di Kubernetes che permette di esporre percorsi HTTP e HTTPS dall'esterno del cluster verso i servizi forniti. L'instradamento del traffico è controllato da regole definite nell'Ingress. Può essere configurato per offrire un indirizzo internet

accessibile dall'esterno ai servizi, per bilanciare il traffico, portare a termine il i protocolli crittografici SSL/TLS e per fornire un hosting virtuale basato sui nomi (quindi più nomi DNS possono essere risolti in un unico indirizzo IP). Se serve esporre porte o protocolli differenti, non è possibile utilizzare questo meccanismo[47].
Per far funzionare correttamente le risorse Ingress, è necessaria la presenza di un controllore Ingress, il quale non è installato e avviato automaticamente all'avvio del cluster[48].

Politiche di rete Le politiche di rete di un cluster Kubernetes specificano come dei gruppi di pod possono comunicare tra loro e con altri host di rete. Le politiche utilizzano etichette per selezionare i pod a cui applicarsi e definiscono le regole per specificare quale traffico è permesso ai pod selezionati. Di default, i pod non sono isolati ed accettano il traffico da qualsiasi sorgente, ma lo diventano quando sono selezionati da una politica. Infatti, il pod selezionato rifiuterà qualsiasi connessione non permessa dalla politica. Le politiche non vanno in conflitto, sono additive, perciò il pod coinvolto sarà limitato dall'unione delle regole di ingresso ed uscita delle politiche che lo affliggono, senza essere influenzata dall'ordine con cui sono definite le politiche stesse[55].

6.3.7 Estensioni di rete

le politiche di rete non possono essere usate da una configurazione base di Kubernetes, ma necessitano dell'installazione delle estensioni di rete (network plugin) apposite. Tra le estensioni esistenti si citano "Flannel" e "Calico".

Flannel Quando utilizzato, dispiega un agente su ogni nodo del cluster ed è responsabile dell'allocazione di una sottorete su ciascun host. Flannel si concentra sulle modalità di comunicazione tra host e non sulla connessione tra container e host, pertanto non supporta le politiche di rete. Ciò permette a Flannel di essere un plugin semplice da configurare e da gestire[11].

Calico Calico fornisce un plugin di rete flessibile che permette di configurare le politiche di rete da adottare, sia supportando le risorse messe a disposizione da Kubernetes che offrendo una estensione delle politiche interoperabili con le precedenti. La configurazione di Calico si estende fino ai minimi particolari, sebbene questa profondità la renda più complessa, ma più potente, rispetto ad altri plugin[5].

6.4 Proxy Envoy

I proxy Envoy sono dei proxy che operano al livello 7 dello stack di rete ISO/OSI (livello applicativo) e che si basano su due principi: "La rete deve essere trasparente per le applicazioni" e "quando si riscontrano problemi di rete e applicativi si deve riuscire facilmente a risalire alla fonte del problema"[75].

Per poter raggiungere l'obiettivo prefissato, Envoy ha le seguenti caratteristiche[75]:

1. **Architettura a processo**: Envoy è un processo a se stante del sistema operativo da eseguire a fianco di ogni server applicativo. Tutti i proxy Envoy formano una rete di comunicazione trasparente attraverso la quale le applicazioni inviano e ricevono messaggi utilizzando *localhost* e non conoscono la topologia di rete. Rispetto all'uso di una libreria, questa architettura rende ininfluente il linguaggio dell'applicazione che il proxy affianca, perciò Envoy permette di collegare in modo trasparente applicazioni scritte in linguaggi diversi che necessiterebbero di particolari configurazioni per interagire (ad esempio un server scritto in C++ che comunica con dei client scritti in Java); inoltre il dispiegamento e l'aggiornamento dei proxy è reso veloce e non tange le applicazioni affiancate, in quanto consiste nel riavviare un processo, al contrario con una libreria potrebbe essere necessario ricompilare l'intero applicativo (operazione che può richiedere molto tempo).

2. **Filtro L3/L4 ISO/OSI**: internamente Envoy è un proxy che opera ai livelli di rete e di trasporto (abbreviati rispettivamente L3 e L4), perciò comprende un meccanismo di di filtri concatenati per eseguire alcuni compiti come proxy TCP/UDP. Alcuni filtri comprendono la gestione delle connessioni HTTP[74] e l'autenticazione attraverso l'uso del protocollo TLS[73].

3. **Filtro L7 HTTP**: Envoy supporta un ulteriore strato di filtraggio per il protocollo HTTP al livello 7 dello stack ISO/OSI (in breve L7), dato che è un componente critico nell'architettura di rete moderna. I filtri possono eseguire azioni come *buffering* (cioè salvare in memoria temporaneamente dei dati mentre si stanno inviando o sono trattati), *rate limiting* (limitazione artificiale della velocità di invio e ricezione dati[127]) e *routing/forwarding*.

4. **Supporto per HTTP/2**: nella modalità HTTP Envoy supporta entrambe le versioni in uso del protocollo, HTTP/1.1 e HTTP/2, e può operare come proxy tra tra client che usano versioni diverse. Questo permette di avere una qualsiasi combinazione di protocollo usato dal client e quello usato dal server, sebbene la configurazione consigliata preveda che tutti i servizi connessi alla mesh utilizzino HTTP/2.

5. **Routing L7 HTTP**: Envoy, quando usato come proxy HTTP, consente di utilizzare un sistema di routing che è capace di instradare e redirezionare richieste in base a parametri del pacchetto come, ad esempio, il percorso, il tipo del contenuto o opzioni di esecuzione di Envoy. Questa funzionalità è usata spesso quando il proxy Envoy fa parte del confine della rete, però non è raro utilizzarla in una service mesh.

6. **Supporto a gRPC**: gRPC è un framework RPC (Remote Procedure Call, permette di eseguire procedure su computer remoti) sviluppato da

Google che utilizza HTTP/2 come protocollo applicativo. Envoy implementa tutte le caratteristiche necessarie per il substrato di instradamento e bilanciamento di carico per le richieste e risposte di gRPC.

7. **Scoperta dei servizi e configurazione dinamica**: Envoy può consumare un insieme stratificato di API per la configurazione dinamica per una gestione centralizzata. Gli strati forniscono ad un proxy aggiornamenti dinamici riguardo: host appartenenti ad un cluster, i cluster stessi, l'instradamento HTTP, i socket di rete in ascolto e le informazioni crittografiche. Per un dispiegamento più semplice, la scoperta degli host può essere effettuata con la risoluzione dei nomi DNS (o eventualmente si può saltare questo passaggio) e gli strati successivi possono essere sostituiti con un file di configurazione statico.

8. **Controllo di attività**: per controllare attivamente lo stato di attività dei servizi, Envoy mette a disposizione un sistema che può effettuare controlli sullo stato di attività dei cluster di servizi. Con l'aggiunta delle informazioni di scoperta dei servizi, Envoy può determinare i bersagli del bilanciamento di carico che hanno performance adeguate. Un controllo passivo è effettuato dal sistema di rilevamento delle anomalie, il quale rimuove dall'insieme dei bersagli del bilanciamento di carico tutti gli host che hanno prestazioni inferiori rispetto agli altri, utilizzando parametri come ad esempio il numero di fallimenti consecutivi e la latenza.

9. **Bilanciamento di carico avanzato**: il bilanciamento di carico tra i componenti di un sistema distribuito è un problema complesso. Essendo un processo a se stante, il proxy Envoy può implementare tecniche di bilanciamento avanzate e renderle accessibili a tutte le applicazioni. Envoy supporta meccanismi di nuovi tentativi automatici (in caso di fallimento, avviene automaticamente un nuovo tentativo), circuit breaking (in caso di sovraccarico o errore, le comunicazioni vengono subito interrotte), rate limiting globale, replicazione di traffico (tra cluster) e rilevazione delle anomalie.

10. **Supporto come proxy di frontiera**: disponendo di meccanismi come algoritmi di bilanciamento di carico, scoperta dei servizi, supporto per la conclusione del protocollo TLS, supporto per HTTP/1.1 e HTTP/2, instradamento HTTP, i proxy Envoy sono adatti anche come proxy di confine di una rete.

11. **Osservabilità**: i problemi possono presentarsi sia a livello di rete che al livello applicativo. Per questo, Envoy include il supporto per i sistemi di statistiche e per il tracciamento distribuito, permettendo così all'amministratore di rete di visualizzare il punto di rottura della struttura.

6.5 Istio

Istio è un software che permette di connettere, mettere in sicurezza, controllare ed osservare servizi; pertanto è una service mesh che si appoggia trasparentemente sulle applicazioni distribuite esistenti[40]. Istio affianca ai pod designati dei server proxy, chiamati *sidecar proxy*, che estendono i proxy Envoy (figura 12). Grazie a questi proxy, Istio può fornire molte funzionalità per il piano di controllo, tra cui:

1. Bilanciamento di carico automatico per il traffico HTTP, gRPC, WebSocket e TCP.
2. Controllo del comportamento del traffico in profondità con molte regole di instradamento, nuovi tentativi, tolleranza agli errori e la simulazione di guasti.
3. Uno strato di politiche aggiuntivo e un'API configurabile che supportano il controllo degli accessi, la limitazione delle richieste e la definizione di tetti massimi per l'uso delle risorse.
4. Monitoraggio automatico, gestione dei log e tracciamento di tutto il traffico all'interno del cluster, compreso il traffico ingress (dall'esterno verso il cluster) ed egress (dal cluster verso l'esterno).
5. Comunicazione tra servizi sicura all'interno di un cluster con un'autenticazione robusta basata sull'identità e autorizzazioni.

6.5.1 Architettura

Essendo una service mesh, Istio è diviso nei piani di controllo e dei dati. Nel secondo, i sidecar proxy permettono ad Istio di estrarre dei segnali riguardanti il comportamento del traffico e utilizzarli per imporre le politiche decisionali o inviarli ai sistemi di monitoraggio. Nel piano di controllo è eseguito il demone `istiod` che gestisce la scoperta, la configurazione e i certificati crittografici dei servizi; inoltre, il demone converte le regole di instradamento di alto livello in configurazioni specifiche per i server Envoy, a cui sono propagate; infine, istiod abilita una robusta autenticazione tra servizi e utenti implementando una gestione delle identità e delle credenziali internamente, in questo modo gli operatori possono imporre delle politiche basate sull'identità dei servizi piuttosto che su identificatori a livello di rete o di trasporto relativamente instabili[35].

6.5.2 Gestione del traffico

Istio si affida ai proxy Envoy dispiegati a fianco dei servizi per gestirne il traffico rendendo non necessario modificare i servizi per modificare le regole di instradamento. Per poter controllare il traffico, deve sapere la posizione di tutti gli endpoint e a quali servizi appartengono; per popolare il proprio registro dei servizi, Istio si collega al sistema di scoperta dei servizi disponibile. Normalmente,

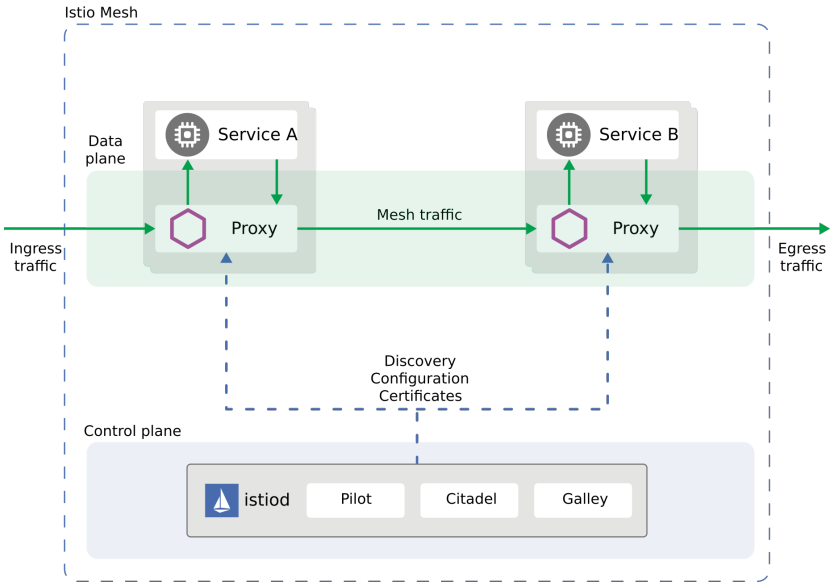

Figura 12: Rappresentazione semplificata dell'architettura di un cluster con Istio

Fonte: https://istio.io/latest/docs/concepts/what-is-istio/

i proxy Envoy distribuiscono il traffico, se ci sono più istanze che offrono un servizio, seguendo il modello round-robin. Quando necessario, è possibile modificare il bilanciamento del carico attraverso l'API apposita fornita da Istio, che fornisce alcune CRD per configurare la mesh.

Virtual Service I servizi virtuali (o virtual service) permettono di configurare le modalità di instradamento delle richieste verso un servizio all'interno della mesh. Ciascun servizio virtuale è composto da una serie di regole che sono valutate in ordine, facendo associare ad Istio una richiesta per un servizio virtuale ad una specifica destinazione reale nella mesh.

Destination Rule Le regole di destinazione, complementari ai servizi virtuali, indicano quali azioni intraprendere per le richieste verso una determinata destinazione reale, perciò sono applicate dopo la valutazione dell'instradamento dei servizi virtuali. Sono quindi usate per specificare un sottoinsieme di servizi identificato da un nome (ad esempio è possibile raggruppare le istanze di un servizio per versione). I sottoinsiemi definiti nelle regole di destinazione si possono utilizzare nei servizi virtuali come destinatari.

Gateway I gateway sono usati per amministrare il traffico in entrata ed in uscita dalla mesh, pertanto la configurazione è applicata ai server Envoy di

confine invece che ai proxy affiancati ai servizi. Generalmente sono usati per gestire il traffico in entrata, ma è possibile anche configurare i gateway di egress.

Service entry Con le service entry è possibile aggiungere voci al registro dei servizi di Istio per i servizi esterni al cluster. In questo modo, i proxy Envoy possono indirizzare il traffico ad essi come se facessero parte della mesh.

Sidecar Applicando una risorsa "sidecar", si configurano l'insieme di porte e protocolli accettati dai proxy Envoy e i servizi che si possono raggiungere.

6.5.3 Sicurezza

Per mettere in sicurezza i microservizi di una mesh, è necessario adottare alcune misure: la criptazione del traffico per difendersi dagli attacchi "Man in the middle"; la configurazione di TLS simmetricamente (quindi entrambi i comunicanti forniscono un certificato) e delle politiche di accesso specifiche per fornire un controllo agli accessi dei servizi flessibile; strumenti di verifica (come i log) per determinare quali azioni sono state intraprese e da chi. La sicurezza in Istio deve raggiungere i seguenti obiettivi:

- Sicurezza di default: non devono essere necessarie modifiche al codice e all'infrastruttura dell'applicazione.
- Difesa in profondità: si integra con i sistemi di sicurezza già esistenti per fornire più strati di protezione.
- Reti non fidate: può erigere soluzioni di sicurezza in reti non fidate.

I componenti della sicurezza di Istio sono una Autorità dei certificati (Certificate Authority, CA) per la gestione delle chiavi e dei certificati; le politiche di autorizzazione ed autenticazione e le informazioni dei nomi dei servizi; i proxy sidecar e di perimetro che sono i punti di imposizione delle politiche per mettere in sicurezza le comunicazioni tra gli utenti ed i server; un insieme di estensioni dei proxy Envoy per gestire la telemetria e i log.[38]

6.5.4 Osservabilità

Istio fornisce una telemetria dettagliata per tutte le comunicazioni dei servizi in una mesh. Questo permette agli operatori di investigare, mantenere e ottimizzare le proprie applicazioni. I tipi di telemetria generati da Istio sono: Metriche, basate sui quattro "segnali aurei" del monitoraggio (latenza, traffico, errori e saturazione) e disponibili in diversi livelli (proxy, servizi, piano di controllo); tracce distribuite, generate per ogni servizio, forniscono agli operatori una visione del flusso delle procedure e delle dipendenze all'interno della mesh; Log di accesso, che registrano ogni richiesta includendo anche informazioni riguardanti la sorgente e la destinazione[37].

6.5.5 Espandibilità

I proxy Envoy usati da Istio possono essere estesi utilizzando l'API WebAssembly, con i seguenti obiettivi[36]:

- Efficienza: un'estensione deve aggiungere latenza, uso di CPU e di memoria in quantità minore possibile.
- Funzionalità: Un'estensione può imporre una politica, ottenere telemetria e eseguire modifiche al contenuto dei pacchetti.
- Isolamento: un fallimento in un plugin non deve influenzare gli altri plugin.
- Configurazione: Un'estensione è configurata utilizzando un'API coerente con le altri di Istio, inoltre è possibile eseguire la configurazione dinamicamente.
- Operatori: un'estensione può essere dispiegata per comportarsi come "log-only" (se si riscontra un errore, viene scritto nel registro dei log ma non viene presa nessun'azione), "fail-open" (se si verifica un'errore, solitamente nell'autenticazione, il traffico non è bloccato, dando priorità all'accesso invece che all'autenticazione), "fail-close" (se si verificano le condizioni di fallimento, il traffico è bloccato e viene impedito l'accesso).
- Sviluppo di estensioni semplice: le estensioni possono essere sviluppate utilizzando più linguaggi di programmazione.

6.6 Ambassador

Il software Ambassador è un piano di controllo specializzato per i proxy Envoy, che traduce le configurazione scritte dall'amministratore (sotto forma di CDR di Kubernetes) in configurazioni per il server Envoy.

Ambassador si affida a Kubernetes per l'adattamento, l'alta disponibilità e la persistenza, infatti la sua configurazione è memorizzata direttamente nel cluster, non è presente un database specifico. È distribuito come singolo container per Kubernetes che contiene il piano di controllo e un proxy Envoy e, normalmente, è gestito come un deployment.

L'architettura di Ambassador non ha stati e ciascuna istanza opera indipendentemente dalle altre. Questo permette di evitare la creazione e la configurazione di un piano di controllo centralizzato e sempre disponibile, che altrimenti sarebbe un onere dell'amministratore del sistema[1].

7 Caso di studio

Lo studio consiste nel verificare se l'aggiunta di Istio ad un cluster kubernetes, e quindi l'affiancamento di un proxy Envoy a tutti i pod, implichi un consumo di risorse hardware e temporali troppo elevato per l'utilizzo del software in un ambiente di produzione.

7.1 Configurazione dei cluster

Sono stati dispiegati due cluster kubernetes con 3 nodi ciascuno: un master e due worker, con Ubuntu server 18.04 come sistema operativo, processori virtuali con 4 core, e 8 GB di RAM. Come plugin di rete, nei cluster è stato installato Calico, in quanto sicuramente compatibile con Istio e non necessita di una configurazione particolare per far comunicare i pod all'interno di un cluster, sebbene sia possibile configurarlo dettagliatamente.

Su uno dei cluster è stato installato anche istioctl, suite di comandi per controllare le funzioni di Istio, e l'operatore (il cui demone è visibile nel pod *Istiod* in figura 14). Per poter ottenere delle metriche di rete, è necessario installare anche un software di monitoraggio che raccolga i dati in un unico luogo: in questo caso è stato scelto Prometheus, utilizzando le configurazioni di base fornite dal sito ufficiale dell'applicativo stesso o da quello di Istio, in base al cluster bersaglio. Queste configurazioni permettono di recuperare rispettivamente le metriche esposte dal server dell'API Kubernetes e dal demone Istiod.

Il software utilizzato come servizio fornito è un software gestionale Enterprise, abbreviato EMT, che si appoggia ad un database Oracle. L'applicazione fornisce la possibilità di rendicontare le ore di lavoro e raggrupparle per progetto; inoltre è presente anche un calendario di eventi interni all'azienda a cui l'utente può iscriversi.

Per ottenere le metriche di latenza del database, è stato usato un semplice programma scritto in Java, lo stesso linguaggio in cui è scritta l'applicazione EMT, dispiegato nel pod *Net-tester* in modo da connettersi al database, eseguire una query e memorizzare il tempo di risposta. Ipotizzando che il tempo di esecuzione della query sia uguale in entrambi i cluster, allora la differenza di latenza è attribuibile alla presenza dei proxy Envoy e la loro gestione da parte di Istio.

Le figure 13 e 14 mostrano una rappresentazione dello stato dei cluster, rispettivamente quello senza Istio e quello con Istio, omettendo la suddivisione in namespace e i pod di default del sistema Kubernetes, in quanto ininfluenti per l'analisi. Il pod di Prometheus è stato inserito in un namespace apposito: in questo modo, non è stato affiancato dal proxy Envoy, in quanto è stato usato solamente come strumento per raccogliere i dati.

7.2 Analisi delle metriche

Per confrontare l'efficienza dei due diversi cluster, sono state raccolte le seguenti metriche per ciascuno dei pod *emt-db*, *emt-webapp*, *net-tester*:

- **RAM utilizzata** dal pod: questa metrica è stata raccolta per controllare che la memoria richiesta dal proxy non sia elevata in confronto al container a cui è affiancato e perciò non ne diminuisca le prestazioni in maniera evidente.

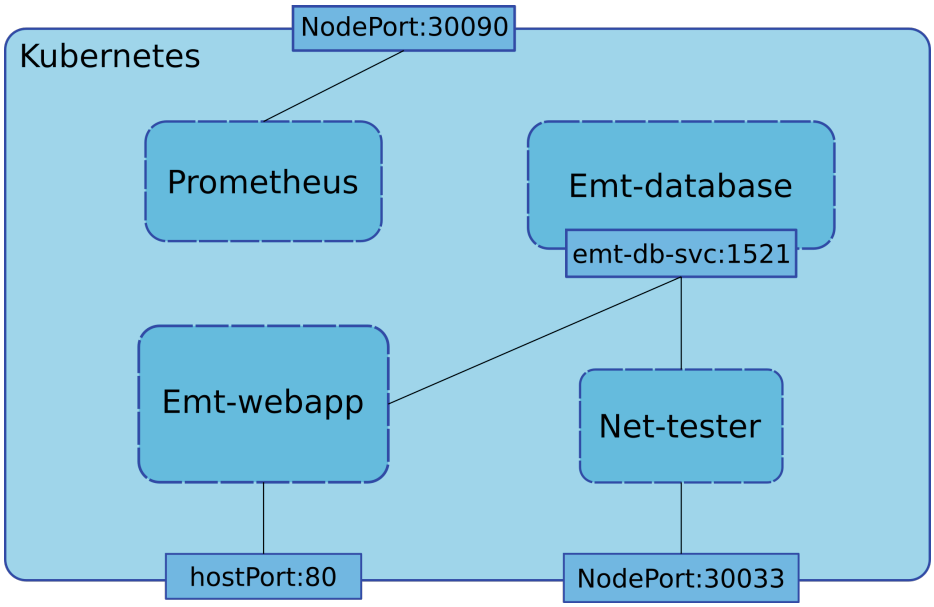

Figura 13: Rappresentazione dello stato del cluster Kubernetes senza Istio

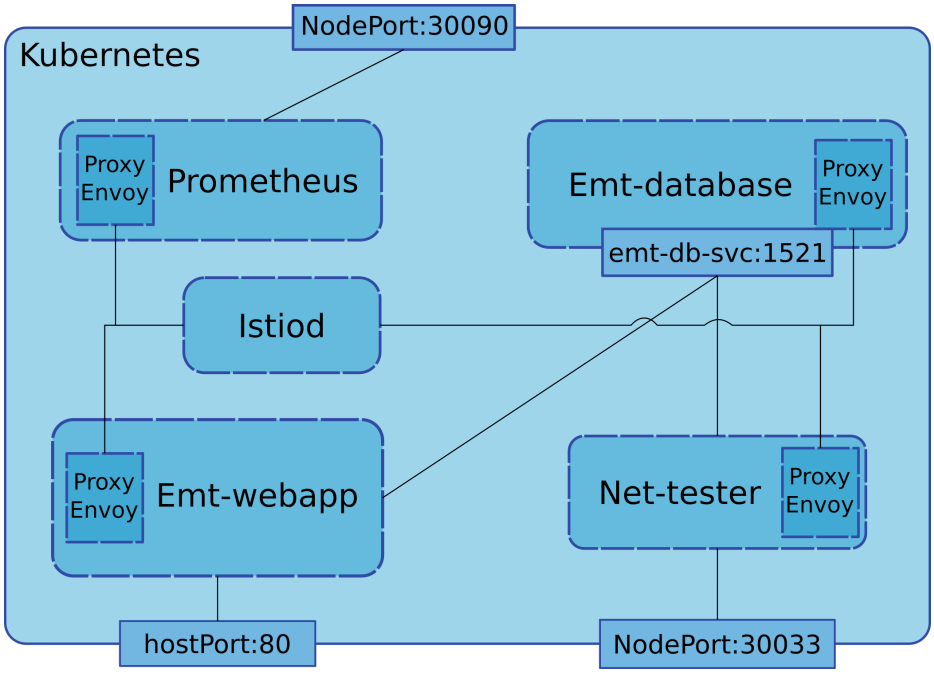

Figura 14: Rappresentazione dello stato del cluster Kubernetes con Istio

- **Percentuale di CPU utilizzata** dal pod: dato che i container di un pod sono processi di uno stesso sistema operativo, il proxy non deve gravare eccessivamente sulle risorse della macchina.
- **Pacchetti trasmessi e ricevuti** dal pod: è stato calcolato anche il numero di pacchetti trasmessi e ricevuti dai pod, in quanto le comunicazioni interne (soprattutto tra i proxy e il demone Istiod) non devono rallentare tutte le richieste interne e dall'esterno verso gli altri container e pod.
- **Latenza** tra i pod: è stata calcolata con l'applicazione nel pod net-tester cronometrando il tempo di risposta alle richieste che effettuava; si può supporre che la latenza tra qualsiasi coppia di pod all'interno di un cluster sia simile in quanto i pod presi in considerazione per questo test non avevano subito configurazioni che favorissero la comunicazione rispetto ad altri pod. Sono stati valutati i dati con un numero variabile di richieste effettuate: una al secondo, dieci al secondo, 100 al secondo e 1000 al secondo.

Tutti i valori analizzati sono stati ottenuti in un intervallo di tempo di 30 minuti, nel quale l'applicazione *net-tester* è stato in funzione eseguendo 10 richieste al secondo (tranne nel caso della latenza come precedentemente specificato) e ha mantenuto sotto sforzo il database per tutta la durata della raccolta dei dati.

7.2.1 RAM utilizzata

Ricordando che i nodi dei cluster utilizzano un sistema operativo GNU/Linux, la memoria ottenuta dai dati non è necessariamente quella effettivamente utilizzata, infatti viene fornita più RAM rispetto alla quantità strettamente necessaria ai processi (quando disponibile) per permettere di fare disk-caching e perciò aumentare la velocità di lettura/scrittura dei dati su disco[33]. Ogniqualvolta un nuovo processo viene avviato, la proprietà di una parte della memoria fornita in eccesso agli altri viene modificata per rispondere alle esigenze di esso. Inoltre, i container Docker sono effettivamente processi, perciò anche ad essi si applica il meccanismo di disk-caching. Perciò in certi casi, pod con più container possono utilizzare meno memoria rispetto a pod con meno container, in base ai processi eseguiti sulla macchina che ospita il pod. Per questo, il pod emt-db con il proxy affiancato, come si vede dalla figura 15, utilizza meno RAM. Al contrario, i pod emt-webapp e net-tester utilizzano più memoria nel caso siano stati affiancati al proxy. Complessivamente, la memoria utilizzata dai pod all'interno di un cluster è simile, perciò si suppone, in caso di un numero congruo di pod rispetto alle specifiche del cluster, che la presenza dei proxy immessi da Istio non sia deleteria per le prestazioni dei servizi forniti.

Come si può evincere dai dati (appendice A), la presenza del proxy Envoy occupa tra i 45 ed i 60 MB di RAM, ergo non impedisce il corretto funzionamento del container affiancato.

Pod emt-db Il pod emt-db ha sempre ricevuto le richieste di net-tester durante la raccolta dati, perciò senza il proxy utilizza, sia per operazioni interne che per il disk-caching, circa 1000 MB di RAM; diversamente nel cluster con Istio, il pod utilizza complessivamente 900 MB di memoria (figura 15).

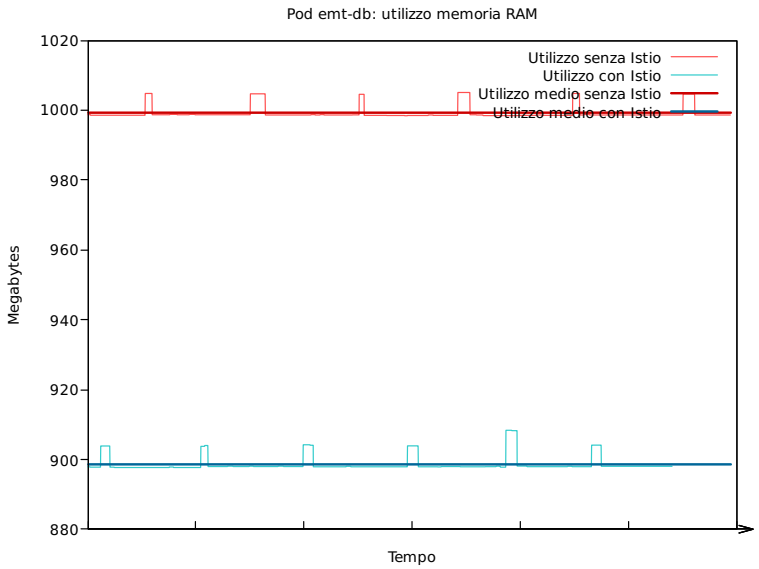

Figura 15: Confronto sull'utilizzo di memoria RAM tra i container emt-db nei due cluster

Pod emt-webapp L'applicativo web emt non è stato utilizzato intensivamente nel periodo di raccolta dati, quindi l'utilizzo di memoria si attesta a poco meno di 250 MB per il cluster senza proxy, ed a circa 300 MB per il cluster con Istio (figura 16).

Pod net-tester Il pod net-tester è sempre stato attivo durante i 30 minuti di lavoro, effettuando richieste al database presente nel pod emt-db. Anche in questo caso, il pod senza proxy ha utilizzato poco meno di 150 MB di RAM, mentre il pod analogo nel cluster con Istio ha utilizzato circa 60 MB in più di memoria (figura 17).

7.2.2 Percentuale di CPU utilizzata

Prometheus non permette di ottenere direttamente la percentuale di CPU utilizzata, ma solo il numero di secondi totali. Ipotizzando che la CPU sia utilizzata al 100% ogni volta che è richiesto, si può calcolare l'utilizzo percentuale medio

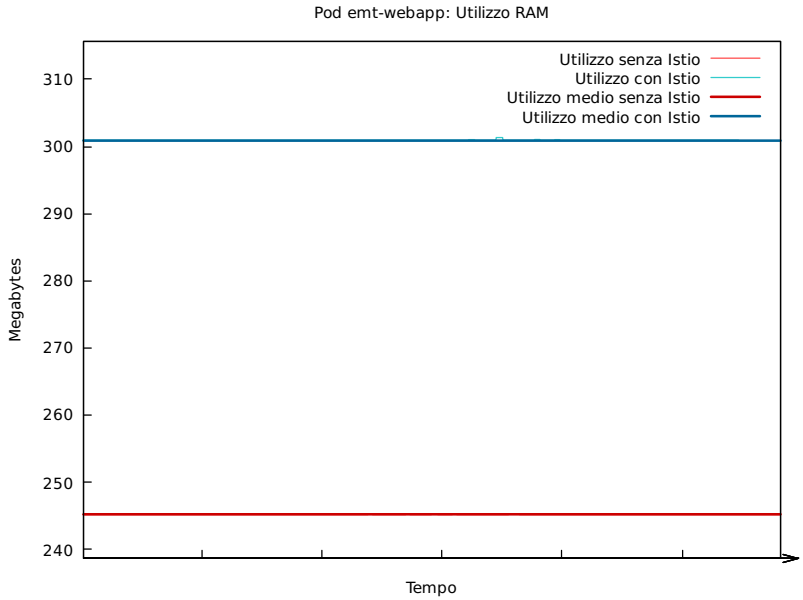

Figura 16: Confronto sull'utilizzo di memoria RAM tra i container emt-webapp nei due cluster

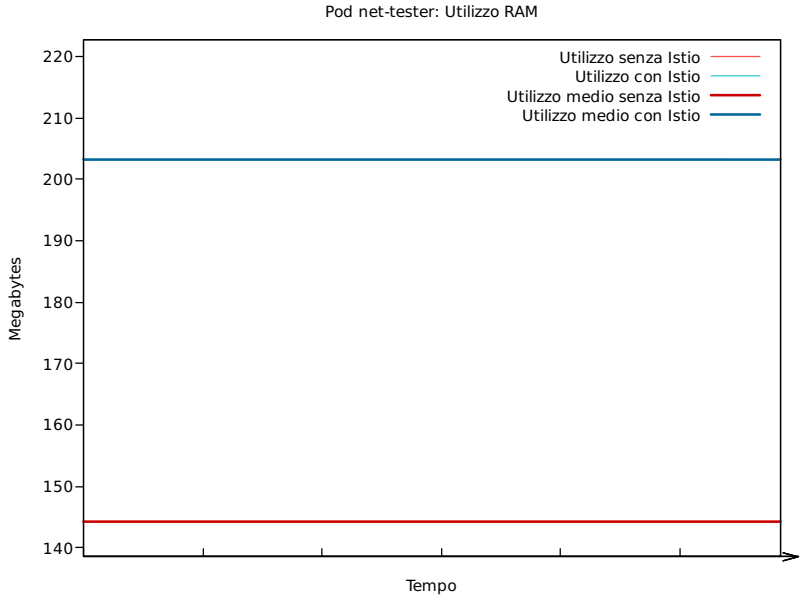

Figura 17: Confronto sull'utilizzo di memoria RAM tra i container net-tester nei due cluster

durante il periodo con la seguente formula:

$$(tempo\_utilizzo\_finale - tempo\_utilizzo\_iniziale)/periodo\_raccolta * 100 \quad (1)$$

con $periodo\_raccolta = 60 * 30 = 1800$ secondi.

I dati ottenuti eseguendo questa operazione sono visibili in forma tabulare alle appendici E e F.

Pod emt-db Il container emt-db utilizza all'incirca l'1.63% di CPU quando è sotto sforzo (probabilmente per la query molto semplice), mentre il container con il proxy di Istio fa raggiungere al pod circa il 5% di CPU utilizzata. Il proxy, tenendo in considerazione che durante il periodo di raccolta dati riceveva ed inviava pacchetti continuamente, non richiede molte risorse di calcolo, in quanto usa circa il 3.5% di CPU.

Pod emt-webapp La percentuale d'uso di processore del container emt-webapp non è alta, in quanto durante la raccolta dati l'applicazione non è stata usata. Nonostante questo, l'aggiunta del proxy ha decuplicato la richiesta di risorse di calcolo che risulta circa del 4.16%, sebbene l'uso iniziale fosse esiguo.

Pod net-tester Il pod net-tester per sua natura non utilizza molto la CPU, ma l'aggiunta del proxy lo ha incrementato di molto, trasformandolo da un container che utilizza meno dell'1% di CPU ad uno che ne utilizza circa il 4%.

I risultati ottenuti mostrano che generalmente i proxy aggiunti da Istio utilizzano circa il 3,5% di CPU indifferentemente dall'utilizzo del container affiancato. Nel caso in cui il processo affiancato non richieda molta potenza di calcolo, l'aggiunta del proxy può anche decuplicare la percentuale d'uso della CPU.

7.2.3 Pacchetti trasmessi e ricevuti

Il numero di pacchetti trasmessi e ricevuti, insieme agli errori riscontrati e alla latenza, permettono di stabilire la qualità della rete: un alto numero di errori può essere dovuto ad uno spaziamento tra pacchetti inconsistente che porta ad un problema di temporizzazione e sincronizzazione[78].

I dati sui pacchetti e i relativi errori sono nelle appendici B, C, D.

Dai dati si può vedere che la presenza di Istio incrementa il numero di pacchetti inviati e ricevuti di qualche decina di migliaia, ma non si sono verificati errori. Perciò, nonostante l'aumento di dati e trasmissioni di pacchetti, la presenza di Istio e quindi l'aggiunta di proxy intermedi tra i pod non ha modificato l'affidabilità della rete.

7.2.4 Latenza

Essendo un nodo aggiuntivo da attraversare, il proxy potrebbe aggiungere una latenza significativa alle comunicazioni. Infatti al posto di avere pacchetti che

sono trasmessi direttamente dal pod mittente al pod destinatario, gli stessi pacchetti dovranno essere prima inviati dal pod mittente al proprio proxy Envoy, poi sarà inviato al proxy del pod destinatario e con una terza comunicazione arriverà a destinazione. La latenza è stata calcolata utilizzando il pod net-tester, facendogli effettuare richieste al database con più thread contemporaneamente, variando il tempo di riposo tra una richiesta e l'altra.

La distanza tra i due pod coinvolti è compresa tra due casi:

- **Limite superiore**: i pod sono dispiegati su nodi diversi, perciò le comunicazioni passano attraverso uno o più cavi di rete. È la distanza massima possibile.
- **Limite inferiore**: i pod sono dispiegati sullo stesso nodo, quindi le comunicazioni avvengono tramite lo stesso kubelet. È la distanza minima che possono assumere.

Sebbene si possono presentare questi due casi indifferentemente, ciò non indice significativamente sulla latenza.

I valori ottenuti comprendono il tempo di query, il tempo di propagazione tra i pod e il tempo di propagazione tra il proxy e il pod, nel caso il primo sia presente. Ipotizzando che:

1. il tempo di query sia costante per ogni richiesta di un cluster, essendo il testo di query sempre uguale, e tra le richieste dei due cluster, dato che la configurazione del database è la stessa (a meno di Istio, che non influisce sul tempo di esecuzione della query);
2. il tempo di propagazione tra i pod sia tendenzialmente costante (essendo entrambi sulla stessa rete locale LAN);

si suppone che la differenza tra i valori di latenza dei cluster sia quindi il tempo aggiuntivo di propagazione introdotto dai proxy Envoy. In questo caso, dato che si hanno comunicazioni senza pod intermediari, per trovare il ritardo di propagazione aggiunto da Istio si deve dividere per due la differenza trovata (in quanto la latenza comprende sia il ritardo della consegna che quello della ricezione).

I quattro casi studiati sono stati configurati per effettuare, come specificato anche in precedenza, 1 richiesta al secondo, 10 richieste al secondo, 100 richieste al secondo e infine 1000 richieste al secondo. Solo i primi due casi hanno fornito risultati attendibili, in quanto negli altri si hanno valori di latenza che superano le decine di secondi. Questo risultato è probabilmente dovuto ad una configurazione del database e dell'applicazione che testava la metrica in questione non adatta all'elevato numero di richieste contemporanee, ma nonostante ciò, la tendenza del cluster con Istio di avere una latenza maggiore rispetto all'altro è rimasta verificata.

1 richiesta al secondo Il database non ha difficoltà nel gestire una richiesta al secondo e men che meno l'applicativo net-tester, il quale ha anche dei tempi

di attesa tra la fine di un thread e l'altro. In questo caso, come si vede dal grafico in figura 18, la differenza di latenza tra le due configurazioni è di circa 10 millisecondi, pertanto è poco influente per il tipo di applicazione trattata.

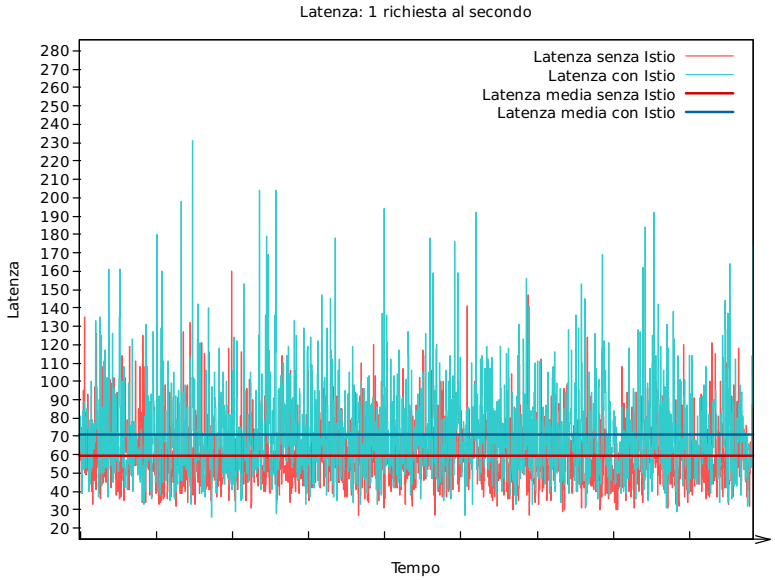

Figura 18: Confronto della latenza dei pod emt-db con 1 richiesta al secondo nei due cluster

10 richieste al secondo Aumentando il numero di richieste al secondo, si nota un leggero aumento di latenza nel cluster con Istio, la cui media si aggira a 80 millisecondi (grafico in figura 19), ma rimane comunque un quantitativo di tempo accettabile per l'esperienza utente.

100 richieste al secondo In questo caso, i valori di latenza sono incrementati a dismisura, ma essendo un evento comune ad entrambi i cluster, si evince che questa situazione è legata a delle configurazioni del database e dell'applicativo net-tester non adatte per sopportare questo numero di richieste, le quali si sovrappongono e si rallentano a vicenda, generando così latenze enormi (vedere grafico in figura 20). Nonostante ciò, rimane verificata la situazione in cui il cluster in cui è installato Istio presenta una latenza maggiore rispetto all'altro, sebbene i dati raccolti non rappresentino una configurazione corretta.

1000 richieste al secondo Similmente al caso precedente, sia per il database che per net-tester, una configurazione non adatta a gestire 1000 richieste al secondo ha portato a raccogliere dati sulla latenza molto elevati (come si vede nel grafico in figura 21). Anche in questo caso, però, si conferma la tendenza del cluster con Istio di incrementare la latenza del servizio.

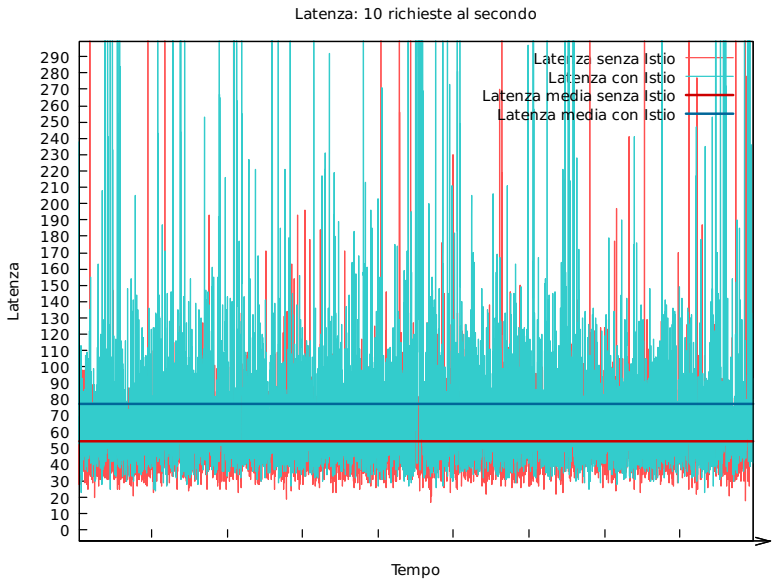

Figura 19: Confronto della latenza dei pod emt-db con 10 richieste al secondo nei due cluster

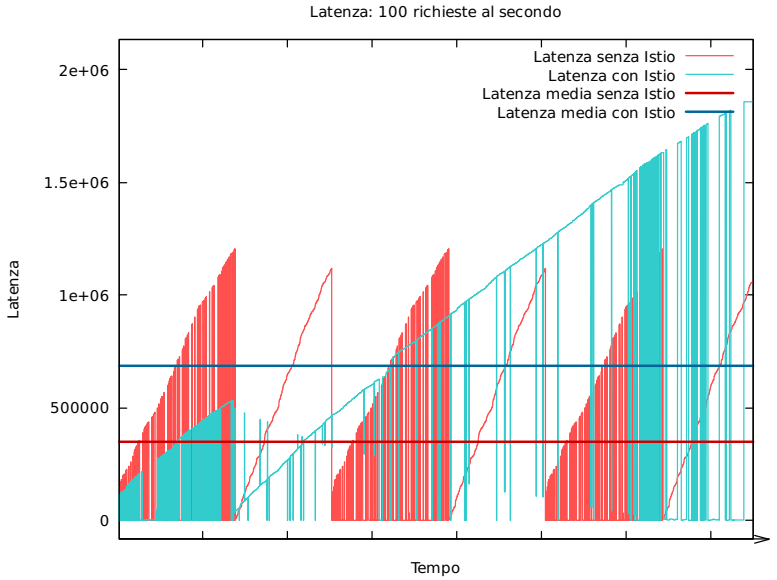

Figura 20: Confronto della latenza dei pod emt-db con 100 richieste al secondo nei due cluster

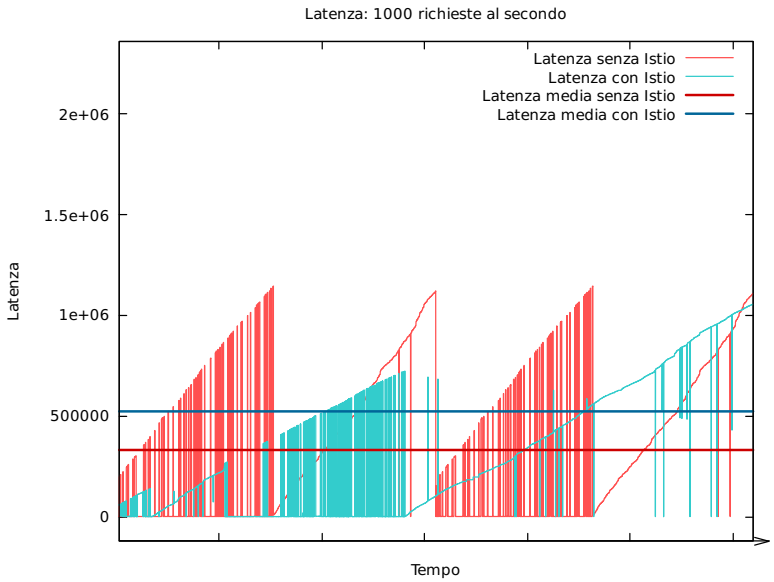

Figura 21: Confronto della latenza dei pod emt-db con 1000 richieste al secondo nei due cluster

8 Conclusioni

Istio è un software per service mesh che offre la possibilità di configurare la propria rete di servizi fino nei minimi dettagli, permettendo così, ad esempio, di controllare il traffico in entrata e in uscita con dei gateway oppure il traffico interno con le destination rule, oltre a poter implementare protocolli di sicurezza, meccanismi di telemetria e di bilanciamento di carico.

Queste funzionalità sono fornite attraverso l'estensione dei proxy Envoy, i quali sono affiancati ai container di un pod.

Il processo di ciascun proxy, come estrapolato dai dati del caso di studio, occupa mediamente all'incirca tra i 40 e i 60 MB di RAM, perciò l'uso di memoria volatile è molto basso, basti confrontarlo con le applicazioni utilizzate: il semplice applicativo net-tester necessita mediamente di circa 150 MB, mentre dall'altro lato il database richiede circa 1GB di memoria disponibile (corrispondente a circa 17 proxy). Per rispondere ad un numero superiore di richieste è necessario aumentare il numero di container o pod del database, perciò l'aggiunta di un proxy non è significativa per la disponibilità di RAM.

Come visto dai dati, i proxy richiedono una quantità di CPU disponibile fissa, circa il 3.5%, che moltiplicata per ogni pod può diventare una quantità importante. Nonostante ciò, i sistemi multi-core odierni (e, in alcuni casi, i server multi-processore) non hanno grosse difficoltà nel gestire Istio, soprattutto se sono dispiegati su più macchine e quindi possono avere una potenza di calcolo maggiore allo stesso prezzo. In ogni caso, l'aggiunta di Istio è molto conve-

niente nel caso il container affiancato debba eseguire operazioni molto onerose in termini di CPU, in quanto l'incremento nell'uso della capacità di calcolo, in percentuale, è minimo.

La presenza dei proxy incrementa enormemente la quantità di pacchetti scambiati tra i pod, ma non introduce errori e non aumenta l'ordine di grandezza della latenza. Questo risultato è probabilmente dovuto al fatto che le macchine siano collegate fisicamente, essendo molto raro il caso in cui i nodi di un cluster siano collegati attraverso connessioni wireless, le quali per la loro natura fisica hanno una maggiore probabilità di riscontrare errori.

Sebbene Istio introduca due passaggi intermedi nel collegamento tra un pod e l'altro, cioè i collegamenti tra container mittente e container destinatario ed il proprio proxy, la latenza rimane contenuta: l'incremento rispetto ad un cluster senza Istio è di circa 20 millisecondi e rimane nello stesso ordine di grandezza (sotto i 100 millisecondi), quindi tendenzialmente non è visibile all'utente.

Concludendo, l'aggiunta di Istio in un cluster porta molti benefici a discapito di un leggero aumento nella richiesta di risorse hardware.

Bibliografia e sitografia

[1] Ambassador. *The Ambassador Edge Stack Architecture*. URL: https://www.getambassador.io/docs/latest/topics/concepts/architecture/.

[2] Atlassian. *Calculating the cost of downtime*. URL: https://www.atlassian.com/incident-management/kpis/cost-of-downtime.

[3] David A. Bader e Robert Pennington. *Cluster Computing: Applications*. URL: https://web.archive.org/web/20080221171731/http://www.cc.gatech.edu/~bader/papers/ijhpca.pdf.

[4] Kamal Benzekki, Abdeslam El Fergougui e Abdelbaki Elbelrhiti Elalaoui. *Software-defined networking (SDN): a survey*. URL: https://onlinelibrary.wiley.com/doi/full/10.1002/sec.1737.

[5] Project Calico. *About Calico*. URL: https://docs.projectcalico.org/introduction/.

[6] CaptainCap. *What is management plane in SDN?* URL: https://stackoverflow.com/questions/30190944/what-is-management-plane-in-sdn.

[7] Eric Carter. *2018 Docker usage report*. URL: https://sysdig.com/blog/2018-docker-usage-report/#whatcontainerruntimesareinuse.

[8] David Castillo. *Stack Overflow - What are Docker image layers?* URL: https://stackoverflow.com/a/33836848.

[9] Cisco. *Border Gateway Protocol (BGP)*. URL: https://www.cisco.com/c/en/us/products/ios-nx-os-software/border-gateway-protocol-bgp/index.html.

[10] Cloudflare. *What Is Function as a Service (FaaS)?* URL: https://www.cloudflare.com/learning/serverless/glossary/function-as-a-service-faas/.

[11] coreos. *Flannel readme*. URL: https://github.com/coreos/flannel/blob/master/README.md.

[12] M. Cotton et al. *RFC 6890: Special-Purpose IP Address Registries*. URL: https://www.rfc-editor.org/rfc/rfc6890.html#section-2.2.2.

[13] Docker. *Best practice for writing a Dockerfile*. URL: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/.

[14] Docker. *Disable networking for a container*. URL: https://docs.docker.com/network/none/.

[15] Docker. *Docker concepts*. URL: https://docs.docker.com/get-started/#docker-concepts.

[16] Docker. *Images and containers*. URL: https://docs.docker.com/get-started/#images-and-containers.

[17] Docker. *Manage data in Docker*. URL: https://docs.docker.com/storage/.

[18] Docker. *Network drivers*. URL: https://docs.docker.com/network/#network-drivers.

[19] Docker. *Swarm mode key concepts*. URL: https://docs.docker.com/engine/swarm/key-concepts/.

[20] Docker. *Swarm mode key concepts - Nodes*. URL: https://docs.docker.com/engine/swarm/key-concepts/#nodes.

[21] Docker. *Use bind mounts*. URL: https://docs.docker.com/storage/bind-mounts/.

[22] Docker. *Use bridge networks*. URL: https://docs.docker.com/network/bridge/.

[23] Docker. *Use host networking*. URL: https://docs.docker.com/network/host/.

[24] Docker. *Use macvlan networks*. URL: https://docs.docker.com/network/macvlan/.

[25] Docker. *Use overlay networks*. URL: https://docs.docker.com/network/overlay/.

[26] Docker. *Use tmpfs mounts*. URL: https://docs.docker.com/storage/tmpfs/.

[27] Docker. *Use volumes*. URL: https://docs.docker.com/storage/volumes/.

[28] Cloud Native Computing Foundation. *Etcd homepage*. URL: https://etcd.io/.

[29] Vikram Fugro. *Container Networking Interface aka CNI*. URL: https://medium.com/@vikram.fugro/container-networking-interface-aka-cni-bdfe23f865cf.

[30] Google. *Cosa sono i container e quali vantaggi offrono*. URL: https://cloud.google.com/containers/.

[31] J. Hawkinson e T. Bates. *RFC 1930: Guidelines for creation, selection, and registration of an Autonomous System (AS)*. URL: https://www.rfc-editor.org/rfc/rfc1930.html.

[32] C. Hedrick. *Routing Information Protocol*. URL: https://tools.ietf.org/html/rfc1058.

[33] Vidar Holen. *Linux ate my ram!* URL: https://www.linuxatemyram.com/index.html.

[34] IBM. *What is Containers as a Service (CaaS)?* URL: https://www.ibm.com/services/cloud/containers-as-a-service.

[35] Istio. *Architecture*. URL: https://istio.io/latest/docs/ops/deployment/architecture/.

[36] Istio. *Extensibility*. URL: https://istio.io/latest/docs/concepts/wasm/.

[37] Istio. *Observability*. URL: https://istio.io/latest/docs/concepts/observability/.

[38] Istio. *Security*. URL: https://istio.io/latest/docs/concepts/security/.

[39] Istio. *What is a service mesh?* URL: https://istio.io/latest/docs/concepts/what-is-istio/#what-is-a-service-mesh.

[40] Istio. *What is Istio?* URL: https://istio.io/latest/docs/concepts/what-is-istio/.

[41] Kubernetes. *cloud-controller-manager*. URL: https://kubernetes.io/docs/concepts/overview/components/#cloud-controller-manager.

[42] Kubernetes. *Container runtime*. URL: https://kubernetes.io/docs/concepts/overview/components/#container-runtime.

[43] Kubernetes. *Control Plane Components*. URL: https://kubernetes.io/docs/concepts/overview/components/#control-plane-components.

[44] Kubernetes. *Deployments*. URL: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.

[45] Kubernetes. *DNS for Services and Pods*. URL: https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/.

[46] Kubernetes. *etcd*. URL: https://kubernetes.io/docs/concepts/overview/components/#etcd.

[47] Kubernetes. *Ingress*. URL: https://kubernetes.io/docs/concepts/services-networking/ingress/.

[48] Kubernetes. *Ingress Controller*. URL: https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/.

[49] Kubernetes. *Installing Addons*. URL: https://kubernetes.io/docs/concepts/cluster-administration/addons/.

[50] Kubernetes. *kube-apiserver*. URL: https://kubernetes.io/docs/concepts/overview/components/#kube-apiserver.

[51] Kubernetes. *kube-controller-manager*. URL: https://kubernetes.io/docs/concepts/overview/components/#kube-controller-manager.

[52] Kubernetes. *kube-proxy*. URL: https://kubernetes.io/docs/concepts/overview/components/#kube-proxy.

[53] Kubernetes. *kube-scheduler*. URL: https://kubernetes.io/docs/concepts/overview/components/#kube-scheduler.

[54] Kubernetes. *kubelet*. URL: https://kubernetes.io/docs/concepts/overview/components/#kubelet.

[55] Kubernetes. *Network Policies*. URL: https://kubernetes.io/docs/concepts/services-networking/network-policies/#sctp-support.

[56] Kubernetes. *Operator pattern*. URL: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/.

[57] Kubernetes. *Persistent Volumes*. URL: https://kubernetes.io/docs/concepts/storage/persistent-volumes/.

[58] Kubernetes. *PersistentVolumeClaims*. URL: https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims.

[59] Kubernetes. *Pods*. URL: https://kubernetes.io/docs/concepts/workloads/pods/#resource-sharing-and-communication.

[60] Kubernetes. *Service*. URL: https://kubernetes.io/docs/concepts/services-networking/service/.

[61] Kubernetes. *Services without selectors*. URL: https://kubernetes.io/docs/concepts/services-networking/service/#services-without-selectors.

[62] Kubernetes. *Volumes*. URL: https://kubernetes.io/docs/concepts/storage/volumes/.

[63] Roland McGrath. *chroot - Linux manual page*. URL: https://www.man7.org/linux/man-pages/man1/chroot.1.html.

[64] Peter Mell e Tim Grace. *The NIST Definition of Cloud Computing*. Special Publication 800-145. Set. 2011. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[65] Microsoft. *Architecture*. URL: https://docs.microsoft.com/en-us/azure/aks/servicemesh-about#architecture.

[66] Microsoft. *Cos'è il modello IaaS*. URL: https://azure.microsoft.com/it-it/overview/what-is-iaas/.

[67] Microsoft. *Cos'è il modello PaaS*. URL: https://azure.microsoft.com/it-it/overview/what-is-paas/.

[68] Microsoft. *Cos'è il modello SaaS*. URL: https://azure.microsoft.com/it-it/overview/what-is-saas/.

[69] Microsoft. *Serverless computing*. URL: https://azure.microsoft.com/en-us/overview/serverless-computing/.

[70] Open Networking. *OpenFlow® Conformance Certification*. URL: https://www.opennetworking.org/product-certification/.

[71] nuagenetworks. *The Tale of Two Container Networking Standards: CNM v. CNI*. URL: https://www.nuagenetworks.net/blog/container-networking-standards/.

[72] Oracle. *Mobile Hub service*. URL: https://www.oracle.com/application-development/cloud-services/mobile/.

[73] Envoy Project. *Authentication filter*. URL: https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/security/ssl#arch-overview-ssl-auth-filter.

[74] Envoy Project. *HTTP connection management*. URL: https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/http/http_connection_management#arch-overview-http-conn-man.

[75] Envoy Project. *What is Envoy*. URL: https://www.envoyproxy.io/docs/envoy/latest/intro/what_is_envoy.

[76] Linux man-pages project. *cgroups - Linux manual page*. URL: https://man7.org/linux/man-pages/man7/cgroups.7.html.

[77] Regional Internet Registries. *Regional Internet Registries Statistics*. URL: https://www-public.imtbs-tsp.eu/~maigron/RIR_Stats/RIR_Delegations/World/ASN-ByNb.html.

[78] Logan Rivenes. *What are the Causes of Packet Loss?* URL: https://datapath.io/resources/blog/causes-of-packet-loss/.

[79] Runnable. *Binding Ports*. URL: https://runnable.com/docker/binding-docker-ports.

[80] John Spacey. *Thin Client vs Thick Client*. URL: https://simplicable.com/new/thin-client-vs-thick-client.

[81] Techopedia. *Virtual Machine Snapshot (VM Snapshot)*. URL: https://www.techopedia.com/definition/16821/virtual-machine-snapshot-vm-snapshot.

[82] Ayman Totounji. *Virtual Machines: A Closer Look*. URL: https://www.cynexlink.com/2017/08/18/virtual-machines-pros-cons/.

[83] Typeof. *Types Of Computer Cluster*. URL: https://www.typesof.com/types-of-computer-cluster/.

[84] Stanford University. *Ethane: A Security Management Architecture*. URL: http://yuba.stanford.edu/ethane/.

[85] VMware. *Memory Overcommitment*. URL: https://pubs.vmware.com/vsphere-51/index.jsp?topic=%2Fcom.vmware.vsphere.resmgmt.doc%2FGUID-895D25BA-3929-495A-825B-D2A468741682.html.

[86] David Ward. *The pros and cons of using a virtualized machine*. URL: https://www.controleng.com/articles/the-pros-and-cons-of-using-a-virtualized-machine/.

[87] Wikipedia. *Address Resolution Protocol*. URL: https://it.wikipedia.org/wiki/Address_Resolution_Protocol.

[88] Wikipedia. *ARM architecture*. URL: https://en.wikipedia.org/wiki/ARM_architecture.

[89] Wikipedia. *As a service*. A list of cloud service models. URL: https://en.wikipedia.org/wiki/As_a_service.

[90] Wikipedia. *Attributes of clusters*. URL: https://en.wikipedia.org/wiki/Computer_cluster#Attributes_of_clusters.

[91] Wikipedia. *Binary translation*. URL: https://en.wikipedia.org/wiki/Binary_translation.

[92] Wikipedia. *Calculating the routing table*. URL: https://en.wikipedia.org/wiki/Link-state_routing_protocol#Calculating_the_routing_table.

[93] Wikipedia. *Classless Inter-Domain Routing*. URL: https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing.

[94] Wikipedia. *Clustered file system*. URL: https://en.wikipedia.org/wiki/Clustered_file_system.

[95] Wikipedia. *Comparison of instruction set architectures*. URL: https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures#Instruction_sets.

[96] Wikipedia. *Computer architecture*. URL: https://en.wikipedia.org/wiki/Computer_architecture.

[97] Wikipedia. *Computer cluster*. URL: https://en.wikipedia.org/wiki/Computer_cluster.

[98] Wikipedia. *Control plane*. URL: https://en.wikipedia.org/wiki/Control_plane.

[99] Wikipedia. *Copy-on-write*. URL: https://en.wikipedia.org/wiki/Copy-on-write.

[100] Wikipedia. *Data plane*. URL: https://en.wikipedia.org/wiki/Data_plane.

[101] Wikipedia. *Distance-vector routing protocol*. URL: https://en.wikipedia.org/wiki/Distance-vector_routing_protocol.

[102] Wikipedia. *Distributed file systems*. URL: https://en.wikipedia.org/wiki/Clustered_file_system#Distributed_file_systems.

[103] Wikipedia. *Distributing maps*. URL: https://en.wikipedia.org/wiki/Link-state_routing_protocol#Distributing_maps.

[104] Wikipedia. *Failure modes*. URL: https://en.wikipedia.org/wiki/Link-state_routing_protocol#Failure_modes.

[105] Wikipedia. *Full virtualization*. URL: https://en.wikipedia.org/wiki/Full_virtualization.

[106] Wikipedia. *Hardware-assisted virtualization*. URL: https://en.wikipedia.org/wiki/Hardware-assisted_virtualization.

[107] Wikipedia. *Hypervisor*. URL: https://en.wikipedia.org/wiki/Hypervisor.

[108] Wikipedia. *Hypervisor - Classification*. URL: https://en.wikipedia.org/wiki/Hypervisor#Classification.

[109] Wikipedia. *Indirizzo MAC*. URL: https://it.wikipedia.org/wiki/Indirizzo_MAC.

[110] Wikipedia. *Infrastructure as a service*. URL: https://en.wikipedia.org/wiki/Infrastructure_as_a_service.

[111] Wikipedia. *Interior gateway protocol*. URL: https://en.wikipedia.org/wiki/Interior_gateway_protocol.

[112] Wikipedia. *Internal scalability*. URL: https://en.wikipedia.org/wiki/Border_Gateway_Protocol#Internal_scalability.

[113] Wikipedia. *IPv4 address exhaustion*. URL: https://en.wikipedia.org/wiki/IPv4_address_exhaustion.

[114] Wikipedia. *IPv4 CIDR blocks*. URL: https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing#IPv4_CIDR_blocks.

[115] Wikipedia. *Link-state routing protocol*. URL: https://en.wikipedia.org/wiki/Link-state_routing_protocol.

[116] Wikipedia. *Linux namespaces*. URL: https://en.wikipedia.org/wiki/Linux_namespaces.

[117] Wikipedia. *List of TCP and UDP port numbers*. URL: https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers.

[118] Wikipedia. *Message Passing Interface*. URL: https://en.wikipedia.org/wiki/Message_Passing_Interface.

[119] Wikipedia. *Mobile backend as a service*. URL: https://en.wikipedia.org/wiki/Mobile_backend_as_a_service.

[120] Wikipedia. *Network address translation*. URL: https://en.wikipedia.org/wiki/Network_address_translation.

[121] Wikipedia. *Network as a service*. URL: https://en.wikipedia.org/wiki/Network_as_a_service.

[122] Wikipedia. *Network-attached storage*. URL: https://en.wikipedia.org/wiki/Network-attached_storage.

[123] Wikipedia. *Operation*. URL: https://en.wikipedia.org/wiki/Border_Gateway_Protocol#Operation.

[124] Wikipedia. *OS-level virtualization*. URL: https://en.wikipedia.org/wiki/OS-level_virtualization.

[125] Wikipedia. *Parallel Virtual Machine*. URL: https://en.wikipedia.org/wiki/Parallel_Virtual_Machine.

[126] Wikipedia. *Port (computer networking)*. URL: https://en.wikipedia.org/wiki/Port_(computer_networking).

[127] Wikipedia. *Rate limiting*. URL: https://en.wikipedia.org/wiki/Rate_limiting.

[128] Wikipedia. *Security*. URL: https://en.wikipedia.org/wiki/Border_Gateway_Protocol#Security.

[129] Wikipedia. *Sequoia (supercomputer)*. URL: https://en.wikipedia.org/wiki/Sequoia_(supercomputer).

[130] Wikipedia. *Service mesh*. URL: https://en.wikipedia.org/wiki/Service_mesh.

[131] Wikipedia. *Shared-disk file system*. URL: https://en.wikipedia.org/wiki/Clustered_file_system#Shared-disk_file_system.

[132] Wikipedia. *Software-defined networking*. URL: https://en.wikipedia.org/wiki/Software-defined_networking.

[133] Wikipedia. *Split horizon route advertisement*. URL: https://en.wikipedia.org/wiki/Split_horizon_route_advertisement.

[134] Wikipedia. *Stability*. URL: https://en.wikipedia.org/wiki/Border_Gateway_Protocol#Stability.

[135] Wikipedia. *Subnetwork*. URL: https://en.wikipedia.org/wiki/Subnetwork.

[136] Wikipedia. *Supernetwork*. URL: https://en.wikipedia.org/wiki/Supernetwork.

[137] Wikipedia. *UnionFS*. URL: https://en.wikipedia.org/wiki/UnionFS.

[138] Wikipedia. *Virtual machine*. URL: https://en.wikipedia.org/wiki/Virtual_machine.

[139] Wikipedia. *VMware ESXi*. URL: https://en.wikipedia.org/wiki/VMware_ESXi.

[140] Wikipedia. *x86*. URL: https://en.wikipedia.org/wiki/X86.

[141] L. Yang et al. *RFC 3746: Forwarding and Control Element Separation (ForCES) Framework*. URL: https://tools.ietf.org/html/rfc3746#page-5.

[142] Hao Zeng et al. "Measurement and Evaluation for Docker Container Networking". Documento di conferenza. College of Computer; National University of Defense Technology; Changsha, China, 2017.

Elenco delle figure

Elenco delle tabelle

Appendice

A Dati raccolti: utilizzo medio di memoria RAM del proxy Envoy

Pod	Utilizzo medio di RAM del proxy
emt-db	46,305
emt-webapp	54,035
net-tester	58,496

Utilizzo medio di RAM del proxy Envoy nei pod del cluster con Istio

B Dati raccolti: Pacchetti ricevuti

Pod	Ricevuti senza Istio	Ricevuti con Istio	Differenza
emt-db	7586	54149	46563
emt-webapp	175	53682	53507
net-tester	18264	67720	49456

Pacchetti ricevuti dai pod

C Dati raccolti: Pacchetti trasmessi

Pod	Trasmessi senza Istio	Trasmessi con Istio	Differenza
emt-db	7397	66419	59022
emt-webapp	235	67720	67485
net-tester	18216	47418	29202

Pacchetti trasmessi dai pod

D Dati raccolti: Errori nella trasmissione e nella ricezione di pacchetti

Pod	Errori in trasmissione senza Istio	Errori in trasmissione con Istio	Errori in ricezione senza istio	Errori in ricezione con Istio
emt-db	0	0	0	0
emt-webapp	0	0	0	0
net-tester	0	0	0	0

Pacchetti ricevuti dai pod

E Dati raccolti: Utilizzo CPU senza Istio

La percentuale è stata calcolata su un massimo di 1800 secondi, cioè la durata del periodo di raccolta dati.

Pod	container	Δ secondi	% secondi	% totale
emt-db	emt-db	29,346	1,63	1,63
emt-webapp	emt-webapp	5,978	0,33	0,33
net-tester	net-tester	7,217	0,40	0,40

Percentuale di CPU usata dai container dei pod senza proxy

F Dati raccolti: Utilizzo CPU con Istio

La percentuale è stata calcolata su un massimo di 1800 secondi, cioè la durata del periodo di raccolta dati.

Pod	container	Δ secondi	% secondi	% totale
emt-db	emt-db	30,402	1,69	5,08
	istio-proxy	60,987	3,39	
emt-webapp	emt-webapp	7,102	0,39	4,16
	istio-proxy	67,776	3,77	
net-tester	net-tester	6,241	0,35	4,05
	istio-proxy	66,477	3,70	

Percentuale di CPU usata dai container dei pod con proxy