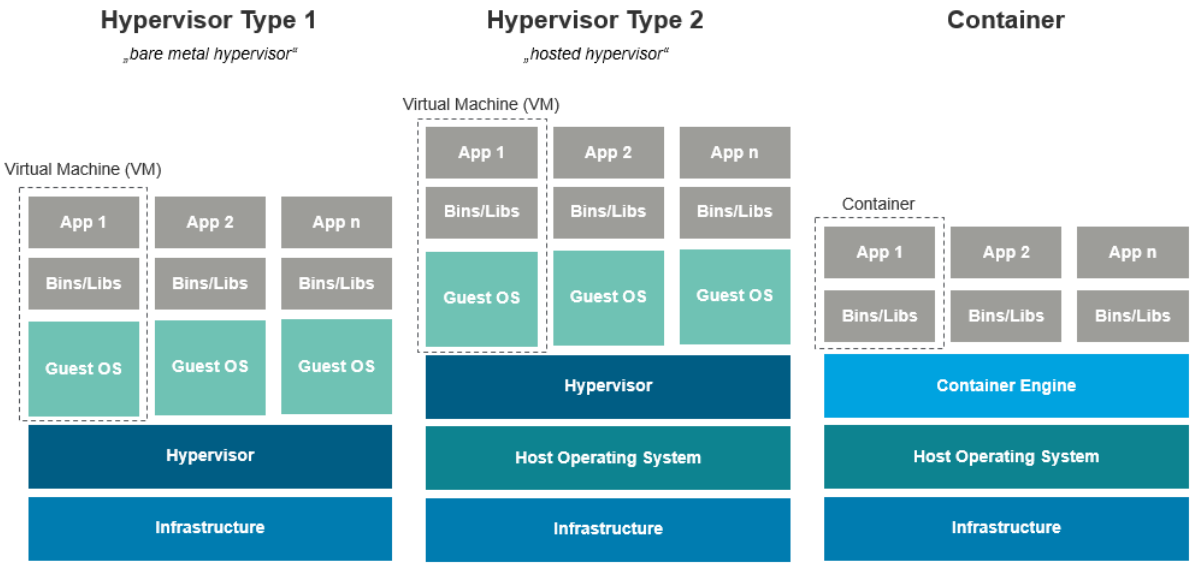
Docker

Struttura

L’architettura di docker è di tipo client-server.Tra i componenti troviamo il Docker Client e il Docker Daemon . Il Docker Client contatta il Daemon utilizzando il comando **docker** che sfrutta delle REST API , è il metodo più utilizzato per comunicare con il docker daemon ed eseguire container , un altro metodo è Docker desktop.
Il docker daemon invece, riceve ed elabora le chiamate da docker client e si occupa di costruire, eseguire e distribuire i nostri container.
La comunicazione tra il docker daemon e il docker client avviene con l’uso di REST API tramite il canale UNIX SOCKET (permette la comunicazione tra componenti dello stesso sistema) o grazie ad un’interfaccia di rete (ad esempio TCP, questo viene utilizzato per le gestione distribuita di container , potendo far comunicare il docker daemon con un client facente parte di un altro sistema).



A differenza delle VM i Container sono leggeri e condividono il kernel dell'host con altri container. Ciò significa che condividono le risorse del sistema operativo dell'host e sono più efficienti in termini di utilizzo di risorse rispetto alle VM.

Containers Docker

Immagini

Per parlare di container docker dobbiamo aver chiaro cosa sia un immagine.
Un immagine è un template di sola lettura contenente le istruzioni per creare e configurare un container docker.
Si possono trovare molte immagini già pronte all' interno dei docker registry come Docker Hub , oppure si possono creare le proprie , spesso partendo da immagini gia esistenti e aggiungendo le configurazioni di cui abbiamo bisogno.
Per creare un' immagine si utilizza un Dockerfile , un file di configurazione contenente le istruzioni per la creazione del container, ogni istruzione nel container corrisponde ad un layer durante la costruzione del container una cosa molto utile è che quando andiamo a modificare un dockerfile ed eseguiamo il rebuild dell'immagine, solo i layer che sono stati modificati vengono ricostruiti.

Containers

Un container è un'istanza eseguibile di un'immagine. Puoi creare, avviare, fermare, spostare o eliminare un container usando l'API o la CLI di Docker. Puoi connettere un container a una o più reti, allegare archiviazione ad esso o persino creare una nuova immagine basata sul suo stato attuale.

Quando avvii un container , vengono eseguite diverse azioni , se per esempio volessimo avviare un container usando il comando **docker run -it ubuntu /bin/bash**
in ordine verrebbero eseguite le seguenti azioni :

- Se non abbiamo l immagine in locale viene scaricata dal registro configurato
- Docker crea un container
- Docker assegna un filesystem di scrittura/lettura isolato al container in base alle istruzioni presenti nel dockerfile, in modo che si possano creare/modificare file e directories al suo interno mentre è in esecuzione
- Viene assegnata al container un'interfaccia di rete collegata a quella principale che gli permette di collegarsi all'esterno utilizzando la connessione internet del sistema host.
- Docker avvia il container ed esegue /bin/bash .

La separazione dei container docker è gestita dai "namespaces", una caratteristica del kernel Linux che consente di isolare e separare diversi aspetti di un sistema operativo.

I container possono essere utilizzati per lo sviluppo in modo da avere ambiente di sviluppo, trasportabile e replicabile , autoconsistente .
Queste permette agli sviluppatori di una determinata applicazione di lavorare all'interno di ambienti di sviluppo uguali , evitando problemi di dipendenze dei pacchetti e configurazioni.
I container utilizzati per questi scopi sono chiamati **dev container** e spesso utilizzano filesystem più estesi , contenenti tutto ciò di cui gli sviluppatori hanno bisogno per lavorare.
Quando si porta un applicazione in produzione , invece, si utilizzano docker images minimali contenenti solamente i binari e le librerie necessarie a far funzionare l'applicazione e

garantire il servizio. Questo diminuisce i rischi dovuti ad attacchi informatici in quanto le librerie e programmi presenti nel container sono ridotti al minimo e questo facilità la manutenzione permettendoci di esporre meno vulnerabilità possibile da sfruttare all'attaccante.

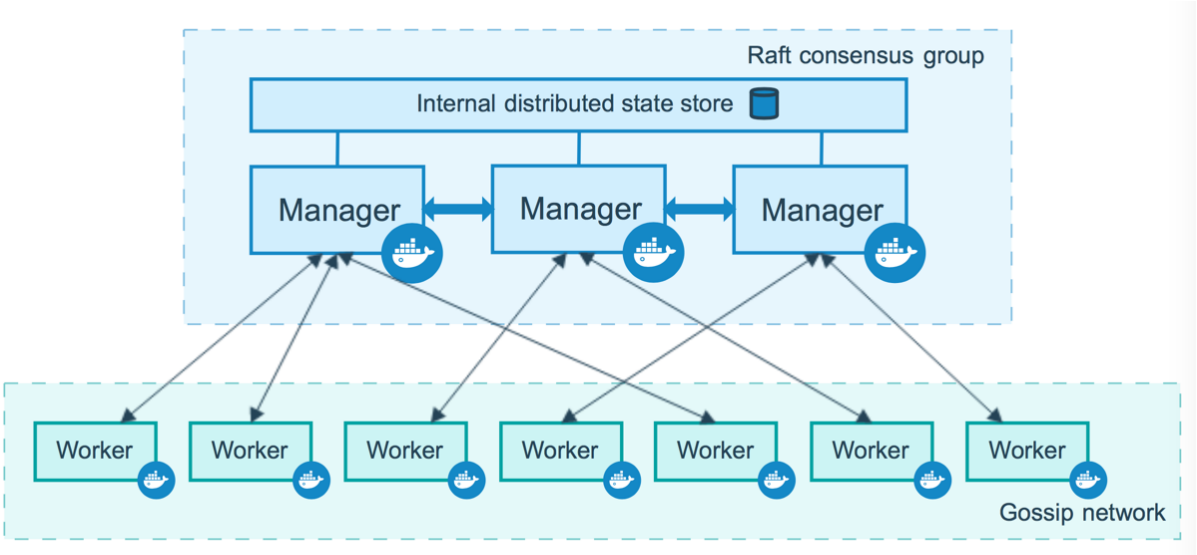
Docker Compose

Compose è uno strumento per definire ed eseguire applicazioni multi-container in Docker. Con compose si utilizza un file YAML per configurare i servizi di un'applicazione. Poi, con un singolo comando, si creano e avviano tutti i servizi dalla configurazione.
Se per esempio avessimo bisogno di sviluppare un'applicazione che necessità di un database e un ngnix server potremmo con un unico file YAML creare due container contenenti uno il servizio mysql e l'altro ngnix e farli comunicare .
Tuttavia compose è uno strumento utilizzato per lo sviluppo e il collaudo , il suo utilizzo per la produzione è sconsigliato.

Docker Swarm

Docker Swarm è una funzione docker che permette l'orchestrazione dei container attraverso la funzionalità di clustering host nativa di docker , ovvero la distribuzione dei container su più macchine fisiche che lavorano assieme.
Un cluster Swarm è formato da due tipi di nodi (i nodi sono le macchine che svolgono le attività):

- **Nodi Manager:** i nodi incaricati della gestione dei nodi di lavoro (o nodi worker) ed eseguono tutti i comandi dell'interfaccia a riga di comando di Docker relativi alla gestione e monitoraggio di uno Swarm
- **Nodi Worker** : I nodi Worker sono quelli che si occupano dell'esecuzione dei container per i servizi containerizzati



I Cluster vengono usati per distribuire i servizi containerizzati su più Host , garantendo un servizio di alta affidabilità e con un elevata scalabilità Questo è un modello molto semplice di cluster , i cluster kubernetes possono essere molto più complessi.

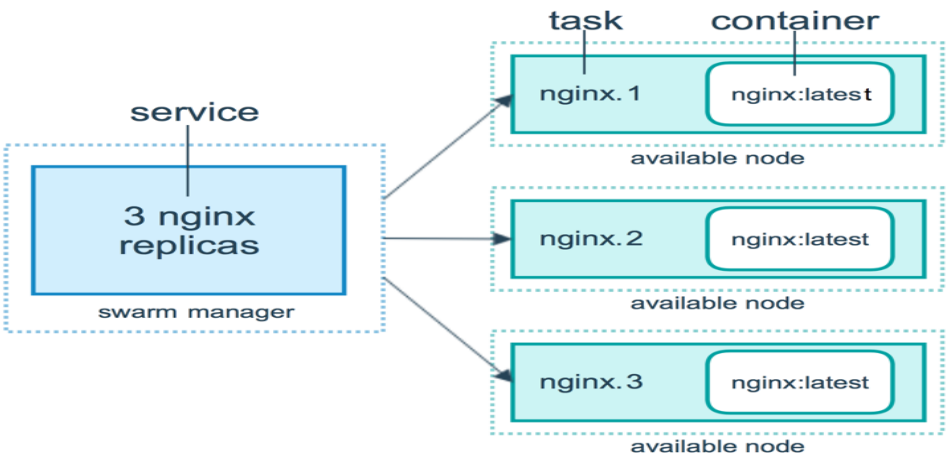
Docker Swarm si avvale di **Raft** , un protocollo di consenso distribuito , ecco alcune delle sue caratteristiche chiave:

- **Leader Election (Elezione del Leader)**: In RAFT, un nodo all'interno del cluster viene eletto come leader. Il leader è responsabile della gestione delle operazioni di scrittura e dell'invio dei dati agli altri nodi. Gli altri nodi sono chiamati "follower" e seguono le istruzioni del leader.
- **Log Replication (Replicazione del Registro)**: Ogni modifica di stato all'interno del cluster viene registrata in un registro replicato. Il leader è responsabile della replica dei dati all'interno del registro in modo che tutti i nodi abbiano una copia coerente delle modifiche.
- **Safety (Sicurezza)**: RAFT è progettato per garantire la sicurezza dei dati. Un'operazione di scrittura non viene considerata confermata finché il leader non ha ricevuto una conferma dalla maggioranza dei nodi follower.

Per distribuire un'applicazione quando Docker Engine è in modalità Swarm, si crea un **service**. Spesso, un service rappresenta l'immagine per un microservizio all'interno del contesto di un'applicazione più ampia. Esempi di servizi potrebbero includere un server HTTP, un database o qualsiasi altro tipo di programma eseguibile che si desidera eseguire in un ambiente distribuito.

Quando si crea un servizio, si specifica quale immagine del container utilizzare e quali comandi eseguire all'interno dei container in esecuzione. Si definiscono anche le opzioni per il servizio, tra cui:

- la porta attraverso cui il servizio è reso disponibile al di fuori del cluster Swarm
- una rete sovrapposta per il servizio per connettersi ad altri servizi nel cluster Swarm
- limiti e riserve di CPU e memoria
- una politica di aggiornamento continuo
- il numero di repliche dell'immagine da eseguire nel cluster Swarm



Quando si distribuisce il servizio all'interno del cluster Swarm, il manager Swarm accetta la definizione del servizio come stato desiderato per il servizio. Successivamente, pianifica il servizio su nodi all'interno del cluster come uno o più **task** (attività replica). Le attività vengono eseguite in modo indipendente l'una dall'altra su nodi all'interno del cluster.