Riassunto di Docker e Kubernates per il corso di TAASS

Gabriele Naretto

2022 Sessione Gennaio-Febbraio

Contents

1 Docker

1.1 Cos'è Docker ?

Col passare del tempo sempre più aziende hanno avuto il bisogno di portare una propria applicazione/sito web, questo per restare al passo con il mondo

digitale (*No application no business* per citare la prof). Ma per fare questo pero bisogna però avere del personale che lavori sui server e ne faccia manutenzione (Che ovviamente ha un costo non indifferente). In aggiunta a questo si consideri che prima dell'avvento di Docker, il compito di semplificare la gestione del ciclo di vita del software (e in particolare di quelli basati su architetture SOA) è stato affidato alle tecnologie di virtualizzazione. Tuttavia, la pratica ha dimostrato come la virtualizzazione abbia in sé una serie di limitazioni che mal si coniugano con le esigenze delle applicazioni moderne. Se da una parte esse offrono un elevato grado d'isolamento (ogni VM è di fatto un'unità a sé) dall'altra comportano un significativo spreco di risorse di calcolo, dato che ogni VM impegna l'hardware ospitante a eseguire il codice del sistema operativo virtualizzato oltre che delle applicazioni che esegue. Ciò limita fortemente il numero di VM che uno stesso host può ospitare senza degradare le prestazioni generali del sistema. Per risolvere tutti questi problemi è nato Docker.

Prendendo la definizione Amazon AWS *"Docker è una piattaforma software che permette di creare, testare e distribuire applicazioni con la massima rapidità. Docker raccoglie il software in unità standardizzate chiamate container che offrono tutto il necessario per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime. Con Docker, è possibile distribuire e ricalibrare le risorse per un'applicazione in qualsiasi ambiente, tenendo sempre sotto controllo il codice eseguito."*

1.2 Come Funziona Docker ?

Docker funziona grazie a dei contenitori detti "Container" che compongono l'insieme dei dati di cui necessita un'applicazione per essere eseguita: librerie, altri eseguibili, rami del file system, file di configurazione, script, ecc. Il processo di distribuzione di un'applicazione si riduce quindi alla semplice creazione di una Immagine Docker, ovvero di un file contenente tutti i dati dell'applicazione. L'immagine viene utilizzata da Docker per creare un Container, un'istanza dell'immagine che eseguirà l'applicazione in essa contenuta. I container sono quindi autosufficienti perché contengono già tutte le dipendenze dell'applicazione e non richiedono quindi particolari configurazioni sull'host, ma sono soprattutto portabili perché essi sono distribuiti in un formato standard (le immagini appunto), **che può essere letto ed eseguito da qualunque server Docker**. La frase in grassetto è di fondamentale importanza perché è proprio il punto forte di Docker, che voi sviluppiate su Windows, IOS o Linux una colta che la vostra immagine sarà Dockerizzata si potrà usare su qualunque sistema operativo. Inoltre, i container sono leggeri perché sfruttano i servizi offerti dal kernel del sistema operativo ospitante, invece di richiedere l'esecuzione di un kernel virtualizzato come avviene nel caso delle VM. Ciò permette di ospitare un gran numero di container nella stessa macchina fisica.

1.3 Altri concetti utili

- Docker Engine: è il core software capace di gestire immagini, containers, volumes, network.
- Networks: spesso bisogna gestire il network dei container per far si che questi possano per esempio comunicare tra loro nel progetto (molto utile per il progetto `https://docs.docker.com/network/`)
- Docker File: il Dockerfile è un documento di testo che contiene tutti i comandi necessari per creare un'immagine con un'applicazione all'interno
- Volumi: in Docker si possono memorizzare sul dispositivo i dati di alcuni container in modo da non perderli a ogni esecuzione `https://docs.docker.com/storage/volumes/`

1.4 Perché usare Docker ?

I vantaggi visti a lezione:

- I containers forniscono l'applicazione (codice sorgente e tutte le relative dipendenze) sotto forma di un'unica unità standardizzata
- Eliminazione delle inconsistenze derivanti dallo specifico ambiente che si utilizza
- Ambienti di sviluppo più puliti e senza incappare in possibili conflitti/problemi
- In ambito "DevOps" svolge un ruolo di assoluta importanza per ridurre quello che viene definito come il "systems development life cycle"

E vediamo quelli più importanti secondo Amazon AWS

- Distribuzione più rapida del software: Gli utenti di Docker in media distribuiscono software con una frequenza sette volte maggiore rispetto a chi non utilizza Docker. Permette infatti di distribuire servizi isolati con qualsiasi frequenza.
- Funzionamento Standardizzato: L'uso di container per le piccole applicazioni semplifica la distribuzione, l'identificazione dei problemi e il rollback per il ripristino.
- Trasferimenti ottimizzati: Le applicazioni basate su Docker possono essere spostate in modo facile da computer di sviluppo locali a distribuzioni di produzione in AWS.
- Spese ridotte: I container Docker semplificano l'esecuzione di codice sui server, migliorandone i livelli di utilizzo e contribuendo a risparmiare soldi.

1.5 Docker Registry e Docker Repository

Una volta Dockerizata un applicazione bisogna avere la possibilità di condividerla con altri, per fare questo introduciamo dei concetti molto semplici

- Un Docker Registry è un servizio di storage per le immagini docker. E' possibile anche gestirsi privatamente un proprio docker registry attraverso l'utilizzo dell'apposito progetto "docker distribution". Altri esempi di docker registry sono: Google Container, Amazon ECR...
- Docker Hub è il public registry messo a disposizione da Docker per qualunque utente o compagnia con la possibilità di poter gestire un numero illimitato di docker repository pubblici(stessa funzione di GitHub utilizzato però per le immagini docker)
- Un Docker Repository è una collezione d'immagini docker che hanno lo stesso nome ma che si differenziano per il differente tag (ovvero versione dell'applicazione)

Negli spike a lezione la prof scaricherà da Docker Hub l'immagine di un DB, questo procedimento è molto utile perché nel progetto molti programmi esterni potranno essere utilizzati tramite Docker (io per esempio ho scaricato un database relazione e uno non relazionale cosi da non dover installarli sul mio pc)

Lista d'immagini disponibili su docker `https://hub.docker.com/search?type=image`

1.6 Tool Utili

- Compatibile con IntelliJ in cui vedremo una finestra interamente dedicata hai servizi.
- Docker Compose: un file che ci permette di lanciare diverse immagini Docker allo stesso tempo
- Docker Machine: un tool che ci permette di installare il docker engine su un host virtuale e che ci permette di gestire docker con dei comandi
- Kompose (Utile per il progetto): tool per la trasformazione automatica di file Docker compose in file di Deployment e Service per Kubernate

2 Kubernates

2.1 Cos'è Kubernates ?

I container sono un buon modo per distribuire ed eseguire le tue applicazioni. In un ambiente di produzione, è necessario gestire i container che eseguono le applicazioni e garantire che non si verifichino interruzioni dei servizi. Per esempio, se un container si interrompe, è necessario avviare un nuovo container. Non

sarebbe più facile se questo comportamento fosse gestito direttamente da un sistema? È proprio qui che Kubernetes viene in soccorso! Kubernetes ti fornisce un framework per far funzionare i sistemi distribuiti in modo resiliente. Kubernetes si occupa della scalabilità, failover, distribuzione delle tue applicazioni. Quindi Kubernetes è di fatto un **Orchestratore di container (Docker)**.

Noi trattiamo sia Docker che Kubernetes perché sono ottimi per sviluppare ed eseguire Microservizi.

2.2 Come funzione Kubernetes ?

Introduciamo ora tutti i componenti di Kubernetes e vediamo cosa fanno.

2.2.1 Cluster

TUTTO É CONTENUTO NEL CLUSTER

Facendo il deployment di Kubernetes, ottieni un cluster (per esempio all'avvio di Minikube si crea il cluster). Un cluster Kubernetes è un insieme di macchine, chiamate nodi, che eseguono container gestiti da Kubernetes. Un cluster ha almeno un Worker Node.

2.2.2 Worker Node

I NODI CHE SONO IN ESECUZIONE + CONTENUTI NEL CLUSTER

Il/I Worker Node ospitano i Pod che eseguono i workload dell'utente. Il/I Control Plane Node gestiscono i Worker Node e tutto quanto accade all'interno del cluster. Per garantire la high-availability e la possibilità di failover del cluster, vengono utilizzati più Control Plane Node.

2.2.3 Kubelet

CONTENUTO NEI NODI

Kubelet é un agente che è eseguito su ogni nodo del cluster. Si assicura che i container siano eseguiti in un pod. La kubelet riceve un set di PodSpecs che vengono forniti attraverso vari meccanismi, e si assicura che i container descritti in questi PodSpecs funzionino correttamente e siano sani. La kubelet non gestisce i container che non sono stati creati da Kubernetes.

2.2.4 Kube-Proxy

CONTENUTO NEI NODI

kube-proxy è un proxy eseguito su ogni nodo del cluster, responsabile della gestione dei Kubernetes Service. I kube-proxy mantengono le regole di networking sui nodi. Queste regole permettono la comunicazione verso gli altri nodi del cluster o l'esterno. Il kube-proxy usa le librerie del sistema operativo quando possible; in caso contrario il kube-proxy gestisce il traffico direttamente.

2.2.5 Pod

CONTENUTO NEI NODI + CONTIENE I CONTAINER

I Pod sono i più piccoli e basilari oggetti deployabili in Kubernetes. Un Pod rappresenta una singola istanza di un processo in esecuzione nel tuo cluster. I pod contengono uno o più container, come i container Docker. Quando un

Pod esegue più container, i container sono gestiti come una singola entità e condividono le risorse del Pod. Generalmente, l'esecuzione di più container in un singolo Pod è un caso d'uso avanzato. I pod contengono anche risorse di rete e di storage condivise per i loro contenitori:

- Network: Ai pod vengono assegnati automaticamente indirizzi IP unici. I container Pod condividono lo stesso spazio dei nomi di rete, compreso l'indirizzo IP e le porte di rete. I container in un Pod comunicano tra loro all'interno del Pod su localhost.
- Storage: I pod possono specificare un insieme di volumi di storage condivisi che possono essere condivisi tra i container.

Si può considerare un Pod come un "logical host" autonomo e isolato che contiene le esigenze sistemiche dell'applicazione che serve. Quando esegui "kubectl get pod" per ispezionare un Pod in esecuzione sul tuo cluster, un Pod può essere in una delle seguenti possibili fasi:

- Pending:Il Pod è stato creato e accettato dal cluster, ma uno o più dei suoi contenitori non sono ancora in esecuzione. Questa fase include il tempo trascorso per scaricare le immagini.
- Running: Il Pod è stato legato a un nodo e tutti i container sono stati creati. Almeno un container è in esecuzione, sta per partire o si sta riavviando.
- Succeeded: Tutti i container nel Pod sono terminati con successo. I Pod terminati non si riavviano.
- Failed: Tutti i contenitori nel Pod sono terminati, e almeno un container è terminato in fallimento. Un contenitore "fallisce" se esce con uno stato diverso da zero.
- Unknown: Lo stato del Pod non può essere determinato.

2.2.6 Control Plane

IL MANAGER DEL CLUSTER + CONTESTO NEL CLUSTER

I componenti del Control Plane sono responsabili di tutte le decisioni globali sul cluster (ad esempio, lo scheduling) oltre che a rilevare e rispondere agli eventi del cluster (ad esempio, l'avvio di un nuovo pod quando il valore replicas di un deployment non è soddisfatto). Inoltre i componenti della Control Plane possono essere eseguiti su qualsiasi nodo (che li rende Control Plane Node) del cluster stesso. Solitamente, per semplicità, gli script d'installazione tendono a eseguire tutti i componenti della Control Plane sulla stessa macchina, separando la Control Plane dai workload dell'utente

2.2.7 Api-server

Contenuto nel Manager

L'API server è un componente di Kubernetes control plane che espone le Kubernetes API. L'API server è il front end del control plane di Kubernetes. La principale implementazione di un server Kubernetes API è kube-apiserver. kube-apiserver è progettato per scalare orizzontalmente, cioè scala aumentando il numero d'istanze. Puoi eseguire multiple istanze di kube-apiserver e bilanciare il traffico tra queste istanze.

2.2.8 kube-scheduler

Contenuto nel Manager

Componente della Control Plane che controlla i pod appena creati che non hanno un nodo assegnato, e dopo averlo identificato glielo assegna. I fattori presi in considerazioni nell'individuare un nodo a cui assegnare l'esecuzione di un Pod includono la richiesta di risorse del Pod stesso e degli altri workload presenti nel sistema, i vincoli delle hardware/software/policy, le indicazioni di affinity e di anti-affinity, requisiti relativi alla disponibilità di dati/Volumes, le interferenze tra diversi workload e le scadenze.

2.2.9 kube-controller-manager

Contenuto nel Manager

Componente della Control Plane che gestisce controllers. Da un punto di vista logico, ogni controller è un processo separato, ma per ridurre la complessità, tutti i principali controller di Kubernetes vengono raggruppati in un unico container ed eseguiti in un singolo processo. Alcuni esempi di controller gestiti dal kube-controller-manager sono:

- Node Controller: Responsabile del monitoraggio dei nodi del cluster, e.g. della gestione delle azioni da eseguire quando un nodo diventa non disponibile
- Replication Controller: Responsabile per il mantenimento del corretto numero di Pod per ogni ReplicaSet presente nel sistema
- Endpoints Controller: Popola gli oggetti Endpoints (cioè, mette in relazioni i Pods con i Services).
- Service Account e Token Controllers: Creano gli account di default e i token di accesso alle API per i nuovi namespaces.

2.2.10 etcd

Contenuto nel Manager

È un database key-value ridondato, che è usato da Kubernetes per salvare tutte le informazioni del cluster. Se il tuo cluster utilizza etcd per salvare le informazioni, assicurati di avere una strategia di backup per questi dati. Puoi trovare informazioni dettagliate su etcd sulla documentazione ufficiale.

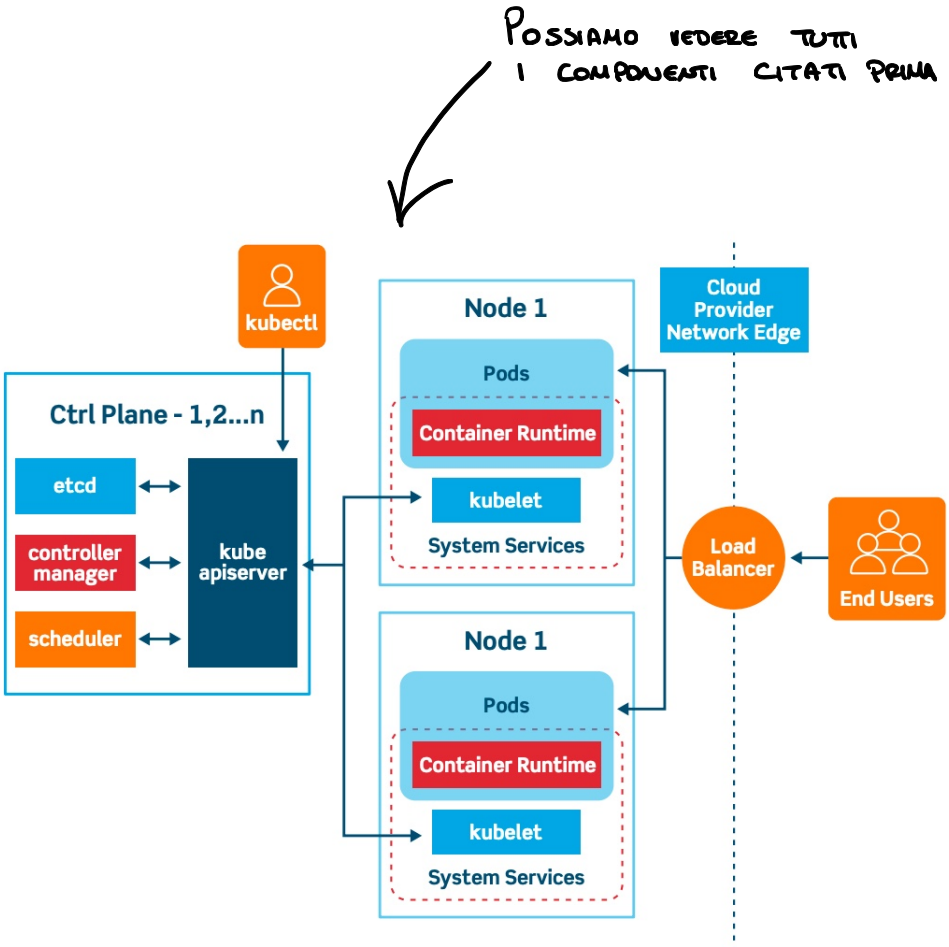

Figure 1: Struttura Docker

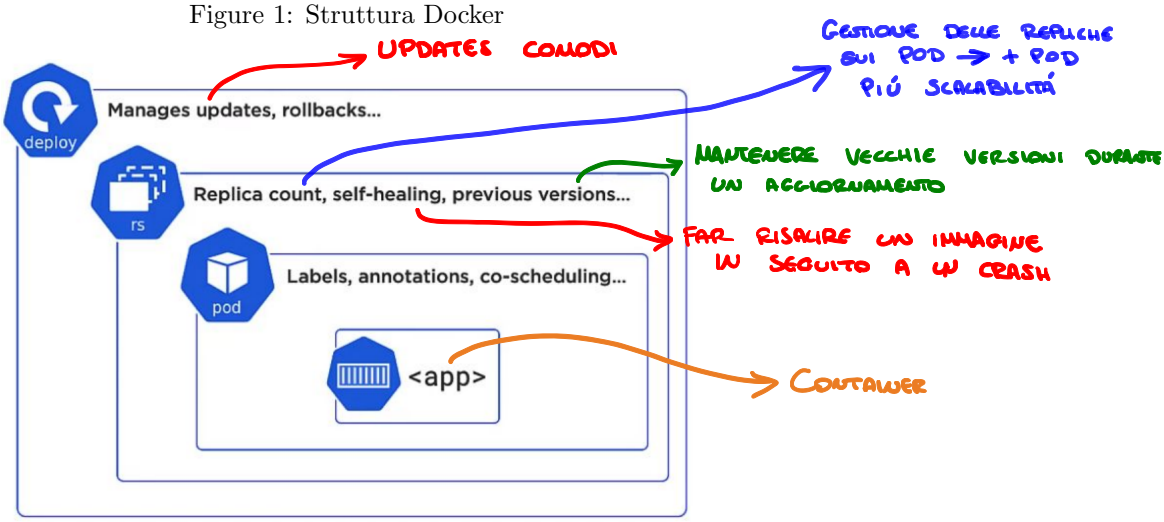

Figure 2: Funzionamento Generale

2.2.11 Riassuntino easy

Quindi in parole povere, al lancio di Kubernetes si crea il Cluster, all'interno del Cluster c'é il Master (Control Plane) che gestisce i nodi(Worker Node), Ogni nodo gestisce un pod che contiene il Container Docker.

E perché tutto questo è bello ? Perché ci permette di lanciare in tranquillità i nostri container Docker senza doverci preoccupare della loro gestione (che verrà specificata grazie hai file di service e deployment che lanceremo), in automatico Kubernetes li rilancerà e gestirà.

2.3 Consigli utili

- Seguite la guida di Kubernetes per farlo funzionare, funziona su tutti i sistemi operativi (si anche WINDOWS, alla faccia vostra possessori di Linux) `https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/`
- in Docker Desktop nelle impostazioni ce una parte dedicata a Kubernetes che ci permette di lanciarlo sul nostro dispositivo da Docker (fidatevi che è stra comodo) PS ricordatevi d'installare kubectl
- Sfruttate il software Kompose se avete un mega Docker compose e volete lanciare il tutto su Kubernetes `https://kompose.io/`