SWARM

Lorenzo Valentini

05/28/2017

Indice

Informazioni introduttive

Relazione di tirocinio:
All'interno di questo documento cercherò di rispondere ad alcune domande riguardanti la tecnologia offerta da Swarm in Docker come strumento per l'orchestrazione di servizi.

Obiettivo

- Come vengono scalati i servizi in Docker Swarm?
- Si può scalare solo orizzontalmente o verticalmente o in entrambe le modalità?
- Nodi di un cluster di macchine in rete può fare parte di più sciami "swarms"?

!! ATTENZIONE:
Verso la fine di aprile 2017 sono stati fissati molti bug presenti in Docker e nei suoi componenti, inoltre, sono stati aggiunti alcuni tool che prima erano disponibili ma non preinstallati. Al momento della mia sperimentazione questi strumenti non erano ancora incorporati. Nella prova da me effettuata utilizzo la versione 1.12 per il set di macchine virtuali utilizzate per simulare il mio cluster. Quella versione include già al suo interno lo SwarmKit ma non ancora tutta la toolbox che di recente è stata resa embedded con tutti i tool per l'automazione e la gestione del cluster e dei servizi es. (docker-machine, docker-compose). Per maggiori informazioni guardare la pagina ufficiale, qui.

Capitolo 1

Introduzione su Swarm e concetti di base

1.1 Che cos'è Swarm?

Swarm è un cluster manager presente in Docker engine dalla versione 1.12 e successive all'interno dello SwarmKit. Come funziona, i sistemi in cui eseguiamo Docker candidati a partecipare ad un cluster di macchine (ad uno "swarm") potranno farlo solo se in modalità Swarm utilizzando i comandi es. (`docker swarm <command_name>` mediante la tipica sintassi per richiamare tools interni a docker). Una macchina per poter partecipare a uno Swarm("sciame") ha due possibilità inizializzarne uno nuovo oppure unirsi ad uno già esistente.
Swarm quindi non è altro che un insieme di macchine Docker engine "nodi" interconnessi dove si possono distribuire dei servizi. Per effettuare la gestione sono messe a disposizione (Docker engine CLI) e (API) che includono al loro interno comandi per effettuare la gestione dei nodi come ad es.(add, remove, init nodes), e per gestire i servizi in esecuzione su di essi.
Quindi possiamo dire che quando usiamo Docker al di fuori della modalità Swarm, i nostri "contenitori"(container) verranno eseguiti con i comandi normali di Docker.

Quando invece eseguiamo Docker in modalità uno Swarm i "contenitori" di immagini saranno gestiti come servizi. Questo significa che io posso eseguire un servizio Swarm o un container("stanalone Docker container") partendo da una medesima istanza "immagine" di partenza.
Portiamo particolarmente attenzione a cosa si intende per "nodo" di uno Swarm, potrebbe essere il chiarimento che stiamo cercando alle domande inizialmente poste.

1.2 Che cos'è un nodo?

Un **nodo** può essere inteso come una singola macchina su cui è in esecuzione un'istanza di Docker engine ogni nodo può far parte di uno "swarm". Ogni macchina la si può immaginare come un "nodo Docker". Si possono eseguire uno o più nodi sopra un singolo computer fisico o all'interno di cloud server, ma un tipico risultato di una semplice distribuzione con Swarm, potrebbe corrisponde a più nodi Docker distribuiti su diversi computer fisici o su cloud in comunicazione tra loro tramite una rete di interconnessione. Tale rete mette in comunicazione i vari nodi di uno swarm sfruttando il paradigma master-worker.
La distribuzione delle applicazioni con Swarm, la si effettua definendo un task servizio e assegnandolo ad un nodo in particolare quello avente funzionalità di manager. Il nodo manager (**manager node**) a questo punto ha il compito di occuparsi della distribuzione delle unità di lavoro chiamate **tasks** ai nodi lavoratori (**worker nodes**).
I nodi manager hanno anche il compito di occuparsi dell'orchestrazione e della gestione delle funzionalità richieste allo "swarm" per lo svolgimento del proprio compito e il mantenimento dello stato. Tra i vari nodi aventi il ruolo di manager ne esiste solo uno eletto come Leader incaricato di condurre l'orchestrazione dei tasks.
I nodi lavoratori ricevono ed eseguono i tasks(compiti) spediti dal nodo manager. Di default anche il nodo manager oltre ad eseguire le ordinarie ope-

razioni di gestione dello sciame esegue anch'esso i servizi come se fosse un semplice "worker nodes". Successivamente questa configurazione può essere modificata in modo che esegua esclusivamente con funzionalità di manager. Un "agente" che non è altro che un servizio in esecuzione in background nello swarm, che per tutti i nodi lavoratori a cui sono stati assegnati dei tasks produce un report informativo in cui tiene traccia dello stato. I worker node notificano continuamente al nodo manager lo stato dei tasks a loro assegnati così che il manager possa operare per mantenere lo stato desiderato per tutti i nodi del sistema questa operazione è detta **load balancing**.

1.3 Servizi e tasks

Un **servizio** in esecuzione su un nodo "worker" è definito come **task**. Questi rappresentano l'unità minima di lavoro svolta da ogni nodo del sistema. I **tasks** consentono di soddisfare richieste multiple degli utenti che utilizzano quel determinato servizio. Quando si crea un servizio, la prima cosa che si deve fare è specificare da quale immagine di base si vuole partire a creare il container Docker e quali comandi lanciare al suo interno.
Dopo aver creato un nuovo servizio o avendone uno già precedentemente generato la fase di replicazione e distribuzione è a carico del nodo Manager, che serve i vari nodi Worker con i tasks in modo da mantenere lo stato della distribuzione del servizio come specificato. Nel caso di servizi GLOBALI, lo swarm eseguirà il task di quel servizio su di un qualsiasi nodo disponibile nel cluster. Quindi riassumendo: un task al suo interno contiene un container Docker più i comandi necessari all'esecuzione di quest'ultimo che rappresenta l'unità minima di lavoro atomica e indivisibile. Il manager scala i compiti ai nodi worker in accordo alle politiche di scheduling definite dall'amministratore di sistema. Un task una volta assegnato ad un nodo, non può essere spostato in un altro nodo può solamente eseguire con successo oppure fallire.

1.4 Load balancing

Il nodo manager, in quel momento candidato alla gestione dello sciame, usa il traffico in ingresso per bilanciare la distribuzione del servizio reso accessibile dall'esterno. Il manager, inoltre, se non specificata può assegnare automaticamente una porta di rete ad ogni servizio aggiunto o in alternativa è possibile specificare una porta sulla quale esporre il servizio, in tal caso è sufficiente in fase di creazione del servizio assegnare mediante l'utilizzo di un flag passato al comando una porta non usata tra quelle disponibili. Nel caso noi lasciassimo la gestione delle porte al nodo manager, sarà lui a provvedere all'assegnamento sciegliendo tra una di queste nel range dalla 30000 alla 32767.
Componenti esterni, per lo scheduling del carico possono anch'essi accedere al servizio tramite la sua interfaccia pubblica resa accessibile all'indirizzo e sulla porta esposta per ogni nodo del cluster, indipendentemente dal fatto che il nodo in quel momento stia eseguendo il servizio o meno. Titti i nodi dello sciame che permettono connessioni in ingresso eseguono un istanza di un task.
Inoltre uno Swarm ha un componente DNS interno che assegna automaticamente ad ogni servizio nello swarm un proprio nome simbolico interno per distinguere tra loro i vari tasks presenti nello swarm. Il sistema interno di bilanciamento del carico distribuisce le richieste dei vari servizi all'interno del cluster sui vari nodi in base alle richieste utilizzando il nome simbolico DNS del servizio.

Capitolo 2

Esempio con Docker Swarm

La prova svolta aveva lo scopo di utilizzare lo strumento di container scheduling Docker Swarm, per osservarne e studiarne il funzionamento in un tipico scenario applicativo. Ad esempio, in questo caso, ho realizzato un semplice servizio web incapsulato mediante la tecnologia dei container offerta da Docker. Questo è stato successivamente scalato su un cluster di macchine virtuali connesse e gestite tramite lo strumento di orchestrazione Swarm. Sempre mediante più VM sono stati emulati dei Client, necessari per effettuare le richieste GET che emulavano il traffico in entrata al Cloud Server. Il cluster di macchine server anch'esse virtualizzate con delle VM come detto in precedenza invece ospitavano il servizi. I servizi sono gestiti e incapsulati come tasks mediante la tecnologia dei container. Cercherò ora di fare in modo chiaro e conciso un breve riassunto di questa sperimentazione pratica. In questo documento non saranno appositamente affrontati tutti i dettagli dei passaggi e le varie fasi necessarie alla preparazione delle macchine. È importante tenere a mente che per la riuscita della dimostrazione sono elementi strettamente fondamentali. Per semplicità suddividerò l'esperimento in fasi:

1. Inizialmente, è stato necessario incorporare l'immagine di un web-server all'interno di un container Docker in modo da ottenere con una semplice operazione di DockerFile scripting il nostro container che incapsula al suo interno web server e i comandi necessari successivamente per la

sua esecuzione. Il container da me costruito quindi si comporrà al suo interno dei seguenti layer: la base di partenza è un'immagine Linux Alpine minimale. Il sever sul quale girerà il nostro applicativo server-side è node.js sul quale farò girare un'applicazione java script app.js che rappresenterà in definitiva il servizio che renderemo disponibile all'esterno. Per comodità ho deciso di renderlo raggiungibile sulle porte 9111:9111.

2. Successivamente, è stato necessario creare un cluster di VM con Virtual Box per simulare la piattaforma di hosting. Una volta configurate e settate le comunicazioni di rete delle varie macchine, ho provveduto all'installazione di Docker » v.1.12 su ognuna di queste. Le VM da me create per l'occasione erano 4 macchine aventi: due 1Gb di RAM e con l'utilizzazione di 1 core della cpu, mentre, le restanti due avevano 512MB di RAM e 1 core a disposizione. Interconnesse tra di loro mediante interfaccia di rete bridge-virtualBox. Il SO di base installato su ogni nodo è la versione mate di Linux Mint 18.0.

3. Creazione e inizializzazione.
   Installato e configurato il sistema è ora possibile creare lo Swarm su cui andremo a distribuire il servizio. Per chi non lo conoscesse Swarm è la soluzione integrata di Docker per la gestione di un cluster in grado di eseguire container o come li definisce Swarm stesso, servizi; un servizio è formato da uno o più container, espone una porta, ed è in grado di scalare facilmente in termini di numero di container. L'intero strato di bilanciamento detto network, nella sua configurazione più semplice, è interamente gestito da Docker Swarm stesso e permette di astrarre il servizio dai container reali che lo fanno girare. Swarm richiede almeno 2 nodi (un manager ed un worker) per funzionare, anche se per motivi di ridondanza è sempre meglio avere almeno un paio di worker e due o più manager.

Con il comando:
`docker swarm init -advertise-addr <IP of my server leader>`
Creo e inizializzo il mio primo nodo master e con semplicità.

```
$ docker swarm init --advertise-addr 10.0.0.3
Swarm initialized:\
current node (jkr2vnv7ntyj0p0zgmdj5t17c) is now a manager.

To add a worker to this swarm, run the following command:

docker swarm join \
--token SWMTKN-1-06hpb763v16zbsplkmbw9yp3uyt5wwti5 \
10.0.0.3:2377

To add a manager to this swarm, run 'docker swarm join-token manager' an
```

Il comando ci fornisce anche in output già i comandi per fare join di nuovi worker al cluster. In questo modo aggiungerò due nodi worker e un ulteriore nodo master in modo da avere tutte e 4 le macchine attive e pronte all'uso.
Il nostro cluster è quindi composto da:

- node1, node2 in funzione di manager.
- node2 e node3 come worker.

Alcuni comandi messi a disposizione per verificare lo stato e l'attuale distribuzione del carico di lavoro sono:
`docker info`
`docker ps`

```
docker node ls
docker node ps <nome del servizio>
```

4. Distribuzione del servizio:
   Seppur possa sembrare strano, avere più manager che worker. Inoltre è da notare che per i cluster Swarm i manager fungono anche da worker. Il che vuol dire: attualmente abbiamo un cluster in grado di supportare 4 container in parallelo per l'esecuzione di un singolo servizio (potrebbero essere anche di più, ma si andrebbe a non avere più parallelismo). L'unica differenza, dunque, tra un worker ed un manager è che sui manager è possibile eseguire i comandi di gestione del cluster e di deploy dei servizi. Su quale worker poi i servizi andranno in esecuzione è a discrezione di Swarm a noi non è dato a priori sapere quale criterio di scheduling lo sciame userà per sciegliere i nodi su cui verranno sclati i tasks.
   Quale modo migliore di testare un cluster Swarm se non creare un servizio che eroghi qualcosa? A questo punto useremo come base di partenza il servizio che abbiamo creato inizialmente nella prima fase useremo il nostro servizio web. Il comando da usare per creare un servizio:
   `docker service create -name myServiceName -replicas 3 -p 9111:9111 node:app0.1`
   I parametri sono molto semplicemente i seguenti:
   `-name`: Il nome che vogliamo dare al servizio. Nel nostro caso lo abbiamo chiamato 'myServiceName'
   `-replicas`: Il numero di repliche (container) che vogliamo deployare. Partiamo con 3
   `-p`: Il mapping delle porte. Node parte di default erogando la porta 9111, noi vogliamo accedervi usando la porta 9111, faremo quindi questo mapping 9111:9111 per maggior chiarezza da ora in poi usero le porte standard per un web server 8080:80
   `node:app0.1`: Il nome del container su cui si basa il servizio

Come conseguenza al comando avremo quindi generato 3 container Linux Alpine (su che nodi al momento non lo sappiamo) che erogano il servizio '`node:app0.1`' sulla porta 8080. Questo significa che contattando la porta 8080 del nostro cluster andremo ad utilizzare uno di questi container. Le logiche di bilanciamento, al momento, sono tutte in mano a Docker Swarm.

5. Scaliamo il servizio:
   I servizi in Docker Swarm possono essere scalati molto velocemente come vedremo nell'esempio sottostante.
   In pratica abbiamo detto a Docker di scalare il servizio `node:app0.1` fino ad avere 20 repliche, in poche parole abbiamo aggiunto 17 tasks ai 3 già presenti. Tutta la loro configurazione di rete e il bilanciamento della network viene fatta in automatico da docker:
   `docker node ls` tutti i servizi in esecuzione
   `docker node ps node:app0.1` vediamo ogni singolo tasks di quel servizio su quale nodo del cluster è in esecuzione con il rispettivo stato.

6. Contattiamo il server: Infine aprendo il browser e contattando il server web all'indirizzo e la porta su cui esso è in ascolto ci risponderà con un oggetto json dove è contenuto un id univoco per l'oggetto e l'Id del container che ha eseguito la richiesta. Quindi infine per vedere quali container e come rispondevano alle richieste è stato sufficiente fare uno script bash che tramite un ciclo for e sfruttando il comando bash curl effettua un numero r di richieste salvando gli oggetti json ricevuti in risposta in un file sucessivamente ispezionabile.

```
#!bin/bash
docker node ps node:app0.1 >> result;
for i in ‘seq 1 20‘;
do
  curl -i -X GET "http://localhost:9111/guid" >> result;
  #echo -e "\n" >> result;
```

```
done;
exit 0;
```

concetti di base di swarm servizio ecc. https://docs.docker.com/glossary/

2.1 Deploy service

Originariamente per effettuare il deploy di servizi si richiedeva necessariamente l'installazione Docker e Swarm come due componenti software distinti su ogni macchina candidata a far parte di un determinato gruppo di lavoro. Poichè questi due sistemi software svolgono compiti differenti ed originariamente erano distribuiti separatamente. Questo portava ad inevitabili complicazioni nella fase di start-up dovute principalmente all'overhead solo della fase di configurazione, rendendo eccessivamente onerosa questa fase preparatoria del lavoro. Ma grazie alla notorietà acquisita da queste tecnologie e dalla continua espansione della cerchia di sviluppatori e utilizzatori, ha fatto si che guidati dalla necessità si sviluppassero dei tools aggiuntivi per migliorare, velocizzare e semplificare l'utilizzo di questi strumenti. Attualmente alcuni di essi fanno a loro volta parte degli strumenti interni offerti da Docker. Ad esempio tool come docker-compose o docker-machine entrambi per automatizzare le fasi di build delle VM e il deploy di servizi e applicazioni. Per ulteriori informazioni in maggior dettaglio consultare la documentazione ufficiale Compose e Machine