Dockerizziamo OpenWhisk: docker run Part 3/3

Apache OpenWhisk è un sistema Serverless Open Source molto flessibile, sviluppato come progetto Open Source alla Apache Software Foundation. Abbiamo già visto nelle prime due parti come compilarlo e come creare una immagine Docker per lanciarlo. In questa terza parte concludiamo spiegando come lanciarlo, che richiede un certo numero di opzioni da specificare.

# Lanciare l'immagine Docker

Siamo pronti per lanciare la nostra immagine Docker, ma dobbiamo ricordarci subito che generalmente le immagini richiedono un certo numero di parametri per funzionare

Quindi se facciamo semplicemente:

docker run sciabarracom/openwhisk-standalone:2020-07-01

non funzionerà! Ecco una breve lista di informazioni che dobbiamo passare all'immagine (e che saranno essenziali per scrivere la configurazione per eseguirlo su Kubernetes):

* Dare un nome e un hostname all’immagine
* Specificare Reti e Porte da utilizzare
* Montare volumi
* Specificare Variabili di ambiente
* Eventualmente passare argomenti alla riga di comando

Utilizzeremo queste informazioni per scrivere uno script di avvio.

## Diamo un nome al nostro container

Quando lanciamo una immagine, con il comando *docker run* viene creato un container a cui viene dato un nome generato casualmente tipo *nice\_nightingale* o giù di lì. Per quanto pittoresco, questo nome di solito non è molto utile quindi è meglio poterlo specificare direttamente.

Inoltre è opportuno dare un "hostname" al container, in quanto questo nome viene utilizzato per "riferirlo" da altri container.Quindi utilizzeremo qualcosa come:

--name openwhisk –h openwhisk --rm

Notare che abbiamo aggiunto il flag –*rm,* che serve per rimuovere il container automaticamente quando viene stoppato. Questo perché OpenWhisk non ha bisogno di preservare nulla (userei un volume in quel caso) ma se provassi a lanciarlo nuovamente specificando un nome fisso otterrei errore in quanto esiste già (dovrei invece ristartarlo in quel caso).

## Impostare la porta e il network

Passiamo adesso al network e alle porte esposte. Occorre sapere che ogni container è eseguito in un network "virtuale" che non è lo stesso in cui gli altri programmi del vostro computer sono eseguiti. Questo fatto per motivi di protezione e isolamento. Quindi, see volete accedere al container dovete attaccarlo al network usato dal vostro computer. Questo network è reso accessibile tramite una rete interna di docker che chiama *bridge*; è appunto un bridge in senso TCP/IP che collega il network della vostra macchina al network interno di Docker.

Poi OpenWhisk standalone richiede due porte TCP: 3232 e 3233. La prima viene usata accedervi con il comando *wsk* con una API, e la seconda per il playground

Quindi occorre specificare, dopo *run* e prima del nome dell'immagine i parametri:

--network bridge -p 3822:3288 –p 3289:3828

Se ci dimentichiamo di specificare questi parametri semplicemente non potremo accedere al servizio che è eseguito dentro il container

## Montare i volumi

Creare una immagine è paragonabile a creare il disco iniziale di una macchina virtuale. Quando viene lanciata, si crea un container, che è un filesystem temporaneo che contiene l'immagine ma è scrivibile. L'applicazione dentro il container può utilizzare questo filesystem. Però è un filesystem temporaneo che viene immediatamente distrutto quando il container viene eliminato.

Se si vogliono preservare dei dati (per esempio i file di database) occorre fornire all'immagine un filesystem non transiente che non viene eliminato. I volumi sono questo. Si utilizza il parametro –v per "montare" un volume in un container. I volumi sono utilizzati anche per montare directory o singoli file, anche speciali (come i socket).

Nel nostro caso, OpenWhisk standalone non ha bisogno di volumi ma ha bisogno di leggere e scrivere in un file speciale: il socket di comunicazione con Docker. Per questo motivo dovremo montarlo come volume. Infatti se lanciamo la nostra immagine senza questo parametro otteniamo subito un errore:

/usr/bin/docker: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?.

La soluzione è aggiungere il parametro:

-v /var/run/docker.sock:/var/run/docker.sock

**Attenzione**: se lanciate il comando da windows dovete raddoppiare il primo slash (*-v //var*)

## Variabili di Ambiente

Un altro importante aspetto da considerare sono le variabili di ambiente che sono il modo privilegiato con cui si passano parametri alle applicazioni sono le variabili di ambiente. In particolare qui dobbiamo specificare la variabile di ambiente *HOST\_EXTERNAL* che è la variabile usata per indicare il nome dell'host pubblico. Nel nostro caso è localhost (lo useremo solo localmente per ora) quindi il parametro da aggiungere è:

-e HOST\_EXTERNAL=localhost

# Conclusioni

Se non avete la pazienza di digitare i vari comandi presentati, potete trovare il Dockerfile giá pronto qui:

https://github.com/openwhisk-blog/learning-kubernetes

Il dockerfile e gli script di supporto sono nella directory *openwhisk-docker-image,* potete eseguire la build con il comando:

docker build . -t sciabarracom/openwhisk-standalone:2020-07-01

Ci mette un po' ma alla fine avrete la vostra immagine docker fatta in casa! Potete comunque risparmiarvi la fatica di fare la build semplicemente utilizzando l’immagine che ho pubblicato su Docker Hub per questo articolo come: *sciabarracom/openwhisk-standalone:2020-07-01.*

Per lanciarla dovrete usare il comando con tutti i parametri che ho descritto prima, e che riepilogandoli sono i seguente:

docker run -ti \

-h openwhisk --name openwhisk --rm \

--network bridge -p 3232:3232 -p 3233:3233 \

-v /var/run/docker.sock:/var/run/docker.sock \

-e HOST\_EXTERNAL=localhost \  
sciabarracom/openwhisk-standalone:2020-07-01

In pratica basta che copiate e incollate questo comando nel prompt e… dovreste avere OpenWhisk locale up-and running senza doverlo ricompilare da zero. A questo punto potete creare funzioni serverless usando *wsk* come descritto nel primo articolo.