Dockerizziamo OpenWhisk: Dockerfile Part 2/3

Apache OpenWhisk è un sistema Serverless Open Source molto flessibile, sviluppato come progetto Open Source alla Apache Software Foundation. Abbiamo già visto il mese scorso come si compila la versione standalone in locale. In questo articolo impariamo ad impacchettarlo in una immagine Docker scrivendo un apposito Dockerfile

# Creare l'immagine Docker di OpenWhisk

Adesso che sappiamo come compilare OpenWhisk in locale è il momento di automatizzare il tutto scrivendo un *Dockerfile.*

Un Dockerfile è un descrittore che Docker utilizza per creare le sue immagini e contiene generalmente tutti i comandi di compilazione necessari più eventuali direttive sempre relativa alla creazione di immagini. Vediamo passo passo come scrivere un Dockerfile per OpenWhisk standalone.

La prima direttiva che vedremo è *FROM* che sostanzialmente indica una immagine base che possiamo usare come punto di partenza. Poiché OpenWhisk è fatto in Scala iniziamo proprio con

FROM scala as builder

WORKDIR

Notare che abbiamo dato un nome all'immagine, perché la utilizzeremo solo come area temporanea per la compilazione e ci serve il nome per riferirla dopo. E notare che abbiamo impostato come directory principale la radice.

A questo punto ci preoccupiamo di aggiungere un po' di componenti che si serviranno per la compilazione. Java (che è usato da scala) è già presente nell'immagine scala, ma mancano *git* (che serve per scaricare i sorgenti) e *npm* più *nodejs* (che servono per compilare il frontend), quindi li aggiungiamo usando la direttiva *RUN:*

RUN apt-get update && apt-get install -y git nodejs npm

L'immagine di scala è basata, come moltissime altre immagini docker, su Linux Debian. Quindi utilizziamo *apt-get install* che è il comando di questa distribuzione per installare pacchetti. Notare che prima di installare dobbiamo precedere il comando con una *apt-get* *update* per aggiornare gli indici per scaricare i pacchetti.

Ci servono anche degli altri componenti che non sono disponibili nel repository ufficiale di Debian. In particolare, ci serve una specifica versione di Docker compilata in modalità statica e la riga di comando di OpenWhisk. Aggiungiamo questi componenti con dei download diretti dei pacchetti binari in questione:

RUN curl [https://download.docker.com/linux/static/stable/x86\_64/docker-18.06.3-ce.tgz](https://download.docker.com/linux/static/stable/aarch64/docker-18.06.3-ce.tgz) | tar xzvf -

RUN curl -sL <https://github.com/apache/openwhisk-cli/releases/download/1.2.0/OpenWhisk_CLI-1.2.0-linux-amd64.tgz> | tar xzvf -

A questo punto possiamo scaricare e compilare: "cloniamo" i sorgenti da GitHub, cambiamo alla directory di lavoro e compiliamo il tutto:

RUN git clone <https://github.com/apache/openwhisk>

RUN cd openwhisk && ./gradlew :core:standalone:build

Ora l'immagine è più o meno pronta, ma non vogliamo portarci dietro tutti i sorgenti e gli artefatti della compilazione. Per questo motivo usiamo la feature di Docker che permette di fare delle build, e poi usare una altra immagine come base per produrre il risultato finale, estraendo dalla prima immagine i file che servono.

Quindi facciamo esattamente questo: aggiungiamo al Dockerfile le direttive per "ripartire" da capo con l'immagine scala, aggiorniamo i pacchetti all'ultima versione e poi ci mettiamo dentro solo quello che serve.

FROM scala

RUN apt-get update && apt-get -y upgrade

COPY --from=builder /openwhisk/bin/openwhisk-standalone.jar /lib/openwhisk-standalone.jar

COPY --from=builder /docker/docker /usr/bin/docker

COPY --from=builder /wsk /usr/bin/wsk

Tutto pronto per "rilasciare". Impostiamo ora la directory principale (usiamo */home*) ed il comando di avvio (spiegato nel prossimo paragrafo).

ADD start.sh /usr/bin/start.sh

WORKDIR /homem

ENTRYPOINT ["/usr/bin/bash", "/usr/bin/start.sh"]

È importante notare che utilizzando *ENTRYPOINT* questo comando viene sempre eseguito quando lanceremo l'immagine con *docker run <immagine> [<parametri>]*. I parametri, come vedremo, verranno propagati al runtime stesso e ci saranno utili in seguito per collegare altri componenti.

## Lo script di avvio

OpenWhisk ha parecchi parametri, che useremo intensamente nei prossimi articoli. Questi parametri possono essere passati come Java System Property, o con variabili di ambiente o con file di configurazione. AI nostri fini, per l'esecuzione di OpenWhisk standalone in una singola immagine docker occorre settare 3 properties:

* Il nome dell'host all'interno dell'immagine docker
* L'indirizzo IP dove il runtime deve stare in ascolto
* L'indirizzo esterno che si usa per connettersi

I primi due possono essere ricavati da un comando Linux: *$(hostname)* e *$(hostname –i)*. Il terzo dipende da una configurazione e quindi va passata con una variabile di ambiente.

Infine, dato che all'interno di una immagine Docker non possiamo lanciare il browser, dovremo chiedere a OpenWhisk di farlo, con il parametro *–no-browser*. Tradotto in codice questo diventa il seguente scritpt *start.sh*:

java $EXTRA\_JVM\_ARGS\

-Dwhisk.standalone.host.name="$(hostname)"\

-Dwhisk.standalone.host.internal="$(hostname -i)"\

-Dwhisk.standalone.host.external="$HOST\_EXTERNAL" \

-jar /usr/lib/openwhisk-standalone.jar --no-browser "$@"

Non useremo gli *EXTRA\_JVM\_ARGS* per ora ma abbiamo lasciato un modo per poterli specificare in seguito se servissero (e serviranno…).

# Conclusioni

Abbiamo visto finora come compilare OpenWhisk (in versione standalone) e poi come usare le istruzioni di compilazione per creare una immagine Docker. Nella prossima parte, terza ed ultima, vedremo come chiudere il cerchio lanciando l’immagine Docker che abbiamo costruito che, come vedremo, non è banale.