Imparare Kubernetes con OpenWhisk: compilazione e immagine docker

Apache OpenWhisk è un sistema Serverless Open Source molto flessibile. È una tipica applicazione distribuita e puo quindi essere installato in Kubernetes come sistema distribuito. Utilizzare Kafka come coda di messaggi e CouchDB come database per mantenere configurazioni e attivazioni.

Tutti i componenti sono replicabili e scalabili e si può creare un ambiente con centinaia di nodi. Per queto motivo è un ambiente ideale da installare per apprendere come funziona Kubernetes in ambienti realisitici. Ho deciso pertanto di scrivere una serie di articolo su come si installa OpenWhisk in Kubernetes allo scopo di presenare un tutorial su Kubernetes.

Non copriró in questa serie invece come funziona OpenWhisk stesso come sistema serverless, anche se è indubbiamente un argomento anch'esso interessante. Delego a questo proposito la lettura del libro in inglese "Learning Apache OpenWhisk", pubblicato da O'Reilly, di cui sono l'autore. Cortesemente Google ne pubblica un ampio estratto di ben 94 pagine che copre tutte le basi, lo trovate facilmente su Google stesso con la chiave"Learning Apache OpenWhisk". Inoltre c'è su YouTube in italiano un corso gratuito che copre Nimbella, che è un servizio commerciale Serveless basato su OpenWhisk. La playlist ha URl bit.ly/intronim

# Compilazione in locale

Iniziamo le operazioni di installazione partendo dalla modalità più semplice, installazione in locale in modalità singola ("standalone"). Questo passo è propedeutico per poi creare una immagine Docker da usare in Kubernetes.

Quindi si può cominciare ad usarlo in maniera molto semplice ed immediata. Tutto quello che vi serve è Docker, un Java Development Kit per compilare il backend, e Node.js per poter compilare l front-end. Se siete uno sviluppatore molto probabilmente avete già tutto questo nella vostra macchina.

Se avete i prerequisiti sulla vostra macchina potete cimentarvi nell'impresa, tutto sommato piuttosto semplice, di installarlo e farlo partire in locale. Potete semplicemente scaricare i sorgenti da qui:

<https://github.com/apache/openwhisk>

Nelle releases troverete la versione 1.0, in formato zip e tar.gz. Vi conviene comunque prendere il master con un bel git clone, in quando è stabile e di solito più aggiornato delle releases, e quindi contiene le ultime novità e bug fix.

Una volta scaricato e spacchettato, o clonato, andate nella directory principale e potete compilare OpenWhisk standalone con

./gradlew :core:standalone:build

Dovete avere un attimo di pazienza (ci mette un po' a compilare) e avrete OpenWhisk versione locale pronto all'uso. Mentre compila vi consiglio anche di andare su:

<https://github.com/apache/openwhisk-cli>

per scaricare la command line interface per il vostro sistema operativo. Una volta scaricata, dezippate il pacchetto e mettete il comandu **wsk** nel *PATH* da qualche parte nel vostro sistema (per esempio tipicamente in  */usr/local/bin* su Mac e Linux, o *C:\Windows\System32* su Windows).

Tutto è pronto per lanciarvi nel mondo del serverless! Potete lanciare OpenWhisk in locale con il comando:

java –jar bin/openwhisk-standalone.jar

Questo dovrebbe normalmente aprire il browser con il playground. Si tratta di una semplice applicazione javascript che vi permette di scrivere una funzione Serverless nei vari linguaggi direttamente nel browser e provarla.

# Sperimentare con il serverless

Una volta che avete fatto hello world nel playground, la cosa migliore è scrivere delle applicazioni con un editor, visto che il playground non è un completo ambiente di sviluppo. E qui viene utile la utility a riga di comando **wsk**.

La cli la dovete innanzitutto configurare. Se leggete accuratamente l'output di OpenWhisk quando lo lanciate (fate attenzione, è verso l'inizio, e viene immediatamente 'scrollato via' dagli altri messaggi che vengono mostrati) troverete il comando per configurare.

A questo punto copiate uno degli esempi che vi presenta il playground, per esempio *hello.js* in un file, e scrivete:

wks action update hello hello.js

per creare l'azione hello. A questo punto la potete invocare e vedere l'output con

wsk action invoke hello –p name Michele -r

per vedere l'output.

# Creare una immagine Docker

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.

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

Notare che abbiamo dato un nome all'immagine, perché la utilizzeremo solo come area temporanea per la compilazione e ci serve ilnome per riferirla dopo.

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 git (che serve perscaricare i sorgenti) e nodejs (che serve per compilare il frontend) no, quindi li aggiungiamo usando la direttiva *RUN:*

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

A questo punto scarichiamo i sorgenti dal github, cambiamo alla directory di lavoro e compiliamo il tutto:

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

WORKDIR openwhisk

RUN ./gradlew :core:standalone:build

Ora l'immagine è pìu o meno pronta solo che non vogliamo portarci dietro tutti i sorgenti e gli artefatti della compilazione. Per questo motivo adesso aggiungiamo al Dockerfile le direttive per metterci dentro solo quello che serve.

WORKDIR /home

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

Aggiungiamo ora un file di startup, start.sh. Ci sono un po' di accortezze e verifiche da fare ma per il momento assumiamo che *start.sh* sia solamente questo:

#!/bin/bash

cd /home

java -jar openwhisk-standalone.jar

Vedremo dopo che non basta, occorre fare dei controlli pre-flight. Ma per il momento aggiungiamo al Dockerfile questo script e lo impostiamo come punto di partenza per provare il tutto.

COPY start.sh /bin/start.sh

ENTRYPOINT ["/bin/start.sh"]