Operatività in Storm SMACS

Murgia Antonio

30/03/2015

1 Introduzione

In questo breve documento tecnico verranno esplicitate le procedure da eseguire per ottenere operatività con gli strumenti: Scala, sbt e Apache Storm. Questo how-to è stato sviluppato in concomitanza con il progetto Storm SMACS con lo scopo di facilitare l'introduzione nel progetto di nuovi sviluppatori.
La guida sarà divisa in due capitoli principali: la prima parte sarà particolamente dedicata a nuovi sviluppatori che vogliono contribuire allo sviluppo; la seconda invece si occuperà di fornire gli strumenti necessari per eseguire il deployment dell'architettura Storm SMACS.

2 Guida allo Sviluppo

Per ottenere operatività all'interno del progetto Storm SMACS è necessario disporre dei seguenti strumenti di base, imprescindibili in qualunque ambiente di lavoro Scala: scalac (il compilatore Scala), sbt (simple build tool) e git, l'arcinoto sistema di versioning. Non ci addentreremo nel processo di installazione e setup poichè gli argomenti sono particolarmente ben coperti e variano rapidamente con l'evoluzione degli strumenti.
Il progetto Storm SMACS è stato sviluppato utilizzando Scala versione 2.11.5 (la più recente al momento della scrittura) e Java SE versione 6. Da questo momento in poi daremo per scontata una conoscenza almeno marginale di ciò che è un tool di project e dependency management come sbt (o maven).
Il progetto Storm SMACS è composto dai seguenti sotto progetti:

- stormSMACS

- ceilometerAPI4s
- scalaStorm (un fork della repository)
- utils
- cfAgent
- sigarAgent

In figura è possibile osservare le dipendenze che esistono tra i progetti. La prima cosa da fare è eseguire il clone delle repository di ognuno di questi progetti, per fare questo sarà necessario eseguire il seguente comando sostituendo a <link> il link di ogni progetto (che sarà disponibile in appendice):

```
git clone <link>
```

Solamente i progetti stormSMACS, cfAgent e sigarAgent sono eseguibili, i restanti rappresentano librerie in cui sono stati posti a fattor comune soluzioni a problemi ricorrenti. Per questo sarà necessario porsi tramite un terminale nella cartella principale dei progetti scalaUtils, scalaStorm e ceilometerAPI4s ed eseguire il seguente comando:

```
sbt publishLocal
```

In questo modo verranno pubblicati in una repository locale i seguenti artefatti (seguendo la convenzione Maven):

- it.unibo.ing.smacs.ceilometerAPI4s
- it.unibo.ing.utils
- com.github.velvia.scala-storm

Fatto questo sarà possibile compilare i progetti stormSMACS, cfAgent e sigarAgent, con il comando:

```
sbt compile
```

I progetti cfAgent e sigarAgent possono essere eseguiti in modo standalone semplicemente digitando il comando:

```
sbt run
```

Entrambi accettano un parametro di tipo intero che indica la porta da utilizzare per il web server, le porte di default sono:

- 9876 : cfAgent
- 9875 : sigarAgent

Per quanto riguarda il testing del progetto stormSMACS è invece necessaria un'installazione funzionante di Apache Storm (la quale verrà coperta nel capitolo successivo). Apache Storm fortunatamente offre anche una modalità di debug, che permette di eseguire l'intera topologia in thread locali per simulare il comportamento nel distribuito. In ogni caso per eseguire una topologia Storm è necessario un singolo file .jar il quale dovrà contenere tutte le dipendenze del progetto.
Per poter generare tale file jar ci avvaleremo di un plugin per sbt: sbt-assembly.
È possibile visitare la repository del progetto su github all'indirizzo: `http://github.com/sbt/sbt-assembly`
In tale pagina sono fornite le istruzioni per installare il plugin, dopo averlo installato sarà sufficiente eseguire il comando:

```
sbt assembly
```

Quindi verrà creato un file con estensione jar nella cartella target/scala-2.11/.
Per eseguire stormsmacs in modalità debug è necessario eseguire il comando:

```
storm jar target/scala-2.11/storm-smacs-assembly-1.0.jar it.unibo.ing.
    ↪ stormsmacs.topologies.Topology src/test/resources/emptyConf.json
```

Andiamo ad esplicitare i parametri passati al comando storm:
Il primo parametro *jar* indica a storm che vogliamo mettere in esecuzione un file jar. Il secondo parametro indica invece il percorso del jar. Il terzo parametro rappresenta la classe che fornisce il punto d'accesso alla topologia Storm e che ne esegue la configurazione. Il quarto parametro (che viene passato direttamente alla nostra topologia) indica un file di configurazione necessario e minimale per poter eseguire stormSMACS.
Di seguito riportiamo il contenuto del file:

```
{
  "name"              : "test",
  "persisterNode"     : {
    "id"  : "fuseki",
    "type": "FUSEKI",
    "url" : "http://10.0.10.13:3030/ds/update"
  },
  "remote"            : false,
  "debug"             : true,
```

```
  "pollTime"          : 60000
}
```

name rappresenta il nome che verrà dato alla topologia in esecuzione, *remote* è un parametro di configurazione di Storm, impostandolo a false, indica che la topologia non è eseguita in un ambiente remoto ma locale. *debug* indica che deve essere eseguito in modalità debug, quindi simulata su thread, ed infine il parametro pollTime indica la frequenza in millisecondi in cui è necessario effettuare il campionamento. Il parametro persisterNode indica l'endpoint SPARQL sul quale verranno salvati i dati raccolti da stormSMACS, in questa configurazione non è importante che rappresenti un endpoint valido. Un file completo può essere visionato nella cartella src/test/resources/confExample.json, all'interno del file sono definiti più nodi che si intende monitorare di diverse topologie, la loro configurazione è particolarmente banale e quindi non verrà trattata (self-explanatory code).
Per quanto riguarda l'utilizzo di un IDE, il progetto non fa affidamento alla presenza di alcuno di essi e può essere quindi utilizzato con il vostro preferito. Personalmente ho giudicato IntelliJ l'IDE più maturo per quanto riguarda il panorama Scala, il quale offre un'ottima integrazione anche con sbt. Ad ogni modo consiglio di non allontanarsi mai troppo dalla linea di comando, la quale tramite sbt offre uno strumento formidabile.

3 Guida al Deployment

Tale guida è stata redatta in data 30/03/2015 e fa riferimento alla versione di Apache Storm 0.9.4.
Essa è il frutto dell'esperienza maturata nello sviluppo del progetto e prende spunto dalla guida in lingua inglese presente all'indirizzo `http://www.michael-noll.com/tutorials/running-multi-node-storm-cluster/`

3.1 Configurazione di un cluster Apache Storm

Per ottenere l'operatività di un cluster Storm è necessario configurare (almeno) 3 nodi, ognuno eseguirà un differente processo, storm nimbus, storm supervisor e zookeeper. Questa guida è stata testata utilizzando *Ubuntu 14.04 LTS*.

3.1.1 Configurazione comune a tutte le macchine

In primo luogo è necessario installare Oracle JDK versione 6 e rimuovere la versione precedentemente installata di default (OpenJDK). Per farlo è necessario eseguire i

seguenti comandi da una shell bash:

```
sudo apt-get purge openjdk*
sudo apt-get install python-software-properties
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java6-installer
```

Quindi per comodità arricchiremo il file `/etc/hosts` in modo da poter riferire le macchine del nostro cluster attraverso un nome e non l'indirizzo IP:

```
192.168.10.1    slave
192.168.10.200  nimbus
192.168.10.250  zkserver
```

Inoltre sarà necessario installare *supervisord* un tool che permette di porre sotto supervision e di eseguire allo startup i processi necessari.

```
sudo apt-get install supervisor
```

3.1.2 Configurazione Zookeeper

Questa procedura va eseguita solamente sulla macchina denominata zookeeper. Eseguiamo l'installazione di Zookeeper tramite il comando:

```
sudo apt-get install zookeeper
```

Dopodichè eseguiamo la configurazione del server Zookeper modificando il file `/etc/zookeeper/conf/zoo.cfg` in modo che contenga:

```
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
# Enable regular purging of old data and transaction logs every 24 hours
autopurge.purgeInterval=24
autopurge.snapRetainCount=5
```

Quindi sarà necessario aggiungere zookeeper ai processi gestiti da supervisord, per fare ciò è necessario creare il file `/etc/supervisor/conf.d/zookeeper.conf` con il seguente contenuto:

```
[program:zookeeper]
command=/usr/share/zookeeper/bin/zkServer.sh start-foreground
```

```
directory=/usr/share/zookeeper/bin/
autorestart=true
autostart=true
```

In questo modo supervisor eseguirà ad ogni reboot il server zookeeper (autostart = true) e ne eseguirà il restart nel caso esso termini inavvertitamente(autorestart = true). Per indicare a supervisor che deve gestire zookeeper è necessario eseguire:

```
sudo supervisorctl reread
sudo supervisorctl update
sudo supervisorctl start zookeeper
```

3.1.3 Configurazione Apache Storm

Questa procedura andrà eseguita su tutte le macchine che eseguiranno il processo Storm, quindi sia nimbus che supervisor. Prima di tutto assicuriamoci di avere installati i tool di sviluppo necessari tramite:

```
sudo apt-get install build-essential
sudo apt-get install uuid-dev
sudo apt-get install autoconf
sudo apt-get install libtool
sudo apt-get install pkg-config
```

Quindi installiamo ZeroMQ necessario per far funzionare Apache Storm:

```
wget http://download.zeromq.org/zeromq-2.1.7.tar.gz
tar -xzf zeromq-2.1.7.tar.gz
cd zeromq-2.1.7
./autogen.sh
./configure
make
sudo make install
```

Quindi andiamo ad installare JZMQ, i binding java per zeroMQ:

```
git clone http://github.com/zeromq/jzmq
cd jzmq
./autogen.sh
./configure
make
sudo make install
```

Quindi installiamo Python 2.6.6 necessario per l'esecuzione del comando `storm`, per farlo installeremo pyenv e tramite esso installeremo python:

```
curl -L https://raw.githubusercontent.com/yyuu/pyenv-installer/master/bin
    ↪ /pyenv-installer | bash
export PATH="$HOME/.pyenv/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv init -)"
sudo apt-get install libssl-dev
sudo apt-get install zlib1g-dev
sudo apt-get install libbz2-dev
sudo apt-get install libreadline-dev
sudo apt-get install libsqlite3-dev
sudo apt-get install llvm
pyenv install 2.6.6
pyenv global 2.6.6
```

Quindi eseguiamo il download e l'installazione della versione 0.9.4 di Apache Storm:

```
wget http://it.apache.contactlab.it/storm/apache-storm-0.9.4/apache-storm
    ↪ -0.9.4.tar.gz
tar -xvf apache-storm-0.9.4.tar.gz
sudo mv apache-storm-0.9.4 /usr/local/
sudo chown -R root:root /usr/local/apache-storm-0.9.4
sudo ln -s /usr/local/storm-0.8.2 /usr/local/storm
```

3.1.4 Configurazione nodo Nimbus

Relativamente al nodo nimbus modificheremo le impostazioni in modo che riflettano la nostra topologia di rete e porremo sotto supervisor i processi `storm nimbus` e `storm ui`. Il file `/usr/local/storm/conf/storm.yaml` dovrà contenere:

```
storm.zookeeper.servers:
    - "zkserver"
nimbus.host: "nimbus"
storm.local.dir: "/usr/local/storm/data"
```

Quindi andiamo a creare il file `/etc/supervisor/conf.d/nimbus.conf` con il seguente contenuto:

```
[program:nimbus]
```

```
command=/usr/local/storm/bin/storm nimbus
directory=/usr/local/storm
autorestart=true
autostart=true
priority=1
```

Ed il file `/etc/supervisor/conf.d/stormui.conf`:

```
[program:stormui]
command=/usr/local/storm/bin/storm ui
directory=/usr/local/storm
autorestart=true
autostart=true
priority=50
```

Quindi eseguiamo i comandi:

```
sudo supervisorctl reread
sudo supervisorctl update
sudo supervisorctl start nimbus
sudo supervisorctl start stormui
```

In questo modo verranno avviati automaticamente e riavviati in caso di crash i processi storm nimbus e storm ui.

3.1.5 Configurazione nodo slave

Relativamente al nodo slave modificheremo le impostazioni in modo che riflettano la nostra topologia di rete e porremo sotto supervisor il processo `storm supervisor`. Il file `/usr/local/storm/conf/storm.yaml` dovrà contenere:

```
storm.zookeeper.servers:
    - "zkserver"
nimbus.host: "nimbus"
storm.local.dir: "/usr/local/storm/data"
supervisor.slots.ports:
    - 6700
    - 6701
```

Quindi andiamo a creare il file `/etc/supervisor/conf.d/supervisor.conf` con il seguente contenuto:

```
[program:supervisor]
command=/usr/local/storm/bin/storm supervisor
directory=/usr/local/storm
autorestart=true
autostart=true
priority=1
```

Quindi eseguiamo i comandi:

```
sudo supervisorctl reread
sudo supervisorctl update
sudo supervisorctl start supervisor
```

In questo modo verrà avviato automaticamente e riavviato in caso di crash il processo storm supervisor.

3.2 Configurazione Apache Jena Fuseki

Per la configurazione dell'endpoint SPARQL andremo ad installare Apache Jena Fuseki.

```
wget http://apache.fastbull.org/jena/binaries/jena-fuseki1-1.1.2-
    ↪ distribution.tar.gz
tar -xvf jena-fuseki1-1.1.2-distribution.tar.gz
sudo mv jena-fuseki1-1.1.2-distribution /usr/local/
sudo chown -R root:root /usr/local/jena-fuseki1-1.1.2-distribution
sudo ln -s /usr/local/jena-fuseki1-1.1.2-distribution /usr/local/fuseki
sudo mkdir /usr/local/fuseki/stormsmacs
```

Quindi andiamo a configurare supervisor per avviare automaticamente fuseki:

```
[program:fuseki]
command=/usr/local/fuseki/fuseki-server --update --loc=stormsmacs /ds
directory=/usr/local/fuseki/
stdout_logfile=/usr/local/fuseki/log.txt
redirect_stderr=true
autorestart=true
autostart=true
```

E quindi:

```
sudo supervisorctl reread
```

```
sudo supervisorctl update
sudo supervisorctl start fuseki
```

3.3 Deployment di sigarAgent

Per il deployment di sigarAgent è necessario ottenere il jar sigarAgent.jar e le librerie necessarie al funzionamento di sigar (che non è possibile accorpare al file jar) ed eseguire:

```
export LD_LIBRARY_PATH="nativelibs"
sudo java -jar sigarAgent.jar
```

3.4 Deployment di cfAgent

Per il deployment di cfAgent è necessario ottenere il jar cfAgent.jar ed eseguire:

```
sudo java -jar sigarAgent.jar
```

4 Esecuzione di Storm SMACS

Per avviare stormSMACS basta semplicemente eseguire sul nodo *nimbus* il comando:

```
storm jar storm-smacs-assembly-1.0.jar it.unibo.ing.stormsmacs.topologies
    ↪ .Topology confFile.json
```

`confFile.json` è il file che rappresenta la topologia delle macchine da monitorare.

A Link delle repository dei progetti

- stormSMACS: `http://github.com/tmnd1991/stormSMACS`
- ceilometerAPI4s `http://github.com/tmnd1991/ceilometerAPI4s/settings`
- scalaStorm `http://github.com/tmnd1991/scalaStorm`
- utils `http://github.com/tmnd1991/scalaUtils`
- cfAgent `http://github.com/tmnd1991/cfAgent`
- sigarAgent `http://github.com/tmnd1991/sigarAgent`