Upute

*Ove upute su napisane za linux, a testirane su nad operacijskim sustavom Ubuntu 20.04.

Potrebni alati za rad:

- Java JDK 1.8.0
- Apache Beam 2.25.0
- Apache Kafka 3.2.0
- Apache Flink 1.10.3
- Apache Spark 2.4.7
- Apache Hadoop 2.7.2
- Apache Maven 3.6.3
- Hazelcast Jet 4.5.3

CSV datoteka s podacima se može pronaći:
https://data.cityofchicago.org/api/views/ggws-77ih/rows.csv?accessType=DOWNLOAD&bom=true&fo

Kod je dostupan na github repozitoriju https://github.com/zokonil/beam-benchmark

Prije pokretanja potrebno je naravno instalirati gore navedenu programsku podršku i postaviti varijable okruženja:

- JAVA_HOME
- MVN_HOME
- FLINK_HOME
- SPARK_HOME
- JET_HOME
- ZOOKEEPER_HOME
- HADOOP_HOME
- YARN_HOME

Dodatno da bi radio Apache Spark potrebno je dodati sljedeće kao varijablu okruženje:

0.0.1

export SPARK_DIST_CLASSPATH=$hadoopclasspath

Prvo je potrebno pokrenuti kafku, a za njega nam je potrebna zookeeper aplikacija, ako su dobro postavljene varijable okruženje potrebno je pokrenuti sljedeću liniju koda za zookeeper:

0.0.2

zookeeper-server-start.sh $KAFKA_HOME/config/zookeeper.properties

Sada nakon što smo osigurali da nam je zookeeper servis pokrenut možemo i pokrenuti sam kafka servis. On se pokreće na sličan način no prije toga ako želite mijenjati broj particija koju će neki od topica imati potrebno je promijeniti num.parititons varijablu u server.properties datoteci koja se nalazi u $KAFKA_HOME/config. Nakon što ste konfigurirali kafku po svojim željama, pokrenite sljedeću komandu:

0.0.3

kafka-server-start.sh $KAFKA_HOME/config/server.properties

Sada je potrebno dodati topic na koji će se pisati i topic iz kojeg će se čitati podaci. Imena ovih topica se mogu promijeniti unutar Constants.java datoteke ako želite. Inače će ime ulaznog topica biti input, a izlaznoga output.
Za stvaranje novog topica pokrenite sljedeće komande:

0.0.4

kafka-topics.sh –create –topic input –replication-factor 1 –partitions N –bootstrap-server localhost:9092

0.0.5

kafka-topics.sh –create –topic output –replication-factor 1 –partitions 1 –bootstrap-server localhost:9092

Razlog iz kojega je N na mjestu particija za kreiranje ulaznog topica je da se poveća paralelizam kod čitanja podataka s ulaznog topica. Kod testiranja je N bio jednak broju paralelnih procesa koji su se odvijali na izvođaču.

Ako se odabere N koji je veći od 1 mora se pripaziti na to da je unutar Beam.java datoteke kod čitanja ulaznih podataka postavljena opcija .withTopicPartitions(topicPartitions)

i potrebno je paziti da je unutar Constants.java datoteke korektno postavljena varijabla NUM_OF_KAFKA_PARTITIONS = N.

Sljedeće moramo upaliti program koji će nam prikazivati rezultate, to radimo tako da se pozicioniramo u mapu gdje se nam nalazi kod i pokrenemo sljedeću komandu:

0.0.6

mvn compile exec-java -D exec.mainClass=ht.test.kafka.Consumer

Sada kad smo postavili pozadinu pokretanja moramo odabrati koji od izvođača želimo pokrenuti.

0.1. Apache Flink

Za pokretanje ovog izvođača postoje dvije metode. Možemo konfigurirati čvorove u slaves datoteci, gdje možemo zapisati ili IP adrese lokalnih računala koje imaju postavljeno okruženje za pokretanje Flink čvora. Ili možemo dodati riječ localhost onoliko puta koliko čvorova želimo dodati u klaster.
Kako bi pokrenuli takav klaster jednostavno moramo pokrenuti sljedeću komandu:

0.1.1

start-cluster.sh

Ovo komanda će se brinuti za pokretanje JobManagera i svih TaskManagera na Flink klasteru. S uspješnim pokretanjem ove komande također ćemo dobiti i web UI na localhost:808x. Naravno nad kojim portom želimo otvoriti ovaj web UI možemo konfigurirati unutar flink-conf.yaml datoteku koja se nalazi u $FLINK_HOME/config. Ovdje možemo i konfigurirati koliko slotova će svaki od taskmanagera imati s varijablom taskmanager.numberOfTaskSlots: N.

Drugi način pokretanja je izmjenom nekih od skripta koje se nalaze u scripts mapi od beam-benchmark repozitorija. Oba dvije skripte se kopiraju u $FLINK_HOME/bin mapu i nakon toga je moguće pokrenuti TaskManager s određenim brojem jezgra procesora. Ovakvo pokretanje flink klastera se radi preko sljedećih komandi:

0.1.2

start-cluster.sh

taskmanager.sh start K

K u ovom slučaju predstavlja jezgre procesora nad kojima će se TaskManager odvijati. Primjer varijable K su : 0; 0,1,2,3; 0-7; 8-11.Također će nakon pokretanja start-cluster.sh komande biti dostupno web sučelje na prije spomenutoj adresi.
Da bi pokrenuli Flink nad našim klasterom ponovo je potrebno pozicionirati se u mapu gdje se nalazi beam-beanchmark repozitorij i pokrenuti sljedeću komandu:

0.1.3

mvn compile exec:java -D exec.mainClass=hr.test.beam.Beam -D exec.args="–runner=FlinkRunner –task=OPERACIJA –flinkMaster=HOST:PORT –parallelism=N" -P flink-runner

kao OPERACIJA se prima jedan od 6 parametara a to su:

- identity
- filtration
- aggregate
- topN
- specificValues
- compositeTransformation

A kao flinkMaster će ako je slučaj lokalnog pokretanja vrijednost biti localhost:808x, PORT je opet ovisan o vašoj konfiguraciji. Nakon što smo pokrenuli ovu komandu možemo pričekati par sekundi i u web sučelju bi trebali vidjeti da je naš program pokrenut.

0.2. Apache Spark

Pokretanje ovog izvođača je slično pokretanju izvođača Apache Flink, dijele sličnu logiku i konfiguraciju, no ovaj puta je jedan način pokretanja pošto nam Spark nudi i više nego dovoljno konfiguracijskih mogućnosti ako želimo limitirati broj jezgri nad određenim čvorom. Kako bi pokrenuli Spark u svoj PATH je potrebno dodati $SPARK_HOME/bin i $SPARK_HOME/sbin. Oba imaju komande koje su bitne za pokretanje klastera i za čitanje konfiguracije. Za pokretanje samog klastera koristima komandu:

0.2.1

start-master.sh

Ovom komandom smo pokrenuli sam klaster no na njemu nema čvorova koji će obavljati posao. Ali što smo dobili pokretanjem ove komande je web sučelje slično onom koje nam je nudio i Spark. Manje je responzivno, ali nudi dovoljno opcija da vidimo što se događa na klasteru u koje vrijeme. Ovo sučelje će biti pokrenuto na localhost:8080 ili na konfiguriranom portu. Spark za konfiguraciju nudi nekoliko datoteka koje se nalaze u $SPARK_HOME/conf. Konfiguracije kao port i host na kojim se pokreće klaster se nalaze u datoteci spark-env.sh. Ovdje su dvije varijable bitne za pokretanje klastera, a one su SPARK_WORKER_CORES, koja postavlja koliko procesorskih jezgri jedan čvor klastera ima pristup i SPARK_WORKER_INSTANCES, koja postavlja broj čvorova koje će SPARK pokrenuti. Kako bi pokrenuli čvorove koristimo sljedeću komandu:

0.2.2

start-slave.sh spark://HOST:PORT

Cijeli URL koji se prosljeđuje start-slave skripti je moguće vidjeti na vrhu web sučelja koje smo malo prije pokrenuli. Ako nam nakon pokretanja skripte u sučelju pokaže da su dodani čvorovi možemo nastaviti s pokretanjem programa na spark klasteru. Za ovo je prvo potrebno stvoriti paket koji ćemo predati na klaster, ovo se radi sljedećom funkcijom:

0.2.3

mvn package -P spark-runner

Ovo će stvoriti datoteku u target mapi, izgenerirane će biti 2 ili 3 .jar datoteke, mi ćemo koristiti onu koja u nazivu ima bundled. Primjer pokretanja programa nad spark klasterom:

0.2.4

spark-submit –class hr.test.beam.Beam –deploy-mode cluster –master spark://HOST:PORT ./target/apache-beam-test-bundled-1.0-SNAPSHOT.jar –task=OPERACIJA –runner=SparkRunner

Opet isto kao i kod Flinka HOST i PORT, odnosno cijeli URL koji se predaje u master varijablu spark-submit skripte se nalazi na web sučelju. Nakon pokretanja ove skripte bi na tim istim sučelju trebala biti vidljiva nova aplikacija. Ako je sve uspješno

prošlo smo pokrenuli našu aplikaciju na Spark klasteru.

** U slučaju da vam ne radi spark-master.sh ili spark-slave.sh komanda potencijalna greška je kod varijabla okruženja. Dodajte sljedeću liniju u okruženje :

0.2.5
export SPARK_DIST_CLASSPATH=$hadoopclasspath

I nakon toga pokrenite :

0.2.6
source file

Ovdje file predstavlja datoteku koju koristite za postavljanje varijabla okruženja. Potencijalno bashrc, profile, zshrc...

0.3. Hazelcast Jet

Za Jet pokretanje je jednostavnije od ostalih, jedino što zapravo moramo osigurati je to što su nam varijable okruženja koje opisuju JET postavljene i pokrećemo sljedeću komandu:

0.3.1
mvn compile exec:java -D exec.mainClass=hr.test.beam.Beam -D exec.args="–runner=JetRunner –jetLocalMode=K –task=OPERACIJA" -P jer-runner

Sa jetLocalMode varijablom K, koja je ustvari broj, postavljamo broj čvorova koji će se pozvati nad klasterom. U konzoli bi trebali dobiti ispis ako i kad se pokrene aplikacija. Ako želite dodatno konfigurirati jet to je moguće tako da se u mapi gdje smo pokrenuli program komandom iznad dodaju konfiguracijske datoteke, kao što su hazelcast.yaml, hazelcast-client.yaml i hazelcast-jet.yaml.

Nakon što smo pokrenuli našu aplikaciju nad jednim od izvođača možemo i napuniti ulazni topic kako bi aplikacija mogla čitati podatke i izračunati što smo tražili od nje. Ovo se radi pokretanjem sljedeće komande:

0.3.2

mvn compile exec:java -D exec.mainClass=hr.test.kafka.Producer

Ova komanda će pokrenuti Producer aplikaciju koja čita određeni broj podataka koji smo specificirali u Constants.java datoteci i šalje ih na ulazni topic kafke. Producer će se ugasiti nakon što proslijedi sve podatke na topic, a naša aplikacija, ako je dobro pokrenuta, će čitati podatke s tog topica. Nakon što završi će prozore u kojem smo pokrenuli Consumer dobiti poruku i izračunati propusnost alata i propusnost sustava.