Store Event Hub Messages to Windows Azure Blob with Trident

# Architecture

# Prerequisites

* An Azure subscription
* Java and JDK
* Maven
* Visual Studio with the Microsoft Azure SDK for .NET
* Git tool

# Configure Windows Azure

## Create the Azure Storage Account

* Create Azure Storage account

## Create the Azure Redis Cache

* Create Azure Redis Cache following the following link: How to Use Azure Redis Cache (<http://azure.microsoft.com/en-us/documentation/articles/cache-dotnet-how-to-use-azure-redis-cache/> )

## Configure Event Hub

* Create Event Hub with partition count 10 and message retention of 1 days.
* Once the event hub has been created, select the event hub you created. Select **Configure**, then create two new **shared access policies** using the following information.

NAME PERMISSIONS

Devices Send

Storm Listen

## Create the HDInsight Storm cluster

* Sign in to the azure and create a storm cluster and create a Storm cluster

# Install Java Dependencies

## Install Microsoft Azure SDK for Java

* Clone Microsoft Azure SDK for Java from <https://github.com/Azure/azure-sdk-for-java>
* Use the following command to install the package into the local Maven store. This will allow us to easily add it as a reference in the Storm project in a later step.

mvn clean install -Dmaven.test.skip=true

## Install Windows Azure Storage libraries for Java

* Clone Windows Azure Storage libraries for Java (<https://github.com/Azure/azure-storage-java>)
* Use the following command to install the package into the local Maven store. This will allow us to easily add it as a reference in the Storm project in a later step.

mvn clean install -Dmaven.test.skip=true

## Install the Event Hub spout

Several of the dependencies used in this project must be downloaded and built individually, then installed into the local Maven repository on your development environment.

In order to receive data from Event Hub, we will use the eventhubs-storm-spout.

* Use Remote Desktop to connect to your Storm cluster, then copy the %STORM\_HOME%\examples\eventhubspout\eventhubs-storm-spout-0.9-jar-with-dependencies.jar file to your local development environment.
* Use the following command to install the package into the local Maven store. This will allow us to easily add it as a reference in the Storm project in a later step.

mvn install:install-file -Dfile=eventhubs-storm-spout-0.9-jar-with-dependencies.jar -DgroupId=com.microsoft.eventhubs -DartifactId=eventhubs-storm-spout -Dversion=0.9 -Dpackaging=jar

## Install Jedis SDK with SSL Support

* Clone jedis sdk with SSL support from <https://github.com/RedisLabs/jedis>
* Use the following command to install the package into the local Maven store. This will allow us to easily add it as a reference in the Storm project in a later step.

mvn clean install -Dmaven.test.skip=true

# Write the code

## Scaffold the Storm topology project

* Use the following Maven command to create the scaffolding for the Trident topology project.

mvn archetype:generate -DgroupId=com.contoso.app.trident -DartifactId=eventhub-blobwriter -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

## Add dependencies in pom.xml

Using a text editor, open the pom.xml file, and add the following to the <dependencies> section. You can add them at the end of the section, after the dependency for junit.

<dependency>

<groupId>org.apache.storm</groupId>

<artifactId>storm-core</artifactId>

<version>0.9.1-incubating</version>

<!-- keep storm out of the jar-with-dependencies -->

<scope>provided</scope>

</dependency>

<dependency>

<groupId>com.google.guava</groupId>

<artifactId>guava</artifactId>

<version>13.0.1</version>

</dependency>

<dependency>

<groupId>commons-collections</groupId>

<artifactId>commons-collections</artifactId>

<version>3.2.1</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.7</version>

<!-- keep out of the jar-with-dependencies -->

<scope>provided</scope>

</dependency>

<dependency>

<groupId>com.microsoft.eventhubs</groupId>

<artifactId>eventhubs-storm-spout</artifactId>

<version>0.9</version>

</dependency>

<dependency>

<groupId>com.google.code.gson</groupId>

<artifactId>gson</artifactId>

<version>2.3</version>

</dependency>

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.5.0.ssl</version>

</dependency>

<dependency>

<groupId>com.microsoft.azure</groupId>

<artifactId>azure-storage</artifactId>

<version>1.3.1</version>

</dependency>

<dependency>

<groupId>com.microsoft.azure</groupId>

<artifactId>azure-management</artifactId>

<version>0.6.0</version>

</dependency>

<dependency>

<groupId>com.microsoft.azure</groupId>

<artifactId>azure-management</artifactId>

<version>0.6.0</version>

</dependency>

<dependency>

<groupId>com.microsoft.azure</groupId>

<artifactId>azure-management-compute</artifactId>

<version>0.6.0</version>

</dependency>

<dependency>

<groupId>com.microsoft.azure</groupId>

<artifactId>azure-management-network</artifactId>

<version>0.6.0</version>

</dependency>

<dependency>

<groupId>com.microsoft.azure</groupId>

<artifactId>azure-management-sql</artifactId>

<version>0.6.0</version>

</dependency>

<dependency>

<groupId>com.microsoft.azure</groupId>

<artifactId>azure-management-storage</artifactId>

<version>0.6.0</version>

</dependency>

<dependency>

<groupId>com.microsoft.azure</groupId>

<artifactId>azure-management-websites</artifactId>

<version>0.6.0</version>

</dependency>

<dependency>

<groupId>com.microsoft.azure</groupId>

<artifactId>azure-media</artifactId>

<version>0.6.0</version>

</dependency>

<dependency>

<groupId>com.microsoft.azure</groupId>

<artifactId>azure-servicebus</artifactId>

<version>0.6.0</version>

</dependency>

<dependency>

<groupId>com.microsoft.azure</groupId>

<artifactId>azure-serviceruntime</artifactId>

<version>0.6.0</version>

</dependency>

<dependency>

<groupId>com.microsoft.windowsazure.storage</groupId>

<artifactId>microsoft-windowsazure-storage-sdk</artifactId>

<version>0.6.0</version>

</dependency>

Note: Some dependencies are marked with a scope of **provided** to indicate that these dependencies should be downloaded from the Maven repository and used to build and test the application locally, but that they will also be available in your runtime environment and do not need to be compiled and included in the JAR created by this project.

## Add plugins in pom.xml

* At the end of the pom.xml file, right before the </project> entry, add the following.

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.2</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-shade-plugin</artifactId>

<version>2.3</version>

<configuration>

<createDependencyReducedPom>true</createDependencyReducedPom>

<transformers>

<transformer

implementation=*"org.apache.maven.plugins.shade.resource.ApacheLicenseResourceTransformer"*>

</transformer>

</transformers>

</configuration>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>shade</goal>

</goals>

<configuration>

<transformers>

<transformer

implementation=*"org.apache.maven.plugins.shade.resource.ServicesResourceTransformer"* />

<transformer

implementation=*"org.apache.maven.plugins.shade.resource.ManifestResourceTransformer"*>

<mainClass></mainClass>

</transformer>

</transformers>

</configuration>

</execution>

</executions>

</plugin>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>1.2.1</version>

<executions>

<execution>

<goals>

<goal>exec</goal>

</goals>

</execution>

</executions>

<configuration>

<executable>java</executable>

<includeProjectDependencies>true</includeProjectDependencies>

<includePluginDependencies>false</includePluginDependencies>

<classpathScope>compile</classpathScope>

<mainClass>${storm.topology}</mainClass>

</configuration>

</plugin>

</plugins>

<resources>

<resource>

<directory>${basedir}/conf</directory>

<filtering>false</filtering>

<includes>

<include>Config.properties</include>

</includes>

</resource>

</resources>

</build>

* This tells Maven to do the following when building the project:
  + Include the /conf/Config.properties resource file. This file will be created later, it will contain configuration information for connecting to Azure Event Hub.
  + Use the maven-compiler-plugin to compile the application.
  + Use the maven-shade-plugin to build an uber jar or fat jar, which contains this project and any required dependencies.
  + Use the exec-maven-plugin, which allows you to run the application locally without a Hadoop cluster.

## Add configuration file

eventhubs-storm-spout reads configuration information from a Config.properties file. This tells it what Event Hub to connect to. While you can specify a configuration file when starting the topology on a cluster, including one in the project gives you a known default configuration.

* In the eventhub-blobwriter directory, create a new directory named conf. This will be a sister directory of src.
* In the conf directory, create file Config.properties - contains settings for event hub
* Use the following as the contents for the Config.properties file.

eventhubspout.username = storm

# shared access key for the above event hub policy

eventhubspout.password = kGhIcUak7qVTyLPeUPnPy7GZEI5iqA0Rf1cJvrhsK5U=

eventhubspout.namespace = hanzeventhub1-ns

eventhubspout.entitypath = hanzeventhub1

eventhubspout.partitions.count = 10

# if not provided, will use storm's zookeeper settings

# zookeeper.connectionstring=localhost:2181

eventhubspout.checkpoint.interval = 10

eventhub.receiver.credits = 1024

storage.blob.account.name = hanzstorage

storage.blob.account.key = w9TEpvGTusvFlGAdCoWdDrwqLzy6er0Zm5YKdDD0YTkQdOj3WufeVrgd2c8q8amLR0o6xD0tBChcIIA+DCgxXA==

storage.blob.account.container = hanzstorm2

#number of blocks in each blob default to 50000

storage.blob.block.number.max = 2

#max bytes in each block default to 4194304 Byte

storage.blob.block.bytes.max = 1024

#Redis cache

redis.host = hanzredis1.redis.cache.windows.net

redis.password = eQoMISLEQf7mwCDetcvIUT+P9WGGK9KGsdf7/UOGkTg=

You should modify the value accorkding to your settings.

## Add EventHubMessage class to support serialization to and from JSON

* Create a new file EventHubMessage.jav in directory \eventhub-blobwriter\src\main\java\com\contoso\app\trident\

package com.contoso.app.trident;

import java.io.Serializable;

public class EventHubMessage implements Serializable {

private static final long serialVersionUID = 1L;

public double lat;

public double lng;

public long time;

public String id;

public EventHubMessage (double lat, double lng, long time, String id) {

super();

this.time = time;

this.lat = lat;

this.lng = lng;

this.id = id;

}

}

## Add BlobWriter class support uploading data to azure blob

Create a new file BlobWriter.java in directory \eventhub-blobwriter\src\main\java\com\contoso\app\trident\

**package** com.contoso.app.trident;

**import** java.io.ByteArrayInputStream;

**import** java.io.InputStream;

**import** java.nio.charset.StandardCharsets;

**import** java.util.ArrayList;

**import** java.util.Properties;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** com.microsoft.azure.storage.AccessCondition;

**import** com.microsoft.azure.storage.CloudStorageAccount;

**import** com.microsoft.azure.storage.blob.BlobRequestOptions;

**import** com.microsoft.azure.storage.blob.BlockEntry;

**import** com.microsoft.azure.storage.blob.BlockListingFilter;

**import** com.microsoft.azure.storage.blob.BlockSearchMode;

**import** com.microsoft.azure.storage.blob.CloudBlobClient;

**import** com.microsoft.azure.storage.blob.CloudBlobContainer;

**import** com.microsoft.azure.storage.blob.CloudBlockBlob;

**import** com.microsoft.azure.storage.core.Base64;

**public** **class** BlobWriter {

**static** **public** **void** upload(Properties properties, String blobname, String blockIdStr, String data) {

Logger logger = (Logger) LoggerFactory.*getLogger*(BlobWriter.**class**);

InputStream stream = **null**;

**try** {

**if** (LogSetting.***LOG\_BLOB\_WRITER*** && LogSetting.***LOG\_METHOD\_BEGIN***) {

logger.info("upload Begin");

}

String accountName = properties.getProperty("storage.blob.account.name");

String accountKey = properties.getProperty("storage.blob.account.key");

String containerName = properties.getProperty("storage.blob.account.container");

String connectionStrFormatter = "DefaultEndpointsProtocol=http;AccountName=%s;AccountKey=%s";

String connectionStr = String.*format*(connectionStrFormatter, accountName, accountKey);

**if** (LogSetting.***LOG\_BLOB\_WRITER***) {

logger.info("upload accountName = " + accountName);

logger.info("upload accountKey = " + accountKey);

logger.info("upload containerName = " + containerName);

logger.info("upload connectionStr = " + connectionStr);

logger.info("upload blobname = " + blobname);

logger.info("upload blockIdStr = " + blockIdStr);

}

**if** (LogSetting.***LOG\_BLOB\_WRITER\_DATA***) {

logger.info("upload data= \r\n" + data);

}

CloudStorageAccount account = CloudStorageAccount.*parse*(String.*format*(connectionStr, accountName, accountKey));

CloudBlobClient \_blobClient = account.createCloudBlobClient();

CloudBlobContainer \_container = \_blobClient.getContainerReference(containerName);

\_container.createIfNotExists();

CloudBlockBlob blockBlob = \_container.getBlockBlobReference(blobname);

BlobRequestOptions blobOptions = **new** BlobRequestOptions();

stream = **new** ByteArrayInputStream(data.getBytes(StandardCharsets.***UTF\_8***));

BlockEntry newBlock = **new** BlockEntry(Base64.*encode*(blockIdStr.getBytes()), BlockSearchMode.***UNCOMMITTED***);

ArrayList<BlockEntry> blocksBeforeUpload = **new** ArrayList<BlockEntry>();

**if** (blockBlob.exists(AccessCondition.*generateEmptyCondition*(), blobOptions, **null**)) {

blocksBeforeUpload = blockBlob.downloadBlockList(BlockListingFilter.***COMMITTED***, **null**, blobOptions, **null**);

}

**if** (LogSetting.***LOG\_BLOB\_WRITER\_BLOCKLIST\_BEFORE\_UPLOAD***) {

**int** i = 0;

String id = **null**;

**for** (BlockEntry e : blocksBeforeUpload) {

i++;

id = e.getId();

logger.info("BlockEntry Before Upload id=" + id + ", Index = " + i);

}

**if** (id != **null**) {

logger.info("BlockEntry Before Upload id=" + id + ", Index = " + i + " --last before");

}

}

blockBlob.uploadBlock(newBlock.getId(), stream, -1);

**if** (!blocksBeforeUpload.contains(newBlock)) {

blocksBeforeUpload.add(newBlock);

}

**if** (LogSetting.***LOG\_BLOB\_WRITER\_BLOCKLIST\_AFTER\_UPLOAD***) {

**int** i = 0;

String id = **null**;

**for** (BlockEntry e : blocksBeforeUpload) {

i++;

id = e.getId();

logger.info("BlockEntry After Upload id=" + id + ", Index = " + i);

}

**if** (id != **null**) {

logger.info("BlockEntry After Upload id=" + id + ", Index = " + i + " --last after");

}

}

blockBlob.commitBlockList(blocksBeforeUpload);

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

**if** (stream != **null**) {

**try** {

stream.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

**if** (LogSetting.***LOG\_BLOB\_WRITER*** && LogSetting.***LOG\_METHOD\_END***) {

logger.info("upload End");

}

}

**static** **public** **void** remove(Properties properties, String blockIdStrFormat, String blobname, String blockIdStr) {

// remove blocks with blockid >= blockIdStr

Logger logger = (Logger) LoggerFactory.*getLogger*(BlobWriter.**class**);

**try** {

**if** (LogSetting.***LOG\_BLOB\_WRITER*** && LogSetting.***LOG\_METHOD\_BEGIN***) {

logger.info("remove Begin");

}

String accountName = properties.getProperty("storage.blob.account.name");

String accountKey = properties.getProperty("storage.blob.account.key");

String containerName = properties.getProperty("storage.blob.account.container");

String connectionStrFormatter = "DefaultEndpointsProtocol=http;AccountName=%s;AccountKey=%s";

String connectionStr = String.*format*(connectionStrFormatter, accountName, accountKey);

logger.info("accountName = " + accountName);

logger.info("accountKey = " + accountKey);

logger.info("containerName = " + containerName);

logger.info("connectionStr = " + connectionStr);

CloudStorageAccount account = CloudStorageAccount.*parse*(String.*format*(connectionStr, accountName, accountKey));

CloudBlobClient \_blobClient = account.createCloudBlobClient();

CloudBlobContainer \_container = \_blobClient.getContainerReference(containerName);

\_container.createIfNotExists();

CloudBlockBlob blockBlob = \_container.getBlockBlobReference(blobname);

BlobRequestOptions blobOptions = **new** BlobRequestOptions();

ArrayList<BlockEntry> blocksBeforeUpload = **new** ArrayList<BlockEntry>();

**if** (blockBlob.exists(AccessCondition.*generateEmptyCondition*(), blobOptions, **null**)) {

blocksBeforeUpload = blockBlob.downloadBlockList(BlockListingFilter.***COMMITTED***, **null**, blobOptions, **null**);

}

**int** blockid = Integer.*parseInt*(blockIdStr);

**int** size = blocksBeforeUpload.size();

// int size = 50000;

**for** (**int** i = size; i >= blockid; i--) {

String idStr = String.*format*(blockIdStrFormat, i);

BlockEntry entry = **new** BlockEntry(Base64.*encode*(idStr.getBytes()), BlockSearchMode.***UNCOMMITTED***);

**if** (blocksBeforeUpload.contains(entry)) {

blocksBeforeUpload.remove(entry);

}

}

blockBlob.commitBlockList(blocksBeforeUpload);

} **catch** (Exception e) {

e.printStackTrace();

}

**if** (LogSetting.***LOG\_BLOB\_WRITER*** && LogSetting.***LOG\_METHOD\_END***) {

logger.info("remove End");

}

}

}

## Add Redis class to support reading/writing from Azure Redis Cache

Create a new file Redis.java in directory \eventhub-blobwriter\src\main\java\com\contoso\app\trident\

**package** com.contoso.app.trident;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Properties;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** redis.clients.jedis.Jedis;

**public** **class** Redis {

**private** **static** **final** Logger ***logger*** = (Logger) LoggerFactory.*getLogger*(Redis.**class**);

**public** **static** **void** main(String[] args) {

*testList*();

}

**public** **static** **void** testList() {

String host = "hanzredis1.redis.cache.windows.net";

String password = "eQoMISLEQf7mwCDetcvIUT+P9WGGK9KGsdf7/UOGkTg=";

List<String> li = **new** ArrayList<String>();

li.add("1");

li.add("2");

li.add("3");

li.add("4");

Redis.*setList*(host, password, "key", li);

List<String> li1 = **new** ArrayList<String>();

li1.add("1a");

li1.add("2a");

li1.add("3a");

li1.add("4a");

Redis.*setList*(host, password, "key", li1);

List<String> li2 = Redis.*getList*(host, password, "key", 50000);

**for** (String s : li2) {

System.***out***.println(s);

}

}

**static** **public** String getHost(Properties properties) {

**if** (LogSetting.***LOG\_REDIS*** && LogSetting.***LOG\_METHOD\_BEGIN***) {

***logger***.info("getHost Begin");

}

String redisHost = properties.getProperty("redis.host");

***logger***.info("getHost Returns " + redisHost);

***logger***.info("getHost End");

**return** redisHost;

}

**static** **public** String getPassword(Properties properties) {

**if** (LogSetting.***LOG\_REDIS*** && LogSetting.***LOG\_METHOD\_BEGIN***) {

***logger***.info("getPassword Begin");

}

String redisPassword = properties.getProperty("redis.password");

**if** (LogSetting.***LOG\_REDIS*** && LogSetting.***LOG\_METHOD\_END***) {

***logger***.info("getPassword End");

}

**return** redisPassword;

}

**static** **public** **void** flushDB(String host, String password) {

**if** (LogSetting.***LOG\_REDIS*** && LogSetting.***LOG\_METHOD\_BEGIN***) {

***logger***.info("flushDB Begin");

***logger***.info("flushDB params: host= " + host);

}

Jedis jedis = **new** Jedis(host, 6380, 3600, **true**); // host, port, timeout,isSSL

jedis.auth(password);

jedis.connect();

**if** (jedis.isConnected()) {

jedis.flushDB();

} **else** {

**if** (LogSetting.***LOG\_REDIS***) {

***logger***.info("flushDB connection error !!!!!");

}

}

jedis.close();

**if** (LogSetting.***LOG\_REDIS*** && LogSetting.***LOG\_METHOD\_END***) {

***logger***.info("flushDB End");

}

}

**static** **public** String get(String host, String password, String key) {

**if** (LogSetting.***LOG\_REDIS*** && LogSetting.***LOG\_METHOD\_BEGIN***) {

***logger***.info("get Begin");

***logger***.info("get params: host= " + host + " key= " + key);

}

String value = **null**;

Jedis jedis = **new** Jedis(host, 6380, 3600, **true**); // host, port, timeout,isSSL

jedis.auth(password);

jedis.connect();

**if** (jedis.isConnected()) {

value = jedis.get(key);

} **else** {

**if** (LogSetting.***LOG\_REDIS***) {

***logger***.info("get connection error !!!!!");

}

}

jedis.close();

**if** (LogSetting.***LOG\_REDIS*** && LogSetting.***LOG\_METHOD\_END***) {

***logger***.info("get returns " + value);

***logger***.info("get Begin");

}

**return** value;

}

**static** **public** **void** set(String host, String password, String key, String value) {

**if** (LogSetting.***LOG\_REDIS*** && LogSetting.***LOG\_METHOD\_BEGIN***) {

***logger***.info("set Begin");

***logger***.info("set params: host= " + host + " key= " + key + " value= " + value);

}

Jedis jedis = **new** Jedis(host, 6380, 3600, **true**); // host, port, timeout,isSSL

jedis.auth(password);

jedis.connect();

**if** (jedis.isConnected()) {

jedis.set(key, value);

} **else** {

**if** (LogSetting.***LOG\_REDIS***) {

***logger***.info("set connection error !!!!!");

}

}

jedis.close();

**if** (LogSetting.***LOG\_REDIS*** && LogSetting.***LOG\_METHOD\_END***) {

***logger***.info("set End");

}

}

**static** **public** List<String> getList(String host, String password, String key, **int** maxLength) {

**if** (LogSetting.***LOG\_REDIS*** && LogSetting.***LOG\_METHOD\_BEGIN***) {

***logger***.info("getList Begin");

***logger***.info("getList params: host= " + host + " key= " + key + " maxLength= " + maxLength);

}

Jedis jedis = **new** Jedis(host, 6380, 3600, **true**); // host, port, timeout,isSSL

List<String> stringList = **null**;

jedis.auth(password);

jedis.connect();

**if** (jedis.isConnected()) {

stringList = jedis.lrange(key, 0, maxLength - 1);

} **else** {

**if** (LogSetting.***LOG\_REDIS***) {

***logger***.info("getList connection error !!!!!");

}

}

jedis.close();

**for** (String s : stringList) {

***logger***.info("getList returns: " + s);

}

**if** (LogSetting.***LOG\_REDIS*** && LogSetting.***LOG\_METHOD\_END***) {

***logger***.info("getList End");

}

**return** stringList;

}

**static** **public** **void** setList(String host, String password, String key, List<String> stringList) {

**if** (LogSetting.***LOG\_REDIS*** && LogSetting.***LOG\_METHOD\_BEGIN***) {

***logger***.info("setList Begin");

***logger***.info("getList params: host= " + host + " key= " + key);

}

**if** (stringList == **null** || stringList.isEmpty()) {

**if** (LogSetting.***LOG\_REDIS***) {

***logger***.info("setList params stringList is empty !!!!!!");

}

} **else** {

**if** (LogSetting.***LOG\_REDIS***) {

**for** (String s : stringList) {

***logger***.info("setList params stringlist: " + s);

}

}

Jedis jedis = **new** Jedis(host, 6380, 3600, **true**); // host, port, timeout,isSSL

jedis.auth(password);

jedis.connect();

**if** (jedis.isConnected()) {

jedis.del(key);

**for** (String str : stringList) {

jedis.lpush(key, str);

}

} **else** {

**if** (LogSetting.***LOG\_REDIS***) {

***logger***.info("setList connection error !!!!!");

}

}

jedis.close();

}

**if** (LogSetting.***LOG\_REDIS*** && LogSetting.***LOG\_METHOD\_END***) {

***logger***.info("setList End");

}

}

}

## Add ByteAggregator class to perform partitionAggregate operation

Create a new file ByteAggregator.java in directory \eventhub-blobwriter\src\main\java\com\contoso\app\trident\

**package** com.contoso.app.trident;

**import** java.util.Map;

**import** java.util.Properties;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** backtype.storm.topology.FailedException;

**import** backtype.storm.tuple.Values;

**import** storm.trident.operation.BaseAggregator;

**import** storm.trident.operation.TridentCollector;

**import** storm.trident.operation.TridentOperationContext;

**import** storm.trident.topology.TransactionAttempt;

**import** storm.trident.tuple.TridentTuple;

@SuppressWarnings("unused")

**public** **class** ByteAggregator **extends** BaseAggregator<BlockState> {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** **static** **final** Logger ***logger*** = (Logger) LoggerFactory.*getLogger*(ByteAggregator.**class**);

**private** **long** txid;

**private** **int** partitionIndex;

**private** Properties properties;

**public** ByteAggregator(Properties properties) {

***logger***.info("Constructor Begin");

**this**.properties = properties;

***logger***.info("Constructor End");

}

@Override

**public** **void** prepare(@SuppressWarnings("rawtypes") Map conf, TridentOperationContext context) {

**if** (LogSetting.***LOG\_INSTANCE*** && LogSetting.***LOG\_METHOD\_BEGIN***) {

***logger***.info("prepare Begin");

}

**this**.partitionIndex = context.getPartitionIndex();

**super**.prepare(conf, context);

**if** (LogSetting.***LOG\_INSTANCE*** && LogSetting.***LOG\_METHOD\_END***) {

***logger***.info("p" + **this**.partitionIndex + ": prepare End");

}

}

**public** BlockState init(Object batchId, TridentCollector collector) {

**if** (LogSetting.***LOG\_BATCH*** && LogSetting.***LOG\_METHOD\_BEGIN***) {

***logger***.info("p" + **this**.partitionIndex + ": init End");

}

**if** (batchId **instanceof** TransactionAttempt) {

**this**.txid = ((TransactionAttempt) batchId).getTransactionId();

}

BlockState state = **new** BlockState(**this**.partitionIndex, **this**.txid, **this**.properties);

// BlobWriter.remove(this.properties, state.blockIdStrFormat, state.block.blobname, state.block.blockidStr);

**if** (LogSetting.***LOG\_BATCH*** && LogSetting.***LOG\_METHOD\_END***) {

***logger***.info(state.partition\_tx\_logStr + "init End");

}

**return** state;

}

**public** **void** aggregate(BlockState state, TridentTuple tuple, TridentCollector collector) {

**if** (LogSetting.***LOG\_MESSAGE*** && LogSetting.***LOG\_METHOD\_BEGIN***) {

***logger***.info(state.partition\_tx\_logStr + "aggregate Begin");

}

String tupleStr = tuple.getString(0);

**if** (tupleStr != **null** && tupleStr.length() > 0) {

String msg = tupleStr + "\r\n";

**if** (state.block.isMessageSizeWithnLimit(msg)) {

**if** (state.block.willMessageFitCurrentBlock(msg)) {

state.block.addData(msg);

} **else** {

// since the new msg will not fit into the current block, we will upload the current block,

// and then get the next block, and add the new msg to the next block

state.block.upload(**this**.properties);

state.needPersist = **true**;

**if** (LogSetting.***LOG\_BLOCK\_ROLL\_OVER***) {

***logger***.info(state.partition\_tx\_logStr + "Roll over from : blobname = " + state.block.blobname + ", blockid = " + state.block.blockid);

}

state.block = state.block.next();

**if** (LogSetting.***LOG\_BLOCK\_ROLL\_OVER***) {

***logger***.info(state.partition\_tx\_logStr + "Roll over to: blobname = " + state.block.blobname + ", blockid = " + state.block.blockid);

}

state.block.addData(msg);

}

} **else** {

// message size is not within the limit, skip the message

}

}

**if** (LogSetting.***LOG\_MESSAGE*** && LogSetting.***LOG\_METHOD\_END***) {

***logger***.info(state.partition\_tx\_logStr + "aggregate End");

}

}

**public** **void** complete(BlockState state, TridentCollector collector) {

**if** (LogSetting.***LOG\_BATCH*** && LogSetting.***LOG\_METHOD\_BEGIN***) {

***logger***.info(state.partition\_tx\_logStr + "complete Begin");

}

**if** (state.block.blockdata.length() > 0) {

state.block.upload(**this**.properties); // upload the last block in the batch

state.needPersist = **true**;

}

**if** (state.needPersist) {

state.persist();

}

collector.emit(**new** Values(1)); // just emit a value

**if** (LogSetting.***LOG\_BATCH*** && LogSetting.***LOG\_METHOD\_END***) {

***logger***.info(state.partition\_tx\_logStr + "complete End");

}

}

}

## Add BlobState class

Create a new file BlobState.java in directory \eventhub-blobwriter\src\main\java\com\contoso\app\trident\

package com.contoso.app.trident;

import java.util.ArrayList;

import java.util.List;

import java.util.Properties;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@SuppressWarnings("unused")

public class BlockState {

public Block block;

public boolean needPersist = false;

private int maxNumberOfBlocks = 6;

private static final String blobidBlockidStrFormat = "%05d\_%05d";

private static final String blobNameFormat = "aaa/blobwriter/%05d/%05d";

public String blockIdStrFormat = "%05d";

private static final Logger logger = (Logger) LoggerFactory.getLogger(BlockState.class);

private String redisHost = null;

private String redisPassword = null;

private String key\_partition\_txid;

private String key\_partitionBlocklist;

private int partitionIndex;

private long txid;

private List<String> blocklist;

private int maxBlockBytes;// 1024 \* 1024 \* 4 / 16;

String partition\_tx\_logStr;

public BlockState(int partitionIndex, long txid, Properties properties) {

this.partition\_tx\_logStr = "p" + partitionIndex + "\_tx" + txid + ": ";

if ((LogSetting.LOG\_INSTANCE || LogSetting.LOG\_BLOCK) && LogSetting.LOG\_METHOD\_BEGIN) {

logger.info(partition\_tx\_logStr + "Constructor Begin");

}

this.partitionIndex = partitionIndex;

this.txid = txid;

this.maxNumberOfBlocks = getMaxNumberOfblocks(properties);

this.maxBlockBytes = getMaxBlockBytes(properties);

redisHost = Redis.getHost(properties);

redisPassword = Redis.getPassword(properties);

this.key\_partition\_txid = "p\_" + String.valueOf(partitionIndex) + "\_txid";

this.key\_partitionBlocklist = "p\_" + String.valueOf(partitionIndex) + "\_blocklist";

this.blocklist = new ArrayList<String>();

String lastTxidStr = Redis.get(redisHost, redisPassword, this.key\_partition\_txid);

if (lastTxidStr == null) { // the very first time the topology is running

this.block = getNewBlock();

} else {

long lastTxid = Long.parseLong(lastTxidStr);

if (txid != lastTxid) { // a new batch, not a replay

Block lastblock = getLastBlockInLastFailedBatch();

this.block = lastblock.next();

} else {// if(txid == lastTxid) a replay, overwrite old block

this.block = getFirstBlockInLastFailedBatch();

}

}

if ((LogSetting.LOG\_INSTANCE || LogSetting.LOG\_BLOCK) && LogSetting.LOG\_METHOD\_END) {

logger.info(this.partition\_tx\_logStr + "Constructor End with blobid=" + this.block.blobid + ", blockid=" + this.block.blockid);

logger.info(this.partition\_tx\_logStr + "Constructor End");

}

}

public void persist() {

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_BEGIN) {

logger.info(this.partition\_tx\_logStr + "persist Begin");

}

Redis.set(redisHost, redisPassword, this.key\_partition\_txid, String.valueOf(this.txid));

Redis.setList(redisHost, redisPassword, this.key\_partitionBlocklist, this.blocklist);

if (LogSetting.LOG\_PERSIST) {

logger.info(this.partition\_tx\_logStr + "set(" + this.key\_partition\_txid + ", " + this.txid + ")");

for (String s : this.blocklist) {

logger.info(this.partition\_tx\_logStr + "addToList(" + this.key\_partitionBlocklist + ", " + s + ")");

}

}

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_END) {

logger.info(this.partition\_tx\_logStr + "this.partition\_tx\_logStr + persist End");

}

}

private Block getNewBlock() {

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_BEGIN) {

logger.info(this.partition\_tx\_logStr + "getNewBlock Begin");

}

Block block = new Block();

block.build();

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_END) {

logger.info(this.partition\_tx\_logStr + "getNewBlock End");

}

return block;

}

private Block getLastBlockInLastFailedBatch() {

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_BEGIN) {

logger.info(this.partition\_tx\_logStr + "getLastBlockInLastFailedBatch Begin");

}

Block block = new Block();

List<String> lastBlobidBlockidList = Redis.getList(redisHost, redisPassword, this.key\_partitionBlocklist, 50000);

if (lastBlobidBlockidList != null && lastBlobidBlockidList.size() > 0) {

String blockStr = lastBlobidBlockidList.get(0);

for (String s : lastBlobidBlockidList) {

if (s.compareTo(blockStr) > 0) {// find the last block written in the last batch

blockStr = s;

}

}

String[] strArray = blockStr.split("\_");

block.blobid = Integer.parseInt(strArray[0]);

block.blockid = Integer.parseInt(strArray[1]);

if (LogSetting.LOG\_GET\_LAST\_BLOCK) {

logger.info(this.partition\_tx\_logStr + "Last record in List(" + this.key\_partitionBlocklist + "): " + blockStr);

logger.info(this.partition\_tx\_logStr + "Last record in List( " + this.key\_partitionBlocklist + "): blobid=" + block.blobid + ", blockid = "

+ block.blockid);

}

} else {

if (LogSetting.LOG\_GET\_LAST\_BLOCK) {

logger.info(this.partition\_tx\_logStr + "List(" + this.key\_partitionBlocklist + ") is null or empty");

}

}

block.build();

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_END) {

logger.info(this.partition\_tx\_logStr + "getLastBlockInLastFailedBatch returns blobid=" + block.blobid + ", blockid=" + block.blockid);

logger.info(this.partition\_tx\_logStr + "getLastBlockInLastFailedBatch End");

}

return block;

}

private Block getFirstBlockInLastFailedBatch() {

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_BEGIN) {

logger.info(this.partition\_tx\_logStr + "getFirstBlockInLastFailedBatch Begin");

}

Block block = new Block();

List<String> lastblocks = Redis.getList(redisHost, redisPassword, this.key\_partitionBlocklist, 50000);

if (lastblocks != null && lastblocks.size() > 0) {

String blockStr = lastblocks.get(0);

for (String s : lastblocks) {

if (s.compareTo(blockStr) < 0) {// find the first block written in the last batch

blockStr = s;

}

}

String[] strArray = blockStr.split("\_");

block.blobid = Integer.parseInt(strArray[0]);

block.blockid = Integer.parseInt(strArray[1]);

if (LogSetting.LOG\_GET\_FIRST\_BLOCK) {

logger.info(this.partition\_tx\_logStr + "First record in List(" + this.key\_partitionBlocklist + "): " + blockStr);

logger.info(this.partition\_tx\_logStr + "First record in List(" + this.key\_partitionBlocklist + "): blobid=" + block.blobid + ", blockid = "

+ block.blockid);

}

} else {

logger.info(this.partition\_tx\_logStr + "List(" + this.key\_partitionBlocklist + ") is null or empty");

}

block.build();

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_END) {

logger.info(this.partition\_tx\_logStr + "getFirstBlockInLastFailedBatch returns blobid=" + block.blobid + ", blockid=" + block.blockid);

logger.info(this.partition\_tx\_logStr + "getFirstBlockInLastFailedBatch End");

}

return block;

}

private int getMaxNumberOfblocks(Properties properties) {

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_BEGIN) {

logger.info(this.partition\_tx\_logStr + "getMaxNumberOfblocks Begin");

}

int maxNumberOfBlocks = 10;

String maxNumberOfBlocksStr = properties.getProperty("storage.blob.block.number.max");

if (maxNumberOfBlocksStr != null) {

maxNumberOfBlocks = Integer.parseInt(maxNumberOfBlocksStr);

}

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_END) {

logger.info(this.partition\_tx\_logStr + "getMaxNumberOfblocks returns " + maxNumberOfBlocks);

logger.info(this.partition\_tx\_logStr + "getMaxNumberOfblocks End");

}

return maxNumberOfBlocks;

}

private int getMaxBlockBytes(Properties properties) {

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_BEGIN) {

logger.info(this.partition\_tx\_logStr + "getMaxBlockBytes Begin");

}

int maxBlockBytes = 1024;

String maxBlockBytesStr = properties.getProperty("storage.blob.block.bytes.max");

if (maxBlockBytesStr != null) {

maxBlockBytes = Integer.parseInt(maxBlockBytesStr);

}

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_END) {

logger.info(this.partition\_tx\_logStr + "getMaxBlockBytes returns " + maxBlockBytes);

logger.info(this.partition\_tx\_logStr + "getMaxBlockBytes End");

}

return maxBlockBytes;

}

public class Block {

public int blobid = 1;

public int blockid = 1;

public String blobname;

public String blockidStr;

public String blockdata;

public Block() {

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_BEGIN) {

logger.info(BlockState.this.partition\_tx\_logStr + "Block Constructor Begin");

}

this.blobid = 1;

this.blockid = 1;

this.blockdata = "";

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_END) {

logger.info(BlockState.this.partition\_tx\_logStr + "Block Constructor End");

}

}

public void addData(String msg) {

if (LogSetting.LOG\_MESSAGE && LogSetting.LOG\_METHOD\_BEGIN) {

logger.info(BlockState.this.partition\_tx\_logStr + "Block.addData Begin");

}

this.blockdata = this.blockdata + msg;

if (LogSetting.LOG\_MESSAGE && LogSetting.LOG\_METHOD\_END) {

logger.info(BlockState.this.partition\_tx\_logStr + "Block.addData End");

}

}

public boolean isMessageSizeWithnLimit(String msg) {

if (LogSetting.LOG\_MESSAGE && LogSetting.LOG\_METHOD\_BEGIN) {

logger.info(BlockState.this.partition\_tx\_logStr + "Block.isMessageSizeWithnLimit Begin");

}

boolean result = false;

if (msg.getBytes().length <= maxBlockBytes) {

result = true;

}

if (LogSetting.LOG\_MESSAGE && LogSetting.LOG\_METHOD\_END) {

logger.info(BlockState.this.partition\_tx\_logStr + "Block.isMessageSizeWithnLimit End");

}

return result;

}

public boolean willMessageFitCurrentBlock(String msg) {

if (LogSetting.LOG\_MESSAGE && LogSetting.LOG\_METHOD\_BEGIN) {

logger.info(BlockState.this.partition\_tx\_logStr + "Block.willMessageFitCurrentBlock Begin");

}

boolean result = false;

int byteSize = (this.blockdata + msg).getBytes().length;

if (byteSize <= maxBlockBytes) {

result = true;

}

if (LogSetting.LOG\_MESSAGE && LogSetting.LOG\_METHOD\_END) {

logger.info(BlockState.this.partition\_tx\_logStr + "Block.willMessageFitCurrentBlock End");

}

return result;

}

public void upload(Properties properties) {

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_BEGIN) {

logger.info(BlockState.this.partition\_tx\_logStr + "Block.upload Begin");

}

BlobWriter.upload(properties, this.blobname, this.blockidStr, this.blockdata);

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_END) {

logger.info(BlockState.this.partition\_tx\_logStr + "BlobState.upload End");

}

}

private void build() {

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_BEGIN) {

logger.info(BlockState.this.partition\_tx\_logStr + "Block.build Begin");

}

this.blockdata = new String("");

this.blobname = String.format(BlockState.blobNameFormat, BlockState.this.partitionIndex, this.blobid);

this.blockidStr = String.format(BlockState.this.blockIdStrFormat, this.blockid);

String blobidBlockidStr = String.format(BlockState.blobidBlockidStrFormat, this.blobid, this.blockid);

BlockState.this.blocklist.add(blobidBlockidStr);

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_END) {

logger.info(BlockState.this.partition\_tx\_logStr + "Block.build End");

}

}

public Block next() {

Block current = this;

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_BEGIN) {

logger.info(BlockState.this.partition\_tx\_logStr + "Block.next Begin");

}

Block nextBlock = new Block();

if (current.blockid < maxNumberOfBlocks) {

nextBlock.blobid = current.blobid;

nextBlock.blockid = current.blockid + 1;

} else {

nextBlock.blobid = current.blobid + 1;

nextBlock.blockid = 1;

}

nextBlock.build();

if (LogSetting.LOG\_BLOCK && LogSetting.LOG\_METHOD\_END) {

logger.info(BlockState.this.partition\_tx\_logStr + "Block.next returns blobid=" + nextBlock.blobid + ", blockid=" + nextBlock.blockid);

logger.info(BlockState.this.partition\_tx\_logStr + "Block.next End");

}

return nextBlock;

}

}

}

## Add BlobWriterTopology class:

Create a new file BlobWriterTopology.java in directory \eventhub-blobwriter\src\main\java\com\contoso\app\trident\

package com.contoso.app.trident;

import java.util.Properties;

import backtype.storm.Config;

import backtype.storm.LocalCluster;

import backtype.storm.StormSubmitter;

import backtype.storm.generated.StormTopology;

import backtype.storm.tuple.Fields;

import com.microsoft.eventhubs.spout.EventHubSpoutConfig;

import com.microsoft.eventhubs.trident.OpaqueTridentEventHubSpout;

import storm.trident.Stream;

import storm.trident.TridentTopology;

public class BlobWriterTopology {

static Properties properties = null;

static int numWorkers = 1;

public static void main(String[] args) throws Exception {

System.out.println("Loding properties from Config.properties");

properties = new Properties();

properties.load(BlobWriterTopology.class.getClassLoader().getResourceAsStream("Config.properties"));

// properties.load(new FileReader(args[1])); // load properties from a different file

if ((args != null) && (args.length > 0)) { // if running in storm cluster, the first argument is the topology name

String topologyName = args[0];

StormTopology stormTopology = buildTopology(topologyName);

Config config = new Config();

numWorkers = Integer.parseInt(properties.getProperty("eventhubspout.partitions.count"));

config.setNumWorkers(numWorkers);

System.out.println("Number of workers = " + numWorkers);

StormSubmitter.submitTopology(topologyName, config, stormTopology);

} else {// if running in local development environment, there is no argument for topology name

String topologyName = "localTopology";

StormTopology stormTopology = buildTopology(topologyName);

Config config = new Config();

config.setMaxTaskParallelism(10);

LocalCluster localCluster = new LocalCluster();

localCluster.submitTopology(topologyName, config, stormTopology);

Thread.sleep(5000000L);

localCluster.shutdown();

}

}

static StormTopology buildTopology(String topologyName) {

Redis.flushDB(Redis.getHost(properties), Redis.getPassword(properties));

TridentTopology tridentTopology = new TridentTopology();

Stream inputStream = null;

// TestSpout spout = new TestSpout();

OpaqueTridentEventHubSpout spout = createOpaqueTridentEventHubSpout(topologyName);

inputStream = tridentTopology.newStream("message", spout);

inputStream.parallelismHint(numWorkers).partitionAggregate(new Fields("message"), new ByteAggregator(properties), new Fields("blobname"));

return tridentTopology.build();

}

static OpaqueTridentEventHubSpout createOpaqueTridentEventHubSpout(String topologyName) {

EventHubSpoutConfig spoutConfig = readConfig();

spoutConfig.setTopologyName(topologyName);

OpaqueTridentEventHubSpout spout = new OpaqueTridentEventHubSpout(spoutConfig);

return spout;

}

static EventHubSpoutConfig readConfig() {

EventHubSpoutConfig spoutConfig;

String username = properties.getProperty("eventhubspout.username");

String password = properties.getProperty("eventhubspout.password");

String namespaceName = properties.getProperty("eventhubspout.namespace");

String entityPath = properties.getProperty("eventhubspout.entitypath");

String zkEndpointAddress = properties.getProperty("zookeeper.connectionstring");

int partitionCount = Integer.parseInt(properties.getProperty("eventhubspout.partitions.count"));

int checkpointIntervalInSeconds = Integer.parseInt(properties.getProperty("eventhubspout.checkpoint.interval"));

int receiverCredits = Integer.parseInt(properties.getProperty("eventhub.receiver.credits"));

System.out.println("Eventhub spout config: ");

System.out.println(" username: " + username);

System.out.println(" password: " + password);

System.out.println(" namespaceName: " + namespaceName);

System.out.println(" entityPath: " + entityPath);

System.out.println(" zkEndpointAddress: " + zkEndpointAddress);

System.out.println(" partition count: " + partitionCount);

System.out.println(" checkpoint interval: " + checkpointIntervalInSeconds);

System.out.println(" receiver credits: " + receiverCredits);

spoutConfig = new EventHubSpoutConfig(username, password, namespaceName, entityPath, partitionCount, zkEndpointAddress, checkpointIntervalInSeconds,

receiverCredits);

return spoutConfig;

}

}

## Add LogSetting class

Create a new file LogSetting.java in directory \eventhub-blobwriter\src\main\java\com\contoso\app\trident\

**package** com.contoso.app.trident;

**public** **class** LogSetting {

**public** **static** **final** **boolean** ***LOG\_MESSAGE*** = **false**;

**public** **static** **final** **boolean** ***LOG\_BATCH*** = **true**;

**public** **static** **final** **boolean** ***LOG\_INSTANCE*** = **true**;

**public** **static** **final** **boolean** ***LOG\_BLOCK\_ROLL\_OVER*** = **false**;

**public** **static** **final** **boolean** ***LOG\_BLOCK*** = **false**;

**public** **static** **final** **boolean** ***LOG\_PERSIST*** = **true**;

**public** **static** **final** **boolean** ***LOG\_GET\_LAST\_BLOCK*** = **true**;

**public** **static** **final** **boolean** ***LOG\_GET\_FIRST\_BLOCK*** = **true**;

**public** **static** **final** **boolean** ***LOG\_BLOB\_WRITER*** = **false**;

**public** **static** **final** **boolean** ***LOG\_BLOB\_WRITER\_DATA*** = **false**;

**public** **static** **final** **boolean** ***LOG\_BLOB\_WRITER\_BLOCKLIST\_BEFORE\_UPLOAD*** = **false**;

**public** **static** **final** **boolean** ***LOG\_BLOB\_WRITER\_BLOCKLIST\_AFTER\_UPLOAD*** = **false**;

**public** **static** **final** **boolean** ***LOG\_REDIS*** = **true**;

**public** **static** **final** **boolean** ***LOG\_METHOD\_BEGIN*** = **false**;

**public** **static** **final** **boolean** ***LOG\_METHOD\_END*** = **false**;

}

# Test the topology with Event Hub

## Create .NET Console to send messages to Event Hub

Create a .NET console application to generate events for 10 devices every second, until you stop the application by pressing a key.

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading.Tasks;

using Microsoft.ServiceBus.Messaging;

using Newtonsoft.Json;

using Microsoft.ServiceBus;

using System.Threading;

using System.Runtime.Serialization;

namespace SendEvents

{

class Program

{

static int numberOfDevices = 1000;

static string eventHubName = "hanzeventhub1";

static string eventHubNamespace = "hanzeventhub1-ns";

static string sharedAccessPolicyName = "devices";

static string sharedAccessPolicyKey = "8Aw7zeko+0VB1Or335RT/98LALc/58suW3ZsXtzOwbQ=";

static void Main(string[] args)

{

var settings = new MessagingFactorySettings()

{

TokenProvider = TokenProvider.CreateSharedAccessSignatureTokenProvider(sharedAccessPolicyName, sharedAccessPolicyKey),

TransportType = TransportType.Amqp

};

var factory = MessagingFactory.Create(ServiceBusEnvironment.CreateServiceUri("sb", eventHubNamespace, ""), settings);

EventHubClient client = EventHubClient.CreateFromConnectionString("Endpoint=sb://hanzeventhub1-ns.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=VremOpcEIYzpxXLIkqjgzT2ZJXBVdTSYxFhkRW6SiY8=", eventHubName);

try

{

List<Task> tasks = new List<Task>();

Console.WriteLine("Sending messages to Event Hub {0}", client.Path);

Random random = new Random();

while (!Console.KeyAvailable)

{

// One event per device

for (int devices = 0; devices < numberOfDevices; devices++)

{

// Create the event

Event info = new Event()

{

lat = -30 + random.Next(75),

lng = -120+random.Next(70),

time = DateTime.UtcNow.Ticks,

diagnosisCode = (310 + random.Next(20)).ToString()

};

// Serialize to JSON

var serializedString = JsonConvert.SerializeObject(info);

Console.WriteLine(serializedString);

EventData data = new EventData(Encoding.UTF8.GetBytes(serializedString))

{

PartitionKey = info.diagnosisCode

};

// Send the message to Event Hub

tasks.Add(client.SendAsync(data));

}

//Thread.Sleep(1000);

};

Task.WaitAll(tasks.ToArray());

}

catch (Exception exp)

{

Console.WriteLine("Error on send: " + exp.Message);

}

}

}

[DataContract]

public class Event

{

[DataMember]

public double lat { get; set; }

[DataMember]

public double lng { get; set; }

[DataMember]

public long time { get; set; }

[DataMember]

public string diagnosisCode { get; set; }

}

}

## Test the topology locally with event hub

To compile and test the file on your development machine, use the following steps.

* Start the SendEvent .NET application to begin sending events, so that we have something to read from Event Hub.
* Start the topology locally using the following command
* mvn compile exec:java -Dstorm.topology=com.contoso.app.trident.BlobWriterTopology
* This will start the topology, read messages from Event Hub, and upload them to azure blob storage
* After verifying that this works, stop the topology by entering Ctrl-C. To stop the SendEvent app, select the window and press any key.

## Test the topology in HDInsight Storm with event hub

On your development environment, use the following steps to run the Temperature topology on your HDInsight Storm Cluster.

1. Use the following command to create a JAR package from your project.

mvn package

This will create a file named eventhub-blobwriter-1.0-SNAPSHOT.jar in the target directory of your project.

1. On your local development machine, start the SendEvents .NET application, so that we have some events to read.
2. Connect to your HDInsight Storm cluster using Remote Desktop, and copy the jar file to the c:\apps\dist\storm<version number> directory.
3. Use the HDInsight Command Line icon on the cluster desktop to open a new command prompt, and use the following commands to run the topology.

cd %storm\_home%

bin\storm jar eventhub-blobwriter-1.0-SNAPSHOT.jar com.contoso.app.trident.BlobWriterTopology BlobWriterTopology

1. To stop the topology, go to the Remote Desktop session with the Storm cluster and enter the following in the HDInsight Command Line.
2. bin\storm kill BlobWriterTopology

# Test the topology without Event Hub

## Add Test Classes for TestSpout

### Add TestSpout class

Create a new file TestSpout.java in directory \eventhub-blobwriter\src\main\java\com\contoso\app\trident\

**package** com.contoso.app.trident;

**import** backtype.storm.task.TopologyContext;

**import** backtype.storm.tuple.Fields;

**import** storm.trident.spout.ITridentSpout;

**import** java.util.Map;

@SuppressWarnings("rawtypes")

**public** **class** TestSpout **implements** ITridentSpout<Long> {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

BatchCoordinator<Long> coordinator = **new** TestCoordinator();

Emitter<Long> emitter = **new** TestEmitter();

@Override

**public** BatchCoordinator<Long> getCoordinator(String txStateId, Map conf, TopologyContext context) {

**return** coordinator;

}

@Override

**public** Emitter<Long> getEmitter(String txStateId, Map conf, TopologyContext context) {

**return** emitter;

}

@Override

**public** Map getComponentConfiguration() {

**return** **null**;

}

@Override

**public** Fields getOutputFields() {

**return** **new** Fields("message");

}

}

### Add TestCoordinator class

* Create a new file TestCoordinator.java in directory \eventhub-blobwriter\src\main\java\com\contoso\app\trident\

**package** com.contoso.app.trident;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** storm.trident.spout.ITridentSpout.BatchCoordinator;

**import** java.io.Serializable;

**public** **class** TestCoordinator **implements** BatchCoordinator<Long>, Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** **static** **final** Logger ***LOG*** = LoggerFactory.*getLogger*(TestCoordinator.**class**);

@Override

**public** **boolean** isReady(**long** txid) {

**return** **true**;

}

@Override

**public** **void** close() {

}

@Override

**public** Long initializeTransaction(**long** txid, Long prevMetadata, Long currMetadata) {

***LOG***.info("Initializing Transaction [" + txid + "]");

**return** **null**;

}

@Override

**public** **void** success(**long** txid) {

***LOG***.info("Successful Transaction [" + txid + "]");

}

}

### Add TestEmitter class

* Create a new file TestEmitter.java in directory \eventhub-blobwriter\src\main\java\com\contoso\app\trident\

**package** com.contoso.app.trident;

**import** com.google.gson.Gson;

**import** storm.trident.operation.TridentCollector;

**import** storm.trident.spout.ITridentSpout.Emitter;

**import** storm.trident.topology.TransactionAttempt;

**import** java.io.Serializable;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.concurrent.atomic.AtomicInteger;

**public** **class** TestEmitter **implements** Emitter<Long>, Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

AtomicInteger successfulTransactions = **new** AtomicInteger(0);

@Override

**public** **void** emitBatch(TransactionAttempt tx, Long coordinatorMeta, TridentCollector collector) {

**for** (**int** i = 0; i < 10000; i++) {

List<Object> eventJsons = **new** ArrayList<Object>();

**double** lat = **new** Double(-30 + (**int**) (Math.*random*() \* 75));

**double** lng = **new** Double(-120 + (**int**) (Math.*random*() \* 70));

**long** time = System.*currentTimeMillis*();

String diag = **new** Integer(320 + (**int**) (Math.*random*() \* 7)).toString();

EventHubMessage event = **new** EventHubMessage(lat, lng, time, diag);

String eventJson = **new** Gson().toJson(event);

eventJsons.add(eventJson);

collector.emit(eventJsons);

}

}

@Override

**public** **void** success(TransactionAttempt tx) {

successfulTransactions.incrementAndGet();

}

@Override

**public** **void** close() {

}

}

## Test the topology locally without using Event Hub

To compile and test the file on your development machine without using Event hub, use the following steps.

* Open file javaBlobWriterTopoloty.java.:
* Uncomment the line under method buildTopology (String topologyName)

// TestSpout spout = new TestSpout();

* Comment the line

OpaqueTridentEventHubSpout spout = createOpaqueTridentEventHubSpout(topologyName);

* Start the topology locally using the following command
* mvn compile exec:java -Dstorm.topology=com.contoso.app.trident.BlobWriterTopology
* This will start the topology, read messages from test spout, and upload them to azure blob storage

## Test the topology in HDInsight Storm with event hub

On your development environment, use the following steps to run the Temperature topology on your HDInsight Storm Cluster.

1. Use the following command to create a JAR package from your project.

mvn package

This will create a file named eventhub-blobwriter-1.0-SNAPSHOT.jar in the target directory of your project.

1. Connect to your HDInsight Storm cluster using Remote Desktop, and copy the jar file to the c:\apps\dist\storm<version number> directory.
2. Use the HDInsight Command Line icon on the cluster desktop to open a new command prompt, and use the following commands to run the topology.

cd %storm\_home%

bin\storm jar eventhub-blobwriter-1.0-SNAPSHOT.jar com.contoso.app.trident.BlobWriterTopology BlobWriterTopology

1. To stop the topology, go to the Remote Desktop session with the Storm cluster and enter the following in the HDInsight Command Line.
2. bin\storm kill BlobWriterTopology