# Ignite On AWS

Table of Contents

[Ignite on AWS 1](#_Toc22548978)

[What is Apache Ignite: - 1](#_Toc22548979)

[Setup Ec2 instances: - 1](#_Toc22548980)

[AWS Static IP Finder file: - 1](#_Toc22548981)

[Public IP Address of Ec2 Instances: - below are the public ip address of ec2 instances which are launched in aws console for ignite cluster, this can be a comma separated value too. 2](#_Toc22548982)

[Download Ignite artifact: - 2](#_Toc22548983)

[Apache Ignite JDBC Thin client: - 2](#_Toc22548984)

[Apache Ignite SQL Example: - 2](#_Toc22548985)

[Apache Ignite + Spark: - 3](#_Toc22548986)

[Reading Spark Data Frame to Ignite: - 3](#_Toc22548987)

[Write Spark Data Frame to Ignite: - 4](#_Toc22548988)

[Full Code: - 4](#_Toc22548989)

[Maven Dependency: - 9](#_Toc22548990)

[GitHub Link: - 11](#_Toc22548991)

[Apache Ignite Resources and Training and Blogs: - 11](#_Toc22548992)

[Apache Ignite Real World Scenario: - 11](#_Toc22548993)

## What is Apache Ignite: -

Apache Ignite is a memory-centric distributed database, caching, and processing platform for transactional, analytical, and streaming workloads delivering in-memory speeds at petabyte scale

## Setup Ec2 instances: -

Below link provides one easier way to setup ignite on AWS

<https://www.gridgain.com/docs/8.7.6//installation-guide/manual-install-on-ec2>

steps mentioned in the above link should be followed for all the number of nodes which are about to get configured on aws which is missing the weblink, let’s say we are launching 5 nodes to launch ignite the same steps should be followed for all 5 instances.

AWS Static IP Finder file: - there is a missing piece for the static ip finder xml document below is something which is working, missing piece is highlighted.

**<beans xmlns="http://www.springframework.org/schema/beans"**

**xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xsi:schemaLocation="**

**http://www.springframework.org/schema/beans**

**http://www.springframework.org/schema/beans/spring-beans.xsd">**

**<bean class="org.apache.ignite.configuration.IgniteConfiguration" >**

**<!-- other properties -->**

**<!-- Explicitly configure TCP discovery SPI to provide a list of nodes. -->**

**<property name="discoverySpi">**

**<bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">**

**<property name="ipFinder">**

**<bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.vm.TcpDiscoveryVmIpFinder">**

**<property name="addresses">**

**<list>**

**<value>172.31.82.246</value>**

**<value>172.31.92.157</value>**

**</list>**

**</property>**

**</bean>**

**</property>**

**</bean>**

**</property>**

**</bean>**

**</beans>**

### Public IP Address of Ec2 Instances: - below are the public ip address of ec2 instances which are launched in aws console for ignite cluster, this can be a comma separated value too.

**Example 1: -**

**<value>172.31.82.246</value>**

**<value>172.31.92.157</value>**

**Comma separated values Example 2: -**

**<value>172.31.82.246,172.31.92.157</value>**

## Download Ignite artifact: -

Below is the weblink to download community edition of Grid Gain ignite artifact which is used to run and create ignite cluster.

<https://www.gridgain.com/resources/download/gce-thank-you?ggType=GCE&file=gridgain-community-8.7.6.zip>

## Apache Ignite JDBC Thin client: -

After launching ignite cluster one way to connect ignite is to use sqline.sh script using JDBC thin client which lets users connect ignite as a client.

Sqline.sh script is found in grid gain community edition artifact file under bin folder, below is one example.

**gridgain-community-8.7.6/bin/sqlline.sh**

**Example: -** below highlighted part is public ip address of any ec2 ignite cluster node.

**./sqlline.sh -u jdbc:ignite:thin://18.206.247.40**

## Apache Ignite SQL Example: -

There is a sql example inside ignite artifact which users can run it using .sqlline.sh

Example: -

gridgain-community-8.7.6/examples/sql

<https://medium.com/@sdarlington/loading-data-into-apache-ignite-c0cb7c065a7>

## Apache Ignite + Spark: -

Apache spark + ignite is a great combination for in memory real time computation, I will explain how to read apache ignite table and write spark data frames to Ignite.

Below JDBC thin client address is private address of any ec2 node of Apache ignite cluster.

Note: - JDBC thin client is the only working approach to connect apache ignite from outside the ignite cluster like a client.

### Reading Spark Data Frame to Ignite: -

**Step 1 Setup some example data and create and write to Ignite Table: -**

def setupExampleData = {  
  
 val cfg2 = new ClientConfiguration().setAddresses("35.171.27.68:10800")  
 val igniteClient:IgniteClient = Ignition.*startClient*(cfg2)  
  
 System.*out*.format(">>> Created cache [%s].\n", *CACHE\_NAME*)  
  
 val cache:ClientCache[Integer, Address] = igniteClient.getOrCreateCache(*CACHE\_NAME*)  
  
 cache.query(new SqlFieldsQuery(String.*format*("DROP TABLE IF EXISTS Person2"))  
 .setSchema("PUBLIC")).getAll  
  
 cache.query(new SqlFieldsQuery(String.*format*("CREATE TABLE IF NOT EXISTS Person2 (id LONG,street varchar, zip VARCHAR, PRIMARY KEY (id) ) WITH \"VALUE\_TYPE=%s\"", *classOf*[Address].getName))  
 .setSchema("PUBLIC")).getAll  
  
 cache.query(new SqlFieldsQuery("INSERT INTO Person2(id,street, zip) VALUES(?,?, ?)").setArgs(1L.asInstanceOf[JLong],"Jameco", "04074").setSchema("PUBLIC")).getAll  
 cache.query(new SqlFieldsQuery("INSERT INTO Person2(id,street, zip) VALUES(?,?, ?)").setArgs(2L.asInstanceOf[JLong],"Bremar road", "520003").setSchema("PUBLIC")).getAll  
 cache.query(new SqlFieldsQuery("INSERT INTO Person2(id,street, zip) VALUES(?,?, ?)").setArgs(3L.asInstanceOf[JLong],"orange road", "1234").setSchema("PUBLIC")).getAll  
  
 System.*out*.format(">>> Data Inserted into Cache [%s].\n", *CACHE\_NAME*)  
  
 val data=cache.query(new SqlFieldsQuery("select \* from Person2").setSchema("PUBLIC")).getAll  
  
 *println*(data.toString)  
}

**Step 2 Read example Ignite data: -**

def sparkReadIgniteWithThinClient(implicit spark: SparkSession)={  
  
 val df = spark.read  
 .format("jdbc")  
 .option("url", "jdbc:ignite:thin://35.171.27.68")  
 .option("fetchsize",100)  
 //.option("driver", "org.apache.ignite.IgniteJdbcDriver")  
 .option("dbtable", "Person2").load()  
  
 df.printSchema()  
  
 df.createOrReplaceTempView("test")  
  
 spark.sql("select \* from test where id=1").show(10)  
  
 spark.sql("select 4,'blah',124232").show(10)  
  
}

### Write Spark Data Frame to Ignite: -

def sparkWriteIgniteWithThinClient(implicit spark: SparkSession)={  
  
 import java.sql.DriverManager  
 val connection = DriverManager.*getConnection*("jdbc:ignite:thin://35.171.27.68")  
  
 import java.util.Properties  
 val connectionProperties = new Properties()  
  
 connectionProperties.put("url", "jdbc:ignite:thin://35.171.27.68")  
  
 spark.sql("select 4 as ID,'blah' as STREET,124232 as ZIP").write.mode(SaveMode.*Append*).jdbc("jdbc:ignite:thin://35.171.27.68",  
 "Person2",connectionProperties)  
  
 spark.read  
 .format("jdbc")  
 .option("url", "jdbc:ignite:thin://35.171.27.68")  
 .option("fetchsize",100)  
 .option("dbtable", "Person2").load().show(10,false)  
  
}

### Full Code: -

package com.ignite.examples.spark  
  
import com.ignite.examples.model.Address  
import org.apache.ignite.{Ignite, Ignition}  
import org.apache.ignite.cache.query.SqlFieldsQuery  
import org.apache.ignite.client.{ClientCache, IgniteClient}  
import org.apache.ignite.configuration.{CacheConfiguration, ClientConfiguration}  
import java.lang.{Long => JLong, String => JString}  
import com.ignite.examples.spark.SparkClientConnectionTest.{*CACHE\_NAME*, *CONFIG*}  
import org.apache.ignite.internal.util.IgniteUtils.*resolveIgnitePath*import org.apache.ignite.spark.IgniteDataFrameSettings.{*FORMAT\_IGNITE*, *OPTION\_CONFIG\_FILE*, *OPTION\_CREATE\_TABLE\_PARAMETERS*, *OPTION\_CREATE\_TABLE\_PRIMARY\_KEY\_FIELDS*, *OPTION\_TABLE*}  
import org.apache.log4j.{Level, Logger}  
import org.apache.spark.sql.{SaveMode, SparkSession}  
  
object SparkIgniteCleanCode {  
  
 private val *CACHE\_NAME* = "SPARK2"  
  
 private val *CONFIG* = "/Users/kalit\_000/Downloads/designing-event-driven-applications-apache-kafka-ecosystem/05/demos/kafka-streams-after/ApacheIgnitePoc/src/main/scala/com/ignite/examples/config/example-ignite.xml"  
  
 def setupExampleData = {  
  
 val cfg2 = new ClientConfiguration().setAddresses("35.171.27.68:10800")  
 val igniteClient:IgniteClient = Ignition.*startClient*(cfg2)  
  
 System.*out*.format(">>> Created cache [%s].\n", *CACHE\_NAME*)  
  
 val cache:ClientCache[Integer, Address] = igniteClient.getOrCreateCache(*CACHE\_NAME*)  
  
 cache.query(new SqlFieldsQuery(String.*format*("DROP TABLE IF EXISTS Person2"))  
 .setSchema("PUBLIC")).getAll  
  
 cache.query(new SqlFieldsQuery(String.*format*("CREATE TABLE IF NOT EXISTS Person2 (id LONG,street varchar, zip VARCHAR, PRIMARY KEY (id) ) WITH \"VALUE\_TYPE=%s\"", *classOf*[Address].getName))  
 .setSchema("PUBLIC")).getAll  
  
 cache.query(new SqlFieldsQuery("INSERT INTO Person2(id,street, zip) VALUES(?,?, ?)").setArgs(1L.asInstanceOf[JLong],"Jameco", "04074").setSchema("PUBLIC")).getAll  
 cache.query(new SqlFieldsQuery("INSERT INTO Person2(id,street, zip) VALUES(?,?, ?)").setArgs(2L.asInstanceOf[JLong],"Bremar road", "520003").setSchema("PUBLIC")).getAll  
 cache.query(new SqlFieldsQuery("INSERT INTO Person2(id,street, zip) VALUES(?,?, ?)").setArgs(3L.asInstanceOf[JLong],"orange road", "1234").setSchema("PUBLIC")).getAll  
  
 System.*out*.format(">>> Data Inserted into Cache [%s].\n", *CACHE\_NAME*)  
  
 val data=cache.query(new SqlFieldsQuery("select \* from Person2").setSchema("PUBLIC")).getAll  
  
 *println*(data.toString)  
 }  
  
 def sparkReadIgniteWithThinClient(implicit spark: SparkSession)={  
  
 val df = spark.read  
 .format("jdbc")  
 .option("url", "jdbc:ignite:thin://35.171.27.68")  
 .option("fetchsize",100)  
 //.option("driver", "org.apache.ignite.IgniteJdbcDriver")  
 .option("dbtable", "Person2").load()  
  
 df.printSchema()  
  
 df.createOrReplaceTempView("test")  
  
 spark.sql("select \* from test where id=1").show(10)  
  
 spark.sql("select 4,'blah',124232").show(10)  
  
 }  
  
 def sparkWriteIgniteWithThinClient(implicit spark: SparkSession)={  
  
 import java.sql.DriverManager  
 val connection = DriverManager.*getConnection*("jdbc:ignite:thin://35.171.27.68")  
  
 import java.util.Properties  
 val connectionProperties = new Properties()  
  
 connectionProperties.put("url", "jdbc:ignite:thin://35.171.27.68")  
  
 spark.sql("select 4 as ID,'blah' as STREET,124232 as ZIP").write.mode(SaveMode.*Append*).jdbc("jdbc:ignite:thin://35.171.27.68",  
 "Person2",connectionProperties)  
  
 spark.read  
 .format("jdbc")  
 .option("url", "jdbc:ignite:thin://35.171.27.68")  
 .option("fetchsize",100)  
 .option("dbtable", "Person2").load().show(10,false)  
  
 }  
  
 def writeJSonToIgniteUsingSpark(implicit spark: SparkSession): Unit = {  
  
 val ignite = Ignition.*start*(*CONFIG*)  
  
 //Load content of json file to data frame.  
 val personsDataFrame = spark.read.json(  
 *resolveIgnitePath*("/Users/kalit\_000/Downloads/designing-event-driven-applications-apache-kafka-ecosystem/05/demos/kafka-streams-after/ApacheIgnitePoc/src/main/resources/person.json").getAbsolutePath)  
  
 *println*()  
 *println*("Json file content:")  
 *println*()  
  
 //Printing content of json file to console.  
 personsDataFrame.show()  
  
 *println*()  
 *println*("Writing Data Frame to Ignite:")  
 *println*()  
  
 //Writing content of data frame to Ignite.  
 personsDataFrame.write  
 .format(*FORMAT\_IGNITE*)  
 .mode(org.apache.spark.sql.SaveMode.*Append*)  
 .option(*OPTION\_CONFIG\_FILE*, *CONFIG*)  
 .option(*OPTION\_TABLE*, "json\_person")  
 .option(*OPTION\_CREATE\_TABLE\_PRIMARY\_KEY\_FIELDS*, "id")  
 .option(*OPTION\_CREATE\_TABLE\_PARAMETERS*, "template=replicated")  
 .save()  
  
 *println*("Done!")  
  
 *println*()  
 *println*("Reading data from Ignite table:")  
 *println*()  
  
 val cache = ignite.cache[Any, Any](*CACHE\_NAME*)  
  
 //Reading saved data from Ignite.  
 val data = cache.query(new SqlFieldsQuery("SELECT id, name, department FROM json\_person")).getAll  
  
 *println*(data.toString)  
  
 //data.foreach { row ⇒ println(row.mkString("[", ", ", "]")) }  
 }  
  
 def readIgniteUsingSpark(implicit spark: SparkSession) = {  
 val json\_person = spark.read  
 .format(*FORMAT\_IGNITE*)  
 .option(*OPTION\_CONFIG\_FILE*, *CONFIG*)  
 .option(*OPTION\_TABLE*, "json\_person")  
 .load()  
  
 *println*()  
 *println*("Data frame content:json\_person")  
 *println*()  
  
 //Printing content of data frame to console.  
 json\_person.show()  
 }  
  
 def readIgniteSparkTableUsingThinClient()={  
 val cfg2 = new ClientConfiguration().setAddresses("35.171.27.68:10800")  
 val igniteClient:IgniteClient = Ignition.*startClient*(cfg2)  
  
 System.*out*.format(">>> Created cache [%s].\n", *CACHE\_NAME*)  
 val cache2:ClientCache[Integer, Address] = igniteClient.getOrCreateCache(*CACHE\_NAME*)  
 val data=cache2.query(new SqlFieldsQuery("select \* from json\_person")).getAll  
 *println*(data.toString)  
 }  
  
  
 def readThinClientTableUsingSpark(implicit spark: SparkSession) = {  
 val personDataFrame = spark.read  
 .format(*FORMAT\_IGNITE*)  
 .option(*OPTION\_CONFIG\_FILE*, *CONFIG*)  
 .option(*OPTION\_TABLE*, "person2")  
 .load()  
  
 *println*()  
 *println*("Data frame thin connection content:person")  
 *println*()  
  
 //Printing content of data frame to console.  
 personDataFrame.show()  
 }  
  
  
 def main(args: Array[String]): Unit = {  
  
 *println*()  
 *println*("Step 1 setupExampleData:")  
 *println*()  
  
 *setupExampleData  
  
 println*()  
 *println*("Step 2 createSparkSession:")  
 *println*()  
  
 //Creating spark session.  
 implicit val spark = SparkSession.*builder*()  
 .appName("Spark Ignite data sources example")  
 .master("local")  
 .config("spark.executor.instances", "2")  
 .getOrCreate()  
  
 // Adjust the logger to exclude the logs of no interest.  
 Logger.*getRootLogger*.setLevel(Level.*ERROR*)  
 Logger.*getLogger*("org.apache.ignite").setLevel(Level.*INFO*)  
  
 *println*()  
 *println*("Step 3 ReadIgniteWithThinClient of Step1 Data:")  
 *println*()  
  
 *sparkReadIgniteWithThinClient*(spark)  
  
 *println*()  
 *println*("Step 4 sparkWriteIgniteWithThinClient of Step1 Data:")  
 *println*()  
  
 *sparkWriteIgniteWithThinClient*(spark)  
  
 *println*()  
 *println*("Step 5 writeJSonToIgniteUsingSpark Using Spark:")  
 *println*()  
  
 *writeJSonToIgniteUsingSpark*(spark)  
  
 *println*()  
 *println*("Step 6 readIgniteUsingSpark Using Spark:")  
 *println*()  
  
 *readIgniteUsingSpark*(spark)  
  
 *println*()  
 *println*("Step 7 readIgniteSparkTableUsingThinClient Using Spark:")  
 *println*()  
  
 *readIgniteUsingSpark*(spark)  
  
 *println*()  
 *println*("Step 8 readThinClientTableUsingSpark Using Spark:")  
 *println*()  
  
 *readThinClientTableUsingSpark*(spark)  
  
  
 //spark.stop()  
  
  
 }  
  
}

### Maven Dependency: -

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>IgnitePOc</groupId>  
 <artifactId>ApacheIgnitePoc</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <build>  
 <sourceDirectory>src/main/scala</sourceDirectory>  
 <testSourceDirectory>src/test/scala</testSourceDirectory>  
 <plugins>  
 <plugin>  
 <!-- see http://davidb.github.com/scala-maven-plugin -->  
 <groupId>net.alchim31.maven</groupId>  
 <artifactId>scala-maven-plugin</artifactId>  
 <version>3.1.3</version>  
 <executions>  
 <execution>  
 <goals>  
 <goal>compile</goal>  
 <goal>testCompile</goal>  
 </goals>  
 <configuration>  
 </configuration>  
 </execution>  
 </executions>  
 </plugin>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.13</version>  
 <configuration>  
 <useFile>false</useFile>  
 <disableXmlReport>true</disableXmlReport>  
 <!-- If you have classpath issue like NoDefClassError,... -->  
 <!-- useManifestOnlyJar>false</useManifestOnlyJar -->  
 <includes>  
 <include>\*\*/\*Test.\*</include>  
 <include>\*\*/\*Suite.\*</include>  
 </includes>  
 </configuration>  
 </plugin>  
  
 <!-- "package" command plugin -->  
 <plugin>  
 <artifactId>maven-assembly-plugin</artifactId>  
 <version>2.4.1</version>  
 <configuration>  
 <descriptorRefs>  
 <descriptorRef>jar-with-dependencies</descriptorRef>  
 </descriptorRefs>  
 </configuration>  
 <executions>  
 <execution>  
 <id>make-assembly</id>  
 <phase>package</phase>  
 <goals>  
 <goal>single</goal>  
 </goals>  
 </execution>  
 </executions>  
 </plugin>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.7.0</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
 <repositories>  
 <repository>  
 <id>maven-hadoop</id>  
 <name>Hadoop Releases</name>  
 <url>https://repository.cloudera.com/content/repositories/releases/</url>  
 </repository>  
 <repository>  
 <id>cloudera-repos</id>  
 <name>Cloudera Repos</name>  
 <url>https://repository.cloudera.com/artifactory/cloudera-repos/</url>  
 </repository>  
 </repositories>  
 <dependencies>  
  
 <dependency>  
 <groupId>org.apache.ignite</groupId>  
 <artifactId>ignite-core</artifactId>  
 <version>2.7.5</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.ignite</groupId>  
 <artifactId>ignite-spring</artifactId>  
 <version>2.7.5</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.ignite</groupId>  
 <artifactId>ignite-indexing</artifactId>  
 <version>2.7.5</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.apache.ignite</groupId>  
 <artifactId>ignite-spark</artifactId>  
 <version>2.7.5</version>  
 </dependency>  
 <dependency>  
 <groupId>com.amazonaws</groupId>  
 <artifactId>aws-lambda-java-core</artifactId>  
 <version>1.1.0</version>  
 </dependency>  
 <dependency>  
 <groupId>com.amazonaws</groupId>  
 <artifactId>dynamodb-streams-kinesis-adapter</artifactId>  
 <version>LATEST</version>  
 </dependency>  
 <dependency>  
 <groupId>com.amazonaws</groupId>  
 <artifactId>amazon-kinesis-client</artifactId>  
 <version>LATEST</version>  
 </dependency>  
 <dependency>  
 <groupId>javax.cache</groupId>  
 <artifactId>cache-api</artifactId>  
 <version>1.0.0</version>  
 </dependency>  
  
 </dependencies>  
</project>

### GitHub Link: -

<https://github.com/kali786516/ApacheIgnitePoc/blob/master/src/main/scala/com/ignite/examples/spark/SparkIgniteCleanCode.scala>

## Apache Ignite Resources and Training and Blogs: -

<https://app.pluralsight.com/course-player?course=apache-ignite-getting-started&author=edward-curren&name=e96dc6ff-e481-4755-b408-011e9706a816&clip=0&mode=live>

<https://medium.com/@sdarlington/loading-data-into-apache-ignite-c0cb7c065a7>

## Apache Ignite Real World Scenario: -

This blog needs some code changes to make it work the above concepts in this document should make it work.

<https://aws.amazon.com/blogs/big-data/real-time-in-memory-oltp-and-analytics-with-apache-ignite-on-aws/>