Securing the Spark Fire Hose

Jack Mannino  Abdullah Munawar
Who Are We?

- Abdullah Munawar
  - Managing Consultant @ nVisium
  - Lifelong Redskins fan

- Jack Mannino
  - CEO @ nVisium
  - Scala fan, looks down on people who don’t Scala :-)  
  - New York Mets fan, numb to disappointment

- Tony Romo
  - Not affiliated with nVisium (thankfully)
  - Injured
  - Made of glass
TL;DR

- Open source general purpose cluster computing framework
  - Spark Core, Spark SQL, GraphX, and MLLib
- Commercial support and a large ecosystem
- Better suited for stream processing and iterative machine learning algorithms than Hadoop/MapReduce
- Why do we care? **We develop software with Spark.**
Why Do People Use Spark?

- https://www.youtube.com/watch?v=zKbds9ZPjLE
A Real-World Spark Architecture

What’s important

- Assuming the data is trusted and valid, authN/authZ, data at rest, in transit, injection and attacks against interpreters, untrusted deserialization
Spark Cluster Mode

http://spark.apache.org/docs/latest/cluster-overview.html
## Spark Exposed Ports and Services

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Default Port</th>
<th>Purpose</th>
<th>Configuration Setting</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser</td>
<td>Application</td>
<td>4040</td>
<td>Web UI</td>
<td>spark.ui.port</td>
<td>Jetty-based</td>
</tr>
<tr>
<td>Browser</td>
<td>History Server</td>
<td>18080</td>
<td>Web UI</td>
<td>spark.history.ui.port</td>
<td>Jetty-based</td>
</tr>
<tr>
<td>Executor / Standalone Master</td>
<td>Driver</td>
<td>(random)</td>
<td>Connect to application / Notify executor state changes</td>
<td>spark.driver.port</td>
<td>Akka-based. Set to &quot;0&quot; to choose a port randomly.</td>
</tr>
<tr>
<td>Driver</td>
<td>Executor</td>
<td>(random)</td>
<td>Schedule tasks</td>
<td>spark.executor.port</td>
<td>Akka-based. Set to &quot;0&quot; to choose a port randomly.</td>
</tr>
<tr>
<td>Executor</td>
<td>Driver</td>
<td>(random)</td>
<td>File server for files and jars</td>
<td>spark.fileserver.port</td>
<td>Jetty-based</td>
</tr>
<tr>
<td>Executor</td>
<td>Driver</td>
<td>(random)</td>
<td>HTTP Broadcast</td>
<td>spark.broadcast.port</td>
<td>Jetty-based. Not used by TorrentBroadcast, which sends data through the block manager instead.</td>
</tr>
<tr>
<td>Executor</td>
<td>Driver</td>
<td>(random)</td>
<td>Class file server</td>
<td>spark.replClassServer.port</td>
<td>Jetty-based. Only used in Spark shells.</td>
</tr>
<tr>
<td>Executor / Driver</td>
<td>Executor / Driver</td>
<td>(random)</td>
<td>Block Manager port</td>
<td>spark.blockManager.port</td>
<td>Raw socket via ServerSocketChannel</td>
</tr>
</tbody>
</table>
object KafkaWordCount {
    def main(args: Array[String]) {
        if (args.length < 4) {
            System.err.println("Usage: KafkaWordCount <zkQuorum> <group> <topics> <numThreads>"")
            System.exit(1)
        }

        StreamingExamples.setStreamingLogLevels()

        val Array(zkQuorum, group, topics, numThreads) = args
        val sparkConf = new SparkConf().setAppName("KafkaWordCount")
        val ssc = new StreamingContext(sparkConf, Seconds(2))
        ssc.checkpoint("checkpoint")

        val topicMap = topics.split(",").map((_, numThreads.toInt)).toMap
        val lines = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap).map(_._2)
        val words = lines.flatMap(_.split(" "))
        val wordCounts = words.map(x => (x, 1L))
        .reduceByKeyAndWindow(_ + _, _, Minutes(10), Seconds(2), 2)
        wordCounts.print()

        ssc.start()
        ssc.awaitTermination()
    }
}
SparkContext

- The initial entry point to an application
- `StreamingContext`, for streaming apps
- By default, one SparkContext per JVM
  - Or, `spark.driver.allowMultipleContexts`...but buggy
- Configuration and bootstrapping the environment
  - Security settings, performance etc.
  - `addFile` and `addJar` to upload dependencies
- Each instance starts a Web UI listener on port 4040
  - Securitah by default? More on that later, but tl;dr - none
RDDs and DAGs

Details for Job 5

Status: SUCCEEDED
Completed Stages: 1

- Event Timeline
- DAG Visualization

Stage 5

makeRDD

pipe
Physical Plan

ComplexJob
including map(), partitionBy(), union(), and join()
Disk and Memory Storage

- Spark uses a combination of in-memory and disk persistence
  - BlockStore
  - MemoryStore
  - DiskStore

- Storage levels determine how your data is persisted
  - NONE
  - DISK_ONLY
  - MEMORY_ONLY
  - MEMORY_AND_DISK
  - *_SER
  - OFF_HEAP

```java
/** *
 * Expose some commonly useful storage level constants.
 */
public class StorageLevels {
    public static final StorageLevel NONE = create(false, false, false, false, 1);
    public static final StorageLevel DISK_ONLY = create(true, false, false, false, 1);
    public static final StorageLevel DISK_ONLY_2 = create(true, false, false, false, 2);
    public static final StorageLevel MEMORY_ONLY = create(false, true, false, false, 1);
    public static final StorageLevel MEMORY_ONLY_2 = create(false, true, false, true, 2);
    public static final StorageLevel MEMORY_AND_DISK = create(true, true, false, false, 1);
    public static final StorageLevel MEMORY_AND_DISK_2 = create(true, true, false, true, 2);
    public static final StorageLevel MEMORY_AND_DISK_SER = create(true, true, false, false, 1);
    public static final StorageLevel MEMORY_AND_DISK_SER_2 = create(true, true, false, true, 2);
    public static final StorageLevel OFF_HEAP = create(true, true, true, false, 1);
}```
Unencrypted Disk Persistence

- Unless you’re using HDFS, there aren’t many options for native disk or in-memory encryption
- Unless explicitly called, it occurs under the following conditions:
  - RDD checkpointing triggered by Spark or the application’s code
    - `RDD.checkpoint()`
  - Data spill during shuffling
  - Lack of available RAM for caching
- Approaches
  - Understand your data classification levels and impact of combining data
  - Build security into the serializers themselves (fast encryption for I/O)
IO Encryption—Sorry, YARN Only

- Encryption capabilities are thanks to YARN, not Spark
- If you’re using YARN, you get some IO encryption

```java
if (master == "yarn" && deployMode == "client") System.setProperty("SPARK_YARN_MODE", "true")
if (_conf.get(IO_ENCRYPTION_ENABLED) && !SparkHadoopUtil.get.isYarnMode()) {
    throw new SparkException("IO encryption is only supported in [YARN mode](https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-client/yarn-client.html), please disable it " +
        "by setting ${IO_ENCRYPTION_ENABLED.key} to false")
}
```
Spark SQL
Spark SQL

- Spark SQL allows you to query structured data from many sources
  - JDBC (MySQL, PostgreSQL)
  - Hive
  - JSON
  - Parquet
  - Optimized Row Columnar (ORC)
  - Cassandra
  - HDFS
  - S3

- Catalyst query compiler introduced in Spark 2.0
  - ANTLR Grammar for SQL, including Spark-specific items
  - Abstract Syntax Tree builder converts ANTLR ParseTrees into expressions

- Lexical analysis for syntax verification and query construction
“If it’s simple, quoting sounds good. This isn’t really a SQL injection problem since it would be up to callers to sanitize inputs from an external source, and Spark is not something you would expose directly to external calls or input. Still this may avoid corner case problems like table names with a special character.”
Spark SQL + Catalyst
primaryExpression : name=(CURRENT_DATE | CURRENT_TIMESTAMP) | CASE value=expression whenClause+ (ELSE elseExpression=expression)? END | CASE whenClause+ (ELSE elseExpression=expression)? END | CAST '(' expression AS dataType ')' | constant | ASTERISK | qualifiedName '.' ASTERISK | '(' expression (',' expression)+ ')' | '(' query ')' | qualifiedName '(' (setQuantifier? expression (',' expression)*)? ')' (OVER windowSpec)? | value=primaryExpression '[' index=valueExpression ']' | identifier | base=primaryExpression '.' fieldName=identifier | '(' expression ')' ;
Query Plan + Catalyst

• Catalyst
  ○ “Finally, rule conditions and their bodies can contain arbitrary Scala code. This gives Catalyst more power than domain specific languages for optimizers, while keeping it concise for simple rules.”

• Scala language tricks used by Catalyst
  ○ Quasiquotes
  ○ Reflection

```scala
def compile(node: Node): AST = node match {
  case Literal(value) => q"$value"
  case Attribute(name) => q"row.get($name)"
  case Add(left, right) => q"${compile(left)} + ${compile(right)}"
}```
Spark SQL + DataSet + DataFrames

- Ability to query structured data inside a spark program using traditional SQL
- Ability to use DataSet API and leverage Java and Scala
- As of Spark 2.0 DataFrames are merely an alias to DataSet

```python
# spark is an existing SparkSession
df = spark.read.json("examples/src/main/resources/people.json")
# Displays the content of the DataFrame to stdout
df.show()
```

Spark SQL vs. DataSet vs. DataFrames

Spark SQL

- Native, plain old
  - sql(sqlText: String): DataFrame
    - scala> sql("SHOW TABLES")
      res0: org.apache.spark.sql.DataFrame = [tableName: string, isTemporary: boolean]

- DataSet
  - val dataset = (0 to 4).toDS
  - // Variant 3: filter operator accepts a SQL query
    dataset.filter("value % 2 = 0").count
  - Enforces type safety at runtime
Spark SQL

- Classic anti-pattern #1 - string concatenation
  - `sql(sqlText: String): DataFrame`

  ```scala
sql("SELECT person FROM table where name = " + name )
  ```

- How do you fix it?
  - Bind your variables?
  - Escape it?
  - Filter it?
Spark SQL—Who’s Down With JDBCRDD?

```scala
/**
 * Converts value to SQL expression.
 */
private def compileValue(value: Any): Any = value match {
  case stringValue: String => s"${escapeSql(stringValue)}"
  case timestampValue: Timestamp => """ + timestampValue + ""
  case dateValue: Date => """ + dateValue + ""
  case arrayValue: Array[Any] => arrayValue.map(compileValue).mkString("", "")
  case _ => value
}

private def escapeSql(value: String): String =
  if (value == null) null else StringTools.replace(value, "\\", "\\\\")
```
private val REF_RE = "\\$\\{(?:(\\w+?):)?(\\S+)}\"".r

private def substitute(input: String, usedRefs: Set[String]): String = {
  if (input != null) {
    ConfigReader.REF_RE.replaceAll(input, { m =>
      val prefix = m.group(1)
      val name = m.group(2)
      val ref = if (prefix == null) name else s"$prefix:$name"
      require(!usedRefs.contains(ref), s"Circular reference in $input: $ref"

      val replacement = bindings.get(prefix)
        .flatMap(_ get(name))
        .map { v => substitute(v, usedRefs + ref) }
        .getOrElse(m.matched)
          Regex.quoteReplacement(replacement)
    )
  } else {
    input
  }
}
Spark SQL

- This is why we can’t have nice things
- Detecting Injection in Spark SQL dynamically may be harder
  - Bang, bang, bang on the website doesn’t work
  - You’re still thinking of the way the internet used to work
- Many stages along with scheduled jobs, Lambda etc. means timing attacks don’t make as much sense and may be less useful
  - This might not be a synchronous thing
  - This might not even be a thing that gets kicked back to the client
    - Logging
    - Predictive analytics: we noticed you like socks, do you need shoes too?
  - Might see results in failing services, jobs taking much longer to complete
User Defined Functions

- “UDF functions are a blackbox for Spark and so it does not even try to optimize them.”
- These functions defy the rules; treat them as unsafe

```scala
val dataset = Seq((0, "hey"), (1, "lascon")).toDF("id", "text")

// Define a regular Scala function
val doQuestionableStuff: String => String = _.toUpperCase

// Define a UDF that wraps the upper Scala function defined above
// You could also define the function in place, i.e. inside udf
// but separating Scala functions from Spark SQL's UDFs allows for easier testing
import org.apache.spark.sql.functions.udf
val questionableUDF = udf(doQuestionableStuff)

// Apply the UDF to change the source dataset
scala> dataset.withColumn("questionable", questionableUDF('text)).show
```
Serialization and the Questionable Chin
Spark <3s Serialization

- Serialization and deserialization happens everywhere
- Speed is paramount
- Lots of inference to deduce class signatures on-the-fly
Unsafe Deserialization

- Kryo is the default serializer
  - Java serialization is also used, too
- By default, Kryo disables type registration
  - Remember, speed is paramount right?
  - `private val registrationRequired = conf.getBoolean("spark.kryo.registrationRequired", false)`
  - `kryo.setRegistrationRequired(registrationRequired)`
- `spark.kryo.registrationRequired`
- `spark.kyro.classesToRegister`
- Open issue: https://github.com/EsotericSoftware/kryo/issues/398
Unsafe Deserialization

- What happens when you lie to Kryo (or other serializers)?
- Invoke anything loaded on the classpath
  - Call zero-argument constructors
  - Call finalize functions
- Example of a few tricks against Kryo
  - https://www.contrastsecurity.com/security-influencers/serialization-must-die-act-1-kryo
- Spark consumes serialized data within developer APIs and internal APIs
Crashing the JVM

```java
private void testJnaFree() throws Exception {
    /*
     * Create a 1-byte buffer from malloc().
     */
    Memory mem = new Memory(1);
    mem.setByte(0, (byte) 0x01);
    assertEquals(0x01, mem.getByte(0));

    /*
     * Confirm that the protections around this API are in place.
     */
    try {
        mem.getByte(5);
        fail("Shouldn't be able to access value outside of 1-byte buffer");
    } catch (IndexOutOfBoundsException e) {
    }

    /*
     * Set the hidden member of this field that points to the memory address
     * where the buffer is supposed to be allocated to some random address.
     */
    FieldHelper memHelper = new FieldHelper(mem);
    memHelper.setValue("peer", RandomUtils.nextLong());

    /*
     * Serialize the now-malicious value to be read in by the target app.
     */
    kryo.writeObject(out, mem);

    /*
     * Deserialize the malicious value.
     */
    in = new Input(out.toBytes());
    Memory rebuiltMem = kryo.readObject(in, Memory.class);

    /*
     * This call will crash the JVM entirely, unless your random() call
     * accidentally landed on a previously allocated buffer. Not likely.
     */
    callFinalize(rebuiltMem);
}
```

Pipe and the Nasty Piranhas
Pipe and the Nasty Piranhas

- Uses `ProcessBuilder` to invoke system processes
- Single shell command at a time
- For added flexibility, developers invoke bash directly
  - `dataRDD.pipe("bash -c \'rm " + input + \"\'")`
- Invoke a file uploaded via `addFile` through SparkContext
- Sink becomes the code you invoke through Spark
  - Executable files
    - Shell scripts
    - R files
    - Python scripts
Pipe and the Nasty Piranhas

- Example of pipe to execute a single command

```scala
scala> val data = List("hi","there","lascon","what's","up")
data: List[String] = List(hi, there, lascon, what's, up)

scala> val dataRDD = sc.makeRDD(data)

scala> val pipeRDD = dataRDD.pipe("bash -c 'uname'")

scala> pipeRDD.collect
Pipe and the Nasty Piranhas

- Avoid `eval()` calls in R, Python, Shell, etc.
  - `eval(badStuff)`
- Avoid directly invoking `bash -c`
- Handle the data safely for the respective interpreter/language
Transport Layer Security

Communications from
- Spark Streaming
- Spark SQL
- Spark Master/Worker communications
- Outbound from Spark

From SecurityManager
// the default SSL configuration - it will be used by all communication layers unless overwritten
private val defaultSSLOptions = SSLOptions.parse(sparkConf, "spark.ssl", defaults = None)

SASL for Block Transfer Service and RPC Endpoints

SASL encryption is currently supported for the block transfer service when authentication (spark.authenticate) is enabled. To enable SASL encryption for an application, set spark.authenticate.enableSaslEncryption to true in the application’s configuration.
Authentication

- Authentication to the following components:
  - REST API
  - Web UI
  - Between nodes

- Authentication
  - YARN
  - SASL
  - HBase + Kerberos
  - Thrift + Hive + Kerberos
  - AWS = IAM makes your life infinitely better!

- Proxy user to Hadoop UGI

- Streaming Interfaces, data sources (ie- Kafka, Cassandra)
Web UI Authentication

- Recall, by default Spark launches a web application when you start a SparkContext.
Web UI Authentication

- By default, no authentication required
- Authentication is implemented with a Servlet Filter
- You must set `spark.ui.filters` to `<your filter path>`
- Custom code to authenticate a user and pass their identity to Jetty (Spark’s Servlet container)
@Override
public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain)
    throws IOException, ServletException {
    HttpServletRequest request = (HttpServletRequest) servletRequest;
    HttpServletResponse response = (HttpServletResponse) servletResponse;

    String authHeader = request.getHeader("Authorization");
    if (authHeader != null) {
        StringTokenizer st = new StringTokenizer(authHeader);
        if (st.hasMoreTokens()) {
            String basic = st.nextToken();

            if (basic.equalsIgnoreCase("Basic")) {
                try {
                    String credentials = new String(Base64.decodeBase64(st.nextToken()), "UTF-8");
                    LOG.debug("Credentials: " + credentials);
                    int p = credentials.indexOf(":");
                    if (p != -1) {
                        String _username = credentials.substring(0, p).trim();
                        String _password = credentials.substring(p + 1).trim();
                    }
                } catch (UnsupportedEncodingException e) {
                    throw new Error("Couldn't retrieve authentication", e);
                }
            } else {
                unauthorized(response);
            }
        } else {
            unauthorized(response, "Bad credentials");
        }
    }
    filterChain.doFilter(servletRequest, servletResponse);
}
Authorization

- Authorization
  - ACLs in Spark UI
  - API access to submit/monitor/modify
- Apache Ranger
- Read/write to RDDs
  - RDDs are immutable and read-only
  - Per-RDD AuthZ strategies
    - Also stage-wise dependencies and patterns to consider
    - Where did the data come from? Who created it? Do you trust it?
- YARN ACLs
  - [https://github.com/apache/spark/commit/a45647746d1efb90cb8bc142c2ef110a0db9bc9f](https://github.com/apache/spark/commit/a45647746d1efb90cb8bc142c2ef110a0db9bc9f)
Streaming Authorization

- Maps Spark partitions to Kafka partitions
- Kafka Partitions -> RDDs don’t always map
  - Relationships and offsets change after shuffle, repartition, etc.
- Considerations
  - Who can publish to the topics you subscribe to?
  - Who can publish to the partitions mapped to Spark?
  - Who can receive the things you publish?
Summary

- Think about the ecosystem and broader architecture
- Think about the data and how badly you want to protect it
- Ensure you understand the impact of default security controls and fix the problematic ones where applicable
- Always think about what’s feeding and talking to Spark
Contact Us

Jack Mannino  
@jack_mannino  
jack@nvisium.com

Abdullah Munawar  
@amanofwar  
abdullah@nvisium.com