Certainly! Integrating a system with the Big Data ecosystem often involves compatibility with popular frameworks like Hadoop and Spark. Here's a brief explanation along with code examples for Hadoop integration and running Spark on YARN.

Hadoop Integration:

Hadoop is a distributed storage and processing framework. If you want your system to interact with Hadoop, you may need to consider Hadoop Distributed File System (HDFS) for storage and MapReduce for processing.

Example Code (using Hadoop Java API):

Assuming you want to read a file from HDFS:

```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class HadoopIntegrationExample {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        conf.set("fs.defaultFS", "hdfs://your-hadoop-cluster");  // Set Hadoop cluster address

        FileSystem fs = FileSystem.get(conf);
        Path filePath = new Path("/path/to/your/file.txt");  // Specify the path to your file in HDFS

        BufferedReader br = new BufferedReader(new InputStreamReader(fs.open(filePath)));
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
        br.close();
    }
}
```

Spark on YARN:

Apache Spark is a fast and general-purpose cluster computing system. YARN (Yet Another Resource Negotiator) is a cluster management technology in Hadoop, and Spark can be configured to run on YARN, leveraging its resource management capabilities.

Example Code (submitting a Spark job to YARN using Spark submit):

Assuming you have a Spark job written in Scala:

```scala
// Your Spark Job
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext

object SparkOnYARNExample {
    def main(args: Array[String]): Unit = {
        val conf = new SparkConf()
          .setAppName("YourSparkJob")
          .setMaster("yarn")  // Set the master to YARN

        val sc = new SparkContext(conf)

        // Your Spark job logic here
        val data = sc.textFile("hdfs://your-hadoop-cluster/path/to/your/input")
        val result = data.flatMap(_.split(" ")).map((_, 1)).reduceByKey(_ + _)

        result.saveAsTextFile("hdfs://your-hadoop-cluster/path/to/your/output")

        sc.stop()
    }
}
```

To submit the job:

```bash
$ spark-submit --class SparkOnYARNExample --master yarn --deploy-mode cluster your-spark-job.jar
```

Ensure that the Hadoop and Spark configurations are properly set in your Spark job. This code assumes you have a Spark job written in Scala and a JAR file containing the compiled code. Adjust the paths and configurations according to your environment.

Certainly! Continuing from the previous explanation, let's delve a bit deeper into Spark on YARN and provide additional examples.

Spark on YARN (continued):

Submitting a Spark job to YARN using Spark submit:

Assuming you have a Spark job written in Python:

```python
# Your Spark Job (example in Python)
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("YourSparkJob") \
    .master("yarn")  # Set the master to YARN \
    .getOrCreate()

# Your Spark job logic here
data = spark.read.text("hdfs://your-hadoop-cluster/path/to/your/input")
word_count = data.selectExpr("explode(split(value, ' ')) as word").groupBy("word").count()

# Save the result back to HDFS
word_count.write.mode("overwrite").csv("hdfs://your-hadoop-cluster/path/to/your/output")

spark.stop()
```

To submit the job:

```bash
$ spark-submit --master yarn --deploy-mode client your-spark-job.py
```

This example assumes you have a Spark job written in Python using the PySpark API.

Spark on YARN with Resource Configuration:

In a production environment, you may need to configure resource parameters for Spark on YARN. Here's an example:

```bash
$ spark-submit \
  --class SparkOnYARNExample \
  --master yarn \
  --deploy-mode cluster \
  --num-executors 5 \
  --executor-memory 2g \
```

```
  --executor-cores 2 \
  --driver-memory 1g \
  your-spark-job.jar
```

Adjust the values (`num-executors`, `executor-memory`, etc.) based on your cluster configuration and workload requirements.

Spark Streaming on YARN:

If you're dealing with real-time data, you might use Spark Streaming. Here's an example:

```scala
// Spark Streaming on YARN (Scala)
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._

val ssc = new StreamingContext(sparkConf, Seconds(1))

// Your Spark Streaming job logic here
val lines = ssc.socketTextStream("localhost", 9999)
val words = lines.flatMap(_.split(" "))
val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)

wordCounts.print()

ssc.start()
ssc.awaitTermination()
```

Submit the Spark Streaming job to YARN using a similar `spark-submit` command as mentioned earlier.

Remember to adapt these examples to your specific use case, ensuring that Hadoop and Spark configurations align with your cluster setup.