# Building An Application Using MapReduce and Workflows

## **Exercise Objectives**

In this exercise, you will:

- Add a MapReduce job and Workflow to the example project
- Run the MapReduce job
- Run the Workflow
- View operations and results in Reactor Dashboard

## **Exercise Steps**

- Add imports
- Modify pom.xml
- Modify ApplicationSpecification
- Modify the Procedure
- Define the MapReduce job
- Define beforeSubmit
- Define Mapper
- Define Reducer
- Define the Workflow
- Build, deploy, run and test

#### Add Imports

Add these imports:

```
import com.continuuity.api.data.dataset.table.Put;
import com.continuuity.api.mapreduce.AbstractMapReduce;
import com.continuuity.api.mapreduce.MapReduceContext;
import com.continuuity.api.mapreduce.MapReduceSpecification;
import com.continuuity.api.schedule.Schedule;
import com.continuuity.api.workflow.Workflow;
import com.continuuity.api.workflow.WorkflowSpecification;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.MD5Hash;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
```

### Modify pom.xml

```
<dependency>
  <groupId>org.apache.hadoop</groupId>
 <artifactId>hadoop-common</artifactId>
 <version>${hadoop.version}</version>
 <scope>provided</scope>
  <exclusions>
    <exclusion>
      <groupId>io.netty</groupId>
      <artifactId>netty</artifactId>
    </exclusion>
    <exclusion>
      <groupId>com.sun.jersey</groupId>
      <artifactId>jersey-server</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.jboss.netty</groupId>
      <artifactId>netty</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

## Modify ApplicationSpecification

Replace .noMapReduce() with:

```
.withMapReduce()
   .add(new SentimentAnalysisMapReduce())
```

Replace .noWorkflow() with:

```
.withWorkflows()
  .add(new SentimentAnalysisWorkflow())
```

## Modify SentimentAnalysisProcedure

Add a new handler:

```
@Handle("reductions")
  public void sentimentReductions(ProcedureRequest request, ProcedureResponder response)
  throws Exception {
    String sentiment = request.getArgument("sentiment");
    if (sentiment == null) {
        response.error(ProcedureResponse.Code.CLIENT_ERROR, "No sentiment sent.");
        return;
    }
    byte[] count = sentiments.get(Bytes.toBytes(sentiment), Bytes.toBytes(sentiment));
    Map<String, Long> resp = Maps.newHashMap();
    if (count == null) {
        resp.put(sentiment, OL);
    } else {
        resp.put(sentiment, Bytes.toLong(count));
    }
    response.sendJson(ProcedureResponse.Code.SUCCESS, resp);
}
```

## Add MapReduce Job

```
public static class SentimentAnalysisMapReduce extends AbstractMapReduce {
  // Annotation indicates the DataSets used in this MapReduce
 @UseDataSet("text-sentiments")
 private SimpleTimeseriesTable textSentiments;
 @UseDataSet("sentiments")
 private Table sentiments;
 private static String sentiment_arg;
 @Override
 public MapReduceSpecification configure() {
   return MapReduceSpecification.Builder.with()
    .setName("SentimentCountMapReduce")
    .setDescription("Sentiment count MapReduce job")
    .useInputDataSet("text-sentiments") // Specify the DataSet for Mapper to read.
    .useOutputDataSet("sentiments") // Specify the DataSet for Reducer to write.
    .setMapperMemoryMB(512)
    .setReducerMemoryMB(1024)
    .build();
```

#### Define the beforeSubmit

```
@Override
public void beforeSubmit(MapReduceContext context) throws Exception {
  Job job = context.getHadoopJob();
  sentiment_arg = context.getRuntimeArguments().get("sentiment");
 if (sentiment_arg == null) {
    sentiment_arg = "positive";
 LOG.info("Start of MapReduce job for sentiment \"" + sentiment_arg + "\"");
  long endTime = System.currentTimeMillis();
  long startTime = endTime - TimeUnit.MILLISECONDS.convert(1, TimeUnit.DAYS);
  context.setInput(textSentiments, textSentiments.getInput(1, Bytes.toBytes(sentiment_arg),
                                                           startTime,
                                                           endTime));
  job.setMapperClass(SentimentMapper.class);
  job.setMapOutputKeyClass(Text.class); // Sets output key of the Reducer class
  job.setMapOutputValueClass(IntWritable.class); // Sets output value of the Reducer class
  job.setReducerClass(SentimentReducer.class);
```

#### Define Mapper

```
* A Mapper that reads the sentiments from the text-sentiments
* DataSet and creates key value pairs, where the key is the
* sentiment and value is the occurrence of a sentence. The Mapper
* receives a key value pair (<byte[], TimeseriesTable.Entry>)
* from the input DataSet and outputs data in another key value
* pair (<Text, IntWritable>) to the Reducer.
public static class SentimentMapper extends Mapper<byte[], TimeseriesTable.Entry,</pre>
                                                   Text, IntWritable> {
  // The output value
 private static final IntWritable ONE = new IntWritable(1);
 public void map(byte[] key, TimeseriesTable.Entry entry, Context context)
 throws IOException, InterruptedException {
  // Send the key value pair to Reducer.
   String sentiment = Bytes.toString(key);
   context.write(new Text(sentiment), ONE);
}
```

#### Define Reducer

## Define onFinish

## Define SentimentAnalysisWorkflow

## Build, Deploy and Test

- Build using mvn clean package
- Deploy the jar to Reactor after stopping any existing Flows and Procedures
- Run the MapReduce job, using the Dashboard to set the sentiment it reduces as a Runtime Argument
- Check the results using the Procedure's *reductions* handler, passing in the same sentiment you are checking as a parameter
- Run the Workflow manually

## **Exercise Summary**

You should now be able to:

- Add MapReduce jobs and Workflows to a project
- Run MapReduce jobs
- Run Workflows
- View operations and results in the Reactor Dashboard

# **Exercise Completed**

Chapter Index