Map Reduce in Action

Maven

- We would be using maven from now onwards for all our Java artifact generation.
- Maven is a build automation tool for Java projects. Maven addresses two aspects of building software: First, it describes how software is built, and second, it describes its dependencies.
- Have your eclipse updated to support Maven-
  - Open Eclipse
  - Go to Help -> Eclipse Marketplace
  - Search for "Maven Integration for Eclipse", choose the one which fits your eclipse version
  - Click "Install" button at "Maven Integration for Eclipse" section
  - Follow the instruction step by step

Demo – Code and build the temperature example

- After Maven is installed, restart eclipse
- Choose File – new Project
- Choose Maven Project
- Select all default options, in the New Maven Project screen provide a Group ID, Artifact ID and if required change the Version.
  - GroupID -> your package name, a unique namespace for your project
  - ArtifcatID -> your project name

Demo – Code and build the temperature example

- Modify pom.xml and include the below
  - Dependency for Hadoop 2.7.2 API's

```
<dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-client</artifactId>
      <version>2.7.2</version>
</dependency>
```

How to find the above XML, open http://mvnrepository.com/, search for Hadoop Client, choose the appropriate version and you would find the above XML.

Demo – Code and build the temperature example

- Modify pom.xml and include the build plugin

```
 <build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.1</version>
            <configuration>
                <source>1.7</source>
                <target>1.7</target>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Demo – Code and build the temperature example

- Add TMapper class, here is the "uglly" code for Tmapper, this needs to be refined more, for now it should suffice for our example.

```
public class TMapper extends Mapper<Text, Text, Text, IntWritable> {

    public void map(LongWritable key, Text value, Context context) throws
    IOException,

    InterruptedException {

    String[] data = value.toString().split(",");

    String state = data[1];

    IntWritable temp = new IntWritable(Integer.parseInt(data[2]));

    context.write(new Text(state), temp);

    }

}
```

Demo – Code and build the temperature example

- Add TReducer

```
public class TReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(final Text key, final Iterable<IntWritable> values,
        final Context context) throws IOException, InterruptedException {
        int sumOfTemperatures = 0;
        int numValues = 0;

        for (IntWritable temperature : values) {
        sumOfTemperatures += temperature.get();
        numValues++;
        }
        int average = sumOfTemperatures / numValues;
        context.write(key, new IntWritable(average));
    }
}
```

Demo – Code and build the temperature example

- Add TDriver (this is the main class, also called as Job class/driver class), you would be running this on the cluster.

```
public class TDriver{
   public static void main(String args[]) throws Exception{
    Configuration conf = new Configuration();
    Job job = new Job(conf, "AvgTemperatureJob");
    String outputDir = “/OUTPUT_" + System.currentTimeMillis();
    String inputFile = "/test/temp.txt";

    FileInputFormat.addInputPath(job, new Path(inputFile));
    job.setInputFormatClass(TextInputFormat.class);


    FileOutputFormat.setOutputPath(job, new Path(outputDir));
    job.setOutputFormatClass(TextOutputFormat.class);

    job.setJarByClass(TDriver.class);
    job.setMapperClass(TMapper.class);
    job.setReducerClass(TReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);


    job.waitForCompletion(true);
    System.out.println("Job Completed, results are stored in the directory - " + outputDir);

  }
}
```

Compilation and other stuff

- Eclipse would automatically compile your code once you save the code
- We need a jar file to run the job on cluster, inorder to generate a jar, we could use maven, choose the project in "Package explorer" of eclipse, and right-click, choose runas-Maven Install.
- Maven would generate the jar in Target folder.
- Copy this jar to your cluster (to edge node), in our case copy this jar to hdtester user dir
- Create a temp.txt file with temperature data (sample data is shared in last class)
- Upload this file to /test folder of hdfs (i.e. start hdfs, yarn and history server before you begin all the operations)
- Run the job using your main class –
  - yarn job <YOURJARFILE> <space> <YOURPACKAGENAME>.TDriver
- Note the output folder name and examine the content using hdfs dfs -cat

So what’s happening here?

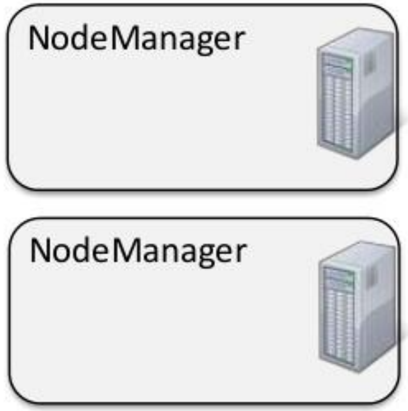
- Data is stored in Hadoop as Blocks on HDFS
- Processing of Data is done using Map Reduce
- Old architecture of Hadoop used JobTracker and TaskTrackers
- New Architecture of Hadoop uses ResourceManager, ApplicationMaster and NodeManagers to run your job
- New Architecture is called YARN (Yet Another Resource Negotiator)
- MapReduce when run on Yarn are called MR2, just to differentiate them from older architectures
- There is no functional difference between MR1 and MR2, its about scalability, scheduling and performance that you prefer MR2. More about the differences between both in later sessions

Yarn

- **Resource Manager (RM)**
  - –Runs on master node
  - –Global resource scheduler
  - –Arbitrates system resources between competing applications

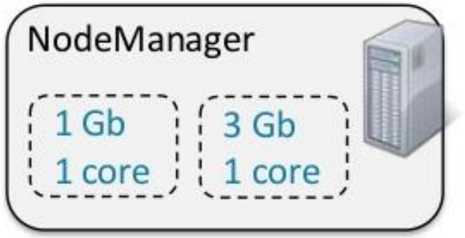
Resource Manager

- **Node Manager (NM)**
  - –Runs on slave nodes
  - –Communicates with RM

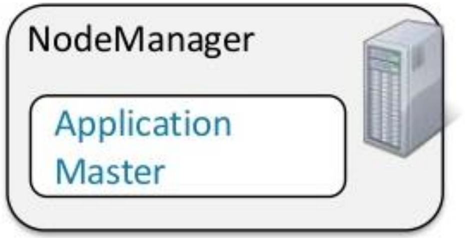


Yarn Application Components

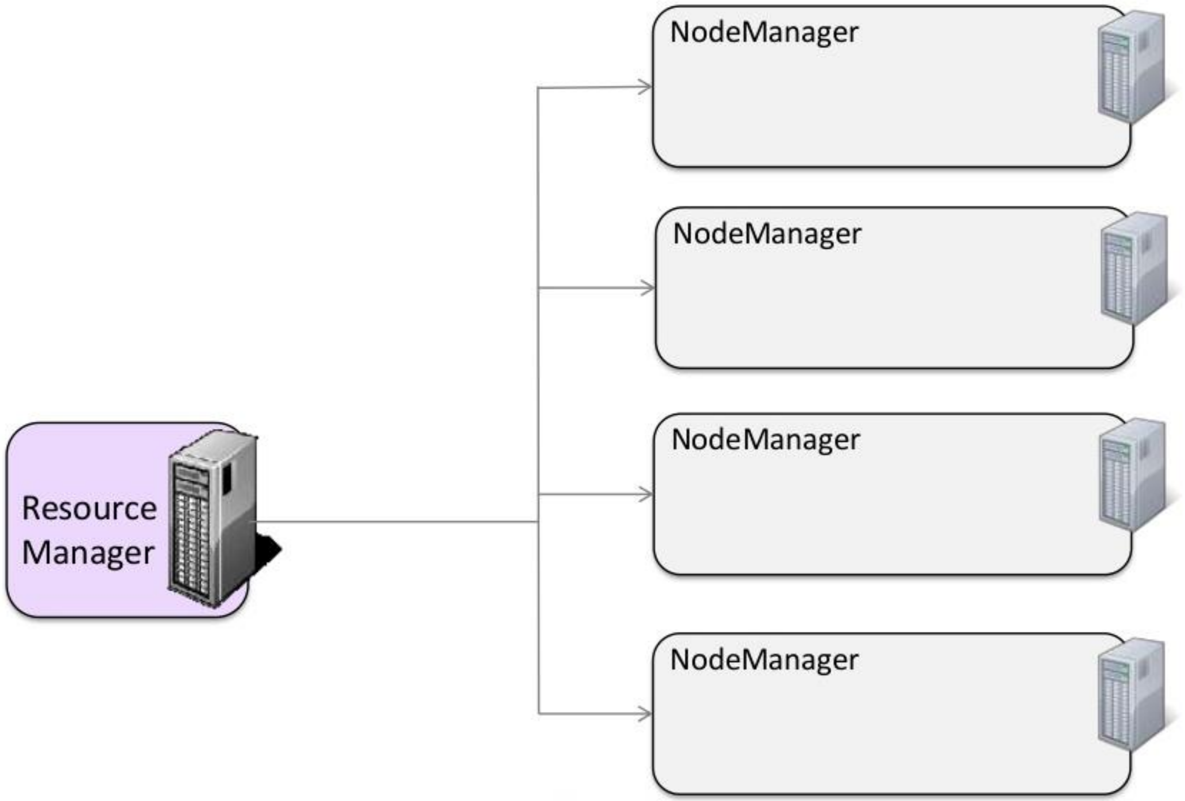
- **Containers**
  - –Created by the RM upon request
  - –Allocate a certain amount of resources (memory, CPU) on a slave node
  - –Applications run in one or more containers



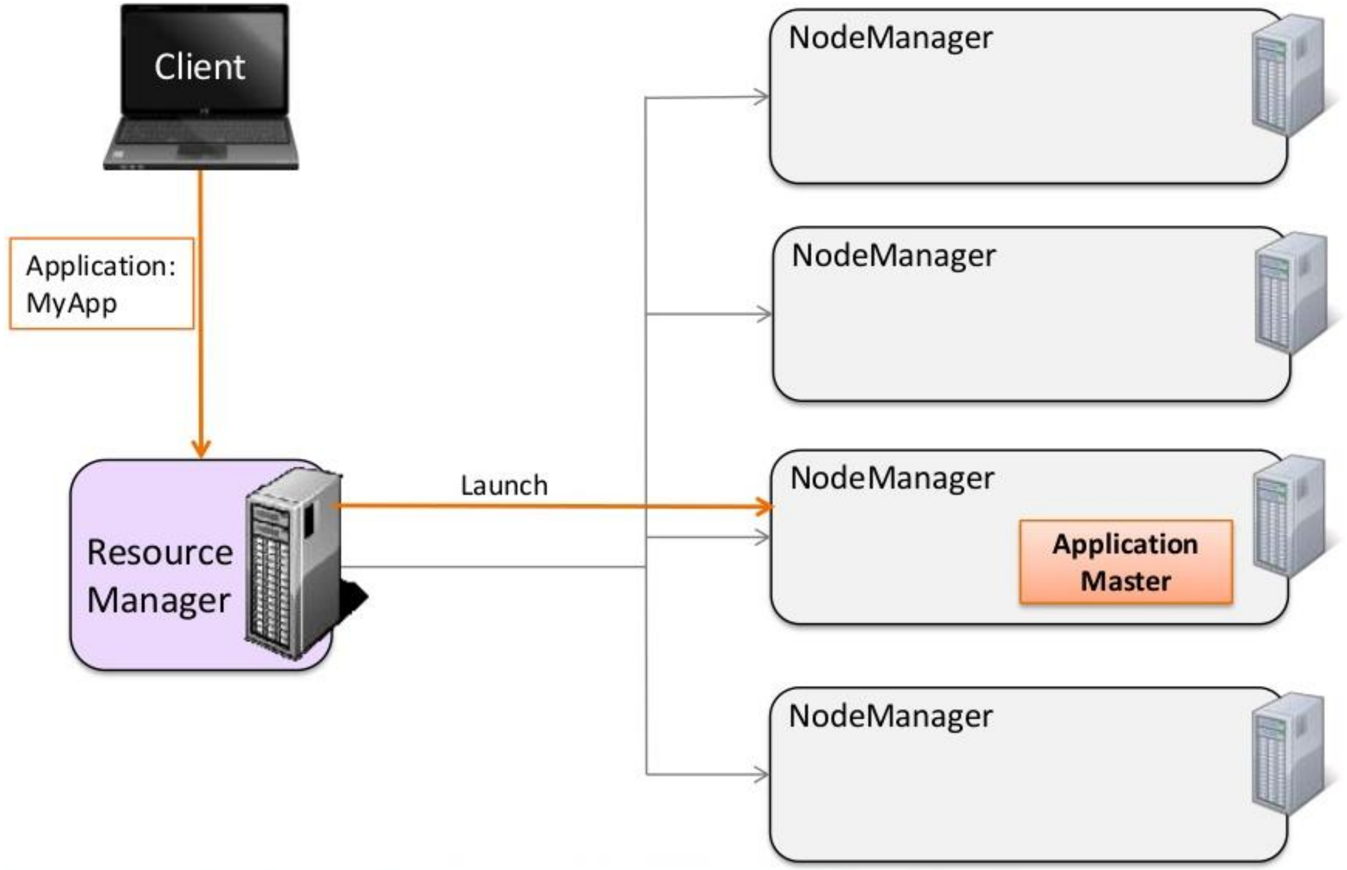
- **Application Master (AM)**
  - –One per application
  - –Framework/application specific
  - –Runs in a container
  - –Requests more containers to run application tasks



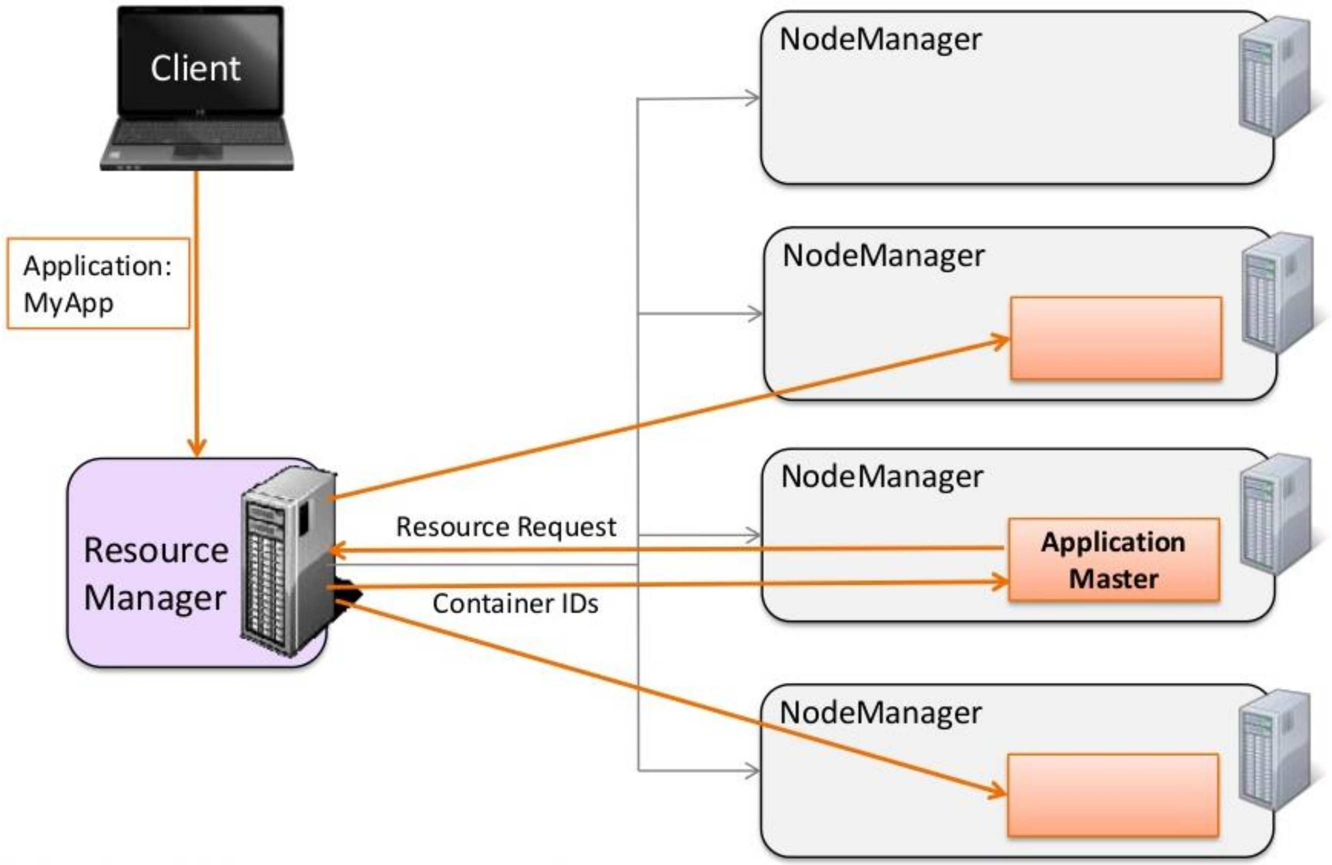
Yarn Cluster



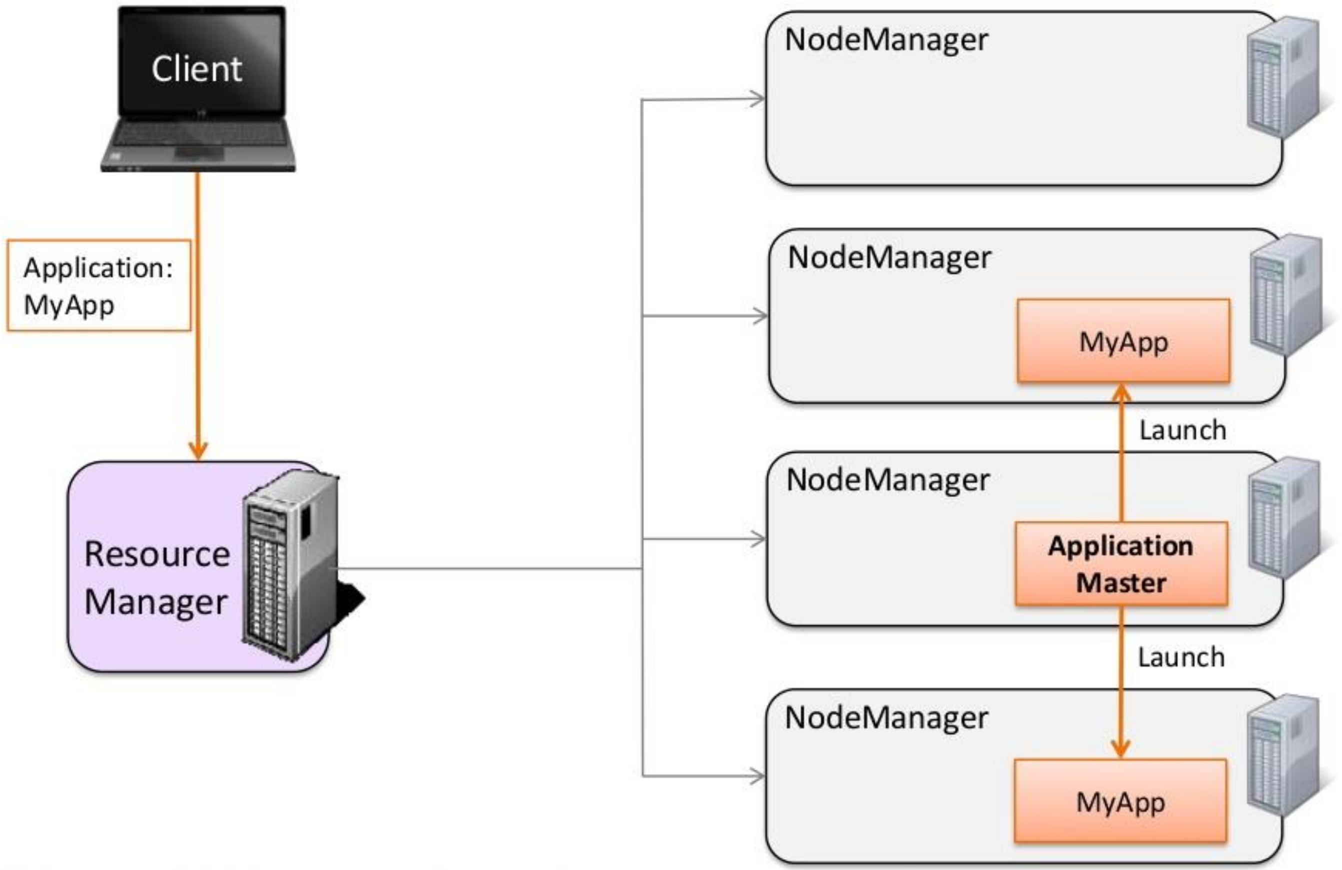
Yarn Cluster – Application Mechanix



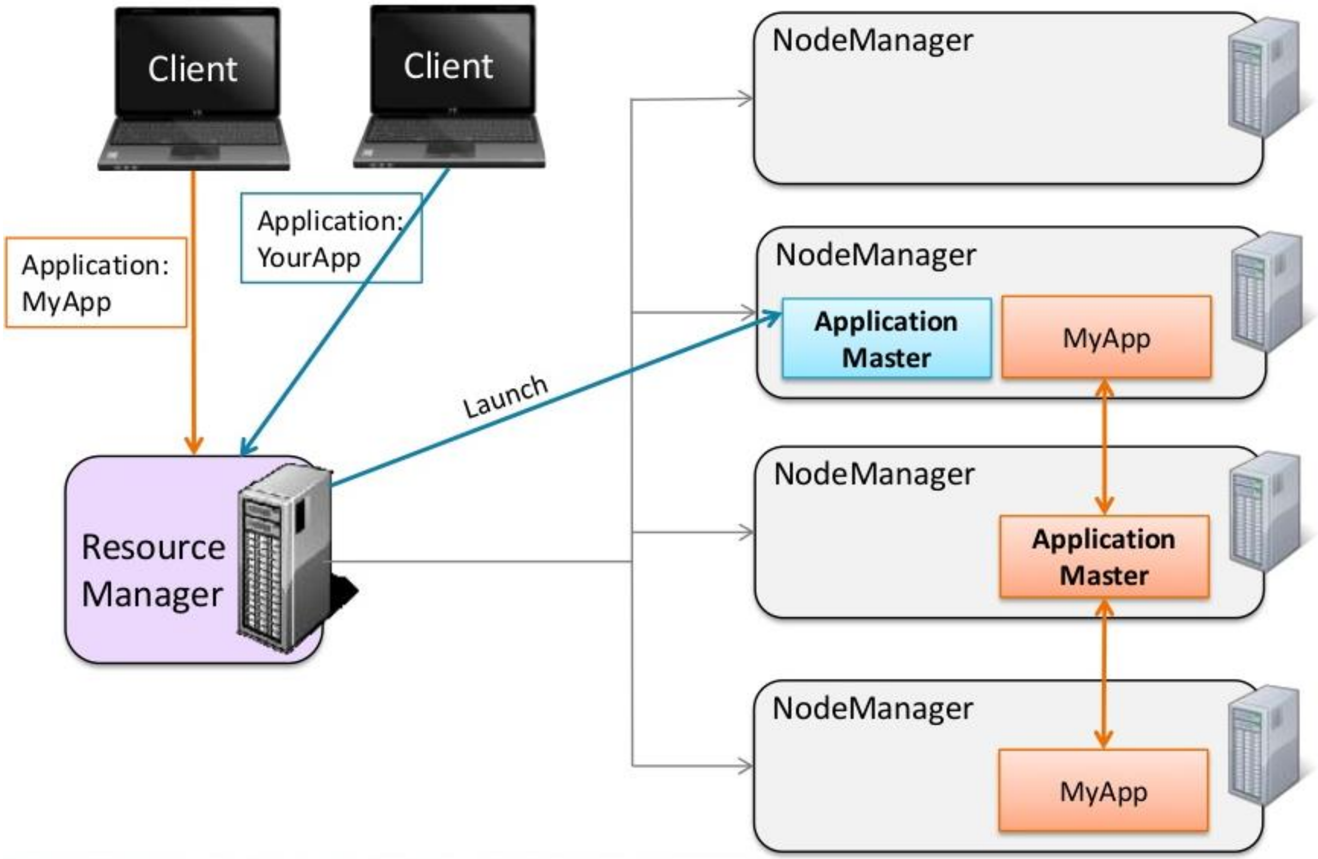
Yarn Cluster – App Lifecycle



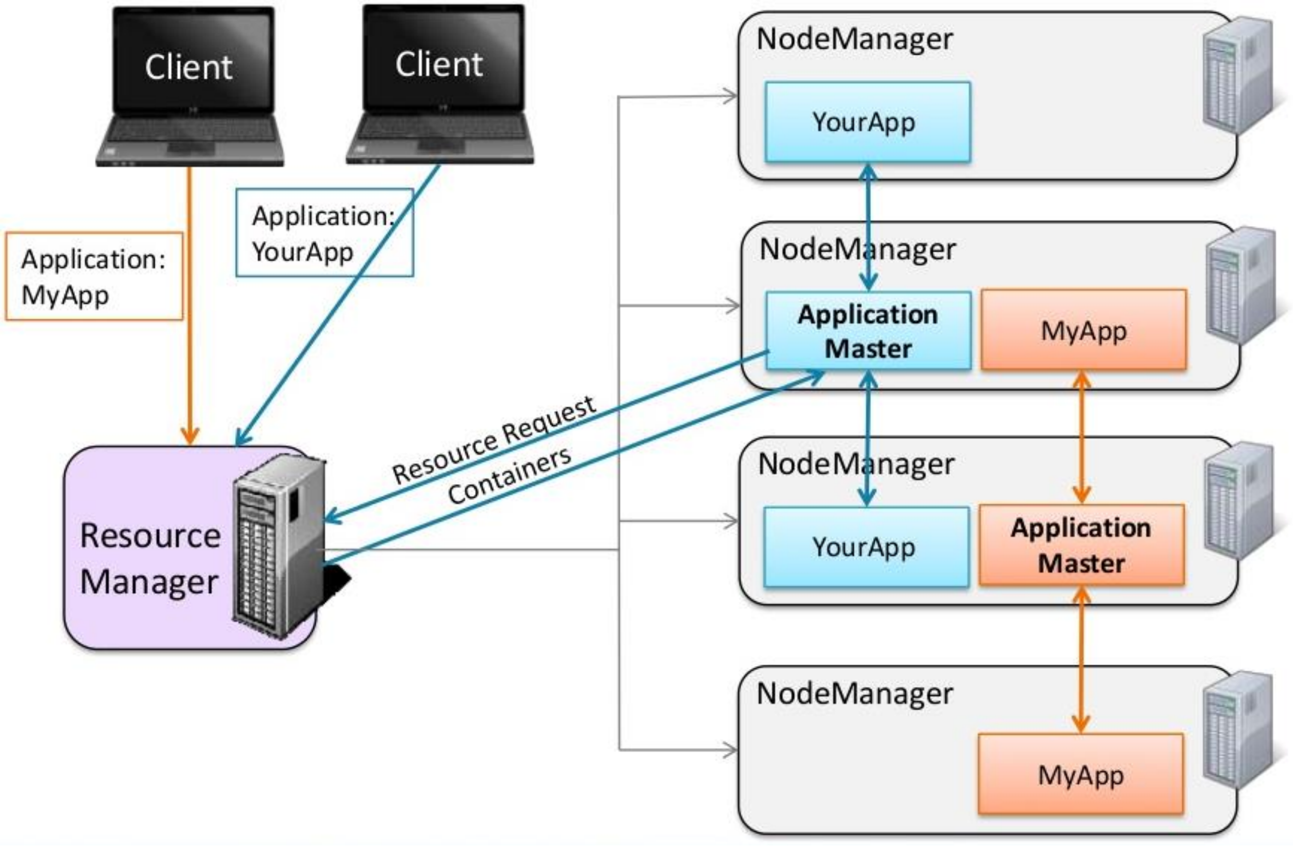
Yarn Cluster – App Lifecycle



Yarn Cluster – App Lifecycle



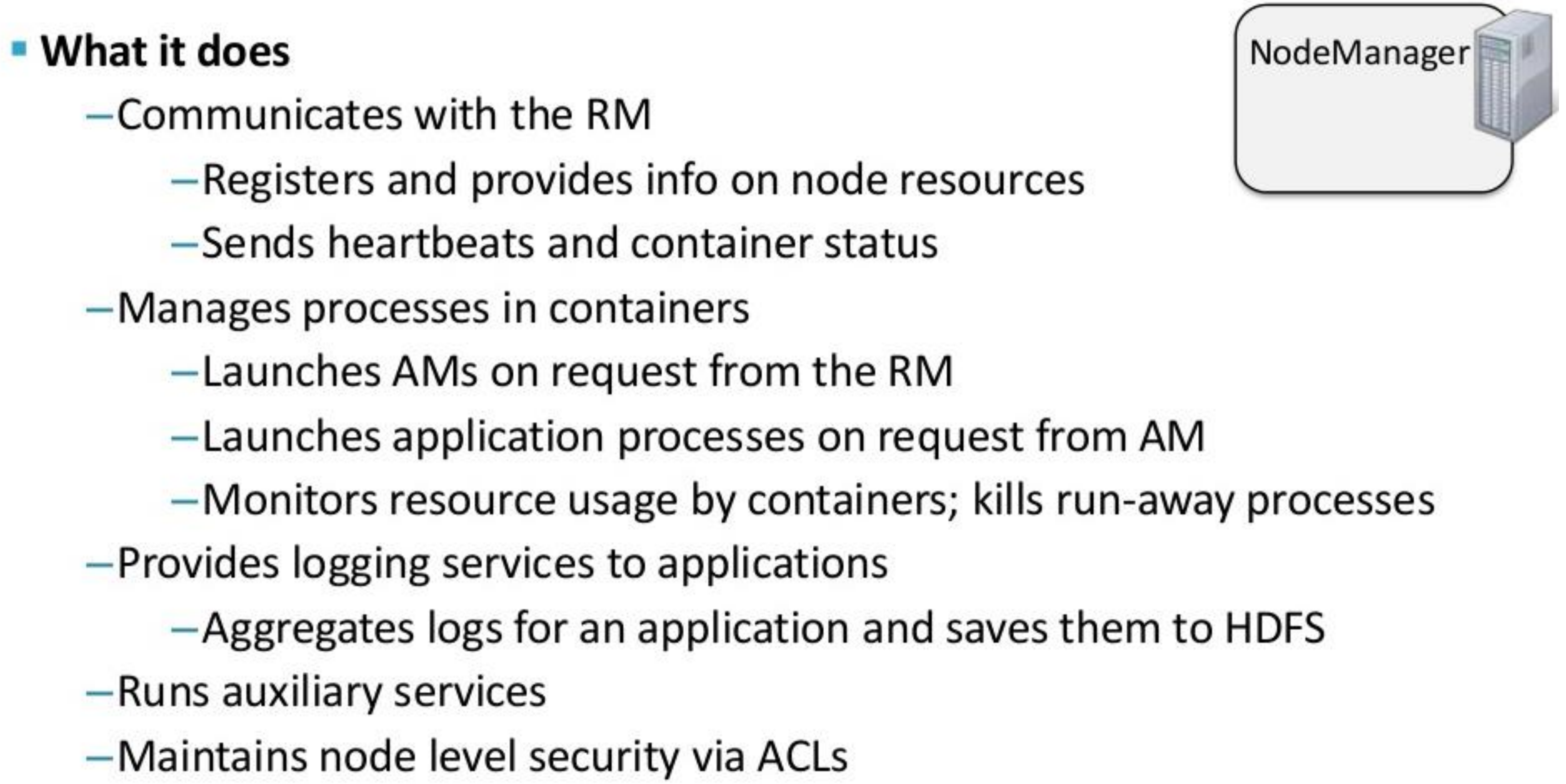
Yarn Cluster – App Lifecycle



Resource Manager

- **What it does**
  - Manages nodes
    - Tracks heartbeats from NodeManagers
  - Manages containers
    - Handles AM requests for resources
    - De-allocates containers when they expire or the application completes
  - Manages ApplicationMasters
    - Creates a container for AMs and tracks heartbeats
  - Manages security

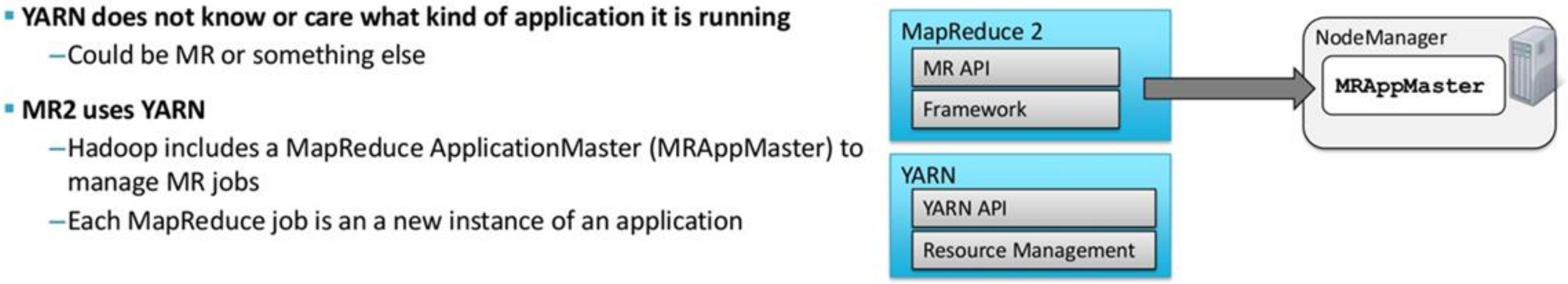
Node Manager



NodeManager

- **What it does**
  - –Communicates with the RM
    - –Registers and provides info on node resources
    - –Sends heartbeats and container status
  - –Manages processes in containers
    - –Launches AMs on request from the RM
    - –Launches application processes on request from AM
    - –Monitors resource usage by containers; kills run-away processes
  - –Provides logging services to applications
    - –Aggregates logs for an application and saves them to HDFS
  - –Runs auxiliary services
  - –Maintains node level security via ACLs

Yarn and MR2



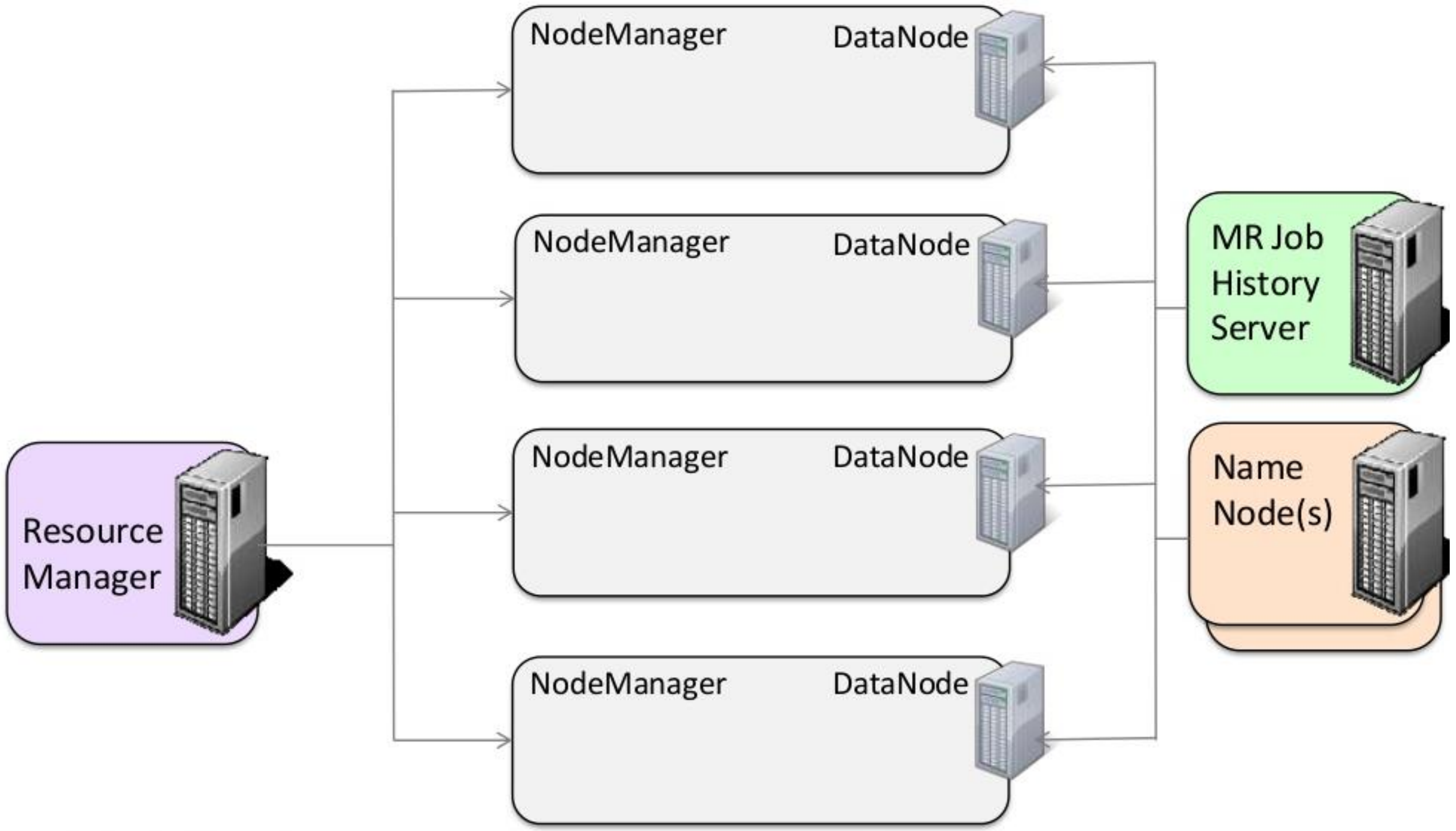
- **YARN does not know or care what kind of application it is running**
  - Could be MR or something else
- **MR2 uses YARN**
  - Hadoop includes a MapReduce ApplicationMaster (MRAppMaster) to manage MR jobs
  - Each MapReduce job is an a new instance of an application

Try running a non MR App (DistShell)

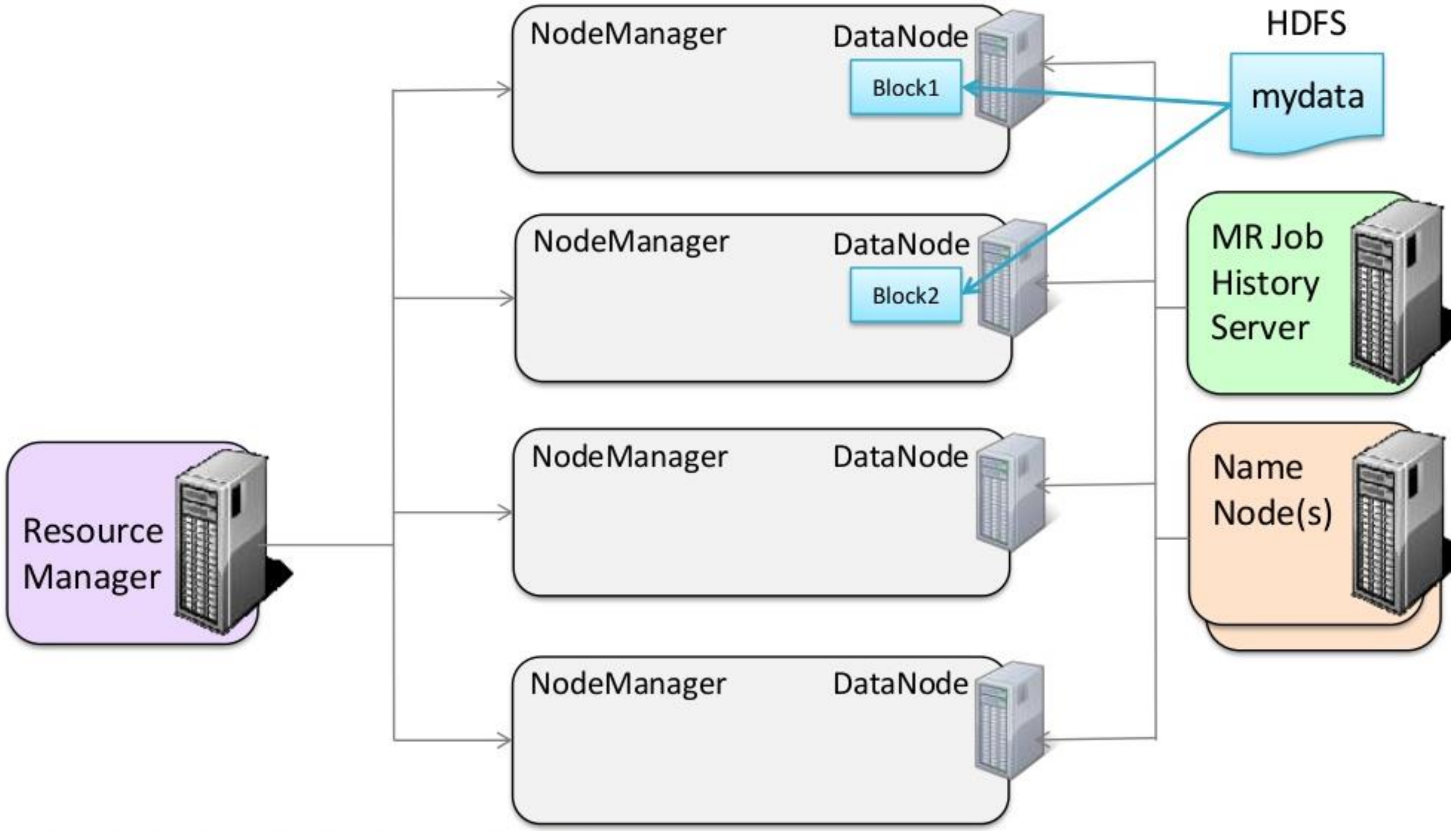
```
yarn jar $HADOOP_HOME/share/hadoop/yarn/*distributedshell-*.jar
org.apache.hadoop.yarn.applications.distributedshell.Client -jar $HADOOP_HOME/share/hadoop/yarn/*distributedshell-*.jar
-shell_command 'ls' -shell_args '-la`
```

Examine the logs using **yarn logs -applicationId <theAppID>**

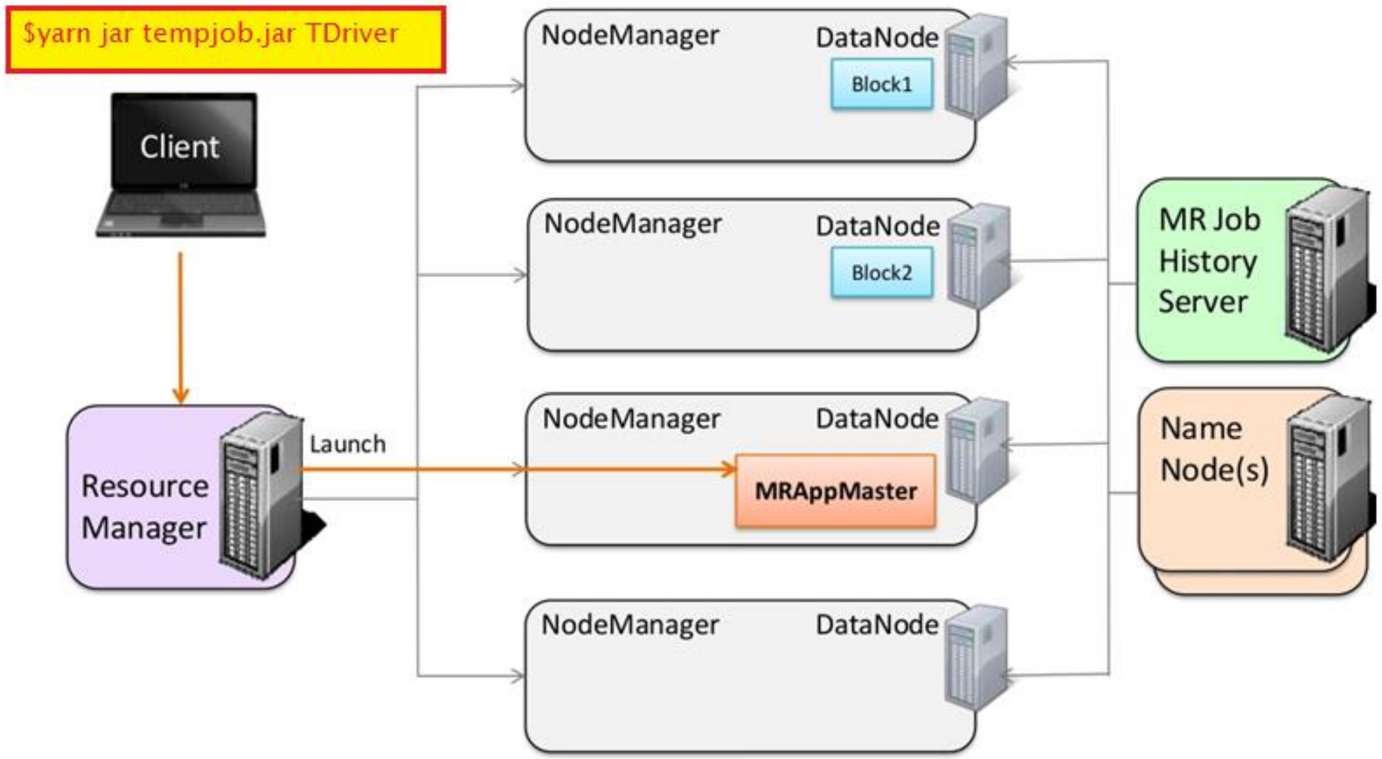
Yarn and MR2


NodeManager
DataNode
NodeManager
DataNode
NodeManager
DataNode
NodeManager
DataNode
Resource Manager
MR Job History Server
Name Node(s)

Yarn and MR2

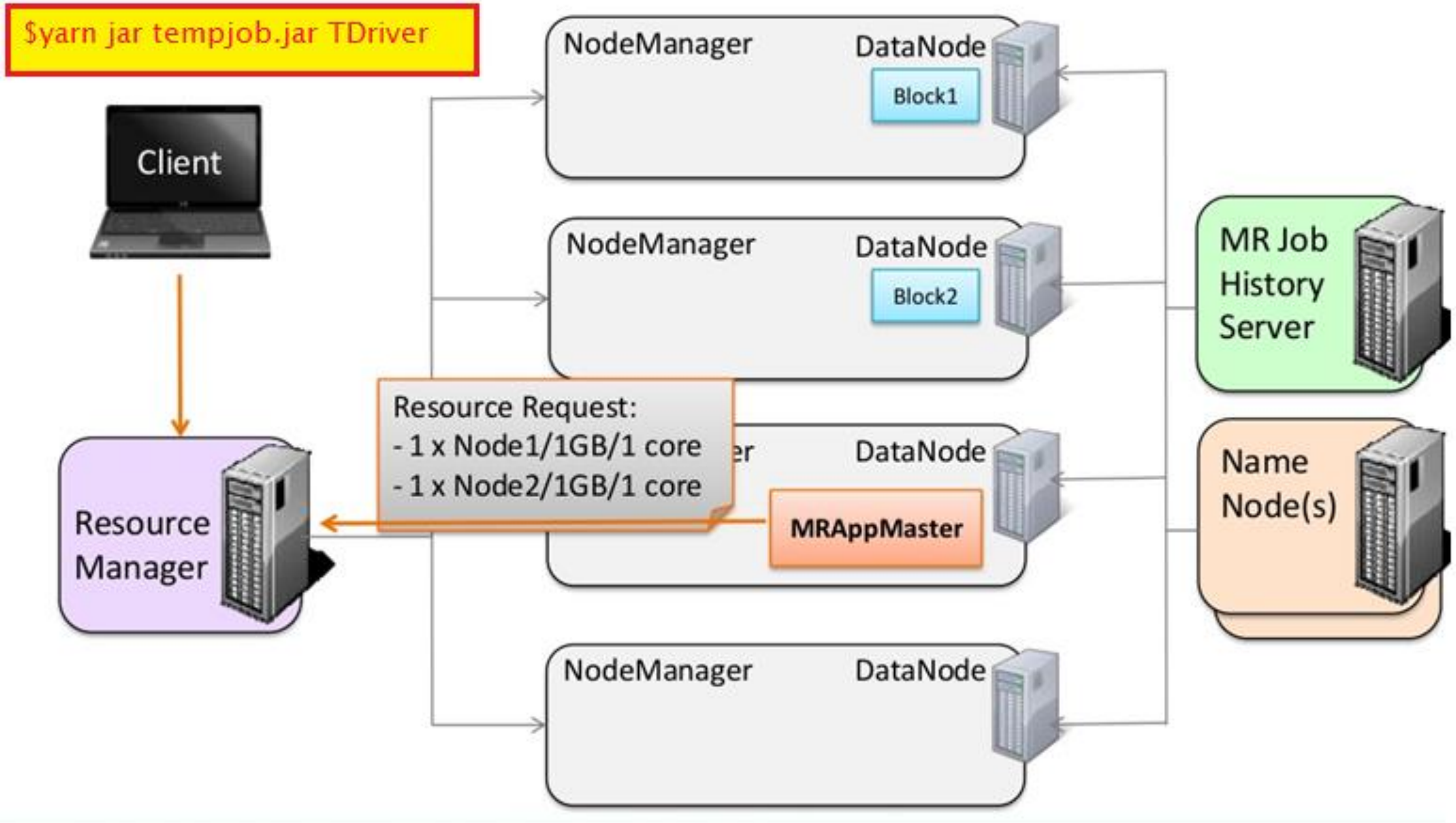

NodeManager
DataNode
Block1
HDFS
mydata
NodeManager
DataNode
Block2
MR Job History Server
Resource Manager
NodeManager
DataNode
Name Node(s)
NodeManager
DataNode

Yarn and MR2

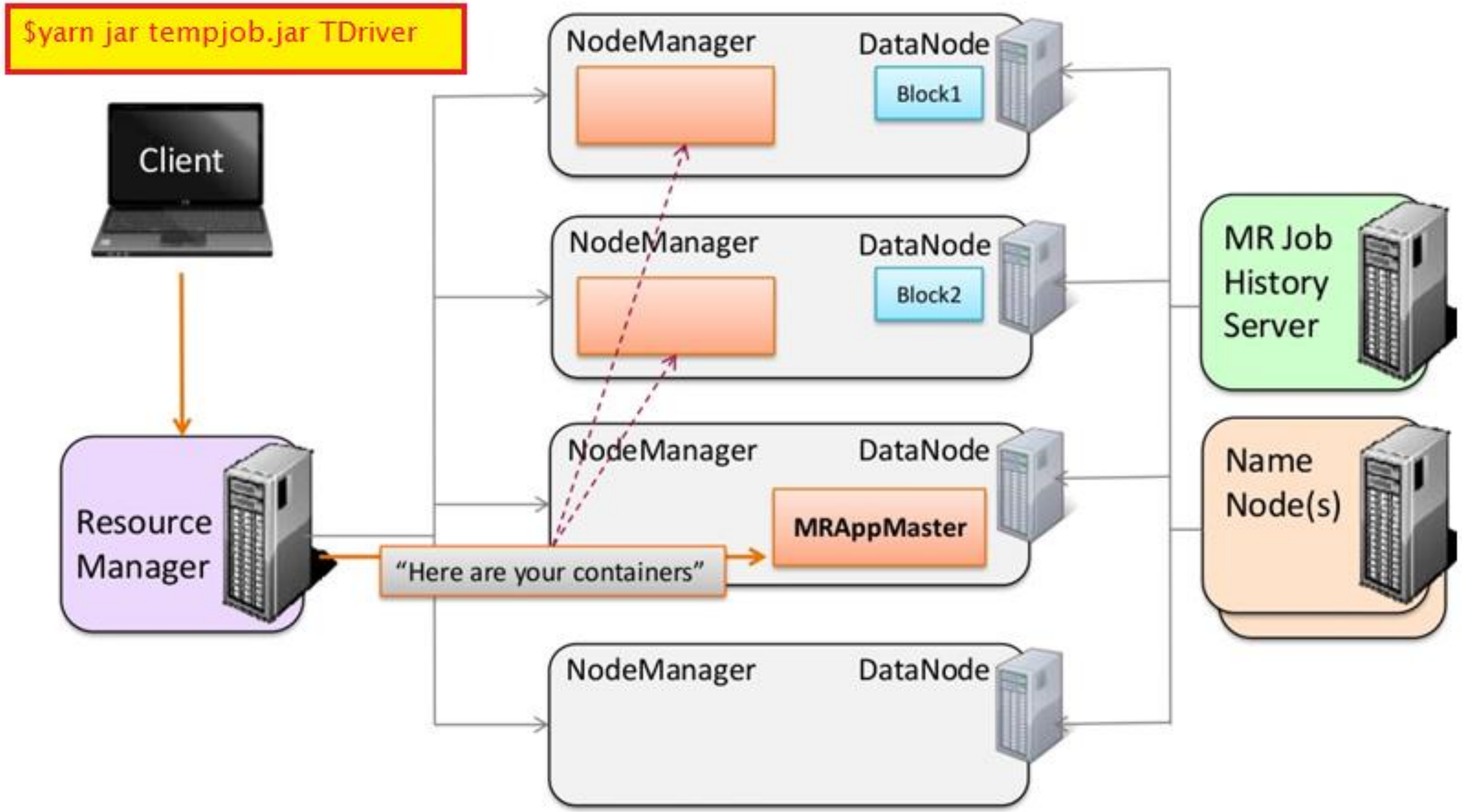


```
$yarn jar tempjob.jar TDriver
```

Yarn and MR2

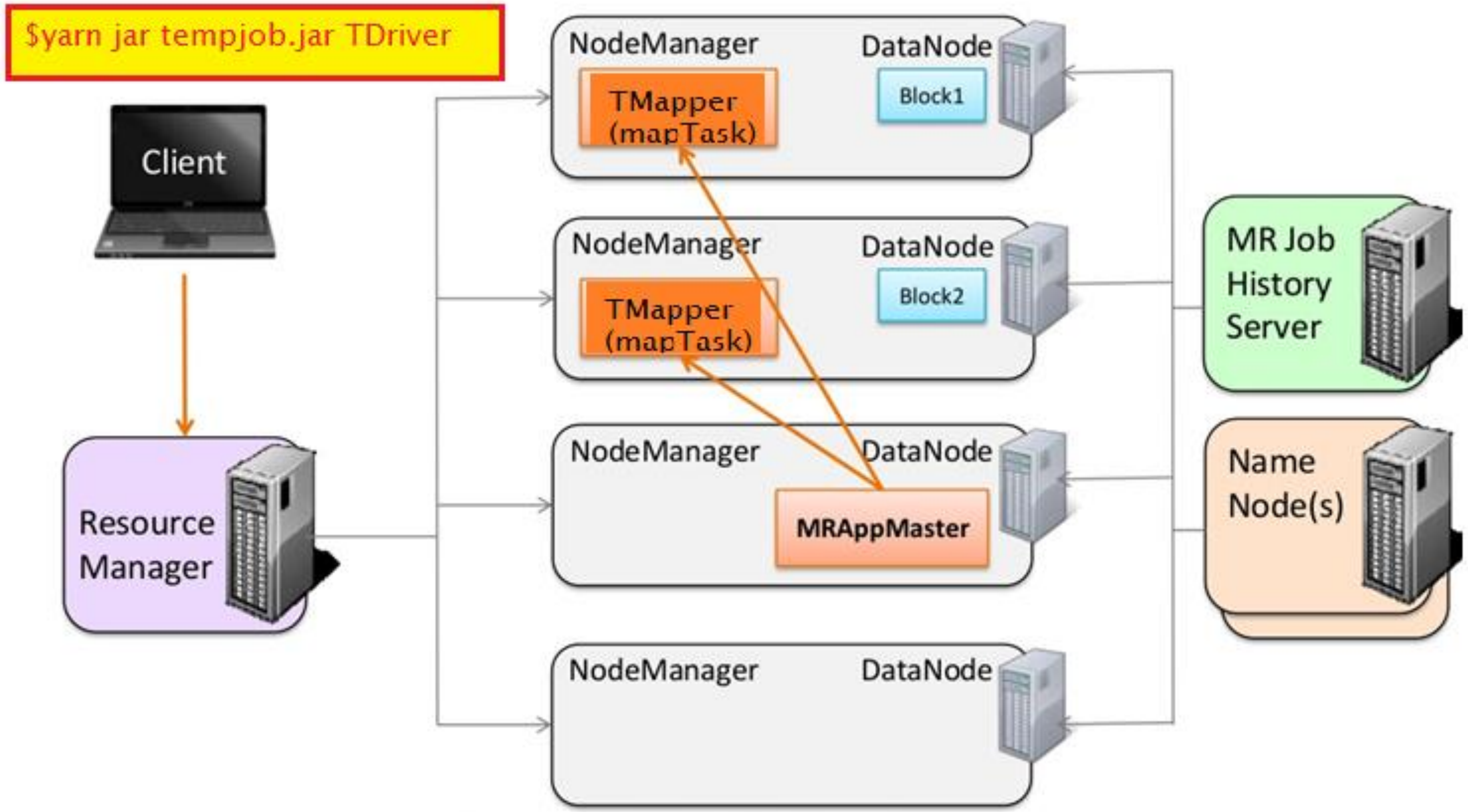

$yarn jar tempjob.jar TDriver
Client
NodeManager
DataNode
Block1
NodeManager
DataNode
Block2
Resource Request:
- 1 x Node1/1GB/1 core
- 1 x Node2/1GB/1 core
er
DataNode
MRAppMaster
Resource Manager
NodeManager
DataNode
MR Job History Server
Name Node(s)

Yarn and MR2



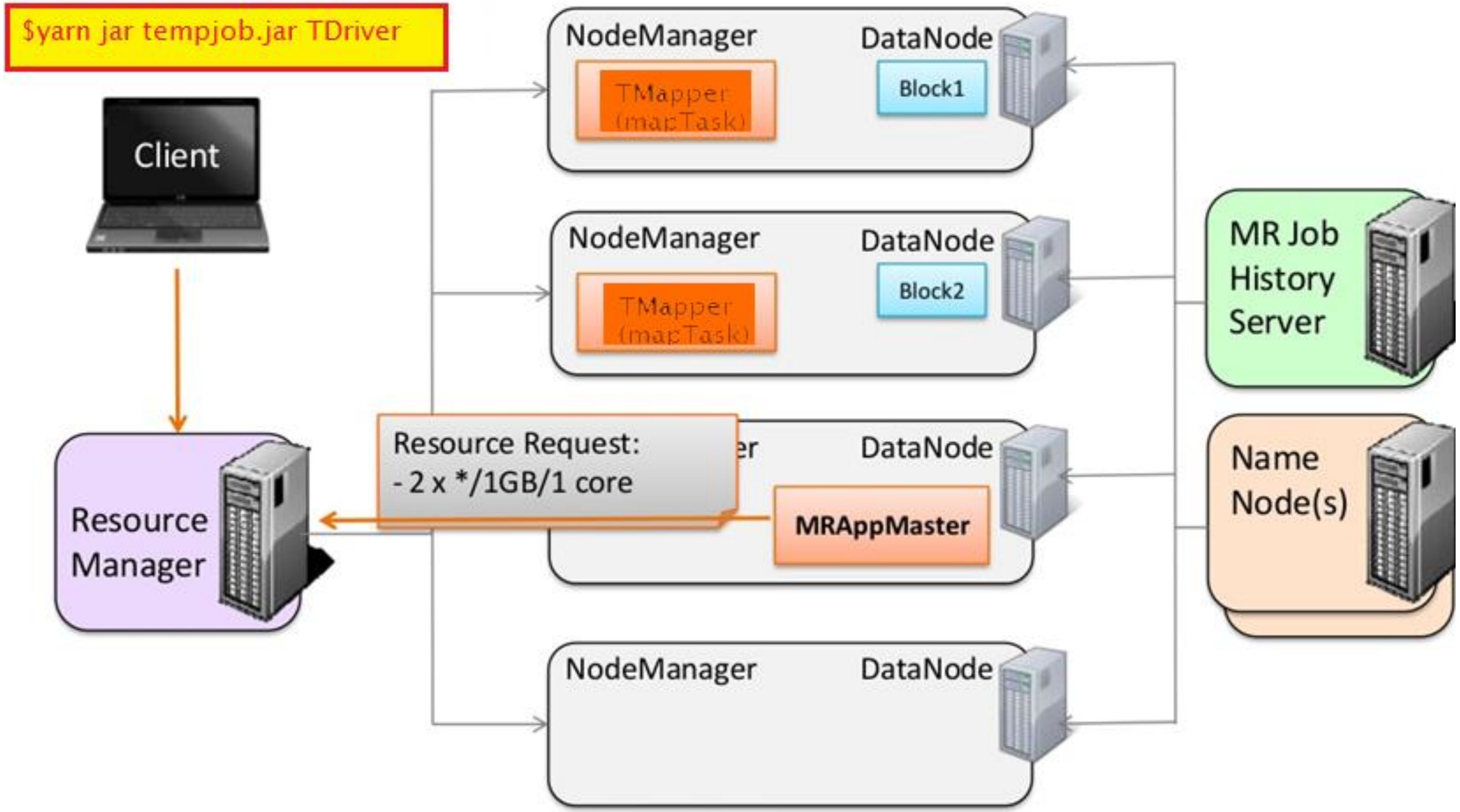
```
$yarn jar tempjob.jar TDriver
```

Yarn and MR2

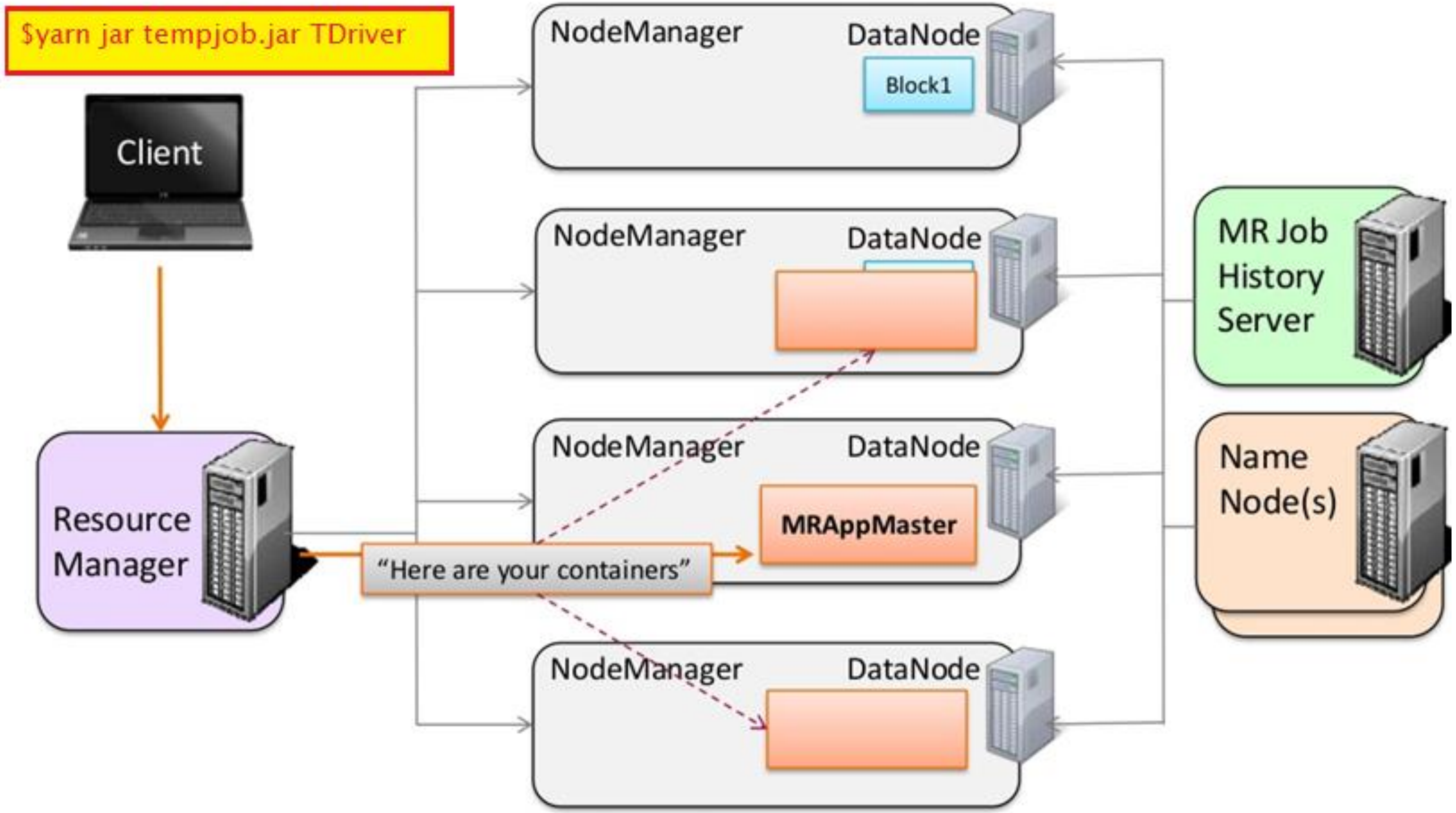


```
$yarn jar tempjob.jar TDriver
```

Yarn and MR2

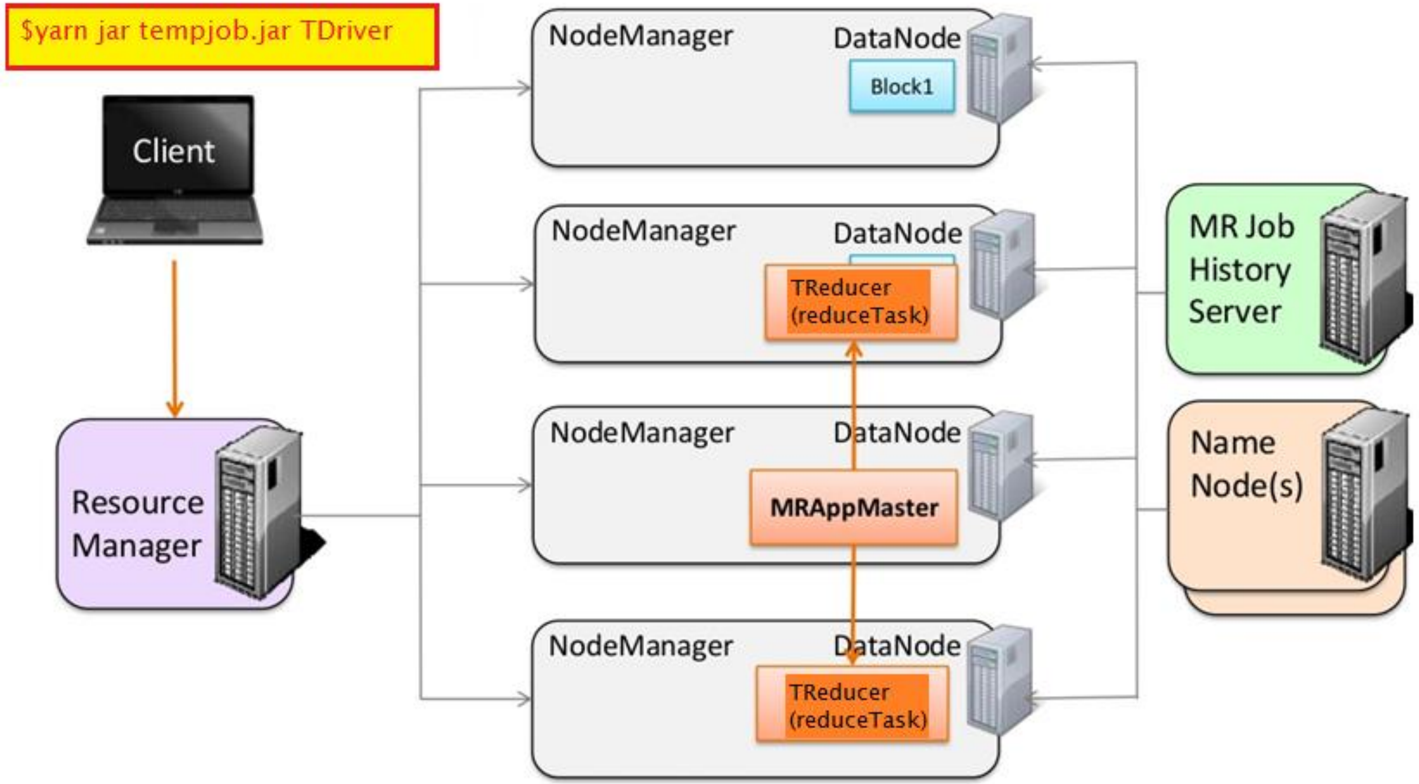

$yarn jar tempjob.jar TDriver
Client
Resource Manager
NodeManager
DataNode
TMapper (mapTask)
Block1
NodeManager
DataNode
TMapper (mapTask)
Block2
Resource Request:
- 2 x */1GB/1 core
er
DataNode
MRAppMaster
NodeManager
DataNode
MR Job History Server
Name Node(s)

Yarn and MR2

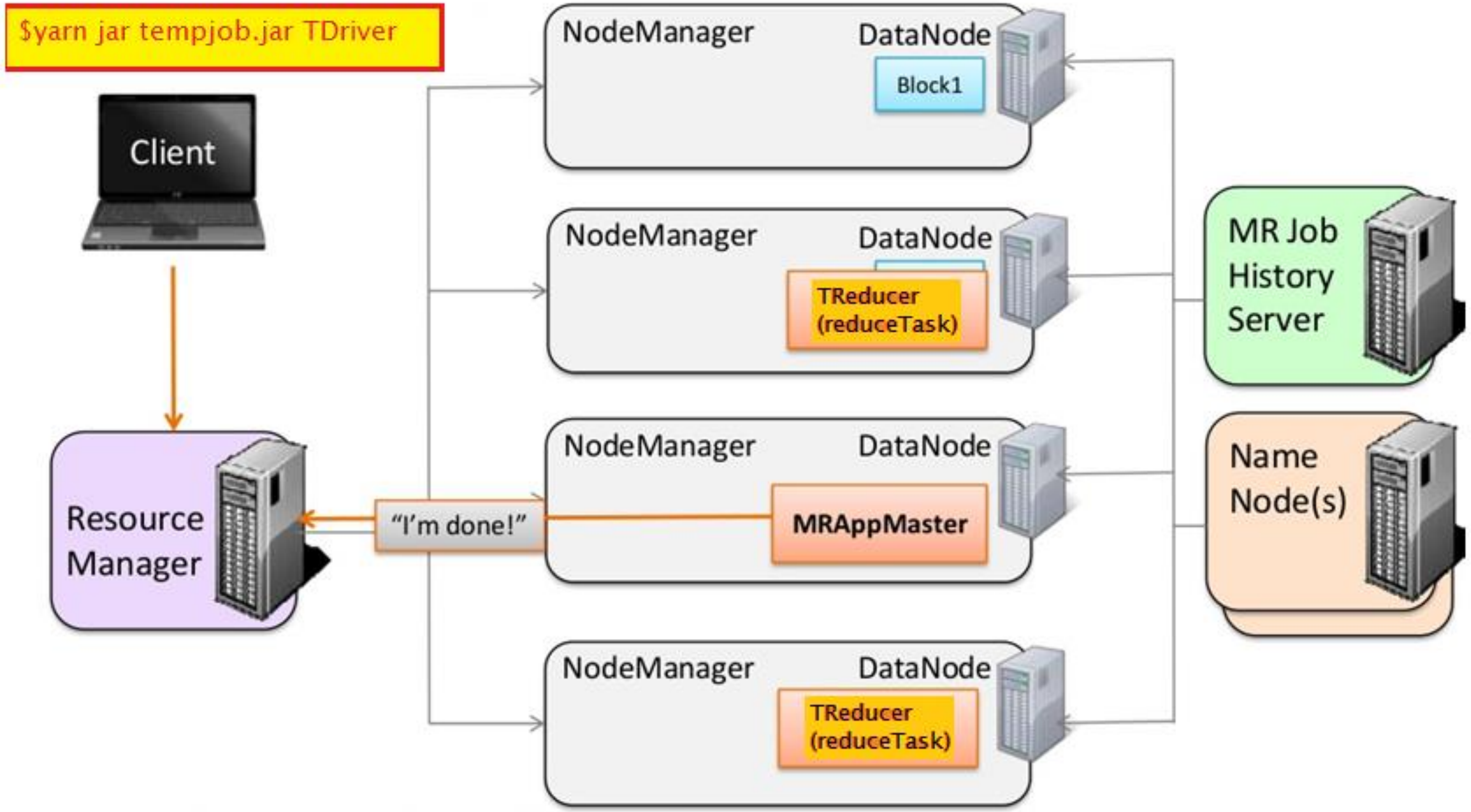


```
$yarn jar tempjob.jar TDriver
```

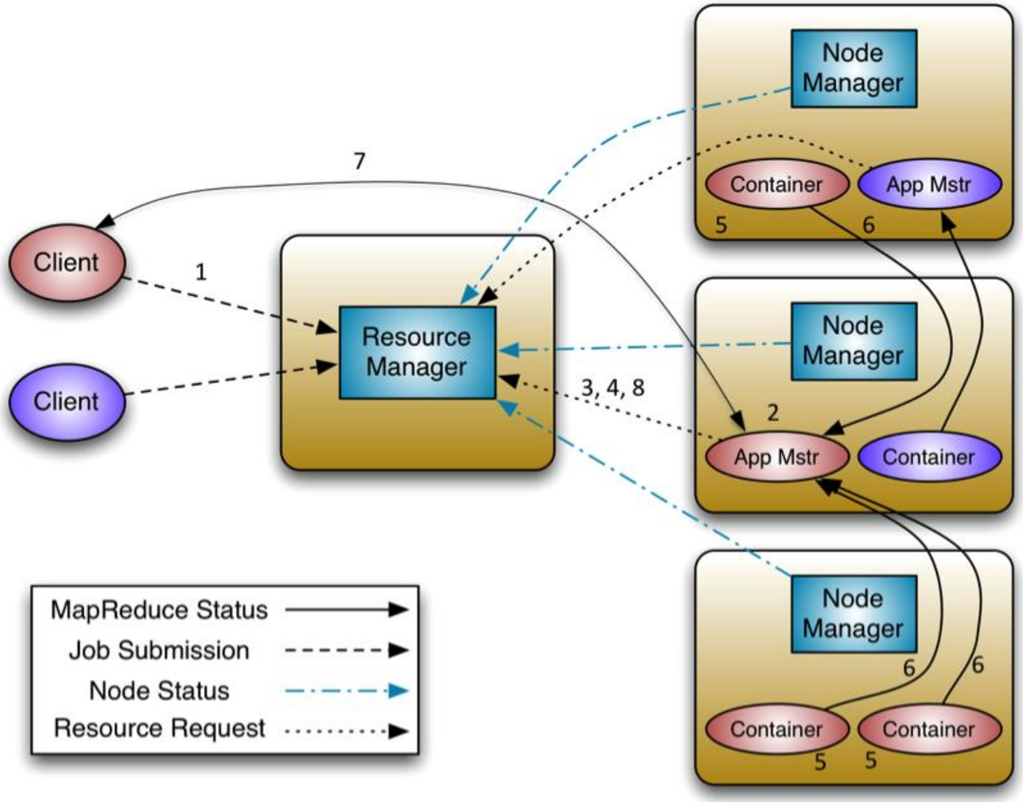
Yarn and MR2


$yarn jar tempjob.jar TDriver
Client
Resource Manager
NodeManager
DataNode
Block1
NodeManager
DataNode
TReducer (reduceTask)
NodeManager
DataNode
MRAppMaster
NodeManager
DataNode
TReducer (reduceTask)
MR Job History Server
Name Node(s)

Yarn and MR2


$yarn jar tempjob.jar TDriver
Client
Resource Manager
"I'm done!"
NodeManager
DataNode
Block1
NodeManager
DataNode
TReducer (reduceTask)
NodeManager
DataNode
MRAppMaster
NodeManager
DataNode
TReducer (reduceTask)
MR Job History Server
Name Node(s)

The LifeCycle of an App in Yarn



Some assignments

- Some more MR patterns
  - Find how many Unigrams are there in a file
  - How many unique bigrams are there in your text file? A bigram is a N-gram of two words.
    - Consider the text “A cat jumped over a wall to catch a rat with no fat”
    - Bigrams are two words, for the above sentence the bigrams are
      - a cat
      - cat jumped
      - jumped over
      - over a
      - a wall
      - wall to
      - ….
  - Find how many anagrams occur in a file. Anagram is a word formed by rearranging the letters of another, such as spar, formed from rasp.

Pending Assignment

- Pick real time data for All India seasonal Annual Min/Max temperatures series from 1901 – 2014) from below linkhttps://data.gov.in/catalog/all-india-seasonal-and-annual-minmax-temperature-series
- The data is layed out year wise, with min and max, but not averages.
- Compute average temperature year wise and spit data in the format of Year, Average.
- We will use this result for some analytics.