# MR UNIT

- ✓ MR unit is a **Testing Framework** for MapReduce applications.
- ✓ MRUnit allows you to test your mapper and reducer classes in **Isolation**, without having to run a full MapReduce job.
- ✓ This can help you to identify and fix bugs early on.

Note: **Isolation** in MRUnit means that you can test your mapper and reducer classes without having to run a full MapReduce job.

**To develop a MapReduce Application Using MRUnit, you can follow these steps:**

1. **Create Your MapReduce Application -** This includes writing your mapper and reducer classes.

2. **Add MRUnit To Your Project -** You can do this using Maven or Gradle.

3. **Write Unit Tests For Your Mapper And Reducer Classes -** MRUnit provides a MapReduceDriver class that you can use to test your mapper and reducer classes separately.

4. **Run Your Unit Tests -** You can use your IDE's built-in test runner or a command-line tool like mvn test or gradle test.

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mrunit.mapreduce.MapDriver;
import org.junit.Test;

public class WordCountMapperTest {

    @Test
    public void testMapper() throws Exception {

        // Create a new MapDriver object.
        MapDriver<Text, IntWritable, Text, IntWritable> driver = new MapDriver<>();

        // Set the mapper class.
        driver.setMapper(WordCountMapper.class);

        // Add some input data.
        driver.addInput(new Text("hello world"));

        // Run the mapper.
        driver.runTest();

        // Verify the output data.
        driver.assertOutput(new Text("hello"), new IntWritable(1));
        driver.assertOutput(new Text("world"), new IntWritable(1));
    }
}
```

- We  can write similar **unit tests for your reducer class**.
- Once we have written unit tests for your mapper and reducer classes, you can **Run** them to verify that your MapReduce application is working as expected.

**Here are some additional tips for developing MapReduce applications using MRUnit:**

- Use MRUnit to test your mapper and reducer classes in **isolation**. This will help you to **Identify And Fix Bugs Early** on.

- Write **Comprehensive Unit Tests That Cover All Of The Possible Cases** that your mapper and reducer classes may encounter.

- Use **MRUnit** to test your MapReduce application as a whole. This will help you to ensure that all of the components of your application are working together correctly

To develop a MapReduce application and **Run It Locally On Test Data**, you can follow these steps:

**Write Your MapReduce Code** - This involves writing two classes: a Mapper class and a Reducer class. The Mapper class takes each input record and splits it into key-value pairs. The Reducer class takes the key-value pairs generated by the Mapper class and combines them into a single output value.

**Compile Your MapReduce Code -** Once you have written your MapReduce code, you need to compile it into a **JAR file**.

✓ This can be done using the following command:

" **javac -d classes src/main/java/*.java jar -cvf mapper-reducer.jar -C classes** “

**Configure Your MapReduce Job** - You need to configure your MapReduce job by creating a **job.xml file**. This file specifies the input and output directories for your job, as well as the Mapper and Reducer classes to use.

**Run your MapReduce job -** To run your MapReduce job, you can use the following command:

" **hadoop jar mapper-reducer.jar mapper.Mapper reducer.Reducer input_directory output_directory** “

- ✓ For example, if your input directory is input and your output directory is output, you would run the following command:

" **hadoop jar mapper-reducer.jar mapper.Mapper reducer.Reducer input output** “

**View Your Job Output -** Once your MapReduce job has finished running, you can view your job output in the output directory.