Amazon Web Services Setup

The initial version of this document was created by Joe Sackett in September of 2017. Current revision: January 2019 by Mirek Riedewald (updated for Spring, 2020 by Gene Cooperman)

In general, you should follow the latest AWS documentation, not this document. This document may be out of date and is only provided for convenience: it will hopefully help you navigate the latest setup process more easily. After all, most steps will be similar :)

This is targeted documentation for getting started with Amazon Web Services, in particular Simple Storage Service (S3) and Elastic MapReduce (EMR) for executing custom Hadoop and Spark programs. You will find links from the AWS documentation as well as explanations and tips for a quick and successful configuration. This document also covers access to all relevant services through the AWS Command Line Interface (CLI) and the AWS SDK providing programmatic control of all AWS resources.

Getting Started

For getting-started guides on AWS, take a look at https://aws.amazon.com/getting-started/

The first step is to create your account. Account options change all the time, but typically free accounts will not work for this course because they limit access to services we will need. Most likely the best option is a student account through AWS Educate (https://aws.amazon.com/education/awseducate/) where you should receive at least $100 (reports in 2020 seem to indicate that this is now $75) in free AWS credit. (Try to avoid accounts with terms such as “free” or “starter” in the title. They often provide only limited functionality. Either way, it is important to read the account description carefully to find out about those restrictions.)

After logging in, you probably will need to set up your account. Whenever presented with a choice of region of the world for a service, pick the closest geographic area (e.g., US-East). Make sure you always choose the same region.

Security

You will need to set up access keys so that your application can be launched directly from your laptop. Take a look at the AWS document about best practices for access keys (https://docs.aws.amazon.com/general/latest/gr/aws-access-keys-best-practices.html as of 01/12/2019) for more information.

S3

“Amazon Simple Storage Service (Amazon S3) is storage for the Internet. You can use Amazon S3 to store and retrieve any amount of data at any time, from anywhere on the web.” This global file storage can be used by EMR programs (e.g., Hadoop Mapreduce and Spark) as well as user programs executing on either your local machine or on EC2. Read the relevant documentation on the AWS Web sites to become more familiar with it.

For this course, when running a MapReduce or Spark program on EMR, input data and code (jar files) need to reside on S3. After program termination, log files and output will also end up on S3. This requires you to create a “bucket” that will contain the corresponding folders. The provided Makefile will do the folder creation and file copying for you, assuming you properly set the variables at the top of the Makefile.

EMR

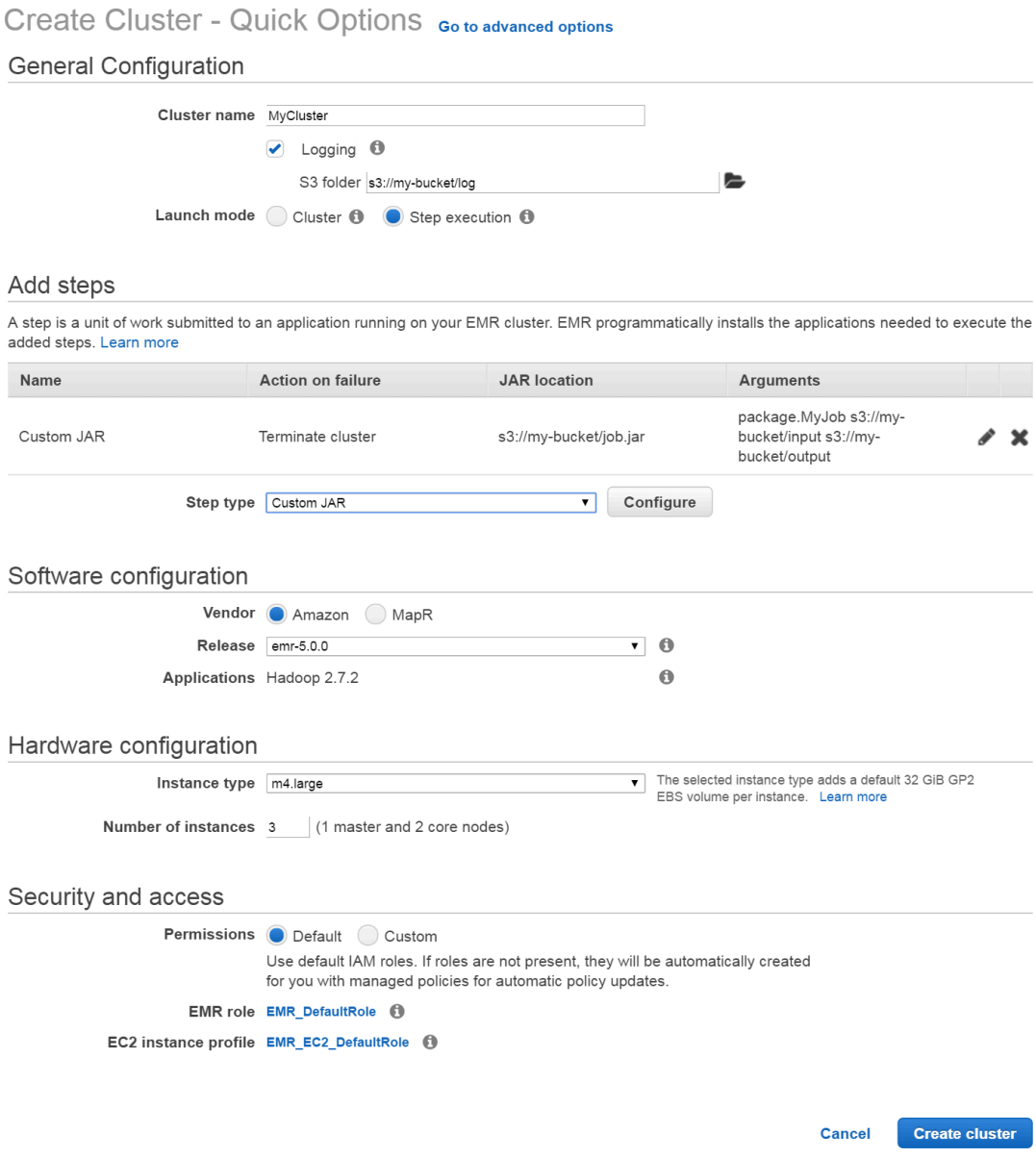
“Amazon Elastic MapReduce (Amazon EMR) is a managed cluster platform that simplifies running big data frameworks, such as Apache Hadoop…” Like for S3, we recommend you first read some of the documentation on AWS to familiarize yourself with the service.

Note: If you want to save money, you can use EC2 instead of EMR. In EC2, machine instances are “barebones,” i.e., you need to install Hadoop and Spark there yourself. You should be able to find scripts on the Web that automate and simplify this setup. We only recommend this for advanced users. EMR is much easier to use, because the required environment is already pre-configured. It costs a little more per machine instance, but if you follow our advice for development and testing before going full scale, then the $100 free credit should get you through the entire course.

Normally, you will use the Makefile for setting up and executing MapReduce and Spark jobs on EMR. However, you may also want to run a small job from the AWS GUI to get a better feel for the environment. This process requires the following main steps:

1. Package your application in a JAR and test that it works locally. It is necessary that your program take command-line parameters for the input and output paths as they will be S3 URIs on EMR.
2. Use the S3 web interface to create a bucket---let’s call it my-bucket here---and two subdirectories (input & log).
    a. Upload your JAR file into the my-bucket root.
    b. Upload your input files into the my-bucket/input folder.
3. Go to the EMR Console and create a cluster. This should approximately require the following (the screenshot is from 2017):
    a. Specify cluster name
    b. Set Log location to: s3://my-bucket/log

c. Select “Launch mode” of “Step execution”
d. Select “Step type” to “Custom JAR”
e. Click configure button. Select the JAR. Assuming your main program accepts input and output folders as parameters, set the Hadoop program command line arguments to include your:
    i. job class
    ii. s3://my-bucket/input
    iii. s3://my-bucket/output
f. Choose the latest release of emr stack (different than graphic below).
g. Click Create Cluster.



The new cluster is now visible in the Cluster List. It will take a few minutes to be provisioned but execute quickly once it spins up. The program output will be in its directory but likely separated into multiple files due to the reducers executing on different machines in the cluster.

Note: it may be necessary to run an EMR job manually from the Web GUI one time in order for AWS to create the default security roles and to determine your region's subnet-id.

AWS Command Line Interface (CLI)

"The AWS Command Line Interface is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts." Read the corresponding documentation on the AWS site to find out how to install and configure CLI.

Configuration should take the following steps (may have changed slightly):

- OBSOLETE: Create IAM account (unless you already did it) - Continue to use your AWS Educate account here.
- Assign necessary privileges to new account (unless you already did it)
- Download keys for this account (unless you already did it)
- Install CLI
- Configure CLI
- Execute CLI commands

OBSOLETE: Create IAM Account - Continue to use your AWS Educate account

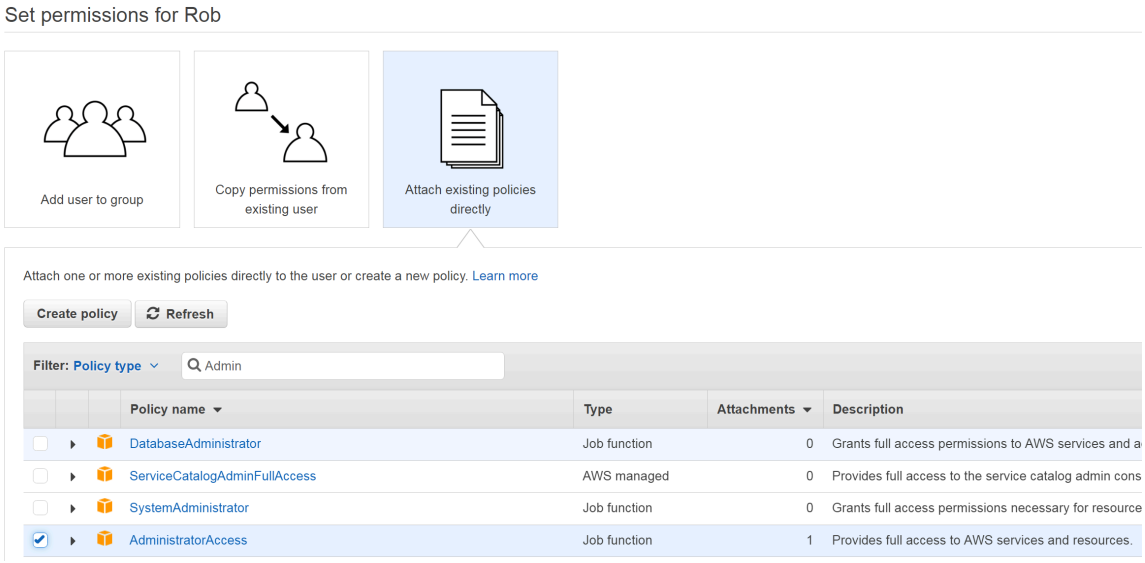
For security purposes, it is important to create an IAM account for working with AWS services rather than using your "root" account created at sign-up. Follow the AWS documentation, which should be something like this:

1. IAM service -> select Users from left menu.
2. Click Add User button at top.
3. Enter your chosen User Name in the top field.
4. Make sure to select Programmatic access and AWS Management Console access checkboxes in the Access Type section.
    a. Choose a Custom Password
    b. Deselect Require Change Password if you want to keep your password.
5. Click button Next: Permissions

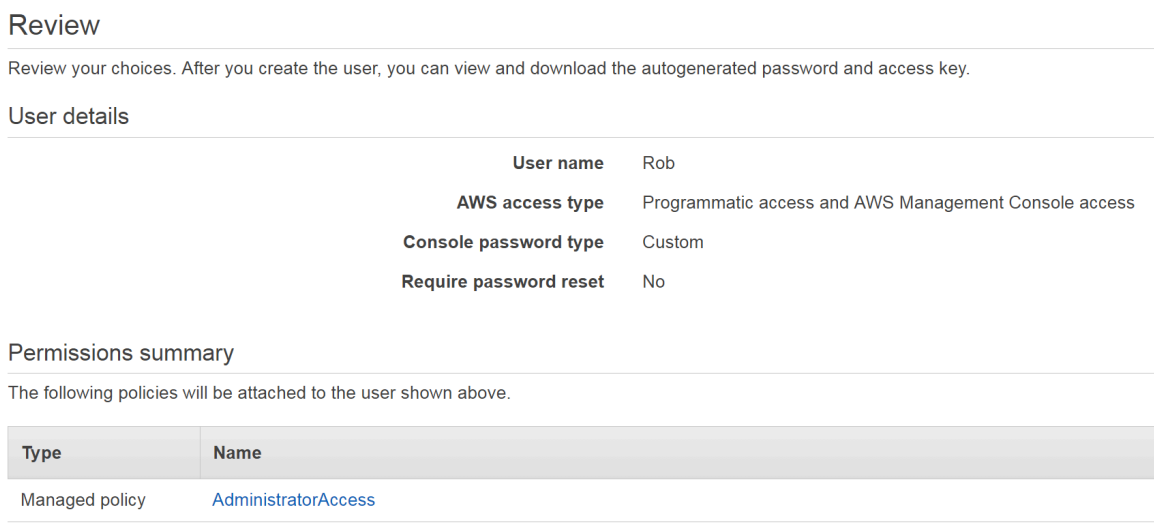
OBSOLETE: Assign Appropriate Privileges to IAM Account

Administrator privileges will allow access and use of the AWS services but prevent access to billing and general account maintenance. The AWS security documentation might also suggest even more limited privileges. Please follow their suggestions. The following steps assume you are assigning administrator privileges:

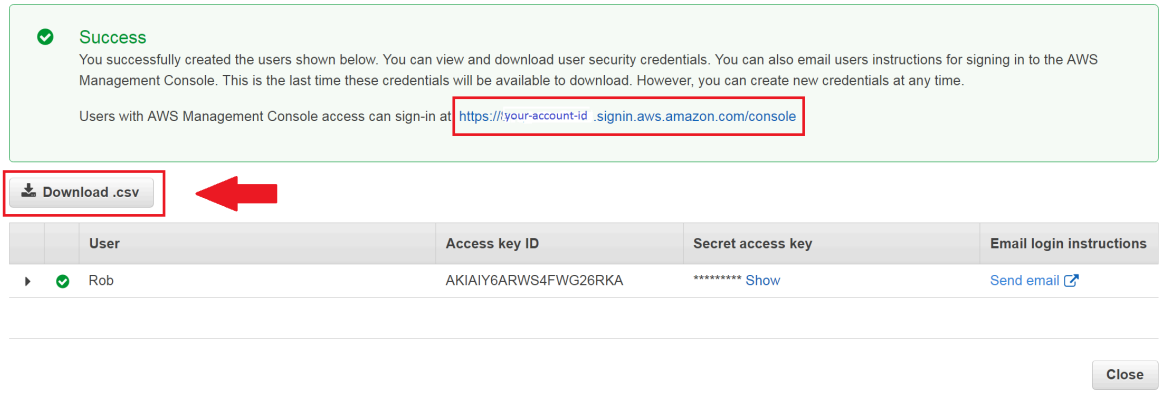
6. Although you may create an Administrator group and assign the new account to it, it is simplest to directly give the new account an Administrator policy. Choose:
   Attach existing policies directly
7. Type Admin partial string to filter the policies in the list below.
8. Select Administrator Access, as shown in the figure below:



9. Click button Next: Review
10. Your review screen should look like figure below, except for User name (of course):



11. Click button Create user
12. You will see completion screen, as shown in figure below:



Download Keys for Account

13. Make sure to immediately download .csv with your new user account's access keys by clicking the Download .csv button.
14. Bookmark the URL highlighted above. This should become your login for AWS console going forward. Use your new User Name and Password as credentials.

Install CLI

15. From a terminal shell ($ prompt) do something similar to this:
    a. `$ cd ~/Downloads`
    b. `$ curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o "awscli-bundle.zip"`
    c. `$ unzip awscli-bundle.zip`
    d. `$ sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws`
    e. Answer affirmatively to the install program queries.

Configure CLI

16. From a terminal shell ($ prompt):
    a. `$ cd`
    b. Assuming your credentials.csv file downloaded in step 13 is in ~/Downloads:
       `$ cat ~/Downloads/credentials.csv`
    c. Substituting the keys in your credentials.csv file:
       $ aws configure
       i. AWS Access Key ID [None]: <your-access-key-id>
       ii. AWS Secret Access Key [None]: <your-secret-access-key>

iii. Default region name [None]: us-east-1
iv. Default output format [None]: text

d. The above step should create two secure files in your ~/.aws directory.

```
$ ls -al ~/.aws
drwx------  2 joe joe 4096 Nov  1 21:49 .
drwxr-xr-x 34 joe joe 4096 Jan 20 19:59 ..
-rw-------  1 joe joe   43 Nov  1 21:48 config
-rw-------  1 joe joe  116 Nov  1 21:49 credentials
```

e. Check that they contain the correct information from configure step above:

```
$ cat ~/.aws/credentials
$ cat ~/.aws/config
```

Execute CLI Commands

17. CLI S3 command to check that configuration works:
    $ aws s3 ls s3://<your-bucket-name>
18. The web services each provide extensive CLI commands. These are a few useful ones:
    a. Make bucket:
    ```
    $ aws s3 mb s3://my-bucket
    ```
    b. Upload file:
    ```
    $ aws s3 cp my-jar.jar s3://my-bucket/
    ```
    c. Delete folder:
    ```
    $ aws s3 rm s3://my-bucket/output/ --recursive
    ```
    d. Create cluster.
    Note that you should create a cluster to execute your MapReduce program via the AWS EMR web interface first. This will give you experience with this UI and allow you to easily gather configuration information (e.g. subnet-id) needed for the CLI example. This create cluster command is very sensitive to whitespace. Do not experimentally add any spaces until it works as documented below. Make sure to customize the red configuration parameters before executing:

```
$ aws emr create-cluster \
    --name "WordCount Cluster" \
    --release-label emr-5.2.1 \
    --instance-groups
'[{"InstanceCount":2,"InstanceGroupType":"CORE","InstanceType":"m4.large"},{"InstanceC
ount":1,"InstanceGroupType":"MASTER","InstanceType":"m4.large"}]' \
    --applications Name=Hadoop \
    --steps
'[{"Args":["your.package.MainClass","s3://my-bucket/input","s3://my-bucket/output"],"T
ype":"CUSTOM_JAR","Jar":"s3://my-bucket/my-jar.jar","ActionOnFailure":"TERMINATE_CLUST
ER","Name":"Custom JAR"}]' \
--log-uri s3://my-bucket/log \
--service-role EMR_DefaultRole \
--ec2-attributes InstanceProfile=EMR_EC2_DefaultRole,SubnetId=your-subnet-id \
--region us-east-1 \
```

```
--enable-debugging \
--auto-terminate
```

e. Download output, following cluster termination:

```
$ mkdir output
$ aws s3 sync s3://my-bucket/output output
```

AWS Java SDK

“The AWS SDK for Java provides a Java API for AWS infrastructure services. Using the SDK, you can build applications on top of Amazon S3, Amazon EC2, Amazon DynamoDB, and more.” There are SDKs for many other languages and even different platforms (e.g., Android). AWS access greatly extends the capabilities of applications. This section will use the SDK to access EC2, S3 and EMR. As always, start by going over the corresponding documentation on AWS.

Download the Java AWS SDK, or, preferably, use Maven or similar dependency-management tools to manage this and all of your dependencies:

```
<dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk</artifactId>
        <version>1.11.31</version>
</dependency>
```

Note: Check Maven and use latest version!

Security

As with CLI, it is very important to get your credentials properly configured. There are many options for this. If you configured your Access Key ID and Secret Access Key as described in the security section above, the SDK should seamlessly pick it up your credentials in your .aws directory. If not, check that you created an IAM role and downloaded id/key.

Example - Create EC2 Instance

This will create a new EC2 instance based on EC2 public image ami-60b6c60a and micro instance within the free tier. With security properly configured, the default credential search should work. Note that you must supply your key and a configured security group. A security group limits the IP address that can connect to the EC2 instance.

```
-----
AWSCredentials credentials = new ProfileCredentialsProvider().getCredentials();
AmazonEC2Client amazonEC2Client = new AmazonEC2Client(credentials);
amazonEC2Client.setEndpoint("ec2.us-east-1.amazonaws.com");
RunInstancesRequest runInstancesRequest = new RunInstancesRequest();
runInstancesRequest.withImageId("ami-60b6c60a")
        .withInstanceType("t2.micro")
```

```
        .withMinCount(1)
        .withMaxCount(1)
        .withKeyName("YourKey")
        .withSecurityGroups("YourSG");
RunInstancesResult runInstancesResult = amazonEC2Client.runInstances(runInstancesRequest);
System.out.println("Id: " + runInstancesResult.getReservation().getInstances().get(0).getInstanceId());
```

Example - Stop EC2 Instance

```
// Same credentials & client as above.
StopInstancesRequest stopInstancesRequest = new StopInstancesRequest(Arrays.asList(new String[]{instanceId}));
StopInstancesResult stopInstancesResult = amazonEC2Client.stopInstances(stopInstancesRequest);
```

Example - Start Existing EC2 Instance

```
StartInstancesRequest startInstancesRequest = new StartInstancesRequest()
        .withInstanceIds(instanceId);
StartInstancesResult startInstancesResult = amazonEC2Client.startInstances(startInstancesRequest);
```

Example - S3 Tasks

```
// Same credentials as above.
AmazonS3Client s3 = new AmazonS3Client(credentials);
Region usEast2 = Region.getRegion(Regions.US_EAST_1);
s3.setRegion(usEast2);
// Create bucket.
s3.createBucket(bucketName);
// List buckets.
for (Bucket bucket : s3.listBuckets()) {
        System.out.println(bucket.getName());
}
// Upload file to bucket.
s3.putObject(new PutObjectRequest(bucketName, key, aFile));
// List object in bucket.
ObjectListing objectListing = s3.listObjects(new ListObjectsRequest()
        .withBucketName(bucketName)
        .withPrefix("My"));
for (S3ObjectSummary objectSummary : objectListing.getObjectSummaries()) {
        System.out.println(objectSummary.getKey() + "  " + "(size = " + objectSummary.getSize() + ")");
}
```

```
// Download and read an object (text file).
S3Object object = s3.getObject(new GetObjectRequest(bucketName, key));
BufferedReader reader = new BufferedReader(new InputStreamReader(object.getObjectContent()));
String line;
while ((line = reader.readLine()) != null) {
        System.out.println(line);
}
// Delete object.
s3.deleteObject(bucketName, key);
// Delete bucket.
s3.deleteBucket(bucketName);
```

AWS SDK Managing EMR Cluster and Steps

This API allows access to all of the EMR functionality. The following shows how to create an EMR cluster:

```
AWSCredentials credentials = new ProfileCredentialsProvider().getCredentials();
AmazonElasticMapReduce client = new AmazonElasticMapReduceClient(credentials);
HadoopJarStepConfig hadoopConfig = new HadoopJarStepConfig()
        .withJar("s3://bucket/YourJar.jar")
        .withMainClass("YourMainClass")
        .withArgs("s3://bucket/input", "s3://bucket/output");
StepConfig customStep = new StepConfig(“Step Name", hadoopConfig);
RunJobFlowRequest request = new RunJobFlowRequest()
        .withName("Cluster Name")
        .withReleaseLabel("emr-4.3.0")
        .withSteps(customStep, customStep2)
        .withLogUri("s3://bucket/logs")
        .withServiceRole("EMR_DefaultRole")
        .withJobFlowRole("EMR_EC2_DefaultRole")
        .withInstances(new JobFlowInstancesConfig().withKeepJobFlowAliveWhenNoSteps(true)
        .withEc2KeyName("YourKey")
        .withInstanceCount(3)
        .withKeepJobFlowAliveWhenNoSteps(true)
        .withMasterInstanceType("m3.xlarge")
        .withSlaveInstanceType("m3.xlarge"));
RunJobFlowResult result = client.runJobFlow(request);
System.out.println("jobFlowId: " + result.getJobFlowId());
// Add step to runniing cluster.
HadoopJarStepConfig hadoopConfig2 = new HadoopJarStepConfig()
        .withJar("s3://bucket/YourJar.jar")
        .withMainClass("YourMainClass")
        .withArgs("s3://bucket/input", "s3://bucket/output");
StepConfig customStep2 = new StepConfig("Step Name", hadoopConfig2);
AddJobFlowStepsRequest request2 = new AddJobFlowStepsRequest()
```

```
        .withJobFlowId(jobFlowId)
        .withSteps(customStep3);
AddJobFlowStepsResult result2 = client.addJobFlowSteps(request);
```