README.md

To create an EC2 instance programatically

The AWS SDK(Software Development Kit) provides an API for Amazon Web Services. Using the SDK, you can easily build applications that work with AWS services.

AWS SDK for Java

1. Create an AWS account

- Create a user and get an access key
- Store it in `~/.aws/credentials`

```
[default]
aws_access_key_id = [ID]
aws_secret_access_key = [KEY]
```

- Set up `~/.aws/config`

```
[default]
region = us-east-1
```

2. Set up a suitable Java Development Environment and set the environment path

3. Install Maven

- Apache Maven is the most popular build and dependency resolution tool for Java like Npm and Pip
- To test the Maven installation `mvn -v`

4. Build a Java project with Maven

```
$ mvn archetype:generate
        -DgroupId=[package-name]
        -DartifactId=[project-name]
        -DarchetypeArtifactId=maven-archetype-quickstart
        -DinteractiveMode=false
```

5. Define a Maven build in *pom.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
...
...
...
<properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
</properties>
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk-bom</artifactId>
      <version>1.11.327</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

```
...
...
  <build>
          <plugins>
              <plugin>
                  <groupId>org.apache.maven.plugins</groupId>
                  <artifactId>maven-shade-plugin</artifactId>
                  <version>3.2.4</version>
                  <executions>
                      <execution>
                          <phase>package</phase>
                          <goals>
                              <goal>shade</goal>
                          </goals>
                          <configuration>
                              <transformers>
                                  <transformer
                                      implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                                      <mainClass>[package-name].[class-name]</mainClass>
                                  </transformer>
                              </transformers>
                          </configuration>
                      </execution>
                  </executions>
              </plugin>
          </plugins>
  </build>
</project>
```

6. Declare dependencies in *pom.xml* - ex) EC2 module

- Example Code

7. Write code

- Create a security group

    - Optinially set up ingress rules
- Create a key pair
- Create an instance with the security group and the key pair attached to it

8. Example code

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.CreateSecurityGroupRequest;
import com.amazonaws.services.ec2.model.CreateSecurityGroupResult;
import com.amazonaws.services.ec2.model.AuthorizeSecurityGroupIngressRequest;
import com.amazonaws.services.ec2.model.AuthorizeSecurityGroupIngressResult;
import com.amazonaws.services.ec2.model.IpPermission;
import com.amazonaws.services.ec2.model.IpRange;
import com.amazonaws.services.ec2.model.CreateKeyPairRequest;
import com.amazonaws.services.ec2.model.CreateKeyPairResult;
import com.amazonaws.services.ec2.model.InstanceType;
import com.amazonaws.services.ec2.model.RunInstancesRequest;
import com.amazonaws.services.ec2.model.RunInstancesResult;
import com.amazonaws.services.ec2.model.Tag;
import com.amazonaws.services.ec2.model.CreateTagsRequest;
import com.amazonaws.services.ec2.model.CreateTagsResult;
import java.util.List;

public static void main( String[] args ) {
        String sgName = "securityGroupForDemo";
        String sgDesc = "This is a security group for demo";
        String keyName = "COMS-6998-demo-key";
        String instanceName = "COMS-6998-demo-instance";
        String amiId = "ami-06b263d6ceff0b3dd"; // Ubuntu 18.04 LTS
        int minInstance = 1;
        int maxInstance = 1;

        createSecurityGroup(sgName, sgDesc);
        createKeyPair(keyName);
        createInstance(instanceName, amiId, sgName, keyName, minInstance, maxInstance);
}

public static void createSecurityGroup(String groupName, String desc) {
        final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();
        CreateSecurityGroupRequest createRequest = new CreateSecurityGroupRequest()
                                                .withGroupName(groupName)
                                                .withDescription(desc);

        CreateSecurityGroupResult createResponse = ec2.createSecurityGroup(createRequest);

 }

 public static void createKeyPair(String keyName) {
        final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();
        CreateKeyPairRequest request = new CreateKeyPairRequest().withKeyName(keyName);
        CreateKeyPairResult response = ec2.createKeyPair(request);
 }

 public static void createInstance(String name, String amiId, String sgName, String keyName, int min, int max) {
        final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();
        RunInstancesRequest runRequest = new RunInstancesRequest()
                                .withImageId(amiId)
                                .withInstanceType(InstanceType.T1Micro)
                                .withMaxCount(min)
                                .withMinCount(max)
                                .withKeyName(keyName)
                                .withSecurityGroups(sgName);

        RunInstancesResult runResponse = ec2.runInstances(runRequest);

        String reservationId = runResponse.getReservation().getInstances().get(0).getInstanceId();

        Tag tag = new Tag()
            .withKey("Name")
            .withValue(name);
        CreateTagsRequest tagRequest = new CreateTagsRequest()
            .withResources(reservationId)
            .withTags(tag);

        CreateTagsResult tagResponse = ec2.createTags(tagRequest);

        System.out.printf("EC2 instance %s started based on AMI %s\n", reservationId, amiId);
}
```

AWS SDK for Python

1. Create an AWS account

- Create a user and get an access key
- Store it in `~/.aws/credentials`

```
[default]
aws_access_key_id = [ID]
aws_secret_access_key = [KEY]
```

- Set up `~/.aws/config`

```
[default]
region = us-east-1
```

2. Install boto3

```
$ pip install boto3
```

3. Write code

- Create a security group
  - Optinially set up ingress rules
- Create a key pair
- Create an instance with the security group and the key pair attached to it

4. Example code

```
import boto3

def createSG(sgName,sgDesc):
    ec2 = boto3.client('ec2')
    res = ec2.create_security_group (
        GroupName = sgName,
        Description = sgDesc
    )

def createKeyPair(name):
    ec2 = boto3.resource('ec2')
    res = ec2.create_key_pair(KeyName = name)

def createEC2(amiId, keyName, sgName, instType = 't1.micro', minInst = 1, maxInst = 1):
    ec2 = boto3.resource('ec2')
    instances = ec2.create_instances (
        ImageId = amiId,
        MinCount = minInst,
        MaxCount = maxInst,
        InstanceType = instType,
        KeyName = keyName,
        SecurityGroups=[sgName]
    )

if __name__ == '__main__':
    sgName = 'securityGroupForDemo'
    sgDesc = 'This is a security group for demo'
    keyName = 'COMS-6998-demo-key'
    amiId = 'ami-06b263d6ceff0b3dd'

    createSG(sgName, sgDesc)
    createKeyPair(keyName)
    createEC2(amiId, keyName, sgName)
    print('Done')
```

Setting Up Dev Environment (Step-by-Step)

This guide is based on the previous guide written by former TA Hyun, and tries to go more step-by-step. You can check out the original by scrolling up to the top.

It'll follow the same steps, but includes some more helpful notes for each.

Java

1. **Create an AWS account**
   - After you create an IAM user, you can configure the AWS CLI by doing

   ```
   aws configure
   ```

   and going through the process. After that, you should be ready for step 2.

2. **Set up a suitable Java Development Environment and set the environment path**

   a. If you already have Java on your machine, skip to 2b.

   First, download the Java macOS Installer here:

   https://www.oracle.com/java/technologies/javase-jdk15-downloads.html

   Go through the installer, and once you're done, confirm that you have the latest Java version (as of writing Java 15)

   ```
   java -version
   ```

   b. Set your JAVA_HOME environment variable

   Then, you need to set your JAVA_HOME environment variable. Take a look at this:

   https://mkyong.com/java/how-to-set-java_home-environment-variable-on-mac-os-x/

   Once you've done that, when you try

   ```
   echo $JAVA_HOME
   ```

You should get an output like:

/Library/Java/JavaVirtualMachines/jdk-15.0.2.jdk/Contents/Home

3. **Install Maven**

   a. Download the .zip file here:

      https://maven.apache.org/download.cgi

      (Choose "Link" for "Binary zip archive")

   b. Follow the installation instructions here:

      https://maven.apache.org/install.html

      You'll notice that we already set the JAVA_HOME env variable as the instructions require. Yay!

      But we also need to add the 'bin' folder with the 'mvn' command to the $PATH env variable. To do this, I moved the entire `apache-maven-3.6.3` folder to my root user directory (at `~/`) and added the following to my `~/.bash_profile` if using bash, or `~/.zshrc` if using zsh:

```
export PATH="/Users/water/apache-maven-3.6.3/bin:$PATH"
```

4. **Build a Java project with Maven**

- [package-name] and [project-name] (specified in Hyun's original document), in this case "demo_package" and "demo_project", can be anything you want.

```
mvn archetype:generate -DgroupId=demo_package -DartifactId=demo_project -DarchetypeArtifactId=maven-archetyp
e-quickstart -DinteractiveMode=false
```

5. **Define a Maven build in pom.xml**

Here's an example pom.xml that lets you use ec2:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <properties>
    <maven.compiler.source>
      1.8
    </maven.compiler.source>
    <maven.compiler.target>
      1.8
    </maven.compiler.target>
  </properties>
  <groupId>demo_package</groupId>
  <artifactId>demo_project</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>demo_project</name>
  <url>http://maven.apache.org</url>
```

```xml
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk-ec2</artifactId>
    </dependency>
  </dependencies>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-bom</artifactId>
        <version>1.11.327</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.2.4</version>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
                <goal>shade</goal>
            </goals>
            <configuration>
              <transformers>
                <transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                  <mainClass>demo_package.demo_class</mainClass>
                </transformer>
              </transformers>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

6. **Declare dependencies in pom.xml - ex) EC2 module**

- As you can see the above pom.xml includes a dependency for ec2. Without this, you cannot import from the ec2 package.

7. **Write code**

- Only now can you `import com.amazonaws.services.ec2.AmazonEC2;` in your App.java file.
- Now, to actually build the project, do:

```
mvn package
```

- To run the project,

```
java -cp target/demo_project-1.0-SNAPSHOT.jar demo_package.App
```

- When you run it, you should see "Hello, World!" in your terminal!

8. **Example code**

   Provided in original document

Python

1. **Create an AWS account**
   - Same as for Java
2. **Install boto3**

   a. If you have already downloaded Python, skip to b. If you have Python 2 but want Python 3, you might want to keep reading.

   To download Python, go to website:

   https://www.python.org/downloads/

   and click the big yellow "Download Python 3.9.1" button. Go through Python installer, then make sure to double click the script that sets up Python on the shell (called Update Shell Profile.command).

   Confirm that you have 3.9 (latest as of the time of writing) by typing `python3`. You should now see 3.9

   Now, let make it so that when you type python in your terminal, it runs python3, not python2. In your `~/.zshrc` (if using zsh) or `~/.bash_profile` (if using bash), add:

```
alias pip=pip3
alias python=python3

(You can also alias git commands like gs=git status)
```

   b. If you don't care about setting up a virtual environment, skip to c. By downloading the boto3 package in a virtual environment rather than the global environment of your machine, your project will be self-contained.

   First, download virtualenv, a program that lets you create virtual environments:

`pip install virtualenv`

Now, in your project directory, setup your virtualenv like so:

`virtualenv NAME_OF_VENV`

Then, activate the virtual environment like this:

`source NAME_OF_VENV/bin/activate`

c. After you've activated your virtual environment, download the boto3 package:

`pip install boto3`

Great! Now you have boto3 installed just for your virtualenv, not your whole machine.

later, you can deactivate your virtualenv by doing:

`deactivate`

or just closing that terminal window.

3. **Write code**

   You can now import boto3 in your python files while your virtual environment is active.

4. **Example code**

   Provided in original document