DevOps: some tips for the first assignment

This document is designed to give you most of what you need to adapt parts of the Boto3 example programs provided in Week 3 and combine with the provided check_webserver.py script to complete the Assignment 1 core specification.

Creating instances

- **Credentials.** Ensure you have set up your credentials correctly. Instructions are given in the *Getting started with Boto3* lab. You can find your credentials file to troubleshoot any issues in a directory called `.aws` off your home directory.
- **Documentation.** `ec2 = boto3.resource('ec2')` assigns an EC2 *ServiceResource* object to the variable `ec2`. To see what you can do with this object, the following link will take you directly to the documentation: http://boto3.readthedocs.io/en/latest/reference/services/ec2.html#service-resource
- The `create_instances()` method of ServiceResource can take any of a large number of optional parameters, some of which you will want to set to suitable values. For example you'll need to set a `KeyName` for SSH access. You can also set `UserData`, `SecurityGroupIDs` and/or `SecurityGroups` (see below).
- **Key name.** When specifying a `KeyName` in `create_instances()`, drop the `.pem` or `.ppk` extension.
- **User Data** can be used to configure instance start-up scripts. When setting the `UserData` parameter in `create_instances()`, you will probably want to use a multi-line string. Python allows you to specify a multi-line string by starting and ending it with three quotation marks – i.e. `"""` You can find out more about User Data here: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html
- **Security group.** When starting a new instance using `create_instances()`, the instance is launched into your VPC's default security group unless you tell it otherwise. You can do this using the `SecurityGroupIDs` or `SecurityGroups` parameter. Each of these parameters is a Python *list* type. To specify a single security group (by ID) which you have already set up, it will look like this:

```
instances = ec2.create_instances(
    ImageId='ami-0fad7378adf284ce0',
    ...
    SecurityGroupIds=['sg-2a07095e4e3fda0c2'],    # HTTP and SSH
    InstanceType='t2.micro')
```

- **Security group creation.** If you do not have a security group already set up, you can do so manually in advance, or you can set one up as part of your script. For this you can use the `create_security_group()` method of EC2 *ServiceResource*.
- **Security groups and VPCs.** Resources in AWS accounts can be segmented into Virtual Private Clouds (VPCs). Every AWS account has a default VPC and new instances launched using the SDK use this default VPC unless instructed otherwise. Note however that your RosettaHUB AWS account has another VPC called "RosettaHUB VPC". When configuring resources (such as security groups) at the management console that you are working in the default VPC. A security group in the RosettaHUB VPC will not be usable by instances in the default VPC.
- **Saving your budget.** When testing scripts that create instances, it is important to clean up after yourself – i.e. terminate such instances. You can do this using the web console or a script. Also make sure you specify micro instances at launch

Managing instances

- The `create_instances()` method will return a **list** of *E2.Instance* objects. This list will usually have one item unless you set MinCount and/or MaxCount to values other than 1. You can extract the individual instances using the usual list operations – e.g. `instances[0]`
- An *E2.Instance* object has several parameters and methods that you can access to manage the instance. See http://boto3.readthedocs.io/en/latest/reference/services/ec2.html#instance.
- **Reload**. Always call the reload() method on an instance before trying to access its properties – just a simple `instance.reload()` (where the object is named `instance`). This will ensure the object's properties are refreshed.
- **Tags**. To help keep track of your instances, use the `create_tags()` method of the instance object to add a *Name* tag to the instance; i.e.
  ```
  name_tag = {'Key': 'Name', 'Value': 'Demo instance'}
  instance.create_tags(Tags=[name_tag])
  ```
  (where the Instance object is named `instance`). Specifying the *Key* as *Name* ensures that the tag is visible in the default instances view on the AWS management console.
- You will need a while loop or a short sleep to wait until the instance state is 'running' before getting the public IP address or DNS name.
- You can get public IP address using the public_ip_address attribute of the Instance class; i.e. if the object is called instance, then it is `instance.public_ip_address` (you could use the public_dns_name attribute either)
- After creating the instance and getting the IP address, you will need to wait until the machine has booted up and started the SSH service before you can connect and issue commands to it. There are a number of ways to do this. For example, you could sleep for a fixed period of time, e.g. `time.sleep(60)`, you could write a loop to keep trying every few seconds, or (best) you could implement a waiter method. For example there if you have an EC2 instance object called `instance`, you could do this with:
  ```
  instance.wait_until_running()
  ```

SCP (Secure copy)

scp allows you to transfer files using SSH. The command to copy a file called `check_webserver.py` in the current working directory on the local machine to an EC2 instance is:

```
scp -i mykeyfile.pem check_webserver.py ec2-user@52.42.23.2:.
```

(notice the colon following by dot at the end, after the IP address or DNS name).

SSH remote command execution

The general syntax for SSH remote command execution on an EC2 instance is:

```
ssh -i mykeyfile.pem ec2-user@ip_address 'enter command here'
```

You'll need the following additional option for the first SSH connection to a new instance:

```
-o StrictHostKeyChecking=no
```

(this is required to suppress the new host key confirmation (yes/no) prompt)

Putting the above together, your first ssh command will look something like this:

```
cmd1 = "ssh -o StrictHostKeyChecking=no -i  mykeyfile.pem
ec2-user@" + ip_address + " 'sudo pwd'"
```

(replace `'sudo pwd'` with a more useful command).

Before executing a remote command from within a Python script, it's a good idea to print the command string to the console (to help with debugging). You should also print the status and output values for the same reason.

You may have noticed that the "yum" package manager on Amazon Linux prompts the user for (yes/no) confirmation before installing a package. This can be suppressed using the yum `-y` option. For example, to remotely start apache on an EC2 instance you might have the following:

```
ssh -o StrictHostKeyChecking=no -i mykeyfile.pem ec2-user@52.42
.23.2 'sudo yum -y systemctl start httpd'
```

Script permissions

After copying a script to your newly-created EC2 instance, you'll need to make it executable. The command for this will be something like:

```
ssh -i mykeyfile.pem ec2-user@52.42.23.2 'chmod 700 check_webserver.py'
```

Use of echo command

The `echo` command can be used to redirect output to a file. For example you can append the text '*hello*' to a file test.txt using

```
            echo 'hello' >> test.txt
```

If you want to overwrite the contents of the file then just use

```
            echo 'hello' > test.txt
```

You can use echo to write multiple lines to a file e.g.

```
 echo '<html>
   <p>To view the uploaded S3 image click</p>
   <a href="https://s3-eu-west-1.amazonaws.com/bucketname/filename">HERE</a>
   </html>' > index.htm
```

A more elegant solution might be to open a file for write as described here: https://developers.google.com/edu/python/dict-files

For your assignment you will of course need to replace the url above with your S3 bucket url.