**How To Create Your Personal Data Science Computing Environment In AWS**

Running a training algorithm is such a time-consuming task when you build a machine learning application. If you are developing it with your computer, you cannot do anything else for a long period of time (hours and hours). Especially, we tend to run it in parallel using all the CPU cores. During training, CPU will be peaking at 100%.

So, here is what we can do. We create our personal data science computing environments in AWS and do training there. We can simply execute the script in the cloud as long as it needs to be. While it is running, we can still user our computer. Running it in the server has also lesser chance of interruptions. You can even start the script before going to bed and it will be finished by the time you come back from work on the next day. You can also set up your own database to get data and write the result back to a table. Sounds good, doesn’t it?

[Amazone Web Services (AWS)](https://aws.amazon.com/) offers easy to use cloud computing services. The computing and database resources are charged by hour. You can quickly launch the resources and terminate them once you finish your task. When you sign up, you have access to heaps of free services for the first 12 months (including Linux and database services) as part of [free tier](https://aws.amazon.com/free/).

Before going into this post, you may need to learn some basic about AWS. There are heaps of AWS courses out there. I recommend [AWS Cloud Practitioner Essentials](https://www.aws.training/learningobject/curriculum?id=16357), which is the official training material from AWS. It will give you the enough foundation knowledge to get it started.

**Architecture**

We are going to using Linux [EC2](https://aws.amazon.com/ec2/) instance to run the program and Postgres [RDS](https://aws.amazon.com/rds/) for Database. We install [Anaconda](https://anaconda.org/anaconda/python) in EC2 instance. EC2 can read and write to Postgres.

We are going to have two subnets across two availability zones. Our local machine needs to have access to both EC2 and Postgres RDS instances.

**Steps**

This is going to be an epic!

1. **Create AWS account**

First of all, you need to [create an AWS account](https://aws.amazon.com/resources/create-account/) (if you don’t have it already). AWS offers a [free tier](https://aws.amazon.com/free/) service for 12 months so that you can experiment and gain some practical experience.

1. **Create Admin User**

The best practice is not to use root account credential (the user who created the account). Use [IAM](https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html) to create an admin group attached with the AdministratorAccess policy. Then, create a user attached to the group.

To understand IAM identities and how to create admin user, go to [**How To Create Admin User In AWS**](http://www.mydatahack.com/how-to-create-admin-user-in-aws/).

Log back into AWS management console with the admin user credential.

1. **Create and Configure VPC and Subnets**

According to the plan, you need to create and configure VPC and Subnets. Then, attach Internet Gateway to the VPC.

For detailed steps, refer to this blog entry: [**How To Create and Configure VPC and Subnets In AWS**](https://www.mydatahack.com/how-to-create-and-configure-vpc-and-subnets-in-aws/).

1. **Create Network ACLs for both subnets and Security Groups for EC2 and RDS instances**

Go to VPC Dashoard and create Network ACLs for subnets and Security Groups for instances according to the plan. Problems with connecting to resources are usually resolved by fixing NACLs or Security Group.

I have the detailed set up examples for this use case: [**How To Configure Network Access Control Lists (NACLs) and Security Groups in AWS**](https://www.mydatahack.com/how-to-configure-network-access-control-lists-nacls-and-security-groups-in-aws/).

1. **Launch Linux EC2 Instance In Subnet A**

You need to launch the instance with correct role in to the correct subnet. Create Elastic IP and attach it to the instance. Once you have the instance, make sure you can SSH to it. Here is the detailed steps: [**How To Launch an EC2 Instance From AMI in AWS**](https://www.mydatahack.com/how-to-launch-an-ec2-instance-from-ami-in-aws).

1. **Attach an EBC volume to EC2 Instance**

This step is optional, but fun. You can get 30GB of free EBC for the first 12 months. So, why not? Here is the detailed steps: [**How To Attach EBS Volume to EC2 Linux Instance In AWS**](https://www.mydatahack.com/how-to-attach-ebs-volume-to-ec2-linux-instance-in-aws).

1. **Launch Postgres RDS instance in Subnet B.**

When you launch an RDS instance, you cannot choose a subnet. Instead, you have to choose Availability Zone. According to our diagram, Subnet B sits in AZ2. So, you have to launch it into AZ2. Then, Subnet B becomes where the database sits. Using subnet group doesn’t really work for this use case.

Apart from that, launching RDS is very easy. Here is the detailed steps: [**How To Launch a RDS Instance In a Specific Subnet**](https://www.mydatahack.com/how-to-launch-a-rds-instance-in-a-specific-subnet).

1. **Install Anaconda to EC2 instance**

We are almost there. Let’s install Anaconda to EC2 instance. Linux has Python 2.7 pre-installed. However, upgrading their Python version is not a good idea. Linux has some dependency on Python and it may break it.

Instead, we create a special folder /anaconda/ and install it there. When we call python, we create a variable called $python3 with the path to Python 3 in Anaconda and use it.

You can obtain the installation path from Anaconda website here. The example url below will be quickly outdated.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

sudo mkdir anaconda

cd anaconda

sudo wget https://repo.continuum.io/archive/Anaconda3-5.0.1-Linux-x86\_64.sh

sudo bash Anaconda3-5.0.1-Linux-x86\_64.sh

[/cc]

Make sure to change the default installation path to /anaconda/anaconda3/. Then, export a variable python3 as below. Type $python3 to see if it works.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

export python3=/anaconda/anaconda3/bin/python

$python3

[/cc]

1. **Install psycopg2**

Make sure to use the pip path for the anaconda. If you just use pip, it will install it to OS’s Python.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

sudo /anaconda/anaconda3/bin/pip install --upgrade pip

sudo /anaconda/anaconda3/bin/pip install psycopg2

[/cc]

1. **Test to see if the EC2 can connect to the Postgres RDS instance with the script below.**

Test to see if the EC2 can connect to the Postgres RDS instance with the script below.

I usually copy and paste the script directly into vi editor. Make sure to run the script with $python3 to use the correct Python. You also need to chmod to excute the script.

[cc lang="python" tab\_size="4" lines="-1"]

import psycopg2

dbname='<database name>'

user='<Username>'

host='<End Point>'

password='<Password>'

conn = psycopg2.connect("dbname={} user={} host={} password={}".format(dbname, user, host, password))

print(conn)

cur = conn.cursor()

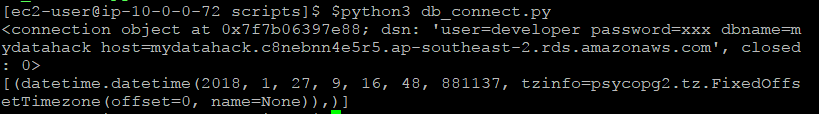
cur.execute('Select NOW();')

record = cur.fetchall()

print(record)

[/cc]

Yeah, the script runs. EC2 can connect to RDS.



Great. Now you have your own AWS environments to run a heavy training algorithm or do whatever you want. Once you finish computing, you can stop instances. While they are not running, you won’t be charged. When you need them again, you just restart them.

Epic!

<strong><u>Next Frontier</u></strong>

<strong>Infrastracture as Code</strong>

<strong>Infrastracture as Code</strong>

I think the most important philosophy of AWS (or any cloud computing platform) is Infrastracture as Code. The entire infrastructure and resources can be coded. Bringing up environments becomes running code. It sounds cool, right? This is the next frontier you should explore.

Resources like EC2 and RDS can be a piece of code. You can even install software and configure it while launching them. In this example, we launched EC2 and installed Anaconda with psycopg2. The whole step can be coded and repeated again and again by running the script (called Bootstrap). You can check out how it can be done here: <strong><a href="https://www.mydatahack.com/how-to-launch-ec2-with-bootstrap-in-aws" target="\_blank">How To Launch EC2 With Bootstrapping in AWS</a></strong>.

The same goes with RDS. You can launch it with a piece of code. Check it out how to do it here: <strong><a href="https://www.mydatahack.com/how-to-launch-postgres-rds-with-aws-command-line-interface-cli" target="\_blank">How To Launch Postgres RDS With AWS Command Line Interface (CLI)</a></strong>.

You can use a tool like CloudFormation to code up the entire infrastructure including VPC, subnet, routing and securities. Learning how to bootstrap EC2 or RDS is the first step toward coding the entire AWS environment. It is also very satisfying to create and terminate resources with a piece of code.

Good Times!