

Deploying a machine learning model in AWS Cloud (EC2 Instance) – Linux

Step 1: Develop a model (Regression or Classification)

The below model predicts “Salary” for the given “Years of Experience”. It’s a regression model. Hence the model is developed by using “Linear Regression”.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Read the input file
df = pd.read_csv("D:\Model Deployment - 01Feb\ML-Deployment-master\ML-Deployment-master\Regression_Deployment\SalaryData.csv")

# Split the dataset for training and testing
train_set, test_set = train_test_split(df, test_size=0.2, random_state=42)

df_copy = train_set.copy()

test_set_full = test_set.copy()
test_set = test_set.drop(["Salary"], axis=1)

train_labels = df_copy["Salary"]

train_set_full = train_set.copy()
train_set = train_set.drop(["Salary"], axis=1)

# Create a model object
lin_reg = LinearRegression()

# Train the model
lin_reg.fit(train_set, train_labels)

# Predict the test data
salary_pred = lin_reg.predict(test_set)

# Display predicted values
salary_pred
```

Step 2: Ensure that the model predicts as expected for the unseen data. I have inputted "10.0" years of experience as input and got the predicted value as "119559.74"

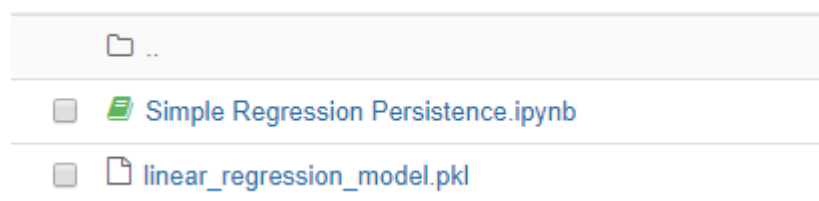
```
lin_reg.predict([[10.0]])
```

Step 3: In this step, we will create the pickle (serialization) file, which will be used for prediction in deployed environment.

```
# Import the required library for pickling
from sklearn.externals import joblib

# Create a pickle object for the model
joblib.dump(lin_reg, "linear_regression_model.pkl")
```

Ensure the the pickled object is created successfully in the project folder.



Step 4: Since the model will be invoked from a webpage, create a required web pages for business user usage. For this simple regression model, we will create 2 html files.

- First file, will take inputs from the user and pass on to the model
- Second file, will display the predicted value (salary, in our case) received from the model.

HTML code snippet:

Index.html

```
<html>
<body>
    <h3>Salary Prediction Form</h3>

<div>
    <form action="/result" method="POST">
        <label for="experience">Years of Experience : </label>
        <input type="text" id="experience" name="experience">
        <br>
        <br>
        <input type="submit" value="Submit">
    </form>
</div>
</body>
</html>
```

Result.html

```
<!doctype html>
<html>
  <body>
    <h1> Predicted Salary is : {{ prediction }}</h1>
  </body>
</html>
```

Note: Both the files should be placed in “templates” folder.

Step 5: Create a flask program to invoke the model

```
#importing libraries
from flask import Flask, render_template, request
from sklearn.externals import joblib

#creating instance of the class
app=Flask(__name__)

#to tell flask what url should trigger the function index()
@app.route('/')
def index():
    return render_template('index.html')

@app.route("/result", methods=['POST'])
def predict():
    print("I am here in predict function")
    print(request.method)
    if request.method == 'POST':
        try:
            data = request.form.to_dict()
            print(data)
            years_of_experience = float(data["experience"])
            print(years_of_experience)

            # load the model from disk
            lin_reg = joblib.load("./linear_regression_model.pkl")
            print(lin_reg.intercept_)

        except ValueError:
            return render_template("result.html", salary="Please enter a
number.")

        print("Prediction = ", lin_reg.predict([[years_of_experience]]))
        Prediction = lin_reg.predict([[years_of_experience]])

        print(type(Prediction))
        salary = Prediction.tolist()
        print(salary)

        return render_template("result.html", prediction=salary[0])

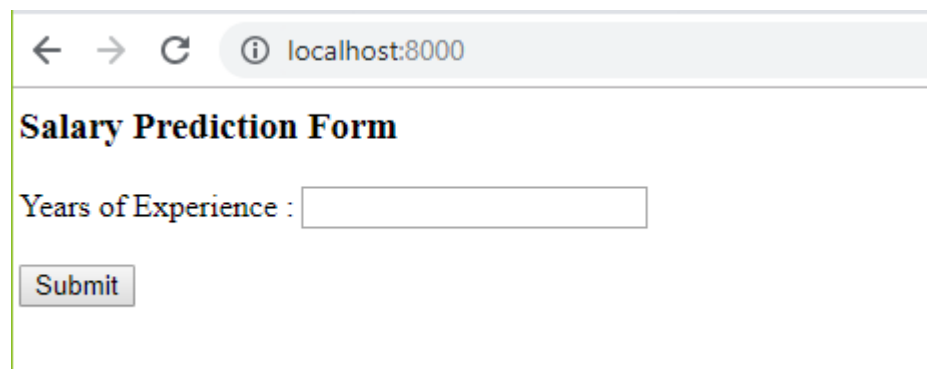
if __name__ == "__main__":
    app.run(port = 8000, debug=False)
```

Step 6: Run the above flask program. The code will create a web server (localhost or your computer) and deploy the model as a web service.

```
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off

* Running on http://127.0.0.1:8000/ (Press CTRL+C to quit)
```

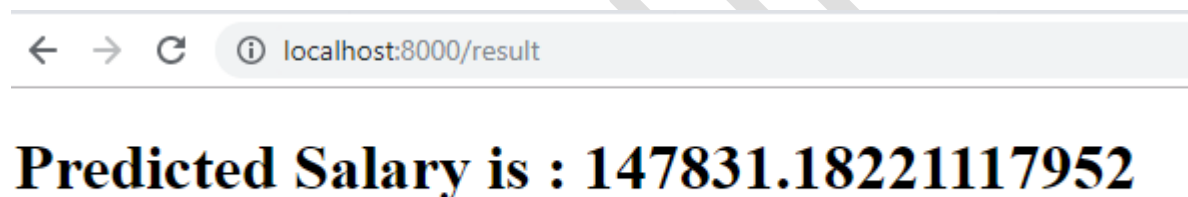
Go to the browser and open "localhost:8000" or <http://127.0.0.1:8000/>



Salary Prediction Form

Years of Experience :

Enter "Years of Experience" value and click "Submit"



Predicted Salary is : 147831.18221117952

Deploying the model in AWS Cloud

Step 1: Choose the AMI

Step 1: Choose an Amazon Machine Image (AMI)

[Cancel and Exit](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace, or you can select one of your own AMIs.


Quick Start

My AMIs


AWS Marketplace

Community AMIs

☒ Free tier only ⓘ

**Amazon Linux**
Free tier eligible

Amazon Linux 2 AMI (HVM), SSD Volume Type - ami-0d9462a653c34dab7 (64-bit x86) / ami-01ee9bf1eba051789 (64-bit Arm)
Amazon Linux 2 comes with five years support. It provides Linux kernel 4.14 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras.
Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

**Amazon Linux**
Free tier eligible

Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type - ami-05695932c5299858a
The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.
Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

1 to 18 of 18 AMIs

64-bit (x86)
64-bit (Arm)

64-bit (x86)

For this exercise, I have used “**Amazon Linux 2 AMI (HVM), SSD Volume Type** - ami-0d9462a653c34dab7 (64-bit x86) / ami-01ee9bf1eba051789 (64-bit Arm)”

Step 2: Create the instance with a security group

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group: ☒ Create a new security group
☐ Select an existing security group

Security group name:

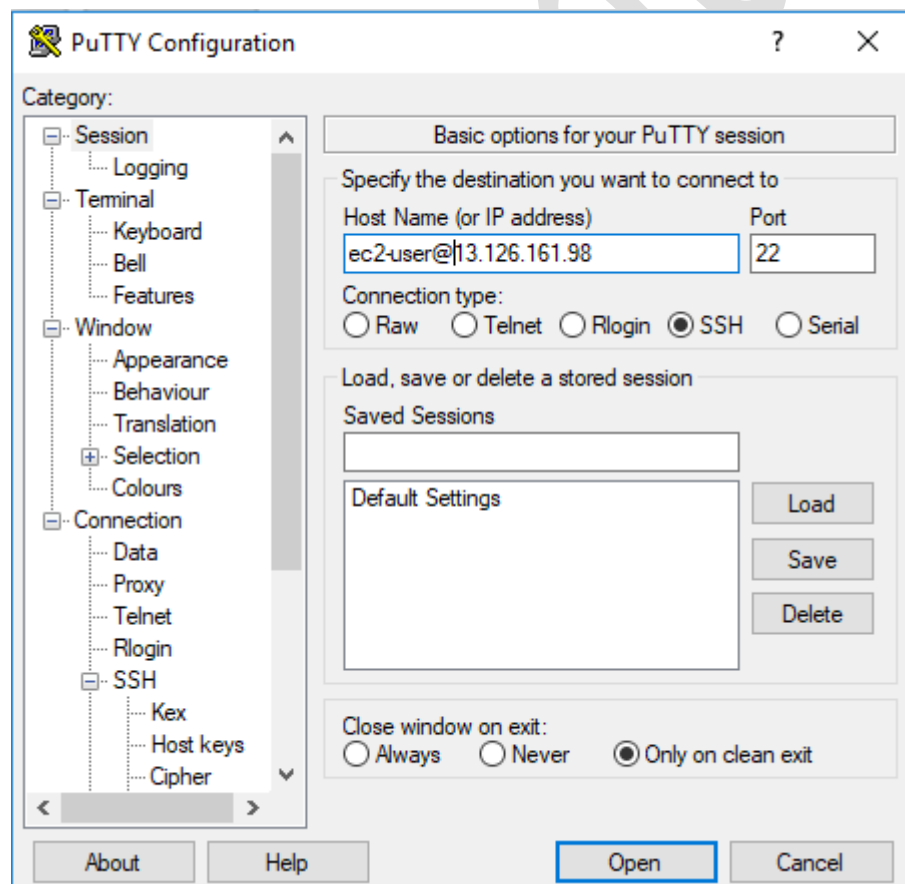
Description:

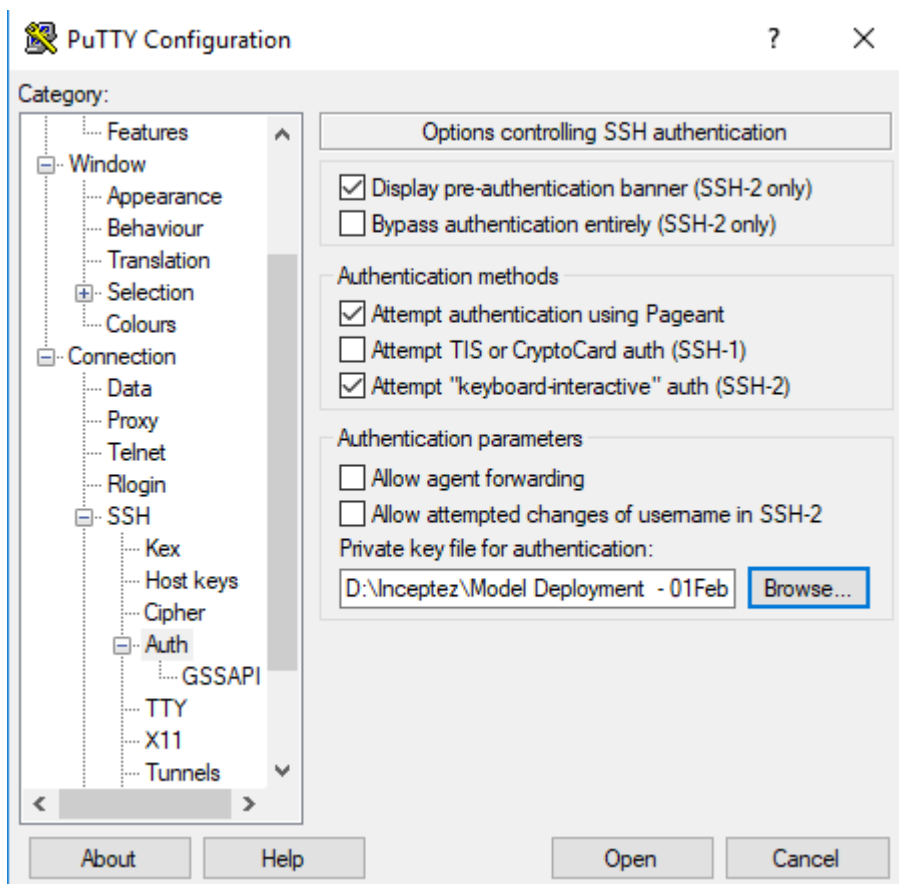
Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
Custom TCP F	TCP	8000	Custom 0.0.0.0/0, ::/0	e.g. SSH for Admin Desktop

Step 3: Ensure that the EC2 instance is up and running

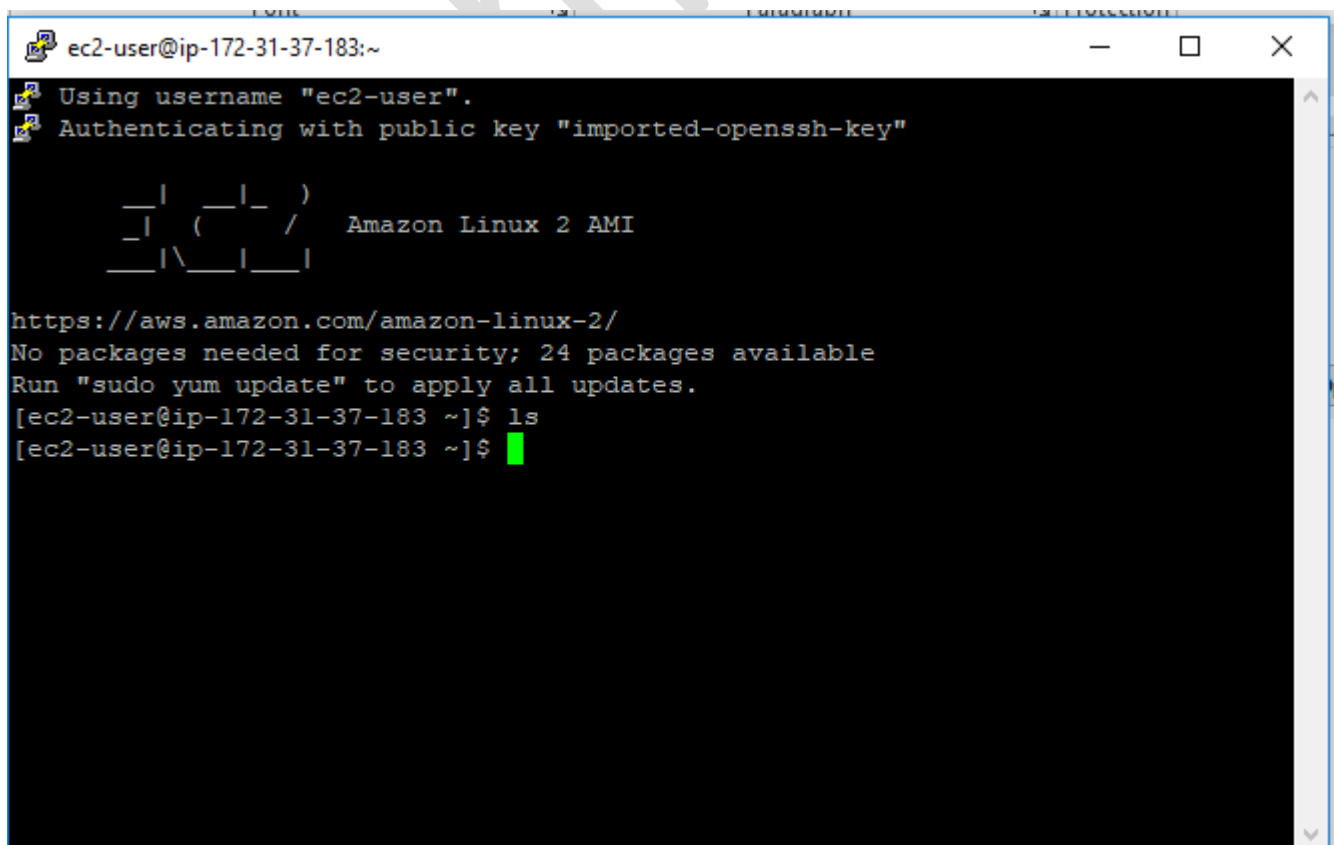
<input checked="" type="checkbox"/>	MachineLear...	i-0914cf15529fc3580	t2.micro	ap-south-1a	● running	✔ 2/2 checks ...	None	
-------------------------------------	----------------	---------------------	----------	-------------	--	---	------	--

Step 4: Since, it is a Linux instance, use PuTTY to connect to the instance. Before that, use PuTTY Gen to generate the “Private Key” by using the .pem file.





Connected to Linux Instance. You will get the below screen.



Note: The newly instantiated Linux machine won't have git, python and all the required packages. These have to be installed.

For installing git, use the below commands:

- `sudo yum upgrade`
- `sudo yum install git`

Then, download the deployable components from the github by using the below command:

- `git clone <github link>`

Install Python by using the below command in Linux:

- `sudo yum install python3.7`
- `python3.7 --version`

Install pip

- `curl -O https://bootstrap.pypa.io/get-pip.py`
- `python3.7 get-pip.py --user`

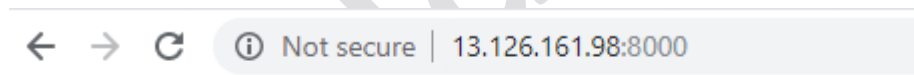
Install and starting the flask server

- `sudo yum install python-setuptools`
- `sudo easy_install pip`
- `sudo pip install flask`
- `pip3.7 install sklearn`

Edit the flask program to include the host as '0.0.0.0'

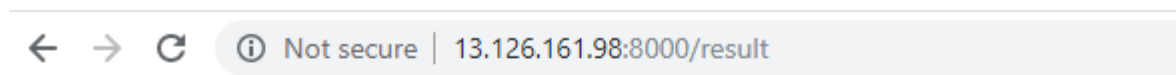
```
if __name__ == "__main__":  
    app.run(host='0.0.0.0', port = 8000, debug=True)
```

Now, invoke the model from anywhere in the world 😊



Salary Prediction Form

Years of Experience :



Predicted Salary is : 119559.73624208657