Run a simple node application

Installing method:

Sudo apt update

Sudo apt install nodejs

Sudo apt install npm.

Step 2:

Created hello-world file

Mv hello-world hello-world.js

Reactjs /nodejs

format \*.js

run we need package.json for that run npm init

npm install

node \*.js

Step 3:added code to hello-world

//Require module

const express = require('express');

// Express Initialize

const app = express();

const port = 8000;

app.listen(port,()=> {

console.log('listen port 8000');

})

//create api

app.get('/', (req,res)=>{

res.send('Hello World');

})

Search ipaddress in browser:8000

Graphical user interface, text, application, chat or text message

Description automatically generated

Run a simple java application

Installing process:

sudo apt update

sudo apt install default-jre

java -version

sudo apt install default-jdk

step 2:

create file:

touch helloworld.java

added code:

nano helloworld.java

**class helloworld         {**

**public static void main(String args[]){**

**System.out.println("Hello Java");**

**}**

**}**

javac helloworld.java

java helloworld

to run code :

java helloworld



**Run a simple python application**

**Setting phython:**

**sudo apt update**

1. **sudo apt -y upgrade**

**The -y flag will confirm that we are agreeing for all items to be installed, but depending on your version of Linux, you may need to confirm additional prompts as your system updates and upgrades.**

**Once the process is complete, we can check the version of Python 3 that is installed in the system by typing:**

**python3 -V**

**sudo apt install -y python3-pip**

**To manage software packages for Python, let’s install pip, a tool that will install and manage programming packages we may want to use in our development projects**

**Python packages can be installed by typing:**

1. **pip3 install package\_name**

**Copy**

**Here, package\_name can refer to any Python package or library, such as Django for web development or NumPy for scientific computing. So if you would like to install NumPy, you can do so with the command pip3 install numpy.**

**There are a few more packages and development tools to install to ensure that we have a robust setup for our programming environment:**

1. **sudo apt install -y build-essential libssl-dev libffi-dev python3-dev**

**Copy**

**Once Python is set up, and pip and other tools are installed, we can set up a virtual environment for our development projects.**

**Step 2 — Setting Up a Virtual Environment**

**Virtual environments enable you to have an isolated space on your server for Python projects, ensuring that each of your projects can have its own set of dependencies that won’t disrupt any of your other projects.**

**Setting up a programming environment provides greater control over Python projects and over how different versions of packages are handled. This is especially important when working with third-party packages.**

**You can set up as many Python programming environments as you would like. Each environment is basically a directory or folder on your server that has a few scripts in it to make it act as an environment.**

**While there are a few ways to achieve a programming environment in Python, we’ll be using the venv module here, which is part of the standard Python 3 library. Let’s install venv by typing:**

1. **sudo apt install -y python3-venv**

**Copy**

**With this installed, we are ready to create environments. Let’s either choose which directory we would like to put our Python programming environments in, or create a new directory with mkdir, as in:**

1. **mkdir environments**

**Copy**

**Then navigate to the directory where you’ll store your programming environments:**

1. **cd environments**

**Copy**

**Once you are in the directory where you would like the environments to live, you can create an environment by running the following command:**

1. **python3 -m venv my\_env**

**Copy**

**Essentially, pyvenv sets up a new directory that contains a few items which we can view with the ls command:**

1. **ls my\_env**

**Copy**

**Output**

**bin include lib lib64 pyvenv.cfg share**

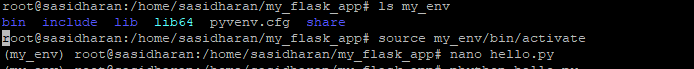
**Together, these files work to make sure that your projects are isolated from the broader context of your server, so that system files and project files don’t mix. This is good practice for version control and to ensure that each of your projects has access to the particular packages that it needs. Python Wheels, a built-package format for Python that can speed up your software production by reducing the number of times you need to compile, will be in the Ubuntu 20.04 share directory.**

**To use this environment, you need to activate it, which you can achieve by typing the following command that calls the activate script:**

1. **source my\_env/bin/activate**

**Your command** **Text

Description automatically generated prompt will now be prefixed with the name of your environment, in this case it is called my\_env. Depending on what version of Debian Linux you are running, your prefix may appear somewhat differently, but the name of your environment in parentheses should be the first thing you see on your line:**

****

**Copy**

**This prefix lets us know that the environment my\_env is currently active, meaning that when we create programs here they will use only this particular environment’s settings and packages.**

**After following these steps, your virtual environment is ready to use.**

**Step 3 — Creating a “Hello, World” Program**

**Now that we have our virtual environment set up, let’s create a traditional “Hello, World!” program. This will let us test our environment and provides us with the opportunity to become more familiar with Python if we aren’t already.**

**To do this, we’ll open up a command line text editor such as nano and create a new file:**

1. **nano hello.py**

**Copy**

**Once the text file opens up in the terminal window we’ll type out our program:**

**hello.py**

**print("Hello, World!")**

**Copy**

**Save the file and exit nano by pressing CTRL + X, Y, and then ENTER.**

**Once you exit out of the editor and return to your shell, you can run the program:**

1. **python hello.py**

**Copy**

**The hello.py program that you created should cause your terminal to produce the following output:**

****

**Run a simple dotnet core application**

**Installation:**

**Before you install .NET, run the following commands to add the Microsoft package signing key to your list of trusted keys and add the package repository.**

**wget https://packages.microsoft.com/config/ubuntu/21.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb**

**sudo dpkg -i packages-microsoft-prod.deb**

**rm packages-microsoft-prod.deb**

**The .NET SDK allows you to develop apps with .NET. If you install the .NET SDK, you don't need to install** **sudo apt-get update;**

**sudo apt-get update**

**sudo apt-get install -y apt-transport-https**

**sudo apt-get update**

**sudo apt-get install -y dotnet-sdk-6.0**

## Create Sample Application

Let’s create a sample application with dotnet core on your Ubuntu system. Create a new console application with the command:

dotnet new console -o Helloworld(helloworld is creating file)

dotnet new console -o HelloWorld

The Above command will create dotnet application on your system. This will create a directory named “helloworld” in under the current directory.

You can change to this directory and start working your application.

Cd helloworld

cd HelloWorld

execute below command to run this application.

dotnet run

A screenshot of a computer

Description automatically generated with medium confidence

dotnet run