**Standard Operating Procedure (SOP) for Building and Running C# .NET Applications in Ubuntu**

**Prerequisites**

1. Ubuntu 20.04 or later installed.
2. Access to the terminal with sudo privileges.
3. Basic understanding of terminal commands.

**Step 1: Update System Packages**

1. Open the terminal and run:
2. sudo apt update

sudo apt upgrade -y

**Step 2: Install .NET SDK**

1. **Add Microsoft Package Repository:**
2. wget https://packages.microsoft.com/config/ubuntu/22.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb

sudo dpkg -i packages-microsoft-prod.deb

1. **Install .NET SDK:**

Update the package list and install the .NET SDK:

sudo apt update

sudo apt install dotnet-sdk-8.0 -y

1. **Verify Installation:**

Check the installed .NET version:

dotnet --version

**Step 3: Create a New C# Application**

1. **Create a New Console Application:**

Run the following command to create a new console application:

dotnet new console -n MyFirstApp

This creates a folder named MyFirstApp with the application structure.

1. **Navigate to the Project Folder:**

cd MyFirstApp

1. **Edit the Program.cs File (Optional):**

Open Program.cs in a text editor to modify the code.

Example code:

using System;

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello, World!");

}

}

**Step 4: Build and Run the Application**

1. **Build the Application:**

dotnet build

Ensure there are no errors in the output.

1. **Run the Application:**

dotnet run

The output should display:

Hello, World!

**Step 5: Create a Web API Application (Optional)**

1. **Create a New Web API Application:**

dotnet new webapi -n MyWebAPI

1. **Navigate to the Project Folder:**

cd MyWebAPI

1. **Run the Web API Application:**

dotnet run

The application will be hosted on http://localhost:5000 and https://localhost:5001.

1. **Test the Default Endpoint:**

Use a browser or curl to test the default endpoint:

curl http://localhost:5000/api/weatherforecast

**Step 6: Install Visual Studio Code (Optional)**

1. **Add Microsoft Repository for VS Code:**
2. wget -qO- https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > microsoft.gpg
3. sudo install -o root -g root -m 644 microsoft.gpg /usr/share/keyrings/
4. echo "deb [arch=amd64 signed-by=/usr/share/keyrings/microsoft.gpg] https://packages.microsoft.com/repos/code stable main" | sudo tee /etc/apt/sources.list.d/vscode.list

sudo apt update

1. **Install VS Code:**

sudo apt install code -y

1. **Launch VS Code:**

code

**Step 7: Publish the Application (Optional)**

1. **Publish the Application for Deployment:**

dotnet publish -c Release -o ./publish

1. **The Output Files:**

The ./publish directory will contain the necessary files for deployment.

**Troubleshooting**

1. **Error: 'Unable to locate package code':**

Ensure the Microsoft repository for VS Code is added correctly and try again.

1. **Permission Denied:**

Use sudo for commands requiring elevated privileges.

1. **Unable to Run Web API:**

Ensure no other services are using the default ports (5000 or 5001). You can specify different ports by editing launchSettings.json or using environment variables.

**Conclusion**

You have successfully installed .NET, created a sample application, and optionally set up a Web API or Visual Studio Code. You can now develop, build, and run C# applications on Ubuntu.

## IN window dotnet run command is working but in ubuntu is'nt why?

The discrepancy between running dotnet run on Windows and Ubuntu could be due to several reasons. Here are the most common ones:

**File Path Differences**

* **Windows uses backslashes (**\**)** in paths, while **Linux uses forward slashes (**/**)**. Ensure your code is using the correct path format for the operating system.

**Case Sensitivity**

* **Linux is case-sensitive** when it comes to file names and paths, whereas Windows is not. Ensure that your file and directory names match exactly in your code.

**Dependencies**

* Make sure all necessary **.NET runtime and dependencies** are installed on Ubuntu.

**File Permissions**

* Ensure the **files and directories have the correct permissions** on Ubuntu. You might need to adjust permissions using chmod.

**Configuration Differences**

* Check if there are any **configuration settings** that differ between your Windows and Ubuntu environments (e.g., environment variables, connection strings).

**Environment Setup**

* Ensure that the **.NET SDK and runtime** are properly installed on Ubuntu.

By addressing these areas, you should be able to resolve the issue and successfully run your application on both Windows and Ubuntu.

## Setting Up a systemd Service for .NET Core Application

This guide will help you set up a systemd service to manage a .NET Core application on your Ubuntu server, enabling it to run on startup and ensuring it listens on all network interfaces (0.0.0.0) for external access.

**Steps to Configure and Start the Service**

**1. Build Your .NET Application**

Before creating the systemd service, ensure that your .NET Core application is built and ready for deployment.

Navigate to your project directory and build the application:

cd ~/SCERS\_RAP\_API

dotnet publish -c Release -o /home/ubuntu/SCERS\_RAP\_API/bin/Release/netcoreapp3.1/publish

Adjust netcoreapp3.1 with the version of .NET Core that you're using.

**2. Create the systemd Service File**

Create a new systemd service file to manage your application. You can do this by creating a .service file in the /etc/systemd/system/ directory.

sudo nano /etc/systemd/system/scers-rap-api.service

**3. Add the Configuration to the Service File**

Copy and paste the following configuration into the service file. It ensures the application will bind to 0.0.0.0:5034 to allow external access:

ini

[Unit]

Description=SCERS RAP API Service

After=network.target

[Service]

WorkingDirectory=/home/ubuntu/SCERS\_RAP\_API

ExecStart=/usr/bin/dotnet /home/ubuntu/SCERS\_RAP\_API/bin/Release/netcoreapp3.1/publish/SCERS\_RAP\_API.dll

Restart=always

Environment=ASPNETCORE\_URLS=http://0.0.0.0:5034

User=ubuntu

Group=ubuntu

SyslogIdentifier=scers-rap-api

[Install]

WantedBy=multi-user.target

**Breakdown of Configuration:**

* **WorkingDirectory**: Specifies the path to your application directory.
* **ExecStart**: The command used to start the .NET application (dotnet command with the path to the published .dll file).
* **Environment**: Sets the environment variable ASPNETCORE\_URLS to listen on 0.0.0.0:5034 for external access.
* **Restart**: Ensures that the application will automatically restart if it crashes.
* **User/Group**: Runs the service as the specified user and group (in this case, ubuntu).
* **SyslogIdentifier**: Sets the syslog identifier for logging.

**4. Reload systemd and Enable the Service**

Once you've saved the file, reload the systemd configuration to recognize the new service and enable it to start on boot.

sudo systemctl daemon-reload

sudo systemctl enable scers-rap-api.service

Then, start the service:

sudo systemctl start scers-rap-api.service

**5. Verify the Service is Running**

To verify that the service is running correctly, check its status:

sudo systemctl status scers-rap-api.service

This should show that the service is active and running.

**6. Verify the Service is Listening on the Correct Port**

To ensure that the application is listening on port 5034 for external access, run the following command:

sudo ss -tuln | grep 5034

You should see output similar to this:

ruby

tcp LISTEN 0 512 0.0.0.0:5034 0.0.0.0:\*

This indicates that the service is bound to 0.0.0.0:5034, allowing access from external machines.

**7. Test External Access**

Now that your application is running, you can test access to the API externally by using curl or your web browser. From a different machine, run:

curl http://54.175.99.194:5034/api/RAP

Replace 54.175.99.194 with your server’s public IP address.

**8. Adjust Firewall Rules (if necessary)**

Ensure that your firewall allows traffic on port 5034. To allow incoming traffic on this port, run:

sudo ufw allow 5034/tcp

sudo ufw reload

If you're using a different firewall, make sure to open port 5034 for incoming traffic.

**9. Check Logs for Debugging**

If the service is not starting or behaving unexpectedly, you can check the logs for errors using:

sudo journalctl -u scers-rap-api.service -f

This will show you the real-time logs of your service.

**Troubleshooting:**

* **Firewall Issues**: Ensure that no firewall is blocking port 5034. If needed, open it using ufw or your firewall manager.
* **Permission Issues**: Ensure that the user specified in the systemd service has sufficient permissions to run the application. You may need to adjust file permissions for the .dll and associated files.
* **Binding Issues**: If your application does not bind to 0.0.0.0:5034, verify that the Environment=ASPNETCORE\_URLS=http://0.0.0.0:5034 line is correctly set in the systemd configuration.

**Conclusion:**

By following the steps above, you have successfully set up a systemd service to run your .NET Core application and configured it to listen on all network interfaces (0.0.0.0) for external access. This allows you to manage the service easily, ensuring it runs on system startup and can be accessed from external clients.

Let me know if you need further assistance!