.NET CORE AZURE VM DEPLOY GUIDE

PETRO KOLOSOV

Abstract. Simple and easy way to deploy your .NET Core web application to the Azure Ubuntu-based virtual machine.

Contents

1. Virtual machine creation

Firstly, it is necessary to create a virtual machine (unexpectedly) where deployment to be hosted on. In this guide is considered free virtual machine of type `Standard B1ms (1 vcpu, 2 GiB memory)` with Ubuntu 20.04 operating system. Definitely it won't be considered step by step creation in this document, however required VM parameter are as follows:

- Size: `Standard B1ms (1 vcpu, 2 GiB memory)`
- OS: `Ubuntu Server 20.04 LTS - Gen2`
- Availability options: `No infrastructure required`
- Authentication type: `SSH public key`
- SSH public key source: `Use existing public key (create it before you created VM)`

Date: May 26, 2022.

2010 *Mathematics Subject Classification.* 26E70, 05A30.

Key words and phrases. Azure, DevOps, Virtual machine, Deploy, Nginx, SSH, CI/CD, Azure pipelines, Github actions .

- Public inbound ports: `HTTP(80), HTTPS(443), SSH(22)`
- OS disk type: `Standard SSD`
- Encryption type: `Default`
- Public IP: `Basic SKU, Static (be sure to create static IP)`
- Select inbound ports: `HTTP(80), HTTPS(443), SSH(22)`
- Boot diagnostics: `Disabled`

Chosen parameters of the virtual machine are collected in order to minimize vm's cost. If you are not sure, refer to the screenshots via the reference [Kol22c].

2. Connect to VM via SSH

In order to configure virtual machine manually (as this guide tends to describe), we have to connect to it via SSH using the specified RSA private and public key-pair. It is assumed that programmer uses `WSL2` under `Windows 10` in order to work with VM via the SSH. By default, SSH keys are stored under the path `c/Users/username/.ssh`. Assume that RSA key-pair is stored there and have the names `id_rsa` and `id_rsa.pub` for private and public keys respectively. In order to interact the VM via SSH it is necessary to copy RSA keypair to the WSL `username/.ssh` folder, we use the commands under WSL

- `cp /mnt/c/Users/pkolosov/.ssh/id_rsa /.ssh/`
- `cp /mnt/c/Users/pkolosov/.ssh/id_rsa.pub /.ssh/`

Then connection is available now using the command

- `ssh -i /.ssh/id_rsa razumovsky_r@MachineStaticIP`

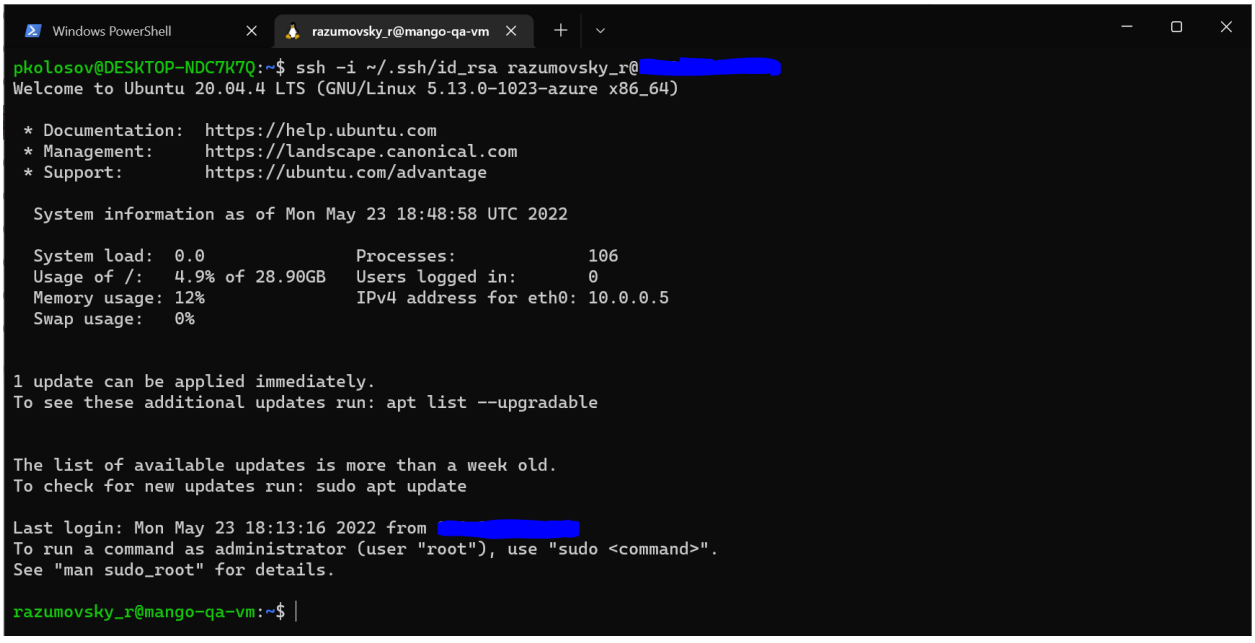

Figure 1. SSH connected successfully.

Therefore, the SSH connection between WSL2 under Windows 10 is established so that we are able to configure our virtual machine as per our needs.

3. Install .NET SDK and Runtime to the Ubuntu 20.04

Next, we should install the .NET SDK (unexpectedly again) in order to run our application. Proceeding, we refer to the Microsoft documentation article named `Install the .NET SDK or the .NET Runtime on Ubuntu` [Cor22], precisely the version is 20.04. As per documentation, consider the following commands to install .NET 6.0 SDK to your Ubuntu VM

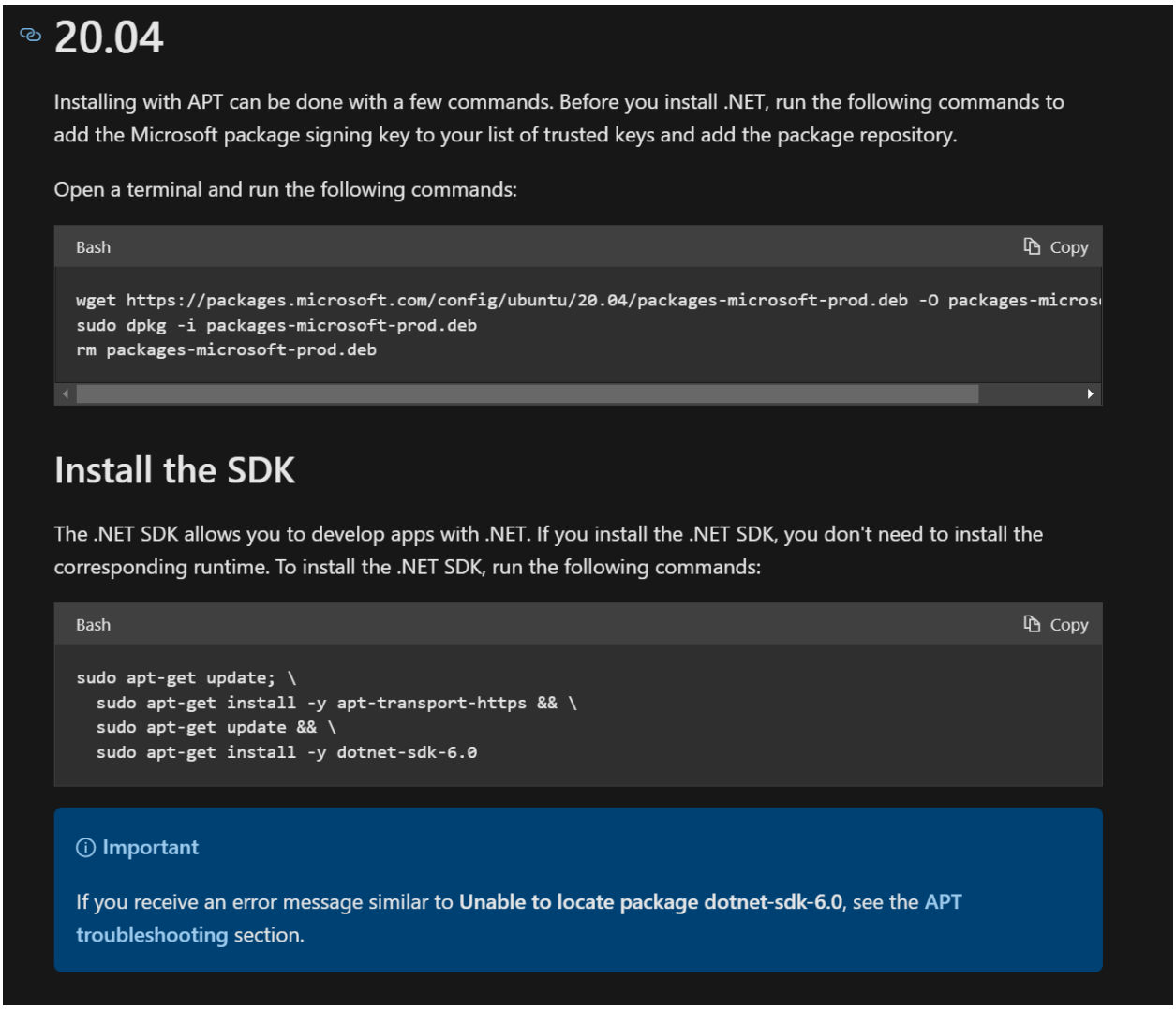

Figure 2. Ubuntu 20.04 install .NET 6.0 SDK MSDN.

Prepare your virtual machine applying the commands

- `wget https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.de`
  `-O packages-microsoft-prod.deb`

- `sudo dpkg -i packages-microsoft-prod.deb`
- `rm packages-microsoft-prod.deb`

The terminal output is as follows

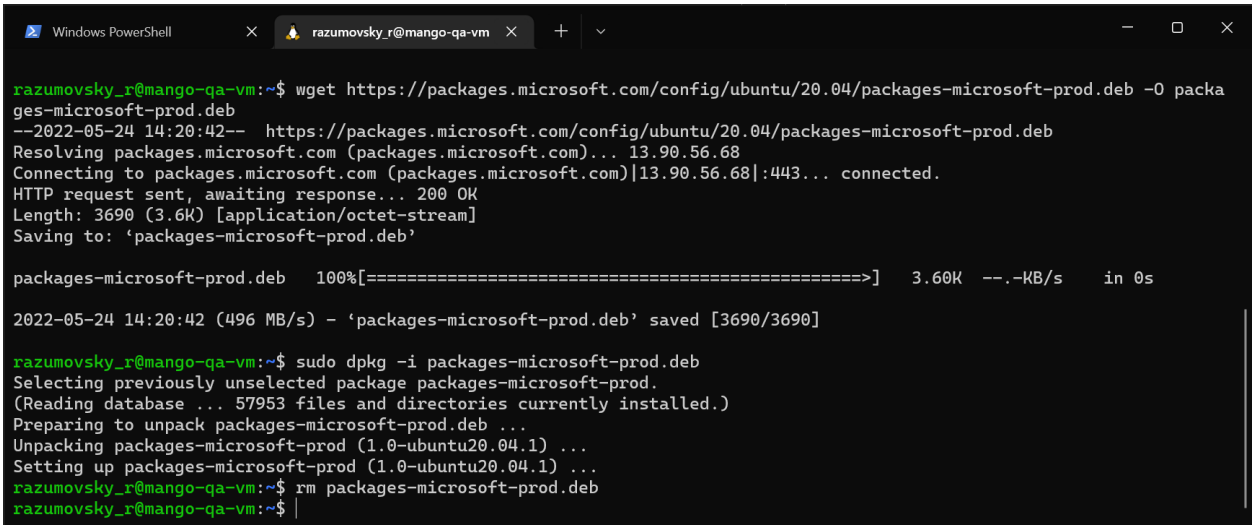

Figure 3. Virtual machine preparation..

Apply the following commands in order to install the SDK

- `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`

The terminal output after .NET 6.0 SDK installation is as follows

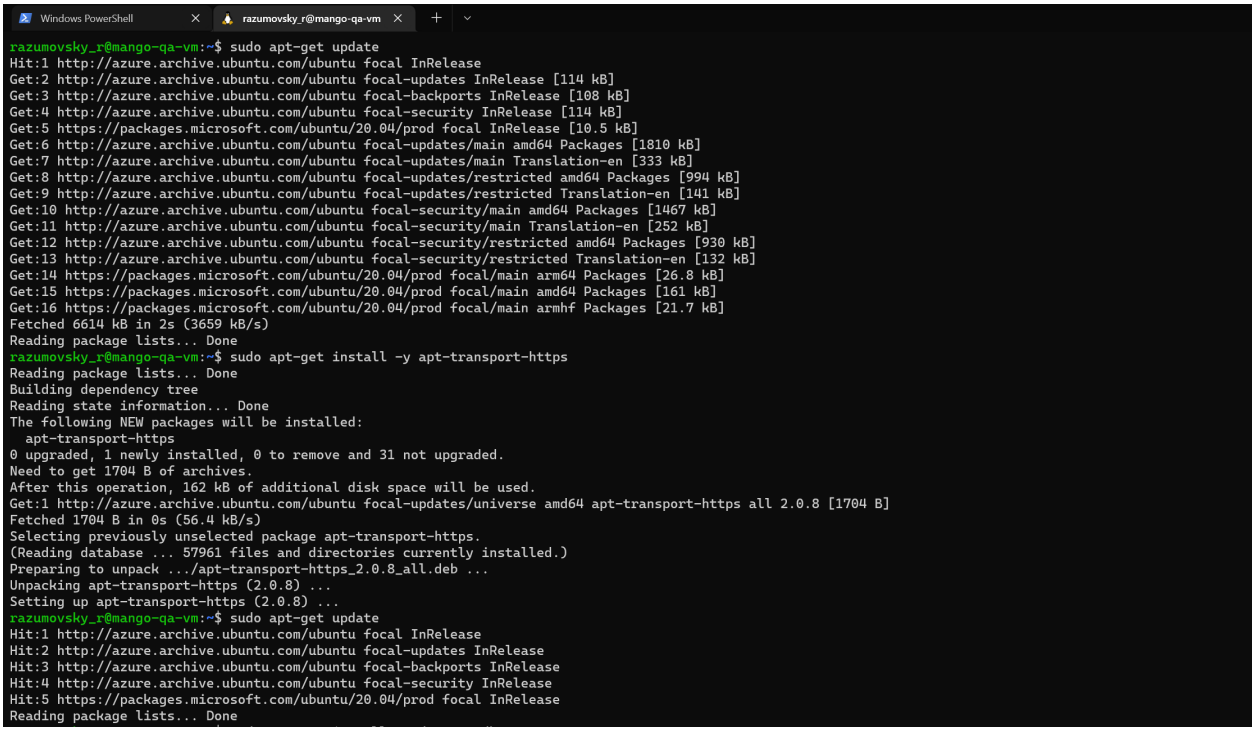

Figure 4. Ubuntu 20.04 install .NET 6.0 SDK terminal output.

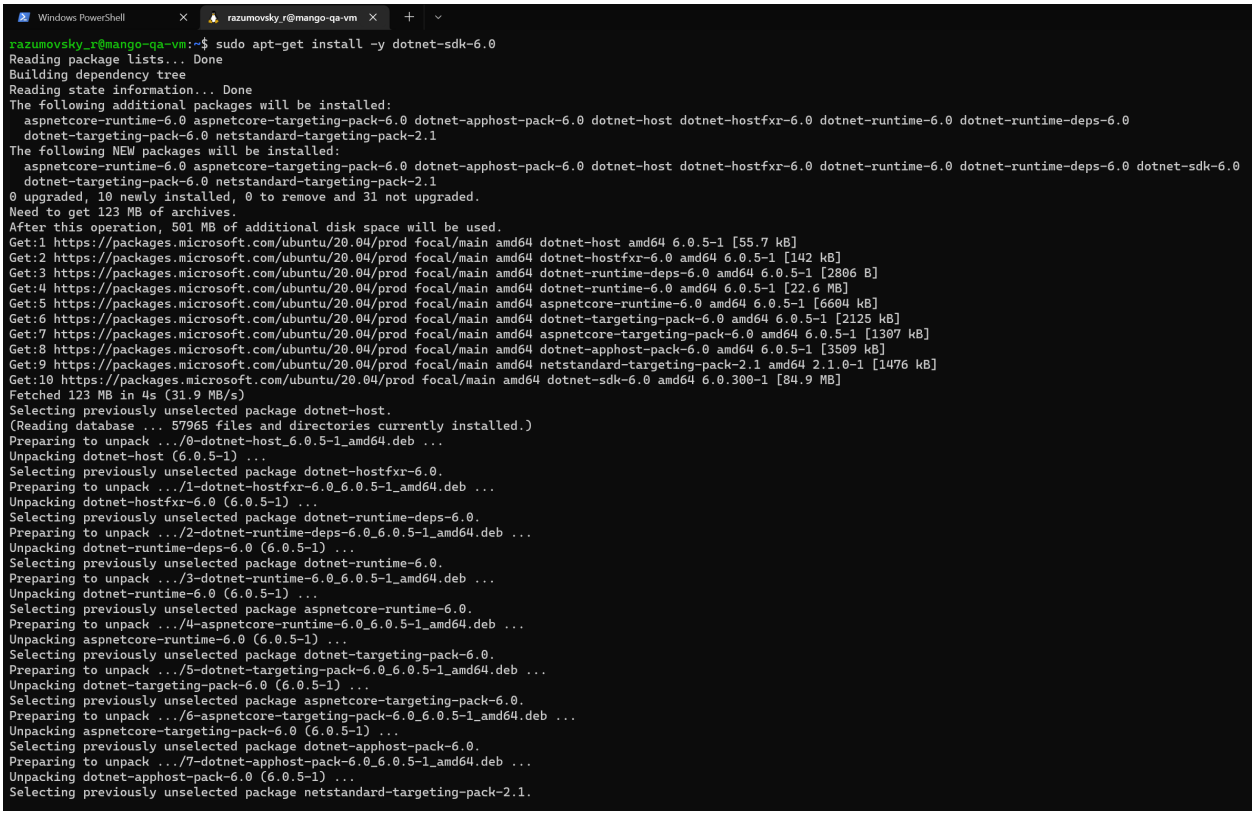

Figure 5. Ubuntu 20.04 install .NET 6.0 SDK terminal output.

```
Selecting previously unselected package netstandard-targeting-pack-2.1.
Preparing to unpack .../8-netstandard-targeting-pack-2.1_2.1.0-1_amd64.deb ...
Unpacking netstandard-targeting-pack-2.1 (2.1.0-1) ...
Selecting previously unselected package dotnet-sdk-6.0.
Preparing to unpack .../9-dotnet-sdk-6.0_6.0.300-1_amd64.deb ...
Unpacking dotnet-sdk-6.0 (6.0.300-1) ...
Setting up dotnet-host (6.0.5-1) ...
Setting up dotnet-apphost-pack-6.0 (6.0.5-1) ...
Setting up netstandard-targeting-pack-2.1 (2.1.0-1) ...
Setting up dotnet-targeting-pack-6.0 (6.0.5-1) ...
Setting up dotnet-runtime-deps-6.0 (6.0.5-1) ...
Setting up aspnetcore-targeting-pack-6.0 (6.0.5-1) ...
Setting up dotnet-hostfxr-6.0 (6.0.5-1) ...
Setting up dotnet-runtime-6.0 (6.0.5-1) ...
Setting up aspnetcore-runtime-6.0 (6.0.5-1) ...
Setting up dotnet-sdk-6.0 (6.0.300-1) ...
This software may collect information about you and your use of the software, and send that to Microsoft.
Please visit http://aka.ms/dotnet-cli-eula for more information.
Welcome to .NET!
---------------------
Learn more about .NET: https://aka.ms/dotnet-docs
Use 'dotnet --help' to see available commands or visit: https://aka.ms/dotnet-cli-docs

Telemetry
---------
The .NET tools collect usage data in order to help us improve your experience. It is collected by Microsoft and shared with the community. You can opt-out of telemetry by setting
the DOTNET_CLI_TELEMETRY_OPTOUT environment variable to '1' or 'true' using your favorite shell.

Read more about .NET CLI Tools telemetry: https://aka.ms/dotnet-cli-telemetry

Configuring...
--------------
A command is running to populate your local package cache to improve restore speed and enable offline access. This command takes up to one minute to complete and only runs once.
Processing triggers for man-db (2.9.1-1) ...
```

Figure 6. Ubuntu 20.04 install .NET 6.0 SDK terminal output.

In order to install the .NET Runtime we refer again to the Microsoft documentation, that is

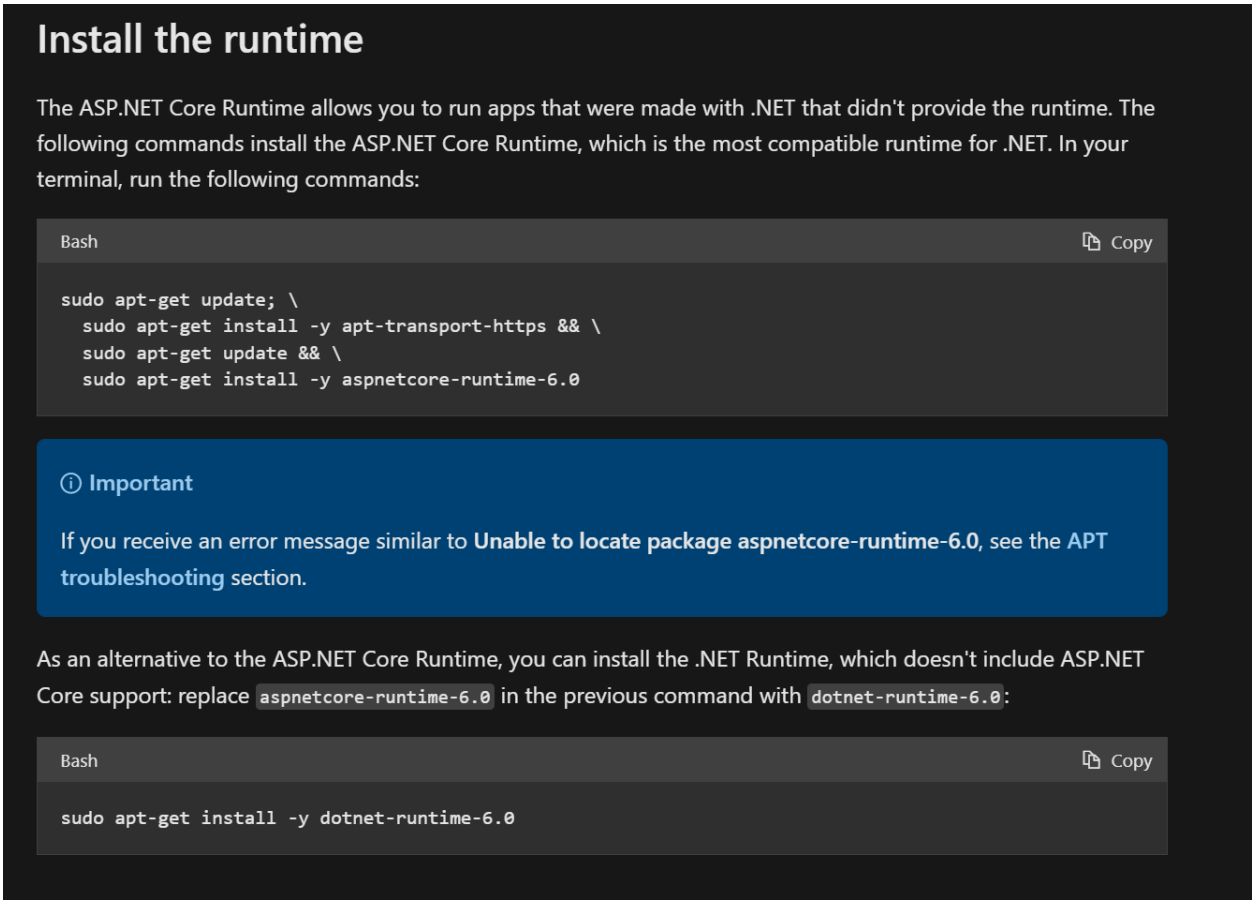

Figure 7. Install the .NET SDK or the .NET Runtime on Ubuntu MSDN.

We install .NET runtime using the commands

- `sudo apt-get update`
- `sudo apt-get install -y apt-transport-https`
- `sudo apt-get update`
- `sudo apt-get install -y aspnetcore-runtime-6.0`

Terminal output as follows

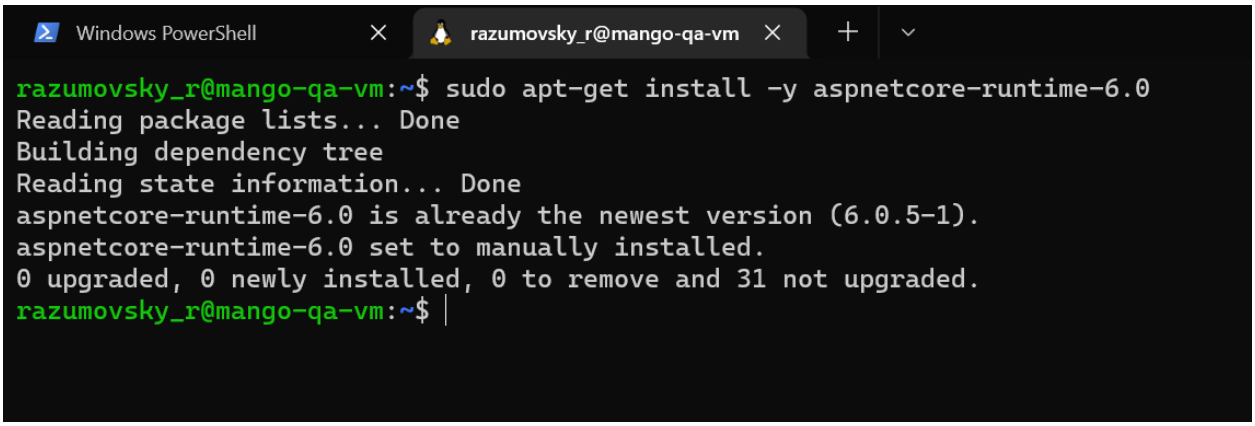

Figure 8. Ubuntu 20.04 install .NET 6.0 Runtime terminal output.

Therefore, the .NET SDK and Runtime are installed so that we are able to run specified .NET app on behalf of our Ubuntu virtual machine.

4. Copy build files to the VM via SSH

Now we have to build our .NET Core Web Application to the specified folder, say `/mango-linux-build/src`. Note that it is much better to build it on behalf of Windows 10 main machine, not WSL 2.0 one. We use the following commands to build .NET Core Web App with `Release` configuration

- `cd E:/RiderProjects/MangoMessengerAPI/MangoAPI.Presentation`
- `dotnet publish "MangoAPI.Presentation.csproj" -r linux-x64 -o /mango-linux-build/src`

Terminal output is as follows

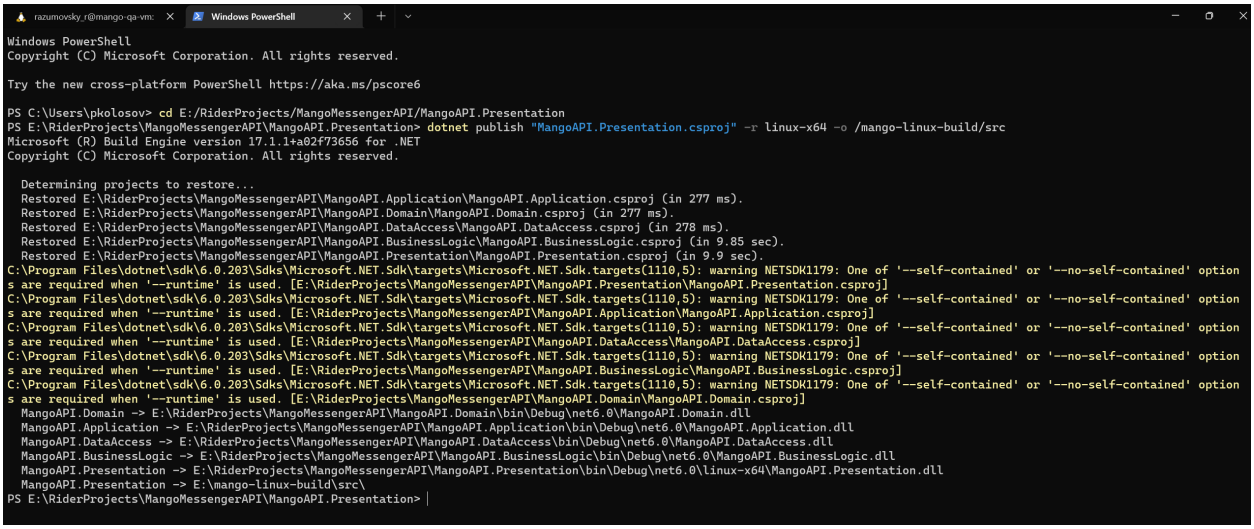

Figure 9. Publish .NET Web app terminal output.

Let's create the folder `mango-backend` where build files to be stored. Do not forget to connect to your Azure VM via SSH. Do not also forget to assign read-write privileges to the folder, using the commands

- `sudo mkdir /mango-backend`
- `sudo chmod a+rwx /mango-backend`

Terminal output:

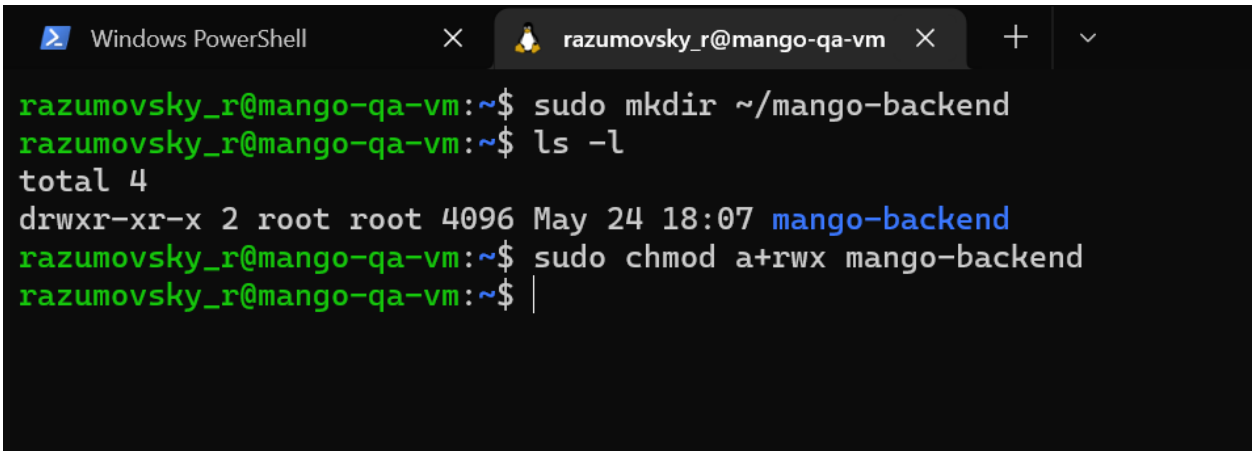

Figure 10. Create folder at remote VM.

As next step consider to copy build files to the remote folder on your Azure VM so that we execute our program after. We copy the build files on behalf of WSL2 this time. In order to copy the build files we use following commands

- `cd /mnt/e/mango-linux-build`

- `scp -r -i /.ssh/id_rsa ./src/* razumovsky_r@VM_IP_ADDRESS:/home/razumovsky_r/mango-backend`

where `id_rsa` is the private key. Terminal output:

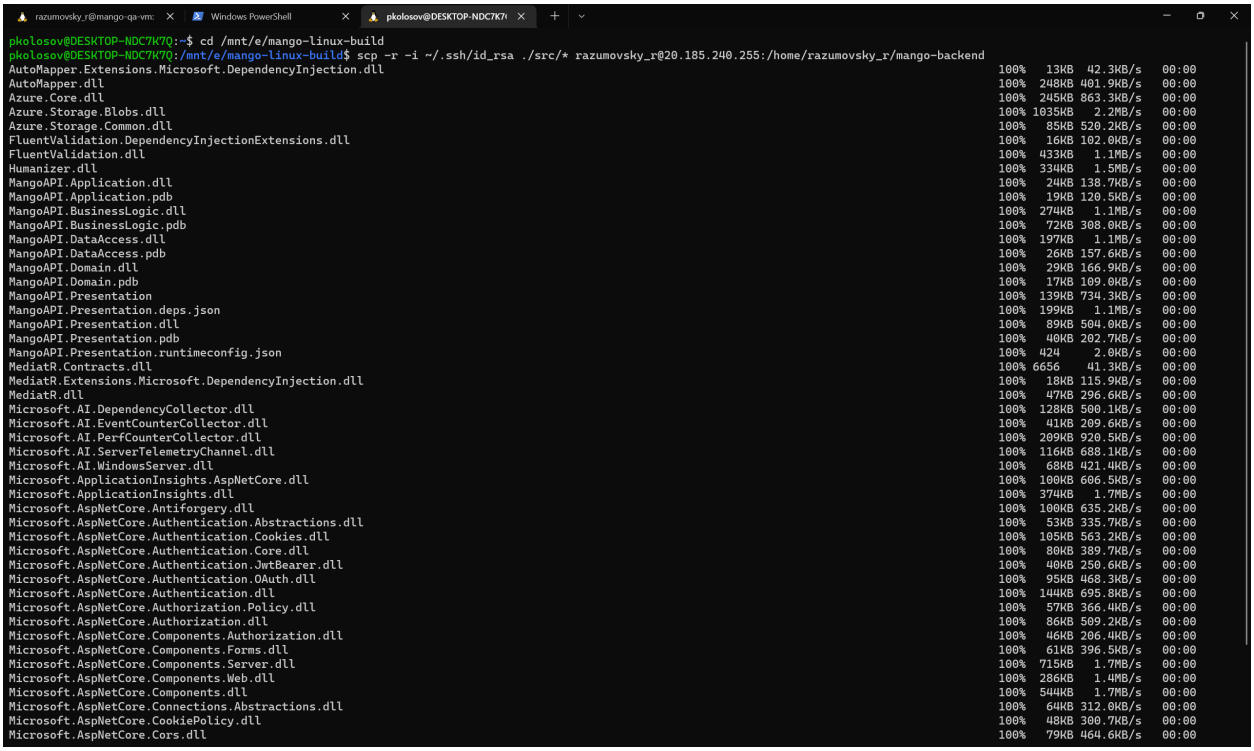

Figure 11. Copy build files via SSH.

Ensure build files are copied successfully to the remote VM, use the command `ls -l mango-backend`. Terminal output:

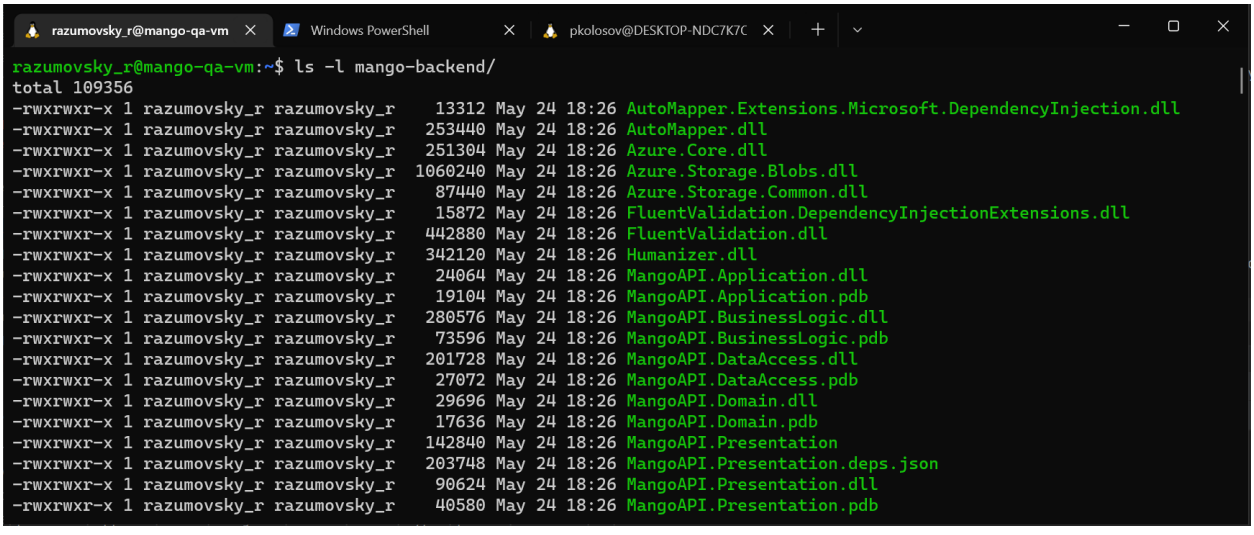

Figure 12. Check files at remote VM.

Therefore, the specified .NET Core web application is copied to the Ubuntu virtual machine so that it can be executed thanks to the previously installed .NET SDKs and runtimes.

5. Configure Ubuntu service

In this section the main aim is to implement an Ubuntu service such that runs our previously built .NET Core web application. It means that we have to configure the environment variables used in our application as well as to configure the firewall rules so that application will be able to communicate with another resources like databases, blobs etc. Ubuntu server refers to the entry point of the web app, that is

```
/home/razumovsky_r/mango-backend/MangoAPI.Presentation
```

Use the command to create service

```
sudo vim /etc/systemd/system/mangoback.service
```

Paste the following text there

```
[Unit]
Description=Mango Messenger Backend Service for Azure Dev Environment
After=network.target

[Service]
Environment=ASPNETCORE_URLS=http://+:8080/
Environment=MANGO_JW_ISSUER="https://front.mangomessenger.company"
Environment=MANGO_JWT_AUDIENCE="https://back.mangomessenger.company"
Environment=MANGO_JWT_SIGN_KEY="d32d7cea-4cb8-4488-aa94-323ffb8cbdf4"
Environment=MANGO_EMAIL_NOTIFICATIONS_ADDRESS="mango@gmail.com"
Environment=MANGO_FRONTEND_ADDRESS="https://front.mangomessenger.company/"
Environment=MANGO_DATABASE_URL="database.connection.string"
Environment=MANGO_SEED_PASSWORD="seedPass"
Environment=MANGO_BLOB_URL="blob.url.connection.string"
Environment=MANGO_BLOB_CONTAINER="container.name"
Environment=MANGO_BLOB_ACCESS="blob.access.url"
Environment=MANGO_MAILGUN_API_KEY="mailgun.api.key"
Environment=MANGO_MAILGUN_API_BASE_URL="https://api.mailgun.net"
Environment=MANGO_MAILGUN_API_BASE_DOMAIN="back.mangomessenger.company"
Environment=MANGO_BACKEND_ADDRESS="https://back.mangomessenger.company/"
Type=simple
WorkingDirectory=/home/razumovsky_r/mango-backend
ExecStart=/home/razumovsky_r/mango-backend/MangoAPI.Presentation
User=razumovsky_r
```

```
Group=razumovsky_r

[Install]
WantedBy=multi-user.target
```

From the vim it should look as follows [Kol22b]

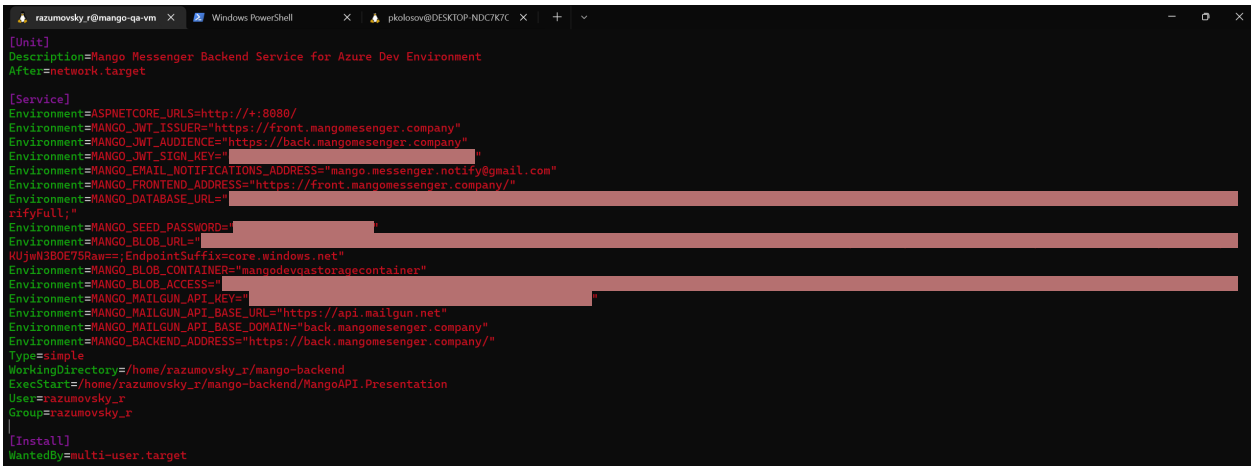

Figure 13. Ubuntu service opened in vim.

Make sure all resources are listening from the outside, check firewall rules on database side prior to run the service. Start and check health of the service using

- `sudo systemctl start mangoback`
- `sudo systemctl status mangoback`

Terminal output:

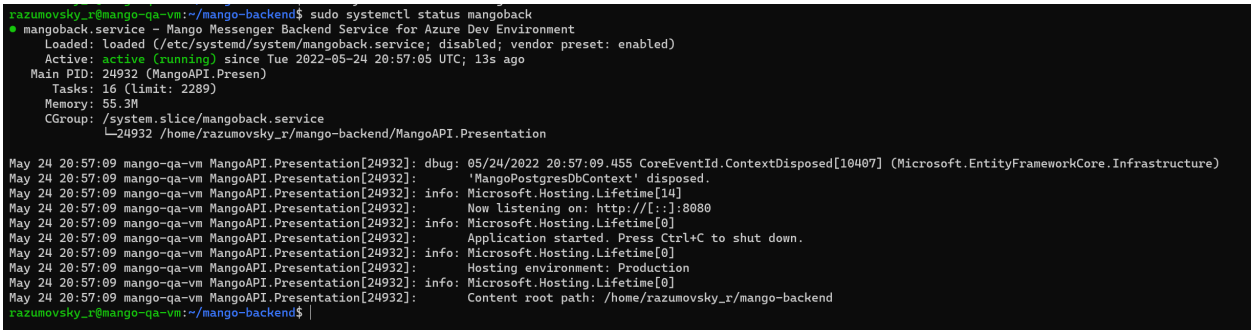

Figure 14. Run ubuntu service and check status, terminal output.

As a result of this section, we have created a specified ubuntu service that runs our previously copied .NET Core web application using installed .NET SDK and runtime.

6. Install and configure Nginx server

Now we have to configure the `nginx` server in order to expose our .NET Core web application to the outside. As a result of this section web app will be exposed and accessible via VM's external IP address. Let's install it using the commands

- `sudo apt update -y`
- `sudo apt install -y nginx build-essential`

Terminal output:

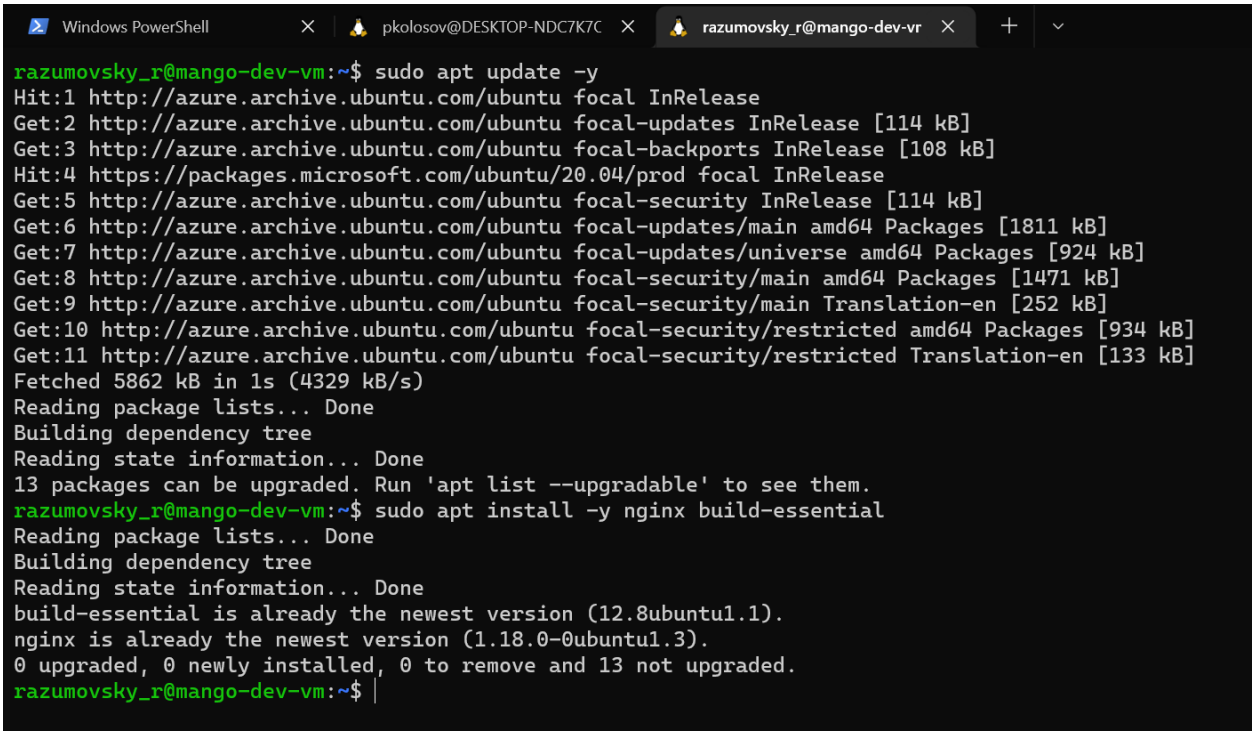

Figure 15. Ubuntu install nginx terminal output.

Next, it is necessary to create nginx configuration [Kol22a] that exposes our application, that is

```
server {
    server_name STATIC_IP_ADDRESS_OF_VM;

    location / {
        include proxy_params;
        proxy_pass http://127.0.0.1:8080;
    }

    location /swagger {
```

```
        include proxy_params;
        proxy_pass http://127.0.0.1:8080;
    }

    location /api {
        include proxy_params;
        proxy_pass http://127.0.0.1:8080;
    }

    location /notify {
        proxy_pass http://127.0.0.1:8080;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

We create it at the following path on behalf of our Azure VM via SSH

```
sudo vim /etc/nginx/conf.d/back.mangomesenger.company.conf
```

Restart nginx and validate its state using the commands

- `sudo systemctl restart nginx`
- `sudo nginx -t`

Terminal output:

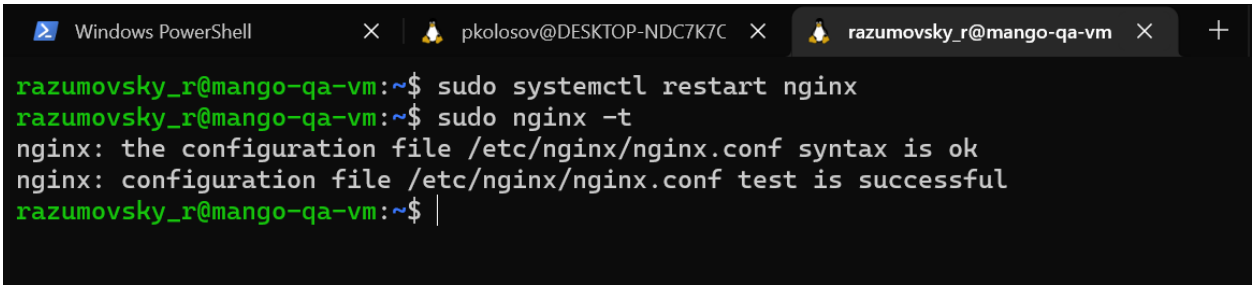

Figure 16. Restart and test nginx terminal output.

Now we must be able to find our application listening to the

```
http://STATIC_IP_ADDRESS_OF_THE_VM
```

And actually it works as expected

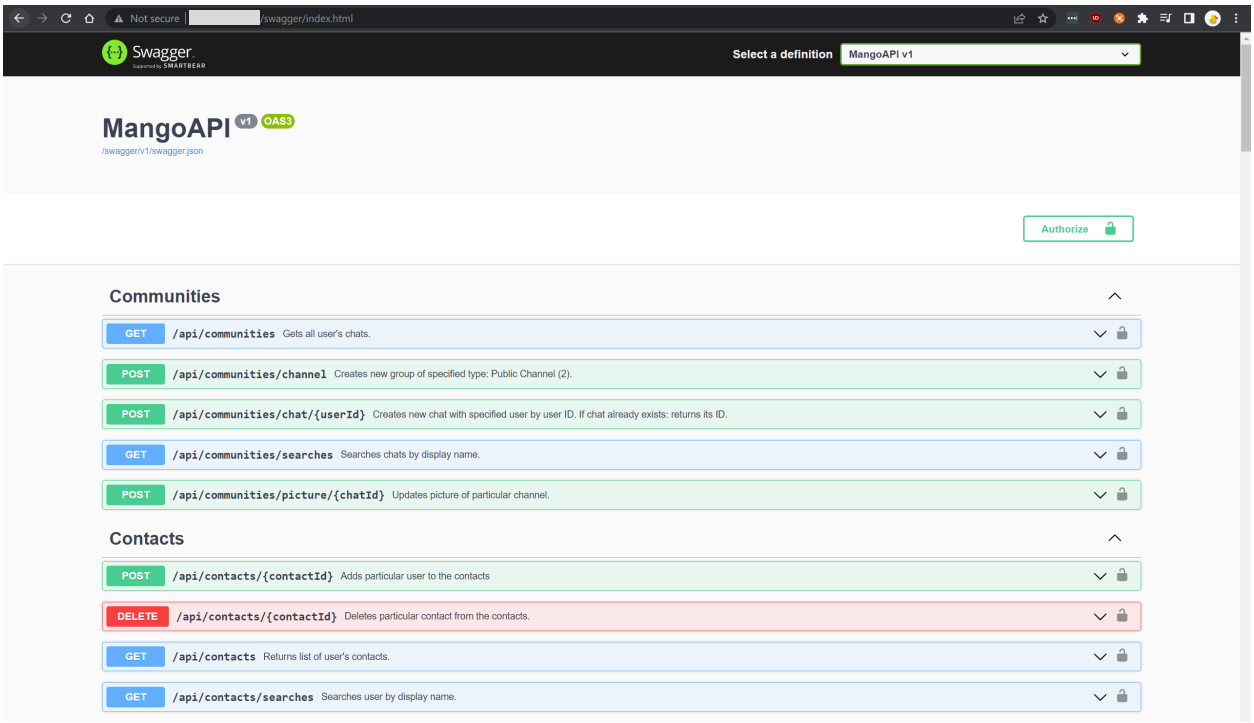

Figure 17. .NET Core web app accessed via browser using static IP address of the virtual machine.

In this section we have installed and configured the `nginx` web server so that it exposes our .NET Core web application (run on behalf of Ubuntu service) from the previous section and makes it available from the web browser under the url `http://STATIC_IP_ADDRESS_OF_THE_VM`.

7. Configure domain name and SSL

In this section our main aim is to assign specified (previously bought) domain name to our .NET Core web application as well as to configure SSL certificate for it. What is domain name?

Domain name – is a string of text that maps to a numeric IP address, used to access a website from client software [Clo22]. The actual address of a website is a complex numerical IP address (e.g. 103.21.244.0), but thanks to DNS, users are able to enter human-friendly domain names and be routed to the websites they are looking for.

For instance, the domain name can be bought on the one of the following resources

- https://www.name.com
- https://www.namecheap.com
- https://get.tech

8. Deploy frontend project

9. Conclusions

Conclusions of your manuscript.

References

[Clo22] Cloudflare. "What is a domain name? — Domain name vs. URL". published electronically at https://www.cloudflare.com/learning/dns/glossary/what-is-a-domain-name/, 2022.

[Cor22] Microsoft Corporation. "Install the .NET SDK or the .NET Runtime on Ubuntu". published electronically at https://docs.microsoft.com/en-us/dotnet/core/install/linux-ubuntu, 2022.

[Kol22a] Petro Kolosov. "Nginx config gist". published electronically at https://gist.github.com/kolosovpetro/f993f02f8cf2e9be9f574791d5a740ce, 2022.

[Kol22b] Petro Kolosov. "Ubuntu service example gist". published electronically at https://gist.github.com/kolosovpetro/c4e863d0bb8876d41f5e5ee479c46db3, 2022.

[Kol22c] Petro Kolosov. "VM Creation screens". published electronically at https://drive.google.com/file/d/1odouQARBd1mV-tV6OTnj5B-bxkl0N40J/view?usp=sharing, 2022.

Email address: kolosovp94@gmail.com

URL: https://razumovsky.me/