

DOCKER

RG - rg-dooker

Frontend - VM

Backend - VM

Database – SQL

+++++

```
File Edit View
Vm => Public ip
Docker engine install
Dockerfile => Image I
Image run >

Image 1 > Frontend
Image 2 > backend

Image1 > Container1
Image2 > Container2
```

+++++

1) Create **rg** - rg-dooker

+++++

2) Create **vm** - vm-dooker

UN - azureuser

PW - Mommy7Daddy!

a) Open powershell

```
ssh azureuser@20.126.139.18
```

UN - azureuser

PW - Mommy7Daddy!

b) Install docker in vm using below url

<https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04>

c) exit

d) **docker --version**

+++++

3) Create **sql database** – sql-dooker

a) Create new server - sql-dooker123

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

Database name * ✓

Server * ⓘ ▼

[Create new](#)

b) SQL authentication

UN - azureuser

PW - Mommy7Daddy!

+++++

4) Now putting backend code in our vm machine from below url

<https://github.com/devopsinsiders/PyTodoBackendMonolith>

File	Commit Message	Commit Hash	Time
.gitignore	added .env	8f314fe	5 months ago
Dockerfile	initial commit		8 months ago
Readme.md	Update Readme.md		last month
app.py	added code		5 months ago
requirements.txt	added .env		5 months ago

a) **git clone** <https://github.com/devopsinsiders/PyTodoBackendMonolith.git>

ls

```

For more help on how to use Docker, head to https://docs.docker.com/go/guides/
azureuser@vm-dooker:~$ git clone https://github.com/devopsinsiders/PyTodoBackendMonolith.git
Cloning into 'PyTodoBackendMonolith'...
remote: Enumerating objects: 59, done.
remote: Counting objects: 100% (59/59), done.
remote: Compressing objects: 100% (52/52), done.
remote: Total 59 (delta 30), reused 21 (delta 5), pack-reused 0
Receiving objects: 100% (59/59), 14.69 KiB | 1.05 MiB/s, done.
Resolving deltas: 100% (30/30), done.
azureuser@vm-dooker:~$ ls
PyTodoBackendMonolith
azureuser@vm-dooker:~$

```

b) cd PyTodoBackendMonolith/

ls

```

azureuser@vm-dooker:~$ cd PyTodoBackendMonolith/
azureuser@vm-dooker:~/PyTodoBackendMonolith$ ls
Dockerfile  Readme.md  app.py  requirements.txt
azureuser@vm-dooker:~/PyTodoBackendMonolith$

```

c) Now we need an image from docker hub that has python and pip installed in it as per prerequisites

Prerequisites

Before getting started, make su

- source_image_reference =
version = "latest" }
- Python
- pip

So by installing python image our time got reduced since image already have pip and python installed in it

d) SEARCH – Docker reference

<https://docs.docker.com/reference/dockerfile/>

e) cat Dockerfile

Just to check what is there inside

```

azureuser@vm-docker:~/PyTodoBackendMonolith$ cat Dockerfile
# Use the official Python image as the base image
FROM python:3.9

# Set the working directory in the container
WORKDIR /app

# Copy the application files into the container
COPY . .

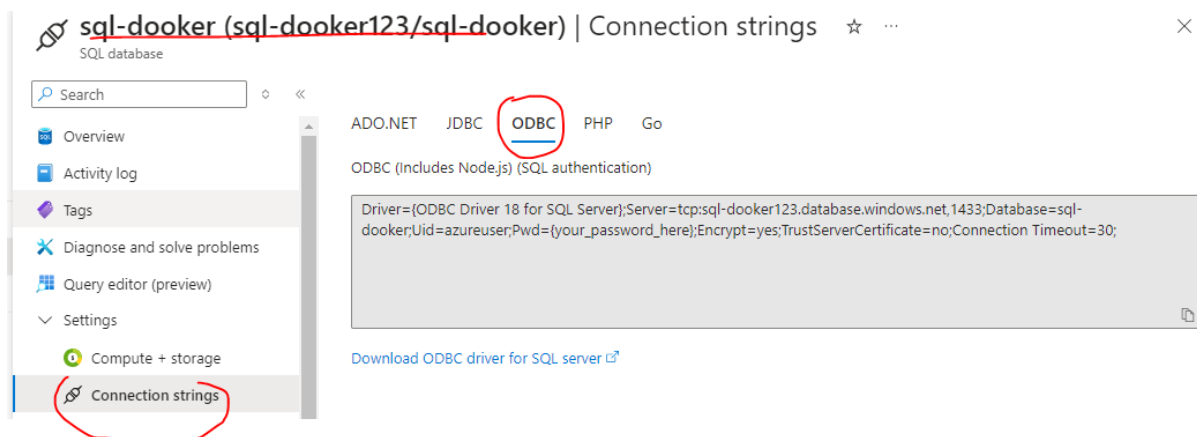
# Install necessary packages
RUN apt-get update && apt-get install -y unixodbc unixodbc-dev
RUN curl https://packages.microsoft.com/keys/microsoft.asc | apt-key add -
RUN curl https://packages.microsoft.com/config/debian/10/prod.list > /etc/apt/sources.list.d/mssql-release.list
RUN apt-get update
RUN ACCEPT_EULA=Y apt-get install -y msodbcsql17

RUN pip install -r requirements.txt

# Start the FastAPI application
azureuser@vm-docker:~/PyTodoBackendMonolith$ client_loop: send disconnect: Connection reset
PS C:\Users\HP>

```

f) Now go to sql db and copy “connection string”



g) **docker images**

To check whether any docker image is there or not

h) Now we will build docker image, so we will build docker in that folder only that contains docker files

docker build -t backendimage .

i) **docker images**

```

root@vm-docker:/home/azureuser/PyTodoBackendMonolith# docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
backendimage        latest          c44e3af7ffaf   About a minute ago  1.12GB
root@vm-docker:/home/azureuser/PyTodoBackendMonolith#

```

j) **nano app.py**

Update connection string by changing password

Driver={ODBC Driver 17 for SQL Server};Server=tcp:sql-dooker123.database.windows.net,1433;Database=sql-dooker;Uid=azureuser;Pwd={Mommy7Daddy!};Encrypt=yes;TrustServerCertificate=no;Connection Timeout=30;

```
# Load environment variables from .env file
load_dotenv()

connection_string = "Driver={ODBC Driver 17 for SQL Server};Server=tcp:sql-dooker123.database.windows.net,1433;Database=sql-dooker;Uid=azureuser;Pwd={mommy70addy!};Encrypt=yes;TrustServerCertificate=no;Connection Timeout=30;"

app = FastAPI()
```

K) We can bring sql db online by clicking and changing time as below

The screenshot shows the Azure portal interface for a SQL database instance named 'sql-dooker (sql-dooker123/sql-dooker)'. The 'Essentials' tab is selected, displaying various settings. The 'Auto-pause delay' is set to '2 hours', which is circled in red. Other visible settings include 'Server name' as 'sql-dooker123.database.windows.net', 'Connection strings', 'Pricing tier' as 'General Purpose - Serverless: Gen5, 1 vCore', and 'Subscription' as 'Free Trial'.

I) **docker run -dp 8000:8000 backendimage**

here in dp – d means detach mode and p means port exposing

also we use i – interactive mode

t – terminal

p- port

The terminal screenshot shows the command `docker run -dp 8000:8000 backendimage` being executed. The output displays the container ID `46a5f71f4483185d445b4aabea1ebdd0a087031930123a2b4bd5a4ea54e1b6c2`. A red circle is drawn around the container ID.

129

vm
port

App port
No change

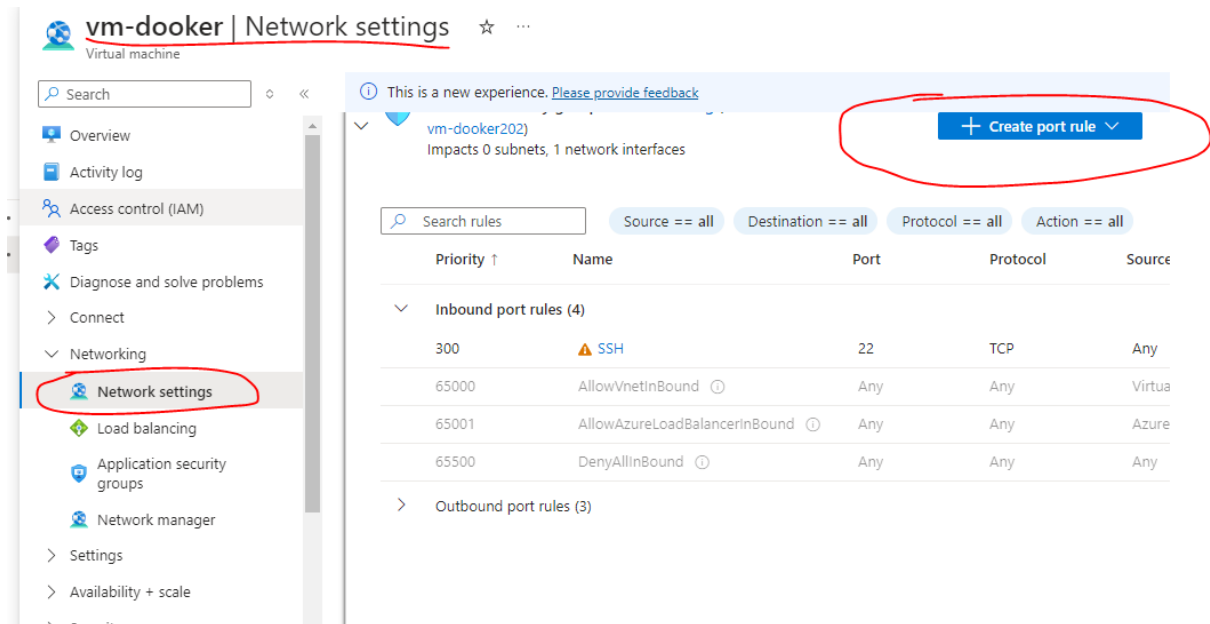
`docker run -dp 8000:8000 backendimage`

m) **docker ps**

docker ps -a

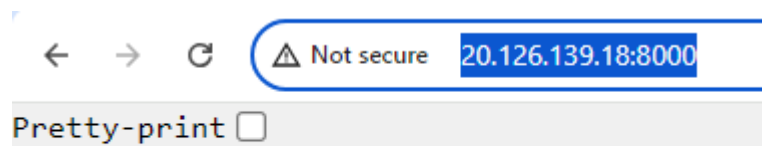
The terminal screenshot shows the output of the `docker ps -a` command. It lists a single container with ID `46a5f71f4483`, image `backendimage`, command `"uvicorn app:app --h..."`, created 7 minutes ago, status `Up 7 minutes`, ports `0.0.0.0:8000->8000/tcp, :::8000->8000/tcp`, and name `sharp_cori`.

n) Open 8000 port on vm



o) Now run on browser – public ip of vm:port

<http://20.126.139.18:8000/>



```
{"detail": "Not Found"}
```