

Dokumentation WebAPI mit Git

Livio Piccolotto

2EI

Schritt 1:

Erstellen eines GitRepository „Minimal API with MongoDB“:


In GitHub ein neues Repository aufsetzen:

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?



[Import a repository.](#)

Required fields are marked with an asterisk (*).

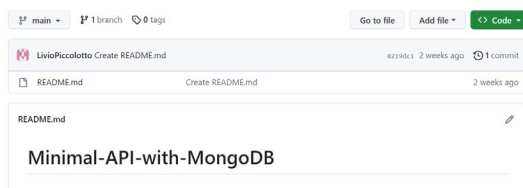
Owner *  LivioPiccolotto / Repository name *
⚠ Your new repository will be created as Minimal-API-with-Mongo-DB.

Great repository names are short and memorable. Need inspiration? How about [scaling-engine](#)?

Description (optional)

- ☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

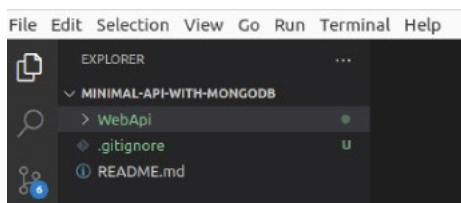
Nun fügt man noch eine Readme Datei hinzu:



Schritt 2: Grundgerüst erstellen

1. Klonen des Projekts:
`git clone https://github.com/LivioPiccolotto/Minimal-API-with-MongoDB.git`
2. In das Verzeichnis wechseln: `cd Minimal-API-with-MongoDB`
3. Erstellen eines .NET Projekts WebApi Template web: `dotnet new web --name WebApi`
4. Im gleichen Verzeichnis ein. gitignore: `dotnet new gitignore`

Ergebnis:

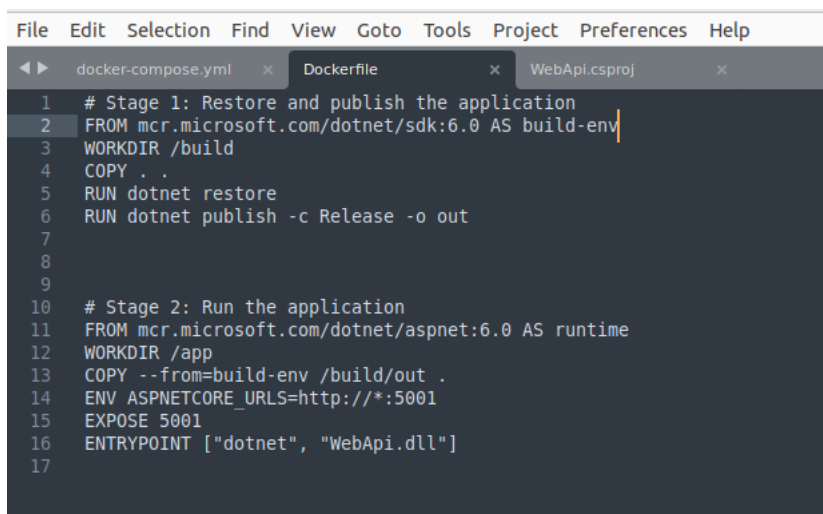


5. Neues Terminal öffnen und in den WebApi Ordner navigieren. Mit *dotnet run* die Anwendung starten.
6. Im Browser sollte nun Hello World angezeigt werden.
7. Mit [ctrl] c kann man die Anwendung wieder schliessen.
8. In „launchSettings.json“ ist das http-Profil definiert. Den Port sollte man auf 5001 wechseln:

```
    },  
    "profiles": {  
      "WebApi": {  
        "commandName": "Project",  
        "dotnetRunMessages": true,  
        "launchBrowser": true,  
        "applicationUrl": "http://localhost:5001",  
        "environmentVariables": {  
          "ASPNETCORE_ENVIRONMENT": "Development"  
        }  
      }  
    },  
  },  
}
```

9. Nun sollte der Aufruf auch über <https://localhost:5001> erreichbar sein.

Schritt 3: Dockerfile erstellen:

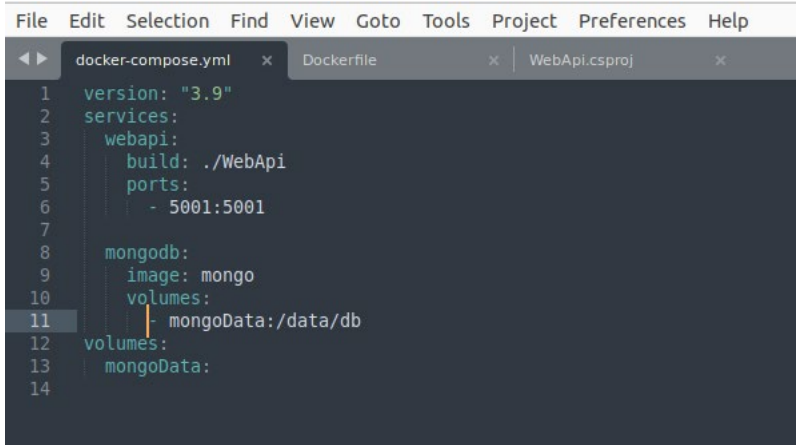


```
File Edit Selection Find View Goto Tools Project Preferences Help  
docke-compose.yml x Dockerfile x WebApi.csproj x  
1 # Stage 1: Restore and publish the application  
2 FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build-env  
3 WORKDIR /build  
4 COPY . .  
5 RUN dotnet restore  
6 RUN dotnet publish -c Release -o out  
7  
8  
9  
10 # Stage 2: Run the application  
11 FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS runtime  
12 WORKDIR /app  
13 COPY --from=build-env /build/out .  
14 ENV ASPNETCORE_URLS=http://*:5001  
15 EXPOSE 5001  
16 ENTRYPOINT ["dotnet", "WebApi.dll"]  
17
```

Diese Dockerfile besteht aus zwei Stufen. In der ersten Stufe wird der Code mit dem .NET SDK kompiliert und veröffentlicht. In der zweiten Stufe wird das resultierende Veröffentlichungspaket in ein ASP.NET-Image kopiert und die Anwendung gestartet, wobei der Container auf Port 5001 lauscht.

Schritt 3: Docker-compose.yml erstellen.

Mit docker compose up soll die Anwendung mit Hilfe des Dockerfiles erzeugt werden und gestartet werden.



```
1 version: "3.9"
2 services:
3   webapi:
4     build: ./WebApi
5     ports:
6       - 5001:5001
7
8   mongodb:
9     image: mongo
10    volumes:
11      - mongoData:/data/db
12 volumes:
13   mongoData:
```

Dieser Code definiert eine Docker-Compose-Datei mit zwei Diensten. Der Dienst "webapi" erstellt ein Docker-Image für eine Web-API-Anwendung und leitet den Port 5001 des Containers auf den Port 5001 des Hosts weiter. Der Dienst "mongodb" verwendet das offizielle "mongo" Docker-Image und bindet ein Volumen "mongoData" an den Datenbankpfad im Container. Diese Konfiguration ermöglicht es, eine Umgebung mit Web-API und MongoDB schnell bereitzustellen und zu verwenden.

Schritt 6: Commit und Push

Alles auf GitPushen.

Befehle: `git config --global use.ernail livio.1616@gmail.com`

`Git config --global user.name Livipicc`