Aufgabe

Bei dieser Aufgabe soll sich näher mit Microservices beschäftigt werden sowie ein kleiner Service selbst erstellt werden.

Auswahl des Tools

Für die Umsetzung der Aufgabe verwende ich **C#** mit **.NET Core 5 (.NET 5)** und Visual Studio. Als Versionsverwaltung (DVC) kommt **Git**[1] zum Einsatz. Für die Veröffentlichung verwende ich **Docker** und hoste die Lösung hierzu in einem eigenen Repository auf Docker[2]

Angestrebte Ergebnisse

Es soll ein einfacher API basierter Microservice erstellt werden. Dieser Service soll einen Rechner für die Grundrechenarten Addition, Subtraktion, Multiplikation und Division verfügbar machen.

Anschließend soll dieser Service in einen Container verpackt und veröffentlicht werden. Der veröffentlichte Container soll über Docker Hub öffentlich verfügbar sein und bei Änderungen des Git Repository automatisiert erstellt werden. Der hierfür benötigte Code wird hierbei direkt von GitHub gezogen.

Umsetzung

Rein formal handelt es sich um eine .NET Core 5 und somit einer ASP.net Anwendung. Die Anwendung basiert auf die Nutzung von Controllern, jedoch wird kein MVC Pattern umgesetzt. Hierbei unterstützt Visual Studio die Entwicklung mit einem soliden und ausführbaren Grundgerüst.

Es kommt **MapController** für das Routing zum Einsatz. Hierbei werden beim Starten der Anwendung die Endpunkte (Endpoints/mögliche Ziele von Aufrufen) anhand der Annotierungen im Code festgelegt. In der nachfolgenden Abbildung ist ein Beispiel für die Route **/Addition** festgelegt.

```
[Route( template: "Addition")]
[ApiController]
public class AdditionController : ControllerBase
{
```

[1] https://github.com/ChristianKitte/CalculatorService
[2] https://hub.docker.com/repository/docker/ckitte/apibasedmicroservice

Es ist hierbei eine feste Konvention, Controller im Verzeichnis **Controllers** und hier den Klassen- und Dateinamen um **Controller** zu erweitern. So liegt der oben abgebildete Controller in **./Controllers/AdditionController.cs**.

Es ist möglich, bereits im Routing eine feste Validierung z.B. des Types vorzunehmen, jedoch wurde dies explizit nicht gemacht, um sich auf das Essenzielle konzentrieren zu können. Auch sind andere Möglichkeiten der Datenübergabe, wie beispielsweise JSON möglich. Hier wurde jedoch bewußt auf **GET** und der Übergabe von genau **zwei Zahlen** und einer **Rechenart** gesetzt, um die Schnittstelle möglichst einfach zu gestalten.

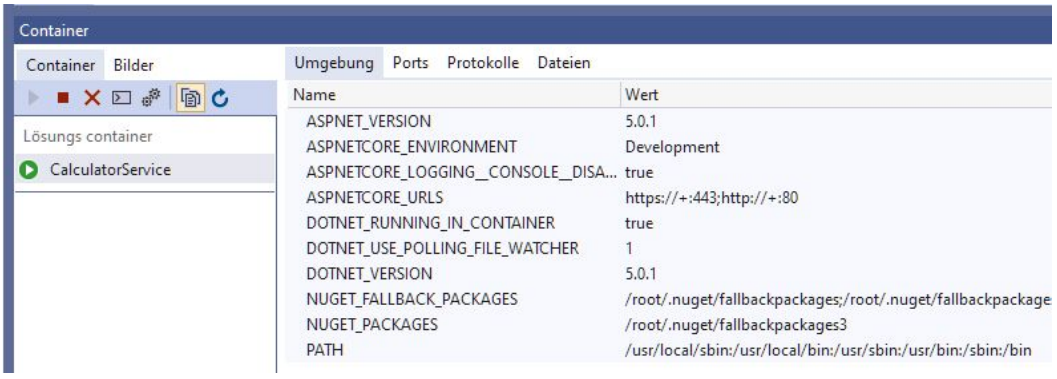
Befehle

Der Rechner unterstützt die vier Grundrechenarten. Die Syntax zur Nutzung des Rechners ist die Webadresse, gefolgt von der Rechenart sowie die Angabe zweier Zahlen, wie im Beispiel zu sehen: **https://domain/[addition/subtract/multiply/divide]/[Zahl 1]/[Zahl 2]** .

Je nach Start des Containers sowie beim lokalen Testen ist hierbei gegebenenfalls der genutzte Port mit anzugeben.

Veröffentlichung mit Docker Anpassungen

Visual Studio unterstützt die Nutzung von **Docker** auf vielerlei Art. So können komplette **Build Scripte für Container** auf Basis der aktuellen Anwendung erstellt und der Container im Anschluss erstellt werden (Separate Vorgänge). Container können **direkt aus der IDE** ohne und mit Debug Unterstützung ausgeführt und sehr komfortabel verwaltet werden, wie im Bild zu sehen ist:



Für die Konfiguration werden zwei Dateien, **Dockerfile** und **.dockerignore** verwendet. Das Erstere definiert das **Buildscript**, das zweit stellt eine **Ausschlussliste** dar, um die Container möglichst schmal zu halten.

Das Buildscript ist insofern besonders interessant, da im ersten Schritt zunächst **zwei** Images herunter geladen werden. Das eine Image wird im späteren Verlauf quasi als Worker für den **Build**, das andere für das **Deployment** verwendet. Ein schönes Beispiel, das zum

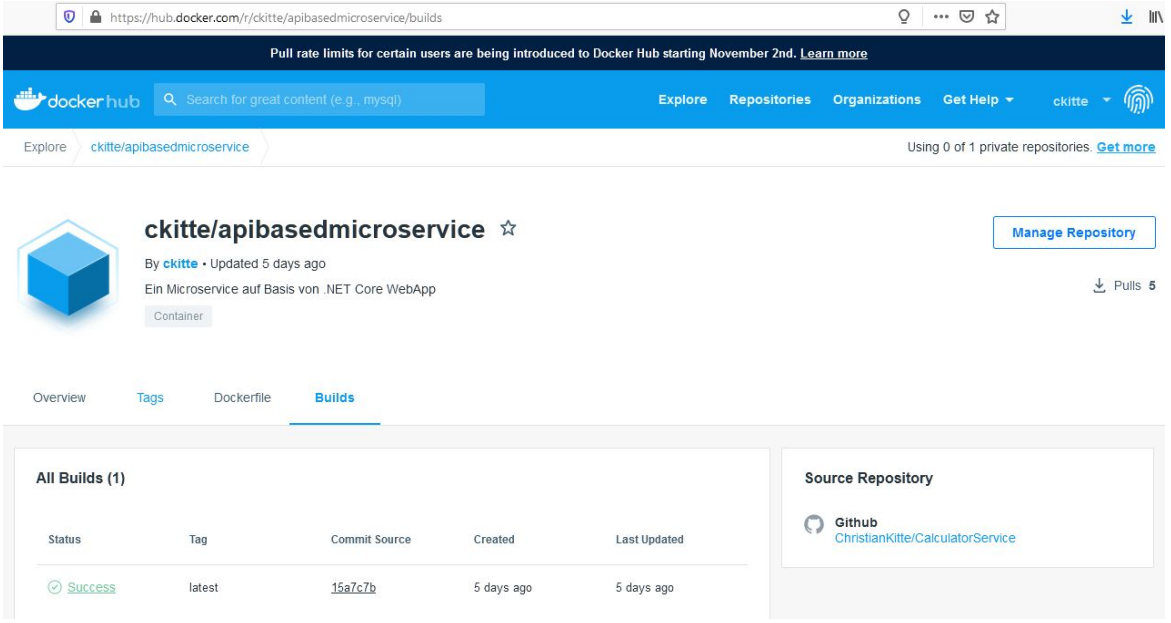
einen lehrreich, zum anderen den Zugang für Personen ohne Erfahrungen mit Docker (wie mich) vereinfacht.

Veröffentlichung mit Docker Repository

Docker ermöglicht die Erstellung eines kostenlosen Kontos mit für den privaten Bereich genügenden Möglichkeiten. Eine Aufstockung des Abonnements ist jedoch eher günstig.

Die **Verbindung von Nutzername und Name des Repository** sorgt hierbei für eindeutige Benennungen der verfügbaren Images. Weiterhin ist es möglich, ein Repository mit einem **GitHub** oder **BitBucket** Konto zu verknüpfen und so **automatisierte Builds** zu generieren. Leider ist es nicht möglich, diese durch einen Badge zu visualisieren, wie es bei GitHub möglich ist.

In die andere Richtung ist es aus GitHub heraus möglich, **Container zu erstellen** und nach Docker **in ein dortiges Repository zu verschieben**. In diesem Fall kann ein **Badge** über den **Erfolg/Misserfolg** direkt Auskunft geben.



Verwendung des Docker Container

Für die Verwendung des Microservices kann nun einfach der entsprechende Container von Docker herunter geladen und ausgeführt werden. Hierzu ist lediglich ein **kostenloser Account** erforderlich.

Die untenstehende Abbildung zeigt den kompletten Vorgang mit allen Anweisungen, die für ein herunterladen und in Betrieb nehmen erforderlich sind.

Beachtenswert sind die erste zwei Zeilen für das Einloggen. Hier muss der **Benutzername** angegeben werden. Es ist etwas ungewöhnlich, dass die **Passworteingabe** zu keinerlei Rückmeldung oder Anzeige führt. In der Powershell funktioniert das EInfügen mit STRG+V zwar, aber auch hier kommt es zu **keinerlei Rückmeldung**.

Mit dem Befehl **docker run -p 8080:80 ckitte/apibasedmicroservice** wird der Service gestartet. Hierbei ist **8080** der Port außerhalb des Containers, hier localhost:8080, und **80** der Port innerhalb des Containers. **Somit horcht der Service im Container am Port 80 und ich spreche ihn von außen mit 8080 an**.

Die letzte Zeile (**warn**) rührt daher, dass die Umleitungen für **HTTPS** nicht konfiguriert wurden. Auch hier sollte der Code möglichst einfach sein und sich zunächst auf die grundsätzliche Umsetzung konzentrieren.

```
PS C:\Users\Christian Kitte> docker login --username ckitte
Password:
Login Succeeded
PS C:\Users\Christian Kitte> docker pull ckitte/apibasedmicroservice
Using default tag: latest
latest: Pulling from ckitte/apibasedmicroservice
6ec7b7d162b2: Already exists
f48adbf33222: Already exists
2a0bbc85b650: Already exists
7ab77a7157c0: Already exists
b70652007fb4: Already exists
bafc2925e2e2: Pull complete
a16f0983c568: Pull complete
Digest: sha256:692c20ae995aa49b3673df2ea59f14eda580a7c9d1a692769a4dde792ac713dc
Status: Downloaded newer image for ckitte/apibasedmicroservice:latest
docker.io/ckitte/apibasedmicroservice:latest
PS C:\Users\Christian Kitte> docker run -p 8080:80 ckitte/apibasedmicroservice
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
warn: Microsoft.AspNetCore.HttpsPolicy.HttpsRedirectionMiddleware[3]
      Failed to determine the https port for redirect.
```

Nach dem Starten des Containers kann nun der Microsservice genutzt werden. Mehr noch können **mehrere** Container gestartet werden, welche auf **verschiedene** Ports lauschen. Das untere Bild zeigt eine **Nutzung des Service an Port 8080**.