Praxisprojekt – Cloud Computing 2

Übersetzungs-APP

mit Azure Cognitive Services Translator

von Thomas Alpert

Inhalt

Einleitung

Die Aufgabenstellung lässt solch folgendermaßen Paraphrasieren. Das Projektziel ist die Entwicklung einer Web-App, die Azure Cognitive Services nutzt und Funktionen: Übersetzung unterstützt. Die Web-App soll in der Lage sein, Benutzereingaben und Antworten in einer Cloud-Datenbank zu speichern. Das Deployment soll in der Azure Cloud-Umgebung erfolgen und mit Terraform automatisiert werden. Die Infrastruktur soll so konfiguriert werden, dass die App in verschiedenen Regionen einfach repliziert werden kann. Das Deployment kann auch als Docker-Container verwendet sein und soll mit Ansible automatisiert werden. Die Konfiguration der App soll mit Ansible erfolgen, bevorzugt über Umgebungsvariablen. Zusammengefasst lässt sich die Aufgabenstellung daraus in die funktionalen und nicht funktionalen Anforderungen untergliedern:

Funktionale Anforderungen:

- Entwicklung eine Web-Applikation
- SaaS: Azure Cognitive Services Translator
- PaaS: beliebige Datenbank
  - → **Keine unnötigen / doppelten Anfragen an Translate-API → Entnahme aus DB**
- Deployment der Web-App in Azure Infrastruktur
- IaaS: Infrastruktur mit Terraform
- Deployment über Ansible

Nicht-Funktionale Anforderungen:

- Infrastruktur as Code Ansatz
- GitHub Repository / (Docker-)Container-Image
- Keine Secrets im Code / Auslagern von Credentials

Theorie / Grundlagen / Technologien

Der Schwerpunkt des Projekts liegt auf dem Thema Cloud Computing und der Entwicklung einer Anwendung auf der Azure Cloud. Dabei werden die verschiedenen Azure Cloud Service-Modelle IaaS, PaaS und SaaS berücksichtigt und genutzt.

Azure Cloud ist eine öffentliche Cloud-Plattform von Microsoft, die eine breite Palette von Cloud-Diensten wie Computing, Storage und Networking anbietet. Die Plattform ermöglicht es Unternehmen und Entwicklern, ihre Anwendungen und Dienste in der Cloud zu hosten und zu verwalten, ohne sich um die zugrunde liegende Infrastruktur kümmern zu müssen.

IaaS (Infrastructure as a Service) ermöglicht die Erstellung virtueller Maschinen in der Cloud, auf denen Anwendungen und Dienste ausgeführt werden können. Der Nutzer ist für die Konfiguration und Wartung der virtuellen Maschinen verantwortlich.

PaaS (Platform as a Service) ist eine höhere Stufe des Cloud-Computing-Modells, bei dem der Nutzer nur Anwendungen und Dienste bereitstellt und sich nicht um die zugrunde liegende Infrastruktur kümmern muss. In diesem Fall verwende ich PaaS um eine Datenbank bereitzustellen. Die Verwendung von PaaS-Datenbanken in Azure ermöglicht es Entwicklern, Datenbanken schnell und einfach zu erstellen und zu verwalten, ohne sich um die Wartung und Skalierung der zugrunde liegenden Infrastruktur kümmern zu müssen.

SaaS (Software as a Service), in unserem Fall der Azure Cognitive Services Translator, ist eine Anwendung, die über die Cloud bereitgestellt wird und vom Benutzer genutzt werden kann, ohne dass dieser die zugrunde liegende Infrastruktur und die Wartung der Anwendung selbst verwalten muss.

Die Umsetzung von Projekten im Themenbereich Cloud Computing mit Azure erfordert in der Regel den Einsatz von Infrastruktur-Service-Tools **wie Ansible und Terraform**.

Ansible ist ein Open-Source-Tool zur Automatisierung von IT-Workflows, das die Bereitstellung, Konfiguration und Verwaltung von Anwendungen und Systemen in der Cloud erleichtert. Mit Ansible können Entwickler und IT-Administratoren die Infrastruktur in Azure automatisch konfigurieren und skalieren.

Terraform ist ein weiteres Open-Source-Tool, das eine Infrastruktur-Code-Plattform darstellt. Es ermöglicht die Erstellung, Änderung und Versionierung von Infrastruktur sicher und effizient. Terraform bietet Unterstützung für eine Vielzahl von Cloud-Plattformen und kann auch in Azure eingesetzt werden, um die Konfiguration und Verwaltung von Infrastruktur-Service-Tools wie SaaS, IaaS und PaaS zu erleichtern.

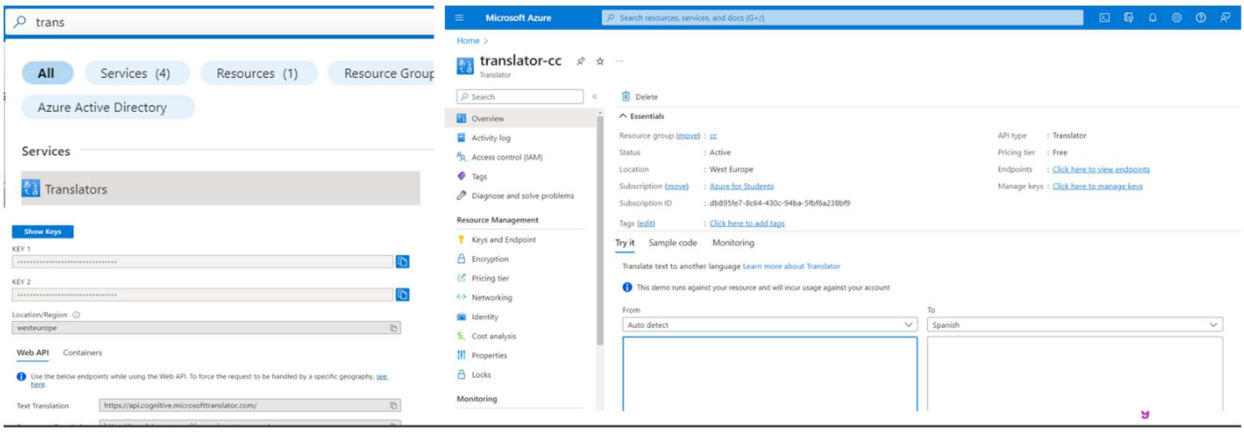
Durch den Einsatz von Ansible und Terraform können IT-Administratoren und Entwickler die Effizienz und Skalierbarkeit von Cloud-Computing-Projekten in Azure verbessern. Diese Tools automatisieren die Bereitstellung und Verwaltung von Anwendungen und Systemen in der Cloud, wodurch Entwickler und IT-Administratoren Zeit und Ressourcen sparen können.

Die Webanwendung wurde als einfache **Node.js-Anwendung** entwickelt, die es ermöglicht, sowohl das Frontend als auch das Backend mit einer Programmiersprache zu erstellen.

Implementierung

1. Azure Translate

Der Azure Translate (SaaS) wurde über das Azure Portal manuell bereitstellt



Die Umsetzung des Projekts erfolgt größtenteils vollständig automatisiert mit **Azure DevOps.**

Azure DevOps ist eine umfassende Plattform, die verschiedene Werkzeuge und Dienste für die Entwicklung von Anwendungen in der Cloud bereitstellt. Sie umfasst Tools für die Verwaltung von Quellcode, Projektmanagement, Build- und Release-Management, Testautomatisierung und vieles mehr.

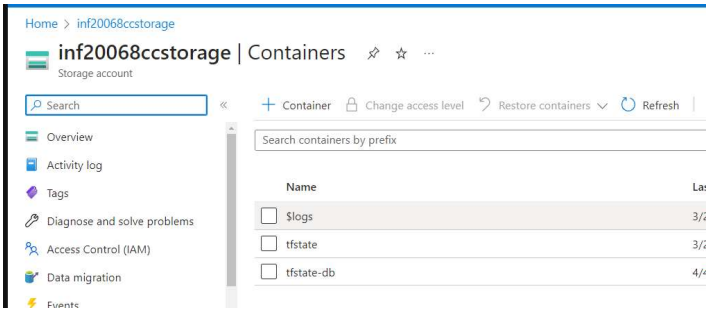
In unserem Fall wurde Azure DevOps zusammen mit **Azure Repos** und **Pipelines** verwendet, um die Infrastruktur mit Terraform und Infrastructure as Code vollständig zu automatisieren. Durch die Verwendung von Pipelines in Azure DevOps können wir alle Schritte, die für die Bereitstellung und Verwaltung unserer Anwendung erforderlich sind (also die Infrastruktur etc.), automatisieren und so menschliche Fehler minimieren.

Vorbedingung Terraform und (Pipelines):

In Azure DevOps Pipeline und Terraform ist es wichtig, das Statefile in einem Azure Storage Account zu speichern, da das Statefile eine wichtige Rolle bei der Verwaltung der Infrastruktur spielt. Der Statefile enthält Informationen über die aktuelle Konfiguration der Infrastruktur und alle Änderungen, die an der Infrastruktur vorgenommen wurden.

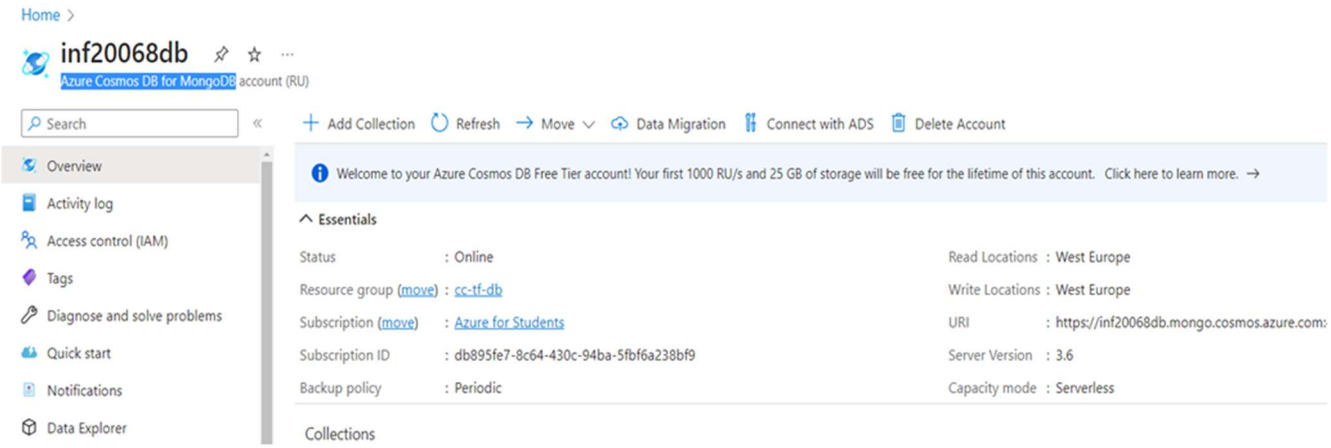
Wenn der Statefile nicht in einem Azure Storage Account gespeichert wird, kann dies zu unerwarteten Ergebnissen führen, da der Statefile möglicherweise nicht immer konsistent oder korrekt ist. Wenn beispielsweise zwei Pipeline-Läufe gleichzeitig Änderungen an der Infrastruktur vornehmen, kann es zu Konflikten kommen, da jede Pipeline ihre eigene Version des Statefiles hat. Dies kann dazu führen, dass Änderungen überschrieben werden oder verloren gehen und die Infrastruktur instabil wird.

Durch die Speicherung des Statefiles in einem Azure Storage Account wird sichergestellt, dass alle Pipelines auf die gleiche Version des Statefiles zugreifen und somit die Konsistenz und Korrektheit der Infrastruktur gewährleistet ist.



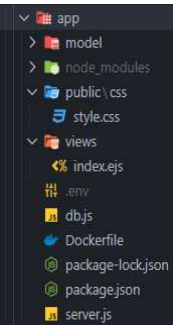
2. Datenbank CosmosDB

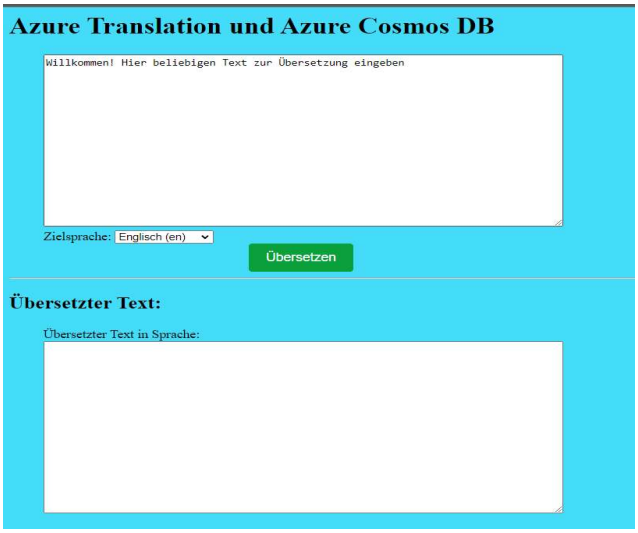
Als Datenbank wurde die auf Mongo/NoSql basierende Azure Cosmos DB gewählt, die mit Terraform und über eine Pipeline bereitgestellt wird.
Die Pipeline (cloud-computing-2\pipelines\database-terraform-pipelines.yml) welche die Terraform Skripte im Ordner *cloud-computing-2\database* ausführt, hat wie rechts im Bild zu sehen und markiert, Terraform mit der Statefile im Storageaccount konfiguriert.
Diese Pipeline ist so konzipiert, dass sie vor der Terraform-Ausführung die Skripte validiert und bei erfolgreicher Validierung die Ressourcen (Azure Cosmos DB) mit Terraform Plan und Terraform Apply in der Azure Cloud bereitstellt.



3. APP

Die mit NodeJS entwickelte APP verfügt über ein einfaches Frontend. In einem einfachen Textfeld wird der zu übersetzende Text eingegeben (die Sprache wird automatisch erkannt), anschließend kann aus einem Drop-Down-Menü die Zielsprache ausgewählt werden und mit einem Klick auf einen Button wird die Übersetzungsanfrage abgeschickt. Der Programmcode der Web-Anwendung ist im Ordner *cloud-computing-2* zu finden





Sicherstellung, dass keine doppelte Einträge in Datenabank landen:

API Key und Datenbank-Zugangsdaten:

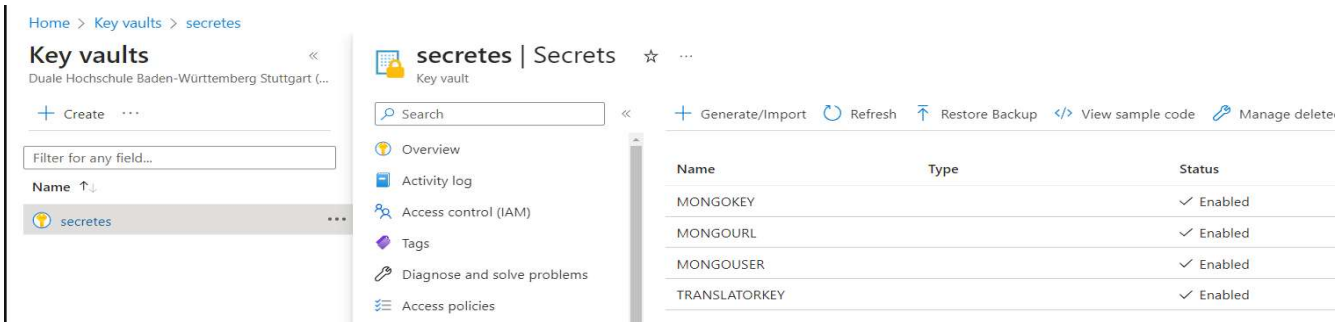
Der bei der Anfrage an den Translate Service benötigte Api Key und die Datenbankzugangsdaten werden im Backend benötigt und müssen als Secret (d.h. geschützte private Daten) behandelt werden. Damit diese nicht hardcoded im Quellcode stehen, werden sie in eine .env ausgelagert**.**

```
16  const translatorKey = process.env.TRANSLATOR_KEY;
17
```

```
6  const mongooseurl = 'mongodb://' + process.env.MONGO_USER +
7
```

```
1  TRANSLATOR_KEY = 9b30ac
2  MONGO_USER = inf20068db
3  MONGO_URL = inf20068db.mongo.cosmos.azure.com:10255
4  MONGO_KEY = EkxvSfS
```

Diese .env Datei darf jedoch nicht in die Versionsverwaltung (Azure Repo) gepusht werden. Der Inhalt dieser Datei (also die Secrets) werden mit Azure Key Vault sicher in der Azure Cloud gespeichert.



Bereitstellung der APP über Docker (CI/CD-Pipeline):

Nachdem die Secrets sicher in Azure Vault ausgelagert wurden, kann das Deployment der Anwendung über eine CD/CD-Pipeline erfolgen. Dazu muss ein Dockerfile (zu finden im Pfad cloud-computing-2/app/Dockerfile) und eine Pipeline (cloud-computing-2\pipelines\build-app-pipeline.yml) erstellt werden.

build-app-pipeline.yml

Contents History Compare Blame

```
10  branches:
11    include:
12    - main
13
14
15  pool:
16    name: default
17
18  jobs:
19  - job: BuildAndConfiure
20    displayName: Build and Configure translator APP
21    steps:
22
23      - task: AzureKeyVault@2
24        inputs:
25          azureSubscription: 'Azure-cc-Connection'
26          KeyVaultName: 'secretes'
27          SecretsFilter: '*'
28          RunAsPreJob: true
29      - task: CmdLine@2
30        displayName: Write Translator API-Key in .env
31        inputs:
32          script: 'touch app/.env && echo $(TRANSLATORKEY) > app/.env'
33
34
35      - task: CmdLine@2
36        displayName: Write Mongo-URL in .env
37        inputs:
38          script: 'echo $(MONGOURL) >> app/.env && echo $(MONGOUSER) >> app/.env && echo $(MONGOKEY) >> app/.env'
39
40      - task: Docker@2
41        inputs:
42          containerRegistry: 'docker'
43          command: 'login'
44      - task: Docker@2
45        inputs:
46          containerRegistry: 'docker'
47          repository: 'tomisboy/tomis
48          command: 'buildAndPush'
49          Dockerfile: '**/Dockerfile'
50          tags: 'cctranslator'
```

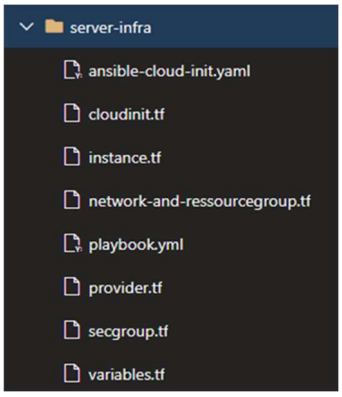
Die **CI/CD Pipeline**, die die Anwendung als Docker Container zur Verfügung stellt, kann wie folgt beschrieben werden:

Die Pipeline dient der automatisierten Bereitstellung und Konfiguration einer Anwendung namens "cctranslator" auf der Docker-Hub-Plattform. Dabei wird der Quellcode der Anwendung aus dem "app"-Ordner des Repositories in den Docker-Container gebaut und anschließend in das private Docker-Repository "tomisboy/tomisboy" gepusht. Vor dem Bauen der Anwendung werden im Schritt "AzureKeyVault" geheime Schlüssel aus dem Azure

Key Vault gelesen und im Schritt "CmdLine" werden die Werte für die Umgebungsvariablen der Anwendung in die Datei ".env" geschrieben. Diese Werte umfassen unter anderem den API-Schlüssel für die Verwendung von Azure Translator API und die Zugangsdaten für den MongoDB-Datenbankzugriff. Die Pipeline wird bei jeder Änderung im "app"-Ordner und auf dem Branch "main" ausgeführt.

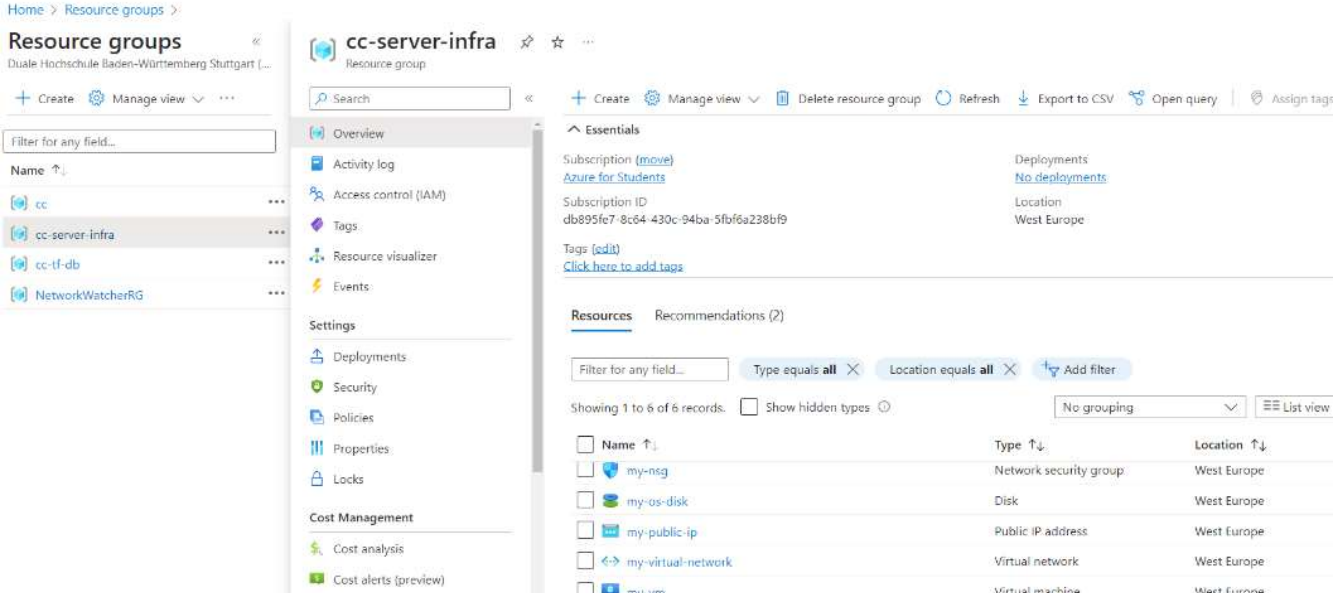
4. Server-Infrastruktur

Wie bei der Datenbank in Punkt 2 erfolgt die Bereitstellung der Infrastruktur ebenfalls über eine Pipeline (\cloud-computing-2\pipelines\terraform-pipelines.yml), die die Terraform-Skripte im Ordner (cloud-computing-2\server-infra)ausführt



- .network-and-ressourcegroup.tf
  Erstellt Netzwerk und Ressourcen-Gruppe für Infrastruktur
- Secgroup.tf
  Erstellt Firewall / Portöffnung auf Port 5000 für NodeJS Anwendung
- Instance.tf
  Erstellt die eigentliche (Linux)-Virtuelle Maschine
- Instance.tf cloudinit.tf und ansible-cloud-init.yaml.
  Führt Cloud-init Script aus welches Ansible und Docker auf Server installiert

Nach Ausführung der Pipeline stehen die Ressourcen in Azure zur Verfügung.



5. Ausrollen und Ausführen der Anwendung

SKU	: Basic
Tier	: Regional
IP address	: 52.174.181.242
DNS name	: -
Associated to	: my-nic
Virtual machine	: my-vm
Routing preference	: Microsoft network

Zu diesem Zeitpunkt ist die Anwendung als Docker-Container im Docker-Hub vollständig einsatzbereit. Mit der erhaltenen IP-Adresse kann über SSH auf den mit Terraform erstellten Server zugegriffen werden. Ansible wird über das Cloud-init-Skript (cloud-computing-2\server-infra\ansible-cloud-init.yaml) bei Erstellung der Instanz mit Terraform auf diese gleich mit installiert.

Im nächsten Schritt werden die Zugangsdaten für das private Repository in Ansible Vault über ein Passwort gespeichert, so dass auf diese zugegriffen werden kann, wenn der Docker-Container über das Playbook (cloud-computing-2\server-infra\playbook.yml) heruntergeladen und auf dem Server ausgeführt wird.

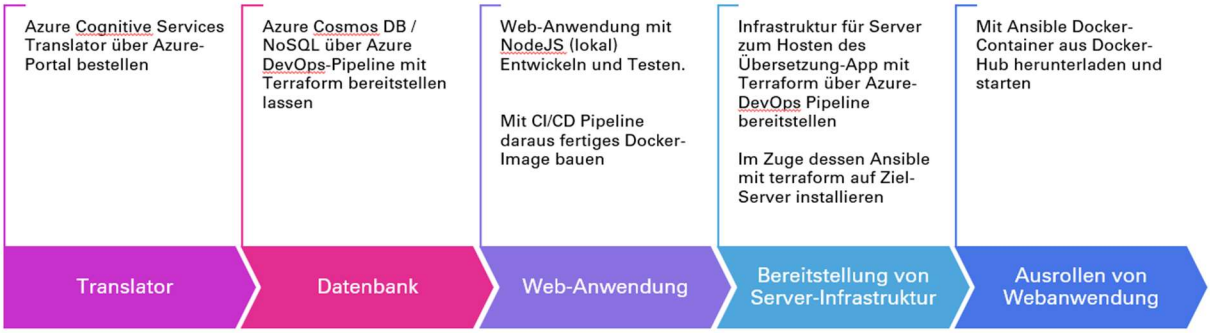
```
ansible-vault create var.yml
```

Nach dem mit `ansible-playbook playbook.yml --ask-vault-pass`

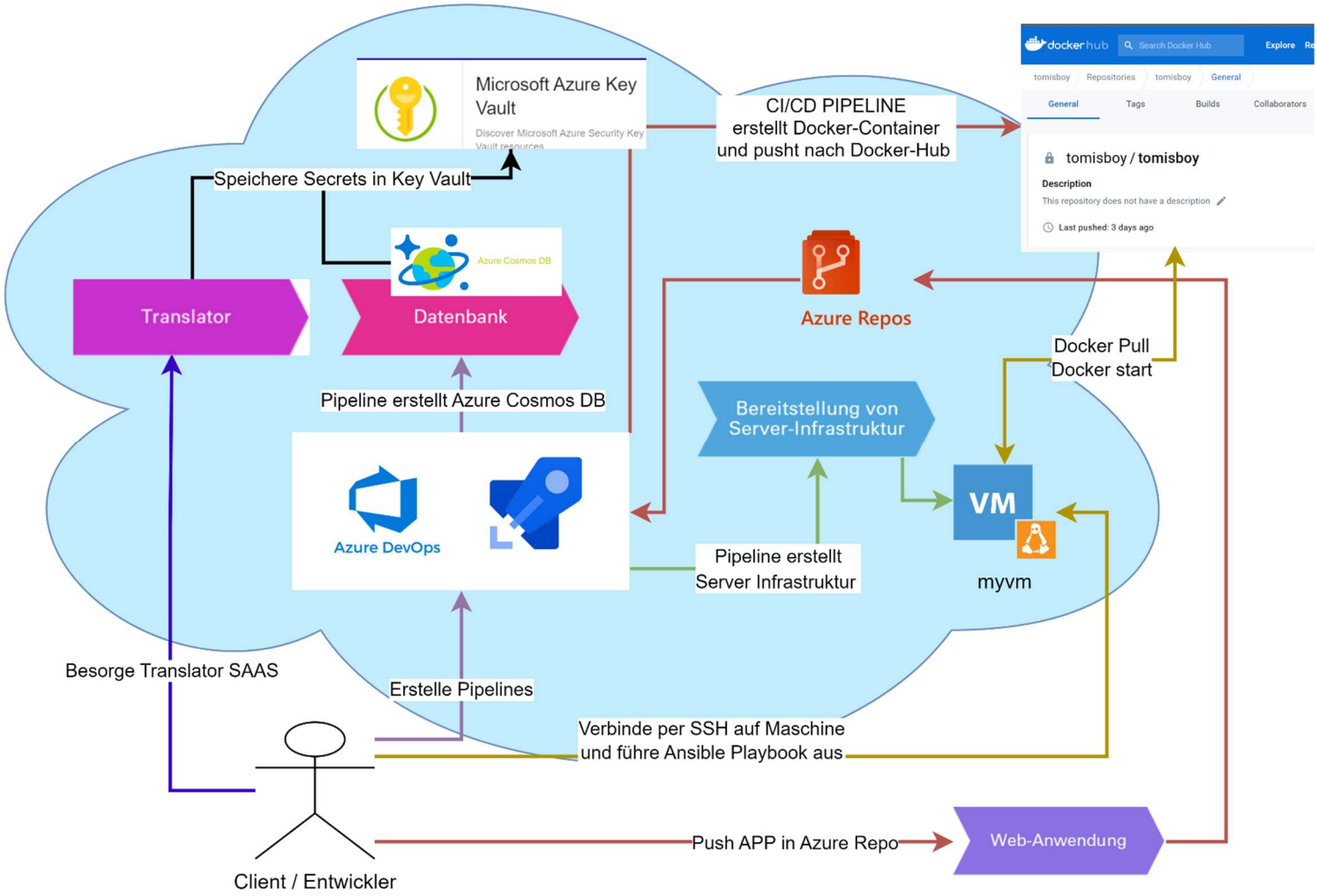
das Playbook ausgeführt wurde steht der Übersetzer über die URL http://52.174.181.242:5000/ zu Verfügung:

```
ubuntu@my-vm:~$ cat playbook.yml
- name: Start Docker Container with Port 5000
  connection: local
  hosts: localhost
  become: true
  vars_files:
    - var.yml

  tasks:
    - name: Log in to Docker registry
      docker_login:
        username: "{{ docker_username }}"
        password: "{{ docker_password }}"

    - name: Delete old Docker Container
      docker_container:
        name: cctranslator
        image: tomisboy/tomisboy:cctranslator
        state: absent

    - name: Pull Docker Image
      docker_image:
        name: tomisboy/tomisboy
        tag: cctranslator
        source: pull
        force_source: yes

    - name: Start Docker Container
      docker_container:
        name: cctranslator
        image: tomisboy/tomisboy:cctranslator
        state: started
        restart_policy: always
        published_ports:
          - "5000:5000"
```

Schaubild und Zusammenfassung



API-Key und Azure Cosmos DB Zugangsdaten als Azure Secret speichern



Fazit / Diskussion

Das Projekt kann als voller Erfolg gewertet werden, alle Anforderungen wurden erfüllt.

Mit dieser Lösung konnte eine flexible Möglichkeit geschaffen werden, wie eine Anwendung in der Praxis ausgerollt werden kann.

Der Einsatz einer CI/CD Pipeline, die uns eine automatisierte Erfahrung beim Ausrollen einer Anwendung in die Cloud bietet, wird in der Praxis immer häufiger eingesetzt.

Darüber hinaus setzen immer mehr Unternehmen in der Praxis (sofern sie auf Azure sind) auf die All-in-One-Lösung Azure Dev Ops, die in diesem Projekt erfolgreich getestet und eingesetzt wurde.