PRAXISPROJEKT

ÜBERSETZUNGS-APP

MIT AZURE COGNITIVE SERVICES TRANSLATOR

Cloud Computing
Thomas Alpert

AGENDA

- Anforderungsanalyse
- Beschreibung der Entwicklungsumgebung und Umsetzung
- Translator API und Datenbank
- Entwicklung der Web-Anwendung
- Bereitstellung von Server-Infrastruktur in Azure
- Ausrollen der Web-Anwendung auf Server und Demo

ANFORDERUNGSANALYSE

Anforderungsanalyse

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

BESCHREIBUNG DER ENTWICKLUNGSUMGEBUNG UND UMSETZUNG

Beschreibung der Entwicklungsumgebung und Umsetzung

Entwicklung Web-Anwendung

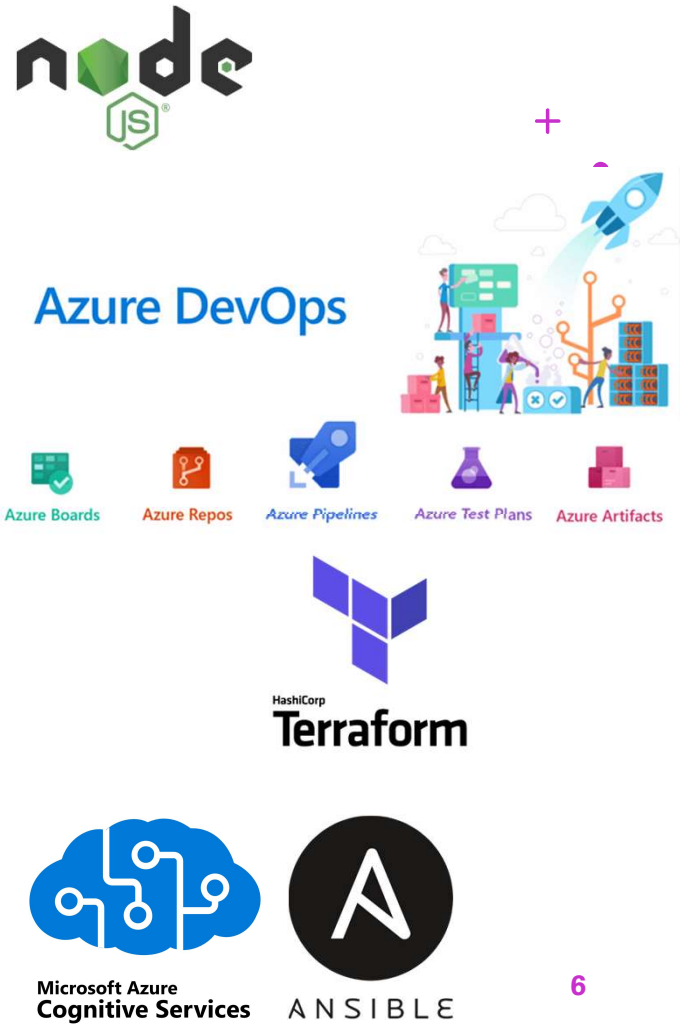
- **NodeJS als Docker-Container**

Automatisierte Infrastruktur-Steuerung:

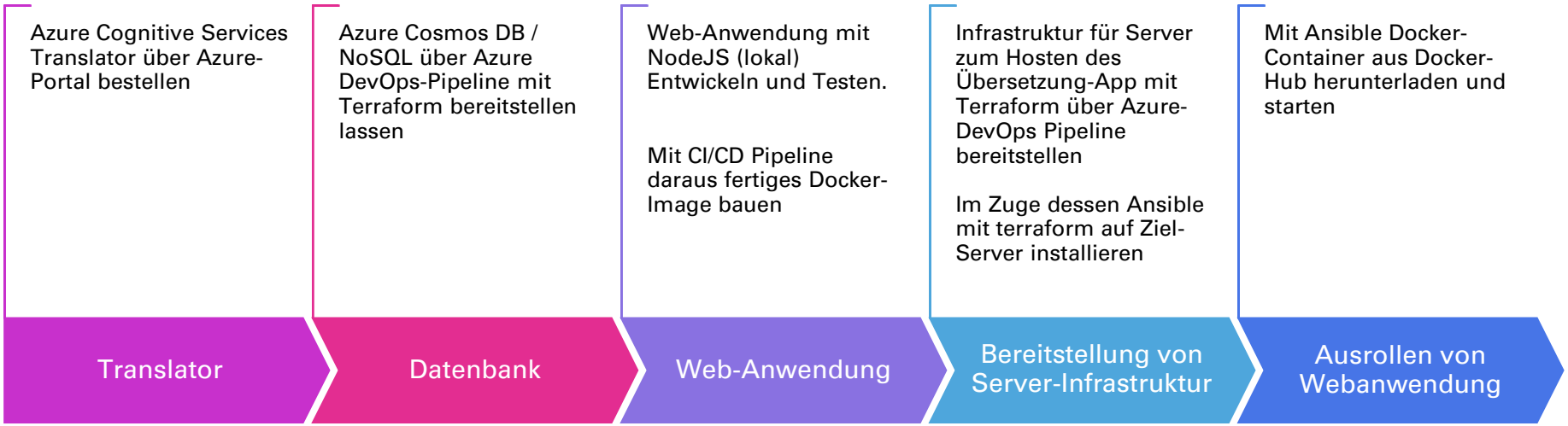
- **Azure DevOps Server**
- **Azure Repo**
- **Bereitstellung von Infrastruktur mit Terraform Pipelines**
  - Azure Cosmos DB
  - Server für Hosting der Webanwendung
- **CI/CD Pipeline**
  - zur Bildung der Anwendung als Docker-Container
  - Stellt Docker-Container in privates Docker-Hub zur Verfügung

Manuelle Prozesse:

- **Erstellen von Storage-Account (terraform-Statefile)**
- **Azure Cognitive Services Translator besorgen**
- **Starten der Docker-Container über Ansible auf Server**
- **API-Keys/DB-Zugangsdaten als Secrets in Azure speichern**



Ablaufplan



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

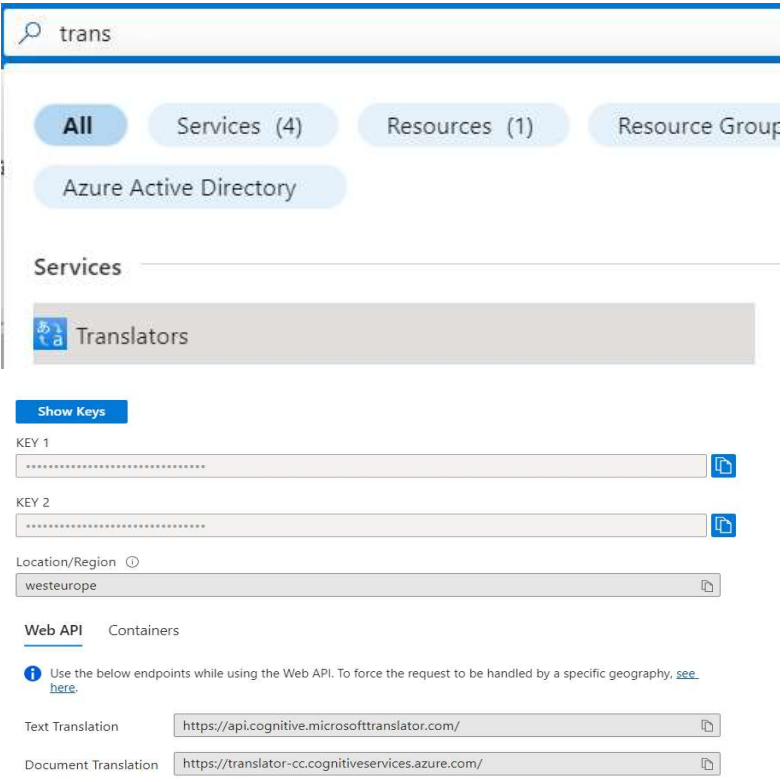
TRANSLATOR API UND DATENBANK

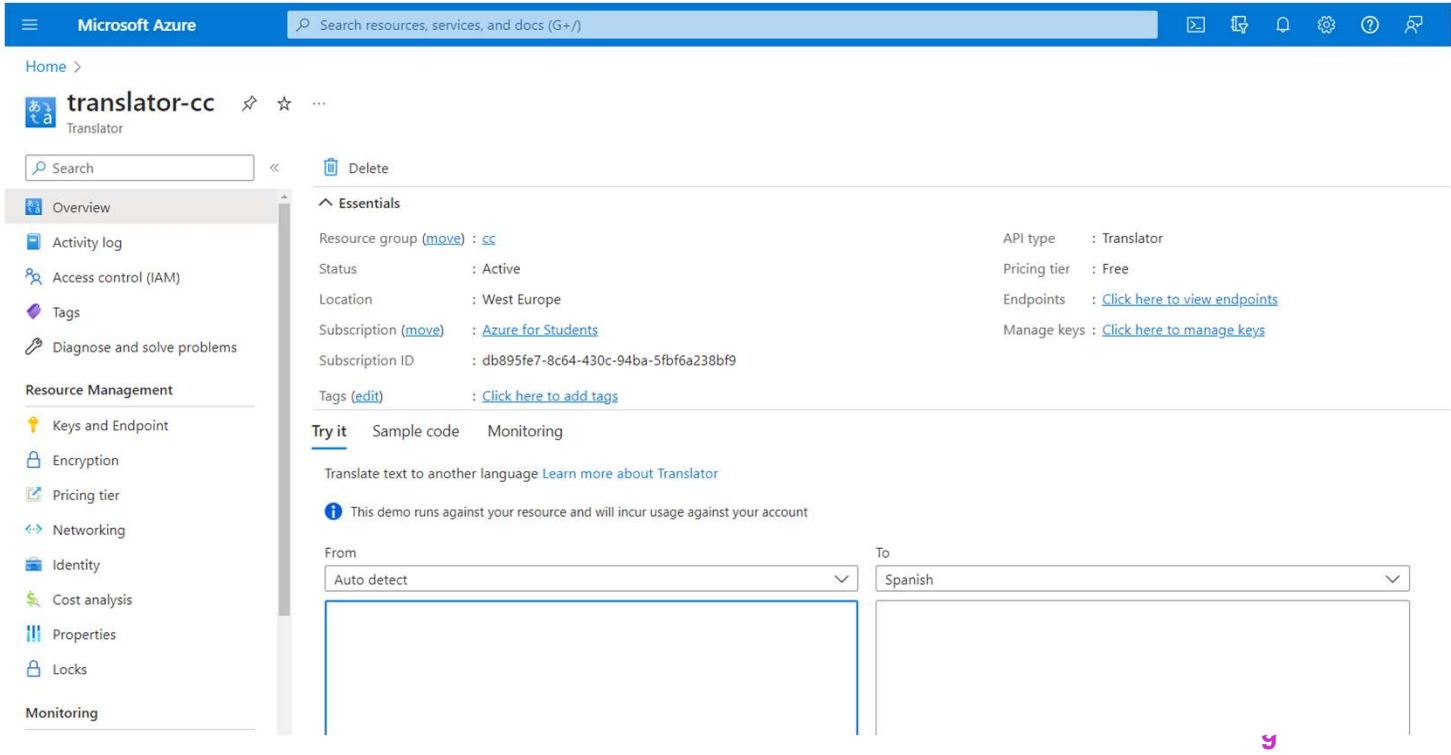
Translator API

Translator

Azure Cognitive Services Translator über Azure-Portal bestellen

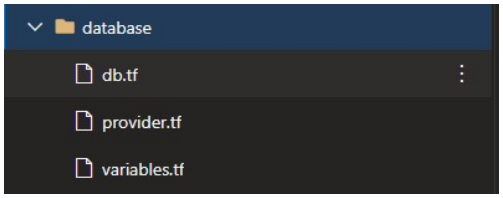
- Einfache Bereitstellung über Azure Portal Azure Cognitive Services Translator
  - API und passender API-Key wird erstellt





Azure Cosmos DB

- Terraform Konfiguration



variables.tf

Contents History Compare Blame

```hcl
# Define variables
variable "resource_group_name" {
  type        = string
  description = "The name of the backend storage account resource group"
  default     = "cc-tf-db"
}

variable "name" {
  type    = string
  default = "inf20068db"
}

variable "location" {
  type    = string
  default = "westeurope"
}

variable "location_name" {
  type    = string
  default = "West Europe"
}

variable "default_experience" {
  type    = string
  default = "Azure Cosmos DB for MongoDB API"
}

variable "is_zone_redundant" {
  type    = bool
  default = false
}
```

db.tf

Contents History Compare Blame

```hcl
resource "azurerm_resource_group" "example" {
  # Name der Resource Group
  name     = var.resource_group_name
  # Standort der Resource Group
  location = var.location
}

# Define resource block for Cosmos DB account
resource "azurerm_cosmosdb_account" "example" {
  name                = var.name
  resource_group_name = var.resource_group_name
  location            = var.location
  offer_type          = "Standard"
  kind                = "MongoDB"
  is_virtual_network_filter_enabled = false
  enable_free_tier    = true
  capacity {
    total_throughput_limit = 1000
  }

  consistency_policy {
    consistency_level = "Session"
  }

  geo_location {
    location          = "${var.location_name}"
    failover_priority = 0
  }

  capabilities {
    name = "EnableMongo"
  }

  capabilities {
    name = "DisableRateLimitingResponses"
  }

  capabilities {
    name = "EnableServerless"
  }


  tags = {
    defaultExperience       = var.default_experience
    "hidden-cosmos-mmspecial" = ""
  }

  enable_automatic_failover = var.is_zone_redundant
}
```

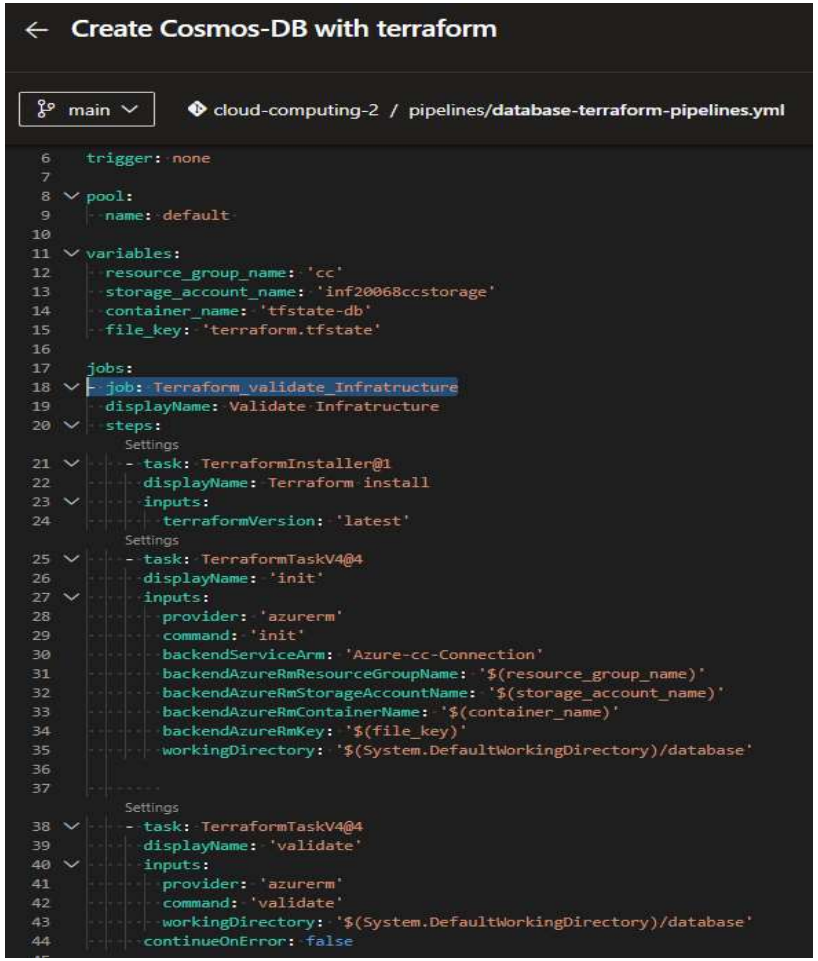
Azure Cosmos DB / NoSQL über Azure DevOps-Pipeline mit Terraform bereitstellen lassen

Azure Cosmos DB

Translator | Datenbank

Azure Cosmos DB / NoSQL über Azure DevOps-Pipeline mit Terraform bereitstellen lassen

- Azure Cosmos DB für MongoDB über Terraform Pipeline erstellt
  - Pipeline mit 2 Jobs
- 1. Job Validierung der Terraform-Skripte

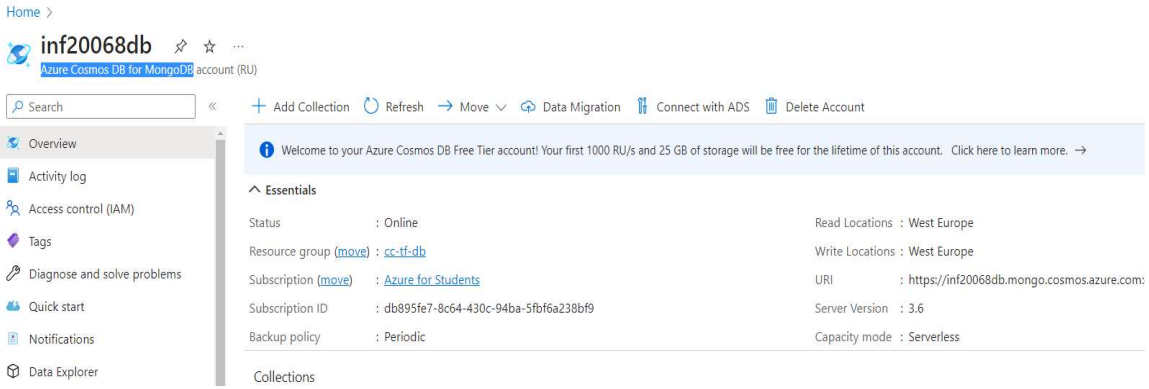


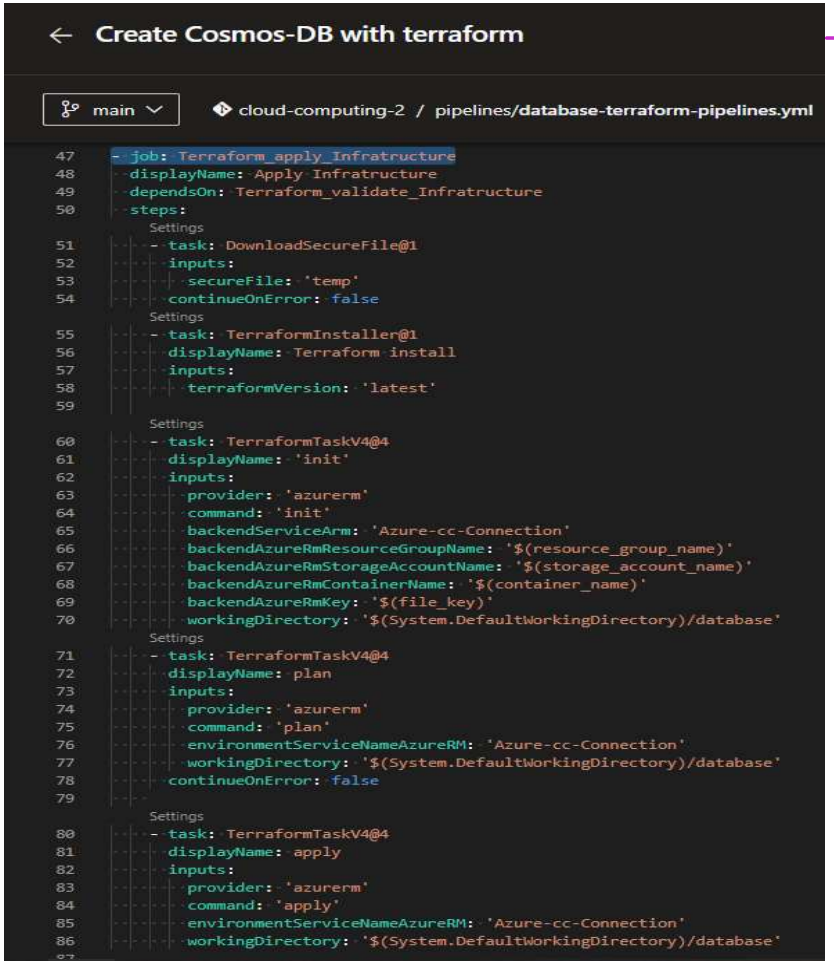
Azure Cosmos DB

Translator | Datenbank

Azure Cosmos DB / NoSQL über Azure DevOps-Pipeline mit Terraform bereitstellen lassen

- Azure Cosmos DB für MongoDB über Terraform Pipeline erstellt
  - Pipeline mit 2 Jobs
- 2. Job Terraform-Skripte ausführen





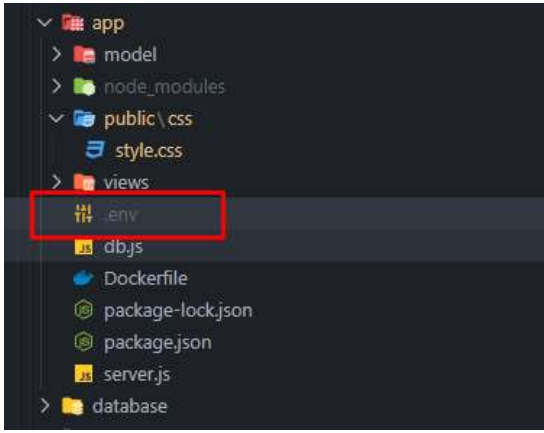
ENTWICKLUNG DER WEB-ANWENDUNG

Web-Anwendung

Web-Anwendung mit NodeJS (Lokal Entwickeln und Testen.

Mit CI/CD Pipeline daraus fertiges Docker-Image bauen

- Web-Anwendung für die Übersetzung mit Azure Cognitive Services Translator mit NodeJS lokal entwickeln.
- Secrets wie API-Key und DB-Zugangsdaten in .env auslagern.
- Diese Datei nicht in GitHub/Azure Repos pushen




```
const translatorKey = process.env.TRANSLATOR_KEY;
// Translate Function
async function translateText(text, endpoint) {

    const response = await axios.post(endpoint, [{ 'Text': text }], {
        headers: {
            'Content-Type': 'application/json',
            'Ocp-Apim-Subscription-Key': translatorKey,
            'Ocp-Apim-Subscription-Region': 'westeurope'
        }
    });
    return response.data[0].translations[0].text;
}
```

```
const crypto = require('crypto');
const db = require('./model/text.js');
const mongooseurl = 'mongodb://' + process.env.MONGO_USER + ':' + process.env.MONGO_KEY + '@' + process.env.MONGO_URL

mongoose
  .connect(mongooseurl, {
    useNewUrlParser: true,
    useUnifiedTopology: true
  })
  .then(() => console.log("Database connected! \n "))
  .catch(err => console.log(err));
```

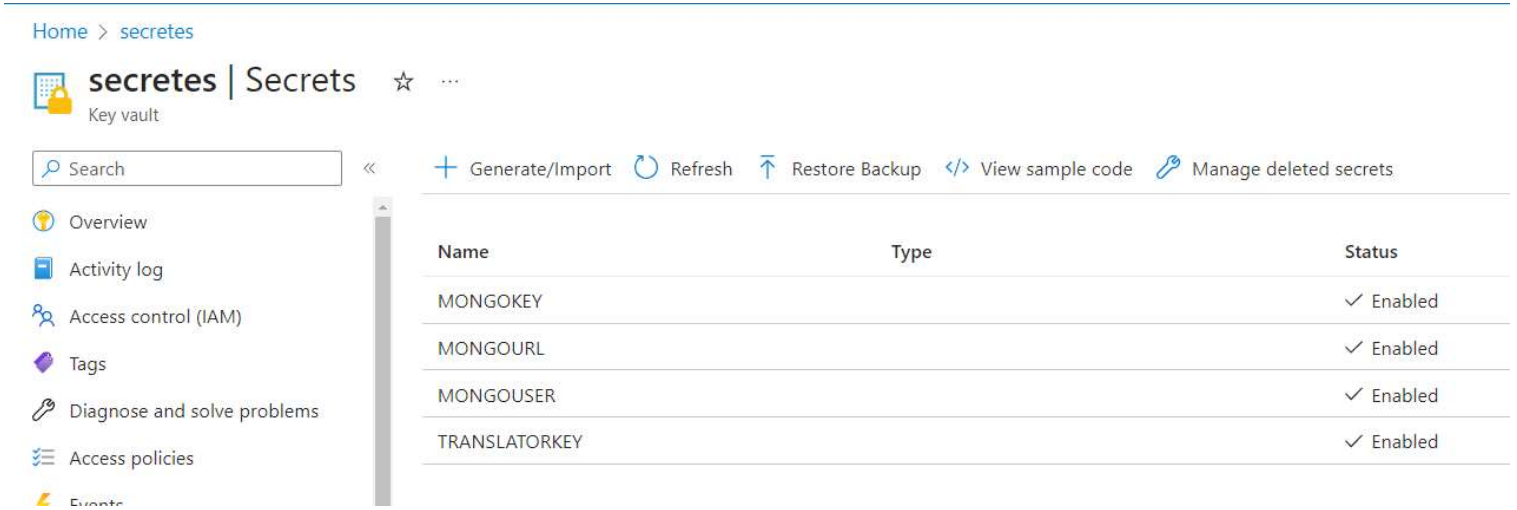
Web-Anwendung

Web-Anwendung mit NodeJS (Lokal Entwickeln und Testen.

Mit CI/CD Pipeline daraus fertiges Docker-Image bauen

- Web-Anwendung mit CI/CI Pipeline als Docker-Container Bereitstellen

- Die Secrets aus der .env Datei müssen in Azure importiert werden



Web-Anwendung CI/CD

Translator | Datenbank | Web-Anwendung

- Bei Erstellung des Containers werden die Secrets aus Azure in die .env Datei geschrieben

- Anschließend Pushen in privates Docker-Hub repo

CI-CD for Application CCTranslator docker build and push

main · cloud-computing-2 / pipelines/**build-app-pipeline.yml**

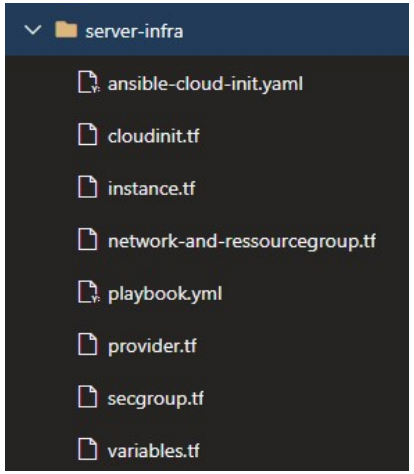
```yaml
pool:
  name: default

jobs:
- job: BuildAndConfiure
  displayName: Build and Configure translator APP
  steps:

    Settings
    - task: AzureKeyVault@2
      inputs:
        azureSubscription: 'Azure-cc-Connection'
        KeyVaultName: 'secretes'
        SecretsFilter: '*'
        RunAsPreJob: true
    Settings
    - task: CmdLine@2
      displayName: Write Translator API-Key in .env
      inputs:
        script: 'touch app/.env && echo $(TRANSLATORKEY) > app/.env'


    Settings
    - task: CmdLine@2
      displayName: Write Mongo-URL in .env
      inputs:
        script: 'echo $(MONGOURL) >> app/.env && echo $(MONGOUSER) >> app/.env && echo $(MONGOKEY) >> app/.env'

    Settings
    - task: Docker@2
      inputs:
        containerRegistry: 'docker'
        command: 'login'
    Settings
    - task: Docker@2
      inputs:
        containerRegistry: 'docker'
        repository: 'tomisboy/tomisboy'
        command: 'buildAndPush'
        Dockerfile: '**/Dockerfile'
        tags: 'cctranslator'
```

BEREITSTELLUNG VON SERVER-INFRASTRUKTUR IN AZURE

Server-Infrastruktur

Infrastruktur für Server zum Hosten des Übersetzung-App mit terraform über Azure-DevOps Pipeline bereitstellen

- Terraform Konfiguration



- Aufteilung der Schritte in einzelne Terraform Skripte

- 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

- cloudinit.tf und ansible-cloud-init.yaml
  - Führt Cloud-init Script aus welches Ansible und Docker auf Server installiert

Server-Infrastruktur

Infrastruktur für Server zum Hosten des Übersetzung-App mit terraform über Azure-DevOps Pipeline bereitstellen

- Pipeline mit 2 Jobs
- 1. Job Validierung der Terraform-Skripte

Create Server-Infrastructure with terraform

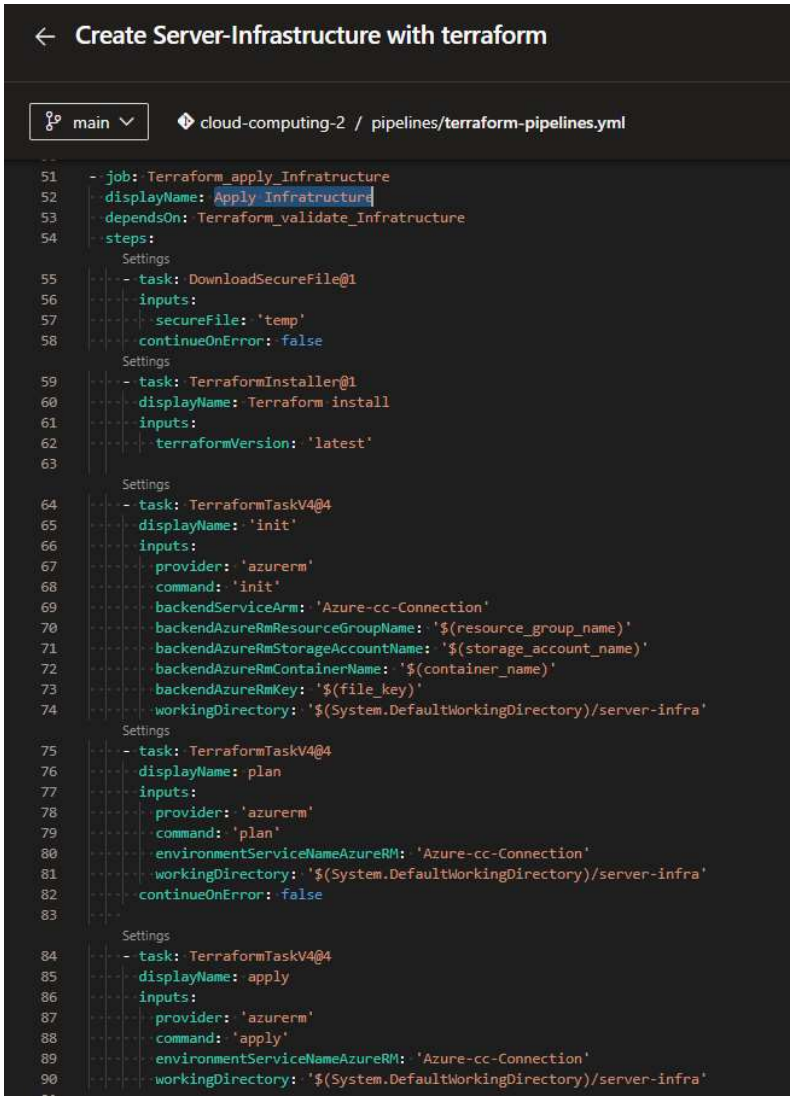
main | cloud-computing-2 / pipelines/terraform-pipelines.yml

```yaml
pool:
  name: default

variables:
  resource_group_name: 'cc'
  storage_account_name: 'inf20068ccstorage'
  container_name: 'tfstate'
  file_key: 'terraform.tfstate'

jobs:
- job: Terraform_validate_Infratructure
  displayName: Validate Infratructure
  steps:
    Settings
    - task: DownloadSecureFile@1
      inputs:
        secureFile: 'temp'
      continueOnError: false
    Settings
    - task: TerraformInstaller@1
      displayName: Terraform install
      inputs:
        terraformVersion: 'latest'
    Settings
    - task: TerraformTaskV4@4
      displayName: 'init'
      inputs:
        provider: 'azurerm'
        command: 'init'
        backendServiceArm: 'Azure-cc-Connection'
        backendAzureRmResourceGroupName: '$(resource_group_name)'
        backendAzureRmStorageAccountName: '$(storage_account_name)'
        backendAzureRmContainerName: '$(container_name)'
        backendAzureRmKey: '$(file_key)'
        workingDirectory: '$(System.DefaultWorkingDirectory)/server-infra'

    Settings
    - task: TerraformTaskV4@4
      displayName: 'validate'
      inputs:
        provider: 'azurerm'
        command: 'validate'
        workingDirectory: '$(System.DefaultWorkingDirectory)/server-infra'
      continueOnError: false
```

Server-Infrastruktur

- Pipeline mit 2 Jobs

- 2. Job Apply der Terraform-Skripte



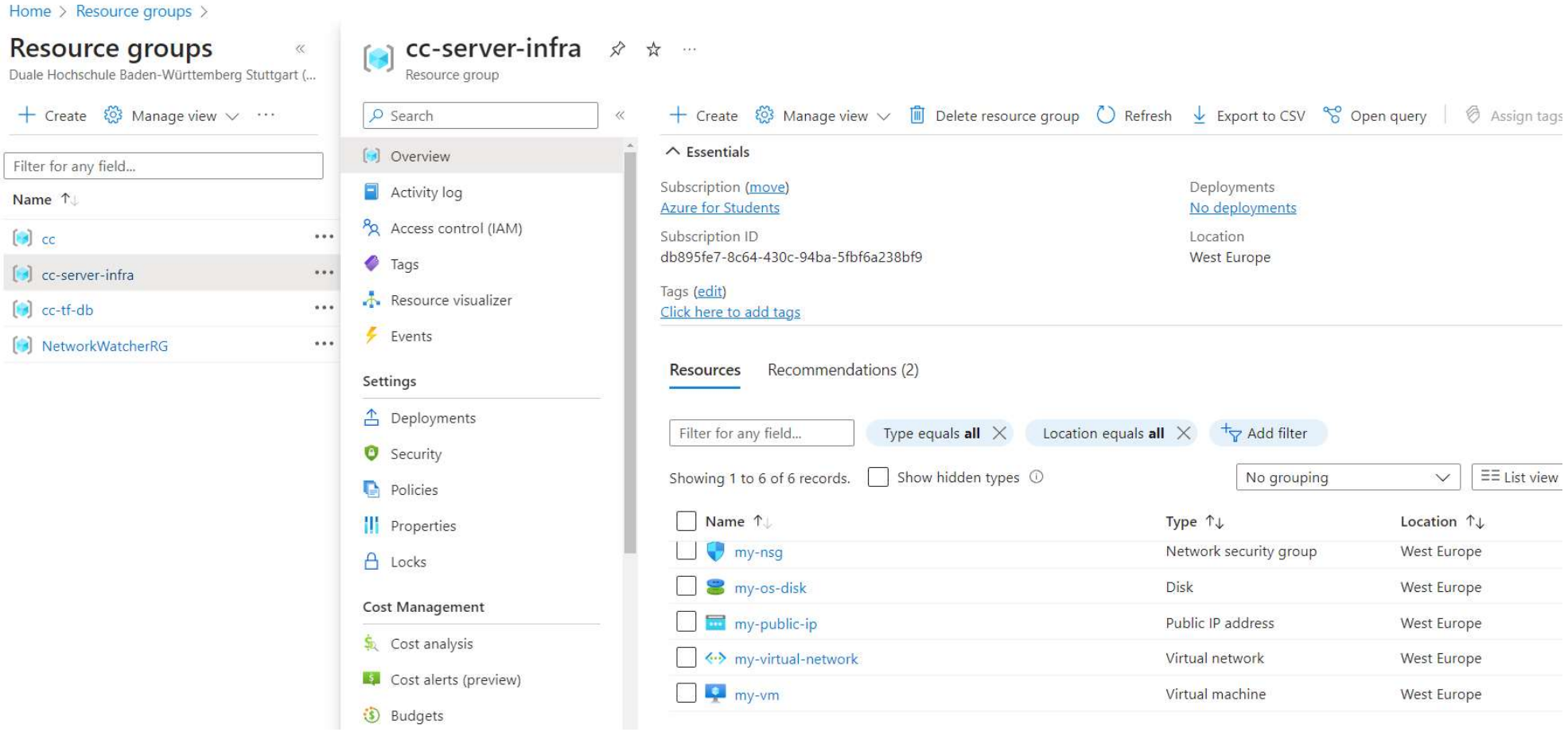
Infrastruktur für Server zum Hosten des Übersetzung-App mit terraform über Azure-DevOps Pipeline bereitstellen

Server-Infrastruktur

Translator | Datenbank | Web-Anwendung | Bereitstellung von Server-Infrastruktur

Infrastruktur für Server zum Hosten des Übersetzung-App mit terraform über Azure-DevOps Pipeline bereitstellen

- Zusammenfassung



- AUSROLLEN DER WEB-ANWENDUNG AUF SERVER

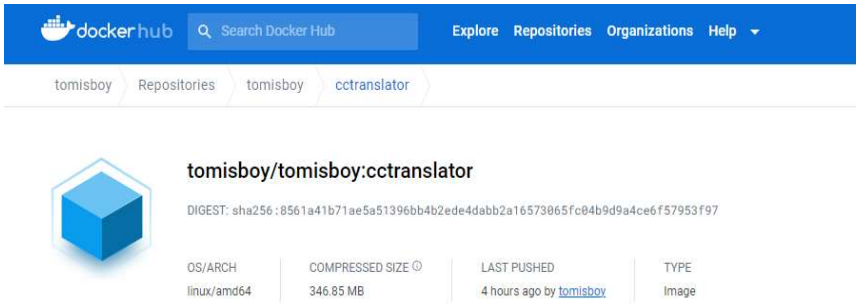
- DEMO

Ausrollen und Ausführen der Anwendung

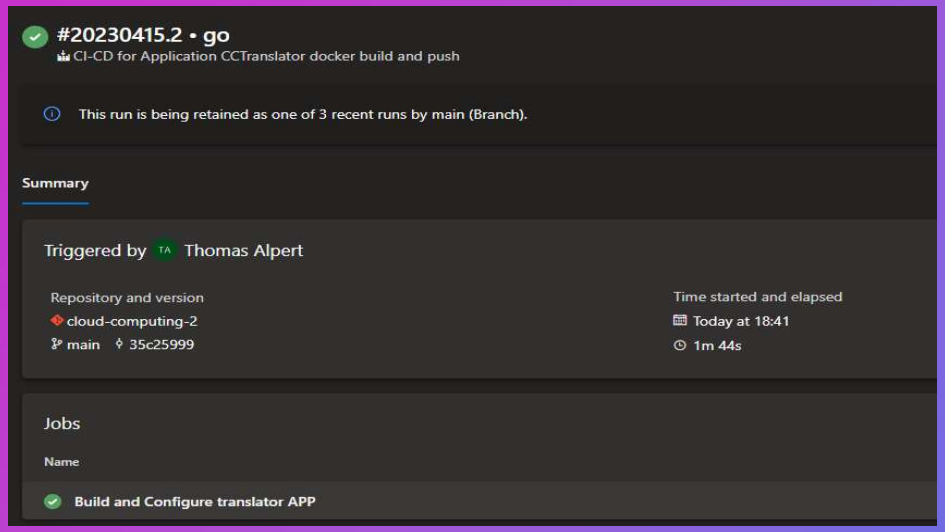
Translator | Datenbank | Web-Anwendung | Bereitstellung von Server-Infrastruktur | **Ausrollen von Webanwendung**

- Lokal entwickelte App auf Azure Repos pushen
- → CI/CD Pipeline stellt Anwendung als Docker-Container bereit
- Ansible: Zugangsdaten zu privatem Docker-Hub Repo als Secret in ansible-vault speichern

```
ubuntu@my-vm:~$ ansible-vault create var.yml
[DEPRECATION WARNING]: Ansible will require Python 3.
from ansible-core in version 2.12. Deprecation warnin
New Vault password:
Confirm New Vault password:
```



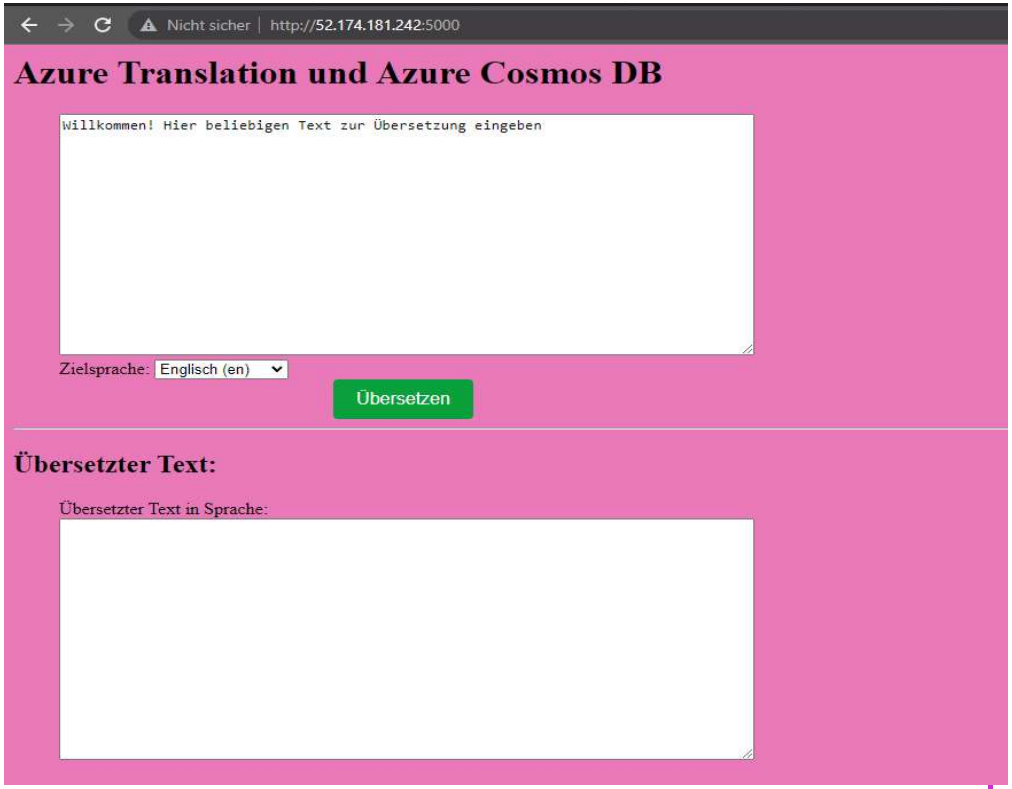

```
ubuntu@my-vm:~$ cat playbook.yml
- name: Start Docker Container with Port 5000
  connection: local
  hosts: localhost
  become: true
  vars_files:
    - var.yml

  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"
```




```
ubuntu@my-vm:~$ ansible-playbook playbook.yml --ask-vault-pass
[DEPRECATION WARNING]: Ansible will require Python 3.8 or newer on the controller starting with
from ansible-core in version 2.12. Deprecation warnings can be disabled by setting deprecation_w
Vault password:
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit loc

PLAY [Start Docker Container with Port 5000] ***********************************************

TASK [Gathering Facts] *********************************************************************
ok: [localhost]

TASK [Log in to Docker registry] ***********************************************************
ok: [localhost]

TASK [Delete old Docker Container] *********************************************************
changed: [localhost]

TASK [Pull Docker Image] *******************************************************************
ok: [localhost]

TASK [Start Docker Container] **************************************************************
changed: [localhost]

PLAY RECAP *********************************************************************************
localhost                  : ok=5    changed=2    unreachable=0    failed=0    skipped=0    resc

ubuntu@my-vm:~$
```

Demo

- Ansible Playbook starten
- URL Aufrufen



VIELEN DANK!

HashiCorp Terraform