

AUTOREN
LOÏS KAUFUNGEN UND MATTIA GALLICCHIO



mammatus - Technische Dokumentation

VIRTUELLE INFRASTRUKTUR

- GITLAB

Gitlab Deployment

Unsere internen System managen wir nicht auf dem gleichen Weg wie die virtuellen Maschinen von Kunden. Für das haben wir Gitlab Pipelines geschriebe, welche uns die Arbeit leichter machen und vor allem das wichtigste, nämlich "Config as Code" bieten.

Aufbau Gitlab Repositorys

Gruppe: Mammatus-cloud

- proxmox - für die Erstellung von Internen Virtuellen Maschinen und deren automatische Konfiguration.
- main-rp - Main Reverse Proxy
- mkdocs - diese Dokumentationsoberfläche
- admin-services - alle unseren interne Services nötig für die Cloud
- mammatus-backend - Unser Backendsystem für das erstellen von Kunden VMs

Proxmox

Wir erstellen hier neue VMs basierend auf einem Cloud-Init Template. Das Deployment läuft via Terraform und die anschliessende Konfiguration mittels unserm Anisble Playbook.

Ausgeführt wird das über das Standart Template von Gitlab. Mehr dazu im Repo.

Ablauf

- Verbindung mit dem Managed TF-Stage von Gitlab
- Init/validate/testing von Terraform
- Änderungen überprüfen --> TF Plan
- Anschliessend das Deployment mit Manuellem bestätigen.
 - Clone des Template und Konfiguration der Hardware.
 - Konfiguration durch Ansible
 - ausgabe der IP adressen

main-rp

Das Repo funktioniert gleich wie "admin-services" (weiter unten) ist einfach abgetrennt.

Konfiguration des Proxys mittels Caddyfile oder REST API.

mkdocs

Beschreibung gibt es im Repo. Kurz zusammen gefasst: mkdocs mit dem Material Design und vielen änderungen und plugins, fertig konfiguriert mit automatischem Deployment auf gitlab-pages. Zugriff mittels custom Domain von uns "docs.mammatus.ch". Dafür mussten einmal ein CNAME und ein TXT auf dem DNS Server hinterlegt werden.

Ansible Playbook

Beschreibung jeweils im Titel der tasks:

```

- become: yes
hosts: all

tasks:
- name: Install VIM
  apt:
    name: vim
    state: latest
    update_cache: true

- name: Install required system packages
  apt:
    pkg:
      - apt-transport-https
      - ca-certificates
      - curl
      - software-properties-common
      - python3-pip
      - virtualenv
      - python3-setuptools
    state: latest
    update_cache: true

- name: Add Docker GPG apt Key
  apt_key:
    url: https://download.docker.com/linux/ubuntu/gpg
    state: present

- name: Add Docker Repository
  apt_repository:
    repo: deb https://download.docker.com/linux/ubuntu focal stable
    state: present

- name: Update apt and install docker-ce
  apt:
    pkg:
      - docker-ce
      - docker-ce-cli
      - containerd.io
      - docker-buildx-plugin
      - docker-compose-plugin
    state: latest
    update_cache: true

- name: Create Docker Dir
  file:
    path: /docker
    state: directory
    owner: user
    mode: u=rwX,g=rX,o=rX
    recurse: yes

- name: adding existing user 'mammadmin' to group docker
  user:
    name: 'user'
    groups: docker
    append: yes

```

admin-services

.gitlab-ci.yml

```

.docker-deploy:
  stage: deploy
  # verbindung aufbau vi SSH auf den angegebenen Docker Host
  before_script:
    - 'command -v ssh-agent >/dev/null || ( apk add --update openssh )'
    - eval $(ssh-agent -s)
    - echo "$SSH_PRIVATE_KEY" | tr -d '\r' | ssh-add -
    - mkdir -p ~/.ssh
    - chmod 700 ~/.ssh
    - ssh-keyscan $SSH_TARGET >> ~/.ssh/known_hosts
    - chmod 644 ~/.ssh/known_hosts
  script:
    # Verbindung hergestellt
    - echo -e '\e[104mMAMMATUS Connecting to Server'
    - ssh $SSH_USER@$SSH_TARGET "hostname"
    - echo -e '\e[104mMAMMATUS Successfully Connected'
    # directory für neue app erstellen wen nötig
    - ssh $SSH_USER@$SSH_TARGET "mkdir -p /docker/$CI_JOB_NAME"
    # bestehende App stoppen
    - ssh $SSH_USER@$SSH_TARGET "docker compose -f /docker/$CI_JOB_NAME/docker-compose.yml down"
  || true"
    - ssh $SSH_USER@$SSH_TARGET "echo -e '\e[104mMAMMATUS $CI_JOB_NAME stopped'"
    # ersetzen der neuen Files
    - ssh $SSH_USER@$SSH_TARGET "rm -rf /docker/$CI_JOB_NAME/*"
    - scp -r apps/$CI_JOB_NAME/* $SSH_USER@$SSH_TARGET:/docker/$CI_JOB_NAME/
    - scp -r apps/$CI_JOB_NAME/./* $SSH_USER@$SSH_TARGET:/docker/$CI_JOB_NAME/
    - ssh $SSH_USER@$SSH_TARGET "ls /docker/$CI_JOB_NAME"
    # App wieder starten
    - ssh $SSH_USER@$SSH_TARGET "docker compose -f /docker/$CI_JOB_NAME/docker-compose.yml up -d"
    - ssh $SSH_USER@$SSH_TARGET "echo -e '\e[104mMAMMATUS $CI_JOB_NAME started'"
  tags:
    - docker-m

```

Mittels dieser Konfiguration wird im Repository für jede App eine Pipeline mit automatischem Deployment aufgebaut.

Neue App hinzufügen

```

# Einfach folgender block hinzufügen pro app
app-name:
  extends: .docker-deploy
  only:
    changes:
      - apps/app-name/**/*

```

achtung

Job name muss gleich sein wie der name des ordners!

mammatus-backend

Das Backend unseres VM Systems geschrieben in Python. Genaueres dazu in der Backend Doku.

