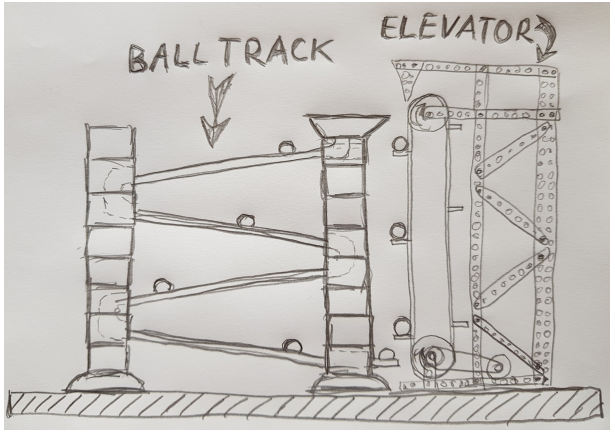
Automatisiertes IoT Flottenmanagement mit Git

Winterchaos 2022, Kriens

Matthias Lüscher

Über mich



- Langweilige Jobs gehören automatisiert:
  → Als Kind habe ich Kugelbahnen mit einem Motor ausgestattet.
  → Im Berufsleben versuche ich, repetitive Aufgaben durch CI/CD von mir fernzuhalten.
- Um mich weiterzubilden, habe ich mein eigenes Open Source Projekt edi gestartet.
- edi wird auch bei meiner beruflichen Tätigkeit eingesetzt.
- Die Freizeit verbringe ich am liebsten zusammen mit meiner Familie in der Natur.
- Kontakt: lueschem@gmail.com

Idee

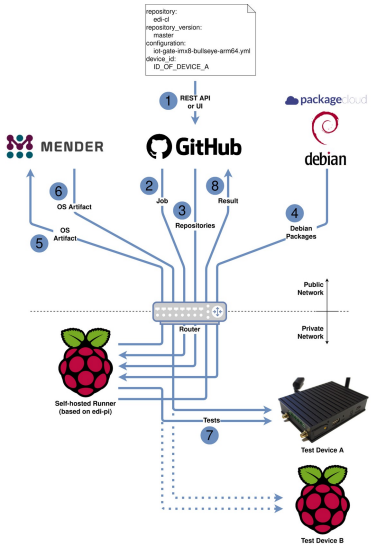
In einem IoT Umfeld soll möglichst viel automatisiert werden:

- Herstellung von massgeschneiderten Betriebssystemen
- Qualitätssicherung
- Konfigurationsmanagement
- Flottenmanagement

Daraus resultieren soll:

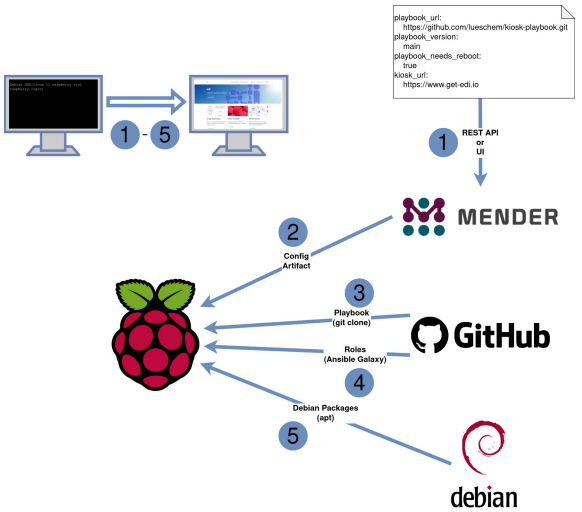
✓ Hohe Qualität
✓ Reproduzierbarkeit
✓ Sicherheit
✓ Reduzierter Aufwand
✓ Kurze Reaktionszeiten

Inhalt



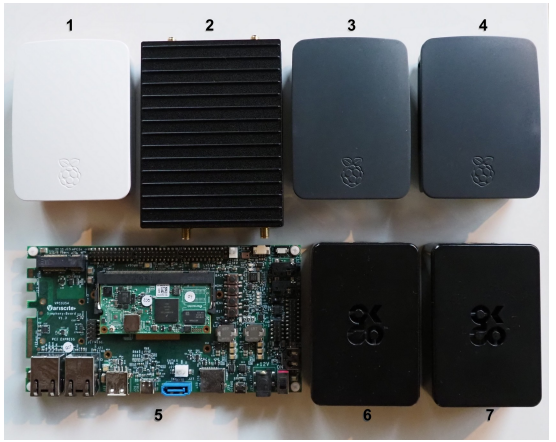
Kontinuierliche Integration

Bau eines Betriebssystems, inklusive Installation auf einem Zielsystem mit abschliessenden Tests zur Qualitätssicherung



Geräteverwaltung

Anpassung eines IoT Gerätes an einen individuellen Anwendungsfall



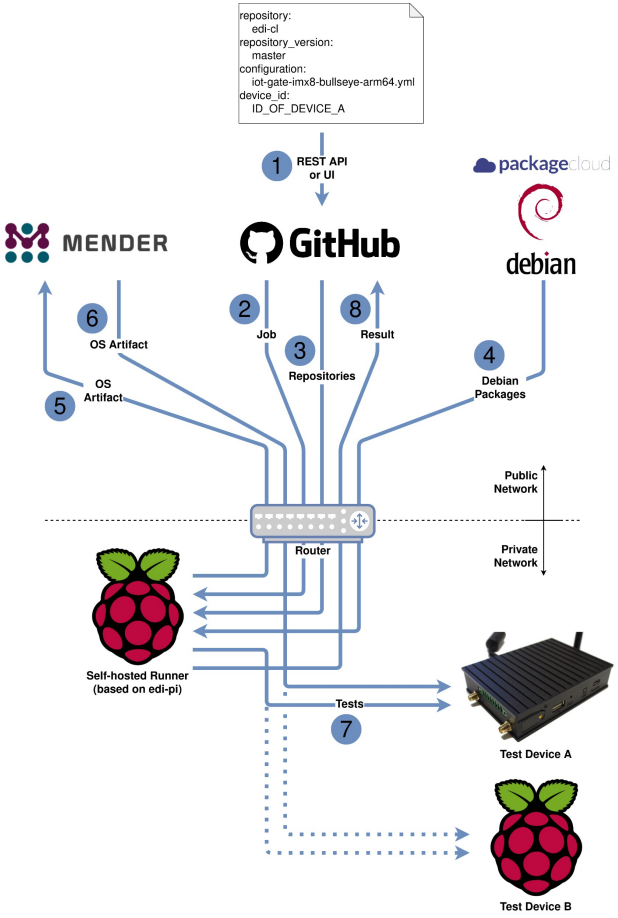
Fortlaufende Auslieferung

Kontinuierliches Aktualisieren einer Geräteflotte basierend auf einer in Git enthaltenen Beschreibung

Kontinuierliche Integration

Kontinuierliche Integration

Überblick: Bau des Betriebssystems ⇨ Online Update ⇨ Tests



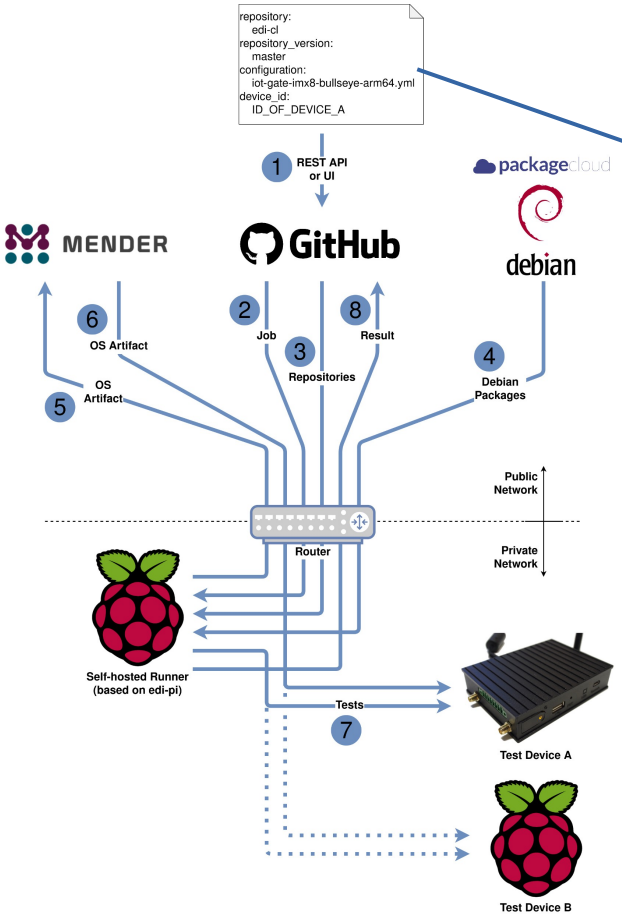
Arbeitsschritte

1. Der Vorgang wird auf GitHub gestartet ([1 (private)], [1 (public)]).
2. Die Arbeit wird an ein Raspberry Pi weitergegeben.
3. Das Raspberry Pi klont ein Git Repository.
4. Das Betriebssystem wird aus zahlreichen Debian Paketen zusammengestellt.
5. Das resultierende Artefakt wird auf Mender hochgeladen.
6. Das Betriebssystem wird auf einem Zielgerät installiert.
7. Das Zielgerät wird ausgiebig getestet ([2]).
8. Sämtliche Resultate werden auf GitHub hochgeladen.

Schlüsselprinzipien

- Sicherheit ([3])
- Reproduzierbarkeit
- Automatisierung
- Qualitätssicherung

Kontinuierliche Integration
Starten des Arbeitsablaufes

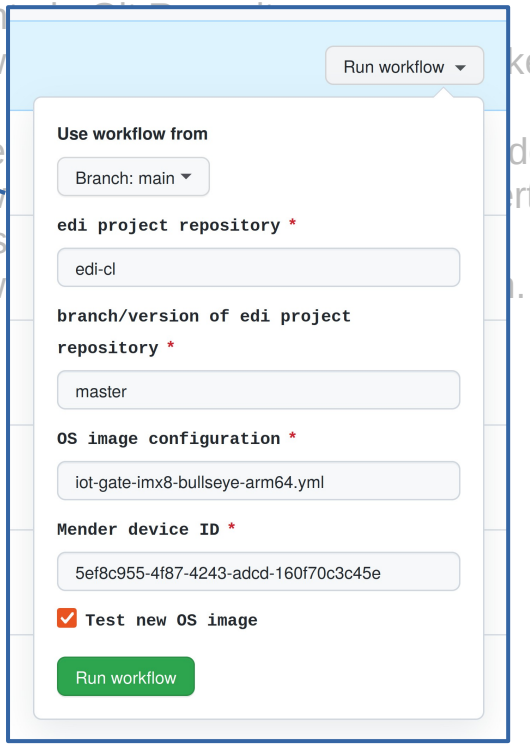


Arbeitsschritte

1. Der Vorgang wird auf GitHub gestartet ([1 (private)], [1 (public)]).
2. Die Arbeit wird an ein Raspberry Pi weitergegeben.
3. Das Raspberry Pi klon
4. Das Betriebssystem w
   zusammengestellt.
5. Das resultierende Arte
6. Das Betriebssystem w
7. Das Zielgerät wird aus
8. Sämtliche Resultate w

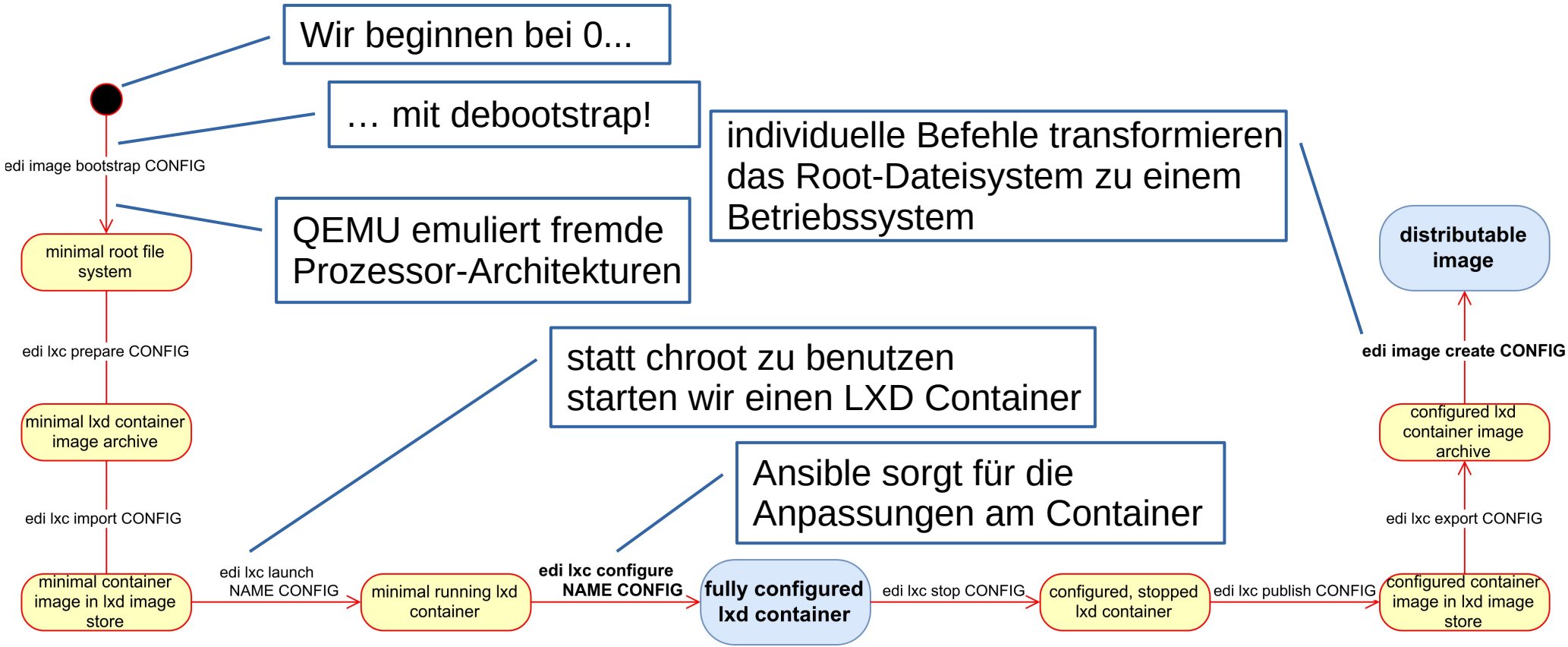
Schlüsselprinzipien

- Sicherheit ([3])
- Reproduzierbarkeit
- Automatisierung
- Qualitätssicherung



Kontinuierliche Integration

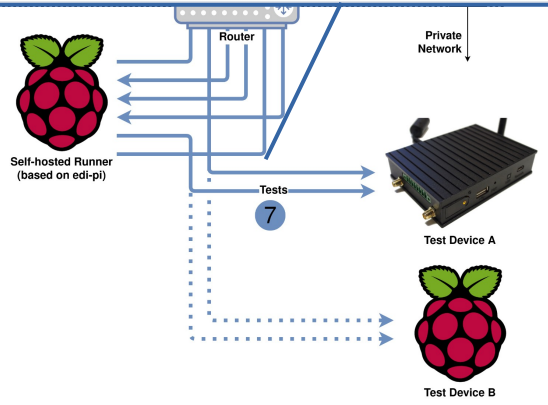
Bau des Betriebssystems



Kontinuierliche Integration

Qualitätssicherung durch zahlreiche Tests

```python
import re
import pytest


def test_root_device(host):
    cmd = host.run("df / --output=pcent")
    assert cmd.rc == 0
    match = re.search(r"(\d{1,3})%", cmd.stdout)
    assert match
    # if the usage is below 50% then the root device got properly resized
    assert int(match.group(1)) < 50


def test_resize_completion(host):
    assert host.file("/etc/edi-resize-rootfs.done").exists


@pytest.mark.parametrize("mountpoint", ["/", "/data", "/boot/firmware", ])
def test_mountpoints(host, mountpoint):
    assert host.mount_point(mountpoint).exists
```



Arbeitsschritte

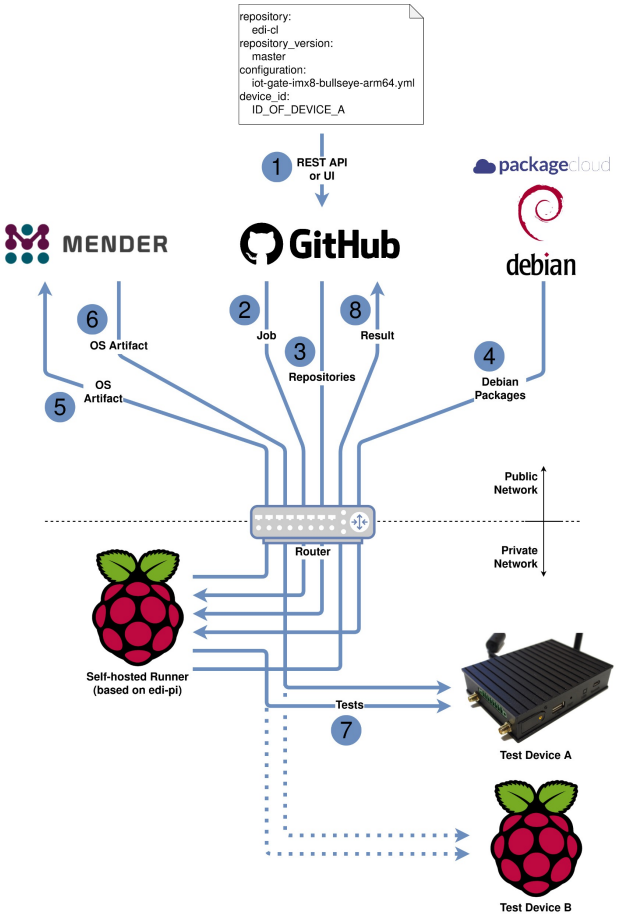
1. Der Vorgang wird auf GitHub gestartet ([1 (private)], [1 (public)]).
2. Die Arbeit wird an ein Raspberry Pi weitergegeben.
3. Das Raspberry Pi klont ein Git Repository.
4. Das Betriebssystem wird aus zahlreichen Debian Paketen zusammengestellt.
5. Das resultierende Artefakt wird auf Mender hochgeladen.
6. Das Betriebssystem wird auf einem Zielgerät installiert.
7. Das Zielgerät wird ausgiebig getestet ([2]).
8. Sämtliche Resultate werden auf GitHub hochgeladen.

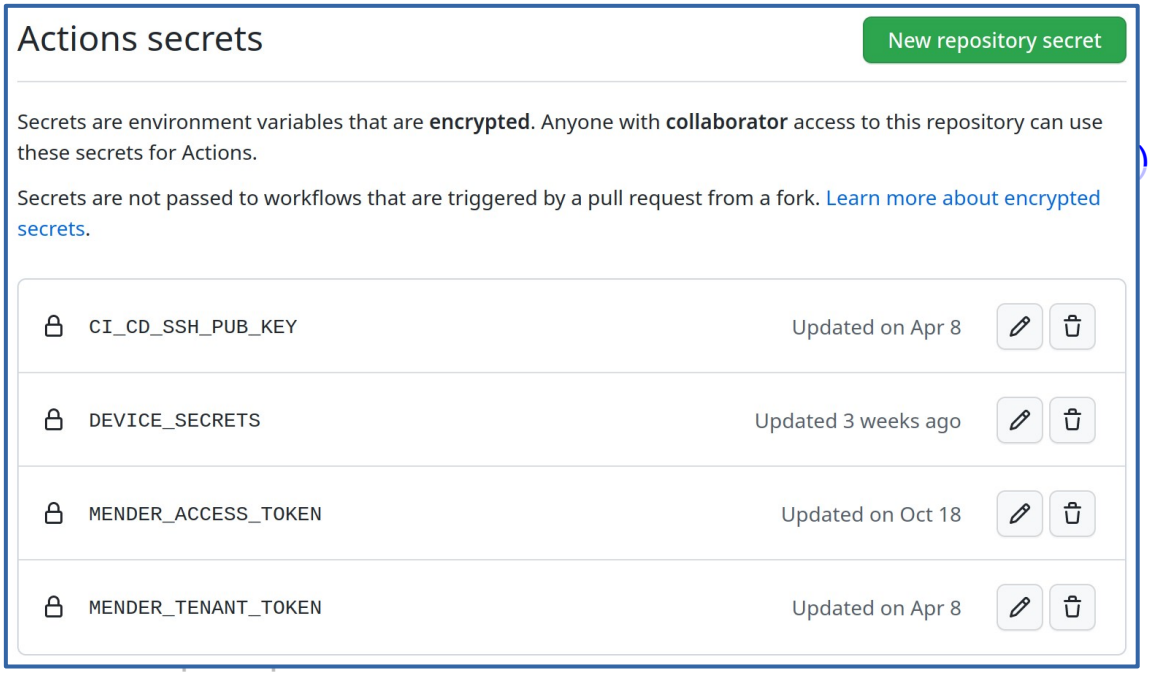
Schlüsselprinzipien

- Sicherheit ([3])
- Reproduzierbarkeit
- Automatisierung
- Qualitätssicherung

Kontinuierliche Integration

Geheimnisse



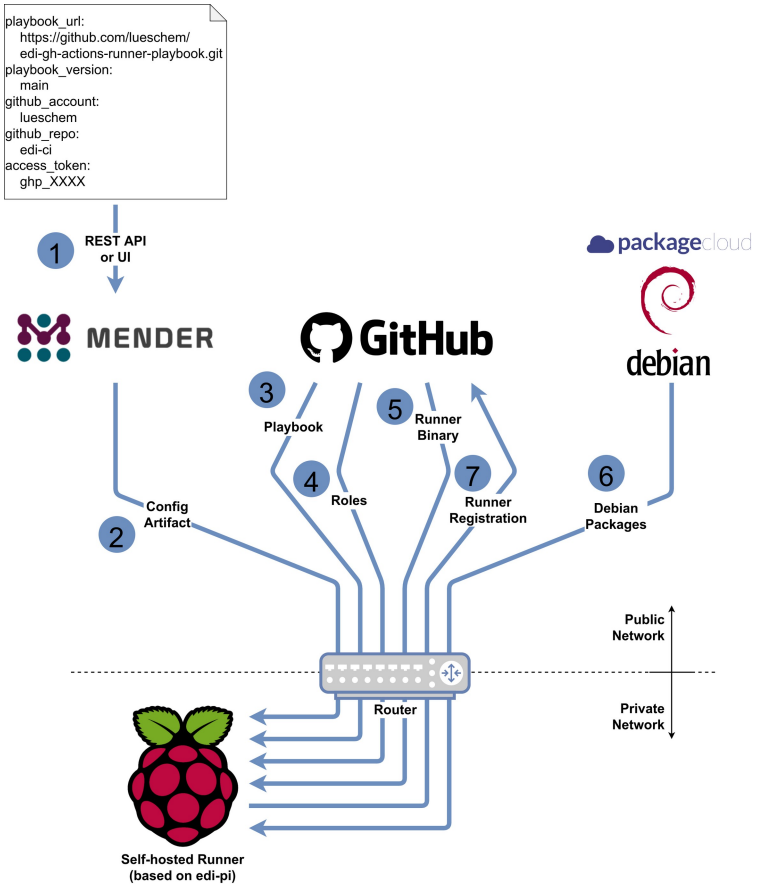


- Sicherheit ([3])
- Reproduzierbarkeit
- Automatisierung
- Qualitätssicherung

Geräteverwaltung

Geräteverwaltung

Beispiel: IoT Gerät soll in einen «GitHub runner» verwandelt werden



Arbeitsschritte

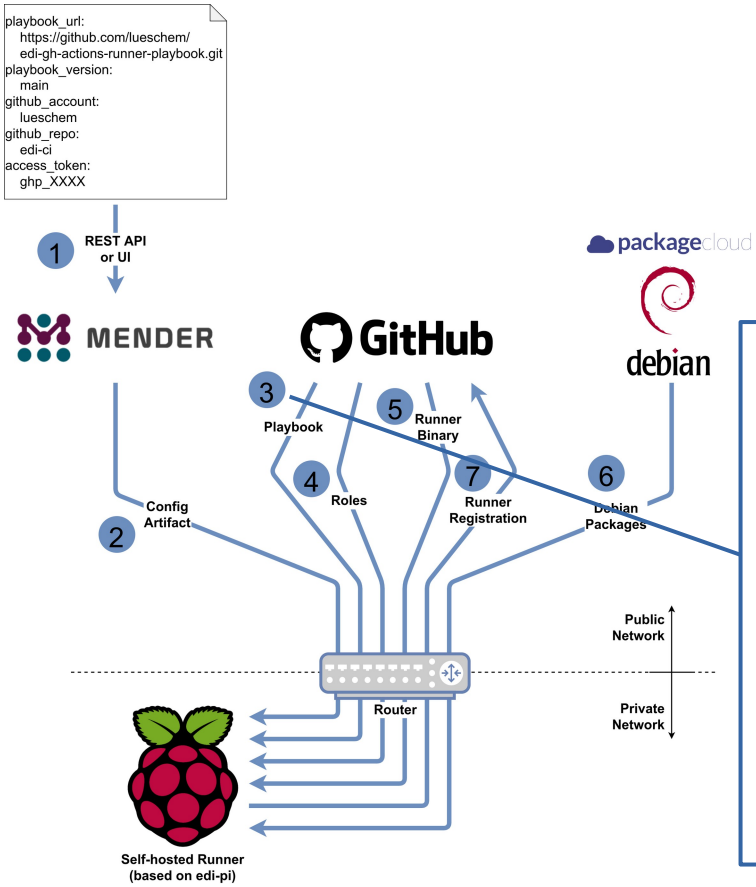
1. Eine Konfiguration wird einem Gerät zugewiesen.
2. Die Konfiguration wird an das Gerät übermittelt.
3. Das Gerät holt ein «Playbook» mittels Git ([1]).
4. Das Gerät holt vom «Playbook» benötigte «Rollen».
5. Das Gerät holt den «.NET GitHub actions runner».
6. Das Gerät installiert zusätzliche Debian Pakete.
7. Der «GitHub actions runner» registriert sich auf GitHub ([2]).

Schlüsselprinzipien

- Idempotenz
- Rückverfolgbarkeit
- Das Gerät kennt sich selbst am besten
- Sicherheit
- Reproduzierbarkeit
- Automatisierung

Geräteverwaltung

Beispiel: IoT Gerät soll in einen «GitHub runner» verwandelt werden



Arbeitsschritte

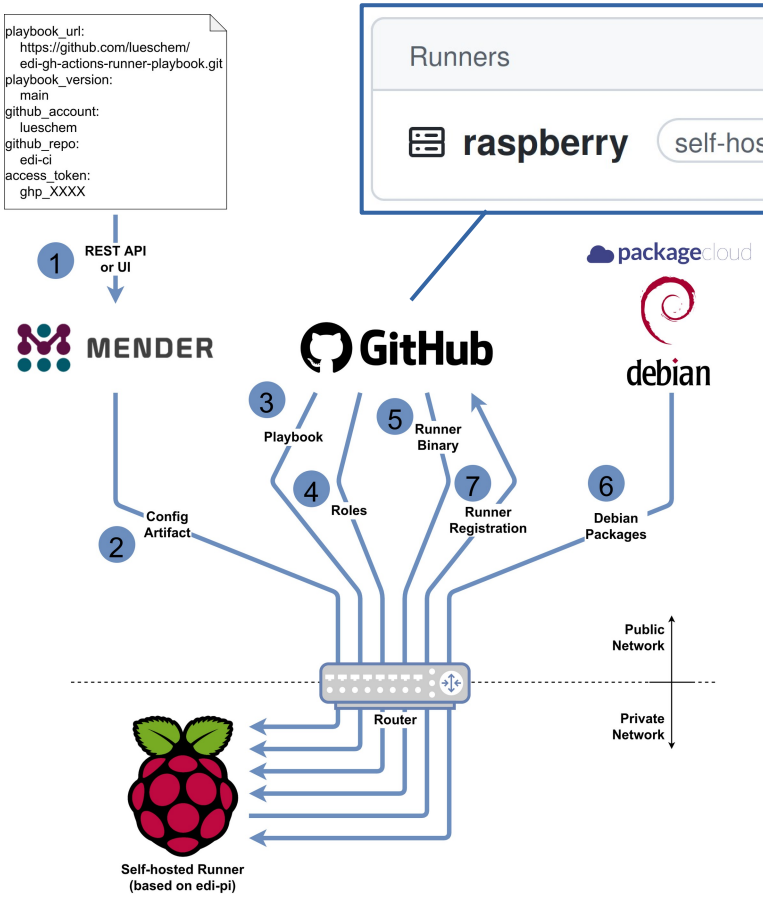
1. Eine Konfiguration wird einem Gerät zugewiesen.
2. Die Konfiguration wird an das Gerät übermittelt.
3. Das Gerät holt ein «Playbook» mittels Git ([1]).
4. Das Gerät holt vom «Playbook» benötigte «Rollen».
5. Das Gerät holt den «.NET GitHub actions runner».

tHub ([2]).

```yaml
---
- name: Install GitHub Actions Runner
  hosts: all

  roles:
    - role: ansible-github_actions_runner
      user: gitops
      become: true
    - role: edi_installer
      become: true
```

Geräteverwaltung

Beispiel: IoT Gerät soll in einen «GitHub runner» verwandelt werden



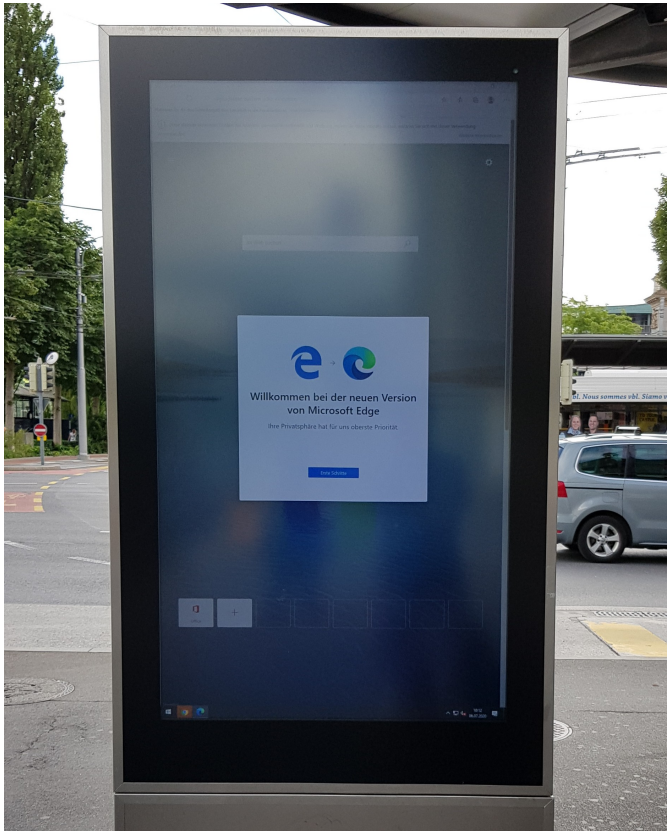
zugewiesen.
übermittelt.
s Git ([1]).
4. Das Gerät holt vom «Playbook» benötigte «Rollen».
5. Das Gerät holt den «.NET GitHub actions runner».
6. Das Gerät installiert zusätzliche Debian Pakete.
7. Der «GitHub actions runner» registriert sich auf GitHub ([2]).

Schlüsselprinzipien

- Idempotenz
- Rückverfolgbarkeit
- Das Gerät kennt sich selbst am besten
- Sicherheit
- Reproduzierbarkeit
- Automatisierung

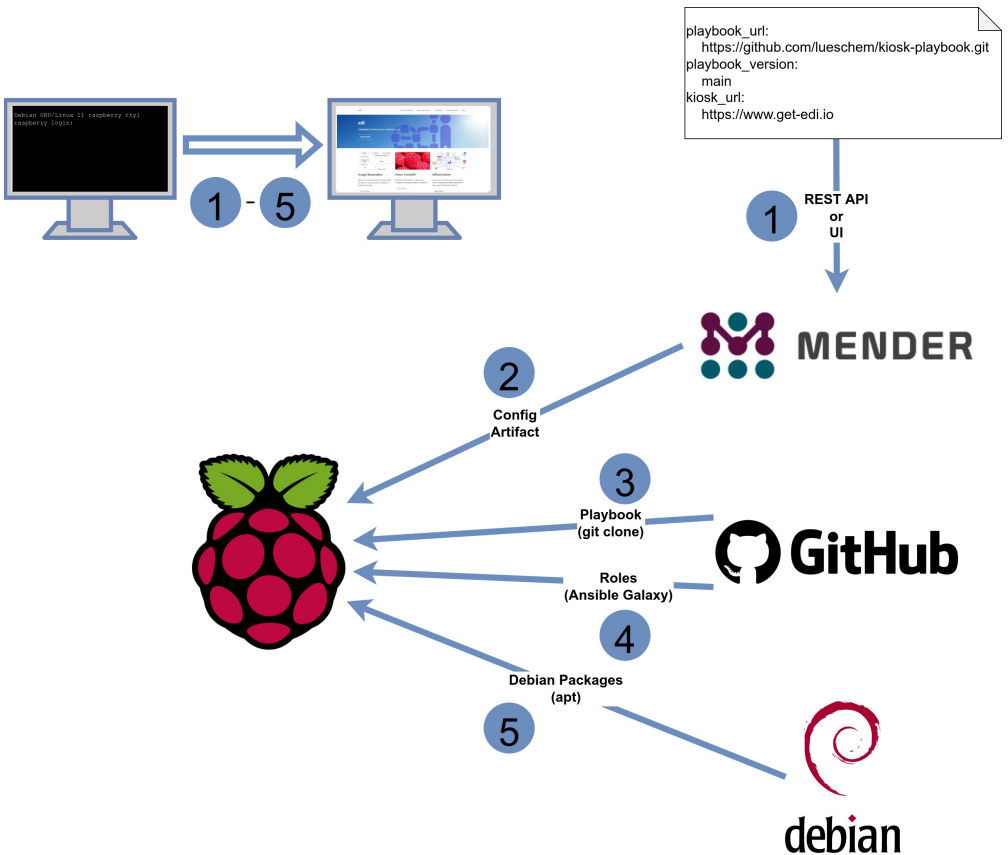
Geräteverwaltung

Beispiel: IoT Gerät soll Informationen darstellen (Kiosk System)



Geräteverwaltung

Beispiel: IoT Gerät soll Informationen darstellen (Kiosk System)



Arbeitsschritte

1. Eine Konfiguration wird einem Gerät zugewiesen.
2. Die Konfiguration wird an das Gerät übermittelt.
3. Das Gerät holt sich ein «Playbook» mittels Git.
4. Das Gerät holt vom «Playbook» benötigte «Rollen».
5. Während der Ausführung des «Playbooks» werden zahlreiche Pakete nachinstalliert.

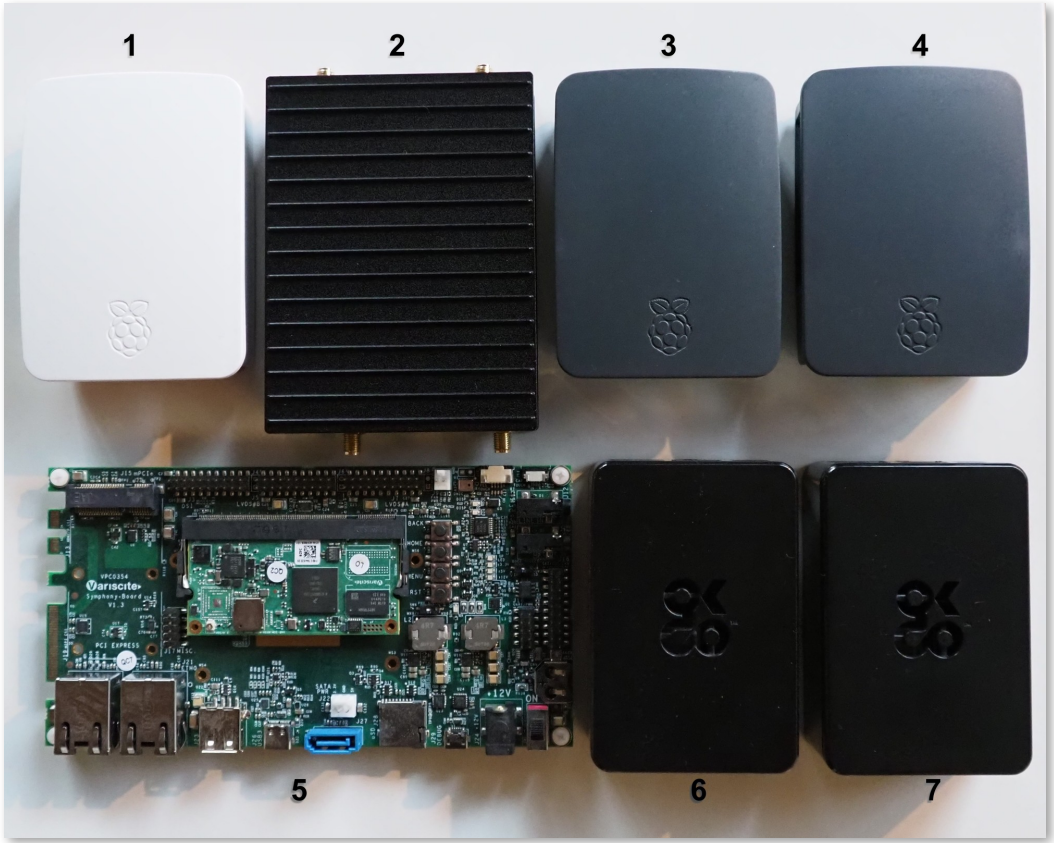
Schlüsselprinzipien

- Idempotenz
- Rückverfolgbarkeit
- Das Gerät kennt sich selbst am besten

Fortlaufende Auslieferung

Beispielflotte

Verschiedene Geräte, unterschiedliche Anwendungsfälle



1. Raspberry Pi 2
   Legacy-Gerät
2. Compulab IOT-GATE-iMX8
   WiFi 6 hotspot
3. Raspberry Pi 3
   Kiosk System
4. Raspberry Pi 3
   Kiosk System
5. Variscite VAR-SOM-MX8M-NANO
   Entwicklungsgerät
6. Raspberry Pi 4
   GitHub actions runner
7. Raspberry Pi 4
   Kiosk System

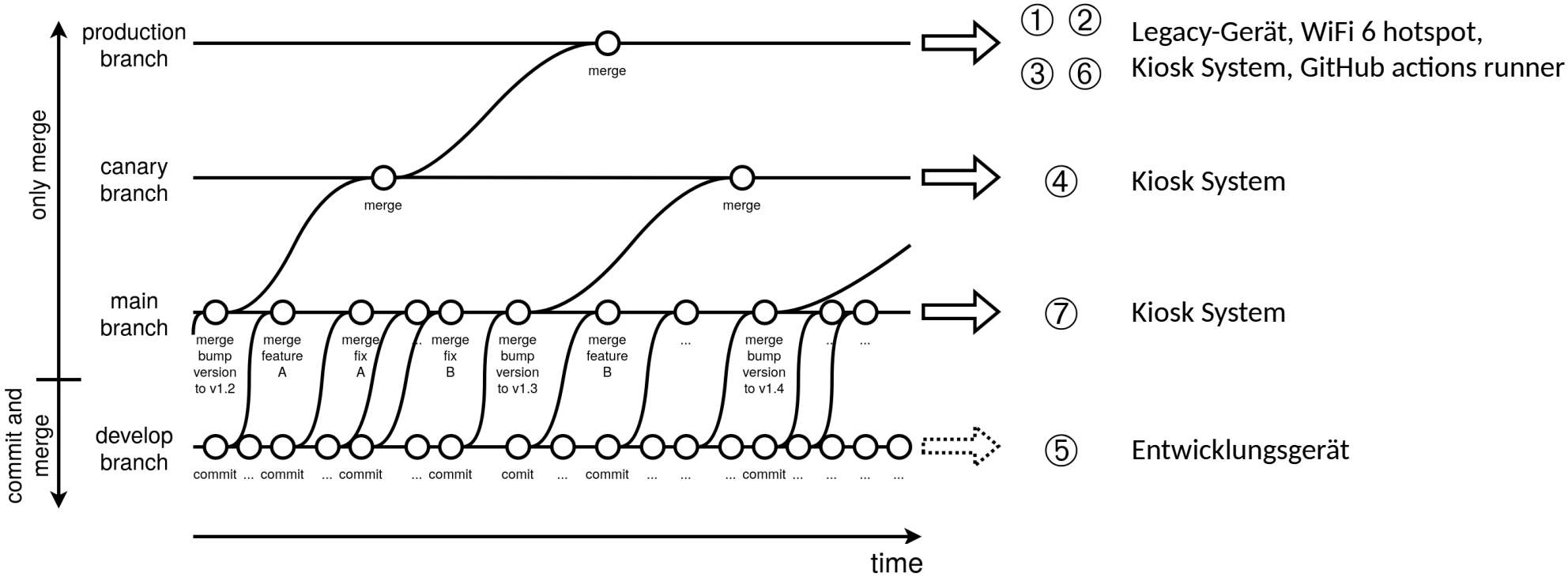
GitOps
Was ist GitOps?

- GitOps ist ein neues Konzept bzw. Schlagwort der IT Gurus.
- Das Ziel ist es, so viele IT Abläufe wie möglich zu automatisieren.
- Die Automatisierung soll basierend auf einem komplett beschriebenen und versionierten Zielzustand ausgeführt werden.
- Git wird gewöhnlich zur Speicherung des Zielzustandes verwendet.
- Eine Reihe weiterer Tools appliziert den Zielzustand auf die Infrastruktur.

→ Die Anwendbarkeit von GitOps beschränkt sich nicht auf die IT Infrastruktur - auch im IoT Umfeld lässt sich das Konzept erfolgreich umsetzen!

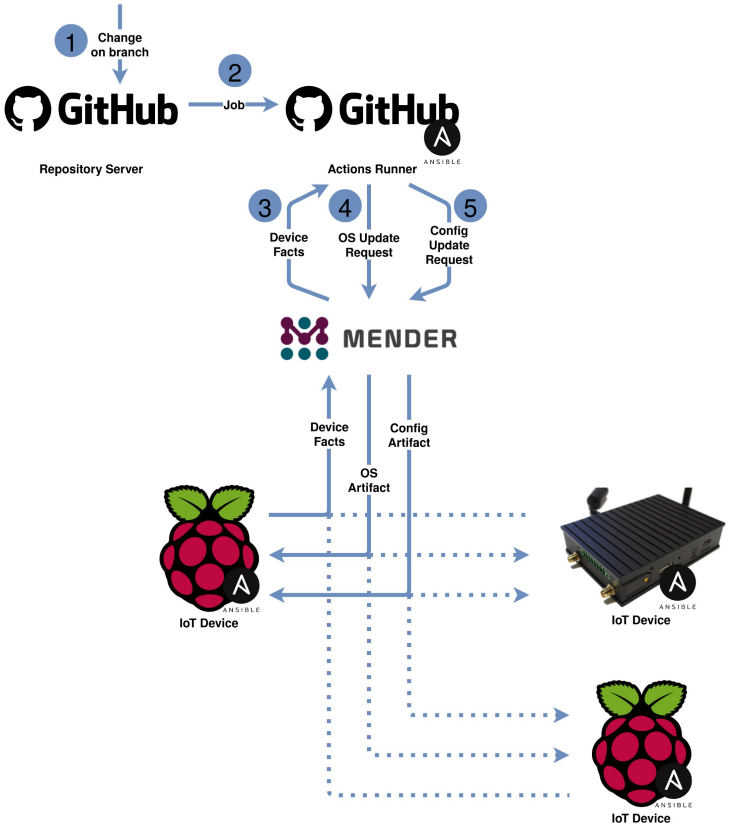
GitOps

Die Flotte wird in einem Git Repository abgebildet



GitOps

Ein Blick hinter die Bühne



Arbeitsschritte

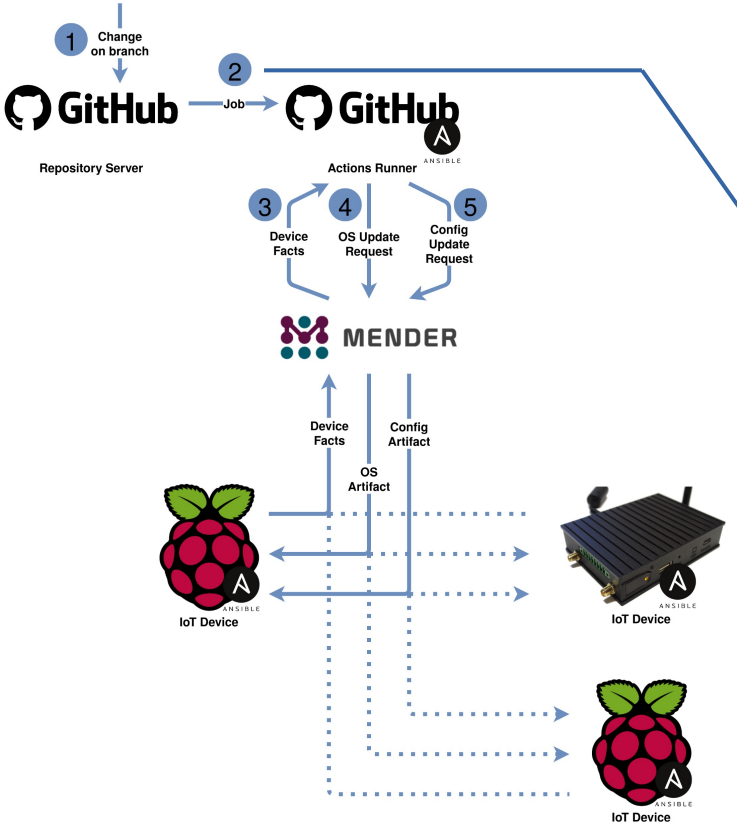
1. Ein Ast im Git Repository wird modifiziert:
   develop/feature Ast: commit
   main/canary/production Ast: merge
2. GitHub schickt einen Auftrag an einen «Runner» ([1]),
   worauf dieser das Flotten Repository klont ([2], [3], [4]).
3. Der aktuelle Zustand der Flotte wird über Mender erfragt.
4. Betriebssystemaktualisierungen werden veranlasst ([5]).
5. Konfigurationsanpassungen werden eingeplant.

Schlüsselprinzipien

- Idempotenz
- Rückverfolgbarkeit
- Schrittweises Ausrollen der Änderungen auf der Flotte
- Ab main Ast werden keine Änderungen mehr vorgenommen
- Der Stellvertreter (hier Mender) nimmt die Befehle für die Flotte entgegen

GitOps

Bereits bekannte Werkzeuge orchestrieren den Vorgang



Arbeitsschritte

1. Ein Ast im Git Repository wird modifiziert:
   develop/feature Ast: commit
   main/canary/production Ast: merge
2. GitHub schickt einen Auftrag an einen «Runner» ([1]),
   worauf dieser das Flotten Repository klont ([2], [3], [4]).

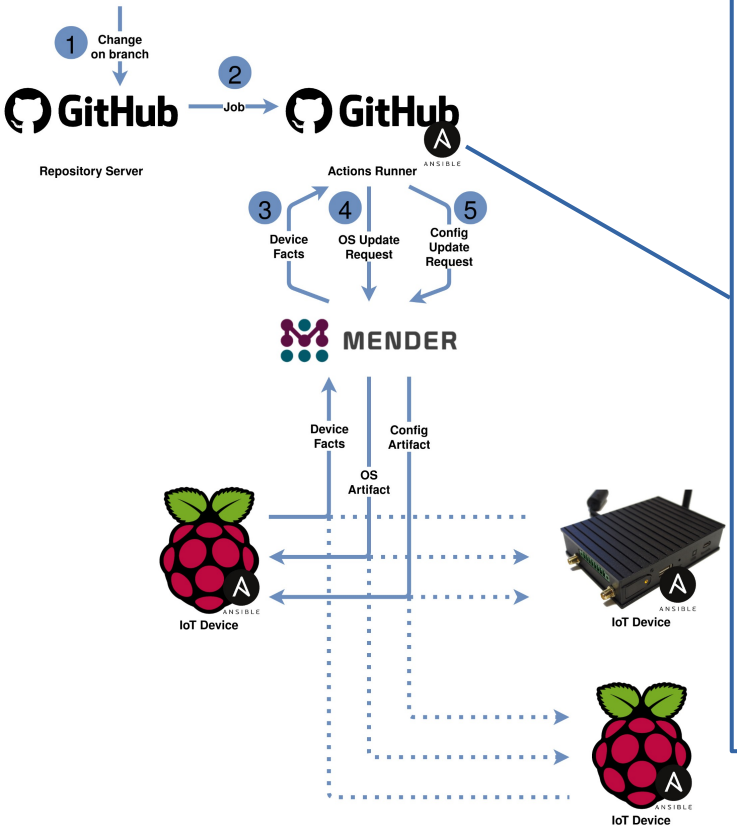
```yaml
name: update fleet
on:
  push:
  workflow_dispatch:

jobs:
  build:
    runs-on: ubuntu-20.04
    steps:
      - name: Check out the fleet management playbook
        uses: actions/checkout@v3
      - name: Install jmespath into venv of ansible-core
        run: |
          source /opt/pipx/venvs/ansible-core/bin/activate
          python3 -m pip install jmespath
      - name: Run the fleet management playbook
        uses: dawidd6/action-ansible-playbook@v2
        with:
          playbook: manage-fleet.yml
          options: --inventory inventory.yml
```

nder erfragt.
nlasst ([5]).
nt.

r Flotte
orgenommen
hle für die

GitOps

Ein Ansible «Playbook» kümmert sich um die Flotte

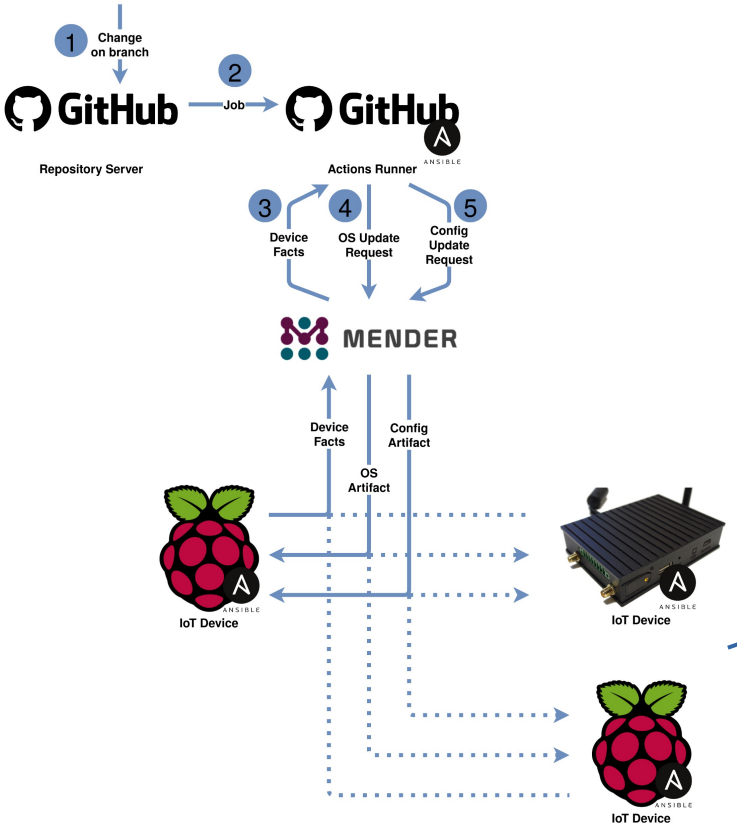



```yaml
---
- name: Apply OS and configuration to fleet.
  hosts: all
  gather_facts: false

  pre_tasks:
    - name: Check for minimum required Ansible version (>=2.10).
      assert:
        that: "ansible_version.full is version_compare('2.10', '>=')"
        msg: "Ansible >= 2.10 is required for this playbook."
      run_once: true

  vars:
    playbook_mode: "{{ lookup('env', 'PLAYBOOK_MODE') | default('dry-run') }}"

  roles:
    - role: gather_fleet_facts
    - role: install_os
      when: subscribed_branch == applied_branch
    - role: apply_configuration
      when: subscribed_branch == applied_branch and configuration.template is defined
```

- Der Stellvertreter (hier Mender) nimmt die Befehle für die Flotte entgegen

GitOps

Das Inventar der Flotte



Arbeitsschritte

1. Ein Ast im Git Repository wird modifiziert:
   develop/feature Ast: commit
   main/canary/production Ast: merge
2. GitHub schickt einen Auftrag an einen «Runner» ([1]),
   worauf dieser das Flotten Repository klont ([2], [3], [4]).

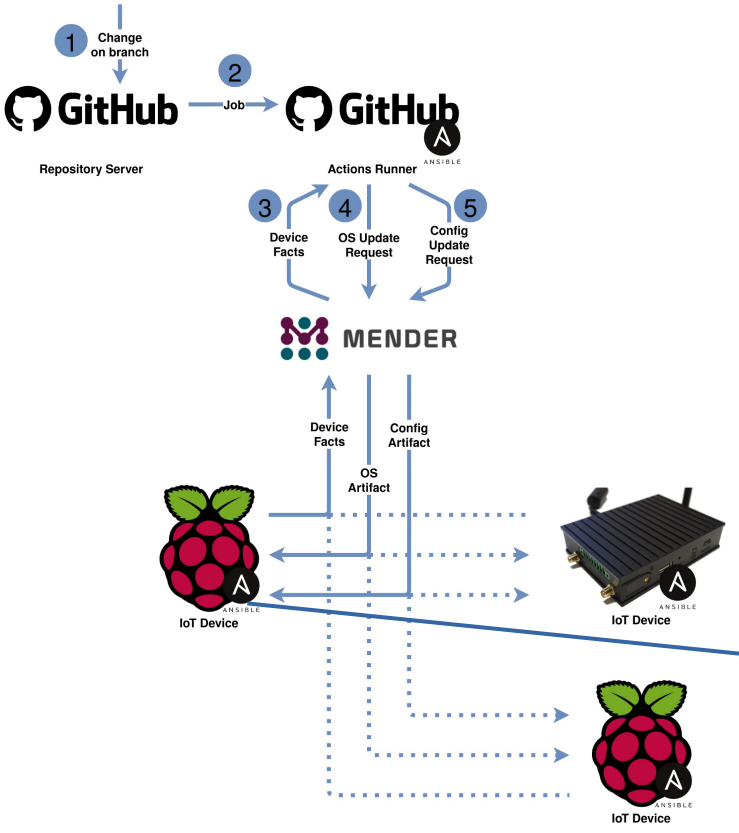
```
all:
  children:
    pi4:
      hosts:
        b8b311de-000e-4914-9a13-1d7e2e23bc5d: # GitHub runner
        3fb4632b-96b9-475d-ac89-02255bd15b6f:
    pi3:
      hosts:
        50a28c2e-3ee8-4559-a5b9-3ce47c881c5d:
        f4580afc-7195-4c8b-b35a-e0248e6bd894:
    pi2:
      hosts:
        048312b5-0456-47a7-9e83-b636f4c0a689:
    iot_gate_imx8:
      hosts:
        5ef8c955-4f87-4243-adcd-160f70c3c45e:
```

der erfragt.
lasst ([5]).
nt.

r Flotte
orgenommen
nle für die

GitOps

Eine individuelle Gerätekonfiguration



Arbeitsschritte

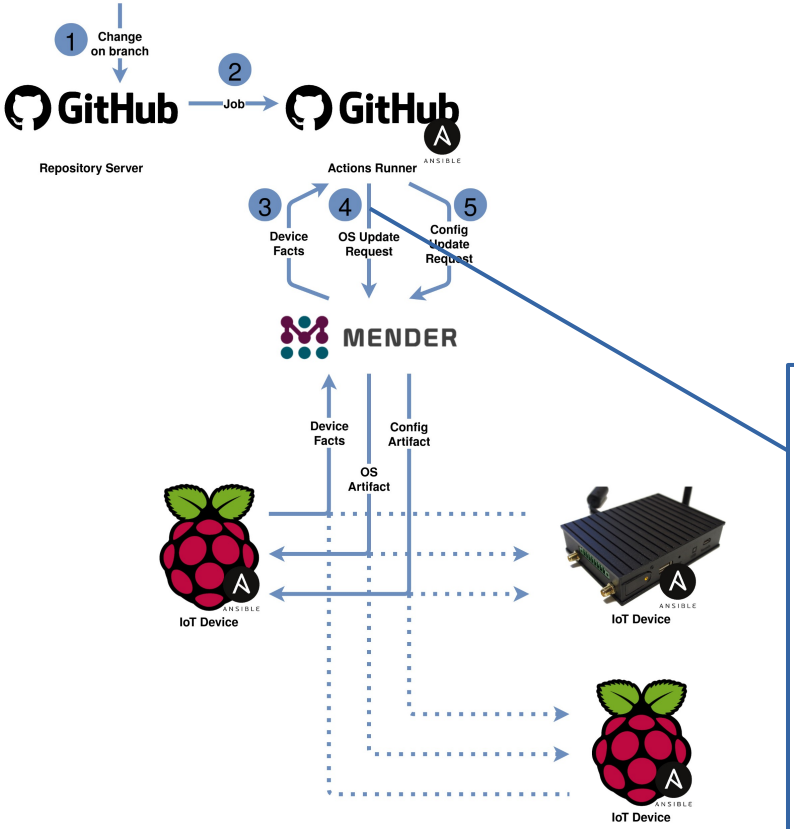
1. Ein Ast im Git Repository wird modifiziert:
   develop/feature Ast: commit
   main/canary/production Ast: merge
2. GitHub schickt einen Auftrag an einen «Runner» ([1]),
   worauf dieser das Flotten Repository klont ([2], [3], [4]).
3. Der aktuelle Zustand der Flotte wird über Mender erfragt.
4. Betriebssystemaktualisierungen werden veranlasst ([5]).
5. Konfigurationsanpassungen werden eingeplant.

```yaml
---
subscribed_branch: main

configuration:
  template: kiosk.json
  parameters:
    kiosk_url: https://www.get-edi.io
```

r Flotte
orgenommen
hle für die

GitOps

Bei Bedarf wird ein neues Betriebssystem installiert



Arbeitsschritte

1. Ein Ast im Git Repository wird modifiziert: develop/feature Ast: commit main/canary/production Ast: merge
2. GitHub schickt einen Auftrag an einen «Runner» ([1]), worauf dieser das Flotten Repository klont ([2], [3], [4]).
3. Der aktuelle Zustand der Flotte wird über Mender erfragt.
4. Betriebssystemaktualisierungen werden veranlasst ([5]).

```yaml
---
mender_server: "https://hosted.mender.io"
subscribed_branch: production

os_image:
  - device_type: pi2-armhf
    image_name: 2022-07-08-1050-pi2-bullseye-armhf
  - device_type: pi3-arm64
    image_name: 2022-07-08-0859-pi3-bullseye-arm64-gitops
  - device_type: pi4-v3-arm64
    image_name: 2022-07-08-0958-pi4-bullseye-arm64-gitops
  - device_type: var-som-mx8m-nano-arm64-v2
    image_name: 2022-07-08-1129-var-som-mx8m-nano-bullseye-arm64
```

GitOps
Zwei Bemerkungen

- Die *Überwachung* der Geräte ist sehr wichtig - aber nicht Teil von dieser Präsentation (z.B. Telegraf und InfluxDB zusammen mit Grafana könnten hier weiterhelfen)!
- Bei einer grossen Flotte würde das *Inventar* und die individuellen *Gerätekonfigurationen* in einer separaten Datenbank abgespeichert.

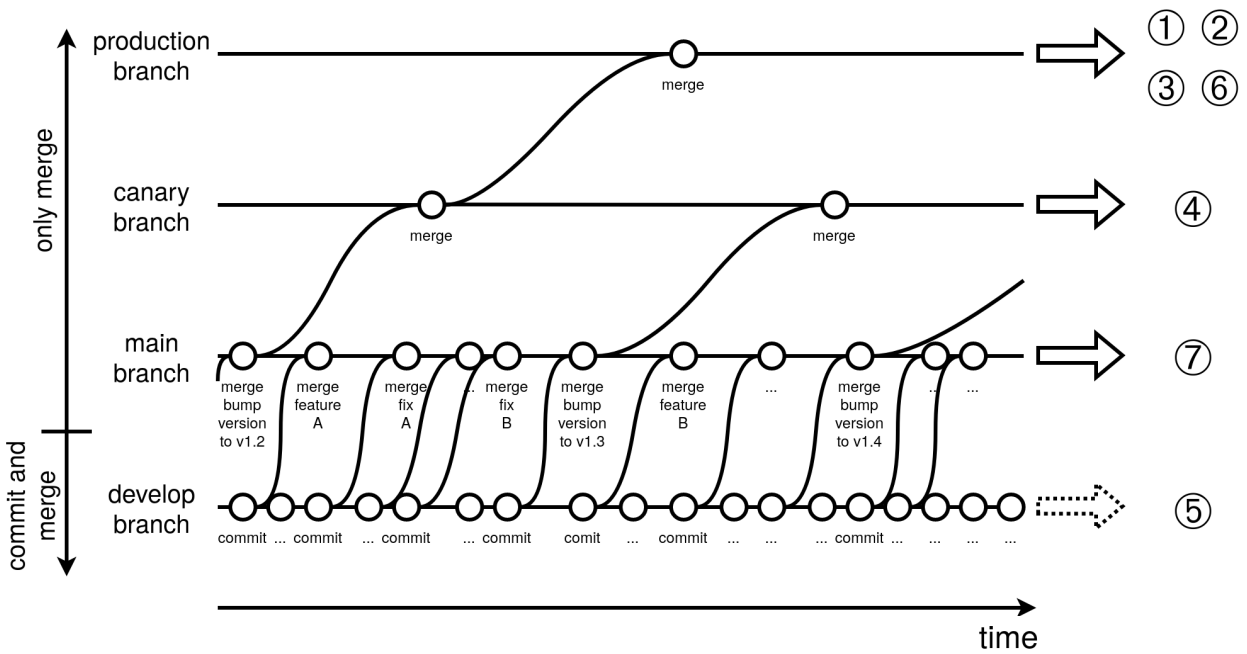
```
all:
  children:
    pi4:
      hosts:
        b8b311de-000e-4914-9a13-1d7e2e23bc5d: # GitHub runner
        3fb4632b-96b9-475d-ac89-02255bd15b6f:
    pi3:
      hosts:
        50a28c2e-3ee8-4559-a5b9-3ce47c881c5d:
        f4580afc-7195-4c8b-b35a-e0248e6bd894:
    pi2:
      hosts:
        048312b5-0456-47a7-9e83-b636f4c0a689:
    iot_gate_imx8:
      hosts:
        5ef8c955-4f87-4243-adcd-160f70c3c45e:
    var_som_mx8m_nano:
      hosts:
        ed531b64-5108-4f1d-9879-f39f56054078:
```

```
---
subscribed_branch: main

configuration:
  template: kiosk.json
  parameters:
    kiosk_url: https://www.get-edi.io
```

Fazit

Flottenmanagement mit GitOps

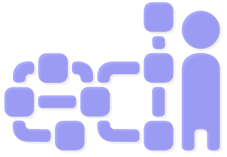
Hauptvorteile I



- Alle Beteiligten arbeiten mit denselben Werkzeugen und sprechen somit die gleiche «Sprache»
- Volle Rückverfolgbarkeit
- Die Arbeit wird auf dem develop Ast getätigt, oberhalb wird nur noch «gemerged»
- Sehr hoher Grad an Automatisierung
- Schrittweises Ausrollen
- Beinahe keine Flüchtigkeitsfehler mehr möglich

Flottenmanagement mit GitOps

Hauptvorteile II



- Mächtige Werkzeugkiste
- Die gewählte Strategie skaliert auch für riesige Flotten
- Alle Werkzeuge sind praxiserprobt
- Die einzelnen Werkzeuge können durch andere Werkzeuge ausgetauscht werden
- Auch nach mehreren Jahren habe ich noch Freude an den gewählten Werkzeugen

Git Repositories

CI Orchestrierung

edi-ci/edi-ci-public

Betriebssystem

edi-pi

edi-var

edi-cl

Playbooks/Rollen

kiosk-playbook

ansible-kiosk

edi-gh-actions-runner-playbook

ansible-github_actions_runner

edi_installer

CD Orchestrierung

edi-cd

Kontinuierliche Integration

Bau eines Betriebssystems, inklusive Installation auf einem Zielsystem mit abschliessenden Tests zur Qualitätssicherung

Geräteverwaltung

Anpassung eines IoT Gerätes an einen individuellen Anwendungsfall

Fortlaufende Auslieferung

Kontinuierliches Aktualisieren einer Geräteflotte basierend auf einer in Git enthaltenen Beschreibung

Weiterführende Informationen

- Embedded Meets GitOps
- Managing an IoT Fleet with GitOps
- Building and Testing OS Images with GitHub Actions
- Surprisingly Easy IoT Device Management

Fragen & Antworten