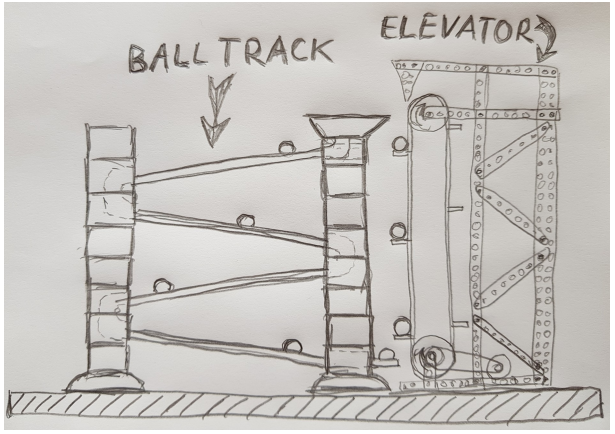
Automating and Managing an IoT Fleet Using Git

Open Source Summit Europe 2022

Matthias Lüscher, Schindler AG

About Me

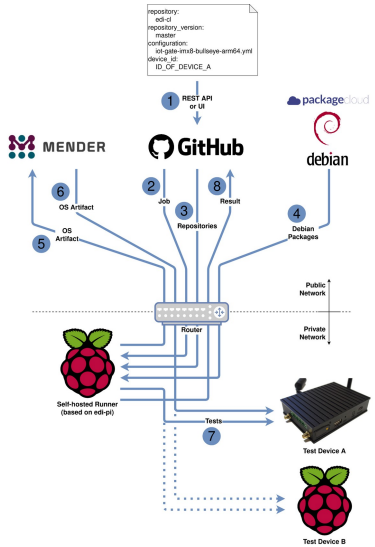


- I prefer to automate boring jobs:
  → E.g. as a child: Operate a ball track using an *elevator*
  → E.g. as a professional: Operate IoT devices that connect *elevators* using CI/CD
- Instead of attending a lot of courses and earning some training awards I decided to create my own open source (automation research) project called edi
- I live in Switzerland and work for Schindler AG as a principal engineer
- During my spare time I enjoy the nature together with my family (biking, hiking, skiing, …)
- Contact: lueschem@gmail.com

Mission:

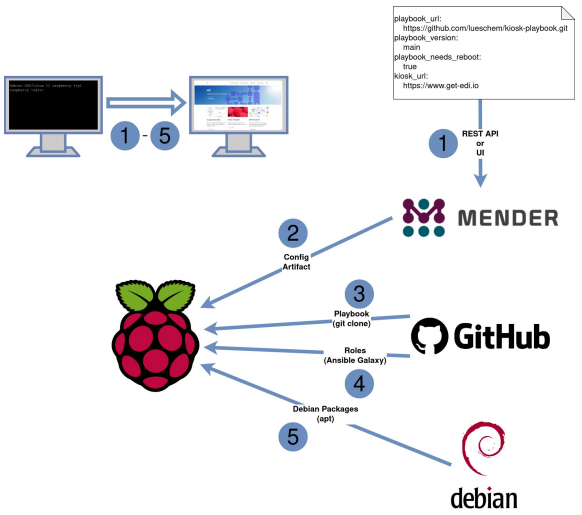
Automate as much as possible in an IoT environment including OS image builds, testing, configuration management and fleet management.

Agenda



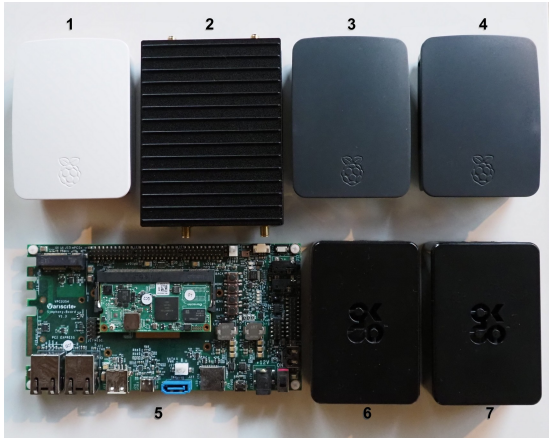
Continuous Integration

Build an OS image for an IoT device, dispatch it to a device and test it



Device Management

Adjust an IoT device for an individual use case



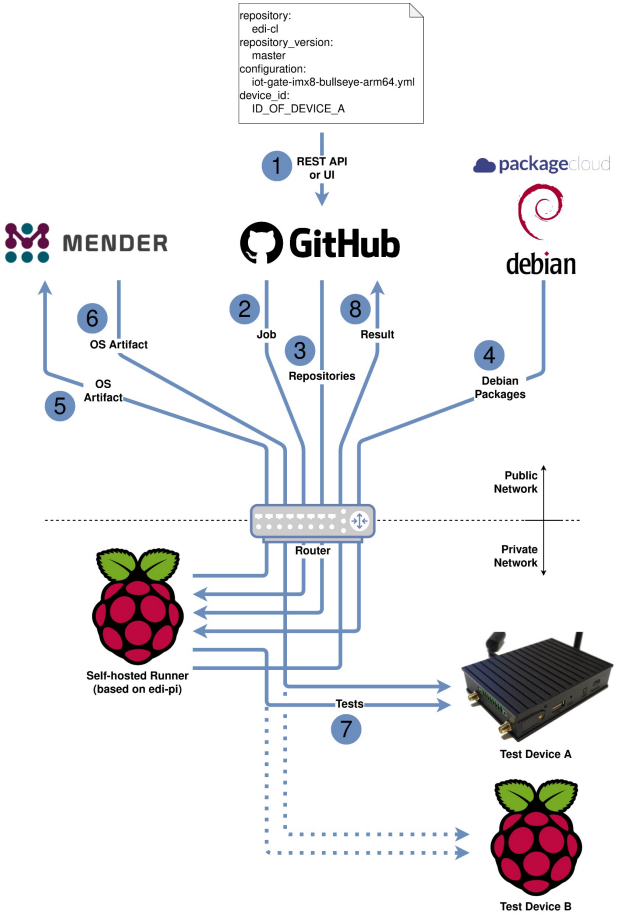
Continuous Delivery

Keep an entire IoT fleet up to date using git

Continuous Integration

Continuous Integration

Overview: OS image → OTA update → test



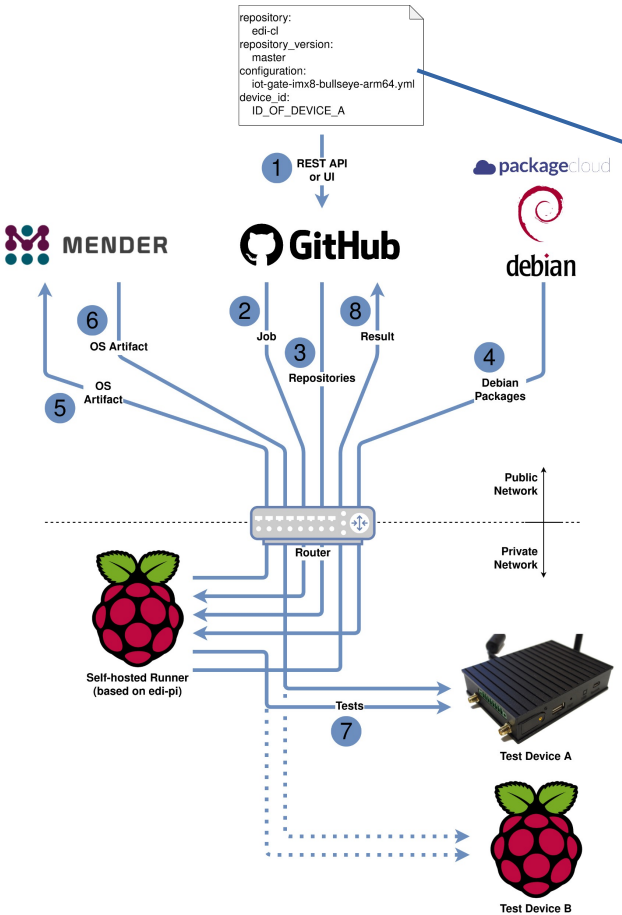
Workflow

1. Start the workflow on GitHub ([1 (private)], [1 (public)])
2. A job gets dispatched to the self-hosted runner
3. The runner clones git repositories
4. During the OS build a lot of Debian packages will be fetched
5. The OS artifact will be uploaded to Mender
6. The OS artifact will be dispatched to the chosen device
7. The device will be thoroughly tested ([2])
8. All the build and test results get uploaded to GitHub

Key Principles

- Security ([3])
- Reproducibility
- Automation
- Quality assurance

Continuous Integration
Start workflow

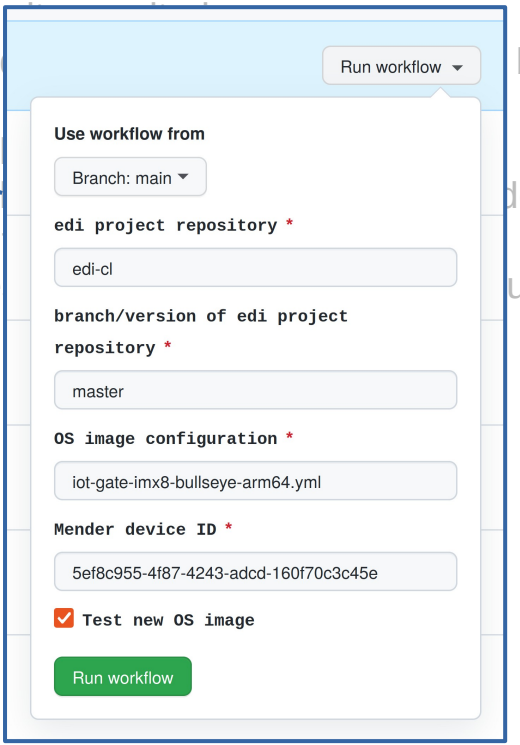


Workflow

1. Start the workflow on GitHub ([1 (private)], [1 (public)])
2. A job gets dispatched to the self-hosted runner
3. The runner clones
4. During the OS buil be fetched
5. The OS artifact wil
6. The OS artifact wi device
7. The device will be
8. All the build and te ub

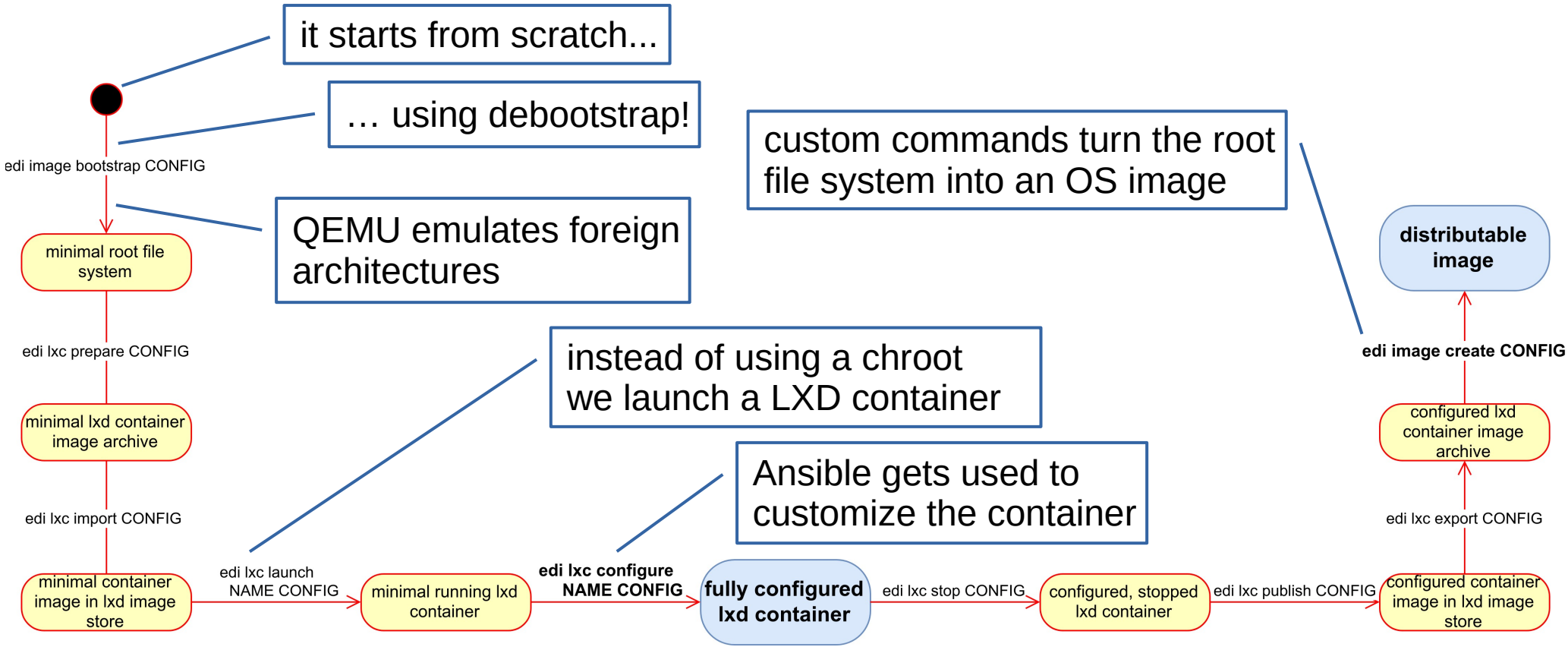
Key Principles

- Security ([3])
- Reproducibility
- Automation
- Quality assurance



Continuous Integration

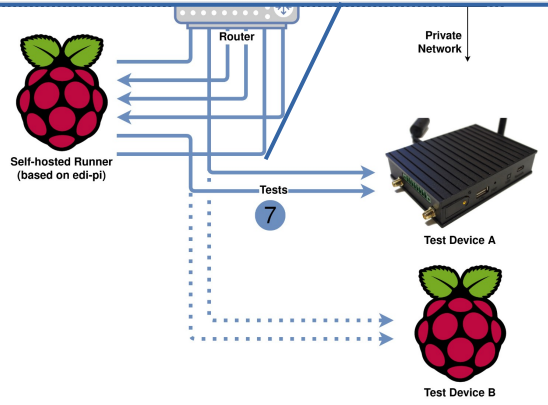
Build the OS image



Continuous Integration

Test the device

```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
```



Workflow

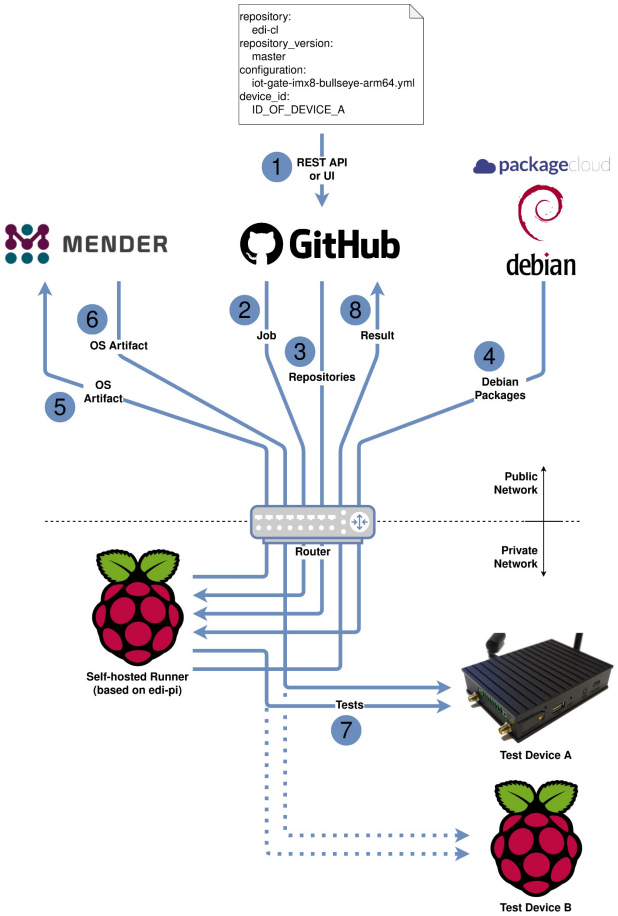
1. Start the workflow on GitHub ([1 (private)], [1 (public)])
2. A job gets dispatched to the self-hosted runner
3. The runner clones git repositories
4. During the OS build a lot of Debian packages will be fetched
5. The OS artifact will be uploaded to Mender
6. The OS artifact will be dispatched to the chosen device
7. The device will be thoroughly tested ([2])
8. All the build and test results get uploaded to GitHub

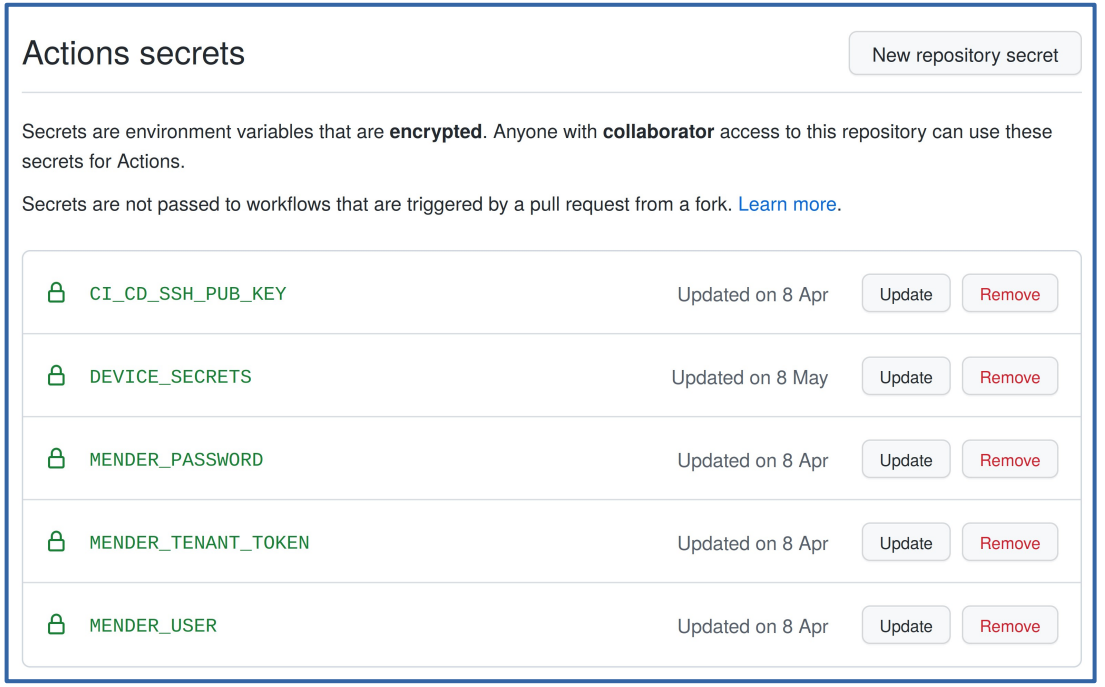
Key Principles

- Security ([3])
- Reproducibility
- Automation
- Quality assurance

Continuous Integration

Handling of secret stuff



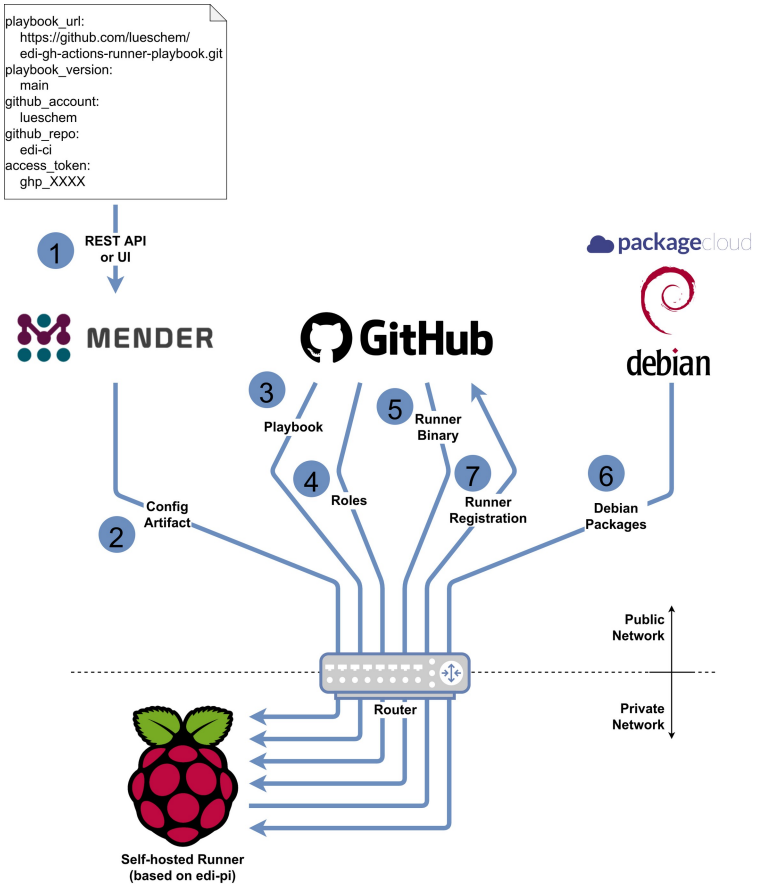


- Security ([3])
- Reproducibility
- Automation
- Quality assurance

Device Management

Device Management

Example: Turn an IoT device into a GitHub runner



Workflow

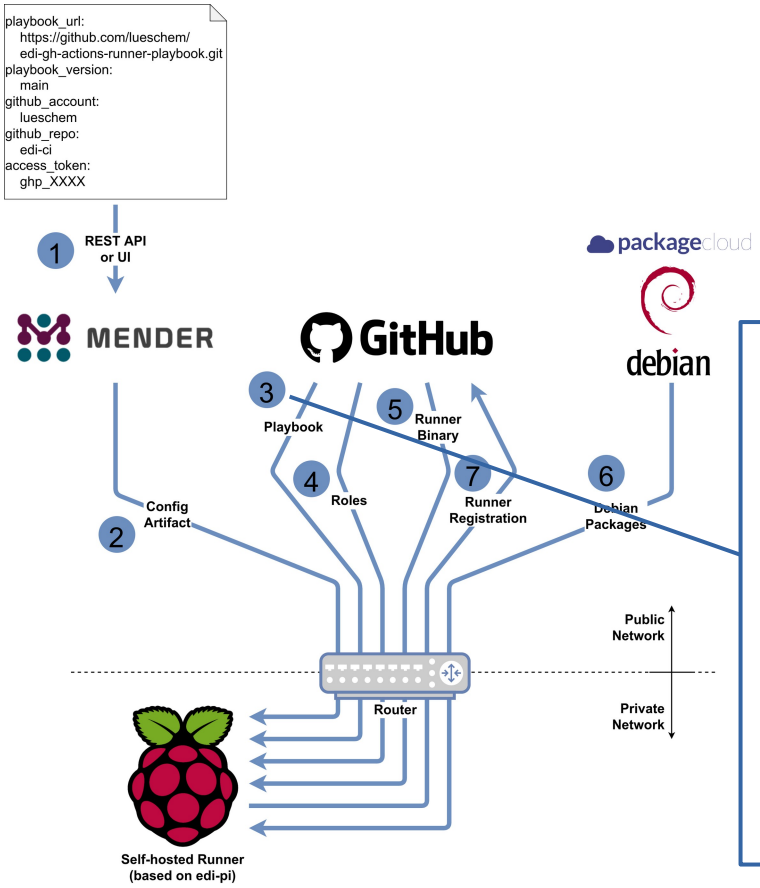
1. Assign a configuration to a device
2. A configuration artifact gets dispatched to the device
3. The device fetches a playbook using git ([1])
4. The device fetches the roles that the playbook requests
5. The device fetches the .NET GitHub actions runner binary
6. The device fetches some additional Debian packages
7. The GitHub actions runner registers itself on GitHub ([2])

Key Principles

- Idempotency
- Traceability
- The device knows a lot about itself
- Security
- Reproducibility
- Automation

Device Management

Example: Turn an IoT device into a GitHub runner



Workflow

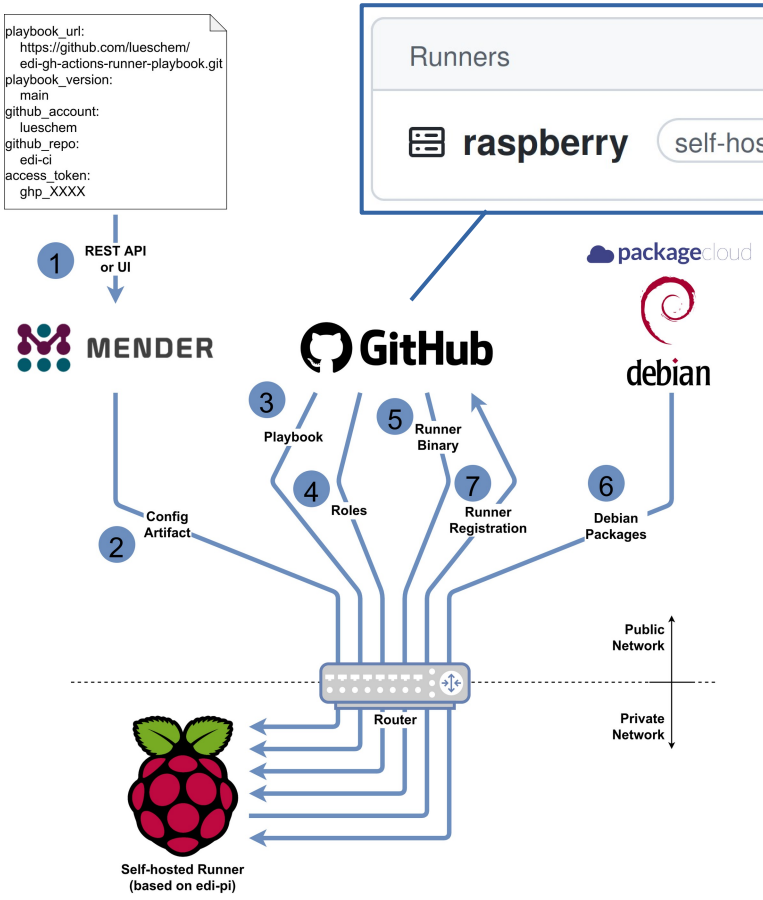
1. Assign a configuration to a device
2. A configuration artifact gets dispatched to the device
3. The device fetches a playbook using git ([1])
4. The device fetches the roles that the playbook requests
5. The device fetches the .NET GitHub actions runner binary

ges
b ([2])

```
1  ---
2  - name: Install GitHub Actions Runner
3    hosts: all
4
5    roles:
6      - role: ansible-github_actions_runner
7        user: gitops
8        become: true
9      - role: edi_installer
10       become: true
```

Device Management

Example: Turn an IoT device into a GitHub runner



ed to the device

git ([1])

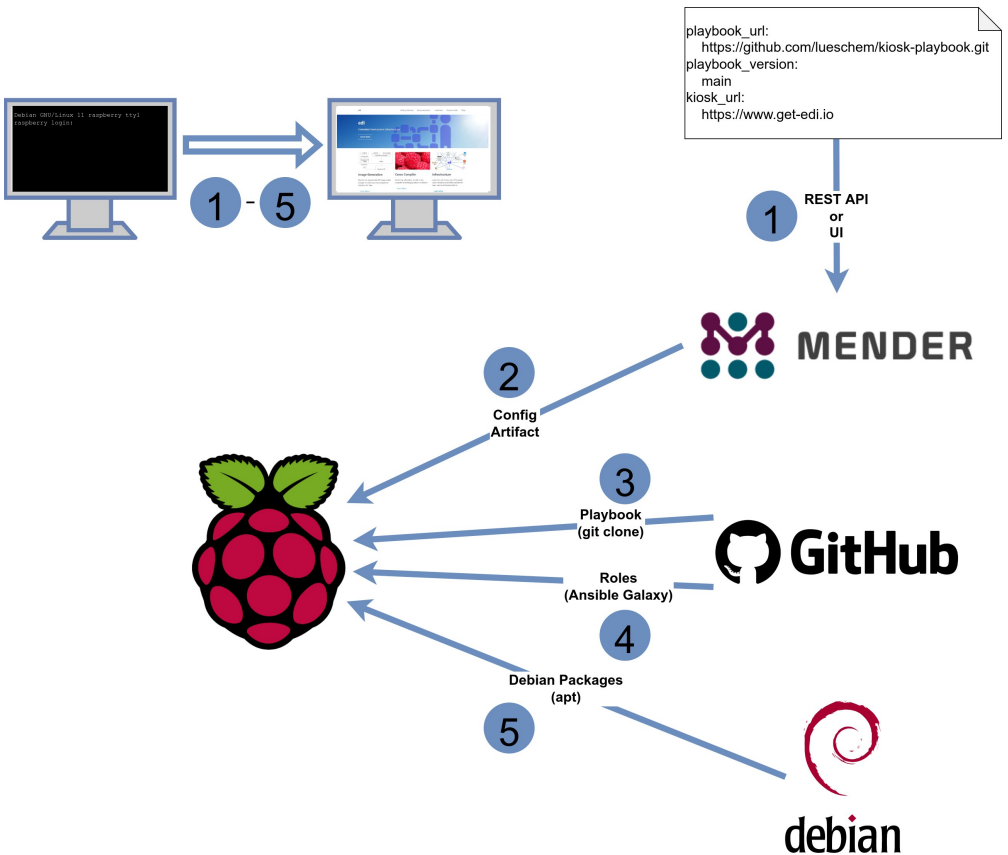
4. The device fetches the roles that the playbook requests
5. The device fetches the .NET GitHub actions runner binary
6. The device fetches some additional Debian packages
7. The GitHub actions runner registers itself on GitHub ([2])

Key Principles

- Idempotency
- Traceability
- The device knows a lot about itself
- Security
- Reproducibility
- Automation

Device Management

Example: Turn a headless device into a kiosk terminal



Workflow

1. Assign a configuration to a device
2. A configuration artifact gets dispatched to the device
3. The device fetches a playbook using git
4. The device fetches the roles that the playbook requests
5. The playbook gets applied and during that process some additional packages might get installed

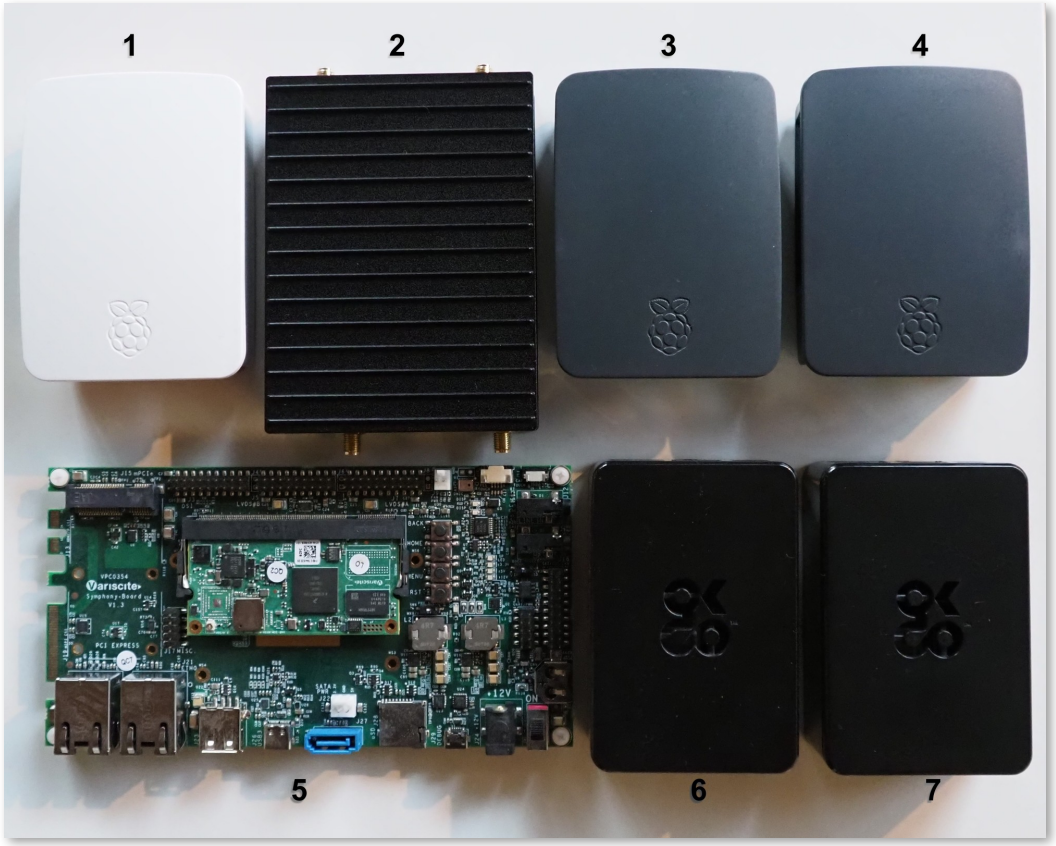
Key Principles

- Idempotency
- Traceability
- The device knows a lot about itself

Continuous Delivery

Demo Fleet

Different devices, different use cases



1. Raspberry Pi 2
   legacy device
2. Compulab IOT-GATE-iMX8
   WiFi 6 hotspot
3. Raspberry Pi 3
   kiosk terminal
4. Raspberry Pi 3
   kiosk terminal
5. Variscite VAR-SOM-MX8M-NANO
   development device
6. Raspberry Pi 4
   GitHub actions runner
7. Raspberry Pi 4
   kiosk terminal

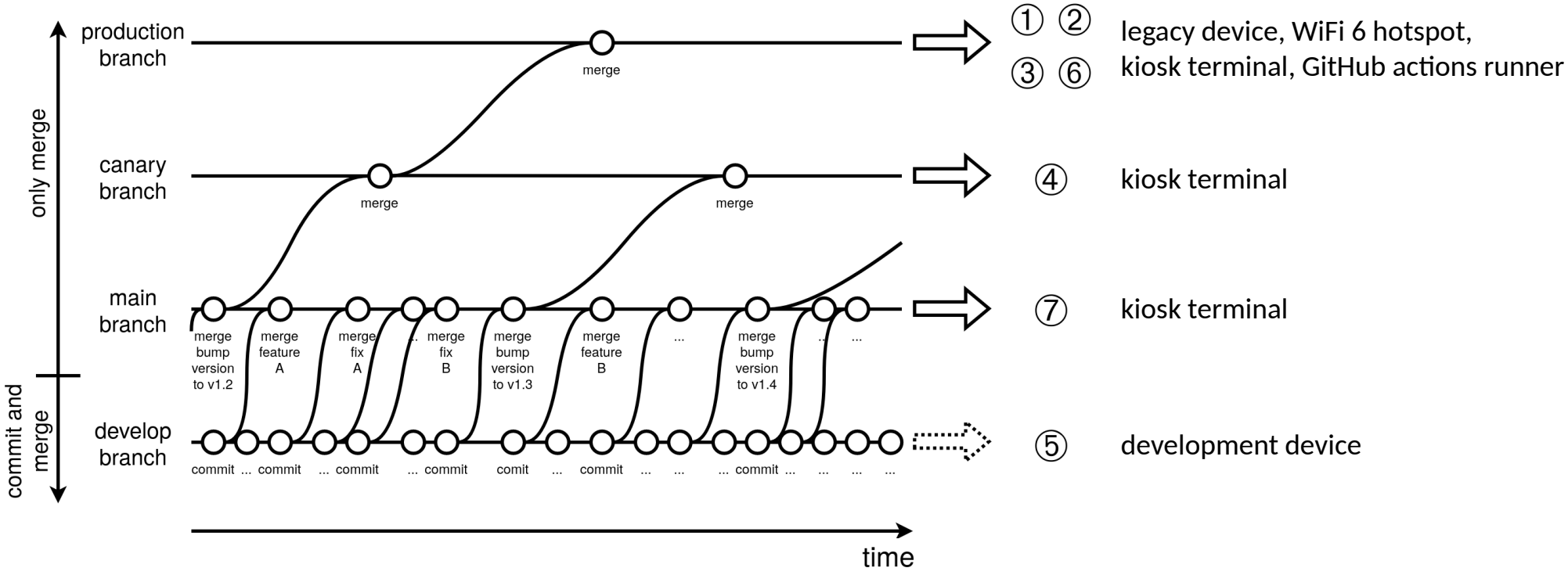
GitOps
What is GitOps?

- A new concept/buzzword in the IT industry
- The goal is to automate as many IT operations as possible
- The automation shall be based on a fully declared and versioned target state
- Git is usually the tool of choice to store the target state
- A bunch of tools are responsible for applying the target state to the infrastructure

→ GitOps is not only applicable within the IT industry -
it can also be very beneficial for embedded and IoT use cases!

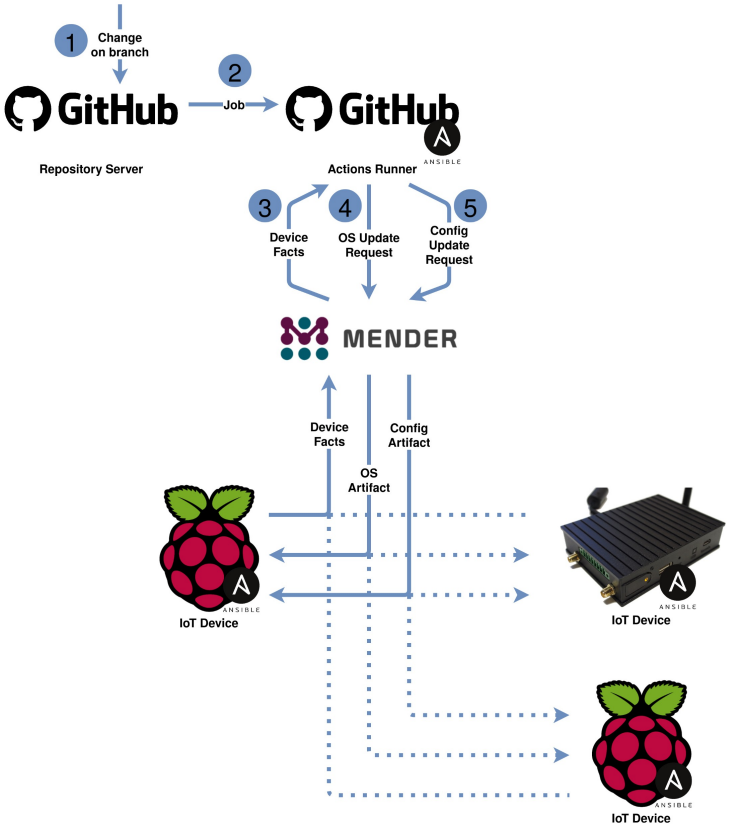
GitOps

Map the fleet to a git repository



GitOps

How it works behind the scene



Workflow

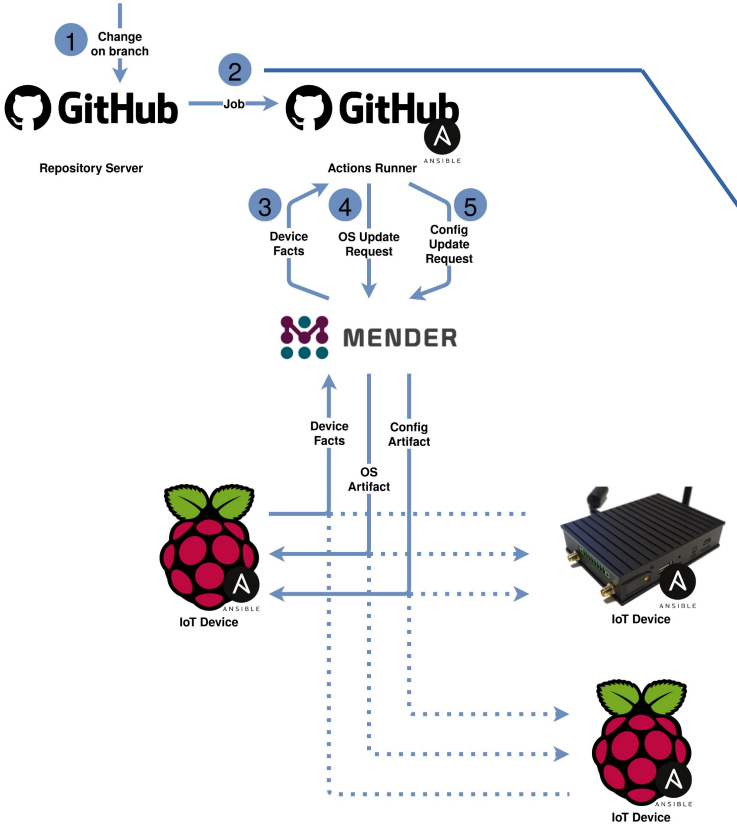
1. A branch gets modified:
   develop/feature branch: commit
   main/canary/production branch: merge
2. GitHub dispatches a job to a runner ([1])
   and the runner clones the fleet repository ([2], [3], [4])
3. The fleet facts get retrieved from Mender
4. OS update requests get scheduled ([5])
5. Configuration update requests get scheduled

Key Principles

- Idempotency
- Traceability
- Staged roll outs
- From main branch and upwards no changes
- Proxy between management server and fleet

GitOps

Already familiar tools take care of the orchestration



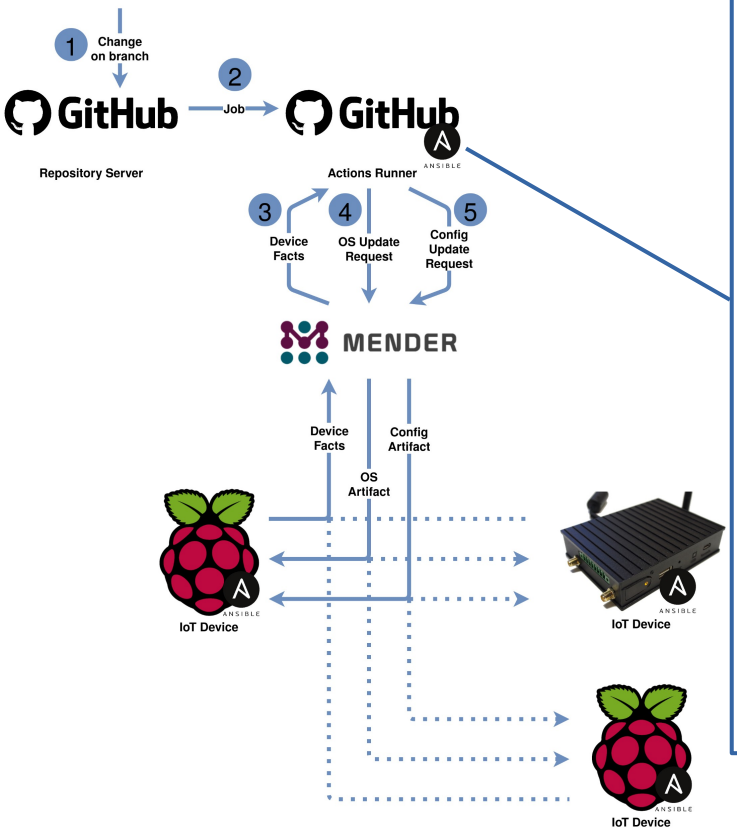
Workflow

1. A branch gets modified:
   develop/feature branch: commit
   main/canary/production branch: merge
2. GitHub dispatches a job to a runner ([1])
   and the runner clones the fleet repository ([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
```

GitOps

An Ansible playbook takes care of the fleet

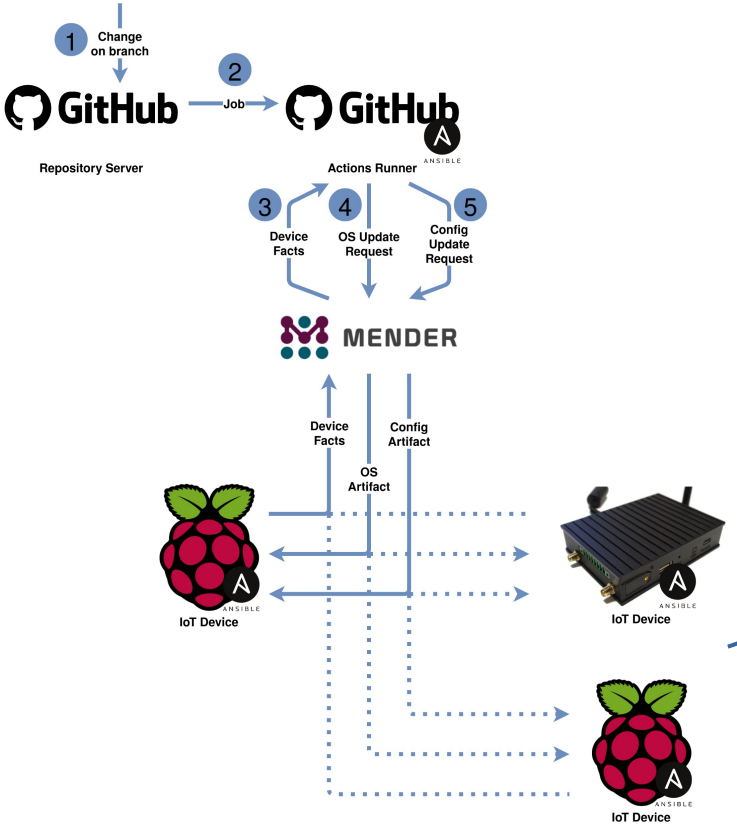



```yaml
---
- name: Apply OS and configuration to fleet.
  hosts: all
  gather_facts: false

  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
```

- Proxy between management server and fleet

GitOps

The inventory of the fleet



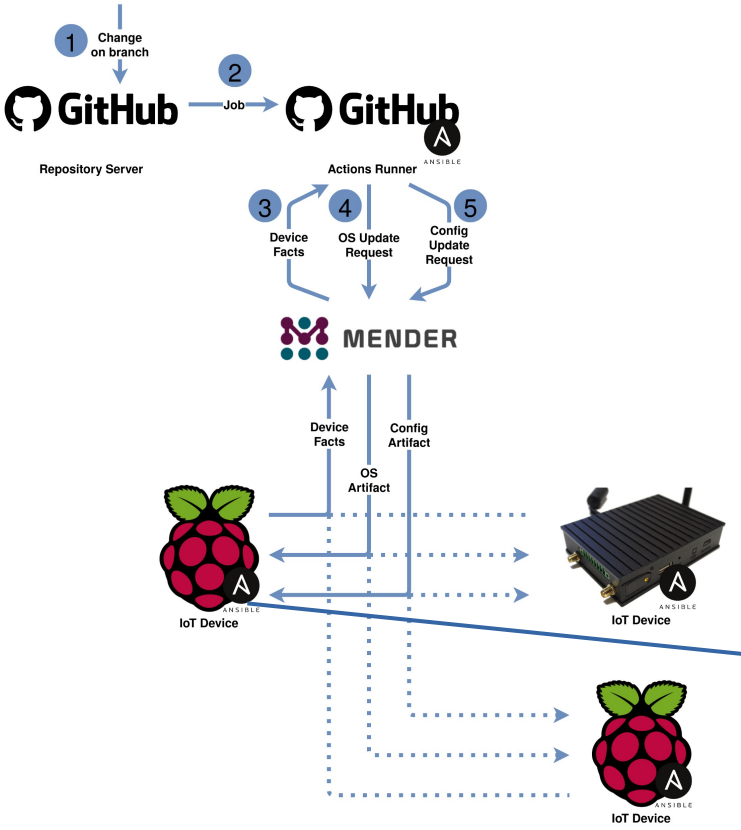
Workflow

1. A branch gets modified:
   develop/feature branch: commit
   main/canary/production branch: merge
2. GitHub dispatches a job to a runner ([1])
   and the runner clones the fleet repository ([2], [3], [4])
3. The fleet facts get retrieved from Mender

```
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:
```

GitOps

An individual device configuration



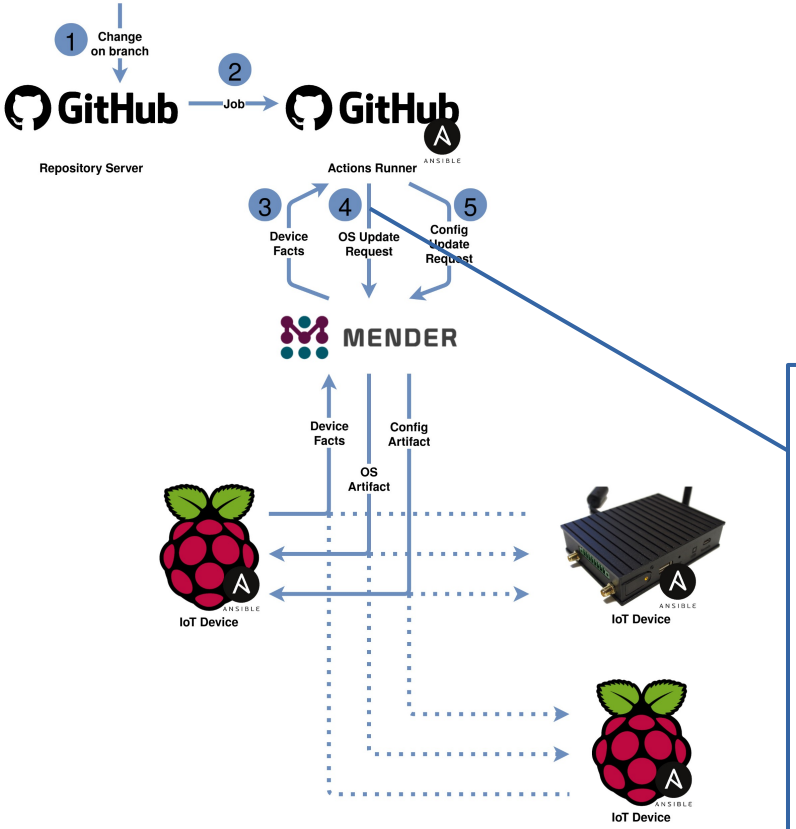
Workflow

1. A branch gets modified:
   develop/feature branch: commit
   main/canary/production branch: merge
2. GitHub dispatches a job to a runner ([1])
   and the runner clones the fleet repository ([2], [3], [4])
3. The fleet facts get retrieved from Mender
4. OS update requests get scheduled ([5])
5. Configuration update requests get scheduled

```yaml
---
subscribed_branch: main

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

GitOps

Eventually an OS update will get dispatched



Workflow

1. A branch gets modified:
   develop/feature branch: commit
   main/canary/production branch: merge
2. GitHub dispatches a job to a runner ([1])
   and the runner clones the fleet repository ([2], [3], [4])
3. The fleet facts get retrieved from Mender
4. OS update requests get scheduled ([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
Some remarks

- The important *monitoring* aspect is out of scope of this presentation!
- On a large fleet the *inventory* and the *individual device configurations* would be offloaded to a separate tool/database.

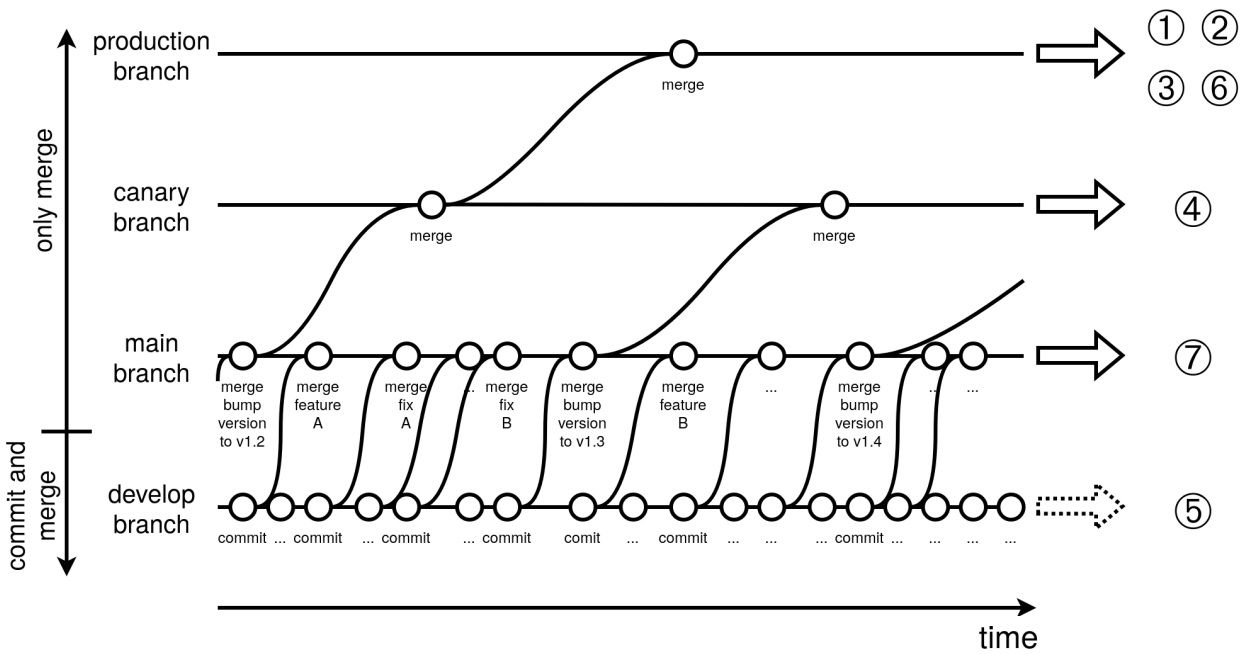
```
all:
  children:
    pi4:
      hosts:
        b8b311de-000e-4914-9a13-1d7e2e23bc5d: # GitHub runner
        3fb4632b-96b9-475d-ac89-02255bd15b6f:
    pi3:
      hosts:
        50a28c2e-3ee8-4559-a5b9-3ce47c881c5d:
        f4580afc-7195-4c8b-b35a-e0248e6bd894:
    pi2:
      hosts:
        048312b5-0456-47a7-9e83-b636f4c0a689: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

Conclusion

GitOps for Fleet Management

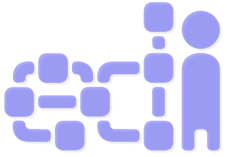
Key benefits I



- Everybody is working on the same git repository/talking the same language
- Full traceability
- No changes introduced beyond the main branch – just merges
- Very high level of automation
- Staged roll outs
- Almost no room for human errors

GitOps for Fleet Management

Key benefits II



GitHub

debian

- Powerful toolbox
- Suitable for a huge fleet
- Components are proven in use
- Components are exchangeable
- Fun to work with

Git Repositories

CI orchestration

edi-ci/edi-ci-public

OS Setup

edi-pi

edi-var

edi-cl

Continuous Integration

Build an OS image for an IoT device, dispatch it to a device and test it

Playbooks/Roles

kiosk-playbook

ansible-kiosk

edi-gh-actions-runner-playbook

ansible-github_actions_runner

edi_installer

Device Management

Adjust an IoT device for an individual use case

CD Orchestration

edi-cd

Continuous Delivery

Keep an entire IoT fleet up to date using git

Links

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

Q&A