UNIKASSEL VERSITÄT

VERTEILTE SYSTEME

Automatisiertes Aufsetzen eines Kubernetes-Clusters auf Raspberry Pis mithilfe von Ansible-Playbooks

Seminararbeit von

KL

Matrikelnummer:

-

Vorgelegt im Fachgebiet

Verteilte Systeme

Betreuer: RH
Betreuer: HB

25. Mai 2020

Universität Kassel

Fachbereich Elektrotechnik und Informatik

Wintersemester 2019/2020

Erklärung

Hiermit versichere ich, dass ich die vorliegende Seminararbeit selbstständig verfasst und nur die angegebenen Hilfsmittel und Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen sind, habe ich als solche kenntlich gemacht. Diese Seminararbeit wurde bis jetzt noch nicht veröffentlicht und wurde bisher noch in keinem anderen Prüfungsamt vorgelegt.

Kassel, den 25. Mai 2020 _____

Inhaltsverzeichnis

1 Einleitung

Dienste von IoT-Projekten stellen hohe Anforderungen an die Infrastruktur, auf der sie laufen. Große Datenmengen, geringe Latenzen oder Hochverfügbarkeit sind Herausforderungen, die sich nicht oder nur kostenintensiv mit klassischem Hosting im Internet oder in der Cloud bewältigen lassen. Eine Alternative stellt der Betrieb der Dienste in einem lokalen Netz dar.

Kubernetes[1] ist eine moderne Technologie, die skalierbare Applikationen ermöglicht. Sie beruht auf dem Prinzip, mehrere Rechner miteinander zu vernetzen und ihre gesamten Ressourcen effizient zu nutzen. Eine besonders günstige Option stellen hierbei Einplatinencomputer wie der Raspberry Pi[2] dar.

Es sind viele Schritte nötig, um einen Kubernetes-Cluster einzurichten und je mehr Worker-Nodes eingerichtet werden sollen, umso häufiger müssen die immer gleichen Schritte durchgeführt werden. Mithilfe des Automatisierungs-Werkzeugs Ansible[3] können diese Aufwände automatisiert und somit vereinfacht und beschleunigt werden. Nachdem mit wenigen Handgriffen das Standardbetriebssystem Raspbian[4] installiert wurde, werden alle weiteren Schritte von Ansible-Playbooks automatisch erledigt.

In dieser Seminararbeit werden zunächst die verwendeten Technologien, Kubernetes und Ansible, kurz vorgestellt (Kapitel 2). Anschließend wird die Vorgehensweise zum Aufsetzen eines Clusters mithilfe der Playbooks übersichtlich zusammengefasst (Kapitel 3). Danach erfolgt eine ausführliche Erläuterung der Funktionsweise der Playbooks (Kapitel 4). Zuletzt werden mögliche Alternativen zu den eingesetzten Technologien vorgestellt (Kapitel 5) und ein Ausblick auf mögliche Weiterentwicklungen gegeben (Kapitel 6).

[1] `https://kubernetes.io/` – 25. Mai 2020
[2] `https://www.raspberrypi.org` – 25. Mai 2020
[3] `https://www.ansible.com/` – 25. Mai 2020
[4] `https://www.raspbian.org/` – 25. Mai 2020

2 Technologien

In diesem Kapitel werden die eingesetzten Technologien vorgestellt. Nacheinander werden die Automatisierungs-Software Ansible, die Virtualisierungs-Software Docker sowie die Orchestrierungs-Software Kubernetes eingeführt. Im Kapitel 4 wird anschließend beschrieben, wie sich ein Kubernetes-Cluster, der Docker-Container verteilt und redundant laufen lassen und skalieren kann, mithilfe von Ansible automatisiert aufsetzen lässt.

2.1 Ansible

Ansible ist eine Software zur Automation von IT-Prozessen und Konfigurationsverwaltung.

Ein mit Ansible ausgerüsteter Steuer-Computer verbindet sich per SSH oder über andere Remote-Protokolle mit anderen Computern und führt auf ihnen Befehle aus, die einen Zielzustand herstellen sollen. Dieser Zielzustand wird in sogenannten *Playbooks* definiert. Dabei handelt es sich um Textdateien im YAML-Format.

Sie bestehen im Wesentlichen aus einer Folge von *Tasks*, die auf vordefinierte Ansible-Module zurückgreifen oder explizit auszuführende Terminal-Befehle enthalten. *Inventory*-Dateien werden angelegt, um die zu verwaltende Infrastruktur zu beschreiben. Dafür werden die zu steuernden Hosts aufgelistet und gegebenenfalls kategorisiert. Playbooks können bei der Ausführung dann auf einzelne Hosts oder bestimmte Gruppen beschränkt werden.

Auf den Remote-Hosts ist außer einem aktiven SSH-Dienst und einem Python-Interpreter keine weitere Software erforderlich. Insbesondere wird kein Ansible-eigener Agent oder ähnliches benötigt, sodass der Aufwand zur Vorbereitung der Hosts für die Verwaltung mit Ansible gering ausfällt.

2.2 Docker

Docker ermöglicht Virtualisierung von Software.

In einem *Docker-Container* laufen Programme isoliert von den Ressourcen das Host-Systems. Neben Sicherheitsaspekten trägt dies auch zu einer erhöhten Portabilität von

Anwendungen bei. Mehrere Dienste, ihre Konfiguration und Abhängigkeiten zu installierten Bibliotheken lassen sich bündeln und auf unterschiedlichen Host-Systemen reproduzierbar ausführen. Ein Container entsteht durch die Instanziierung eines *Docker-Images*. Diese sind vergleichbar mit einer Bauanleitung für das System, das im Container laufen soll.

In öffentlichen Verzeichnissen wie Docker Hub stehen zahlreiche Images zur Verfügung, die als Grundlage für eigene, darauf aufbauende Images dienen.

2.3 Kubernetes

Kubernetes ermöglicht den ausfallsicheren Betrieb virtualisierter Anwendungen mit automatischer Skalierung.

Ein Kubernetes-Cluster besteht aus mehreren, vernetzten Rechnern (*Nodes*), die jeweils mit Docker oder einer vergleichbaren Virtualisierungssoftware ausgestattet sind. Neben einer Vielzahl an *Worker-Nodes* hat der Cluster eine Kontrollebene, die auf einem *Master-Node* läuft.

Diese Kontrollebene verwaltet die Worker-Nodes und alle darauf laufenden Dienste. Sie kann Dienste mit einer vorgegebenen Anzahl an Instanzen auf die Worker-Nodes verteilen, die Ausführung überwachen und bei Bedarf (beispielsweise bei einem Hardwareausfall) automatisch neue Instanzen hochfahren. Anfragen an die im Cluster laufenden Dienste werden ebenfalls von der Kontrollebene entgegengenommen und die Last gleichmäßig auf laufende Instanzen verteilt.

3 Anwendung

Um mithilfe der Playbooks aus diesem Projekt einen Kubernetes-Cluster einzurichten, müssen zunächst die WiFi-Infrastruktur und die Raspberry Pis vorbereitet werden. Dazu werden die SD-Karten einzeln mit Raspbian beschrieben und mit dem Playbook `local-raspbian.yaml` für den Headless-Betrieb[5] konfiguriert, mit Strom versorgt und gestartet. Sobald alle Raspberry Pis online sind, wird Kubernetes mit dem Playbook `kubernetes.yaml` aufgesetzt.

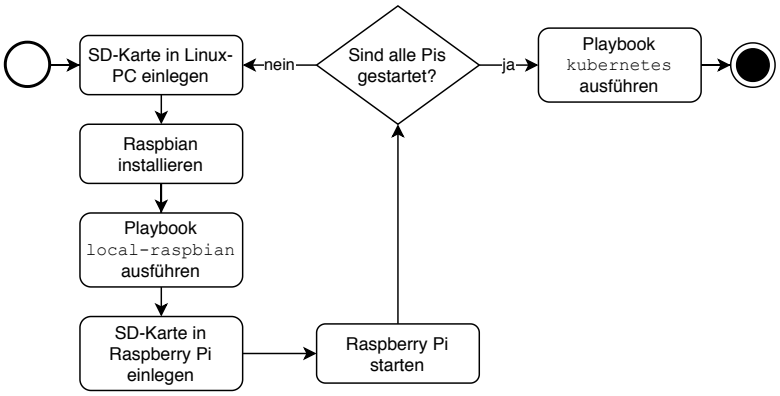

Abbildung 3.1: Ablaufdiagramm zur Einrichtung eines Clusters

3.1 Vorbereitung

Zum erfolgreichen Ausführen dieser Anleitung müssen folgende Voraussetzungen erfüllt sein:

Raspberry Pis in beliebiger Anzahl und ebenso viele Speicherkarten und Netzteile stehen bereit.

Ein Linux-Rechner steht bereit, um die Speicherkarten zu beschreiben und die Ansible-Playbooks auszuführen. Dafür sind Balena Etcher[6] und Ansible[7] installiert, das Git-Repository zu diesem Projekt mit den Verzeichnissen `inventory` und `playbook` ist lokal verfügbar, außerdem ist ein Image von Raspbian Lite[8] heruntergeladen.

[5] Betrieb ohne Bildschirm
[6] `https://www.balena.io/etcher/` – 25. Mai 2020
[7] `https://www.ansible.com/` – 25. Mai 2020
[8] `https://downloads.raspberrypi.org` – 25. Mai 2020

Ein Terminal mit `playbook` als Arbeitsverzeichnis ist geöffnet.

Ein WiFi-Access Point mit Internetzugriff ist in Betrieb. Seine Einstellungen (IP, SSID, WPA2-Key) entsprechen den Angaben in den Dateien `inventory/group_vars/all.yaml` und `playbook/local-raspbian.yaml`. Der zuvor erwähnte Linux-Rechner ist mit dem Access Point verbunden.

Die Inventory-Datei `inventory/k8s-cluster.yaml` bildet den derzeitigen Cluster ab – enthält also keine Einträge unter `hosts`, falls ein neuer Cluster eingerichtet werden soll:

```
nodes:
  hosts:
```

Codeausschnitt 3.1: Leere Inventory-Datei

Ein SSH-Schlüsselpaar ist generiert. Der private Schlüssel ist auf dem Linux-PC unter `$HOME/.ssh` hinterlegt und der öffentliche Schlüssel im Playbook `local-raspbian.yaml` in dem Array `sshKeys` angegeben.

3.2 Raspbian installieren und Cluster einrichten

Die folgenden Schritte müssen für jeden Raspberry Pi einzeln durchgeführt werden.

1. Speicherkarte in den Linux-Rechner einlegen.
2. Raspbian-Image mit Balena Etcher auf der Speicherkarte installieren.
3. Raspbian-Playbook ausführen:

   ```
   sudo ansible-playbook -i ../inventory/k8s-cluster.yaml local-raspbian.
       yaml
   ```

4. Speicherkarte in den Raspberry Pi einsetzen und starten.

Wenn alle Raspberry Pis gestartet sind, wird die Installation mit dem Kubernetes-Playbook fortgesetzt:

```
ansible-playbook -i ../inventory/k8s-cluster.yaml kubernetes.yaml
```

Der Kubernetes-Cluster ist anschließend einsatzbereit.

3.3 Weitere Nodes hinzufügen

Sollen zu einem fertigen Cluster weitere Nodes hinzugefügt werden, kann auch dafür diese Anleitung ab Abschnitt 3.1 verwendet werden. Die Inventory-Datei darf dann nicht leer sein, sondern muss die bereits vorhandenen Nodes enthalten.

4 Umsetzung

Um eine maximale Automatisierung zu erreichen, werden so viele Arbeitsschritte wie möglich von Ansible-Playbooks übernommen. Zwei relevante Playbooks übernehmen unterschiedliche Aufgaben und unterscheiden sich grundsätzlich in ihrer Funktionsweise: Bevor mit `kubernetes.yaml` die eigentliche Einrichtung des Clusters per gleichzeitigem Zugriff auf die Raspberry Pis via SSH vorgenommen werden kann, müssen die Systeme mit `local-raspbian.yaml` zunächst für den Headless-Betrieb vorbereitet werden. Dies geschieht ausschließlich über lokale Aktionen mit direktem Zugriff auf das Dateisystem der SD-Karten vom Linux-Rechner aus.

4.1 Raspbian einrichten

Das originäre Raspbian kann sich mangels Zugangsdaten (SSID und WPA-Key) nicht mit dem WiFi verbinden. Gewöhnlicherweise werden diese Daten nach dem ersten Systemstart per Hand eingegeben. Da dieser Schritt auf jedem einzelnen Raspberry Pi durchgeführt werden müsste und das Anschließen eines Monitors und einer Tastatur erfordern würde, entsteht dabei ein Aufwand, der sich durch Automatisierung vermeiden lässt.

Die WiFi-Konfiguration kann alternativ vorgenommen werden, indem die entsprechende Konfigurationsdatei von Raspbian direkt auf der SD-Karte angepasst wird. Da die SD-Karte ohnehin zunächst an einem separaten PC mit einem System-Image beschrieben werden muss, bietet sich dieser Zeitpunkt an, um weitere Konfigurationen vorzunehmen. Neben der Einrichtung des kabellosen Netzwerks können hier auch weitere Schritte erledigt werden, die in den folgenden Unterkapiteln näher beschrieben werden. Die Reihenfolge kann dabei - mit Ausnahme des Mountens und Unmountens der Partitionen - beliebig geändert werden.

Für diese Schritte wurde das Playbook `local-raspbian.yaml` entwickelt. Da der Rasperry Pi zum Ausführungszeitpunkt dieses Playbooks nicht läuft, werden sämtliche enthaltenen Tasks auf dem Ansible-Host ausgeführt, nicht auf Remote-Hosts. Alle Aktionen erfolgen mittels direktem Zugriff auf das Dateisystem, anstatt ein laufendes System anzusprechen. Dadurch beschränken sich die nutzbaren Ansible-Module auf jene, die das Dateisystem betreffen.

4.1.1 Partitionen mounten

Ein Raspbian-System besteht aus zwei Partitionen: eine Hauptpartition (`rootfs`) und eine Boot-Partition, die einen Zugriff auf häufig benötigte Einstellungen ermöglicht, ohne den Raspberry Pi erst starten zu müssen. Beide Partitionen tragen eine eindeutige UUID, über die sie im Playbook zunächst mithilfe des Ansible-Moduls `mount` gemountet werden.

4.1.2 Statische IP-Adresse setzen

Standardmäßig werden IP-Adressen dynamisch über DHCP bezogen. Da Kubernetes feste IP-Adressen voraussetzt, wird stattdessen eine statische IP-Adresse vergeben. Dafür wird das Ansible-Inventory ausgelesen, auf die höchste bisher vergebene IP-Adresse 1 addiert und die resultierende Adresse sowie die Adresse des Routers und die Subnetz-Maske in die Datei `/etc/dhcpcd.conf` geschrieben. Hierfür kommt das Ansible-Module `lineinfile` zum Einsatz (siehe Codausschnitt 4.2).

```
nodes:
  hosts:
    raspi1:
      ansible_host: "{{ ipSubnet }}101"
    raspi2:
      ansible_host: "{{ ipSubnet }}102"
    raspi3:
      ansible_host: "{{ ipSubnet }}103"
```

Codeausschnitt 4.1: Ansible-Inventory mit drei Hosts

```
- name: Configure static IP address
  lineinfile:
    path: /mnt/{{ rootfsPartition }}/etc/dhcpcd.conf
    state: present
    line: "{{ item }}"
  with_items:
    - interface wlan0
    - static ip_address={{ ipSubnet }}{{ newIp }}/24
    - static routers={{ ipSubnet }}1
    - static domain_name_servers={{ ipSubnet }}1
```

Codeausschnitt 4.2: Ansible-Task zur Einrichtung einer statischen IP-Adresse

Zusätzlich wird in Abhängigkeit von der IP-Adresse ein sprechender Name als Hostname gewählt und dieser in den Dateien `/etc/hostname` und `/etc/hosts` eingetragen.

4.1.3 SSH-Daemon aktivieren

Raspberry Pis werden oft für IoT-Projekte verwendet und dabei im Internet öffentlich zugänglich gemacht. Da auch das Standardpasswort oft nicht geändert wird, ist aus Si-

cherheitsgründen der SSH-Dienst in Raspbian standardmäßig deaktiviert. Er wird jedoch von Ansible benötigt. Der in Raspbian vorinstallierte Daemon lässt sich aktivieren, indem eine inhaltsleere Datei mit dem Namen `ssh` auf der Boot-Partition angelegt wird. Ansible ermöglicht das Anlegen von Dateien mittels des `file`-Moduls und der Option `state: touch`.

4.1.4 WiFi konfigurieren

Damit ein WiFi-Gerät wie der Raspberry Pi eine Verbindung mit einem Access Point herstellen kann, müssen der Name des Netzwerks (SSID) und der Zugangsschlüssel konfiguriert werden. Raspbian liest diese Informationen aus der Datei `/etc/wpa_supplicant/wpa_supplicant.conf`. Mithilfe von `lineinfile` wird der Inhalt der zu Beginn des Playbooks definierten Variablen `ssid` und `psk` dort hinterlegt.

Zusätzlich wird das Land definiert, in dem das Gerät betrieben wird, damit das System die korrekten Frequenzbänder nutzt[9]. Ohne diese Konfiguration ist das WiFi-Modul nicht betriebsfähig. Raspbian registriert den Eintrag in der Konfigurationsdatei jedoch nicht ohne Weiteres. Daher ist zusätzlich ein Eintrag in der Datei `/etc/rc.local` nötig, wodurch bei jedem Systemstart der Befehl `rfkill unblock wifi` ausgeführt und der WiFi-Adapter freigegeben wird.

4.1.5 Swapfile deaktivieren

Kubernetes ist aus Gründen der Performanz[10] nicht zum Einsatz auf Systemen mit Swap-Speicher vorgesehen. Findet der Dienst eine aktivierte Swap-Partition oder Swap-Datei, wird der Startvorgang mit einer Fehlermeldung abgebrochen. In Raspbian ist standardmäßig eine Swap-Datei aktiviert. Ihre Größe wird über einen Eintrag in der Konfigurationsdatei `/etc/dphys-swapfile` definiert. Indem dort der Wert der Variable `CONF_SWAPSIZE` auf `0` gesetzt wird, wird die Swap-Datei deaktiviert.

4.1.6 Control Groups aktivieren

Docker greift zur Verwaltung von Ressourcen auf Linux Control Groups (cgroups) zurück. Dieses Kernel-Feature ist in Raspbian standardmäßig deaktiviert. Um es zu aktivieren, werden in der Datei `cmdline.txt` auf der Boot-Partition mehrere Kernel-Parameter ergänzt. Dafür wird zunächst mithilfe eines regulären Ausdrucks im Modul `lineinfile` im Check-Mode sichergestellt, dass die Parameter noch nicht vorhanden sind. Gegebenenfalls werden sie anschließend, ebenfalls mit `lineinfile`, hinzugefügt.

[9] `https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md` – 25. Mai 2020

[10] `https://github.com/kubernetes/kubernetes/issues/7294` – 25. Mai 2020

4.1.7 SSH-Keys hinterlegen

Eine Alternative zur standardmäßigen Anmeldung mit Benutzernamen und Passwort ist die Authentifizierung per SSH-Schlüsselpaar. Unter anderem benötigt Ansible dadurch keine Passwörter, die manuell beim Ausführen eines Playbooks eingegeben oder im Inventory hinterlegt werden müssen. Damit ein SSH-Server einen Schlüssel akzeptiert, muss der zugehörige öffentliche Schlüssel im Home-Verzeichnis des Nutzers in der Datei `.ssh/authorized_keys` eingetragen werden. Im Playbook wird dieser öffentliche Schlüssel zu Beginn als Variable definiert und mit diesem Task in die Datei geschrieben.

4.1.8 Partitionen unmounten

Zum Schluss werden die zwei Partition ausgeworfen, sodass die Speicherkarte unmittelbar nach Ausführung des Playbooks sicher entfernt werden kann. Die Karte kann dann in den Raspberry Pi eingelegt, dieser mit Strom versorgt und gebootet werden.

4.2 Kubernetes-Cluster einrichten

Sobald alle einzurichtenden Nodes online sind, kann die Einrichtung des Clusters beginnen. Diese Aufgabe übernimmt das Playbook `kubernetes.yaml`, welches die nötigen Schritte auf allen Nodes simultan ausführt.

4.2.1 Master-Flag setzen

Zunächst wird in diesem Playbook ein Master-Flag gesetzt. Der Wert hängt von der Konfiguration der `masterIp` im Ansible-Inventory ab. Nur für den Host, auf dem später der Kubernetes-Master laufen soll, erhält das Flag den Wert `true`. Mithilfe des Flags werden später einzelne Schritte exklusiv auf dem Master-Node (beispielsweise das Initialisieren des Clusters) oder exklusiv auf den Worker-Nodes ausgeführt.

4.2.2 Docker installieren

Um eine unnötige Neuinstallation von Docker zu vermeiden, wird mit dem Systemaufruf `which docker` über das Modul `command` zunächst geprüft, ob Docker bereits installiert ist. Das Kommando `which` drückt über seinen Rückgabewert aus, wie viele seiner Argumente *nicht* gefunden wurden. Mit der Option `register: whichDocker` wird der Rückgabewert in einer Ansible-Variable registriert, um sie als Bedingung zur Ausführung weiterer Tasks verwenden zu können. Ist Docker noch nicht installiert, ist der Rückgabewert `1`. Rückgabewerte ungleich `0` werden jedoch von Ansible als Fehler interpretiert. Daher ist es nötig, Ansible mit der Option `ignore_errors: yes` anzuweisen, diesen vermeintlichen Fehler zu ignorieren.

Docker stellt ein Installationsscript[11] zur Verfügung, das mithilfe von `curl` heruntergeladen und in der lokalen Datei `get-docker.sh` gespeichert wird. Anschließend wird das Script ausgeführt. Diese beiden Schritte werden durch die Bedingung `when: whichDocker.failed == true` übersprungen, falls Docker bereits installiert ist.

Zuletzt wird mit dem Modul `systemd` sichergestellt, dass der Docker-Daemon gestartet ist.

4.2.3 Kubernetes installieren

Kubernetes wird mit dem Paketverwalter `apt` des Betriebssystems installiert. Die Pakete sind nicht über die offiziellen Apt-Repositorys verfügbar. Daher wird zunächst mit den Ansible-Modulen `apt_key` und `apt_repository` das Kubernetes-Repository hinzugefügt. Anschließend werden die benötigten Pakete mittels `apt` heruntergeladen und installiert. Danach wird mit einem Aufruf des Kommandos `kubeadm init` der Cluster initialisiert. Dies geschieht ausschließlich auf dem Master-Node, indem durch `when: master is defined` eine Abhängigkeit vom Master-Flag geschaffen wird (siehe Abschnitt 4.2.1). Im Anschluss fordert `kubeadm` dazu auf, die generierte Konfigurationsdatei von `/etc/kubernetes/admin.conf` ins Home-Verzeichnis zu kopieren. Diesen Schritt übernimmt Ansibles `copy`-Modul. Danach ist der Master-Node lauffähig.

4.2.4 Kubernetes-Nodes verbinden

Um die Worker-Nodes mit dem Master zu verbinden, generiert der Master einen Join-Befehl. Er enthält die IP-Adresse das Master-Nodes und einen zeitlich begrenzt gültigen Schlüssel. Wird er auf einem Kubernetes-Knoten ausgeführt, baut er mit dem Master eine sichere Verbindung auf, macht sich mit diesem bekannt und wird als Worker-Node zum Cluster hinzugefügt.

Der Join-Befehl wird mit einem Aufruf von `kubeadm token create` auf dem Master-Knoten generiert. Die Ausgabe des Befehls – also der Join-Befehl – wird in der Ansible-Variable `join_command` registriert und in der lokalen Datei `/tmp/join-command` zwischengespeichert. Diese Datei wird wiederum auf die übrigen Knoten kopiert (`copy`) und dort ausgeführt (`command: sh /tmp/join-command.sh`). Anschließend sind alle Nodes dem Cluster beigetreten.

4.2.5 Virtuelles Netzwerk installieren

Kubernetes-Dienste, die auf unterschiedlichen Nodes laufen, sind voneinander isoliert und können unteinander nicht kommunizieren. Um dies zu ermöglichen, benötigt Ku-

[11] `https://get.docker.com` – 25. Mai 2020

bernetes ein virtuelles Netzwerk. Es gibt unterschiedliche Implementierungen, die zusätzlich installiert werden können. Die hier gewählte Option heißt Flannel[12]. Die Installation erfolgt über den Befehl `kubectl apply` mit Angabe einer YAML-Datei, die zur Einrichtung benötigte Informationen enthält. Diese wird von den Flannel-Entwicklern bereitgestellt. Nach Abschluss der Installation ist der Cluster fertig eingerichtet und einsatzbereit.

4.3 Herausforderungen

In diesem Kapitel soll auf Schwierigkeiten eingegangen werden, die bei der Umsetzung auftraten und auch in Zukunft noch relevant sein könnten.

4.3.1 Update von Raspbian

Während der Arbeiten an dem Projekt wurde eine neue Version von Raspbian veröffentlicht. Dadurch ergaben sich mehrere Probleme.

Im Zuge der Veröffentlichung wurden die Namen der Besitzer der Apt-Repositorys geändert. `apt` sieht hierin ein Sicherheitsrisiko und verlangt eine manuelle Bestätigung, um zum Beispiel mit dem Aktualisieren der installierten Pakete fortzufahren. Da das Kubernetes-Playbook eine solche Aktualisierung durchführt, war eine unbeaufsichtigte Ausführung nicht mehr möglich. Es lag also nahe, von vornherein mit aktuellerer Software zu arbeiten und dazu die neue Raspbian-Version ins Projekt zu übernehmen. Dies verlief jedoch nicht reibungslos.

Zum einen haben sich die UUIDs der zwei Partitionen im Image geändert. Da das Playbook `local-raspbian.yaml` diese verwendet, um die Partitionen zu mounten, mussten sie händisch angepasst werden. Es ist zu erwarten, dass dieser Schritt mit jedem neuen Raspbian-Release notwendig ist.

Weiterhin kam nach dem Update keine WiFi-Verbindung mehr zustande. Die Ursache war die fehlende Definition des Landes in der Konfigurationsdatei `wpa_supplicant.conf`. Mit der alten Version war dies noch nicht erforderlich, erst die neue setzte die Konfiguration voraus. Es reichte zudem nicht aus, das Land festzulegen, sondern es musste auch die in Abschnitt 4.1.4 beschriebene Lösung mit `rfkill` eingeführt werden, um das neue Raspbian wieder mit dem WiFi-Netzwerk zu verbinden.

[12] `https://github.com/coreos/flannel` – 25. Mai 2020

5 Alternativen

5.1 Alternativen zu Ansible

Ansible hat mehrere Konkurrenzprodukte, zum Beispiel Puppet[13] und Chef[14].

Alle sind etabliert und eignen sich zur Automatisierung von Konfigurationen über Netzwerkverbindungen. Ein wesentlicher Unterschied besteht in der grundlegenden Funktionsweise. Während Ansible genau dann arbeitet, wenn ein Playbook auf dem Steuerungsrechner ausgeführt wird (Push-Prinzip), setzen Puppet und Chef Agenten-Software auf den Remote-Hosts voraus, die selbst den Zeitpunkt der Konfiguration bestimmen Pull-Prinzip). Dafür muss dauerhaft ein Puppet- beziehunsgsweise Chef-Server bereitstehen, um zum gegebenen Zeitpunkt die nötigen Konfigurationen bereitzustellen. Puppet und Chef eignen sich dadurch eher für Anwendungsfälle, in denen Zeit keine wichtige Rolle spielt oder nicht alle Remote-Hosts gleichzeitig verfügbar sind, zum Beispiel Büro-Rechner, die für begrenzte Dauer und zu unterschiedlichen Zeiten verwendet werden.

In diesem Projekt laufen alle Remote-Hosts gleichzeitig und eine unverzügliche Ausführung der Konfiguration auf Knopfdruck ist erwünscht. Das macht Ansible zu einer guten Wahl.

5.2 Alternativen zu Kubernetes

Kubernetes ist die etablierteste Software zur Orchestrierung von Containern.[15] Eine der Alternativen heißt Docker Swarm[16] und ist Bestandteil von Docker. Die Funktionsumfänge beider Produkte sind ähnlich. Ausfallsicherheit durch Redundanz, Load Balancing, Skalierbarkeit, Monitoring und automatisiertes Ersetzen von fehlgeschlagenen Instanzen findet man auf beiden Seiten.

Die Handhabung von Docker Swarm ist an die von Docker angelehnt und leichter zu erlernen. Die Skalierung erfolgt bei Docker Swarm schneller, allerdings nur manuell. Kubernetes beherrscht automatisierte Skalierung anhand der Auslastung. Während Docker

[13] `https://puppet.com/` – 25. Mai 2020
[14] `https://www.chef.io/` – 25. Mai 2020
[15] `https://platform9.com/blog/kubernetes-docker-swarm-compared/` – 25. Mai 2020
[16] `https://docs.docker.com/engine/swarm/` – 25. Mai 2020

Swarm nur Docker-Container unterstützt, kann Kubernetes auch mit anderer Virtualisierungs-Software wie Podman betrieben werden.

Fleet war ein Cluster-Manager und Bestandteil von Container Linux (früher CoreOS). Die Technologie wird seit 2017 nicht mehr weiterentwickelt. In der Dokumentation wird stattdessen der Einsatz von Kubernetes empfohlen.[17]

[17] `https://coreos.com/fleet/docs/latest/` – 25. Mai 2020

6 Fazit

Die für dieses Projekt entwickelten Playbooks ermöglichen es, mit wenigen Handgriffen in kurzer Zeit einen Kubernetes-Cluster einzurichten. Die benötigte Zeit für die gesamte Einrichtung beträgt circa 10 Minuten pro Raspberry Pi plus etwa 30 Minuten für die Einrichtung des Clusters. Der überwiegende Teil dieser Zeit ist Wartezeit. Der Einrichtungsvorgang läuft fast komplett selbstständig ab. Eine manuelle Einrichtung des Clusters nimmt mehrere Stunden in Anspruch.

Der Ersparnis von wenigen Stunden bei der Anwendung der Ansible-Playbooks muss aber auch der Aufwand gegenübergestellt werden, der nötig ist, um diese Playbooks zu entwickeln. In meinem Fall waren dafür weit über zehn Stunden nötig. Für die Einrichtung von wenigen oder nur einem Cluster lohnt es sich nicht, die Schritte zu automatisieren. Erst mit der häufigeren Nutzung der Playbooks, gegebenenfalls durch mehrere Nutzer, amortisieren sich die Aufwände.

6.1 Ausblick

An verschiedenen Stellen bietet das Projekt Verbesserungspotenziale. Einige davon sollen nun erläutert werden.

6.1.1 Raspbian mit Ansible installieren

Bislang wird zu Beginn der Einrichtung mit Balena Etcher ein separates Werkzeug eingesetzt, um Raspbian auf der Speicherkarte zu installieren (siehe Abschnitt 3.2). Erst danach wird Ansible zur weiteren Einrichtung verwendet.

Es wäre auch möglich, im Ansible-Playbook `local-raspbian.yaml` mit einem Systemaufruf von `dd` das Raspbian-Abbild auf die Speicherkarte zu schreiben. Der vorhergehende Schritt mit Balena Etcher könnte dadurch ersetzt werden. Ein Nachteil wäre, dass der Anwender die Datenträgerbezeichnung (zum Beispiel `/dev/sdb`) selbst angeben müsste.

6.1.2 Bestehenden Kubernetes-Cluster erhalten

Das Ansible-Playbook `kubernetes.yaml` richtet stets einen neuen Kubernetes-Cluster ein. Falls schon vor Ausführung des Playbooks ein Cluster existiert, wird dieser gelöscht. In manchen Situationen kann dieses Verhalten unerwünscht sein, zum Beispiel, wenn der Cluster um zusätzliche Nodes erweitert werden soll. Besser wäre es, vor dem Initialisieren des Clusters zu überprüfen, ob bereits ein Cluster existiert und gegebenenfalls die Neueinrichtung zu überspringen.

6.1.3 Globale Ansible-Variablen

Beim Beitritt der Worker-Nodes zum Cluster (siehe Abschnitt 4.2.4) wird der vom Master-Node generierte Join-Befehl in eine Skriptdatei geschrieben, die auf die Worker-Nodes kopiert und dort ausgeführt wird. Diese Vorgehensweise ist nicht optimal.

Effizienter wäre es, den Befehl in einer Ansible-Variable zu registrieren und ihn dann direkt auf den Worker-Nodes auszuführen. Die Herausforderung dabei ist, dass Variablen in Ansible grundsätzlich nicht Host-übergreifend verwendet werden können. Ansible-Facts sollten es ermöglichen, diesen Umstand zu umgehen. Dies hat aber zum Zeitpunkt der Umsetzung dieses Projekts nicht funktioniert.