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

3. April 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 3. April 2020 _____

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Kubernetes 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. In Produktivumgebungen werden dazu leistungsstarke Maschinen oder Cloud-Instanzen eingesetzt. Im Entwicklungsbetrieb kann es jedoch von Vorteil sein, aus finanziellen Gründen auf die Leistungsfähigkeit und die gute Anbindung eines Rechenzentrums zu verzichten und stattdessen auf lokal betriebene Systeme zu setzen. Eine besonders günstige Option stellen hierbei Einplatinencomputer wie der Raspberry Pi dar.

Es sind viele Schritte nötig, um einen Kubernetes-Cluster einzurichten und je mehr Nodes eingerichtet werden sollen, umso häufiger müssen die immer gleichen Schritte durchgeführt werden. Mithilfe des Automatisierungs-Werkzeugs Ansible können diese Aufwände automatisiert und somit vereinfacht und beschleunigt werden. Nachdem mit wenigen Handgriffen das Standardbetriebssystem Raspbian 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).

2 Technologien

2.1 Ansible

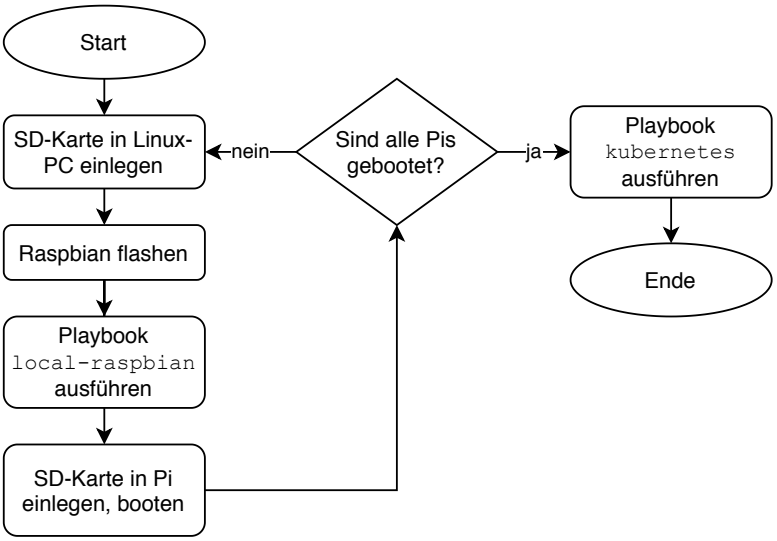
Ansible macht Sachen automatisch

2.2 Kubernetes

Kubernetes schubst Container

3 Anwendung

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



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 flashen und die Ansible-Playbooks auszuführen. Dafür sind Balena Etcher[1] und Ansible[2] installiert, das Git-Repo zu diesem Projekt mit den Verzeichnissen `inventory` und `playbook` ist ausgecheckt und ein Image von Raspbian Lite[3] ist heruntergeladen.

[1] `https://www.balena.io/etcher/` – 3. April 2020
[2] `https://www.ansible.com/` – 3. April 2020
[3] `https://downloads.raspberrypi.org` – 3. April 2020

- **Ein Terminal** mit `playbook` als Current Working Directory 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 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:
  ```

  Leere Inventory-Datei

Die benötigte Zeit für die gesamte Einrichtung beträgt ca. 10 Minuten pro Raspberry Pi plus ca. 30 Minuten für den gesamten Cluster.

3.2 Raspbian installieren

Die Schritte in diesem Abschnitt müssen für jeden Raspberry Pi einzeln durchgeführt werden.

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

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

   CLI-Befehl zur Ausführung des Raspbian-Playbooks
4. Speicherkarte in den Raspberry Pi einsetzen und booten.

3.3 Cluster aufsetzen

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

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

CLI-Befehl zur Ausführung des Raspbian-Playbooks

Der Kubernetes-Cluster ist anschließend einsatzbereit.

3.4 Weitere Nodes hinzufügen

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

4 Umsetzung

Um das Ziel der maximalen 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 Nodes via SSH vorgenommen werden kann, müssen die Systeme mit `local-raspbian.yaml` zunächst auf den Headless-Betrieb vorbereitet werden. Dies geschieht ausschließlich über lokale Aktionen mit direktem Zugriff auf das Dateisystem der SD-Karten.

4.1 Raspbian einrichten (`local-raspbian.yaml`)

Das unveränderte Raspbian kann sich mangels Zugangsdaten (SSID und WPA-Key) nicht mit dem WiFi verbinden. Gewöhnlicherweise werden diese Daten nach dem ersten Start des Systems per Hand eingegeben. Da dieser Schritt aber auf jedem einzelnen Raspberry Pi durchgeführt werden muss und zudem das Anschließen eines Monitors und einer Tastatur erfordert, entsteht dabei ein nicht zu vernachlässigender Aufwand, der sich vermeiden lässt.

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

Für diese Schritte gibt es ein eigenes Playbook. Da der Rasperry Pi zum Ausführungszeitpunkt dieses Playbooks nicht läuft, werden sämtliche Tasks darin nur auf dem Ansible-Host ausgeführt, nicht auf Remote-Hosts; und alle Aktionen erfolgen mittels direktem Eingriff ins Dateisystem, anstatt die Shell eines laufenden Systems anzusprechen. Dadurch beschränken sich die nutzbaren Ansible-Module auf jene, die das Dateisystem

betreffen. Das Playbook trägt das Namenspräfix `local-`, um diesen Umstand zu signalisieren.

4.1.1 Partitionen mounten

Ein Raspbian-System besteht aus zwei Partitionen: eine Hauptpartition (`rootfs`), deren Struktur der eines gewöhnlichen Linux-Systems entspricht und eine Boot-Partition, die dank ihres plattformübergreifend unterstützten FAT-Dateisystems leichten Zugriff auf häufig benötigte Einstellungen ermöglicht, ohne den Raspberry Pi erst booten 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 das Ansprechen über Hostnamen vom Netzwerk abhängt und nicht immer zuverlässig funktioniert, 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.

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

Aus Sicherheitsgründen ist der SSH-Dienst von Raspbian von Haus aus deaktiviert. Er wird jedoch von Ansible benötigt. Da SSH ein häufig genutztes Feature ist, lässt sich der Daemon in Raspbian unkompliziert aktivieren. Dazu muss lediglich eine inhaltsleere Datei `ssh` auf der Boot-Partition angelegt werden. 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[1]. Ohne diese Konfiguration verweigert das WiFi-Modul den Dienst. 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 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 und Kubernetes greifen 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 die Kernel-Parameter `cgroup_enable=cpuset cgroup_enable=memory cgroup_memory=1` 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.

4.1.7 SSH-Keys hinterlegen

Die Authentifizierung per SSH-Schlüsselpaar kann aus verschiedenen Gründen eine attraktive Alternative zur klassischen Anmeldung mit Benutzernamen und Passwort darstellen. 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 nun in diese Datei geschrieben.

[1] `https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md`

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 nun in den Pi eingelegt und dieser mit Strom versorgt und gebootet werden.

4.2 Kubernetes-Cluster einrichten (`kubernetes.yaml`)

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 können später einzelne Schritte exklusiv auf dem Master-Node bzw. auf allen anderen Nodes ausgeführt werden, beispielsweise das Initialisieren des Clusters.

4.2.2 Zeitzone korrigieren

Raspbian befindet sich werksseitig nicht in der korrekten Zeitzone. Das Ansible-Modul `timezone` schafft mit dem Parameter `name: Europe/Berlin` Abhilfe.

4.2.3 apt update && apt upgrade

Mit dem Modul `apt` werden Software-Pakete auf den aktuellen Stand gebracht.

4.2.4 Docker installieren

Um eine unnötige Neuinstallation 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. Ist Docker also noch nicht installiert, ist der Rückgabewert `1`. Jedoch werden Rückgabewerte ungleich `0` als Fehler interpretiert. Daher ist es nötig, Ansible mit der Option `ignore_errors: yes` anzuweisen, diesen vermeintlichen Fehler zu ignorieren.

Docker stellt unter `https://get.docker.com` ein Installationsscript 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.

4.2.5 Kubernetes installieren

Kubernetes wird über `apt` 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 (vgl. 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.

4.2.6 Kubernetes-Nodes verbinden

Um die restlichen Knoten mit dem Master zu verbinden, kann der Master einen Join-Befehl generieren. 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 Arbeiterknoten 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.

4.2.7 Virtuelles Netzwerk installieren

Kubernetes bringt kein eigenes virtuelles Netzwerk für die im Cluster laufenden Dienste mit. Es gibt unterschiedliche Implementierungen, die zusätzlich installiert werden können. Die hier gewählte Option heißt Flannel. Die Installation erfolgt über den Befehl `kubectl apply` mit Angabe einer YAML-Datei, die zur Einrichtung benötigte Informationen enthält. Nach Abschluss der Installation ist der Cluster fertig eingerichtet und einsatzbereit.

4.3 Herausforderungen

In diesem Kapitel soll auf erwähnenswerte 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 in Ordnung, 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 kabellos ins Netz zu bringen.

4.3.2 Veraltetes Kubernetes-Paket

Das installierte Kubernetes-Paket ist für die Ubuntu-Version Xenial Xerus, also 16.04 ausgelegt. Obwohl diese bereits vier Jahre alt ist, gibt es noch kein aktuelleres Release. Das stellt (bisher) kein Hindernis dar, sollte aber zumindest erwähnt werden.

5 Alternativen

5.1 Ansible

Puppet?

Chef?

5.2 Kubernetes

Docker Swarm?

Fleet?

Ubuntu?

6 Zusammenfassung

6.1 Ausblick

Image flashen per Ansible

Globale Variablen (kubeadm join)

6.2 Fazit

Alles cool