DevOps Position Interview Task

Ahmad Reza Parsi Zadeh

Introduction

In this task, firstly, I am going to describe **what Cloud-init is**, **what QEMU is used for**, **developing scripts for VM and templates creation, how to shrink and the size of the mentioned template, describing what webhook is and its implementation, and describing our steps during this task.**

What is Cloud-init?

Cloud-init is a service used for customizing Linux-based operating systems in the cloud. It allows you to customize virtual machines provided by a cloud vendor by modifying the generic OS configuration on boot. Canonical initially developed cloud-init for Ubuntu but expanded to most major Linux and FreeBSD operating systems. Cloud-init is a service that sets up the VM instance with the wanted configuration and software ready to use. The service starts at boot and uses the metadata provided by the cloud provider or the direct user.

It does so by executing scripts, most commonly from the **cloud-config** file. Therefore, to change any default settings, you need to edit the cloud-config file on your VM image.

- Relationships between items are defined by **indentations with whitespaces**.
- A **pipe character** (**|**) before a text indicates it should be interpreted as is.
- Text blocks are **indented**.
- A **leading dash** (**-**) identifies members of a list.
- A **colon** (**:**) + **space** + **value** is used to create associative array entries.

What is QEMU

QEMU is a free and open-source emulator. It emulates the machine's processor through dynamic binary translation and provides a set of different hardware and device models for the machine, enabling it to run a variety of guest operating systems.

Customized Ubuntu Images using QEMU and Cloud-Init

In the following, I am going to explain how we can utilize QEMU and Cloud-Init to create and customize an Ubuntu Image. Before commencing, make sure you have already installed QEMU or optionally KVM and packer.

I. Setup File

This file which is written in YAML format, is responsible for setting up our cloud configuration.

- the autoinstall is Ubuntu's AutoInstall / Subiquity configuration section which will set:
  - Keyboard locale to en_US and layout to US
  - Install SSH Server and allow Password login (will be used by Packer)
  - packages will install some necessary packages on first-boot. Install qemu-guest-agent to help out with login
  - late-commands will be triggered at the end of the installation. Install the Bootloader Manager (efibootmgr). Also define the sequence that tells the Boot Manager how it should setup the boot sequence. This will tell the manager where the OS is and when should it be loaded
  - user-data is the actual section where the Cloud-Init configuration takes place.
- The cloud-init configuration above should do the following:
  - change the hostname to packerubuntu
  - set the timezone Asia/Tehran
  - Force change the password for the user called Woody1 through chpasswd
  - Describe which users need to be created:
    - Woody1 should be create with the password packerubuntu
    - Woody1 should be granted sudo access without requirements for password
    - Woody2 should be created with packerubuntu

II. Packer File

Packer configuration will set all the necessary values to download the ISO Image from Ubuntu Artifacts Repository, give the boot command options when the ISO is first booted, tell Ubuntu that it will do an automatic installation rather than anticipating the user to intervene. The Template file tells Packer where to find and download the ISO file.

Finally, the command below should be run on CLI:

“***packer build -force ubuntu.pkr.hcl***”

At the end you will have qcow2 image which you can use to load it on your bare-metal machine or use qemu commands to simply boot it up and test it.

After some usage disk image file sizes grows. Even we delete files in the vm the size in the host system remains the same. Because the host system do not know how much and which part of the disk image file is unused. Shrink is done with convert command and copying existing disk image in to new one.

```
qemu-img convert -O qcow2  ubuntu1.qcow2 ubuntu2.qcow2
```

Another useful feature of the qemu-img is disk compression. Compressing disk images will make more space for other VMs. But compression will little performance loss. Here, I used convert command to compress.

```
qemu-img convert -O qcow2 -c  ubuntu1.qcow2 ubuntu2.qcow2
```

Creating a Linux Template

In this part, I am going to describe the procedure of creating a linux template from scratch.

In this scenario I intend to continue with Ubuntu. As a result, I have to **1. Update** the OS.

Following commands are responsible for this:

```
sudo -i
apt-get update
apt-get upgrade -y
apt-get install -y acpid ntp
reboot
```

For **2. Networking** part, we should set template network interface configuration to DHCP so Cloudstack infrastructure can assign one on boot.

Now It is **3. Host Management.** Here we should set a generic name to the template VM during installation, this will ensure components such as LVM do not appear unique to a machine. It is recommended that the name of "localhost" is used for installation.

```
hostname localhost
echo "localhost" > /etc/hostname
```

One of most important parts, **4. Password Management.** It is a good practice to remove any non root users that come with the OS (such as ones created during the Ubuntu installation). First ensure the root user account is enabled by giving it a password and then login as root to continue. Once logged in as root, any custom user can be removed.

```
deluser myuser --remove-home
```

Other parts should be **5. SSH keys management** which Cloudstack can create key pair and push certificates to instances. Also Cloudstack can push **6. user-data** during instance creation. **7. Partition management,** volumes can autorextend after reboot when partition is extended in the GUI.

8. Cleanup steps should be run when all Main Template configuration is done and just before the shutdown step. After shut down Final template should be created. If the Main Template is started or rebooted before Final template creation all cleanup steps have to be rerun. First, we should remove information unique to the Main Template such as network MAC addresses, lease files and CD block devices, the files are automatically generated on next boot.

```
rm -f /etc/udev/rules.d/70*
rm -f /var/lib/dhcp/dhclient.*
```

At this time, turns into ensure all Templated VMs do not have the same SSH keys, which would decrease the security of the machines dramatically.

```
rm -f /etc/ssh/*key*
```

However, It is good practice to remove old logs from the Main Template.

```
cat /dev/null > /var/log/audit/audit.log 2>/dev/null
cat /dev/null > /var/log/wtmp 2>/dev/null
logrotate -f /etc/logrotate.conf 2>/dev/null
rm -f /var/log/*-* /var/log/*.gz 2>/dev/null
```

Now, we force the user to change the password of the VM after the template has been deployed.

```
passwd --expire root
```

The next step clears the bash commands you have just run.

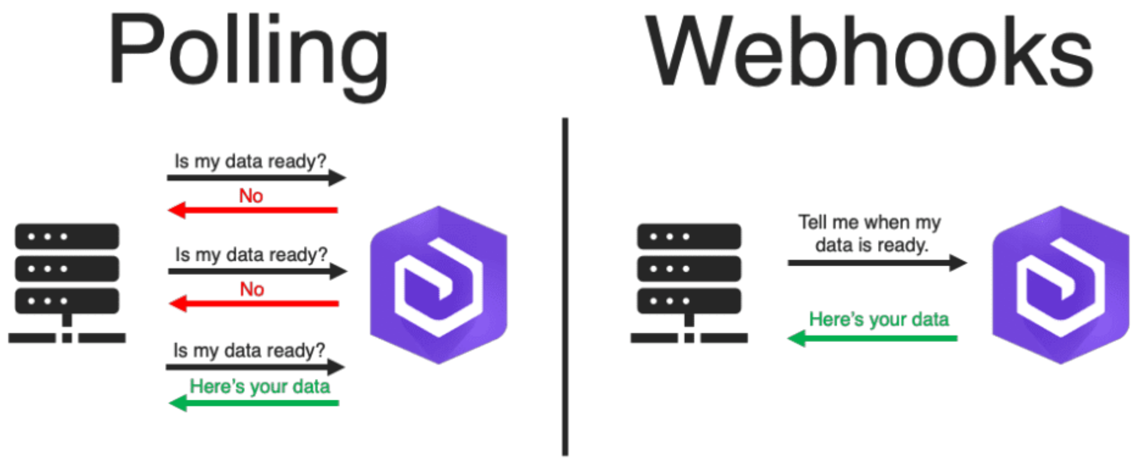
```
history -c
unset HISTFILE
```

For the final step of this process, Shutdown the Main Template.

```
halt -p
```

What is Webhook and its implementation

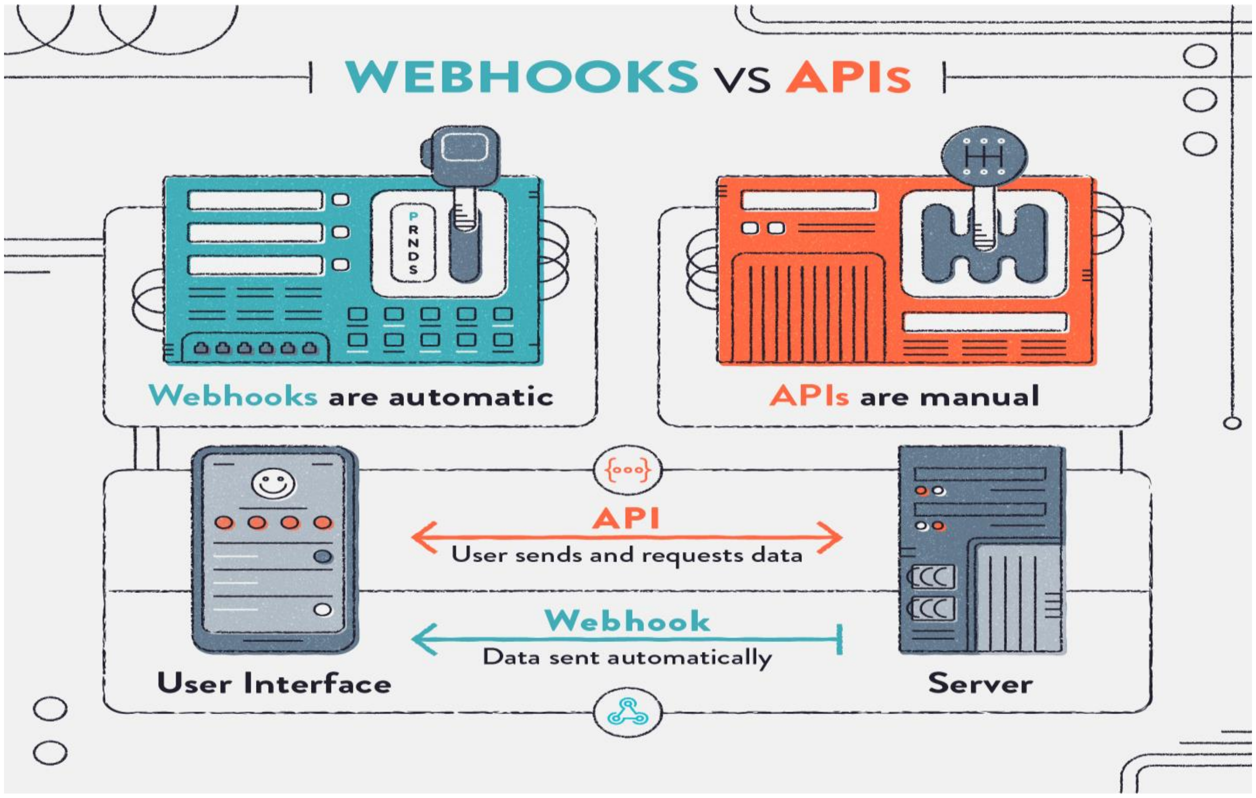
A webhook is an API concept growing in popularity. As more and more of what we do on the web includes events, webhooks are becoming even more applicable. These are incredibly useful and resource-light ways to implement event reactions. a webhook (also called a web callback or HTTP push API) is a way for an app to provide other applications with real-time information. A webhook delivers data to other applications as it happens, meaning you get data immediately—unlike typical APIs where you would need to poll for data very frequently to get it in real-time. This makes webhooks much more efficient for the provider and consumer. The only drawback to webhooks is the difficulty of initially setting them up.



Webhooks are sometimes referred to as reverse APIs because of the ability to give you what amounts to an API spec, and you must design an API for the webhook to use. The webhook will make an HTTP request to your app (typically a POST), and then you'll have to interpret it.

The first step in consuming a webhook is giving the webhook provider a URL to deliver a request. This is often done through a back-end panel or an API, meaning you also need to set up a URL in your app accessible from the public web.

Most webhooks will POST data to you in one of 2 ways: as JSON (typically) or XML (blech) to be interpreted, *or* as form data (application/x-www-form-urlencoded or multipart/form-data). Either way, your provider will tell you how it delivers it (or even give you a choice in the matter). The good news is both of these are fairly easy to interpret, and most web frameworks will do the work for you. If the web frameworks don't, you may need to call on a function or 2.

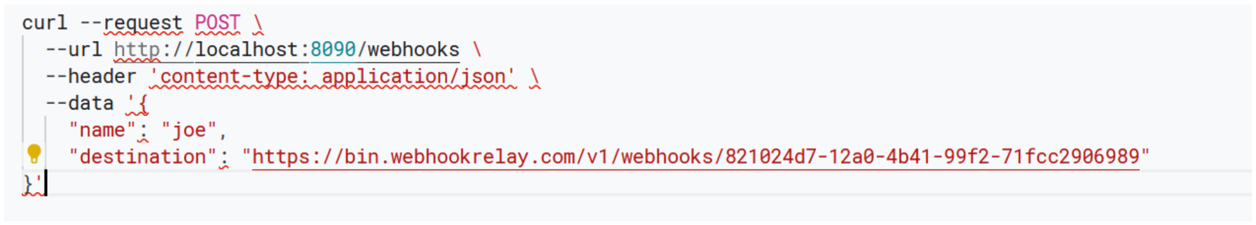


You can find the implementation in the Go programming language in the project file. To use our webhook producer we need an endpoint that can receive and help us debug it. For this task, start by using this free service: https://bin.webhookrelay.com/.
Upon opening the link you should be redirected to a unique URL that is your bin. Get the bin address, you will need it in a minute.

```
$ go run main.go

Create webhooks on http://localhost:8090/webhooks
```

Then, I add the bin:



```
curl --request POST \
  --url http://localhost:8090/webhooks \
  --header 'content-type: application/json' \
  --data '{
    "name": "joe",
    "destination": "https://bin.webhookrelay.com/v1/webhooks/821024d7-12a0-4b41-99f2-71fcc2906989"
}'
```

So, I consume that you can see the requests in your webhooks bin page.

Conclusion

In this project, first I introduced Cloud-init and what it is capable of. then, I continued to QEMU and how we can use QEMU and Packer (especially Hashicorp packer) and YAML files for creating a customized Ubuntu image then I moved forward to the process of creating a Linux template and what is webhook and how is its implementation. In other directories of the project, you can find the necessary scripts for each part of the project and its demands.