6.1 Remote management

Quick instructions

1. Build image: `docker build -t ansible-exercise .`
2. Run image: `docker run -d -p 2222:22 ansible-exercise`
3. Run Ansible: `ansible-playbook playbook.yml -i hosts`

If running with a rebuilt container, you may need to remove `[localhost]:2222` from known host keys (usually in `~/.ssh/known_hosts`) before running Ansible.

The image

In my solution to this exercise, I decided to use the AlmaLinux 8 init image as the base for my Docker container, as it is aimed for running multiple services and comes with systemd enabled.

As visible in the Dockerfile, there are a couple of differences to the example commands; as AlmaLinux doesn't use `apt-get` for package management, dependencies are installed instead with `dnf`. Similarly instead of `sudo`, the `ssluser` user is inserted into the `wheel` group. In addition to the supplied `sed` command, two more were added to configure the SSH server to enable key-based login and block password-based login. Public-key generation was also included, to help the SSH-based Ansible work without issue. Naturally the Ansible user's public SSH key had to be included in the image;, this was done with a `COPY` Dockerfile directive.

As the image has an init system included, there is no need to start a web server – the SSH server runs persistently inside the container.

The playbook

The included Ansible playbook is very simple, consisting of mainly Ansible variable setting and the two required tasks (actually, this became three tasks in my solution): Ensuring the `git` package and checking the host's uptime. I added the third task to simply print the uptime within the playbook's output.

The set variables are used for configuring Ansible to work with minimal manual work; the Ansible user, its password (for `sudo` use), its private SSH key and the correct Python interpreter. The playbook is instructed to use the Paramiko provider for SSH connections, this should prevent Ansible asking to install `sshpass` upon running the playbook.

Results

First container, first playbook run:

```
PLAY [Assignment]***************************************************

TASK [Gathering Facts]*********************************************
ok: [localhost]

TASK [Ensure git]**************************************************
changed: [localhost]

TASK [Get uptime]**************************************************
changed: [localhost]

TASK [Print uptime]************************************************
ok: [localhost] => {
      "msg": "up 4 weeks, 3 days, 6 hours, 5 minutes"
}

PLAY RECAP
********************************************************************
localhost                  : ok=4      changed=2   unreachable=0
failed=0    skipped=0   rescued=0   ignored=0
```

First container, second playbook run:

```
PLAY [Assignment] **************************************************

TASK [Gathering Facts] *********************************************
ok: [localhost]

TASK [Ensure git] **************************************************
ok: [localhost]

TASK [Get uptime] **************************************************
changed: [localhost]

TASK [Print uptime] ************************************************
ok: [localhost] => {
      "msg": "up 4 weeks, 3 days, 6 hours, 8 minutes"
}

PLAY RECAP *********************************************************
localhost                  : ok=4      changed=1   unreachable=0
failed=0    skipped=0   rescued=0   ignored=0
```

Second container, first playbook run:

```
PLAY [Assignment] ************************************************

TASK [Gathering Facts] *******************************************
ok: [localhost]

TASK [Ensure git] ************************************************
changed: [localhost]

TASK [Get uptime] ************************************************
changed: [localhost]

TASK [Print uptime] **********************************************
ok: [localhost] => {
      "msg": "up 4 weeks, 3 days, 6 hours, 15 minutes"
}

PLAY RECAP *******************************************************
localhost                  : ok=4      changed=2   unreachable=0
failed=0    skipped=0   rescued=0   ignored=0
```

Second container, second playbook run:

```
PLAY [Assignment] ************************************************

TASK [Gathering Facts] *******************************************
ok: [localhost]

TASK [Ensure git] ************************************************
ok: [localhost]

TASK [Get uptime] ************************************************
changed: [localhost]

TASK [Print uptime] **********************************************
ok: [localhost] => {
      "msg": "up 4 weeks, 3 days, 6 hours, 16 minutes"
}

PLAY RECAP *******************************************************
localhost                  : ok=4      changed=1   unreachable=0
failed=0    skipped=0   rescued=0   ignored=0
```

Conclusions

The output of task “Ensure git” is expected on two consecutive runs; as the package isn’t present in the image, the first run installs it and the second only checks that it is installed. This is typical of the idempotent and declarative style Ansible is usually used in.

What isn’t expected is the uptime command – if the containers worked like more traditional virtual machines, the output shouldn’t be more than a couple of minutes in this exercise. My first instinct was to assume that the uptime actually comes from the base image that is used; I checked the image’s tag page in Docker Hub and found that the image was pushed on Oct 4, 2022 at 4:08 pm. At the time of writing this report, that is around 26 days in the past, so we are still short by a few days compared to the uptime I got. I didn’t find a simple `FROM` directive in the image’s Dockerfile, so I have no definitive answer as to what causes this uptime. I can however make a guess that this is somehow related to the cycle time of the AlmaLinux project and its process of publishing images.

In the exercise, I found it most difficult to have the SSH server running in the container, as this is not something that is commonly done. However, I found it was a good decision to use the AlmaLinux-init image for this purpose. I already have experience of RHEL-compatible distributions so the modifications to the Dockerfile were not an issue. Using Ansible wasn’t difficult, as I have used the tool prior and didn’t have to rely on tutorials.