vmadm README

2022-06-18 08:31

Contents

1 vmadm README

vmadm is a tool to create and destroy virtual machines running under a local libvirt. Virtual machines are described in specification files, using YAML:

```yaml
foo:
  cpus: 4
  memory_mib: 4096
  image_size_gib: 100

bar:
  cpus: 1
  memory_mib: 512
  image_size_gib: 1
```

All the machines in a specification file are created or destroyed at once.

1.1 Usage

Given a specification file `machines.yaml`, to create virtual machines run:

```
$ vmadm new machines.yaml
```

To delete them:

```
$ vmadm delete machines.yaml
```

Creating a VM creates a disk image of qcow2 format, based on a base image, also of qcow2 format. Deleting the VM deletes the image file as well. Image files are named after the VM and put into the configured image directory, unless the specification file names an image file explicitly.

To get built-in command line help:

```
$ vmadm help
$ vmadm --help
```

A base image is an image with some operating system already installed. It should use [cloud-init][] on first boot to configure hostname and SSH keys, or at least not mind that there is an extra ISO disk with cloud-init configuration attached to the VM. It should open an SSH port when it has booted. Other than that, vmadm doesn't care what it is. For Debian, the pre-made OpenStack cloud-image at https://cloud.debian.org/[1] works well. You need to download the base image yourself, vmadm doesn't do that for you.

1.2 Configuration

The default configuration file is `vmadm/config.yaml` under the XDG configuration directory; by default, this is `~/.config/vmadm/config.yaml`. The configuration file may specify the following fields:

- `default_base_image` – path to the base image to use by default
- `default_image_gib` – default size of new image for a VM, in GiB
- `default_memory_mib` – default amount of memory for a VM, in MiB
- `default_cpus` – default number of CPUs for a VM
- `default_generate_host_certificate` – should SSH host certificates be generated by default?
- `default_networks` – networks to which VM should be added
- `image_directory` – directory where VM image files are put
- `authorized_keys` – list of filenames to SSH public keys, to be put into the default user's `authorized_keys` file in the VM
- `ca_key` – path name to default CA *private* key

1.3 Specification fields

The specification file is YAML and may specify the following fields, all of which override some default from the configuration.

- `ssh_key_files` – overrides `authorized_keys`

[1] `https://cloud.debian.org/`

- `image_size_gib` – overrides `default_image_giv`
- `memory_mib` – overrides `default_memory_mib`
- `cpus` – overrides `default_cpus`
- `base` – overrides `default_base_image`
- `image` – overrides default image file name; must include
- `image` – overrides default image file name; must include path name, is not put into the image directory by default
- `generate_host_certificate` – override host certification setting
- `networks` – networks to which VM should be added; if this field and the `default_networks` field in the config are not specified, add to the `default` network
- `ca_key` – overrides default CA key
- `rsa_host_key` – RSA host key to install on host
- `rsa_host_cert` – RSA host certificate to install on host
- `dsa_host_key` – DSA host key to install on host
- `dsa_host_cert` – DSA host certificate to install on host
- `ecdsa_host_key` – ECDSA host key to install on host
- `ecdsa_host_cert` – ECDSA host certificate to install on host
- `ed25519_host_key` – Ed25519 host key to install on host
- `ed25519_host_cert` – Ed25519 host certificate to install on host

The various `host_key` and `host_cert` fields specify *private* host keys and certificates to be installed in the new VM. The public key is computed from the private key, so there's no need to specify it explicitly. The fields should contain the text of the key or certificate, not its filename.

If *any* host key is specified, no host certificate is generated: the `generate_host_certificate` setting is ignored. If no host keys is specified, an Ed25519 key is generated and signed with the specified CA certificate. The generated key and certificate are installed in the new VM.

In other words, if you specify any host keys, you get to do everything by hand. If you want to keep things easy, don't specify any host keys and let vmadm generate a host key and host certificate for a VM.

1.4 Logging

To turn on logging, set the environment variable `VMADM_LOG`:

```
$ VMADM_LOG=vmadm::libvirt vmadm list
 DEBUG vmadm::libvirt > connecting to libvirtd qemu:///system
 DEBUG vmadm::libvirt > listing all domains
$
```

vmadm uses the `env_logger` Rust library for logging, which is documented at https://docs.rs/env_logger[2]. The environment variable can enable by log level, or by code module, or both. Setting it to `trace` gives the most detailed logging.

[2] `https://docs.rs/env_logger`

2 Using host certificates

Host certificates allow you to access a newly created VM without having to accept its host key. This is especially useful the VM gets recreated and the host key changes. You need to configure your SSH client to trust certificates made with a given SSH CA key, but that is a one-time operation.

You need to create an SSH key used as a CA key for host certification. Run this command:

```
$ mkdir -m 0700 ~/.ssh/ca
$ ssh-keygen -f ~/.ssh/ca/vmadm_ca -t ed25519 -N ''
```

This creates a key **without a passphrase**, because vmadm does not currently support CA keys with passphrases.

Keep the CA key secure. Don't use it for anything else.

Add the following to the `known_hosts` file your SSH client uses, all on one one:

```
@cert-authority * XXXX
```

where `XXX` is the public key part of the CA key, as stored in `~/.ssh/ca/vmadm_ca.pub` in the example above. This tells your client that the CA key on the line should be accepted for all hosts (`*`). You can restrict it to only some hosts if you prefer.

3 Setup of host

The host where vmadm is run needs to have libvirt running and you must have access to the `qemu:///system` connection. The Debian wiki has some useful documentation:

- https://wiki.debian.org/libvirt[3]
- https://wiki.debian.org/KVM[4]

I set up my own libvirt hosts using an Ansible role: http://git.liw.fi/ansibleness/tree/ansible/roles/vmhost[5]. It works on Debian. The short version:

- install
  - `libvirt` (Debian packages `libvirt-daemon-system`, `libvirt-daemon`, `libvirt-daemon`)
  - `virt-install` (Debian package `virtinst`)
  - `qemu-img` (Debian package `qemu-uttils`)
  - NSS lookups for VMs (Debian package `libnss-libvirt`)
  - SSH client (Debian package `openssh-client`)
- make sure you are in the `libvirt` group

[3] `https://wiki.debian.org/libvirt`
[4] `https://wiki.debian.org/KVM`
[5] `http://git.liw.fi/ansibleness/tree/ansible/roles/vmhost`

- edit `/etc/nsswitch.conf` to have `libvirt libvirt_guest` in the `hosts` line

4 Legalese

[6] `COPYING`