Migrating with virsh

First, my setup.

We have a couple of semi-new servers for hypervisors, but we also wanted a couple of backup machines so we could migrate to them, do maintenance, then migrate back. So, a real mixture of cpu's.

All of our backend stores are on an iSCSI server which is **not** managed by libvirt. However, all of the hypervisors are running a fairly recent version of Devuan Linux, which is a version of Debian without System-D, so the paths to the block devices are all the same. Devuan/Debian creates a soft link in /dev/disk/by-path/ with human decipherable names which don't change (so far).

Baseline CPU Definition

The different CPU's cause a problem, so I needed a baseline cpu definition for the virtuals that would work on all of the different hypervisors. Red Hat has a way of doing it, but their explanation stinks, so here is a recipe that works as of today (2022).

By default, if you do not define anything special, your xml file (the virtual definition) will define your cpu's to use whatever capabilities your host has. The code block, inside of your definition, looks like this.

```
--other stuff
<cpu mode='host-model' check='partial'>
  <model fallback='allow'/>
</cpu>
-- other stuff
```

However, this will keep you from migrating if the target machine does not have the same, or a superset of, the capabilities your source has. To get around this, we will replace the above block with a cpu definition that will work on all of the machines.

1. On all hypervisor machines, run the following command. This creates a capabilities file (with your machines hostname as the filename)
    1. `virsh capabilities > `**`hostname`**`.capabilities.xml`
2. Copy all files created to one of the machines
3. Combine the files, then run cpu-baseline on the result

```
cat *.xml > all.xml && virsh cpu-baseline all.xml > common.xml
```

4. Edit each of your virtual definitions, replacing the <cpu></cpu> block with the contents of common.xml.

When I ran this comparing two machines with an Intel Xeon X5650 and a Xeon E5-2430, the result was

```
  <cpu mode='custom' match='exact' check='full'>
    <model fallback='forbid'>Nehalem</model>
    <vendor>Intel</vendor>
    <feature policy='require' name='vme'/>
    <feature policy='disable' name='ds'/>
    <feature policy='disable' name='acpi'/>
    <feature policy='require' name='ss'/>
    <feature policy='disable' name='ht'/>
    <feature policy='disable' name='tm'/>
    <feature policy='disable' name='pbe'/>
    <feature policy='disable' name='dtes64'/>
    <feature policy='disable' name='monitor'/>
    <feature policy='disable' name='ds_cpl'/>
    <feature policy='disable' name='vmx'/>
    <feature policy='disable' name='smx'/>
    <feature policy='disable' name='est'/>
    <feature policy='disable' name='tm2'/>
    <feature policy='disable' name='xtpr'/>
    <feature policy='disable' name='pdcm'/>
    <feature policy='require' name='pcid'/>
    <feature policy='disable' name='dca'/>
    <feature policy='require' name='arat'/>
    <feature policy='require' name='pdpe1gb'/>
    <feature policy='require' name='rdtscp'/>
    <feature policy='require' name='x2apic'/>
    <feature policy='require' name='hypervisor'/>
  </cpu>
```

I literally copied that block, did a *virsh edit* on each of my virtuals, found and deleted the existing <cpu></cpu> block, and pasted the above in.

Now, all my virtuals are set up to use a common set of cpu capabilities.

Get the correct version of netcat

On my Devuan machine, installing *netcat* gives me something that is not compatible with what *virsh migrate* needs. You need the BSD version of netcat, which you can install with the command:

```
apt -y install netcat-openbsd
```

Failing that, you will get an error about an incompatible netcat (nc) which does not have the -U flag.

Set up root access between hypervisors

I like to use ssh, and since I'm in a fairly secure environment, I don't mind giving root on all the machines access to each other without a password. It uses Public Key pairs to do this.

On each machine, run the command

```
ssh-keygen -t rsa -b 4096
```

. Feel free to change the type (rsa) and key length (4096) for your purposes.

Now, copy the contents of /root/.ssh/id_rsa.pub from **each** machine to /root/.ssh/authorized_keys on **all machines**. So, for example, if you have three machines, each will have two lines in their /root/.ssh/authorized_keys file, each line containing the contents of id_rsa.pub from the other two machines.

Test this by logging into each machine as root, then issue the following command for each of the other machines. This simply give a directory listing of /root/, which is all we need for a test.

```
ssh hostname 'ls'
```

If something breaks, fix this before you go further.

Finally, test the virsh connection between machines. Assuming virtlib running on both machines and ssh access, this should work:

```
virsh -c qemu+ssh://hostname/system list
```

. This should return a list of all virtuals running on the TARGET machine.

Notes

- Make sure you have the same bridges defined on all hypervisors, with the same names:

  ```
  virsh net-list
  ```

- Make sure your paths are all the same on all hypervisors:

  ```
  ls /dev/disk/by-path
  ```

- There are several GUI and WebUI tools to do the migration, but the only one I've used (virt-manager) has some serious issues.
  - It assumes you want to copy your block devices to the target, and fails when it tries to do it. However, I was able to shut down on one machine and start up on the second, once I had used –persistent one time to copy the config.
  - I have some machines with the disks encrypted. The message to enter my passphrase doesn't show up on the virt-manager console screen

Test the migration

Ok, you know ssh works, and you have the correct version of cpu definition on all machines, and you have checked the configuration. Now it is time to test. On the source machine:

```
virsh migrate --live --persistent --verbose  runningVirt
qemu+ssh://target/system
```

Replace *runningVirt* with the name of a test virtual you want to migrage, and *target* with the same name you used for your ssh tests. You should see a notification that the virtual is being moved, with a progress. On a very small (4G RAM) under no load, it took about a second.

- **–live** simply says “do this without shutting the virtual down
- **–persistent** copies the virtual's definition to **target** so it can be shut down/started there.
- **–verbose** lets you see what is going on, which I like
- **qemu+ssh:** says to make the move over ssh
- **system** gives you access to the system on the target, allowing migrate to start the machine

There are a lot of additional things you can do with migrate. From the command prompt,

```
man virsh
```

and search for migrate.

Links

- https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/sect-managing_guest_virtual_machines_with_virsh-guest_virtual_machine_cpu_model_configuration
- https://www.geekpills.com/virtulization/kvm-guest-cpu-doesnt-match-specification-missing-features-hlertm

From:
https://wiki.linuxservertech.com/ - **Unix Server Tech Knowledge Base**

Permanent link:
https://wiki.linuxservertech.com/unix/virtualization/virtlib/migrate

Last update: **2022/05/23 02:22**