Fedora Cloud Documentation

Release 0.1

Fedora Cloud Working Group

December 13, 2016

Contents

This repo contains the documentation for Fedora Cloud users.

Contents:

CHAPTER 1

What is Fedora Cloud?

Fedora Cloud provides few different images of Fedora Project which can be consumed in private and public cloud infrastructures. The following list contains the different kind of images available for the users.

Cloud Base This is the minimal image of Fedora, it has the bare minimal packages required to run on any cloud environment.

Atomic Image Atomic image is a lightweight, immutable platform, designed with the sole purpose of running containerized applications. This can also be used in any public or private cloud environment. To learn more you can visit the Project Atomic project page.

Vagrant images We also provide Vagrant images for both cloud base, and atomic. Both VirtualBox, and libvirt is supported by the two different image we publish for Vagrant.

Docker image If you do docker pull fedora, then you will get the latest Fedora image for docker. This image is also created by the Fedora Cloud team.

This Documenation is created with the help of Major Hayden. Thanks to Major Hayden :).

CHAPTER 2

Networkd

This chapter shows how to convert Network manger to Networkd and various use cases of Networkd

2.1 Comparing systemd-networkd and NetworkManager

- NetworkManager has been around for quite some time and systemd-networkd is relatively new.
- Re-configuring existing network interfaces is challenging for both NetworkManager and systemd-networkd
- Accessing raw systemd-networkd configuration files is more straightforward than NetworkManager
- systemd-networkd's memory and CPU usage is extremely low (good for containerized environments)
- systemd-networkd can handle various sysctl activities automatically, like IP forwarding
- Tunnels and Overlays - NetworkManager has more options for advanced tunnels, like vpnc, openconnect, and openvpn - systemd-networkd makes gre, vlan, vxlan, and other overlay technologies much easier to implement
- NetworkManager's logging is more verbose by default, which can be good for troubleshooting
- systemd-networkd is meant to be configured without a GUI

2.2 Configuration

Fedora Cloud Base Image. Download it from here.

2.3 Switching from NetworkManager to systemd-networkd

Start by ensuring that NetworkManager and network scripts don't start on reboot:

```
# systemctl disable NetworkManager
# systemctl disable network
```

Ensure that systemd-networkd starts on the next boot:

```
# systemctl enable systemd-networkd
```

Enable the resolver and make a symlink:

```
# systemctl enable systemd-resolved
# systemctl start systemd-resolved
# rm -f /etc/resolv.conf
# ln -s /run/systemd/resolve/resolv.conf /etc/resolv.conf
```

Be sure to configure your network interfaces in `/etc/systemd/network` and then reboot.

2.4 systemd-networkd use cases

Here are some sample use cases for systemd-networkd and example configurations.

Simple DHCP on a single interface

For an interface `eth0`, a single `.network` file is needed:

```
# cat /etc/systemd/network/eth0.network
[Match]
Name=eth0

[Network]
DHCP=yes
```

Static address on a single interface

For an interface `eth0`, a single `.network` file is needed:

```
# cat /etc/systemd/network/eth0.network
[Match]
Name=eth0

[Network]
Address=192.168.0.50/24
Address=2001:db8:dead:beef::/64

# These are optional but worth mentioning
DNS=8.8.8.8
DNS=8.8.4.4
NTP=pool.ntp.org
```

You can also split up the addresses into separate blocks:

```
# cat /etc/systemd/network/eth0.network
[Match]
Name=eth0

[Network]
DNS=8.8.8.8
DNS=8.8.4.4
NTP=pool.ntp.org

[Address]
Address=192.168.0.50/24

[Address]
Address=2001:db8:dead:beef::/64
```

Or add static routes:

```
# cat /etc/systemd/network/eth0.network
[Match]
Name=eth0

[Network]
DNS=8.8.8.8
DNS=8.8.4.4
NTP=pool.ntp.org

[Address]
Address=192.168.0.50/24

[Address]
Address=2001:db8:dead:beef::/64

[Route]
Destination=10.0.10.0/24
Gateway=192.168.50.1

[Route]
Destination=10.0.20.0/24
Gateway=192.168.50.1
```

Do DHCP on all network devices

You can use wildcards almost anywhere in the `[Match]` block. For example, this will cause systemd-networkd to do DHCP on all interfaces:

```
[Match]
Name=eth*

[Network]
DHCP=yes
```

Bridging

Let's consider an example where we have `eth0` and we want to add it to a bridge. This could be handy for servers where you want to build containers or virtual machines and attach them to the network bridge.

Start by setting up our bridge interface, `br0`:

```
# cat /etc/systemd/network/br0.netdev
[NetDev]
Name=br0
Kind=bridge
```

Now that we have a bridge device, let's configure the network for the bridge:

```
# cat /etc/systemd/network/br0.network
[Match]
Name=br0

[Network]
IPForward=yes
DHCP=yes
```

The `IPForward=yes` will take care of the sysctl forwarding setting for us (`net.ipv4.conf.br0.forwarding = 1`) automatically when the interface comes up.

Now, let's take the ethernet adapter and add it to the bridge:

```
# cat /etc/systemd/network/eth0.network
[Match]
Name=eth0

[Network]
Bridge=br0
```

Simply reboot the system and it will come up with `eth0` as a port on `br0`.

Bonding

Configuring a bonded interface is very similar to configuring a bridge. Start by setting up the individual network adapters:

```
# /etc/systemd/network/ens9f0.network
[Match]
Name=ens9f0

[Network]
Bond=bond1
```

```
# /etc/systemd/network/ens9f1.network
[Match]
Name=ens9f1

[Network]
Bond=bond1
```

Now we can create the network device for the bond:

```
# /etc/systemd/network/bond1.netdev
[NetDev]
Name=bond1
Kind=bond

[Bond]
Mode=802.3ad
TransmitHashPolicy=layer3+4
MIIMonitorSec=1s
LACPTransmitRate=fast
```

Once the device is defined, let's add some networking to it:

```
# /etc/systemd/network/bond1.network
[Match]
Name=bond1

[Network]
DHCP=yes
BindCarrier=ens9f0 ens9f1
```

The **BindCarrier** is optional but recommended. It gives systemd-networkd the hint that if both bonded interfaces are offline, it should remove the bond configuration until one of the interfaces comes up again.

2.5 Status & Diagnostics

All of the output from systemd-networkd will appear in your system journal. Any errors when setting up interfaces or configuring routes will be printed there. The `networkctl` command allows you to check your network devices at a

glance. Here's an example of a fairly complicated network setup:

```
# networkctl
IDX LINK             TYPE               OPERATIONAL SETUP
  1 lo               loopback           carrier     unmanaged
  2 enp3s0           ether              off         unmanaged
  3 enp1s0f0         ether              degraded    configured
  4 enp1s0f1         ether              degraded    configured
  5 br1              ether              routable    configured
  6 br0              ether              routable    configured
  7 gre0             ipgre              off         unmanaged
  8 gretap0          ether              off         unmanaged
  9 gre-colocation   ipgre              routable    configured
 12 vlan100          ether              routable    configured
 13 tun1             none               routable    unmanaged
 14 tun0             none               routable    unmanaged
 15 vlan200          ether              routable    configured
```

You'll find two physical network cards (`enp1s0f0` and `enp1s0f1`) each attached to a bridge (`br0` and `br1`, respectively). The physical network adapters show up as degraded because they don't have network addresses directly assigned – that assignment is done on the bridge. The `gre0` and `gretap0` devices are created automatically to handle the gre tunnel `gre-colocation`. There are also two VLANs configured within systemd and attached to a bridge. The `tun` interfaces are OpenVPN interfaces and they are not configured by systemd-networkd (hence the unmanaged setup).

Further Reading

- ArchLinux systemd-networkd documentation
- Upstream systemd-networkd documentation

CHAPTER 3

Atomic image examples

This chapter will contain various examples related to Atomic project.

3.1 Setting up Fedora Atomic cluster with Kubernetes and Vagrant

You can use Vagrant to setup a Fedora Atomic cluster. Use the following steps to set it up.

```
$ git clone https://github.com/kubernetes/contrib.git
$ cd contrib/ansible/vagrant
$ export OS_IMAGE=fedoraatomic # (regular fedora, regular centos, centos atomic are other options)
$ vagrant up --no-provision --provider=libvirt # (virtualbox, openstack and aws are other provider op
$ vagrant provision kube-master
```

This should get you a working Atomic cluster ready. For more details follow this blog post.

3.2 Setting up Fedora Atomic cluster with Kubernetes and preinstalled hosts

In case you don't want to use Vagrant, you can systems preinstalled with Fedora Atomic, and then update the inventory file (check the section above for git repo link), and use the same.

CHAPTER 4

Atomic Command Cheat Sheet

This chapter contains cheat sheet for atomic commands

The atomic command ‘‘/usr/bin/atomic‘‘ defines the entrypoint of Project Atomic hosts

- `atomic host upgrade` upgrades to a newer version.
- `atomic host rollback` rollbacks to the previous version.
- `atomic host status` shows the current status of the installed atomic host.
- `atomic run <name>` executes container image run method.
- `atomic install <name>` installs a container on atomic host with systemd unit file to run it as service.
- `atomic uninstall <name>` uninstalls the container from the atomic host.
- `atomic info <name>` returns LABEL Information of the image.
- `atomic images` displays all the container image present on the atomic host.
- `atomic scan <name>` scans the image or container
- `atomic stop <name>` executes container image stop method
- `atomic mount` mounts container image to a specific directory
- `atomic diff` shows difference between to container images, RPMs or file diff
- `atomic push` pushes (upload) the latest image to the repository
- `atomic pull` pulls the latest image from the repository
- `atomic top` shows stats about processes running inside container
- `atomic storage` manages container storage
- `atomic unmount` unmount container image
- `atomic update` updates to the latest container image of the repository
- `atomic version` displays “Name Version Release” Label of the image
- `atomic verify` verifies that the image is fully updated

We have blog post for few commands. Check this.

CHAPTER 5

Cloud Base Image examples

This chapter will contain various examples related to Cloud Base Image.

5.1 How to do docker-storage-setup in Fedora 22 cloud image?

docker-storage-setup helps to create a LVM thin pool, which can be then used by docker for storage of containers, and images. By starting docker, it automatically starts this service. We can also make sure that it uses a specific block device, and volume group. In this example I am running Fedora 22 Cloud Base image on an Openstack environment, I added a new volume */dev/vdb* to the instance.

```
# cat <<EOF > /etc/sysconfig/docker-storage-setup
DEVS=/dev/vdb
VG=docker-vg
EOF
# sudo docker-storage-setup
  Volume group "vda1" not found
  Cannot process volume group vda1
Checking that no-one is using this disk right now ... OK

Disk /dev/vdb: 5 GiB, 5379194880 bytes, 10506240 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xc8ed8872

Old situation:

>>> Script header accepted.
>>> Created a new DOS disklabel with disk identifier 0x39ca0d62.
Created a new partition 1 of type 'Linux LVM' and of size 5 GiB.
/dev/vdb2:
New situation:

Device     Boot Start      End  Sectors Size Id Type
/dev/vdb1        2048 10506239 10504192   5G 8e Linux LVM

The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
  Physical volume "/dev/vdb1" successfully created
```

```
  Volume group "docker-vg" successfully created
  Rounding up size to full physical extent 8.00 MiB
  Logical volume "docker-poolmeta" created.
  Logical volume "docker-pool" created.
  WARNING: Converting logical volume docker-vg/docker-pool and docker-vg/docker-poolmeta to pool's da
  THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)
  Converted docker-vg/docker-pool to thin pool.
  Logical volume "docker-pool" changed
```

5.2 How to build a network router and firewall with Fedora 22 and systemd-networkd?

Major Hayden explains that in this blog post.

CHAPTER 6

Docker related queries

SELinux rule for volumes on host

```
# chcon -Rt svirt_sandbox_file_t /path/to/volume
```

Without this you will see permission error on a Fedora host.

CHAPTER 7

Vagrant

Vagrant is a useful development tool that lowers the barrier to entry to new contributors of your project. With Vagrant, new contributors don't have to spend much time configuring their development environment, but can quickly get one automatically provisioned for them with a few simple commands.

7.1 Using Vagrant in Fedora

Fedora's Cloud working group provides Fedora Vagrant boxes for libvirt and Virtualbox. You can see the Fedora 24 Vagrant images here.

To quickly get started with Vagrant on a Fedora host, the Fedora developers have conveniently packaged vagrant-libvirt and a handful of handy Vagrant plugins for you. All you need to do is to install vagrant-libvirt and the plugins you wish to use, write a Vagrantfile for your project, and type "vagrant up" to get started. Here's an example Vagrantfile:

```
$ cat Vagrantfile
# -*- mode: ruby -*-
# vi: set ft=ruby :

# On your host:
# git clone https://github.com/fedora-infra/bodhi.git
# cd bodhi
# cp Vagrantfile.example Vagrantfile
# vagrant up
# vagrant ssh -c "cd /vagrant/; pserve development.ini --reload"

Vagrant.configure(2) do |config|
  config.vm.box_url = "https://download.fedoraproject.org/pub/fedora/linux/releases/24/CloudImages/x8
  config.vm.box = "f24-cloud-libvirt"
  config.vm.network "forwarded_port", guest: 80, host: 80
  config.vm.synced_folder ".", "/vagrant", type: "sshfs"

  config.vm.provider :libvirt do |domain|
      domain.cpus = 4
      domain.graphics_type = "spice"
      domain.memory = 1024
      domain.video_type = "qxl"
  end

  config.vm.provision "shell", inline: "echo hello_world > /home/vagrant/hello_world.txt"

  # Uncomment the following block if you have a playbook at devel/ansible/playbook.yml you want Vagra
  # # Ansible needs the guest to have these
```

```
  # config.vm.provision "shell", inline: "sudo dnf install -y libselinux-python python2-dnf"
  #
  # config.vm.provision "ansible" do |ansible|
  #     ansible.playbook = "devel/ansible/playbook.yml"
  # end
end
```

In this example, we're using the Fedora 24 libvirt box and we're assuming that your project has an ansible playbook at the relative path `devel/ansible/playbook.yml`. Writing ansible playbooks is beyond the scope of this document, and you are free to use config.vm.provision lines to configure your Vagrant guest if you like.

To get started with the above example, simply write the code to a file called Vagrantfile, install `vagrant-libvirt` and `vagrant-sshfs`, and run `vagrant up`:

```
$ cd ~/devel/my_project
$ vim Vagrantfile  # Write the above Vagrant code here
$ sudo dnf install vagrant-libvirt vagrant-sshfs
$ vagrant up
```

Once your guest is running, you can ssh into the guest. Your code directory from the host will be shared into the guest at the path `/vagrant`:

```
$ vagrant ssh
$ [vagrant@localhost ~]$ ls /vagrant/Vagrantfile
/vagrant/Vagrantfile
```

Now you can edit your project on your host, and the changes you make will be shared into the guest's `/vagrant` folder live. This allows you to use your editor of choice (even graphical editors!) on the host, while keeping everything that might "dirty" you host system contained in the guest virtual machine. This example `Vagrantfile` has also set up a port forward from 80 in the guest to 80 on the host, so if there were a web application listening in the guest on port 80, you could browse to http://localhost on the host to access it.

When you are done with your Vagrant guest, you can destroy it:

```
$ vagrant destroy
```

It is good practice to check in a `Vagrantfile.example` file in your project's source code, rather than the `Vagrantfile` itself. This allows new developers a quick way to get started by just copying your example into place, but it also allows each contributor to make the changes they prefer to their individual Vagrantfiles.

CHAPTER 8

Irssi: IRC Client on Atomic Host

Irssi is a Terminal Based IRC Client for Unix/Linux based systems. This page shows how to run Irssi on Atomic Host.

8.1 Steps

First you will need to boot up an atomic host

- Copy the **Sources** down from here Fedora Dockerfile for Irssi.
- **Perform the build with the following command:** `docker build -t fedora/irssi .`
- **After the build is successful run the container with the following command:** `atomic run fedora/irssi`

This will start Irssi.

8.2 Irssi Commands

To Connect to Server:

```
/connect irc.freenode.net
```

To Connect to Channel:

```
/join #fedora
```

To leave Channel:

```
/part #fedora
```

To Set nick:

```
/set nick username
```

To Identify nick:

```
/msg nickserv identify password
```

To Change nick:

```
/nick username1
```

8.3 More Commands

Command	Description
/ban	Sets or lists bans for a channel
/clear	Clears a channel buffer
/disconnect	Disconnects from the network that has focus
/exit	Disconnects client from all networks and returns to shell prompt
/join	Joins a channel
/kick	Kicks a user out
/kickban	Kickbans a user
/msg	Sends a private message to a user
/names	Lists the users in the current channel
/query	Opens a query window with a user or closes a current query window
/topic	Displays/edits the current topic
/unban	Unbans everyone
/whois	Displays user information
/window close	Forces closure of a window

Further Reading: To know more about Irssi visit https://irssi.org.

CHAPTER 9

How to contribute to this guide?

This guide is maintained by volunteers on github. It is written using reStructred Text, and sphinx project. If you want to add full doc in any part of this document, feel free to submit a pull request. You can also add links to blog posts which explains how to do any particular task on Cloud.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search