Stacki Tutorial #4: Configuring Routing (Networking Part 2)

Introduction

In a previous tutorial, we went over the basics of configuring networking. That tutorial mentioned that there was a lot more in Stacki to deal with the networking component of servers. This tutorial is aimed at covering some of those additional bits. Specifically, this tutorial covers routing.

If you have not gone through the first networking tutorial, this tutorial will still be of benefit. Having a working backend server that Stacki installed is the key to getting the most out of this tutorial, not having studied the previous one.

```
Naming hosts/host groups

Most stacki commands take the name of a host. This can be in any of the following forms:

hostname - Actual host DNS name host-0-0.local – This machine.
hostname* - All hosts whose name match the pattern. Host* - All machines whose names start with
host.
appliance name - all machines of that appliance type backend – All machines of appliance type
backend.
```

Pre-requisites

This tutorial assumes:

1. You have downloaded Stacki from www.stacki.com
2. You have followed the directions here to install a Stacki server.
3. You have followed the directions here to install backend server(s).
4. You have access to a command line on the Stacki server.
5. You have information about the network(s) you want to add to Stacki.

These are the same requirements as the fir st networking tutorial. That is not a coincidence, since both tutorials cover networking. To make this tutorial stand-alone, we will be reviewing adding networks to Stacki, but otherwise there is no direct overlap in content between the two tutorials. If you completed the first networking tutorial on the same system you will use for this one, skip ahead to “Modifying Network Connections”.

Let's Get Started!

Log in to the Stacki server, and type the following command:

```
stack list network
```

You should see output like this, assuming you've not changed anything yet:

```
[root@stackidon ~]# stack list network
NETWORK      SUBNET         NETMASK        MTU     DNSZONE SERVEDNS
private:     10.1.1.0       255.255.255.0          1500    local           True
public:      192.168.0.0    255.255.255.0          1500    stackiq.com     False

[root@stackidon ~]#
```

These are all of the networks Stacki knows about by default. The private network is used for installations, the public so that you can log into the stacki server remotely, and get updates in the future. This tutorial will show you how to create new networks, and to modify the "public" network of these. Since "private" is used to do installations, we'll not do anything with that one until a later tutorial, when we have more information about what can be done with networks. By default, all servers exist on the private network. To see this, type the following…

```
Stack list host interface backend*
```

You should see output like the following:

```
[root@stackidon ~]# stack list host interface backend*
HOST         SUBNET  IFACE   MAC     IP      NETMASK        MODULE NAME            VLAN OPTIONS
CHANNEL
backend-0-0: private eth0    08:00:27:6e:88:08 10.1.1.253 255.255.255.0 ------
backend-0-0 ----                        -------       -------
backend-0-1: private eth0    08:00:27:3a:e0:e4 10.1.1.254 255.255.255.0 ------
backend-0-1 ----                         ------- -------

[root@stackidon ~]#
```

Notice that the interface has a lot of information on it. We'll get to that in a future installment.

To add a network to the list of networks, simply type:

```
stack add network protected 10.10.10.1 255.255.255.0
```

As you likely guessed already, 'protected' will be the network name, '10.10.10.1 the base address, and 255.255.255.0 the netmask – meaning the network covers all of the 10.10.10.* network.

```
[root@stackidon ~]# stack add network protected 10.10.10.1 255.255.255.0

[root@stackidon ~]# stack list network
NETWORK    SUBNET      NETMASK       MTU   DNSZONE     SERVEDNS
private:   10.1.1.0    255.255.255.0 1500  local       True
protected: 10.10.10.1  255.255.255.0 1500  protected   False
public:    192.168.0.0 255.255.255.0 1500  stackiq.com False

[root@stackidon ~]#
```

Now stacki knows about the protected network, and we can tell backend machines to attach NICs to it. Let's do so. First, add the interface to the stacki database:

```
stack add host interface backend-0-0 iface=eth1 ip=10.10.10.100 subnet=protected name=prot1
```

This tells stacki to add eth1 to server backend-0-0, give it ip 10.10.10.100, attach it to subnet protected, and name it prot1 locally.

Now if you type:

```
stack list host interface backend-0-0
```

You'll see our new interface in the list, but if you go to a terminal on backend-0-0, you will not find it listed with ifconfig. That is because Stacki knows about it, but hasn't updated the server yet. To update the server type:

```
Stack sync host network backend-0-0
```

It will tell the host about its new interface, and bring the interface up. If you go to the backend servers' terminal and type ifconfig, you will see eth1 is now there, and it is fully functional (assuming you put it on the right network with a valid IP).

```
[root@stackidon ~]# stack add host interface backend-0-0 iface=eth1 ip=10.10.10.100 subnet=pro-
tected name=prot1
[root@stackidon ~]# stack list host interface backend-0-0
SUBNET    IFACE MAC               IP           NETMASK       MODULE NAME        VLAN OPTIONS
CHANNEL
private   eth0  08:00:27:6e:88:08 10.1.1.253   255.255.255.0 ------ backend-0-0 ---- ------- ---
----
protected eth1  ----------------- 10.10.10.100 255.255.255.0 ------ prot1       ---- ------- ---
----
[root@stackidon ~]# stack sync host network backend-0-0
[root@stackidon ~]#
```

Modifying Network Connections

The first tutorial did a good job of setting up network connections and showing how they work, this tutorial will focus on changing them so that the configuration can be adaptive to infrastructure changes.

Above, we added a network protected. This is great for initial base config – quick and easy. But what if the networking changes, or you have more specific information – like VLANs – to track. That's where set host interface comes in. We'll take the subcommands one at a time, but in short, set host interface allows complete control of the networking interface from the stacki server command line.

> Note:
> A note, if your organization is using spreadsheets to manage server configurations, information entered via the command line will be lost if ever someone issues the command stack remove host [hostname]. While it is rare, there are cases where you might want to delete the host and re-add it later, so this warning is worth note. The best solution to this problem would be to script any customized command lines you use, and place the scripts into version control for re-use when servers are returned under stacki management.

First, let's talk about routing. The initial configuration of the private network has routing configured to a usable default. But if you add a network like protected, you're going to want to provide a route to the gateway, and perhaps more.

There are a variety of ways you can add routing information to stacki, depending upon your needs. Let's say that you want all servers stacki is installing to share a route without concern for any particulars of the server. You would use:

```
stack add route [address] [gateway] netmask=[value]
```

The netmask is optional, and is assumed to define a single host if left blank.

So to add a route to the 10.10.100.0/24 network for our protected network, we would use:

```
stack add route 10.10.100.0 protected netmask=255.255.255.0
```

As with all networking changes, this new route will be applied to servers when one of the following happens:

- they are installed
- they are re-installed
- stack sync host network is called

Note – in all of these examples we are using the network name as the gateway. There are other options, including the IP address. See the stack add route listing on the wiki if you prefer a different method of specifying the gateway.

Sometimes you just don't want the route applied to every server. This is usually the case when there are a variety of appliance types in stacki and some of them need specialized access to a network. As such, there is an appliance route command that functions much the same, but only on machines installed as certain appliance types. We only have backend defined, but we'll use that to show you how it works.

The command is:

```
stack add appliance route [appliance] [address] [gateway] netmask=[value]
```

Notice that this is the same as the global add route, with just the name of the appliance type it should apply to added in.

```
stack add appliance route backend 10.10.100.0 protected netmask=255.255.255.0
```

Would add the exact same route, but only to machines installed as "backend" appliances – with the backend pallet, essentially.

The same is true for specific hosts. The command to add a route to just one host is:

```
stack add host route [host] [address] [gateway] netmask=[value]
```

Once again, the netmask is optional and defaults to a host – a /1 netmask.

To add the same route we have been using to host backend-0-0, the command would be:

```
stack add host route backend-0-0 10.10.100.0 protected netmask=255.255.255.0
```

Remember that these commands will update the stacki database, but changes will not be pushed out to the host until one of the three th ings discussed above happens.

Since this is a tutorial, let's look at the database now with:

```
stack list host route backend-0-0
```

our new route will be there, ready to go. We can then go to the console for backend-0-0 and ping something on the new network. We can also list the routing table to show our new route.

```
ssh backend-0-0
ping 10.10.100.50
route
```

> Reminder:
> These changes are in the Stacki database, but should you ever delete the host from the database and then go to re-add it, they will have to be re-executed. If you are in a dynamic environment, it is a good idea to build the commands into a script that can be executed in such a scenario, and check that script into version control along with the server definition spreadsheet.

The following is a sample of using these commands that routes out of the public network through 192.168.0.1. Items of interest are highlighted in red for easy access.
Once again, the netmask is optional and defaults to a host – a /1 netmask.

```
[root@stackidon ~]# stack list host route backend-0-0
HOST         NETWORK         NETMASK         GATEWAY  SOURCE
backend-0-0: 0.0.0.0         0.0.0.0         10.1.1.1 G
backend-0-0: 10.10.100.0     255.255.255.0   -------- G
backend-0-0: 192.168.0.215   255.255.255.255 10.1.1.1 G
backend-0-0: 224.0.0.0       255.255.255.0   eth0     G
backend-0-0: 255.255.255.255 255.255.255.255 eth0     G
[root@stackidon ~]# stack add host route backend-0-0 0.0.0.0 192.168.0.1 netmask=0.0.0.0
[root@stackidon ~]# stack list host route backend-0-0
HOST         NETWORK         NETMASK         GATEWAY     SOURCE
backend-0-0: 0.0.0.0         0.0.0.0         192.168.0.1 H
backend-0-0: 10.10.100.0     255.255.255.0   ----------- G
backend-0-0: 192.168.0.215   255.255.255.255 10.1.1.1    G
backend-0-0: 224.0.0.0       255.255.255.0   eth0        G
backend-0-0: 255.255.255.255 255.255.255.255 eth0        G
[root@stackidon ~]# stack sync host network backend-0-0
[root@stackidon ~]# ssh backend-0-0
Last login: Fri Aug 21 13:41:41 2015 from stackidon.local
[root@backend-0-0 ~]# ping www.google.com
PING www.google.com (216.58.217.132) 56(84) bytes of data.
64 bytes from iad23s43-in-f132.1e100.net (216.58.217.132): icmp_seq=1 ttl=50 time=146 ms
64 bytes from iad23s43-in-f132.1e100.net (216.58.217.132): icmp_seq=2 ttl=50 time=45.0 ms
64 bytes from iad23s43-in-f132.1e100.net (216.58.217.132): icmp_seq=3 ttl=50 time=56.8 ms
64 bytes from iad23s43-in-f132.1e100.net (216.58.217.132): icmp_seq=4 ttl=50 time=68.5 ms
^C
--- www.google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 4953ms
rtt min/avg/max/mdev = 45.046/79.316/146.819/39.852 ms
[root@backend-0-0 ~]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
255.255.255.255 *               255.255.255.255 UH    0      0        0 eth0
stackidon.stack stackidon.local 255.255.255.255 UGH   0      0        0 eth0
192.168.0.0     *               255.255.255.0   U     0      0        0 eth1
10.1.1.0        *               255.255.255.0   U     0      0        0 eth0
224.0.0.0       *               255.255.255.0   U     0      0        0 eth0
link-local      *               255.255.0.0     U     1002   0        0 eth0
link-local      *               255.255.0.0     U     1003   0        0 eth1
default         192.168.0.1     0.0.0.0         UG    0      0        0 eth1
[root@backend-0-0 ~]#
```

That's it for this tutorial! You now can route your way to whatever access is needed!