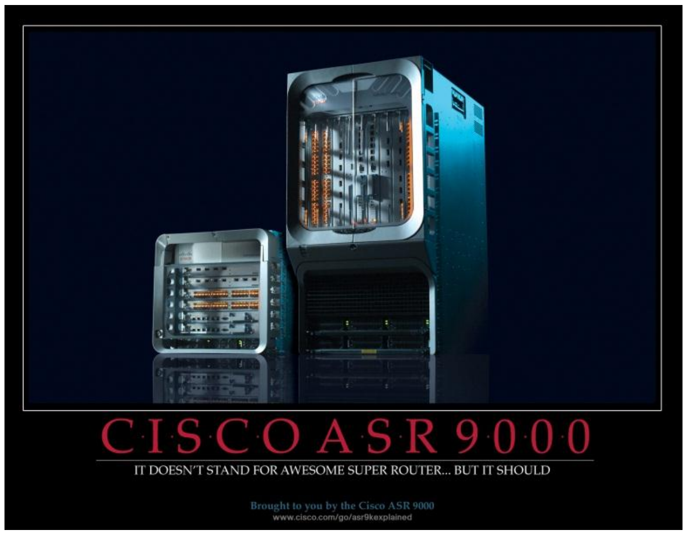


CISCO IOS XR

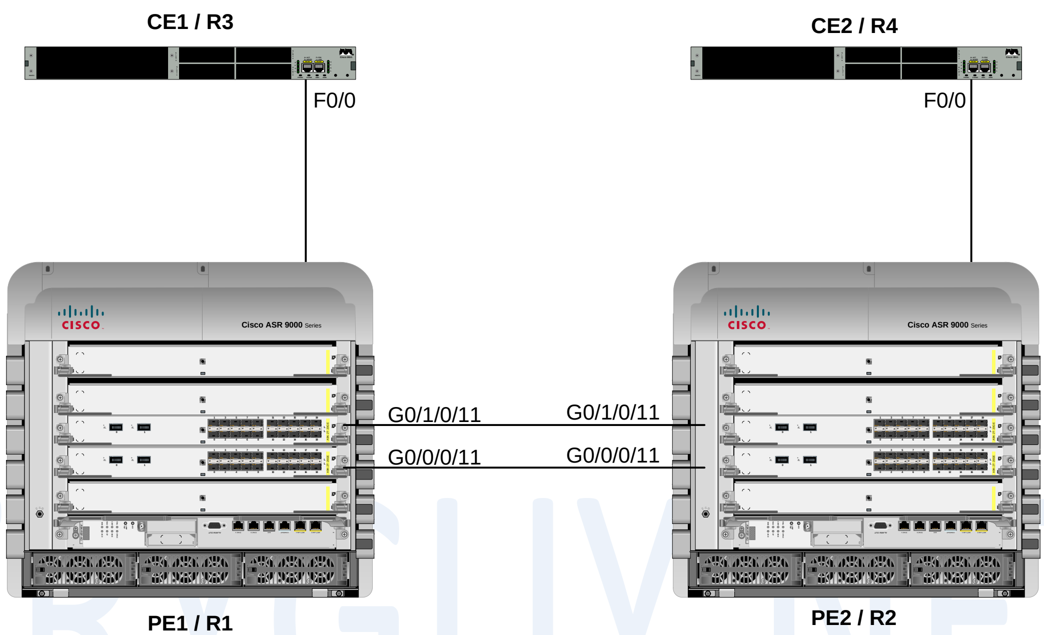
Jeffrey Fry

FryGuy.Net

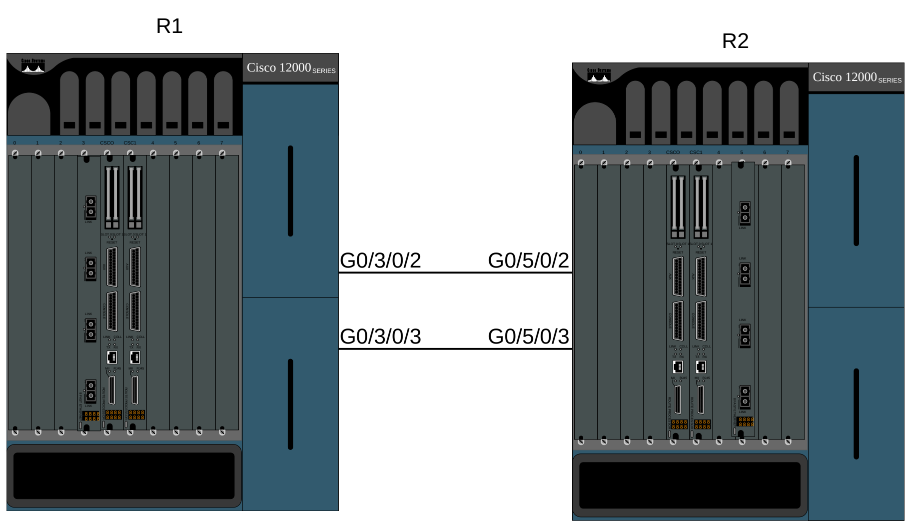
Table of Contents

There are two topologies that have been used. This is because of what I had access to changed over time. For this lab, here is a topology diagram of what was used:

ASR9000 and Cisco 2811



Cisco 12000



1. Cisco IOS XR Introduction and Comparison to IOS

Let's start with the basic difference between Cisco IOS and Cisco IOS XR code, the Operating System.

In Cisco IOS, the kernel is monolithic, meaning everything in installed in a single image and all processes share the same address space. There is no memory protection between processes, so if one crashes it can impact all other processes on the box – thus forcing or causing a reload of the entire router. The other thing with monolithic code is that it has a *run to completion scheduler,* so the kernel will not preempt a process that is running; the process must make a kernel call before another process has a chance to execute.

In Cisco IOS XR, the kernel is based on an OS called QNX Neutrino that runs some very powerful and reliable systems. QNX runs – per their News Release at http://www.qnx.com/news/pr_1329_3.html - things from EKG machines, to Air Traffic Control systems, and among other things – automated beer bottle inspection systems. IOS XR offers modularity and memory protection between processes, threads and supports preemptive scheduling as well as the ability to restart a failed process. Protocols like BGP, OSPF, OSPFv3, RIBv4, RIBv6, etc all run in separate spaces – if one has a fault, it will not impact the others. Also, an added bonus, if you run multiple routing protocol instances (like OSPF), each process will run in its own memory space – this is an important feature of Service Providers – any fault with one customer process will not impact another.

Another big difference between IOS and IOS XR is the configuration model. IOS is a single stage model meaning that as soon as you make a change, it is applied to the active running config. With IOS XR, you have a running (active) config that you cannot modify directly, all your changes are made in a staging area first before being committed to the running config. After you make your changes, you commit them and promote the staging config to the active config. Before the change is made active, the IOS XR will run a sanity check on it making sure that the commands are correct to a certain degree, if there is a problem it will tell you so that you can correct the error

```
% Failed to commit one or more configuration items. Please use 'show configuration failed' to view the errors
```

Here is a table of some of the other significant differences between IOS and IOS XR

IOS XR	IOS
Config changes do not take place immediately	Configuration changes take place immediately
Config changes must be COMMITted before taking effect	No commit, changes immediate
You can check your configuration before applying it	No verification, immediate.
Two stage configuration	Single stage
Configuration Rollback	Not easy to do, has to be manually configured and not guaranteed
Feature centric	Interface centric

2. Cisco IOS XR Prompt and Hostname Differences

Let's cover the prompt real quick as that is a bit different than what people are used to.

Let's look at the standard IOS prompt vs. the IOS XR prompt.

IOS: **Router#**

IOS-XR: **RP/0/7/CPU0:ios#**

As you can see the prompt is a bit different. In standard IOS you have the hostname, but in IOS XR you get a bit more information. It breaks down as follows:

Prompt Syntax:

- Type – type of interface card (Usually **RP** for Route Processor)
- Rack – What Rack number this is installed in in a multishelf system, typically **0** if standalone
- Slot – Slot the RP is installed in (**7** in this example)
- Module – What execute the user commands or port interface. Usually **CPU0** or CPU1
- Name – Hostname of the router, default here is **IOS**

Ok, now let's change the hostname on typical IOS so you can see the difference. Going forward, **BLUE** text is prompts and router feedback, **RED** are commands entered.

```
Router#
Router#conf t
*Mar 29 16:32:51.507: %SYS-5-CONFIG_I: Configured from console by console
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#hostname R1
R1(config)#
```

As you can see, in IOS the hostname changed immediately after hitting Enter.

So, let's change the hostname to R1 on IOS XR code:

```
RP/0/7/CPU0:ios#
RP/0/7/CPU0:ios#conf t
Thu Mar 29 16:00:43.844 UTC
RP/0/7/CPU0:ios(config)#hostname R1
RP/0/7/CPU0:ios(config)#
```

Notice that the hostname did not change? In IOS XR you need to COMMIT your changes in order for them to take effect. But before we commit them, let's do a show config quick

```
RP/0/7/CPU0:ios(config)#
RP/0/7/CPU0:ios(config)# sh config
Thu Mar 29 16:03:53.060 UTC
Building configuration...
!! IOS XR Configuration 4.1.1
hostname R1
end

RP/0/7/CPU0:ios(config)#
```

Pretty cool, the router will show you the changes you are about to make, this is your staging config changes.

Now we can COMMIT the changes

```
RP/0/7/CPU0:ios(config)#commit
Thu Mar 29 16:03:04.182 UTC
RP/0/7/CPU0:R1(config)#
```

See, once you entered COMMIT, the hostname change from IOS to R1.

3. Basic Configuration Options

Ok, we have seen the basic COMMIT option – but what other options do we have for configuration mode? Well, we have a few to choose from.

First, what if I am making changes and decide I don't want them? You have a few options. First you could just exit all the way out.

```
RP/0/7/CPU0:R1(config)#exit
Uncommitted changes found, commit them before exiting(yes/no/cancel)?
[cancel]: no
```

And once you exit out, all your changes are lost.

Ok, that is one option. Another is clear. To demonstrate we will create loopback 666:

```
RP/0/7/CPU0:R1#conf t
Sun Apr  1 22:18:52.956 UTC
RP/0/7/CPU0:R1(config)#int loop666
RP/0/7/CPU0:R1(config-if)#ip add 6.6.6.6/32
```

Ok, let's check the candidate configuration:

```
RP/0/7/CPU0:R1(config-if)#show config
Sun Apr  1 22:19:03.438 UTC
Building configuration...
!! IOS XR Configuration 4.1.1
interface Loopback666
 ipv4 address 6.6.6.6 255.255.255.255
!
end

RP/0/7/CPU0:R1(config-if)#
```

OK, we have it in the candidate configuration now. We changed our mind about that – so lets clear it.

```
RP/0/7/CPU0:R1(config-if)#clear
```

Now check the candidate configuration again.

```
RP/0/7/CPU0:R1(config)#show config
Sun Apr  1 22:19:34.733 UTC
Building configuration...
!! IOS XR Configuration 4.1.1
end

RP/0/7/CPU0:R1(config)#
```

There, all gone!

Now, what if we want to make a change but we want to be sure we don't lose connection to the router? Well, we can do a commit confirm, this way if we do lose connection our change will be rolled back!

```
RP/0/7/CPU0:R1#conf t
Sun Apr  1 22:23:01.154 UTC
RP/0/7/CPU0:R1(config)#int loop 666
RP/0/7/CPU0:R1(config-if)#ip add 6.6.6.6/32
```

Now, lets look at our commit confirmed options:

```
RP/0/7/CPU0:R1(config-if)#commit confirmed ?
  <30-65535>  Seconds until rollback unless there is a confirming commit
  minutes     Specify the rollback timer in the minutes
  <cr>        Commit the configuration changes to running
```

See, we can have a few seconds or a few minutes. Pretty cool!

```
RP/0/7/CPU0:R1(config-if)#commit confirmed 30
Sun Apr  1 22:23:19.344 UTC
```

Now, lets see if we have loop666:

```
RP/0/7/CPU0:R1(config-if)#do show int loop666
Sun Apr  1 22:23:34.353 UTC
Loopback666 is up, line protocol is up
  Interface state transitions: 1
  Hardware is Loopback interface(s)
  Internet address is 6.6.6.6/32
  MTU 1500 bytes, BW 0 Kbit
     reliability Unknown, txload Unknown, rxload Unknown
  Encapsulation Loopback,  loopback not set,
  Last input Unknown, output Unknown
  Last clearing of "show interface" counters Unknown


RP/0/7/CPU0:R1(config-if)#
```

Yup, its there. Now we can wait a few seconds (30 or so) and do the show interface command again.

```
RP/0/7/CPU0:R1(config-if)#do show int loop666
Sun Apr  1 22:25:09.361 UTC
Interface not found (Loopback666)

RP/0/7/CPU0:R1(config-if)#
```

All gone!

Ok, now lets commit it this time.

```
RP/0/7/CPU0:R1#conf t
Sun Apr  1 22:26:20.749 UTC
RP/0/7/CPU0:R1(config)#int loop666
RP/0/7/CPU0:R1(config-if)#ip add 6.6.6.6/32
RP/0/7/CPU0:R1(config-if)#commit confirmed 30
Sun Apr  1 22:26:32.913 UTC
RP/0/7/CPU0:R1(config-if)#
```

Lets see if the interface is there:

```
RP/0/7/CPU0:R1(config-if)#do show int loop666
Sun Apr  1 22:26:38.421 UTC
Loopback666 is up, line protocol is up
  Interface state transitions: 1
  Hardware is Loopback interface(s)
  Internet address is 6.6.6.6/32
  MTU 1500 bytes, BW 0 Kbit
     reliability Unknown, txload Unknown, rxload Unknown
  Encapsulation Loopback,  loopback not set,
  Last input Unknown, output Unknown
  Last clearing of "show interface" counters Unknown
```

Yup, now we can commit it again to make it stay.

```
RP/0/7/CPU0:R1(config-if)#commit
Sun Apr  1 22:26:40.299 UTC

% Confirming commit for trial session.
RP/0/7/CPU0:R1(config-if)#
```

And lets make sure it is still there.

```
RP/0/7/CPU0:R1#sh int loop 666
Sun Apr  1 22:27:09.232 UTC
Loopback666 is up, line protocol is up
  Interface state transitions: 1
  Hardware is Loopback interface(s)
  Internet address is 6.6.6.6/32
  MTU 1500 bytes, BW 0 Kbit
     reliability Unknown, txload Unknown, rxload Unknown
  Encapsulation Loopback,  loopback not set,
  Last input Unknown, output Unknown
  Last clearing of "show interface" counters Unknown


RP/0/7/CPU0:R1#
```

Look at that, IOS XR has a commit confirmed – ***ju***st like someo***n***e else d***oes*** as well.

Few other things that is nice to know.

You can configure the system in exclusive mode, this way only you can be making changes and nobody else. To do this, just enter configure exclusive

```
RP/0/7/CPU0:R1#configure exclusive
```

You can add comments and notations to your commit that will show up in the rollback.

```
RP/0/7/CPU0:R1#conf t
Sun Apr  1 22:32:23.941 UTC
RP/0/7/CPU0:R1(config)#int loop 667
RP/0/7/CPU0:R1(config-if)#ip add 6.6.6.7/32
RP/0/7/CPU0:R1(config-if)#exit
RP/0/7/CPU0:R1(config)#commit comment Created Loopback 667 For  Testing
Sun Apr  1 22:33:34.589 UTC
RP/0/7/CPU0:R1(config)#
```

Now, if a comment has been added, you can see it via the show configuration history last x detail command

```
RP/0/7/CPU0:R1#sh configuration history last 1 detail
Sun Apr  1 22:36:04.053 UTC
 1)  Event: commit        Time: Sun Apr  1 22:33:36 2012
     Commit ID: 1000000230 Label:
     User: user            Line: con0_7_CPU0
     Client: CLI          Comment:  Created Loopback 667 For Testing

RP/0/7/CPU0:R1#
```

Ok, let's quickly look at loading a configuration from the disk and overwriting an existing configuration.

I have copied a config to disk0a: called newconfig.txt. What I want to do is install this configuration as the running config on the router.

```
1626        -rwx  204        Wed Oct 17 01:21:30 2012  newconfig.txt
```

So to start, lets delete the existing configuration

```
RP/0/RSP0/CPU0:R1(config)#commit replace
Wed Oct 17 01:21:43.406 UTC

This commit will replace or remove the entire running configuration. This
operation can be service affecting.
Do you wish to proceed? [no]: y
RP/0/RSP0/CPU0:ios(config)#
RP/0/RSP0/CPU0:ios(config)#exit
```

Ok, so now we are at an unconfigured device. Now we can load the config on the disk to the running config.

```
RP/0/RSP0/CPU0:ios(config)#load disk0a:/newconfig.txt
Loading.
204 bytes parsed in 1 sec (203)bytes/sec
```

The configuration is now loaded into the candidate config. Let us check what is there and then commit it.

```
RP/0/RSP0/CPU0:ios(config)#show confi
Wed Oct 17 01:26:17.539 UTC
Building configuration...
!! IOS XR Configuration 4.1.2
hostname R1
domain name lab.cfg
interface Loopback100
 ipv4 address 100.100.100.100 255.255.255.255
!
end

RP/0/RSP0/CPU0:ios(config)#commit
Wed Oct 17 01:26:22.174 UTC
RP/0/RSP0/CPU0:R1(config)#
```

There, we have loaded the config and applied the changes.

I have loaded another file to the router called ReplaceConfig.txt. This is a new configuration for the router, one that we want to replace the existing config with.

```
RP/0/RSP0/CPU0:R1#conf t
Wed Oct 17 01:37:23.638 UTC
RP/0/RSP0/CPU0:R1(config)#load disk0a:/ReplaceConfig.txt
Loading.
283 bytes parsed in 1 sec (282)bytes/sec
RP/0/RSP0/CPU0:R1(config)#show config
Wed Oct 17 01:37:38.571 UTC
Building configuration...
!! IOS XR Configuration 4.1.2
hostname Router1
domain name NewLab.CFG
interface Loopback100
 ipv4 address 101.101.101.101 255.255.255.255
!
interface TenGigE0/0/0/0
 ipv4 address 200.200.200.202 255.255.255.0
!
end
```

```
RP/0/RSP0/CPU0:R1(config)#commit replace
Wed Oct 17 01:37:41.577 UTC

This commit will replace or remove the entire running configuration. This
operation can be service affecting.
Do you wish to proceed? [no]: y
RP/0/RSP0/CPU0:Router1(config)#
```

What other options to loaf configuration are there? Well, here is a list:

```
RP/0/RSP0/CPU0:Router1(config)#load ?
  WORD             Load from file
  bootflash:       Load from bootflash: file system
  commit           Load commit changes
  compactflash:    Load from compactflash: file system
  compactflasha:   Load from compactflasha: file system
  configuration    Contents of configuration
  diff             Load from diff file
  disk0:           Load from disk0: file system
  disk0a:          Load from disk0a: file system
  disk1:           Load from disk1: file system
  disk1a:          Load from disk1a: file system
  ftp:             Load from ftp: file system
  harddisk:        Load from harddisk: file system
  harddiska:       Load from harddiska: file system
  harddiskb:       Load from harddiskb: file system
  lcdisk0:         Load from lcdisk0: file system
  lcdisk0a:        Load from lcdisk0a: file system
  nvram:           Load from nvram: file system
  rcp:             Load from rcp: file system
  rollback         Load rollback changes
  tftp:            Load from tftp: file system
RP/0/RSP0/CPU0:Router1(config)#
```

You can load from the local disk, RCP, TFTP, FTP, etc if you want.

4. Configuring an interface Basic IPv4 and IPv6 address

First we will take a look at what interfaces we have and review them quickly. We can use the same IOS command we are already familiar with – show ip interface brief

```
RP/0/7/CPU0:R1#
RP/0/7/CPU0:R1#sh ip int br
Thu Mar 29 18:12:04.883 UTC

Interface                 IP-Address      Status                Protocol
MgmtEth0/7/CPU0/0         unassigned      Shutdown              Down
MgmtEth0/7/CPU0/1         unassigned      Shutdown              Down
MgmtEth0/7/CPU0/2         unassigned      Shutdown              Down
GigabitEthernet0/3/0/0    unassigned      Down                  Down
GigabitEthernet0/3/0/1    unassigned      Down                  Down
GigabitEthernet0/3/0/2    unassigned      Up                    Up
GigabitEthernet0/3/0/3    unassigned      Up                    Up
MgmtEth0/6/CPU0/0         unassigned      Shutdown              Down
MgmtEth0/6/CPU0/1         unassigned      Shutdown              Down
MgmtEth0/6/CPU0/2         unassigned      Shutdown              Down
RP/0/7/CPU0:R1#
```

Here you can see that we have an RP in Slot 6 and 7 (Mgmt) and a 4-port Gig card in Slot 3. For this lab, interfaces G0/3/0/2 and G0/3/0/3 are pre-cabled to another router and are currently UP/UP right now.

Let configure an IP address on G0/3/0/2 of 150.1.12.1 with a mask of 255.255.255.0

First, let’s look at the running config on the interface now:

```
RP/0/7/CPU0:R1#
RP/0/7/CPU0:R1#sh run int g0/3/0/2
Thu Mar 29 18:38:29.942 UTC
% No such configuration item(s)

RP/0/7/CPU0:R1#
```

As you can see, it says No such config, it is telling you that it is unconfigured.

```
RP/0/7/CPU0:R1#conf t
Thu Mar 29 18:38:31.891 UTC
RP/0/7/CPU0:R1(config)#int g0/3/0/2
RP/0/7/CPU0:R1(config-if)#ip add 150.1.12.1/24
```

Notice, on IOS XR you can use / for the subnet, no more entering 255.255.255.0 :

```
RP/0/7/CPU0:R1(config-if)#show config
Thu Mar 29 18:38:44.248 UTC
Building configuration...
!! IOS XR Configuration 4.1.1
interface GigabitEthernet0/3/0/2
 ipv4 address 150.1.12.1 255.255.255.0
!
end
RP/0/7/CPU0:R1(config-if)#
```

Another cool thing with IOS-XR is you can find out where you are any time you want just by entering PWD

```
RP/0/7/CPU0:R1(config-if)#pwd

Thu Mar 29 19:31:24.666 UTC
interface GigabitEthernet0/3/0/2
RP/0/7/CPU0:R1(config-if)#
RP/0/7/CPU0:R1(config-if)#comm
Thu Mar 29 18:38:46.216 UTC
RP/0/7/CPU0:R1(config-if)#
```

Now, let's check the running config on that interface again:

```
RP/0/7/CPU0:R1#sh run int g0/3/0/2
Thu Mar 29 18:42:43.763 UTC
interface GigabitEthernet0/3/0/2
 ipv4 address 150.1.12.1 255.255.255.0
!

RP/0/7/CPU0:R1#
```

Let's PING our neighbor now – 150.1.12.2

```
RP/0/7/CPU0:R1#ping 150.1.12.2
Thu Mar 29 18:44:39.570 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.12.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/8/12 ms
RP/0/7/CPU0:R1#
```

Now, lets configure a loopback for R1 of 1.1.1.1/32

```
RP/0/7/CPU0:R1#conf t
Thu Mar 29 19:25:19.486 UTC
RP/0/7/CPU0:R1(config)#int l0
RP/0/7/CPU0:R1(config-if)#ip add 1.1.1.1/32
RP/0/7/CPU0:R1(config-if)#exit
RP/0/7/CPU0:R1(config)#exit
```

```
Uncommitted changes found, commit them before exiting(yes/no/cancel)?
[cancel]:yes
RP/0/7/CPU0:R1#
```

Notice this time I did not commit the change, but the system knew I was making changes and asked me if I wanted to commit them. I simply responded with YES and it saved them for me. If I did not want to save them, I could have entered NO and all the changes would have been tossed out. If I would have selected CANCEL, I would go back into edit mode.

Now time to configure some IPv6 addresses – first 2001:1:1:12::1/64

```
RP/0/7/CPU0:R1#conf t
Thu Mar 29 19:26:21.184 UTC
RP/0/7/CPU0:R1(cconfig)#int g0/3/0/2
RP/0/7/CPU0:R1(config-if)#ipv6 address 2001:1:1:12::1/64
RP/0/7/CPU0:R1(config-if)#exit
RP/0/7/CPU0:R1(config)#commit
Thu Mar 29 19:26:39.769 UTC
RP/0/7/CPU0:R1(config)#exit
```

And now we can try to PING our neighbor at 2001:1:1:12::2

```
RP/0/7/CPU0:R1#ping 2001:1:1:12::2
Thu Mar 29 19:29:11.893 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001:1:1:12::2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/16/68 ms
RP/0/7/CPU0:R1#
```

Let’s add one under our loopback interface as well – well use 2001::1/128

```
RP/0/7/CPU0:R1#conf t
RP/0/7/CPU0:R1(config)#int l0
RP/0/7/CPU0:R1(config-if)#ipv6 add 2001::1/128
RP/0/7/CPU0:R1(config-if)#commit
Thu Mar 29 19:30:49.920 UTC
RP/0/7/CPU0:R1(config-if)#
```

5. Interface Bundles

Etherchannels are also different between IOS and IOS XR. In typical IOS, they would be configured as such:

```
interface port-channel 1
 IP add 10.1.1.1 255.255.255.0
interface FastEthernet0/0
 channel-group 1
interface FastEthernet0/1
 channel-group 1
```

IOS XE is a little different then IOS as you can choose LACP:

```
interface GigabitEthernet0/0/2
 channel-group 12 mode active
 no shut
interface GigabitEthernet0/0/3
 channel-group 12 mode active
 no shut
interface Port-channel12
 ip address 10.1.1.1 255.255.255.252
```

And with IOS XR, it is a bit different again. So, for this example we will configure Ethernet Bundle 200

First on PE2:

```
RP/0/RSP0/CPU0:PE2#conf t
```

First up though, let's reset the interfaces back to factory by using the no interface command:

```
RP/0/RSP0/CPU0:PE2(config)#no int g0/0/0/11
RP/0/RSP0/CPU0:PE2(config)#commit
```

Instead of a port-channel interface, we do a bundle-ether interface

```
RP/0/RSP0/CPU0:PE2(config)#int bundle-ether 200
RP/0/RSP0/CPU0:PE2(config-if)#ip add 150.1.12.2 255.255.255.0
```

Now let's look at our bundle options:

```
RP/0/RSP0/CPU0:PE2(config-if)#bundle ?
  load-balancing  Load balancing commands on a bundle
  maximum-active  Set a limit on the number of links that can be active
  minimum-active  Set the minimum criteria for the bundle to be active
  shutdown        Bring all links in the bundle down to Standby state
  wait-while      Set the wait-while timeout for members of this bundle
```


Ok, since this is a bundle, we should put restrictions around the max and min links. Normally this is not a problem, but if you had to guarantee bandwidth (say 4G, then you might consider having the min links set to 4, and if you dropped below 4 the interface would go down).

```
RP/0/RSP0/CPU0:PE2(config-if)#bundle maximum-active links 2
RP/0/RSP0/CPU0:PE2(config-if)#bundle minimum-active links 1
```

Now let's take a quick look at our load balancing hash options:

```
RP/0/RSP0/CPU0:PE2(config-if)#bundle load-balancing hash ?
  dst-ip  Use the destination IP as the hash function
  src-ip  Use the source IP as the hash function
```

So, for this example we will use the src-ip

```
RP/0/RSP0/CPU0:PE2(config-if)#bundle load-balancing hash src-ip
```

Now, let's assign the interfaces to the bundle

```
RP/0/RSP0/CPU0:PE2(config-if)#int g0/1/0/11
```

Just like port-channels, the bundle ID should match the interface number you created. But here we will also look at what bundle options we have:

```
RP/0/RSP0/CPU0:PE2(config-if)#bundle id 200 mode ?
  active   Run LACP in active mode over the port.
  on       Do not run LACP over the port.
  passive  Run LACP in passive mode over the port.
```

There are three ways that LACP will link aggregate:

Switch 1	Switch 2	Notes
Active	Active	This is the recommended configuration.
Active	Passive	Link will aggregate once negotiation is done
On	On	Aggregation will happen, but not reccomded

We will use LACP in ACTIVE mode as that is what is recommended by Cisco:

```
RP/0/RSP0/CPU0:PE2(config-if)#bundle id 200 mode active
RP/0/RSP0/CPU0:PE2(config-if)#no shut
```

And do the same for G0/0/0/11:

```
RP/0/RSP0/CPU0:PE2(config-if)#int g0/0/0/11
RP/0/RSP0/CPU0:PE2(config-if)#bundle id 200 mode ac
RP/0/RSP0/CPU0:PE2(config-if)#no shut
```

```
Now let's check our config before we commit:
RP/0/RSP0/CPU0:PE2(config-if)#show config
Fri Apr 27 01:46:28.451 UTC
Building configuration...
!! IOS XR Configuration 4.1.2
interface Bundle-Ether200
 ipv4 address 150.1.12.2 255.255.255.0
 bundle load-balancing hash src-ip
 bundle maximum-active links 2
 bundle minimum-active links 1
!
interface GigabitEthernet0/0/0/11
 bundle id 200 mode active
 no shutdown
!
interface GigabitEthernet0/1/0/11
 bundle id 200 mode active
 no shutdown
!
end

RP/0/RSP0/CPU0:PE2(config)#commit
Fri Apr 27 01:46:44.692 UTC

Now we can do the other Router, PE1
RP/0/RSP0/CPU0:PE1(config)#no int g0/0/0/11
RP/0/RSP0/CPU0:PE1(config)#commit
Fri Apr 27 01:49:05.892 UTC
RP/0/RSP0/CPU0:PE1(config)#int bundle-ether 200
RP/0/RSP0/CPU0:PE1(config-if)#ip add 150.1.12.1/24
RP/0/RSP0/CPU0:PE1(config-if)#bundle maximum-active links 2
RP/0/RSP0/CPU0:PE1(config-if)#bundle minimum-active links 1
RP/0/RSP0/CPU0:PE1(config-if)#bundle load-balancing hash src-ip
RP/0/RSP0/CPU0:PE1(config-if)#int g0/1/0/11
RP/0/RSP0/CPU0:PE1(config-if)#bundle id 200 mode act
RP/0/RSP0/CPU0:PE1(config-if)#no shut
RP/0/RSP0/CPU0:PE1(config-if)#int g0/0/0/11
RP/0/RSP0/CPU0:PE1(config-if)#bundle id 200 mode act
RP/0/RSP0/CPU0:PE1(config-if)#no shut
RP/0/RSP0/CPU0:PE1(config-if)#exit
RP/0/RSP0/CPU0:PE1(config)#show config
Fri Apr 27 01:50:34.351 UTC
Building configuration...
!! IOS XR Configuration 4.1.2
interface Bundle-Ether200
 ipv4 address 150.1.12.1 255.255.255.0
 bundle load-balancing hash src-ip
 bundle maximum-active links 2
 bundle minimum-active links 1
!
```

```
interface GigabitEthernet0/0/0/11
 bundle id 200 mode active
 no shutdown
!
interface GigabitEthernet0/1/0/11
 bundle id 200 mode active
 no shutdown
!
end

RP/0/RSP0/CPU0:PE1(config)#commit
Fri Apr 27 01:50:37.705 UTC
RP/0/RSP0/CPU0:PE1(config)#
```

Now, let’s look at our bundle interface:

```
RP/0/RSP0/CPU0:PE1#sh int bundle-eth 200
Fri Apr 27 01:51:04.668 UTC
Bundle-Ether200 is up, line protocol is up
  Interface state transitions: 1
  Hardware is Aggregated Ethernet interface(s), address is 6c9c.ed2d.0bab
  Internet address is 150.1.12.1/24
  MTU 1514 bytes, BW 2000000 Kbit (Max: 2000000 Kbit)
     reliability 255/255, txload 0/255, rxload 0/255
  Encapsulation ARPA,
  Full-duplex, 2000Mb/s
  loopback not set,
  ARP type ARPA, ARP timeout 04:00:00
    No. of members in this bundle: 2
        GigabitEthernet0/0/0/11 Full-duplex      1000Mb/s     Active
        GigabitEthernet0/1/0/11 Full-duplex      1000Mb/s     Active
  Last input 00:00:18, output 00:00:18
  Last clearing of "show interface" counters never
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     15 packets input, 1792 bytes, 50 total input drops
     0 drops for unrecognized upper-level protocol
     Received 2 broadcast packets, 13 multicast packets
              0 runts, 0 giants, 0 throttles, 0 parity
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     12 packets output, 1408 bytes, 0 total output drops
```

As we can see, we are UP and have a full-duplex bandwidth of 2Gs.

So, let's PING!

```
RP/0/RSP0/CPU0:PE1#ping 150.1.12.2
Fri Apr 27 01:51:12.692 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.12.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/16 ms
RP/0/RSP0/CPU0:PE1#
```

Cool! The bundle is working.

Now we can check out some of the details:

```
RP/0/RSP0/CPU0:PE1#sh bundle bundle-ether 200
Fri Apr 27 02:13:16.767 UTC

Bundle-Ether200
  Status:                                    Up
  Local links <active/standby/configured>:   2 / 0 / 2
  Local bandwidth <effective/available>:     2000000 (2000000) kbps
  MAC address (source):                      6c9c.ed2d.0bab (Chassis pool)
  Minimum active links / bandwidth:          1 / 1 kbps
  Maximum active links:                      2
  Wait while timer:                          2000 ms
  Load balancing:
    Link order signaling:                    Not configured
    Hash type:                               Src-IP
  LACP:                                      Operational
    Flap suppression timer:                  Off
    Cisco extensions:                        Disabled
  mLACP:                                     Not configured
  IPv4 BFD:                                  Not configured

  Port                  Device           State        Port ID         B/W, kbps
  --------------------  ---------------  -----------  --------------  ----------
  Gi0/0/0/11            Local            Active       0x8000, 0x0002     1000000
      Link is Active
  Gi0/1/0/11            Local            Active       0x8000, 0x0001     1000000
      Link is Active
RP/0/RSP0/CPU0:PE1#
```

Now we can look at LACP:

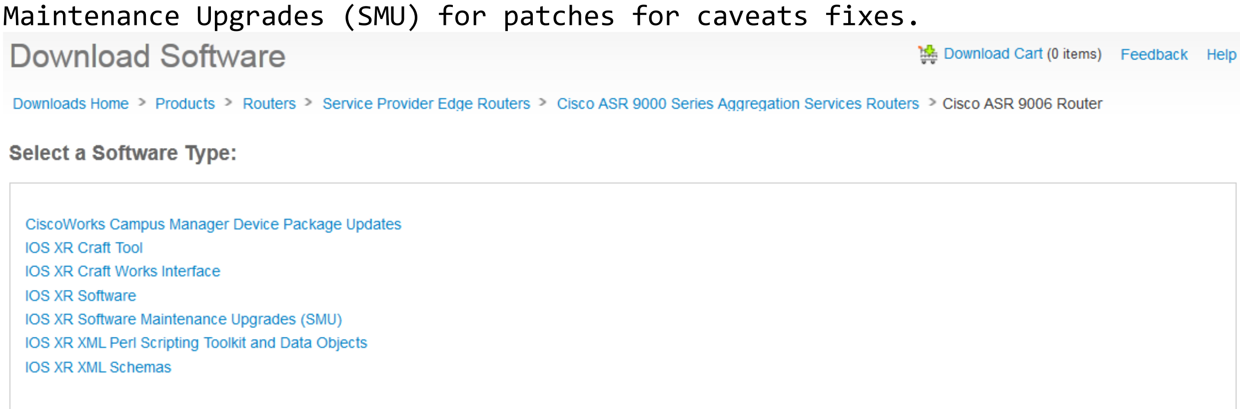
```
RP/0/RSP0/CPU0:PE1#show lacp
Fri Apr 27 01:52:35.115 UTC
State: a - Port is marked as Aggregatable.
       s - Port is Synchronized with peer.
       c - Port is marked as Collecting.
       d - Port is marked as Distributing.
       A - Device is in Active mode.
       F - Device requests PDUs from the peer at fast rate.
       D - Port is using default values for partner information.
       E - Information about partner has expired.

Bundle-Ether200

  Port          (rate)  State    Port ID       Key    System ID
  --------------------  -------- ------------- ------ ------------------------
Local
  Gi0/0/0/11       30s  ascdA--- 0x8000,0x0002 0x00c8 0x8000,6c-9c-ed-2d-0b-ac
   Partner         30s  ascdA--- 0x8000,0x0003 0x00c8 0x8000,6c-9c-ed-2d-1f-cc
  Gi0/1/0/11       30s  ascdA--- 0x8000,0x0001 0x00c8 0x8000,6c-9c-ed-2d-0b-ac
   Partner         30s  ascdA--- 0x8000,0x0004 0x00c8 0x8000,6c-9c-ed-2d-1f-cc

  Port                  Receive    Period Selection  Mux       A Churn P Churn
  --------------------  ---------- ------ ---------- --------- ------- -------
Local
  Gi0/0/0/11            Current    Slow   Selected   Distrib   None    None
  Gi0/1/0/11            Current    Slow   Selected   Distrib   None    None
RP/0/RSP0/CPU0:PE1#
```

6. Software installation, PIE packages, and patches

As part of any system, from time to time you need to install updates, patches, and upgrade code. The joys of IOS XR code is that you can actually installed patches that fix bugs, you can perform in-service upgrades and not take down the router (provided you have a dual-supervisor router), as well as add new services to the code. All the necessary PIE packages can be found in the main image, they are not available separately.

You can get the main image from CCO Support and Downloads. To navigate to the download, select:
Products -> Routers -> Service Provider Edge Routers -> ASR 9000 -> ASR 9006

Then select IOS XR Software for the main images or IOS XR Software Maintenance Upgrades (SMU) for patches for caveats fixes.



Once you select the IOS XR Software, the most recent version of code will be presented on the screen. Select the version that you need and proceed to download it. If you get an error that a contract is required, please open a Cisco TAC case requesting access, they will need the serial number of the chassis in order to prove support.

Once you have the image on your computer, we will now need to transfer it. Since the image is over 400 Megs as of 4.1.2, and 4.2.0 is over 700 Megs, TFTP is probably not going to cut it (most TFTP apps do not support files over 32 megs). What you might need to do is find an FTP server program to use - I recommend FileZilla – but that is ultimately up to you. Once you have your FTP server setup and ready to go, we now need to get the image copied. For this example, I am using a username of Cisco and a password of cisco

```
RP/0/RSP0/CPU0:R1#  copy ftp://1.1.1.2/ASR9K-iosxr-k9-4.1.2.tar compactflash:
Tue Apr 10 02:00:23.038 UTC
Source username: [anonymous]?cisco
Source password: cisco
Destination filename [/compactflash:/ASR9K-iosxr-k9-4.1.2.tar]? (just hit
enter)
```

The file copy will now start and will take some time (you will see CCCCCCCCCCCCCCCCCCC) – these are large images, so patience is a virtue.

Once the file copy is complete, check the compact flash to make sure the images transferred successfully.

```
RP/0/RSP0/CPU0:R2#dir compactflash:
Tue Apr 10 02:01:37.766 UTC

Directory of compactflash:

131104   -rw-  9216      Sun Jan  2 08:01:19 2000  Test
6        drwx  4096      Tue Jan  4 23:33:44 2000  LOST.DIR
131264   -rw-  453611520 Thu Apr  5 22:14:28 2012  ASR9K-iosxr-k9-4.1.2.tar

1022427136 bytes total (568795136 bytes free)
```

Now that we have the image, we need to extract the tar file. That is done from ADMIN mode. You enter admin mode by typing admin at the command promt.

```
RP/0/RSP0/CPU0:R2#admin
Tue Apr 10 02:03:27.052 UTC
RP/0/RSP0/CPU0:R2(admin)#
```

Once there, we can install the tar image using the install command:

```
RP/0/RSP0/CPU0:ios(admin)#install add tar compactflash:ASR9K-iosxr-k9-
4.1.2.tar
```

Once you enter that command, the system will start to process the file and show output:

```
Mon Apr  9 21:29:41.420 UTC
Install operation 1 '(admin) install add tar
/compactflash:ASR9K-iosxr-k9-4.1.2.tar' started by user 'admin' via CLI at
21:29:41 UTC Mon Apr 09 2012.
Info:     The following files were extracted from the tar file
Info:     '/compactflash:ASR9K-iosxr-k9-4.1.2.tar' and will be added to the
Info:     entire router:
Info:
Info:         asr9k-mcast-p.pie-4.1.2
Info:         asr9k-mpls-p.pie-4.1.2
Info:         asr9k-mini-p.pie-4.1.2
Info:         asr9k-mini-p.vm-4.1.2 (skipped - not a pie)
Info:         asr9k-doc-p.pie-4.1.2
```

```
Info:         asr9k-video-p.pie-4.1.2
Info:         asr9k-mgbl-p.pie-4.1.2
Info:         asr9k-optic-p.pie-4.1.2
Info:         asr9k-upgrade-p.pie-4.1.2
Info:         asr9k-k9sec-p.pie-4.1.2
Info:         README-ASR9K-k9-4.1.2.txt (skipped - not a pie)
Info:
The install operation will continue asynchronously.
```

This operation will happen in the background, you will be returned to the command prompt. Once the process is finished, the similar text will appear on the prompt:

```
P/0/RSP0/CPU0:ios(admin)#Info:     The following packages are now available
to be activated:
Info:
Info:         disk0:asr9k-mcast-p-4.1.2
Info:         disk0:asr9k-mpls-p-4.1.2
Info:         disk0:asr9k-mini-p-4.1.2
Info:         disk0:asr9k-doc-p-4.1.2
Info:         disk0:asr9k-video-p-4.1.2
Info:         disk0:asr9k-mgbl-p-4.1.2
Info:         disk0:asr9k-optic-4.1.2
Info:         disk0:asr9k-upgrade-p-4.1.2
Info:         disk0:asr9k-k9sec-p-4.1.2
Info:
Info:     The packages can be activated across the entire router.
Info:
Install operation 1 completed successfully at 21:38:52 UTC Mon Apr 09 2012.
```

Now that we have the image there, we need to see what inactive PIEs we have to install and activate. The command here is show install inactive summary

```
RP/0/RSP0/CPU0:ios(admin)#sh install inactive summary
Mon Apr  9 21:59:10.354 UTC
Default Profile:
  SDRs:
    Owner
  Inactive Packages:
    disk0:asr9k-upgrade-p-4.1.2
    disk0:asr9k-optic-4.1.2
    disk0:asr9k-doc-p-4.1.2
    disk0:asr9k-k9sec-p-4.1.2
    disk0:asr9k-video-p-4.1.2
    disk0:asr9k-mpls-p-4.1.2
    disk0:asr9k-mgbl-p-4.1.2
    disk0:asr9k-mcast-p-4.1.2
```

Now we should be able to activate and install one of the PIE images, here we will activate the MPLS one.

```
RP/0/RSP0/CPU0:ios(admin)#install activate disk0:asr9k-mpls-p-4.1.2
Mon Apr  9 21:59:43.108 UTC
Install operation 2 '(admin) install activate disk0:asr9k-mpls-p-4.1.2'
started
by user 'admin' via CLI at 21:59:43 UTC Mon Apr 09 2012.
Error:    Cannot proceed with the operation because the upgrade package
Error:    disk0:asr9k-upgrade-p-4.1.2 is present on boot disk.
Error:    The disk0:asr9k-upgrade-p-4.1.2 package should only be used when
Error:    upgrading from software versions prior to 4.0.0. Once the upgrade
is
Error:    complbe immediately doved.  No
Error:    further install operations will be allowed until this is completed.
Error:
Error:    Remove the package disk0:asr9k-upgrade-p-4.1.2 from the entire
router
Error:    by executing the 'install remove disk0:asr9k-upgrade-p-4.1.2'
command
Error:    in admin mode.
Error:    No further install operations will be allowed until this is
Error:    completed.
Install operation 2 failed at 21:59:44 UTC Mon Apr 09 2012.
```

Ahh, we got an error! The error output tells us that we need to remove the upgrade package from the disk via the install remove command:

```
RP/0/RSP0/CPU0:ios(admin)#install remove disk0:asr9k-upgrade-p-4.1.2
Mon Apr  9 22:00:13.538 UTC
Install operation 3 '(admin) install remove disk0:asr9k-upgrade-p-4.1.2'
started by user 'admin' via CLI at 22:00:13 UTC Mon Apr 09 2012.
Info:     This operation will remove the following package:
Info:         disk0:asr9k-upgrade-p-4.1.2
```

Now we need to confirm it by just hitting enter:

```
Proceed with removing these packages? [confirm] (just hit enter to confirm)
The install operation will continue asynchronously.
```

Now if we do a show install summary, it will tell us that we are in the process of doing something:

```
RP/0/RSP0/CPU0:ios(admin)#sh install summary
Mon Apr  9 22:00:22.060 UTC
Default Profile: Currently affected by install operation 3
  SDRs:
    Owner
  Active Packages:
    No packages.
```

Once completed, we will be notified on the cli

```
RP/0/RSP0/CPU0:ios(admin)#Install operation 3 completed successfully at
22:00:39 UTC Mon Apr 09 2012.
```

Now, we should be able to install the MPLS PIE

```
RP/0/RSP0/CPU0:ios(admin)#install activate disk0:asr9k-mpls-p-4.1.2
Mon Apr  9 22:03:38.202 UTC
Install operation 4 '(admin) install activate disk0:asr9k-mpls-p-4.1.2'
started
by user 'admin' via CLI at 22:03:38 UTC Mon Apr 09 2012.
Info:     Install Method: Parallel Process Restart
The install operation will continue asynchronously.
RP/0/RSP0/CPU0:ios(admin)#RP/0/RSP0/CPU0:Apr  9 22:04:32.428 :
insthelper[65]: ISSU: Starting sysdb bulk start session
Info:     The changes made to software configurations will not be persistent
Info:     across system reloads. Use the command '(admin) install commit' to
Info:     make changes persistent.
Info:     Please verify that the system is consistent following the software
Info:     change using the following commands:
Info:         show system verify
Info:         install verify packages
RP/0/RSP0/CPU0:Apr  9 22:04:45.933 : instdir[229]: %INSTALL-INSTMGR-4-
ACTIVE_SOFTWARE_COMMITTED_INFO : The currently active software is not
committed. If the system reboots then the committed software will be used.
Use 'install commit' to commit the active software.
Install operation 4 completed successfully at 22:04:45 UTC Mon Apr 09 2012.
```

If you want to see the status of the install, you can use the *show install request* command and it will show you the percentage complete.

```
RP/0/RSP0/CPU0:c20.newthk01(admin)#sh install request
Sat May 12 00:43:54.386 UTC
Install operation 4 '(admin) install activate disk0:asr9k-mpls-p-4.1.2'
started
by user 'neteng' via CLI at 00:42:50 UTC Sat May 12 2012.
The operation is 85% complete
The operation can still be aborted.
RP/0/RSP0/CPU0:c20.newthk01(admin)#
```

Once the installation is complete, we need to COMMIT the installation using the install commit command

```
RP/0/RSP0/CPU0:ios(admin)#install commit
Mon Apr  9 22:07:17.014 UTC
Install operation 5 '(admin) install commit' started by user 'admin' via CLI
at
22:07:17 UTC Mon Apr 09 2012.
\ 100% complete: The operation can no longer be aborted (ctrl-c for
options)RP/0/RSP0/CPU0:Apr  9 22:07:20.238 : instdir[229]: %INSTALL-INSTMGR-
4-ACTIVE_SOFTWARE_COMMITTED_INFO : The currently active software is now the
same as the committed software.
Install operation 5 completed successfully at 22:07:20 UTC Mon Apr 09 2012.
```

Now if we look at our show install active summary command, we now have the MPLS PIE

```
RP/0/RSP0/CPU0:R2(admin)#sh install active summary
Tue Apr 10 02:12:38.009 UTC
Default Profile:
  SDRs:
    Owner
  Active Packages:
    disk0:asr9k-mini-p-4.1.2
    disk0:asr9k-k9sec-p-4.1.2
    disk0:asr9k-mpls-p-4.1.2

RP/0/RSP0/CPU0:R2(admin)#
```

When it comes to patches, they are rather easy as well. They pretty much follow the same process as packages. Copy the file to flash, install the tar, then activate the patch.
For this example, we will copy the CSCtu30994 - rn_preorder_key_successor_int function is constantly looping per the readme file.

First up, lets copy it from the TFTP server to our CompactFlash card:

```
RP/0/RSP0/CPU0:ASR01#copy tftp: compactflash:
Tue May 15 06:12:19.645 UTC
Address or name of remote host [192.168.1.1]? (enter)
Source filename [/tftp:]?asr9k-p-4.1.2.CSCtu30994.tar
Destination filename [/compactflash:/asr9k-p-4.1.2.CSCtu30994.tar]?
Accessing tftp://10.100.100.17/asr9k-p-4.1.2.CSCtu30994.tar
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
911360 bytes copied in      6 sec (   134936)bytes/sec
```

Once copied, lets switch to ADMIN mode.

```
RP/0/RSP0/CPU0:ASR01#admin
Tue May 15 06:15:01.739 UTC
```

Now we can add the TAR files

```
RP/0/RSP0/CPU0:ASR01(admin)#install add tar compactflash:asr9k-p-
4.1.2.CSCtu30994.tar
Tue May 15 06:15:03.744 UTC
/compactflash:asr9k-p-4.1.2.CSCtu30994.tar' started by user 'admin' via CLI
at
06:15:04 UTC Tue May 15 2012.
Info:     The following files were extracted from the tar file
Info:     '/compactflash:asr9k-p-4.1.2.CSCtu30994.tar' and will be added to
the
Info:     entire router:
Info:
Info:         asr9k-p-4.1.2.CSCtu30994.pie
```

```
Info:         asr9k-p-4.1.2.CSCtu30994.txt (skipped - not a pie)
Info:
The install operation will continue asynchronously.
```

And once the TAR has been added, the following message will appear:

```
Info:     The following package is now available to be activated:
Info:
Info:         disk0:asr9k-p-4.1.2.CSCtu30994-1.0.0
Info:
Info:     The package can be activated across the entire router.
Info:
Install operation 27 completed successfully at 06:15:39 UTC Tue May 15 2012.
```

Now we can activate this patch:

```
RP/0/RSP0/CPU0:ASR01(admin)#install activate disk0:asr9k-p-4.1.2.CSCtu30994-
1.0.0
Tue May 15 06:15:45.276 UTC
Install operation 28 '(admin) install activate
disk0:asr9k-p-4.1.2.CSCtu30994-1.0.0' started by user 'admin' via CLI at
06:15:45 UTC Tue May 15 2012.
Info:     Install Method: Parallel Process Restart
The install operation will continue asynchronously.

Info:     The changes made to software configurations will not be persistent
Info:     across system reloads. Use the command '(admin) install commit' to
Info:     make changes persistent.
Info:     Please verify that the system is consistent following the software
Info:     change using the following commands:
Info:         show system verify
Info:         install verify packages
```

Once the install is done we need to commit it:

```
RP/0/RSP0/CPU0:ASR01(admin)#install commit
Tue May 15 06:17:06.359 UTC
Install operation 29 '(admin) install commit' started by user 'admin' via CLI
at 06:17:06 UTC Tue May 15 2012.
\ 100% complete: The operation can no longer be aborted (ctrl-c for
options)RP/0/RSP0/CPU0:May 15 06:17:09.967 : instdir[233]: %INSTALL-INSTMGR-
4-ACTIVE_SOFTWARE_COMMITTED_INFO : The currently active software is now the
same as the committed software.
Install operation 29 completed successfully at 06:17:09 UTC Tue May 15 2012.
```

And like that we are patched.

Now, that was not one that required a reload, if you have one of them like CSCtw84381, here you will be prompted that you need to reload.

```
RP/0/RSP0/CPU0:ASR01(admin)#install activate disk0:asr9k-p-4.1.2.CSCtw84381-1.0.0
Tue May 15 06:30:37.867 UTC
Install operation 35 '(admin) install activate
disk0:asr9k-p-4.1.2.CSCtw84381-1.0.0' started by user 'admin' via CLI at
06:30:38 UTC Tue May 15 2012.
Info:     This operation will reload the following nodes in parallel:
Info:         0/RSP0/CPU0 (RP) (SDR: Owner)
Info:         0/0/CPU0 (LC) (SDR: Owner)
Info:         0/1/CPU0 (LC) (SDR: Owner)
```

See, it is asking you to proceed - hit enter for Y

```
Proceed with this install operation (y/n)? [y] (enter)
Info:     Install Method: Parallel Reload
The install operation will continue asynchronously.
```

Once the install is complete, the router will reload and you will need to relogin. Do not forget to do INSTALL COMMIT!!!

Note from the Cisco website

(http://www.cisco.com/en/US/docs/routers/asr9000/software/asr9k_r3.9/system_management/command/reference/yr39asr9k_chapter14.html)

Install operations are activated according to the method encoded in the package being activated. Generally, this method has the least impact for routing and forwarding purposes, but it may not be the fastest method from start to finish and can require user interaction by default. To perform the installation procedure as quickly as possible, you can specify the parallel-reload keyword. This action forces the installation to perform a parallel reload, so that all cards on the router reload simultaneously and then come up with the new software. This impacts routing and forwarding, but it ensures that the installation is performed without other issues.

7. Licensing

The ever loving Cisco licensing – well, not just Cisco but all vendors have some type of licensing. With the IOS XR in this case, we need a license to run VRF interfaces on our line cards. In order to request a license, you need to have a PAK key that you purchase, once you have that you will need to gather some information to request the license key.

From the command promt, enter the admin mode

```
RP/0/RSP0/CPU0:R2#admin
Tue Apr 17 01:34:35.939 UTC
```

From there, enter the command show license udi

```
RP/0/RSP0/CPU0:R2(admin)#show license udi
Tue Apr 17 01:34:38.950 UTC

Local Chassis UDI Information:
  PID         : ASR-9010-AC
  S/N         : FOXXXXXAAAA
  Operation ID: 1

RP/0/RSP0/CPU0:R2(admin)#
```

This information will be used on the Cisco License site – www.cisco.com/go/license (CCO Account required). Once you have submitted the PAK request, Licensing@cisco.com will send you the license file as an attachment within a few hours.

Once you have the file, you will need to copy it to the router via TFTP or some other method. The license file will also include the instructions to add it, I have included them here as well.

```
RP/0/RSP0/CPU0:R1#copy  tftp: compactflash
Wed Apr 11 05:23:23.259 UTC
Address or name of remote host []?1.1.1.2
Source filename [/tftp:]?foo.lic
Destination filename [/disk0a:/usr/compactflash]? (enter)
Accessing tftp://1.1.1.2/foo.lic
C
1199 bytes copied in      0 sec
RP/0/RSP0/CPU0:R1#admin
```

Next we can use the license add command from Admin mode

```
RP/0/1/CPU0:CRS#
RP/0/1/CPU0:CRS(admin)#license add compactflash:/foo.lic
RP/0/1/CPU0:Mar 16 16:01:37.077 : licmgr[252]: %LICENSE-LICMGR-6-
LOAD_LICENSE_FILE_OK : All licenses from license file compactflash:/foo.lic
added successfully

License command "license add compactflash:/foo.lic sdr Owner" completed
successfully.
RP/0/1/CPU0:CRS(admin)#
```

Now we need to see if is has been added via the show license command

```
RP/0/1/CPU0:CRS(admin)#show license

FeatureID: foo (Slot based, Permanent)
 Available for use         1
 Allocated to location     0
 Active                    0
  Pool: Owner
   Status: Available     1    Operational:    0
  Pool: sdr1
   Status: Available     0    Operational:    0
```

Once the license has been successfully added, we now need to assign it to a line card slot. Again, this is done from Admin config mode

```
RP/0/RSP0/CPU0:R1(admin)#config
```

To assign the license, the command is license (License) location (LocationID). In our case, we are going to apply A9K-iVRF-LIC. The question mark will show you what location are available for this license.

```
RP/0/RSP0/CPU0:R1(admin-config)#license A9K-iVRF-LIC location ?
  0/0/CPU0     Fully qualified location specification
  0/1/CPU0     Fully qualified location specification
  0/RSP0/CPU0  Fully qualified location specification
  WORD         Fully qualified location specification
  all          all locations
```

Now we can apply the licenses that we have to 0/0 and 0/1:

```
RP/0/RSP0/CPU0:R1(admin-config)#license A9K-iVRF-LIC location 0/0/CPU0
RP/0/RSP0/CPU0:R1(admin-config)#license A9K-iVRF-LIC location 0/1/CPU0
RP/0/RSP0/CPU0:R1(admin-config)#commit
Thu Apr 19 03:13:44.883 UTC
RP/0/RSP0/CPU0:R1(admin-config)#exit
RP/0/RSP0/CPU0:R1(admin)#exit
```

Once installed, we can check using the *show license* command.

```
RP/0/RSP0/CPU0:R1#sh license
Thu Apr 19 03:13:51.432 UTC
FeatureID: A9K-iVRF-LIC (Slot based, Permanent)
  Total licenses 2
  Available for use         0
  Allocated to location     0
  Active                    2
  Store name             Permanent
  Store index               1
    Pool: Owner
      Total licenses in pool: 2
      Status: Available     0    Operational:    2
      Locations with licenses: (Active/Allocated) [SDR]
              0/1/CPU0        (1/0) [Owner]
              0/0/CPU0        (1/0) [Owner]



RP/0/RSP0/CPU0:R1#
```

There they are, assigned to 0/1 and 0/0 as requested.

8. Aliases

From IOS, Aliases can sometimes make life easier on you and your support staff. In IOS XR, aliases get ramped up a bit, but first lest cover the basics.

For this example, we can create an alias to show all the IPV4 interfaces in a brief using a single command, SHV4BR

```
RP/0/7/CPU0:R1#conf t
Mon Apr 16 15:05:26.064 UTC
RP/0/7/CPU0:R1(config)#alias SHV4BR show ipv4 int brief
RP/0/7/CPU0:R1(config)#commit
Mon Apr 16 15:05:44.043 UTC
```

Now, lets test the command:

```
RP/0/7/CPU0:R1#shv4br
```

As you can seem the system will re-enter the command from the alias

```
RP/0/7/CPU0:R1#show ipv4 int brief
Mon Apr 16 15:05:49.094 UTC

Interface                      IP-Address      Status                Protocol
Loopback0                      1.1.1.1         Up                    Up
Loopback100                    100.100.100.100 Up                    Up
Loopback666                    6.6.6.6         Up                    Up
Loopback667                    6.6.6.7         Up                    Up
MgmtEth0/7/CPU0/0              unassigned      Shutdown              Down
MgmtEth0/7/CPU0/1              unassigned      Shutdown              Down
MgmtEth0/7/CPU0/2              unassigned      Shutdown              Down
GigabitEthernet0/3/0/0         unassigned      Down                  Down
GigabitEthernet0/3/0/1         unassigned      Down                  Down
GigabitEthernet0/3/0/2         150.1.12.1      Up                    Up
GigabitEthernet0/3/0/3         unassigned      Up                    Up
MgmtEth0/6/CPU0/0              unassigned      Shutdown              Down
MgmtEth0/6/CPU0/1              unassigned      Shutdown              Down
MgmtEth0/6/CPU0/2              unassigned      Shutdown              Down
RP/0/7/CPU0:R1#
```

Now, onto a cool feature, interface alias!
We will create an alias for interface GigabitEthernet0/3/0/2, our connection to R2 on this router.

```
RP/0/7/CPU0:R1#conf t
Mon Apr 16 15:23:26.451 UTC
RP/0/7/CPU0:R1(config)#alias R2Connection gig0/3/0/2
RP/0/7/CPU0:R1(config)#commit
```

Now, let's see what happens when we do a show int for that alias

```
RP/0/7/CPU0:R1#sh int r2connection
RP/0/7/CPU0:R1#sh int gig0/3/0/2
Mon Apr 16 15:24:00.745 UTC
GigabitEthernet0/3/0/2 is up, line protocol is up
<--- SNIP -→
     0 output buffer failures, 0 output buffers swapped out
     5 carrier transitions
```

Pretty neat, but it gets better – we can actually configure that alias as well!

```
RP/0/7/CPU0:R1#conf t
Mon Apr 16 15:24:06.626 UTC
RP/0/7/CPU0:R1(config)#int r2connection
RP/0/7/CPU0:R1(config)#int gig0/3/0/2
RP/0/7/CPU0:R1(config-if)#exit
RP/0/7/CPU0:R1(config)#exit
RP/0/7/CPU0:R1#
```

Now, there is another trick with IOS XR, and that is variables!

So, what can we do with Variables and Aliases? Well, if there is a command that you use quite often – say show interface, why not change it to an alias with a variable.

For this example, we will create sint (show interface) and use variable (var1).

But first, let us look at what happens when you add a question mark (?) to the end of the command in configuration mode:

```
RP/0/RSP0/CPU0:c21.lab(config)#alias sint ?
  LINE  Alias body with optional parameters e.g,(name) show $name
```

As you can see, it even tells you that you can use variables, might not be obvious, that that is what (name) is.

So, let us create our alias:

```
RP/0/RSP0/CPU0:c21.lab(config)#alias sint (var1) show interface $var1
RP/0/RSP0/CPU0:c21.lab(config)#commit
RP/0/RSP0/CPU0:c21.lab(config)#
```

Now we can test it on Interface Bundle-Eth 100:

```
RP/0/RSP0/CPU0:c21.lab#sint(Bundle-Eth100)
RP/0/RSP0/CPU0:c21.lab#show interface Bundle-Eth100
Bundle-Ether100 is up, line protocol is up
  Interface state transitions: 3
  Hardware is Aggregated Ethernet interface(s), address is 6c9c.ed2d.0bab
  Internet address is 157.238.206.3/31
  MTU 1514 bytes, BW 20000000 Kbit (Max: 20000000 Kbit)
```

```
     reliability 255/255, txload 0/255, rxload 0/255
  Encapsulation ARPA,
  Full-duplex, 20000Mb/s
  loopback not set,
  ARP type ARPA, ARP timeout 04:00:00
    No. of members in this bundle: 2
        TenGigE0/0/0/0       Full-duplex      10000Mb/s    Active
        TenGigE0/1/0/0       Full-duplex      10000Mb/s    Active
  Last input 00:00:00, output 00:00:00
  Last clearing of "show interface" counters never
  5 minute input rate 1000 bits/sec, 1 packets/sec
  5 minute output rate 1000 bits/sec, 1 packets/sec
     1509709 packets input, 641971670 bytes, 411 total input drops
     0 drops for unrecognized upper-level protocol
     Received 5 broadcast packets, 1298355 multicast packets
              0 runts, 0 giants, 0 throttles, 0 parity
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     1518092 packets output, 642666596 bytes, 0 total output drops
     Output 6 broadcast packets, 1300886 multicast packets
     0 output errors, 0 underruns, 0 applique, 0 resets
     0 output buffer failures, 0 output buffers swapped out
     0 carrier transitions


RP/0/RSP0/CPU0:c21.lab#
```

Now, another trick we can do is nested aliases!
Lets modify the alias *sint* to show the interface as well as the configuration.

```
RP/0/RSP0/CPU0:c21.lab(config)#alias sint (var1) show interface $var1; show
run int $var1
RP/0/RSP0/CPU0:c21.lab(config)#commit
RP/0/RSP0/CPU0:c21.lab(config)#
```

Now we can run that same command [sint(bundle-eth100)]again.

```
RP/0/RSP0/CPU0:c21.lab#sint(bundle-eth100)
RP/0/RSP0/CPU0:c21.lab#show interface bundle-eth100
Bundle-Ether100 is up, line protocol is up
  Interface state transitions: 3
  Hardware is Aggregated Ethernet interface(s), address is 6c9c.ed2d.0bab
  Internet address is 157.238.206.3/31
  MTU 1514 bytes, BW 20000000 Kbit (Max: 20000000 Kbit)
     reliability 255/255, txload 0/255, rxload 0/255
  Encapsulation ARPA,
  Full-duplex, 20000Mb/s
  loopback not set,
  ARP type ARPA, ARP timeout 04:00:00
```

```
    No. of members in this bundle: 2
        TenGigE0/0/0/0       Full-duplex      10000Mb/s    Active
        TenGigE0/1/0/0       Full-duplex      10000Mb/s    Active
  Last input 00:00:00, output 00:00:00
  Last clearing of "show interface" counters never
  5 minute input rate 1000 bits/sec, 1 packets/sec
  5 minute output rate 1000 bits/sec, 1 packets/sec
     1509849 packets input, 642030906 bytes, 411 total input drops
     0 drops for unrecognized upper-level protocol
     Received 5 broadcast packets, 1298474 multicast packets
              0 runts, 0 giants, 0 throttles, 0 parity
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     1518235 packets output, 642727790 bytes, 0 total output drops
     Output 6 broadcast packets, 1301008 multicast packets
     0 output errors, 0 underruns, 0 applique, 0 resets
     0 output buffer failures, 0 output buffers swapped out
     0 carrier transitions


RP/0/RSP0/CPU0:c21.lab#show run int bundle-eth100
interface Bundle-Ether100
 ipv4 address 157.238.206.3 255.255.255.254
 bundle load-balancing hash src-ip
 bundle maximum-active links 2
 bundle minimum-active links 1
!

RP/0/RSP0/CPU0:c21.lab#
```

As you can see, it did the *show interface bundle-eth100* and *show run interface bundle-eth100*

9. Wildcard Masks

A really cool thing with IOS XR is interface wildcards.

If you want to only see the Loopback interfaces, all of them. Normally you would do something like Show int br | in Loop, but with XR you can use a wildcard (*)

```
RP/0/7/CPU0:R1#sh int l* br
Mon Apr 16 17:21:08.088 UTC
      Intf       Intf        LineP              Encap  MTU        BW
      Name       State       State               Type (byte)    (Kbps)
----------------------------------------------------------------------
      Lo0          up          up             Loopback  1500          0
      Lo100        up          up             Loopback  1500          0
      Lo666        up          up             Loopback  1500          0
      Lo667        up          up             Loopback  1500          0
      Lo1000        up          up             Loopback  1500          0

RP/0/7/CPU0:R1#
```

This works the same if you want to see this in the running config:

```
RP/0/7/CPU0:R1#sh run in l*
Mon Apr 16 17:21:53.360 UTC
interface Loopback0
 ipv4 address 1.1.1.1 255.255.255.255
 ipv6 address 2001::1/128
!
interface Loopback100
 ipv4 address 100.100.100.100 255.255.255.255
!
interface Loopback666
 ipv4 address 6.6.6.6 255.255.255.255
!
interface Loopback667
 ipv4 address 6.6.6.7 255.255.255.255
!
interface Loopback1000
 vrf LAB
 ipv4 address 111.111.111.111 255.255.255.255
!

RP/0/7/CPU0:R1#
```

10.Processes

So, since IOS XR is based on QNX, the SHOW PROCESSES command is a bit different then you would see in IOS. In IOS XR, you can actually query processes and see what is going on.

Here is a way to see what BGP is doing :

```
RP/0/RSP0/CPU0:R2#sh processes bgp
Tue Apr 24 01:23:06.343 UTC
                  Job Id: 1039
                     PID: 2941214
         Executable path: /disk0/iosxr-routing-4.1.2/bin/bgp
              Instance #: 1
              Version ID: 00.00.0000
                 Respawn: ON
           Respawn count: 4
  Max. spawns per minute: 12
            Last started: Tue Apr 10 05:31:26 2012
           Process state: Run
           Package state: Normal
       Started on config: ipc/gl/ip-bgp/meta/speaker/default
                    core: MAINMEM
               Max. core: 0
               Placement: Placeable
            startup_path: /pkg/startup/bgp.startup
                   Ready: 0.338s
               Available: 25.582s
        Process cpu time: 63.719 user, 1.074 kernel, 64.793 total
JID  TID CPU Stack pri state        TimeInState    HR:MM:SS:MSEC   NAME
1039  1    1  312K  10 Receive        0:01:01:0795    0:00:00:0249 bgp
1039  2    1  312K  10 Receive      331:51:39:0224    0:00:00:0000 bgp
1039  3    0  312K  10 Receive      331:51:39:0223    0:00:00:0001 bgp
1039  4    1  312K  10 Sigwaitinfo  331:51:39:0129    0:00:00:0000 bgp
1039  5    0  312K  10 Receive        0:00:01:0764    0:00:00:0005 bgp
1039  6    1  312K  10 Receive        0:00:01:0760    0:00:00:0016 bgp
1039  7    0  312K  10 Receive        0:00:23:0239    0:00:02:0242 bgp
1039  8    0  312K  10 Receive        0:00:03:0321    0:00:02:0280 bgp
1039  9    1  312K  10 Receive        0:01:01:0796    0:00:00:0005 bgp
1039  10   1  312K  10 Receive        0:00:01:0786    0:00:00:0008 bgp
1039  11   0  312K  10 Receive        0:00:01:0786    0:00:00:0004 bgp
1039  12   0  312K  10 Receive        0:00:01:0786    0:00:00:0001 bgp
1039  13   1  312K  10 Receive        0:00:02:0861    0:00:00:0006 bgp
1039  14   1  312K  10 Receive        0:00:03:0189    0:00:59:0750 bgp
1039  15   0  312K  10 Receive        0:00:01:0816    0:00:00:0223 bgp
1039  16   1  312K  10 Receive      331:51:39:0047    0:00:00:0000 bgp
1039  17   1  312K  10 Receive        0:00:27:0039    0:00:00:0000 bgp
1039  18   1  312K  10 Receive       75:14:02:0621    0:00:00:0002 bgp
```

Now, since IOS XR is based on a flavor of Unix, we have a command similar to TOP called monitor processes.

```
RP/0/RSP0/CPU0:R2#monitor processes
Tue Apr 24 01:27:41.959 UTC
Computing times...
```

(Screen Clears and the following data is refreshed)

```
287 processes; 1320 threads; 1086 timers, 6265 channels, 8489 fds
CPU states: 99.6% idle, 0.2% user, 0.1% kernel
Memory: 4096M total, 2762M avail, page size 4K

      JID TIDS Chans  FDs Tmrs   MEM   HH:MM:SS   CPU  NAME
        1   13   291  204    1     0  998:56:18  0.12% procnto-600-smp-instr
    65744    1     1   11    0    1M    0:00:00  0.09% ptop
      340    2    16   20    4  340K    0:41:16  0.05% sc
       60   15    44   20    7    4M    0:18:05  0.04% eth_server
       95   22   276   35    4  924K    0:02:03  0.00% sysmgr
      152    4    16   24    6  700K    0:00:08  0.00% canb-server
       71    2     7   11    1  236K    0:00:04  0.00% mdio_sup
      202    9    39   62   14    1M    0:00:03  0.00% ether_ctrl_mgmt
       89    1     6    3    1  104K    0:00:55  0.00% serdrvr
      355    4     9   15    2  260K    0:00:00  0.00% ssm_process
```

11. Remote Access Services – Telnet and SSH

We need to have a way to remote access this device, and by default SSH and TELNET are not enabled.

First up, the easy one – telnet.

```
RP/0/RSP0/CPU0:R1(config)#telnet ipv4 server max-servers 10
```

And like that, we can telnet.

Ok, onto SSH – but before setting up SSH, we need to generate an RSA key. This is a bit different as you do not do this from config mode.
First up, add your domain-name if you do not have one:

```
RP/0/RSP0/CPU0:R1(config)#domain name fryguy.net
RP/0/RSP0/CPU0:R1(config)#commit
RP/0/RSP0/CPU0:R1#crypto key generate rsa
Sat Apr 21 00:36:07.790 UTC
The name for the keys will be: the_default
  Choose the size of the key modulus in the range of 512 to 2048 for your
General Purpose Keypair. Choosing a key modulus greater than 512 may take a
few minutes.

How many bits in the modulus [1024]: 2048
Generating RSA keys ...
Done w/ crypto generate keypair
[OK]

RP/0/RSP0/CPU0:R1#
```

Once we have generated the RSA key, we can now enable the SSH service:

```
RP/0/RSP0/CPU0:R1#conf t
Sat Apr 21 00:40:33.845 UTC
RP/0/RSP0/CPU0:R1(config)#ssh server v2
RP/0/RSP0/CPU0:R1(config)#commit
Sat Apr 21 00:40:39.939 UTC
```

And like that, SSH services are now enabled.

Ok, but what if we wanted to limit who has access to the box by IP address, that is where control-plane security comes in. For this example, I will allow 10/8 to access the device.

```
RP/0/RSP0/CPU0:R1(config)#control-plane
RP/0/RSP0/CPU0:R1(config-ctrl)#management-plane
RP/0/RSP0/CPU0:R1(config-mpp-inband)#int g0/1/0/18
RP/0/RSP0/CPU0:R1(config-mpp-inband-if)#allow SSH peer
RP/0/RSP0/CPU0:R1(config-ssh-peer)# address ipv4 10.0.0.0/8
```

```
RP/0/RSP0/CPU0:R1(config-ssh-peer)# allow Telnet peer
RP/0/RSP0/CPU0:R1(config-telnet-peer)#address ipv4 10.0.0.0/8
RP/0/RSP0/CPU0:R1(config-telnet-peer)#exit
RP/0/RSP0/CPU0:R1(config-mpp-inband)#comm
Sat Apr 21 01:09:45.163 UTC
```

And now to test, from a device on the 10/8 network:

```
user@host [~]> ssh admin@10.1.1.1
admin@10.1.1.1's password:
RP/0/RSP0/CPU0:R1#
```

There you go, SSH access from only the 10.0.0.0/8 subnet.

And, when it comes close the expiry timer, you will get a message:

```
RP/0/RSP0/CPU0:R1#
*
* The idle timeout is soon to expire on this line
*

Received disconnect from 10.2.2.2: 11:
user@host [~]>
```

12. TACACS Configuration (default and non-default VRF)

Ok, so you want to secure your IOS-XR device using TACACS.
The first example I will use will be using the default VRF for TACACS authorization and the second will be using a different VRF. For these examples, the tacacs server is at IP 192.168.100.100 and the password is TacacsPassword

First up, we need to configure our source interface for TACACS, here we will use loopback0 and the default VRF.

```
RP/0/RSP0/CPU0:PE2(config)#tacacs source-interface Loopback0 vrf default
```

Now we can configure our TACACS server and Password

```
RP/0/RSP0/CPU0:PE2(config)#tacacs-server host 192.168.100.100
RP/0/RSP0/CPU0:PE2(config-tacacs-host)#key 0 TacacsPassword
RP/0/RSP0/CPU0:PE2(config-tacacs-host)#exit
RP/0/RSP0/CPU0:PE2(config)#
```

Time to create a local console authenticaion method, this way console does not rely on TACACS.
You may or may not want to do this, but I am showing it for these examples.

```
RP/0/RSP0/CPU0:PE2(config)#aaa authentication login console local
RP/0/RSP0/CPU0:PE2(config)#aaa authorization commands console none
```

Apply the console loging to the line console

```
RP/0/RSP0/CPU0:PE2(config)#line console
RP/0/RSP0/CPU0:PE2(config-line)#login authentication console
RP/0/RSP0/CPU0:PE2(config-line)#authorization commands console
RP/0/RSP0/CPU0:PE2(config-line)#exit
RP/0/RSP0/CPU0:PE2(config)#
```

Now we can start to configure our AAA for login, here I am using default

```
RP/0/RSP0/CPU0:PE2(config)#aaa authentication login default group tacacs+ local
```

Now for some command authorization, if you want it

```
RP/0/RSP0/CPU0:PE2(config)#aaa authorization commands default group tacacs+
```

And accounting as well.

```
RP/0/RSP0/CPU0:PE2(config)#aaa accounting exec default start-stop group tacacs+
RP/0/RSP0/CPU0:PE2(config)#aaa accounting system default start-stop group tacacs+
RP/0/RSP0/CPU0:PE2(config)#aaa accounting commands default start-stop group tacacs+
```

Since this is IOS XR, I strongly suggest using a commit confirmed here!

```
RP/0/RSP0/CPU0:PE2(config)#commit confirmed minutes 2
Thu Oct 18 03:22:57.487 UTC
RP/0/RSP0/CPU0:PE2(config)#
```

From another terminal, SSH into the box using a TACACs account, and if successful, commit again.

```
RP/0/RSP0/CPU0:PE2(config)#commit
Thu Oct 18 03:23:22.951 UTC

% Confirming commit for trial session.
RP/0/RSP0/CPU0:PE2(config)#
```

That is normal TACACS, now time to add in the challenges of a VRF.

First up, we need to set our source interface, for this one I will use a different Loopback, Lo100 and use VRF CustA

```
RP/0/RSP0/CPU0:PE2(config)#tacacs source-interface Loopback100 vrf CustA
```

Now we can configure our TACACS server

```
RP/0/RSP0/CPU0:PE2(config)#tacacs-server host 192.168.100.100
RP/0/RSP0/CPU0:PE2(config-tacacs-host)#key 0 TacacsPassword
RP/0/RSP0/CPU0:PE2(config-tacacs-host)#exit
RP/0/RSP0/CPU0:PE2(config)#
```

Now we need to create a server group for the ACS box. This tells it what VRF the server is in.

```
RP/0/RSP0/CPU0:PE2(config)#aaa group server tacacs+ ACS
RP/0/RSP0/CPU0:PE2(config-sg-tacacs)# server 192.168.100.100
RP/0/RSP0/CPU0:PE2(config-sg-tacacs)# vrf CustA
```

Now we can configure our local logins for the console:

```
RP/0/RSP0/CPU0:PE2(config)#aaa authentication login console local
RP/0/RSP0/CPU0:PE2(config)#aaa authorization commands console none
RP/0/RSP0/CPU0:PE2(config)#line console
RP/0/RSP0/CPU0:PE2(config-line)# login authentication console
RP/0/RSP0/CPU0:PE2(config-line)# authorization commands console
```

Here I would commit the configs that we have done.

```
RP/0/RSP0/CPU0:PE2(config)#commit
```

And finally configure our AAA for login

```
RP/0/RSP0/CPU0:PE2(config)#aaa authentication login default group ACS local
RP/0/RSP0/CPU0:PE2(config)#aaa authorization commands default group ACS none
RP/0/RSP0/CPU0:PE2(config)#aaa accounting exec default start-stop group ACS
RP/0/RSP0/CPU0:PE2(config)#aaa accounting system default start-stop group ACS
RP/0/RSP0/CPU0:PE2(config)#aaa accounting commands default start-stop group
ACS
RP/0/RSP0/CPU0:PE2(config)#
```

And finally do the commit confirmed here again

```
RP/0/RSP0/CPU0:PE2(config)#commit confirmed minutes 2
```

Test remote access via SSH, and if all works - commit it to save

```
RP/0/RSP0/CPU0:PE2(config)#commit

% Confirming commit for trial session.
```

And we are done!

13.Access Lists

Access lists - these are the same as IOS Extended access lists.

Sorry, not much to say here but you should already be familiar with these.

```
RP/0/RSP0/CPU0:R1(config)#ipv4 access-list RemoteAccess
RP/0/RSP0/CPU0:R1(config-ipv4-acl)#permit tcp 216.167.0.0/24 any eq ssh
RP/0/RSP0/CPU0:R1(config-ipv4-acl)#commit
```

14.OSPF

Time for some OSPF configs, these will build off the previous configs we just did. For this lab, the other router, R2, was preconfigured to support the connections.

We will place our loopback and out g0/3/0/2 interface into OSPF process LAB and area 0.0.0.0

```
RP/0/7/CPU0:R1#
RP/0/7/CPU0:R1#conf t
Thu Mar 29 19:37:52.671 UTC

Define our OSPF process name
RP/0/7/CPU0:R1(config)#router ospf LAB

Now to define our area first
RP/0/7/CPU0:R1(config-ospf)#area 0.0.0.0

Now we can place the interfaces into the area, no need to entering subnets
RP/0/7/CPU0:R1(config-ospf-ar)#inter loo0
RP/0/7/CPU0:R1(config-ospf-ar-if)#inter g0/3/0/2
RP/0/7/CPU0:R1(config-ospf-ar-if)#exit
RP/0/7/CPU0:R1(config-ospf-ar)#exit
RP/0/7/CPU0:R1(config-ospf)#exit
RP/0/7/CPU0:R1(config)#commit
Thu Mar 29 19:38:15.182 UTC
RP/0/7/CPU0:R1(config)#
```

Now to look at our IP Protocols running:

```
RP/0/7/CPU0:R1#sh ip proto
Thu Mar 29 19:38:24.113 UTC

Routing Protocol OSPF LAB
  Router Id: 1.1.1.1
  Distance: 110
  Non-Stop Forwarding: Disabled
  Redistribution:
    None
  Area 0.0.0.0
    Loopback0
    GigabitEthernet0/3/0/2
RP/0/7/CPU0:R1#
```

We can see what we have OSPF LAB running with a RouterID of 1.1.1.1 (our loopback). It tells us what interfaces are in Area 0.0.0.0 as well.

Now to see if we neighbored up with R2:

```
RP/0/7/CPU0:R1#sh ip ospf nei
Thu Mar 29 19:38:33.557 UTC

* Indicates MADJ interface

Neighbors for OSPF LAB

Neighbor ID Pri State    Dead Time  Address     Interface
2.2.2.2     1   FULL/DR  00:00:37   150.1.12.2  GigabitEthernet0/3/0/2
    Neighbor is up for 00:00:12

Total neighbor count: 1
RP/0/7/CPU0:R1#
```

Yup, we have a neighbor of R2 (2.2.2.2) up and in FULL/DR.
Time to look at the routing table:

```
RP/0/7/CPU0:R1#sh ip route
Thu Mar 29 19:41:06.047 UTC

Codes: C - connected, S - static, R - RIP, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - ISIS, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, su - IS-IS summary null, * - candidate default
       U - per-user static route, o - ODR, L - local, G  - DAGR
       A - access/subscriber, (!) - FRR Backup pathc

Gateway of last resort is not set
L    1.1.1.1/32 is directly connected, 00:15:37, Loopback0
O    2.2.2.2/32 [110/2] via 150.1.12.2, 00:02:43, GigabitEthernet0/3/0/2
C    150.1.12.0/24 is directly connected, 01:02:19, GigabitEthernet0/3/0/2
L    150.1.12.1/32 is directly connected, 01:02:19, GigabitEthernet0/3/0/2
RP/0/7/CPU0:R1#
```

We can see we have a route to R2 loopback interface (2.2.2.2/32), now we should be able to PING it from our Loopback0 interface.

```
RP/0/7/CPU0:R1#ping 2.2.2.2 source lo0
Thu Mar 29 19:41:21.828 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
RP/0/7/CPU0:R1#
```

For reference, here is a similar IOS config for the same thing:

```
R1(config)#router ospf 1
R1(config-router)#net 1.1.1.1 0.0.0.0 a 0.0.0.0
R1(config-router)#net 150.1.12.0 0.0.0.255 a 0.0.0.0
R1(config-router)#^Z
R1#
*Mar 29 20:18:29.698: %SYS-5-CONFIG_I: Configured from console by console
R1#
R1#
R1#p 2.2.2.2 so l0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
Packet sent with a source address of 1.1.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms
R1#
```

OSPF Advanced Features

I just wanted to take a minute and discuss some of the other features available for OSPF.

Network Point-to-Point, Point-to-Multipoint, broadcast, non-broadcast

```
RP/0/7/CPU0:R1(config)#router ospf LAB
RP/0/7/CPU0:R1(config-ospf)#area 0.0.0.0
RP/0/7/CPU0:R1(config-ospf-ar)#int g0/3/0/2
RP/0/7/CPU0:R1(config-ospf-ar-if)#network ?
  broadcast           Specify OSPF broadcast multi-access network
  non-broadcast       Specify OSPF NBMA network
  point-to-multipoint Specify OSPF point-to-multipoint network
  point-to-point      Specify OSPF point-to-point network
```

As you can see, all the normal OSPF network interface types are there. You just need to configure them under the OSPF process instead of the interface like in normal IOS.

Authentication

IOS XR also supports OSPF authentication, both area and interface. In this example we will create an MD5 interface authentication.

```
RP/0/7/CPU0:R1(config)#router ospf LAB
RP/0/7/CPU0:R1(config-ospf)#area 0.0.0.0
RP/0/7/CPU0:R1(config-ospf-ar)#int g0/3/0/2
```

Need to enable MD5 authentication

```
RP/0/7/CPU0:R1(config-ospf-ar-if)#authentication message-digest
```

Then set our MD5 key #1 to Cisco

```
RP/0/7/CPU0:R1(config-ospf-ar-if)#message-digest-key 1 md5 Cisco
RP/0/7/CPU0:R1(config-ospf-ar-if)#exit
RP/0/7/CPU0:R1(config-ospf-ar)#commit
```

Now, lets look at the interface and make sure we have MD5 authentication enabled.

```
RP/0/7/CPU0:R1#sh ospf LAB int g0/3/0/2
Sun Apr  1 18:31:01.235 UTC

GigabitEthernet0/3/0/2 is up, line protocol is up
  Internet Address 150.1.12.1/24, Area 0.0.0.0
  Process ID LAB, Router ID 1.1.1.1, Network Type BROADCAST, Cost: 1
  Transmit Delay is 1 sec, State BDR, Priority 1, MTU 1500, MaxPktSz 1500
  Designated Router (ID) 2.2.2.2, Interface address 150.1.12.2
  Backup Designated router (ID) 1.1.1.1, Interface address 150.1.12.1
  Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
    Hello due in 00:00:01
  Index 1/1, flood queue length 0
  Next 0(0)/0(0)
  Last flood scan length is 1, maximum is 1
  Last flood scan time is 0 msec, maximum is 0 msec
  LS Ack List: current length 0, high water mark 3
  Neighbor Count is 1, Adjacent neighbor count is 1
    Adjacent with neighbor 2.2.2.2  (Designated Router)
  Suppress hello for 0 neighbor(s)
  Message digest authentication enabled
    Youngest key id is 1
  Multi-area interface Count is 0
RP/0/7/CPU0:R1#
```

As you can see above, we do. This is all very similar to IOS, so as you can see, the jump to XR is more knowing where to configure something then how to configure something.

Now, lets check our neighbor state

```
RP/0/7/CPU0:R1#sh ospf LAB neighbor
Sat Mar 31 18:37:07.753 UTC

* Indicates MADJ interface

Neighbors for OSPF LAB

Neighbor ID     Pri   State           Dead Time   Address         Interface
2.2.2.2         1     EXSTART/DR      00:00:36    150.1.12.2      
GigabitEthernet0/3/0/2
    Neighbor is up for 00:00:31

Total neighbor count: 1
RP/0/7/CPU0:R1#
```

Then make sure we are getting a route

```
RP/0/7/CPU0:R1#sh route ipv4 ospf
Sat Mar 31 18:37:15.279 UTC

O    2.2.2.2/32 [110/2] via 150.1.12.2, 00:00:06, GigabitEthernet0/3/0/2
RP/0/7/CPU0:R1#
```

And finallying PINGing R2 loopback from ours

```
RP/0/7/CPU0:R1#ping 2.2.2.2 so l0
Sat Mar 31 18:37:19.151 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/5 ms
RP/0/7/CPU0:R1#
```

Cost

Just like normal IOS, we can change the OSPF cost on an interface – but same thing here; it is done under the OSPF process

```
RP/0/7/CPU0:R1#
RP/0/7/CPU0:R1#conf t
Sun Apr  1 18:35:17.061 UTC
RP/0/7/CPU0:R1(config)#router ospf LAB
RP/0/7/CPU0:R1(config-ospf)#area 0.0.0.0
RP/0/7/CPU0:R1(config-ospf-ar)#int loop0
RP/0/7/CPU0:R1(config-ospf-ar-if)#cost ?
  <1-65535>  Cost
```

15.EIGRP

First thing we need to do is delete the OSPF process,that is if you still have it.

```
RP/0/7/CPU0:R1#conf t
Thu Mar 29 20:07:53.797 UTC
RP/0/7/CPU0:R1(config)#no router ospf LAB
RP/0/7/CPU0:R1(config)#commit
```

Once that is deleted, we can now continue with EIGRP configuration.

Just like IOS, we need to give it a process ID

```
RP/0/7/CPU0:R1(config)#router eigrp 1
```

Here is where the difference starts, we need to select the Address family first

```
RP/0/7/CPU0:R1(config-eigrp)#address-family ipv4
```

Enter no auto-summary (this is habitual to be honest)

```
RP/0/7/CPU0:R1(config-eigrp-af)#no auto-summary
```

Then assign the interfaces you want in EIGRP

```
RP/0/7/CPU0:R1(config-eigrp-af)#int l0
RP/0/7/CPU0:R1(config-eigrp-af-if)#int g0/3/0/2
RP/0/7/CPU0:R1(config-eigrp-af-if)#exit
RP/0/7/CPU0:R1(config-eigrp-af)#exit
RP/0/7/CPU0:R1(config-eigrp)#exit
RP/0/7/CPU0:R1(config)#commit
Thu Mar 29 20:08:59.108 UTC
RP/0/7/CPU0:R1(config)#exit
```

Now lets look at our IP Protocols:

```
RP/0/7/CPU0:R1#sh ip protocols
Thu Mar 29 20:09:25.058 UTC

Routing Protocol: EIGRP, instance 1
 Default context AS: 1, Router ID: 1.1.1.1
  Address Family: IPv4
   Default networks not flagged in outgoing updates
   Default networks not accepted from incoming updates
   Distance: internal 90, external 170
   Maximum paths: 4
   EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
   EIGRP maximum hopcount 100
   EIGRP maximum metric variance 1
   EIGRP NSF: enabled
```

```
    NSF-aware route hold timer is 480s
    NSF signal timer is 20s
    NSF converge timer is 300s
    Time since last restart is 00:00:25
   SIA Active timer is 180s
   Interfaces:
    Loopback0
    GigabitEthernet0/3/0/2
```

When you issue the same command under IOS, you have Routing for Networks instead of Interfaces:

```
R1#sh ip protocols
Routing Protocol is "eigrp 1"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Default networks flagged in outgoing updates
  Default networks accepted from incoming updates
  EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
  EIGRP maximum hopcount 100
  EIGRP maximum metric variance 1
  Redistributing: eigrp 1
  EIGRP NSF-aware route hold timer is 240s
  Automatic network summarization is not in effect
  Maximum path: 4
  Routing for Networks:
    1.1.1.1/32
    150.1.12.0/24
  Routing Information Sources:
    Gateway         Distance      Last Update
    (this router)         90      00:00:22
    150.1.12.2            90      00:00:04
  Distance: internal 90 external 170
R1#
```

Now, let’s look at our routing table on IOS XR.

```
RP/0/7/CPU0:R1#sh ip route
Thu Mar 29 20:09:31.763 UTC

Codes: C - connected, S - static, R - RIP, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - ISIS, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, su - IS-IS summary null, * - candidate default
       U - per-user static route, o - ODR, L - local, G  - DAGR
       A - access/subscriber, (!) - FRR Backup path

Gateway of last resort is not set

L    1.1.1.1/32 is directly connected, 00:44:02, Loopback0
```

```
D    2.2.2.2/32 [90/130816] via 150.1.12.2, 00:00:11, GigabitEthernet0/3/0/2
C    150.1.12.0/24 is directly connected, 01:30:45, GigabitEthernet0/3/0/2
L    150.1.12.1/32 is directly connected, 01:30:45, GigabitEthernet0/3/0/2
RP/0/7/CPU0:R1#
```

There, we have a route to R2's loopback. Lets PING it from our loopback to test connectivity.

```
RP/0/7/CPU0:R1#ping 2.2.2.2 so l0
Thu Mar 29 20:09:36.232 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/2/5 ms
RP/0/7/CPU0:R1#
```

And like that basic EIGRP is done.

Now, lets add IPv6 to the EIGRP process.

```
RP/0/7/CPU0:R1#conf t
Thu Mar 29 20:27:16.966 UTC
RP/0/7/CPU0:R1(config)#router eigrp 1
RP/0/7/CPU0:R1(config-eigrp)#address-family ipv6
RP/0/7/CPU0:R1(config-eigrp-af)#int l0
RP/0/7/CPU0:R1(config-eigrp-af-if)#int g0/3/0/2
RP/0/7/CPU0:R1(config-eigrp-af-if)#commit
Thu Mar 29 20:27:28.352 UTC
RP/0/7/CPU0:R1(config-eigrp-af-if)#
```

I will be honest here; the correct command to show routes is *show route Protocol*. Once you add IPv6, you really should to start to use the correct commands. ☺

```
RP/0/7/CPU0:R1#sh route ipv6
Thu Mar 29 20:29:31.952 UTC

Codes: C - connected, S - static, R - RIP, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - ISIS, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, su - IS-IS summary null, * - candidate default
       U - per-user static route, o - ODR, L - local, G  - DAGR
       A - access/subscriber, (!) - FRR Backup path

Gateway of last resort is not set

L    2001::1/128 is directly connected,
      00:58:42, Loopback0
D    2001::2/128
```

```
      [90/130816] via fe80::2d0:79ff:fe01:3a78, 00:01:43,
GigabitEthernet0/3/0/2
C    2001:1:1:12::/64 is directly connected,
      01:02:52, GigabitEthernet0/3/0/2
L    2001:1:1:12::1/128 is directly connected,
      01:02:52, GigabitEthernet0/3/0/2
RP/0/7/CPU0:R1#
```

Cool – we have an IPv6 route to R2 loopback (2001::2/128)
 Lets ping that interface   from our loopback interface

```
RP/0/7/CPU0:R1#ping 2001::2 sou 2001::1
Thu Mar 29 20:31:56.602 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001::2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/4/10 ms
RP/0/7/CPU0:R1#
```

There you go; we have IPv4 and IPv6 connectivity now.

16. RIP (yeah yeah, why would you want to run this? Think – PE-CE)

Ok, time for the next routing protocol – RIP. Why would you use IOS XR for RIP? Well, if you have a CE device that only has a few networks, RIP is a perfect protocol. Keep in mind that IOS XR is code built for a Service Provider network, so PE-CE relationships are what these routers are about.

Plus – it is just good to know different options.

```
RP/0/7/CPU0:R1#conf t
Thu Mar 29 20:37:44.801 UTC
RP/0/7/CPU0:R1(config)#router rip
RP/0/7/CPU0:R1(config-rip)#int l0
RP/0/7/CPU0:R1(config-rip-if)#int g0/3/0/2
RP/0/7/CPU0:R1(config-rip-if)#^Z
Uncommitted changes found, commit them before exiting(yes/no/cancel)?
[cancel]:yes
```

Notice I did not do a COMMIT, but since the router knows I was making changes it asked me.

```
RP/0/7/CPU0:R1#
```

Let's check our IP protocols:

```
RP/0/7/CPU0:R1#sh ip proto
Thu Mar 29 20:39:20.919 UTC

Routing Protocol RIP
  1 VRFs (including default) configured, 1 active
  6 routes, 3 paths have been allocated
  Current OOM state is "Normal"
  UDP socket descriptor is 42
    VRF              Active    If-config If-active Routes     Paths
Updates
    default          Active    2         2         6          3
30s
```

Now lets look at the RIP process:

```
RP/0/7/CPU0:R1#sh rip
Thu Mar 29 20:39:24.892 UTC

RIP config:
Active:                    Yes
Added to socket:           Yes
Out-of-memory state:        Normal
```

```
Version:                     2
Default metric:              Not set
Maximum paths:               4
Auto summarize:             No
Broadcast for V2:           No
Packet source validation:   Yes
NSF:                         Disabled
Timers: Update:              30 seconds (0 seconds until next update)
        Invalid:             180 seconds
        Holddown:            180 seconds
        Flush:               240 seconds
RP/0/7/CPU0:R1#
```

Now here is something interesting, the RIP version is 2, yet I did not specify it in the config. This is because IOS XR code only supports v2.

```
RP/0/7/CPU0:R1(config)#router rip
RP/0/7/CPU0:R1(config-rip)#ver?
                                                      ^
% Invalid input detected at '^' marker.
```

Now, let's look at the routing table using the proper command:

```
RP/0/7/CPU0:R1#sh route ipv4
Thu Mar 29 20:40:08.877 UTC
Codes: C - connected, S - static, R - RIP, B – BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E – EGP
       i - ISIS, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, su - IS-IS summary null, * - candidate default
       U - per-user static route, o - ODR, L - local, G  - DAGR
       A - access/subscriber, (!) - FRR Backup path
Gateway of last resort is not set
L    1.1.1.1/32 is directly connected, 01:14:39, Loopback0
R    2.2.2.2/32 [120/1] via 150.1.12.2, 00:01:25, GigabitEthernet0/3/0/2
C    150.1.12.0/24 is directly connected, 02:01:22, GigabitEthernet0/3/0/2
L    150.1.12.1/32 is directly connected, 02:01:22, GigabitEthernet0/3/0/2

RP/0/7/CPU0:R1#
```

As you can see, we have a RIP route to R2 L0 2.2.2.2/32 interface. Time for a PING!

```
RP/0/7/CPU0:R1#ping 2.2.2.2 so l0
Thu Mar 29 20:40:15.606 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/3/4 ms
RP/0/7/CPU0:R1#
```

And like that RIP is configured.

17.IS-IS

Time for some IS-IS routing! Between IS-IS and OSPF, those are the two most coming SP core routing protocols.

```
RP/0/7/CPU0:R1#conf t
Thu Mar 29 22:09:12.786 UTC
```

First we need to name our process

```
RP/0/7/CPU0:R1(config)#router isis LAB
```

Then configure our Network Entity (Area)

```
RP/0/7/CPU0:R1(config-isis)#net 49.0000.0000.0001.00
```

Then we assign the interfaces to the process, as well as the address family.

```
RP/0/7/CPU0:R1(config-isis)#int l0
RP/0/7/CPU0:R1(config-isis-if)#address-family ipv4
RP/0/7/CPU0:R1(config-isis-if-af)#exit
RP/0/7/CPU0:R1(config-isis-if)#address-family ipv6
RP/0/7/CPU0:R1(config-isis-if-af)# exit
RP/0/7/CPU0:R1(config-isis-if)#int g0/3/0/2
RP/0/7/CPU0:R1(config-isis-if)#address-family ipv4
RP/0/7/CPU0:R1(config-isis-if-af)# exit
RP/0/7/CPU0:R1(config-isis-if)#address-family ipv6
RP/0/7/CPU0:R1(config-isis-if-af)# exit
RP/0/7/CPU0:R1(config-isis-if)#exit
```

Notice I did not specify an IS-IS Level when I started, but we can set this to Level-2

```
RP/0/7/CPU0:R1(config-isis)#is-type level-2-only
```

Now, when we show the config, you will notice Level-2 is set to the top of the config when applied, not in the order I entered it.   This is the beauty of a staging config, you can enter some things in the wrong order but they will be applied in the correct order.

```
RP/0/7/CPU0:R1(config-isis)#sh config
Thu Mar 29 22:10:22.326 UTC
Building configuration...
!! IOS XR Configuration 4.1.1
router isis LAB
 is-type level-2-only
 net 49.0000.0000.0001.00
 interface Loopback0
  address-family ipv4 unicast
  !
  address-family ipv6 unicast
  !
 !
```

```
 interface GigabitEthernet0/3/0/2
  address-family ipv4 unicast
  !
  address-family ipv6 unicast
  !
 !
!
end
```

Now, let us commit our changes.

```
RP/0/7/CPU0:R1(config-isis)#commit
RP/0/7/CPU0:R1(config-isis)#exit
RP/0/7/CPU0:R1(config)#exit
RP/0/7/CPU0:R1#
```

Time to check our IS-IS adjancies.

```
RP/0/7/CPU0:R1#sh isis adjacency
Thu Mar 29 22:16:21.989 UTC

IS-IS LAB Level-2 adjacencies:
System Id      Interface        SNPA           State Hold Changed  NSF IPv4 IPv6
                                                                       BFD  BFD
GSR-R2         Gi0/3/0/2        00d0.7901.3a78 Up    9    00:05:52 Yes None None

Total adjacency count: 1
RP/0/7/CPU0:R1#
```

We can see we are adjacent with R2 via IPv4 and IPv6. Lets look at the IPv4 IS-IS routing table and then PING the loopback of R2:

```
RP/0/7/CPU0:R1#sh route ipv4 isis
Thu Mar 29 22:17:15.545 UTC

i L2 2.2.2.2/32 [115/20] via 150.1.12.2, 00:06:36, GigabitEthernet0/3/0/2
RP/0/7/CPU0:R1#
RP/0/7/CPU0:R1#ping 2.2.2.2 so l0
Thu Mar 29 22:17:37.226 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
RP/0/7/CPU0:R1#
```

Ok, that worked – now we can do the same for IPv6. First we should look at the IPv6 IS-IS routes and then ping the loopback of R2.

```
RP/0/7/CPU0:R1#sh route ipv6 isis
Thu Mar 29 22:17:43.918 UTC

i L2 2001::2/128
      [115/20] via fe80::2d0:79ff:fe01:3a78, 00:07:05, GigabitEthernet0/3/0/2
RP/0/7/CPU0:R1#ping 2001::2 so 2001::1
Thu Mar 29 22:17:49.763 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001::2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/9/34 ms
RP/0/7/CPU0:R1#
```

Compare that to IOS ISIS config:
Configure the process and set the level and NET

```
R1(config)#router isis
R1(config-router)#net 49.0000.0000.0001.00
R1(config-router)#is-type level-2
```

Then change context to the G0/1 interface and enable ISIS for IPv4 and IPv6

```
R1(config-router)#int g0/1
R1(config-if)#ip router isis
R1(config-if)#ipv6 router isis
```

Then change context to the Loop0 interface and enable ISIS for IPv4 and IPv6

```
R1(config-if)#int l0
R1(config-if)#ip router isis
R1(config-if)#ipv6 router isis
R1(config-if)#^Z
```

Few more steps, and configuring things under the process make much more sense than under an interface.

18.BGP iBGP and eBGP

BGP, this is where it starts to get different with IOS XR.
First up, configuring an iBGP peering with R2's 150.1.12.2 in AS1 and advertise our loopback interface.

```
RP/0/7/CPU0:R1(config)#
RP/0/7/CPU0:R1(config)#router bgp 1
```

Let's define the network we want to advertise, under the address family:

```
RP/0/7/CPU0:R1(config-bgp)#address-family ipv4 unicast
RP/0/7/CPU0:R1(config-bgp-af)#net 1.1.1.1/32
RP/0/7/CPU0:R1(config-bgp-af)#exit
```

Now, we can configure the neighbor. Notice all the commands for the neighbor are under the neighbor now – not next to the neighbor.

```
RP/0/7/CPU0:R1(config-bgp)#nei 150.1.12.2
RP/0/7/CPU0:R1(config-bgp-nbr)#remote-as 1
RP/0/7/CPU0:R1(config-bgp-nbr)#address-family ipv4 unicast
RP/0/7/CPU0:R1(config-bgp-nbr-af)#exit
RP/0/7/CPU0:R1(config-bgp-nbr)#comm
Thu Mar 29 22:47:05.147 UTC

RP/0/7/CPU0:R1(config-bgp)#exit
RP/0/7/CPU0:R1(config)#exit
```

Now, time to see if we have a neighbor established:

```
RP/0/7/CPU0:R1#sh bgp nei 150.1.12.2
Thu Mar 29 22:48:13.338 UTC

BGP neighbor is 150.1.12.2
 Remote AS 1, local AS 1, internal link
 Remote router ID 2.2.2.2
  BGP state = Established, up for 00:00:24
  Last read 00:00:24, Last read before reset 00:00:00
  Hold time is 180, keepalive interval is 60 seconds
  Configured hold time: 180, keepalive: 60, min acceptable hold time: 3
  Last write 00:00:24, attempted 19, written 19
  Second last write 00:00:24, attempted 53, written 53
  Last write before reset 00:00:00, attempted 0, written 0
  Second last write before reset 00:00:00, attempted 0, written 0
  Last write pulse rcvd  Mar 29 22:47:49.296 last full not set pulse count 4
  Last write pulse rcvd before reset 00:00:00
  Socket not armed for io, armed for read, armed for write
  Last write thread event before reset 00:00:00, second last 00:00:00
  Last KA expiry before reset 00:00:00, second last 00:00:00
  Last KA error before reset 00:00:00, KA not sent 00:00:00
  Last KA start before reset 00:00:00, second last 00:00:00
```

```
  Precedence: internet
  Neighbor capabilities:
    Route refresh: advertised and received
    4-byte AS: advertised and received
    Address family IPv4 Unicast: advertised and received
  Received 2 messages, 0 notifications, 0 in queue
  Sent 2 messages, 0 notifications, 0 in queue
  Minimum time between advertisement runs is 0 secs

 For Address Family: IPv4 Unicast
  BGP neighbor version 0
  Update group: 0.2 Filter-group: 0.1  No Refresh request being processed
  Route refresh request: received 0, sent 0
  0 accepted prefixes, 0 are bestpaths
  Cumulative no. of prefixes denied: 0.
  Prefix advertised 0, suppressed 0, withdrawn 0
  Maximum prefixes allowed 524288
  Threshold for warning message 75%, restart interval 0 min
  AIGP is enabled
  An EoR was not received during read-only mode
  Last ack version 1, Last synced ack version 0
  Outstanding version objects: current 0, max 0
  Additional-paths operation: None

  Connections established 1; dropped 0
  Local host: 150.1.12.1, Local port: 33432
  Foreign host: 150.1.12.2, Foreign port: 179
  Last reset 00:00:00
```

Cool, neighbor is up and active.
Now, time to check our BGP summary to see what routes we have.

```
RP/0/7/CPU0:R1#sh ip bgp
Thu Mar 29 22:48:51.876 UTC
BGP router identifier 1.1.1.1, local AS number 1
BGP generic scan interval 60 secs
BGP table state: Active
Table ID: 0xe0000000   RD version: 4
BGP main routing table version 4
BGP scan interval 60 secs

Status codes: s suppressed, d damped, h history, * valid, > best
              i - internal, r RIB-failure, S stale
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network            Next Hop            Metric LocPrf Weight Path
*> 1.1.1.1/32         0.0.0.0                  0         32768 i
*>i2.2.2.2/32         150.1.12.2               0    100      0 i

Processed 2 prefixes, 2 paths
RP/0/7/CPU0:R1#
```

Cool, we have a route to R2 Loopback interface. Lets PING it!

```
RP/0/7/CPU0:R1#ping 2.2.2.2 so l0
Thu Mar 29 22:52:17.899 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/3 ms
RP/0/7/CPU0:R1#
```

We have connectivity!

Here is the IOS XR Config:

```
RP/0/7/CPU0:R1#sh run | begin bgp
Thu Mar 29 22:56:17.937 UTC
Building configuration...
router bgp 1
 address-family ipv4 unicast
  network 1.1.1.1/32
 !
 neighbor 150.1.12.2
  remote-as 1
  address-family ipv4 unicast
  !
 !
!
end

RP/0/7/CPU0:R1#
```

Here is the same IOS config. With a single neighbor it is pretty simple.

```
router bgp 1
network 1.1.1.1 mask 255.255.255.255
 neighbor 150.1.12.2 remote-as 1
```

Now for eBGP – here is where it starts gets interesting!

First we need to configure an IGP so that we can establish Loopback connectivity – for this we will use ISIS:

```
RP/0/7/CPU0:R1(config)#router ISIS LAB
RP/0/7/CPU0:R1(config-isis)#net 49.0000.0000.0001.00
RP/0/7/CPU0:R1(config-isis)#interface l0
RP/0/7/CPU0:R1(config-isis-if)#address-family ipv4
RP/0/7/CPU0:R1(config-isis-if)#exit
RP/0/7/CPU0:R1(config-isis)#interface g0/3/0/2
RP/0/7/CPU0:R1(config-isis-if)#address-family ipv4
RP/0/7/CPU0:R1(config-isis-if)#exit
RP/0/7/CPU0:R1(config-isis)#is-type level-2
RP/0/7/CPU0:R1(config-isis)#commit
RP/0/7/CPU0:R1(config-isis-if-af)#exit
RP/0/7/CPU0:R1(config-isis-if)#exit
```

```
RP/0/7/CPU0:R1(config-isis)#exit
```

Now we need to configure an interface to advertise via BGP – here we will create Loop100 with an IP of 100.100.100.100/32

```
RP/0/7/CPU0:R1(config)#int loop100
RP/0/7/CPU0:R1(config-if)#ip add 100.100.100.100/32
RP/0/7/CPU0:R1(config-if)#comm
Thu Mar 29 23:12:31.681 UTC
RP/0/7/CPU0:R1(config-if)#exit
```

Now to configured eBGP.
We will peer with R2 loopback's (2.2.2.2) and their remote AS of 2.
First we define our BGP processed (ID 1)

```
RP/0/7/CPU0:R1(config)#
RP/0/7/CPU0:R1(config-if)#router bgp 1
```

Define the networks we want to advertise

```
RP/0/7/CPU0:R1(config-bgp)#address-family ipv4 unicast
RP/0/7/CPU0:R1(config-bgp-af)#net 100.100.100.100/32
RP/0/7/CPU0:R1(config-bgp-af)#exit
```

Now we can configure our neighbor

```
RP/0/7/CPU0:R1(config-bgp)#nei 2.2.2.2
RP/0/7/CPU0:R1(config-bgp-nbr)#remote-as 2
RP/0/7/CPU0:R1(config-bgp-nbr)#ebgp-multihop
RP/0/7/CPU0:R1(config-bgp-nbr)#up loopback 0
RP/0/7/CPU0:R1(config-bgp-nbr)#address-family ipv4 un
```

And finally commit our changes.

```
RP/0/7/CPU0:R1(config-bgp-nbr-af)#comm
Thu Mar 29 23:18:06.455 UTC
RP/0/7/CPU0:R1(config-bgp-nbr-af)#exit
RP/0/7/CPU0:R1(config-bgp-nbr)#exit
RP/0/7/CPU0:R1(config-bgp)#exit
RP/0/7/CPU0:R1(config)#exit
RP/0/7/CPU0:R1#
```

Ok, now that we have that configured – time to look at our routing table, we should see a route to 200.200.200.200/32.

```
RP/0/7/CPU0:R1#sh ip route
Thu Mar 29 23:24:25.533 UTC

Codes: C - connected, S - static, R - RIP, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - ISIS, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, su - IS-IS summary null, * - candidate default
       U - per-user static route, o - ODR, L - local, G  - DAGR
       A - access/subscriber, (!) - FRR Backup path
```

```
Gateway of last resort is not set

L    1.1.1.1/32 is directly connected, 02:26:47, Loopback0
i L2 2.2.2.2/32 [115/20] via 150.1.12.2, 00:13:05, GigabitEthernet0/3/0/2
L    100.100.100.100/32 is directly connected, 00:11:53, Loopback100
C    150.1.12.0/24 is directly connected, 02:27:12, GigabitEthernet0/3/0/2
L    150.1.12.1/32 is directly connected, 02:26:47, GigabitEthernet0/3/0/2
RP/0/7/CPU0:R1#
```

Hmm, no route – why is that? Is the neighbor up?
Lets check:

```
RP/0/7/CPU0:R1#sh ip bgp summ
Thu Mar 29 23:25:12.041 UTC
BGP router identifier 1.1.1.1, local AS number 1
BGP generic scan interval 60 secs
BGP table state: Active
Table ID: 0xe0000000   RD version: 7
BGP main routing table version 7
BGP scan interval 60 secs

BGP is operating in STANDALONE mode.


Process       RcvTblVer   bRIB/RIB   LabelVer  ImportVer  SendTblVer  StandbyVer
Speaker               7          7          7          7           7           7

Some configured eBGP neighbors (under default or non-default vrfs)
do not have both inbound and outbound policies configured for IPv4 Unicast
address family. These neighbors will default to sending and/or
receiving no routes and are marked with '!' in the output below.
Use the 'show bgp neighbor <nbr_address>' command for details.

Neighbor        Spk    AS MsgRcvd MsgSent   TblVer  InQ OutQ  Up/Down  St/PfxRcd
2.2.2.2           0     2       7       6        7    0    0 00:03:09          0!

RP/0/7/CPU0:R1#
```

Yup, we are up for over 3 minutes now – but wait, we have an ! mark there – no routes received.
It says to use the *show bgp neighbors address* for details. Let’s see what that says.

```
RP/0/7/CPU0:R1#sh bgp neighbors 2.2.2.2
Thu Mar 29 23:26:12.572 UTC

BGP neighbor is 2.2.2.2
 Remote AS 2, local AS 1, external link
 Remote router ID 2.2.2.2
  BGP state = Established, up for 00:04:10
  Last read 00:00:05, Last read before reset 00:00:00
```

```
  Hold time is 180, keepalive interval is 60 seconds
  Configured hold time: 180, keepalive: 60, min acceptable hold time: 3
  Last write 00:00:05, attempted 19, written 19
  Second last write 00:01:05, attempted 19, written 19
  Last write before reset 00:00:00, attempted 0, written 0
  Second last write before reset 00:00:00, attempted 0, written 0
  Last write pulse rcvd  Mar 29 23:26:07.793 last full not set pulse count 14
  Last write pulse rcvd before reset 00:00:00
  Socket not armed for io, armed for read, armed for write
  Last write thread event before reset 00:00:00, second last 00:00:00
  Last KA expiry before reset 00:00:00, second last 00:00:00
  Last KA error before reset 00:00:00, KA not sent 00:00:00
  Last KA start before reset 00:00:00, second last 00:00:00
  Precedence: internet
  Enforcing first AS is enabled
  Neighbor capabilities:
    Route refresh: advertised and received
    4-byte AS: advertised and received
    Address family IPv4 Unicast: advertised and received
  Received 8 messages, 0 notifications, 0 in queue
  Sent 7 messages, 0 notifications, 0 in queue
  Minimum time between advertisement runs is 30 secs

 For Address Family: IPv4 Unicast
  BGP neighbor version 7
  Update group: 0.2 Filter-group: 0.1  No Refresh request being processed
  eBGP neighbor with no inbound or outbound policy; defaults to 'drop'
  Route refresh request: received 0, sent 0
  0 accepted prefixes, 0 are bestpaths
  Cumulative no. of prefixes denied: 1.
    No policy: 1, Failed RT match: 0
    By ORF policy: 0, By policy: 0
  Prefix advertised 0, suppressed 0, withdrawn 0
  Maximum prefixes allowed 524288
  Threshold for warning message 75%, restart interval 0 min
  An EoR was not received during read-only mode
  Last ack version 7, Last synced ack version 0
  Outstanding version objects: current 0, max 0
  Additional-paths operation: None

  Connections established 1; dropped 0
  Local host: 1.1.1.1, Local port: 58277
  Foreign host: 2.2.2.2, Foreign port: 179
  Last reset 00:00:00
  External BGP neighbor may be up to 255 hops away.
RP/0/7/CPU0:R1#
```

Ahh, the neighbor is up but there is a line that says:

eBGP neighbor with no inbound or outbound policy; defaults to 'drop'

Here is the first difference with IOS XR – eBGP peers must have a Route-Policy (route-map) configured to permit routes in and out of them.

Instead of a route-map like IOS, IOS XR uses a Route Policy Language (RPL) – that is more powerful and easier than IOS. Let's configure a very simple one to pass everything:

```
RP/0/7/CPU0:R1(config)#route-policy PASS
RP/0/7/CPU0:R1(config-rpl)#pass
RP/0/7/CPU0:R1(config-rpl)#exit
RP/0/7/CPU0:R1(config)#commit
Thu Mar 29 23:28:08.400 UTC
```

Cool – that was easy. Now lets apply that to the eBGP neighbor:

```
RP/0/7/CPU0:R1(config)#router bgp 1
RP/0/7/CPU0:R1(config-bgp)#nei 2.2.2.2
RP/0/7/CPU0:R1(config-bgp-nbr)#address-family ipv4 unicast
RP/0/7/CPU0:R1(config-bgp-nbr-af)#route-policy PASS out
RP/0/7/CPU0:R1(config-bgp-nbr-af)#route-policy PASS in
RP/0/7/CPU0:R1(config-bgp-nbr-af)#commit
Thu Mar 29 23:28:32.865 UTC
```

Now, lets look at the routing table for BGP

```
RP/0/7/CPU0:R1#sh route ipv4 bgp
Thu Mar 29 23:29:43.865 UTC

B    200.200.200.200/32 [20/0] via 2.2.2.2, 00:01:06
RP/0/7/CPU0:R1#
```

Cool, we have a route to R2's Loopback100 interface. PING time!

```
RP/0/7/CPU0:R1#ping 200.200.200.200 sou loop100
Thu Mar 29 23:30:10.013 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.200.200.200, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/3/5 ms
RP/0/7/CPU0:R1#
```

Look at that, we have connectivity!

A similar IOS config would look like this:

```
router bgp 1
 no synchronization
 bgp log-neighbor-changes
 network 100.100.100.100 mask 255.255.255.255
 neighbor 2.2.2.2 remote-as 2
 neighbor 2.2.2.2 ebgp-multihop 255
 neighbor 2.2.2.2 route-map PASS in
 neighbor 2.2.2.2 route-map PASS out
```

```
ip prefix-list PASS seq 5 permit 0.0.0.0/0 le 32

route-map PASS permit 10
 match ip address prefix-list PASS
```

19.Route Filtering

Ok, now that BGP has been covered, lets talk about filtering routes received from our neighbor. Here I have created some additional Loopbacks on R2 that are being advertised to R1:

```
RP/0/7/CPU0:R1#sh ip route bgp
Fri Mar 30 13:13:36.797 UTC

B    200.100.200.100/32 [20/0] via 2.2.2.2, 00:00:42
B    200.200.200.200/32 [20/0] via 2.2.2.2, 13:45:00
B    200.200.200.203/32 [20/0] via 2.2.2.2, 00:00:42
B    200.200.200.204/32 [20/0] via 2.2.2.2, 00:00:42
B    200.200.200.205/32 [20/0] via 2.2.2.2, 00:00:42
B    200.200.200.206/32 [20/0] via 2.2.2.2, 00:00:42
B    200.200.200.207/32 [20/0] via 2.2.2.2, 00:00:42
B    200.200.200.208/32 [20/0] via 2.2.2.2, 00:00:42
B    200.200.200.209/32 [20/0] via 2.2.2.2, 00:00:42
B    200.200.200.210/32 [20/0] via 2.2.2.2, 00:00:42
RP/0/7/CPU0:R1#
```

As you can see, we are getting a bunch of 200.200.200.x/32 routes now as well as a 200.100.200.100/32 route. For this exercise, lets filter our all the 200.200.200.x routes we are receiving from our neighbor.

Ok, lets create a prefix-set for the loopback we want to permit:

```
RP/0/7/CPU0:R1(config)#conf t
RP/0/7/CPU0:R1(config)#prefix-set R2Loopbacks
```

In IOS XR you can add comments via the #

```
RP/0/7/CPU0:R1(config-pfx)## These are the R2 Loopbacks that we will allow
RP/0/7/CPU0:R1(config-pfx)#200.100.200.100/32
RP/0/7/CPU0:R1(config-pfx)#end-set
```

Now that we have the prefix-set done we can create the route-policy

```
RP/0/7/CPU0:R1(config)#route-policy R2Loopbacks
```

Notice that IOS XR can use IF statements, you can just imagine how powerful IF and ELSE statements can be when route filtering.

```
RP/0/7/CPU0:R1(config-rpl)#if destination in R2Loopbacks then
RP/0/7/CPU0:R1(config-rpl-if)#pass
RP/0/7/CPU0:R1(config-rpl-if)#endif
RP/0/7/CPU0:R1(config-rpl)#end-policy
```

After we end the policy, we need to commit it

```
RP/0/7/CPU0:R1(config)#commit
```

Now that we have the policy committed with no errors, we can apply it to the neighbor. We could have waited to commit, but I chose to commit there to make sure all was OK.

```
RP/0/7/CPU0:R1(config)#router bgp 1
RP/0/7/CPU0:R1(config-bgp)#neighbor 2.2.2.2
RP/0/7/CPU0:R1(config-bgp-nbr)#address-family ipv4 un
RP/0/7/CPU0:R1(config-bgp-nbr-af)#route-policy R2Loopbacks in
RP/0/7/CPU0:R1(config-bgp-nbr-af)#exit
RP/0/7/CPU0:R1(config-bgp-nbr)#exit
RP/0/7/CPU0:R1(config-bgp)#exi
RP/0/7/CPU0:R1(config)#commit
Fri Mar 30 13:27:01.945 UTC
RP/0/7/CPU0:R1(config)#
```

Now, lets look at our BGP routing table:

```
RP/0/7/CPU0:R1#sh ip route bgp
Fri Mar 30 13:27:22.601 UTC

B    200.100.200.100/32 [20/0] via 2.2.2.2, 00:14:28
RP/0/7/CPU0:R1#
```

There we go, only getting the 200.100.200.100/32 from R2 now.

In IOS this would have looked like:

```
R1(config)#ip prefix-list R2Loopbacks permit 200.100.200.100/32
R1(config)#route-map R2Loopbacks
R1(config-route-map)#match ip add prefix-list R2Loopbacks
R1(config-route-map)#exit
R1(config)#router bgp 1
R1(config-router)#nei 2.2.2.2 route-map R2Loopbacks in
R1(config-router)#^Z
R1#sh ip route b
*Mar 30 14:08:53.048: %SYS-5-CONFIG_I: Configured from console by console
(After a few minutes waiting for BGP)
R1#sh ip route bgp
     200.100.200.0/32 is subnetted, 1 subnets
B       200.100.200.100 [20/0] via 2.2.2.2, 00:00:20
R1#
```

While that might not be so bad, the power of RPL grows. This is just a quick intro; future posts will have more and more about RPL. Some other things that we might see are:

```
route-policy check ASPath
   if as-path passes-through '65500' then
    drop
   else
    pass
  endif
end-policy
```

20.VRF lite and Dot1Q Trunks

Ok, time for some VRF lite basics and we can throw in some Dot1Q trunks to go with it.

First, let's create our VRF called LAB
```
RP/0/7/CPU0:R1(config)#vrf LAB
```

Now we need to enable the address family for this VRF, there IPv4 Unicast
```
RP/0/7/CPU0:R1(config-vrf)#address-family ipv4 un
RP/0/7/CPU0:R1(config-vrf-af)#exit
```

Now we need to enable the IPv6 address family for this VRF
```
RP/0/7/CPU0:R1(config-vrf)#address-family ipv6 unicast
```

Now we can create our Dot1Q trunk to the other router:
```
RP/0/7/CPU0:R1(config-vrf-af)#int g0/3/0/3.100
```

Little different then IOS, but this actually makes more sense
```
RP/0/7/CPU0:R1(config-subif)#dot1q vlan 100
RP/0/7/CPU0:R1(config-subif)#ip add 150.1.21.1/24
RP/0/7/CPU0:R1(config-subif)#ipv6 add 2001:1:1:21::1/64
RP/0/7/CPU0:R1(config-subif)#vrf LAB
```

Notice that I applied the VRF LAB command after configuring the IP addresses. If this was IOS, I would have lost all that work – but since its IOS XR, nothing takes effect until after you COMMIT the changes. ☺

Lets look at what will be applied and then commit it.
```
RP/0/7/CPU0:R1(config-subif)#show config
Fri Mar 30 14:12:06.649 UTC
Building configuration...
!! IOS XR Configuration 4.1.1
vrf LAB
 address-family ipv4 unicast
 !
 address-family ipv6 unicast
 !
!
interface GigabitEthernet0/3/0/3.100
 vrf LAB
 ipv4 address 150.1.21.1 255.255.255.0
 ipv6 address 2001:1:1:21::1/64
 dot1q vlan 100
!
end
```

```
RP/0/7/CPU0:R1(config-subif)#comm
Fri Mar 30 14:12:12.700 UTC
RP/0/7/CPU0:R1(config-subif)#
RP/0/7/CPU0:R1#
```

Now we should try to PING over the VRF. Remember, when PINGing over a VRF you need to specify the VRF in the PING command.

```
RP/0/7/CPU0:R1#ping vrf LAB 2001:1:1:21::2
Fri Mar 30 14:17:37.291 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001:1:1:21::2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/13/49 ms

RP/0/7/CPU0:R1#ping vrf LAB 150.1.21.2
Fri Mar 30 14:17:40.858 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.21.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/3/5 ms
RP/0/7/CPU0:R1#
```

Full connectivity, there we go!
Ok, now we can toss a routing protocol over the link – say OSPF PID 100
First though, create a loopback we can advertise over that VRF,

```
RP/0/7/CPU0:R1#conf t
Fri Mar 30 14:39:31.441 UTC
RP/0/7/CPU0:R1(config)#int loop1000
RP/0/7/CPU0:R1(config-if)#ip add 111.111.111.111/32
RP/0/7/CPU0:R1(config-if)#vrf LAB
```

Now we can configure OSPF
First, define the process identifier, here 100

```
RP/0/7/CPU0:R1(config-if)#router ospf 100
```

Now to change context to VRF LAB

```
RP/0/7/CPU0:R1(config-ospf)#vrf LAB
```

Configure our Router-ID

```
RP/0/7/CPU0:R1(config-ospf-vrf)#router-id 111.111.111.111
RP/0/7/CPU0:R1(config-ospf-vrf)#area 0.0.0.0
```

And then assign the interfaces to the area

```
RP/0/7/CPU0:R1(config-ospf-vrf-ar)#int loop1000
RP/0/7/CPU0:R1(config-ospf-vrf-ar)#int g0/3/0/3.100
RP/0/7/CPU0:R1(config-ospf-vrf-ar-if)#exit
RP/0/7/CPU0:R1(config-ospf-vrf-ar)#exit
```

Now we can check our config:

```
RP/0/7/CPU0:R1(config-ospf-vrf)#show configuration
Fri Mar 30 14:40:49.074 UTC
Building configuration...
!! IOS XR Configuration 4.1.1
interface Loopback1000
 vrf LAB
 ipv4 address 111.111.111.111 255.255.255.255
!
router ospf 100
 vrf LAB
  router-id 111.111.111.111
  area 0.0.0.0
   interface Loopback1000
   !
   interface GigabitEthernet0/3/0/3.100
   !
  !
 !
!
end
```

And finally commit the changes:

```
RP/0/7/CPU0:R1(config-ospf-vrf)#commit
Fri Mar 30 14:40:57.984 UTC
```

Now, lets check our routes:

```
RP/0/7/CPU0:R1#sh route vrf LAB ipv4
Fri Mar 30 14:41:40.746 UTC

Codes: C - connected, S - static, R - RIP, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - ISIS, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, su - IS-IS summary null, * - candidate default
       U - per-user static route, o - ODR, L - local, G  - DAGR
       A - access/subscriber, (!) - FRR Backup path

Gateway of last resort is not set

L    111.111.111.111/32 is directly connected, 00:00:40, Loopback1000
C    150.1.21.0/24 is directly connected, 00:29:26,
GigabitEthernet0/3/0/3.100
L    150.1.21.1/32 is directly connected, 00:29:26,
GigabitEthernet0/3/0/3.100
O    222.222.222.222/32 [110/2] via 150.1.21.2, 00:00:37,
GigabitEthernet0/3/0/3.100

RP/0/7/CPU0:R1#
```

Cool, we have a route to R2's Loop1000 of 222.222.222.222/32. Ping test time!

```
RP/0/7/CPU0:R1#ping vrf LAB 222.222.222.222 so loop1000
Fri Mar 30 14:41:50.331 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 222.222.222.222, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/2/4 ms
RP/0/7/CPU0:R1#
```

There you go, we have connectivity!

21. Basic MPLS - LDP

Ok, time for some MPLS! For this lab I will be using the OSPF routing protocol first, then we can move to IS-IS next. All we will be doing here is configuring LDP

First up, lets enabled LDP on all OSPF interfaces. Normally you would do this under each interface, but here we will use the MPLS LDP AUTOCONFIG command. This is a good command to use as it ensures that you do not miss configuring LDP on an interface.

```
RP/0/7/CPU0:R1#
RP/0/7/CPU0:R1#conf t
Sun Apr  1 18:58:04.084 UTC
RP/0/7/CPU0:R1(config)#router ospf LAB
```

Under the OSPF LAB process, we need to configure mpls ldp autoconfig and then commit it.

```
RP/0/7/CPU0:R1(config-ospf)#mpls ldp auto
RP/0/7/CPU0:R1(config-ospf)#commit
Sun Apr  1 18:58:12.277 UTC
RP/0/7/CPU0:R1(config-ospf)#exit
RP/0/7/CPU0:R1(config)#exit
```

Ok, now lets see what interfaces have LDP on them

```
RP/0/7/CPU0:R1#sh mpls ldp int
Sun Apr  1 18:58:18.902 UTC
MPLS LDP application must be enabled to use this command
```

Ahh, we configured the command but never enabled MPLS LDP. Remember, if a process is not needed – it does not run. So, lets enable the process

```
RP/0/7/CPU0:R1#conf t
Sun Apr  1 18:58:22.811 UTC
RP/0/7/CPU0:R1(config)#mpls ldp
```

Now, one thing to note on IOS XR, LDP is the only label protocol supported, TDP is not available.

```
RP/0/7/CPU0:R1(config-ldp)#label ?
  accept     Configure inbound label acceptance control
  advertise  Configure outbound label advertisement control
  allocate   Configure label allocation control
  <cr>
RP/0/7/CPU0:R1(config-ldp-lbl)#tag?
                                                           ^

% Invalid input detected at '^' marker.
RP/0/7/CPU0:R1(config-ldp)#comm
```

Now, lets check our interfaces and check for a neighbor.

```
RP/0/7/CPU0:R1#sh mpls ldp int
Sun Apr  1 19:04:23.402 UTC
Interface GigabitEthernet0/3/0/2 (0x4000500)
   Enabled via config: IGP Auto-config
Interface GigabitEthernet0/3/0/3.100 (0x4000700)
   Disabled
```

Yup, we see it's enabled on g0/3/0/2 via IGP Auto-config. Nice!
Now, let's check for a neighbor

```
RP/0/7/CPU0:R1#sh mpls ldp neighbor
Sun Apr  1 19:04:27.582 UTC

Peer LDP Identifier: 2.2.2.2:0
  TCP connection: 2.2.2.2:35051 - 1.1.1.1:646
  Graceful Restart: No
  Session Holdtime: 180 sec
  State: Oper; Msgs sent/rcvd: 13/24; Downstream-Unsolicited
  Up time: 00:05:57
  LDP Discovery Sources:
    GigabitEthernet0/3/0/2
  Addresses bound to this peer:
    2.2.2.2          150.1.12.2       200.100.200.100  200.200.200.200
    200.200.200.201  200.200.200.202  200.200.200.203  200.200.200.204
    200.200.200.205  200.200.200.206  200.200.200.207  200.200.200.208
    200.200.200.209  200.200.200.210

RP/0/7/CPU0:R1#
```

There you can see we have LDP neighbor with router-id 2.2.2.2 (R2) on G0/3/0/2. You can also see the ports we are using for this communication. Our local port is 646 and the remote port is 35051.

LDP Authentication

Ok, now onto neighbor password for LDP (both directed and all)

Lets configure a password for our neighbor, 2.2.2.2, of cisco

```
RP/0/7/CPU0:R1#conf t
Sun Apr  1 19:39:35.480 UTC
```

This is done under the LDP section

```
RP/0/7/CPU0:R1(config)#mpls ldp
RP/0/7/CPU0:R1(config-ldp)#nei 2.2.2.2 password cisco
RP/0/7/CPU0:R1(config-ldp)#comm
Sun Apr  1 19:40:04.498 UTC
```

Now here is something different than normal IOS, as soon as you enable authentication – the LDP session resets and enables the password. With IOS, you would need to clear the LDP session and allow it to re-establish.

```
RP/0/7/CPU0:Apr  1 19:40:06.205 : tcp[400]: %IP-TCP-3-BADAUTH : Invalid MD5
digest from 2.2.2.2:57032 to 1.1.1.1:646
```

Ok, I made the change to R2 so the passwords match, now we can look at our neighbor

```
RP/0/7/CPU0:R1#sh mpls ldp neighbor
Sun Apr  1 19:40:33.961 UTC

Peer LDP Identifier: 2.2.2.2:0
  TCP connection: 2.2.2.2:57491 - 1.1.1.1:646; MD5 on
  Graceful Restart: No
  Session Holdtime: 180 sec
  State: Oper; Msgs sent/rcvd: 7/18; Downstream-Unsolicited
  Up time: 00:00:10
  LDP Discovery Sources:
    GigabitEthernet0/3/0/2
  Addresses bound to this peer:
    2.2.2.2          150.1.12.2       200.100.200.100  200.200.200.200
    200.200.200.201  200.200.200.202  200.200.200.203  200.200.200.204
    200.200.200.205  200.200.200.206  200.200.200.207  200.200.200.208
    200.200.200.209  200.200.200.210

RP/0/7/CPU0:R1#
```

As you can see, next to the TCP connection, it now says ***MD5 on***. Previously nothing was after the port number.

You can also configure a password for any LDP session, that is done like follows:

```
RP/0/7/CPU0:R1#conf t
Sun Apr  1 19:40:45.561 UTC
RP/0/7/CPU0:R1(config)#mpls ldp
RP/0/7/CPU0:R1(config-ldp)#neighbor ?
  A.B.C.D   IP address of neighbor
  password  Configure password for MD5 authentication for all neighbors
RP/0/7/CPU0:R1(config-ldp)#neighbor password cisco
RP/0/7/CPU0:R1(config-ldp)#comm
Sun Apr  1 19:40:57.167 UTC
```

Now any LDP session must have a password. Now remember this in case you need to do directed LDP session some time down the road.

ISIS

Changing Metrics on an interface.

To change a metric on an interface in IS-IS, it is pretty simple. Just like before, all configuration are done under the routing protocol section of the config, interface subsection, and address family.

```
RP/0/7/CPU0:R1#conf t
Sun Apr  1 22:40:33.251 UTC
RP/0/7/CPU0:R1(config)#router ISIS LAB
RP/0/7/CPU0:R1(config-isis)#int g0/3/0/2
RP/0/7/CPU0:R1(config-isis-if)#address-family ipv4 un
RP/0/7/CPU0:R1(config-isis-if-af)#metric 20
RP/0/7/CPU0:R1(config-isis-if-af)#
```

And to check:

```
RP/0/7/CPU0:R1#sh isis interface g0/3/0/2
Sun Apr  1 22:42:11.124 UTC

GigabitEthernet0/3/0/2      Enabled
  Adjacency Formation:      Enabled
  Prefix Advertisement:     Enabled
  <--SNIP - Information removed for brevity -->

  IPv4 Unicast Topology:    Enabled
    Adjacency Formation:    Running
    Prefix Advertisement:   Running
    Metric (L1/L2):         20/20
    MPLS LDP Sync (L1/L2):  Disabled/Disabled
  IPv6 Unicast Topology:    Enabled
    Adjacency Formation:    Running
    Prefix Advertisement:   Running
    Metric (L1/L2):         10/10
    MPLS LDP Sync (L1/L2):  Disabled/Disabled

  IPv4 Address Family:      Enabled
    Protocol State:         Up
    Forwarding Address(es): 150.1.12.1
    Global Prefix(es):      150.1.12.0/24
  IPv6 Address Family:      Enabled
    Protocol State:         Up
    Forwarding Address(es): fe80::201:c9ff:fee8:dd7c
    Global Prefix(es):      2001:1:1:12::/64

  LSP transmit timer expires in 0 ms
  LSP transmission is idle
  Can send up to 9 back-to-back LSPs in the next 0 ms
RP/0/7/CPU0:R1#
```

As you can see, IPv4 now has a metric of 20 whereas IPv6 has the default metric of 10.

Passive Interfaces
Now, typically in ISIS you make the loopback interface passive.

To make an interface passive, is very simple.

```
RP/0/7/CPU0:R1#conf t
Sun Apr  1 22:45:10.308 UTC
RP/0/7/CPU0:R1(config)#router isis LAB
```

Change to the interface uder the protocol

```
RP/0/7/CPU0:R1(config-isis)#int loop0
```

And set it as passive.

```
RP/0/7/CPU0:R1(config-isis-if)#passive
RP/0/7/CPU0:R1(config-isis-if)#commit
```

Authentication
Time to configure IS-IS authentication. Again, all this is done under the routing process – makes keeping all relevant changes very close together.

```
RP/0/7/CPU0:R1(config)#router ISIS LAB
RP/0/7/CPU0:R1(config-isis)#inter g0/3/0/2
```

Now, to configure authentication we need to set the hello-password. As you can see we have some options listed – but for this lab we will use hmac-md5.

```
RP/0/7/CPU0:R1(config-isis-if)#hello-password ?
  WORD        The unencrypted (clear text) hello password
  accept      Use password for incoming authentication only
  clear       Specifies an unencrypted password will follow
  encrypted   Specifies an encrypted password will follow
  hmac-md5    Use HMAC-MD5 authentication
  keychain    Specifies a Key Chain name will follow
  text        Use cleartext password authentication
RP/0/7/CPU0:R1(config-isis-if)#hello-password hmac-md5 cisco
```

Now before we commit, let's look at our neighbors

```
RP/0/7/CPU0:R1(config-isis-if)#do show isis neighbors
Sun Apr  1 22:49:07.800 UTC

IS-IS LAB neighbors:
System Id      Interface        SNPA           State Holdtime Type IETF-NSF
GSR-R2         Gi0/3/0/2        00d0.7901.3a78 Up    7        L2   Capable

Total neighbor count: 1
RP/0/7/CPU0:R1(config-isis-if)#commit
Sun Apr  1 22:49:10.443 UTC
RP/0/7/CPU0:R1(config-isis-if)#
RP/0/7/CPU0:R1#
```

You may or may not have to clear the process; I did not and was able to catch this in the log with regards to ISIS neighbor authentication failure.

```
RP/0/7/CPU0:Apr  1 22:52:58.265 : isis[1003]: %ROUTING-ISIS-5-
AUTH_FAILURE_DROP : Dropped L2 LAN IIH from GigabitEthernet0/3/0/2 SNPA
00d0.7901.3a78 due to authentication TLV not found
```

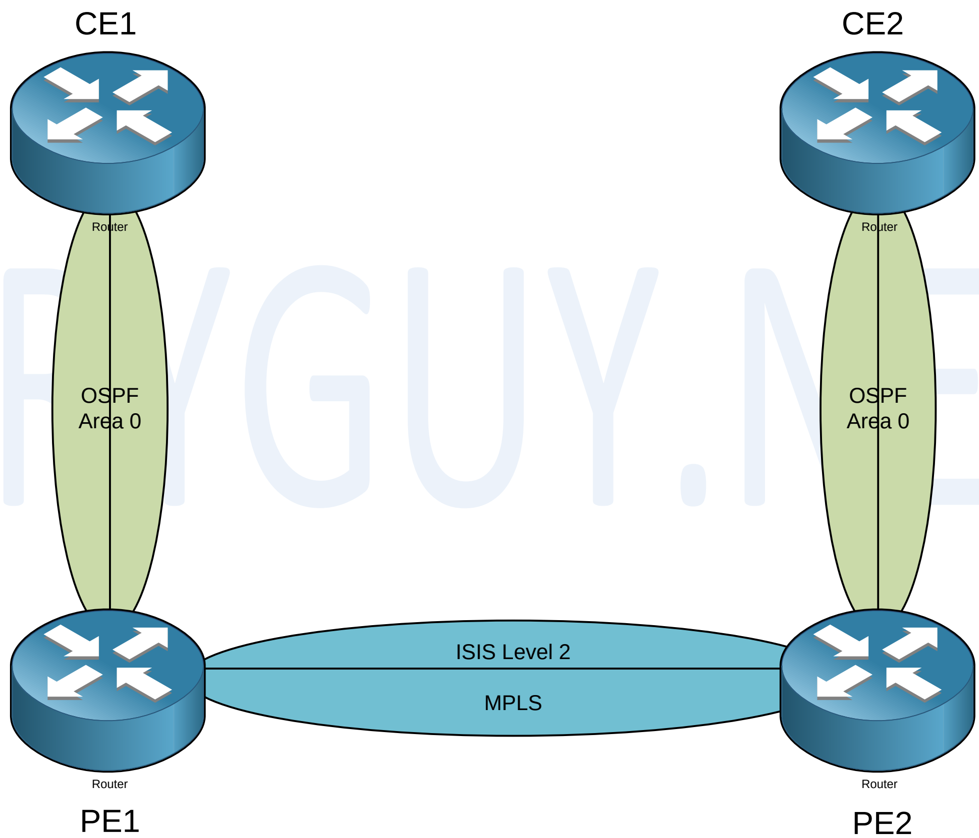
Once I configured the password on the other router, we have neighbors again!

```
RP/0/7/CPU0:R1#sh isis neighbors
Sun Apr  1 22:55:55.066 UTC

IS-IS LAB neighbors:
System Id      Interface        SNPA           State Holdtime Type IETF-NSF
GSR-R2         Gi0/3/0/2        00d0.7901.3a78 Up    7        L2   Capable

Total neighbor count: 1
RP/0/7/CPU0:R1#
```

22.MPLS VPN

Next up is MPLS VPN; actually VPNv4 routes are what these actually are since we are only passing IPv4 traffic in this example.

So we have this diagram below – CE1 and CE2 are the customer routers and both are running OSPF in Area 0. Then need to talk to each other but do not have a direct connection available, so they have contracted us to provide connectivity via MPLS between them. What we will now do is build a pseudo MPLS network between PE1 and PE2, establish an iBGP peering, create the associated customer VRF and then peer with the customer via OSPF Area 0.



So, first up lets configure CE1 using an IP of 10.3.3.3/32 for the loopback and 10.1.13.3/24 for the link facing PE1.

```
CE1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
```

First up, Loopback 0. Since this is IOS, you will need to use the full dotted decimal subnet mask

```
CE1(config)#int loop0
CE1(config-if)#ip add 10.3.3.3 255.255.255.255
```

Now for the interface facing the PE (here f0/0)

```
CE1(config-if)#int f0/0
CE1(config-if)#ip add 10.1.13.3 255.255.255.0
CE1(config-if)#no shut
CE1(config-if)#^Z
CE1#
```

Ok, now lets get CE2 done since it basically the same – but here we will use 10.4.4.4/32 and 10.4.24.4/24

```
CE2#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
```

First up, Loopback 0. Since this is IOS, you will need to use the full dotted decimal subnet mask

```
CE2(config)#int loop0
CE2(config-if)#ip add 10.3.4.4 255.255.255.255
```

Now for the interface facing the PE (here f0/0)

```
CE2(config-if)#int f0/0
CE2(config-if)#ip add 10.1.24.4 255.255.255.0
CE2(config-if)#no shut
CE2(config-if)#^Z
CE2#
```

Now we can do the OSPF configs for these routers. Since this is a lab, I am just going to put the 10/8 network in Area 0. So, first up - CE1

```
CE1(config)#router ospf 1
CE1(config-router)#net 10.0.0.0 0.255.255.255 a 0
CE1(config-router)#
```

And now CE2:

```
CE2(config)# router ospf 1
CE2(config-router)#net 10.0.0.0 0.255.255.255 a 0
CE2(config-router)#
```

Easy part done, now to build the PE nework.

For the PE network we are going to use ISIS for our internal routing protocol and then use BGP on top of that to connect the routers together to pass the VPNv4 routes. Why ISIS you ask? It is because you can use one process for IPv4 and IPv6 traffic. With OSPF you need to run two processes, OSPF for IPv4 and OSPFv3 for IPv6. A single process makes it easier to support as well as if new protocols come around, ISIS won't really care since it is not IP based (CLNS based).

Now it is time to get some IP addresses on PE1. We will use G0/1/0/11 for connection to PE2 and also create Loopback0. The IPs for the connection to PE2 will be 150.1.12.0/24 and the Loopback will be 150.1.1.1/32.

```
RP/0/RSP0/CPU0:R1#conf t
Fri Apr 20 00:34:18.971 UTC
RP/0/RSP0/CPU0:R1(config)#int g0/1/0/11
RP/0/RSP0/CPU0:R1(config-if)#ip add 150.1.12.1/24
RP/0/RSP0/CPU0:R1(config-if)#no shut
RP/0/RSP0/CPU0:R1(config-if)#commit
Fri Apr 20 00:34:25.555 UTC
RP/0/RSP0/CPU0:R1(config-if)#
RP/0/RSP0/CPU0:R1(config-if)#int loop0
RP/0/RSP0/CPU0:R1(config-if)#ip add 150.1.1.1/32
RP/0/RSP0/CPU0:R1(config-if)#commit
Fri Apr 20 00:34:39.839 UTC
RP/0/RSP0/CPU0:R1(config-if)#
```

Ok, lets get PE2 done now and test the interface connectivity. After we confirm that, we can do ISIS.

```
RP/0/RSP0/CPU0:R2#conf t
Fri Apr 20 00:35:39.031 UTC
RP/0/RSP0/CPU0:R2(config)#int g0/1/0/11
RP/0/RSP0/CPU0:R2(config-if)#ip add 150.1.12.2/24
RP/0/RSP0/CPU0:R2(config-if)#no shut
RP/0/RSP0/CPU0:R2(config-if)#int loop0
RP/0/RSP0/CPU0:R2(config-if)#ip add 150.2.2.2/32
RP/0/RSP0/CPU0:R2(config-if)#comm
Fri Apr 20 00:35:54.565 UTC
RP/0/RSP0/CPU0:R2(config-if)#
```

Ok, now lets do a PING test to see if we have connectivity:

```
RP/0/RSP0/CPU0:R2#ping 150.1.12.1
Fri Apr 20 00:36:09.946 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.12.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms
RP/0/RSP0/CPU0:R2#
```

Good, now onto ISIS.
For this we will use ISIS area 49.0000.0000.000X.00 where X = Router number and Level-2 only area.

PE1:
Lets define the routing process - Core

```
RP/0/RSP0/CPU0:R1(config)#router isis Core
```

Set are Network ID

```
RP/0/RSP0/CPU0:R1(config-isis)#net 49.0000.0000.0001.00
```

And our IS Type
```
RP/0/RSP0/CPU0:R1(config-isis)#is-type level-2
```

Set the loopback interface into ISIS and place it in PASSIVE mode
```
RP/0/RSP0/CPU0:R1(config-isis)#int loop0
RP/0/RSP0/CPU0:R1(config-isis-if)#passive
RP/0/RSP0/CPU0:R1(config-isis-if)#address-family ipv4 un
RP/0/RSP0/CPU0:R1(config-isis-if-af)#exit
```

Now for g0/1/0/11
```
RP/0/RSP0/CPU0:R1(config-isis-if)#int g0/1/0/11
RP/0/RSP0/CPU0:R1(config-isis-if)#address-family ipv4 unicast
RP/0/RSP0/CPU0:R1(config-isis-if-af)#exit
```

And finally, commit our changes.
```
RP/0/RSP0/CPU0:R1(config-isis-if)#commit
Fri Apr 20 01:02:31.714 UTC
```

Now, lets get PE2 setup the same way:
```
RP/0/RSP0/CPU0:R2(config)#router isis Core
RP/0/RSP0/CPU0:R2(config-isis)#net 49.0000.0000.0002.00
RP/0/RSP0/CPU0:R2(config-isis)#is-type level-2
RP/0/RSP0/CPU0:R2(config-isis)#int loop0
RP/0/RSP0/CPU0:R2(config-isis-if)#passive
RP/0/RSP0/CPU0:R2(config-isis-if)#address-family ipv4 u
RP/0/RSP0/CPU0:R2(config-isis-if-af)#exit
RP/0/RSP0/CPU0:R2(config-isis-if)#exit
RP/0/RSP0/CPU0:R2(config-isis)#int g0/1/0/11
RP/0/RSP0/CPU0:R2(config-isis-if)#address-family ipv4 un
RP/0/RSP0/CPU0:R2(config-isis-if-af)#exit
RP/0/RSP0/CPU0:R2(config-isis-if)#commit
```

Ok, lets check our ISIS neighbors
```
RP/0/RSP0/CPU0:R2#sh isis neighbors
Fri Apr 20 01:10:31.813 UTC

IS-IS Core neighbors:
System Id      Interface    SNPA           State Holdtime Type IETF-NSF
R1             Gi0/1/0/11   6c9c.ed26.ab91 Up    22       L2   Capable

Total neighbor count: 1
RP/0/RSP0/CPU0:R2#
```

Yup, all neighbored up. Time to check the routes:
```
RP/0/RSP0/CPU0:R2#sh ip route isis
Fri Apr 20 01:10:54.269 UTC

i L2 150.1.1.1/32 [115/10] via 150.1.12.1, 00:07:06, GigabitEthernet0/1/0/11
RP/0/RSP0/CPU0:R2#
```

Cool, we have a Level2 route to 150.1.1.1 via R1. Now, lets PING to make sure.

```
RP/0/RSP0/CPU0:R2#ping 150.1.1.1 so l0
Fri Apr 20 01:11:26.132 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.1.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms
RP/0/RSP0/CPU0:R2#
```

Connectivity is working, cool!

Next up, LDP.
First up PE1:

```
RP/0/RSP0/CPU0:R1(config)#mpls ldp
```

Like all other IOS XR commands, you assign the interfaces under the protocol.

```
RP/0/RSP0/CPU0:R1(config-ldp)#int g0/1/0/11
RP/0/RSP0/CPU0:R1(config-ldp-if)#comm
Fri Apr 20 01:18:00.216 UTC
RP/0/RSP0/CPU0:R1(config-ldp-if)#
```

Ok, PE2

```
RP/0/RSP0/CPU0:R2(config)#mpls ldp
RP/0/RSP0/CPU0:R2(config-ldp)#int g0/1/0/11
RP/0/RSP0/CPU0:R2(config-ldp-if)#comm
Fri Apr 20 01:18:08.116 UTC
```

Now lets check our LDP neighbors:

```
RP/0/RSP0/CPU0:R2#sh mpls ldp neighbor
Fri Apr 20 01:21:21.957 UTC

Peer LDP Identifier: 150.1.1.1:0
  TCP connection: 150.1.1.1:646 - 150.2.2.2:43857
  Graceful Restart: No
  Session Holdtime: 180 sec
  State: Oper; Msgs sent/rcvd: 12/10; Downstream-Unsolicited
  Up time: 00:00:26
  LDP Discovery Sources:
    GigabitEthernet0/1/0/11
  Addresses bound to this peer:
    150.1.1.1        150.1.12.1

RP/0/RSP0/CPU0:R2#
```

Cool, we have a LDP session with PE1 and we can see the IPs bound to the peer.

We are getting there, we still have BGP, VRF, and the OSPF configuration to do yet. We will save the BGP part until last – so for now, VRF time.

For this example, we will call our VRF R3R4 since we are connecting R3 (CE1) and R4 (CE2).

PE1 up first:

```
RP/0/RSP0/CPU0:R1#conf t
Fri Apr 20 01:38:35.869 UTC
```

Lets define the name of our VRF

```
RP/0/RSP0/CPU0:R1(config)#vrf R3R4
```

Now we need to configure the appropriate address family, ipv4 unicast.

```
RP/0/RSP0/CPU0:R1(config-vrf)# address-family ipv4 unicast
```

Now we need to define our route-targets that we are going to import, and export. What is a route-target? Quickly it is a 64-bit BGP community that is used for tagging prefixes, making every prefix unique and also allows the remote PE routers to know what routes belong to what VRF (import).

For this example, we will use 100:100 for both.

```
RP/0/RSP0/CPU0:R1(config-vrf-af)#  import route-target
RP/0/RSP0/CPU0:R1(config-vrf-import-rt)#   100:100
RP/0/RSP0/CPU0:R1(config-vrf-import-rt)#  export route-target
RP/0/RSP0/CPU0:R1(config-vrf-export-rt)#   100:100
```

And commit the changes.

```
RP/0/RSP0/CPU0:R1(config-vrf-export-rt)#commit
Fri Apr 20 01:38:39.866 UTC
```

Now we can create the same VRF with the same route-targets:

```
RP/0/RSP0/CPU0:R2(config)#vrf R3R4
RP/0/RSP0/CPU0:R2(config-vrf)# address-family ipv4 unicast
RP/0/RSP0/CPU0:R2(config-vrf-af)#  import route-target
RP/0/RSP0/CPU0:R2(config-vrf-export-rt)#   100:100
RP/0/RSP0/CPU0:R2(config-vrf-export-rt)#  export route-target
RP/0/RSP0/CPU0:R2(config-vrf-export-rt)#   100:100
RP/0/RSP0/CPU0:R2(config-vrf-export-rt)#comm
Fri Apr 20 01:45:18.380 UTC
```

Ok, time to check to see if the VRF is there:

```
RP/0/RSP0/CPU0:R2#sh vrf R3R4
Fri Apr 20 01:45:52.204 UTC
VRF         RD                  RT                         AFI   SAFI
R3R4        100:100             import  100:100            IPV4 Unicast
                                export  100:100            IPV4 Unicast
RP/0/RSP0/CPU0:R2#
```

Yup, we have a VRF. Now we can assign the interfaces facing the CE routers to the appropriate VRF, configure the IP addresses, and then do a PING test across the interface.

```
PE1:
RP/0/RSP0/CPU0:R1#conf t
Fri Apr 20 01:48:48.712 UTC

Lets get to our interface, G0/1/0/19
RP/0/RSP0/CPU0:R1(config)#interface GigabitEthernet0/1/0/19

Now we can assign the VRF of R3R4
RP/0/RSP0/CPU0:R1(config-if)# vrf R3R4

Configure our IP
RP/0/RSP0/CPU0:R1(config-if)# ipv4 address 10.1.13.1 255.255.255.0

Since this is a 100M link, we will need to hard code it for the GBICs sake.
RP/0/RSP0/CPU0:R1(config-if)# speed 100

And Commit our changes
RP/0/RSP0/CPU0:R1(config-if)#comm
Fri Apr 20 01:48:51.120 UTC
RP/0/RSP0/CPU0:R1(config-if)#

Once that is done, let do PE2 the same way.
PE2:
RP/0/RSP0/CPU0:R2#conf t
Fri Apr 20 01:48:45.677 UTC
RP/0/RSP0/CPU0:R2(config)#interface GigabitEthernet0/1/0/19
RP/0/RSP0/CPU0:R2(config-if)# vrf R3R4
RP/0/RSP0/CPU0:R2(config-if)# ipv4 address 10.1.24.2 255.255.255.0
RP/0/RSP0/CPU0:R2(config-if)# speed 100
RP/0/RSP0/CPU0:R2(config-if)#commit
Fri Apr 20 01:48:51.059 UTC
RP/0/RSP0/CPU0:R2(config-if)#

Now we can test a ping from PE1 to CE1 and PE2 to CE2.
RP/0/RSP0/CPU0:R1#ping 10.1.13.3
Fri Apr 20 01:57:01.969 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.1.13.3, timeout is 2 seconds:
UUUUU
Success rate is 0 percent (0/5)
RP/0/RSP0/CPU0:R1#

Hmm, that failed – why?  Well, when an interface lives in a VRF, you need to
PING from that VRF. Lets try that again using VRF R3R4
RP/0/RSP0/CPU0:R1#ping vrf R3R4 10.1.13.3
Fri Apr 20 01:57:11.522 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.1.13.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/2 ms
```

There, that worked. Lets check R2

```
RP/0/RSP0/CPU0:R2#ping vrf R3R4 10.1.24.4
Fri Apr 20 01:56:49.742 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.1.24.4, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms
RP/0/RSP0/CPU0:R2#
```

Ok, we have connectivity. Now we can get OSPF working between the PE and the CEs.

First up, PE1

We need to specify what we want to call our OSPF process, here I just used R3R4

```
RP/0/RSP0/CPU0:R1(config)#router ospf R3R4
```

Now we need to configure OSPF for the VRF

```
RP/0/RSP0/CPU0:R1(config-ospf)# vrf R3R4
```

Now for the area

```
RP/0/RSP0/CPU0:R1(config-ospf-vrf)#  area 0
```

And then place the interfaces that we want in area 0

```
RP/0/RSP0/CPU0:R1(config-ospf-vrf-ar)#   interface GigabitEthernet0/1/0/19
```

And commit our changes (I just hit CTRL-Z)

```
Uncommitted changes found, commit them before exiting(yes/no/cancel)?
[cancel]:yes
RP/0/RSP0/CPU0:R1#
```

Ok, that is PE1 – now for PE2

```
RP/0/RSP0/CPU0:R2#conf t
Fri Apr 20 02:13:03.521 UTC
RP/0/RSP0/CPU0:R2(config)#router ospf R3R4
RP/0/RSP0/CPU0:R2(config-ospf)# vrf R3R4
RP/0/RSP0/CPU0:R2(config-ospf-vrf)#  area 0
RP/0/RSP0/CPU0:R2(config-ospf-vrf-ar)#   interface GigabitEthernet0/1/0/19
RP/0/RSP0/CPU0:R2(config-ospf-vrf-ar-if)#exit
RP/0/RSP0/CPU0:R2(config-ospf-vrf-ar)# exit
RP/0/RSP0/CPU0:R2(config-ospf-vrf)# exit
RP/0/RSP0/CPU0:R2(config-ospf)#com
Fri Apr 20 02:13:14.843 UTC
RP/0/RSP0/CPU0:R2(config-ospf)#
```

Ok, PE2 done. Now we can check for OSPF neighbor in that VRF.
To do that, we need to use the following command: *show ospf (OSPF Process) vrf (VRF Name) neighbor*

```
RP/0/RSP0/CPU0:R1#sh ospf R3R4 vrf R3R4 neighbor
Fri Apr 20 02:18:16.826 UTC

* Indicates MADJ interface

Neighbors for OSPF R3R4, VRF R3R4

Neighbor ID     Pri   State           Dead Time   Address         Interface
10.3.3.3        1     FULL/DR         00:00:39    10.1.13.3
GigabitEthernet0/1/0/19
    Neighbor is up for 00:00:05

Total neighbor count: 1
RP/0/RSP0/CPU0:R1#
```

Ok, lets check PE2:

```
RP/0/RSP0/CPU0:R2#sh ospf R3R4 vrf R3R4 neighbor
Fri Apr 20 02:19:19.588 UTC

* Indicates MADJ interface

Neighbors for OSPF R3R4, VRF R3R4

Neighbor ID     Pri   State           Dead Time   Address         Interface
10.4.4.4        1     FULL/DR         00:00:39    10.1.24.4
GigabitEthernet0/1/0/19
    Neighbor is up for 00:00:03

Total neighbor count: 1
RP/0/RSP0/CPU0:R2#
```

Ok, both PE routers are neighbored up with the CE routers.

Now, if we look at CE1's routing table – what will we see?

```
CE1#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static
route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
```

```
C       10.1.13.0/24 is directly connected, FastEthernet0/0
C       10.3.3.3/32 is directly connected, Loopback0
CE1#
```

We only see our local routes, nothing from CE2 yet. This is because we have not built the VPNv4 session between PE1 and PE2 yet. We need to configure BGP VPNv4 in order to get the two PE routers to pass the tagged routes to each other. So, onto BGP we go!

For this we will peer with PE2 loopback (150.2.2.2) using AS 1.
PE1 first:

```
RP/0/RSP0/CPU0:R1#conf t
Fri Apr 20 02:21:32.174 UTC
```

First we define our BGP process and AS number

```
RP/0/RSP0/CPU0:R1(config)#router bgp 1
```

Enable vpv4 address family

```
RP/0/RSP0/CPU0:R1(config)#address-family vpnv4 unicast
```

Now we can configure our neighbor and all the info.

```
RP/0/RSP0/CPU0:R1(config-bgp-af)# neighbor 150.2.2.2
RP/0/RSP0/CPU0:R1(config-bgp-nbr)#  remote-as 1
```

Remember to specify the loopback as the update-source

```
RP/0/RSP0/CPU0:R1(config-bgp-nbr)#  update-source Loopback0
```

Now we can enable VPNv4 address family with that neighbor

```
RP/0/RSP0/CPU0:R1(config-bgp-nbr)#  address-family vpnv4 unicast
```

Now we can configure the VRF parameters that BGP needs to know
First define our VRF

```
RP/0/RSP0/CPU0:R1(config-bgp-nbr-af)# vrf R3R4
```

Assign our Route Distinguisher (RD)

```
RP/0/RSP0/CPU0:R1(config-bgp-vrf)#  rd 100:100
```

Enable IPv4 Unicse for this VRF

```
RP/0/RSP0/CPU0:R1(config-bgp-vrf)#  address-family ipv4 unicast
```

And finally redistribute our OSPF learned routes into BGP VRF R3R4

```
RP/0/RSP0/CPU0:R1(config-bgp-vrf-af)# redistribute ospf R3R4 match internal
external
RP/0/RSP0/CPU0:R1(config-bgp-vrf-af)# ^Z

Uncommitted changes found, commit them before exiting(yes/no/cancel)?
[cancel]:yes
RP/0/RSP0/CPU0:R1#
```

Ok, now that that is done – we need to do the same thing on PE2

```
RP/0/RSP0/CPU0:R2(config)#router bgp 1
RP/0/RSP0/CPU0:R2(config-bgp)# address-family ipv4 unicast
RP/0/RSP0/CPU0:R2(config-bgp-af)# address-family vpnv4 unicast
RP/0/RSP0/CPU0:R2(config-bgp-af)# neighbor 150.1.1.1
RP/0/RSP0/CPU0:R2(config-bgp-nbr)#  remote-as 1
RP/0/RSP0/CPU0:R2(config-bgp-nbr)#  update-source Loopback0
RP/0/RSP0/CPU0:R2(config-bgp-nbr)#  address-family vpnv4 unicast
RP/0/RSP0/CPU0:R2(config-bgp-nbr-af)# vrf R3R4
RP/0/RSP0/CPU0:R2(config-bgp-vrf)#  rd 100:100
RP/0/RSP0/CPU0:R2(config-bgp-vrf)#  address-family ipv4 unicast
RP/0/RSP0/CPU0:R2(config-bgp-vrf-af)#   redistribute ospf R3R4 match internal
external
RP/0/RSP0/CPU0:R2(config-bgp-vrf-af)#exit
RP/0/RSP0/CPU0:R2(config-bgp-vrf)#exit
RP/0/RSP0/CPU0:R2(config-bgp-vrf)#comm
Fri Apr 20 02:27:10.491 UTC
```

Ok, since this is a VPNv4 neighbor we need to check to see if we are neighbored up:

```
RP/0/RSP0/CPU0:R2#sh bgp vpnv4 unicast  summary
Fri Apr 20 02:28:05.467 UTC
BGP router identifier 150.2.2.2, local AS number 1
BGP generic scan interval 60 secs
BGP table state: Active
Table ID: 0x0   RD version: 3889240856
BGP main routing table version 25
BGP scan interval 60 secs

BGP is operating in STANDALONE mode.


Process       RcvTblVer   bRIB/RIB   LabelVer  ImportVer  SendTblVer
StandbyVer
Speaker              25         25         25         25           25
25

Neighbor        Spk    AS MsgRcvd MsgSent   TblVer  InQ OutQ  Up/Down
St/PfxRcd
150.1.1.1         0     1   14168   14173       25    0    0 00:00:48
2

RP/0/RSP0/CPU0:R2#
```

Yup, we are up and we can see what we are receiving 2 prefixes as well!
Wonder what they are? To find out, use the *show bgp vpnv4 unicast* command

```
RP/0/RSP0/CPU0:R2#sh bgp vpnv4 unicast
Fri Apr 20 02:29:01.202 UTC
BGP router identifier 150.2.2.2, local AS number 1
BGP generic scan interval 60 secs
```

```
BGP table state: Active
Table ID: 0x0   RD version: 3889240856
BGP main routing table version 25
BGP scan interval 60 secs

Status codes: s suppressed, d damped, h history, * valid, > best
              i - internal, r RIB-failure, S stale
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network            Next Hop            Metric LocPrf Weight Path
Route Distinguisher: 100:100 (default for vrf R3R4)
*>i10.1.13.0/24       150.1.1.1                0    100      0 ?
*> 10.1.24.0/24       0.0.0.0                  0         32768 ?
*>i10.3.3.3/32        150.1.1.1                2    100      0 ?
*> 10.4.4.4/32        10.1.24.4                2         32768 ?

Processed 4 prefixes, 4 paths
RP/0/RSP0/CPU0:R2#
```

Nice, we can see we have routes from CE1 and CE2.
Now, lets see if CE1 has routes to CE2

```
CE1#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static
route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C       10.1.13.0/24 is directly connected, FastEthernet0/0
C       10.3.3.3/32 is directly connected, Loopback0
CE1#
```

Nope, hmm. What did we forget? I know, we redistributed OSPF into BGP, but we did not redistribute BGP into OSPF. Lets get that fixed.

PE1:

```
RP/0/RSP0/CPU0:R1(config)#router ospf R3R4
RP/0/RSP0/CPU0:R1(config-ospf)#vrf R3R4
RP/0/RSP0/CPU0:R1(config-ospf-vrf)# redistribute bgp 1
RP/0/RSP0/CPU0:R1(config-ospf-vrf)#comm
Fri Apr 20 02:31:44.637 UTC
RP/0/RSP0/CPU0:R1(config-ospf-vrf)#
```

And on PE2:

```
RP/0/RSP0/CPU0:R2(config)#router ospf R3R4
RP/0/RSP0/CPU0:R2(config-ospf)#vrf R3R4
RP/0/RSP0/CPU0:R2(config-ospf-vrf)# redistribute bgp 1
RP/0/RSP0/CPU0:R2(config-ospf-vrf)# ^Z
Uncommitted changes found, commit them before exiting(yes/no/cancel)?
[cancel]:yes
RP/0/RSP0/CPU0:R2#
```

Ok, lets check CE1 for routes to CE2 now

```
CE1#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static
route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
C       10.1.13.0/24 is directly connected, FastEthernet0/0
C       10.3.3.3/32 is directly connected, Loopback0
O IA    10.4.4.4/32 [110/12] via 10.1.13.1, 00:00:51, FastEthernet0/0
O IA    10.1.24.0/24 [110/11] via 10.1.13.1, 00:00:51, FastEthernet0/0
CE1#
```

There they are, lets do a PING

```
CE1#ping 10.4.4.4 so l0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.4.4.4, timeout is 2 seconds:
Packet sent with a source address of 10.3.3.3
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/4 ms
CE1#
```

Nice, we can PING!

Now, one other thing that you should notice with the CE1 routing table, routes to CE2 are seen as O IA, OSPF InterArea routes. This is what is expected when you run the same CE OSPF process ID over a MPLS network – the BGP will carry the extra attributes creating what is called a Super Backbone. When we decode the BGP route information using the show bgp vpnv4 unicast vrf R3R4 10.1.13.0/24 command, we get the following output - notice the extended community information, this is where the extra information is carried. We will actually pull up both 10.1.13.0 and 10.1.14.0 so you can see.

```
RP/0/RSP0/CPU0:R2#sh bgp vpnv4 unicast vrf R3R4 10.1.13.0
Fri Apr 20 02:45:47.177 UTC
BGP routing table entry for 10.1.13.0/24, Route Distinguisher: 100:100
Versions:
  Process           bRIB/RIB  SendTblVer
  Speaker                 24          24
Last Modified: Apr 20 02:27:22.347 for 00:18:24
Paths: (1 available, best #1)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  Local
    150.1.1.1 (metric 10) from 150.1.1.1 (150.1.1.1)
      Received Label 16001
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported
      Received Path ID 0, Local Path ID 1, version 24
      Extended community: RT:100:100 OSPF route-type:0:2:0x0 OSPF router-
id:150.1.1.1
RP/0/RSP0/CPU0:R2#sh bgp vpnv4 unicast vrf R3R4 10.1.24.0
Fri Apr 20 03:23:37.990 UTC
BGP routing table entry for 10.1.24.0/24, Route Distinguisher: 100:100
Versions:
  Process           bRIB/RIB  SendTblVer
  Speaker                 20          20
    Local Label: 16001
Last Modified: Apr 20 02:19:16.347 for 01:04:21
Paths: (1 available, best #1)
  Advertised to peers (in unique update groups):
    150.1.1.1
  Path #1: Received by speaker 0
  Advertised to peers (in unique update groups):
    150.1.1.1
  Local
    0.0.0.0 from 0.0.0.0 (150.2.2.2)
      Origin incomplete, metric 0, localpref 100, weight 32768, valid,
redistributed, best, group-best, import-candidate
      Received Path ID 0, Local Path ID 1, version 20
      Extended community: RT:100:100 OSPF route-type:0:2:0x0 OSPF router-
id:150.2.2.2
RP/0/RSP0/CPU0:R2#
```

There is a way to prevent this from happening and that is to create a Domain-ID for the OSPF process on one of the PE routers.

```
RP/0/RSP0/CPU0:R2#conf t
Fri Apr 20 03:30:11.463 UTC
```

Navigate to the OSPF VRF process

```
RP/0/RSP0/CPU0:R2(config)#router ospf R3R4
RP/0/RSP0/CPU0:R2(config-ospf)#vrf R3R4
```

Now, lets see what Domain-id types we have – See RFC 4577 for more info on these.

```
RP/0/RSP0/CPU0:R2(config-ospf-vrf)#domain-id type ?
  0005   Type 0x0005
  0105   Type 0x0105
  0205   Type 0x0205
  8005   Type 0x8005
RP/0/RSP0/CPU0:R2(config-ospf-vrf)#domain-id type 0005 value ?
  WORD  OSPF domain ID ext. community value in Hex (6 octets)
```

Now lets set it to a value

```
RP/0/RSP0/CPU0:R2(config-ospf-vrf)#domain-id type 0105 value AABBCCDDEEFF
```

When you do this, the routes on CE are now E2 routes:

```
CE1#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static
route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
C       10.1.13.0/24 is directly connected, FastEthernet0/0
C       10.3.3.3/32 is directly connected, Loopback0
O E2    10.4.4.4/32 [110/2] via 10.1.13.1, 01:04:24, FastEthernet0/0
O E2    10.1.24.0/24 [110/1] via 10.1.13.1, 01:04:24, FastEthernet0/0
CE1#
```

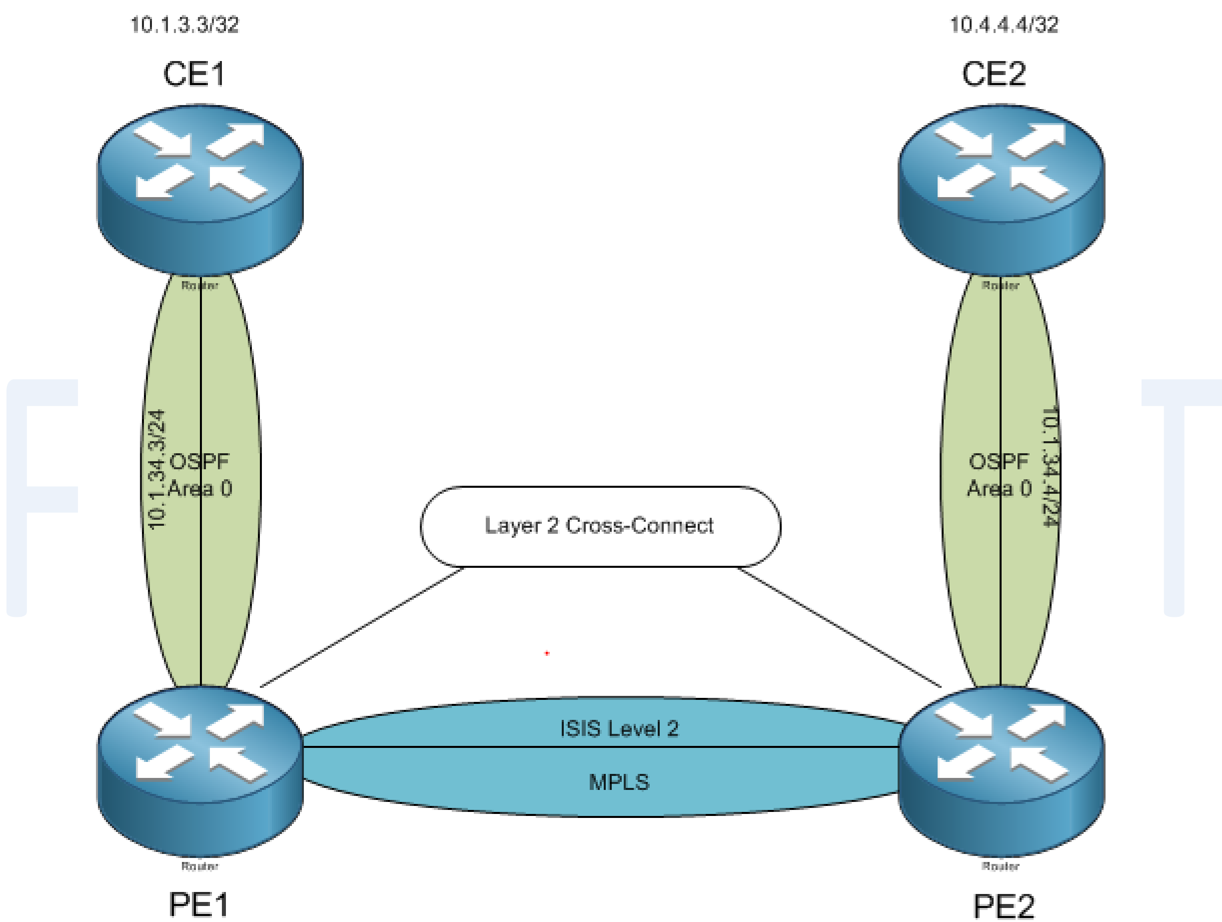
This can also work in reverse, if you want to create a SuperBackbone but the OSPF processes are different, you can set the domain-id to be the same.

```
RP/0/RSP0/CPU0:R1(config-ospf-vrf)#domain-id type 0105 value AABBCCDDEEFF

CE1#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS
       ia - IS-IS inter area, * - candidate default, U - per-user
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set
```

```
     10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
C       10.1.13.0/24 is directly connected, FastEthernet0/0
C       10.3.3.3/32 is directly connected, Loopback0
O IA    10.4.4.4/32 [110/12] via 10.1.13.1, 00:00:04, FastEthernet0/0
O IA    10.1.24.0/24 [110/11] via 10.1.13.1, 00:00:04, FastEthernet0/0
CE1#
```
There, back to IA routes again.
There is much more to domain-id, but I will save that for another day.

23.L2VPN

Ok, now it is time for some L2VPN. Here we will use the same diagram as before, but instead of providing MPLS VPN between CE1 and CE2, we are going to create a L2VPN so that CE1 and CE2 think that they are directly connected to each other.



First up, CE1

```
CE1(config-if)#int f0/0
CE1(config-if)#ip add 10.1.34.3 255.255.255.0
CE1(config-if)#int l0
CE1(config-if)#ip add 10.1.3.3 255.255.255.255
CE1(config-if)#router ospf 1
CE1(config-router)#net 10.0.0.0 0.255.255.255 a 0
CE1(config-router)#
```

Now CE2

```
CE2(config-if)#ip add 10.1.34.4 255.255.255.0
CE2(config-if)#int l0
CE2(config-if)#ip add 10.4.4.4 255.255.255.255
CE2(config-if)#router ospf 1
CE2(config-router)#net 10.0.0.0 0.255.255.255 a 0
CE2(config-router)#
```

Ok, now time for the PE routers.
PE1:
First, lets reset the interface to all its defaults:

```
RP/0/RSP0/CPU0:PE1(config)#default interface g0/1/0/19
RP/0/RSP0/CPU0:PE1(config)#commit
```

Now PE2:

```
RP/0/RSP0/CPU0:PE2(config)#default interface g0/1/0/19
RP/0/RSP0/CPU0:PE2(config)#commit
```

Now, lets kill our OSPF sessions. On both routers:
PE1:

```
RP/0/RSP0/CPU0:PE1(config)#no router ospf 100
RP/0/RSP0/CPU0:PE1(config)#commit
```

PE2:

```
RP/0/RSP0/CPU0:PE2(config)#no router ospf 100
RP/0/RSP0/CPU0:PE2(config)#commit
```

OK, now we can build out L2VPN cross-connects.
Fist up, we need to get to the L2VPN configuration

```
RP/0/RSP0/CPU0:PE1(config)#l2vpn
```

Now to configure our X-Connect group

```
RP/0/RSP0/CPU0:PE1(config-l2vpn)#xconnect group R3R4
```

And our Point-to-Point settings

```
RP/0/RSP0/CPU0:PE1(config-l2vpn-xc)#p2p R3_to_R4
```

Place the interface in the P2P group R3_to_R4

```
RP/0/RSP0/CPU0:PE1(config-l2vpn-xc-p2p)#interface g0/1/0/19
```

Specify our Neighbor for this with a pseudowire ID (think of it as a circuit ID) and then commit our changes

```
RP/0/RSP0/CPU0:PE1(config-l2vpn-xc-p2p)#neighbor 150.2.2.2 pw-id 304
RP/0/RSP0/CPU0:PE1(config-l2vpn-xc-p2p-pw)#comm
```

Now, this is unique to our CE devices, we need to specify the speed in order to get the interfaces up as the CE routers here do not support Gigabit Ethernet

```
RP/0/RSP0/CPU0:PE2(config-if)#int g0/1/0/19
RP/0/RSP0/CPU0:PE2(config-if)#spee 100
RP/0/RSP0/CPU0:PE2(config-if)#comm
```

Now for PE2:

```
RP/0/RSP0/CPU0:PE2(config)#l2vpn
RP/0/RSP0/CPU0:PE2(config-l2vpn)#xconnect group R3R4
RP/0/RSP0/CPU0:PE2(config-l2vpn-xc)#p2p R3_to_R4
RP/0/RSP0/CPU0:PE2(config-l2vpn-xc-p2p)#interface g0/1/0/19
```

The Pseudo-wire ID must match.

```
RP/0/RSP0/CPU0:PE2(config-l2vpn-xc-p2p)#neighbor 150.1.1.1 pw-id 304
RP/0/RSP0/CPU0:PE2(config-l2vpn-xc-p2p-pw)#comm
RP/0/RSP0/CPU0:PE2(config-if)#int g0/1/0/19
RP/0/RSP0/CPU0:PE2(config-if)#spee 100
RP/0/RSP0/CPU0:PE2(config-if)#comm
```

Now we can look at our L2VPN Cross-connects

```
RP/0/RSP0/CPU0:PE1#sh l2vpn xconnect

Tue Apr 24 03:40:36.619 UTC
Legend: ST = State, UP = Up, DN = Down, AD = Admin Down, UR = Unresolved,
SB = Standby, SR = Standby Ready

XConnect                  Segment 1                Segment 2
Group   Name       ST   Description       ST   Description            ST
--------------------    --------------------   -------------------------
R3R4    R3_to_R4   UP   Gi0/1/0/19        UP   150.2.2.2       304    UP
------------------------------------------------------------------------
RP/0/RSP0/CPU0:PE1#
```

There you go, that looks good. Now, can we PING between CE1 and CE2?

```
CE1#p 10.1.34.4

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.1.34.4, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/4 ms
```

Yup, PING works. Hmm, wonder what CDP looks like?

```
CE1#sh cdp nei
Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge
                  S - Switch, H - Host, I - IGMP, r - Repeater

Device ID     Local Intrfce    Holdtme   Capability   Platform   Port ID
CE2           Fas 0/0          160       R S I        2811       Fas 0/0
CE1#
```

Now if we look at OSPF:

```
CE1#sh ip ospf neighbor

Neighbor ID    Pri    State      Dead Time    Address      Interface
```

```
10.4.4.4        1   FULL/DR   00:00:33    10.1.34.4   FastEthernet0/0
```

All neighbored up! That means we should be able to PING between loopback interfaces:

```
CE1#p 10.4.4.4 so l0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.4.4.4, timeout is 2 seconds:
Packet sent with a source address of 10.1.3.3
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/4 ms
CE1#
```

24.NHRP (HSRP & VRRP)

Next Hop Resolution Protocol comes in two fashions on the IOS XR. The first is Cisco proprietary and called Hot-Standby Router Protocol or HSRP and the other is the industry standard called Virtual Router Redundancy protocol or VRRP.

This is something that many customers use in order to maintain the availability of a default gateway on the network. If your customer uses a static route to a next hop, you might be using this as well.

Like everything else with IOS XR, NHRP is handled a little differently. With IOS, you configure your standby commands under the interfaces; in IOS XR you use ROUTER HSRP or Router VRRP.

For this lab we will use interface Te0/1/0/0 and a subnet of 150.1.12.0/24.

First up R1:

```
RP/0/RSP0/CPU0:R1(config)#int tenGigE 0/1/0/0
RP/0/RSP0/CPU0:R1(config-if)#ip add 150.1.12.1/24
RP/0/RSP0/CPU0:R1(config-if)#no shut
RP/0/RSP0/CPU0:R1(config-if)#commit
RP/0/RSP0/CPU0:R1(config-if)#
```

Now R2:

```
RP/0/RSP0/CPU0:R2(config)#int tenGigE 0/1/0/0
RP/0/RSP0/CPU0:R2(config-if)#ip add 150.1.12.2/24
RP/0/RSP0/CPU0:R2(config-if)#no shut
RP/0/RSP0/CPU0:R2(config-if)#commit
```

Now we should test PING from R1 to R2:

```
RP/0/RSP0/CPU0:R1#ping 150.1.12.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.12.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/2 ms
RP/0/RSP0/CPU0:R1#
```

Good, we have connectivity.

HSRP is up first!

First thing, from config mode, enter router hsrp

```
RP/0/RSP0/CPU0:R1(config)#router hsrp
```

Now we tell it what interface we are configuring HSRP under

```
RP/0/RSP0/CPU0:R1(config-hsrp)#interface tenGigE 0/1/0/0
```

And then we configure our HSRP ID and associated information. Instead of using the standby command, we are using the HSRP command.

```
RP/0/RSP0/CPU0:R1(config-hsrp-if)#hsrp 100 ipv4 150.1.12.100
RP/0/RSP0/CPU0:R1(config-hsrp-if)#hsrp 100 priority 150
RP/0/RSP0/CPU0:R1(config-hsrp-if)#hsrp 100 preempt
RP/0/RSP0/CPU0:R1(config-hsrp-if)#hsrp 100 authentication cisco
RP/0/RSP0/CPU0:R1(config-hsrp-if)#exit
RP/0/RSP0/CPU0:R1(config-hsrp)#exit
RP/0/RSP0/CPU0:R1(config)#exit
```

And save your changes!

```
Uncommitted changes found, commit them before exiting(yes/no/cancel)?
[cancel]:yes
```

Ok, now to configure R2

```
RP/0/RSP0/CPU0:R2(config)#router hsrp
RP/0/RSP0/CPU0:R2(config-hsrp)#interface tenGigE 0/1/0/0
RP/0/RSP0/CPU0:R2(config-hsrp-if)#hsrp 100 ipv4 150.1.12.100
RP/0/RSP0/CPU0:R2(config-hsrp-if)#hsrp preempt
RP/0/RSP0/CPU0:R2(config-hsrp-if)#hsrp authentication cisco
RP/0/RSP0/CPU0:R2(config-hsrp-if)#comm
```

Back to R1 to validate HSRP

```
RP/0/RSP0/CPU0:R1#sh hsrp
                     P indicates configured to preempt.
                     |
Interface      Grp Pri P State   Active addr     Standby addr    Group addr
Te0/1/0/0      100 150 P Active  local           150.1.12.2      150.1.12.100
RP/0/RSP0/CPU0:R1#
```

There we go, we now have HSRP on R1 active with R2 as standby.

Time for some VRRP

One thing cool about VRRP, you don't have to burn an IP address just for the virtual. You can use an actual physical IP address of a router. If that router goes off-line, then the other router will just assume the IP address.

R1 up first, and we will use the R1 Te0/1/0/0 IP address for the virtual. First though, we need to remove HSRP and save the changes.

```
RP/0/RSP0/CPU0:R1(config)#no router hsrp
RP/0/RSP0/CPU0:R1(config)#commit
```

Ok, to configure VRRP the command is router vrrp

```
RP/0/RSP0/CPU0:R1(config)#router vrrp
```

Again, then you tell it what interface

```
RP/0/RSP0/CPU0:R1(config-vrrp)#interface tenGigE 0/1/0/0
```

Since VRRP likes IPV6, we need to use the address-family command

```
RP/0/RSP0/CPU0:R1(config-vrrp-if)#address-family ipv4
```

Then configure our VRRP ID

```
RP/0/RSP0/CPU0:R1(config-vrrp-address-family)#vrrp 1
```

Assign the IP, here I am using the same IP as our physical interface

```
RP/0/RSP0/CPU0:R1(config-vrrp-virtual-router)#address 150.1.12.1
RP/0/RSP0/CPU0:R1(config-vrrp-virtual-router)#text-authentication cisco
RP/0/RSP0/CPU0:R1(config-vrrp-virtual-router)#commit
RP/0/RSP0/CPU0:R2(config-vrrp-virtual-router)#
```

Now for R2, but this time we will decrease the priority so that R1 is the active router

```
RP/0/RSP0/CPU0:R2(config)#no router hsrp
RP/0/RSP0/CPU0:R2(config)#commit
RP/0/RSP0/CPU0:R2(config)#router vrrp
RP/0/RSP0/CPU0:R2(config-vrrp)#interface tenGigE 0/1/0/0
RP/0/RSP0/CPU0:R2(config-vrrp-if)#address-family ipv4
RP/0/RSP0/CPU0:R2(config-vrrp-address-family)#vrrp 1
RP/0/RSP0/CPU0:R2(config-vrrp-virtual-router)#address 150.1.12.1
RP/0/RSP0/CPU0:R2(config-vrrp-virtual-router)#text-authentication cisco
RP/0/RSP0/CPU0:R2(config-vrrp-virtual-router)#priority 50
RP/0/RSP0/CPU0:R2(config-vrrp-virtual-router)#commit
RP/0/RSP0/CPU0:R2(config-vrrp-virtual-router)#
RP/0/RSP0/CPU0:R2#
```

Now back to R1 to see the VRRP status:

```
RP/0/RSP0/CPU0:R1#show vrrp
IPv4 Virtual Routers:
                      A indicates IP address owner
                      | P indicates configured to preempt
                      | |
Interface   vrID Prio A P State    Master addr     VRouter addr
Te0/1/0/0      1  255 A P Master   local           150.1.12.1
IPv6 Virtual Routers:
                      A indicates IP address owner
                      | P indicates configured to preempt
                      | |
Interface   vrID Prio A P State    Master addr     VRouter addr
RP/0/RSP0/CPU0:R1#
```

Nice, R1 is the MASTER. Wonder what R2 says?

```
RP/0/RSP0/CPU0:R2#sh vrrp
IPv4 Virtual Routers:
                      A indicates IP address owner
                      | P indicates configured to preempt
                      | |
Interface   vrID Prio A P State    Master addr     VRouter addr
Te0/1/0/0      1   50   P Backup   150.1.12.1      150.1.12.1
IPv6 Virtual Routers:
                      A indicates IP address owner
                      | P indicates configured to preempt
                      | |
Interface   vrID Prio A P State    Master addr     VRouter addr
RP/0/RSP0/CPU0:R2#
```

It says it's the backup! Nice.

This page intentionally left black for notes.