jncip-dc-notes

Release

June 22, 2019

Table of Contents

Welcome to Tyler Christiansen's notes for the JNCIP-DC exam. These notes are open source. I would love to see additional collaboration and contribution from others who are either pursuing the exam or who are far more knowledgeable than myself. If you're interested in contributing, please see the *Contributing* section below.

These notes have been taken and organized in the order of the listed JNCIP-DC exam objectives. The primary reference has been the QFX5100 Series book, but other references such as the MX Series book and official Juniper documentation have been used. I've tried to remember to credit any sources I've had while taking notes, but I'm sure I've left sources off. If you're aware of a particular source for something, please let me know so that I can attribute credit.

Note Links to Amazon may be present. They are *not* affiliate links. I do not make any money if you follow them. I do not make any money for any link you follow on this site. There is no monetization whatsoever here. I'm only interested in publishing my JNCIP-DC notes as open source.

These are my personal notes – they're not guaranteed to be accurate or representative of the exam in any way, shape, or form. If you find something that is incorrect, please let me know via e-mail or GitHub. You can even submit a pull request to correct the error!

CHAPTER 1

Contributing

I'll take contributions however I can get them, but here are a few areas in which I'm particularly interested:

- Diagrams: I'm really terrible at making diagrams. I actually tried to to make a few for this project, but I became far too frustrated with the tools available to me (I don't have Visio). If you can contribute diagrams, it would be incredibly helpful.
- Technical Corrections: These are my notes for an exam; I'm not an authority on the technologies, and I could definitely be wrong. I don't have a technical reviewer. If you find technical errors, please let me know either via e-mail, a GitHub issue, or a pull request to correct the issue. All I ask is that for any technical correction, a source to the correct information be provided.
- Grammatical Correction: This may seem odd, but if there's a grammar issue, I'd appreciate a correction.
- Labs and Configurations: More labs and configuration examples are always better! I'm trying to provide examples, but it's sometimes difficult to capture. In some cases, I can't provide any labs or configuration snippets because it requires physical gear (such as Virtual Chassis) and I only have access to vQFX10k. So if you can contribute labs or configuration snippets, they would be greatly appreciated!

If you have any other contribution to make, it's 100% welcome. The best place to contribute and collaborate is the GitHub repository. Issues can be reported, and if you're familiar with GitHub and/or ReStructured Text (the markup language in which these notes are written), you can submit pull requests to contribute.

All contributions will be added to the *Contributors* page unless the contributor requests otherwise.

1.1 Data Center Deployment and Management

This blueprint item covers a number of topics:

- *Zero-Touch Provisioning*
- *High Availability*
- *Monitoring*
- *Analytics*

Unfortunately, the blueprint doesn't indicate to what level those items should be known, and all of those options seem to require physical hardware, which I do not have. So this guide will only be theory until

someone contributes configuration sections.

1.1.1 Zero-Touch Provisioning

ZTP is a process for upgrading software and applying configuration automatically on first boot. This is accomplished by using a stadard DHCP server with a little bit of extra configuration so that each `DISCOVER` is associated with the correct switch so that the correct configuration can be retrieved.

When configuring the DHCP server, the following DHCP options can be used:

- `12`: Switch Hostname
- `42`: NTP Server
- `43.00`: Software image filename
- `43.01`: Configuration file filename
- `43.02`: Symbolic link flag for filename
- `43.03`: Transfer mode (HTTP, FTP, TFTP)
- `43.04`: Alternate software image filename
- `66`: DNS FQDN for the HTTP/FTP/TFTP server
- `150`: IP Address for the HTTP/FTP/TFTP server

The reason that both `44.00` and `44.04` can both be used is that some DHCP servers may not support `44.00`. If both options are defined, `44.00` takes precedence.

When setting the `44.00` or `44.04` location to a symlink, you also need to set the `44.02` option to the value `symlink` so that the system knows that the image is a symlink.

`44.03` lets you specify whether the file transfer method will be FTP, TFTP, or HTTP. The default is TFTP.

If DHCP options `66` and `150` are both specified, then `150` takes precedence.

When the configuration file starts with a shebang (`#!`), Junos will attempt to run the file as a script. This means you can use one Python or shell script to dynamically configure your switches instead of hand-crafting a configuration file for each switch.

By default, a QFX5100 will attempt to perform ZTP through both its management interface and its revenue ports. This means you can perform ZTP through a dedicated OOB management network (recommended) or in-band.

After the switch knows the details of its ZTP process, it first downloads the required configuration file and then the required software image (if necessary). If a software image was downloaded, then the switch performs the software upgrade. Finally, it applies the configuration file that was downloaded.

ZTP can also be performed by Junos Space with Network Director. When this is done, the switch is automatically added to Network Director for future management.

For more information on ZTP (including configuration examples), see the following blogs/articles:

- Zero Touch Provisioning: How to Build a Network Without Touching Anything
- Zero Touch Provisioning on Juniper devices using Linux

You can also read Chapter 6 of the QFX5100 Series book from O'Reilly [1].

1.1.2 High Availability

The QFX5100 has a virtualized control plane. There is a Linux host running KVM, and Junos runs as a

1 Juniper QFX5100 Series

VM. The hypervisor can run up to four VMs: two of them are reserved for Junos, one is a guest of the operator's choice, and the last is reserved.

Each Junos VM has four management interfaces. The first two, `em0` and `em1`, map to the physical management ports on the switch. The third, `em2`, is used for communicating with the hypervisor. The last interface, `em3`, is used when performing a Topology-independent In-Service Software Upgrade, or TISSU. The second Junos VM is only created during a TISSU. In order to perform a TISSU, NSR, NSB, and GRES must be configured.

The high-level process for TISSU is as follows [2]:

1: Create the backup Junos VM running the new version requested 2: Synchronize state between the Junos VMs using `ksyncd` 3: Makes the new VM the master RE 4: Renames the slot ID of the new VM from `1` to `0` 5: The former master Junos VM is shut down

When performing a TISSU, keep the following in mind:

- Downgrades and rollbacks are not supported
- TISSU should not be used when transitioning between different base images (e.g., `standard` to `enhanced-automation`)
- The CLI is inaccessible during a TISSU
- Log files are located in `/var/log/vjunos-log.tgz`
- BFD timers need to be >= 1 second [3]
- `system internet-options no-tcp-reset drop-all-tcp` must not be configured [3]

A configuration sample for enabling NSR, NSB, and GRES is below.

```
system {
    commit synchronize;
}
chassis {
    redundancy {
        graceful-switchover;
    }
}
routing-options {
    nonstop-routing;
}
protocols {
    layer2-control {
        nonstop-bridging;
    }
}
```

Once the configuration is in place, you can perform a TISSU with the `request system software in-service-upgrade <path-to-image>` command.

For more information, see Chapter 2 of the Juniper QFX5100 Series book [1].

1.1.3 Monitoring

Junos supports streaming telemetry as well as more traditional methods of monitoring such as SNMP.

TODO: Add configuration for SNMPv2c, SNMPv3, and Streaming Telemetry.

2 Understanding In-Service Software Upgrade (ISSU)
3 Performing an In-Service Software Upgrade (ISSU) with Non-Stop Routing

1.1.4 Analytics

The QFX5100 supports two methods of analytics: sFlow and Enhanced Analytics. These are described below, but it's important to understand that they should be used together as neither provides the entire picture on its own.

sFlow

The QFX5100 supports sFlow, which samples every `n` packets. This sampled data is exported every 1500 bytes or every 250ms. Any alerting on this data must be performed off-box with the sFlow collector. On the QFX5100, only switchports can be sampled. Layer 3 interfaces cannot be sampled. The first 128 bytes of the packet are sampled, and this includes information such as the source and destination MACs, IPs, and Ports. Higher sampling rates require more processing power.

To combat this on high-traffic switches, the QFX5100 can dynamically adjust sample rates based on interface traffic. This is known as adaptive sampling. An agent checks the interfaces every 5 seconds. A list of the top five interfaces is created. An algorithm reduces the load by half for the top five interfaces and allocates those samples to lower traffic interfaces.

An sFlow configuration is shown below.

```
protocols {
    sflow {
        polling-interval 10;
        sample-rate {
            ingress 50;
            egress 50;
        }
        collector 10.0.0.30 {
            udp-port 9000;
        }
        interfaces xe-0/0/0.0;
    }
}
```

Note The `polling-interval` tells Junos how frequently, in seconds, to poll for data; the `sampling-rate` tells Junos how many packets to sample.

Juniper Enhanced Analytics

The QFX5100 also supports Juniper Enhanced Analytics. This system can poll as frequently as every 8ms, and data is exported as soon as it is collected. You can set thresholds on-box down to 1ns. Enhanced Analytics can monitor:

- Traffic statistics
- Queue depth
- Latency
- Jitter

This data can be streamed using the following formats:

- Protobuf
- JSON
- CSV

- TSV

Enhanced Analytics is performed by two systems: the Analytics Daemon and the Analytics Manager.

The Analytics Daemon (`analyticsd`) collects information from the Analytics Manager's ring buffers and exports it to collectors.

The Analytics Manager runs in the PFE and collects the data that is placed into ring buffers for `analyticsd` to collect.

TODO: Add configuration examples for Enhanced Analytics.

For more information, see Chapter 9 of the Juniper QFX5100 Series book [1].

1.2 Multi-Chassis LAG (MC-LAG)

Multi-Chassis LAG is a technology that allows two independent systems to present themselves as a single system while still operating indepedently. This is in contrast to Virtual Chassis, which makes multiple independent systems operate as a single system. At the highest level, MC-LAG works by forcing the two upstream devices to use the same LACP System ID, which makes the downstream device think that the upstream devices are just one device.

Before diving into the specifics of an MC-LAG configuration, it is critical to know which settings *must match* and which settings *must be unique* between peers. The following section describes which must match and which must be unique [4].

Must match:

- LACP System ID
- LACP Admin Key
- MCAE ID
- MCAE Mode
- VLANs
- Redundancy Group ID [5]
- Service ID [5]

Must be unique:

- MCAE Chassis ID
- MCAE Status Control
- Local ICCP IP
- Peer ICCP IP
- MC-LAG Protection

Finally, a diagram is worth a thousand words. If you have the Juniper MX Series book from O'Reilly [6], I can highly recommend Figure 9-4 as a reference. I won't reproduce it here because that would be incredibly rude and inconsiderate, so please pick up that book if you can. Chapter 9, MC-LAG, is definitely worth it! Just in case you have a Safari subscription, here's a link straight to the diagram. But again, definitely read the entire chapter. It's absolutely worth it.

4 Juniper Ambassador's Cookbook 2019, Recipe #5
5 Juniper MX Series, Chapter 9, ICCP Hierarchy Section
6 Juniper MX Series, Second Edition

Note I'm hoping the authors don't mind me including a link directly to the diagram in the Safari Online book. But if you're one of the authors or the publisher and want that removed, *please* reach out to me and let me know and I will gladly remove it.

1.2.1 Terminology

This section explains some of the terminology and configuration elements listed above. [5]

Service ID

The Service ID must match between two ICCP peers. The Service ID exists for use cases where MC-LAG is used in more than one routing instance. This ID is global.

Redundancy Group ID

The Redundancy Group ID must match between two ICCP peers. It contains a grouping of MC-LAG bundles and their associated VLANs. It is useful because it allows the ICCP peers to send an update once instead of sending an update for each MC-LAG. For example, when a MAC address is learned, the MAC only needs to be sent to the ICCP peer one time for that redundancy group instead of multiple times for each MC-LAG in the group. This ID is not required to be global, but can encompass one or more MC-LAG bundles.

Note In vQFX 18.1R1.9, on which this document is based, there can only be one Redundancy Group ID configured. Although a list is supported, the `commit check` will fail with an error if more than one is provided. This is probably because only one bridge domain is supported, and a Redundancy Group is a broadcast domain. In the MX router, multiple Redundancy Group IDs are supported.

MCAE ID

The Multi-Chassis Aggregated Ethernet Identifier must match between peers. This number must be unique per MC-LAG.

MCAE Chassis ID

The MCAE Chassis ID uniquely identifies a chassis. Therefore, this ID must be unique between ICCP peers.

LACP System ID

This ID is used by LACP to identify the system to which a remote belongs. This ID must match between peers. It is used to "trick" the remote LACP peer into thinking it is talking to only one system.

LACP Admin Key

This is used in conjunction with the LACP System ID to uniquely identify an LACP peer. It must match between peers.

MCAE Mode

`active-active` or `active-standby`. Must match between peers.

Note Only `active-active` is supported on the QFX and EX Series. However, both `active-active` and `active-standby` are supported on the MX series with MPCs. `active-active` with a DPC on an MX is not supported; you must use `active-standby` in those scenarios. [??]

?? Understanding Multichassis Link Aggregation Groups

Status Control

Controls whether the node will be the active or standby node if ICCP fails. Only two options exist: `active` or `standby`. One node must be `active` and the other must be `standby`. The `standby` node will change its *LACP System ID*.

MC-LAG Protection

Multi-chassis Link Protection ensures the appropriate behavior of the node configured as *Status Control* `standby` when the ICL goes down. For example, if the ICL goes down, should the `standby` node change its *LACP System ID*, thereby taking its MC-LAG link down, or should it leave it as-is, making it actively receive traffic? Link Protection uses an out-of-band mechanism to determine if the remote ICCP peer is still up or not. If the ICL is down but the remote ICCP peer is still up, then the `standby` node will change its *LACP System ID*. However, if the ICL is down and the remote ICCP peer is also down, then the `standby` node will *not* change its *LACP System ID*. This will ensure that there is no outage.

1.2.2 Guidelines

The follow sections describe high-level, general guidelines when deploying MC-LAG. None of these recommendations are hard-and-fast rules, but they should be taken into consideration with any MC-LAG deployment.

Inter-Chassis Link Redundancy

Although not required, it is strongly recommended that the ICL consist of at least two links. This ICL link needs to have VLANs trunked across it. It needs to have the downstream VLANs trunked plus the ICCP VLAN. The reason for trunking the downstream VLANs is in case one of the MC-LAG links goes down and traffic needs to go between the switches.

Compensating for Lack of LACP support

Generally speaking, your downstream LACP bundle will be a single-interface LAG per upstream device. If the downstream device does not support LACP, you can use the `force-up` flag. However, if the downstream device *does* support LACP, it is strongly recommended to use LACP and leave the `force-up` option off.

Layer 3 Connectivity

For simple layer 3 connectivity, such as when the downstream device is a server, both of the upstream MC-LAG peers can have their gateways configured with the same IP address when `mcae-mac-synchronization` is enabled on the VLAN. Unlike VRRP, this will keep traffic local to the switch that receives the packet. This works by allowing one upstream switch to respond with the peer switch's MAC address. The end result is that each upstream switch treats a packet as if it is its own. `mcae-mac-synchronization` in this scenario is required because, without it, ARP requests are not sniffed or replicated by the peers.

Note DHCP Relay is not supported with `mcae-mac-synchronization`. If a DHCP Relay is required, you must use VRRP.

When routing protocols are required with an MC-LAG, `mcae-mac-synchronization` is no longer an option. This is because the MAC synchronization option works similarly to an anycast service, but routing protocols require 1:1 direct relationships. For this reason, when protocol adjacency is required (such as OSPF between the aggregation and core layers), VRRP is required. In these instances, the devices should peer to the VRRP VIP, not the real IP. To compensate for potential issues, a static ARP address for the *remote* peer's IRB MAC and real IP should be configured.

Spanning Tree Protocol

Although the goal of MC-LAG is to remove the need for a Spanning Tree Protocol, it is highly recommended that STP is enabled to prevent loops caused by miswiring as well as to prevent unintentional propagation of BPDUs.

When configuring STP, disable STP on the ICL. This is because STP could cause the traffic on the ICL to be blocked, thereby breaking the MC-LAGs. Configure all MC-LAG interfaces as `edge`. A downstream device should not be able to cause a loop in a properly designed network. Turn on `bpdu-block-on-edge`. This is to prevent malicious or unintentional BPDU propagation.

Note Do *not* configure MSTP or VSTP. This can cause loops when not configured appropriately on all devices, including downstream devices. [??]

1.2.3 Configuration

In this example, there are four vQFX switches: vQFX-1 through vQFX-4. vQFX-1 and vQFX-2 are our MC-LAG head-end devices. Their ICL is `ae0`, which consists of members `xe-0/0/0` and `xe-0/0/1`.

They run an MC-LAG down to vQFX-3 on `ae1` and to vQFX-4 on `ae2`. `ae1` consists of `xe-0/0/2` on vQFX-1 and `xe-0/0/4` on vQFX-2. `ae2` is made up of `xe-0/0/3` on vQFX-1 and `xe-0/0/5` on vQFX-2. The interface numbers are the same on vQFX-3 and vQFX-4 as their upstream devices with the exception of the bundle interface. On vQFX-3 and vQFX-4, that is `ae0`. You can see this below in the `show lldp neighbors` output for each device:

```
root@vQFX-1> show lldp neighbors
Local Interface    Parent Interface    Chassis Id          Port info          System
Name
xe-0/0/0           ae0                 02:05:86:71:68:00   xe-0/0/0           vQFX-2
xe-0/0/1           ae0                 02:05:86:71:68:00   xe-0/0/1           vQFX-2
xe-0/0/2           ae1                 02:05:86:71:bc:00   xe-0/0/2           vQFX-3
xe-0/0/3           ae2                 02:05:86:71:c5:00   xe-0/0/3           vQFX-4
{master:0}

root@vQFX-2> show lldp neighbors
Local Interface    Parent Interface    Chassis Id          Port info          System
Name
xe-0/0/4           ae1                 02:05:86:71:bc:00   xe-0/0/4           vQFX-3
xe-0/0/5           ae2                 02:05:86:71:c5:00   xe-0/0/5           vQFX-4
xe-0/0/1           ae0                 02:05:86:71:fa:00   xe-0/0/1           vQFX-1
xe-0/0/0           ae0                 02:05:86:71:fa:00   xe-0/0/0           vQFX-1

{master:0}

root@vQFX-3> show lldp neighbors
Local Interface    Parent Interface    Chassis Id          Port info          System
Name
xe-0/0/4           ae0                 02:05:86:71:68:00   xe-0/0/4           vQFX-2
xe-0/0/2           ae0                 02:05:86:71:fa:00   xe-0/0/2           vQFX-1

{master:0}

root@vQFX-4> show lldp neighbors
Local Interface    Parent Interface    Chassis Id          Port info          System
Name
xe-0/0/5           ae0                 02:05:86:71:68:00   xe-0/0/5           vQFX-2
xe-0/0/3           ae0                 02:05:86:71:fa:00   xe-0/0/3           vQFX-1
```

```
{master:0}
```

The MC-LAG to vQFX-3 provides normal layer 3 gateway services. We use MAC address synchronization here. The MC-LAG toward vQFX-4, however, runs OSPF. For this, we remove MAC address synchronization and implement VRRP with static ARP bindings. See *Layer 3 Connectivity* for more details about this.

The final test will be pinging between vQFX-3 and vQFX-4's `ae0` interfaces.

Note I don't have Visio or Omnigraffle, and I found making diagrams in anything else highly frustrating. So for now, there are no diagrams. :(However, if you'd like to contribute some, they would be greatly appreciated!

The very first thing to do with any LAG in Junos is to set the number of LAG interfaces:

```
# qfx1
chassis {
    aggregated-devices {
        ethernet {
            device-count 3;
        }
    }
}
```

```
# qfx2
chassis {
    aggregated-devices {
        ethernet {
            device-count 3;
        }
    }
}
```

```
# vQFX-3
chassis {
    aggregated-devices {
        ethernet {
            device-count 1;
        }
    }
}
```

```
# vQFX-4
chassis {
    aggregated-devices {
        ethernet {
            device-count 1;
        }
    }
}
```

Next, we configure the *Service ID* on the upstream devices:

```
# qfx1
switch-options {
    service-id 1;
}
```

```
# qfx2
switch-options {
    service-id 1;
}
```

Next, we'll configure the ICL. For redundancy, we'll configure the ICL as an LACP bundle:

```
# qfx1
interfaces {
    xe-0/0/0 {
        ether-options {
            802.3ad ae0;
        }
    }
    xe-0/0/1 {
        ether-options {
            802.3ad ae0;
        }
    }
    ae0 {
        aggregated-ether-options {
            lacp {
                active;
                periodic slow;
            }
        }
        unit 0 {
            family ethernet-switching {
                interface-mode trunk;
                vlan {
                    members all;
                }
            }
        }
    }
}
```

```
# qfx2
interfaces {
    xe-0/0/0 {
        ether-options {
            802.3ad ae0;
        }
    }
    xe-0/0/1 {
        ether-options {
            802.3ad ae0;
        }
    }
    ae0 {
        aggregated-ether-options {
            lacp {
                active;
                periodic slow;
            }
        }
        unit 0 {
            family ethernet-switching {
```

```
                interface-mode trunk;
                vlan {
                    members all;
                }
            }
        }
    }
}
```

Note If you have a device with different line cards/slots/ASICs, consider spreading your bundle members across them for greater resiliency.

Now that the ICL has its layer 1 and layer 2 configuration done, we need to ensure it can route packets for ICCP. Since ICCP only requires TCP/IP connectivity to establish, this could be done between loopback interfaces. However, for the example, we'll just use an IRB interface.

```
# qfx1
interfaces {
    irb {
        unit 100 {
            family inet {
                address 10.0.0.0/31;
            }
        }
    }
}
vlans {
    v100 {
        vlan-id 100;
        l3-interface irb.100;
    }
}
```

```
# qfx2
interfaces {
    irb {
        unit 100 {
            family inet {
                address 10.0.0.1/31;
            }
        }
    }
}
vlans {
    v100 {
        vlan-id 100;
        l3-interface irb.100;
    }
}
```

Next, we need to configure ICCP:

```
# qfx1
protocols {
    iccp {
        local-ip-addr 10.0.0.0;
        peer 10.0.0.1 {
```

```
            session-establishment-hold-time 50;
            redundancy-group-id-list 100;
            liveness-detection {
                minimum-receive-interval 300;
                transmit-interval {
                    minimum-interval 300;
                }
            }
        }
    }
}
```

```
# qfx2
protocols {
    iccp {
        local-ip-addr 10.0.0.1;
        peer 10.0.0.0 {
            session-establishment-hold-time 50;
            redundancy-group-id-list 100;
            liveness-detection {
                minimum-receive-interval 300;
                transmit-interval {
                    minimum-interval 300;
                }
            }
        }
    }
}
```

Note that we're configuring BFD for faster failure detection in this example. By default, this will create a new BFD session that runs on the control plane. However, if your platform supports it and you are not running ICCP via loopback interfaces (and ICCP is using IPs that are directly connected), you can push this down to the PFE with the `set protocols iccp peer <ip> liveness-detection single-hop` command.

Note In production, a best practice would be to configure ICCP between the loopback interfaces to ensure the ICCP session stays up, even if the ICL is down.

At this point, everything has been done such that ICCP should come up. This can be verified with the `show iccp status` command:

```
root@vQFX-1> show iccp

Redundancy Group Information for peer 10.0.0.1
  TCP Connection       : Established
  Liveliness Detection : Up
  Redundancy Group ID          Status
    100                         Up

Client Application: lacpd
  Redundancy Group IDs Joined: None

Client Application: MCSNOOPD
  Redundancy Group IDs Joined: None

Client Application: l2ald_iccpd_client
  Redundancy Group IDs Joined: None
```

```
{master:0}

root@vQFX-2> show iccp

Redundancy Group Information for peer 10.0.0.0
  TCP Connection       : Established
  Liveliness Detection : Up
  Redundancy Group ID          Status
    100                         Up

Client Application: lacpd
  Redundancy Group IDs Joined: None

Client Application: MCSNOOPD
  Redundancy Group IDs Joined: None

Client Application: l2ald_iccpd_client
  Redundancy Group IDs Joined: None

{master:0}
```

Before going on, we have two minor things to finish up: configuring *MC-LAG Protection* and configuring *Spanning Tree Protocol*:

```
# qfx1
multi-chassis {
    multi-chassis-protection 10.0.0.1 {
        interface ae0;
    }
}
protocols {
    rstp {
        interface ae0 {
            disable;
        }
        interface all {
            mode point-to-point;
        }
        bpdu-block-on-edge;
    }
}
```

```
# qfx2
multi-chassis {
    multi-chassis-protection 10.0.0.0 {
        interface ae0;
    }
}
protocols {
    rstp {
        interface ae0 {
            disable;
        }
        interface all {
            mode point-to-point;
        }
        bpdu-block-on-edge;
    }
```

```
}
```

All that's left to finish our MC-LAG configuration is to start bringing up MCAEs! First, we'll work on the MC-LAG toward vQFX-3.

```
# qfx1
interfaces {
    xe-0/0/2 {
        ether-options {
            802.3ad ae1;
        }
    }
    ae1 {
        aggregated-ether-options {
            lacp {
                active;
                periodic slow;
                system-id 00:00:00:11:11:11;
                admin-key 1;
            }
            mc-ae {
                mc-ae-id 1;
                redundancy-group 100;
                chassis-id 0;
                mode active-active;
                status-control active;
                init-delay-time 15;
            }
        }
        unit 0 {
            family ethernet-switching {
                interface-mode access;
                vlan {
                    members v1000;
                }
            }
        }
    }
    irb {
        unit 1000 {
            family inet {
                address 192.168.100.0/31;
            }
        }
}
protocols {
    rstp {
        interface ae1 {
            edge;
        }
}
vlans {
    v1000 {
        vlan-id 1000;
        l3-interface irb.1000;
        mcae-mac-synchronize;
    }
}
```

```
# qfx2
interfaces {
    xe-0/0/4 {
        ether-options {
            802.3ad ae1;
        }
    }
    ae1 {
        aggregated-ether-options {
            lacp {
                active;
                periodic slow;
                system-id 00:00:00:11:11:11;
                admin-key 1;
            }
            mc-ae {
                mc-ae-id 1;
                redundancy-group 100;
                chassis-id 1;
                mode active-active;
                status-control standby;
                init-delay-time 15;
            }
        }
        unit 0 {
            family ethernet-switching {
                interface-mode access;
                vlan {
                    members v1000;
                }
            }
        }
    }
    irb {
        unit 1000 {
            family inet {
                address 192.168.100.0/31;
            }
        }
}
protocols {
    rstp {
        interface ae1 {
            edge;
        }
}
vlans {
    v1000 {
        vlan-id 1000;
        l3-interface irb.1000;
        mcae-mac-synchronize;
    }
}
```

The lines highlighted are specific to MC-LAG; everything else is just normal LACP configuration. Note that on both vQFX-1 and vQFX-2 the *LACP System ID*, *LACP Admin Key*, *MCAE ID*, *Redundancy Group ID*, and *MCAE Mode* are all the same.

However, the *MCAE Chassis ID* and *Status Control* are unique between the two devices.

Next we'll configure vQFX-3, which is just standard LACP configuration:

```
# vQFX-3
interfaces {
    xe-0/0/2 {
        ether-options {
            802.3ad ae0;
        }
    }
    xe-0/0/4 {
        ether-options {
            802.3ad ae0;
        }
    }
    ae0 {
        aggregated-ether-options {
            lacp {
                active;
                periodic slow;
            }
        }
        unit 0 {
            family inet {
                address 192.168.100.1/31;
            }
        }
    }
}
routing-options {
    static {
        route 0.0.0.0/0 next-hop 192.168.100.0;
    }
    router-id 3.3.3.3;
}
```

We add a static default route so that we can ping vQFX-4 later once its configuration is complete. But first, let's make sure we can ping our gateway!

```
root@vQFX-3> ping 192.168.100.0 rapid count 5
PING 192.168.100.0 (192.168.100.0): 56 data bytes
!!!!!
--- 192.168.100.0 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 306.100/365.522/380.486/29.711 ms

{master:0}
```

Success! Let's move on to configuring the MCAE toward vQFX-4, which is slightly more complicated due to running OSPF. We'll split up the layer 2 configuation and the layer 3 configuration to make it a little easier to digest.

First, let's configure the upstream devices:

```
# qfx1
interfaces {
    xe-0/0/3 {
        ether-options {
            802.3ad ae2;
        }
```

```
    }
    ae2 {
        aggregated-ether-options {
            lacp {
                active;
                periodic slow;
                system-id 00:00:00:22:22:22;
                admin-key 2;
            }
            mc-ae {
                mc-ae-id 2;
                redundancy-group 100;
                chassis-id 1;
                mode active-active;
                status-control standby;
                init-delay-time 15;
            }
        }
        unit 0 {
            family ethernet-switching {
                interface-mode access;
                vlan {
                    members v2000;
                }
            }
        }
    }
}
protocols {
    rstp {
        interface ae2 {
            edge;
        }
}
vlans {
    v2000 {
        vlan-id 2000;
        l3-interface irb.2000;
    }
}
```

```
# qfx2
interfaces {
    xe-0/0/5 {
        ether-options {
            802.3ad ae2;
        }
    }
    ae2 {
        aggregated-ether-options {
            lacp {
                active;
                periodic slow;
                system-id 00:00:00:22:22:22;
                admin-key 2;
            }
            mc-ae {
                mc-ae-id 2;
                redundancy-group 100;
```

```
                chassis-id 0;
                mode active-active;
                status-control active;
                init-delay-time 15;
            }
        }
        unit 0 {
            family ethernet-switching {
                interface-mode access;
                vlan {
                    members v2000;
                }
            }
        }
    }
}
protocols {
    rstp {
        interface ae2 {
            edge;
        }
}
vlans {
    v2000 {
        vlan-id 2000;
        l3-interface irb.2000;
    }
}
```

The MC-LAG-specific configurations are highlighted above.

Note Notice in this example that we did not configure `mcae-mac-synchronization` on the VLAN. This is because we will be using VRRP due to the OSPF requirement, and these two configurations are mutually exclusive on the QFX series switches.

Now let's examine the layer 3 configuration on vQFX-1 and vQFX-2:

```
# qfx1
interfaces {
    irb {
        unit 2000 {
            family inet {
                address 192.168.200.2/29 {
                    arp 192.168.200.3 l2-interface ae0.0 mac 02:05:86:71:93:00;
                    vrrp-group 1 {
                        virtual-address 192.168.200.1;
                        priority 200;
                        accept-data;
                    }
                }
            }
        }
    }
}
routing-options {
    router-id 1.1.1.1;
}
protocols {
```

```
    ospf {
        area 0.0.0.0 {
            interface irb.2000;
            interface irb.1000 {
                passive;
            }
        }
    }
}
```

```
# qfx2
interfaces {
    irb {
        unit 2000 {
            family inet {
                address 192.168.200.3/29 {
                    arp 192.168.200.2 l2-interface ae0.0 mac 02:05:86:71:62:00;
                    vrrp-group 1 {
                        virtual-address 192.168.200.1;
                        priority 100;
                        accept-data;
                    }
                }
            }
        }
    }
}
routing-options {
    router-id 2.2.2.2;
}
protocols {
    ospf {
        area 0.0.0.0 {
            interface irb.2000;
            interface irb.1000 {
                passive;
            }
        }
    }
}
```

The only line that is different compared to standard layer 3 configuration is the static ARP entry. Recall from *Layer 3 Connectivity* that we need a static ARP entry pointing to the remote device's IRB MAC address via the ICL. That's all this configuration line does: create a static ARP entry for the real IP of the remote switch with its IRB MAC via the ICL.

We can now examine vQFX-4's configuration:

```
# vQFX-4
interfaces {
    xe-0/0/3 {
        ether-options {
            802.3ad ae0;
        }
    }
    xe-0/0/5 {
        ether-options {
            802.3ad ae0;
        }
```

```
    }
    ae0 {
        aggregated-ether-options {
            lacp {
                active;
                periodic slow;
            }
        }
        unit 0 {
            family inet {
                address 192.168.200.4/29;
            }
        }
    }
}
routing-options {
    router-id 4.4.4.4;
}
protocols {
    ospf {
        area 0.0.0.0 {
            interface ae0.0;
        }
    }
}
```

This is just standard configuration. No special tricks required! Let's check OSPF:

```
root@vQFX-1> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
192.168.200.4    irb.2000               Full      4.4.4.4          128    30
192.168.200.3    irb.2000               Full      2.2.2.2          128    36

{master:0}

root@vQFX-2> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
192.168.200.4    irb.2000               Full      4.4.4.4          128    32
192.168.200.2    irb.2000               Full      1.1.1.1          128    34

{master:0}

root@vQFX-4> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
192.168.200.2    ae0.0                  Full      1.1.1.1          128    33
192.168.200.3    ae0.0                  Full      2.2.2.2          128    39

{master:0}
```

Finally, let's ensure we can ping between vQFX-3 and vQFX-4:

```
root@vQFX-3> ping 192.168.200.4 rapid count 5
PING 192.168.200.4 (192.168.200.4): 56 data bytes
!!!!!
--- 192.168.200.4 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 239.258/425.293/601.453/129.111 ms

{master:0}
```

```
root@vQFX-4> ping 192.168.100.1 rapid count 5
PING 192.168.100.1 (192.168.100.1): 56 data bytes
!!!!!
--- 192.168.100.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 114.584/234.628/287.023/67.643 ms

{master:0}
```

And that's it! MC-LAG configuration for simple gateway and advanced layer 3 routing is working.

1.2.4 Conclusion

We've brushed the surface of MC-LAG, hopefully enough for the JNCIP-DC. For more detailed information, check out Juniper MX Series, Second Edition [6].

The following blogs were helpful in compiling these notes:

- MC-LAG on vQFX (EVE-NG)
- MC-LAG Lab – Basic L2 Connectivity
- MC-LAG Lab – Advanced IRB FunctinNality

1.3 Layer 2 Fabrics

This blueprint item primarily covers the following topics:

- *Virtual Chassis*
- *Virtual Chassis Fabric*

Note Both *Virtual Chassis* and *Virtual Chassis Fabric* are hardware-based architectures. Unfortunately, I do not have access to real hardware; all of my labbing is based on the vQFX0000. This means that although there may be configurations presented, I don't currently have a way of validating. If you have access to physical gear and can validate the configurations (or present more interesting topologies/configurations), I would love to see them contributed!

1.3.1 Virtual Chassis

The EX4300, QFX3500, QFX3600, and QFX5100 can form a mixed mode Virtual Chassis. If you are familiar with mixed mode Virtual Chassis from the Enterprise track (EX4200/EX4500/EX4550), the concept is very similar. Up to 10 switches are supported in a stack, although some switches, such as the QFX5100-96S, may not be supported.

The first step in creating a mixed mode Virtual Chassis is to tell each individual switch that it will be participating in a mixed mode VC with the `request virtual-chassis mode mixed reboot` command. However, when building a new stack, the `request virtual-chassis mode mixed all-members reboot` command can be used to set all members in the stack to mixed mode at the same time.

Note When operating in a mixed mode Virtual Chassis, the scaling numbers are reduced to the lowest common denominator. This severely limits the scalability of a deployment. The lowest common denomi-

nator is the lowest scaling factor of the smallest possible PFE. This means that if you have a mixed Virtual Chassis of QFX3500 and QFX5100, even if there is no EX4300, each device will still be limited to the maximum scale of an EX4300.

The benefits of a mixed mode Virtual Chassis are the same as those of a regular Virtual Chassis:

- Redundant Routing Engines
- NSR and NSB
- One control plane for multiple data planes
- Potential elimination of xSTP

By default, the 40G QSFP+ ports on an EX4300 are enabled for Virtual Chassis; however, on the QFX5100, these ports are disabled for Virtual Chassis. When a mixed mode VC contains QFX5100 switches, only the QFX5100 can become an RE. As with many protocols consisting of primary and secondary nodes, the VC mastership process follows an election process. The first tie-breaker is priority, which is 128 by default. Higher is better. If the priority values are the same, the next factor to consider is which node was the master prior to a reboot. Next, the member with the highest uptime; however, the difference in uptime must be more than 60 seconds. Finally, all else being equal (and difference in uptime being less than 60 seconds), the member with the *lowest* MAC address will be elected as the master. The backup is elected according to the same criteria.

When implementing Virtual Chassis, you can use the `virtual-chassis auto-sw-upgrade` configuration to automatically upgrade members with the `LC (Line Card)` state when their software version does not match that of the Master RE. Additionally, Split Detection can be disabled with the `virtual-chassis no-split-detection` configurtation. However, this should only be done when there are only two members in the Virtual Chassis.

On a QFX5100, you will need to set ports as Virtual Chassis Ports with the `request virtual-chassis vc-port set pic-slot 0 port 48` operational command.

Virtual Chassis supports a special deviation of ISSU called NSSU: Non-Stop Software Upgrade. For NSSU to work, the physical topology must be a ring – it cannot be a braid. The master and backup must be adjacent; this means that the roles of each switch must be deterministic. For this reason, only pre-provisioned Virtual Chassis is supported. Additionally, both NSR and GRES must be configured; NSB is optional. To initiate an NSSU, issue the `request system software nonstop-upgrade [<path_platform1> <path_platform2>]` command.

1.3.2 Virtual Chassis Fabric

Virtual Chassis Fabric is an extension of Virtual Chassis. It works with QFX3500, QFX3600, QFX5100, and EX4300 series switches. New switches added to a VCF are automatically discovered and brought online.

Note The node limit is unclear. In Chapter 5 of the O'Reilly QFX5100 Series book [??], a passage indicates that the maximum number of switches is 32; however, recent Juniper documentation publications [??] indicate that the limit is 20. Because the documentation is more recent than the published book, even though there is no confirmed and published errata [??], readers should assume a limit of 20 nodes.

Like *Virtual Chassis*, VCF uses IS-IS between switches. The links between switches, however, come up as "Smart Trunks." This is part of what enables VCF to perform unequal cost multipath load balancing in certain designs and failure scenarios. The other technology that enables unueqal cost multipath is Adaptive Load Balancing. ALB hashes TCP flowlets to different links.

?? Juniper QFX5100 Series
?? Planning a Virtual Chassis Fabric Deployment
?? Errata for Juniper QFX5100 Series

These flowlets are tracked in a hash bucket table. This table can hold hundreds of thousands of entries – enough to prevent "elephant flows" from overloading a given link. When the flowlet egresses the switch, the hash table is updated with a timestamp and the link via which it egressed. When a new packet for the same flowlet egresses, it is checked against an expiration or inactivity timer. If the time since the last packet was seen is greater than the inactivity timer, then the flowlet is hashed to a new uplink. The egress link selection is also based on a moving average of the load and queue depth on each interface.

ALB is disabled by default; to enable it in a VCF, use the `set fabric-load-balance flowlet` configuration command.

Not all switches can be spine switches, but all switches can be leaf switches. A general rule of thumb is that a fiber-based QFX5100 can be a leaf switch or a spine switch; any other switch can only be a leaf switch.

Provisioning Options

When configuring a VCF, you have three options: auto-provisioned, pre-provisioned, and non-provisioned. Each has its own benefits and drawbacks; auto-provisioned is less secure, while the non-provisioned mode is more configuration-intensive and less predictable.

With an auto-provisioned VCF, you must specify the role and serial number for each spine switch; the leaf switches are automatically added. The Virtual Chassis Ports are automatically discovered and added.

With a pre-provisioned VCF, you specify each spine *and* leaf member. Virtual Chassis Ports are also automatically discovered and added. Configuring a VCF in this mode is the same as configuring a *Virtual Chassis* in pre-provisioned mode.

Note If you do not want links between switches to be converted to VCPs automatically, delete the LLDP configuration before powering on additional switches.

Note If you're using a mixed mode *Virtual Chassis Fabric*, you need to disable the VCPs on any EX4300 switch in order for the VCPs to autonegotiate successfully. Converting VCPs to network interfaces is covered in the *Data Plane* section.

The non-provisioned mode is similar to the pre-provisioned mode, except that the Virtual Chassis Ports are not automatically discovered and added, and the roles are not automatically defined; instead, a priority-based election process occurs.

To create a VCF, you need to set the master RE switch into the VCF mode with `request virtual-chassis mode fabric reboot`. At least one leaf switch needs to be installed next, and it should be cabled to the second spine switch before bringing up the second spine switch.

Note If you need a Mixed Mode *Virtual Chassis Fabric*, such as when building a fabric with the QFX5100 and EX4300, you need to use the `request virtual-chassis mode fabric mixed reboot` operational command. When operating a mixed mode *Virtual Chassis Fabric*, you can set the master's mode to `mixed`, then add all members, and then set all switches to `mixed` mode at the same time with the operational `request virtual-chassis mode fabric mixed all-members reboot` command.

Mastership Election

In auto-provisioned and pre-provisioned, modes, the QFX5100 that has the highest uptime is elected the master. The QFX5100 with the second-highest uptime is elected the backup. Any other QFX5100s in the spine role are line cards. If one of the masters fails, then one of the QFX5100 spines operating as a line card will be elected the new backup following the same uptime rules.

For a non-provisioned VCF, the following rules dictate master selection:

1: Highest priority (default is 128) 2: QFX5100 operating as master prior to reboot 3: QFX5100 with longest uptime (greater than one minute) 4: QFX5100 with lowest MAC address

For the backup RE, the process is repeated.

Note You might notice that this the same mastership election process as for *Virtual Chassis*.

Control Plane

`vccpd` runs on all nodes and is based on IS-IS. It is responsible for topology discovery. It also distributes any VCCP-specific state information. For unicast traffic, shortest path first is used; however, to support BUM traffic, bidirection multicast trees are used. Finally, for control plane traffic, a unique Class of Service queue is automatically created and used. All of this operational complexity is abstracted by *Virtual Chassis Fabric*.

When deploying a VCF, GRES, NSR, and NSB are used to keep the master and backup REs in sync.

For console access, each switch runs a virtual console server. When you attach to the console of any member switch, this virtual console server software automatically redirects your connection to the master RE. Once you're on the master RE, you can access a specific node with the `request session member <id>` command.

As with *Virtual Chassis*, the OOB management interface becomes a `vme` interface.

When a switch is removed, its member ID does not get released automatically. If you want to release the member ID to be used by the next switch attached, you can use the `request virtual-chassis recycle member-id <id>` operational cmmand.

When adding a new switch, the software versions must be compatible. You can either upgrade the devices manually, or you can use the `auto-sw-upgrade` configuration. When using this, you must have the images for each series (EX4300, QFX3500, QFX5100) in your fabric on the master RE or a remote URL. Use the `set virtual-chassis auto-sw-upgrade ex-4300 <path>` configuration command to set the path for an EX4300. Replace `ex-4300` with `qfx-3` or `qfx-5` for the QFX3500 or QFX5100, respectively.

When performing a software upgrade, the Non-Stop Software Upgrade (NSSU) feature can be used if using the `preprovisioned` mode. Additionally, `no-split-detection` (covered in the *Fabric Partition* section) must be configured.

Data Plane

Virtual Chassis Fabric has a concept of "Smart Trunks." When two or more links between two devices are connected, they will automatically form a LAG. Each path is weighted based on the bandwidth ratio. Traffic is distributed across multiple unequal paths, taking into account the minimum possible bandwidth on any links in the path.

A 16 byte Fabric Header is added to each packet received or sent by an ingress or egress device, similar to MPLS. It contains the incoming member ID, incoming port ID, destination member ID, and destination port ID, among other fields.

For load balancing hashing, the following fields are used:

Layer 2+Fabric Header:

- Source MAC
- Destination MAC
- Ethertype
- VLAN ID
- Incoming Port ID

- Incoming Member ID

Layer 3+4:

- Source IP
- Destination IP
- Source Port
- Destination Port
- Protocol
- Incoming Port ID
- Incoming Member ID
- Next Header (IPv6 Only)

If you need to convert an interface to a VCP, the `request virtual-chassis vc-port set pic-slot <id> port <id> member <id>` command can be used. The `member <id>` corresponds to the FPC number in the interface's representation. To do the opposite, replace `set` with `delete`. For example, `request virtual-chassis vc-port delete pic-slot 0 port 1 member 7`.

Finally, MAC learning is similar to a *Virtual Chassis*: when a member learns a new MAC address, it notifies the master of the MAC address. The master then programs all other members with the MAC-to-interface entry.

BUM Traffic

BUM traffic is distributed according to a Multicast Distribution Tree (MDT). There are multiple trees in a *Virtual Chassis Fabric*, each rooted at each switch. Therefore, there are `N` MDTs, where `N` is the number of switches in the *Virtual Chassis Fabric*. Each switch can load balance across all of the available MDTs for sending BUM traffic. This traffic is hashed based on the VLAN ID.

Note In a *Virtual Chassis Fabric*, all members receive a copy of all BUM traffic.

Fabric Partition

Sometimes, a fabric may become partitioned or "split." This occurs when one or more switches become isolated from one or more other switches in the fabric. When this happens, one of the new fabrics will remain active, and the others will be deactivated.

Note "Isolated" refers to communications via the Virtual Chassis Ports. Even if IP connectivity would otherwise exist, the fabric is considered partitioned if it cannot communicate over the VCPs.

To determine which fabric will remain active, the following rules are evaluated, in order:

1: The fabric contains both the master and the backup RE from the
previous fabric

2: The fabric contains the original master RE and at least half of the
members from the previous fabric

2: The fabric contains the backup master RE and at least half of the
members from the previous fabric

If your design can function when a partition happens, you can disable the default behavior with the `set virtual-chassis no-split-detection` configuration command. This disables the deactivation of partitioned fabrics described above.

1.4 Layer 3 Fabrics

This blueprint item primarily covers the following topics:

- *3-Stage Clos Architecture*
- *IP Fabric Routing*
- *IP Fabric Scaling*
- *IP Fabric Best Practices*

1.4.1 3-Stage Clos Architecture

A Clos network was originally a circuit switching architecture for the PSTN. It provided for connecting one input to one output with no blocking. In other words, connectivity was always possible. This was because of the intermediate switches that connected ingress and egress switches.

In IP networking, this concept has been applied in the "spine-leaf" architecture. In this design, a number of interconnecting transport switches (spines) connect to ingress and egress switches (leafs). They provide multiple paths for communications and can be designed in a way that is "non-blocking" if zero oversubscription is desired. These fabrics make better use of their links. They are classified as `n-stage`, where `n` is (more or less) the number of network devices a packet will traverse when ingressing and egressing from any point in the fabric to any point in the fabric. A 3-stage architecture consists of three network devices from point A to point Z: the ingress leaf, a spine, and an egress leaf. This means that you can have predictable network characteristics such as hop count and latency. For example, for intra-fabric communications, any destination is exactly two hops from its source (excluding advanced services such as overlay networking, firewalls, load balancers, etc.).

1.4.2 IP Fabric Routing

When designing an IP fabric, there are two primary designs, though both leverage BGP.

iBGP

In an iBGP design, an IGP such as OSPF or IS-IS is used to provide multipathing. iBGP peerings are formed over loopback interfaces, and route reflectors can be used to increase the scale of the solution. Route reflection, however, introduces its own issues.

When the same prefix is received from multiple leafs by a route reflector spine, it will (by default) only advertise the best path to the non-route-reflector spines. This results in suboptimal traffic forwarding, and in congested scenarios may result in avoidable congestion and degradation of service. To resolve this issue, BGP ADD-PATH can be used. BGP ADD-PATH causes the router to prefix a unique path identifier to a prefix advertisement; because of this, all routers at the spine level must support and be configured for BGP ADD-PATH.

eBGP

With eBGP, there is no IGP – BGP acts in a similar role as an IGP. In this design, each leaf node is given a unique ASN and all spines either share the same ASN or have unique ASNs per node.

Note The QFX5100 Series book [??], particularly Chapter 7: IP Fabrics (Clos), seems to suggest that the spine switches should each have a unique ASN. However, multiple other sources such as RFC7938 [??] and BGP

?? Juniper QFX5100 Series
?? RFC7938: Use of BGP in Large-Scale Data Centers, Section 5.2: EBGP Configuration for Clos Topology
?? BGP in the Data Center, Chapter 2, ASN Numbering Model

in the Data Center [??] suggest giving the spines the same ASN while allowing the leafs to have unique ASNs. This is likely to avoid the nead for `multipath multiple-as` on every leaf and to simplify spine configuration. However, even following this numbering scheme, the spines will ultimately require `multipath multiple-as`.

You might notice that because of this design, you will still need extra configuration to handle the scenario where a prefix is received from multiple switches. To work around this, use the `set protocols bgp group <name> multipath multiple-as` BGP configuration. The biggest difference between this an the requirement for ADD-PATH in an *iBGP* is that ADD-PATH is a feature negotiated between peers, whereas `multipath multiple-as` is not. This results in less complexity and fewer places for things to go wrong. It also increases vendor and platform interoperability since ADD-PATH is not guaranteed to be supported on a given platform or vendor, but even without `multipath multiple-as`, the fabric will still function (suboptimally). Additionally, if the spine switches are given the same ASN instead of a unique ASN per spine switch, the number of places that require `multipath multiple-as` might be reduced depending on your design and requirements.

Note `multipath multiple-as` is recommended on all nodes. You *can* get away with only deploying it in certain places, but that doesn't mean you should. One of the benefits of deploying an IP fabric is reproducibility, and when you start tailoring to such a degree, it increases the cognitive load on engineers supporting the fabric because they can no longer assume similar behavior amongst nodes.

1.4.3 IP Fabric Scaling

The scalability of an IP fabric is largely limited by three things:

- the acceptable oversubscription ratio
- the number of uplinks available per leaf per pod
- the number of ports per spine per pod

The acceptable oversubscription ratio can determine both how many servers you attach to a leaf and how many spines you need to have, both of which influence and are influenced by your required speeds downstream toward servers. For example, assume you have a 48x10G switch with 6x40G uplinks. This gives you an oversubscription ratio of 480G:240G, or 2:1, if you have six spine switches. However, perhaps it's unacceptable to have a 2:1 oversubscription ratio. Maybe you require no oversubscription. In this case, you can either select a leaf switch with more 40G uplinks (and add more spine swithces), or you can decide that you will not attach any servers to half of the 10G ports.

Note You can get clever and, if your switch supports it, break out all of the 40G ports to 10G ports. Then use only 10G spines. Split your total 10G ports in half; this number is the number of spines you need. For example, given the same leaf port density above and breaking out the 40G ports, you would have 72x10G interfaces. Dividing this by two, there are 36 ports downstream toward servers and 36 ports upstream toward spines. This means that you need either 36 spines *or* multiple ports stacked on spines, perhaps on a chassis-based spine such as an MX240. The end result is that you are "wasting" 12x10G ports compared to the 2:1 oversubscription model, and you likely need more or larger spines. In reality, 36 server ports in a 48U rack is probably the most (or close to the most) you'll get anyway. All of this said, though, for the purposes of this exam, it's likely best to stick with the number of "downstream" and "upstream" ports as noted by the difference in port speeds (i.e., 10G vs. 40G or 40G vs 100G).

1.4.4 IP Fabric Best Practices

There is one critical consideration to make when selecting links between spines and leafs: bandwdith. If

you go the *iBGP* route (and you use OSPF or use different metrics for different bandwidths in IS-IS), your ECMP will be affected if you have mixed uplink bandwidth from your leafs (such as 10G and 40G). The result is that you will have some unutilized links (except in failure scenarios).

However, with eBGP, link bandwidth is not considered for path calculation. If your design uses different bandwidth links, this might initially seem like a good thing because it results in all links being utilized. However, they are not intelligently utilized. You can easily exceed the utilization of the lower capacity links, resulting in either service degradation or complete outage conditions (depending on your SLAs).

For this reason, in either design, it is critical to ensure that all paths to a given destination from all possible ingress points have the same total link bandwidth. This makes managing oversubscription and scaling much easier.

All swithces at a given stage (spine, leaf, etc.) should be of the same model. This isn't strictly required so long as they can all support the same features, but if you choose to use different switches, you will be limited by the smallest number of ports available. Another consideration is that some switches may hash traffic differently than others, leading to unpredictable load sharing characteristics. In other words, it's okay if your spines and leafs are different models, but all of your spines should be the same model, and all of your leafs should be the same model.

Note *Technically* you aren't limited by the number of ports. However, this section is all about best practices. So, in the spirit of best practices, don't try to mix and match models in the same pod.

If you have a use case for different spine switches in different pods, then you can use different spine switches in different pods. An example of this might be for "bug diversity." In other words, you don't want all of your global data centers to be hit by the same platform/OS bug at the same time, so you might deploy different vendors or platforms in each fabric pod. However, inside of any given pod, you should still use the same model switch.

When connecting leaf switches to spine switches, every leaf should be connected to every spine. Again, you can get away with not doing this, but to do so is to invite heartache. If you're thinking of doing it for scaling reasons, consider implementing a 5-stage IP fabric instead. This brings some different complexity (not covered here), but it also brings easier and greater scalability without the risks of unintentionally oversubscribing or modifying the predictability of some of your fabric switches but not others.

1.4.5 Configuration

TODO: Add configuration examples for an IGP with iBGP, eBGP with iBGP,
and straight eBGP.

1.5 VXLAN

This blueprint item primarily covers the following topics:

- L2VPN Control Planes
- *Multicast Control Plane*
- *Data Plane*

Note Although the VXLAN section of the blueprint indicates L2VPN Control Planes, they are not covered here; instead, they are covered on the *EVPN VXLAN Signaling* page. I'm unaware of a different L2VPN control plane that uses VXLAN for encapsulation; perhaps the blueprint means MPLS and/or GRE L2VPNs, but that isn't clear since the item is listed under VXLAN.

Additionally, this page covers a brief introduction to VXLAN and why it exists in the *Introduction to VXLAN* section.

Finally, because I'm not quite sure where in the blueprint some items might (or might not) fall, the following sections are written to describe platform-dependent concepts:

- *VxLAN L2 and L3 Gateways*
- *Hardware Positioning*
- *Asymmetric vs Symmetric IRB*
- *Edge vs. Central Routing and Bridging*

1.5.1 Introduction to VXLAN

Modern data centers require greater scale and security than in years past. Traditionally, operators have dealt with spanning tree in the data center. While spanning tree itself is not a terrible technology and has its uses, it does result in unused links, which means network architects and operators needed to vastly overprovision their network links in order to have enough bandwidth available to avoid congestion. This may have come in the form of using 40G links where 10G links would have otherwise been enough or in the form of using multiple ports in a LAG. Both solutions are expensive and can have disastrous failover consequences, such as a LAG member failing and resulting in insufficient bandwidth on the active path (degredation of service) or link failures causing TCNs to be genereated (brief outages as MAC tables are flushed throughout the network). Additionally, the scale of every switch in the data center must also account for the total possible number of MAC bindings, ARP bindings, and IP addresses across all devices in the broadcast domain, which may result in purchasing more expensive hardware than might otherwise be required.

Layer 3 Fabrics and Layer 2 L2VPNs

One solution to the above problem is *Layer 3 Fabrics*; however, a purely layer 3 fabric introduces its own issues: complexity in the form of requiring rack awareness; IP planning difficulties; and an inability to support highly available components that require layer 2 adjacency, such as clustered firewalls, load balancers, and VRRP-based applications such as MySQL with `keepalived` (which uses VRRP, a protocol that communicates to the `224.0.0.18` multicast group, which is link-local and not routable).

In order to solve the problems that an IP fabric introduces, a layer 2 overlay can be used. This is something that's been done in the service provider realm for several years in the forms of L2VPN/Pseudowires and VPLS. These technologies use MPLS to transport layer 2 frames across a network without requiring the devices in the middle to know the source and destination MAC addresses, ARP bindings, etc. However, devices in the data center traditionally may not have MPLS functionality all the way to the server point of attachment, and MPLS features are frequently expensive, so a different way was devised.

Note Cost isn't the only reason VXLAN was created. Complexity is another; MPLS requires additional protocols and requires that every device support it end-to-end; VXLAN does not. With VXLAN, only the ingress and egress devices need to know how to deal with VXLAN; every other device in the middle just sees an IP packet. Additionally, VXLAN requires protocols that are frequently used anyway: either a multicast routing protocol or BGP.

Encapsulation

VXLAN transports layer 2 frames by encapsulating them into an IP packet with a UDP header. With VXLAN, an operator maps a VLAN ID to a VNI. Recall that VLAN IDs are 12 bits in length and can thus support up to 4,096 VLANs. VXLAN VNIs (Virtual Network Identifiers), on the other hand, are 24 bits in length and can support up to approximately 16 million unique IDs. Ultimately, with enough planning, this means that your tenant scale is now limited to ~4,096 per switch and 16 million across the entire foot-

print.

1.5.2 Multicast Control Plane

Now that we know a little bit about the motivation of VXLAN, we can look at the control plane protocols that support it. A key requirement of a layer 2 overlay (or VPN) is the ability to transport BUM traffic. With VXLAN, there are two primary ways to accomplish this: multicast and ingress (or head-end) replication. This page covers multicast, while the *EVPN VXLAN Signaling* page covers ingress replication.

PIM

The multicast control plane for VXLAN exists primary to support the *Data Plane*, covered later. The only way to use multicast for BUM traffic is to run PIM in your fabric. You associate each VNI with a particular multicast group. This is because when a segment needs to send a BUM packet, it must be delivered to all interested receivers. For this reason, the same VNI must be associated with the same group on all VTEPs. Eventually, with enough VNIs, you may run into issues with scaling the number of multicast groups. You will start mapping multiple VNIs to a single multicast group, which will result in inefficient forwarding of BUM traffic to uninterested VTEPs.

The best way to deploy an RP is to use *Anycast RP*. You use PIM-SM and will need reachability via some IGP. The RP should also be configured on spines, not leafs. There are many options for configuring RP selection, such as static, bootstrap RP, or auto RP. For the sake of simplicity, when combined with *Anycast RP*, a static configuration is likely the easiest option.

Note that the QFX does not support PIM BiDir [7].

Anycast RP

Anycast RP uses the same IP address for the RP on multiple devices. Anycast RP can be configured either with or without MSDP; for the sake of simplicity (and assuming this exam isn't focusing heavily on multicast), we'll only examine the design without MSDP.

When using Anycast RP, you must configure at least two loopback IP addresses: a unique IP per spine and a shared IP per spine. The shared IP per spine is the Anycast IP, and the unique IP is used as the router ID and so that the routers can sync their multicast states. When configuring the shared anycast IP, you have two options: you can either configure the unique IP as the `primary` IP; or you can configure the anycast shared IP under a different unit.

You must configure an `rp-set` for `pim-anycast` that includes all other Anycast RP members; the IP address used when defining the other members should be the unique IP address of the spine.

Note Instead of configuring an `rp-set` with each spine's unique IP address, you can use MSDP. However, that is out of scope for this guide as it seems unlikely (though not impossible) to appear on the exam. For completeness, you should read the Example: Configuring Multiple RPs in a Domain with Anycast RP configuration guide from Juniper. Additionally, the configuration of MSDP is likely more complicated than configuring an `rp-set` given the scale.

1.5.3 Data Plane

This section describes the VXLAN data plane for two different operations: *Data Plane for Known Unicast Traffic* and *Multicast Data Plane for BUM Traffic*. It also contains a section explaining the default 802.1q stripping behavior and how to override this in the *802.1Q Stripping* section.

802.1Q Stripping

It's worth noting that the default behavior of a VTEP is to strip the VLAN ID before encapsulating a frame in a VxLAN packet. The reverse is also true: if a VxLAN packet is decapsulated and a VLAN ID is

7 Junos Feature Explorer - Bidirectional PIM

present, the frame is discarded. This behavior can be modified on both the QFX and MX platforms.

On the MX, both the encapsulating and decapsulating behaviors are modified at the `[bridge-domains <name> vxlan]` hierarchy. For example, `set bridge-domains bd273 vxlan encapsulate-inner-vlan` will preserve the inner VLAN ID, and `set bridge-domains bd273 vxlan decapsulate-accept-inner-vlan` will accept decapsulated frames with a VLAN ID present.

On a QFX, the hierarchy level for preserving VLAN IDs when encapsulating is done at the `[vlans <name> vxlan]` level, while accepting decapsulated frames with a VLAN ID present is done at the `[protocols l2-learning]` level. For example, `set vlans vlan273 vxlan encapsulate-inner-vlan` and `set protocols l2-learning decapsulate-accept-inner-vlan`.

Note The option is the same between both the MX and the QFX, but where it is applied is different. The MX can perform these actions separately on multiple bridge domains, while the QFX applies them at a global level.

Data Plane for Known Unicast Traffic

VXLAN traffic is encapsulated and decapsulated by a VTEP. A VTEP can be either a hardware (such as a QFX5100) or software (such as Open vSwitch) device. This encapsulation consists of a new VXLAN header, a new UDP header, a new IP header, and a new Ethernet header. The information for each header is described in the list below.

- Ethernet Header: This is a normal Ethernet header; its source MAC will be the SMAC of the egress switchport; the destination MAC will be the DMAC of the next hop IP address.
- IP Header: The source IP will be the local VTEP; the destination IP will be the remote VTEP that contains the remote host of the original IP packet.
- UDP Header: The source port is a random port based on a hashing algorithm of the original payload headers, and that hashing algorithm may depend on the specific hardware platform. The destination port is usually the well-known (registered) VXLAN port of `4789`, though this can be changed on some implementations of VXLAN (especially those that are software-based, such as the Linux kernel or Open vSwitch).
- VXLAN Header: This is an 8 byte header that contains the VNI, a 24-bit field. 8 bits are used for flags, and the other 32 bits are reserved for future use.

The ingress VTEP adds these headers, and the egress VTEP removes them. Any devices between the VTEPs treat the packet as a normal packet, forwarding according to that platforms hashing algorithm for the outter packet.

Multicast Data Plane for BUM Traffic

The multicast data plane is simple compared to the *Multicast Control Plane*. At a high level, a BUM frame is encapsulated in a new VXLAN packet, including new layer 2, 3, and 4 headers. When it arrives on the destination VTEP(s), it is decapsulated, revealing the original segment. This is similar to the operations in the *Data Plane for Known Unicast Traffic*. The biggest difference is that the destination IP is not that of the destination VTEP but is instead that of the multicast group associated with the VNI.

Note In addition to BUM traffic, the multicast topology is used to discover remote VTEPs.

When dealing with a software VTEP, an IGMP Join is translated into a PIM Register by the PIM DR. These PIM Register messages are unicast to the RP. The RP decapsulates the message and forwards it to the destination VTEP. At this point, the tree switches from an RPT (`(*,G)` state) to an SPT (`(S,G)` state).

Note Multicast isn't heavily called out in the JNCIP-DC syllabus [??], so I expect it to be covered very lightly. There seems to be a heavier focus in both documentation and the syllabus on *EVPN VXLAN Signaling*. For this reason, these notes do not attempt to be an exhaustive multicast reference or even a primer.

1.5.4 VxLAN L2 and L3 Gateways

VxLAN has two types of gateways: Layer 2 and Layer 3.

VxLAN Layer 2 Gateway

A Layer 2 Gateway is what bridges a VLAN to a VNI (or vice versa). It is the stitching point for converting a legacy layer 2 network (a VLAN) to an overlay layer 2 network (a VxLAN VNI).

VxLAN Layer 3 Gateway

A VxLAN Layer 3 Gateway routes traffic between two VNIs. In a Juniper network, the VxLAN Layer 3 Gateway is frequently a QFX10k or MX Series device running at the spine layer; in this architecture, your spine switches must also be VTEPs. This presents a complexity and scaling issue. The *Hardware Positioning* section below discusses where to place specific Juniper hardware in the network.

Note VxLAN Layer 3 Gateway is sometimes simply referred to as VxLAN routing. In other materials, you may see it referred to as RIOT (Routing In and Out of Tunnels).

1.5.5 Hardware Positioning

Hardware positioning depends on the design. The design depends on hardware selection. This cyclical dependency can be difficult to tease apart. However, there is a simple rule of thumb:

If you want your leaf/top-of-rack layer to provide VxLAN Layer 3 gateway services, do not use the following hardware platforms:

- EX4300 (Broadcom Trident 2)
- EX4600 (Broadcom Trident 2)
- QFX5100 (Broadcom Trident 2)
- QFX5200 (Broadcom Tomahawk)

This is because of the ASICs used. These switches all use either the Broadcom Trident 2 or Tomahawk chipsets, which do not support the required functionality in hardware. If you already have an investment in these platforms, then it would be best to use them for VxLAN Layer 2 gateway services and implement a spine layer with the QFX10k or MX Series devices for your VxLAN Layer 3 gateway services. However, if you are starting fresh, consider the QFX5110 or QFX10k Series devices for your leaf/top-of-rack layer as they will allow you to perform VxLAN Layer 3 gateway services, thus reducing the traffic tromboning for inter-VxLAN routing in your data center fabric.

Note There are ways around this with other vendors, but it is unclear if Juniper supports these workarounds.

If you want VxLAN Layer 3 gateway services, the following platforms can be used:

- QFX5110 Series (Broadcom Trident 2+)
- QFX10000 Series (Juniper Q5)

?? JNCIP-DC Syllabus

- MX Series (Juniper Trio)

The QFX5110 is based on the Broadcom Trident 2+ and can perform VxLAN Layer 3 gateway services. The QFX10000 Series switches run a custom Juniper ASIC known as the Q5. This is also capable of performing the VxLAN Layer 3 gateway function. The MX Series uses Trio, which is a custom chipset that is incredibly flexible and can perform the VxLAN Layer 3 gateway services.

Ultimately, most of these platforms have some limitations to consider [8]. A general rule of thumb is to use the Trident 2 and Tomahawk chipsets for Layer 2 VxLAN and other chipsets for Layer 3 VxLAN services or to use newer chipsets for every tier.

1.5.6 Asymmetric vs Symmetric IRB

Note Juniper's original implementation used the *Asymmetric IRB* model. With the QFX10k and MX Series, they have implemented *Symmetric IRB*. See *Hardware Positioning* for additional hardware-specific details.

Asymmetric IRB

Asymmetric IRB performs bridging and routing on the ingress VTEP, but only briding on the egress VTEP. When a host wants to send a packet to a host in a different broadcast domain, it sends the packet with the destination MAC set to that of its default gateway as normal. When the ingress VTEP receives the frame, it performs a route lookup and encapsulates the frame in a VxLAN header, setting the VNI to that of the egress VNI. When the egress VTEP receives the VxLAN packet, it decapsulates it and bridges it directly onto the destination VLAN (known by the VLAN-to-VNI mapping). This happens in the opposite direction as well. The end result is that the VNI used when transporting a frame between two layer 2 domains will always be the egress VNI.

This model suffers a significant scalability penalty because it must have all VNIs configured even if there is not a host in that segment attached to the switch. This is because the ingress node must know about the egress VNI on the egress VTEP. However, its configuration is simpler, and depending on the hardware platform, *Symmetric IRB* might incur a performance penalty due to requiring one additional lookup on the egress VTEP compared to the Asymmetric IRB model.

Asymmetric IRB may sometimes be described as `bridge-route-bridge`. This refers to the lookups performed when moving traffic between two layer 2 segments. The ingress VTEP performs a bridging and routing operation, while the egress VTEP only performs a bridging operation.

Symmetric IRB

With Symmetric IRB, there is a dedicated Layer 3 VNI that is used for all layer 3 routing between any two layer 2 VNIs for the same tenant. This results in more configuration for the devices, and it also requires an additional hardware lookup when compared to *Asymmetric IRB*, but it is more scalable because it does not require the egress VNI to be configured on an ingress VTEP if there is not a host attached to that VNI locally.

Symmetric IRB may sometimes be described as `bridge-route-route-bridge`. This refers to the ingress VTEP performing a briding and routing operation and then the egress VTEP performing a routing and bridging operation.

1.5.7 Edge vs. Central Routing and Bridging

In VxLAN, there are two mechanisms for routing traffic: edge and central. These terms refer to how traffic in an overlay is routed. In both cases, the IP and MAC address must be the same for active/active forwarding; however, with *Central Routing and Bridging*, you can use active/standby gateway solutions

8 VXLAN Constraints on QFX Series and EX Series Switches

such as VRRP.

Central Routing and Bridging

Central Routing and Bridging refers to doing VxLAN routing on a central set of devices, such as the spines or a pair of special border leafs. There can be a scalability and complexity issue if this model is implemented on spines as it means that all spines must also be VTEPs and must support advanced VxLAN features. Bandwidth utilization is also a concern because if a tenant has multiple VNIs, even if the VMs are located on the same leaf switch, they must both go up to the spine switch in order to talk to each other. When implementing on a dedicated pair of border leafs, bandwidth utilization becomes an even larger consideration as traffic must go up to the spines, down to the border leafs, and back again. This type of design is reminiscent of some of the classic traffic tromboning and hairpinning issues.

Note While this is probably good for a vendor, it may not be great for an operator. It increases operational complexity and the cost of the solution. It may also be a significant waste of bandwidth.

CRB may be the correct design if you need very high ACL scale or when advanced services (load balancing, NAT, firewalls, etc.) are not native to your VxLAN overlay.

Edge Routing and Bridging

Edge Routing and Bridging may sometimes be called Anycast Gateway or Distributed Gateway. In this model, each leaf performs VxLAN routing. In order to minimize potential disruption, it is imperative that all VTEPs use the same IP address and MAC address for the IRB interface for a given Layer 3 Gateway.

1.5.8 Configuration

TODO: Add configuration examples for a VXLAN multicast control plane using OSPF as the IGP and *Anycast RP*.

1.6 EVPN VXLAN Signaling

Coming soon!

1.7 Technology-Oriented Labs

TODO: Flesh this out more. A lot more.

Right now, this page is a sort of scratch pad on things that are likely to be on the exam. It's lab-oriented, with some specific implementation details.

1.7.1 Lab 1 (Multicast and CRB with QFX)

This lab should focus on multicast for BUM replication and VTEP discovery. It should implement VxLAN routing at the spine layer. The spines should be QFX10k Series devices.

1.7.2 Lab 2 (Multicast and CRB with MX)

This lab should focus on multicast for BUM replication and VTEP discovery. It should implement VxLAN routing at the spine layer. The spines should be MX Series devices.

1.7.3 Lab 3 (Multicast and ERB)

This lab should focus on multicast for BUM replication and VTEP discovery. It should implement VxLAN routing at the leaf layer. Use either the QFX10k or QFX5110 Series for the leaf layer.

1.7.4 Lab 4 (EVPN and CRB with QFX)

This lab should focus on BGP EVPN and ingress replication for VTEP discovery. It should implement VxLAN routing at the spine layer. The spines should be QFX10k Series devices.

1.7.5 Lab 5 (EVPN and CRB with MX)

This lab should focus on BGP EVPN and ingress replication for VTEP discovery. It should implement VxLAN routing at the spine layer. The spines should be MX Series devices.

1.7.6 Lab 6 (EVPN and ERB)

This lab should focus on BGP EVPN and ingress replication for VTEP discovery. It should implement VxLAN routing at the leaf layer. Use either the QFX10k or QFX5110 Series for the leaf layer.

1.7.7 Lab 7 (EVPN Multihoming with QFX)

This lab should focus on EVPN using a QFX at the leaf layer. Use any of labs 3 through 6 as the base topology and add EVPN Multihoming at the leaf layer with a QFX Series device.

1.7.8 Lab 8 (EVPN Multihoming with MX)

This lab should focus on EVPN using an MX at the leaf layer. Use any of labs 3 through 6 as the base topology and add EVPN Multihoming at the leaf layer with an MX Series device.

1.7.9 Lab 9 (DCI with MX Series EVPN Stitching)

Implement a DCI solution with lab 7 or 8 as a starting point and MX Series devices as WAN routers. They should stitch a VxLAN EVPN to an MPLS EVPN solution.

1.7.10 Lab 10 (DCI with MX Series L3VPN)

Implement a DCI solution with lab7 or 8 as a starting point and MX Series devices as WAN routers. The VxLAN EVPN solution should simply ride over a normal MPLS L3VPN.

1.7.11 Lab 11 (DCI with VxLAN EVPN and MX)

Implement a DCI solution with lab7 or 8 as a starting point and MX Series devices as spines. The solution should be implemented using only VxLAN BGP EVPN.

1.7.12 Lab 12 (DCI with VxLAN EVPN and QFX)

Implement a DCI solution with lab7 or 8 as a starting point and QFX10k Series devices as spines. The solution should be implemented using only VxLAN BGP EVPN.

1.8 Additional Resources

Although I've tried to provide references on each page, they can become lost pretty easily. For that

reason, this page includes references for books, feature guides, solutions guides, videos, and blogs that might be helpful when studying for the JNCIP-DC exam.

1.8.1 Data Center Deployment or Management

The QFX5100 Series book from O'Reilly is a great resources for this section of the blueprint. Check chapters 6 (Network Automation) and 9 (Network Analytics) specifically.

A good blog post for ZTP is this one from NextHeader. It includes a topology diagram, switch outputs, configuration of an ISC DHCP Server on Ubuntu, and pcaps.

As you'll see throughout this list of resources, the blog over at https://jncie.tech/ has a blog on ZTP: JNCIE TECH - Zero Touch Provisioning.

Finally, there is an older Juniper Day One book on Deploying Zero Touch Provisioning. It's focused on the EX and SRX Series, but it should still serve as a useful reference.

1.8.2 Multichassis LAG

For MC-LAG, check out the MX Series book from O'Reilly. Chapter 9 is all about Multi-Chassis Link Aggregation.

This will cover you on the MX side, but there's still MC-LAG on the QFX to worry about. It's very similar, so it shouldn't be too difficult. Another great resource is the official Multichassis Link Aggregation Feature Guide.

The following list of blogs can be useful, too:

- JNCIE TECH - MC-LAG Overview
- JNCIE TECH - MC-LAG Lab - Basic L2 Connectivity
- JNCIE TECH - MC-LAG Lab - Advanced IRB Functionality
- Christians Juniper Blog - MC-LAG on vQFX (EVE-NG)

1.8.3 Layer 2 Fabrics

For Virtual Chassis, a great resource is Understanding Mixed EX Series and QFX Series Virtual Chassis. This will help you understand some limitations of mixed mode Virtual Chassis with EX and QFX Series switches. For configuration, check Configuring a QFX Virtual Chassis. For a more general Virtual Chassis read, check out the Junos Enterprise Switching book, specifically Chapter 4, EX Virtual Chassis. Finally, the Day One: EX Series Up and Running book has two chapters on Virtual Chassis - Chapters 4 and 5.

For Virtual Chassis Fabric, you can read the QFX5100 Series book. Chapter 5 is dedicated to Virtual Chassis Fabric. Another good reference is the Day One: Data Center Fundamentals book. Chapter 5 covers fabric architectures, including *Multichassis LAG*, Virtual Chassis, and Virtual Chassi Fabric. Finally, the Virtual Chassis Fabric Feature Guide is a great resource for all things Virtual Chassis Fabric. For a dive into best practices, check the Best Practices: Virtual Chassis Best Practices Guide.

For blogs, https://jncie.tech comes in again with JNCIE TECH - VCF.

1.8.4 Layer 3 Fabrics

This one is a potentially large topic, and there are a number of resources for it, including white papers, RFCs, and books.

First, there is the Clos IP Fabrics with QFX5100 Switches white paper. This is all about the layer 3 underlay with a strong focus on BGP. Both eBGP and iBGP (with route reflectors) are covered.

Next, in terms of books, there is the QFX5100 Series book. Chapter 7 is all about IP fabrics, although its

content seems to be largely the same as the afore-mentioned white paper. Another good reference is Chapter 6 of the Day One: Data Center Fundamentals book. This chapter is, again, completely dedicated to Layer 3 fabrics.

Note For a much more in-depth treatise on BGP in the Data Center, see the book of the same name. This book focuses on Free-Range Routing (FRR, the routing software used in Cumulus), but 100% of the theory applies here.

Finally, there is informational RFC 7938, Use of BGP for Routing in Large-Scale Data Centers.

If blogs are more your speed, the only one I've found that seems appropriately scoped for this topic only is JNCIE TECH - IP Fabric. Juniper QFX, IP-Fabric and VXLAN – Part may be helpful as well, but it also includes some multicast configuration, which I generally lump in with VxLAN.

1.8.5 VxLAN

This is a massive topic. The list of resources here will intentionally ignore EVPN as that is listed as a separate topic in the syllabus.

For books, the QFX5100 Series is again a great resource. Chapter 8 covers Overlay Networking. Day One: Data Center Fundamentals covers VxLAN in Chapter 7, Overlay Networking.

For blogs, we have a few to choose from:

- Juniper QFX, IP-Fabric and VXLAN – Part 1
- Juniper QFX, IP-Fabric and VXLAN – Part 2
- JNCIE TECH - VXLAN Multicast

These next two are from Cumulus, but they should still help explain gateway placement options:

- VXLAN Designs: 3 Ways to Consider Routing and Gateway Design (Part 1)
- VXLAN Designs: 3 Ways to Consider Routing and Gateway Design (Part 2)

The next set are specific to Cisco, but if you're familiar with NX-OS, you might find them helpful. They're also good for general theory.

- The Network Times - VXLAN Part I: Why do we need VXLAN?
- The Network Times - VXLAN Part III: The Underlay Network – Multidestination Traffic: Anycast-RP with PIM
- The Network Times - VXLAN Part V: Flood and Learn

If videos are your speed, here are a list of YouTube resources:

- Introduction to Cloud Overlay Networks - VxLAN by David Mahler
- VxLAN Playlist by Network Direction
- VxLAN 101 by Ivan Pepelnjak

1.8.6 EVPN VxLAN Signaling

For books, we start with the Day One: Data Center Fundamentals book, which covers EVPN in Chapter 9. From there, we can look to the QFX10000 Series book. Chapter 6 covers Ethernet VPN. We also have the This Week: Data Center Deployment EVPN/VXLAN book.

Note EVPN in the Data Center is a great book for learning about EVPN. Its primary focus is FRR, but all of the theory and concepts apply to Junos as well.

The following guides will also be useful:

- Solution Guide: Infrastructure as a Service: EVPN and VXLAN
- EVPN Feature Guide
- Cloud Data Center Architecture Guide
- EVPN LAG Multihoming in EVPN-VXLAN Cloud Data Center Infrastructures

For blog posts, we again have a large number of posts to dive into:

- Christians Juniper Blog - EVPN-VXLAN on (v)QFX-Series Devices
- Dan Hearty - Juniper QFX10K | EVPN-VXLAN | MAC Learning Verification | Single-Homed Endpoint
- Dan Hearty - Juniper QFX10K | EVPN-VXLAN | EVPN Anycast Gateway Verification
- Dan Hearty - Juniper QFX10k | EVPN-VXLAN | IRB Routing | BGP
- JNCIE TECH - EVPN-VXLAN Lab - Basic L2 Switching
- JNCIE TECH - EVPN-VXLAN Lab - RT Assignment Methods
- JNCIE TECH - EVPN-VXLAN Lab - IRB Functionality
- JNCIE TECH - MX EVPN-VXLAN Basic Config
- JNCIE TECH - QFX EVPN Basic Config
- JNCIE TECH - EVPN-VXLAN RT Communities
- Lab on EVPN – VXLAN on QFX5100 Switches
- VXLAN Routing with EVPN: Asymmetric vs. Symmetric Model (this is a Cumulus post, but it's still very helpful)

Next, some Cisco Nexus-centric blog posts:

- The Network Times - VXLAN Part VI: VXLAN BGP EVPN – Basic Configurations
- The Network Times - VXLAN Part VII: VXLAN BGP EVPN – Control Plane Operation
- The Network Times - VXLAN Part VIII: VXLAN BGP EVPN – External Connection
- The Network Times - VXLAN Part XII: Routing Exchange – Intra/Inter-L2VNI, EVPN-to-IP, EVPN-to-VPNv4
- The Network Times - VXLAN Part XIV: Control Plane Operation in BGP EVPN VXLAN Fabric
- The Network Times - VXLAN Part XV: Analysis of the BGP EVPN Control Plane Operation

Some videos from YouTube that might help:

- Juniper Networks EVPN - VXLAN Architecture from Tech Field Day
- Building Blocks in EVPN for Multi-Service Fabrics from NANOG 75

1.8.7 Data Center Interconnect

DCI is a pretty big topic with quite a few ways to implement. Most of the materials I've seen so far seem

to focus on straight VxLAN EVPN connectivity. However, there's at least one blog post from JNCIE TECH (listed below) that covers EVPN stitching.

Books that may be useful:

- Day One: Using Ethernet VPNs for Data Center Interconnect
- Day One: MPLS Up and Running
- Day One: MPLS for Enterprise Engineers

Note Two of the books above are on MPLS basics. For better or worse, it looks like a portion of this track relies on MPLS. I've added the two references references above in case you are coming directly from the Enterprise track, which is a prerequisite for the DC track but has no MPLS coverage.

Blog posts:

- JNCIE TECH - MX EVPN-MPLS Basic Config
- JNCIE TECH - MX EVPN IRB Functionality
- JNCIE TECH - EVPN-VXLAN to EVPN-MPLS Stitching

Videos from YouTube:

- BGP EVPN in Datacenter and Layer 3 Data Center Interconnect from NANOG 66 (This is Cisco, but theory should mostly apply)

1.8.8 Data Center Architecture and Security

This seems to be a pretty nebulous topic. The only items in the syllabus that are listed seem security-related, so I'm just going to focus on that. First, a list of Day One books:

- Day One: Configuring Junos Policies and Firewall Filters
- This Week: Hardening Junos Devices, 2nd Edition
- Day One: Deploying BGP Routing Security

Next, the MX Series book has an entire chapter dedicated to Routing Engine Protection and DDoS Prevention (Chapter 4).

Finally, a couple of blog posts:

- iNetZero - EVPN-VXLAN Inter-tenant Routing on Juniper QFX/MX
- Dan Hearty - Using JUNOS Firewall Filters for Troubleshooting & Verification | QFX5110

1.8.9 Miscellaneous

Some topics that are probably important but don't seem to be called out explicitly in the syllabus:

- Oversubscription: Day One: Data Center Fundamentals has this covered in Chapter 4, while QFX5100 Series covers it in Chapter 3, Performance and Scaling.
- Virtual Machine Traffic Optimization (VMTO): Comparing Layer 3 Gateway & Virtual Machine Traffic Optimization (VMTO) for EVPN/VXLAN and EVPN/MPLS. I'm honestly not sure where this fits in, but it's listed under `Additional Resources` on the JNCIP-DC Certification Page.

1.9 Contributors

This page lists any contributors to these notes, whether they are issues reported, pull requests submitted, or suggestions made. The following format is used:

- <Name> (@<twitter_handle>): <contribution_message> (link to GitHub issue(s) and/or pull request(s))

All of this is optional. Anything that is either not relevant or requested to be left out will not be present in the entry.

- Steve Rossen (@steve): It's Saturday night and you are studying: Issue #1 Closed, Won't Fix.

- *Search Page*