

A Practical Guide to Compute Express Link Memory Devices: Student Lab Guide

What you will learn

- An Introduction to CXL Memory Devices
- CXL Memory Benefits and Use Cases
- The CXL Architecture
- Hands-On Lab: Emulating CXL Devices in QEMU
- Continue Your Learning Path

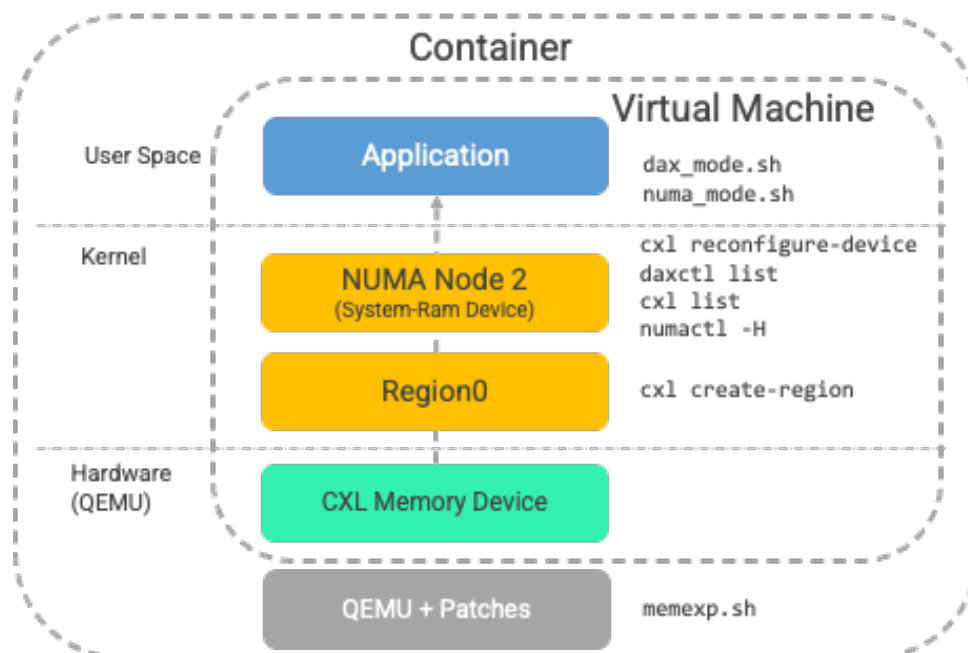
Lab Overview

1. ssh to your assigned lab host instance
2. Install podman
3. Pull the MemVerge CXL Expansion container image
4. Run the Container
5. Start a Guest VM with a single CXL Memory Expander
6. Login to the Guest VM

7. Create a Region
8. Explore the CXL Device using Linux tools
9. Start an Application using CXL and DRAM memory

Lab Environment

The lab software stack provides a ready-to-use QEMU binary and Virtual Machine image using features and functionality of QEMU and the Linux Kernel that are not currently upstreamed, but are available in development branches and patches.



Shell Prompts

The lab instructions predicate each command with the location it should be executed:

`[laptop]$` Run the command(s) on your desktop or laptop

`[container]$` Run the command(s) inside the container

`[vm]$` Run the commands(s) inside the virtual machine

Instructions

SSH to your assigned lab host. The hostname, ip address, username, and password will be provided separately.

```
[laptop]$ ssh user@hostname  
Password:
```

If this is the first time logging in to the host, update the package cache. Otherwise, this step can be skipped.

```
$ sudo apt update
```

Install Podman. You can use Docker as an alternative.

```
[host]$ sudo apt install -y podman
```

Start the Container & Virtual Machine. The Virtual Machine automatically starts when the container starts.

```
// Pull the Container image
[host]$ podman pull docker.io/mvpool/qemu_cxl_memexp

// Start the container as a daemon
[host]$ podman run -d --name cxllab qemu_cxl_memexp

// Connect to the container
[host]$ podman exec -it cxllab bash

// Allow the Virtual Machine time to start. Wait 2-3mins before
continuing.

// Connect to the Virtual Machine (Please be patient! It may
take a few mins)
[container]$ ssh -p 2222 fedora@localhost
Password: password

// If you get the following error, wait another minute and try
again. The VM has not yet booted.
# ssh -p 2222 fedora@localhost
ssh: connect to host localhost port 2222: Connection refused
```

Should the virtual machine fail to start, run `./memexp.sh`, located in the container's root, to start it.

Create a Region

```
[vm]$ ./create_region.sh
```

The `create_region.sh` script automatically provisions the memory in the 'system-ram' mode so it appears as another memory NUMA Node in the operating system.

CXL Device Discovery

The following commands allow you to discover CXL devices installed or attached to a host.

```
// List the CXL devices
[vm]$ cxl list
[vm]$ cxl list -vvv

// List the DAX devices
[vm]$ daxctl list

// Check the PCI device(s)
[vm]$ lspci | grep -i cxl
35:00.0 CXL: Intel Corporation Device 0d93 (rev 01)

// Get more info about the PCI/CXL device
[vm]$ lspci -s 35:00.0 -vvv
```

If the `lspci` command is not installed, run:

```
[vm]$ sudo dnf install -y pciutils
```

Note: The output from `lspci` is limited inside Virtual Machines. For example, your virtual machine will return something similar to the following:

```
[vm]$ lspci -s 35:00.0 -vvv
35:00.0 CXL: Intel Corporation Device 0d93 (rev 01) (prog-if 10
[CXL Memory Device (CXL 2.x)])
    Subsystem: Red Hat, Inc. Device 1100
    Physical Slot: 0
    Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop-
ParErr- Stepping- SERR+ FastB2B- DisINTx+
    Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort-
<TAbort- <MAbort- >SERR- <PERR- INTx-
    Latency: 0
    Region 0: Memory at fe840000 (64-bit, non-prefetchable)
[size=64K]
    Region 2: Memory at fe800000 (64-bit, non-prefetchable)
[size=256K]
    Region 4: Memory at fe850000 (32-bit, non-prefetchable)
[size=4K]
    Capabilities: <access denied>
    Kernel driver in use: cxl_pci
    Kernel modules: cxl_pci
```

Understanding the Memory Layout

Now that you have configured and provisioned an emulated CXL device inside a virtual machine, you will now learn what it looks like from the operating system and how applications can use it.

List the NUMA nodes. Notice how there is a NUMA node with no CPUs and some memory. This is because CXL Type 3 Memory Devices have no CPUs. Run:

```
[vm]$ numactl -H
```

In the following example, Node 1 is the CXL device.

```
[vm]$ numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3
node 0 size: 3901 MB
node 0 free: 3062 MB
node 1 cpus:          <-- A CXL.mem Device has no CPUs
node 1 size: 4096 MB
node 1 free: 4096 MB
node distances:
node    0    1
   0:   10   20
   1:   20   10 <-- CXL Device
```

Each NUMA node is broken into memory blocks by the Kernel. The default size is 2GiB blocks. To see the memory layout, use the following commands

```
[vm]$ lsmem
```

You will see an output similar to the following:

```
[vm]$ lsmem
```

RANGE	SIZE	STATE	REMOVABLE
BLOCK			
0x0000000000000000-0x000000007fffffffff	2G	online	yes
0-15			
0x0000000100000000-0x000000017fffffffff	2G	online	yes
32-47			
0x0000000390000000-0x000000048fffffffff	4G	online	yes 114-
145			

Memory block size:	128M
Total online memory:	8G
Total offline memory:	0B

To see the ZONE mode and which NUMA node the memory is assigned to, use:

```
[vm]$ lsmem -o+ZONES,NODE
```

You will see something similar to this:


```
[vm]$ lsmem -o+ZONES,NODE
```

RANGE		SIZE	STATE	REMOVABLE
BLOCK	ZONES	NODE		
0x0000000000000000-0x0000000007ffffff		128M	online	yes
0	None	0		
0x0000000008000000-0x000000007ffffff		1.9G	online	yes
1-15	DMA32	0		
0x0000000100000000-0x000000017ffffff		2G	online	yes
32-47	Normal	0		
0x0000000390000000-0x000000048ffffff		4G	online	yes
114-145	Movable	1		

```
Memory block size:      128M
```

```
Total online memory:    8G
```

```
Total offline memory:   0B
```

Use the `--all` option to see each individual memory block. ie: `lsmem -o+ZONES,NODE --all`

Start an Application

For this lab, we use the `memhog` utility, which allocates and then frees memory. When an application is run with `numactl`, we can control which memory NUMA nodes are used for memory allocations.

Allocate memory from DRAM and CXL using a 50:50 Interleave policy:

```
$ numactl --membind 0,1 --interleave memhog 1g
```

Run `memhog` again, using memory allocated entirely from CXL

```
$ numactl --membind 1 --interleave memhog 1g
```

Run `numastat` to confirm memory has been touched in the CXL NUMA Node (Node 1), for example:

```
[vm]$ numastat
```

	node0	node1
numa_hit	1154086	262175
numa_miss	0	0
numa_foreign	0	0
interleave_hit	980	0
local_node	1154086	0
other_node	0	262175

Explore the Kernel

If you are interested in learning how the Kernel works and how CXL is surfaced through the Kernel, explore the following directories and files within them to learn more.

Investigate `/dev`

```
[vm]$ ls /dev/dax*  
[vm]$ ls /dev/cxl/
```

`sysfs` has a lot of useful information. Explore and have fun.

```
[vm]$ ls /sys/bus/node/devices/node1/  
[vm]$ ls /sys/bus/cxl/devices/  
[vm]$ ls /sys/bus/acpi/devices/
```

Understanding DeviceDAX and System-RAM Namespace Modes

A Region is the Linux term for what APCI and UEFI call a system physical address range that may belong to a single memory device or interleaved amongst many. This is a similar concept to a concat-stripe amongst one or more disks.

Depending upon the physical device characteristics - SLD, MLD, DCD, etc, a Region may be partitioned using Namespaces, similar to how disks can be partitioned.

A Namespace can be provisioned to operate in one of four modes - `fsdax`, `devdax`, `sector`, and `raw`. The supported modes depend on the physical device. CXL Type 3 Memory Devices currently support `devdax` only.

The DeviceDAX (devdax) namespace mode creates a single character device file (`/dex/dax{X.Y}`). Applications can directly `mmap()` the device and perform load/store operations, just like any other memory mapped address range.

A more useful mode for applications that use a `malloc()` approach to allocating memory is to convert the devdax namespace to a `system-ram` mode. In system-ram mode, the memory is surfaced through the Kernel as a new NUMA Node, which any application can use.

Two scripts in the container's root allow you to change between `devdax` and `system-ram` modes.

```
// Convert the 'System-Ram' device to a 'devdax'
[vm]$ cd
[vm]$ ./dax_mode.sh

// Convert the 'devdax' device to a 'System-Ram' node
[vm]$ cd
[vm]$ ./numa_mode.sh
```

Read the relevant man pages and visit <https://pmem.io/> to learn more about this https://pmem.io). The same concepts for persistent memory can be applied to CXL Memory devices.

Continue Your Learning Path

We encourage you to continue learning and exploring. The Container and Virtual Machine images provided in this lab can be run on your laptop, desktop, or server. Here are some additional resources you will find useful to learn more. You will find several growing and enthusiastic communities.

- CXL Consortium: <https://www.computeexpresslink.org/>
- Linux Kernel CXL Mailing List: <https://lore.kernel.org/linux-cxl/>
- Linux Kernel CXL Driver:
<https://github.com/torvalds/linux/tree/master/drivers/cxl>
- NDCTL Linux Tools: <https://github.com/pmem/ndctl>
 - Includes `cxl` & `daxctl`
- QEMU: <https://www.qemu.org/>
 - Join the Community: <https://www.qemu.org/contribute>
- Intel Performance Counter Monitor (PCM) <https://github.com/intel/pcm>