SR-IOV Live Migration
<table>
<thead>
<tr>
<th>Table of contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Live Migration</td>
</tr>
<tr>
<td>Live Migration</td>
</tr>
<tr>
<td>MLX5 VF Live Migration</td>
</tr>
<tr>
<td>Requirements</td>
</tr>
<tr>
<td>NVCONFIG</td>
</tr>
<tr>
<td>Kernel Configuration</td>
</tr>
<tr>
<td>QEMU</td>
</tr>
<tr>
<td>Host Preparation</td>
</tr>
<tr>
<td>Over libvirt</td>
</tr>
<tr>
<td>Directly over QEMU</td>
</tr>
<tr>
<td>Running the Migration</td>
</tr>
<tr>
<td>Over libvirt</td>
</tr>
<tr>
<td>Directly over QEMU</td>
</tr>
<tr>
<td>Migration with MultiPort vHCA</td>
</tr>
<tr>
<td>Notes</td>
</tr>
</tbody>
</table>
Live migration refers to the process of moving a guest virtual machine (VM) running on one physical host to another host without disrupting normal operations or causing other adverse effects for the end user.

Using the Migration process is useful for:

- load balancing
- hardware independence
- energy saving
- geographic migration
- fault tolerance

Migration works by sending the state of the guest virtual machine's memory and any virtualized devices to a destination host physical machine. Migrations can be performed live or not; in the live case, the migration will not disrupt the user operations and it will be transparent to it as explained in the sections below.

**Non-Live Migration**

When using the non-live migration process, the Hypervisor suspends the guest virtual machine, then moves an image of the guest virtual machine's memory to the destination host physical machine. The guest virtual machine is then resumed on the destination host physical machine, and the memory the guest virtual machine used on the source host physical machine is freed. The time it takes to complete such a migration depends on the network bandwidth and latency. If the network is experiencing heavy use or low bandwidth, the migration will take longer than desired.

**Live Migration**

---

**Note**

This feature is supported in Ethernet mode only.
When using the Live Migration process, the guest virtual machine continues to run on the source host physical machine while its memory pages are transferred to the destination host physical machine. During migration, the Hypervisor monitors the source for any changes in the pages it has already transferred and begins to transfer these changes when all of the initial pages have been transferred.

It also estimates transfer speed during migration, so when the remaining amount of data to transfer will take a certain configurable period of time, it will suspend the original guest virtual machine, transfer the remaining data, and resume the same guest virtual machine on the destination host physical machine.

**MLX5 VF Live Migration**

The purpose of this section is to demonstrate how to perform basic live migration of a QEMU VM with an MLX5 VF assigned to it. This section does not explains how to create VMs either using libvirt or directly via QEMU.

**Requirements**

The below are the requirements for working with MLX5 VF Live Migration.

<table>
<thead>
<tr>
<th>Components</th>
<th>Description</th>
</tr>
</thead>
</table>
| Adapter Cards | • ConnectX-7 ETH  
                   • BlueField-3 ETH |
| Note | The same PSID must be used on both the source and the target hosts (identical cards, same CAPs and features are needed), and have the same firmware version. |
| Firmware | • 28.41.1000  
            • 32.41.1000 |
| Kernel | Linux v6.7 or newer |
### Components

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iproute2 version 6.2 or newer</td>
</tr>
<tr>
<td>QEMU 8.1 or newer</td>
</tr>
<tr>
<td>Libvirt 8.6 or newer</td>
</tr>
</tbody>
</table>

### Setup

**NVCONFIG**

SR-IOV should be enabled and be configured to support the required number of VFs as of enabling live migration. This can be achieved by the below command:

```
mlxconfig -d *<PF_BDF>* s SRIOV_EN=1 NUM_OF_VFS=4 VF_MIGRATION_MODE=2
```

where:

<table>
<thead>
<tr>
<th>SRIOV_EN</th>
<th>Enable Single-Root I/O Virtualization (SR-IOV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_OF_VFS</td>
<td>The total number of Virtual Functions (VFs) that can be supported, for each PF.</td>
</tr>
<tr>
<td>VF_MIGRATION_MODE</td>
<td>Defines support for VF migration.</td>
</tr>
<tr>
<td></td>
<td>• 0x0: DEVICE_DEFAULT</td>
</tr>
<tr>
<td></td>
<td>• 0x1: MIGRATION_DISABLED</td>
</tr>
<tr>
<td></td>
<td>• 0x2: MIGRATION_ENABLED</td>
</tr>
</tbody>
</table>
Needs to be compiled with driver MLX5_VFIO_PCI enabled. (i.e. CONFIG_MLX5_VFIO_PCI).

To load the driver, run:

```bash
modprobe mlx5_vfio_pci
```

**QEMU**

Needs to be compiled with VFIO_PCI enabled (this is enabled by default).

**Host Preparation**

As stated earlier, creating the VMs is beyond the scope of this guide and we assume that they are already created. However, the VM configuration should be a migratable configuration, similarly to how it is done without SRIOV VFs.

**Note**

The below steps should be done before running the VMs.

**Over libvirt**

1. Set the PF in the "switchdev" mode.

   ```bash
   devlink dev eswitch set pci/<PF_BDF> mode switchdev
   ```

2. Create the VFs that will be assigned to the VMs.
3. Set the VFs as migration capable.

1. See the name of the VFs, run:

```
    devlink port show
```

2. Unbind the VFs from mlx5_core, run:

```
    echo '<<BDF>>' > /sys/bus/pci/drivers/mlx5_core/unbind
```

3. Use devlink to set each VF as migration capable, run:

```
    devlink port function set pci/<PF_BDF>/1 migratable enable
```

4. Assign the VFs to the VMs.

1. Edit the VMs XML file, run:

```
    virsh edit <VM_NAME>
```

2. Assign the VFs to the VM by adding the following under the "devices" tag:

```
    <hostdev mode='subsystem' type='pci' managed='no'>
          <driver name='vfio'/>
          <source>
            <address domain='0x0000' bus='0x08' slot='0x00' function='0x2'/>
          </source>
          <address type='pci' domain='0x0000' bus='0x09' slot='0x00' function='0x0'/>
```

```
5. Set the destination VM in incoming mode.

1. Edit the destination VM XML file, run:

```
    virsh edit <VM_NAME>
```

2. Set the destination VM in migration incoming mode by adding the following under "domain" tag:

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
    [...]  
    <qemu:commandline>
        <qemu:arg value='--incoming'/>
        <qemu:arg value='tcp:<DEST_IP>:<DEST_PORT>'/>
    </qemu:commandline>
</domain>
```

**Note**

To be able to save the file, the above "xmlns:qemu" attribute of the "domain" tag must be added as well.
6. Bind the VFs to mlx5_vfio_pci driver.

1. Detach the VFs from libvirt management, run:

```
virsh nodedev-detach pci_<VF_BDF>
```

2. Unbind the VFs from vfio-pci driver (the VFs are automatically bound to it after running "virsh nodedev-detach"), run:

```
echo '<VF_BDF>' > /sys/bus/pci/drivers/vfio-pci/unbind
```

3. Set driver override, run:

```
echo 'mlx5_vfio_pci' > /sys/bus/pci/devices/<VF_BDF>/driver_override
```

4. Bind the VFs to mlx5_vfio_pci driver, run:

```
echo '<VF_BDF>' > /sys/bus/pci/drivers/mlx5_vfio_pci/bind
```

**Directly over QEMU**

1. Set the PF in "switchdev" mode.

```
devlink dev eswitch set pci/<PF_BDF> mode switchdev
```

2. Create the VFs that will be assigned to the VMs.
3. Set the VFs as migration capable.

1. See the name of the VFs, run:

```
devlink port show
```

2. Unbind the VFs from mlx5_core, run:

```
echo '<VF_BDF>' > /sys/bus/pci/drivers/mlx5_core/unbind
```

3. Use devlink to set each VF as migration capable, run:

```
devlink port function set pci/<PF_BDF>/1 migratable enable
```

4. Bind the VFs to mlx5_vfio_pci driver:

1. Set driver override, run:

```
echo 'mlx5_vfio_pci' > /sys/bus/pci/devices/<VF_BDF>/driver_override
```

2. Bind the VFs to mlx5_vfio_pci driver, run:

```
echo '<VF_BDF>' > /sys/bus/pci/drivers/mlx5_vfio_pci/bind
```

---

**Running the Migration**
**Over libvirt**

1. Start the VMs in source and in destination, run:

   ```
   virsh start <VM_NAME>
   ```

2. Enable switchover-ack QEMU migration capability. Run the following commands both in source and destination:

   ```
   virsh qemu-monitor-command <VM_NAME> --hmp "migrate_set_capability return-path on"
   virsh qemu-monitor-command <VM_NAME> --hmp "migrate_set_capability switchover-ack on"
   ```

3. **[Optional]** Configure the migration bandwidth and downtime limit in source side:

   ```
   virsh qemu-monitor-command <VM_NAME> --hmp "migrate_set_parameter max-bandwidth <VALUE>"
   virsh qemu-monitor-command <VM_NAME> --hmp "migrate_set_parameter downtime-limit <VALUE>"
   ```

4. Start migration by running the migration command in source side:

   ```
   virsh qemu-monitor-command <VM_NAME> --hmp "migrate -d tcp:<DEST_IP>:<DEST_PORT>"
   ```

5. Check the migration status by running the info command in source side:

   ```
   virsh qemu-monitor-command <VM_NAME> --hmp "info migrate"
   ```
Directly over QEMU

1. Start the VM in source with the VF assigned to it:

   ```
   qemu-system-x86_64 [...] -device vfio-pci,host=<VF_BDF>,id=mlx5_1
   ```

2. Start the VM in destination with the VF assigned to it and with the "incoming" parameter:

   ```
   qemu-system-x86_64 [...] -device vfio-pci,host=<VF_BDF>,id=mlx5_1 -incoming tcp:<DEST_IP>:<DEST_PORT>
   ```

3. Enable switchover-ack QEMU migration capability. Run the following commands in QEMU monitor, both in source and destination:

   ```
   migrate_set_capability return-path on
   ```

   ```
   migrate_set_capability switchover-ack on
   ```

4. [Optional] Configure the migration bandwidth and downtime limit in source side:

   ```
   migrate_set_parameter max-bandwidth <VALUE>
   ```

---

**Note**

When the migration status is "completed" it means the migration has finished successfully.
5. Start migration by running the migration command in QEMU monitor in source side:

```
migrate -d tcp:<DEST_IP>:<DEST_PORT>
```

6. Check the migration status by running the info command in QEMU monitor in source side:

```
info migrate
```

![Note]

When the migration status is "completed" it means the migration has finished successfully.

---

**Migration with MultiPort vHCA**

Enables the usage of a dual port Virtual HCA (vHCA) to share RDMA resources (e.g., MR, CQ, SRQ, PDs) across the two Ethernet (RoCE) NIC network ports and display the NIC as a dual port device.

MultiPort vHCA (MPV) VF is made of 2 "regular" VFs, one VF of each port. Creating a migratable MPV VF requires the same steps as regular VF (see steps in section Over libvirt). The steps should be performed on each of the NIC ports. MPV VFs traffic cannot be configured with OVS. TC rules must be defined to configure the MPV VFs traffic.

**Notes**
Note

In ConnectX-7 adapter cards, migration cannot run in parallel on more than 4 VFs. It is the administrator's responsibility to control that.

Note

Live migration requires same firmware version on both the source and the target hosts.