Virtual machines on SEAPATH

This page describes the configuration and deployment of virtual machines on SEAPATH.

SEAPATH uses QEMU and Libvirt to deploy virtual machines. With this technology, two elements are needed:

- A VM disk image. This file represents the disk of the virtual machine, it will contain the kernel as well as the filesystem.
- A domain XML format. This file describes all the devices needed by the VM. It can be memory, cpu information, interfaces ...

Build disk image

SEAPATH can host custom virtual machines. If you already have an image, skip to the next section.

However, if you need a virtual machine for testing or for deploying your application, you can use SEAPATH default disk image file :
Information on how to build a disk image in detail is available here for a Yocto VM, or here for a Debian VM.

Configure the virtual machine with domain XML file

If you use a custom VM, it is likely that you already have this XML file. We advise to still read the below information to have recommendations about VM configuration on SEAPATH.

In order to create your own XML for your virtual machines, here are two useful links:

- The official documentation on the domain XML format of libvirt: https://libvirt.org/formatdomain.html
- Examples of XML for SEAPATH: are provided https://github.com/seapath/ansible/blob/debian-main/templates/vm

A preconfigured XML using Ansible template is also provided. It does not have the flexibility of a fully handwritten XML, but is useful for testing or prototyping. For more information, read this page: Deploy with preconfigured XML.

Cluster specific

When you use SEAPATH in cluster, some part of the XML configuration is done by SEAPATH. This is the case for the *name*, *uuid* and disk.

The fields *name* and *uuid* will be ignored if there are present in the XML configuration file.

The disk configuration shall not be configured inside the XML otherwise the VM deployment will fail.

Note that SEAPATH only supports one disk per VM in cluster for the moment.

Virtual machine and slices

If you configured SEAPATH with machine-rt and machine-nort slices (see Scheduling and priorities), you must put the VM into it. This is done using libvirt resources. The VM will then only have access to the CPU associated with the slice.

Possible values:

- `/machine/nort`
- `/machine/rt`

Example, for a virtual machine with the real-time:

```
<resource>
    <partition>/machine/rt</partition>
</resource>
```

CPU tuning

In the project, this element will be used to limit the virtual machine (more details here).

- The `vcpupin` element specifies which host's physical CPUs the domain vCPU will be pinned to. Use this value only for performance driven VMs.
- The `vcpusched` element specifies the scheduler type for a particular vCPU. Use this value only for performance driven VMs.
- The `emulatorpin` element specifies which of host physical CPUs the emulator will be pinned to. The emulator describes the management processes of the VM (watchdog, creation, timers, modification). This value is not useful, most of the time.

If you configured CPU restriction for slices (see Scheduling and priorities), all the CPU ranges that you provide for `emulatorpin` and `vcpupin` must be part of the allowed CPUs of the slice where the VM is. By default, the VM is part of the `machine` slice, but it can be in the `machine-rt` or `machine-nort` slice following your configuration.

If you deploy a machine with real time vcpu scheduler, you must set the `emulatorpin` cpu range. It can be set on the system cpu range or on specific cores.
Remember that, for maximal performance, each vCPU must be scheduled alone on its core. Emulatorpin must then be set on another core.

Deployment of VM with Ansible

To deploy the virtual machines, use the playbooks `ansible/playbooks/deploy_vms_cluster.yaml` and `ansible/playbooks/deploy_vms_standalone.yaml` depending on your setup. These are supposed to work with the vm example inventories (see here for yocto version and here for debian version)

These playbooks will call the library `ansible/library/cluster_vm.py` which wraps the vm_manager tool.

Manage virtual machines on the cluster

- Check the execution of the resource:

```
crm status
```

- Get the status of the resource:

```
vm-mgr status --name NAME
```

- Delete VM in the cluster:

```
vm-mgr remove --name NAME
```

For more information about the vm_manager tool, check out this page : The vm_manager tool