



Linux Stacks for Intel® SGX

2021.12.08 Release

Linux Stacks for Intel® SGX

2021.12.08 Release

- [Introduction](#)
- [Prerequisites](#)
 - [Memory DIMM Configurations](#)
 - [BIOS Setup](#)
- [Software Stack Components](#)
- [Manual Setup for Development](#)
 - [Download *intelsgxstack.repo*](#)
 - [SGX SDK](#)
 - [SGX Development User Setup](#)
 - [Gramine](#)
 - [Gramine Busybox](#)
 - [KVM Stack](#)
 - [Setup the SGX KVM host](#)
 - [Create Virtual Machines with SGX support](#)
 - [Kubernetes Stack](#)
 - [Install *kubeadm*](#)
 - [Install NFD, SGX device plugin](#)
- [Ansible Mass Deployment](#)
 - [High Level Flow](#)
 - [Assumptions & Limitations](#)
 - [Prerequisites](#)
 - [Run Ansible in Docker on controller node](#)
 - [Config Variables](#)
 - [Config Variables for Virtual Machines Creation](#)
 - [Variables for *vm_guests*](#)
 - [VM Template](#)
 - [Deploy the KVM host virtualization stack](#)
 - [Deploy Kubernetes stack](#)
 - [Demos and utilities](#)
 - [Known Issues](#)
 - [1. Kubernetes stacks fail to be deployed](#)
 - [2. Error passing *kubernetes_join_command* when using *docker-playbook.sh*](#)
 - [3. Red Hat does not show host names of registered virtual machines](#)
 - [4. QEMU SGX arguments not backwards compatible](#)
- [Additional Information](#)
 - [Statement](#)

Introduction

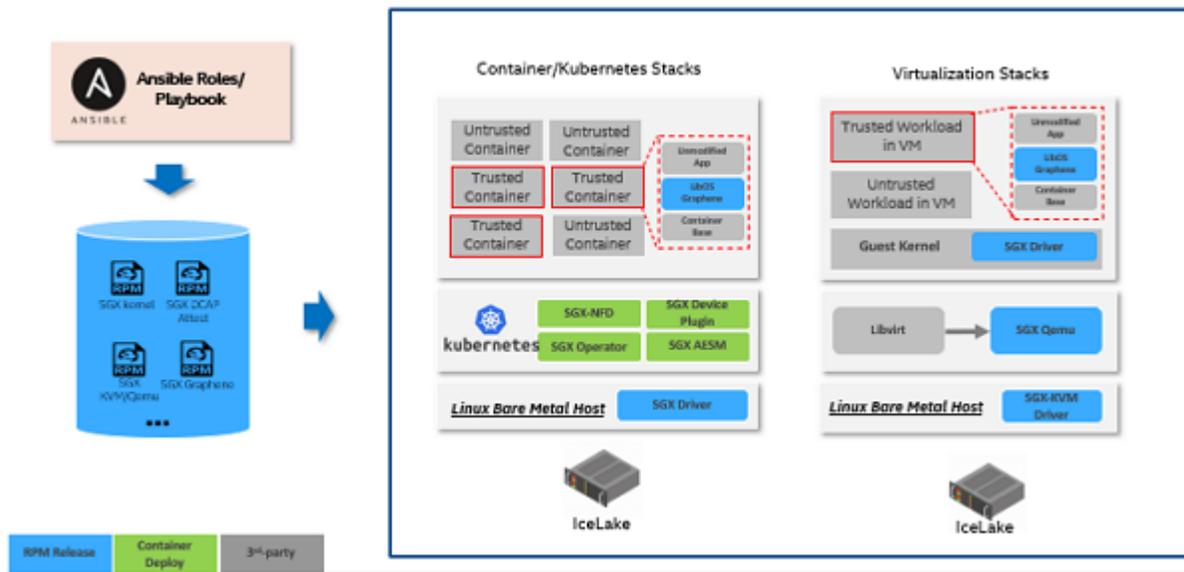
This document introduces the setup and deployment of the Linux* Stacks for Intel® SGX on a 3rd Generation Intel® Xeon® Scalable Processors server.

Note: You can find more detail about SGX (Software Guard Extension) at [Intel® Software Guard Extension](#)

The cloud stacks include:

- Virtualization stack: KVM based virtualization environment to run an SGX enclave workload in a VM guest. The Libvirt/ QEMU* components act as the orchestrator.
- Containers Stack: A container based Kubernetes* environment to run an SGX enclave workload in a container. Kubernetes is the orchestrator for the SGX device plugin.

The stack components are packaged into RPMs and are delivered via an RPM repository server. This document introduces the setup for development and Ansible* deployment for mass provisioning in the cloud environment.



*Note: The Stacks are presently implemented only for the RHEL 8.4 and RHEL 8.5 Operating Systems

Prerequisites

NOTE: The following configurations are for reference only; please use the updated platform configurations from the [Intel® Software Guard Extensions SDK for Linux*](#) documentation.

Memory DIMM Configurations

On 3rd Generation Intel® Xeon® Scalable Processors platforms, the SGX supported memory configuration is shown as follows:

IMC#	IMC0				IMC1				IMC2				IMC3			
Channel	Chan 0 (A)		Chan 1 (B)		Chan 0 (C)		Chan 1 (D)		Chan 0 (E)		Chan 1 (F)		Chan 0 (G)		Chan 1 (H)	
DDR4	Slot0	Slot1														
8	DDR4															
12	DDR4	DDR4	DDR4													
16	DDR4	DDR4														

BIOS Setup

Ensure the following BIOS settings are set as shown:

- TME enable:

Advanced → Socket Configuration → Processor Configuration → TME, MK-TME, TDX → Total Memory Encryption → Enable

NOTE: SGX will be visible only if TME is enabled.

- Disable UMA-Based Clustering (Otherwise SGX will be grayed out):

Advanced → Socket Configuration → Common RefCode Configuration → UMA-Based Clustering → Disable

- Enable SGX:

Advanced → Socket Configuration → Processor Configuration → SW Guard Extensions(SGX) → Enable

- Disable Patrol scrub (Only LCC & HCC):

Advanced → Socket Configuration → Memory RAS Configuration → Patrol Scrub → Disable

- Disable Mirroring:

Advanced → Socket Configuration → Memory RAS Configuration → Mirror Mode → Disable

- Enable Memory ECC

Software Stack Components

All software components are based on Open Source licensed software, installed from the public DNF repository:

<https://download.01.org/intelsgxstack/2021-12-08/rhel>

Component	Package Name	Description
kernel-5.15.5	intel-mvp-mainline-kernel-sgx-kvm	Kernel for KVM host
linux-sgx-sdk-2.15.1	intel-mvp-linux-sgx-sdk	SGX SDK/PSW
qemu-kvm-6.2.0-rc2	qemu-kvm	QEMU KVM with SGX compatible with kernel-5.15.5
gramine-1.0	intel-mvp-gramine	Gramine LibOS

NOTE:

- The kernel will create SGX device node names:
 - /dev/sgx_enclave
 - /dev/sgx_provision

For compatibility with legacy node names, create a udev rules file such as 10-sgx.rules:

```
SUBSYSTEM="misc",KERNEL="enclave",MODE="0666"  
SUBSYSTEM="misc",KERNEL="provision",GROUP="sgx_prv",MODE="0660"  
SUBSYSTEM="misc",KERNEL="sgx_enclave",MODE="0666",SYMLINK+="sgx/enclave"  
SUBSYSTEM="misc",KERNEL="sgx_provision",GROUP="sgx_prv",MODE="0660",SYMLINK+="sgx/provision"
```

Manual Setup for Development

Download *intelsgxstack.repo*

Download the file *intelsgxstack.repo* from <https://download.01.org/intelsgxstack/2021-12-08/rhel> and copy it to the folder */etc/yum.repos.d*. More details [here](#).

SGX SDK

Install the SDK:

```
sudo dnf install intel-mvp-linux-sgx-sdk
```

This installs the SGX-SDK in the default folder `/opt/intel/sgxsdk`. Sample code is installed under the `/opt/intel/sgxsdk/SampleCode` directory with read-only permissions for normal users. Each user should make separate copies to modify, build, and experiment with the sample codes.

Note: To be able to run the code samples you need to have these packages installed:

```
sudo dnf install libsgx-launch libsgx-urts
```

It is also recommended to source the SDK environment:

```
source /opt/intel/sgxsdk/environment
```

SGX Development User Setup

The first step is to verify we have SGX legacy nodes and correct ownership:

```
$ sudo ls -al /dev/sgx*
crw-rw-rw-. 1 root root    10, 126 May  7 15:47 /dev/sgx_enclave
crw-rw----. 1 root sgx_prv 10, 125 May  7 15:47 /dev/sgx_provision

/dev/sgx:
total 0
drwxr-xr-x. 2 root root   80 May  7 15:47 .
drwxr-xr-x. 21 root root 3040 May  7 15:47 ..
lrwxrwxrwx. 1 root root   14 May  7 15:47 enclave -> ../sgx_enclave
lrwxrwxrwx. 1 root root   16 May  7 15:47 provision -> ../sgx_provision
```

If we do not have the group `sgx_prv` and the node names `/dev/sgx/enclave` and `/dev/sgx/provision`, we need to create them. Users that are members of the group `sgx_prv` do not need to run SGX applications with root privileges.

```
sudo groupadd sgx_prv
```

Next, create the file `/etc/udev/rules.d/10-sgx.rules` and reload the udev rules:

```
$ cat /etc/udev/rules.d/10-sgx.rules
SUBSYSTEM=="misc",KERNEL="enclave",MODE="0666"
SUBSYSTEM=="misc",KERNEL="provision",GROUP="sgx_prv",MODE="0660"
SUBSYSTEM=="misc",KERNEL="sgx_enclave",MODE="0666",SYMLINK+="sgx/enclave"
SUBSYSTEM=="misc",KERNEL="sgx_provision",GROUP="sgx_prv",MODE="0660",SYMLINK+="sgx/provision"

udevadm trigger
```

To be able to run and/or develop SGX based applications as a non-privileged user, follow these steps:

Create a user account and add the user to desired groups:

```
sudo useradd <user>
sudo passwd <user>
sudo usermod -aG wheel,sgx_prv <user>
```

Finally, log out and log in as the new user.

Gramine

Prerequisite: kernel with SGX support.

This release contains Gramine RPM built from the repository: <https://github.com/gramineproject/gramine>. Gramine RPM is provided to allow testing and evaluation on RHEL platforms without any out-of-tree SGX drivers.

To install:

```
sudo dnf install intel-mvp-gramine
```

You can view the SGX information by issuing the following command:

```
is_sgx_available
```

Gramine Busybox

Gramine Busybox demonstrates how to distribute packaged Gramine based applications. To run busybox with Gramine as an example application, please follow the steps below.

NOTE: The following steps are applicable to RHEL/CentOS. Other distributions may require modified steps.

1. Verify your user account meets the conditions as described in *SGX Development User Setup*.
2. Install several pre-requisites:

```
sudo dnf install epel-release nss-mdns binutils python3
```

3. Install gramine-busybox. This will install the busybox binary, the manifest file, and it will sign the application:

```
sudo dnf install intel-mvp-gramine-busybox
```

4. Change directory to the location where the gramine-busybox files were installed:

```
cd /usr/libexec/gramine-busybox
```

5. Run some Busybox commands within an SGX enclave:

Example: List available commands:

```
gramine-sgx busybox
```

Example: Get Busybox kernel version

```
gramine-sgx busybox uname -a
```

Example: Run the Busybox shell within an SGX enclave:

```
gramine-sgx busybox sh
```

A shell should be running within an SGX enclave. You can run any busybox shell command at this point. To exit, hit Ctrl+C or type the exit command.

NOTE: There is no shell prompt displayed in the busybox shell. Just go ahead and type various commands (ls, uname, busybox, etc.) after you see the line: `error: Using insecure argv source. Gramine will continue application execution, but this configuration must not be used in production!`

KVM Stack

Setup the SGX KVM host

You can install KVM SGX kernel manually via:

```
sudo dnf install intel-mvp-mainline-kernel-sgx-kvm
```

This will install the latest kernel, which in this case should be the kernel-5.15.5. This kernel supports KVM SGX.

If planning to install Virtual Machines (VMs), you will also need to install QEMU with SGX support:

```
sudo dnf install qemu-kvm
```

Create Virtual Machines with SGX support

To install Virtual Machines, you will generally need to follow the instructions from [here](#).

There is no need to configure/build the kernel nor QEMU, as this has already been done. However, make sure your `/etc/libvirt/qemu.conf` contains:

```
cgroup_device_acl = [  
  "/dev/null", "/dev/full", "/dev/zero",  
  "/dev/random", "/dev/urandom",  
  "/dev/ptmx", "/dev/kvm", "/dev/kqemu",  
  "/dev/rtc", "/dev/hpet", "/dev/sgx_vepc", "/dev/sgx_provision"  
]
```

QEMU needs to read and write to the `/dev/sgx_vepc` device, which is owned by root with file mode 600. This means you must configure QEMU to run as root. In order to do that also ensure `/etc/libvirt/qemu.conf` contains:

```
user = "root"
```

You may need to manually install bridged network named `br0` bridging the Ethernet device on the KVM host computer if you plan on creating Virtual Machines and make them accessible from outside of the host computer. You can find details on setting up bridged networks in Linux [here](#).

Kubernetes Stack

Install *kubeadm*

Follow the instructions from [here](#) to install kubeadm. Then create a Kubernetes master or slave to join an existing Kubernetes cluster.

Install NFD, SGX device plugin

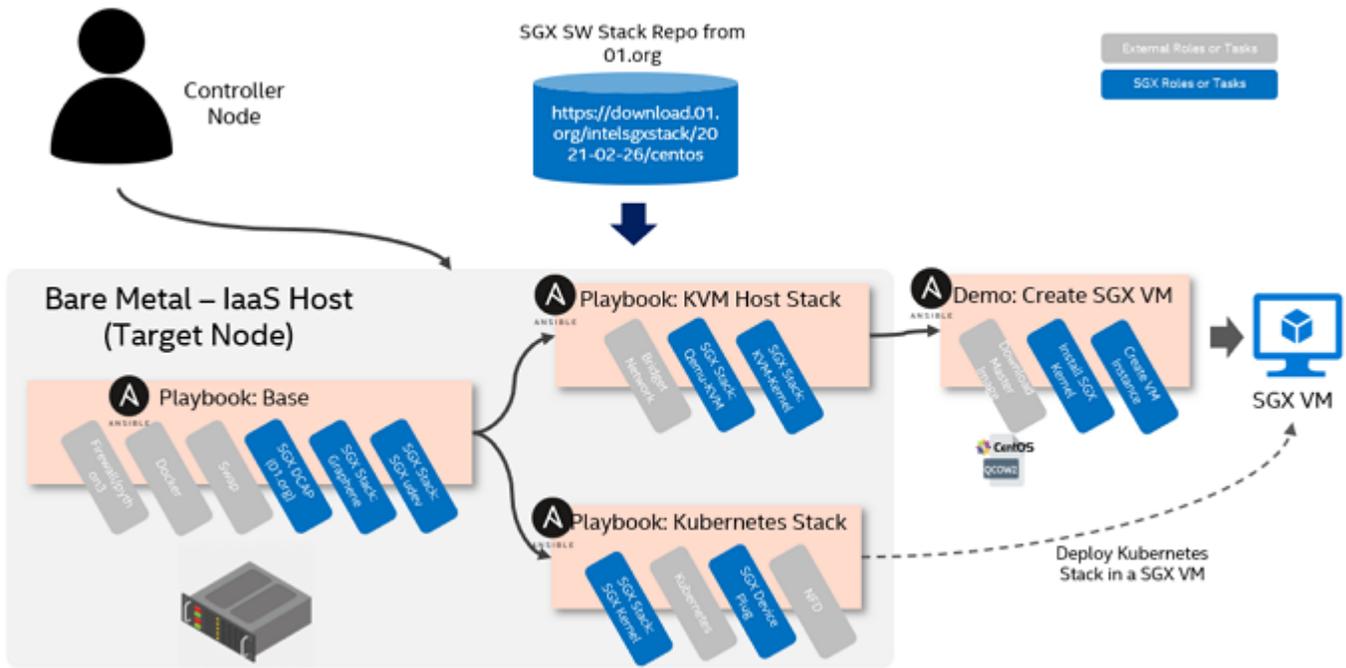
Follow the instructions from [here](#) for installing device plugins.

Ansible Mass Deployment

To facilitate the mass deployment in data center, this guide leverages [Ansible](#). Ansible is an IT automation tool that can configure systems, deploy software, and orchestrate more advanced IT tasks. Please get familiar with the “Ansible concepts” from the [official documentation](#).

High Level Flow

The ansible playbooks are running from the controller node within the same network as the target node, and the whole deployment workflow is as follows:



There are two types of playbook:

- Stack playbooks (in stacks directory)

Deploy KVM host or Kubernetes stacks on target bare metal node. The base playbook provides some common configurations or installations for both stacks. The Kubernetes stack could also be deployed in an SGX VM.

- Demo playbooks (in demos directory).

Assumptions & Limitations

- *The target node has been provisioned with RHEL via OS mass provisioning tool. Only releases 8.4/8.5 are supported.*
- *The target node could be the 3rd Generation Intel® Xeon® Scalable Processors server or later platform with SGX enabled in BIOS.*

Prerequisites

Please read the [Prerequisites](#) for both controller node and managed nodes.

Download the compressed file for ansible roles/playbooks from <https://download.01.org/intelsgxstack/2021-12-08/sgx-cloud-stack-deploy.tar.bz2>, and extract it on the control node.

```
tar xjvf sgx-cloud-stack-deploy.tar.bz2
```

Run Ansible in Docker on controller node

Prerequisites: Docker is installed on the controller node and the user is a member of the `docker` group.

The `docker-playbook.sh` script allows you to run `ansible-playbook` from a docker container. The container `ansible-playbook` will be built at the initial running of `docker-playbook.sh`. You can also use the following command to re-generate the container:

```
./docker-playbook.sh rebuild
```

NOTE: Container runs as user `sgxdev`. If you see errors such as:

```
docker: Error response from daemon: OCI runtime create failed: container_linux.go:348: starting container process caused "chdir to cwd (\"/home/sgxdev/deployment\") set in config.json failed: permission denied": unknown.
ERRO[0000] error waiting for container: context canceled: _
```

Please make sure the folder you are running the shell script from has sufficient permissions:

```
chmod o+rx .
```

Setup an SSH account on the target managed nodes

The managed nodes are network devices (and/or servers) you manage with Ansible. Managed nodes are also sometimes called "hosts". You do not need to install any software on the target managed nodes beyond setting up an SSH account via `ssh-copy-id <user@target_node_address>`.

For example, to set up an SSH account on a target node called `node1.some-cluster.com`:

- On the target managed node, create an admin user:

```
sudo adduser sgxdev
sudo usermod -aG wheel sgxdev
sudo passwd sgxdev
```

- On the control node, setup SSH passwordless login:

```
# If you already have an ssh key, you can skip the ssh-keygen command
ssh-keygen
ssh-copy-id sgxdev@node1.some-cluster.com
```

- Verify SSH based passwordless login:

```
ssh sgxdev@node1.some-cluster.com ls
```

NOTE: For the `ssh-copy-id` command to work the managed node has to have SSH password authentication enabled, this can be verified by running:

```
sudo cat /etc/ssh/sshd_config | grep PasswordAuthentication
```

NOTE: Passwordless login does not mean passwordless sudo privileges. For that, on each managed node you need to include the line `%sgxdev ALL=(ALL) NOPASSWD: ALL` in the file `/etc/sudoers` via `visudo`. Alternatively, on each managed node create a separate file with the line and place the file in the folder `/etc/sudoers.d`.

```
sudo bash -c 'echo "%sgxdev ALL=(ALL) NOPASSWD: ALL" > /etc/sudoers.d/sgxdev'
```

NOTE: Please refer to the Ansible document [check your SSH connections](#) for detail.

Define the inventory

Inventories are commonly formatted in either INI or YAML.

On the control node, create an inventory file in the default location:

```
sudo mkdir -p /etc/ansible  
echo <name-or-ip-of-my-system> | sudo tee /etc/ansible/hosts
```

Please refer to the Ansible document [How to build your inventory](#).

Config Variables

Variables are passed into the playbooks or roles via options "-e". Please evaluate the variables in all new or existing roles or playbooks carefully. The following table gives the variables most often used for deploying SGX cloud stacks.

name	value	default value	applicable stack	note
net_ethernet_dev	The name string of bridge slave ethernet interface	enp1s0	kvm-host	Setup bridge network br0 on net_ethernet_dev, Note: Please check the interface name on your target managed node.
sgx_repo_file_url	URL of repo file to be used for deployment	https://download.01.org/intelsgxstack/2021-12-08/rhel/intelsgxstack.repo	all	Specify a different DNF Repo URL if needed.
kubernetes_role	Deploy master/node	master	Kubernetes	
k8s_username	Non-root user account name	k8s_master	Kubernete master	
k8s_password	Non-root user accout password	k8s_master	Kubernetes master	
k8s_web_ui_create	Create Kubernetes Web Dashboard	true	Kubernetes master	
k8s_web_ui_manifest_file	Web Dashboard version to create	https://raw.githubusercontent.com/kubernetes/dashboard/v2.2.0/aio/deploy/recommended.yaml	Kubernetes master	
k8s_web_ui_enable_skip_login	Allow read-only login to Dashboard without user credentials	true	Kubernetes master	
k8s_web_ui_banner	Optionally displayed banner	Kubernetes with SGX support	Kubernetes master	
k8s_web_ui_port	Assign fixed Web Dashboard port	random	Kubernetes master	There are no range checks, so you should know what you are doing

Config Variables for Virtual Machines Creation

Variable	Description	Default
<code>vm_action</code>	Create or destroy action, one of [create destroy]	<code>create</code>
<code>vm_location</code>	The directory of VM image on the KVM host	<code>/var/lib/libvirt/images/</code>
<code>vm_image</code>	The file name of VM image on the KVM host	<code>CentOS-8-GenericCloud-8.4.2105-20210603.0.x86_64.qcow2</code>
<code>vm_image_url</code>	The remote download URL for VM image	<code>https://cloud.centos.org/centos/8/x86_64/images/CentOS-8-GenericCloud-8.4.2105-20210603.0.x86_64.qcow2</code>
<code>vm_os_variant</code>	VM os variant. Should be one of reported by "osinfo-query os" [^1]	<code>rhel8.4</code>
<code>vm_root_password</code>	The root password for demo VM image	<code>Intelinux123</code>
<code>vm_resize_partition</code>	Disk partition to resize [^2]	<code>/dev/sda1</code>
<code>vm_kernel</code>	Linux kernel for VMs [^3]	<code>kernel</code>
<code>vm_guests</code>	List of VM configurations	See below "VM Template"
<code>vm_repo_file_url</code>	URL of an additional repo file to be used to install packages to VM (optional)	None
<code>vm_no_log</code>	Keep sensitive values out of logs	<code>true</code>
<code>copy_host_files</code>	Copy files from Host to VMs [^4]	<code>empty</code>
<code>sm_username</code>	User name (ID) for RHEL subscription manager [^5]	None
<code>sm_password</code>	User password for RHEL subscription manager [^5]	None

[^1]: Currently supported values are: `rhel8.4`, `rhel8.5` and `centos`

[^2]: EFI based images typically need `/dev/sda3`

[^3]: If `vm_kernel` is specified as an empty string, no new kernel is installed in VMs.

[^4]: Implemented via `virt-customize --copy-in`, for details see <https://libguestfs.org/virt-customize.1.html>

[^5]: Values for username/password obtained by subscribing to Red Hat at <https://www.redhat.com>

Variables for vm_guests

Name	Description	Default	Notes
<code>name</code>	domain name of the VM	must be specified	compulsory, if domain already exists then new VM creation is ignored
<code>hostname</code>	hostname of the VM	must be specified	compulsory
<code>memory_kib</code>	amount of RAM for VM	must be specified	compulsory, no checks for overcommitment
<code>vcpu</code>	number of CPUs	2	no checks for overcommitment
<code>qcow_disk_file</code>	disk file name	<code>name.qcow2</code>	
<code>disk_size</code>	disk size in GB	none	Shrinking of an image is ignored. No resizing of VM image if <code>disk_size</code> not defined
<code>mac_address</code>	MAC address of VM	'52:54:00:xx:yy:zz' with xx:yy:zz randomized	
<code>autostart</code>	start VM automatically after host reboot	yes	optional
<code>running</code>	start VM after VM created	yes	optional
<code>epc_size</code>	SGX EPC size	512M	
<code>customize_extra</code>	commands to pass to <code>virt-customize</code>	empty	optionally customize individual VMs (create users, ssh access,...) [^6]

[^6] See the complete list of available commands here: <https://libguestfs.org/virt-customize.1.html>

VM Template

```
vm_guests:
- name : vm01
  hostname : 'test-vm01'
  memory_kib: 16777216
  vcpu: 4
  qcow_disk_file: 'test-vm01.qcow2'
  mac_address: '52:54:00:c5:9e:e6'
  autostart: 'yes'
  epc_size: '256M'
  disk_size: '12G'
- name : vm02
  hostname : 'test-vm02'
  memory_kib: 16777216
  vcpu: 8
  qcow_disk_file: 'test-vm02.qcow2'
  mac_address: '52:54:00:c5:9e:e7'
  autostart: 'no'
  epc_size: '512M'
  disk_size: '20G'
```

Deploy the KVM host virtualization stack

- Guest VMs use the bridge network on the KVM host; the bridge will be created by the playbook on top of the host's Ethernet interface. By default it uses the value "enp1s0", if your network device name is different, you must specify it via the `net_ethernet_dev` variable. For example:

```
./docker-playbook.sh -i inventory/nodes stacks/kvm-host.yml -e "net_ethernet_dev=<host-ethernet-device-name>"
```

- The default user that will be used by the playbook is "sgxdev", if want to use a different user, please specify via "-u <user>":

```
./docker-playbook.sh -i inventory/nodes -u <user> stacks/kvm-host.yml -e "net_ethernet_dev=<host-ethernet-device-name>"
```

NOTE: Whether using the user `sgxdev` or other, please make sure the `<user>` exists on managed node and is part of `sudo` group. Ansible playbooks need to run with administrator privilege, so either use `-K` for password prompt or modify `sudoers` for passwordless `sudo`. Note that `-K` can only be used when managing a single node, or all managed nodes use identical `sudo` password for `<user>`. Read more details in the [Ansible Documentation](#).

- By default, the SGX RPM packages will be downloaded from one of the [public repositories](#), based on the OS installed on managed nodes. This address can be customized via variable the `sgx_repo_file_url`:

```
./docker-playbook.sh -i inventory/nodes stacks/kvm-host.yml \  
-e "net_ethernet_dev=<host-ethernet-device-name>" \  
-e 'sgx_repo_file_url="https://my-dnf-server-address.com/my-repo-file.repo"'
```

Deploy Kubernetes stack

The Kubernetes stack can be deployed via the `stacks/kubernetes.yml` playbook:

```
./docker-playbook.sh stacks/kubernetes.yml
```

By default, the command will setup a Kubernetes master node with SGX and the NFD (Node Feature Discovery) device plugin. By default it will also install web-ui dashboard.

The playbook also supports setting up a Kubernetes slave node and joining it into an existing Kubernetes cluster via a customized join command, for example:

```
ansible-playbook stacks/kubernetes.yml -e "kubernetes_role=node" \  
-e 'kubernetes_join_command="kubeadm join 10.239.85.45:6443 \  
--token 1ddbem.v1tikqrbutr6sq7e \  
--discovery-token-ca-cert-hash sha256:b02cba3d2c98aebbd0f26be97af19fee68ae3fcb1f2a04b684160f534edb32"'
```

NOTE: The parameter `kubernetes_join_command` will not work with `./docker-playbook.sh` when running `ansible-playbook` tool within `docker`. Please either run `ansible-playbook` on the controller node directly without `docker` or modify the `kubernetes_join_command` in `stacks/kubernetes.yml` without passing via command line. Another option is to use `./docker-playbook.sh` and pass `kubernetes_join_command` via a playbook file. See the file `demos/join-k8-master.yml` for an example.

In the deployment of the slave node, it will assume SGX and the NFD (Node Feature Discovery) device plugin were already installed. Refer to the [SGX plugin documentation](#) for more information.

After the SGX node has joined the cluster, the NFD device plugin will discover the SGX and SGXLC features.

Demos and utilities

Create/Destroy SGX VM

Two playbooks are provided to demonstrate creation of virtual machines:

```
demons/create-sgx-vm-rhel8.4.yml  
demons/create-sgx-vm-rhel8.5.yml
```

For legacy purposes the following playbook is also provided:

```
demons/create-sgx-vm-centos8.yml
```

The number of VMs, host names, memory sizes, vCPU counts, EPC sizes, root password and other options can be customized by editing the playbook files. You need to provide repository file for the SGX kernel. This can be done either by providing URL of the repository or by copying the repository file from the KVM host to the VMs via `"copy_host_files"`. It is safer to provide the URL, this is demonstrated below.

To create RHEL virtual machines you also need to provide RHEL credentials. The credentials are obtained via registration/subscription to RHEL. The playbook file assumes the cloud image file is already placed in the host folder as specified by `"vm_location"`. The location defaults to `/var/lib/libvirt/images`. Otherwise, you need to provide URL of the image. Beware that Red Hat provided download URLs expire quickly, so you may need to host the downloaded cloud image yourself.

Example command to create RHEL8.4 Virtual Machines on KVM host:

```
./docker-playbook.sh demos/create-sgx-vms-rhel8.4.yml
-e "vm_repo_file_url=https://download.01.org/intelsgxstack/2021-12-08/rhel/intelsgxstack.repo"
-e "sm_username=<my_rhelusername>" -e "sm_password=<my_rhelfpassword>"
```

The virtual machines will be registered with Red Hat, as can be verified by viewing <https://access.redhat.com/management/systems>

Example command to destroy all VMs created by the file `demos/create-sgx-vms-rhel8.4.yml`:

```
./docker-playbook.sh demos/create-sgx-vms-rhel8.4.yml -e "vm_action=destroy"
```

The virtual machines will be un-registered from Red Hat, as can be verified by viewing <https://access.redhat.com/management/systems>

Create/Destroy SGX VM ready to be managed by Ansible

A playbook that demonstrates creation of VMs with user `sgxdev` with passwordless ssh connection and passwordless sudo privileges. All you need to do is provide your own public ssh key. The playbook also demonstrates how to install additional packages. Edit the playbook file `demos/create-sgx-k8s-vms-rhel8.4.yml` with your public ssh key and run the playbook:

```
./docker-playbook.sh -i <host> demos/create-sgx-k8s-vms-rhel8.4.yml \
-e "vm_repo_file_url=https://download.01.org/intelsgxstack/2021-12-08/rhel/intelsgxstack.repo" \
-e "sm_username=<my_rhelusername>" -e "sm_password=<my_rhelfpassword>"
```

NOTE: The virtual machines created at this point are simple cloud images, with no Kubernetes software deployed. The Kubernetes deployment will be demonstrated in the next step.

Destroy all created VMs once they are not needed:

```
./docker-playbook.sh demos/create-sgx-k8s-vms-rhel8.4.yml -e "vm_action=destroy"
```

Create/Destroy Kubernetes cluster using virtual machines

After you have successfully created virtual machines using `demos/create-sgx-k8s-vms-rhel8.4.yml`, it is easy to set up a simple Kubernetes cluster. The first step is to create an inventory file `host-k8s-cluster` containing the IP addresses of our virtual machines (use the IP addresses as reported by the VMs creation):

```
$ cat host-k8s-cluster
10.165.59.128 ansible_user=sgxdev kubernetes_role=master
10.165.57.88  ansible_user=sgxdev kubernetes_role=node
10.165.57.93  ansible_user=sgxdev kubernetes_role=node
```

Next, deploy Kubernetes on all target nodes. The deployment will also create a default account for a user `"k8s_master"` with the password `"k8s_master"`.

NOTE: You can override the default account values and specify your own values.

```
./docker-playbook.sh -i host-k8s-cluster stacks/kubernetes.yml \
-e "k8s_username=joe" -e "k8s_password=intelsgx" \
-e "sgx_repo_file_url=https://download.01.org/2021-12-08/rhel/intelsgxstack.repo"
```

The deployment will print out the Kubernetes join command.

At this point you can query the cluster. This can be done via command line interface from a terminal or via web ui dashboard.

Terminal:

Log into `k8_vm_master` (use IP address as reported for the `k8_vm_master`).

Example:

NOTE: You can use the dashboard to view the nodes.

Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)	Memory limits (bytes)	Pods	Created
a4b101722cac-k8s-vm-rhel-worker1	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux feature.node.kubernetes.io/cpu-cpuid.ADX: true Show all	True	100.00m (1.25%)	100.00m (1.25%)	50.00Mi (0.32%)	50.00Mi (0.32%)	4 (3.64%)	an.hovt.399
a4b101722cac-k8s-vm-rhel-worker2	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux feature.node.kubernetes.io/cpu-cpuid.ADX: true Show all	True	100.00m (1.25%)	100.00m (1.25%)	50.00Mi (0.32%)	50.00Mi (0.32%)	4 (3.64%)	an.hovt.399
a4b101722cac-k8s-vm-rhel-master	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux feature.node.kubernetes.io/cpu-cpuid.ADX: true Show all	True	950.00m (23.75%)	100.00m (2.50%)	290.00Mi (1.84%)	390.00Mi (2.48%)	13 (11.82%)	an.hovt.399

Cleanup hosts

Use the following playbook to do a partial cleanup for SGX stack deployment:

```
./docker-playbook.sh demos/cleanup-hosts.yml
```

The playbook will not uninstall any kernel. You can customize the cleanup by editing the script file `files/cleanup-hosts.sh`.

ATTENTION: This will remove docker installation on all managed nodes.

Show the IP address for all VMs

Use the following playbook to show the IP address for all VMs on all managed nodes:

```
./docker-playbook.sh demos/show-vm-ip-address.yml
```

Verify the VMs were created and are running

You can verify that the VMs were created successfully and are running by running the following command from the managed node:

```
sudo virsh list --all
```

Known Issues

1. Kubernetes stacks fail to be deployed

If the managed node is the type of master role, run `kubeadm reset default` to clean up the previous Kubernetes deployment. If the managed node is the type of slave role, make sure the `kubernetes_join_command` was provided.

2. Error passing `kubernetes_join_command` when using `docker-playbook.sh`

When deploying the SGX stack to a Kubernetes slave node, the join command line needs to be passed via `-e`. It will fail if using `docker-playbook.sh` which is running `ansible-playbook` in `docker`. There are three workarounds below, either one should work:

- Run `ansible-playbook` on the controller node directly instead of `docker-playbook.sh`. However, this requires installation of Ansible and all dependencies like `Dockerfile` does for `docker-playbook`. Ansible installation depends on the control node OS. For example, the following sequence of commands would install Ansible on RHEL:

```
sudo dnf install python3 python3-pip ansible
sudo python3 -m pip install -r ./requirements_pip.txt
ansible-galaxy install -r ./requirements_role.yml
ansible-galaxy collection install -r ./requirements_collection.yml
```

- Modify the `kubernetes_join_command` in `stacks/kubernetes.yml` instead of passing it via `-e`.
- Run a playbook file with an explicit join command initialized to the desired value. See `demos/join-k8-master.yml` as an example.

3. Red Hat does not show host names of registered virtual machines

You can check your registered RHEL systems at <https://access.redhat.com/management/systems>. You will see all your registered virtual machines, however they may be showing the name `(none)`. To be able to identify the virtual machines by their host names, you need to re-register each virtual machine with the credentials obtained from Red Hat registration:

```
subscription-manager register --username=<username> --password=<password> --auto-attach --force
```

4. QEMU SGX arguments not backwards compatible

Previous versions of VMs with SGX support may not start. In this case, you need to modify/edit the VM XML file to reflect the new QEMU command syntax.

Modify the following old style syntax:

```
<qemu:commandline>
  <qemu:arg value='-cpu' />
  <qemu:arg value='host,host-phys-bits,+sgx,+sgx-debug,+sgx-exinfo,+sgx-kss,+sgx-mode64,+sgx-provisionkey,
+sgx-tokenkey,+sgx1,+sgx2,+sgxlc' />
  <qemu:arg value='-object' />
  <qemu:arg value='memory-backend-epc,id=mem1,size=512M,prealloc' />
  <qemu:arg value='-sgx-epc' />
  <qemu:arg value='id=epc1,memdev=mem1' />
</qemu:commandline>
```

to look like this:

```
<qemu:commandline>
  <qemu:arg value="-cpu" />
  <qemu:arg value="host,+sgx,+sgx-debug,+sgx-exinfo,+sgx-mode64,+sgx-provisionkey,+sgx-tokenkey,
+sgx1,+sgx2,+sgxlc" />
  <qemu:arg value="-object" />
  <qemu:arg value="memory-backend-epc,id=mem1,size=512M,prealloc=on" />
  <qemu:arg value="-M" />
  <qemu:arg value="sgx-epc.0.memdev=mem1" />
</qemu:commandline>
```

The editing can be done either by editing the VM XML file directly by using *Virtual Machine Manager* GUI editor, or use command line editor:

```
# virsh edit vm-name
```

For more information on SGX QEMU arguments, please consult <https://www.qemu.org/docs/master/system/i386/sgx.html>

Additional Information

Visit us at <https://01.org/intel-sgx-stacks/>

Connect with us on the Linux Stacks for Intel® SGX mailing list to keep up to date on releases and features!

You can sign up here:

<https://lists.01.org/postorius/lists/intel-sgx-stacks.lists.01.org/>

Or email us at intel-sgx-stacks@lists.01.org.

Statement

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. *Other names and brands may be claimed as the property of others.