Linux Stacks for Intel® SGX

2021.02.26 Release

Linux Stacks for Intel® SGX

2021.02.26 Release

Introduction

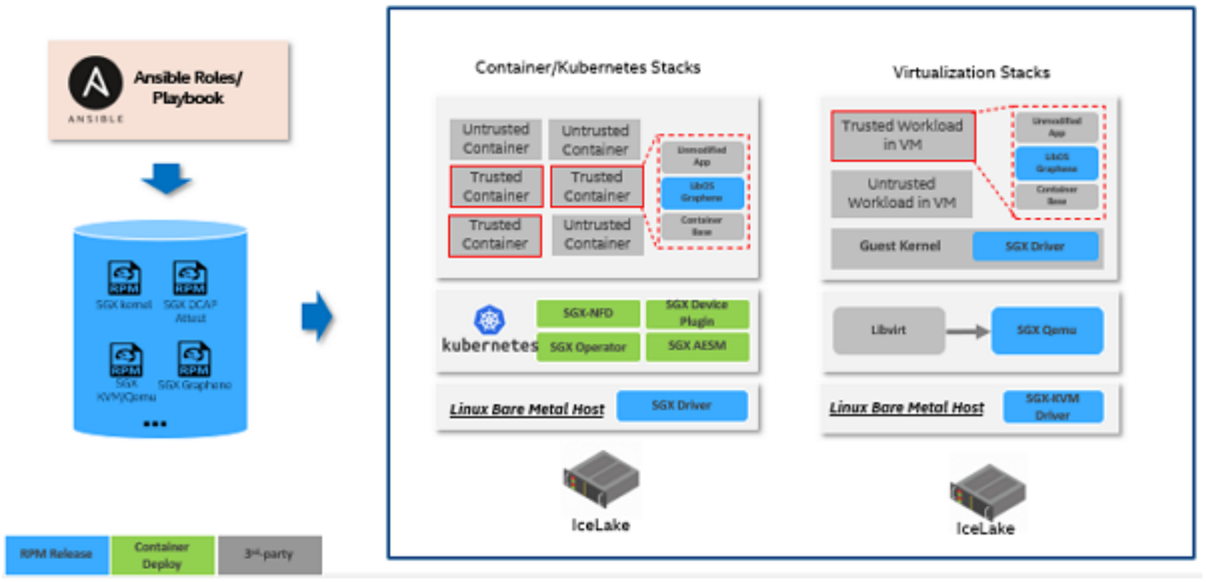
This document introduces the setup and deployment of the Linux* Stacks for Intel® SGX on a 3rd Generation Intel® Xeon® Scalable Processors server.

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.



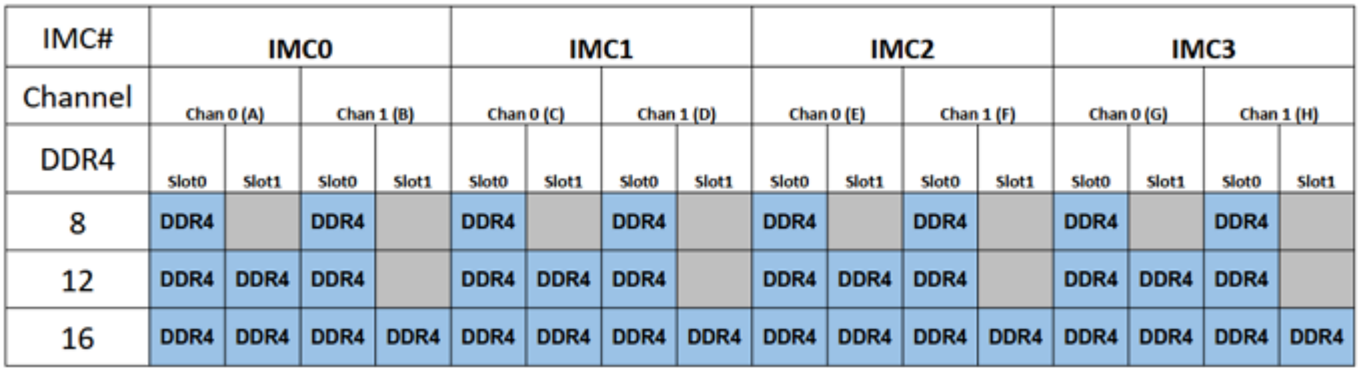
*Note: The Stacks are presently implemented only for the CentOS 8.2/8.3. Operating System**

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	Slot0	Slot1	Slot0	Slot1	Slot0	Slot1	Slot0	Slot1	Slot0	Slot1	Slot0	Slot1	Slot0	Slot1
8	DDR4		DDR4		DDR4		DDR4		DDR4		DDR4		DDR4		DDR4	
12	DDR4	DDR4	DDR4		DDR4	DDR4	DDR4		DDR4	DDR4	DDR4		DDR4	DDR4	DDR4	
16	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	DDR4	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/2021-02-26/centos`.

Component	Package Name	Description
linux-5.11-rc5	kernel	Kernel for bare metal or Kubernetes with in-kernel SGX support
kvm-sgx-kernel-5.11.0-rc3	kvm-sgx-kernel	Kernel for KVM host
linux-sgx-sdk-2.13	linux-sgx-sdk	SGX SDK/PSW
qemu-sgx-v5.2.0	qemu-kvm	QEMU KVM with SGX
graphene	graphene	Graphene LibOS

NOTE::

- Both kernels will create SGX with 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 intelsgxstack.repo file from `https://download.01.org/intelsgxstack/2021-02-26/centos` and copy it to the appropriate directory for your system/distribution. More details here

KVM Stack

Setup the SGX KVM host

You can now install KVM SGX kernel manually via:

```
sudo dnf install kvm-sgx-kernel
```

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

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

Create SGX VM

To install Virtual Machines, you will generally need to follow the instructions from here.

Of course, there is no need to configure/build the kernel and QEMU, as this has already been done. However, with the new KVM kernel please 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_virt_epc", "/dev/sgx_provision"
]
```

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 SGX kernel on a Kubernetes host

A Kubernetes host can be a VM or a bare metal host with SGX support. Since SGX in-kernel driver is already integrated in the 5.11.0 kernel, install the kernel with:

```
sudo dnf install kernel-5.11.0-rc5.2.el8
```

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

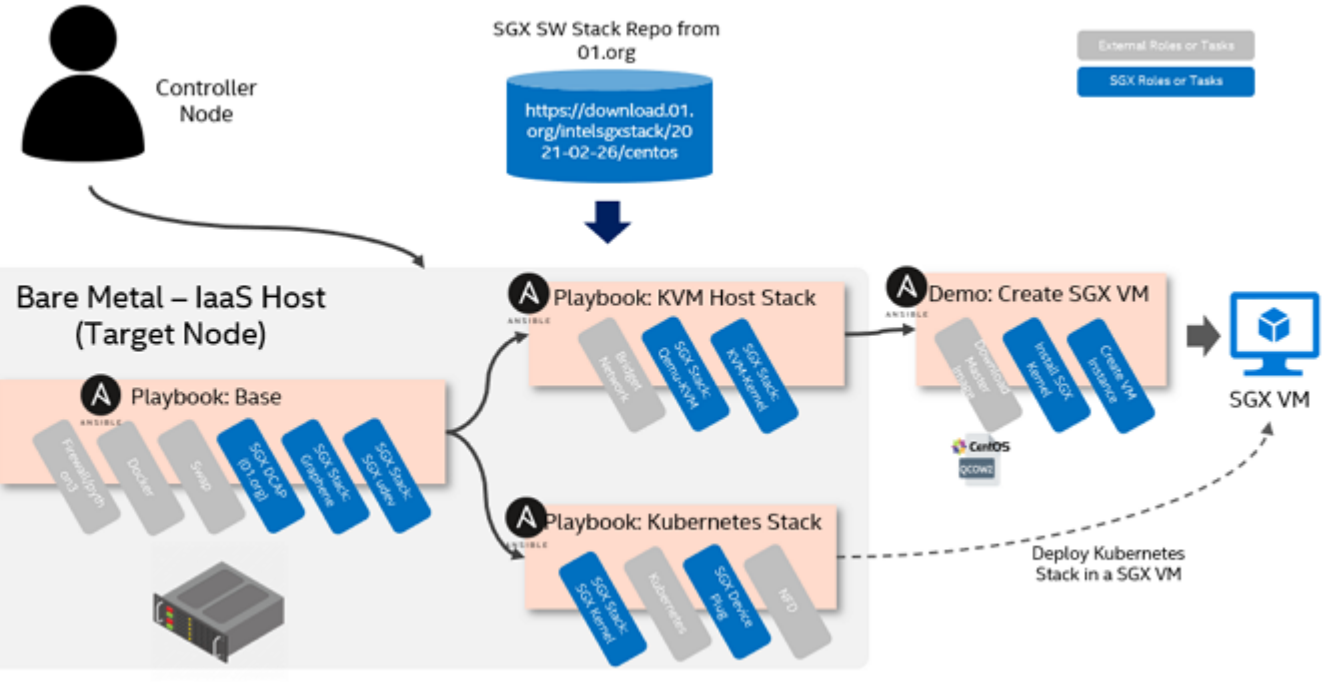
Follow the instructions from here for installing device plugins.

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).

Assumption & Limitations

- *The target node has been provisioned with CentOS 8.2 via OS mass provisioning tool.*
- *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/2021-02-26/centos/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

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
```

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
  # For some distributions you may need to use the group "sudo" instead of "wheel"
  sudo usermod -aG wheel sgxdev
  # Some distributions ask the user to create a password when adding the user, if
  # that was not the case, create a password for the new user
  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: 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.***
`repo_server_url`	The repo server URL when `dnf_repo` is `repo_server`	`https://download.01.org/intelsgxstack/2021-02-26/centos`	all	Specify a different DNF Repo URL if needed.
`k8s_role`	`master|slave`	`master`	Kubernetes	If deploying the master node, the NFD and SGX device plugins will be installed. If deploying slave nodes, please specify the `kubernetes_join_command` to join the cluster.
`kubernetes_join_command`	The join command string for slave nodes	dummy join command	Kubernetes slave	Create with: `kubeadm token create --print-join-command` as admin.

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 sgxdev 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, since the ansible playbook needs to run with administrator privliege. Read more details in the Ansible Documentation.

- By default, the SGX RPM package will be downloaded from the repositories at https://download.01.org/intelsgxstack. This address could be customized via variable repo_server_url:

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

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.

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 "k8s_role=slave" -e 'kubernetes_join_command="kubeadm join 10.239.85.45:6443 --token 1ddbem.v1tikqrbutr6sq7e --discovery-token-ca-cert-hash sha256:b02cba3d2c98aebbd0f26be97af19fee68ae3fcb1f2f2a04b684160f534edb32"'
```

NOTE: The parameter `kubernetes_join_command` will not work with `./docker-playbook.sh` when running `ansible-playbook` tool within docker. So 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.

In the deployment of the slave node, it will assume SGX and the NFD (Node Feature Discovery) device plugin were already installed. Please 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

On an SGX KVM host, you can continue to create SGX VM via demo playbook:

```
./docker-playbook.sh demos/create-sgx-vm.yml
```

The host name, memory size, vCPU count, and EPC size can be customized at `demos/create-sgx-vm.yml`, and the default password for user `root` is `Intelinux123`.

Also, you can destroy the SGX VMs via:

```
./docker-playbook.sh demos/destroy-sgx-vm.yml
```

Cleanup hosts

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

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

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 SGX 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. VM guest with SGX fail to be created

Executing a Host update *will* replace the kvm-sgx-kernel 5.11.0-rc3 with *non* KVM kernel kernel-5.11.0-rc5, which will cause VMs to fail. Users should *never* update kernel on the KVM host. To prevent kernel update, exclude kernel from updates:

```
sudo dnf update --exclude=kernel*
```

However, if the Host kernel should be updated, the KVM kernel is still present, but is no longer the default kernel. The remedy here is to make it the default kernel again.

2. Kubernetes stacks fail to be deployed

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

3. Fail pass `kubernetes_join_command` via `-e` when using `docker-playbook.sh`

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

- Run `ansible-playbook` on the controller node directly instead of `docker-playbook.sh`. But it need install all dependencies like Dockerfile does for `docker-playbook`.
- Modify the `kubernetes_join_command` in `stacks/kubernetes.yml` instead of passing it via `-e`.

Statement