APPLICATION NOTE

Intel Corporation

Intel® Speed Select Technology – Base Frequency Configuration Automation on OpenStack* Compute Host

Authors

Martin Kennelly

Ivens Zambrano

Collaborators/Inputs

Sohaib Iqbal

Mathana Nair Sreedaran

David Hunt

Chris MacNamara

Adrian Hoban

Stephen Finucane (Red Hat*)

1 Introduction

Select SKUs of 2nd generation Intel® Xeon® Scalable processor (5218N, 6230N, and 6252N) offer a capability called Intel® Speed Select Technology - Base Frequency (Intel® SST-BF). This document describes an implementation solution that automates the configuration of the Intel® SST-BF feature.

This document describes the Intel® SST-BF configuration and provisioning process using Ansible scripts, including the rationale for the approach. It provides a step-by-step description of the actions taken to automate and ease the configuration of an OpenStack managed node with Intel® SST-BF.

Intel® SST-BF is a CPU feature designed to unlock software bottlenecks. In the same Thermal Design Power (TDP), a subset of cores are selected for higher guaranteed frequency and the remaining cores operated at a reduced frequency. These higher frequency cores can be used to address many use cases, including:

- Network Function Virtualization (NFV) Data Plane, Control Plane and Open vSwitch* (OVS*) use cases
- Pipeline software architectures
- Frequency bound workloads such as software based crypto
- Priority threads for run to completed such as unbalanced downlink or uplink threads
- Packet distribution and workload distribution in software
- Scenarios where polling user space drivers can be consolidated

Automating this feature in orchestrated cloud platforms allows workloads to be placed on optimized nodes to ensure performance and deterministic frequency requirements. Automating the configuration for Intel® SST-BF gains relevance in the data center, where the System Administrator needs to support the configuration of a number of identical nodes on a heterogeneous cloud.

This document covers the following aspects of Intel® SST-BF provisioning, configuration, and usage on an OpenStack environment by using Ansible* to handle the automation process:

- Configuring platform nodes by setting the CPU to the right frequency boundaries via direct CPU interaction using the Kernel file system (sysfs).
- Configuring OpenStack to provide tenants with the option to request and provision workloads on cores configured with different frequency levels.
- Presenting an automation process usage example of a particular OVS-DPDK use case: Pinning OVS-DPDK to a set of cores on a node running OpenStack compute process and sharing the CPU with virtualized workloads.

The resource orchestration layer described in this document is focused on OpenStack, however, the flow can be ported to other resource orchestrators, such as Kubernetes*.

This document is part of the Network Transformation Experience Kit, which is available at: https://networkbuilders.intel.com/

Table of Contents

Figures

Tables

1.1 Intended Audience

This white paper is intended for System Administrators looking for a mechanism to facilitate the deployment and configuration of Intel® SST-BF in OpenStack cloud environments.

1.2 Terminology

Table 1. Terminology

ABBREVIATION	DESCRIPTION
BIOS	Basic Input / Output System
CPU	Resource class representing an amount of dedicated CPUs for a single guest
NFV	Network Function Virtualization
Nova	Compute project for OpenStack [https://docs.openstack.org/nova/latest/]
OpenStack Placement	REST API stack and data model used to track resource provider inventories and usages, along with different classes of resources.
OVS*	Open vSwitch*
PCPU	Resource class representing an amount of dedicated CPUs for a single guest.
PIP	Python* package installer
PMD	Poll Mode Driver
SST-BF	Intel® Speed Select Technology – Base Frequency [https://www.intel.com/content/www/us/en/architecture-and-technology/speed-select-technology-article.html]
Sys-Admin	System Administrator
TDP	Thermal Design Power
Under-cloud	Term describing a group of physical nodes forming a cluster, including hardware platform, network, operating system, and set of configurations.
VCPU	Resource class representing a unit of CPU resources for a single guest approximating the processing power of a single physical processor.

1.3 Reference Documents

Table 2. Reference Documents

NO.	REFERENCE	SOURCE
[1]	Intel® Speed Select Technology – Base Frequency Enhancing Performance Application Note	https://builders.intel.com/docs/networkbuilders/intel-speed-select-technology-base-frequency-enhancing-performance.pdf
[2]	Intel® Speed Select Technology - Base Frequency with Kubernetes* Application Note	https://builders.intel.com/docs/networkbuilders/intel-speed-select-technology-base-frequency-with-kubernetes-application-note.pdf
[3]	Intel® Speed Select Technology - Base Frequency Priority CPU Management for Open vSwitch* (OVS*) Application Note	https://networkbuilders.intel.com/network-technologies/network-transformation-exp-kits#intel-technology-enablement
[4]	Standardize CPU resource tracking	https://blueprints.launchpad.net/nova/+spec/cpu-resources
[5]	OpenStack Traits	https://github.com/openstack/os-traits
[6]	Python* script to configure Intel® SST-BF on the host (sst_bf.py)	https://github.com/intel/CommsPowerManagement/blob/master/sst_bf.py
[7]	Intel® SST-BF Provisioning Artifacts (available on both Ansible and GitHub*)	Ansible: https://galaxy.ansible.com/intel/sst_bf_openstack_setup_automation GitHub: https://github.com/intel/sst-bf-openstack-setup-automation
[8]	OpenStack Placement API	https://docs.openstack.org/placement/latest/

2 Document Overview

Automating configuration processes is fundamental for data center deployment flow. The under-cloud automation described in this document (for configuring Intel® SST-BF) allows the System Administrator (Sys-Admin) to perform a predictable and repeatable group of tasks and to enhance adoption of silicon technologies at scale.

For Intel® SST-BF, there is a series of steps that require full understanding of the system file structure, the distribution of CPU cores, the relationship between the core thread handlers and then, how this information can be presented to the orchestrators.

3 Intel® SST-BF Provisioning Automation

3.1 Prerequisites

To use the Intel® SST-BF functionality, you need:

- Intel® Speed Select - Base Frequency technology is available on select SKUs of 2nd generation Intel® Xeon® Scalable processor (formerly codenamed Cascade Lake).
- Linux* Kernel version 5.1 or above.
- Enable the Intel® SST-BF feature in the BIOS. Follow the steps described in the Intel® SST-BF Enhancing Performance Application Note.
- Access to Python* script to configure Intel® SST-BF on the host [sst_bf.py]
- Python 3.6 or above

To use Ansible automation artifacts, you need:

- Python 3.6 or above
- Ansible 2.5 or above

Required OpenStack and OpenStack project API versions include:

- OpenStack Train
- Placement API 1.6 or above

3.2 Provisioning Automation Flow

The automation of under-cloud provisioning typically takes a series of steps to perform: configure host hardware, set up networking, start node support processes, monitor tools, and set up the cloud process. The flow presented in this document describes how to take an existing Playbook for a resource orchestrator (OpenStack) and insert a group of extra Roles to perform automation.

The Ansible Playbook and Roles steps described in this document can be inserted in the sequence of Ansible Roles used to deploy the resource orchestrator. Figure 1 shows where these automation steps are inserted in the under-cloud provisioning flow.

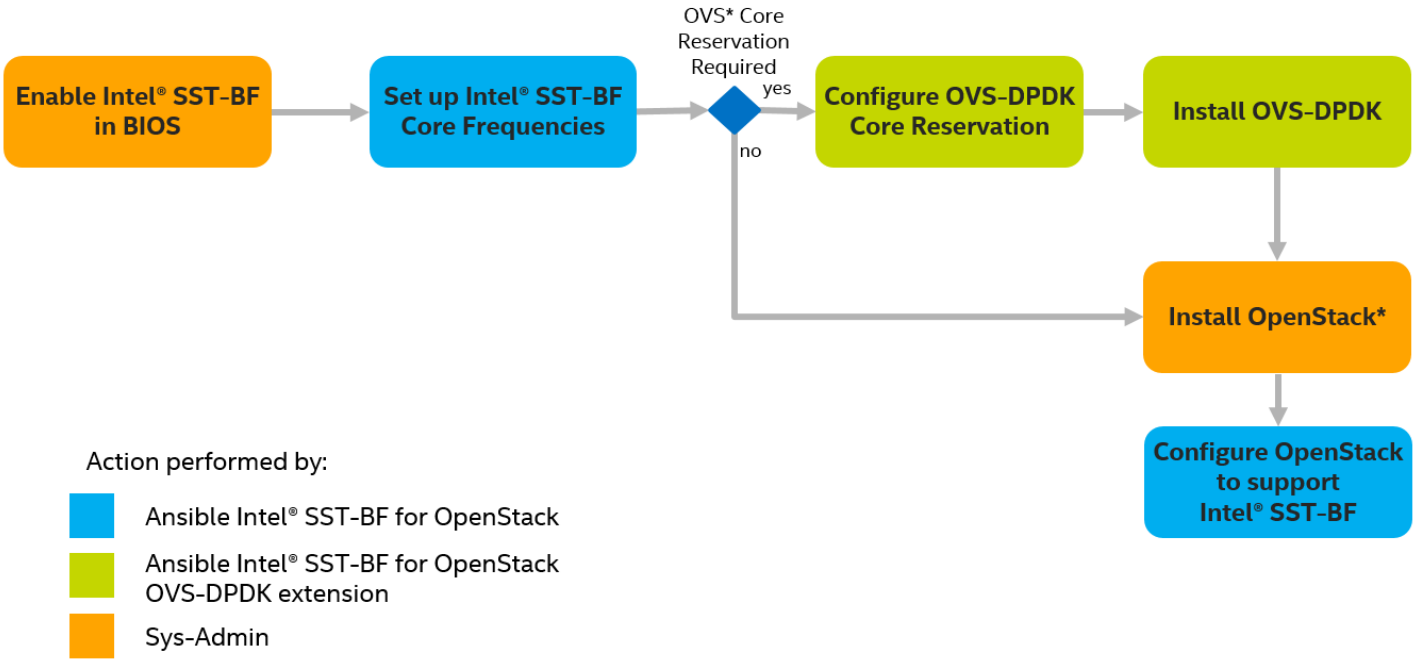

Figure 1. High Level View of Intel® SST-BF Deployment Automation Flow with OpenStack*

The activities in Figure 1 are summarized in Table 3 and are described in detail in upcoming sections of this document, with particular focus on the activities colored in blue.

Table 3. Configuration Activities

ACTIVITY	DESCRIPTION
Enable Intel® SST-BF in BIOS	Initial step required from the System Administrator to change the BIOS configuration as specified in section 3.2 of document [1] in Table 2.
Set up Intel® SST-BF Core Frequencies	Initial automated step to set up the core configuration to high and normal priority frequencies identified by the kernel driver in sysfs. Frequency setting is done through software using the sst_bf.py script, [6] in Table 2.
Configure OVS-DPDK Core Reservation	Before proceeding to the OVS-DPDK installation, it is required to reserve either high or normal priority cores and pin them to an OVS-DPDK Poll Mode Driver (PMD) process. The lcore process will be pinned to a user-defined number of normal priority frequency cores. (Optional step)
Install OVS-DPDK	OVS-DPDK setup and interface assignment is completed during this step. **Note:** OVS-DPDK is not installed by default during an OpenStack installation process. This step is provided to ease the OVS-DPDK installation process as part of this flow. (Optional step)
Install OpenStack	OpenStack is installed by the System Administrator using either an automation set or manual steps.
Configure OpenStack to support Intel® SST-BF	The Intel® SST-BF configuration with the identification of high and normal priority tiers will be appended to the OpenStack Nova configuration file provisioned by the OpenStack installation mechanism on the compute node. The Custom-Traits configuration to the node and the creation of the flavors related to the specific cloud configuration is done as part of this activity.

Figure 2 shows the execution of Ansible Roles from a deployment perspective, which is done from an Ansible Controller that can access a set of compute hosts on an OpenStack cloud. Each of these nodes are called an *OpenStack Nova Target*.

The Ansible execution engine uses a set of artifacts (Playbook, Roles, and Scripts) to complete the activities "Set up Intel® SST-BF Core Frequencies" and "Configure OpenStack to Support Intel® SST-BF" (listed in Table 3) and produce a Nova node configured to provide CPU pinning for the High and Normal Frequency tiers on the CPU and represented in the Nova Configuration file.

The OVS-DPDK artifacts are optional to this flow; they are used only if OVS-DPDK is required on the node and a specific Core Frequency tier is required for it to run. The steps in the flow covered by this group of artifacts are: "Configure OVS-DPDK Core Reservation" and "Install OVS-DPDK" in Table 3.

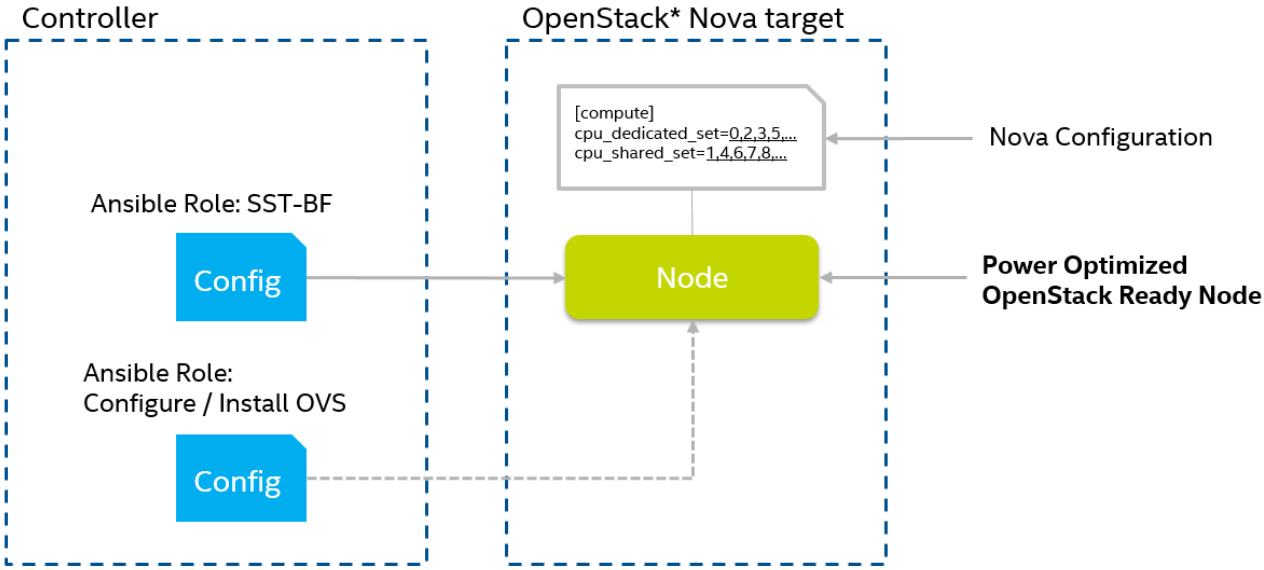

Figure 2. Logical Deployment View

4 OpenStack*: Exposing and Using the Intel® SST-BF Feature

It is important to understand what changes will applied to the system and OpenStack, before getting into the details of the automation steps and the Ansible automation artifacts. To achieve the final goal, the automation scripts:

- Provision OpenStack with a set of Node Traits.
- Provide a characterization of the CPU cores distribution to the OpenStack compute agent (Nova) based on the CPU sysfs structure.

The following two sections describe how OpenStack Node Traits are used and how the CPU core configuration is related to the CPU standard resource tracking in OpenStack.

4.1 Node Traits

In an OpenStack cloud deployment, the different capabilities of a node included in the compute cluster are identified as "Node Traits" in the placement database. The traits are then used to match required workload characteristics to a specific node in the cluster. The standard traits library can be found at https://github.com/openstack/os-traits. The library can be installed through pip and then directly report to OpenStack Placement by Nova.

The traits required to use Intel® SST-BF are added to the placement database using *Custom Traits*. To use custom traits, each trait must be added to the database through the placement API directly or using the OpenStack CLI as shown in the examples below.

Note: A custom trait is **not** part of the standard traits library upstreamed to OpenStack.

The solution described in this document sets the Custom Trait "**CUSTOM_CPU_X86_INTEL_SST_BF"** to clearly identify a node with an **Intel® SST-BF capable CPU.**

Example using the API:

```
#create the trait
PUT /traits/CUSTOM_{name}
#Read the current traits assigned to resource provider identified by UUID
GET /resource_providers/{uuid}/traits
#Response body:
{
    "resource_provider_generation": 0,
    "traits": [
        "CUSTOM_<NAME_A>",
        "CUSTOM_<NAME_B>"
    ]
}
#assign the trait to a resource provider identified by UUID
PUT /resource_providers/{uuid}/traits
#Request body appending the new trait:
{
    "resource_provider_generation": 0,
    "traits": [
        "CUSTOM_<NAME_A>",
        "CUSTOM_<NAME_B>",
        "CUSTOM_<NAME_C>"
    ]
}
```

Note: To use the API, the request header must include the authentication token obtained in an authentication request with the user name, password and project as specified in https://docs.openstack.org/api-quick-start/api-quick-start.html

Example using the CLI:

```
#create the trait
openstack trait create <name>
#read the current traits assigned to the resource provider
openstack resource provider trait list [--sort-column SORT_COLUMN] <uuid>
#assign the trait to the resource provider explicitly appending to the current list obtained in
the previous step
openstack resource provider trait set [--sort-column SORT_COLUMN] [--trait CUSTOM_<trait>]
<uuid>
```

Note: To use the CLI, the System Administrator is required to authenticate with the OpenStack platform using the user name, password and project.

When querying the placement API for trait identification, the information from the providers will be consistent:

```
GET /allocation_candidates?resources=VCPU:8,MEMORY_MB:1024,DISK_GB:4096&required=
CUSTOM_CPU_X86_INTEL_SST_BF
```

It will return a group of nodes matching the request:

```
"provider_summaries": {
    "88a5187d-e0a4-426d-bed4-54e7e89b2adb": {
        "resources": {
            "PCPU": {
                "capacity": 24,
                "used": 4
            },
            "VCPU": {
                "capacity": 56,
                "used": 2
            },
            "MEMORY_MB": {
                "capacity": 10240,
                "used": 0
            }
        },
        "traits": [
            "HW_CPU_X86_SSE",
            "HW_CPU_X86_SSE2",
            "HW_CPU_X86_AVX",
            "HW_CPU_X86_AVX2",
            "HW_CPU_X86_AVX512",
            "CUSTOM_CPU_X86_INTEL_SST_BF",
            "CUSTOM_CPU_FREQUENCY_FIXED_HIGH_DEDICATED", ...
        ]
    },
    "0d684632-eca3-40a9-ab6b-b7457227143c": {
        "resources": {
            "PCPU": {
                "capacity": 24,
                "used": 10
            },
            "VCPU": {
                "capacity": 56,
                "used": 20
            },
            "MEMORY_MB": {
                "capacity": 32768,
                "used": 0
            }
        },
        "traits": [
            "HW_CPU_X86_SSE",
            "HW_CPU_X86_SSE2",
            "HW_CPU_X86_AVX",
            "HW_CPU_X86_AVX2",
            "HW_CPU_X86_AVX512",
            "CUSTOM_CPU_X86_INTEL_SST_BF",
            "CUSTOM_CPU_FREQUENCY_VAR_HIGH_SHARED", ...
        ]
    }
}
```

Similarly, on the CLI:

```
openstack resource provider list --required HW_CPU_X86_AVX --required
CUSTOM_CPU_X86_INTEL_SST_BF
```

4.2 Standardize CPU resource tracking

A mechanism to split the pool of cores in OpenStack is defined in the cpu-resources spec “Standardize CPU resource tracking" and introduced in the OpenStack Train release (for details, see [4] in Table 2).

Previously, the VCPU resource class was used to represent “shared” or “dedicated” CPU cores on a node, meaning it was not possible to mix instance types on one host. With this change, Nova uses two different resource classes in the Placement API for host CPUs:

- VCPU to represent CPU cores that can be shared by multiple workloads
- PCPU to represent dedicated CPU cores

This resource tracking mechanism introduced in OpenStack Train allows the Cloud Admin to define:

- which host cores should be used for dedicated workloads (exclusive core pinning for critical execution)
- which host cores should be used for shared workloads (time sharing core cycles between independent workloads)

On the CPU side of things, the Intel® SST-BF functionality allows you to set up the frequencies of the CPU cores to operate on different tiers and optimize the overall power consumption and critical workload throughput, and also create deterministic frequency boundaries on the CPU cores. The two features (Intel® SST-BF and CPU resources tracking) can be combined to allow the Cloud Admin to set up the OpenStack core classification and match the dedicated list of cores to either high frequency or normal frequency tiers and match the shared list of cores to either high frequency or normal frequency tiers.

Figure 3 illustrates these configuration options.

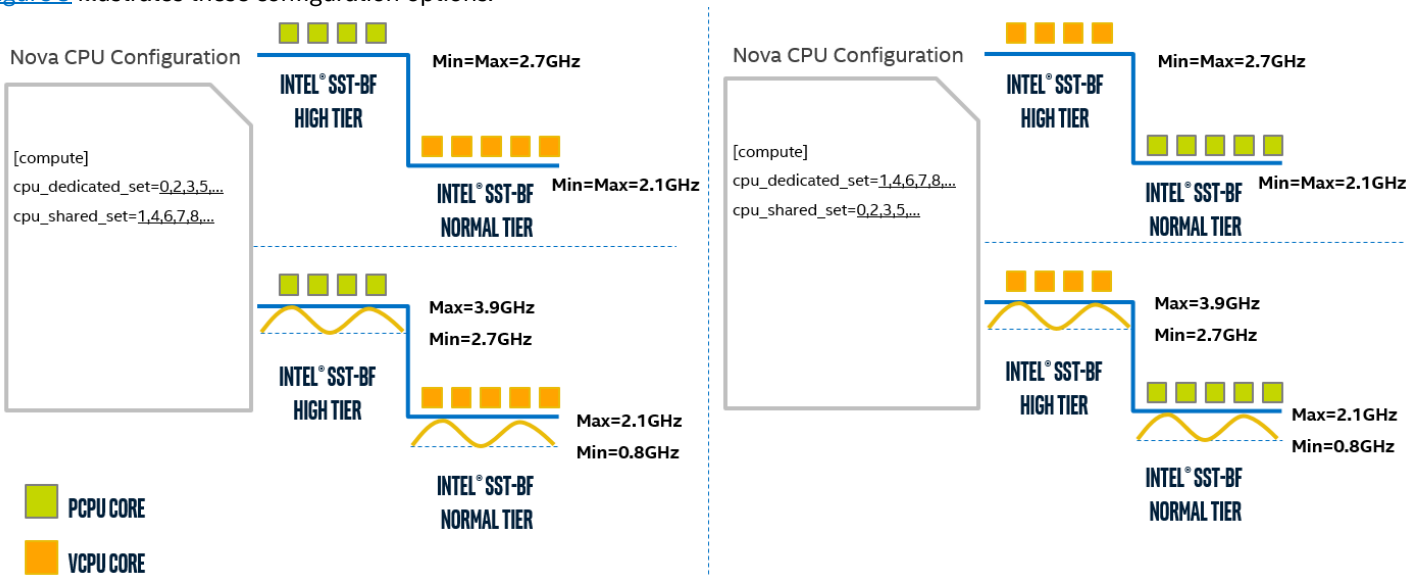

Figure 3. Configuration Options for CPU Resource Tracking

The core distribution ratio for the shared cores (VCPU) is provided under the [DEFAULT] section of the Nova configuration file.

```
[default]
    cpu_allocation_ratio={float}
```

To provide the information to the placement API, the Cloud Administrator must provide the information to the System Administrator in order to define the core distribution. The automation process triggered by the System Administrator takes care of injecting the resource class definition into the Nova configuration file in the [compute] section.

```
[compute]
...
    cpu_dedicated_set={#, #-#, ^#}
    cpu_shared_set={#, #-#, ^#}
...
```

The resource provider will have the following information in the DB:

```
COMPUTE NODE provider
        PCPU:
            total: {integer}
            reserved: 0
            min_unit: {integer > 0 }
            max_unit: {integer <= $total }
            step_size: {integer > 0 && integer <= $total }
            allocation_ratio: 1.0
        VCPU:
            total: {integer}
            reserved: {integer}
            min_unit: {integer > 0 }
            max_unit: {integer <= $total }
            step_size: {integer > 0 && integer <= $total }
            allocation_ratio: {float}
```

Core	sysfs base	sysfs max	sysfs min
0	2100	2100	2100
1	2700	2700	2700
2	2100	2100	2100
3	2100	2100	2100
4	2100	2100	2100
5	2100	2100	2100
6	2700	2700	2700
7	2700	2700	2700
8	2700	2700	2700
9	2100	2100	2100
10	2100	2100	2100
11	2100	2100	2100
12	2100	2100	2100
13	2100	2100	2100
14	2700	2700	2700
15	2100	2100	2100
16	2700	2700	2700
17	2100	2100	2100
18	2100	2100	2100
19	2100	2100	2100
20	2100	2100	2100
21	2100	2100	2100
22	2100	2100	2100
23	2100	2100	2100
24	2100	2100	2100
25	2100	2100	2100
26	2700	2700	2700
27	2700	2700	2700
28	2100	2100	2100
29	2100	2100	2100
30	2100	2100	2100
31	2700	2700	2700
32	2100	2100	2100
33	2700	2700	2700
34	2700	2700	2700
35	2700	2700	2700
36	2100	2100	2100
37	2100	2100	2100
38	2100	2100	2100
39	2100	2100	2100
40	2100	2100	2100
41	2700	2700	2700
42	2100	2100	2100
43	2100	2100	2100
44	2100	2100	2100
45	2100	2100	2100
46	2700	2700	2700
47	2700	2700	2700
48	2700	2700	2700
49	2100	2100	2100
50	2100	2100	2100
51	2100	2100	2100
52	2100	2100	2100
53	2100	2100	2100
54	2700	2700	2700
55	2100	2100	2100
56	2700	2700	2700
57	2100	2100	2100
58	2100	2100	2100
59	2100	2100	2100
60	2100	2100	2100
61	2100	2100	2100
62	2100	2100	2100
63	2100	2100	2100
64	2100	2100	2100
65	2100	2100	2100
66	2700	2700	2700
67	2700	2700	2700
68	2100	2100	2100
69	2100	2100	2100
70	2100	2100	2100
71	2700	2700	2700
72	2100	2100	2100
73	2700	2700	2700
74	2700	2700	2700
75	2700	2700	2700
76	2100	2100	2100
77	2100	2100	2100
78	2100	2100	2100
79	2100	2100	2100

Figure 4. Intel® SST-BF enabled CPU frequencies per core

Using this approach, the System Administrator could set up the cores as they were characterized by the kernel driver matching the Base Frequency of the cores. On an Intel® SST-BF capable CPU with the frequencies split as shown in Figure 4, the Nova configuration file entry could look like:

```
[compute]
cpu_dedicated_set=1,6-8,14,16,26-
27,31,33-35,41,46-48,54,56,66-67,71,73-75
cpu_shared_set=0,2-5,9-13,15,17-25,28-
30,32,36-40,42-45,49-53,55,57-65,68-
70,72,76-79
```

In this case, the "cpu_dedicated_set" holds the high priority tier (2.7 GHz) cores from the Intel® SST-BF enabled CPU, and "cpu_shared_set" holds the list of normal priority tier cores (2.1 GHz). However, the selection of sets depends on the configuration option and is reflected as an OpenStack Node Trait.

In order to use Intel® SST-BF properly, a combination of custom traits is always needed. The trait "**CUSTOM_CPU_X86_INTEL_SST_BF**" identifies the presence of an Intel® SST-BF capable CPU on the node and the extra 4 traits will expose the distribution of the cores between the shared and dedicated lists.

Table 4 describes the set of required custom traits and Figure 5 provides a visual reference.

Table 4. Custom Traits for Intel® SST-BF Configurations

ACTIVITY	DESCRIPTION
CUSTOM_CPU_FREQUENCY_FIXED_HIGH_DEDICATED	The `cpu_dedicated_set` holds the list of cores set to **high** priority frequency. • Min and Max boundary are set to the high BF tier. The `cpu_shared_set` holds the list of cores set to **normal** priority frequency. • Min and Max boundary are set to the normal BF tier.
CUSTOM_CPU_FREQUENCY_FIXED_HIGH_SHARED	The `cpu_dedicated_set` holds the list of cores set to **normal** priority frequency. • Min and Max boundary are set to the normal BF tier. The `cpu_shared_set` holds the list of cores set to **high** priority frequency. • Min and Max boundary are set to the high BF tier.
CUSTOM_CPU_FREQUENCY_VAR_HIGH_DEDICATED	The `cpu_dedicated_set` holds the list of cores set to **high** priority frequency. • Min boundary is set to a given minimum CPU frequency. (Depends on SKU reference and recommended configuration.) • Max boundary is set to the high BF tier. The `cpu_shared_set` holds the list of cores set to **normal** priority frequency. • Min boundary is set to a given minimum CPU frequency. (Depends on SKU reference and recommended configuration.) • Max boundary is set to the normal BF tier.
CUSTOM_CPU_FREQUENCY_VAR_HIGH_SHARED	The `cpu_dedicated_set` holds the list of cores set to **normal** priority frequency. • Min boundary is set to a given minimum CPU frequency. (Depends on SKU reference and recommended configuration.) • Max boundary is set to the normal BF tier. The `cpu_shared_set` holds the list of cores set to **high** priority frequency. • Min boundary is to a given minimum CPU frequency. (Depends on SKU reference and recommended configuration.) • Max boundary is set to the high BF tier.

Figure 5 provides a visual reference to select the appropriate Intel® SST-BF configuration and its corresponding representation as Custom Traits depending on the level of frequency determinism required and the Nova core distribution. Any of the four options described in Table 4 should be always accompanied by the Custom Trait: "**CUSTOM_CPU_X86_INTEL_SST_BF**".

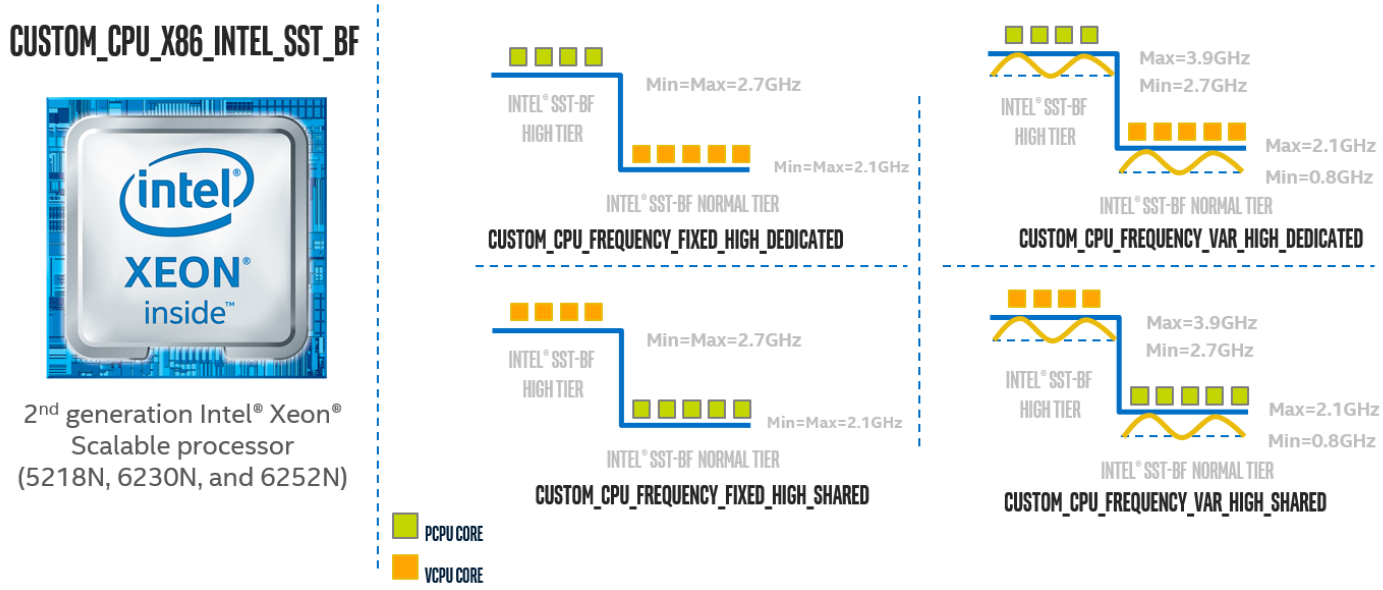

Figure 5. Visual Reference for Nova Node Custom Traits Combinations when using Intel® SST-BF

5 Intel® SST-BF Provisioning Automation for OpenStack* using Ansible*

The configuration of the node characteristics and provisioning of entries for the OpenStack artifacts are done using Ansible by providing the configuration requirements in the form of Ansible Roles. To get the artifacts, refer to link [7] in Table 2 or through Galaxy [https://galaxy.ansible.com/intel/sst_bf_openstack_setup_automation].

The flow shown in color in Figure 6 represents the steps described in this section. The blue boxes are covered by the Ansible Roles to be described. The Ansible Role to deploy OpenStack (orange box) could be integrated as part of the overall flow through the main Ansible Playbook.

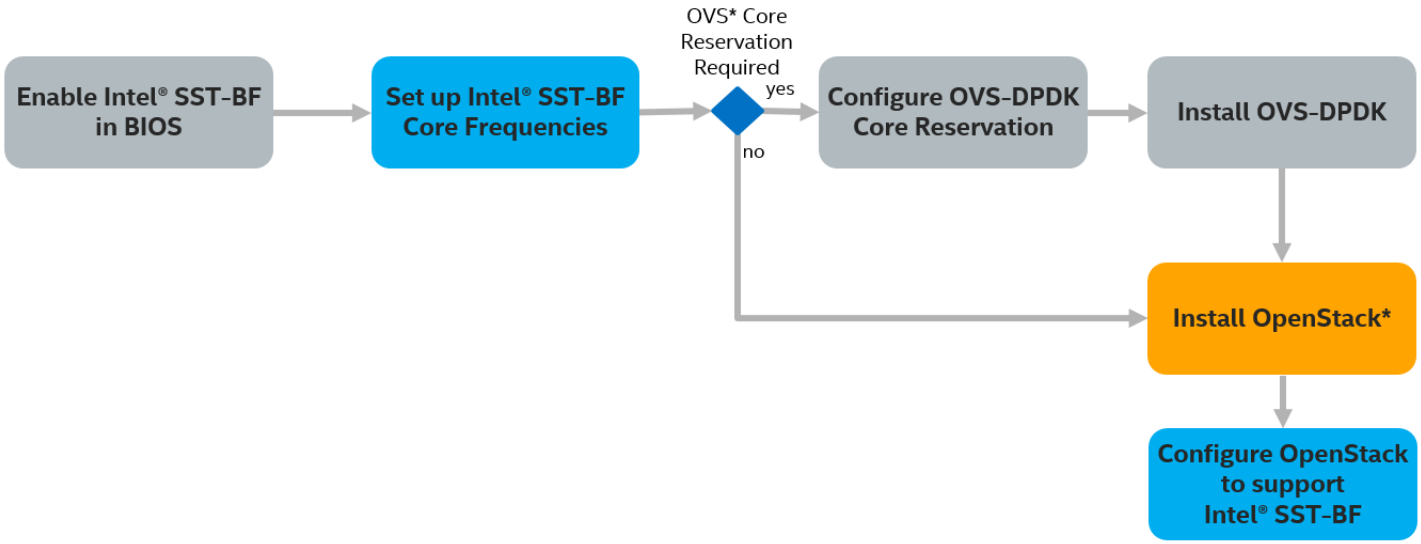

Figure 6. General Deployment Flow for Intel® SST-BF and OpenStack*

Using the Galaxy application in a Linux environment, the Roles can be pulled directly by issuing the command:

```
ansible-galaxy install intel.sst_bf_openstack_setup_automation
```

The Ansible Role utilizes a Python script to configure Intel® SST-BF on the host (see the sst_bf.py documention).

Note: For air-gapped deployments, this repository must be present in /tmp on the target host under the folder name CommsPowerManagement.

The sst_bf.py script will be automatically pulled when triggering the Ansible flow. In the example solution shown below, the high priority cores are set to 2700 minimum/maximum and normal priority cores are set to 2100 minimum/maximum on the target host.

A Playbook example to trigger the configuration of Intel® SST-BF on a host ready for OpenStack would look like this:

```
- name: Configure SST-BF for OpenStack
  hosts: nova_compute
  gather_facts: yes
  user: root
  tasks:
      - name: Set up SST-BF Core Frequencies
        include_role:
          name: "intel.sst_bf_openstack_setup_automation"

# [Insert Role to install OpenStack Nova Compute here]
# OpenStack Nova compute must be installed before running the next task

      - name: Configure OpenStack to Support Intel SST-BF
        vars:
          configure_os_only: true
        include_role:
          name: "intel.sst_bf_openstack_setup_automation"
```

The task "Set up Intel® SST-BF Core Frequencies" configures the BF boundaries using the default configuration described in Table 4 for **CUSTOM_CPU_FREQUENCY_FIXED_HIGH_DEDICATED.**

As shown in Table 5, a set of parameters could be added to this task to override the default values:

Table 5. Set up Intel® SST-BF Core Frequencies Parameters

ACTIVITY	DEFAULT	DESCRIPTION
configure_os_only	false	When true, we write high/normal priority cores lists into the Nova Configuration file, trait info to Placement and set up sst-bf flavors with Intel® SST-BF specific configuration options for future VM requests.
skip_ovs_dpdk_config	true	Skip Open vSwitch*-DPDK.
offline	false	Air-gap deployment. Clone/copy CommsPowerManagement to /tmp of target if true.
sst_bf_profile	FREQUENCY_FIXED_HIGH_DEDICATED	Contains a set of values that control which Intel® SST-BF profile we apply to the target host. The possible values are: FREQUENCY_FIXED_HIGH_DEDICATED FREQUENCY_FIXED_HIGH_SHARED FREQUENCY_VAR_HIGH_DEDICATED FREQUENCY_VAR_HIGH_SHARED This will be translated to the corresponding traits: CUSTOM_CPU_FREQUENCY_FIXED_HIGH_DEDICATED CUSTOM_CPU_FREQUENCY_FIXED_HIGH_SHARED CUSTOM_CPU_FREQUENCY_VAR_HIGH_DEDICATED CUSTOM_CPU_FREQUENCY_VAR_HIGH_SHARED

Note: The Cloud Administrator will inject the username, password, and project into the environment variables to interact with OpenStack as part of the initial flow steps.

The task "Configure OpenStack to Support Intel® SST-BF" will provision the Nova configuration file with the core split details and add the required traits to Placement.

As shown in Table 6, a set of parameters could be added to this task to override the default values:

Table 6. Configure OpenStack* to Support Intel® SST-BF Parameters

ACTIVITY	DEFAULT	DESCRIPTION
configure_os_only	false	When true, we write high/normal priority cores lists into the Nova Configuration file, trait info to Placement and set up sst-bf flavors with Intel® SST-BF specific configuration options for future VM requests.
nova_conf_path	/etc/nova/nova-cpu.conf	Nova Configuration file location.
restart_nova	true	Option to restart nova after configuration changes.
nova_service_name	devstack@nova-cpu.conf	Systemctl nova service name for restart after configuration file changes.

Finally, to trigger the configuration, the System Administrator will execute the command:

```
ansible-playbook playbook.yaml
```

6 Intel® SST-BF Use Case Automation: OVS-DPDK Core Separation

In a cloud environment where OVS-DPDK is part of the components that support the networking layout, the Sys-Admin can choose to set up the OVS-DPDK PMD on either high or normal tier after setting up the core configuration as shown in Figure 7.

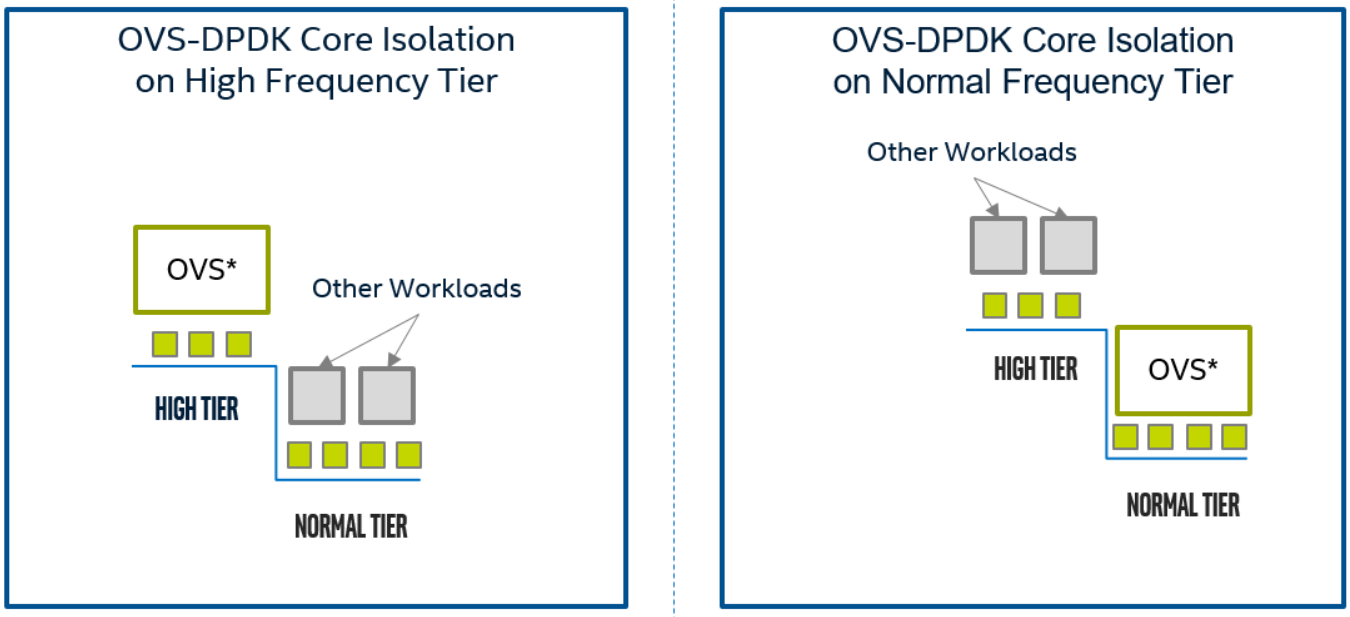

Figure 7. OVS* Core Pinning Model on top of an Intel® SST-BF enabled CPU

Figure 8 shows how this flow is inserted after the "Set up Intel® SST-BF Core Frequencies" task.

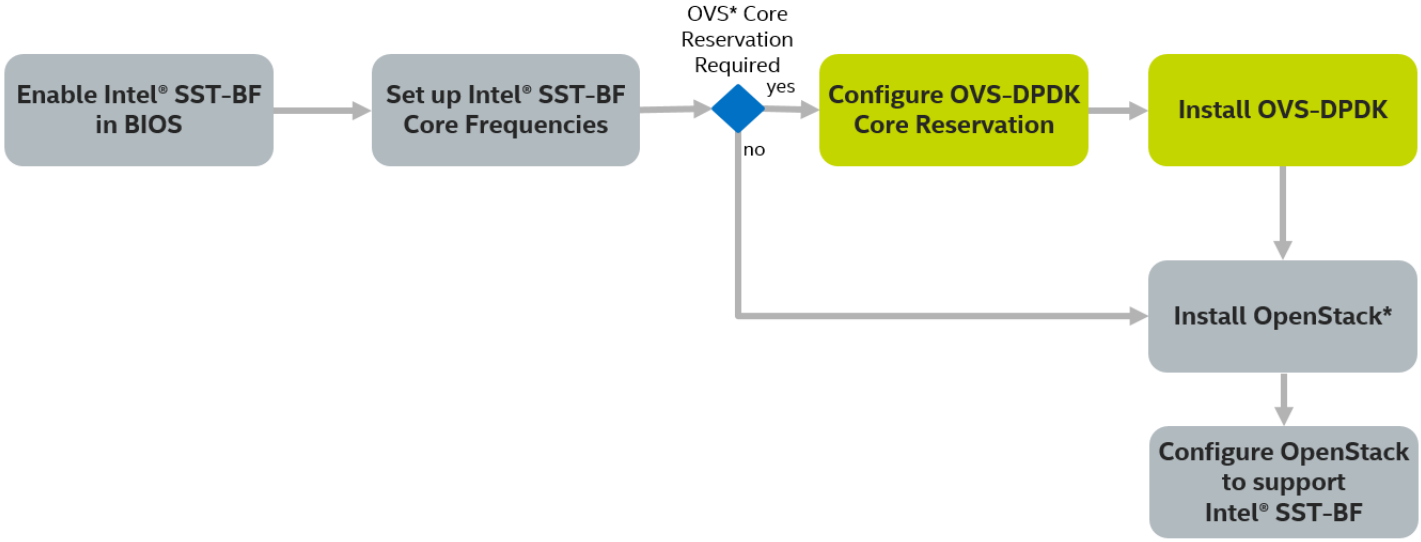

Figure 8. Configuration Flow Tasks for OVS*-DPDK Provisioning

To make this configuration possible, the automation process uses the "Configure OVS-DPDK Core Reservation" (for details, see [3] in Table 2) and through the automation artifacts we get a simplified process to produce the right set up.

The initial steps are the same as specified in Section 5.

Using the Galaxy application in a Linux environment, the Roles can be pulled directly issuing the command:

```
ansible-galaxy install intel.sst_bf_openstack_setup_automation
```

The Ansible Role utilizes a Python script to configure Intel® SST-BF on the host (see the sst_bf.py documention).

Note: For air-gapped deployments, this repository must be present in /tmp on the target host under the folder name CommsPowerManagement.

A Playbook example to trigger the configuration of Intel® SST-BF and to include the tasks to configure an existing OVS-DPDK installation would look like this:

```
- name: Set up SST-BF Core Frequencies
  hosts: nova_compute
```

```
  user: root
  gather_facts: yes
  tasks:
      - name: Set and get SST-BF
        vars:
          skip_ovs_dpdk_config: false
        include_role:
          name: "intel.sst_bf_openstack_setup_automation"

# [ Insert Role to install OpenStack Nova Compute here ]
# Openstack Nova compute must be installed before running the next task

      - name: Configure OpenStack to Support Intel SST-BF
        vars:
          configure_os_only: true
        include_role:
          name: "intel.sst_bf_openstack_setup_automation"
```

Note: The parameter highlighted in yellow will trigger the tasks related to OVS-DPDK configuration.

A full installation of OVS-DPDK can be triggered with the provided automation tasks. Using this set of parameters in the Playbook will set up the OpenStack Compute host with Intel® SST-BF, install OVS-DPDK from the distribution's supported repository, pin physical cores to DPDK's PMD, and write the remaining cores to Nova. You must define "host_description" dictionary to suit your target node.

```
- name: Configure SST-BF for OpenStack
  hosts: nova_compute
  user: root
  gather_facts: yes
  tasks:
      - name: Set and get SST-BF
        vars:
          skip_ovs_dpdk_config: false
          ovs_dpdk_installed: false
        include_role:
          name: "intel.sst_bf_openstack_setup_automation"

# [ Insert Role to install OpenStack Nova Compute here ]
# Openstack Nova compute must be installed before running the next task

      - name: Configure OpenStack to Support Intel SST-BF
        vars:
          configure_os_only: true
        include_role:
          name: "intel.sst_bf_openstack_setup_automation"
```

For either option, an extra set of parameters (see Table 7) are available to be overridden at the Playbook level:

Table 7. Playbook Extra Configuration Parameters for OVS*

PARAMETER	DEFAULT	DESCRIPTION
skip_ovs_dpdk_config	true	Skip Open vSwitch*-DPDK
ovs_dpdk_installed	true	If an existing installation of Open vSwitch*-DPDK exists or not
ovs_core_high_priority	true	If true then pin high priority cores to PMD otherwise choose normal priority cores
ovs_dpdk_nr_1g_pages	16	Number of 1 GB hugepages to reserve for DPDK
ovs_dpdk_nr_2m_pages	2048	Number of 2 MB hugepages to reserve for DPDK
ovs_dpdk_driver	vfio-pci	Driver to bind to NIC
ovs_service_name	openvswitch-switch	Systemctl service name for Open vSwitch*
dpdk_service_name	dpdk	Systemctl service name for DPDK
ovs_datapath	natdev	Userspace datapath type for Open vSwitch* bridge creation
ovs_dpdk_interface_type	dpdk	Interface type for DPDK

When configuring OVS-DPDK, a description of the host will be a required variable to be defined, because there are host specific parameters. A default template file is located at "defaults/main/main.yml" and the information will be added in the section "host_description".

```
host_description:
  numa_nodes:
    0:
      interfaces:
        eno1:
          pci_address: "0000:3d:00.0"
        eno2:
          pci_address: "0000:3d:00.1"
      dpdk_socket_mem: 1024
      no_physical_cores_pinned: 4
    1:
      dpdk_socket_mem: 0
      no_physical_cores_pinned: 2
  bridge_mappings:
    ovs-brnew: 'eno1'
```

The configuration of these entries is described in Table 8.

Table 8. OVS* Configuration Parameters

ACTIVITY	REQUIRED	DESCRIPTION
host_description	yes	Description of target node's assets needed to configure Open vSwitch*-DPDK.
numa_nodes	yes	Description of target NUMA nodes describing interfaces, socket memory and number of cores to pin to PMD. numa_nodes dictionary must contain one or more NUMA nodes.
interfaces	no	One or more interfaces need to be defined if interfaces is defined. This information will be leveraged to build bridges. This dict will contain key value pairs. The key is the interface name.
pci_address	no	A PCI address for a given interface needs to be defined if the interface is attached to a bridge in bridge_mappings.
dpdk_socket_mem	yes	DPDK allocated socket memory.
no_physical_cores_pinned	yes	Number of physical cores to pin to associated NUMA node.
Bridge_mappings	yes	Bridge for DPDK including one or more bridge - interface definitions. An interface defined here must have an associated definition in numa_nodes.

Once the OVS-DPDK configuration and installation flow are finished, the cores assigned to the OVS-DPDK PMD process will be excluded from the core sets (cpu_shared_set and cpu_dedicated_set) in OpenStack. This mechanism will avoid any possibility of the OVS-DPDK processes clashing with either the OpenStack processes or with any visualized workload managed by OpenStack. The physical cores exclusively pinned to the OVS-DPDK PMD are also isolated from kernel processes, which requires a restart of the Nova Compute host.

7 Deploying a Workload using OpenStack* APIs on an Intel® SST-BF Ready Node

After the Ansible automated configuration flow is done (with or without OVS-DPDK) and the Nova Compute agent is running, the placement database is ready, and the Cloud-Admin has been given access to spin up new workloads, there are two required steps: flavor definition and request the compute API for the creation of a VM.

The flavor definition considering the core split is done using the following parameters:

For PCPU:

```
$ openstack flavor create --vcpus <number> <flavor_name_pcpu>
$ openstack flavor set --property hw:cpu_policy=dedicated --property
trait:CUSTOM_CPU_X86_INTEL_SST_BF=required --property trait:
CUSTOM_CPU_FREQUENCY_FIXED_HIGH_DEDICATED=required <flavor_name_pcpu>
```

For VCPU:

```
$ openstack flavor create --vcpus <number> <flavor_name_vcpu>
$ openstack flavor set --property hw:cpu_policy=shared --property
trait:CUSTOM_CPU_X86_INTEL_SST_BF=required --property trait:
CUSTOM_CPU_FREQUENCY_FIXED_HIGH_DEDICATED=required <flavor_name_vcpu>
```

Note: At the time this document is written, it is not possible to create a single flavor with both VCPU and PCPU.

After the flavor is defined, the Cloud Admin can create a new server by issuing a server create command:

```
$ openstack server create --flavor=<flavor_name_*> <server_name>
```

A subset of <number> CPU Cores will be assigned to the virtual server. If using a VCPU flavor, the cores to fulfill the request will be mapped to the pool of cores listed on the cpu_shared_set described in the Nova configuration file. Otherwise when using a PCPU flavor, the cores will come from the ones listed in the cpu_dedicated_set entry.

8 Summary

The automation process described in this document opens the door for scale adoption of Intel® CPU features by enabling the configuration of Intel® SST-BF on select SKUs of 2nd generation Intel® Xeon® Scalable processor (5218N, 6230N, and 6252N). This document also explains how to use Intel® SST-BF in concert with new features in OpenStack for tracking CPU resources.

Our main goal is to help Sys-Admins and Cloud-Admins perform complex configuration tasks that can be error-prone and time-consuming by creating a set of artifacts and Automation recipes. Simplifying these tasks at the Admin level enables the final user of an OpenStack cloud environment (that is, the Tenant) to benefit from running workloads on top of Intel® Architecture without having to break the abstraction layers provided by the Cloud.

The Ansible Roles, Playbook examples, and Scripts referenced in this document are available at the links presented in Table 2. We invite you to experiment with them to achieve a fully functional configuration of Intel® SST-BF and OpenStack. In addition, you can include OVS-DPDK as an option to optimize networking traffic and data flow processing.

1019/DN/PTI/PDF

341449-001US