IMPLEMENTATION GUIDE

Data Center
Rack Scale Design

Provisioning Intel® Rack Scale Design Bare Metal Resources in the OpenStack Environment

NOTE: If you are familiar with Intel® Rack Scale Design and OpenStack* and just want to know how to install components to enable OpenStack composition and bare metal provisioning, please skip to "Installing the Intel RSD OpenStack Components" below.

Introduction

Intel Rack Scale Design (Intel® RSD) is an open source Data Center architecture that enables dynamic composition of logical systems, or "nodes" from disaggregated pools of compute, storage, and (in an upcoming release) accelerator resources. It is based on open, scalable and secure infrastructure management APIs that support a software defined infrastructure with interoperability across hardware and software vendors. Intel RSD makes the Data Center more economical, flexible, simpler to manage, and easier to scale out on demand.

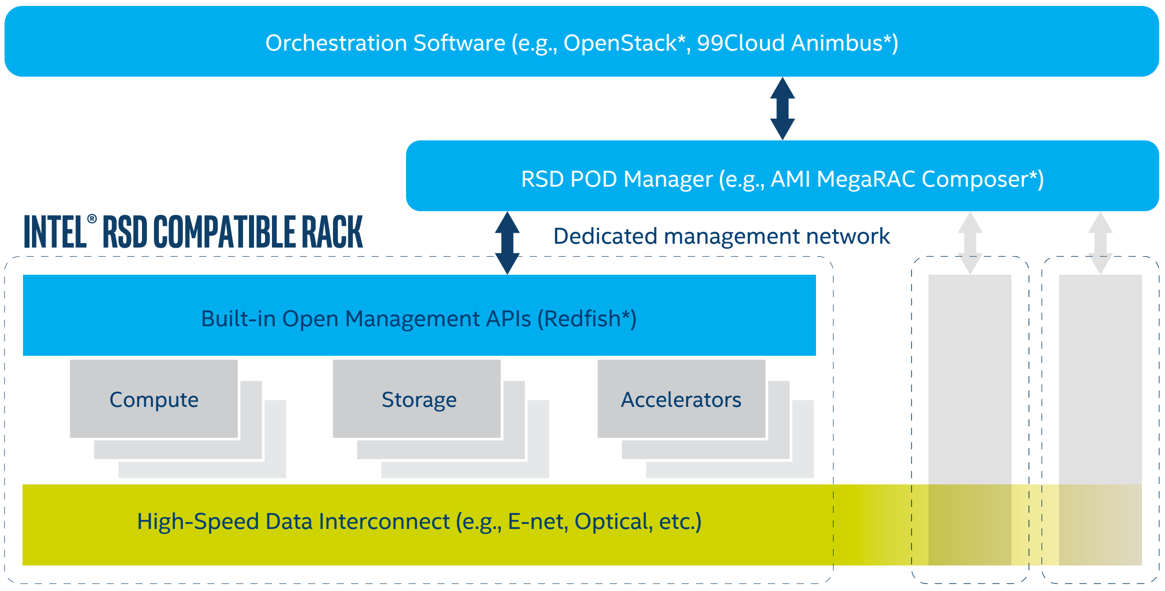
Orchestration Software (e.g., OpenStack*, 99Cloud Animbus*)
RSD POD Manager (e.g., AMI MegaRAC Composer*)
INTEL® RSD COMPATIBLE RACK
Dedicated management network
Built-in Open Management APIs (Redfish*)
Compute
Storage
Accelerators
High-Speed Data Interconnect (e.g., E-net, Optical, etc.)

Figure 1. Intel Rack Scale Design is an industry-aligned specification for composable disaggregated infrastructure.

Today, Data Center managers using cloud-oriented virtualized environments like OpenStack* are asking how they can employ Intel RSD in their existing public, private and hybrid clouds. In this paper we describe how to deploy Intel® RSD-based equipment in an existing OpenStack environment, using OpenStack's Ironic* service for logical node composition and bare metal provisioning. Readers should keep in mind that as the Intel RSD specification continues to evolve it will address more usage models, technologies and products. In addition, there are many integration efforts currently under way to support Intel RSD systems as native platforms in a wide range of virtual and cloud environments.

Table of Contents

Traditional vs. Intel RSD Bare Metal Provisioning

Even in cloud environments like OpenStack, operators still need to provision bare metal servers for specific purposes:

- High-performance computing clusters
- Applications that require access to hardware devices that can't be virtualized
- Database hosting (some databases run poorly on a hypervisor)
- Single tenant, dedicated hardware to meet performance, security, dependability or regulatory requirements
- Or simply to rapidly deploy an OpenStack cloud infrastructure

The OpenStack environment provides a component called Ironic that supports operators with a set of services to provision servers, either for incorporation into the virtual cloud environment, or to run dedicated applications directly on bare metal. To provision a traditional bare metal server, operators physically install and make physical power and network connections to the target servers. Then they "enroll" these servers in the Ironic database, i.e., they provide a description of the assets, including CPU type, number of sockets and cores, size of DRAM and attached SSDs or hard disk storage, accelerator resources, port address, and so on, either manually or using an in-band script. When a new server is needed to support a workload, the Nova service in OpenStack will search the Ironic database for a server that matches the workload requirements, provision it, and deploy the workload.

To understand how Ironic works with Intel RSD based equipment, we need to keep in mind that Intel RSD provides a degree of flexibility that is not available in traditional fixed-configuration servers. Intel RSD is a disaggregated architecture, so rather than fixed configuration servers, equipment is visualized as pools of resources by type, e.g., compute, storage, and (in the future) accelerator resources. When fully deployed Intel RSD will allow operators to define or "compose" logical nodes in real time with any mix of resources available in the resource pools. Rather than choosing a "best fit" existing server, Intel RSD allows operators to create logical nodes with the exact resource requirements "on-the-fly" and then enroll the node in the Ironic database for future consumption. The current version of Intel RSD (2.2) enables storage pooling and composition over PCIe interconnects. Upcoming versions will support NVMe over Fabric*, FPGA pooling, network function virtualization (NFV) management and additional features.

Composing a logical node from pooled compute and storage resources and making it available to Nova* requires one extra step compared to provisioning traditional bare metal servers. This is the composition step, which is accomplished by a simple operation using a modified OpenStack Client (OSC) command line interface (CLI) to define the new node with the desired resources (compute, DRAM, storage, etc.). Once this is accomplished, the operator enrolls the new node as usual, and then provisions it with a call to Nova, which invokes Ironic "under the covers."

Intel RSD and Redfish

At this point we should clarify the relationship between Intel RSD and Redfish. Redfish is an open standard for data center management created by the Distributed Management Task Force (DMTF)*. Essentially, Redfish is the successor to IPMI* (Intelligent Platform Management Interface), which is about 20 years old. Redfish is intended for hyperscale environments and is more scalable and secure than IPMI due to its web-friendly, RESTful APIs.

Intel is a longstanding member of DMTF, and a major contributor to Redfish. New functionality and APIs needed to implement the Intel RSD vision, including composition of logical servers from disaggregated resources, is defined, prototyped and tested by Intel and its partners, and then submitted to the Redfish committee for inclusion in the standard. This extends Redfish and encourages the convergence of Intel RSD and Redfish over time. As with any standards effort, Redfish submissions require review, discussion and approval by the governing committees, so there is always a time lag between the latest Intel RSD APIs and reference code, and the approved version of Redfish. Consequently, at any point in time there may be some differences in the management interface (e.g., the URIs).

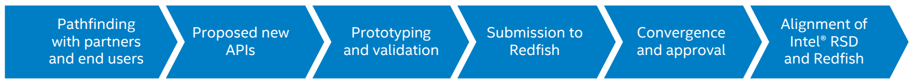
Pathfinding with partners and end users
Proposed new APIs
Prototyping and validation
Submission to Redfish
Convergence and approval
Alignment of Intel® RSD and Redfish

Figure 2. Intel RSD is a collaborative initiative by which pathfinding activities with Intel's ecosystem partners and definitional customers result in proposed functions and APIs submitted to the DMTF Redfish working group.

Intel RSD also differs from Redfish by identifying APIs as required, optional or recommended, and by providing a conformance test suite. This eliminates ambiguity, and helps ensure that the required capabilities have been properly implemented and that components of the Intel RSD software stack from suppliers will work properly together.

(The following discussion assumes that the target equipment implements Intel RSD version 2.2 APIs.)

Discovery, Composition, Provisioning and Deployment of Workloads on Intel RSD Resources with OpenStack

To enable Intel RSD in the OpenStack environment, Intel created a python library called "rsd-lib" and an OpenStack Client (OSC) plugin called "python-rsdclient" and upstreamed them to the OpenStack project. The rsd-lib library extends the standard Ironic Redfish Sushy driver, enabling it to make composition calls to the POD Manager (PODM) software component of Intel® RSD. PODM then carries out node composition requests by communicating with hardware resources through Intel RSD APIs exposed by PSME software components running on hardware modules within the rack. The python-rsdclient plugin provides the user commands for node composition and Ironic enrollment. The diagram in Figure 3 below shows the Intel RSD bare metal provisioning process:

1. User requests a new composed node with specific parameters (CPU, DRAM, storage, etc.) using the OSC RSD CLI, which sends requests to PODM.
2. PODM composes the node and returns node status to the user.
3. The user enrolls the new node into the Ironic database using the OSC Ironic CLI.
4. The user commissions a workload on the new node using Nova, like any other server.
5. Nova invokes Ironic to provision the node and then deploys the workload.

Installing the Intel RSD OpenStack Components

Note: The changes to enable Intel RSD bare metal provisioning have been upstreamed to the OpenStack community. As of release of this application note, the Intel RSD modified OpenStack client and Ironic Sushy extensions are in the OpenStack review process. This situation may change as these submissions are promoted to official plug-in status. You may have to search Github* to find the current location of this package. We expect these changes to be incorporated into the “Queens” release of OpenStack.

The Python* library for Intel® Rack Scale Design, rsd-lib, extends the existing Redfish Sushy driver to include functionality for Intel RSD PODM APIs. Capabilities include logical node composition and decomposition, remote storage discovery and composition, and attaching and detaching NVMe drives over PCIe to logical nodes. The rsd-lib library is located at https://github.com/openstack/rsd-lib and is made available to OSC automatically through the RSD OSC plugin—no additional user actions are required.

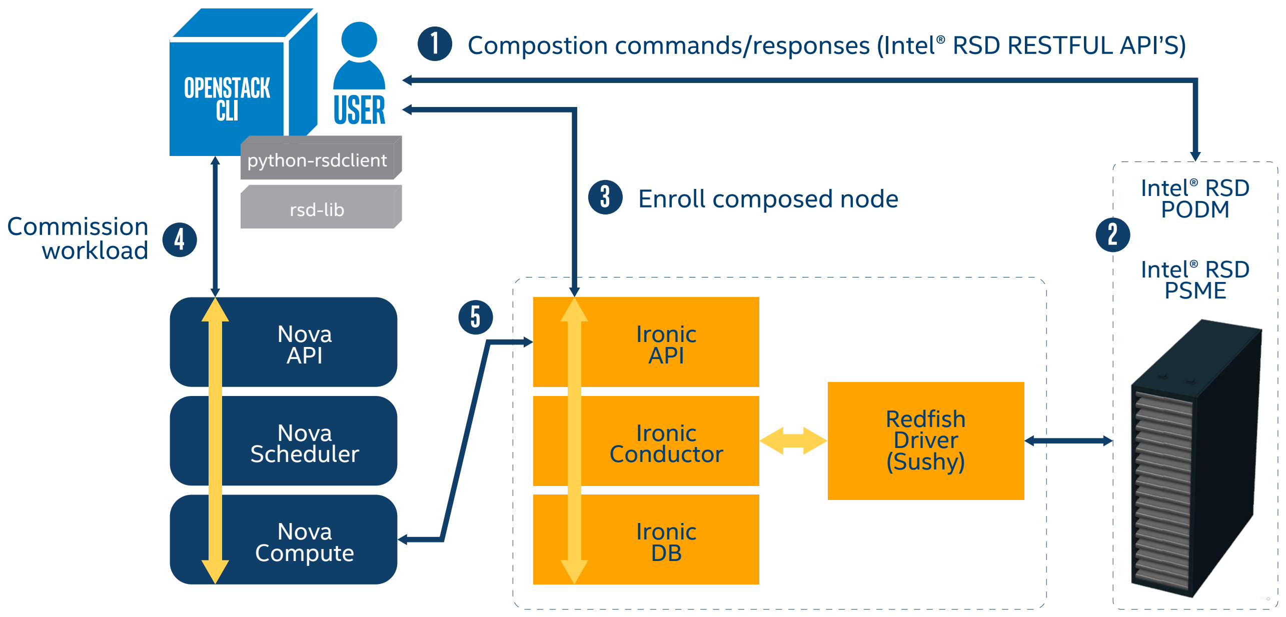

Figure 3. Python libraries upstreamed by Intel to the OpenStack project provide the ability to dynamically compose, enroll and provision logical nodes using OSC.

Users interact with Intel® Rack Scale Design compatible hardware through the RSD plugin for OSC, which provides the extended RSD command line interface (CLI). To use OpenStack RSD commands, install the normal python-openstackclient and the python-rsdclient RSD plugin on any node that has a network connection to OpenStack:

```
# pip install python-openstackclient
# pip install python-rsdclient
```

Next, initiate your user session by logging in to the RSD OpenStack client. First, provide your OpenStack username, password, project, and auth endpoint. You can use configuration options --os-username, --os-password, --os-project-id (or --os-project-name), and --os-auth-url, or set the corresponding environment variables:

```
$ export OS_USERNAME=user
$ export OS_PASSWORD=password
$ export OS_PROJECT_NAME=project              # or OS_PROJECT_ID
$ export OS_PROJECT_DOMAIN_ID=default
$ export OS_USER_DOMAIN_ID=default
$ export OS_IDENTITY_API_VERSION=3
$ export OS_AUTH_URL=http://auth.example.com:5000/identity
```

Then, you have to provide Intel PODM username, password, SSL certificate with admin privilege, and PODM URL. You can use configuration options --rsd-username, --rsd-password, --rsd-verify, and --rsd-url, or set the corresponding environment variables:

```
$ export RSD_USERNAME=admin
$ export RSD_PASSWORD=password
$ export RSD_VERIFY=False     # or RSD_VERIFY=<path to SSL certificate>
$ export RSD_URL=https://localhost:8443/
```

Once logged in to the CLI, users can issue commands to compose logical nodes and enroll them to the Ironic database (see examples below). The PODM access endpoint (URL) can be specified in a Linux environment variable, or it can be entered as a CLI parameter when composition is initiated in the OSC CLI.

Enrolling Composed Nodes

Once a new node is composed, it must be enrolled in the Ironic database prior to provisioning, just like any new server resource. However, there is one difference users must be aware of—the /redfish/v1/Nodes/<node ID> endpoint for the composed node returned by PODM cannot be used directly, since it is specific to Intel RSD, and the Ironic driver for Redfish uses standard Redfish only (see "Intel® and Redfish" section above).

However, users can determine the correct Redfish URIs by displaying the system UUIDs for composed nodes using the OSC RSD CLI. The URIs can then be used to retrieve the Redfish system IDs (e.g. /redfish/v1/Systems/<system ID>), which can be used to enroll the new nodes with Ironic. For example:

Get the list of nodes that have been composed:

```
$ openstack rsd node list
+----------+--------+--------------------------------------+-------------+
| Identity | Name   |                 UUID                 | Description |
+----------+--------+--------------------------------------+-------------+
|    2     |  Test  | fd011520-86a2-11e7-b4d4-5d323196a3e4 |    None     |
+----------+--------+--------------------------------------+-------------+
```

Use the UUIDs of the composed nodes you want to enroll to retrieve the correct Redfish system IDs

```
$ openstack node show fd011520-86a2-11e7-b4d4-5d323196a3e4 | grep –i "system"
```

Here we're using grep to isolate just the systems URI from the large amount of information potentially returned for each node, so the response should contain a string representing the systems URI. For example:

```
/redfish/v1/Systems/<system ID>
```

Then enroll the composed nodes using the Redfish systems URIs:

```
$ openstack baremetal node create --driver redfish --driver-info \
  redfish_address=https://example.com --driver-info \
  redfish_system_id=/redfish/v1/Systems/<system ID> --driver-info \
  redfish_username=admin --driver-info redfish_password=password
```

General enrollment documentation can be found at https://docs.openstack.org/project-install-guide/baremetal/draft/enrollment.html

OpenStack RSD CLI Examples

Compose node command allows the user to issue node composition command through OpenStackClient(OSC):

```
$ openstack rsd node compose [--name <name>]
                             [--description <description>]
                             [--processor <processor requirements>]
                             [--memory <memory requirements>]
                             [--remote-drives <remote drives requirements>]
                             [--local-drives <local drives requirements>]
                             [--ethernet <ethernet requirements>]
```

Attach specific resource to existing composed node:

```
$ openstack rsd node attach [--resource <resource uri>]
                            [--capacity <size>]
                            <node>
```

Detach specific resource from existing composed node:

```
$ openstack rsd node detach [--resource <resource uri>]
                            <node>
```

Delete composed node allows the user to delete composed node(s) by specifying <node_uuid>:

```
$ openstack rsd node delete <node> [<node> ...]
```

Show composed node detail command allows the user to get composed node details by specifying <node_uuid>:

```
$ openstack rsd node show <node>
```

List composed node command allows the user to list all composed node with brief info:

```
$ openstack rsd node list
+----------+------+--------------------------------------+-------------+
| Identity | Name |                 UUID                 | Description |
+----------+------+--------------------------------------+-------------+
|    2     | Test | fd011520-86a2-11e7-b4d4-5d323196a3e4 |    None     |
+----------+------+--------------------------------------+-------------+
```

List storage services command allows the user to list all storage services brief info like below shows:

```
$ openstack rsd storage list
+----------+-----------------+-----------------------------+
| Identity |      Name       |         Description         |
+----------+-----------------+-----------------------------+
|    2     | Storage Service | Storage Service for Testing |
+----------+-----------------+-----------------------------+
```

Show storage detail command allows the user to display the storage service details:

```
$ openstack rsd storage show <storage service>
```

List fabric command allows the user to list all fabrics with brief info:

```
$ openstack rsd fabric list
```

Show fabric detail command allows the user to display the fabric details:

```
$ openstack rsd fabric show <fabric>
```

To get a list of available sub-commands and options, run:

```
$ openstack help rsd
```

To get usage and options of a command, run:

```
$ openstack help rsd <sub-command>
```

Product Availability

Intel RSD compatible products are available now from major vendors including Dell EMC*, Ericsson*, HPE*, Huawei*, Inspur*, Quanta*, Radisys*, Supermicro*, Wiwynn* and others. Learn more about how the Intel RSD architecture can accelerate your transition to an open, modular, software-defined Data Center at: https://www.intel.com/intelrsd. Or contact your local Intel representative to discuss how Intel® can help you to meet the demands of the Digital Transformation.