GRAPHCORE

Graphcore OpenStack Reference Design for IPU-POD Systems

Version latest

Graphcore Ltd

Aug 25, 2023

CONTENTS

This document illustrates a reference configuration of a Graphcore POD_{64} rack deployed with OpenStack, the open-source cloud computing infrastructure management software. The POD_{64} rack contains 64 IPUs (Intelligence Processing Units) and OpenStack is used to manage these IPU resources via its API and UI.

CHAPTER

ONE

CONTENTS

1.1 Overview

OpenStack can be deployed in a variety of different configurations, and Graphcore does not endorse or support any particular configuration. OpenStack is also not a pre-requisite for using Graphcore technology, however, OpenStack is often used as an underlying infrastructure in datacentres. The reference design described in this document can be used as a guideline to derive your own configuration of a Graphcore IPU solution on your existing implementation of OpenStack (or alternative cloud platform).

1.1.1 Goal

The aim of this document is to show a high-level example of configuring an IPU Pod (IPU-POD$_{64}$) with OpenStack. This includes capturing all the details of the design and implementation of the installation: networking, deployment, administration and monitoring features.

The broader, more general details of the OpenStack deployment are not described; however, the fundamentals which it implements are covered such that it could be replicated either manually or with a different automation scheme.

1.1.2 Use cases

IPU Pods support multiple tenants with multiple users.

Use case 1:

Provision of multiple user tenancies which each provide CPU and IPU compute in a discrete, secure and flexible virtual IPU Pod (vPOD). Each vPOD requires:

- One or more Virtual Machines (Poplar hosts) to execute Poplar workloads
- One or more IPU-Machines (IPU-M2000 or Bow-2000)
- Access to IPU-Machines from Poplar hosts to execute Poplar workloads on IPUs
- A secure network providing ssh access to Poplar hosts
- Access to VIPU and IPU-M management services for administrators
- Access to VIPU user services for Poplar hosts
- Secure network storage with secure network connections (optional)

1.1.3 Generic cloud overview

The overall architecture of components is shown below. Virtual IPU Pods are constructed with resources from several physical hardware units:

- CPU from a cluster of COTS (commercial-off-the-shelf) compute servers
- CPU from high-specification Poplar servers installed in the IPU Pod (IPU-POD$_{64}$) clusters
- IPUs from IPU-Machines in the IPU Pod (IPU-POD$_{64}$) clusters
- Network storage volumes from dedicated high-performance NFS appliances
- Virtual disk volumes mounted from Ceph storage cluster
- Control and data planes overlaid on dedicated 1 Gb (or 10 Gb) and 100 Gb network infrastructures, respectively.

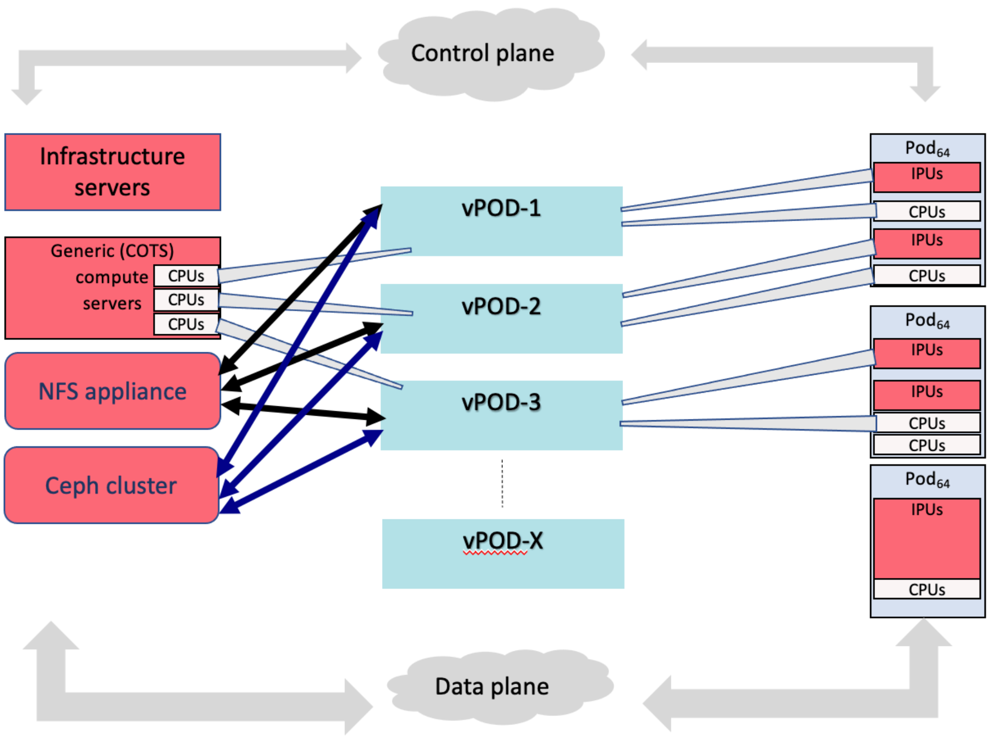

Fig. 1.1: Generic cloud overview - control and data planes

1.1.4 Cloud physical overview

The reference IPU cloud consists of a number of high-power racks containing IPU Pods, switches, storage and other servers. Additional rack(s) contain the OpenStack infrastructure hosts and storage hardware.

A typical layout is shown in Fig. 1.2. Note that each IPU-POD$_{64}$ is referred to as a logical rack in other sections of this document.

All 1G switches are connected to a dedicated data centre network (not shown) for switch management.

The data centre management network also provides internet connectivity.

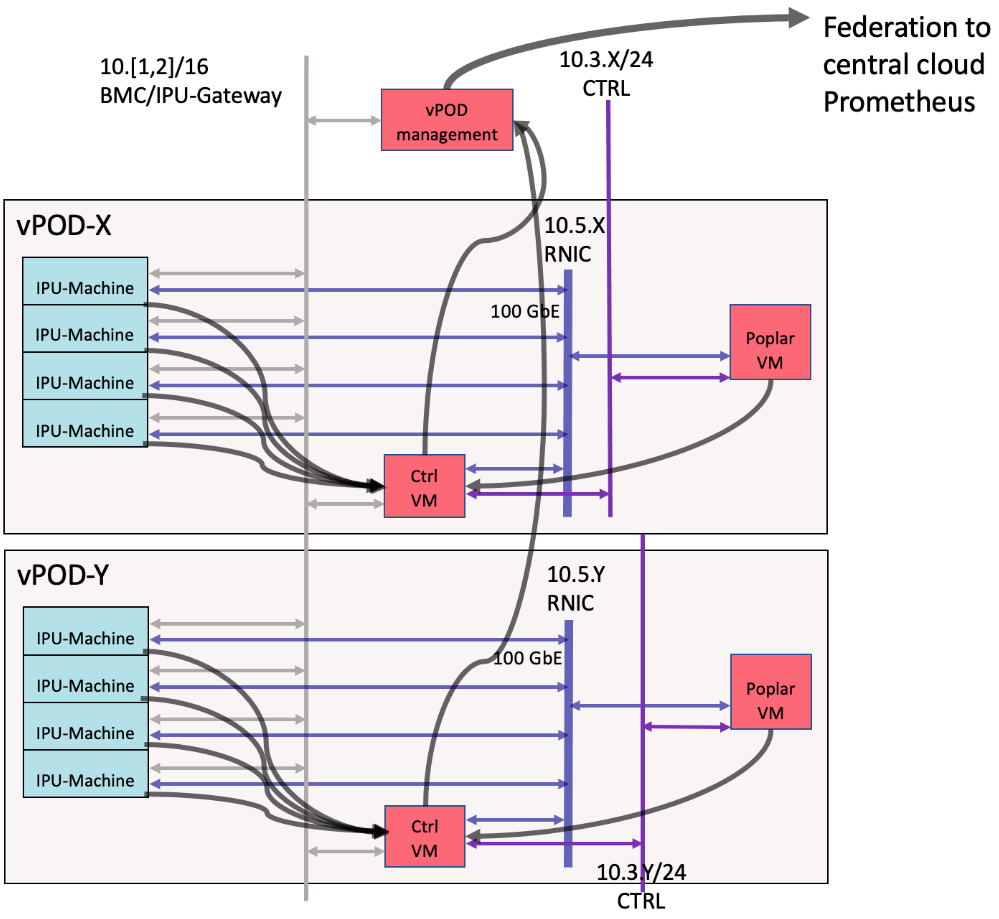

Fig. 1.2: Cloud physical overview

1.1.5 Acronyms and abbreviations

Term	Description
IPU-Machine	An IPU-Machine is a blade with 4 IPUs such as the IPU-M2000 or Bow-2000
IPU Pod	A rack solution containing IPU-Machines, one or more host servers (also called Poplar servers), network switches and IPU Pod software
Poplar server	Host server that is used by end-users to run machine learning workloads on IPU-Machines
RDMA	Remote DMA
RNIC	RDMA Network Interface Controller: 100GbE interface used to communicate between Poplar servers and IPU-Machines for fast data transfers
RoCE	RDMA over Converged Ethernet: protocol used to transfer data between Poplar servers and IPU-Machines
ToR	Top of Rack: term used in combination with the ToR RDMA switch that is placed on top of the IPU-Machines
VIRM	V-IPU Resource Manager: V-IPU agent running on an IPU-Machine
V-IPU Controller	A service that is managing IPU-Machines using VIRMs
VLAN	Virtual LAN - subnetwork grouping devices on separate physical local area networks
VxLAN	Virtual Extensible LAN - network virtualization that improves scalability associated with large cloud computing deployments
vPOD	Virtual POD. A group of IPU-Machines with a dedicated V-IPU Controller and one or more Poplar servers. One vPOD is used by one tenant. A vPOD can have 1, 2, 4, 8, 16 or more IPU-Machines.

1.2 Graphcore components and dependencies

This section describes the components (hardware and software provided by Graphcore) and dependencies (provided by a third party) required for the IPU Pod/OpenStack reference design.

1.2.1 Graphcore components

Hardware provided by Graphcore

IPU-Machines installed in physical IPU Pods with up to 16 IPU-Machines per IPU-Pod.

Software provided by Graphcore

Software	Version	Notes
Poplar SDK	3.1 or later	Poplar development environment and tools. 3.1 provides support for Ubuntu 20.04
V-IPU	1.18.1	VIRM client, agents and management utilities
IPU-M software	2.6.0	Firmware and software for the IPU-Machines

1.2.2 Dependencies

Third party hardware

Hardware	Notes
Dell R6525 server with dual-socket AMD Epyc2 CPUs	Host (Poplar) server(s). There can be up to 4 host servers. Used to host user VMs
Dell R640 server with single socket Intel Xeon Gold 5218R	OpenStack infrastructure servers
Arista 7060X ToR switch (32x100G + 2 10G)	Used to connect host server(s) and IPU-Machines for data traffic
Arista 7010T management switch (48p 1G+ 4x1/10G)	Used to connect host server(s) and IPU-Machines for management traffic and for access to external networks
Any generic 10G management switch	Used in OpenStack infrastructure rack to support servers with 10G, not 1G, interfaces

Third party software

Software	Compatible Versions	Notes
OpenStack	Wallaby	Wallaby supports Debian 11, RHEL 8.2, Ubuntu 20.04, CentOS Stream 8, Rocky Linux 8.6 (these are supported by IPU software)
Ubuntu	20.04	Used as a host system for VMs on Poplar server(s) and as a system inside VMs
Prometheus	Kolla-ansible/ wallaby-stable	Provided as part of Kolla Ansible playbooks for OpenStack
Grafana	Kolla-ansible/ wallaby-stable	Provided as part of Kolla Ansible playbooks for OpenStack

1.3 Physical installation

1.3.1 Graphcore IPU Pod installation

All the IPU Pods in this reference design are physically built as standard IPU-Machine (IPU-M2000 or Bow-2000) IPU Pod$_{64}$ configurations as described in the IPU-POD64 Reference Design: Build and Test Guide and Bow Pod64 Reference Design: Build and Test Guide, respectively.

Note that the following may be subject to change in subsequent revisions of the build and test guides:

- 4x R6525 AMD servers are used with 512 GB RAM each. The RAM is installed with 8x DIMM modules to provide 8 symmetrical NUMA nodes.
- Both 100 Gb RNIC ports on each Poplar server are connected to the ToR switch.
- Both management network ports on each IPU-Machine are connected to the management switch although only one management network port is currently used. This port is shared by the BMC and IPU-Gateway virtual interfaces via a micro-switch built into the IPU-Machine. The IPU-Machine firmware includes an option to split this to use the separate interfaces, but this has not been tested for this design.
- Each R6525 server is configured to PXE boot from a 1 Gb (or 10 Gb) interface. The physical connection is described in the relevant build and test guide. There are several 1 Gb interfaces on these hosts and the network cards vary so the logical interface can appear differently to the OS.
- Dell iDRAC settings:
  - Firmware upgrades:
    - BIOS: 3.9.3
    - iDRAC: 6.10.00.00
    - Mellanox ConnectX-5: 16.26.1040
  - Hyperthreading is enabled
  - NUMA per socket (NPS) = 4
  - Optimised PCIe settings:
    - PCIe preferred I/O bus: enabled
    - PCIe preferred I/O bus value: 161 (PCIe slot that contains the RNIC card)
  - RAID for boot and OS drives
  - IDRAC service account for Kayobe connection, and IPMI enabled:
    - Role: adminstrator
    - IPMI LAN privilege: adminstrator
  - Tuned for maximum performance:
    - CPU power management: maximum performance
    - Turbo boost: enabled
    - C states: disabled

The MAC addresses of all the IPU-Machine interfaces need to be gathered as part of the installation to allow for DHCP to be configured. The 1 GbE interface MACs are listed on labels but the RNIC interface MACs can only be found after network access has been setup. This means that there are several options for the first install:

- Connect the IPU-Machines to the 100 GbE ToR switch and power them on. The MAC addresses can then be seen by the switch and gathered from the switch management interface.
- Install and configure as a standalone bare-metal IPU Pod with factory IP addressing (as per the build and test guides) then query the RNIC MAC addresses. You can then redeploy the IPU Pod under the Openstack Ironic automation. You can also perform hardware tests on the IPU-Machines at this stage.

- Deploy the IPU Pod with Ironic and Neutron OpenStack projects using the 1 GbE interface MACs gathered from labels, but without the RNIC information. After deployment gather the RNIC interfaces and update the DHCP configuration and re-deploy.

1.3.2 Infrastructure hosts

Ceph nodes (x5)

For the reference design these are Dell R640 Intel 96 core servers with 768 GB RAM, 2x 100 GbE, 7x 1 TB NVME, 2x 500 GB SSD (as hardware RAID1).

Each server should provide >= 7 TB of NVME storage using multiple devices since this is better for Ceph OSDs.

These nodes also operate as Hypervisors for general compute purposes.

Section 1.6.2, Ceph clusters provides more details.

OpenStack control nodes (x3)

For the reference design these are Dell R640 Intel 96 core servers with 768 GB RAM, 2x 100 GbE, 7x 1 TB NVME, 2x 500 GB SSD (as hardware RAID1). These may be over-specified in terms of memory and CPU for your system.

These nodes run the OpenStack Overcloud core control services (as bare-metal containers).

Three servers are needed to provide HA.

Seed and Ansible control nodes (x2)

You will require servers with a minimum of:

- 8 core CPU
- 12 GB memory
- 100 GB free disk space (partitioned if possible to separate the Docker volume storage from the host OS)
- 1x 1 Gb Ethernet

In the reference design the servers are Dell PowerEdge R440 with Xeon Silver 4210 2.2 GB CPU, 128 GB RAM, Dual 25 GbE Ethernet, 2x 480 GB SATA SSD.

Note: Only one server is required for a single cloud installation. In our example two were used to allow for an independent development setup.

This host runs a pre-built seed VM in which OpenStack docker containers are run to manage the OpenStack cloud. This is also referred to as the Undercloud. This host must have access to power management and provisioning/control networks for the remote servers to be provisioned into the OpenStack cloud.

1.3.3 Core networking

Multi-rack interconnect

The Pod$_{64}$ racks (described in Section 1.3.1, Graphcore IPU Pod installation) and the OpenStack infrastructure racks have 8x 100 GbE uplinks to a shared 400 GbE spine. Ideally this would be connected as 4x 100 GbE to each of a pair of MLAG 400 GbE switches to provide redundant connectivity with a minimum 400 GbE throughput. Fig. 1.3 shows a general network architecture for multi-rack connectivity. This includes options for storage, disaggregated compute and other services including the OpenStack infrastructure.

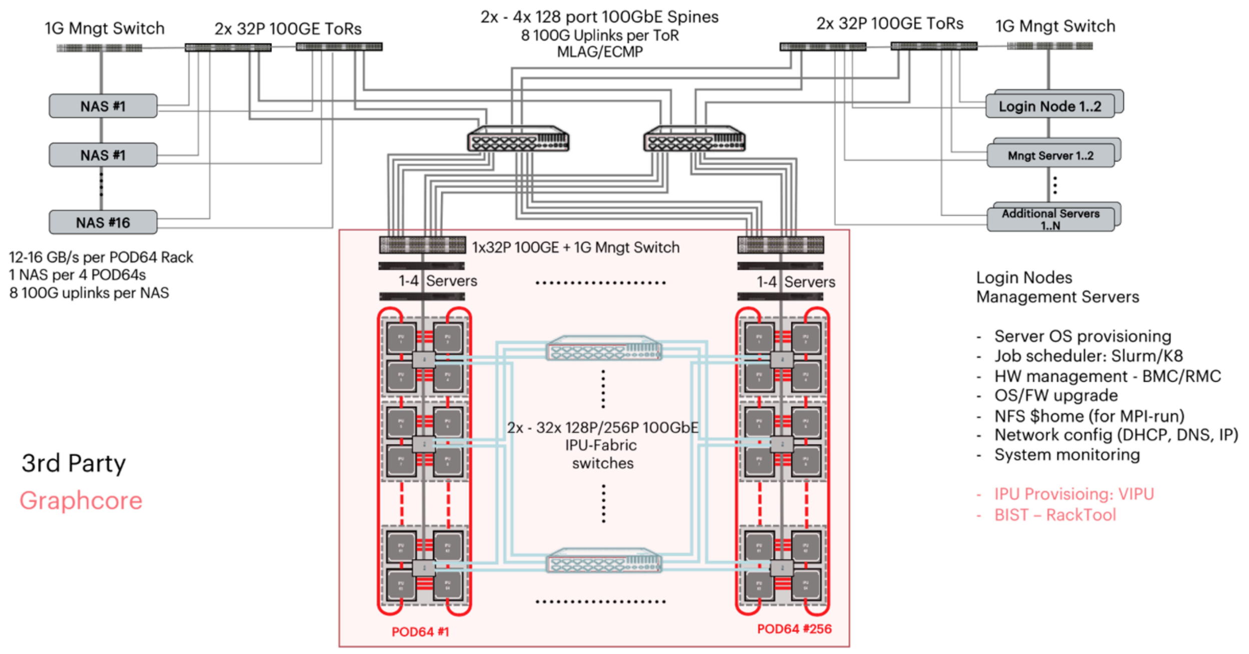

Fig. 1.3: Multi-rack connectivity

1.3.4 Power and cooling

Each Pod$_{64}$ rack requires power at:

- 1500 W per IPU-M2000 (1700 W for Bow-2000)
- 1400 W per Poplar host
- 333 W (max) for Arista 7060 switch
- 52 W (max) for Arista 7010T switch

1.3.5 Network components

This section describes networks, endpoints, physical devices and the relationships between them.

1 GbE management network

This network has the following devices and services connected:

- Poplar servers: 2x 1 GbE interfaces and IPMI interface
- Infrastructure servers: 2x 1 GbE interfaces and IPMI interface
- IPU-Machines: will have either
  - A single connection to the 1 GbE presenting two MAC addresses, one for IPU BMC and one for the IPU-Gateway; or
  - Two 1 GbE connections, one for BMC and one for the IPU-Gateway (untested in this reference architecture)
- PDUs in each rack: management interface
- 100 GbE switch: management interface

The purpose of this network is to provide connections for managing devices.

100 GbE data network

This network has the following devices and services connected:

- Poplar servers (Hypervisors): 2x RNIC 100 GbE interfaces (configured as a bonded MLAG pair)
- IPU-Machines: 100 GbE interfaces
- OpenStack control plane with virtual networks plus block, object and file storage
- High performance 3rd party storage (optional)

Networks and VLANs

The physical switches (1 GbE management switch and 100 GbE data switch) provide physical network access to the IPU-Machines, infrastructure hosts and Poplar servers. To separate data and management traffic, and to separate the various tenants, VLANs are added as provider networks within OpenStack Neutron. Neutron will then provide IPAM and DHCP for these provider networks, along with any required routing.

A typical VLAN set would be:

- Overcloud networks
  - Provisioning networks (for deploying and managing host OS)
  - Out-of-band management network (for IPMI power management)
  - 'Internal' network for OpenStack control plane communication
  - 'Tunnel' network for VXLAN tunnels between Open vSwitch instances
  - 'Storage' network for Ceph connectivity
  - Additional networks for other OpenStack services if enabled, for example:
    - Octavia (LBaaS)
    - Manila (NFSaaS)
- Public storage (if exposing Ceph to user workloads)
- External network (for exposing workloads)
- External high-performance storage
- VLAN range for VLAN based provider networks to be used in user tenancies

1.4 Virtualised Pods

Virtualised Pods (vPODs) are implemented as several separate instances, in an isolated virtual network. Section 1.8.9, vPOD logical networks contains diagrams and further details.

1.4.1 Poplar instances

Poplar instances are end-user instances where users can run Poplar jobs. Each vPOD has one or more Poplar instances. They can be either bare-metal or virtual instances.

Each Poplar instance is connected to the following networks (see Section 1.8.4, IP addressing and VLANs for more details):

- vPOD IPU data network: Uses 100 GbE VLAN and RoCE.
- vPOD storage network: Uses 100 GbE VLAN. The storage appliance has a dedicated VLAN for this network.
- vPOD control network: A virtual 1 GbE network which also provides a default route to the internet. External connections come in via this network.

In an Ironic bare-metal node, this is the only 1 GbE network connection as only one network connection is supported for each physical interface.

RNIC port configuration

To connect to the IPUs using the Graphcore IPU-over-Fabric (IPUoF) protocol, access needs to be via an RDMA enabled network. To achieve this from within the VM, SR-IOV will be used. The virtual function (VF) that is passed through to a VM will only have access to a single IPU VLAN. Moreover, because Mellanox Connect-X 5 is used, the VF can be connected via a bond (called VF LAG) to the RDMA enabled network. In this case, both members of the bond are connected to a single ToR switch in the IPU Pod rack.

Disaggregation

The Poplar instances are usually running on the physical servers which share the 100 GbE and 1 GbE local switches with the IPU-Machines in the vPOD to ensure minimal shared traffic with other vPODs. However, the network infrastructure is fully connected so the Poplar instance could be located on any IPU Pod or in the general compute clusters. This allows for a fully disaggregated setup (if required) but also allows for other hosts to be used in the case of a hardware failure within an individual IPU Pod.

1.4.2 Control instance

The control instance is for vPOD management and is not normally accessible to end users. It is always a virtual machine and requires only one vCPU. More vCPUs can be provided to speed-up activities such as IPU-Machine upgrades which run one IPU-Machine upgrade per available vCPU in parallel.

The control instance is connected to the following networks (see Section 1.8.4, IP addressing and VLANs for more details):

- Shared IPU-Machine BMC network.
- Shared IPU-Machine IPU-Gateway network.
- vPOD control network: A virtual network which also provides a default route to the internet. External connections come in via this network.
- vPOD storage network: Uses 100 GbE VLAN, simple TCP/IP only.
- vPOD IPU data network: Uses 100 GbE VLAN and RoCE.

- Standard OS Node exporter for monitoring and alerts.

The control instance runs:

- The vipu-server, accessed over the vPOD control network from the Poplar hosts.
- A Prometheus instance which collects data for the whole vPOD (see Section 1.7.1, Monitoring).
- When required, IPU-Machine maintenance and upgrades.

The control instance stores Graphcore rack_tool configurations for the IPU-Machines and is normally located on a general compute cluster as there are low performance requirements for its connectivity.

1.5 OpenStack deployment

The figure below, taken from the OpenStack software page, shows an overview of OpenStack services:

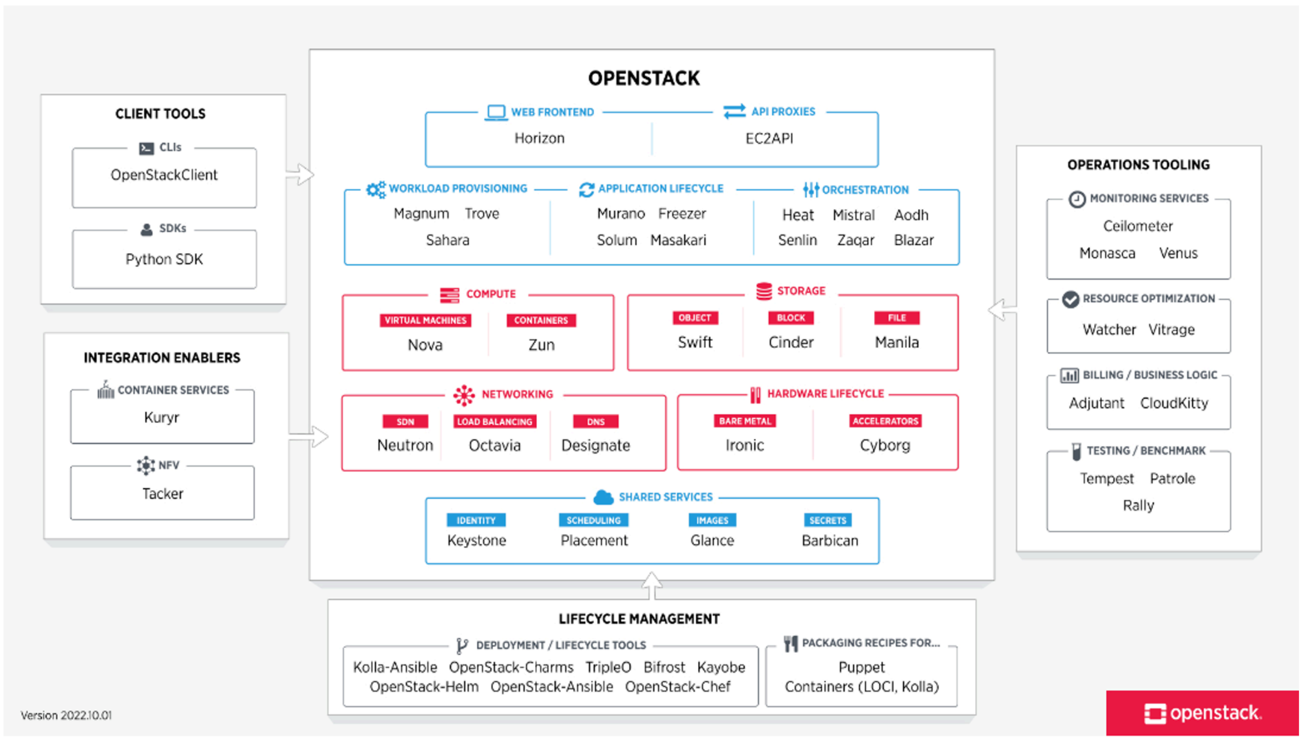

Fig. 1.4: OpenStack overview

To integrate Graphcore IPU-Machines with OpenStack, the following Openstack interfaces and services are used.

1.5.1 Ironic

A custom Openstack Ironic driver was written for IPU-Machines which allows for them to be managed as bare-metal devices. This currently implements:

- Simple DHCP control of the network interfaces
- Simple Redfish power control (not yet tested)

The Ironic driver can be found in Section 1.9.6, IPU-Machine Ironic driver.

1.5.2 Nova

The following Openstack Nova items and features were used to ensure optimal communication with the IPU-Machines:

- Pinned pages for user VMs. All free pages allocated to the VMs are pinned.
- HugePages (1 GB) - on a 512 GB system there should be 464x 1 GB HugePages provisioned on the hypervisor for the expected workloads.
- SRIOV pass-through of RNIC Virtual Functions (VF).
- Dedicated physical CPUs:
  - nova_cpu_dedicated_set in nova.conf, with an equal number of CPUs dedicated from each NUMA zone.
  - hw:cpu_policy: select dedicated in the flavor configuration (to enable dedicated resources).

Note that it is also possible (and tested) to use over-subscribed vCPUs. Pinned pages are a requirement to allow for direct NIC connections - see Section 1.8.6, 100 GbE networks with RDMA over Converged Ethernet (RoCE).

1.5.3 Neutron (Open vSwitch)

OpenStack SDN is conventionally implemented as a set of Neutron services. Our recommendation is to use the Open vSwitch (OVS) Neutron ML2 driver.

In a virtualised OVS-based OpenStack deployment, instance network configuration is terminated in the associated hypervisor using Open vSwitch. OpenStack networking for VM instances and other software-defined infrastructure is expressed as flow rules for the Open vSwitch bridges.

Our reference design makes use of VF-LAG with OVS hardware offload. This gives maximum performance and resilience by exposing the VFs via a bonded network connection, enabling Open vSwitch to offload intensive activities onto the dedicated silicon in the Mellanox ConnectX-5 NIC.

For the hardware offloads to be enabled, OVS has to be connected to the bond directly, without using Linux network bridges. This means that the Overcloud (hypervisors/control nodes) and workload cannot communicate using the bonded 100 GbE interface, and must instead use an additional network interface. For low bandwidth applications such as Octavia the existing 1 GbE network interfaces could be used for this traffic, but for high bandwidth applications (such as sharing the Overcloud Ceph cluster via CephFS or RADOS), additional high performance NICs will have to be provisioned.

1.6 Storage

1.6.1 High performance storage appliance

To provide shared storage with sufficient performance to allow for IPU Pod processing, a storage appliance with the following features is used in this reference design:

- NFS performance to serve a single client at up to 10 GB/s (using nconnect=16).
- Support 8-way active LAG connectivity (see Section 1.8.7, Link aggregation) to provide up to 800 Gb/s connection bandwidth.
- S3-compatible object storage performance to serve a single client (with multiple threads + connections) at up to 6 GB/s.
- Support for multiple VLANs, subnets and access interfaces which can be configured dynamically via an API. One of each per vPOD is required.
- Support for multiple NFS filesystems which can be configured dynamically via an API and which can have access restricted to specified subnets.

- Support for dynamic creation of object storage access accounts via an API.
- Automated snapshotting of NFS filesystems without performance impact.

1.6.2 Ceph clusters

A Ceph cluster built on a number of COTS servers with local NVME storage is provided. This provides all the block-storage services needed by OpenStack Cinder, and can provide specific virtual disk volumes to virtual machines as required.

Ceph is deployed using an open-source Ansible collection as containerised services. In this reference design v1.10.0 of the collection is used: https://github.com/stackhpc/ansible-collection-cephadm.

Access to the Ceph cluster is provided over the 100 GbE Ethernet physical network to ensure high performance access (up to 4 GB/s for a virtual volume mounted on a VM).

If required, this cluster can also be used to provide CephFS file storage which can be managed with OpenStack Manila.

Note: Since the cluster runs in the OpenStack infrastructure networks it cannot be directly exposed to end-user tenancies, so a router must be deployed to route client traffic to the Ceph subnet.

1.6.3 NVME drives

Each physical Poplar server has 7x NVME drives installed (minimum 1 TB each).

- Poplar Hypervisor: NVME devices are configured as RAID0 or RAID6 volumes and used to provide ephemeral disks to each Poplar VM (via OpenStack Nova).
- Ceph Hypervisor: most of the NVME capacity is passed to the Ceph cluster and the remaining portion to Nova.
- Bare-metal Poplar: NVME devices are provided to the operating system and configured as RAID0 (or RAID6) and mounted as /localdata (a Graphcore convention).

1.7 Monitoring and alerts

1.7.1 Monitoring

Prometheus is used for monitoring the vPOD infrastructure and is presented in a Grafana instance for each vPOD. Prometheus is then federated from multiple vPODs into a Prometheus collector and Grafana instance for whole-cloud monitoring.

Prometheus exporters are installed and federated as follows:

- Prometheus in the OpenStack infrastructure is enabled through Kayobe which automatically deploys a containerised HA instance of Prometheus/Grafana across the control plane hosts.
- Each hypervisor has a containerised node-exporter instance providing monitoring of each hypervisor. This is also deployed through Kayobe.
- Every VM instance which uses a provided OS image contains an OS-standard node-exporter.
- On each vPOD:
  - IPU metrics are served via the IPU-Gateway interface and gathered from http://<IPU-Machine IPU-Gateway IP>:2112/metrics.
  - The Poplar VMs use node-exporter.

  - Control VMs federate Poplar VM data with IPU-Machine data together on a single Prometheus instance.

- A single vpod-management VM is created which federates all vPOD Prometheus data together.
- Ceph clusters (where deployed) have a built-in Prometheus exporter.

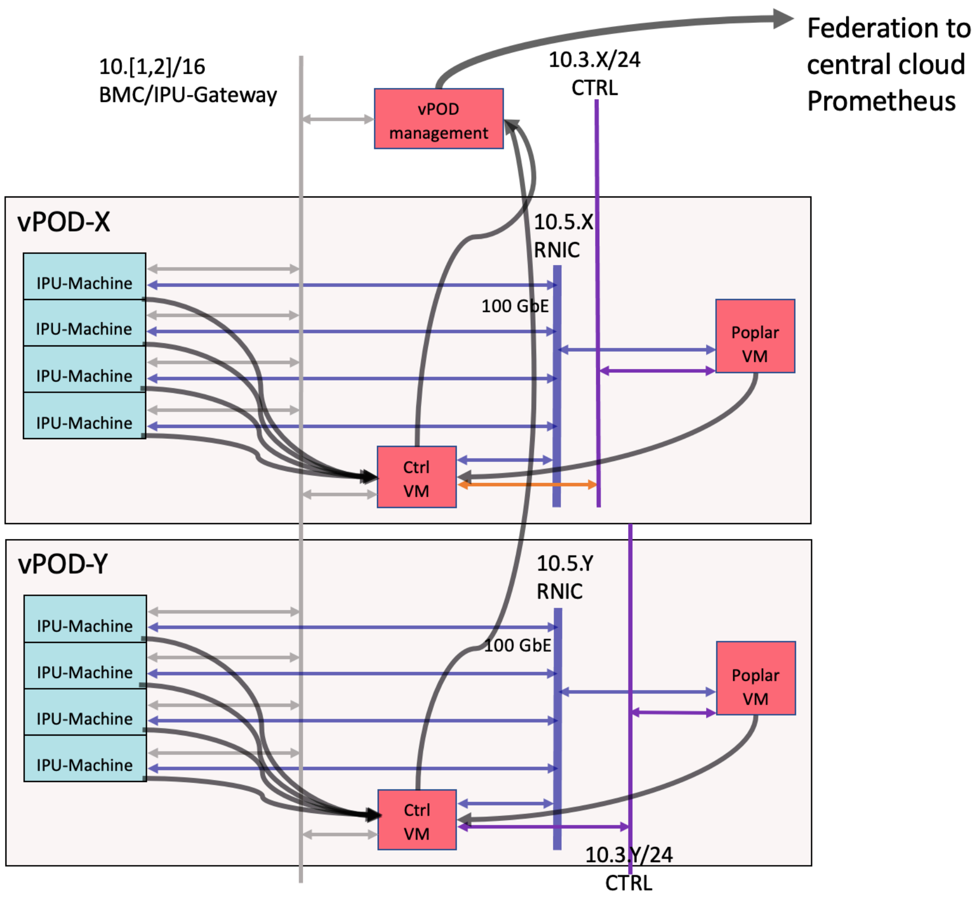

Fig. 1.5: Monitoring with Prometheus

1.7.2 Alerting

Alerts for critical and upcoming issues are configured in the federated Prometheus monitoring and can be sent to email, slack, and so on.

Log aggregation

Each control VM (within each vPOD) aggregates the syslog from each IPU-Machine in the vPOD.

- Each IPU-Machine BMC is configured to deliver syslog using rsyslogd configuration.
- The IPU-Gateway in each IPU-Machine is configured to deliver syslog using rsyslogd configuration.

The rack_tool application is used to set up both these configurations either during or after IPU-Machine software updates.

The syslogs from all hypervisors and VMs (including aggregated vPODs) are collected by a central log analysis service (for example Elastic Stack) to allow for deep-log checking, trend analysis and alerting.

1.8 Networking

As the IPU-Machines in each IPU Pod are network appliances, network architecture and implementation are crucial to the correct function and performance of AI applications.

1.8.1 Requirements for reference design

1. Each vPOD must be completely isolated from any other vPOD. No shared networks except for the internet access network. This restriction also applies to the high-performance storage appliance which must be accessed via a dedicated VLAN, subnet and access interface for each vPOD. See Section 1.6, Storage for more information.
2. Poplar hosts (virtual or bare-metal) must have access to full-bandwidth RNICs without performance loss due to filtering, firewalls or other security measures.
3. 100 GbE network infrastructure for the RDMA network must provide Priority Flow Control to support correct IPU-over-Fabric (IPUoF) operation.
4. IPU-Machine BMC and IPU-Gateway management ports must not be accessible from the Poplar hosts to increases the security of the IPU-Machines.

a. They are connected to the control instance of each vPOD as this is required for the vipu-server and for maintenance and administration.

b. As they are separated from the end-users they can share VLANs and subnets which allows for simple monitoring and management over a ‘back-end’ network.

5. An independent secure access path (for ssh) to vPOD control instances must be provided that doesn’t rely on the health of Poplar instances (which are externally accessible).

1.8.2 Physical connectivity

All 100 GbE connectivity to the Top of Rack (ToR) switch within each POD$_{64}$ uses copper cabling by default, but using fibre and transceivers is also supported as an option.

All the 100 GbE connections from IPU Pod ToR switches to SPINE switches use optical cabling and Innolight transceivers (TR-ZC13H-N00 100G QSFP28 DR1 on the ToR switch end and T-OP4CNH-N00 OSFP DR4 on the SPINE switch end – connecting 4-to-1 cables).

There are two tested options for connecting the 1 GbE and 10 GbE management switches together across the racks:

- A dedicated site core network. All management switches have uplinks to a separate core network for rack-rack traffic and for switch remote management.
- Management switches are connected only to the 100 GbE switches in the same rack and VLANs are used to route management traffic over the shared 400 GbE SPINEs. This includes switch management traffic.

Whilst the second option reduces the physical connection complexity, it does make the management network dependent on the 400 GbE infrastructure meaning that a failure in the latter will impact the ability to even access switch and server management interfaces.

1.8.3 Openstack Overcloud networks

Control plane networks

There are 2 tested options for these networks:

- They are configured in OpenStack by the Kayobe host config to use VLANs that reside on the 100 GbE physical infrastructure, and share this with application workloads. This provides greater throughput and reduced bottlenecks in the control nodes but increases the impact of any failures in that infrastructure.
- They are configured to use VLANs solely on the physical management (1 GbE + 10 GbE) infrastructure. Whilst this increases isolation from workload data traffic, it has been shown to be vulnerable to the case where the control plan and control nodes are under heavy load. This can result in API protocol timeouts or other errors. An example would be when Cinder is copying disk images to hypervisors for VM launch.

Data plane networks

The data plane networks include any networks that are created inside tenant OpenStack projects. These include:

- IPU RDMA traffic
- Network storage traffic
- Internet access
- VM to VM management traffic, vPOD interactive logins

These networks are bound to the 100 GbE physical network in the Neutron ML2 configuration.

1.8.4 IP addressing and VLANs

The Graphcore standard global addressing policy for IPU Pods is used for the vPOD-local networks:

- 10.1.<logical rack number>.0/16: IPU-M BMC network shared by all vPODs.
- 10.2.<logical rack number>.0/16: IPU-M Gateway management shared by all vPODs.
- 10.3.<vPOD number>.0/24: vPOD-specific control network with an associated unique VLAN.
- 10.5.<vPOD number>.0/16: vPOD-specific IPU data RDMA network with an associated unique VLAN.

The logical rack number denotes the physical install location of the POD_{64} containing the IPU-Machines and the vPOD number is a unique number given to each vPOD.

Note: Multiple vPODs may use IPU-Machines from a single logical rack. In this case, the logical rack number will be common but the vPOD number will be distinct.

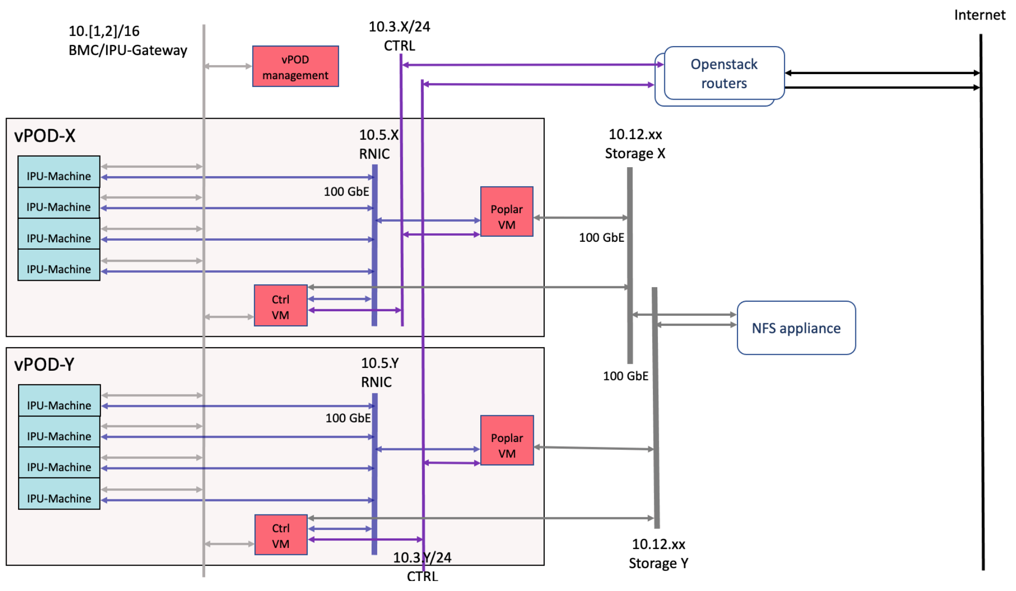

Fig. 1.6: Logical network view of vPODs

In addition, each vPOD may use this convention for a storage network:

- 10.12.<vPOD number>.0/24: 100 GbE storage network where a unique number is allocated to the vPOD and a unique VLAN is associated with this subnet.

DHCP is provided by Openstack Neutron and used for all vPOD interfaces including those on the IPU-Machines. This requires a configuration to be populated with MAC addresses for all IPU-Machine interfaces (BMC, IPU-Gateway and RNIC) to ensure IPs are assigned in standard order. For example, IPU-Machine #12 in logical rack 25 will be assigned:

- 10.1.25.12 for BMC
- 10.2.25.12 for IPU-Gateway
- 10.5.25.12 for RNIC

Addresses for interfaces on virtual machines are not constrained in this way and are allocated randomly from a DHCP pool by Neutron.

Warning: DNS within vPODs

Due to the number of networks connected to some VMs, there may be issues with DNS. Some OS clients have a limit on the number of listed DNS servers which can lead to an inability to resolve.

To avoid this issue, each VM is configured to only resolve via the 10.3 control network. This provides a relevant address for control and Poplar instances but not for IPU-Machines. Consequently these must be addressed by IP when creating VIPU agents or other admin tasks.

1.8.5 Traffic control

For correct IPUoF operation and performance, the 100 GbE network must implement Priority Flow Control but not any other negotiated traffic policies.

1.8.6 100 GbE networks with RDMA over Converged Ethernet (RoCE)

This is implemented with a vPOD-specific VLAN directly on the network switches for each vPOD.

All VMs are configured with direct mode Neutron ports when connecting to the IPU RDMA network. This allows for the required RDMA protocols to be used to the IPU-Machines.

Note: Each Mellanox card has a fixed number of Virtual Functions (VFs) available which limits the number of virtual machines which can be connected using direct mode on a single server or hypervisor.

The use of direct mode prevents the application of security groups (on current network cards) so access to the VLAN must be restricted explicitly by the infrastructure.

Neutron direct mode can also used for storage networks to obtain maximum performance.

1.8.7 Link aggregation

IPU Pod ToR switches

All connections from ToR 100 GbE switches use 8x 100 GbE links with 2x 4-way LAG to a pair of redundant 400 GbE SPINE switches.

Arista switches support active/active configurations providing up to 800 Gb/s of bandwidth and 400 Gb/s in fall-back mode in the case of a failures.

Poplar servers

For both Poplar bare-metal servers and where Poplar servers are used as hypervisors, a 2-way LAG is used to bond the pair of 100 GbE RNICs on each server giving up to 200 Gb/s aggregate throughput.

For hypervisors this bonded interface then carries all the VLANs associated with both the hypervisor (Overcloud networks) and the virtual machines (IPU and storage networks). See Section 1.5.3, Neutron (Open vSwitch) for more details.

1.8.8 Mellanox Connect-X 5

This network card is used in all physical Poplar servers, Ceph nodes and control nodes.

Virtual function passthrough

The Mellanox driver is modified to allow the maximum number of bonded-VF to be used (set 64 per card, default is 16).

See Section 1.5.3, Neutron (Open vSwitch) for related information.

1.8.9 vPOD logical networks

Security groups

The following traffic restrictions are enforced for VMs on the networks in a vPOD. This is configured in OpenStack Neutron.

Protocol	Port	Source/Destination	Reason
vPOD Controller VM interface in IPU-Machine BMC network			
Egress			
ICMP		IPU-Machine BMC addresses	Check connectivity
TCP	22 (ssh)	IPU-Machine BMC addresses	Remote access to shell
TCP	443 (https)	TBC	TBC
TCP, UDP	53 (dns)	DNS agent provided by Neutron	
Ingress			
UDP	514	IPU-Machine BMC addresses	Collect incoming logs from
vPOD Controller VM interface in IPU-Machine IPU-Gateway network			
Egress			
ICMP		Any	Check connectivity
TCP	22 (ssh)	IPU-Machine IPU-Gateway addresses	Remote access to shell
TCP	2112	IPU-Machine IPU-Gateway addresses	Collect metrics from Prom
TCP	8080	IPU-Machine IPU-Gateway addresses	Manage hardware using V-
TCP, UDP	53	DNS agent provided by Neutron	
Ingress			
TCP	22	Global management server	For remote administration
TCP	2113	Global management server	Expose metrics by Promet
TCP	3000	Global management server	Expose Grafana
TCP	9090	Global management server	Expose Prometheus for fe
UDP	514	IPU-Machine IPU-Gateway addresses	Collect incoming logs from
vPOD Controller VM interface in vPOD management network			
Egress			
Any	Any	Any	Egress to any allowed
Ingress			
TCP	22	Any	Remote access to shell
TCP	8090	Poplar VMs	Access for Poplar VMs to V
Poplar VM interface in vPOD management network			
Egress			
Any	Any	Any	Egress to any allowed
Ingress			

Table 1.1 – continued from previous page

Protocol	Port	Source/Destination	Reason
TCP	22	Any	Remote access to shell
TCP	9100	V-IPU Controller	Expose node exporter for I
Poplar VM in IPU data network			
No filtering possible: DIRECT NIC without security groups			
Poplar VM in storage network (examples only for NFS)			
Egress			
TCP, UDP	NFS	NFS server	Allow access to NFS storag
No ingress			

1.9 Appendix

This appendix contains example configuration files for the switches, servers and IPU-Machines.

1.9.1 Example SPINE switch configurations

Spine downlink member interface

```
interface Ethernet5/1
description Link-POD123-100G
dcbx mode ieee
flowcontrol send off
flowcontrol receive off
speed 100g-2
channel-group 123 mode active
priority-flow-control on
priority-flow-control priority 0 no-drop
priority-flow-control priority 1 no-drop
priority-flow-control priority 2 no-drop
priority-flow-control priority 3 no-drop
priority-flow-control priority 4 no-drop
priority-flow-control priority 5 no-drop
priority-flow-control priority 6 no-drop
priority-flow-control priority 7 no-drop
```

Spine port-channel

```
interface Port-Channel123
description LINK-POD123-100G
switchport trunk allowed vlan 1000-2000 # VLANs for overcloud/providers
switchport mode trunk
mlag 123
```

1.9.2 Example ToR switch configurations

IPUM access port

```
interface Ethernet9/1
description IPUMs Compute
dcbx mode ieee
switchport access vlan 1234 # managed by Ironic/Neutron/NGS
switchport trunk allowed vlan none
priority-flow-control on
priority-flow-control priority 0 no-drop
priority-flow-control priority 1 no-drop
priority-flow-control priority 2 no-drop
priority-flow-control priority 3 no-drop
priority-flow-control priority 4 no-drop
priority-flow-control priority 5 no-drop
priority-flow-control priority 6 no-drop
priority-flow-control priority 7 no-drop
spanning-tree portfast
```

Hypervisor access port

```
interface Ethernet1/1
description host1
dcbx mode ieee
flowcontrol send off
flowcontrol receive off
error-correction encoding reed-solomon
switchport access vlan 1002 # Overcloud provisisoning VLAN
channel-group 20 mode active
lacp port-priority 16000
priority-flow-control on
priority-flow-control priority 0 no-drop
priority-flow-control priority 1 no-drop
priority-flow-control priority 2 no-drop
priority-flow-control priority 3 no-drop
priority-flow-control priority 4 no-drop
priority-flow-control priority 5 no-drop
priority-flow-control priority 6 no-drop
priority-flow-control priority 7 no-drop
spanning-tree portfast
```

Hypervisor Portchannel

```
interface Port-Channel20
description T:host1:bond
switchport trunk allowed vlan 1000-2000 # VLANs for overcloud/providers
switchport mode trunk
port-channel lacp fallback individual
port-channel lacp fallback timeout 5
spanning-tree portfast
```

1.9.3 Mellanox ConnectX-5 configuration

Parameters:

- <PCIe device address>: Each device in the system will have a different PCIe address and therefore a unique <PCIe device address> parameter.
- <number of configured VFs>: The number of Virtual Functions (VFs) will depend on the maximum number of VFs the ConnectX-5 card supports and how many you choose to set per node.

Enable SRIOV:

```
mlxconfig -y -d <PCIe device address> set SRIOV_EN=1
```

Set number of VFs:

```
mlxconfig -y -d <PCIe device address> set NUM_OF_VFS=<number of configured VFs>
```

Enable LLDP DCBX PFC:

```
mlxconfig -y -d <PCIe device address> set LLDP_NB_DCBX_P1=TRUE LLDP_NB_TX_MODE_P1=2 LLDP_NB_RX_
↪MODE_P1=2 LLDP_NB_DCBX_P2=TRUE LLDP_NB_TX_MODE_P2=2 LLDP_NB_RX_MODE_P2=2
```

1.9.4 Dell R640 Intel virtualisation

Hypervisor configuration

Key nova.conf for R640 hypervisor:

```
[libvirt]
cpu_mode = host-passthrough
cpu_model_extra_flags = topoext
[compute]
# dedicate all CPUs, except the first 4 from each socket (along with it's paired thread)
cpu_dedicated_set = 0-95,^0,^48,^1,^49,^2,^50,^3,^51,^4,^52,^5,^53,^6,^54,^7,^55
```

Virtual Machine flavour configs (Terraform for OpenStack)

When creating virtual machines, the following metadata will create a large high-performance instance.

```
resource "openstack_compute_flavor_v2" "graphcore_flavor_r640_xlarge" {
    name = "r640.xlarge"
    vcpus = 80
    ram = 573440
    disk = 40
    ephemeral = 0
    is_public= false
    extra_specs = {
        "trait:HW_CPU_X86_INTEL_VMX " = "required"
        "hw:cpu_policy" = "dedicated"
        "hw:numa_nodes" = 2
        "hw:cpu_sockets" = 2
        "hw:cpu_threads" = 2
        "hw:cpu_thread_policy" = "prefer"
        "hw_rng:allowed" = "True"
        "hw:mem_page_size" = "1GB"
        "hw:pci_numa_affinity_policy" = "preferred"

    }
}
```

1.9.5 Dell R6525 AMD virtualisation

Hypervisor configuration

Key configuration for nova.conf:

```
[libvirt]
cpu_mode = host-passthrough
cpu_model_extra_flags = topoext
[compute]
# dedicate all CPUs except one from each NUMA zone (along with it's paired thread)
cpu_dedicated_set = 0-255,^0,^16,^32,^48,^64,^80,^96,^112,^128,^144,^160,^176,^192,^208,^224,^240
```

Virtual Machine flavour configs (Terraform for OpenStack)

When creating virtual machines, the following metadata will create a large high-performance instance that uses all the available CPU and RAM from a hypervisor.

Note the NUMA nodes settings matches the physical configuration of the host.

```
resource "openstack_compute_flavor_v2" "graphcore_flavor_r6525_full" {
    name = "r6525.full"
    flavor_id = "18d98749-f2c5-4bcc-a4bf-94eb65d5a101"
    vcpus = 240
    ram = 491520
    disk = 100
    ephemeral = 4300
    is_public= false
    extra_specs = {
        "trait:HW_CPU_X86_AMD_SVM" = "required"
        "hw:cpu_policy" = "dedicated"
        "hw:numa_nodes" = 8
        "hw:cpu_sockets" = 2
        "hw:cpu_threads" = 2
        "hw:cpu_thread_policy" = "require"
        "hw_rng:allowed" = "True"
        "hw:mem_page_size" = "1GB"
        "hw:pci_numa_affinity_policy" = "preferred"

    }
}
```

1.9.6 IPU-Machine Ironic driver

Ironic can be found here: https://github.com/openstack/ironic/tree/stable/wallaby

We use a modified version of the public driver, as shown below:

```
Patch:
ironic/drivers/redfish.py | 73 +++++++++++++++++++++++++++++++++++++++++++++
setup.cfg                 |  2 ++
2 files changed, 75 insertions(+)

diff --git a/ironic/drivers/redfish.py b/ironic/drivers/redfish.py
index d51e58b6f2..23afa0f07a 100644
--- a/ironic/drivers/redfish.py
+++ b/ironic/drivers/redfish.py
@@ -14,12 +14,20 @@
#    under the License.

from ironic.drivers import generic
+from ironic.common import states
+from ironic.drivers import base
+from ironic.drivers import generic
+from ironic.drivers import hardware_type
```

(continues on next page)

(continued from previous page)

```
+from ironic.drivers.modules import fake
from ironic.drivers.modules import agent
from ironic.drivers.modules import inspector
from ironic.drivers.modules import ipxe
from ironic.drivers.modules import noop
from ironic.drivers.modules import noop_mgmt
from ironic.drivers.modules import pxe
+from ironic.drivers.modules.network import flat as flat_net
+from ironic.drivers.modules.network import neutron
+from ironic.drivers.modules.network import noop as noop_net
from ironic.drivers.modules.redfish import bios as redfish_bios
from ironic.drivers.modules.redfish import boot as redfish_boot
from ironic.drivers.modules.redfish import inspect as redfish_inspect
@@ -27,6 +35,7 @@
from ironic.drivers.modules.redfish import power as redfish_power
from ironic.drivers.modules.redfish import raid as redfish_raid
from ironic.drivers.modules.redfish import vendor as redfish_vendor
+from ironic.drivers.modules.storage import noop as noop_storage


class RedfishHardware(generic.GenericHardware):
@@ -70,3 +79,67 @@ def supported_vendor_interfaces(self):
    def supported_raid_interfaces(self):
        """List of supported raid interfaces."""
        return [redfish_raid.RedfishRAID, noop.NoRAID, agent.AgentRAID]
+
+class RedfishNetworkAppliance(hardware_type.AbstractHardwareType):
+    """Redfish appliance moved between networks and rebooted using Ironic."""
+
+    @property
+    def supported_power_interfaces(self):
+        return [redfish_power.RedfishPower, fake.FakePower]
+
+    @property
+    def supported_inspect_interfaces(self):
+        """List of supported power interfaces."""
+        # TODO(johng): maybe we only want the port detection?
+        return [redfish_inspect.RedfishInspect, noop.NoInspect]
+
+    @property
+    def supported_network_interfaces(self):
+        """List of supported network interfaces."""
+        return [neutron.NeutronNetwork, flat_net.FlatNetwork,
+                noop_net.NoopNetwork]
+
+    @property
+    def supported_boot_interfaces(self):
+        """List of classes of supported boot interfaces."""
+        return [fake.FakeBoot]
+
+    @property
+    def supported_deploy_interfaces(self):
+        """List of supported deploy interfaces."""
+        return [NetworkOnlyDeploy]
+
+    @property
+    def supported_management_interfaces(self):
+        return [noop_mgmt.NoopManagement]
+
+    @property
+    def supported_raid_interfaces(self):
+        return [noop.NoRAID]
+
+    @property
+    def supported_rescue_interfaces(self):
+        return [noop.NoRescue]
+
+    @property
+    def supported_storage_interfaces(self):
+        return [noop_storage.NoopStorage]
```

(continues on next page)

(continued from previous page)

```
+
+
+class NetworkOnlyDeploy(fake.FakeDeploy):
+    """Class for only doing the network part of a typical deployment.
+    This does only the network setup,
+    then (optionally?) reboots the appliance via redfish,
+    letting DHCP do the heavy lifting.
+    """
+
+    @base.deploy_step(priority=100)
+    def deploy(self, task):
+        task.driver.network.configure_tenant_networks(task)
+        task.driver.power.reboot(task)
+
+    def tear_down(self, task):
+        task.driver.network.unconfigure_tenant_networks(task)
+        # TODO(johng): should we power it off?
+        task.driver.power.reboot(task)
+        return states.DELETED
\ No newline at end of file
diff --git a/setup.cfg b/setup.cfg
index 9d99f6dfff..12c8015fd1 100644
--- a/setup.cfg
+++ b/setup.cfg
@@ -91,6 +91,7 @@ ironic.hardware.interfaces.deploy =
    fake = ironic.drivers.modules.fake:FakeDeploy
    iscsi = ironic.drivers.modules.iscsi_deploy:ISCSIDeploy
    ramdisk = ironic.drivers.modules.pxe:PXERamdiskDeploy
+    network-only = ironic.drivers.redfish:NetworkOnlyDeploy

ironic.hardware.interfaces.inspect =
    fake = ironic.drivers.modules.fake:FakeInspect
@@ -102,6 +103,7 @@ ironic.hardware.interfaces.inspect =
    irmc = ironic.drivers.modules.irmc.inspect:IRMCInspect
    no-inspect = ironic.drivers.modules.noop:NoInspect
    redfish = ironic.drivers.modules.redfish.inspect:RedfishInspect
+    redfish-network-appliance = ironic.drivers.redfish:RedfishNetworkAppliance

ironic.hardware.interfaces.management =
    fake = ironic.drivers.modules.fake:FakeManagement

setup.cfg | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)

diff --git a/setup.cfg b/setup.cfg
index 12c8015fd1..fd835d426c 100644
--- a/setup.cfg
+++ b/setup.cfg
@@ -103,7 +103,6 @@ ironic.hardware.interfaces.inspect =
    irmc = ironic.drivers.modules.irmc.inspect:IRMCInspect
    no-inspect = ironic.drivers.modules.noop:NoInspect
    redfish = ironic.drivers.modules.redfish.inspect:RedfishInspect
-    redfish-network-appliance = ironic.drivers.redfish:RedfishNetworkAppliance

ironic.hardware.interfaces.management =
    fake = ironic.drivers.modules.fake:FakeManagement
@@ -186,6 +185,7 @@ ironic.hardware.types =
    redfish = ironic.drivers.redfish:RedfishHardware
    snmp = ironic.drivers.snmp:SNMPHardware
    xclarity = ironic.drivers.xclarity:XClarityHardware
+    redfish-network-appliance = ironic.drivers.redfish:RedfishNetworkAppliance

ironic.database.migration_backend =
    sqlalchemy = ironic.db.sqlalchemy.migration
```

1.10 Revision history

Version	Date	Notes
0.5	15th December 2021	Early access release
1.0	28th February 2023	General release
1.1	3rd April 2023	Minor corrections and additions

1.11 Trademarks & copyright