It's a Cloud..
it's a SuperComputer..
no, it's SuperCloud!

Jacob Anders - CSIRO

Erez Cohen – Mellanox

August Simonelli - Red Hat

About CSIRO

- CSIRO is Australia’s national science agency,
- Australia’s biggest consumer of scientific computing resources, with ~2PF of on-premise compute and ~50PB of storage,
- to lead in science, CSIRO needs the best, uncompromising IT.

Traditional workloads

- CPU/GPU compute (*focus: **CPU/GPU performance***)
- MPI workloads (*focus: **interconnect performance***)
- HTC (*focus: minimising job wait times*)
- sequential I/O (*focus: gigabytes per second*)
- **homogenous**, single-image systems.

Emerging fields of research

- cybersecurity research — *(focus: **security** / isolation)*
- machine learning — *(focus: **adaptability**)*
- bioinformatics — *(focus: **interactive** workloads)*

(all of the above: I/O operations per second + gigabytes per second
***heterogenous, loosely coupled** system*
***self service** capability an asset)*

At the crossroads

- Conflicting requirements lead to disconnected fields with own tools,
- The evolution of computing proves the best outcomes are achieved when development efforts converge, leading to creation of more powerful, multi-purpose tools.

The Vision

- We want to make HPC more cloud-like.. (or cloud more HPC-like)
- Create one system that can support a wide range of workloads:
    - bare metal HPC,
    - high performance storage,
    - batch queue and interactive workloads,
    - virtualised and containerised applications,
    - multi-tenancy and fabric isolation,
- Combine Supercomputing and Cloud into one, creating the **SuperCloud**.

How are HPC and Cloud different?

- Interconnect,
- Performance,
- Multi-tenancy and isolation,
- Homogeneous vs heterogeneous model,
- Batch queue vs DevOps.

Interconnect - Mellanox

Erez Cohen

SuperCloud requires SuperNetwork

- Lots of data -> High throughput
- Fast -> Low latency
- Efficient -> HW offload
  - Transport
  - Virtualization
  - Storage
  - Security
- Flexible -> Software Defined
- Standard, off the shelf, open source

InfiniBand – The High Performance SDN Network

- InfiniBand address all our needs
  - SDN by design
  - Up to 200Gbps with sub 1us latency
  - Fully HW offloadable
  - Open standard, open source
  - Bare metal and Vitalization
  - Work with CPU and GPU based computing
  - While high performance, designed for general networking use cases

InfiniBand Accelerates Leading HPC and AI Systems

World's Top 3 Supercomputers

OAK RIDGE National Laboratory
U.S. DEPARTMENT OF ENERGY

1

Summit CORAL System
World's Fastest HPC / AI System
9.2K InfiniBand Nodes

TOP 500 The List.

Lawrence Livermore National Laboratory
U.S. DEPARTMENT OF ENERGY

2

Sierra CORAL System
#2 USA Supercomputer
8.6K InfiniBand Nodes

TOP 500 The List.

国家超级计算无锡中心
National Supercomputing Center in Wuxi

3

Wuxi Supercomputing Center
Fastest Supercomputer in China
41K InfiniBand Nodes

TOP 500 The List.

openstack CSIRO Mellanox TECHNOLOGIES Red Hat

InfiniBand Accelerates Record-Breaking AI Systems

ImageNet training record breakers

facebook

P100 x 256, EDR InfiniBand

Scaling efficiency **~90 %**

Preferred Networks

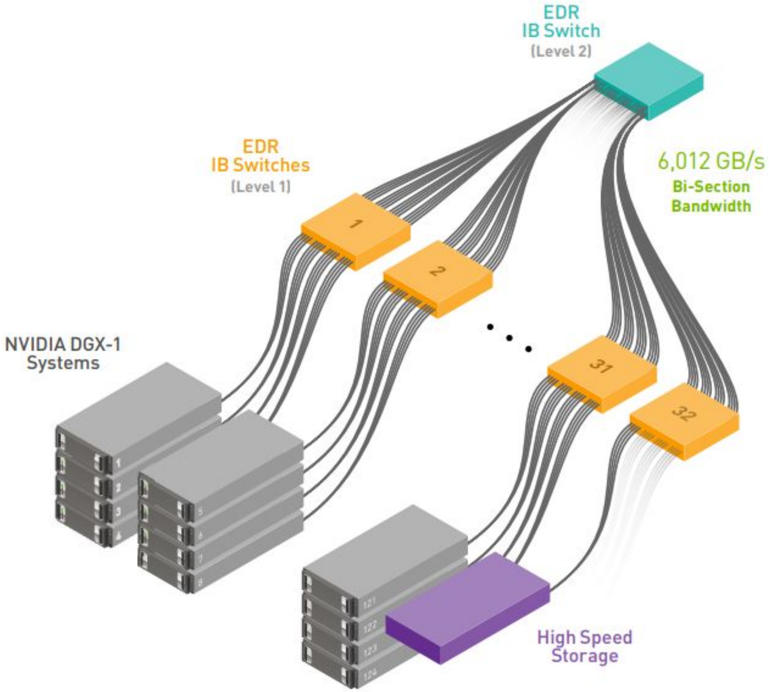
P100 x 1024, FDR InfiniBand

Scaling efficiency **80 %**

SONY

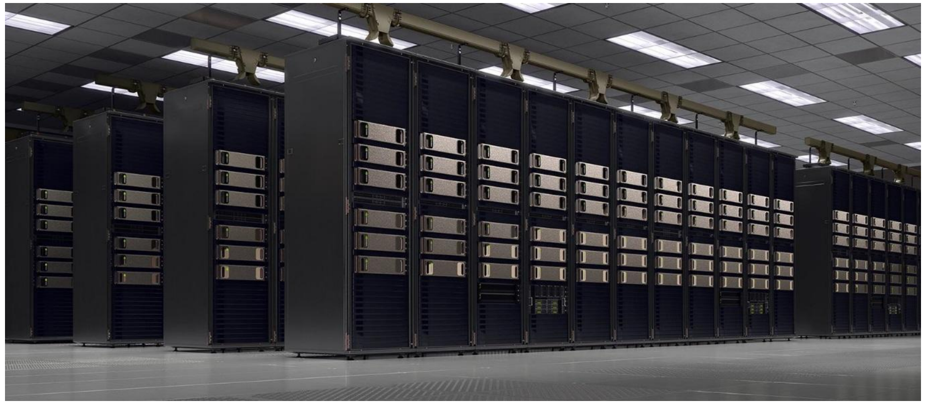
V100 x 1088, EDR InfiniBand

Scaling efficiency **91.62 %**



NVIDIA DGX SATURNV

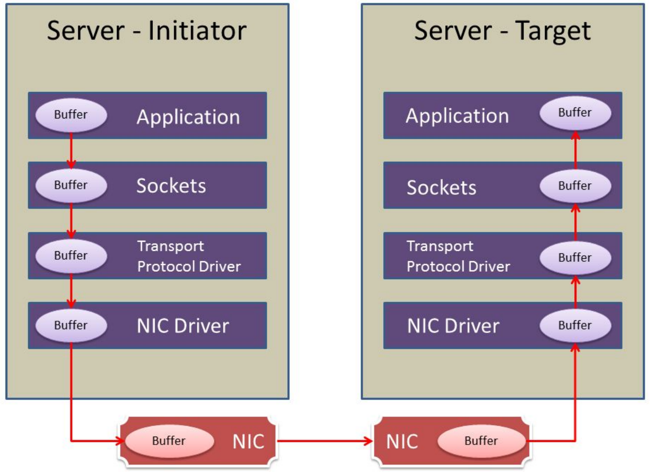
- 124 DGX-1 nodes interconnected by 32 L1 TOR Switches, in 2016
- Mellanox 36 port EDR L1 and L2 switches, 4 EDR per system
- Upgraded to 660 NVIDIA DGX-1 V100 Server Nodes, in 2017
- 5280 V100 GPUs, 660 PetaFLOPS (AI)



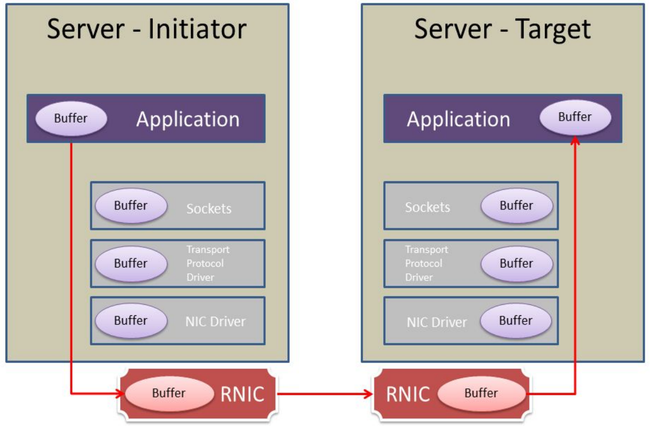
InfiniBand At A Glance

- Industry standard defined by the InfiniBand Trade Association
- Defines System Area Network architecture
- High Bandwidth Links – up to 200G (HDR)
- Remote Direct Memory Access (RDMA)
    - Full CPU Offload - Hardware Based Transport Protocol
    - Kernel Bypass - Ultra low latency
    - Remote memory Read/Write
- Reliable, lossless, self-managing fabric
- All major HPC, ML, Storage and big data frameworks heavily utilizing RDMA

TCP/UDP

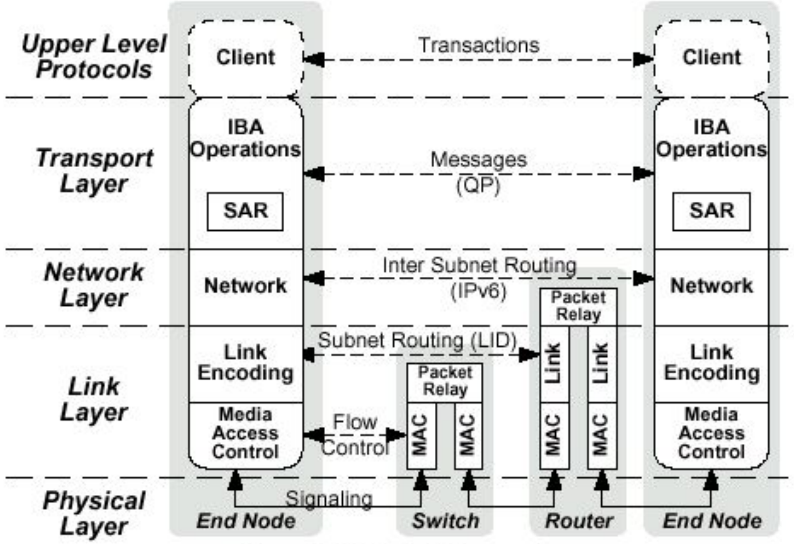


RDMA



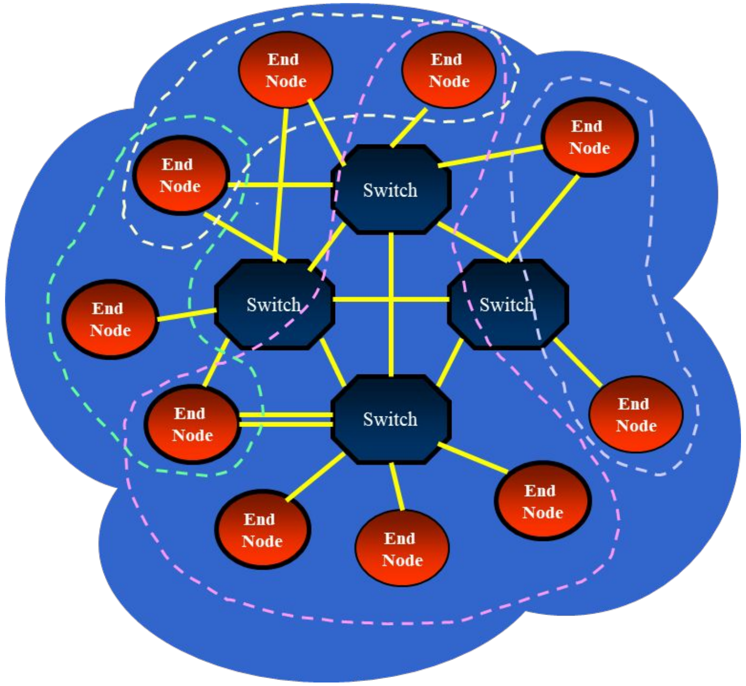
(The very) Basics of InfiniBand

- Network addressing
  - Global Unique ID (GUID) – Fixed, global address (~MAC)
  - Local ID (LID) – Transient ID (~IP)
- Network partitioning
  - Partition key (pkey) – Network segmentation (~VLAN)
- Verbs
  - RDMA API (~sockets)
- Upper Layer Protocols (ULPs)
  - Shim layer to connect legacy network and storage to verbs



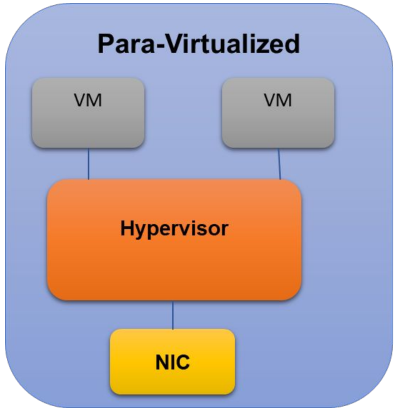
SubnetManager (SM) – InfiniBand SDN

- InfiniBand was design as an SDN network from scratch
- The SM assigns network addresses, segmentation, QOS etc.
- Fabric management is done in band
  - No need for external network or CPU
- Neutron integration available from Mellanox
- Out of the box support for all topologies
  - FAT tree, hypercube, Torus, Dragonfly+ and more

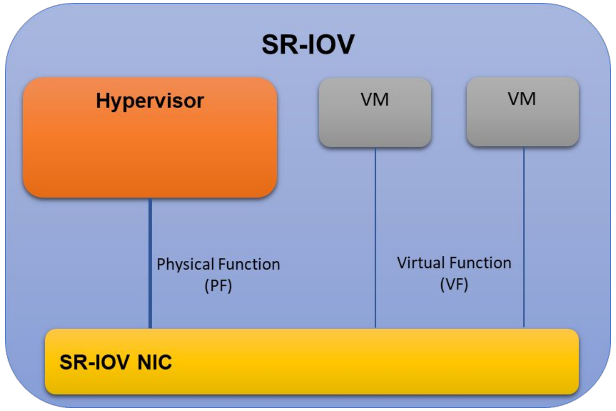


RDMA – Critical For Modern Applications

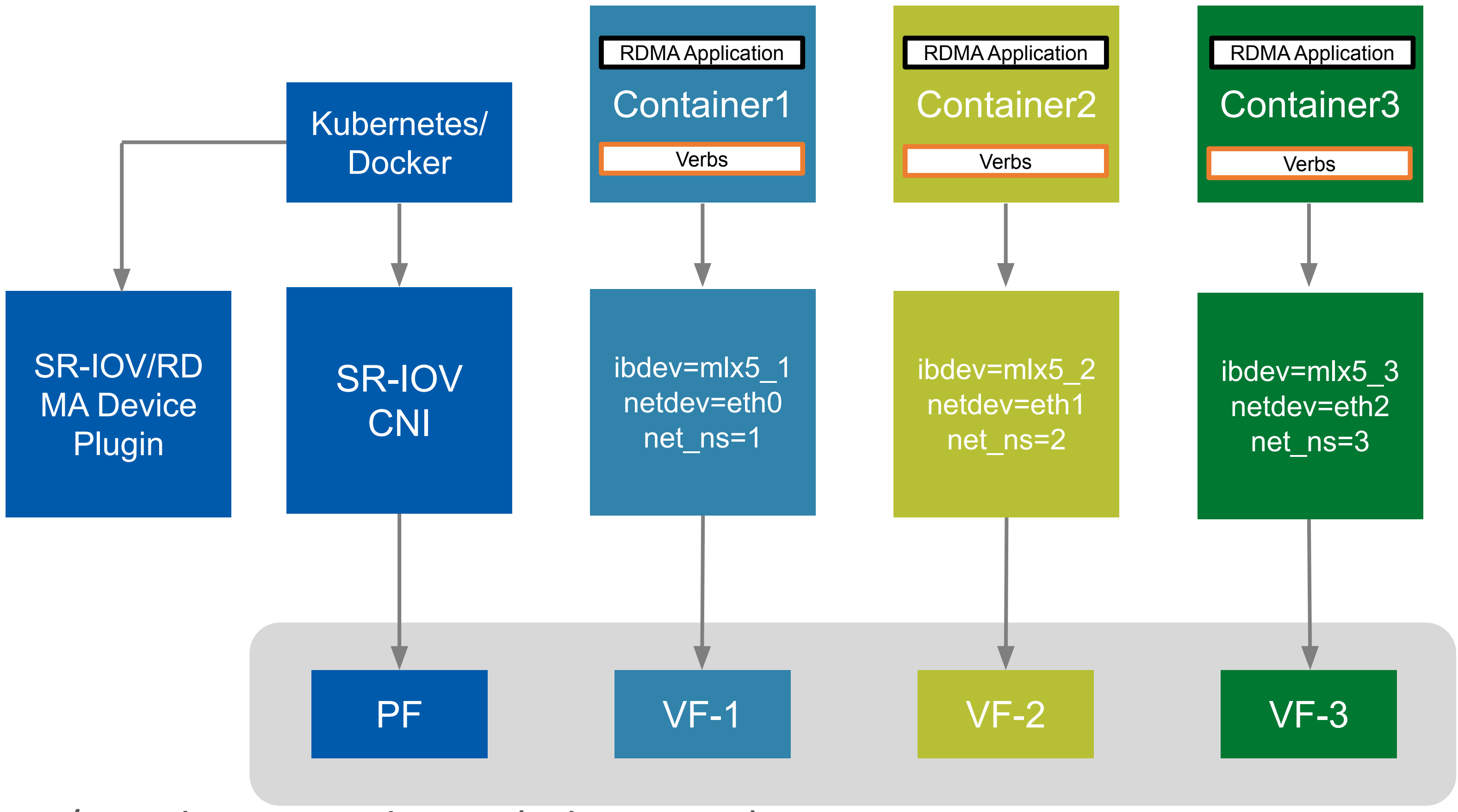
- All SupperCloud applications should be able to use RDMA
  - Bare metal, VM, Containers



- SR-IOV is the main vehicle to provide RDMA and other advance services to applications
  - Fully automated for OpenStack and Containers (CNI)
  - Additional offload for the hypervisor



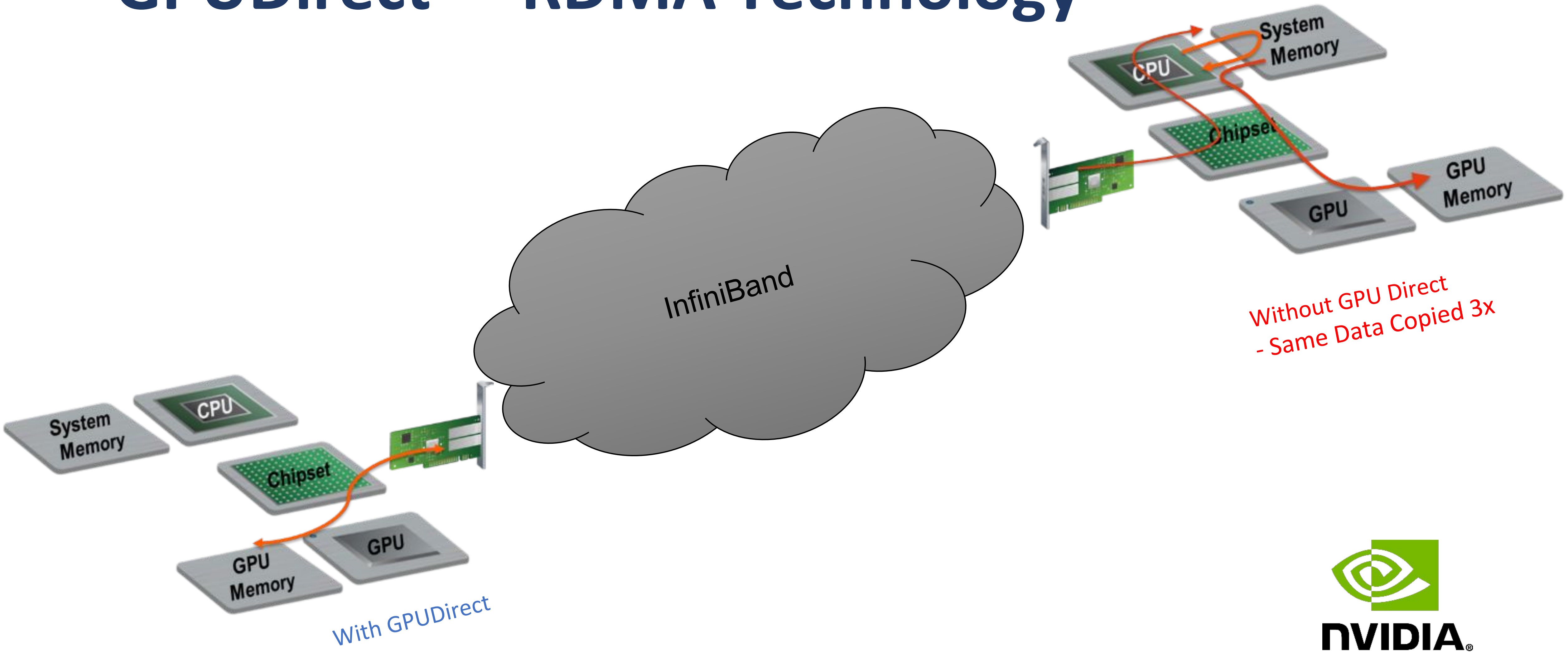
RDMA at the Container Level



Mellanox ConnectX Adapter Card with SR-IOV Enabled

- Every container/POD has an IB device (mlx5_1,2,3)
- Isolation is on the driver level

GPUDirect™ RDMA Technology



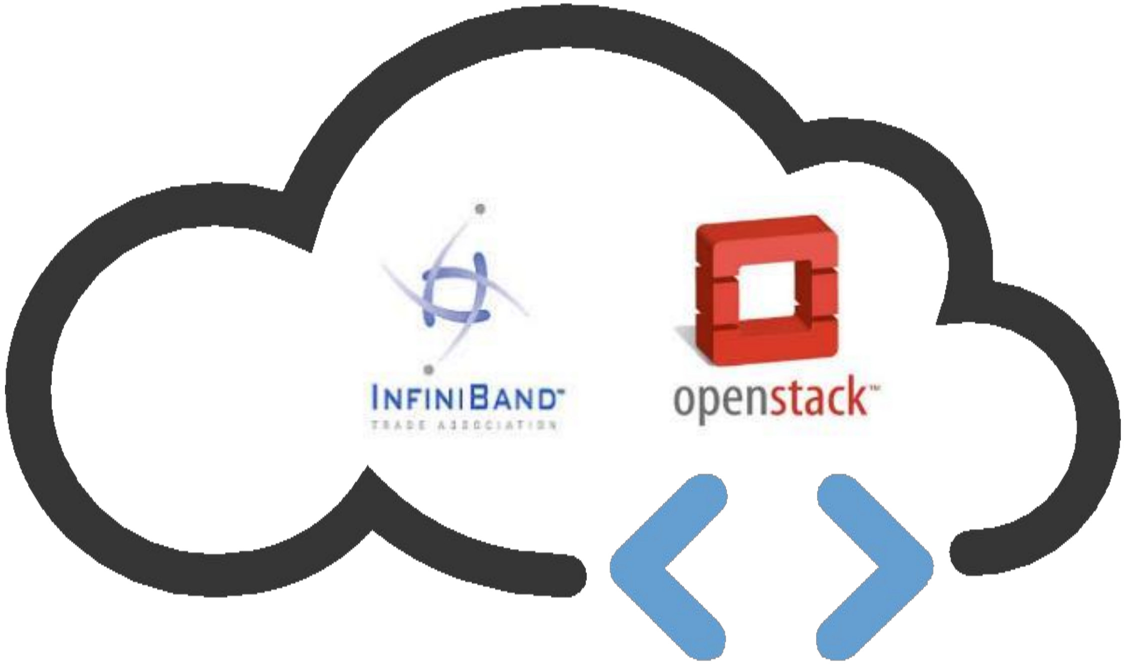
Ironic IB Support

- Ironic: OpenStack Bare Metal Provisioning Program
- Initially developed to provision bare metal servers as part of OpenStack deployment
- Provision servers similarly to Virtual Machine
  - API driven
  - All HW exposed to user including GPUs, FPGAs etc.
  - GPUDirect available
- Support multi-tenancy
- InfiniBand support for Ironic enables HPC/ML over OpenStack!
  - SW defined data center with bare metal performance!

OpenStack Over InfiniBand – The Route To Extreme Performance

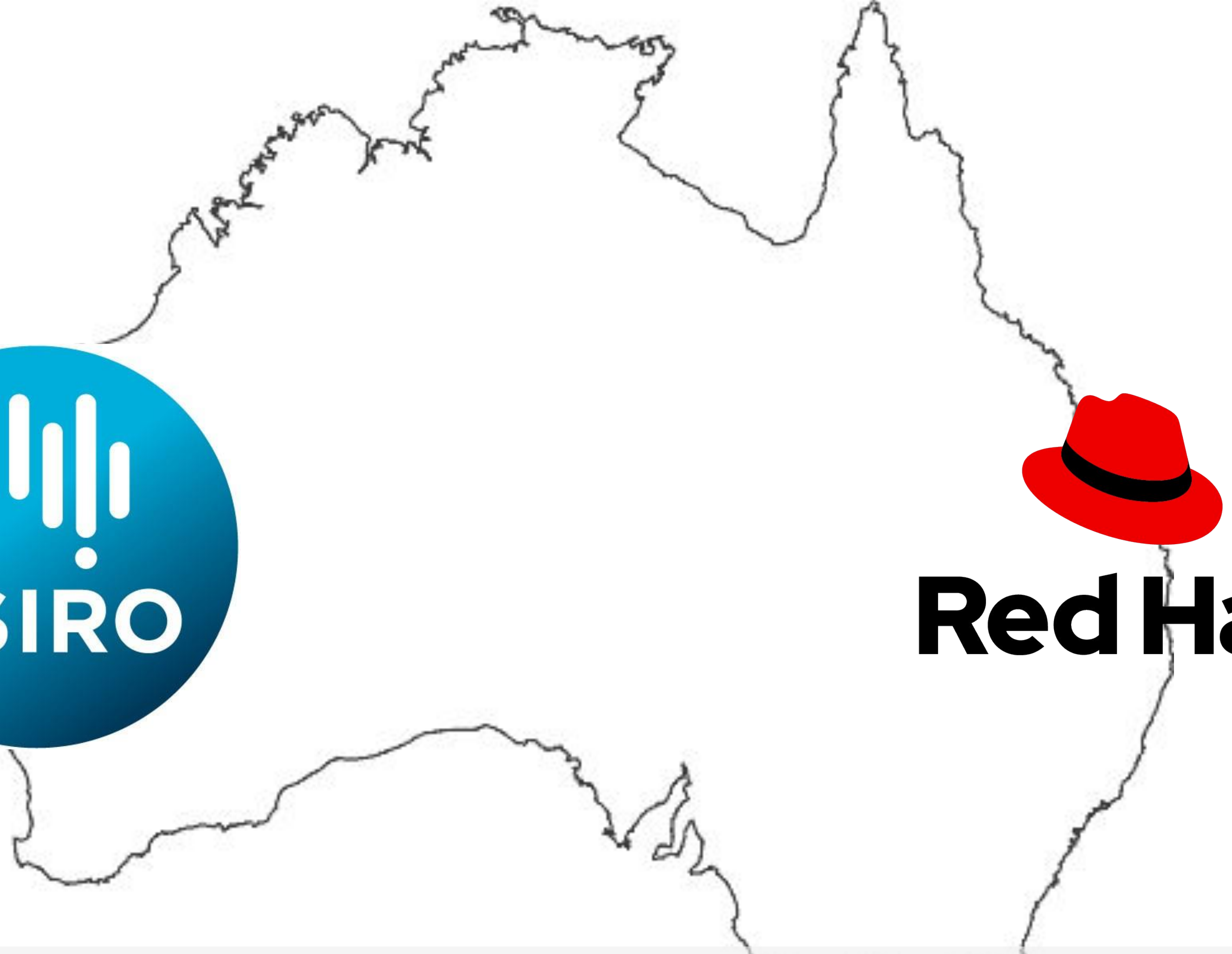
- Transparent InfiniBand integration into OpenStack
  - Since Havana…

- MAC to GUID mapping
- VLAN to pkey mapping
- InfiniBand SDN network
  - Integrated with Neutron
  - Automated with Mellanox UFM fabric manager



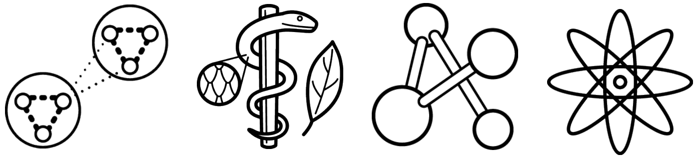
Red Hat

August

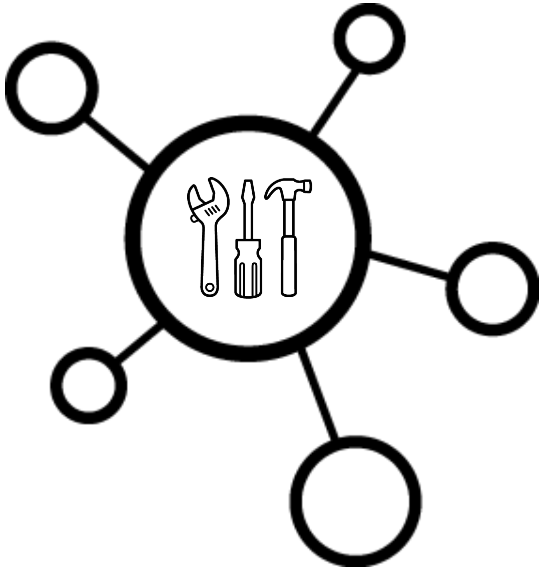


<u>Supporting</u> CSIRO's vision to combine supercomputing and cloud.

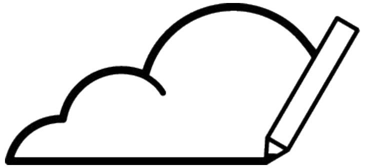
We want to make HPC more cloud-like.



COMMUNITY



We want to make everything more cloud-like.



We have the tools to do it all!

So we decided if we work together we could optimise the best of all of these worlds.

COMMUNITY

Downstream Support via
Architecture, Training,
Product Management, Etc

Red Hat

Upstream Support via
Project Leadership,
Engineering, Testing, Etc

Act as an extension of the CSIRO team letting them focus on what they are good at: innovation!

So we decided if we work together we could optimise the best of all of these worlds.

CSIRO

COMMUNITY

Downstream Support via
Architecture, Training, Product Management, Etc

Red Hat

Upstream Support via
Project Leadership, Engineering, Testing, Etc

This means finding ways to use the same tooling for HPC workloads as for cloud workloads.

Working Together.

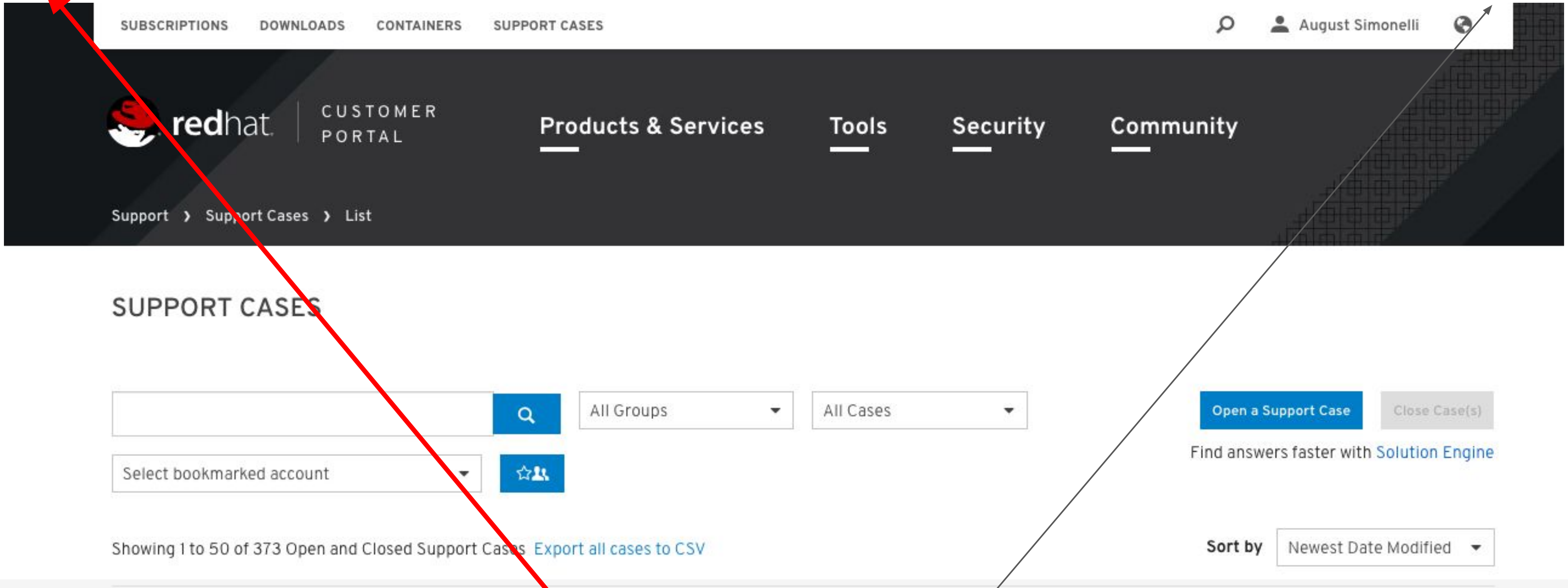
CSIRO using code from upstream identifies an issue and hacks the code to get it working ;)

```
            a malformed response from the agent.
        :raises: AgentAPIError when agent failed to execute specified command.
        :returns: A dict containing command result from agent, see
                  get_commands_status for a sample.
        """
        url = self._get_command_url(node)
        body = self._get_command_body(method, params)
        request_params = {
            'wait': str(wait).lower()
        }
        LOG.debug('Executing agent command %(method)s for node %(node)s',
                  {'node': node.uuid, 'method': method})

        try:
            response = self.session.post(url, params=request_params, data=body)
        except requests.ConnectionError as e:
            msg = (_('Failed to connect to the agent running on node %(node)s '
                     'for invoking command %(method)s. Error: %(error)s') %
                   {'node': node.uuid, 'method': method, 'error': e})
            LOG.error(msg)
            raise exception.AgentConnectionFailed(reason=msg)
        except requests.RequestException as e:
            msg = (_('Error invoking agent command %(method)s for node '
                     '%(node)s. Error: %(error)s') %
                   {'method': method, 'node': node.uuid, 'error': e})
            LOG.error(msg)
            raise exception.IronicException(msg)

        # TODO(russellhaering): real error handling
        try:
            result = response.json()
        except ValueError:
```

Red Hat

Working Together.

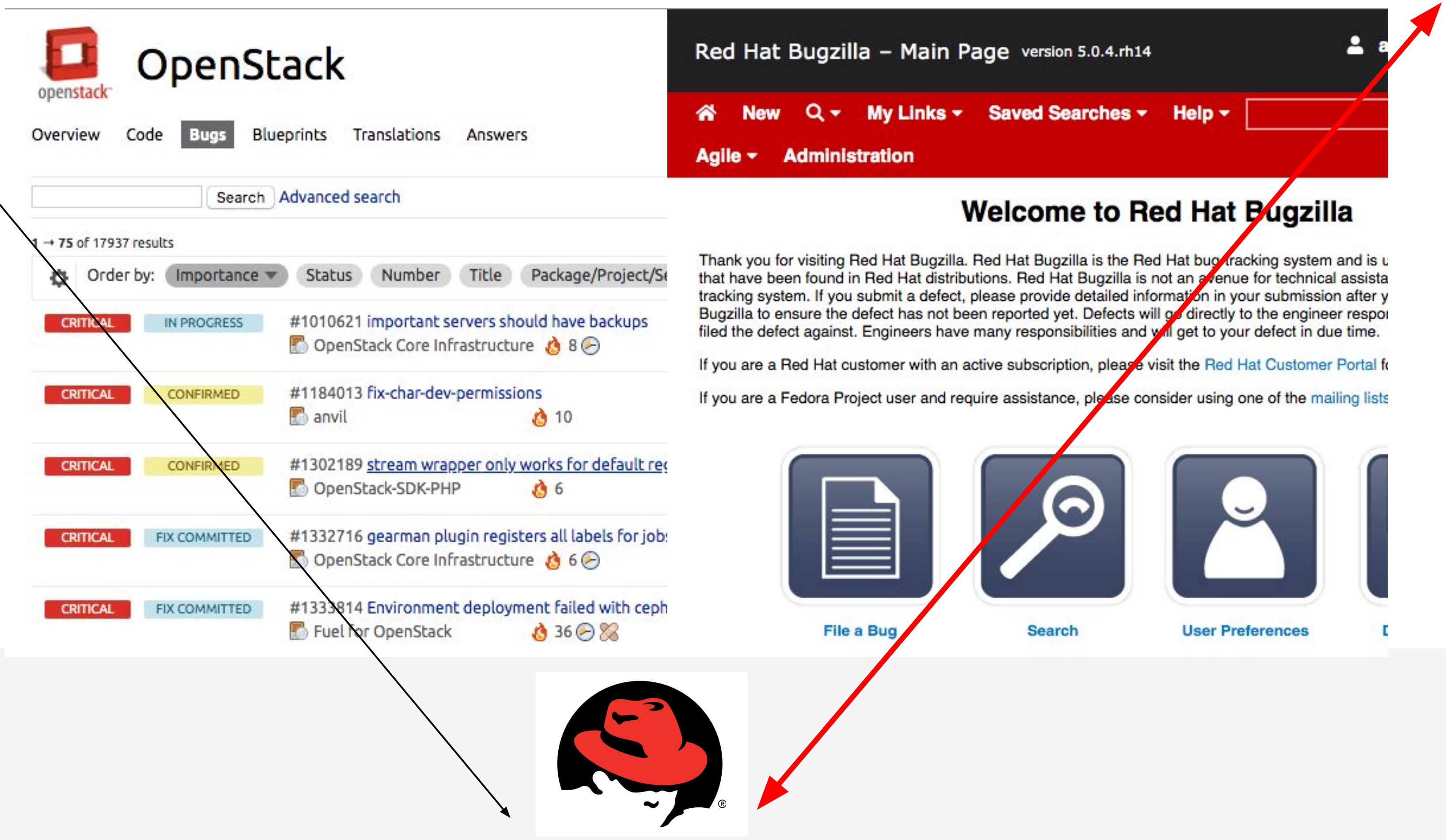
CSIRO opens an internal case and works closely with the Red Hat teams to test and review the possible change.



Red Hat

Working Together.

Red Hat and upstream community work together to bring the best version of the change to reality.



KEY AREAS OF FOCUS FOR SUPERCLOUD

INTEGRATION AND CERTIFICATION

BARE METAL DEPLOYMENTS

IRONIC

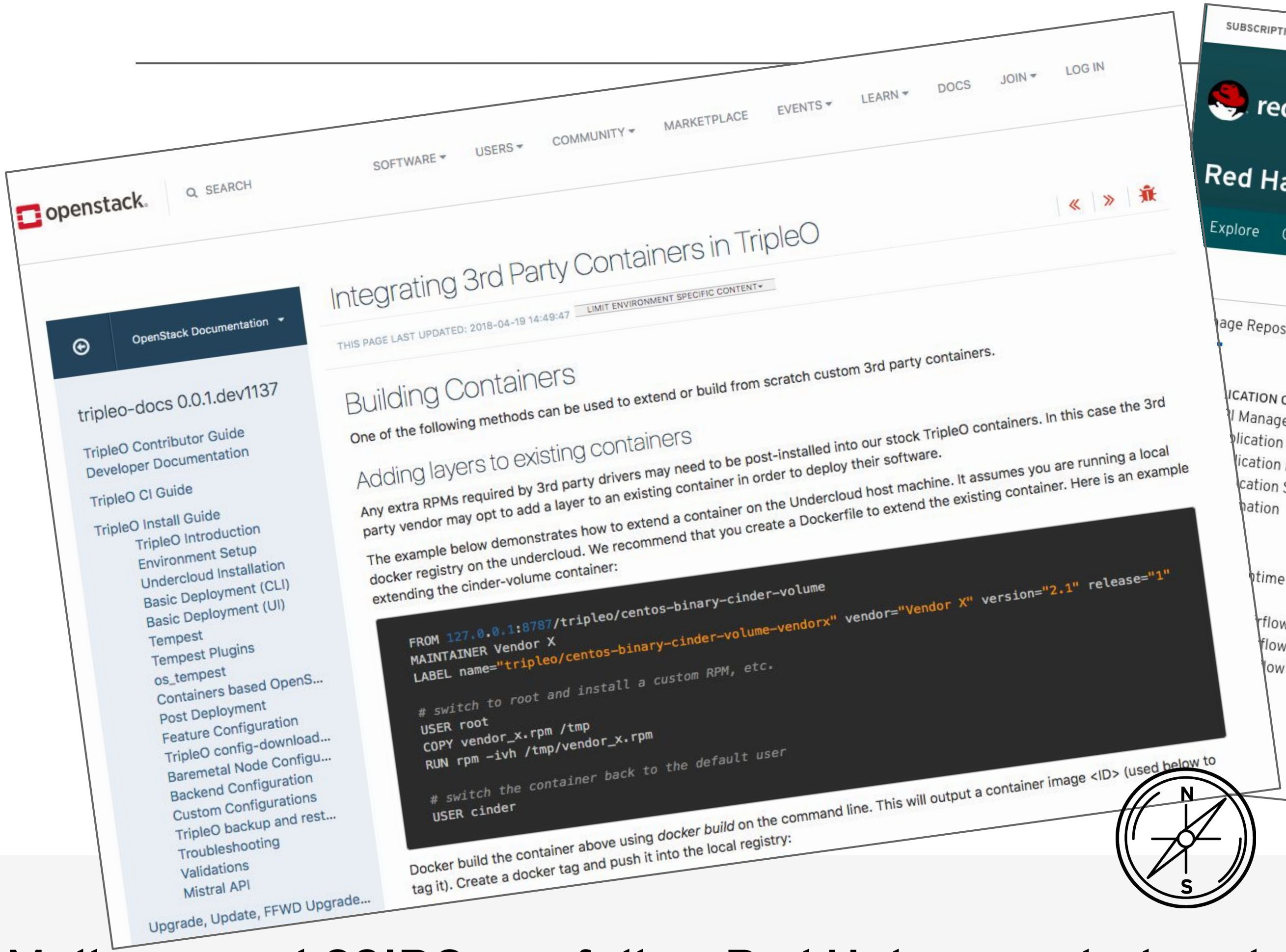
an OpenStack Community Project

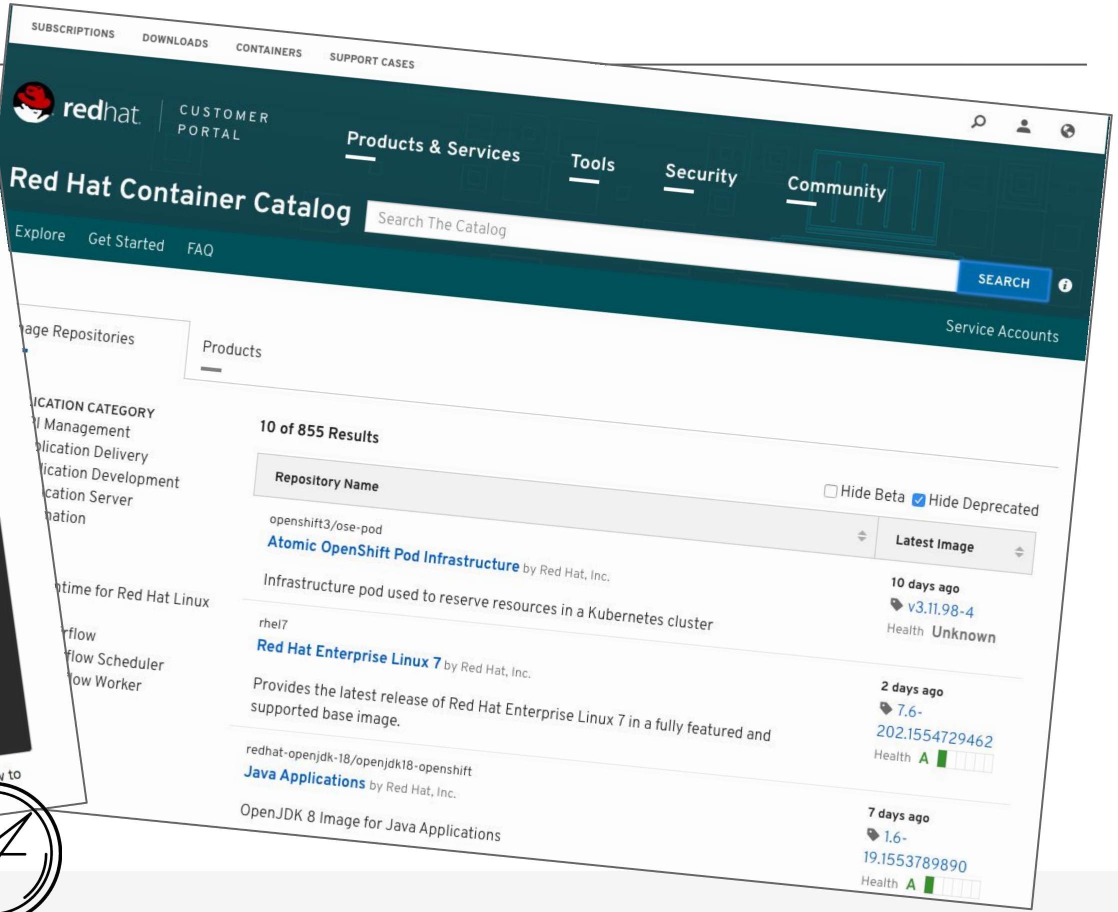
AUTOMATION AND LIFECYCLE

TRIPLEO

an OpenStack Community Project

INTEGRATION GUIDANCE AND CERTIFICATION





Mellanox and CSIRO can follow Red Hat supported upstream documentation to generate a certifiable offering to commercial customers and a tested and reliable upstream solution for the community.

SUPPORTING CSIRO'S OPERATIONAL FRAMEWORK FOR MANAGING OPENSTACK

TripleO provides a well-defined, supported, best practice OpenStack deployment to SuperCloud.

TRIPLEO
an OpenStack Community Project

Ironic provides integrated bare metal as a service capabilities from TripleO for SuperCloud.

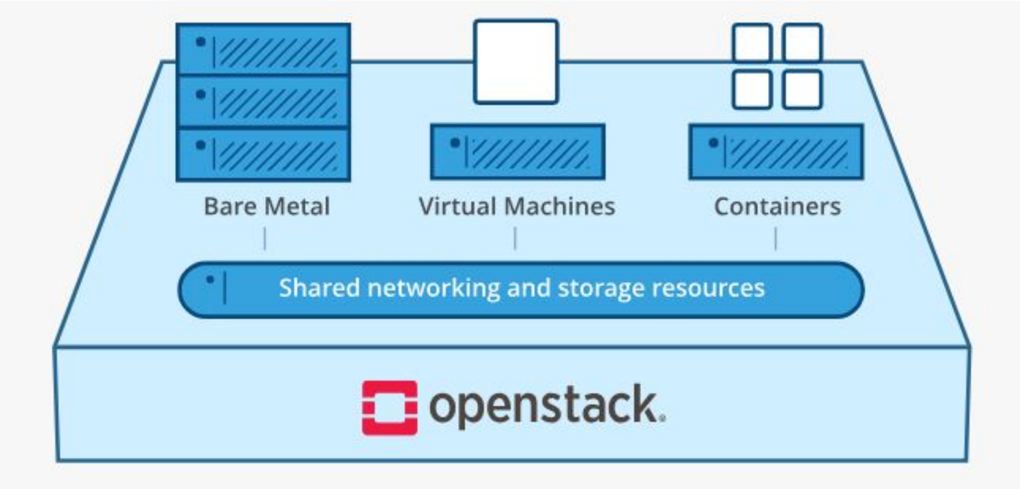
IRONIC
an OpenStack Community Project

Spend less time dealing with the "plumbing."

System review

Jacob

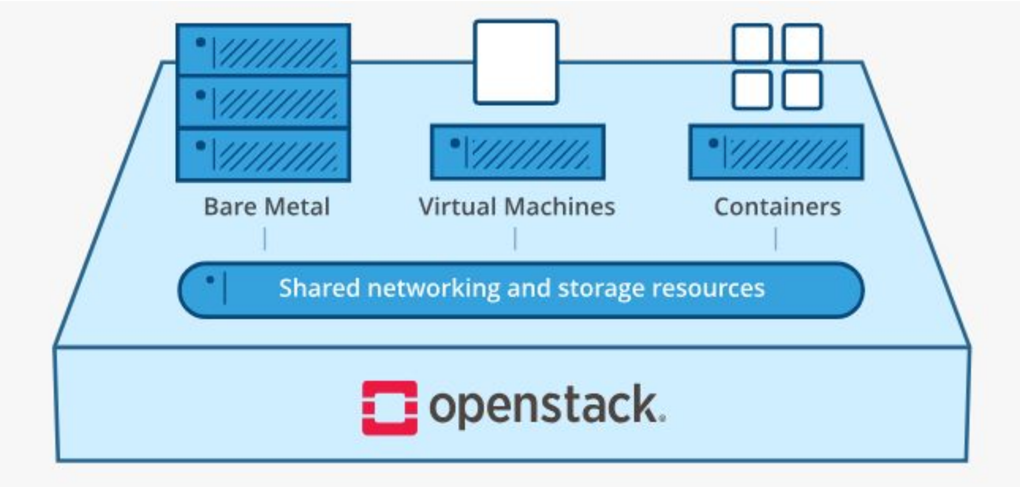
SuperCloud Design



Based on OpenStack with **bare-metal provisioning** and **SDN InfiniBand**:

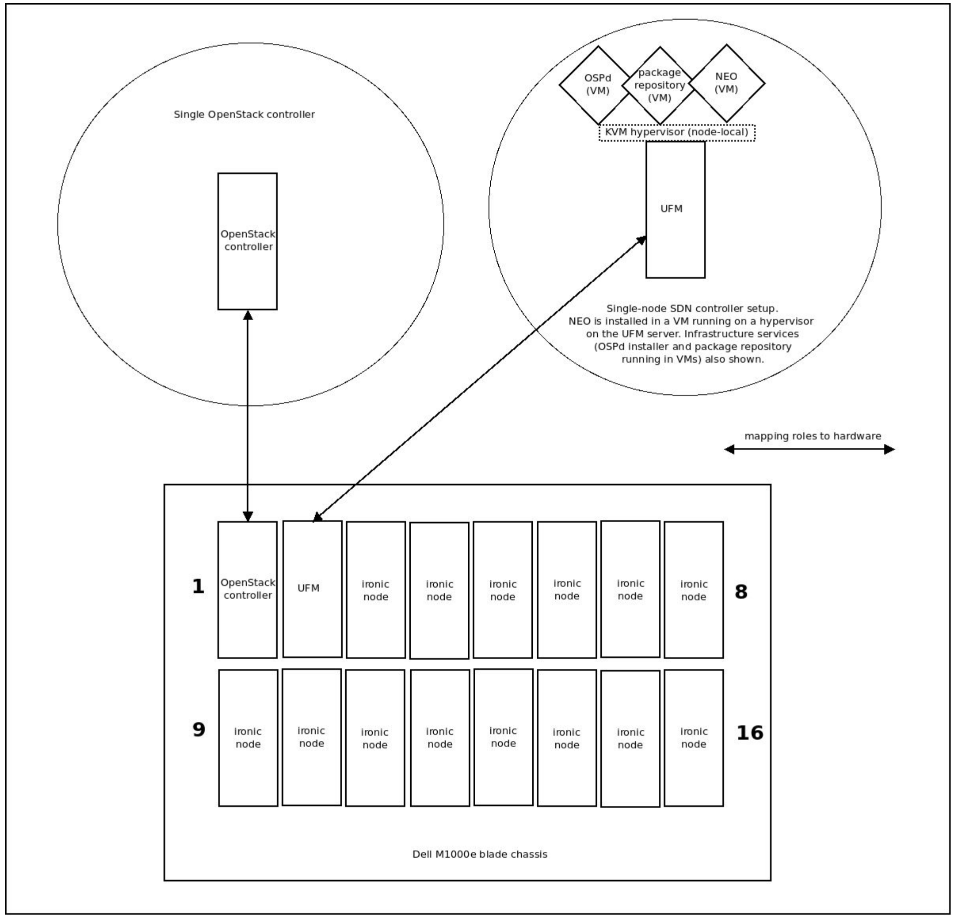
- Offers **performance** of **HPC** with the **flexibility of Cloud**,
- Software Defined Networking (SDN) allows running **workloads in isolation** for **multi-tenancy,**
- Supports a wide variety of workloads with **no trade-offs** between performance and functionality

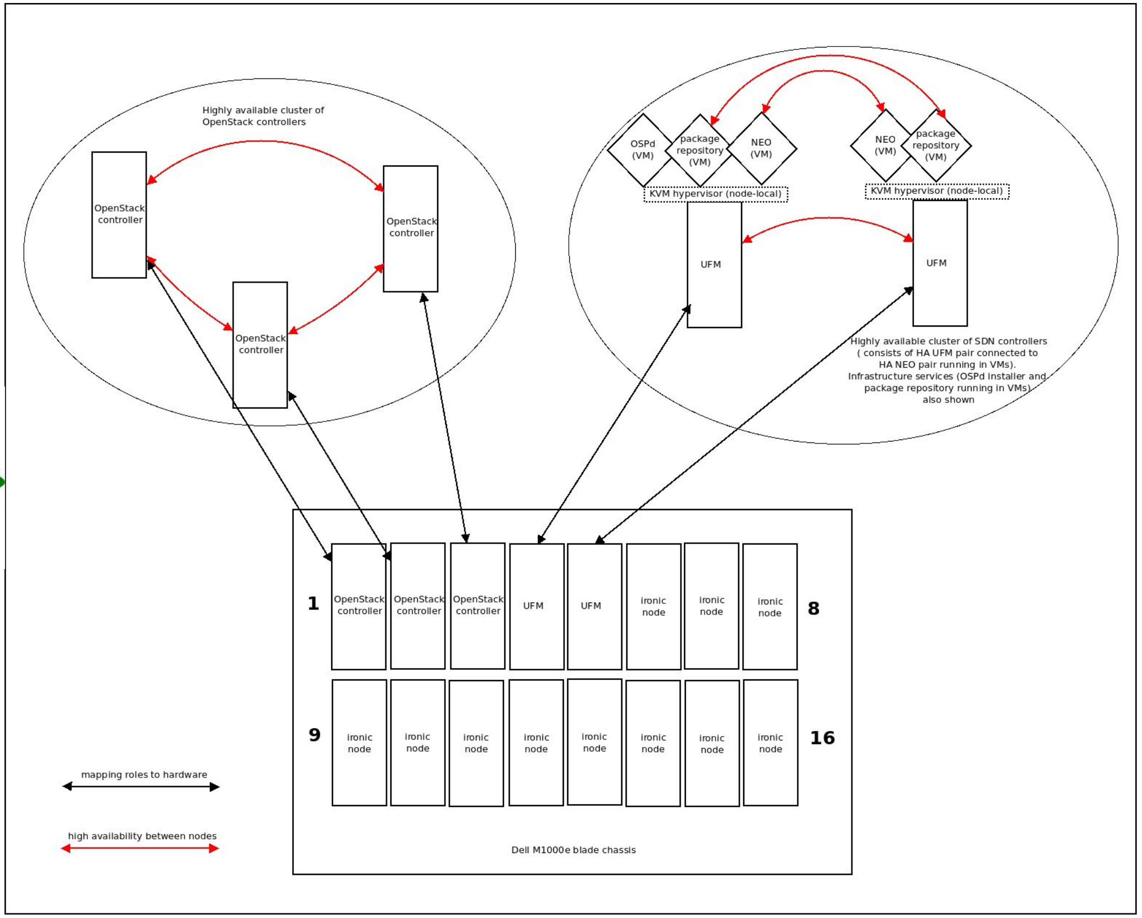
SuperCloud Design (2)



- **Core infrastructure is minimal** - we start with controllers and a pool of bare-metal,
- **nova-compute** is running in bare-metal instances,
- **Storage** is running in bare-metal instances and **software-defined** for **maximum flexibility**
- building upon the technology **first presented in Vancouver:** https://www.openstack.org/videos/vancouver-2018/ironing-the-clouds-a-truly-performant-bare-metal-openstack-1

Making SuperCloud highly available using built-in TripleO controller clustering





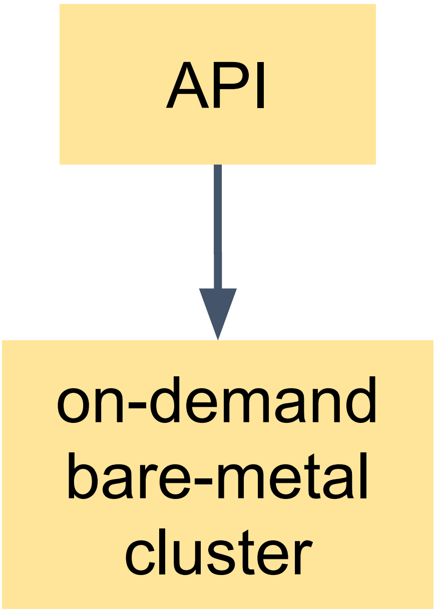
Adding InfiniBand connected containers (baremetal, Skylake, ConnectX5)

```
# docker run -it --privileged  mellanox/mofed421_docker:latest bash
root@41bf48b224a8:/tmp# ib_write_bw 172.17.0.2
---------------------------------------------------------------------------
                    RDMA_Write BW Test
 Dual-port        : OFF          Device         : mlx5_0
 Number of qps    : 1            Transport type : IB
 Connection type  : RC           Using SRQ      : OFF
 TX depth         : 128
 CQ Moderation    : 100
 Mtu              : 4096[B]
 Link type        : IB
(...)
---------------------------------------------------------------------------
local: LID 0x03 QPN 0x008d PSN 0x163d5b RKey 0x00e106 VAddr 0x007f923c550000
remote: LID 0x03 QPN 0x008c PSN 0x68b1bf RKey 0x00e409 VAddr 0x007fdbbded0000
------------------------------------------------------------------------------------
-
 #bytes     #iterations    BW peak[MB/sec]    BW average[MB/sec]   MsgRate[Mpps]
 65536      5000             11741.50             11739.07             0.187825
------------------------------------------------------------------------------------
-
root@41bf48b224a8:/tmp#
```

Bringing Infrastructure-as-Code to bare-metal

SuperCloud **brings Infrastructure-as-Code** methodology **to bare-metal**,

- Allows "**programming the hardware**" without the need for virtualisation layer
- Entire bare-metal systems can be programmatically **created and deleted** as required, including **compute, networking and storage,**
- **Infrastructure details** can be **abstracted away -** think of **Python programmer's** perspective on **Assembly code**



Example 1: software-defined Slurm on bare-metal

The first example will demonstrate creating a software-defined bare-metal Slurm HPC cluster using ElastiCluster package

- **ElastiCluster interacts** directly **with OpenStack APIs** to create the required compute, networking and storage resources,
- The **infrastructure** is **described with** simple **code,**
- The cluster can be **scaled up and down at runtime,**
- **Resources can be** easily **released** when no longer required **and repurposed** into other software-defined systems

Example 1: software-defined Slurm on bare-metal

```
$ git clone git://github.com/gc3-uzh-ch/elasticluster
$ cd elasticluster
$ pip install -e .
$ vim ~/elasticluster/.config

[ cloud / openstack ]
provider = openstack
auth_url = http://192.168.2.10:5000/v3
project_name = SCA19
username = sca19
password =*******
region_name = RegionOne
[ login / cloud-user ]
image_user = cloud-user
image_sudo = True
user_key_name = sca19
```

Example 1: software-defined Slurm on bare-metal

```
[ setup / slurm ]
provider = ansible
master_groups = slurm_master, ganglia_master
worker_groups = slurm_worker, ganglia_monitor
submit_groups = slurm_submit
global_ var _multiuser_cluster = yes

[ cluster / slurm ]
setup = slurm
master_nodes = 1
worker_nodes = 4
ssh_to = master
cloud = openstack
flavor = baremetal
network_ids = d4569eaa-0972-410f-afc3-98828a081eea
security_group = default
image_id =5d0625a1-e814-4e6f-b8a8-fd86597b303f
```

Demo: software-defined Slurm on bare-metal

Example 2: ephemeral hypervisors

The second example demonstrates the creation of ephemeral hypervisors in a bare-metal SuperCloud instances using ansible

- SuperCloud has no native virtualisation capability,
- If virtualisation is required, secondary virtualisation capability can be added using this method,
- Capacity can be scaled to match the current demand.

Example 2: ephemeral hypervisors

```
- name: OpenStack infrastructure
  hosts:
    - ansible
  vars:
    state: present
  tasks:
  roles:
    - role: 050-neutron-ports
      new_state: "{{ state }}"
    - role: 060-baremetal-instances
      new_state: "{{ state }}"
    - role: 080-update-inventory
- hosts: hypervisors
  gather_facts: no
  tasks:
  roles:
    - role: 084-connection-wait
```

```
- name: Nova-compute deployment
  hosts:
    - hypervisors
  vars:
    state: present
  tasks:
  roles:
     - role: 085-networking
       new_state: "{{ state }}"
     - role: 100-yum-install
       new_state: "{{ state }}"
     - role: 120-configfiles
       new_state: "{{ state }}"
     - role: 130-services
     - role: 140-reboot
       new_state: "{{ state }}"
     - role: 150-services-check
```

Demo: ephemeral hypervisors

Example 3: deploying a containerised interactive workload

The final example will demonstrate deploying a containerised precision medicine workload in a bare-metal instance running on SuperCloud

- This workflow was **written by Dr. Denis Bauer (CSIRO Health & Biosecurity), Piotr Szul (Data61)** and their team,
- based on **VariantSpark, a ML library** for detecting disease genes,
- The analysis steps are illustrated on the HipsterIndex dataset, which **simulates how complex diseases work** that are caused by the interplay of multiple genes.

Example 3: deploying a containerised interactive workload

```
demo_instance_secgroup: "default"
demo_instance_flavor: "baremetal"
demo_instance_image: "rhel-7.5-baremetal"
demo_instance_count: 1
demo_instance_name: "varspark"
demo_instance_wait: "true"
demo_instance_creation_timeout: "900"
demo_container_image: "varspark-demo"
demo_container_port: "8888"
demo_key_name: "ansible"
```

Example 3: deploying a containerised interactive workload

```
- name: Ensure {{ demo_instance_name }} instance(s) are {{ new_state }}
  os_server:
    state: "{{ new_state }}"
    cloud: "{{ demo_os_cluster }}"
    name: "{{ demo_instance_name }}"
    image: "{{ demo_instance_image  }}"
    key_name: "{{ demo_key_name }}"
    timeout: "{{ demo_instance_creation_timeout }}"
    wait: "{{ demo_instance_wait }}"
    flavor: "{{ demo_instance_flavor }}"
    security_groups: "{{ demo_instance_secgroup }}"
    network: "{{ demo_network_name }}"
  tags:
    - instances
```

ANSIBLE

docker

Example 3: deploying a containerised interactive workload

```
- name: Pull the VariantSpark image
  docker_image:
    name: "{{demo_container_image}}"
  tags:
    - docker
- name: Run the VariantSpark image
  docker_container:
    name: varspark
    image: "{{demo_container_image}}"
    state: started
    ports:
     - "{{demo_container_port}}:{{demo_container_port}}"
  tags:
    - docker
```

Summary

The benefits of SuperCloud

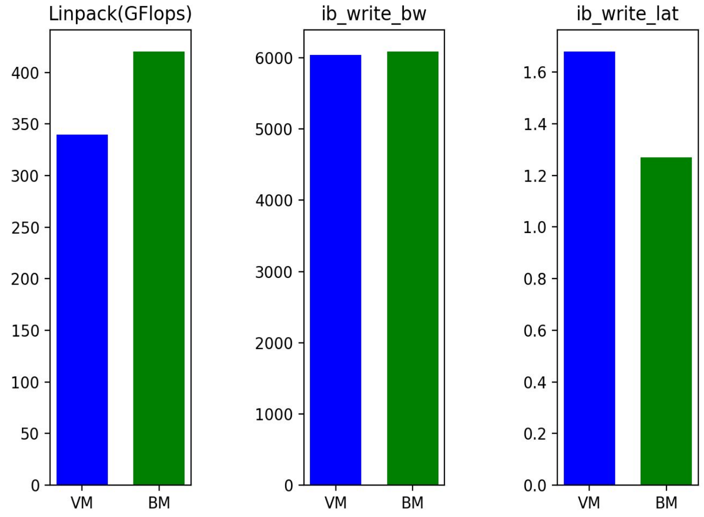
Less overheads and more flexibility: variety of workloads on a single system

```
# openstack server list
+------------------+--------+---------------------------+-------------+
| Name             | Status | Networks                  | Flavor Name |
+------------------+--------+---------------------------+-------------+
| elasticluster04  | ACTIVE | sca19=192.168.3.9         | m1.small    |
| slurm-worker001  | ACTIVE | sca19=192.168.3.14        | baremetal   |
| slurm-worker002  | ACTIVE | sca19=192.168.3.3         | baremetal   |
| slurm-worker003  | ACTIVE | sca19=192.168.3.6         | baremetal   |
| slurm-worker004  | ACTIVE | sca19=192.168.3.5         | baremetal   |
| slurm-master001  | ACTIVE | sca19=192.168.3.11        | baremetal   |
| supercloud05     | ACTIVE | internalapi=192.168.2.23  | baremetal   |
| supercloud04     | ACTIVE | internalapi=192.168.2.26  | baremetal   |
| ansible01        | ACTIVE | sca19=192.168.3.10        | m1.small    |
| varspark01       | ACTIVE | sca19=192.168.3.8         | baremetal   |
+------------------+--------+---------------------------+-------------+
```

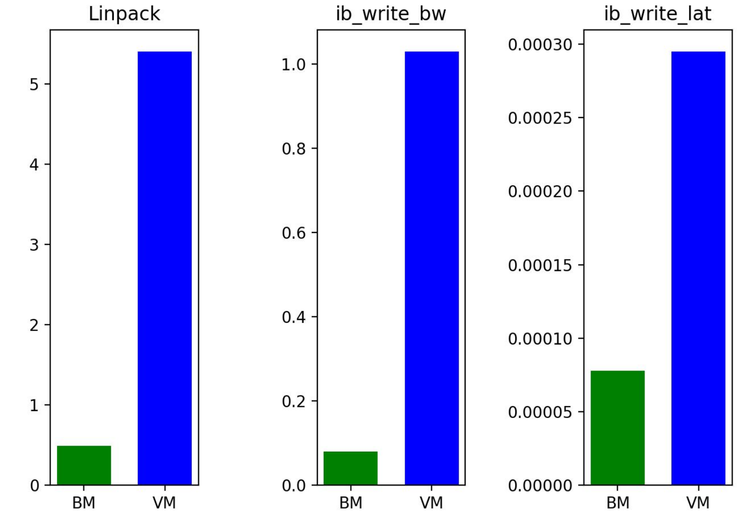
Performance benefits of making virtualisation optional

- With the appropriate tuning, **virtualisation overheads** can be reduced, but **can rarely be eliminated**. VMs show more variability

performance (Broadwell/CX3):



standard deviation:



Performance benefits of making virtualisation optional

Table 6. Testing summary

Test	virtual machine	bare-metal	efficiency(%)
Linpack	339.7	420.15	81
ib_write_bw	6043.13	6088.13	99
ib_write_lat	1.68	1.27	76

Table 7. Testing summary - standard deviation (%)

Test	bare-metal	virtual machine	ratio
Linpack	0.49	5.4	11x
ib_write_bw	0.08	1.03	13x
ib_write_lat	0.000078	0.000295	4x

The Bigger Picture

- Bare-metal capability allows running HPC workloads with **performance matching supercomputers**,
- Flexible, API-based delivery model allows **greater flexibility**, which is a strong asset for emerging fields of science such as **Precision Medicine, Machine Learning** and **Cybersecurity Research**,
- Combining bare-metal performance and API-driven provisioning **brings Infrastructure-as-Code to bare-metal**, laying foundation for **Software Defined HPC,**
- **SuperCloud = API-driven-datacentre.**

Future work

- Increasing the adoption of containerised workloads,
- Improving bare-metal instance boot times (bare-metal execution is fast, bare-metal provisioning - not so fast… yet),
- Enhancing the integration with existing HPC cluster management,
- Orchestration of scaling nova and Slurm up and down,
- Further integration work upstream.

Thank You