Building the Future on Bare Metal

How Ironic Delivers Abstraction and Automation using Open Source Infrastructure

Introduction

Within the history of computing, one persistent theme has been the layering of abstractions. Wrapping critical technologies in new interfaces makes computing easier to use, more reliable, and more secure. Direct hardware programming in turn uses the API to communicate over interfaces using low level system libraries and common driver abstractions. At the application programming layer, APIs are used to build applications, which are networked with web or RESTful APIs. The applications are hosted in virtualized servers. Virtualization is made lighter-weight and easier to scale with the introduction of containers over bare metal hypervisors. The application container itself is even abstracted away with the recent rise of functions as a service or serverless computing.

Yet, there is one fundamental piece that underlies everything in this software abstraction layer: the need for properly configured and managed hardware to host the applications. This is a universal problem in computing, no matter if you're using a device as small as a watch or a phone, running a personal computer, or hosting an entire cluster of servers delivering millions of requests, every application starts directly or indirectly with the thin provisioning of the physical hardware, known as ready to use bare metal. Despite all of the advancements in computational abstraction, bare metal management is still a fundamental problem that must be addressed. This is similar to construction domains where no matter what framework or building materials you use, the foundation must be solid, stable and secure to withstand the load, stress and strain, along with the vagaries of nature including earthquake, flood, lightening and now even pandemic.

While being a fundamental or foundational hosting problem, we can still bring the tools of abstraction and automation to bear upon solving it, making the provisioning of an entire data center or any remote site as easy as typing a single command into a terminal or pressing "submit" on a web form.

In this paper, we will explore how the Open Infrastructure community has addressed the bare metal provisioning problem with entirely free open source software. We will discuss the issues operators or enterprises face in discovering and provisioning servers, how the OpenStack community has solved these issues with the Ironic project, specific case studies of Ironic use in production and the future of open infrastructure and hardware management.

The Motivation

Lack of standardization of bare metal API leads to a proliferation of system installers to provision a host system with storage and networking on day-1. Subsequent addressing of life cycle activity for day-2 management, such as updating and upgrading of firmware, is then ad-hoc and leads to large downtime,deterring system availability for business and critical usage of the platform. Additionally, remote automation with non standard hardware or API leads to less efficient parallel deployments.

The other aspects are the labour cost and the pain points faced by data center staff to work from within the data center. It takes longer to resolve issues at untimely hours on site or to manage remotely due to non standard hardware and lack of standard tools and solutions.

> At a conference a few years ago, I sat down to dinner next to someone I did not know. He started to tell me of his job and his long hours in the data center. He asked me what I did, and I told him I worked as a software engineer in open source. And he started talking about some tooling he recently found that took tasks that would normally take nearly two weeks for racks of servers, to just a few hours. He simply glowed with happiness because his quality of life and work happiness had exploded since finding this Bare Metal as a Service tooling called Ironic.
>
> As a contributor, this is why we contribute. To make those lives better. - Julia Kreger, Ironic PTL

How a Community Came Together

Within the OpenStack community, an early desire existed to extend the Compute abstractions to physical machines. The community wanted to provide a common pattern and method of access in order to request what are ultimately "Compute" resources consisting of entire physical machines, as opposed to a portion of a physical machine that would normally be provided as a Virtual Machine. This started as the "nova-baremetal" component.

As time went on and the patterns required to securely provide physical resources differed from those of providing a virtual machine, shortcomings popped up.. Many standard operations during the lifetime of a virtual server are not as valuable or feasible for physical machines, like live migration or shelving. Eventually, in the case of a virtual machine, you simply delete the virtual machine instance when you are done with it. With a physical machine, the host has to be cleaned and reconfigured to a known base state before the physical machine can be provided to a new user.

These required workflows and the resulting conflict in use patterns ultimately drove the creation of the "Ironic" project to serve as a vendor neutral space where these common workflows could exist in code along with specific patterns required to support individual vendors' hardware and features integrated within that hardware.

Field Study

The problem of automated, unattended machine setup is not new. Many projects and companies have tried their hand at this, apparently, hard problem. It seems that the complexity comes from the many moving parts that are involved in machine provisioning. Apart from that, the desired machine configuration can differ a great deal from one to another.

A typical machine provisioning might include:

- picking a most suitable server from a pool of available servers
- applying specific configuration for hardware (like network interfaces or RAID controllers) and low-level software (like firmware or BIOS)
- loading appropriate software (operating system, drivers, applications) as well as user data into it
- applying necessary updates to minimize vulnerabilities in software
- configuring network stack, monitoring, applications, etc.

Perhaps guided by the task at hand and constrained by the underlying technology, machine provisioning projects share some similarities. They typically rely on common standards for hardware management of main motherboards or system cards, using Out Of Band (OOB) interfaces like IPMI[1] or Redfish[2]. They use the PXE suite for network booting, configuration management software (Ansible, Puppet etc) for machine customization, and newer implementations to offer a REST API as an integration point.

To name a few high-profile open source provisioning projects, Cobbler[3] and Foreman[4] initially focused on automated Red Hat Enterprise Linux machine life cycle management. Newer implementations such as Canonical's MaaS[5] and Ironic are designed to work with cloud software to represent bare metal machines to the operator in the same way as cloud instances.

In this paper, we will be focusing on how the Ironic project supports the use-cases of the Open Infrastructure operators community, covering the work flows and integrations that are deemed necessary. Note this is a new initiative to delink integrated compute-service Nova and Ironic from integrated OpenStack.

Why Bare Metal?

When we look at the original desire that caused the creation of the Ironic project, it was to support Bare Metal As A Service (BMaaS), or ultimately "Compute" resources on distinct physical machines.

Often people wonder, "Why can't a VM be used?" or, "Why do you need a whole physical server?" The reasons vary, but they tend to be one of a few different fundamental reasons.

Performance

While providing an additional abstraction layer which aims to add flexibility and increase overall efficiency, virtualizing resources reduces the overall performance available to the application. This virtualization tax can be reduced by various techniques, such as pinning cores, exploiting NUMA structures or enabling huge pages, but virtual machines can never provide the full resources of the underlying hardware to the hosted applications. Also, reducing the inferred additional latency and jitter may come at a significant additional complexity -- and hence cost -- which has to be balanced against the advantages of a virtualized environment. This is obviously a concern in general, but in High Performance Computing (HPC) environments, getting the most out of the purchased hardware is of utmost importance, and hence virtualization is often not a realistic option.

In addition to the general loss of performance, the variation of the performance over time caused by a "noisy neighbor" can also pose an issue for the hosted application. For example, debugging the performance of an application is not feasible if the jitter of the underlying system is larger than the optimisation gain.

Security

Concern of a *shared* common platform is a strong motivator for use of *distinct* physical machines. Any time an application or tenant's workload is run, be it inside a container or a virtual machine, escape attacks may exist. This may be to search for data or to pivot and attack additional systems. These vulnerabilities may not be just of the underlying platform or operating system code, but could even be something within the processor of the physical machine, like the recent Spectre/Meltdown side-channel attacks[6]. A similar argument holds true for networking and storage. Using (and hence managing) physical resources directly may be desirable when the virtual separation of software-defined solutions is not regarded as sufficient.

Compliance and Stack Independence

Service contracts or industry/government regulations may require wholly distinct physical systems for a number of reasons:

- to support different roles out of security considerations
- to ensure a baseline performance for the hosted application
- for proper cost/performance modeling
- due to license restrictions

A related compliance example is mission-critical equipment that requires full independence from any shared infrastructure, such as an underlying hypervisor. This ensures that in case of an incident, the hosted service is not blocked or impacted by the performance or required actions of another service.

Non-virtualizable Resources

Ultimately, some components of an infrastructure cannot (or should not) be virtualized. This includes the nodes for boot-strapping the infrastructure, but also the lowest layer of the infrastructure

stack, which are usually the hypervisors themselves.

For other infrastructure services, such as storage or databases, virtualization may not make much sense either: whole-node virtual machines would be required and the gains of the additional virtualization layer may not outweigh the drawbacks of the aforementioned virtualization tax or the additional complexity of a horizontal service split. Customers want acceleration resources, such as GPUs or FPGAs, because of performance or latency. But not all acceleration devices work nicely with virtualization or containerization.

Life Cycle Management

Although there can be good reasons to prefer bare metal nodes over virtualized resources, the main drawback of physical nodes is the complexity to manage their life cycle. From the moment a physical server is installed to the moment it is removed, there are several phases a server typically goes through. Standard operations which have to be performed include:

- auto-discovery and registration (with inventory systems or network databases)
- health check and burn-in (component completeness and stress testing)
- benchmarking (e.g. as part of acceptance)
- hardware configuration (e.g. RAID setups or BIOS settings)
- using and configuring Object Storage or key-value storage instead of RAID for Local or Global reliable and resilient persistence
- provisioning to the end user or services
- hardware health monitoring
- repairs and component replacements (including update of inventory databases)
- BIOS and/or UEFI firmware updates
- retirement (e.g. secure data erasure) and removal from above databases

Each of these steps comes with its own difficulties and challenges, so let’s pick the provisioning to end users/services as an example for a closer look.

A common element in layered data center operations is the need to allocate servers from one team to another in an efficient way, like when the servers are ready for production and can be handed over to the end user/service. Interfaces must be defined to file the initial request for resources, follow an approval workflow, do the actual hand-over of servers including their credentials, or accurately account for the used resources. Furthermore, the demand for resources can fluctuate over time, resulting in either poor performance utilization of ‘dedicated’ physical nodes or delayed/impacted business outcomes as workloads wait to be scheduled on the limited number of physical nodes. The reallocation of resources between different users then becomes an additional requirement, and with this the need for secure data erasure between users, the reset of any network configuration or the need to reset credentials in integrated systems, such as Baseboard Management Controllers (BMCs). The complexity of these tasks and workflows clearly varies with the environment, but automation and interface support is essential to have this done in an efficient and scalable way.

How Ironic Helps

Ironic was born as an OpenStack project to replace the original baremetal driver included in Nova (the OpenStack compute instances project). It allows operators to provision bare metal machines instead of virtual machines. Ironic is fully integrated with the rest of the stack primarily thanks to a virtualization driver for Nova that makes the CLI completely transparent to the final user. It can also integrate with other OpenStack projects, like Neutron (networking as a service), Glance (machine images management) and Swift (object store).

Ironic provides generic drivers ("interfaces") that support standards like IPMI and Redfish, used to manage any type of bare metal machine, no matter the brand. At the same time, it's officially supported by different vendors that help maintain not only the Ironic code-base, but also their own interfaces included in the Ironic code to provide full compatibility with their specific features.

Ironic is developed in Python, it is open source, and it uses gerrit[7] for code review. To ensure reliability of the code, Ironic uses the powerful Zuul[8] CI engine tool to run the basic unit and functional tests, and also to simulate bare metal machines using advanced virtualization techniques to be able to run more complex tests with different deployment scenarios, including upgrades and multinode environments.

Ironic has evolved and grown since it was "just" a way to provide bare metal machines to OpenStack users, finding ways to effectively become a standalone bare metal as a service system, capable of providing the same features as a full hardware management application.

Robust API

Ironic is API-driven and API-first, and it includes a full set of RESTful APIs that provide a common vendor agnostic interface, allowing provisioning and management of bare metal machines for their entire lifecycle, from enrollment to retirement. It takes into account possible multiple reconfigurations and reuse of the same device, where a node can be re-provisioned for different use cases over its life.

A non-comprehensive list of features provided by the Ironic API includes:

- Auto-discovery, to automatically register bare metal machines ("nodes") in Ironic
- Hardware introspection, to collect information on the node’s hardware and store it in the Ironic database
- Benchmarking and health-check on hardware components
- Cleaning, to provide a purged node before deploying an operating system on top of it
- Configuration and provisioning, including custom images installation
- End-of-life support up to node retirement

Tools

All the tools described here use Ironic in some form to provision and manage bare metal hosts.

TripleO[9]

Omni-comprehensive tool to deploy a full OpenStack environment. Ironic is used to deploy and manage the bare metal hosts that constitute the base of the cloud infrastructure and to manipulate network switches configuration, as well as the cloud bare metal hosts provider integrated in the OpenStack cloud.

Metal3[10]

Bare metal provisioning and enablement project that aims to provide a Kubernetes[11] native API to manage bare metal hosts. The tool includes a Bare Metal Operator (Metal3 SDK generated stack) as a component and proposes to include Ironic as an optional module to drive machine instances. Note Metal3 (pronounced “metal cubed”) is adding support to enable external management of bare metal hosts from the Cluster API, which again uses Kubernetes' operating model but focuses on different plug-in Cloud controllers using Custom Resource Definitions (CRDs) to support containerized Control planes such as OpenStack, Azure, Amazon Web Services, Google Cloud etc. In this use case, the first supported bare metal driver is Ironic. Largely this use of ironic has been focused on the OpenShift Container Platform installation, but the community is presently growing and evolving.

Airship[12]

A collection of tools that help provisioning and managing a cloud infrastructure starting from bare metal hosts. In Airship 2.0, Ironic is used via Metal3.io as a bare metal provisioning component integrated in the Airship platform.

Bifrost[14]

An Ansible playbooks based tool that helps deploy bare metal servers using predefined images and standalone Ironic.

Kayobe[15]

A suite that combines different tools (like Bifrost, mentioned before, and Kolla) together to be able to deploy OpenStack services in containers on top of bare metal. Kayobe configuration and workflows are all Ansible-driven, providing a consistent interface at every level.

Clients

OpenStackSDK[16]

A generic OpenStack client library used to build clients to interact with OpenStack cloud services. It has a dedicated bare metal module that is kept up-to-date with the latest features developed in Ironic.

Gophercloud[17]

This is the equivalent of the OpenStack SDK but for golang.It allows go developers to directly interact with OpenStack clouds, and supports various services including Ironic.

Ironicclient[18]

Official client, written in python. Provides an API module, ironicclient itself, that is used to build clients, such as the official openstackclient, that includes a bare metal subset of instructions to interact with Ironic, and a standalone CLI that allows interaction with Ironic without having to install the openstackclient itself.

Automation

Ansible[19]

Open-source declarative automation tool used to manage configurations and deployments. It is platform-agnostic and highly customizable with modules written in Python. Ironic provides a set of modules to be used with it.

Terraform[20]

"Infrastructure as code" open-source tool that allows users to define and provision an infrastructure using its own configuration language (HCL). It supports multiple cloud providers, including OpenStack. Metal3 has developed its own terraform provider to be used with Ironic.

Puppet Ironic[21]

Puppet[22] is yet another open-source software to automate infrastructure configuration and deployment. Based on an agent-master system, it is customizable with modules written in Ruby. Ironic has its own puppet module to help OpenStack deployments.

Usage Patterns

Cluster Installation

While more and more applications are migrating to cloud solutions, the problem of installing and managing the underlying clouds becomes even more important. Before becoming a part of a cluster, hardware has to be configured and provisioned in a consistent and efficient manner, while still allowing a certain degree of site-specific customizations.

To solve some of these problems and aid in creating clusters within the OpenStack community, multiple tools have been built upon Ironic to support specific workflow cases from bulk basic Operating System installation (Bifrost), cluster deployment of OpenStack (TripleO and Kayobe) and Kubernetes (Metal3).

Cluster Expansion

Expansion of existing clusters requires integration into existing environments. This requires that information about the new hardware as well as the physical connectivity are made available to applications and personnel performing this expansion. This activity can almost be described as an act of inspection of the hardware to enable this validation. Additional actions, provided by the “cleaning” framework in Ironic, may also be required to make sure new hardware behaves consistently with the existing hardware.

Bare Metal as a Service

While virtual machines and containers can replace hardware for many applications, there are still cases where bare metal instances have to be provided to end users of a cloud solution. Providing them in a fashion consistent with other cloud features provides a smooth user experience and allows for simpler automation. To enable that, Ironic can function as a backend for the OpenStack Compute service (Nova), while optional integration with OpenStack Networking (Neutron), OpenStack Image service (Glance) and OpenStack Block Storage (Cinder) brings the bare metal experience as close as possible to virtual machines.

Additionally, exposing bare metal machines to potentially untrusted tenants puts a strong emphasis on security aspects. The pluggable cleaning process of Ironic provides an ability to return bare metal machines to a known state between tenants, including actions such as removing information from hard drives, changing networks and/or resetting firmware settings.

Containers on Bare Metal

Containers and container orchestration frameworks, such as Kubernetes, address the needs of cloud-native applications for fast configuration changes, smooth upgrades or automatic scaling. While such clusters are usually built on top of virtual machines, this may add extra cost and complexity to the underlying cloud infrastructure or introduce an undesired performance loss due to the virtualization tax. A bare metal management API, such as the one provided by Ironic, allows for a transparent move from virtual to bare metal clusters as the underpinning infrastructure. One example of this is OpenStack Magnum as the cluster orchestration engine where the compute endpoint would either instantiate virtual or physical instances depending on the specified template. Equally, such an API can be used for the integration with other provisioning tools, such as Metal3 for Kubernetes, and hence eliminate the intermediate virtualization layer when deploying containerized applications.

Edge Equipment Management

The Edge architectures pose new challenges for hardware management. Remote connectivity through an inherently insecure medium precludes or limits the usage of many traditional bare metal provisioning and management technologies, such as PXE or IPMI, while encrypting and authentication at each stage becomes a requirement. Redfish, an open bare metal management standard developed by the Distributed Management Task Force (DMTF), allows addressing many of these challenges by providing a robust set of features over the battle-proven HTTPS protocol. Deployment using virtual media devices, recently implemented in Ironic, allows completely bypassing the initial insecure and unreliable deployment stages, using only TLS-encrypted communications.

Shared Hardware Management

In a middle ground between owning hardware management and bare metal as a service, hardware belonging to different tenants can be physically located and managed in a single data center. In this case, complete separation between tenants is required, while each tenant should still have full access to machines belonging to them. The “ownership” concept allows non-administrative users access to the bare metal API limited to only the nodes that they own.

Hardware Fault Management

As data centers scale up and the number of sites where equipment under management is deployed increases, hardware failures become the norm. Identifying failed equipment, marking it in Ironic and replacing it (while preserving the identity of the replaced component) are necessities for daily operations.

Case Studies

CERN

What is matter made of? Why is there more matter than antimatter? What happened in the first moments after the big bang? Finding answers to these and other fundamental questions about our universe is the mission of CERN[23], the European Organisation for Nuclear Research.

To achieve its mission, CERN has built and operates the largest particle physics laboratory in the world, located at the Franco-Swiss border close to Geneva. Here, the organisation provides and maintains a complex of hierarchical particle accelerators, experiment detectors and their surrounding infrastructure to enable thousands of scientists worldwide to advance our knowledge about what the universe is made of and how it works.

To analyse the data produced in these experiments requires close to 200 computing centres all over the world, joining their compute power in the Worldwide LHC Computing Grid. The CERN data centre is at the centre of this scientific infrastructure and features around 230,000 cores in more than 15,000 servers. More than 90% of the computing resources at CERN are provided by a private cloud based on OpenStack, a deployment the CERN IT department has run in production since 2013.

There were several reasons to recently complete the service offering of the cloud service by a system for bare metal provisioning:

- Simplify the procurement and provisioning workflows for physical machines
- Integrate bare metal servers into resource accounting
- Satisfy special use cases and enable new ones

Workflow Simplification: The Hardware Life Cycle at CERN

Before physical machines enter production at CERN, new hardware has to undergo verification, burn-in and benchmarking. The reasoning behind these steps include:

- Ensuring that the hardware complies with the technical specification
- Identifying broken components, such as a malfunctioning CPU
- Finding systematic errors in a delivery, such as a firmware issue
- Provoking early failures

Once the servers have made it through this process, they are allocated to their end users. This is typically done by changing ownership in one of our internal databases. It is then up to the end users to install, configure and monitor the hardware. If users don’t need their machines any longer or if the machines reach their end of life, they are given back to the procurement team (by reverting the ownership change). The machines are then cleaned, e.g. disks are wiped or IPMI passwords are reset, and either allocated to a new use case, donated or disposed. Figure 1 gives an overview of the steps during this life cycle.

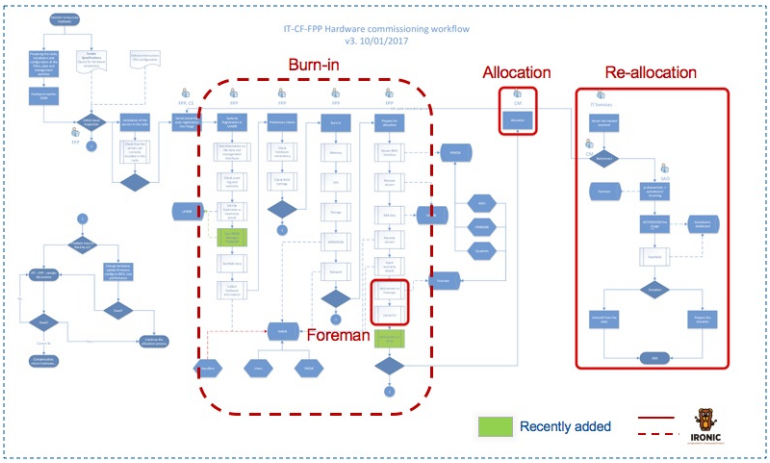


Figure 1: Procurement and resource allocation workflow.

The whole process is complex by nature, but relies in addition on various tools written in-house and also requires human intervention at various steps. The introduction of Ironic as the main tool for physical resource management is intended to reduce the complexity, the maintenance load and the need for human intervention to unblock certain steps during the process.

After initial registration into our network databases, the nodes are now enrolled in Ironic, which can do the verification of the technical specification (via inspection rules), run the burn-in and benchmarking (as manual cleaning steps), and, in conjunction with Nova, simplifies the allocation and re-allocation process. Requestors of physical machines will use a single form to request compute resources and get their physical machines via the same OpenStack instance creation process they use for virtual machines. If machines are not needed any longer, the physical instances can be destroyed, which will trigger Ironic's automatic cleaning. The machines are marked as free and are available for new use cases. The final retirement (with more extensive cleaning) also becomes a manual cleaning step. With the state machine in Ironic, its cleaning framework and its integration with Nova, a large fraction of the resource provisioning workflow is hence reduced to a few API calls.

Resource Accounting

Along with the introduction of an OpenStack based private cloud service in the CERN IT department a few years ago, the policy that all servers shall be virtual was established. Accounting questions such as 'which service is using how many resources?' became much easier to answer with OpenStack as the single pane of glass for compute resource provisioning and management.

Since physical machines were allocated outside of OpenStack, a separate source had to be maintained and consulted in order to get the full accounting details. Things became even more complicated when physical machines were assigned to a new use case, as this needs to be recorded and tracked in order to always have an accurate picture of resource allocation.

With physical instances managed in Nova and provisioned by Ironic, the number of sources for the overall accounting is reduced to one. Servers changing use cases do not require active tracking any longer, since the corresponding nodes in Ironic will have instances in known projects and can be attributed accordingly. The resource accounting is made simpler and more consistent by using Ironic.

As Ironic also supports '*adoption*', i.e. the integration of already existing servers without the need to instantiate them through Nova or Ironic, getting a full picture of the physical infrastructure and the corresponding assignments does not need to wait for a full fleet replacement.

Special and New Use Cases

Some of the services in the CERN IT department are exempt from the aforementioned '*virtual first*' policy. This includes the service providing the compute infrastructure, i.e. OpenStack compute nodes (but not the OpenStack control plane), as well as the storage services, e.g. disk or database servers, or mission critical services which are either in areas outside of the standard network connectivity or which cannot depend on other services, e.g. due to boot-strapping or security reasons.

There are, however, use cases where none of the requirements above apply, but where it is still sensible to have these services on physical machines. One example is the code calibration and performance monitoring services of the Large Hadron Collider (LHC) experiments. In order to satisfy the computing needs to analyze the many PetaBytes of data produced by the LHC experiments at CERN and to use the resources at hand in the most efficient way, it is crucial to optimize the code as much as possible. In order to detect the impact of code, operating system, or compiler changes, a stable platform which provides reproducible results is required. Since the hypervisors in our OpenStack deployment are over-committed in terms of CPU, virtual machines do not provide such a stable platform (even without overcommit it is probably desirable to reduce the number of layers and remove virtualization). Other use cases where physical servers are a sensible choice may include servers with GPUs or special hardware in our HPC clusters.

The CERN IT department's OpenStack cloud service also supports the creation of container clusters via OpenStack Magnum. While the service started with support for Swarm and Mesos, the vast majority of these clusters today use Kubernetes. The Kubernetes clusters are, however, created via OpenStack Heat on top of virtual machines. This adds an additional (not necessarily desired) layer of abstraction. With Ironic, it is now possible to create such clusters directly with physical machines, or even in a hybrid mode where only the master nodes are virtual machines and the minions are physical machines. One example for an application which makes use of this approach is the CERN IT department's batch processing service. The combination of virtual and physical machine provisioning via Nova and Ironic here allows for maximizing the efficient use of the allocated resources.

Current Service Status and Future Plans

The bare metal provisioning at CERN based on Ironic has been in production for around two years now, with more than 5,000 nodes currently enrolled. With only minor modifications, e.g. to leverage our central PXE infrastructure rather than one managed on the Ironic controllers, we use the upstream releases and have successfully upgraded Ironic multiple times.

Figure 2: The Ironic dashboard used in the CERN IT department.

While Ironic in its current configuration has already achieved most of the goals outlined above and has been established as the framework to manage the whole life cycle of physical servers in the CERN IT department, there are several areas where we expect further benefits. In addition to the adoption of existing physical nodes, the currently evolving graphical console support, the introduction of Redfish as the IPMI successor and the development of a hardware inventory system, working closely with Ironic is of special interest to resource provisioning services in the CERN IT department.

https://superuser.openstack.org/articles/Ironic-bare-metal-case-study-cern/

StackHPC[24]

Software RAID Support in Ironic

Ironic operates in a curious world. Each release of Ironic introduces even more inventive implementations of the abstractions of virtualization. However, bare metal is wrapped up in hardware-defined concrete: devices and configurations that have no equivalent in software-defined cloud. To exist, Ironic must provide pure abstractions, but to succeed it must also offer real-world circumventions.

For decades, the conventional role of an HPC system administrator has included deploying bare metal machines, sometimes at large scale. Automation becomes essential beyond trivial numbers of systems to ensure repeatability, scalability and efficiency. Thus far, that automation has evolved in domain-specific ways, loaded with simplifying assumptions that enable large-scale infrastructure to be provisioned and managed from a minimal service. Ironic is the first framework to define the provisioning of bare metal infrastructure in the paradigm of cloud.

So much for the theory: working with hardware has always been a little hairy, never as predictable or reliable as expected. Software-defined infrastructure, the method underpinning the modern mantra of agility, accelerates the interactions with hardware services by orders of magnitude. Ironic strives to deliver results in the face of unreliability (minimising the need to ask someone in the data centre to whack a machine with a large stick).

HPC Infrastructure for Seismic Analysis

As a leader in the seismic processing industry, ION Geophysical[25] maintains a hyperscale production HPC infrastructure, and operates a phased procurement model that results in several generations of hardware being active within the production environment at any time. Field failures and replacements add further divergence. Providing a consistent software environment across multiple hardware configurations can be a challenge.

ION is migrating on-premise HPC infrastructure into an OpenStack private cloud. The OpenStack infrastructure is deployed and configured using Kayobe, a project that integrates Ironic (for hardware deployment) and Kolla-Ansible (for OpenStack deployment), all within an Ansible framework. Ansible provides a consistent interface to everything, from the physical layer to the application workloads themselves.

This journey began with some older-generation HPE SL230 compute nodes and a transfer of control to OpenStack management. Each node has two HDDs. To meet the workload requirements, these are provisioned as two RAID volumes - one mirrored (for the OS) and one striped (for scratch space for the workloads).

Each node also has a hardware RAID controller, and standard practice in Ironic would be to make use of this. However, after considerable effort it was found:

- The hardware RAID controller is managed using the ssacli tool.
- The RAID controller requires a proprietary kernel driver.
- The driver was not available for the latest CentOS releases.
- The server hardware included a ‘personality board’ that had silently failed on many systems, preventing the automated reconfiguration of hardware RAID on those systems.

Taking these and other factors into account, it was decided that the hardware RAID controller was unusable for this migration. Thankfully, Ironic developed a software-based alternative.

Provisioning to Software RAID

Linux servers are often deployed with their root filesystem on a mirrored RAID-1 volume. This requirement exemplifies the inherent tensions within the Ironic project. The abstractions of virtualization demand that the guest OS is treated like a black box, but the software RAID implementation is Linux-specific. However, not supporting Linux software RAID would be a limitation for the primary use case. Without losing Ironic's generalised capability, the guest OS “black box” becomes a white box in exceptional cases such as this. Recent work led by CERN has contributed software RAID support to the Ironic Train release.

The CERN team has documented the software RAID support on their tech blog[26].

In its initial implementation, the software RAID capability is constrained. A bare metal node is assigned a persistent software RAID configuration, applied whenever a node is cleaned and used for all instance deployments. Prior work involving the StackHPC team to develop instance-driven RAID configurations[27] is not yet available for software RAID. However, the current driver implementation provides exactly the right amount of functionality for Kayobe's cloud infrastructure deployment.

The Method

RAID configuration in Ironic is described in greater detail in the Ironic Admin Guide. A higher-level overview is presented here.

Software RAID with UEFI boot is not supported until the Ussuri release, so BIOS-mode booting must be configured when deploying Train OpenStack.

A series of compute nodes, each with two physical spinning disks, were provisioned according to the CERN blog article. Two RAID devices were specified in the RAID configuration set on each node; the first for the operating system, and the second for use by Nova as scratch space for VMs.

```
{
  "logical_disks": [
    {
      "raid_level": "1",
      "size_gb"   : 100,
      "controller": "software"
    },
    {
      "raid_level": "0",
      "size_gb"   : "MAX",
      "controller": "software"
    }
  ]
}
```

The RAID configuration was then applied with the following cleaning steps:

```
[{
  "interface": "raid",
  "step": "delete_configuration"
 },
 {
  "interface": "deploy",
  "step": "erase_devices_metadata"
 },
 {
  "interface": "raid",
  "step": "create_configuration"
 }]
```

A RAID-1 device was selected for the OS so that the hypervisor would remain functional in the event of a single disk failure. RAID-0 was used for the scratch space to take advantage of the performance benefit and additional storage space offered by this configuration. It should be noted that this configuration is specific to the intended use case, and may not be optimal for all deployments.

As noted in the CERN blog article, the mdadm package was installed into the Ironic Python Agent (IPA) ramdisk for the purpose of configuring the RAID array during cleaning. mdadm was also installed into the deploy image to support the installation of the grub2 bootloader onto the physical disks for the purposes of loading the operating system from either disk should one fail. Finally, mdadm was added to the deploy image ramdisk, so that when the node booted from disk, it could pivot into the root filesystem.

Open Source, Open Development

As an open-source project, Ironic depends on a thriving user base contributing back to the project. Our experiences covered new ground: hardware not used before by the software RAID driver. Inevitably, new problems were found.

The first observation was that configuration of the RAID devices during cleaning would fail on about 25% of the nodes from a sample of 56. The nodes which failed logged the following message:

```
mdadm: super1.x cannot open /dev/sdXY: Device or resource busy
```

Where X was either a or b and Y either 1 or 2, denoting the physical disk and partition number respectively. These nodes had previously been deployed with software RAID, either by Ironic or by other means.

Inspection of the kernel logs showed that in all cases, the device marked as busy had been ejected from the array by the kernel:

```
md: kicking non-fresh sdXY from array!
```

The device which had been ejected or may not have been synchronised and appeared in `/proc/mdstat` as part of a RAID-1 array. The other drive, having been erased, was missing from the output. It was concluded that the ejected device had bypassed the cleaning steps designed to remove all previous configuration, and had later resurrected itself, thereby preventing the formation of the array during the `create_configuration` cleaning step.

For cleaning to succeed, a manual workaround of stopping this RAID-1 device and zeroing signatures in the superblocks was applied:

```
mdadm --zero-superblock /dev/sdXY
```

Removal of all pre-existing states greatly increased the reliability of software RAID device creation by Ironic. The remaining question was why some servers exhibited this issue and others did not. Further inspection showed that although many of the disks were old, there were no reported SMART failures. The disks passed self tests and although generally close, had not exceeded their mean time before failure (MTBF). No signs of failure were reported by the kernel other than the removal of a device from the array. Actively seeking errors, for example by running tools such as `badblocks` to exercise the entire disk media, showed that only a very small number of disks had issues. Benchmarking, burn-in and anomaly detection may have identified those devices sooner.

Further research may help us identify whether the disks that exhibit this behaviour are at fault in any other way. An additional line of investigation could be to increase thresholds such as retries and timeouts for the drives in the kernel.

The second issue observed occurred when the nodes booted from the RAID-1 device. These nodes, running IPA and deploy images based on Centos 7.7.1908 and kernel version 3.10.0-1062, would show degraded RAID-1 arrays, with the same message seen during failed cleaning cycles:

```
md: kicking non-fresh sdXY from array!
```

A workaround for this issue was developed by running a Kayobe custom playbook against the nodes to add `sdXY` back into the array. In all cases, the ejected device was observed to resync with the RAID device. The state of the RAID arrays is monitored using OpenStack Monasca, ingesting data from a recent release candidate of Prometheus Node Exporter containing some enhancements around MD/RAID monitoring.[28] Software RAID status can be visualised using a simple dashboard:

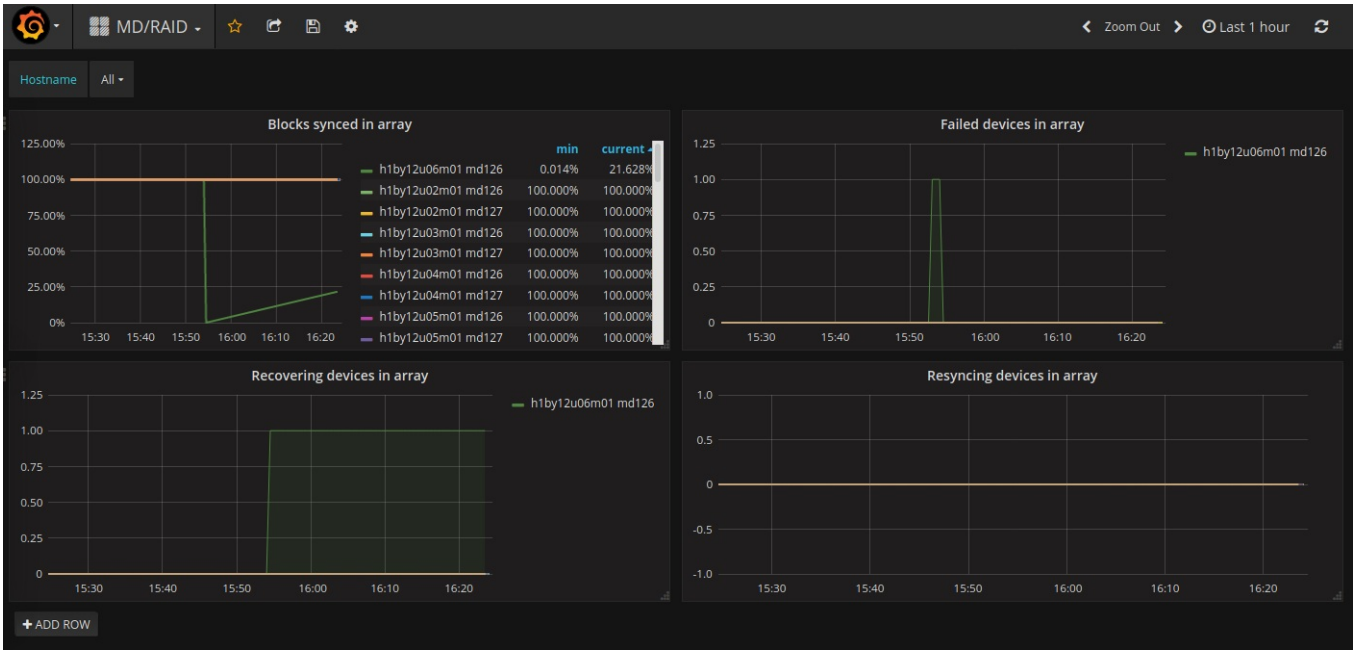


Figure: Monasca MD/RAID Grafana dashboard. The plot in the top left shows the percentage of blocks synchronised on each RAID device. A single RAID-1 array can be seen recovering after a device was forcibly failed and added back to simulate the failure and replacement of a disk. Unfortunately it is not yet possible to differentiate between the RAID-0 and RAID-1 devices on each node since Ironic does not support the name field for software RAID[29] . The names for the RAID-0 and RAID-1 arrays therefore alternate randomly between md126 and md127. Top right: The simulated failed device is visible within seconds. This is a good metric from which to generate an alert. Bottom left: The device is marked as recovering while the array rebuilds. Bottom right: No manual re-sync was initiated. The device is seen as recovering by MD/RAID and does not show up in this figure.

The root cause of these two issues is not yet identified, but they are likely to be connected and related to an interaction between these disks and the kernel MD/RAID code.

Open Source, Open Community

Software that interacts with hardware soon builds up an extensive "case law" of exceptions and workarounds. Open projects like Ironic survive and indeed thrive when users become contributors. Equivalent projects that do not draw on community contribution have ultimately fallen short.

The original contribution made by the team at CERN (and others in the OpenStack community) enabled StackHPC and ION Geophysical to deploy infrastructure for seismic processing in an optimal way. To this original work StackHPC added experiences, documentation improvements and more robust handling of RAID device creation. Even small contributions, when shared back, help to further strengthen the project.

https://superuser.openstack.org/articles/openstack-Ironic-bare-metal-program-case-study-stackhpc/

SuperCloud

Introduction

SuperCloud is a system architecture aiming to combine Supercomputing and Cloud Computing, with advantages of both and disadvantages of neither approach. It was proposed by Jacob Anders, who at the time worked for CSIRO. SuperCloud Proof-of-Concept has been presented at the OpenStack Summit in Vancouver (2018) and the Open Infrastructure Summit in Denver (2019), and provided a valuable test platform for establishing performance benchmarks and validating various Infrastructure-as-Code workflows, from bare metal HPC-on-demand to ephemeral hypervisors.

Rationale

Traditionally, Supercomputers used to be focused entirely on performance (and hence running directly on the hardware) at the expense of the flexibility they offered. In a supercomputing environment, users can expect excellent performance but rarely have the luxury of bringing their own operating system image or requesting their workload to run in an isolated network environment. Cloud Computing systems, on the other hand, offer users nearly unlimited flexibility, but this is most commonly achieved by the use of virtualization, which often carries significant performance penalty, making these systems less suitable for hosting High Performance Computing (HPC) workloads.

Implementation

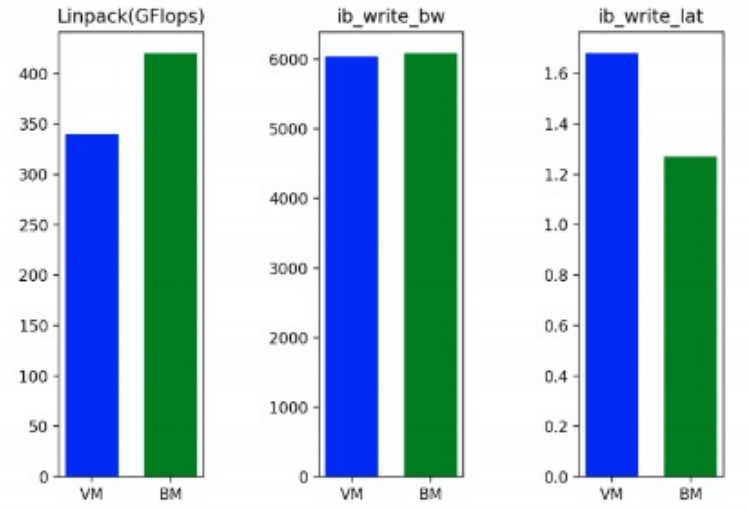
To address these challenges, SuperCloud builds upon Ironic bare metal service and combines it with InfiniBand Software Defined Networking. The system can provision cloud instances directly on the hardware, with no need of virtualization, achieving the level of performance previously only seen on classic HPC systems. It is capable of doing so while enabling the users to run any operating system and workload required, provisioning resources in software defined networks which can be isolated, private, shared or publicly accessible just like they would in a classic virtualized cloud environment.

SuperCloud is a foundation for the Infrastructure-as-Code tools to build upon. This has the potential of enabling the users to programmatically request bare metal compute, networking, storage and software resources through a single, unified set of APIs. This is a key capability which has the potential to consolidate today's isolated islands of Software Defined Networking (SDN), software defined storage (SDS) and software-as-a-service (SaaS) and enable IT teams to move into a new paradigm of building systems - Software Defined Computing. In this new paradigm, HPC cluster

management software, job schedulers and high-performance parallel file systems move up the stack and become cloud native applications which can run side-by-side with hypervisors running VMs and container nodes, all contained within one standard operating environment. At the same time, HPC compute and HPC storage become much more heterogeneous, flexible and dynamic - and can not only consistently deliver top performance, but also quickly adapt to ever-changing needs of the scientific community, addressing the challenges posed by the lack of flexibility.

Performance

In the SuperCloud environment and configuration, preliminary benchmarks showed promising results - CPU performance while running Linpack was approximately 19% higher in bare metal compared to virtual machines. While interconnect bandwidth was only marginally higher on bare metal (approximately 1%) due to the use of SRIOV technology in the virtualized environment, interconnect latency was far superior (approximately 23% lower) in bare metal compared to virtual machines. A summary of these tests is included in the diagram below.



Sample Workloads

To demonstrate the capabilities of bare metal cloud, consider the following example use cases:

Infrastructure-as-Code on Bare Metal

It wouldn't be far from the truth to say that Cloud Computing was born out of the need for automation. Application developers required a method of programmatically requesting compute, networking and storage resources through a set of API endpoints so that they can easily deploy and scale their applications with no involvement of the infrastructure team. Virtualization largely helps with this task by creating a convenient abstraction layer, however, until recently, the situation was not equally simple for bare metal deployments. While many fully or partially automated methods of hardware deployment existed, most were limited and/or often platform specific, and were not able to match the feature sets of cloud environments.

The addition of Ironic bare metal provisioning to OpenStack dramatically changes this landscape. Now, standard Infrastructure-as-Code tools developed for virtualized clouds can be used to provision bare metal. Being able to reuse existing tools and abstractions is very powerful, as proven by the examples below.

ElastiCluster - Bare Metal HPC on Demand

Elasticluster (https://elasticluster.readthedocs.io/en/latest/) is a software package developed at the University of Zurich, which can deploy on-demand, software-defined High Performance Computing clusters to EC2, GCE and OpenStack clouds. While ElastiCluster was developed with virtualized instances in mind, an Infrastructure-as-Code approach and related abstractions make it very easy to adapt it to running on bare metal. The only modifications required are specifying a "baremetal" flavor for the cluster nodes and adjusting the timeout values to cater for longer boot times of bare metal systems.

With small modifications, OpenStack-based HPC on demand can be used with significantly faster bare metal compute, without sacrificing network multi-tenancy. This is a great example of how Ironic allows reuse of existing code in a new, more performant context.

Ephemeral Hypervisors

By design, a SuperCloud system does not have any primary virtualization capability; it is bare metal only. However, the ability to run bare metal compute instances means that a secondary virtualization capability can be added, if desired. Hypervisors can be provisioned dynamically, by creating bare metal instances connected to the internal API network and configuring nova-compute service appropriately. The major benefit of combining Infrastructure-as-Code capability with bare metal provisioning is the ability to build and maintain one pool of hardware which can be quickly and easily reconfigured to fulfil a variety of very different roles. While a user might want to run a number of high performance servers in an ElastiCluster system, another set of nodes can be running elastic hypervisors or container nodes. This has the potential to improve resource utilisation and reduce operational overheads that are typically observed when different workloads require different types of services to run on them.

Red Hat[30]

As covered in the OpenStack Baremetal Logo Program case study SuperUser article[31] , Redhat's use of Ironic largely came from the need to help support customers' needs of automating the installation of its OpenStack Platform[32] product, which was also logical because of the use of the TripleO[33] project. This use has continued to grow with the Metal3[34] project to help facilitate the automated installation of the RedHat's OpenShift[35] product.

But Ironic does not just make it easier for our installer tools to provision the bare metal hardware needed for clusters being deployed; it also provides an API and mechanisms to support a variety of use cases from within a running cloud, ultimately allowing cloud users to gain access to the dedicated resources they need in an API driven, repeatable and reliable way.

Airship[36]

The goals of Airship are to enable operators to predictably deliver raw infrastructure as a resilient cloud and to efficiently manage the life cycle of the resulting platform, following cloud-native principles such as real-time upgrades with no downtime to services. To achieve this, Airship integrates best-in-breed open source tooling, presenting an easy-to-use, flexible and declarative interface to infrastructure management.

A fundamental piece of this puzzle is the provisioning and management of bare metal servers. Airship initially used a declarative wrapper around a traditional package-based bare metal provisioner (MaaS). However, this did not provide the desired immutability and predictability of image-based deployments. To address this, Airship 2.0 integrates the Metal3 project. Metal3 presents a declarative model for bare metal, and drives Ironic (in standalone mode) to efficiently realize provisioning. To further model Kubernetes clusters declaratively, Airship uses the Kubernetes Cluster API (CAPI). CAPI broadens Airship's goal to be flexible, general-purpose tooling by providing implementations that stand up Kubernetes clusters across the range of public cloud providers, OpenStack clusters and bare metal provisioning.

From an Airship perspective, the net-net is that CAPI allows it to manage infrastructure and workloads consistently across these different environments. This opens up use cases such as sharing Containerized Network Function (CNF) workloads across private bare metal clusters and elastic public clouds, as well as many others that were not previously possible. The Airship and Metal3 communities have worked closely to ensure that Metal3 integrates as seamlessly as the bare metal provider for the Kubernetes Cluster API.

Airship's need for bare metal capabilities was driven by many of the benefits mentioned elsewhere in this whitepaper – in particular, the need to squeeze every last bit of performance out of physical assets, as well as the ability to physically locate the infrastructure close to end users. These are critical ingredients for a successful, low-latency 5G network, which was the initial key use case for Airship. In addition, as detailed in the Edge Usage Pattern below, the ability to drive secure, remote provisioning over the WAN led Airship 2.0 to adopt a Redfish-based bootstrap procedure.

Finally, infrastructure is nothing without a workload to utilize it. Airship provides a declarative YAML interface and CLI to manage the lifecycle of any Helm-based or raw Kubernetes manifest-based workloads, unified with its management of servers, Kubernetes nodes and network configuration. It provides the Treasuremap project, which has reusable configuration for common workloads such as OpenStack, Logging and Monitoring and Databases. Airship 2.0 is reframing Treasuremap into a library of composable intent for operators to rapidly consume and customize to meet their unique needs.

And More!

Many more companies use Ironic in production as recorded in various other Superuser articles.

Platform9[39]

Platform 9 provides a software-as-a-service-based service to deploy and operate OpenStack hybrid clouds for KVM, VMware and public cloud environments.

https://superuser.openstack.org/articles/Ironic-bare-metal-case-study-platform9/

ChinaMobile[40]

ChinaMobile is a leading telecommunications provider in mainland China.

https://superuser.openstack.org/articles/openstack-Ironic-bare-metal-program-case-study-china-mobile/

VEXXHOST[41]

VEXXHOST provides infrastructure-as-a-service OpenStack public cloud, private cloud, and hybrid cloud solutions to customers, from small businesses to enterprises across the world.

https://superuser.openstack.org/articles/openstack-Ironic-bare-metal-program-case-study-vexxhost/

The Future

Open Infrastructure

When we look at the foundations of infrastructure, common patterns exist. These patterns exist in large part due to standards and ways of operation that became the standard or de-facto standard. In a sense, it comes down to form following function, just like function following form. One could almost look at it as a yin-yang relationship or each conductor of a twisted-pair cable. It is hard to believe that twisted-pair cable, invented in 1881[42] , is still used to this day because it provides a wonderful foundation.

And so, it is not impossible to imagine a future where these patterns continue to exist as there is no one singular vendor or supplier, nor can there be in our society. And realistically, the only path technology can truly take is for further continuous innovation and democratization. This is similar to how the quality of twisted-pair cable has improved over the many years it has existed to meet the needs of what was built on top of the foundation it provides.

With no singular vendor or solution, and with the very nature of humanity to have uniqueness, the necessity of open is realized. Open provides the common ground, the level playing field. The context of common need to work together is what binds us and stresses the need for open-ness. Ultimately, regardless of if we build composable systems, ships to the moon or start missions to populate Mars with cats, the need to work together with an Open Infrastructure remains a necessity for success.

In the Short Term

While we may all wish to populate the planet Mars with cats, fundamental layers of access and management are needed to serve as the structure and mechanisms. Standards like Redfish will only bring consistency across the vendors and consumers that wish to engage in that effort through utilizing the realization that the common means allow the equipment to be leveraged faster and more efficiently. That does not discount others' efforts that are contrary to standards, because they are also engines of innovation.

As we look forward to the management of bare metal, the common feature set as driven by the market is what brings mutual value to everyone. This is where tools like Ironic bring tremendous value, seeking to enable users and operators to leverage the common feature set through an open common mechanism.

Like any effort, this does take substantial dedication and commitment to the contributors of Ironic. The common use case, the common means and the common need will continue to drive us forward.

Thanks

A special thanks to those that contributed to this document in both words, commentary and questions. Without everyone's effort, this paper would not have not been possible.

Contributors

This document contains the words of many individuals, both known and unknown, as this document was available from within the community for anonymous editing.

한승진, SK Telecom
Jacob Anders, CSIRO
Alex Bailey, AT&T
Pete Birley, AT&T
Manuel Buil, SUSE
Ella Cathey, Cathey.Co
Chris Dearborn, Dell Technologies
Fatih Degirmenci, Ericsson
Ilya Etingof, Red Hat
Chris Hoge, OpenStack Foundation (OSF)
Chris Jones, Red Hat
Arkady Kanevsky, Dell Technologies
Julia Kreger, Red Hat
Simon Leinen, SWITCH
Matt McEuen, AT&T
Noor Muhammad Malik, Xflow Research
Quang Huy Nguyen, Fujitsu
Richard Pioso, Dell Technologies
Riccardo Pittau, Red Hat
Prakash Ramchandran, Dell Technologies
Doug Szumski, StackHPC
Dmitry Tantsur, Red Hat
Stig Telfer, StackHPC
Kaifeng Wang, ZTE
Arne Wiebalck, CERN
Wes Wilson, OpenStack Foundation (OSF)

References

1. https://www.intel.com/content/www/us/en/products/docs/servers/ipmi/ipmi-home.html
2. https://www.dmtf.org/standards/redfish
3. https://cobbler.github.io
4. https://theforeman.org
5. https://maas.io/
6. https://meltdownattack.com/
7. https://www.gerritcodereview.com/
8. https://zuul-ci.org/
9. https://docs.openstack.org/tripleo-docs/latest/
10. https://metal3.io/
11. https://kubernetes.io/
12. https://www.airshipit.org/
13. https://thenewstack.io/metal3-uses-openstacks-ironic-for-declarative-bare-metal-kubernetes/
14. https://thenewstack.io/metal3-uses-openstacks-ironic-for-declarative-bare-metal-kubernetes/
15. https://docs.openstack.org/bifrost/latest/
16. https://docs.openstack.org/kayobe/latest/index.html
17. https://docs.openstack.org/openstacksdk/latest/
18. http://gophercloud.io/
19. https://docs.openstack.org/python-ironicclient/latest/
20. https://www.ansible.com/
21. https://www.terraform.io/
22. https://github.com/openstack/puppet-ironic
23. https://puppet.com/
24. http://home.cern
25. https://www.stackhpc.com/
26. https://www.iongeo.com/
27. http://techblog.web.cern.ch/techblog/post/ironic_software_raid/
28. https://www.stackhpc.com/bespoke-bare-metal.html
29. https://github.com/prometheus/node_exporter/tree/v1.0.0-rc.1
30. https://docs.openstack.org/ironic/train/admin/raid.html#optional-properties
31. https://www.redhat.com/en
32. https://superuser.openstack.org/articles/openstack-ironic-bare-metal-program-case-study-red-hat/
33. https://www.redhat.com/en/technologies/linux-platforms/openstack-platform
34. http://tripleo.org/
35. https://metal3.io/
36. https://www.openshift.com/
37. https://www.airshipit.org/
38. https://www.verizonmedia.com/
39. https://superuser.openstack.org/articles/how-verizon-media-rocks-bare-metal/
40. https://platform9.com/

41. https://www.chinamobileltd.com/en/global/home.php
42. https://vexxhost.com/
43. Patent US244426A - A. G. Bell - July 1881 - https://patents.google.com/patent/US244426A/en
44. Distributed Management Task Force - Redfish - https://www.dmtf.org/standards/redfish