openstack
CLOUD SOFTWARE

2015

Exploring Opportunities: Containers and OpenStack

OPENSTACK WHITE PAPER

Contributors:

Kathy Cacciatore, *Consulting Marketing Manager,* OpenStack Foundation
Paul Czarkowski, *Cloud Engineer,* Blue Box, An IBM Company
Steven Dake, *Kolla Project Technical Lead (PTL), Principal Engineer - OpenStack,* Cisco Systems, Inc.
John Garbutt, *Nova PTL, Principal Engineer,* Rackspace
Boyd Hemphill, *Technology Evangelist,* StackEngine
John Jainschigg, *Technology Solutions Marketing,* Mirantis Inc.
Andre Moruga, *Director of Program Management/Server Virtualization,* Odin
Adrian Otto, *Magnum PTL, Distinguished Architect,* Rackspace
Craig Peters, *Director of Product Management,* Mirantis Inc.
Brian E. Whitaker, *Founder,* Zettabyte Content LLC

Executive Summary

> *"The important thing for us as a community is to think about OpenStack as an integration engine that's agnostic," Collier said. "That puts users in the best position for success. Just like we didn't reinvent the wheel when it comes to compute, storage and networking, we'll do the same with containers."*
> **- Mark Collier,** *COO,* OpenStack Foundation

Containers are certainly a hot topic. The OpenStack® User Survey indicates over half of the respondents are interested in containers in conjunction with their OpenStack clouds for production uses. Thanks to new open source initiatives, primarily Docker, containers have gained significant popularity lately among Developer and Ops communities alike.

The Linux® kernel has supported containers for several years, and now even Microsoft® Windows® is following suit. However, container use in the enterprise remains an emerging opportunity since standards are still being formed, the toolset ecosystem around containers is relatively new, and ROI is uncertain.

Containers are an evolving technology and OpenStack is evolving to support them, just as it has supported other emerging technologies in the past. Rather than create new vertical silos to manage containers in their data centers, IT organizations find value in OpenStack providing a cross-platform API to manage virtual machines, containers and bare metal.

Trevor Pott, writing for The Register, provides perspective.

> *"OpenStack is not a cloud. It is not a project or a product. It is not a virtualization system or an API or a user interface or a set of standards. OpenStack is all of these things and more: it is a framework for doing IT infrastructure – all IT infrastructure – in as interchangeable and interoperable a way as we are ever likely to know how."*[1]

Container support is just another example of the basic value proposition for OpenStack - that by utilizing OpenStack as the foundation of a cloud strategy, you can add in new, even experimental technologies, and then deploy them to production when the time is right, all with one underlying cloud infrastructure - without compromising multi-tenant security and isolation, management and monitoring, storage and networking and more.

In order to support accelerating interest in containers and highlight opportunities, this paper offers readers a comprehensive understanding of containers and container management in the context of OpenStack. This paper will describe how various services related to containers are being developed as first-class resources in current and upcoming releases of OpenStack.

[1] http://www.theregister.co.uk/2015/07/09/openstack_overview/

What are containers?

Containers are isolated, portable environments where you can run applications along with all the libraries and dependencies they need. Containers aren't virtual machines. In some ways they are similar, but there are even more ways that they are different. Like virtual machines, containers share system resources for access to compute, networking, and storage. They are different because all containers on the same host share the same OS kernel, and keep applications, runtimes, and various other services separated from each other using kernel features known as namespaces and cgroups. Docker added the concept of a container image, which allows containers to be used on any host with a modern Linux kernel. Soon Windows applications will enjoy the same portability among Windows hosts as well. The container image allows for much more rapid deployment of applications than if they were packaged in a virtual machine image.

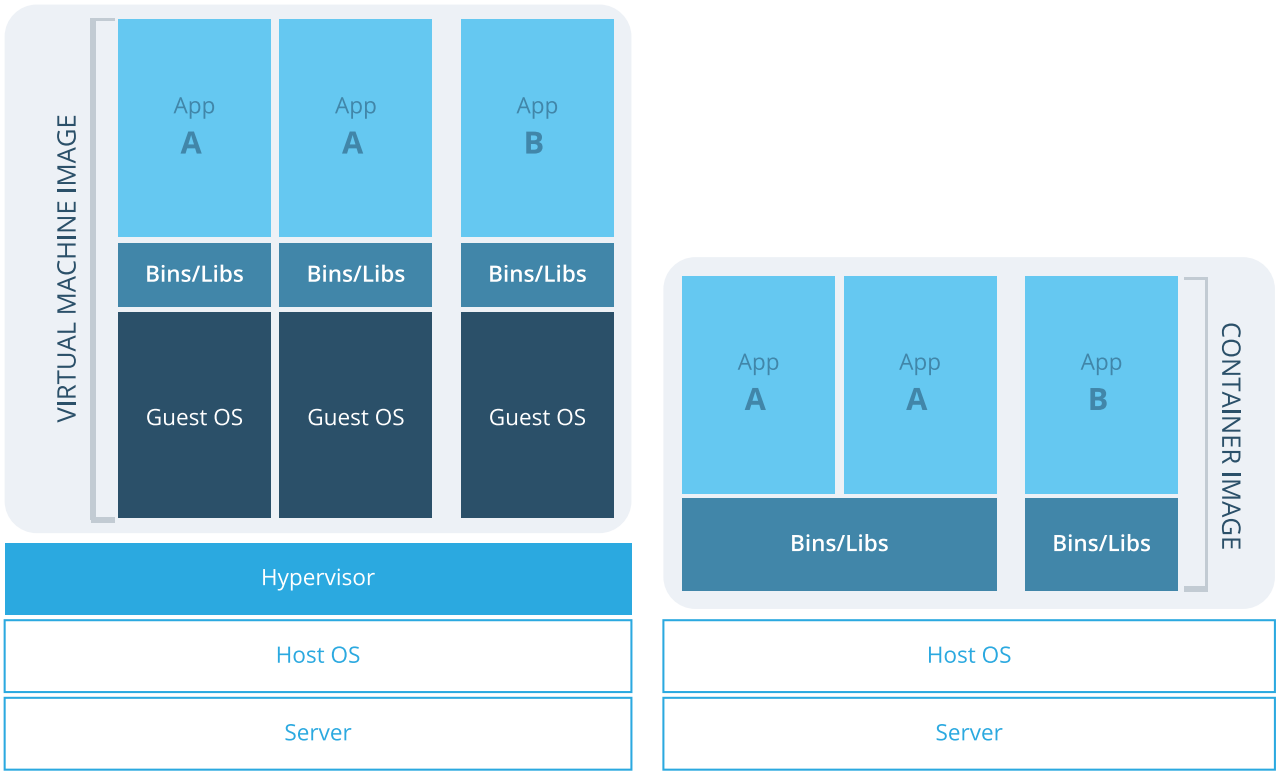

Figure 1: Containers vs. VMs

Today, containers are being used for two major purposes.

An entire system – operating system, applications, services, etc. – can exist inside a container. These are called, appropriately enough, **system or OS containers**. System containers' use cases are effectively similar to those of virtual machines.

However, **application containers** are different. Containers are a way of bundling and running applications in a more portable way. They can be used to break down and isolate parts of applications, called microservices, which allow for more granular scaling, simplified management, superior security configurations, and solving a class of problems previously addressed with configuration management (CM) tools. They are not a replacement for virtualization or CM.

A developer can put an application or service inside a container, along with the runtime requisites and services the application requires, without having to include a full operating system. This allows container images to be small, usually just a few megabytes in size compared to virtual machine images which can be orders of magnitude larger.

Containers have been around for years, but they didn't become popular until various vendors started defining *container images*. Conceptually a container image can be thought of as a snapshot of a container's filesystem that can be stored on disk. The container filesystem is arranged in layers, like how a series of commits are arranged in a git repository. This allows the container image to indicate which parent image it is derived from, allowing it to be very small by comparison. All it needs are the bits that are different from its parent. This is why they can be so much smaller. Container images allow tools like Docker to simplify container creation and deployment, using a single command to launch the app with all its requisites. The concept of the container image, and the layering features associated with that concept, was really the missing piece needed to bring containers to the mainstream.

Administrators and developers are interested in containers for two major reasons.

1. Application containers, compared with virtual machines, are very lightweight – minimizing compute, storage, and bandwidth requirements. Since multiple containers leverage the same kernel (Linux today, with Windows soon), containers can be smaller and may require less processing, RAM, and storage than virtual machines because they can be used without any hardware virtualization. They allow more dynamic systems than virtual machines allow, because the chunks of data that need to be moved around to use containers are so much smaller than virtual machine images.

2. The other advantage is that containers are *portable*, effectively running on any hardware that runs the relevant operating system. That means developers can run a container on a workstation, create an app in that container, save it in a container image, and then deploy the app on any virtual or physical server running the same operating system - and expect the application to work.

Containers offer deployment speed advantages over virtual machines because they're smaller megabytes instead of gigabytes. Typical application containers can be started in seconds, whereas virtual machines often take minutes. Containers also allow direct access to device drivers through the kernel, which makes I/O operations faster than with a hypervisor approach where those operations must be virtualized. Even in environments with hundreds or thousands of containers, this speed advantage can be significant and contributes to overall responsiveness new workloads can be brought online quickly and make boot storms become a thing of the past.

Containers create a proliferation of compute units, and without robust monitoring, management, and orchestration, IT administrators will be coping with "container sprawl", where containers are left running, mislocated or forgotten. As a result, some third-party ecosystem tools have become so synonymous with containers that they need to be mentioned, in the context of OpenStack.

The three most common are **Docker Swarm**, **Kubernetes**, and **Mesos**.

Docker[2] popularized the idea of the container image. They provide a straightforward way for developers to package an application and its dependencies in a container image that can run on any modern Linux, and soon Windows, server. Docker also has additional tools for container deployments, including Docker Machine, Docker Compose, and Docker Swarm. At the highest level, Machine makes it easy to spin up Docker hosts, Compose makes it easier to deploy complex distributed apps on Docker, and Swarm enables native clustering for Docker.

Kubernetes[3] (originally by Google, now contributes to the Cloud Native Computing Foundation[4]) is an open source orchestration system for Docker containers. It handles scheduling onto nodes in a compute cluster and actively manages workloads to ensure that their state matches the user's' declared intentions.

Apache Mesos[5] can be used to deploy and manage application containers in large-scale clustered environments. It allows developers to conceptualize their applications as jobs and tasks. Mesos, in combination with a job system like Marathon, takes care of scheduling and running jobs and tasks.

OpenStack refers to these three options as Container Orchestration Engines (COE). All three of these COE systems are supported in OpenStack Magnum, the containers service for OpenStack, that allows your choice of COE to be automatically provisioned in a collection of compute instances where your containers are run.

[2] https://opensource.com/resources/whatdocker
[3] http://kubernetes.io/
[4] http://www.linuxfoundation.org/newsmedia/announcements/2015/07/newcloudnativecomputingfoundationdrivealignmentamong
[5] http://opensource.com/business/14/9/opensourcedatacentercomputingapachemesos

It's worth mentioning that the container ecosystem, even for companies like Docker, remains a work in progress. For example, a fundamental standard for container images is under development. In June 2015, 21 companies formed the Open Container Initiative[6] to address this issue. Docker is donating its container format and runtime, runC, to the OCI to serve as the cornerstone of this new effort . As container technology matures, a fundamental goal for ongoing OpenStack development is to ensure that tools like Docker, Kubernetes and Mesos work well within OpenStack. OpenStack, as a fundamental framework for IT infrastructure, remains hardware and software agnostic so it can manage everything.

[6] https://www.opencontainers.org/

Value of Containers Within an OpenStack Infrastructure

Cloud infrastructure provides a complete data center management solution in which containers, or hypervisors for that matter, are only part of a much bigger system. OpenStack includes multi-tenant security and isolation, management and monitoring, storage and networking and more. These services are needed for any cloud / data center management regardless of whether containers, virtual machines or bare metal servers are being used.

Containers are a complement to existing technology that bring a new set of benefits. OpenStack enables organizations to select and use the right tool for the job. OpenStack supports containers on bare metal or virtual machines. Operators must be aware that containers don't have the same security isolation capabilities as virtual machines, which means that containers can not be viewed as a direct substitute for virtual machines. As an example, service providers often run containers in VMs in order to provide robust protection of one tenant's processes from poorly behaved or malicious code in other containers[7]. Another approach is to use a bay in OpenStack Magnum to arrange a group of virtual machines or bare metal (Ironic) instances that are only used by one tenant to address this risk. OpenStack supports all of these configurations in the role of the overall data center manager - virtual machines deliver compute resources and containers aid application deployment and management. For many organizations, containers on OpenStack enable additional flexibility and deployment agility without having to spin up a separate container-only infrastructure.

Organizations could use containers for the following reasons:

- Containers provide deterministic software packaging and fit nicely with an immutable infrastructure model.
- Containers are excellent for encapsulation of microservices.
- For portability of containers on top of OpenStack virtual machines as well as bare metal servers (Ironic) using a single, lightweight image.

One of the benefits of using an orchestration framework with containers, is that it can allow switching between OpenStack or bare metal environments at any given point in time, abstracting the application away from the infrastructure. In this way, either option can be selected by pointing the orchestration engine to the target environment of choice. OpenStack Orchestration Service (Heat) provides support for Docker orchestration starting from the Icehouse release. With Google's recent sponsorship of the OpenStack Foundation[8] and developer contributions, the Kubernetes orchestration engine is integrated with OpenStack as well. In fact, with OpenStack Magnum containers-as-a-service, the default bay type is a Kubernetes bay.

[7] https://cloud.google.com/compute/docs/containers/container_vms
[8] http://www.openstack.org/blog/2015/07/google-bringing-container-expertise-to-openstack/

What are the use cases in OpenStack?

Since complex mission critical software systems have about a five-year maturation cycle, it's still early to determine all the opportunities for containers to provide benefit. Having said that, many organizations are using containers because containers are a continuation of the "doing more with less" trend that began with virtualization. Organizations of all sizes that use containers agree on several use cases where benefits are specific and measurable.

According to a survey by ZDNet[9], respondents perceived benefits from the ability to deploy applications faster (54 percent); reduced effort to deploy applications (40 percent); streamlined development and testing (38 percent); reduced application deployment costs (31 percent); and server consolidation (25 percent).

Development is a very clearcut opportunity for container technology. Essentially, a developer can create an application container, containing the app, runtimes, libraries, etc., and move it to any machine - physical or virtual. Since containers are truly stateless, developers don't have to worry about compatibility and containers can be used as easily provisioned, immediately disposable development environments on any kind of IT infrastructure. This speeds up ramp time for new developers as well as increasing overall development productivity.

In build/continuous integration environments, containers enable organizations to rapidly test more system permutations as well as deliver increased parallelism, increasing innovation and feature velocity.

For quality assurance, containers enable better black box testing as well as help organizations shift from governance to compliance.

Because containers can be stateless, they also contribute to the shift toward immutable infrastructure. Thousands of containers can be created using a single consistent container image. Changes to the image can immediately be layered upon all the container instances. Old container images can be discarded as needed.

Stateless containers also facilitate high availability. Containers can be run on different underlying hardware, so if one host goes down, administrators can route traffic to live application containers running elsewhere.

[9] http://www.zdnet.com/article/customers-reporting-interest-in-cloud-containers-linux-and-openstack-for-2015/

There are noticeable cost advantages as well. Essentially, administrators can create and destroy container resources in their data center without worrying about costs. With typical data center utilization at 30%, it is easy to bump up that number by deploying additional containers on the same hardware. Also, because containers are small and launch quickly, clusters can scale up and down in more affordable and granular ways, simply by running or stopping additional containers.

To that point, containers also enable density improvements. Instead of running a dozen or two dozen virtual machines per server, it's possible to run hundreds of application containers per server. There are a few implications to this possibility. One is that enterprises might be able to make use of older, or lower performing hardware – thereby reducing costs. Another implication is that an enterprise might be able to use fewer servers, or smaller cloud instances, in order to accomplish their objectives.

There are a number of sophisticated users who have started to use containers at scale, in production. Rackspace is using OpenStack to provision containers at scale in production products, including Rackspace Private Cloud, Rackspace Public Cloud, and Rackspace Cloud Databases. Pantheon, a website management platform serving over 100,000 Drupal and WordPress sites, is powered by 1,000,000+ containers on virtual machines and bare metal, provisioned in exactly the same way with their OpenStack-based CLI and RESTful API.

Containers with OpenStack Today

OpenStack is arguably the leading cloud framework for adopting and adapting new technologies. The community makes decisions on technology relevance and creates new projects to support widespread adoption among OpenStack deployments. In 2014, the OpenStack community decided that containers were an important technology to support and that decision has resulted in several projects to ensure containers - and the third-party ecosystem around containers - will be supported in OpenStack clouds.

OpenStack is undergoing a continuous evolution toward full-fledged container support. Today it supports LXC and Virtuozzo system containers. Docker application containers and Docker Swarm, Kubernetes and Mesos container orchestration are available with the Liberty release of Magnum[10].

Building a Container Hosting Environment with OpenStack Compute

OpenStack Compute (Nova) manages the compute resources for an OpenStack cloud. Those resources may be virtual machines (VMs) from hypervisors such as KVM, Xen, VMware® vSphere® and Hyper-V® or from container technology like LXC and OpenVZ (Virtuozzo)[11]. The latter system container technologies are supported by Nova via libvirt; libvirt is an open source API, daemon and management tool for managing platform virtualization available on most Linux distributions.

These are suitable for use cases with the requirement to treat a container like a lightweight virtual machine, allowing use in a similar way to on-demand virtual machines. The created system container environment would typically be a network-connected server, with one or more applications, an IP address and some form of remote access (ssh). Containers are usually created from templates or images that determine the structure and contents of the container. System containers are useful to run a fleet of identical or different flavors of distros with little overhead. Example uses are a service provider that needs maximum density to stay profitable, or a user that wants to get bare metal performance and still needs to use some form of virtualization for manageability.

Rackspace Private Cloud is using LXC containers in production for all Infrastructure components of an OpenStack powered cloud, supporting Icehouse, Juno, and Kilo as production ready releases. Instead of using a microservice model, as would be defined by the larger Docker community, Rackspace has made the decision to use containers like disposable bare metal. They have found that containers are amazing at providing scale within an OpenStack cloud and allows deployers to better utilize resources.

[10] http://lists.openstack.org/pipermail/openstackdev/2015March/058714.html
[11] http://docs.openstack.org/developer/nova/supportmatrix.html

Containers with OpenStack Tomorrow

The OpenStack community formed a Containers team in May 2014, with a clear vision for advancing container technology in OpenStack, and providing new services and tools on track with the latest technological advances. The objective is to allow users to create and manage containers with an experience consistent with what they expect from using the Nova service to get virtual machines. The aim is to offer developers a single set of compatible APIs to manage their workloads, whether those run on containers, virtual machines or bare metal. Overall, OpenStack aims at providing the infrastructure framework for the next ten to twenty years. As new technologies like containers emerge and become relevant, the community will work on supporting them, taking a consistent and open approach.

The top three areas of focus are:

- Provide comprehensive support for running containerized workloads on OpenStack.
- Simplify the setup needed to run a production multi-tenant container service.
- Offer modular choice to OpenStack cloud operators who have not yet established a definitive containers strategy.

There are multiple OpenStack projects leveraging container technology to make OpenStack better: Magnum, Kolla and Murano. Basically:

- Magnum is designed to offer container specific APIs for multi-tenant containers-as-a-service with OpenStack. Figure 2 shows how Magnum integrates with other OpenStack components.
- Kolla is designed to offer a dynamic OpenStack control plane where each OpenStack service runs in a Docker container.
- Murano is an application catalog solution that allows for packaged applications to be deployed on OpenStack, including single-tenant installations of Kubernetes.

With the Liberty release, Magnum and Murano will be production-ready.

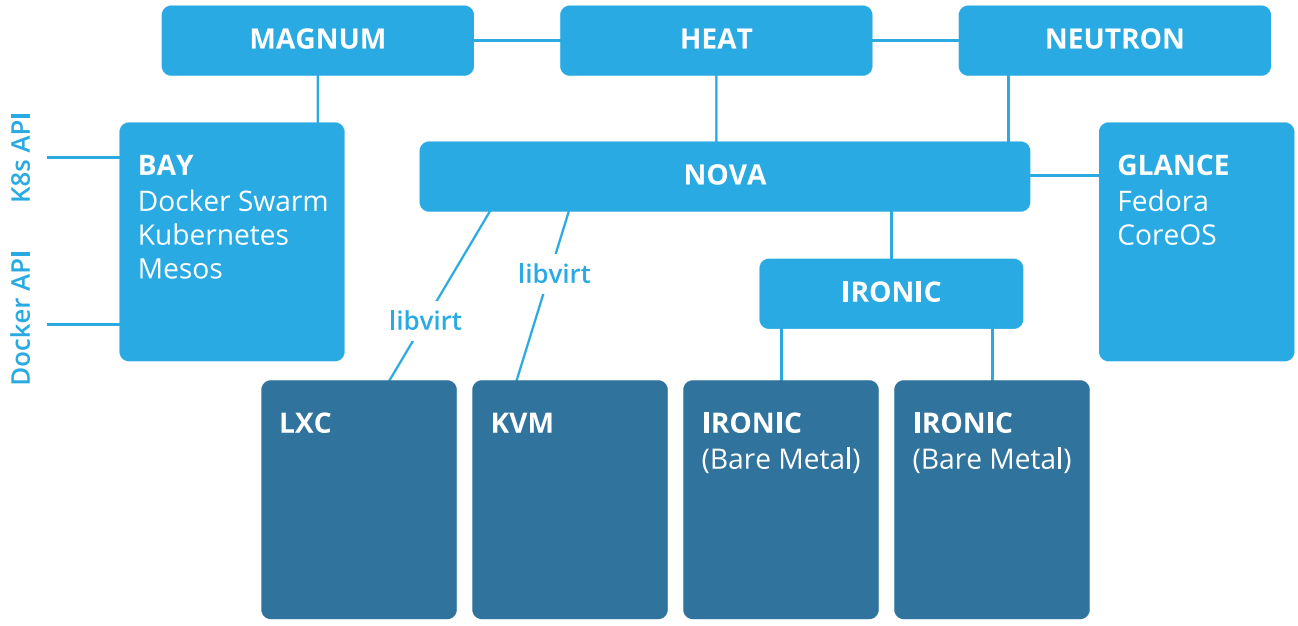

Figure 2: OpenStack Container-as-a-Service Support Architecture

Magnum

Magnum is an OpenStack API service that adds multi-tenant integration of prevailing container orchestration software for use in OpenStack clouds. Magnum allows multiple container technologies in OpenStack to be used concurrently, on a variety of Nova instance types. Magnum makes orchestration engines, including Docker Swarm, Kubernetes, and Mesos, available through first class resources in OpenStack. Magnum provides container specific features that are beyond the scope of Nova's API, and implements its own API to surface these features in a way that is consistent with other OpenStack services. It also allows for native APIs and native tools to be used directly, so that container tooling (like the Docker CLI client) does not need to be redesigned to work with OpenStack.

Containers started by Magnum are run on top of an OpenStack resource called a bay. Bays are collections of Nova instances that are created using Heat. Magnum uses Heat to orchestrate an OS image which contains Docker Swarm, Kubernetes or Mesos, and runs that image in either virtual machines or bare metal in a cluster configuration. Magnum simplifies the required integration with OpenStack, and allows cloud users who can already launch cloud resources such as Nova instances, OpenStack Block Storage (Cinder) volumes, OpenStack Database Service (Trove) databases, etc. to create bays where they can start application containers.

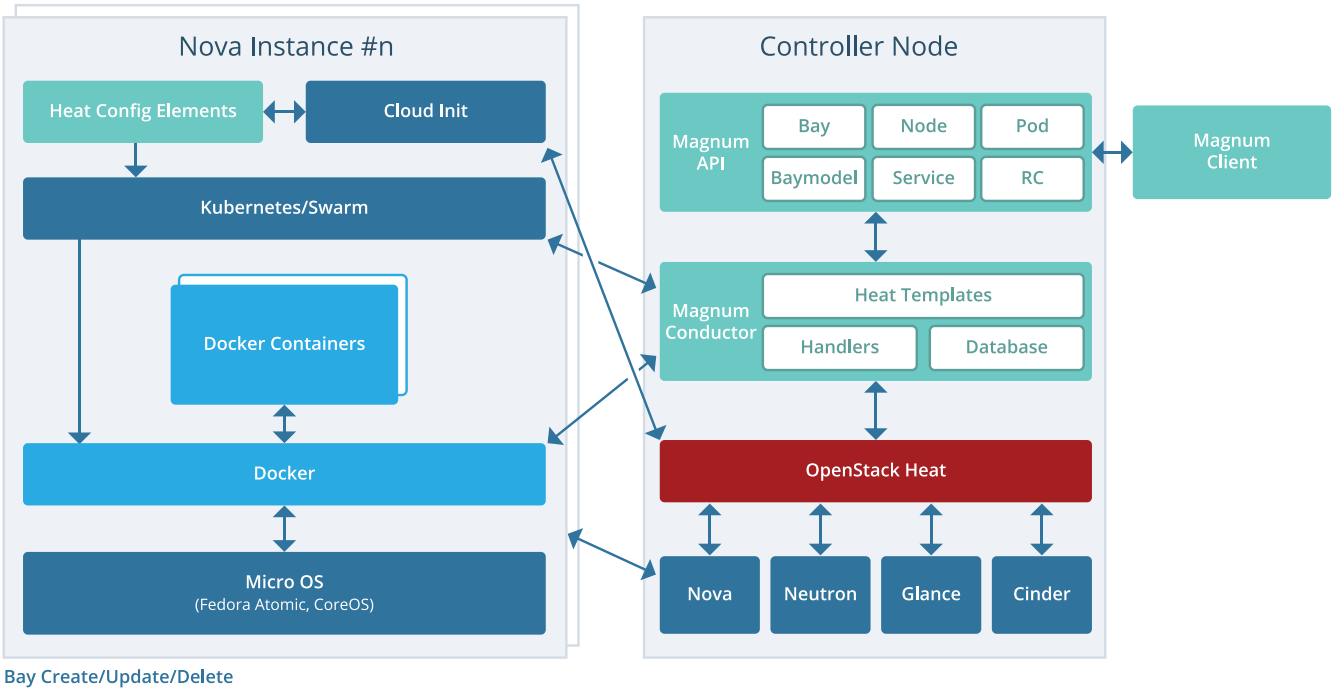

Figure 3: Magnum Architecture

Magnum leverages Docker Swarm, Kubernetes, and Mesos as components, but differs in that Magnum also offers an asynchronous OpenStack API that uses OpenStack Identity Service (Keystone), and includes a complete multi-tenancy implementation. It does not perform orchestration internally, and instead relies on Heat.

The same identity credentials used to create IaaS resources can be used to run containerized applications using Magnum, via built-in integration with Keystone. Some examples of advanced features available with Magnum are the ability to scale an application to a specified number of instances, to cause your application to automatically re-spawn an instance in the event of a failure, and to pack applications together more tightly than would be possible using virtual machines.

The second release of Magnum (Kilo-2) is available for download[12]. It includes significant test code coverage, multi-tenancy support, scalable bays, support for CoreOS Nodes, 8 bit character support, and 61 other enhancements, bug fixes, and technical debt elimination.

Magnum is at the point today, prior to Liberty, where OpenStack-based public cloud providers can start to leverage it. These users are contributing to Magnum, making containers accessible for individual IT departments later this year and beyond. Experienced users assert it is very reliable and really helps to run an app in an immutable infrastructure.

Magnum Networking

Magnum leverages OpenStack Networking (Neutron) capability when creating bays. This allows each node in a bay to communicate with the other nodes. If the Kubernetes bay type is used, a Flannel overlay network is used which allows Kubernetes to assign IP addresses to containers in the bay while allowing multihost communication between containers. Work is currently underway to leverage new networking features from the Docker community (libnetwork) to provide a native client experience while leveraging OpenStack networking, and offer a consistent experience between containers running outside of OpenStack cloud environments. This capability is expected in the OpenStack Mitaka release timeframe.

Magnum Security and Multi-tenancy

Resources such as containers, services, pods, bays, etc. started by Magnum can only be viewed and accessed by users of the tenant that created them. Bays are not shared, meaning that containers will not run on the same kernel as neighboring tenants. This is a key security feature that allows containers belonging to the same tenant to be tightly packed within the same pods and bays, but runs separate kernels (in separate Nova Instances) between different tenants. This is different than using a system like Kubernetes without Magnum, which is intended to be used only by a single tenant, and leaves the security isolation design up to the implementer.

[12] https://github.com/openstack/magnum/releases/tag/2015.1.0b2

Using Magnum provides the same level of security isolation as Nova provides when running Virtual Machines belonging to different tenants on the same compute nodes.

If Nova is currently trusted to isolate workloads between multiple tenants in a cloud using a hypervisor, then Magnum can also be trusted to provide equivalent isolation of containers between multiple tenants. This is because Magnum uses Nova instances to compose bays, and does not share bays between different tenants.

For more information or to get involved, please visit: **https://wiki.openstack.org/wiki/Magnum**

Kolla

Deploying and upgrading OpenStack has always been a complex task. With the advent of the core and projects structure of the OpenStack software[13], there is no longer an integrated OpenStack release. Projects can be approved and released on their own cadence, with most projects still opting to do releases at the end of the 6-month development cycles. Operators will be able to select from a number of projects to custom-build their deployment. This introduces more flexibility and choice but also complexity in deployment and ongoing operations.

Kolla is a new approach to deploying OpenStack within containers that results in new fast, reliable and composable building blocks. Kolla simplifies deployment and ongoing operations by packaging each service, for the most part, as a micro-service in a Docker container.

Containerized micro-services and Ansible orchestration allows operators to upgrade a service by building a new Docker container and redeploying the system. Different versions and package mechanisms such as distribution packaging, RDO, and from-source can be supported individually. Deploying OpenStack with Kolla does not reconfigure anything else in the deployed operating system, unlike other configuration management tools, lowering risk. And Kolla containers are easy to create.

Kolla provides immutability, because the only things that change are the configuration files loaded into a container and how those configuration changes modify the behavior of the OpenStack services. This turns OpenStack from an imperative system to a declarative system; either the container runs or doesn't. Kolla is implemented with data containers that can be mounted on the host operating system. For example, databases, OpenStack Image Service (Glance) information, Nova compute VMs and other persistent data can be stored in data containers which can be backed up and restored individually, again lowering risk and extending immutability throughout the OpenStack infrastructure.

[13] https://www.openstack.org/blog/2015/02/tc-update-project-reform-progress/

Kolla comes out of the box as a highly opinionated deployment system meant to work with the configuration of four pieces of information. For more experienced operators, Kolla can be completely customized by configuration augmentation, enabling an operator to customize their deployment to suit their needs as their experience with OpenStack increases with the execution of one operation.

Kolla's lead container distribution is CentOS but containers are built and tested for CentOS, Fedora, Oracle Linux, Red Hat Enterperise Linux, and Ubuntu container runtimes in both source and distro packaging models. Deployment occurs to an operating system that matches the container run-time operating system to preserve system call, IOCTL, and netlink compatibility. Deployment to micro-operating systems such as CoreOS, Fedora Atomic, and Red Hat Enterprise Linux Atomic are planned for the future.

Kolla implements Ansible deployment of the following infrastructure and basic services with a high availability model using redundancy to protect against faults:

- RabbitMQ
- MariaDB with Galera Replication
- Keepalived
- HAProxy
- Keystone
- Glance
- Magnum
- Nova
- Neutron with both OVS and LinuxBridge support
- Horizon
- Heat
- Cinder
- Swift
- Ceilometer

Kolla is ready for evaluation but not deployment by operators. A deployment-ready Kolla is anxiously awaited by operators worldwide.

For more information or to get involved, please visit: **https://wiki.openstack.org/wiki/Kolla** or meet the developers and users on **#kolla** on Freenode IRC.

Murano

Murano is an OpenStack project that provides an application catalog for app developers and cloud administrators to publish cloud-ready applications in a browsable, categorized repository available within OpenStack Dashboard (Horizon); and for administrators to obtain additional apps easily from public repositories such as the OpenStack Community App Catalog (apps.openstack.org), Google Container Repository, and Docker Hub/Registry. Murano provides developers and operators with the ability to control full application lifecycles, while allowing users - including inexperienced ones - a simple, self-service way of deploying reliable application environments with the push of a button.

Murano thus enables management and self-service delivery both of conventional application stacks and container oriented environments and PaaS solutions - including Kubernetes, Mesos, Cloud Foundry, and Swarm, on OpenStack. It can coordinate the use of all the Docker drivers within the context of an application through Heat or Python plugins. App and services developers use containers to run services using the container management tools of their choice, and Murano then acts as a lightweight abstraction so that the developer can provide guidance to the operator about how to handle app/service lifecycle actions such as upgrade, scale-up/down, backup, and recover.

Murano is available today as OpenStack packages in Juno and Kilo. An organization's cloud Application Catalog can be populated by importing Murano packages from a local repository, or from the OpenStack Community Application Catalog.

Murano environments may be as simple as a single VM or may be complex, multi-tier applications with auto-scaling and self healing.

Because each application and service definition includes all of the information the system needs for deployment, users will not have to work through various IT departments in order to provision a cloud service, nor are users required to provide detailed IT specifications. They are only required to provide business and organization requirements.

Installing third party services and applications can be difficult in any environment, but the dynamic nature of an OpenStack environment can make this problem worse. Murano is designed to solve this problem by providing an additional integration layer between third-party components and OpenStack infrastructure. This makes it possible to provide both Infrastructure-as-a-Service and Platform-as-a-Service from a single control plane.

For users, the application catalog is a place to find and self-provision third-party applications and services, integrate them into their environment, and track usage information and costs. The single control plane becomes a single interface from which they can provision an entire fully-functional cloud-based application environment.

From the third-party tool developer's perspective, the application catalog provides a way to publish applications and services, including deployment rules and requirements, suggested configuration, output parameters and billing rules. It will also provide a way to track billing and usage information. In this way, these third party developers can enrich the OpenStack ecosystem to make it more attractive for users, and users can get more out of their OpenStack clusters more easily, fostering adoption of OpenStack itself.

Social-as-a-Service company Lithium has a VM-based application running on an OpenStack private cloud today. They are working to transition to a sleeker, container-based model, using Docker and Kubernetes container orchestration and clustering. Containers make sense for Lithium because they enable development, testing and deployment on a single, simple and standardized runtime platform that's easily and quickly deployed and inherently lightweight: important as the scale and performance requirements of Lithium's platform continue growing, and as formerly-stateful aspects of their architecture evolve towards stateless micro-services. Kubernetes works for them for similar reasons. And, it offers a common scheduler across Lithium's private and public clouds, letting them build out a more transparently-integrated hybrid solution on multiple IaaSs - OpenStack, AWS, and Google Compute Engine. In the future, Lithium wants to explore bringing OpenStack APIs up into the Kubernetes/Docker layer to create fullstack DevOps tooling.

For more information or to get involved, please visit:
https://wiki.openstack.org/wiki/Murano

Conclusion and Next Steps

OpenStack, due to its exceptional flexibility, remains unparalleled in its ability to encompass emerging technologies. OpenStack users are discovering that there's value to one platform for virtual machines, containers, and bare metal. As OpenStack continues to evolve, it will deepen and mature its support for containers as well as other emerging capabilities.

If you are using OpenStack to manage your IT infrastructure, and are interested in exploring containers, current maturing OpenStack projects can help. Magnum and Murano provide state of the art services for running applications, whether from an integrated catalog or external repository, and Kolla is evolving into what many believe is the future of OpenStack deployment. They will continue to mature as container technology advances.

For those who want to tap into OpenStack software's container capabilities now, the services are available for exploration.

To learn more, visit each projects' wiki or join the Containers team at:
https://wiki.openstack.org/wiki/ContainersTeam