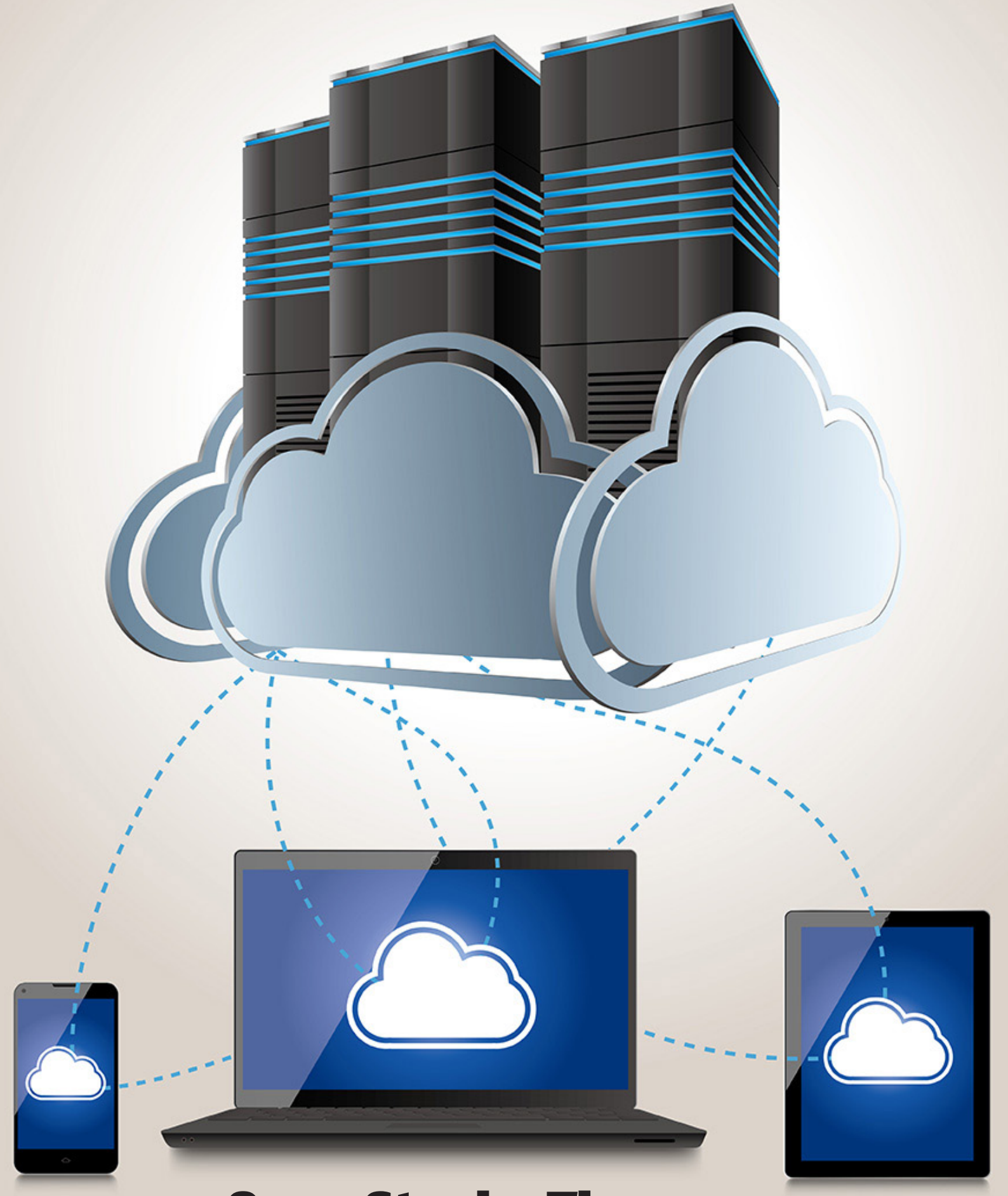

IVCANDY/ISTOCK

OpenStack: The essential guide

OpenStack Cinder: Block storage on the open-source cloud platform

OpenStack Cinder 101: The fundamentals of Cinder, how it is implemented, how to provision it, how it works with third-party storage arrays, its features and more

The OpenStack platform is an open-source collaboration to develop a private cloud ecosystem, delivering IT services at web scale. OpenStack is divided into a number of discrete projects, each with a code name with parallels to the purpose of the project itself. Virtual machines – or compute – are delivered through a project called Nova. In early OpenStack implementations, Nova virtual machines were stateless, that is they were not kept on persistent storage, and a Nova virtual machine would lose its contents when it was shut down.

As Nova developed, a feature called nova-volume was introduced to store virtual machines on persistent media, similar to the way Amazon Web Services Elastic Cloud Compute (EC2) stores instances on persistent media known as Elastic Block Store (EBS). The nova-volume feature was eventually superceded by a separate project called Cinder that delivers persistent block-level storage to OpenStack environments.

How Cinder works

Cinder performs a number of operations in OpenStack environments. In the first instance, it acts as a piece of middleware, providing application programming interfaces (APIs) that allow Cinder volumes to be created through use of the Cinder client software. A single Cinder volume is associated with a single Nova compute instance or virtual machine. Cinder keeps track of the volumes in use within OpenStack using a MySQL database created on the Cinder services controller. Through the use of a common interface and APIs, Cinder abstracts the process of creating and attaching volumes to Nova compute instances. This means storage can be provided to OpenStack environments through a variety of methods.

By default, Cinder volumes are created on a standard Linux server that runs Logical Volume Manager (LVM). This allows physical disks to be combined to implement redundant array of independent disks (RAID) data protection and to carve out logical volumes from a physical pool of space, called a volume group. Cinder volumes are created from a volume group called cinder-volumes, with the OpenStack administrator assigned the task of deciding exactly how this LVM group is mapped onto physical disk.

Cinder and external storage

Cinder can also manage external storage resources, either from a physical external storage array or from software-based storage implementations. This is achieved through the use of a Cinder driver that maps Cinder requests to the commands required on the external storage platform – in fact, the default LVM implementation is simply another Cinder driver. Support is available for iSCSI and Fibre Channel protocols, with specific support based on the capabilities of the supplier's storage hardware (see the support matrix described later).

Storage suppliers have been quick to provide Cinder support, enabling a wide range of storage hardware to be used in OpenStack deployments. Depending on the implementation, the driver allows OpenStack to automate the process of creating volumes and assigning them to Nova virtual machines.

Cinder keeps track of the volumes in use within OpenStack using a MySQL database created on the Cinder services controller

Some hardware platforms require storage administrators to create a pool – or pools – of storage for OpenStack to use – traditional arrays that use pools of RAID groups, for example. A list of supported platforms is available but this isn't exhaustive and many suppliers are not mentioned. You should check with your storage supplier for specific information on Cinder support and the features their drivers offer.

The use of external storage for OpenStack provides the ability to take advantage of native features on the storage platform where available, such as data deduplication, compression, thin provisioning and quality of service.

External storage isn't limited to physical hardware appliances; block storage can be assigned to OpenStack from a variety of software-based systems, both commercial and open-source. This includes Ceph and GlusterFS. Ceph, for example, is implemented through the use of Rados Block Device (RBD), a device driver in the Linux kernel that talks natively with a Ceph storage cluster.

OpenStack Cinder features

With each successive release of OpenStack (the most recent being Kilo – versions are named after successive letters of the alphabet), new features have been added to Cinder. Some of these have been implemented through a second version of the Cinder API, as version one didn't have support for the newer features.

Version one and two APIs provide commands to create, update, delete and extend volumes, as well as attach and detach them to instances – Nova virtual machines. Volumes can be assigned volume types, allowing them to be matched to a specific storage provider, where an OpenStack deployment takes storage from multiple providers. Alternatively, volume types can be used to differentiate between different classes of storage, based on, for example, physical characteristics such as RAID protection or performance.

Cinder provides the ability to take snapshots of Nova instances. For external storage platforms, this is achieved by using the native snapshot process of the underlying storage platform. The Juno release of OpenStack introduced the ability to group Cinder volumes into a consistency group, allowing all the volumes to be taken as a single snapshot. To date, only a few suppliers support this functionality.

Cinder supports the ability to take Nova instance backups. Unfortunately, this process is limited to using an object store as the backup target, with restores that require restoration of the entire volume. This may prove limiting in many circumstances and is one reason why Manila – the OpenStack file services project – could provide a more appropriate way to manage application data.

Cinder in OpenStack distributions

OpenStack distributions are available from a wide range of suppliers – well over 20 at last count. Each supplier provides support for a specific release of OpenStack and for each of the core OpenStack components. Cinder is a core component and ships with each distribution. The OpenStack marketplace provides a list of suppliers and their offerings. This also lists each of the projects and their supported levels as well as the supported level of the APIs. Today almost all suppliers support the version two API for Cinder.

Cinder provides great flexibility to add storage to OpenStack environments, through native LVM support or via an external appliance or software. However, as with all components of OpenStack, Cinder requires time and effort to understand and configure correctly. ■

With each successive release of OpenStack, new features have been added to Cinder

The Juno release of OpenStack introduced the ability to group Cinder volumes into a consistency group

OpenStack Swift 101: The object store for OpenStack apps

We run the rule over the OpenStack Swift object storage architecture, its key components, how it achieves resiliency, and the data protection methods in use and development

OpenStack is a collection of projects that deliver the components required to deploy a service-based private cloud. Code is delivered through twice-yearly alphabetically codenamed releases that introduce new projects, features and enhancements, typically in April and October.

The basic elements of OpenStack include compute (Nova, which delivers virtual machines), networking (Neutron) and storage (handled by Cinder, Swift and Manila). Cinder delivers support for block-based storage, allowing virtual machine states to be maintained across the creation and destruction of instances. It is an evolution of what were originally called “Nova volumes”.

Block-based storage is great for storing a virtual machine image but less flexible for storing application data

Block-based storage is great for storing a virtual machine image but less flexible for storing application data. Cinder volumes can’t be shared between running Nova instances, making it difficult to distribute access to data in an environment designed around the transient nature of an individual virtual machine or instance.

Swift object storage

To meet the needs for application data storage in OpenStack, the Swift project delivers a reliable, scalable and multi-user accessible object store. The OpenStack project has also recently introduced Manila, a scale-out file storage system. Swift is implemented as an object store, distributed across multiple nodes in an OpenStack infrastructure, using commodity disk storage components such as hard-disk drives and solid-state disks.

The term ‘object store’ implies no specific data format and content is effectively stored as binary objects with associated metadata. Data is stored in and retrieved from a Swift cluster using ReST-based API calls that are based on standard HTTP/S web protocols.

The use of ReST (Representational State Transfer) means that each object within the Swift object store can be accessed through a unique URL, which includes a reference to the object (the object ID) and its location. The open-source version of Swift distributed with OpenStack allows user-generated object IDs to be used when referencing objects in the store.

As OpenStack is by nature a multi-tenant environment, objects can be stored within Swift with some degree of hierarchy. A Swift object store is divided into accounts (also known as tenants or projects) and containers.

The use of containers provides the ability to apply storage policies to object data – for example, to set the number of replicas kept of each object. Policies are established at the container level. Note that containers in this context are not related to those being popularised by companies such as CoreOS and Docker; they are analogous to buckets used in public object stores such as Amazon Web Services.

Swift is implemented through a number of separate service components that deliver the scale-out and resiliency capabilities expected of object stores. These

include container servers, account servers, proxy servers and object servers, which are combined into an entity known as a “ring”.

Actual object data is stored on object servers, with other services used to implement features such as metadata management and distributed data access and protection. A service doesn’t imply a separate server. Some services can be run on the same hardware infrastructure, but high levels of resiliency are achieved by running multiple services across separate hardware appliances.

Separating data access services from data storage services allows a Swift instance to scale out in both capacity and performance. Data resiliency is implemented through the use of zones. A zone describes the sub-component of a Swift ring used to store one copy of data.

Resiliency is achieved by creating multiple redundant copies of data (called replicas) and distributing replicas across redundant components (zones) in the infrastructure. This can mean either a single disk drive or separate server, which provides the ability to create high availability through the geographic dispersal of data between datacentres. Requests to read data objects are delivered by the nearest, most consistent copy of that object.

Data consistency in Swift

In common with many object stores, Swift implements the concept of eventual consistency for data that is replicated between zones. Block-based storage is focused on the idea of either synchronous (immediate) or asynchronous replication (time-delayed) consistency.

Eventual consistency is similar to asynchronous replication in that the consistency of data is managed in the background, separate from the writing and reading of objects.

Object replicas are created as background tasks and replication completed as system resources (including network bandwidth) allow. This kind of replication is more suited to the scale-out Swift model, where individual servers may be offline or inaccessible as part of normal operations.

Most commercial object store systems now support the protection of data using erasure coding. Data protection using replicas is expensive in terms of storage capacity (especially with flash storage) whereas erasure coding provides data protection with only a fractional overhead in capacity. The trade-off comes in performance, as erasure coding uses algorithms in both the reading and writing of data that transform an object into a set of shards that are distributed across the infrastructure.

Erasure coding is currently only supported in beta mode within Swift, so end users should be careful about deploying it in production environments. However, we can expect erasure coding to be a future standard in Swift deployments, especially those at scale where the space/cost savings are the most beneficial.

This is not to say that improvements are not being made to existing data protection features. The Grizzly release of OpenStack, for example, introduced more granular controls to manage replica counts.

Commercial alternatives to Swift

Swift is an open-source platform, with a large amount of the support and coding coming from SwiftStack, which provides commercial support for Swift deployments. Other platforms are available that support the Swift API and can be used to replace or emulate the use of an open-source Swift deployment.

Separating data access services from data storage services allows a Swift instance to scale out in both capacity and performance

Implementations of the Swift ReST API are supported in object store platforms from Scality (since the Juno OpenStack release), Cleversafe, Cloudian, EMC Isilon, Hitachi HCP and others.

The benefits of using a commercial storage provider are obvious. Data is protected by hardware and operational processes with which the customer is already familiar. And hardware can be shared with OpenStack and non-OpenStack environments to allow data to be exchanged or moved in and out of a Swift-supported environment while providing data access through traditional protocols such as NFS and SMB.

Data is protected by hardware and operational processes with which the customer is already familiar

Using external storage also gives the ability to make use of features such as backup, encryption and mature role-based access controls that are still somewhat scarce in the open-source implementation of Swift.

One thing to bear in mind when using external storage is that there is no requirement to use Swift. It is perfectly possible to use other object-based APIs such as the S3 API from Amazon Web Services. Although the APIs are not directly interoperable, code changes to use either standard are minor in nature.

Simplivity adds ROBO hyper-converged box and OS enhancements
DataCore teams up with Curvature to offer recycled storage hardware
X-IO's Iglu Blaze adds replication to ISE HDD, flash and hybrid arrays
Panasas boosts scale-out NAS offerings with flash and disk AS18. ■

OpenStack Manila: File access storage for the open-source cloud

OpenStack Manila is the file level access method in development by the open-source cloud platform. What is it, how does it work and when will it be ready?

April 2015 saw the latest release of the OpenStack cloud computing platform, codenamed Kilo. This release introduced Manila, which brings support for shared file systems into OpenStack to complement its existing storage offerings, extending and improving its ability to consume external shared storage resources.

The Manila project introduces the concept of shared file systems into OpenStack. Until now, the two main storage projects were Cinder and Swift. Cinder provides application programming interface (API) support to manage block storage, specifically systems using block-based protocols, such as Fibre Channel and iSCSI.

Block storage provides high-performance access for virtual machines (VMs) and data, but a Cinder volume/LUN is limited to access by only a single Nova guest (virtual machine). This restriction exists because there is no inherent locking or synchronisation process built into block-level protocols. Block devices also have restrictions on capacity, making them difficult to increase and almost impossible to decrease in size.

Swift provides object storage support, making it suitable for storing large binary objects at scale. However, Swift storage isn't suitable for transactional data or to store VMs, as objects are typically immutable and updated in their entirety. Object stores are also not usually suitable for small objects due to overheads for data protection methods, such as erasure coding, and are relatively inefficient when using simple protection approaches, such as replicas.

Manila bridges the gap between block and object by providing the ability to map external storage systems using file-based NAS (NFS/SMB) protocols to Nova hosts and guests. File shares can be distributed between hosts and guests, as NAS protocol manages locking and data integrity processes required to provide multiple concurrent access to data.

Evolution of Manila

The Manila project was started in 2012 and developed as a fork of the Cinder project, as many of the concepts and API calls were anticipated to share much in common between file shares and LUNs/volumes. Since then the project has evolved, and in August 2014 it was classed as "incubated", effectively achieving final development status before becoming a full core project of the OpenStack platform. Although Manila is available in the Kilo release, it hasn't reached full core status, but that should happen later this year.

Manila bridges the gap between block and object by providing the ability to map external storage systems

The main function of Manila is to provide Nova compute instances with access to shared file-based storage. The architecture is designed to use external storage resources, such as commercial hardware from the likes of NetApp and EMC, or to use software-based implementations such as Ceph and GlusterFS.

OpenStack, and its Swift and Cinder object and block storage lead the way, but are not the only options for building open-source cloud platforms.

Manila effectively provides the orchestration components that manage the creation of file shares and the mapping of shares to Nova compute instances and does not sit within the data path. This functionality is implemented as a set of APIs, a command line interface (CLI) and integration into the OpenStack Horizon dashboard. Manila uses concepts and terms familiar to anyone implementing NAS, such as share (an instance of a file system), share access rule (an ACL) and security service (eg, LDAP or Microsoft Active Directory).

In addition, share network is used to describe the networking implementation associated with a share, and is used as one way to implement multi-tenancy support. Networking multi-tenancy is implemented using standard features such as VLANs and VXLAN.

Manila provides automated provisioning of file shares to Nova hosts (the servers running virtual machines). At this stage of project development, mapping file shares to Nova guests (virtual machines) is a manual process, although a number of proposals have been made on how the mapping could be automated.

Manila also provides opportunities for hardware suppliers to make their products valid in OpenStack deployments

Using Manila

The use of Manila makes it easier for developers to implement systems that scale at the virtual machine layer, but still need access to shared storage resources to deliver resiliency and availability. These features can be delivered using Manila without having to implement complex application-based replication and redundancy processes.

Manila also provides opportunities for hardware suppliers to make their products valid in OpenStack deployments and, as a result, we have seen a significant amount of development time provided by NetApp (on Clustered ONTAP), EMC (VNX) and IBM (Spectrum Scale), as well as OpenStack community members, such as Mirantis. In fact, the project lead for Manila is a NetApp employee. Meanwhile, NetApp and Mirantis have provided 29% and 35% of the source code of the project respectively.

Manila is not restricted to deployment in traditional storage arrays, and there is currently supplier development activity on the Ceph and GlusterFS open-source storage platforms. This includes support for protocols outside of traditional NAS (NFS/SMB), for example using device drivers built into the KVM hypervisor that use the native Ceph protocol.

The open-source project NFS-Ganesha that implements an NFS server in user space can also be used to abstract underlying NFS server hardware, although this introduces latency and more complexity in the data path. It is also possible to implement Manila support without an external storage array, using the generic driver provided with Manila. This driver creates a file share using a Nova VM and external block-based storage, with each file share creating a new VM.

Early days for Manila

Manila is available in the OpenStack Kilo distribution, but as discussed earlier it is not fully adopted as a core project. Users can try out features and see how they fit into their environment (assuming there is driver support), although there are still issues to resolve around networking multi-tenancy and provision of shares to Nova guests.

Users can try out features and see how they fit into their environment

There is also a lot of thought required about how external NAS storage should be integrated into OpenStack deployments. Manila provides only the conduit for provisioning and mapping new shares. It does not provide (at this stage) any integration with backup, or data protection, other than support for snapshots. ■

VMware vs OpenStack: How they handle storage

The opportunities and challenges presented by the two virtualisation environments when it comes to storage, backup and disaster recovery

The software-defined datacentre represents the current phase in the evolution of IT delivery to the business. It is an approach largely driven by virtualisation and the rise of VMware – and to a lesser extent, Microsoft Hyper-V – in the datacentre. Those platforms, VMware in particular, have attained an air of unassailability as virtualisation has swept the world's datacentres. But there is another virtualisation environment beginning to gain significant attention – the open-source, modular cloud platform OpenStack.

VMware's vSphere and OpenStack form extensive ecosystems, built up to deliver, operate and manage virtualised compute, storage and networking, as well as the systems management tools that deliver resource management, monitoring and alerting. And while OpenStack feels like it is hardly out of the blocks compared with the incumbent virtualisation platforms, the question is begged: to what extent is OpenStack a potential replacement for VMware?

The relevance of that question is reinforced by the example of PayPal, which has – in part – replaced vSphere with OpenStack. It is a question that has to be answered by looking at the opportunities and challenges for IT organisations inherent in the two platforms. A key aspect of this is how they implement and manage storage in the two environments. Here, we look at VMware vs OpenStack, and assess their keys strengths and weaknesses in storage.

VMware vSphere

In a vSphere configuration, storage is mapped to the VMware hypervisor ESXi using a number of standard protocols. Today, these include Fibre Channel, iSCSI, Fibre Channel over Ethernet (FCoE) and Network File Sytem (NFS). Fibre Channel connectivity follows a pretty standard configuration, with each ESXi host accessing storage through a logical unit number (LUN) or volume, which are then mapped to datastores, the storage container for a VMware virtual machine (VM).

Although vSphere provides a rich range of connectivity options, historically it has taken a fairly traditional approach to storage management. At first in the vSphere platform, storage was managed externally to the vSphere environment, typically by storage administrators who provided LUNs preconfigured to application performance and availability requirements. This configuration work was manual in nature, with a significant amount of pre-planning required.

In successive releases, vSphere evolved to provide a degree of automation to the functions of storage management. VMs could be rebalanced for capacity and input/output (I/O) load across a vSphere cluster using policies within Storage DRS. Meanwhile, Storage I/O Control allowed the prioritisation of application I/O that implements a basic quality of service.

But VMware has been on a journey to provide more intelligence between the storage and the hypervisor, and has incorporated a series of application programming interfaces (APIs) against which storage suppliers can develop their

In a vSphere configuration, storage is mapped to the VMware hypervisor ESXi using a number of standard protocols

platforms. These include vStorage APIs for Array Integration (VAAI), vStorage APIs for Storage Awareness (VASA) and vStorage API for Data Protection (VADP). These APIs allow the hypervisor to direct storage to manage virtual machines more effectively. VAAI provides I/O offload capabilities; VASA provides the hypervisor with information on the capabilities of the storage platform; and VADP provides the ability to take application consistent backups and snapshots. Probably the least developed area in the vSphere storage ecosystem is that of storage provisioning.

Pretty much every storage array supplier implements the layout and provisioning of its storage in a different way. There is little or no consistency available, outside of what can be achieved with the Storage Management Initiative Specification (SMI-S) standard. This makes it difficult for VMware to develop a standard API within the platform for storage provisioning. Solutions up to now have been to implement plugins in the vSphere management interface that allow a call-out to the storage array for provisioning, which is a manual process.

Pretty much every storage array supplier chooses to implement the layout and provisioning of its storage in a different way

However, in vSphere 6 we have seen the release of Virtual Volumes (VVOLs), a technology that will vastly simplify the provisioning process. A VVOL is a logical representation of part of a virtual machine. A minimum of three VVOLs are needed to represent a single VM, each one mapping to the configuration data, VM swap space and at least one virtual machine disk. VMware has worked with storage array suppliers to enable the creation and deletion of VVOLs from within the vSphere ecosystem, removing the need for a significant amount of storage administrator work and laying the foundation for implementing policy-based VM management at the storage layer.

OpenStack

The OpenStack project is focused on developing an ecosystem that allows customers to deploy applications in a software-defined datacentre. The platform is divided into a number of projects, each of which delivers part of the infrastructure.

Compute is delivered through Nova, networking through Neutron, with OpenStack storage delivered by two components called Swift (object) and Cinder (block). Cinder provides persistent block storage to OpenStack environments. The persistent feature is important because vanilla OpenStack virtual machines are transient and are destroyed on shutdown, but Cinder allows local server storage to be used for persistent VMs.

External storage suppliers can provide block storage to Cinder through the use of a Cinder driver. This is typically implemented as a piece of middleware that translates Cinder API calls into commands on the underlying storage platform.

Over time, successive releases of OpenStack have introduced new functionality and suppliers support this in specific releases. Protocol support is also supplier and release-specific, and includes iSCSI, Fibre Channel, FCoE, NFS, plus a range of bespoke implementations such as Rados Block Device for Ceph and GlusterFS. Object storage support is provided through the Swift component. Like Cinder, Swift can be deployed natively in the platform using commodity servers and storage, or it can be delivered by external storage suppliers.

Other OpenStack components are still in early development. Backup support is being developed through Raksha and a shared-file system through Manila. These will complement the existing Cinder and Swift components by providing the ability to share data between virtual machines and to integrate a VM-consistent backup function. So, if an organisation contemplated replacing an existing vSphere platform with OpenStack, how would that impact upon storage? From a persistent storage perspective, external arrays already deployed could be repurposed for use with OpenStack, subject to the availability of a Cinder driver.

But at this stage in the evolution the two platforms, vSphere offers more mature features and is pushing towards a higher degree of integration with external storage platforms through the use of features such as VASA and VAAI. This means OpenStack deployments could require additional management overhead compared with vSphere. But of course, many organisations may look at OpenStack as a way to reduce hardware costs and eliminate the need for external storage altogether.

VMware's vSphere has no direct object storage support, but within the infrastructure there is not necessarily any need for an object platform, unless demanded by individual VMs and applications. If it was, this could be implemented quite easily through the deployment of SwiftStack or another object store compatible with Amazon Simple Storage Service.

Differences in data protection

Today, the major difference in the two infrastructures is in the area of data protection. While vSphere provides native capabilities to manage virtual machine backups with application consistency, OpenStack at this stage in development pretty much leaves backup to the customer. There is a good reason for this – vSphere is still focused on the monolithic deployment of a single VM for a single application, compared with OpenStack, where a virtual machine is simply one of potentially many servers in a scale-out application. These OpenStack deployments assume that data is stored separately from the VM and more readily backed up outside the virtual machine.

Disaster recovery is another area in data protection that customers will have to implement themselves. There is no equivalent in OpenStack of vSphere Site Recovery Manager – the VMware component that manages the failover of storage at the array level. Again, the assumption is that disaster recovery can be provided by the application.

In summary, IT organisations looking to move to OpenStack will find a very different operational model, with potential savings in infrastructure costs replaced by an increased amount of operational management and application redesign. This means that for most, rip-and-replace is not a viable option. Companies such as PayPal have the developer resources in place to manage a transition to OpenStack and the latest news indicates this is the case.

PayPal developed its own OpenStack deployment and started a transition away from VMware, in part because of the benefits of being able to customise its own platform – something that is out of reach for the majority of IT organisations. ■

OpenStack deployments could require additional management overhead compared with vSphere

IT organisations looking to move to OpenStack will find a very different operational model