Managed infrastructure with IBM Cloud OpenStack Services

S. Cash
V. Jain
L. Jiang
A. Karve
J. Kidambi
M. Lyons
T. Mathews
S. Mullen
M. Mulsow
N. Patel

In this paper, we discuss how OpenStack®, an open source cloud computing software platform, can be used to deliver a managed private cloud by describing the IBM Cloud OpenStack Services offering. IBM Cloud OpenStack Services is an enterprise-class infrastructure-as-a-service, running on OpenStack in a selection of worldwide IBM SoftLayer® data centers. This managed service provides the necessary levels of control so that businesses can focus on applications and services, not on the infrastructure and cloud management systems beneath them. We discuss the design challenges that customers encounter when choosing managed private clouds and illustrate an architecture that addresses these challenges using open source technologies.

Introduction

An enterprise-class infrastructure-as-a-service (IaaS) cloud computing solution offers the reliability, availability, security, compliance, and performance of a dedicated private cloud, combined with the flexibility, scalability, deployment automation, and economic benefits of a public multi-tenant cloud. A managed infrastructure allows businesses to use the power of cloud computing without the challenge of needing many experts in a range of IT areas. In addition to the benefits of a cloud deployment, such as agility and elasticity, enterprises require a high level of security to meet their compliance requirements, high resiliency to meet their availability requirements, and isolation to avoid unpredictable performance.

Adoption of mobile [1] and social computing [2], plus a rapid rise in the number of Internet-connected devices (the "Internet of Things") [3], is creating an increase in the amount of data and enabling new analytics-driven insights [4]. Organizations are challenged with balancing traditional investments in "systems of record," the authoritative data sources for business information, with new compute models involving "systems of engagement," created through new public-facing applications. Businesses need rapid access to new resources such as compute, storage, networks, platforms, and services. These resources must seamlessly integrate with the back-end systems for the adopting organizations to continue to benefit from their prior investment. Ananthanarayanan et al. [5] examine the performance of Internet-scale file systems for cloud-based analytics applications. Sefraoui et al. [6] provide a comparative study of multiple open source IaaS cloud solutions. Open standards solutions can help integrate these innovative systems of engagement with traditional systems of record, preventing vendor lock-in while maximizing interoperability. Sobeslav and Komarek [7] provide a survey of open source automation and configuration management tools in cloud systems. Building on open source and standards can provide support for the availability, flexibility, and portability required by enterprise workloads.

As businesses adopt a strategy of high-performance interoperability between legacy information technology systems and off-premises cloud functionality, they are seeking IaaS models where they can focus on their core competencies of building and managing their applications without concerns about the underlying infrastructure. Armbrust et al. [8] suggest the use of multiple cloud providers to achieve high availability of applications. Businesses expect that the data in the cloud is managed with the same rigor and security as on-premises data and often prefer private hosting environments that are highly secure, scalable, and built using dedicated hardware. To efficiently use the available

Digital Object Identifier: 10.1147/JRD.2016.2515418

resources, we need to consolidate virtual machines (VMs) in the minimal number of physical servers, reducing the runtime power consumption and cost. Corradi et al. [9] discuss VM consolidation in an OpenStack** cloud and the guided prevention of performance degradation.

In this paper, we describe the key attributes of IBM Cloud OpenStack Services, the architectural overview of the offering, and the design of the key components. Then, we discuss the development and operations (DevOps) challenges and distributed management. Finally, we provide details on the offering's performance measurements.

IBM Cloud OpenStack Services

IBM Cloud OpenStack Services is based on OpenStack technology to meet the flexibility and control needs of enterprises. This technology provides the benefit of open standards, ready skills availability, vast open source tooling, and an ever-expanding ecosystem of related services. A key challenge was to design the offering so that IBM retains control over the managed components while providing the access required for an OpenStack cloud. IBM Cloud OpenStack Services provides a managed IaaS, where the customers are responsible for consuming compute, storage, and network resources based on their requirements. IBM manages the hardware, hypervisor, storage, network, and security to meet the customers' requirements.

We designed the service as a turnkey managed offering with setup and configuration of "bare-metal" servers on SoftLayer** and services in a high-availability configuration completed within 72 hours. While automation allows a private cloud to be deployed in less than half a day, ensuring that the customer's hardware choices are available in its selected data center and are configured correctly can take up to 72 hours. IBM Cloud OpenStack Services offers speed, agility, and elastic scaling of compute and storage to meet workload demands. Capabilities are sold by the node and incremental storage volume, allowing customers to meet changing business demands with the capacity and granularity they need.

This offering is designed to be highly available and comes with a service-level agreement assurance of 99.95% for all components managed by IBM, including 1) the Horizon web portal (dashboard), 2) the OpenStack application programming interface (API), 3) the compute node hypervisors, and 4) the virtual private network (VPN) and firewall gateways. With redundant components and geographical dispersing of instances, we can offer a higher service level than relying solely on monitoring to restart failed components.

IBM Cloud OpenStack Services includes an enterprise-ready software defined network (SDN) service integration with OpenStack. This allows for broad network configuration flexibility, including self-service modifications as business needs change.

Security is a prime concern for enterprises moving to the cloud [10, 11]. IBM Cloud OpenStack Services provides an environment with security and privacy. Security has been a key attribute in every major aspect of the design. In addition to providing security isolation, a dedicated environment means no interference from other tenants contending for resources. Performance depends solely on the business' workload, not on the demands of other tenants.

We designed and built this offering to support the full DevOps lifecycle and to work with environments relying on DevOps. The integration of development, deployment, and operations can enable a more seamless and rapid transition from concept to production, which can improve time-to-market and produce higher quality deployments. Enabling DevOps-style integration between development and delivery allows earlier discovery of defects and increases development quality by letting developers see how their code runs in a production-like environment much more quickly than when using periodic release methods. IBM Cloud OpenStack Services uses this approach, coupled with rapid, incremental updates, to reduce defects and deploy releases that are ready for production use.

Architecture

Overview

IBM Cloud OpenStack Services is hosted on SoftLayer [12], a global cloud infrastructure computing provider, and is managed by IBM. Compute and storage components are dedicated to a single tenant, as shown in **Figure 1**. The architecture integrates the following key technologies:

- Kernel-based virtual machine (KVM) [13] based hypervisor to host virtual memories.
- SDN to provide multiple isolated networks, routing between those networks, and bring-your-own IP addresses.
- Ceph [14] clustered storage to provide support for block and image storage.
- Swift [15] based private object storage.
- Chef [16] and Jenkins [17] based DevOps for rapid deployment.

The base is a set of KVM hypervisors hosted on RHEL** (Red Hat Enterprise Linux**) servers and managed by OpenStack. KVM is generally accepted as an enterprise-ready hypervisor due to the thriving community backing and support. The IBM Cloud OpenStack Services 2.0 release is based on OpenStack Juno. We provision a customer's instance in a separate SoftLayer account

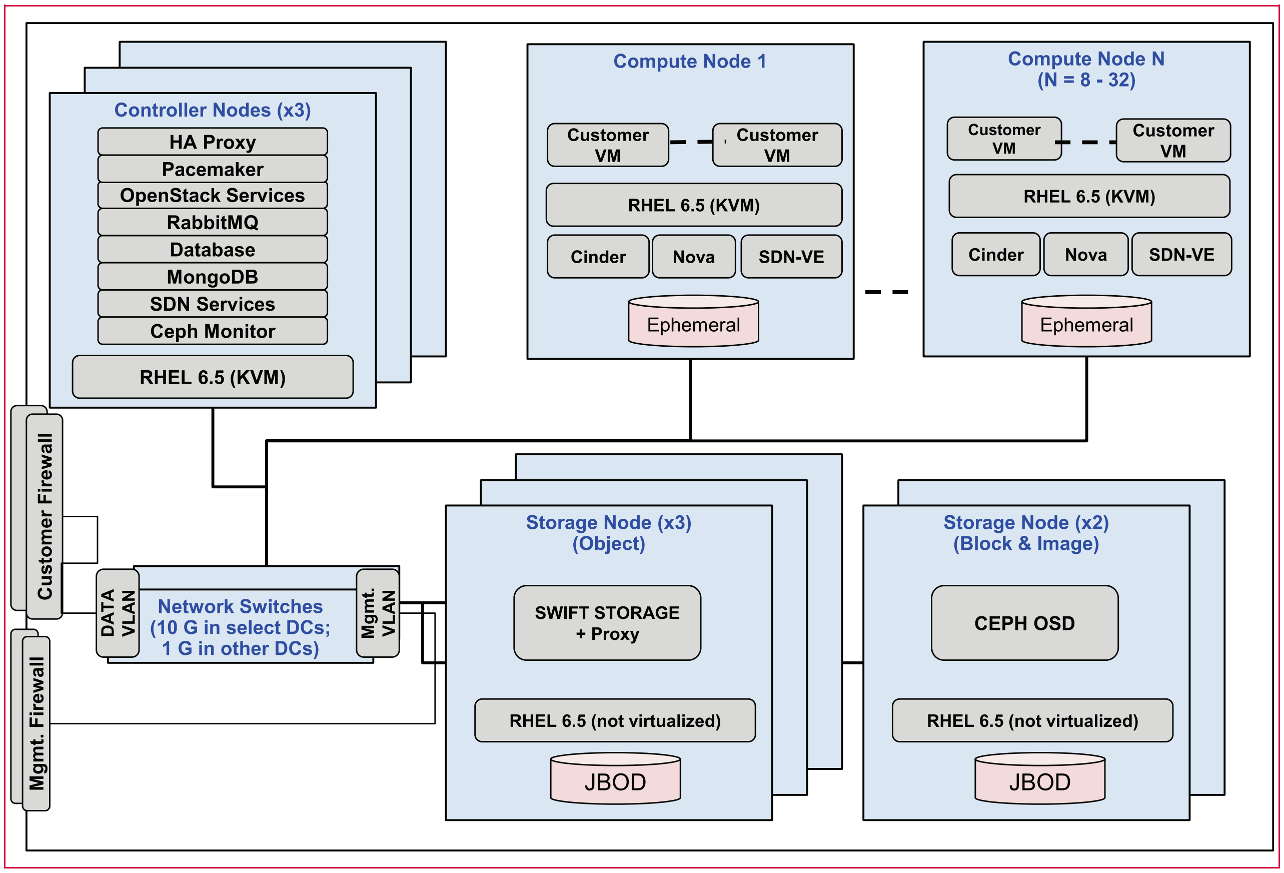

Figure 1

Hardware architectural layout of IBM Cloud OpenStack Services. (CEPH OSD: Ceph Object Storage Daemon; JBOD: Just a Bunch of Disks; VM: virtual memory; VLAN: virtual local area network; HA: high availability; SDN: software-defined network; RHEL: Red Hat Enterprise Linux; VE: virtual environment; DCs: data centers; 10 G: 10 gigabit Ethernet; 1 G: 1 gigabit Ethernet.)

to provide a predictable network layout and to provide isolation between different instances, making each instance private. These instances consist of management, compute, and storage nodes with their numbers varying based on the size and high availability characteristics of the offering. The management nodes host high availability proxy (HAproxy), OpenStack, and SDN components.

Controller

Since we are providing cloud IaaS to enterprise customers, high availability is the basic requirement for OpenStack services. High availability systems are designed to minimize system downtime and data loss. A crucial aspect of high availability is the elimination of a single point of failure. In the context of an OpenStack controller, this means that the OpenStack services need to be highly available. OpenStack currently meets such availability requirements for its own infrastructure services, meaning that an uptime of 99.99% is feasible for the OpenStack infrastructure itself [18]. This requires the OpenStack controller architecture to be designed for such expectations. Applications deployed on IBM Cloud OpenStack Services can be made highly available through the use of Heat-based auto-scaling, which uses monitoring to detect when new instances need to be deployed on failure or workload spikes.

OpenStack services are interconnected by Representational State Transfer (REST)ful HTTP based APIs and Advanced Message Queuing Protocol (AMQP) based Remote Procedure Call (RPC) messages. Redundancy for stateless OpenStack API services is implemented through the combination of Virtual Internet Protocol (VIP) management using Pacemaker [19] and load balancing using HAProxy. Stateful OpenStack components, such as the state database and messaging server, rely on their respective active/passive modes for high availability.

The challenge in providing high availability for stateful services is the data consistency during network partition. In the SoftLayer environment, a redundant network may not be available on every type of bare-metal server. Cost may also limit the choice of redundant hardware. A typical approach to resolve this issue, as described by Coulouris et al. [20], is to use a quorum-consensus approach.

We chose MySQL** with Galera** to provide a highly available OpenStack database [21]. This provides synchronous database replication in an active-active, multi-master environment. Galera replication manages the data availability internally, while HAProxy manages the access availability. Since IBM Cloud OpenStack Services is a co-located private cloud, three MySQL nodes are sufficient to deal with network partition.

For the choice of message queue, Rostanski et al. [22] present a performance evaluation of RabbitMQ** [23] in high availability-enabling and redundant configurations. We chose RabbitMQ as the messaging server because it is the default of the OpenStack Oslo RPC messaging system [24]. The three RabbitMQ nodes are running in the active/active mode with a mix of disk and RAM nodes for message queue in order to have balance between durability and good performance. RabbitMQ Queue Mirroring supports the high availability of the RabbitMQ cluster.

The three-node approach for OpenStack database and message queue divides the OpenStack controller into three availability zones. In the context of IBM Cloud OpenStack Services, the availability zone fits into three bare-metal nodes. Each OpenStack controller component runs in the VM. The replicas are distributed to the three nodes. To improve efficiency and to reduce the management complexity, components are grouped together to reduce the total number of VMs. The grouping of components into a VM does have performance impact due to the competition of compute resources from each component within the VM. The problem is no different than running OpenStack components within a bare-metal server. It is important to size and tune each VM and the hosting bare-metal server properly.

The IBM Cloud OpenStack Services three-node OpenStack controller architecture is shown in **Figure 2**. The grouping of the components is based on the service boundary. The OpenStack stateless services—including Keystone, Nova-API, Glance-API, Glance-registry, Cinder-API, Neutron-server, Heat-API, and Ceilometer-API—are grouped into a single operations VM called *OpenStack Service (OPS) Node*. The database, message queue, SDN services, Horizon, and load balancer run in separate VMs. We are exploring the use of containers (i.e., Docker containers) to reduce hypervisor overhead for VMs when running the services.

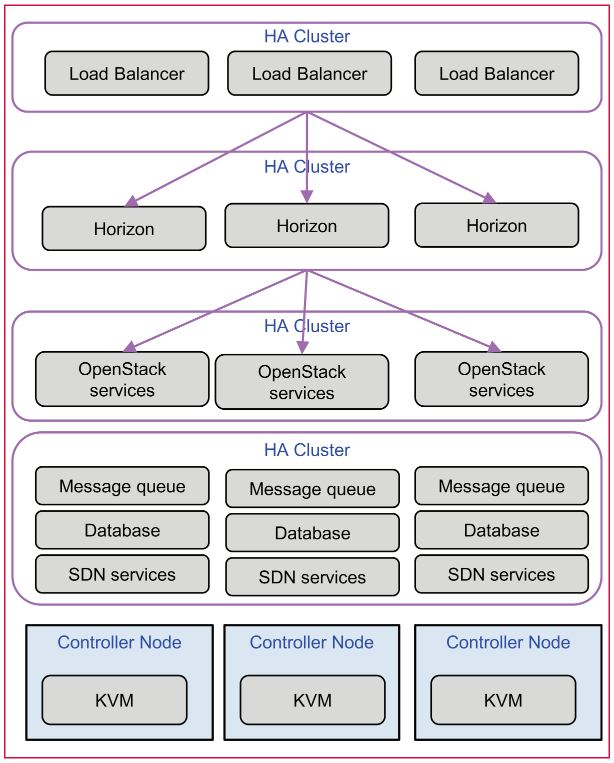

Figure 2

High-availability (HA) architecture of the OpenStack controller.

Software defined network

IBM Cloud OpenStack Services runs on the SoftLayer network environment that provides customers with access to two networks: the Internet and the SoftLayer private network. However, we can project a much more sophisticated view of the network to our customers by using the capabilities of SDN Virtual Environments (SDN-VE) [25], which is an IBM-developed network virtualization solution. Using these capabilities, IBM Cloud OpenStack Services instances can have multiple projects, each with its own distinct software-defined Neutron networks, all deployed on a common physical network infrastructure. SDN-VE uses an on-demand address dissemination protocol in the control plane that is built to serve the scaling needs of private and public clouds.

SDN-VE is a network overlay solution based on the Virtual Extensible Local Area Network (VXLAN) encapsulation [26]. Packets sent by tenant endpoints, VMs, or instances in IBM Cloud OpenStack Services are intercepted by tunnel endpoints (TEPs), which then encapsulate those packets in an outer header and transport or "tunnel" the encapsulated packet to another TEP to which the destination is attached. While the TEPs are in the physical network, the endpoints that are connected to the TEPs are effectively detached from the physical network due to the tunneling operation. All tenant addresses are qualified by a virtual network identifier (VNI/VNID) that is carried in the outer VXLAN

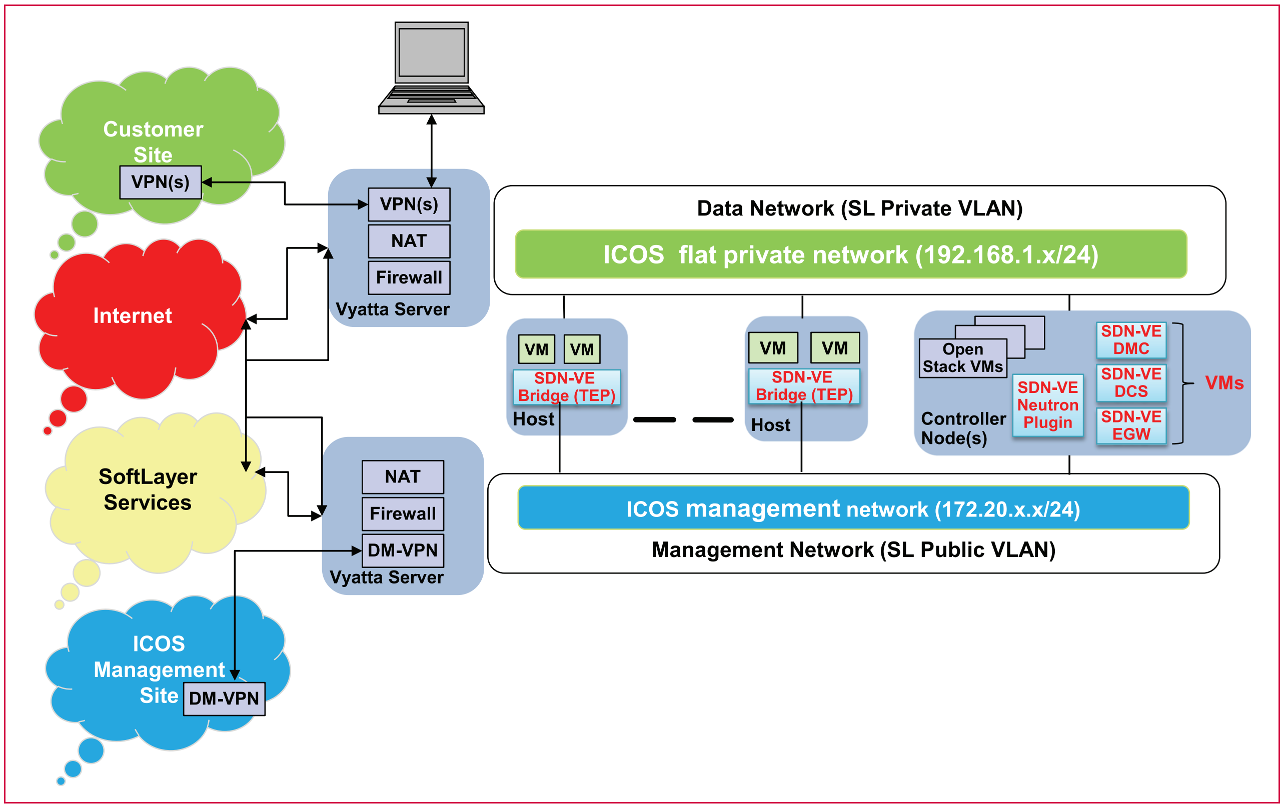

Figure 3

IBM Cloud OpenStack Services network architecture using SDN-VE. (DMC: DOVE Management Console/Controller; DCS: Distributed Connectivity Service; EGW: External Gateway, SL: SoftLayer; NAT: Network Address Translation; OpenVPN: Open Virtual Private Network, VPN: Virtual Private Network; TEP: Tunnel Endpoint; VLAN: Virtual Local Area Network.)

header, allowing each tenant network to have its own addressing schemes and network semantics.

This allows SDN-VE to virtualize the network over any physical network, as long as there is simple IP (Internet Protocol) connectivity in the physical and underlay network, including the SoftLayer network as it is today or how it may evolve in the future. These same concepts can be easily extended to an architecture that supports a public cloud offering. The same physical network can be configured in different ways for different offerings, yet the client/virtual networks can provide the same standardized infrastructure and usage models. IBM Cloud OpenStack Services integrates with SDN-VE via OpenStack Neutron [27], and this provides yet another standardized API interface to integrate with SDN-VE.

SDN-VE solution has four components:

- The Open DOVE [28] Management Console (DMC) provides a command line interface and a web user interface. It supports native and OpenStack APIs. The DMC is highly available and runs in an active-standby configuration.
- The Open DOVE Connectivity Service (DCS) provides address and policy dissemination service to the data plane elements. The DCS is clustered for load balancing and high availability. In IBM Cloud OpenStack Services, only two nodes are deployed in the cluster.
- Gateways connect the virtual overlay network to the physical network. In IBM Cloud OpenStack Services, the external gateway (EGW) is used. The EGW provides network address translation (NAT) and floating Internet protocol translations [29]. The DMC, DCS, and EGW are all delivered as virtual appliances.
- A "deployer" appliance has been developed to install and perform rolling upgrades of these appliances.

Finally, each compute host must run the SDN-VE agent, which securely interacts with the DMC and the DCS to program the Linux bridge and Virtual Extensible Local Area Network (VXLAN) module on the host to provide virtual overlay networks to VMs on that host. **Figure 3** shows how these components are deployed in our single-tenant solution. Neutron APIs allow a variety of use cases [30]. IBM Cloud OpenStack Services uses two of the most common use cases, multiple shared networks and per-tenant routers with private networks. SDN-VE supports these use cases with the combination of functionality within the TEPs and the EGW. SDN-VE enables this use case as the interface between the router and the tenant private network is handled at the TEP on the compute host, while the external interface of the router that connects it to the external network (the address that supports NAT and floating IP) is implemented at the EGW.

Storage

Balancing the cloud storage data availability, durability, scalability, and management requirements of IBM Cloud OpenStack Services against the technology and cost constraints imposed by underlying cloud infrastructure software and hardware architectures involves many critical decisions.

Since IBM Cloud OpenStack Services is deployed on top of SoftLayer hardware, the hardware choices are limited by the contents of the SoftLayer catalog. Hardware costs are especially significant in a single-tenant cloud where the underlying hardware cannot be shared. The SoftLayer networking infrastructure also strongly influences the final designs. IBM Cloud OpenStack Services storage nodes are restricted to two physical network adapters on the SoftLayer "private" network and can additionally use two physical network adapters on the "public" network.

For the software architecture, we made the key decisions to use Ceph block storage software and OpenStack Swift. Ceph is the most popular block storage choice in OpenStack deployments across the industry [31]. Swift is the core OpenStack component specifically focused on providing an object storage solution.

Ceph is based on the underlying RADOS object store [32]. It is capable of providing a unified storage subsystem. By using the RADOS Gateway (RadosGW) component of Ceph, we can use one set of storage servers to provide both object and block APIs. This has clear cost advantages and simplifies cluster management. Although the provided object API is Swift-compatible, full API support is not yet available. The RadosGW approach does not use the Swift proxy tier, and the ecosystem of extensible Swift middleware is not supported. OpenStack Swift is designed to excel specifically at object storage, which allows for optimization. For these reasons, IBM Cloud OpenStack Services uses both Ceph and Swift. Different storage nodes are used for each to avoid hardware over-commitment and complexity.

One goal of the block storage system is to achieve certain durability service-level agreements. We replicate data with multiple copies and check consistency on a regular basis. However, replication increases cost and cluster-wide consistency checking impacts performance. A data durability model [33] was used to determine that a two replica Ceph configuration can achieve a six 9s (99.9999%) durability level, meeting general customer requirements. This reduces the amount of Ceph storage nodes required by one third, compared with using a replica count of three, and still meets the initial durability objectives set by our offering team at a reduced cost. Architecturally, Ceph supports the definition of storage pools, which include specifying the number of replicas to use for a given pool. For customers who desire the higher durability guarantees provided by three replicas, we hope to be able to provide such an offering in the future. We expect that the combination of cache tiers with erasure coding [34] can enable IBM Cloud OpenStack Services to achieve equivalent durability with better price and performance.

IBM Cloud OpenStack Services also uses Ceph for its Glance image repository, obtaining the benefits of the Ceph copy-on-write and snapshot capabilities. IBM Cloud OpenStack Services uses drives that are locally-attached to the compute nodes for ephemeral storage. A motivating factor for this decision is the desire for data locality. We found that booting a VM from a locally-attached ephemeral drive is most efficient using "qcow2" images, while booting from a Ceph volume is most efficient using images in "raw" format. The use of an ephemeral drive does incur an overhead during cold start when the image is first copied from Glance to the compute node. Each time images are added or changed, Glance must redistribute the new images to the required compute nodes. The network becomes a bottleneck during this image transfer. The three Glance controller nodes ease this network overhead. There is no overhead when booting from a Ceph volume using raw images because only the required content is streamed to the compute nodes. However, with boot from volume, the required content is streamed over the network for each VM instance. These are areas for possible future improvement.

The design decisions that were made for the Swift configuration must account for similar factors as were considered for Ceph. Cost and anticipated scale are both factors in deciding to deploy the Swift proxy, account, container, and object servers on the same node. At higher scale, these are often distributed across multiple nodes. Data durability modeling influences the decision of the number of replicas to use. In the case of Swift, the decision is to follow the common industry practice of using three replicas. The fact that Swift object storage is an optional part of our offering somewhat mitigates the cost concerns of needing three storage nodes per object. Similar to Ceph pools, Swift storage policies provide the architectural tool to potentially provide a reduced durability (two replica) offering in the future. Swift adoption of erasure codes similarly offers the promise of providing equivalent durability at reduced cost.

A final illustrative design point involves Swift storage expansion and heterogeneous node sizes. The entry point IBM Cloud OpenStack Services Swift offering uses a 2U hardware building block that can accommodate eight 4 terabyte (TB) drives for Swift objects. As the object storage usage grows and additional nodes are needed to provide more capacity, it quickly becomes inefficient from a cost perspective to add more of these 2U building blocks. A 4U hardware building block, accommodating thirty-two 4 TB drives, is much more efficient. However, using three nodes with 8 drives each, and a fourth node with 32 drives, presents potential data availability challenges if the total cluster drive capacity is to be used. We want to avoid the situation where multiple replicas of a given object reside on the single large node. To avoid complexity in managing this kind of grossly imbalanced cluster, IBM Cloud OpenStack Services can migrate the nodes to the larger chassis size when sufficient capacity growth is required, and can gradually retire the 2U units. Handling heterogeneous cluster node sizes is a potential future operational enhancement.

IBM Cloud OpenStack Services meets its needs for highly available, durable, and scalable storage using the OpenStack Swift and Ceph software solutions. Both solutions provide for fault-tolerance and self-healing, and the Chef-based automation helps with ease of operation. Deployed on top of commodity bare-metal servers from SoftLayer, the technologies have proven to be very adaptable to the specific needs of our offering.

Security

Security remains the number one concern when enterprises move to the cloud [35]. Security is a top priority of our IBM Cloud OpenStack Services offering. Our goal is to make it more secure than traditional on-premises IT environments. The only way to achieve this is to automate and integrate the best security technologies into our DevOps, which we refer to as security development and operations (secDevOps).

Building the IBM Cloud OpenStack Services systems starts with provisioning the many different bare-metal systems (BMSs) and VMs. All are provisioned with the same base operating system image. We call this singular image “Lucy” because it is the genesis of all BMSs and VMs within IBM Cloud OpenStack Services. We made O-ACEML (Open Group—Automated Compliance Expert Markup Language) [36] technology more secure. O-ACEML met our criteria of working with IBM’s ITCS104 security compliance standard. O-ACEML automatically configures the systems and provides exact reporting by referencing the human readable sections of the compliance standard and determining how the automation program configured the settings to meet the standard. The other security technology automated in our secDevOps is security scanning. The results of the security scan and configuration settings are compared against a known secure baseline, then are logged in logstash to streamline compliance and auditing.

Building IBM Cloud OpenStack Services starts with two pairs of redundant dedicated physical Vyatta** firewall gateways. One pair provides a redundant protected VPN connection to the customer’s site. The other pair provides a VPN connection for IBM Cloud OpenStack Services support to monitor and maintain the customer’s environment.

When secDevOps builds an IBM Cloud OpenStack Services environment, each component is built as a consistent, uniform, and fully automated process using Jenkins and Chef:

- The security compliant and secure Lucy image is applied.
- Chef installs and configures the component’s functionality, e.g., installs OpenStack Nova to create a Nova VM.
- O-ACEML is run to check all of the security configuration settings. If any of the settings are not correctly configured, O-ACEML reapplies the security configuration and rechecks whether failure results in a build error.
- A security scanner is signaled, which scans the system and compares it to a known secure baseline scan for that component type. Any differences are flagged as errors. Otherwise, the scan output is logged and archived, making compliance audits simple and fast.

When the entire environment is built, a final automated scan is performed on critical access points. Only then are the customers given their dual authenticating credentials, allowing them to access their environment using their customer-facing Vyatta VPN. The IBM-facing Vyatta VPN is used by IBM Cloud OpenStack Services support for continuous monitoring and service.

DevOps

Challenges

The solution requires a high level of integration between development and operations. This integration extends not only to tools and technology but also to processes and organizational responsibilities. The challenges faced in building the solution generally fall into two groups: distributed management and continuous integration. These two areas, while clearly interrelated, are very different problem domains with different and unique requirements for each.

Distributed management

IBM Cloud OpenStack Services is a distributed and scalable solution for providing OpenStack-based IaaS across multiple worldwide data centers. It is crucial that the service be implemented consistently in all data centers, and that the delivery team has a single point-of-control access to these data centers. Meeting these requirements necessitated the development of a distributed management infrastructure that allows both targeted control over any individual instance of the service, referred to as a cloud instance (CI), and distributed control over all CIs and data centers.

We were faced with the challenge of being able to provision new data centers and new CIs within existing data centers, with the speed required to onboard new customers and expand into new geographies.

Guiding principles

In building the infrastructure for distributed management, we focused on key principles. The first principle was consistency. Speed and flexibility provide little value if the result cannot be consistent across target endpoints. This drove us to ensure both recoverability, by having a transaction model with well-understood and robust compensation flows, and traceability, by having centralized generation and storage of diagnostic output to enable easy debugging and diagnosis of any operational failures.

Another key principle was simplicity. By reducing the number of flows and the number of operations within any given flow, we improve consistency and maintainability. This must be an ongoing activity. In many cases, we began with a complex flow meeting key requirements for consistency and recoverability. We then iteratively refined and optimized the flow, removing and replacing unnecessary steps so that the flow remained consistent and recoverable. At each stage, we focused on making the smallest possible change that produced measurable improvement.

We focused heavily on maximum automation, developed and managed centrally. We strove to have as few manual operations as possible to operate and manage the solution. We ensured that all development teams (those focused on management tooling and those working on user function) understood this as a firm requirement and accounted for it in their development cycles. For every new component, we required that the deployment and maintenance operations be automated. This extended beyond regular maintenance to security and compliance. For example, we provided an audit trail that could be integrated with centralized log collection.

Approach

We followed an incremental and an iterative approach throughout solution development. In many cases, we built the final solution piece by piece, finding workable paths and then expanding on them as we better understood the problem domain. A good example of this is the distributed storage platform for distribution of builds and installables to the various data centers. We began by using a single network storage instance which could be shared by the development platforms for staging to a common area. We selected resynchronization to distribute code to endpoints, as resynchronizing provides for fully recoverable transfers. In the early stages of the project, we did not know the geographic distribution of our data centers. Although resynchronizing might not be the eventual optimum solution, it was sufficient as a start. As the solution expanded geographically, it became clear that pseudo-synchronous access to installables was not workable from a single storage instance, so we built more storage instances and kept these synchronized using resynchronization-based replication. This let us stage and co-locate new code with new data centers, providing for much faster and robust access at actual deployment time.

In many cases, a solution which may have appeared to be feasible during the initial design stages turned out to be impractical, for reasons that were not always foreseeable. In one case, we discovered that the VPN client being used by the developers was not sufficiently reliable over a specific network path, and we had to explore both alternate clients and specific routing entries to provide the developers with stable access to their environments.

This model is further illustrated by the core management tooling. We began with Jenkins as a central management console. We chose Jenkins for its simplicity of deployment and use, and for its flexibility. Jenkins ships as a single Java** 2 Platform Enterprise Edition (J2EE**) archive, which can be readily deployed in a Tomcat server and does not require a great deal of skill or expertise to get working. This meant that it was a quick way to setup centralized execution of scripts across our initial inventory of management sites and CIs. A developer can write a script in one of several languages (primarily Ruby, Chef, Python, or shell) and can enable the parallel execution of the script across multiple targets with very little work. The developer can focus on what really matters—the logic in the script—as opposed to mastering a more complex than necessary integration API or tool. Jenkins also provided a central view of all of the job definitions (including their execution such as when, where, by whom) and their output, supporting our traceability and serviceability needs.

As we extended the geographical distribution of our sites, we realized that execution from a single point on the network across geographies was neither reliable nor workable from a speed standpoint. Unreliable wide area network links and network congestion are key reasons we came to this conclusion. As a result, we created a tiered infrastructure, essentially “store-and-forward” for execution, with the master Jenkins server controlling flow execution at multiple Jenkins slaves, each populated with an identical and centrally managed set of job definitions. This architecture provided much greater reliability and scalability.

As we progressed, we realized that our approach allowed us to target execution to a subset of our sites. We needed this flexibility to allow development teams to branch as required and to allow testing on a restricted set of endpoints. Since we retained the ability to manage job definition and execution centrally, we did not endanger our target goal of consistency. Instead, we were able to improve our continuous integration by staging job execution across sites and CIs as required.

Continuous integration

Development of a service offering does not readily lend itself to fixed release cycles. A service offering comprised of running instances with existing customers must meet requirements for reliability, availability, and serviceability) while allowing for the introduction of new function, bug fixes, and other maintenance operations. We implemented a continuous integration model based on sound agile development practices and modern DevOps methods. This uses the integration of new drivers every two weeks, each of which has gone through a full cycle of testing prior to deployment. The deployment mechanism supports this approach by rolling out new drivers to existing sites without interrupting service, a key component of successful continuous integration.

Performance

OpenStack requires configuration tuning to achieve the required concurrency and performance gains. The number of workers, threads, pool sizes, and timeouts must be configured for each component. System limits must be increased to achieve high scalability. We used the Cloud Rapid Experimentation and Analysis Toolkit (*cbtool*) [37] to stress the system with increasing resource workload and to measure and improve the performance of the system under different concurrent load scenarios. We found bottlenecks and tuned the systems by making configuration changes. We also provided numerous OpenStack code fixes to allow more than 100 VMs simultaneous burst provisioning under load.

First, we used separate tests that provisioned VMs with predefined concurrency up to capacity and de-provisioned the VMs from full capacity to empty using Noload (no benchmark application), Coremark** [38], Filebench [39], and Netperf [40]. Filebench was executed against local and CEPH block storage. The Noload test provides a baseline that we can compare with later tests. We gradually increase the number of VMs on the compute nodes with specific benchmark load on the VM instances.

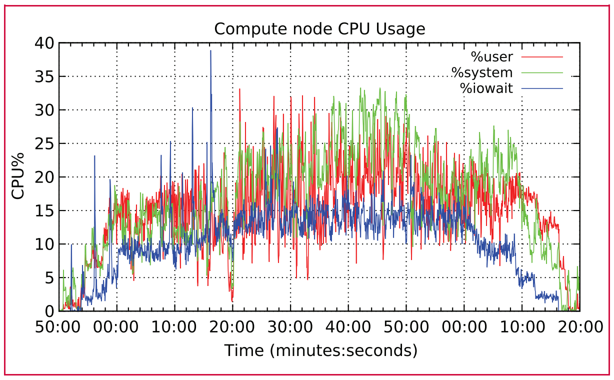

Figure 4

Performance measurement with simultaneous Coremark, Netperf, and Ephemeral load with 750 VMs on 24 compute nodes and three controller nodes. CPU usage of one compute node is illustrated.

This load puts stress on specific resources on the compute nodes, allowing us to find bottlenecks and improve performance. Provisioning under this type of load uncovered numerous issues with Libvirt and OpenStack.

Next, we ran benchmarks with the I/O Intensive Yahoo!** Cloud Serving Benchmark (YCSB) workload with Cassandra [41], a compute intensive workload with K-Means [42] as the Hadoop** workload, and the Daytrader [43, 44] Web Application workload for characterizing the performance of a J2EE application built around an online stock trading system. For YCSB, the load driver runs in a separate VM and generates the data set for the underlying NoSQL Cassandra database VMs. For K-Means, HiBench runs on the "name node" of the Hadoop cluster. It generates the data set on the underlying Hadoop instances and executes the K-Means clustering algorithm. For Daytrader, the client VM generates load against the WebSphere* Application Server VMs with DB2* as the database.

When *cbtool* receives a request to provision an application instance (AI), it instructs IBM Cloud OpenStack Services to create the multiple VM instances that belong to this AI. When the application running in these VMs is ready, the AI is considered to have been provisioned. Then *cbtool* invokes a script to generate the data set that is used by the workload driver during the run. After each run, the workload driver reports the collected metrics. The tool simultaneously runs multiple AIs with different workloads.

The experimental setup consisted of 24 compute nodes, each with 32 processors, 128 GB RAM and three controllers each with 4 CPUs and 16 GB RAM. **Figure 4** shows the CPU usage of one compute node out of 24 for an experiment with a mix of equal numbers of Coremark, Netperf, and Filebench VMs. A total of

750 VMs were provisioned in around 44 minutes. There is a wait time of 20 minutes when the load continues to run on the AIs. Finally, we de-provisioned the VM instances for the remainder of the run of around 30 minutes. Figure 4 shows the increasing CPU usage (system, user, and iowait) on the compute node as up to 32 VM instances are started and run the benchmark applications. The iowait is caused by the VM instances that are running Filebench, most of the system usage is caused by Netperf, and the user CPU usage is caused by Coremark. The three OPS nodes (not shown) were not heavily loaded. This experiment shows that IBM Cloud OpenStack Services can handle provisioning and deprovisioning requests under a heavy VM load. With full CPU utilization on OPS nodes and using the Noload image, we can provision the 750 VM instances in 25 minutes on the same setup.

Conclusion

We have shown how we used OpenStack and the support it provides to build a highly available and secure cloud that can meet the needs of an enterprise. It provides the flexibility to design an architecture that is pluggable and allows the selection of technologies such as Ceph and SDN-VE. The administrative model and granular security policies can be designed in a way that allows the cloud operator to manage key aspects of the offering while retaining the complete OpenStack experience for the private cloud user. We believe that the key to a cloud offering is to build a framework of continuous integration and distributed management.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of OpenStack Corporation, SoftLayer, Inc. (an IBM Company), Red Hat, Inc., Linus Torvalds, MySQL Corporation, Codership Oy, Pivotal Software, Vyatta, Inc., Oracle America, EDN Embedded Microprocessor Benchmark Consortium, Yahoo!, Apache Software Foundation, VMware, or The Open Group in the United States, other countries, or both.

References

1. C. Doukas, T. Pliakas, and I. Maglogiannis, “Mobile healthcare information management utilizing cloud computing and android OS,” in *Proc. IEEE Annu. Int. Conf. EMBC*, 2010, pp. 1037–1040.
2. C. Pascu, “An empirical analysis of the creation, use and adoption of social computing applications,” Inst. Prospective Technol. Studies, Seville, Spain, Publ. 7 23415 EN, 2008, pp. 1–92.
3. L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.
4. S. LaValle, E. Lesser, R. Shockley, M. Hopkins, and N. Kruschwitz, “Big data, analytics and the path from insights to value,” *MIT Sloan Manage. Rev.*, vol. 52, no. 2, p. 21, 2013.
5. R. Ananthanarayanan, K. Gupta, P. Pandey, H. Sarkar, M. Shah, and R. Tewari, “Cloud analytics: Do we really need to reinvent the storage stack,” in *Proc. Workshop HotCloud Topics Comput.*, 2009, vol. 9, p. 15.
6. O. Sefraoui, M. Aissaoui, and M. Eleuldj, “OpenStack: Toward an open-source solution for cloud computing,” *Int. J. Comput. Appl.*, vol. 55, no. 3, pp. 38–42, Oct. 2012.
7. V. Sobeslav and A. Komarek, “OpenSource automation in cloud computing,” in *Proc. 4th Int. Conf. Comput. Eng. Netw.*, 2015, pp. 805–812.
8. M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A view of cloud computing,” *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.
9. A. Corradi, M. Fanelli, and L. Foschini, “VM consolidation: A real case based on OpenStack cloud,” *Future Gener. Comput. Syst.*, vol. 32, pp. 118–127, Mar. 2014.
10. B. Kandukuri, B. Reddy, V. Ramakrishna Paturi, and A. Rakshit, “Cloud security issues,” in *Proc. IEEE Int. SCC*, 2009, pp. 517–520.
11. T. Mather, S. Kumaraswamy, and S. Latif, *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*. Newton, MA, USA: O’Reilly Media, Inc., 2009.
12. *SofLayer: About SoftLayer*. [Online]. Available: http://www.softlayer.com/about-softlayer.
13. *KVM*. [Online]. Available: http://www.linux-kvm.org/page/Main_Page.
14. *Ceph*. [Online]. Available: http://ceph.com/docs/master/start/intro/.
15. *OpenStack:Swift*. [Online]. Available: http://docs.openstack.org/developer/swift/.
16. *Chef*. [Online]. Available: http://docs.chef.io/.
17. *OpenStack: Jenkins*. [Online]. Available: http://docs.openstack.org/infra/system-config/jenkins.html.
18. *Mirantis, Logic Setup for Controller Nodes*. [Online]. Available: http://docs.mirantis.com/fuel/fuel-3.2.1/reference-architecture.html#controller-nodes.
19. *OpenStack: Pacemaker*. [Online]. Available: http://docs.openstack.org/high-availability-guide/content/ch-pacemaker.html.
20. G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*. Reading, MA, USA: Addison-Wesley, 2001.
21. *OpenStack: MySQL*. [Online]. Available: http://docs.openstack.org/high-availability-guide/content/s-mysql.html.
22. M. Rostanski, K. Grochla, and A. Seman, “Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ,” in *Proc. FedCSIS*, 2014, pp. 879–884.
23. *OpenStack: RabbitMQ*. [Online]. Available: http://docs.openstack.org/high-availability-guide/content/s-rabbitmq.html.
24. B. Turakhia, *RabbitMQ vs Apache ActiveMQ vs Apache qpid*. [Online]. Available: http://bhavin.directi.com/rabbitmq-vs-apache-activemq-vs-apache-qpid/.
25. IBM Corporation, *SDN-VE*. [Online]. Available: http://www.redbooks.ibm.com/redbooks/pdfs/sg248203.pdf.
26. M. Mahalingam, *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*. [Online]. Available: https://tools.ietf.org/html/rfc7348.
27. *OpenStack: Neutron*. [Online]. Available: https://wiki.openstack.org/wiki/Neutron.
28. *Open DOVE Management Network*. [Online]. Available: https://github.com/opendaylight/opendove.
29. *OpenStack: Floating IP Configuration*. [Online]. Available: http://docs.openstack.org/admin-guide-cloud/content/nova-associate-public-ip.html.
30. *OpenStack: Chapter 7, Network Node*. [Online]. Available: http://docs.openstack.org/icehouse/training-guides/content/operator-network-node.html#operator-openstack-networking-use-cases.
31. *OpenStack Superuser, OpenStack User Survey Insights*: Nov. 2014. [Online]. Available: http://superuser.openstack.org/articles/openstack-user-survey-insights-november-2014.

32. S. Weil, L. Andrew, B. Scott, and C. Maltzahn, “RADOS: A fast, scalable, and reliable storage service for petabyte-scale storage clusters,” in *Proc. ACM PDSW*, 2007, pp. 35–44. [Online]. Available: http://www.pdsi-scidac.org/SC07/resources/rados-pdsw.pdf.
33. V. Galinanes, *Ceph, Development: Final Report*. [Online]. Available: http://tracker.ceph.com/projects/ceph/wiki/Final_report.
34. S. Weil, *Storage Tiering and Erasure Coding in Ceph*. [Online]. Available: http://www.socallinuxexpo.org/scale/13x/presentations/storage-tiering-and-erasure-coding-ceph.
35. M. J. Skok, *Breaking Down the Barriers to Cloud Adoption*. [Online]. Available: http://mjskok.com/news/news-breaking-down-barriers-to-cloud-adoption.
36. *Open Automated Compliance Expert Markup Language (O-ACEML) Standard*. [Online]. Available: https://www2.opengroup.org/ogsys/catalog/c111.
37. *Cloud Rapid Experimentation and Analysis Toolkit*. [Online] Available: https://github.com/ibmcb/cbtool.
38. *Coremark*. [Online]. Available: https://github.com/ibmcb/cbtool/tree/master/scripts/coremark.
39. *Filebench*. [Online]. Available: https://github.com/ibmcb/cbtool/tree/master/scripts/filebench.
40. *Netperf*. [Online]. Available: https://github.com/ibmcb/cbtool/tree/master/scripts/netperf.
41. *Cassandra YCSB*. [Online]. Available: https://github.com/ibmcb/cbtool/tree/master/scripts/cassandra_ycsb.
42. *KMeans Hadoop*. [Online]. Available: https://github.com/ibmcb/cbtool/tree/master/scripts/hadoop.
43. *Daytrader—A More Complex Application*. [Online] Available: http://geronimo.apache.org/GMOxDOC22/daytrader-a-more-complex-application.html.
44. *GitHub—CBTool, IBM Daytrader*. [Online] Available: https://github.com/ibmcb/cbtool/tree/master/scripts/ibm_daytrader.

Received May 20, 2015; accepted for publication June 16, 2015

Sandy Cash *IBM Cloud, Raleigh, NC 27703 USA (lhcash@us.ibm.com)*. Mr. Cash is an architect and software engineer who has worked extensively in cloud computing, information technology service management, and information technology operations. His projects have included the web infrastructure for the Sydney Olympics, Wimbledon, and the Masters Golf Tournament; multiple private cloud engagements for major enterprise customers; SmartCloud* Enterprise; IBM Cloud OpenStack Services; and IBM Bluemix*. Mr. Cash is interested in automated lifecycle management for virtualized and cloud services.

Vinit Jain *IBM Cloud, Austin, TX 78758 USA (vjain@us.ibm.com)*. Mr. Jain is the architect responsible for the overall architecture of IBM Cloud OpenStack Services for dedicated compute. He is an expert in the area of networking and has led IBM teams in delivering innovations in the areas of cloud solutions, high availability, server virtualization, and enterprise networking technology. He is an IBM Master Inventor with more than 50 issued patents. Mr. Jain has degrees in computer science from IIT-Delhi and the University of Maryland.

Liang Jiang *IBM Cloud, Austin, TX 78757 USA (ajiang@us.ibm.com)*. Mr. Jiang is a Senior Technical Staff Member in the Cloud Infrastructure Services organization. He received a B.S. degree in computer science from Shandong University in China in 1994 and an M.S. degree in computer science from Fudan University in China in 1997. After working for Sun Microsystems for three years, he moved to the United States in 2000 and joined the IBM Systems and Technology Group at Austin, where he worked on the AIX* operating system. After spending 14 years in AIX services and base kernel development, he joined Cloud Infrastructure Services development, where he works on Kernel-based Virtual Machine virtualization and OpenStack object storage.

Alexei Karve *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (karve@us.ibm.com)*. Mr. Karve is a Senior Software Engineer working on cloud computing technologies—OpenStack performance, deduplication, peer-to-peer data transfer, containers, and virtualization. He joined IBM in May 1993 and has worked on several projects in systems management and system software. Mr. Karve has degrees in computer science and engineering from IIT-Kanpur and BITS Pilani.

Jay Kidambi *IBM Cloud, San Jose, CA 95120 USA (jkidambi@us.ibm.com)*. Mr. Kidambi is a Technical Lead and Software Development Manager in the Cloud Networking group. He leads the development of Software Defined Networks-Virtual Environments, which is a multi-hypervisor network virtualization technology based on overlays, and development of 5000 V, which is a third-party virtual switch for the VMware vSphere** platform. Prior to joining IBM, he held various technical leadership roles in the areas of networking and virtualization at Blade Network Technologies and at Hewlett Packard. Mr. Kidambi has a master’s degree in electrical and computer engineering from the University of California, Davis.

Michael Lyons *IBM Cloud, Austin, TX 78758 USA (mlyons@us.ibm.com)*. Mr. Lyons is a Senior Technical Staff Member working on Cloud Storage Infrastructure Services. He received a B.S. degree in computer science and mathematics from Duke University in 1985 and an M.B.A. degree from the University of Texas in 1989. He subsequently joined IBM, where he has worked on operating systems and cloud services. Mr. Lyons has received multiple IBM Outstanding Technical Achievement awards and is author or coauthor of four patents.

Thomas Mathews *IBM Cloud, Austin, TX 78758 USA (tmathews@us.ibm.com)*. Mr. Mathews is a Distinguished Engineer leading the overall technical strategy and architecture for the Cloud Infrastructure Services software development organization within the IBM Cloud Division. He is currently focusing on developing OpenStack-based cloud services on SoftLayer. Mr. Mathews has extensive technical experience and background based in OpenStack and has been responsible for the architecture and development of numerous OpenStack-based products offered by IBM. He also has extensive systems and infrastructure knowledge grounded in years spent designing and developing system software for IBM’s UNIX** business. Mr. Mathews joined IBM in 1985 after receiving a B.S. degree in computer science from the University of Texas at El Paso in 1984.

Shawn Mullen *IBM Cloud, Austin, TX 78758 USA (smullen@us.ibm.com)*. Mr. Mullen has worked for a variety of computer companies and organizations, including the last 25 years for IBM. His current role is Cloud Security Architect for the Cloud Infrastructure Services organization, leading the architecture and development of security products and technology. His previous role was security architect for Power Systems*, where he owned and developed the security roadmap, product strategy, and architecture, and where he led the development of products such as PowerSC*. Mr. Mullen has played leadership roles in standards organizations, to reach consensus with competitors and customers. He has more than 100 granted U.S. patents, and he graduated from University of New Mexico in 1986.

Matt Mulsow *IBM Cloud, Austin, TX 78758 USA (mamulsow@us.ibm.com)*. Mr. Mulsow is the lead architect for the public cloud in the Cloud Infrastructure Services organization. He leads a team of developers that have rapidly delivered an OpenStack-based public cloud available as a beta cloud today in IBM Bluemix. Mr. Mulsow joined IBM in April 2015 as one of the early leads in the cloud organization. He has played a key technical role in several projects including Jumpgate, the SoftLayer-to-OpenStack translation layer, and Metis, the database-as-a-service offering, for which he functioned as team lead. Mr. Mulsow graduated from Texas Tech University in 2008 with degrees in electrical engineering and computer science.

Niraj Patel *IBM Cloud, Austin, TX 78758 USA (nirajdp@us.ibm.com)*. Mr. Patel is a Technical Lead for IBM Cloud OpenStack Services for dedicated compute. He is an expert in the area of automation and orchestration of the cloud deployments. In his previous roles in IBM, he has led teams to deliver innovations in the areas of server virtualization, server management, systems management northbound ecosystems, and APIs. He received a B.S. degree in computer science from Mumbai University, India, in 1994 and an M.S degree in computer science from North Carolina A&T State University in 2003. Mr. Patel has received multiple IBM Outstanding Technical Achievement Awards and is author or coauthor of more than 10 patents.