If you have tried OpenStack – as recommended in the second article of this series [1] – you may be thinking: If you can roll out metal as a service (MaaS) and Juju OpenStack within a few hours, the solution cannot be too complex. However, although the MaaS-Juju setup as described simplifies requirements in some places, it initially excludes many functions that a production OpenStack environment ultimately needs.

In the third part of this series, I look at how the lessons learned from a first mini-OpenStack can be transferred to real setups in data centers. To do so, I assume the deployment of a production environment with Juju and MaaS, accompanied by the unfortunate restriction that the solution costs money as of the 11th node that MaaS needs to manage. Of course, it would also be possible to focus on other deployment methods. As an alternative, Ansible could be used on a bare-bones Ubuntu to roll out OpenStack, but then you would have the task of building the bare metal part yourself. Anyone who follows this path might not be able to apply all the tactics used in this article to their setup. However, most of the advice will work in all OpenStack setups, no matter how they were rolled out.

Matching Infrastructure

The same basic rules apply to OpenStack environments as for any conventional setup: Redundant power and a redundant network are mandatory. When it comes to network hardware in particular, you should plan big rather than attempting to scrimp from the outset. Switches with 48 25Gb Ethernet ports are now available on the market (Figure 1), and if you want a more elegant solution, you can set up devices with Cumulus [2] and establish Layer 3 routing. Each individual node uses the Border Gateway Protocol (BGP) to distribute the routes to itself on the entire network. This fabric principle [3] is used, for example, by major providers such as Facebook or Google to achieve an optimal network setup. If you are planning a production OpenStack environment, you should
provide at least two availability zones, which can mean two rooms within the same data center, but located in different fire protection zones. Distributing the availability zones to independent data centers would be even better. This approach is especially important because an attractive Service Level Agreement (SLA) can hardly be guaranteed with a single availability zone.

Matching Hardware

One challenge in OpenStack is addressing hardware requirements. A good strategy is to allocate the required computers to several groups. The first group would include servers running controller services, which means all OpenStack APIs as well as the control components of the software-defined network (SDN) and software-defined storage (SDS) environments. Such servers do not need many disks, but they must be of high enough quality to support 10 drive writes per day (DWPD) without complaint. Add to this the hypervisor nodes on which virtual machines (VMs) run: They need a good portion of CPU and RAM, but the storage media are unimportant.

Whether the planned setup will combine storage and computing on the same node (hyper-converged) is an important factor. Anyone choosing Ceph should provide separate storage nodes that feature a correspondingly large number of hard disks or SSDs and a fast connection to the local network. CPU and RAM, on the other hand, play a subordinate role. Finally, small servers that handle tasks such as load balancing do not require large boxes. The typical pizza boxes are usually fine, but they should be equipped with fast network cards.

Modular System

Basically, the recommendation is to define a basic specification for all servers. It should include the type of case to be used, the desired CPU and RAM configuration, and a list of hard disks and SSD models that can be used. Such a specification makes it possible to build the necessary servers as modular systems and request appropriate quotations. However, be careful: Your basic specification should also take into account eventualities and give dealers clear instructions in points of dispute. If you want to build your setup on UEFI, the network cards may need different firmware from those normally installed by the vendor. If you commit the dealer to supplying the servers with matching firmware for all devices, you can save a huge amount of work. However, if you need to install new firmware on two cards per server, it will take at least 20 minutes per system. Given 50 servers, this translates to a huge amount of time.

No Cloud Without a Net

Once the topics of hardware and infrastructure have been dealt with successfully, OpenStack planners then face the first really unappetizing topic: SDN. In traditional setups in which virtualization is the focus, vendors have hitherto worked with pretty ugly hacks, such as messing around with virtual LANS (VLANs) at the hardware level and then connecting VMs with bridges to the corresponding interfaces at the host level. This principle no longer works in clouds and is painfully apparent in public clouds, where it is inconceivable for you to create VLANs provider-side on customer request and then use your own virtual network in the cloud. The magic word for clouds is “decoupling,” wherein the network hardware of the setup is decoupled from its software. The physical network setup is completely flat; look for VLANs and similar mechanisms in vain. The setup described earlier with Layer 3 routing is a move in the same direction.

The hardware functions are replaced by software: SDN solutions are part of the setup and, in the background, configure virtual extensible LANS (VxLANs) or generic routing encapsulation (GRE) tunnels so that VMs from different customers can run on the same servers without ever seeing each other’s traffic. Cloud users click together their entire virtual network topology as they need it. After configuring the setup for the first time, there is no additional overhead for the vendor.

Open vSwitch Default

In the first two articles of the series [1] [4], SDN was not mentioned explicitly because it comes as part of the package with the Juju-MaaS setup. OpenStack Neutron, which manages SDN networks on the OpenStack side, uses Open vSwitch in the default configuration. Because the typical Open vSwitch setup does not have a central switching point, a good deal of overhead traffic takes
place between the individual network nodes (e.g., for the ARP protocol). The approach is adequate for small setups, like the one in the second article of the series, but it can create a problem for large environments with more than 20 nodes. Various vendors from the OpenStack community have recognized the problem and now offer alternatives. The existing solutions are divided into two categories: Most approaches rely on Open vSwitch, but they add additional components to compensate for its deficits. The other camp counters with completely independent solutions – first and foremost OpenContrail [5] by Juniper.

**Many Question Marks**

The technically most elegant approach is undoubtedly OpenContrail, which relies on open, standardized protocols such as BGP and multi-protocol label switching (MPLS), thus enabling a smooth and well-integrated SDN in OpenStack. In practice, however, OpenContrail fails because of its enormous complexity, and earlier OpenContrail versions did not perform well in terms of stability. Finally, OpenContrail cannot be used with just any old OpenStack cloud because it imposes strict requirements on the setup, and packages are not available for all standard distributions. Anyone who is currently building an OpenStack cloud is likely to turn to a solution based on Open vSwitch. The bad news is that a clear recommendation for a product is simply impossible. The selection of SDN solutions is huge. MidoNet [6] (Figure 2) by Midokura, VMware’s NSX [7], and various other products are admin favorites right now. In the end, your only option is to evaluate different solutions and take all factors into consideration. The question is whether an SDN solution suits the intended deployment scenario without problem. Also, the features offered are sometimes very different. Factors such as stability and the ability to upgrade from one major version of the SDN solution to another also contribute to the worries. Last but not least, price is important: Almost all prominent SDN solutions cost money, burdening your budget, and require time to evaluate.

The importance of the initial decision in favor of a particular SDN environment cannot be overstated, because it cannot be replaced in a running OpenStack environment. Solutions such as MidoNet, NSX, or PLUMgrid typically store their configurations in a custom format in a database that other services cannot parse. Once you’ve rolled out MidoNet, you need to live with it – or rebuild the cloud based on another SDN environment and accept the fact that you lose a customer’s entire network configuration.

**Fewer Problems with SDS**

Like networks, storage presents new challenges in the cloud. In conventional setups, centralized storage connected to the rest of the installation via NFS, iSCSI, or Fibre Channel is quite common. Nothing is wrong with these kinds of network storage, even in clouds, except that they do not usually scale well and come with large price tags. In addition to storage for the VMs, a second storage service is usually required in clouds for storing binary objects (object storage). At least two protocols are available: OpenStack Swift and various Amazon S3 look-alikes. From an admin’s perspective, it is desirable to offer both storage types – storage for VMs and for objects – using the same storage service. Typical storage area networks (SANs) do not offer object storage, so you must find another solution. In contrast to the SDN scenario, a de facto standard has established itself in recent years: Ceph [8] dominates large OpenStack clouds in virtually every respect. Red Hat has done much in recent years to make the solution known and popular: In addition to numerous new features, much has also been done with regard to stability. Additionally, Ceph is very versatile: Via the RBD interface, it acts as back-end storage for VMs thanks to its native connection to OpenStack. With the aid of the RADOS gateway, which is now known as Ceph Object Gateway, it uses HTTP(S) to deliver binary objects in line with the Amazon S3 or OpenStack Swift protocol. Because Red Hat has shortened the CephFS feature list, even the Posix-compatible file system is available as a kind of replacement for NFS.

**Seamlessly Scalable**

Ceph’s greatest advantage is still cluster scalability virtually without limits during operation. When space runs out, you simply add an appropriate number of additional servers to create more space.
Because Ceph now supports erasure coding, the old rule of thumb “gross times three equals net” no longer applies: The overhead for replication can be reduced significantly, but if you do use erasure coding, you need to be prepared to compromise: The recovery of failed nodes takes significantly longer and generates significant load on the CPU and RAM of the storage nodes.

The biggest advantage of Ceph is undoubtedly that Red Hat provides the software free of charge and ready for different distributions. On standard Ubuntu systems, packages provided directly by the vendor can be installed and used. As explained in the second article of the series, Ubuntu’s Autopilot based on MaaS and Juju is the tool of choice for Ceph deployment: If you select Autopilot Ceph and then assign the Ceph roles to appropriate computers, the result at the end of the setup will be a working Ceph cluster that is connected directly to your OpenStack installation (Figure 3).

**Ceph Tips and Tricks**

If you want to build an OpenStack solution, you would do well to opt for Ceph initially because it covers most daily needs and can be optimized through customization. A classic approach, for example, is storing the journals for individual hard disks in Ceph – object storage devices (OSDs) in Ceph-speak – on fast SSDs, which results in good performance for write operations in the cluster and considerably reduces the time required for write operations (i.e., latency).

The factors already mentioned in the “Matching Hardware” section need to be pointed out again: Ceph is not well suited to operations in a hyper-converged system, because the storage nodes need to be independent and they should be able to talk with the OpenStack nodes through high-bandwidth network paths so that the failure of a node and the resulting network traffic do not block the network.

Many admins use their own servers for the monitor servers (MONs), the guarded dogs that monitor the cluster and enforce its quorum; however, MONs require virtually no resources, so they can run on the OSD nodes, making it possible to save money.

On the other hand, you should not scrimp when it comes to equipping the cluster with disks. Many admins succumb to the temptation of using cheaper consumer devices instead of enterprise-grade SATA or SAS disks, which can turn out to be a stumbling block in everyday life. Consumer disks differ from their enterprise colleagues, especially in terms of failure behavior.

A dying consumer disk tries to keep working for as long as possible, whereas an enterprise disk declares itself dead with major malfunctions, completely denying service. In a Ceph cluster, you want this enterprise behavior: Ceph needs the disk to fail completely before it notices the defect and implements the many redundancy measures. A disk that is only slightly broken will mean hanging writes, in the worst case, and will make troubleshooting more difficult.

When it comes to Ceph and OpenStack, not all that glitters is gold because the combination is fraught by technical problems: If you run a database in a VM on Ceph, for example, you are unlikely to be satisfied by the performance. The underlying Controlled Replication Under Scalable Hashing (CRUSH) algorithm in Ceph is extremely prone to latency, and small database writes unfortunately make this more than clear.

The problem can only be circumvented by providing alternatives to Ceph, such as local storage or Fibre Channel-attached local storage. However, this kind of setup would easily fill a separate article, and I can only point out here that such approaches are possible.

**High Availability**

Now that I’ve reached the end of this OpenStack series, I’ll look at a topic that is probably familiar to most admins from the past. After all, high availability (HA) plays an important role in practically every IT setup. Redundancy has already been mentioned several times in the three articles – no wonder: Many components of a classic OpenStack environment are implicitly highly available. Ceph, for example, takes care of HA itself, in that several components need to fail at the same time to impair the service.

SDN is not quite as clear cut: Whether and how well the SDN layer handles HA depends to a decisive extent on the solution you choose, but practically all SDN stacks have at least basic HA features and can field

![Figure 3: Ubuntu’s Autopilot is designed for Ceph and installs a complete Ceph cluster at the push of a button. No extra installation work is needed for storage.](image-url)
a gateway node failure. You can then move the networks assigned to the failed node to other nodes.

Is everything great in terms of HA, then? Well, not quite, because the services that define OpenStack itself need HA. In the meantime, most OpenStack services are designed such that multiple instances can exist in the cluster. However, this does not work with a central component: Although the API services can be started multiple times, it is simply impossible to handle many IP addresses from a client perspective. The usual approach is to deploy a load balancer upstream of the API services that distributes incoming requests to its various back ends. Of course, it must also be highly available: If it is a normal Linux machine, a cluster manager like Pacemaker offers a useful approach. In the case of commercial products, the manuals reveal how HA can work.

MySQL and RabbitMQ

MySQL and RabbitMQ also are not to be forgotten in terms of HA. Although they are not directly part of OpenStack, they are important parts of virtually any OpenStack installation. For MySQL, HA can be achieved either by means of a cluster setup and a special load balancer like MaxScale (Figure 4), MySQL Group Replication (a new feature as of version 5.5), or Galera and its integrated clustering.

With RabbitMQ, the facts look less encouraging: The service has a clustered mode, but it has proven unreliable on several occasions. As an alternative to RabbitMQ, Oqipd could be a candidate, but it also does not offer a convincing HA history. In terms of the Advanced Message Queuing Protocol (AMQP), it is – at least initially – more sensible to monitor the respective service (typically RabbitMQ) and alert the admin via a corresponding monitoring system. Alternatively, an appropriate HA setup can be implemented via a service IP and Pacemaker – even if this means taking on additional complexity in the form of Pacemaker.

HA for VMs

One of the big criticisms aimed at OpenStack is the lack of a feature for deploying high-availability VMs at the OpenStack level (i.e., the failure of a hypervisor node does not automatically cause OpenStack to restart the VM on another host). But not because the OpenStack developers would be unable to implement the appropriate functionality. However, they pursued a different approach and pointed out that a virtual environment in OpenStack must be designed in such a way that it can cope with the failure of individual VMs without self-destructing. Ultimately, the discussion is rather ideological rather than technical. Nevertheless, users should keep it in the back of their minds.

Conclusions

At the end of this series of OpenStack articles, one thing is clear: Gaining an initial overview of the OpenStack components is not particularly difficult; nor is it complicated to create a small OpenStack environment based on MaaS and Juju or other tools. However, it is a real challenge to configure and operate OpenStack in the data center in such a way that the provider and the customer are given proven added value.

While planning your platform, you must answer central questions about deployment, the OpenStack distribution you use, SDN and SDS tools, and everyday maintenance tasks. This initial investment pays off later with an OpenStack environment that is up and running and lets you support many customers without the hassle that would result outside of a cloud. The path is long and stony, but in the end, genuine added value lies at the end of the rainbow.

---

**Info**


[8] Ceph: (http://ceph.com)

---

**The Author**

Martin Gerhard Loschitz is Head of Cloud with innovo Cloud, where he focuses on high availability, distributed storage, and OpenStack.

---