Lectures On Cloud Computing with OpenStack and Amazon Web Service

by Prof. Dr. Harald Richter

Foreword

It is the hope that this script will be a good entry into the subsequent subject. However, it is not a book. It shall only free the audience from writing-up by hand, so that listeners can focus better on the matter. Each student has different strong and weak sides. Therefore, he/she will make personal remarks and explanations at different points in the script as needed. The document at hand will help him or her to do this with ease.

Neither the script nor the web can substitute the visit of the lectures. Many times, it has been evident that autodidactic learning of an extensive subject only by means of written documents, by the web or by videos requires much more time and engagement than visiting the lectures on a regular basis, especially if a personal contact during the lectures and the practical trainings is made possible.

Part I - Introduction into Cloud Computing

1 What is Cloud Computing (CC)?

- Historically, CC evolved -among others roots- from so-called Utility Computing

1.1 First Utility Computing

Def.: In Utility Computing, a service provider allows its customers to use IT resources in the provider's data centers via the Internet.

Example: IT resources are computing power, data storage, networking, operating systems and software applications in a computer cluster.

Def.: An IT resource is called web resource if it is accessed via http or REST requests. Each web resource is identified by an Uniform Resource Identifier (URI) which is unique for it.

- Users are renting web resources on a pay-as-you-go basis, similar as gas or electricity

Def.: Pay-as-you-go means that only the consumed resources such as the compute time or the bytes on a disk storage have to be paid.

- Utility Computing and thus CC have the following features:

1.) Renting of web resources is the common business model

2.) Scalability of web resources is easily possible

   - all provider's services appear to be 'infinite' to the users
   - users can request dynamically more or less resources via the Internet
   - user requests are satisfied within seconds or at most minutes

3.) Smart metering of web resources is provided
   - consumed resources are precisely metered and billed according to their effective utilization

4.) Automation of management of web resources is made possible
   - System administrator tasks are highly automated and accomplished by simple user actions via the Internet

- The aforementioned technologies are modern and effective means for the provisioning, management and usage of web resources

1.2 The New Features of Cloud Computing (CC)

- On top of Utility Computing, CC has put several new features, such as auto-scaling of web resources, for example

Def.: Auto-scaling means that the number of web resources, such as VMs e.g., are automatically adjusted due to their utilization. The goal of auto-scaling is to avoid overload of a web resource.

- Over the last decade, CC evolved to many different services

- **A comprehensive definition of CC stems from the U.S. National Institute of Standards, NIST:**

Def.: „Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.“

- **This definition is good, but not clear enough for a CC novice**
- **For a more intuitive definition, it is better to ask the following two question:**
  - **Does the CC user engage another company called Cloud Service Provider (CSP) to satisfy his IT needs?**

or

  - **Is a so-called cloud operating system installed on user computers at his own premise?**

- **The answers to that questions decide about the flavor of CC the user has**
- **Furthermore, there are only a few CSPs of importance and only a few cloud operating systems which are widespread**

Example: Amazon Web Service (AWS) is the most important commercial CSP, while OpenStack is the most prominent open-source cloud operating-system.

- **Please note that most of the Internet-accessible services are <u>not</u> CC**

Example: No CC services are delivered by Facebook, Twitter, YouTube, Uber, Instagram, WhatsApp, Dropbox, BitBucket or GitHub.

Example: Apple‘s iCloud provides only a few and only very simple CC services

- **However, many of the services of the providers listed above are implemented internally using CC**

1.3 Cloud Service Provider (CSP)

- **A CSP is a company that has multiple computing or data centers around the world**

Comment: We talk explicitly about a data center if the storage of data is the main purpose, such as it is the case in big data. We talk about a computing center if the processing of data is the main purpose, such as via supercomputers. If big data is processed in supercomputers both notions coincide.

- **A CSP provides IT services to paying customers via the Internet such as storage, databases, computing, data distribution, ...**

Example: The big CSPs are Amazon, Microsoft, IBM and Google. Smaller CSPs are SAP and German Telecom.

- **A CSP‘s customer is typically a company, but private users exist as well**
- **Amazon is the inventor of CC and still the market and technology leader in CC, followed by Microsoft Azure Cloud, IBM Cloud, Google Cloud Platform and SAP HANA cloud**

Example: AWS offers about 65 services from all fields of IT that are in production, i.e. out of beta tests. OpenStack offers about 46 services from many IT domains.

- **CSPs use internally either proprietary or public cloud OSes for their computing/data centers to implement services for their customers**
- **All of them have thin-provisioning as their business-model**

1.3.1 Thin Provisioning

- **Typically, a big CSP has millions of customers he satisfies by means of a trick called thin provisioning**
- **Thin provisioning means that a CSP has much more registered customers than he can actually satisfy simultaneously**
- **This is possible because CSP services are world-wide accessible, and because there is always a whole continent on the planet where users are sleeping and thus not active in the CSP cloud**
- **Additionally, not every user needs 24h/7d CC services**

⇒ **less computers can serve more customers by over-committing the CSPs physical hardware**

- **Thin provisioning allows for high hardware utilization and is thus financially very attractive for a CSP**

- **Thin provisioning is also useful for the user, because the resulting price for using it is low**

Example: A so-called virtual machine (VM) with Gigaflops of computing power and Terabytes of disk storage, as well as other virtual IT resources is available at AWS auctions for only 10 $ per month.

- **Thin provisioning and its resulting price/performance ratio is one of the reasons why CC is such widespread**
- **Thin provisioning is implemented in software by so-called multi-tenancy software**

Comment: See extra chapter about multi-tenancy software.

- **Depending on the answers to our previous two questions, the CC features are as described below**

1.4 CC via a CSP (not via own cloud OS)

Def.: CC via a CSP means for a user to have Internet access to a so-called public or hybrid cloud at various levels of service. These levels are Infrastructure-as-a Service, Platform-as-a-Service and Software-as-a-Service.

1.4.1 Public Cloud

Def.: A public cloud provides for users general IT resources via the Internet, such as computing capability, storage capacity, networking management, operating system support and sector-specific software-applications.

- **Typically, a public cloud is geographically distributed over dozends of data/computing centers of the CSP**
- **The exact location where a specific web resource is handled can be unknown to the user and is typically also not important, with exception of data security**

1.4.2 Private Cloud

- **For the definition of a hybrid cloud, we need to define a private cloud first**

Def.: A private cloud is a locally concentrated computer cluster which executes a cloud OS. The cloud OS allows for numerous IT management functions initiated by a central admin console.

Comment: Admin = Cloud Administrator.

1.4.3 Hybrid Cloud

Def.: A hybrid cloud is the combination of a private with a public cloud. Sensitive data are kept on the user premise in a private cloud, while public cloud services are though accessible via the Internet.

- **Hybrid clouds extend the user‘s on-premise features significantly by ‘infinite‘ CSP resources and their numerous services**
- **A drawback of hybrid clouds is their lower data security compared to a private cloud**

1.4.4 Infrastructure-as-a Service (IaaS)

- **IaaS is the basic level a CSP can provide for a user as general service offer**
- **With IaaS, the CSP provides for customer PCs, servers, hard drives, switches, routers or even whole data/computing centers**

Example: Amazon‘s EC2 in AWS delivers IaaS to customers.

- **Typically, IaaS resources are all virtual, i.e. the functionality of PCs or servers is emulated by software**

⇒ **Virtual IaaS resources do not really exist, they are just code on the CSP‘s hardware**

- **Furthermore, each CSP bundles a set of virtual resources into a so-called virtual machine (VM)**
- **The user rents one or many VMs from a CSP**
- **On special request, also real resources can be sometimes obtained as pieces of CSP hardware**
- **Such physical resources are located in one or more CC center**
- **Alternatively to VMs, also so-called Linux Containers (LXCs) can be rented**

- While demand for real resources is very rare, LXCs are getting more and more attention
- In IaaS, all resources, i.e. virtualized ones, LXCs, or physical ones are maintained and administered by the user without help of the CSP

1.4.4.1 Resource Administration by the User

- Furthermore, the user has to install so-called guest operating systems on top of his VMs, together with the software development frameworks he needs, such as Eclipse or Microsoft Visual Studio for example
- The installation of a guest OS is not needed because it is always the undelying host OS that mimics to be an individual guest OS
- Additionally, he installs application software and cares for all updates and licenses by himself
- Furthermore, the user can increase or decrease the number of virtual and LXC resources easily and quickly via the internet
- After all installations, he has a set of resources for himself and his project
- As a result, in IaaS the user has full control over his virtual or LXC resources and can determine their type and features arbitrarily during creation

1.4.4.2 The Role of the CSP

- The CSP meters and bills the actually consumed computing, storage and network resources (pay-as-you-go)

- Furthermore, the CSP ensures also a high resource-availabilty of more than 99,99 %
- This implies automatic backups made by the CSP, distributed file systems, redundant PCs and servers, as well as redundant power supplies and disjoint geographical locations of computing centers

1.4.4.3 Nested Virtualization

- On top of his virtual IT infrastructure, a user can install an own cloud OS, thereby creating a cloud inside of a cloud
- This is called nested virtualization
- Nested virtualization is possible with IaaS but not with PaaS or SaaS because VMs are behaving as if they were real PCs or servers

1.4.5 Platform-as-a-Service (PaaS)

- PaaS is a more abstract service of the CSP for his customers which does not focus on the provisioning of bare VMs or LXCs
- In PaaS, a ready-to-use IT infrastructure is available for customers, with all guest OSes and software development frameworks already installed
- The CSP cares also for all switch and router settings, for public and private IP and MAC addresses, for firewalls and software updates
- The user can focus instead on installing and running his own sector-specific applications, or he can focus on writing his own software

Example: Microsoft Azure provides PaaS for software developers, not for arbitrary users.

- Typically, PaaS has auto-scaling of IT resources
- With auto-scaling, the user's IT resources depend on the number of active applications started by him or on the number of software developers that are currently active in his project
- In PaaS, the user knows nothing about the underlying physical cloud and its software and also nothing about the effectively consumed resources
- At the end of the month, the user is billed on basis of a „flat rate", for example

1.4.6 Software-as-a-Service (SaaS)

- SaaS is the most abstract, but also the most expensive service level
- In SaaS, a CSP provides for everything and maintains even the user's sector-specific applications
- The user has to care for nothing, for example also not for his software licences

Example: The Oracle Database Cloud or Google Docs are providing SaaS.

- Users at the PaaS or SaaS level use a CSP's cloud without seeing anything from it
- Sometimes, these users claim that they are doing CC which is wrong in a strict sense, because they are only using CC services from a CSP but not its virtual infrastructure
- Please note, although Google is a commercial CSP its "Docs" SaaS is free at first sight

- However at 2nd sight, it is clear that Google deals with your personal data you have to give at registration
- Additionally, it is clear that Google has a connection to the U.S. National Security Agency (NSA) and can transmit your data to NSA at any time, as all other U.S.-based CSPs may do
- Only in case that it is assured by contract that your data are stored and processed in Germany, German IT protection laws will apply
- But this is also not really a guarantee for full data security because there are possibilities to bypass German laws

1.4.7 Special Access-Offers

- In addition to public or hybrid clouds exist Community Clouds, Virtual Private Clouds and Multi Clouds as special access offers from some CSPs
- These offers will be explained subsequently

1.4.7.1 Community Cloud

- A community cloud is a semi-public cloud run by a specialized CSP
- The community cloud has restricted access for users, only cloud members are entitled to register

Example: The cloud of the GWDG computing center in Göttingen is a community cloud.

- The advantage of a community cloud is the cost-sharing between its members compared to a private cloud each member would have for himself otherwise

1.4.7.2 Virtual Private Cloud

- A virtual private cloud (VPC) is the attempt of some CSPs to deliver more data security to their customers
- The CSPs guarantees that a VPC can only be accessed via a VPN connection that is additionally secured by IPsec

Comment: A VPN connection establishes a virtual private network which is as a tunnel between the user PC and a VPN gateway.

Comment: IPsec allows for data encryption on ISO layer 3. It is used here o encrypt data in the VPN.

- Furthermore, the CSP adds firewalls around the customer‘s resources

Example: Amazon‘s Virtual Private Cloud is just the classical AWS EC2 service with an additional IPsec-based VPN-access, together with Linux security groups and access lists for the VMs.

1.4.7.3 Multi Cloud

- A multi cloud allows to access services from multiple CSPs via a single interface
- Its API creates a federation of clouds

Example: A multi cloud can bundle the services from AWS and from Google Cloud Platform in one API.

- The advantages of a multi cloud are for a user:
  - He is not fixed to one CSP
  - He has a broader offer of services
  - He has 100 % availabilty of at least some services because no two CSPs will collapse at the same time

1.5 CC with own Cloud OS (not via CSP)

- If a cloud OS is installed on a proprietry computer cluster, an own cloud is obtained
- The cloud-OS transforms each ordinary computer cluster into a fully-fletched cloud
- The service level an own cloud OS typically provides for its users is IaaS
- The cloud OS can be either open source or commercial software

Example: While OpenStack is the dominant open-source cloud OS, IBM Cloud has the same role for a commercial cloud OS. The private cloud versions of Microsoft Azure and the SAP cloud platform are exsiting as well but less important compared to IBM cloud.

- A private cloud is devoted to its owning organization or company only
- Typically, a private cloud is located on the customer‘s premise

- Users who are running a cloud OS on their premise can truly claim that they are doing CC
- However, hosting and management of the physical hardware of a private cloud can also be out-sourced to an IT specialist, i.e. to another company which is located somewhere else

2 What are the Pros and Cons of CC?

- There are common advantages for both flavors of CC that are explained before, ie. for CC with CSP and for CC with own cluster and cloud OS
- These advantages are:

1.) The access of CC resources and services is independant of the user‘s Internet device or physical location
   - all devices with browser are possible to access the cloud and its services
   - all locations in the world with Internet access are allowed

⇒ the user has a much higher flexibility and agility than with standard server equipment

2.) The total costs of ownership are lower

3.) Administration is much easier than for a computer cluster without cloud OS

4.) Isolation between VMs and to the Internet can be perfect, even if the VMs are situated on the same server

⇒ Data security between multiple tenants in the same cloud is very high

Comment: A tenant is a group of users who share cloud resources and their access privileges. Own chapters about tenants and multi-tenancy will follow. In short, a tenant can be seen as a joint software project between a user group.

- However, there are also a common disadvantages for both flavours of CC, i.e. for CC via a CSP and for CC via own cluster and cloud OS:

1.) A cloud is a distributed system whose VMs must communicate even in the same project via TCP/IP, although the VMs may be physically hosted by the same server or the same cabinet or the same computing center

⇒ Efficient data exchange with low latency between VMs is normally not possible

2.) Because of that, clouds are typically not used for High-Performance Computing (HPC)

3.) If VMs are engaged instead of LXC containers then high software-overhead results

4.) On the contrary, if LXC containers are used instead of VMs then data security between multiple tenants is significantly lower

- All other pros and cons of CC depend on the chosen flavour of CC and will be explained subsequently

2.1 CC via CSP

- In case of CC via a CSP, a customer has numerous advantages compared to own hardware and software

2.1.1 Pros

- A company can change its location without changing its IT and without any interruption of its business, at least from the computers point of view
- In AWS, for example, users have much more and better services than they could ever provide by themselves
- Users have a much higher IT reliability and robustness than in own installations
- Users have much less costs because:
  - no expensive hardware or no own data/computing center has to be provisioned
  - only simple PCs with Internet access are needed to profit from CSP services
  - no own personnel is required for the company's IT, at least in case of PaaS and SaaS
  - prices of IT resources in public clouds are very moderate because of thin provisioning
  - no capital-intense investments are required for own IT, only smaller running costs for rented IT occur

⇒ Deployment, operating and maintenance of company IT is much cheaper and more simplified by means of CC

- The CSP bills costs precisely and transparently for consumed resources
- Users have much higher flexibility with respect to growing or shrinking IT needs because of the following facts:
  - resources can be requested by the user without CSP intervention = On-Demand Self-service
  - resources can be obtained or returned within minutes or even seconds = Rapid Elasticity
  - resources are available on all layers of abstraction, from virtual IaaS hardware to sector-specific SaaS applications
  - resources can be automatically adapted to the demand = Auto-Scaling
  - resource demand can grow 'unlimited' without problems = Infinite Scaling
- CSP centers are much more energy-efficient than user clusters because so-called server consolidation via VM migration is taken place frequently

2.1.1.1 Server Consolidation via VM Migration

- VM migration means that a whole VM is moved to another computer
- A VM migration can even be accomplished while it is running and encompasses the following steps:

1.) VM's execution is stopped on a physical server

2.) The whole VM including its data is stored in one file (= "Snapshot")

3.) The file is copied to another physical server, and the VM is activated again

- The VM does not notice its migration
- After having migrated all VMs from those servers that are under-utilized to servers that are better utilized, unused servers can be switched off, thus saving power
- The remaining servers are then fully utilized which is cost-effective
- Server consolidation works the better the more VMs can be moved and the more servers are available as sources and targets of VM migration
- Big CSPs can always consolidate servers because of the millions of VMs and the dozends of computing centers they have

Example: AWS has at the time of this writing 16 regions worldwide. In the Frankfurt region alone are 3 computing centers deployed.

- Owners of a private cloud cannot profit that much from VM migration because of scarcity of VMs and of computing centers

2.1.2 Cons

- The arguments that are speaking against CC via a CSP are as follows:

1.) Data security is a critical issue if the underlying data/computing centers are not located on the territory of the Federal Republic of Germany

  - Even if the data center is in Germany, data security remains critical if the CSP performs automatic backups in other countries of the same region, because other coutries have weaker data security laws

2.) No interoperability exists between services of different CSPs
  - To port the own software infrastructure and applications from one CSP to another CSP is a challenge

⇒ **Therefore, a so-called vendor-lock-in results as soon as a decisions was made for being a customer of a CSP**

3.) If a CSP has technical problems, the user must stop his own business until cloud services are restored by the CSP

4.) If a CSP gets bankrupt the customers will probably go also bankrupt because company-proprietary data cannot be accessed and processed any more

2.1.2.1 Vendor-Lock-in

Def.: Vendor-Lock-in: If a customer cannot change its CSP any more because of the resulting high complexity or costs. As a result, the customer is bound to his CSP forever.

2.2 CC with own Cloud OS

- In the following, the pros and cons of CC with own cluster and cloud OS are explained
- In order to do that, it must be defined first what an own cloud OS is

Def.: An own cloud OS is a software to realize a private cloud or the private part of a hybrid cloud on the own premise by means of an own computer cluster.

Comment: OpenStack is used by many private users and companies to create own private clouds, and it is sometimes also used by CSPs to create public clouds.

2.2.1 Pros for own Cloud OS

- **Typically, only companies or institutions have an own cluster and cloud OS as CC users**
- **In case of CC with own cluster and cloud OS, a CC user has also many advantages, which are different to that of a CSP cloud**
- **These advantages are:**

1.) Data Security
- **this is not an issue because everything is under control of the CC user**

2.) Single Sign-On
- **the same user name and password is valid in the whole cluster instantaneously**

3.) Single Identity Management
- **all user names, passwords and access rights, i.e. the whole so-called authentication and authorization can be administered centrally from one console only, even for big clouds**

4.) Single Update

- **the update of the guest OSes in the VMs of his cloud can be automated by updating one "master" VM and replicating it as needed**

5.) Single Network Management
- **the cluster's virtual network management including virtual switches, routers, MAC and IP address allocation is easily accomplished from a central console via an existing cloud service (e.g. Neutron)**

6.) Easy Integration
- **the integration of new hardware into the cloud is simplified because of the central console and existing cloud services that alleviate that**

7.) Easy Performance Analysis and Tuning
- **the analysis and tuning of the performance of the cloud is significantly simplified because of existing cloud OS services**

8.) Easy Accounting
- **accounting and billing in the cloud for user departments is fully automated because of cloud OS services**

9.) Single Data Storage and Access
- **sharing of data in files within the same project is simple because of ready-to-use shared virtual file systems with automated data replication to distributed locations**

10.)Load Balancing

- load balancing between servers is possible and simple because of a cloud-wide meta-scheduler (e.g. Nova in OpenStack), because of existing load-measuring tools, and because of simple VM migrations between servers

11.)Easy Overall Administration

- The multitude of existing cloud OS services, together with the central admin console simplify the overall handling of the computer cluster without any extra software required

⇒ **A cloud OS offers many cluster management advantages for the admin of the CC user**

2.2.2 Cons for own Cloud OS

- **However, CC with own cluster and cloud OS has also drawbacks:**

1.) Difficult Installation

- Although diverse helper tools exist, the installation of a cloud OS such as OpenStack is a challenge and normally only possible for experts

2.) High Complexity

- Cloud OSes are mostly highly complex

Example: The OpenStack project has more than 80 Tsd. registered development members and comprises > 20 Mio. lines of code.

- Management services can be highly complex as well

Example: OpenStack's "Neutron" networking service has an API with about 244 REST-based request "calls".

3.) Slow APIs

- Because of REST, high-speed data transfers between management services or between user apps and management services are not possible

4.) Inefficient Inter-VM Coupling

- A cloud is a distributed system which implies inter-VM coupling via TCP/IP
- An efficient inter-VM data-exchange is not possible because of the high latency each TCP/IP protocol stack has
- TCP/IP would not be needed because the cloud can be in the same cabinet or in the same computing center

5.) Normally No High-Performance Computing

- Clouds are made for general-purpose computing, but not for high-performance computing (HPC) because of high latency in inter-VM data-exchange
- Only if a virtual shared memory between VMs is engaged, HPC is also possible

Example: Ivshmem is the only existing virtual shared-memory between VMs on the same server. It is based on a virtual PCI device in each guest OS and a common Linux SHM in the host OS of a server.

Comment: SHM = standard Linux shared memory.

3 Technologies of CC

- **All CC technologies were created to serve users and so-called tenants, and to maximize profit for the CSP**

Comment: A tenant is basically a project and will be explained later.

3.1 Common Technologies for Both, CSP Clouds and Own Cloud

- **There are some common software technologies for both, CSP clouds and own clouds which are:**

1.) Tenants and multi-tenancy comes into existence for resource pooling

2.) Linux containers (LXCs) or alternatively server virtualization can be chosen

3.) Auto-scaling for automatic resource allocation to users is possible

3.2 Tenant and Multi-Tenancy

- **Multi-tenancy is a property of CC**

3.2.1 Tenant

- **The tenant concept is derived from so-called multi-user jobs in mainframes of the 1960s**

Example: The code of a text editor existed only once in the main memory of a 1960 mainframe and was shared by many users at the same time, albeit exhibiting to each of them only his private edit file.

- **From that background, the definition of the term tenant is:**

Def.: A tenant is a group of users who are sharing both, cloud resources and access privileges.

Example: Users in a tenant are working in a joint task for which they are sharing the cloud's disk storage for data exchange.

- **In OpenStack, the notion of "tenant" evolved to the simpler term "project"**

3.2.2 Multi-Tenancy

Def.: Multi-tenancy is a feature of a cloud OS in order to support multiple isolated tenants at the same time. It means that the cloud serves multiple projects simultaneously.

- **Multi-tenancy has to provide shared resources and privileges within each tenant and isolated resources between tenants**
- **It leads to sets of user groups with cooperation inside of each group and full isolation between groups**

Example: Each application in Google Docs is a SaaS service that allows for multi-tenancy, because it can share documents within the same collaboration group, and because it separates documents between different groups.

Example: Neutron is an OpenStack multi-tenancy IaaS-service that allows for a shared vLAN for each user in the same tenant. It provides separate vLANs for different tenants. However, technically all vLANs all implemented by the same physical LAN.

Example: Linux Control groups (Cgroups) provide for every user group shared resources and privileges and isolate different Cgroups from each other.

- **Multi-tenancy leads to efficient cloud usage because it allows for resource pooling**

Def.: Resource pooling is for thin provisioning: The same physical resources such as computing power, disk storage or networking are shared between the users of a tenant and between multiple tenants.

- **Resource pooling reduces the costs of a CSP because the price of a resource is divided by the number of users and projects for it**
- **However, each shared resource has the potential risk to become a single-point-of-failure**

3.3 Linux Containers (LXCs)

- **Linux containers (LXCs) are based on Cgroups and on namespace isolation which are both kernel functions**

3.3.1 Cgroups

- **A Cgroup is a set of processes that is created and owned by the same user group**
- **All processes of a Cgroup can be frozen (checkpointed) for a later restart**
- **Furthermore, a Cgroup can be created by a Linux command in such a way that it does not exceed a prescribed main memory limit**
- **Additionally, high-priority Cgroups are also possible by a Linux command**
- **A high-priority Cgroup is a user group with a larger CPU share or more disk I/O bandwidth than other Cgroups**
- **Finally, the Cgroup's resource usage is measured by the Linux kernel for accounting and billing**

3.3.2 Namespace Isolation

- **In Linux, namespaces exist for:**

1.) Process IDs (PIDs), which allows for different PID namespaces

Comment: a namespace is the validity domain for everything that has a name. The domain is called "scope".

2.) User IDs (UIDs) in order to establish different UID namespaces

3.) Disk mounts to obtain different virtual file systems

4.) Server names to allow for different hostnames on the same physical server

5.) Inter-process communications (IPCs) to get different sets of IPCs that are isolated from each other

6.) Inter-networking to obtain different network controllers, IP tables and routing tables in order to create different network equivalents

- Namespaces allows for isolation between process groups such that they can access resources from the own group but not fromother groups
- The access isolation is accomplished by the fact that only the own namespaces are visible to a specific Cgroup

3.3.3 LXC Functionalities

- The LXC functionality in the kernel organizes the set of all processes on a physical machine into groups such that Cgroups are isolated from each other by means of Cgroup features and different namespaces

⇒ Each process group sees only its "own" Linux kernel, and other groups are invisible

- The "own" Linux kernel is also named "Linux container" or LXC
- Of course, there is only one Linux kernel for all LXCs
- On top of each LXC, standard Linux libraries and tools can be executed which mimic together with the kernel a whole Linux OS for every process group individually

- Inside of each process group, Linux access rights for resources must be set such that members of the same group can cooperate easily

⇒ LXCs make Linux to a multi-tenancy operating-system

⇒ LXCs are not a server virtualization

- The 1st advantage of LXC is the higher data security between process groups compared to Linux without LXCs
  - However, LXC containers do not protect data as secure as VMs do
- The 2nd advantage of LXCs is the low overhead compared to VMs
- The 3rd advantage is that a guest OS is not needed
- On top of LXCs, open source middleware has evolved that augment the features of LXCs

Example: "Docker" is the most widespread Container-Software that is based on LXCs.

3.4 Virtualization by means of VMs

- Virtualization means to have VMs not LXCs, together with virtual networks which connect them with each other and with the Internet
- Virtualization implies complex software technology that incures extra overhead compared to LXCs and that needs extra explanations (see Chapter 4 "Virtual Machines (VMs)"

- Finally, virtualization implies also to have a perfect isolation between VMs which is highly secure w.r.t. to data security

3.5 Auto-Scaling

- Auto-Scaling of resources is possible in all relevant clouds or cloud OSes
- In AWS:
  - auto-scaling is, for example, possible w.r.t. the number of VMs in the Elastic Compute Cloud (EC2)
  - also the capacities for storage and networking can be auto-scaled, as in some other AWS services
  - the throughput of the DynamoDB database, for example, can be auto-scaled due to the database input traffic
- In MS Azure:
  - the number of active PaaS applications can be auto-scaled on basis of any metric
  - advanced auto-scaling according to time and date is also possible
- In OpenStack:
  - various auto-scaling policies are possible by means of the "Senlin" service

Comment: the AWS auto-scaling API is also available in Senlin

Example: The OpenStack "Heat" service uses the "Senlin" service for auto-scaling the number of VMs in a cloud.

3.6 Disjoint Software Technologies

- Beside the software technologies which are common to both CC models, there are also other technologies where a distinction has to be made between CC via a CSP and CC via own cloud OS because they are model-specific

3.7 Specific Technologies for CC via CSPs

- AWS has:
  - language and platform-specific APIs incorporated into so-called Software Development Kits (SDKs), together with classical http(s) requests and even SOAP for some older services

Comment: The SOAP protocol allows for data exchange and remote procedure calls (RPCs) in distributed systems via the Internet.

  - Most services have also REST-based APIs, such as the AWS object store "S3" or the content delivery service "Cloud Front"
- MS Azure has:
  - language and platform-specific APIs as AWS has and HTTP(s) and REST
- Both CSPs provide for on-demand self-service via their APIs

3.7.1 Language and Platform-specific APIs

- **AWS:**
- **Several programming languages are supported by AWS by means of the SDKs**
- **Additionally, the AWS API Gateway allows to define any API for the compute cloud EC2, for the serverless "Lambda" execution environment or for web applications, for example**
- **AWS API calls are made manually by user requests via SDKs or automatically by content changes either in S3 or in DynamoDB, or by state changes in the workflow software named "Step Functions", for example**

Comment: "Step Functions" allows to define arbitrary actions of AWS services in arbitrary sequences. Such a thing is called workflow.

- **MS Azure:**
- **MS Azure has also SDKs and an API Gateway as AWS does**
- **Additionally, the Azure Active Directory (AD) allows to secure REST-requests by registering at the AD with credentials**

Comment: Credentials are username/password or a SSH key.

3.8 Specific Technologies for CC via own Cloud OS

- **CC via own cloud OS has typically only REST-based APIs, SDKs do not exist**

- **Furthermore, the own cloud OS as well as other software such as the hypervisor are visible to the owner of the private cluster**
- **The cloud OS location in the CC-user software-stack is described subsequently**

3.8.1 Location of Own Cloud OS

- **The own cloud OS is logically located above the "host OS" of the physical servers and the hypervisor and other software, but below the user apps which are executed in VMs or LXCs**

Comment: For host OS see a later chapter. In short, it is what the physical server executes.

- **The own cloud OS is always executed in the host OS of all physical servers and never in a "guest OS" of a VM**

Comment: For guest OS see a later chapter. In short, it is what the VM server executes.

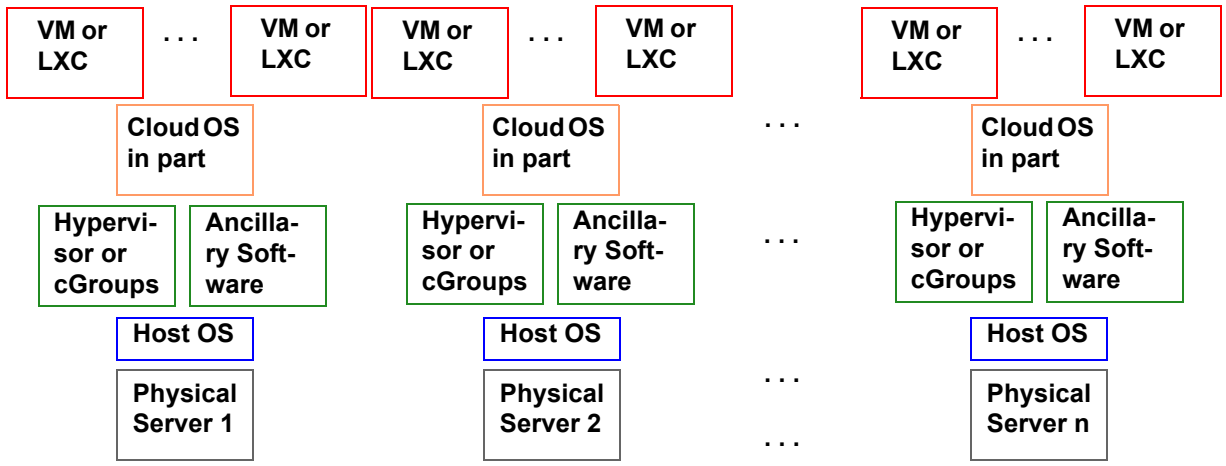
- **In contrast to classical OSes, an own cloud OS is mainly a collection of management services that in turn rely on a host OS, a hypervisor and other ancillary software**

Example: OpenStack Neutron relies on "Linux network bridge" or "OpenvSwitch" as ancillary software.

Comment: Linux network bridge is part of the kernel and emulates an ISO layer 2 bridge between a guest OS and itself. OpenvSwitch emulates a complete switch over which VMs can communicate with each other and with host OS.

Example: Neutron only configures and manages its ancillary software, but does not do much by itself during runtime. The true work in OpenStack networking is made by the Hypervisor, "Linux network bridge" or "OpenvSwitch".

- The resulting block diagram of the software stack is shown below



- Services of the cloud OS are active by themselves during runtime only if they do not have ancillary software as workers, for example because such software does not exist
- Then they must implement all API calls to themselves alone

Example: OpenStack's object store service "Swift" reads and writes data objects by itself because there is no other software that could do this.

- Most but not all cloud OS management service are accessible via a REST-based API

Comment: See the subsequent chapter about REST protocol for an explanation what REST is.

Example: OpenStack has had about 24 different REST APIs at the time when it comprised 32 management services.

- REST APIs of services may use JSON or yaml data structures as vehicle for longer call and return parameters
- Cloud OS services are mainly for the CC user, but services can also call each other, either by using the REST API or cloud OS-internal communication means

Example: The OpenStack "Keystone" service is called by any other OpenStack service to get authentication and authorization tokens.

3.8.2 Basic Services in own Cloud OS

- Always, a few basic services have to be installed on all servers in an own cloud

Example: OpenStack requires always some components of the "Nova" and "Neutron" services on every "node".

Def.: In Openstack and other cloud OSes, a node is a physical computer.

- Most of the other services are optional

- Additionally, some nodes have a special role in a cloud OS:

Example: OpenStack has in each cloud one or more "controller" nodes, one or more "network" nodes and one or more "compute" nodes.

- In contrast to classical OSes, a cloud OS allows to administer only virtual resources, because a cloud OS needs the hypervisor to penetrate to physical resources
- This help is granted, e.g., in case of trying to perform IO in a guest OS

Example: If a virtual Ethernet network interface card ("vNIC") trys to send a data frame to the Internet via TCP/IP, then the hypervisor translates this into a real Internet access via the physical Ethernet network interface card (NIC).

- Finally, all cloud OS management-services are coupled to a central admin console
- This console allows to monitor and control the whole cloud which is very convenient

4 Virtual Machines (VMs)

- The most important software technology for private and public clouds is the virtualization of physical servers or PCs by means of VMs
- Virtualization has three aspects that have to be accomplished all together:

1.) Virtualization of the physical main memory into VMs' main-memories

2.) Virtualization of physical CPUs and cores into VMs' CPUs and cores

3.) Virtualization of physical peripheral devices, such as network interfaces, switches and routers into VMs' vNICS, virtual switches and routers

- The listed 3 directions are implemented by the hypervisor
- In some cases, the hypervisor has a user space "emulator" as companion software that mimics a PC or server
- If both exist, emulator and hypervisor cooperate closely with each other

Example: KVM is the hypervisor for a Linux host OS, and QEMU is its emulator that creates a virtual PC.

Example: Hyper-V is the hypervisor for a Windows Server host OS. It has no emulator.

4.1 What is a VM?

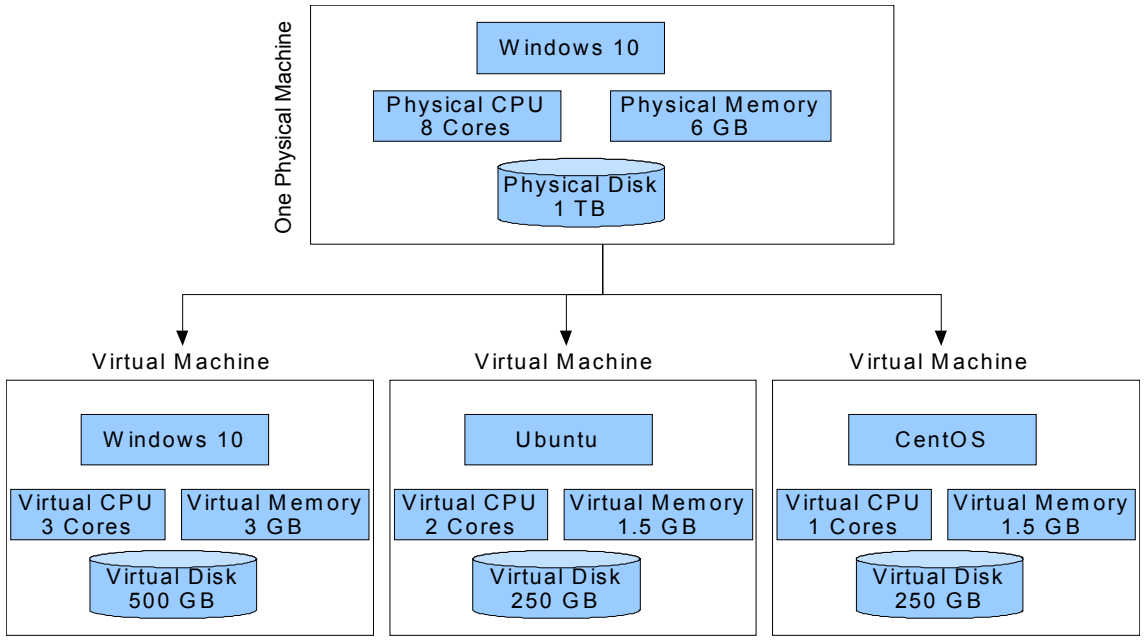
Def.: A VM is a collection of virtual IT resources created by software in a real computer. Each VM is created, maintained, optionally moved and terminated by a hypervisor which is also the VM monitor.

- In the context of CC, a VM is also called **"instance"**

Comment: OpenStack and AWS use "instance" instead of "VM". We are using VM because it is shorter.

- On every VM, a guest OS and all user applications are running as on real hardware
- Furthermore, a single real PC or server can accommodate multiple VMs/instances simultaneously

Example: Three VMs are emulated by one PC as shown in subsequent block diagram.



- **In the example above, there is no over-commitment of resources because the number of virtual cores and the accumulated sizes of virtual main memory and virtual hard drives is not higher than that of the underlying real hardware**

Comment: To over-commit = to over-book

⇒ **Thin provisioning does not exist in the example above**

- **A virtual multicore CPU can be created inside of a VM by adding multiple "vCPUs" to a VM**

Def.: A vCPU is the virtual counterpart of a real core.

Example: In case of an Intel i3, i5 or i7 CPU and a so-called hyperthreading factor of 2, each real core can emulate 2 vCPUs without being over-committed.

- **Although being on top of the same host OS, each guest OS can be of different type or version, as well as each VM setup**
- **Each VM has its own guest OS => there are as many guest OSes as VMs exist**

4.2 Advantages of Virtualization

- **The advantages of virtualization are lying mostly at the side of the CSP, but the user has also benefits**
- **A CSP and its system admins have the following advantages:**

1.) There is no visible change for CSP customers before and after virtualization

- **Normally, the user can find out that hardware was virtualized only by run time measurements**
  - **they show that his applications have become slower**

2.) User software licenses are in a VM as valid as they have been on real hardware

3.) A reduction of the electricity bill results
- the reason for that is server consolidation
- continuous server consolidation is important for CSPs to save costs
- energy saving alone would already justify virtualization in a computing center

4.) System stability is higher
- If a VM crashes this does not affect other VMs
- If a VM trys to intrude an other VM this will be not possible
- The reason is the perfect isolation of VMs between each other, even in the same computer

5.) Backups are simplified
- a whole VM including all user files can be backuped in one “image“ file only
  - the VM image is a so-called snapshot of the VM and all of its content

6.) Restarts are simplified
- periodic snapshots can be made and help the admin as follows:
  - in case of a VM or application crash, a VM snapshot which was taken before the crash can be re-run which is called Checkpoint and Restart
  - restarting a whole VM is as easy as starting an .exe file
  - it is not needed to restart the whole physical hardware, instead the VM launch is sufficient

7.) VM migration for server consolidation or for maintainance is easily accomplished:
- from every VM, a snapshot can be taken by one command which is issued from the central admin console
- this VM image is copied to a 2nd physical machine and run again
- after the 1st physical machine is free of VMs, it can be easily maintained or shut down

8.) A cloud OS on top of virtualization can create, administer, maintain and terminate virtual resources by means of its powerful management services
- already existing VMs can be made visible also to a cloud OS that is installed later
- the combined advantages of virtualization and of cloud services are available to an admin

9.) Cyber security is improved
- attacks are much more difficult ,because login into a VM from outside is typically accomplished via an SSH key such as RSA for example

Comment: a RSA key is safer than user name/password.

- isolation between VMs and between different virtual network equipment is perfect
- if combined with a cloud OS, VMs and and virtual network equipment can be completey isolated from the Internet by not donating a public IP address to the VM => the hacker does not even see the VM

10.)Some hypervisors allow users to created VMs freely w.r.t. architecture and size because nothing really exists
- even a virtual shared memory between VMs is possible

Comment: W.r.t. = with respect to.

4.2.1 Setup of a VM

- Typically, a VM consists of the following parts:

1.) One or more virtual sockets for virtual CPU chips

2.) One or more virtual-cores (vCPUs) per virtual CPU chip

Comment: Also real servers have several sockets and several cores per socket. In contrast to that, real PCs have only one socket that accomodates one CPU.

Comment: a vCPU is sometimes also called "logical processing unit".

3.) Virtual main memory, including a virtual memory management unit (vMMU)

4.) Virtual periphery, including virtual hard drives and network interface cards (vNICs)

- These virtual components have indirect access to a real window on a real screen, to a real keyboard and to a real mouse, as well as to a real network interface card and to a real hard drive
- Indirect access is possible by means of a hypervisor and its companion emulator if existent
- In such a case, the emulator does most of the work for PC or server emulation

4.3 Disadvantages of Virtualization

- Virtualization, howewer, has some disadvantages as well:

1.) Enormous software overhead for its realization is required

⇒ Very high structural complexity results and thus increased execution times

- Because only about 10-20 PCs can be emulated by one server, the high complexity may not always pay out in comparison to LXCs, from which about 100-200 can be emulated by the same server
- in order to reduce performance degradation, additional hardware units exist on the mother board, the periphery and in the cores which are called hardware accelerators for virtualization

2.) Accelerators for virtualization are mostly indispensable, but they increase price and complexity of hardware

- For these reasons, virtualization is under strong market pressure by LXCs and their further improvements, such as "LXD" or "Docker"

Comment: "LXD" is from Canonical Ltd., "Docker" is from Docker Ltd.

- This occurs although they are not as safe as VMs

4.4 Host OS and Guest OS

- On every real server or PC, only one host OS exists
- Typically, the host OS is a standard OS such as Linux or Windows
- The host OS is indispensable as the lower layer for the cloud OS because it contains the hypervisor in its kernel, or it is even the host OS and the hypervisor combined

Example: Linux is the host OS for OpenStack. It has KVM as hypervisor kernel module and QEMU in user space for VM emulation. XEN is another host OS that is its own hypervisor at the same time. Windows is a third host OS that has Hyper-V as hypervisor.

- On any emulated PC or server, i.e. VM, multiple user applications are runnable under the same guest OS as on real hardware

Comment: There is a difference between simulation and emulation. Simulation happens solely by software, while emulation is with hardware assistance or fully in hardware. As a consequence, a simulation is never real-time capable, while emulation can be.

- Technically, each VM is a process in host OS, which means it has its own address space and is isolated from other processes by the memory management unit of the CPU core that executes the host OS
- Together with the emulator (if existent), the hypervisor ensures that the VM knows nothing about the underlying host OS, nothing about to be a host OS process only, and nothing about the real hardware

- Typically, communication between VMs on the same server is possible only via the Internet, i.e. via TCP/IP, provided the VM has a vNIC with indirect Internet access

Comment: Indirect Internet access is possible for a VM if its vNIC has a valid MAC address, a public IP address and if its hypervisor can access a real NIC that is connected to an Internet gateway.

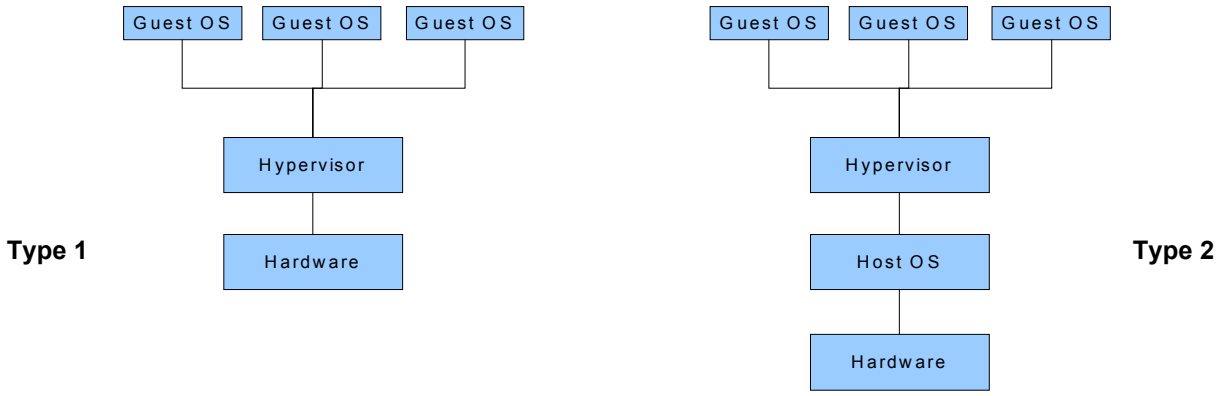
4.5 Hypervisor

- The hypervisor is located always below the VMs but above the hardware
- An emulator, if existent, contains a single VM in its address space
- One has to differentiate between type 1 und type 2 hypervisors

Def.: A type 1 hypervisor does not need a host OS, instead it is also the host OS by itself. It is located directly on top of a physical machine and has all device drivers needed to operate the hardware. It also does not have an emulator companion.

- Both hypervisor types allow to create, to configure, to start, to stop and to terminate VMs on a single physical machine

Example: XEN and Hyper-V are type 1 hypervisors, while KVM is a type 2 hypervisor.



- If a type 1 hypervisor is present, then the device drivers of the guest OSes must be stubs only which is called paravirtualization of the guest OS

Comment: An own chapter about paravirtualization will follow.

  - In that case, the guest-OS device-drivers stubs are such that they call the device-drivers of the type 1 hypervisor instead of doing own IO

- Typically, a type 1 hypervisor implies the paravirtualization of the guest OS

Example: XEN is a type 1 hypervisor and a micro OS kernel at the same time. It provides for its guest OSes the most essential functionalities of an OS, such as IO via device dri-

vers, process scheduling via time sharing, memory management via own pages tables, and some other important kernel functions.

- In contrast to a type 1 hypervisor stands the type 2 hypervisor:

Def.: Type 2 hypervisor: It is executed inside of a standard host OS kernel and is therefore not a micro kernel. Instead it needs a host OS and its device drivers. It may have also a software emulator as companion to mimic a computer on which a guest OS can be executed.

Example: KVM is usually a type 2 hypervisor. KVM is used by Linux and is part of its kernel. XEN, as type 1 hypervisor, is used by Amazon AWS.

- Both hypervisor types provide for so-called server Virtualization

Comment: Server virtualization will be explained later.

- For a type 2 hypervisor, the device drivers of the guest OSes always remain unmodified
- Additionally, all guest-OS device-driver calls are intercepted by the hypervisor as soon as the guest OS tries to perform them

⇒ Type 1 hypervisors are executing guest-OS IO-calls as proxies via their own drivers. Type 2 hypervisors are doing guest-OS IO-calls as proxies via the underlying host OS.

- Both types need and have system privileges to execute the complete instruction set of the physical CPU core
- Both types can thus access all physical files and devices, including main memory
- To summarize:
  - type 1 hypervisors do not need an emulator or a host OS and enforce typically paravirtualization of the guest OSes
  - type 2 hypervisors need a host OS and sometimes also a emulator, but never use paravirtualization
  - both hypervisor types are responsible for their VMs, but not for what happens in and between physical machines in a cluster
    - this is the task of the various host OSes and a cloud OS (if present)

4.5.1 ESXi, Hyper-V and KVM/QEMU

- ESXi is the latest type 1 hypervisor of VMware Inc.

Comment: VMware is the technology pioneer and market leader in commercial virtualization solutions and therefore important.

  - ESXi is used in their "vSphere" product for server virtualization
  - ESXi mimics a virtual server and not a PC

Comment: The difference between server and PC is the higher performance. A server typically has multiple CPUs with a shared memory between.

  - it is used by "vCenter" which is the major part of vSphere

  - among other tasks, vCenter is used to manage VMs created by ESXi
- Hyper-V is the standard hypervisor for MS Windows
  - it is of type 1 because it exists directly on top of the hardware
  - however, it is not a microkernel, and it does also not imply paravirtualization
  - instead, it follows a more complex virtualization concept by creating a so-called parent partition
  - in the parent partition, Windows Server is executed as host OS of a physical server
    - Windows Server is a fully-fletched host OS and not a microkernel
  - the parent partition spawns multiple child partitions, each with an own guest OS which can be Linux or Windows e.g.
  - in each child partition, one VM is located
  - guest-OS device-driver calls are forward via a so-called VMBus to the host OS and executed in the host OS
  - this is similar to paravirtualization
  - the VMBus connects host OS with guest OS
  - its endpoint in host OS is coupled to a "Virtualization Service Provider", i.e. to a Windows Server
  - its endpoint in guest OS is coupled to a "Virtualization Service Consumer", i.e. to a VM
- KVM is the standard type 2 hypervisor for Linux and has QEMU as emulator companion
  - KVM/QEMU mimic together an older PC that has an IDE bus as hard drive interface and an

PCI bus for its other periphery

Comment: IDE was the prerunner of parallel ATA, which was in turn the prerunner of SATA. PCI was the prerunner of PCI-X, which was the prerunner of PCIe. Because of these old technologies, the emulated PC is also old.

- QEMU is included in any Linux distribution as indispensible user module
- An own QEMU instance is needed for every VM because the whole VM is incorporated into QEMU's address space in the host OS
- Each QEMU and thus each VM is a host OS process
- However, there exists only 1 KVM per physical server for all QEMUs/VMs

4.6 Paravirtualization

- For paravirtualization, all original guest-OS device-drivers must be replaced by stubs

Comment: A stub is rudimentary software that is not fully functional. It has only the API of a fully operational software of the same name but is basically empty.

Example: KVM provides by means of "virtio" several alternative stub device drivers for Linux as guest OS. XEN provides the same for Linux and Windows as guest OS.

Comment: Virtio is another Linux kernel module that cooperates with KVM.

- The alternative device drivers are stubs because they only hand-over their call parameters to QEMU and get back the results from KVM/QEMU
- They are not processing these call parameters or results

- Stub device drivers have the same API as the original drivers, but they do not drive anything since they are running in guest OS which has no access to hardware
- In contrast to the original device drivers, they know that they are running in a guest OS and cooperate therefore actively with that QEMU that is responsible for the guest OS

⇒ intercepting of interrupts or of priviledged instructions by the hypervisor is not needed since the stubs do not want to control real hardware

- This saves time which is why paravirtualization is an efficient software solution for virtualization

Example: If a TCP/IP packet has to be sent from a VM, its data is forward via the subsequent chain: Guest-OS application -> Guest-OS device-driver stub -> QEMU -> KVM -> host-OS device-driver -> NIC. QEMU hands over the call parameters to KVM, and KVM calls the proper host OS device driver as a proxy and returns the status result via QEMU to the driver stub.

- Hypervisors have standardized APIs for paravirtualization

Example: XEN provides a "VMI API" for the driver stubs in guest OS. KVM has its virtio API for the same purpose.

- With the advent of more and better hardware accelerators for server virtualization paravirtualization has lost importance

4.6.1 Virtio for Paravirtualization

Def.: Virtio is a Linux kernel module for the virtualization of peripheral devices which is called "IO virtualization". Virtio has own device drivers that are only stubs. It is used for paravirtualization.

- There exist Virtio stub drivers for block devices such as virtual hard drives, for network devices such as vNICs, and for other PCI devices
- Virtio cooperates with KVM on the kernel level
- KVM in turn has two APIs, one for all QEMUs in user space and one for virtio in system space
- Please note, as soon as virtio is called by KVM, KVM mutates into a type 1 hypervisor
- As a consequence, KVM can realize paravirtualization by means of virtio
- Because of that, KVM is either a type 1 or a type 2 hypervisor
- The difference between KVM+virtio and XEN is that the Linux kernel is not a micro kernel as XEN but a fully-fletched OS
- The advantage of KVM+virtio+QEMU (= type 1 hypervisor) compared to KVM+QEMU (= type 2 hypervisor) is its speed which is much higher because of paravirtualization, provided that no hardware accelerator for virtualization exists

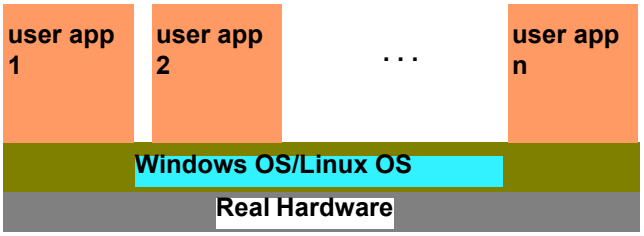
5 Server Virtualization

- A PC is capable to accommodate about 1-5 VMs, and then its resources are used-up
- Servers instead can host about 10-20 VMs which is called "server virtualization"
- A server that was virtualized typically does not provide physical resources to users but only virtual ones

5.1 Comparison between Classical Computer and Virtualized Server

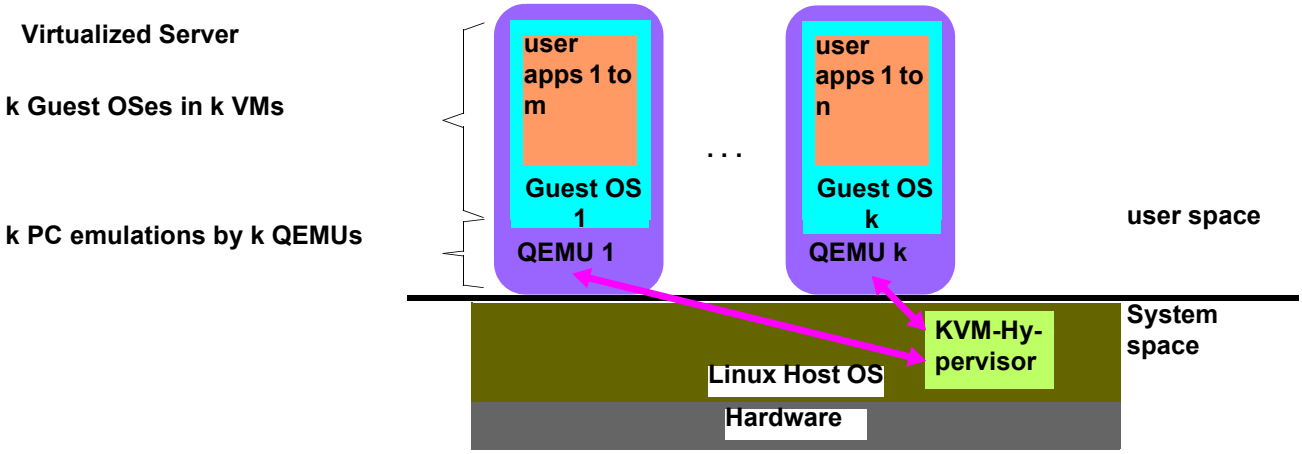
- Without server virtualization, we have a classical computer as we know it:

Classical Computer



In a classical computer, a single host OS supports n user applications, and host-OS device-drivers control real hardware

- Virtualized servers are more complex:
- In a virtualized server, for example by means of KVM/QEMU, more software units exist



- **KVM/QEMU interact with each other and with these software units as follows:**

1.) KVM is executed as a loadable kernel-module in system space

2.) Multiple QEMUs are executed in user space, one for each VM

3.) Each QEMU is a host OS process in which a complete PC is emulated as VM

4.) In each VM, one guest OS is running, together with user applications in time-sharing

5.) In the standard configuration, each QEMU emulates an old PC with PCI and IDE buses

- **in this PC, QEMU can be configured to emulate a multi core CPU, RAM and periphery**

- **it is optionally also possible that QEMU emulates multiple CPUs with shared memory between them, in order to mimic a server**
  - **however, it is not possible to emulate hard drive capacities that are in sum bigger than the real drive of the underlying physical machine**
- **if too many virtual resources have to be emulated simultaneously, the server becomes over-booked, and its performance drops drastically because of continuously paging in and out of hard drive records**
  - **this paging will not stop because physical memory is over-booked**

6.) KVM communicates with all QEMUs via its QEMU API

7.) Each VM is isolated from other VMs as follows:

- **the isolation is accomplished by KVM/QEMU and by the fact that each VM is a host OS process**
- **typically, VMs can communicate only via TCP/IP with each other, even if located in the same physical machine**
- **additionally, KVM intercepts all attempts of guest OSes to access real hardware**
  - **for example, KVM blocks attempted IO accesses and executes them by itself by means of a proper host-OS device-driver call**
  - **KVM returns the result of the host-OS device-driver call to that QEMU whose VM has initiated the IO access attempt by its guest-OS device-driver**
  - **The host-OS device-driver result is forward by QEMU to the guest-OS device-driver**
  - **The same holds also for all guest-OS access-attempts to real main memory**

⇒ **Each guest OS and its applications have the impression that they are accessing real hardware which is a fake**

5.2 How the Host OS sees Guest OSes, QEMUs and User Applications

- **For Linux as host OS, the situation is as follows:**
  - **Each guest OS in host OS is a host OS thread of its QEMU process**

Comment: A thread is a weaker version of a process. A thread shares name space and all other process resources with its father process.

  - **All user applications of a guest OS are host OS threads of their guest OS process in host OS**

- **In Linux, threads are scheduled as processes and similarily treated but differentiated from "normal" processes by a "parent process ID":**

1.) A user application has a host-OS process-ID and the parent process ID of its guest OS

2.) The guest OS has a host-OS process-ID and the parent process ID of its QEMU

3.) Each QEMU has a host-OS process-ID only and is thus a father process and no thread

5.3 Privileged CPU instructions in Guest OS

- **KVM intercepts privileged CPU instructions of the guest OSes as follows:**

1.) KVM intercepts all attempts of a guest-OS device-driver to execute a privileged CPU instruction which would be needed to perform real IO or to access physical memory

2.) Additionally, KVM intercepts all privileged CPU instructions of a guest OS that deal with the real memory management unit (MMU) and with real page tables

3.) Finally, KVM intercepts all privileged CPU instructions of a guest OS that have to do with system traps, timers and interrupts

4.) Instead KVM executes all privileged instructions by itself and by the help of the host-OS device-drivers and forwards the result to the respective QEMU

⇒ **KVM is using corresponding host OS calls to replace the guest OS calls that are intercepted by KVM**

5.) QEMU forwards the result to its VM and thus to the guest OS or to the application which has made the attempt

⇒ **The effect of the host OS call is the same as the guest OS call has intended. Thus no guest OS understands that it is running only inside of a QEMU host-OS process and that its calls have no direct effect.**

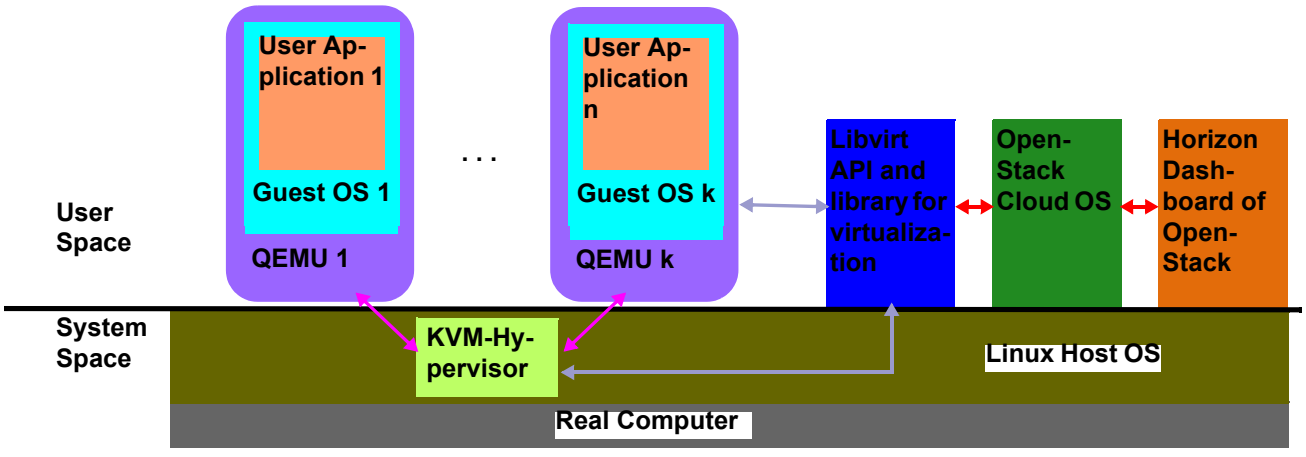
- **In case of an IO attempt by means of a guest-OS device-driver call, IO virtualization is used as part of server virtualization**

5.4 Server Virtualization in a Cloud

- In case that a cloud OS such as OpenStack is present on top of virtualization, the situation in the computer changes again
- In this case, a software called libvirt is the mediator between OpenStack and the virtualized server
- Libvirt is the API and library for server virtualization that helps OpenStack in the management of VMs
- Libvirt becomes thus one of the main ancillary software stacks OpenStack has
- With libvirt, VMs can be created, started, stopped and changed more easily than with KVM/QEMU alone
- Libvirt in turn uses KVM/QEMU, for example, as underlying actuators that do the job
- Libvirt supports other hypervisors as well
- Furthermore, Libvirt has a comfortable CLI named "virsh" and a GUI named "virt-Manager"

Comment: CLI = Command Line Interpreter; GUI = graphical user interface.

- The "Horizon" dashboard, which is the central operator console in OpenStack, is using the libvirt CLI to create an own GUI that is even more abstract and comfortable than that of virt-Manager



- The central dashboard provides an identical "look and feel" for all OpenStack services

5.5 Virtualization of Memory, CPU and IO

- Server Virtualization has the three sub-tasks of memory virtualization, CPU virtualization and IO virtualization
- All three sub-tasks together are needed to mimic a PC or server on a physical computer

Def.: Server virtualization is the creation of multiple VMs on a single physical machine by software, such that the VMs share the machine's resources without noticing that.

- The sharing of physical resources between multiple VMs is a challenge because no hardware component was originally designed to be shared between multiple VMs
- In the worst case, the host OS, all guest OSes and the user applications want to access the same physical resource, such as a NIC e.g., simultaneously

⇒ **A safe serialization of access requests to this physical resource must be established. Side-effects such as deadlocks are not tolerable.**

- As a consequence, a purely software-based server virtualization becomes difficult and slow
- This fact is explained subsequently by showing the internal complexity of memory virtualization, CPU virtualization and IO virtualization

5.5.1 Memory Virtualization

Def.: Memory virtualization is the creation of a classical virtual main memory for every VM, although there is only one real main memory that is already virtual for the host OS.

- The virtual main memory of a VM is based on double virtualization, i.e. on a double illusion because:
  - Virtual guest OS memory addresses have to be translated into host OS addresses that are already virtual, which is why they have to be translated a 2nd time into physical main memory addresses
  - "Physical guest OS addresses" of virtual device registers have to be translated into physical host OS addresses of real device registers
- Each main memory access of a guest OS and its user applications has to be intercepted by the hypervisor and is executed by that QEMU that is responsible for the respective VM in its user address space
- Additionally, all accesses to physical device addresses have to intercepted by KVM and handled by QEMU
- Furthermore, all page fault exceptions, DMAs und interrupts in a guest OS have to be intercepted and handled by the hypervisor as well
  - this is needed because the guest-OS page-tables, its memory addresses and interrupt vectors are all wrong
  - they do not reflect the situation in the real hardware and in the host OS
  - they may even collide with the page-tables, memory addresses and interrupt vectors of

other VMs

- Finally, if the user wants to have a virtual shared memory between virtual CPUs in the same physical server this has to be emulated by KVM/QEMU as well

Comment: The only possibility for a virtual shared memory between VMs in the same server is “ivshmem“ which is a QEMU feature.

- To handle all the aformentioned tasks in pure software is difficult and slow

5.5.2 CPU Virtualization

Def.: CPU virtualization is the software emulation of multiple CPUs, cores and hyperthreads for a virtual PC or server by a small set of real processors or cores.

Comment: There is some confusion in CPU virtualization w.r.t. terminology: OpenStack calls the emulated cores in a CPU “vCPUs“, although they are only cores. Furthermore, a VM is termed by OpenStack an “instance“ but not a VM.

- CPU virtualization is difficult because each QEMU and its VM have only user space privileges while cores have an instruction set that requires sometimes system privileges
- As a consequence, VMs cannot execute all CPU instructions
- This implies the described complex cooperation between KVM and each QEMU in case of access privilege violations
- CPU virtualization implies also to create a virtual Translation Look-aside Buffer (TLB) inside of a virtual MMU

Comment: The main component of a MMU is the TLB.

- However, the virtual TLB requires frequent updates which is why there is a hardware assistant in every recent core to accomplish this quickly: the IOMMU
- With QEMU, it is even possible that one CPU of type x emulates another CPU of type y from a different vendor
- However, this is very time-consuming because of full software emulation

5.5.3 IO Virtualization

Def.: IO virtualization is the mapping of virtual IO resources, such as network interfaces or hard drives of the VMs onto really existing periphery.

- The mapping is based on multiplexing the real periphery between VMs without interferences
- This is a challenge because the periphery was never built to be multiplexed between host OS processes, i.e. VMs
- The host OS can only serialize IO accesses to real periphery, because it is not possible to time-share real periphery between its processes in the middle of an IO operation

Example: Multiplexing of a vNIC between host OS processes requires the serialization of data frames in the vNIC during different ongoing message transmissions.

- Multiplexing real periphery between VMs requires also the serialization of record reads/ writes on the hard disk during ongoing file IO operations

- Multiplexing finally implies input and output queues and a queue scheduler for vNIC frames and for hard drive records which the host OS does not have
- The software is complex that has to handle these tasks alone, i.e. without hardware assistant
- Such assistant is called **hardware accelerator for virtualization**

Comment: See subsequent chapter for hardware accelerator for virtualization.

5.5.3.1 Example of IO Virtualization by Sending an Email from a VM

- If a QEMU VM wants to send an email, everything is as on real hardware except the last step where the guest-OS device-driver wants to access the vNIC of a VM by means of a privileged instruction
- Then KVM intercepts this instruction and calls the host-OS Ethernet device-driver (if not paravirtualized)
- Then the host-OS Ethernet device-driver sends the email through the real Ethernet interface
- Finally, KVM generates a virtual "done" interrupt and status message for the guest-OS device-driver of the vNIC, so that it gets the impression that it would control real hardware

5.6 Hardware Accelerators for Server Virtualization

- The goal of hardware accelerators for server virtualization is to obtain a close entanglement between hardware and software so that hypervisors are unburdened
- This speeds-up the execution of VMs significantly and simplifies the software as well
- Because of such accelerators, there is nowadays nearly no difference between an application running in guest OS or in host OS

Example: KVM/QEMU, XEN, ESXi and Hyper-V support hardware accelerators by cooperating with them in the cores, on the motherboards, in the BIOS and at the periphery.

- Because of these benefits, more and more accelerators for virtualization have come into existence
- Meanwhile they have fully superseded purely software-based server virtualization
- Without these accelerators, cloud computing would not be feasable because of speed, i.e. they are important
- In order to use them, the BIOS must support them and the BIOS configuration flags must be set properly which is not easy

5.6.1 Accelerators for Memory Virtualization

- In the physical core of any recent multi core CPU is a 2nd memory management unit (MMU) called IOMMU which is responsible for the real-time mapping of "physical" guest-OS IO-addresses into physical host-OS IO-addresses that really exist

- Additionally, other accelerators for memory virtualization exist

Example: The Intel "Extended Page Tables" EPT and the AMD "Rapid Virtualization Indexing" RVI are other commercial hardware accelerators for memory virtualization.

5.6.2 Accelerators for CPU Virtualization

- There exist also commercial hardware accelerators for CPU virtualization in recent CPU chips

Example: The hardware accelerators for CPU virtualization from Intel are the "Virtual Machine Extension" (VMX) and the "Virtual Machine Control Structure" (VMCS). They are subsumed under the brand "Intel Virtualization Technology" VT-x. AMD has similar solutions.

5.6.3 Accelerators for IO Virtualization

- There are three main methods in hardware accelerators for IO virtualization:

1.) In PCIe devices is an "Address Translation Service" (ATS) for the real-time mapping of guest-OS PCIe-device register-addresses that are virtual into host-OS PCIe-device register-addresses that are real

2.) In PCIe devices are multiple input and output queues in hardware for multiplexing disjunct VM accesses

   - With multiple input und multiple output queues in hardware, the multiplexing of a single PCIe device between multiple VMs is easily possible because every data item can always be obtained and delivered from and to the proper VM without hypervisor intervention

3.) With the previous two main methods and the IOMMU, guest-OS device-drivers can even be entitled to directly control host OS devices:
   - This is called Single Root I/O Virtualization (SR-IOV)

- All three main methods together unburden the hypervisor significantly

5.6.3.1 SR-IOV

- With SR-IOV, commands from a guest-OS device-driver are forward via QEMU and KVM to the real device without participating the responsible host-OS device-driver
- Furthermore, SR-IOV multiplexes device access requests from VMs including their data such that each VM gets its data automatically by hardware
- SR-IOV thus enables guest-OS device-drivers from multiple VMs to control the same physical PCIe device in a round-robin manner

⇒ SR-IOV eliminates the calling of the resonsible host-OS device-drivers via QEMU/ KVM which reduces communication and overhead significantly

- SR-IOV must be supported by the BIOS, the motherboard and the CPU in order to work
- SR-IOV is included in newer PCIe devices, but not in the low-price category
- Other important commercial hardware accelerators for IO virtualization from Intel are:

1.) I/O Acceleration Technology (IOAT)

2.) Virtual Machine Device Queues (VMDq)

3.) Virtualization Technology for Directed I/O VT-d

- They were bundled in the brand „Intel Virtualization Technology for Connectivity, VT-c“
- AMD has similar technologies

5.6.4 Status of Hardware Accelerators for Server Virtualization

- The user of a VM and thus the user of a cloud typically does not know or does not understand the technical difficulties incurred by virtualization
- Instead, he expects from a VM in a cloud the same speed, functionality, main memory size, hard drive size, memory protection and cyber security as from a real PC
- The status of hardware accelerators for server virtualization is that a VM can execute a user application meanwhile nearly as fast as the physical computer could do, but for 10-20 VMs at the same time
- Hardware accelerators do not only speed-up server virtualization, they also simplify the emulation software needed significantly

5.6.5 Summary of Hardware Accelerators

- IOMMU, ATS, hardware queues, SR-IOV and a better BIOS ensure together that each VM gets only its data, although the real periphery is multiplexed between VMs
- The I/O Acceleration Technology (IOAT), Virtual Machine Device Queues (VMDq), the Virtualization Technology for Directed I/O (VT-d), the Virtual Machine Extension (VMX), the Virtual Machine Control Structure (VMCS) and the Extended Page Tables (EPT) additionally simplify the job of the hypervisor and the emulator (if existing)
- However, even with those accelerators cloud computing is still not high performance, w.r.t. to inter-VM communication in the same server or between servers

Comment: If more performance is needed, it is possible that a hypervisor and LXCs can coexist peacefully in the same server and thus in the same cloud.

5.7 Inter-vCPU and Inter-VM Communication on the Same Server

- The basic concept of every cloud is to isolate VMs from each other
- This is good for stability and data security
- VMs can typically communicate only as a distributed system, i.e. via TCP/IP, although being in the same server, in the same rack or in the same computing center
- TCP/IP was not made and is not indended as effective mean for this use case
- A problem arises if the VMs of different users that are working in the same project want to exchange data quickly

- This is normally not possible in a cloud
- At least, communication between the vCPUs ("cores") of the same virtual CPU is easy because each vCPU is implememented in QEMU as a child process of the CPU father process
- All child processes and their father process are sharing the same memory address space, which allows for simple and fast data exchange by shared variables between vCPUs of the same VM
- However, a shared memory typically does not exist between VMs
- Therefore, Inter-VM data exchange instead is normally slow because of TCP/IP and other software overhead
- This is bad for many cloud applications, as well as for High-performance Computing (HPC) which is thus normally not possible in a cloud
- In Amazon's AWS and in VMWare's vCloud, there is no option to improve this situation
- In OpenStack, however, there is a way-out by using virtual shared memory implemented via "Inter-VM Shared Memory (ivshmem)"
- Ivhsmem is situated in a virtual PCI device and mapped onto physical shared memory in host OS
- This is technically exceptionally complex, but about 10 times faster than TCP/IP which makes ivhsmem important
- It is part of QEMU and thus of Linux
- However, ivshmem is scarcely used because it is "unknown territory" to most user

- This is why, inter-VM communication on the same server without ivshmem has to be discussed first

5.7.1 Inter-VM Communication Without Ivshmem on the Same Server

- In principle, there exist two software instances in a cloud that could be able to perform inter-VM communication inside the same server if ivshmem is not present
- These instances are the QEMUs and OpenStack's "Neutron" network service
- In fact, it is the QEMUs that are responsible for inter-VM communication inside the same server but not Neutron
- Furthermore, without ivshmem, there are two sources of overhead: TCP/IP and KVM/QEMU
- The latter create a huge overhead because of the following steps:
  - QEMU uses an ancillary software called "macvtap" in so-called "bridge mode" for inter-VM communication in the same server
  - Macvtap is a crossing of the Linux "macvlan driver" and a "tap device"
  - Macvtap replaces the vNIC device-driver API of macvlan by a classical Linux tap API which makes it easier for the user to handle the API
  - This is due to the nature of a Linux tap device which is explained below

5.7.1.1 Linux Tap Device

- A Linux tap device mimics the API of a physical or virtual Ethernet card (NIC or vNIC) but does nothing inside, i.e. it is empty
- A Linux tap device is a user-space software-interface and delivers a virtual endpoint to which a NIC or vNIC can be connected
- Instead of passing frames to and from a NIC or vNIC, frames are only read from and written to a tap driver, which is an API only to which ancillary software can be connected subsequently

Example: The Linux bridge that is employed by Neutron as ancillary software is using a Linux tap device as its API.

- The Linux kernel makes each tap device available to user space via the /dev/tap<N> device file, where <N> is the index of the software-only tap interface

5.7.1.2 Macvlan Driver

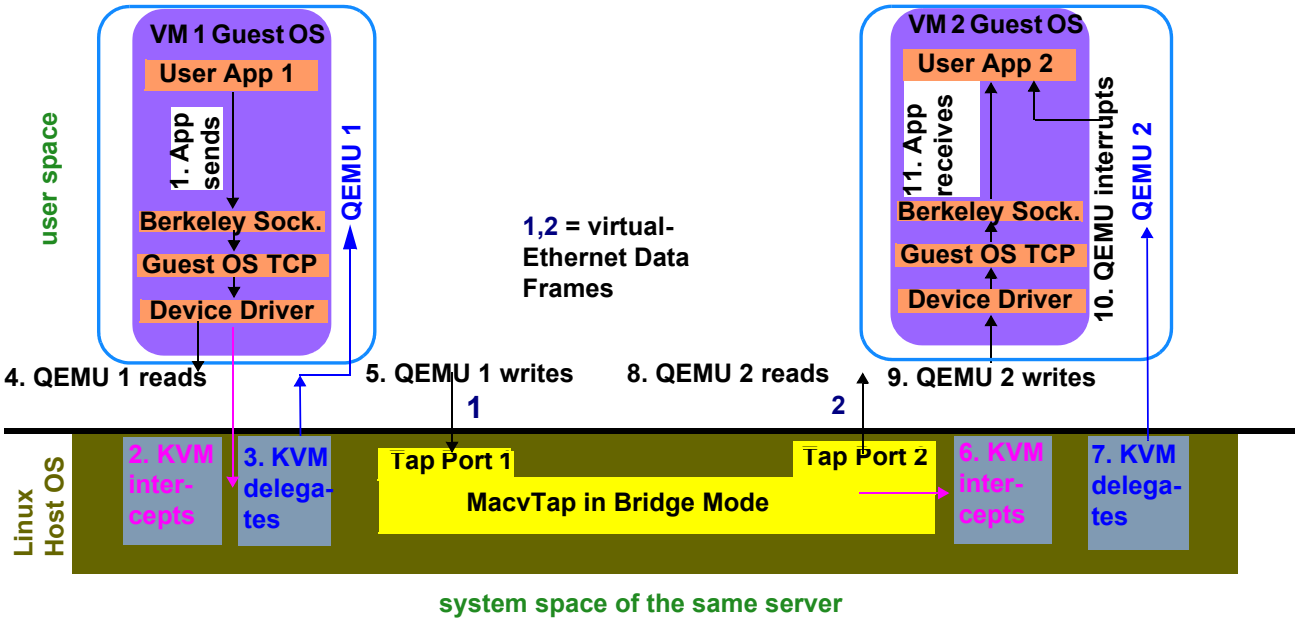
- The macvlan device driver connects virtual network interfaces (vNICs) to a single physical network interface (NIC) by multiplexing multiple vNIC data streams onto one real NIC
- Each vNIC has its own MAC address which is distinct from other vNIC's MAC addresses
- Frames sent to or from the vNICs are mapped onto the same real NIC

5.7.1.3 Macvtap

- Macvtap is a vNIC device driver with a tap interface to any user-space code
- The output of macvtap can be connected to a second tap from other software, as it is the case for each vNIC
- A user space code, such as QEMU for example, can open the /dev/tap<N> device file and send and receive Ethernet frames over it
- Originally, it has been the system-space TCP/IP protocol stack that has sent Ethernet frames over a NIC, but with macvtap it is user-space code that can do the same
- When the kernel executes the macvtap device driver instead of sending frames directly to a NIC, it makes macvtap available to a user-space code via /dev/tap<N>
- In case of a cloud, the user space code is QEMU
- When QEMU reads an Ethernet frame from the file descriptor of /dev/tap<N>, it emulates what a real network card would do:
  - It triggers an interrupt in the VM, and then the guest OS can read data from the vNIC

5.7.1.4 Inefficient Inter-VM Communication as a Result

- In the next figure, the resulting inefficient inter-VM communication on the same server via KVM/QEMU and macvtap is shown
- As a result, an overhead of 11 sequential steps is needed to accomplish a single inter-VM send-operation



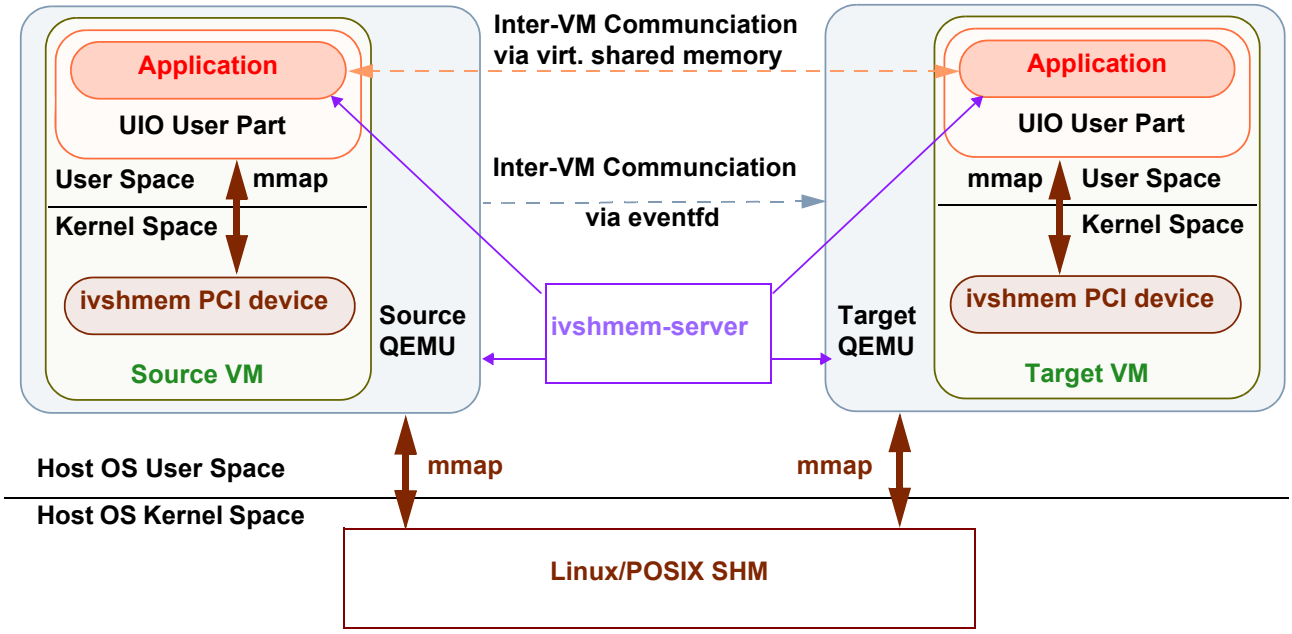
- The same block diagram holds in principle also for the case that additionally Open-Stack is present, but then even more software components are involved as overhead

5.7.2 Inter-VM Communication With Ivshmem on the Same Server

- Ivshmem is the preferred means for inter-VM communication on the same server
- It allows for a zero-copy VM-to-Host communication, which is highly efficient with respect to bandwidth and latency, because no internal data buffer, no TCP/IP and no KVM/QEMU overhead exist
- By going the way from VM 1 to host to VM 2, a shared memory communication between the two VMs comes into existence by means of ivshmem, thus avoiding any copy of data buffers
- In libvirt, an ivshmem between two VMs can be created easily
- Furthermore, ivshmem is compatible with OpenStack and its way to define and launch VMs

Example: If a user creates a VM by means of the Horizon dashboard of OpenStack, a libvirt configuration file is output by the dashboard in xml format. This file has to be post-processed by the user before it is input to KVM and QEMU, and thus before the VM is launched. The post-processing has to add the ivshmem features by a few xml tags, and ivshmem comes into existence.

- In the next figure, the resulting fast inter-VM communication is shown
- It works as follows:
- The address space of the guest-OS application is first "mmapped" to ivhsmem virtual PCI-memory



Comment: mmap is a Linux system call that allows, among other functions, the mapping of a device file, such as from /dev/ or /udev/ e.g., into a user space buffer (=byte array).

- mmap is used to open, read and write the ivshmem PCI device file from user space
- By this, the virt. memory inside of the virt. PCI device can be read and written

- In a second step, the ivshmem PCI device file is mmapped again, but this time to a system space buffer that resides in Linux/POSIX SHM and that really exists in Host OS
- The resulting effect is that opening, reading and writing the user space buffer of an application in a VM, opens reads and writes a system space buffer in SHM from host OS
- If two VMs perform the same pair of mmap calls, two VMs are connected via the same SHM buffer and can perform inter-process communication via shared variables residing in SHM

5.7.3 SHM Access Synchronization

- Each SHM shared-memory requires a read/write and write/write synchronization in case of simultaneous accesses of the same variable in order to avoid reading while writing and to prevent from writing after writing in reverse order
- This access-synchronization is accomplished via an ivshmem server which is also part of QEMU
- The ivshmem server allows that a VM can send a Host OS eventfd (= “interrupt“) to another VM

Comment: An eventfd is a Linux kernel feature that is similar to an interrupt but simpler.

- By sending ready/done eventfds back and forth between VMs, a mutual exclusion of a shared variables can be achieved which implements read/write and write/write synchronization

- Although the shared variables reside in physical Linux SHM, SHM semaphores for mutual exclusion are not possible, because semphores don‘t work between VMs, since they would be only system variables in two different guest OSes
- The result of 1 variable write at the source + 1 read at the target + 2 ready/done eventfds is that ivshmem is much faster than macvtap
- ivshmem is even faster than Linux SHM itself because normally a Linux system call would be needed to access an SHM buffer in kernel space
- This is avoided here, because the kernel space buffer is directly mmapped into user space which is highly intricate, but also highly effective
- This is one of the rare cases where a kernel call is not needed to access kernel memory
- However, in order to avoid the kernel call to SHM, the user applications must be disguised as the user part of a Linux uio device-driver

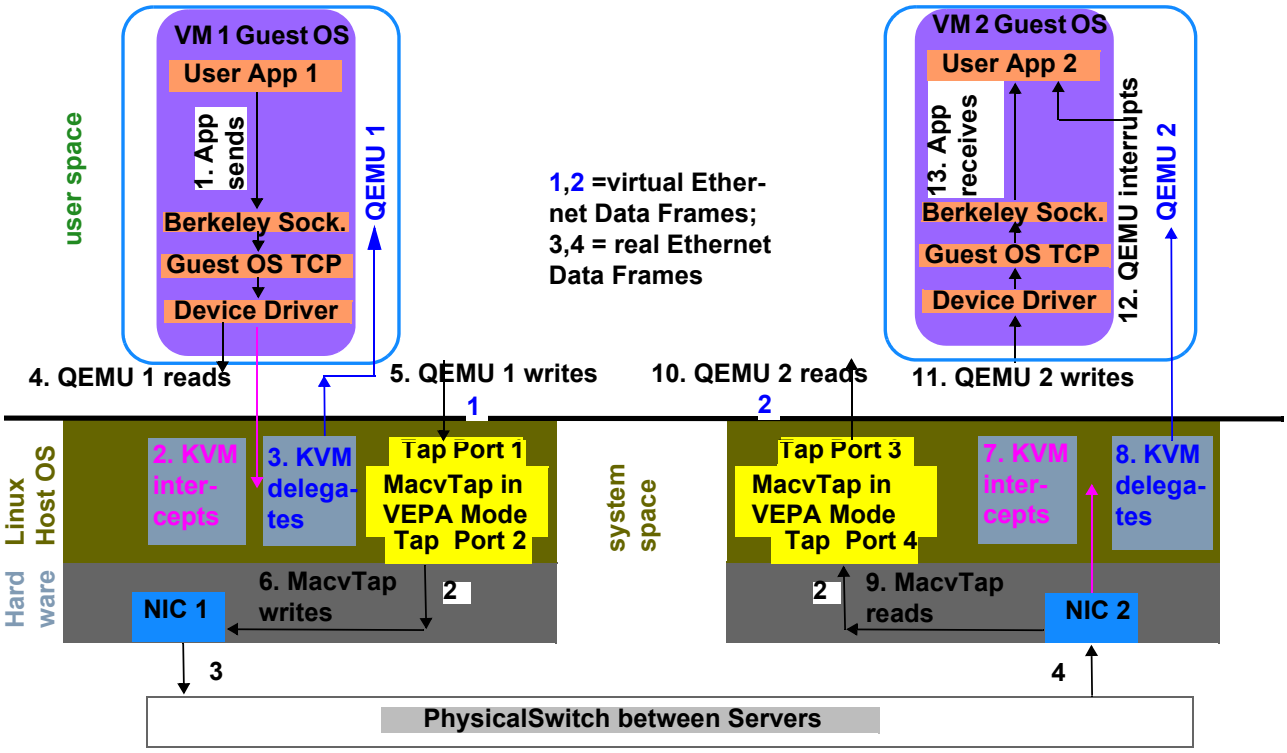
Comment: A uio device-driver is a software that allows to control a physical device from user space. This is possible because the uio device-driver has a companion in kernel space that does the control job as a proxy.

- In order to work, the kernel part of a uio device driver must be written by the user in addition to its application and loaded into the kernel before the first control attempt can take place
- The user part of the Linux uio device-driver does not try to execute a privileged instruction
- Instead, it delegates this to its kernel part

- Because of that tricky construction, KVM does not intercept the user part of uio, although it indirectly controls real hardware, even with the help of host OS
- Finally, the user part of uio allows a so-called blocking read while waiting for an eventfd
- This blocking read does not consume CPU cycles
- Instead the host OS deschules the user part of uio, together with its calling application, and reschedules both automatically when an eventfd has arrived
- This is a highly effective way of communication w.r.t. CPU cycles because no cycle is wasted for waiting in a process

5.8 Inter-VM Communication between Servers without Cloud OS

- Between servers, ivshmem is not possible because there is no shared main memory
- Instead, two physical Ethernet NICs and one external switch are engaged
- A block diagram for the resulting inter-VM communication is given subsequently
- The essence of the block diagram below is that QEMU calls again macvtap, but switches it from bridge mode to so-called VEPA mode which performs non-local communication
- Afterwards, the sending macvtap forwards all virtual Ethernet frames to the physical Ethernet NIC
- According to this block diagram, a total of 13 steps is needed in order to send one TCP packet from the sender on the left side to a receiver on the right side



- This is not efficient

5.9 Inter-VM Communication between Servers with Cloud OS

- If a cloud OS is additionally present the efficiency is even lower for inter-VM communication between servers because of increased software overhead
- In the following two sub-chapters, first the communication efficiency problem in general CSP clouds is discussed and afterwards a specialization for OpenStack is made

5.9.1 Communication Efficiency Problem in CSP Clouds

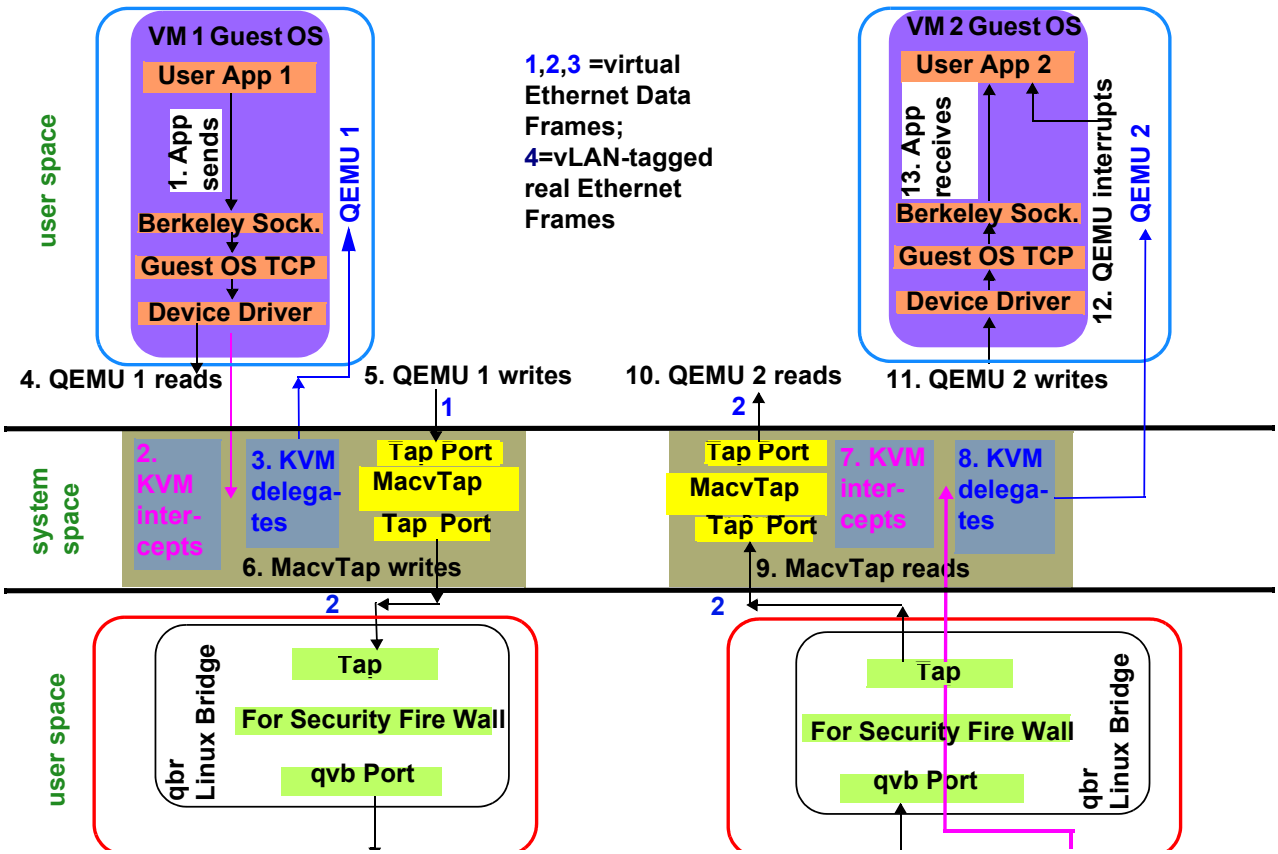
- In Amazon‘s AWS EC2 cloud, only rel. slow inter-VM communication between servers is possible that prevents from HPC and from fast data exchange between VMs
- The same holds for Microsoft Azure, VMWare‘s vCloud and all other commercial clouds

5.9.2 Communication Efficiency Problem in OpenStack

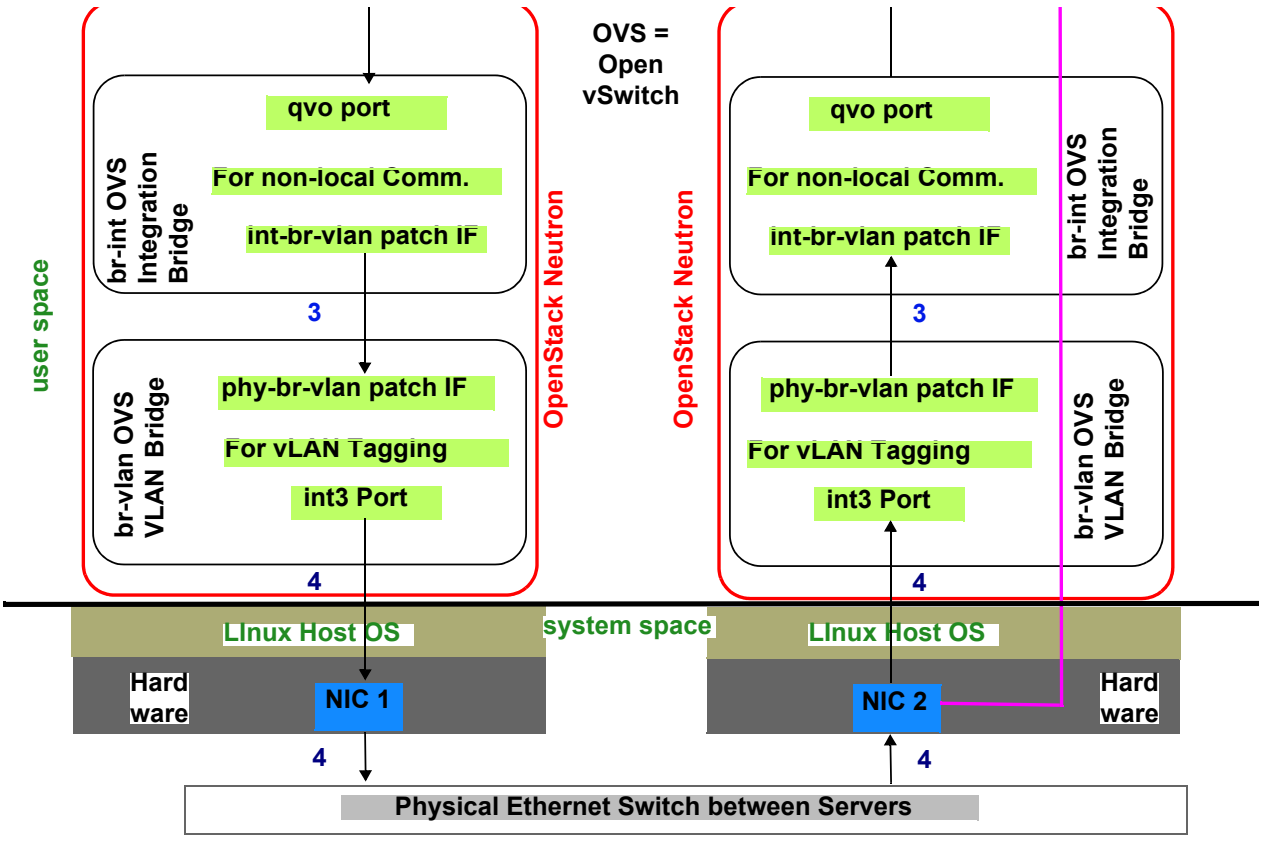
- The next block diagram shows the official way of inter-VM communication between servers in OpenStack

Comment: For reasons of size, this block diagram is distributed onto two foils.

- The diagram illustrates the interplay between two VMs, their QEMUs, KVM and OpenStack‘s Neutron
- The basic rationale of the block diagrams is as follows:


VM 1 Guest OS
User App 1
1. App sends
Berkeley Sock.
Guest OS TCP
Device Driver
QEMU 1
user space
1,2,3 =virtual Ethernet Data Frames;
4=vLAN-tagged real Ethernet Frames
VM 2 Guest OS
User App 2
13. App receives
Berkeley Sock.
Guest OS TCP
Device Driver
12. QEMU interrupts
QEMU 2
4. QEMU 1 reads
5. QEMU 1 writes
10. QEMU 2 reads
11. QEMU 2 writes
1
2
system space
2. KVM inter-cepts
3. KVM delega-tes
Tap Port
MacvTap
Tap Port
6. MacvTap writes
Tap Port
MacvTap
Tap Port
7. KVM inter-cepts
8. KVM delega-tes
9. MacvTap reads
2
2
user space
qbr Linux Bridge
Tap
For Security Fire Wall
qvb Port
Tap
For Security Fire Wall
qvb Port
qbr Linux Bridge

85


OVS = Open vSwitch
user space
br-int OVS Integration Bridge
qvo port
For non-local Comm.
int-br-vlan patch IF
3
br-vlan OVS VLAN Bridge
phy-br-vlan patch IF
For vLAN Tagging
int3 Port
4
OpenStack Neutron
OpenStack Neutron
br-int OVS Integration Bridge
qvo port
For non-local Comm.
int-br-vlan patch IF
3
br-vlan OVS VLAN Bridge
phy-br-vlan patch IF
For vLAN Tagging
int3 Port
4
LInux Host OS
system space
LInux Host OS
Hard ware
NIC 1
NIC 2
Hard ware
4
4
Physical Ethernet Switch between Servers

86

1.) QEMU switches macvTap again in VEPA Mode, but this time macvTap does not write to the physical Ethernet NIC

2.) Instead, MacvTap writes to the Neutron network service of OpenStack which has inserted a thick layer of software between the Tap port of macvTap and the physical NIC

3.) In that layer, vLANs are implemented by Neutron in order to perfectly isolate VMs of different tenants from each other w.r.t. to data exchange

4.) Additionally, a fire wall is added by Neutron to increase cyber security

5.) The listed features are implemented by ancillary software Neutron has which are "Linux Bridge" and "Open vSwitch"

- The essence is that again 13 steps are needed for a single send, but this time a huge software stack of Neutron has to be traversed additionally, compared to the case without cloud OS
- It is clear that OpenStack's official way of inter-VM communication between servers is not efficient at all
- However, with a private OpenStack cloud, it is possible to influence the way inter-VM communication is accomplished

5.9.3 Alternative Inter-VM Communication between Servers in OpenStack

- In a private cloud, the cloud owner can replace a 1 or 10 Gb/s Ethernet by an "Infiniband" interconnect, for example with 50 Gb/s bandwidth and 10 μs Latency

Comment: Infiniband is a commercial alternative to Ethernet used in the HPC domain.

- This compensates in part the software communication overhead by hardware
- However, such hardware installations are not possible in public clouds
- This has to be considered if a decision between a public or private cloud has to be made
- Furthermore, the owner of a private OpenStack cloud can install in each guest OS a communication library that supports Infiniband from user space, such as "MVAPICH2-Virt" from Ohio State University, for example
- MVAPICH2-Virt is a high-performance message-passing software-interface according to the MPI standard

Comment: MPI = Message Passing Interface. It is a wide-spread library for high-performance computing.

- MVAPICH2-Virt supports HPC in a cloud with Infiniband as interconnect and can deal with VMs and containers
- Infiniband as hardware and MVAPICH2-Virt as software are together much faster than the official way of inter-VM communication via Ethernet and Neutron
- Also the software complexity of this solution is lower

- **IF HPC on VMs or fast data exchange between VMs on different server is needed then this could be accomplished by OpenStack + Infiniband + MVAPICH2-Virt**

Part II - OpenStack as Cloud OS

6 Overview on OpenStack and its Services

- OpenStack (http://www.openstack.org) is a large open source project that started in 2010
- Every 6 months, a new release appears
- The update frequency of OpenStack is very high, and each new release can contain fundamental changes
- OpenStack comprises > 20 Mio. lines of code
- This is about the size of the Linux kernel
- OpenStack provides an Infrastructure-as-a-Service (IaaS) by means of "services"§
- It is mostly written in Python 2.6 and Python 3 and thus interpreted

Comment: Python is a popular object-oriented script language from the 1990s. In contrast to C, C++, C# and Java, it is not based on { and } for BEGIN and END but on code indentation.

- Data structures that have to be exchanged to and from OpenStack services are often written in "JSON" or yaml format

Comment: JSON = JavaScript Object Notation. It is a syntax convention for data interchange that is simpler than XML and easier to read. It is not a markup language as XML, but syntactically compatible to JavaScript. Yaml is as JSON but without { and } for BEGIN and END and based on code indentation instead. Therefore, Python and yaml are a perfect match.

- Calling an OpenStack service means often for the user to create such a JSON/yaml object and to use it as input parameter for the service

- OpenStack has more than 45 services with steadily increasing tendency
- The key feature of these services is that most of them, but not all, are calls via a REST-based protocol

Comment: For REST see chapter below.

- Each service with REST communication has an individual API, i.e. an own set of REST-based calls
- Because of REST, OpenStack can be considered to be simply a set of web services
- Those few OpenStack services without own REST API are called cloud-internally by means of a messaging library called "oslo.messaging"
- An other way to look at OpenStack and all other cloud OSes is to denominate them as a distributed system

Comment: The key feature of a distributed system is that its computers and software components are coupled by TCP/IP.

6.1 REST Protocol

- REST is a superset of http because also PUT, DELETE and PATCH are working
- In case of http, these calls are typically not allowed and thus blocked by a web server
- Additionally, an URI (Uniform Resource Identifier) is used instead of an URL

Def.: An URI is a generalization of an URL with addresses of a predefined syntax that are an abstract or concrete web resource instead of a file.

Example: A concrete web resource is a single file that is referenced to via the path and name in the URL of its http request.

Example: An abstract web resource is a user storage such as drop box that is accessible via the the web. This resource is abstract because the user only knows which files he has and how much space is left, but he does not know where his files are physically located.

- Finally, REST is also a set of defined rules ("architectural style") that create a stateless web server
- There are 9 REST calls in total called "requests": GET, HEAD, PUT, POST, DELETE, TRACE, OPTIONS, CONNECT and PATCH
- Many service APIs have three sets of REST calls: for cloud-internal admins, for cloud-internal users, and for the public

6.2 Securing REST Requests and VMs

- All REST requests for the public are secured by https, the others are not
- Instead, the REST requests for admins and for cloud-internal users are secured by short-living certificates
- Typically, an OpenStack REST request to a specific service is made from inside a user project and not from the public Internet
- Each user project comprises multiple VMs that belong to the project members

- The cloud-internal VMs in turn are protected from Internet attacks by a cloud firewall, by vLANs and by floating IP and MAC addresses

Comment: A floating IP is temporily assigned to a specific VM by Neutron from a pool of public IP addresses the cloud has. Afterwards, Neutron can assign the same IP address to another VM.

6.3 Calling and Controlling an OpenStack Service

- Most but not all OpenStack services can be called via its service-specific REST API
- For those services without API, a Universal Unique Identifier (UUID) is used as address instead of a REST URI

Def.: A Universal Unique Identifier (UUID) is a 16-Byte number in hexadezimal notation which is split into 5 groups that are separated by „-". Everybody can create an UUID because it contains data, time and MAC address of the computer where it was created. Additionally, pseudo random numbers or any user names may be included.

Example: 550e8400-e29b-11d4-a716-446655440000 would be a valid UUID.

Example: Each VM has an UUID, as well as each Neutron vLAN.

- In addition to its UUID, a VM has also a name, while a Neutron vLAN has not
- Furthermore, all OpenStack services can be controlled locally via an service-specific CLI

Def.: CLI = command line interface = Command line interpreter for string-based commands.

- **Finally, many OpenStack services can be controlled remotely via a GUI in the "Horizon dashboard"**

Comment: For an explanation of the Horizon dashboard, see the subsequent chapter about it.

6.3.1 Extended UUID Usage for Information Items

- **The UUID addressing scheme is also used to address important information items inside of OpenStack, which are not a service and have therefore no API and no CLI**

Example: Every user-visible software-component on the user-, project- and service levels of OpenStack, as well as every important internal information item have an own UUID.

- **The UUID of an information item allows to reference to it or to call it via OpenStack-internal Remote Procedure Calls or to notify it via oslo.messaging**

Comment: Remote Procedure Calls (RPCs) are the classical way for interprocess-communication in a distributed system.

- **UUIDs for information items and URIs for services complement each other**
- **Finally, those items which have neither URI and nor UUID are referenced by the name they have**

Example: Each Heat stack has a name, every VM has also a name. However, both have also an UUID.

Comment: For an explanation about what a Heat stack is see the subsequent chapter about Heat.

6.3.2 Calling via REST API

- **Each REST request that is issued by a caller to a callee has URI in the request header and an UUID in the request body**
- **The same UUID re-appears in the log files of both, caller and callee**
- **This allows for tracking the service request in order to debug the distributed system that is established by caller and callee**
- **Furthermore, not all 9 REST calls are typically engaged in all services but subsets of it**

Example: An OpenStack service called Murano has only 5 of 9 calls: GET, POST, PUT, DELETE and PATCH. Murano lists the user applications that are installed in the cloud. Each user application is called a "package" in murano.

Example: GET http://myCloudController.de:8082/v1/catalog/packages would by a syntactically valid REST request with http://myCloudController.de as the URI to the web resource where Murano may reside. The port 8082 is the default port for Murano where it listens for requests.

- **The example above would list the installed packages**

- The same effect would be achieved if the full URI would be entered into the browser URL line
- Each REST request must include a port number in its URI where the service is listening
- An (incomplete) list of OpenStack-services and their ports is under: https://github.com/openstack-infra/tripleo-ci/blob/master/test-environments/tls-endpoints-public-ip.yaml

⇒ **Most OpenStack services have at least one host OS port where they are listening for API requests**

- In newer OpenStack releases, the node which has the "Keystone" service installed, contains a host OS file of name /etc/keystone/default_catalog.templates in which the ports for all services are listed

Comment: Keystone is the OpenStack service that provides short-term certificates to other OpenStack services in order to authenticate and authorize them.

6.3.3 Calling via Host OS Shell

- If a user wants to call an OpenStack service by hand from the Host OS shell, then he should use the open source tool "cURL"

Def.: cURL allows to copy the content of an URL or URI to a file and vice versa. "File" as output can also be the own display. "File" as input can be an string or a JSON/yaml object that is entered at the shell prompt by hand.

- cURL generates automatically a GET request if the source is the URI and the destination is the file, and a POST request for the reverse direction
- All other REST methods must be explicitly given as cURL parameter
- cURL option flags are explained under: https://curl.haxx.se/docs/manpage.html

Example: curl -H "X-Auth-Token:01234567" "http://myCloudController.de:13000/v2.0/tenants" is a REST request to the Keystone service that would list all projects (i.e. tenants) of the users in the cloud. -H is the "header option" of cURL, and it is needed to authenticate the call as being issued by an admin. 01234567 is his example certificate that is shorter than a true UUID for reasons of simplicity. Under port number 13000, Keystone is listening for REST requests at the API network.

- An (incomplete) list of OpenStack REST APIs is under: https://docs.openstack.org/queens/api/
- The positive implication of OpenStack being a set of web services is that its calls always follow the same simple scheme: <method> <URI>
- This scheme holds also if the API of a specific OpenStack service changes or if the underlying host OS changes

⇒ **There is a static OpenStack API calling scheme despite changing versions of services and host OSes**

6.3.4 Calling via GUI

- Beside the sets of individual service APIs, OpenStack provides also a GUI via its "Horizon" service

Comment: GUI = Graphical User Interface

- Horizon is a "dashboard" for OpenStack which exhibits different GUIs for admins and users
- Horizon allows to control many OpenStack services and all VMs either directly via its GUI or via service-specific user-plugins that create special GUIs

6.3.4.1 Caveats

- Because of the fact that OpenStack is a set of web services that are called via REST requests, speed and latency of its APIs are low
- As a consequence, clouds in general and OpenStack in special are normally not suited for fast real-time operation or HPC or big data transfers

6.3.5 Calling via CLI

- Finally, in addition to API and Horizon GUI, practically all OpenStack services provide also a CLI for the direct control of them

6.3.6 Calling via UUID

- Calling a service or one of its components via an UUID is accomplished via the OpenStack-internal communication library which is oslo.messaging

Comment: Oslo.messaging provides for inter-process communication in the same node and between nodes. For a deeper explanation see a following chapter about it.

6.4 Overview on OpenStack Services

- By means of its services, OpenStack allows to provide, configure and manage indirectly huge amounts of VMs, virtual hard drives, network components and user applications

Example: In a cluster of 10 Tsd. physical servers, an OpenStack cloud can be installed that hosts hundreds of thousands of VMs.

- The attribute "indirectly" holds for the following reason: each OpenStack service is a user space process only, without system priviledges in the host OS

⇒ A cloud can provide, configure and manage only virtual resources but not real ones

- Furthermore, a distinction must be made whether an OpenStack service pertains to a so-called "ancillary software" or not

Def.: Ancillary software are open source programs that are installed and work on behalf of an OpenStack service.

1.) **If an OpenStack service pertains to ancillary software with own read/write operations on data then the service itself has no OpenStack API call for handling data (no read or write), but only for the provisioning, configuration and management of the service**

Example: The OpenStack service Neutron uses Linux Bridge and OpenVSwitch as ancillary software. The OpenStack service "Nova" uses libvirt and KVM/QEMU as ancillary software.

2.) **The same holds for services that pertain to existing software standards or to user applications that have their own API for reading and writing**

Example: The OpenStack database service "Trove" can provision and manage databases, but they are still accessed by SQL and not by Trove REST calls. This means Trove is not a competitor to SQL, but preserves the existing standard for relational databases.

⇒ **An OpenStack service instead an existing software is not just another API for the same functionality, but it complements the existing API w.r.t. to cluster-wide handling. The OpenStack service does not compete with the existing software, but uses it as ancillary software.**

3.) **If an OpenStack services pertain to new functionalities which can be provided only by the service then the usage of these functionalities and the handling of the respective data with read and write operations is provisioned in its REST API**

Example: A service called Swift can not only provision and manage an OpenStack object store, but it can also read, write and delete the objects inside of it. In fact, the Swift

API is the only way to access these objects. No ancillary software, no standard software and no user application is available that can do that. Therefore, Swift is no competitor to s.th. existing.

Def.: An object store is a web store for objects. Objects in such a store comprise of structured or unstructured data and meta information about the data.

- **The first main purpose of OpenStack, and any other cloud OSes as well, is to extend existing ancillary software and applications that are written for a single computer to a whole cluster of computers by giving powerful tools for cluster-wide provisioning, configuration and management of existing software and apps in VMs**
  - **Data which can be accessed by ancillary software, by a standard software or via a user application is not accessible by the respective cloud service**
- **The second main purpose of OpenStack, and any other cloud OSes, is to allow for new, cluster-wide services that did not exist before**
  - **Only in this case, the REST API includes not only provisioning, configuration and management but also usage and access of data**

6.4.1 Incomplete list of OpenStack services

- An incomplete list of OpenStack services is:

Function	Name	Function	Name	Function	Name
Maintaining an Application Catalog	*Murano*	Databases	*Trove*	Orchestration for Network Virtual Functions (NFV)	*Tacker*
Bare Metal Support	*Ironic*	Access to the Domain Name System (DNS)	*Designate*	Object Storage	*Swift*
Block Storage	*Cinder*	Authentication and Authorization (Identity)	*Keystone*	General Orchestration	*Heat*
Clustering	*Senlin*	VM Image Repository	*Glance*	Searching in the Internet	*Searchlight*
VM Creation and Management	*Nova*	Cloud Infrastructure Optimization	*Watcher*	Shared File Systems	*Manila*
Container Infrastructure Management	*Magnum*	Data-Security-Key-Manager	*Barbican*	Telemetry	*Ceilometer*

Function	Name	Function	Name	Function	Name
Containers	*Zun*	Load-Balancer	*Octavia*	Telemetry Events	*Panko*
Data Processing	*Sahara*	Messaging	*Zaqar*	Workflow Interpreter	*Mistral*
Orchestration for Data Protection	*Karbor*	Networking	*Neutron*	GUI	*Horizon*

- In addition to OpenStack services, there are also OpenStack projects which may cumulate later into a new service, or which will be merged in a later release into an existing service
- OpenStack projects are listed under: https://www.openstack.org/software/
- Finally, there are also OpenStack libraries which are a software infrastructure for services and projects
- The key feature of every service is that it is one central software for all VMs that is specialized on a specific task, and that can be controlled by the dashboard (Horizon)
- This simplifies significantly the management of VMs

Example: In a cloud of 1000 VMs, a new user has to be added to the Keystone service only one time by means of Horizon but not 1000 times.

6.5 The 10 Most-Important OpenStack Services

- **The following is a subjective selection of the 10 most important OpenStack services which were taken from https://docs.openstack.org/queens/projects.html**

1.) Horizon allows to provide, configure and manage VMs and cloud services via a web-based GUI

- **the Horizon dashboard allows the user to login into the cloud**
- **it gives to users and admins the "look and feel" of OpenStack**
- **Horizon has three distinct sets of web pages for users, admins and "settings"**
- **it has additionally an API to many but not to all OpenStack services**
- **it provides for Python classes for easy incorporation of new services and projects**
- **finally, there is a tool called "OpenStack Client" which provides a CLI for many but not all OpenStack services**

2.) A meta scheduler named "Nova" which statically allocates VMs to real servers

Comment: Each operating system has a scheduler that allocates tasks to cores in order to execute them in a round-robin manner. A meta scheduler does not work on the level of the host OS or guest OS, but on the cluster level and allocates VMs to servers, but not to cores inside of them.

- **VMs are created by the user in Horizon, and Nova processes the VM creation request, and schedules statically the requested VM to a specific server in the cluster**
  - **by means of Nova, each VM gets a physical server in a cluster that will execute the VM**
  - **inside of the assigned server, the local host OS scheduler decides which core in which**

CPU will be executing the VM
  - **Nova in turn uses libvirt and KVM/QEMU to implement the VMs**
  - **with each VM, also a virtual hard drive is created as part of each VM**
  - **however, this virtual hard drive is ephemeral, because data will be lost when the VM is terminated**
- **Bare metal servers can also be created by Nova for a specific user or project, and Nova uses the "Ironic" service for their management**
- **Finally, there is some support for containers in Nova**

3.) A network Manager named "Neutron" which manages all virtual network equipment

- **such equipment consists of vNICs, vLANs, virt. switches, virt. routers, virt. gateways, as well as static and floating MAC and IP addresses of VMs**
- **Neutron provides for each VM a vLAN over which it can connect to other VMs in the same cloud or to the Internet**
- **it can operate either on ISO layer 2 or additionally on layer 3**
- **it provides also for a Software Defined Networking (SDN) for virtual network equipment**

Comment: SDN was originally created to ease the configuration and management of real network equipment. It is a concept that is used frequently in compute and data centers. SDN divides the coupling and management of network components inside of the center into two parts: the data plane and the control plane.

4.) An identity management system named „Keystone" which provides for every VM user and for every OpenStack service authentication and authorization

Def.: Authentication is the creation and checking of username/password or of a RSA key for VM login. Furthermore, authentication identifies OpenStack services by an individually created short-term token without it cannot run.

Def.: Authorization is the creation and checking of access rights inside of a guest OS with respect to reading, writing, executing and deleting of files. Furthermore, Authorization allows services to perform specific operations or excludes them from doing so.

- Finally, Keystone maintains a user-accessible catalog where every installed OpenStack service must register, and it enters its URI to that catalogue, so that the service becomes accessible from inside and outside of the cloud

5.) A 1st storage service named "Cinder" which provisions a reliable and distributed file system for disk records in virtual hard drives

Comment: A disk record is a fixed amount of storage of 2, 4 or 8 KB, depending on the respective disk formatting.

Example: Relational databases are optimized for record-based disks.

- Cinder provides, configures and manages virtual volumes for the whole cloud cluster
  - optionally, virt. volumes are accessible from multiple VMs at the same time
  - virt. volumes can also be configured as RAIDs

Comment: RAID = Redundant Array of Inexpensive Disks. The RAID method was originally created for real disk drives. It boosts either reliability or bandwidth of the drive.

- Cinder does not have calls for read, write or delete records in a virt. volume

  - This is accomplished via standard "readf", "printf" and other system calls of the respective Guest OS
- it provides for each volume a cluster-wide unique address (UUID) under which readf, printf etc. are working and can access the virt. volume and its records for configuration and management only

Comment: UUID = Universal Unique Identifier

- it backups whole virt. volumes by means of "volume snapshots"

6.) A 2nd storage service named "Swift" which allows for a reliable and distributed file system for objects in virtual hard drives

Def.: Swift objects are data together with metadata that explain to the user the stored data.

Example: A photo with title, an email with header, or a file-backup, together with the names of the creator and its backup hard drive are Swift objects.

- Swift distributes its objects redundantly over virtual drives in the cloud
- in contrast to Cinder, it allows to read, write and delete data
- it is made for handling Petabytes in big data applications

Comment: A Petabyte = 1024 Terabytes $\approx 10^{15}$ bytes.

7.) A 3rd storage service named "Glance" to store again objects

- In contrast to Swift, Glance objects are stored in JSON format as keyword/value pairs

- Glance is mostly used to store whole VMs in a single file which is called "image"
- Glance can use Swift as storage backend
- it provides also for a repository for VM images in which users can search for a specific VM
- VMs can be launched from existing glance images in the repository by means of Horizon

8.) A 4th storage service named "Trove" that provides, configures and manages both, SQL and "No-SQL" databases as well

Def.: A No-SQL database does not store properties that belong logically together by means of entries in the same row of a table. Instead, it follows other structuring principles that depend on the concrete No-SQL database. There exist various principles for data structuring such as keyword/value pairs or object-orientation.

- Trove uses Cinder as storage backend because SQL databases are optimized for record-oriented storages

9.) A measurement service named "Ceilometer" which allows to collect and display any kind of statistical data from the cloud

- Ceilometer allows to collect performance data for cloud tuning

Example: Typical statistical data are virt. CPU utilization, virt. hard drive accesses per minute or vNIC bandwidths.

- it also allows to collect resource-usage data a CSP may need to bill customers if OpenStack is used as the CSP's cloud OS
- the definition of what has to be collected is given by a user or admin and accomplished by

so-called "counter variables"

- as soon as admin or user has defined a counter variable by a set of rules, Ceilometer updates the counter variable autonomously

10.)A batch processing system called "Heat" that can execute so-called "templates", as well as scripts in various languages, in order to make software-controlled calls to OpenStack services and to host-OS shell-commands

Def.: Heat scripts are named "stack". The supported scripting languages are Chef, Puppet and Ansible.

Comment: Chef, Puppet and Ansible are scripting languages that are optimized for the automation of application deployment and configuration management.

- by Heat, OpenStack services and VMs can be automatically deployed, started and reconfigured

Example: Heat allows to provision at a specific point-in-time a prescribed amount of VMs and increases their number later or decommissions them by auto-scaling, in order to adapt to user habits.

- The listed 10 most important OpenStack services will be outlined in more detail in subsequent chapters

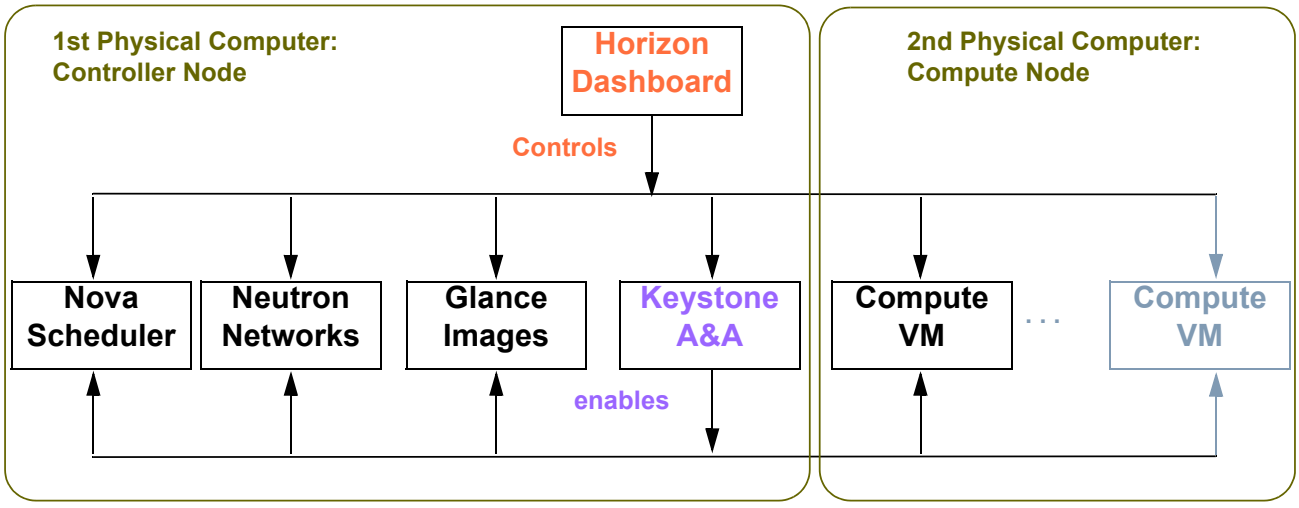
6.6 The Smallest Possible OpenStack System

- The smallest possible OpenStack System consists from a software point-of-view of 5 of the 10 most important services: Nova, Neutron, Glance, Keystone and Horizon
- These services have to be installed at least on one physical server which acts as a so-called controller node for OpenStack
- By means of the controller node, multiple VMs can be launched in turn that act as so-called compute nodes
- Typically, VMs are launched on other physical servers or PCs than the controller node
- As a consequence, the smallest OpenStack System consists from a hardware point-of-view of two physical computers, one that is the controller node and the other that is the compute node with several VMs
- For testing purposes, it is even possible to install the controller node not in a physical computer but in a VM, and to install the compute nodes even in the same VM as nested VMs inside of a VM
- Such an "all-in-one VM" is the truely smallest possible OpenStack system, but this system is far away from a good performance
- To the smallest system, more physical servers with VMs can be added as needed
- It is also possible that some VMs are specialized in a specific OpenStack service

Example: With exception of Horizon, Keystone, Nova and Neutron, all other OpenStack services such as Cinder, Swift, Glance, Trove, for example, can specialize a pure com-

pute VM to a record-based storage-VM, to an object-based storage-VM, to an image VM or to a database VM, respectively.

- In fact, compute VMs on which one or more OpenStack services are installed (with exception of Horizon, Keystone, Nova and Neutron) are specialized compute VMs
- The block diagram of the smallest OpenStack system of performance is as below:



- Inside of each compute node, user applications are executed in VMs
- Instead of VMs, also containers or bare metal servers are possible, but this is still rare
- In addition to general compute VMs, specialized VMs are also possible

6.7 Controller Node and Other Nodes

- In order to be able to control one or more compute node or specialized node from a controller, each of such node must be physically connected to the controller
- Typically, the physical controller NIC is connected to any node NIC via a physical switch
- On top of the physical network is a virtual network that couples controller node services with compute node and specialized node services
- Both, the virtual and the physical networks employ TCP or UDP-based application protocols

6.7.1 "AMQP" and "ZMTP"

- In OpenStack, the main application protocol is REST, but two other application protocols exist as well: "AMQP" and "ZMTP"

Comment: AMQP and ZMTP are so-called "messaging protocols" and will be explained later.

- In order to integrate a VM of a compute node or specialized node into the virtual network of Neutron, a specific component of Neutron must be present in that node

- This part is the "Linux Bridge" which is inserted by Neutron between KVM and each VM's vNIC
- It connects the vNIC of a compute VM to the vNIC in the controller node, either directly in software if it is the same server, or indirectly via a physical switch in case of different servers

6.8 Minimum Software Stack in a Compute Node

- Beside the Linux Bridge for networking, a second software component is needed
- This component is part of Nova and called "Nova-Compute"
- It also is inserted by OpenStack between KVM and the VMs
- The minimum software infrastructure required for a compute node is therefore on the host OS level: KVM, Neutron-Linux Bridge and Nova-Compute
- The latter two are user space processes in the host OS
- On top of that infrastructure, we have several QEMUs with VMs and their guest OSes which are also host-OS user-space processes
- In each guest OS, we have multiple user applications running
- All user apps are also host-OS user-space processes

6.9 Intra-Service and Inter-Service Communication

- Beside the REST API most of the services have, there is one more OpenStack mean for communication which is called oslo.messaging

6.9.1 Oslo.Messaging

- oslo.messaging provides for both, intra-service and inter-service communication
- Intra-service communication is required by large services that consist of several processes as components, so that these processes can exchange data

Example: Nova is a large service that has 4 components and a database. All of them are deamon processes that can even be executed on different physical computers. This is accomplished by means of intra-service communication via oslo.messaging.

- oslo.messaging is an OpenStack library for Remote Procedure Calls (RPCs) and for notifications

Def.: A RPC is a function call a client makes at computer X that is transmitted over the Internet to a RPC server Y which executes it. The result is returned to the client at X, thus establishing a two-way communication. Typically, the RPCs is blocking the client as a local function call would do, until the result has returned.

Def.: A notification is a one-way communication from a sender to one or more clients in the form of a message. Typically, notifications do not block the sender because they are processed asynchronously by the receiver.

- This means that intra-service communication is possible either by synchronous RPCs or by asynchronous notifications

Example: Nova uses RPCs and notifications, ceilometer uses notifications only.

- RPCs and notifications are directed to other components of the same service or to other services
- In oslo.messaging, RPC and notifications are implemented by sending one or more messages either by means of AMQP or alternatively by ZMTP

6.9.1.1 Advanced Message Queuing Protocol (AMQP)

- AMQP is a protocol for connection-oriented message-exchange on ISO layer 7
- It supports a so-called “message broker“ to publish messages to those receivers that have subscribed to it
- Messages are queued at the message broker in a mail-box-like manner to allow for asynchronous communication between it and the receivers
- AMQP allows also for routing between multiple message brokers that may be located between sender and receivers
- Additionally, it provides for flow control on a message base between sender and receiver to avoid receiver overrun
- Finally, it supports authentification of the sender at each receiver and encryption of the message itself

- In case of transmission problems, a message is delivered either at-most-once, exactly-once or at-least-once, depending on AMQP's setting
- For better interoperability between heterogenous computer systems, AMQP converts variables of standard data types into an own intermediate exchange format and transmits them serially as bytes (= marshalling)
- Additional metadata allow to annotate this own intermediate format in order to create user-specific data-types as extensions
- In OpenStack, AMQP is implemented by RabbitMQ as ancillary software which is a AMQP message broker

6.9.1.2 ZeroMQ Message Transfer Protocol (ZMTP)

- ZMTP is as AMQP an ISO layer 7 protocol for message exchange, but without waiting in queues of intermediate message brokers and with other features

Comment: ZeroMQ = Zero Message Queue, i.e. no waiting in a message queue.

- Thus, ZMTP complements AMQP but does not replace it because it is made for other use cases
- As AMQP, ZMTP supports data security as well as metadata for data type extensions
- However, it generalizes the Berkeley sockets on layer 7 from one-to-one communication means to many-to-many transmission by allowing to define sets of senders and sets of reveivers

Comment: Berkeley sockets are the standard mean of communication with the Internet for any system and user process in any operating system.

- Between a sender set and a receiver set, three transaction schemes are supported which are publish-subscribe, request-response and push-pull
- Publish-subscribe allows to connect a set of message publishers to a set of subscribers
  - each publisher can send a message to all subscribers
  - and each subscriber in turn can receive messages from all publishers
- Furthermore, request/response transactions allow to forward a task processing request from a client in the client set to a server in the server set
  - this is well-suited for implementing RPCs
- Additionally, push/pull transactions allow one client to scatter at the same time a set of task processing requests to multiple servers and to gather the results back later automatically
  - this is well-suited for implementing distributed programming due to the master-slave principle
- In OpenStack, ZMTP is implemented by ZeroMQ which is part of the oslo.messaging library

6.9.1.3 Notifications

- A notification is sent by its originator via a so-called “message bus“ by means of a Python “notifier class“ which is also part of oslo.messaging
- If AMQP is used, there is one message bus per message broker and a routing of notifications between multiple brokers, if existing, i.e. between message buses takes place
- A notification consists of two parts: an “envelope“ with a fixed structure defined by oslo.messaging and a “payload“ defined by the originator which is emitting the notification

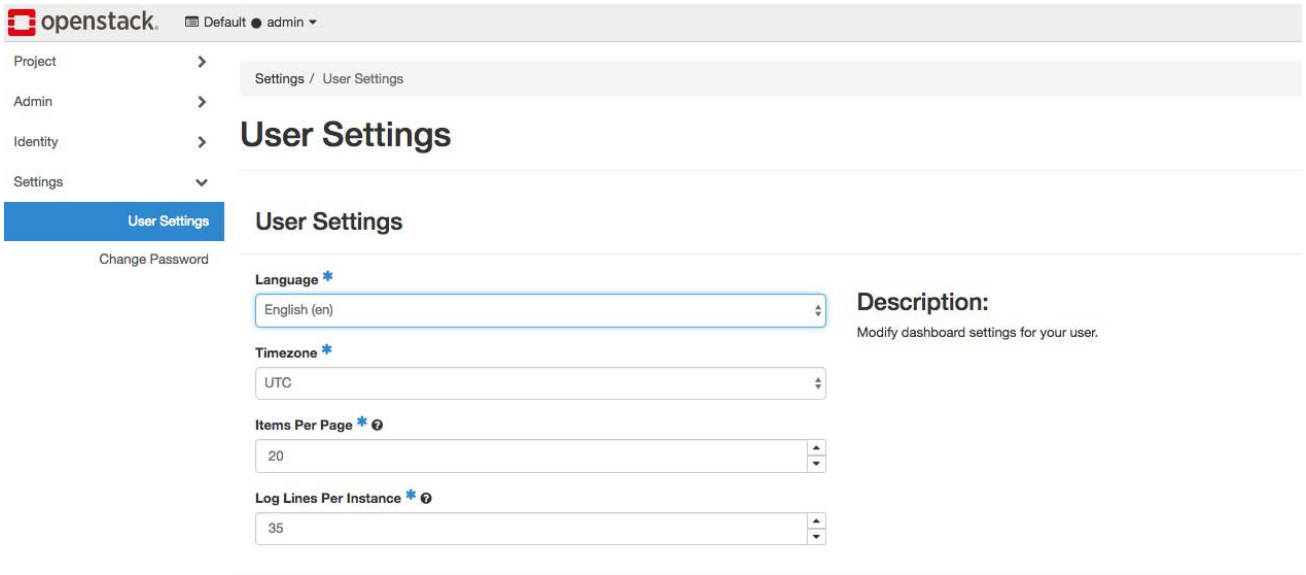
6.10 The Horizon Service

- Horizon is web-based and uses the Django framework

Comment: Django is a popular Python framework for the creation of web pages with a database behind.

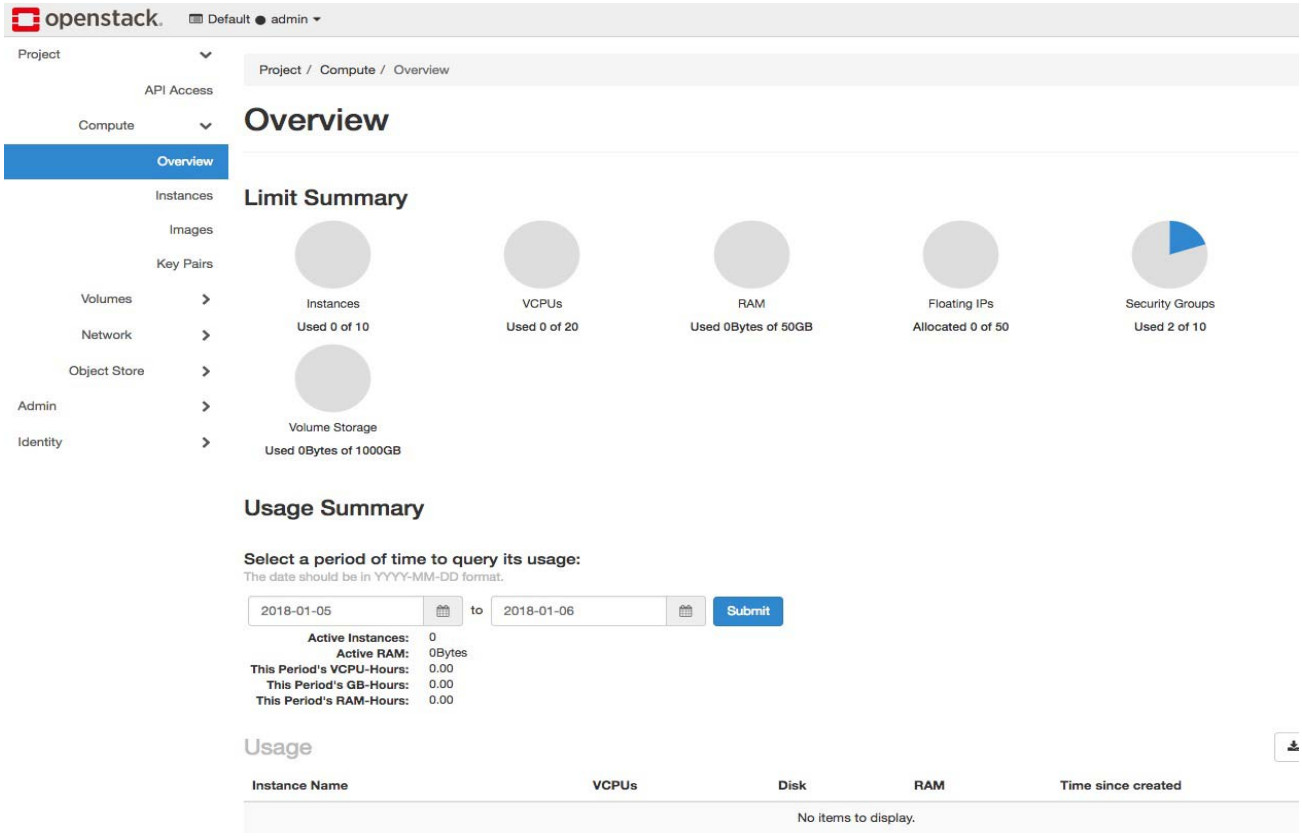
6.10.1 Horizon GUI

- The first web page that may be displayed after login is the “settings tab“:

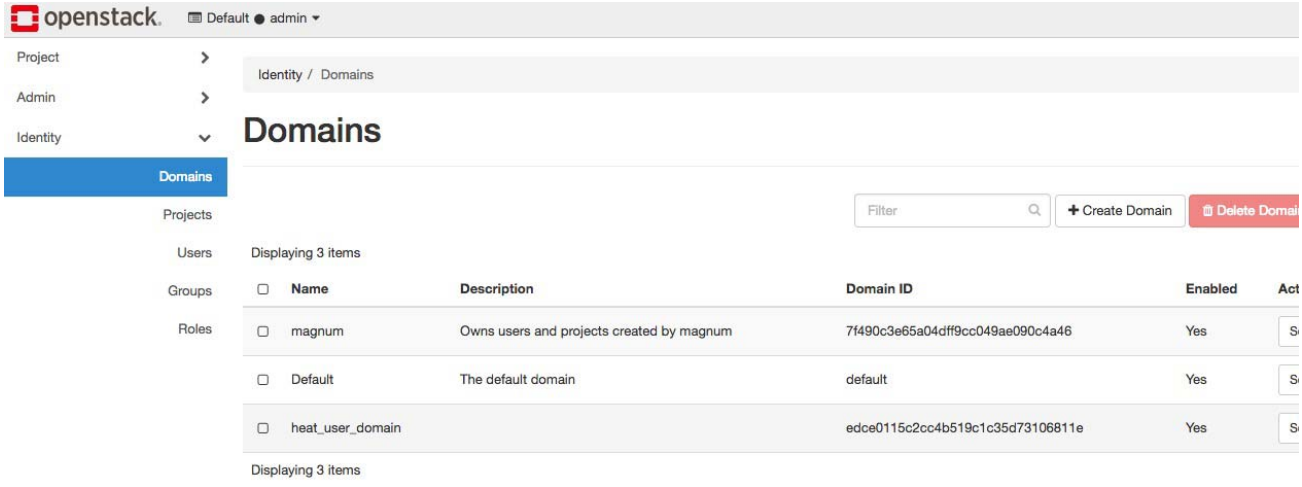


- From the settings tab, the “project tab“ and the “identity tab“ are visible to the user

- **The user web-page overview of the project tab looks as follows:**



- **The user web-page of the "domains" of the identity tab is:**



Comment: Domains are administrative units of the keystone service. A domain comprises a set of projects (tenants) and their users. Domains are similar to a company's main departments. A cloud user can get the role of the administrator for that domain, which is similar to the head of a main department, because he/ she can create projects, users, and groups inside of the domain and can assign roles to its users and groups they have to fulfil.

Comment: A group is an administrative unit inside of a domain. While domains are similiar to main departments in a company, OpenStack groups are similar to the work groups (=departments) of a main department. Both, domains and their groups are methods to fine-grain structure the roles of its members by means of their access rights.

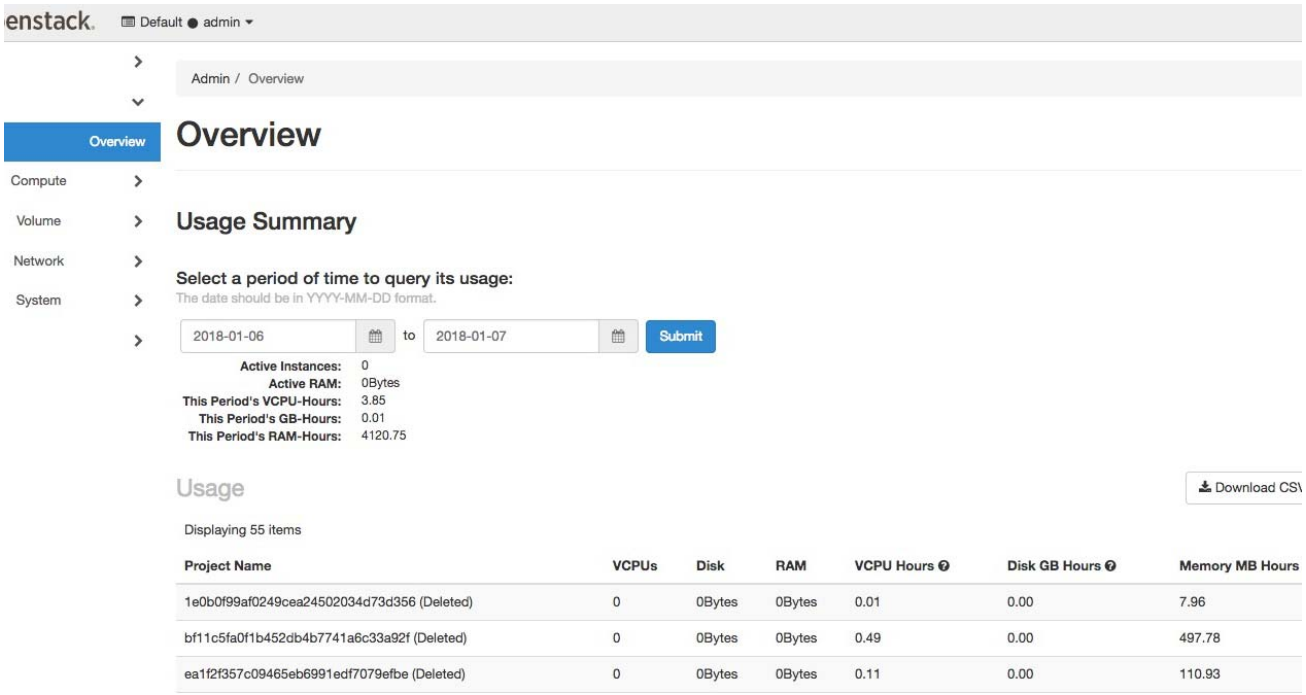
Comment: A role is a specific set of user rights and privileges so that the user is able to perform a predefined set of operations in the cloud. Keystone issues a token to a user that includes a list of roles. When a user calls an OpenStack service, Keystone checks first whether the user is entitled to do so. Furthermore, a role can be applied to all users of a group. Then it is called group role.

- **Admins have similar, but not identical tabs and sub-tabs because more powerful tabs and options are available to them**

Example: The options "delete networks", "delete routers" or "delete instances" are only available to an admin, as well as the admin tab.

Example: The compute tab for admins has different sub-tabs compared to the user. Also the overview tab is different.

Example: Admins have the following overview GUI:



6.10.2 List of Tabs and Sub-tabs for the User GUI (not the Admin GUI)

- **The following list is from https://docs.openstack.org/horizon/latest/user/log-in.html**

6.10.2.1 Compute Tab

1.) **Overview**: View reports for the own project

2.) **Instances**: View, launch, create a snapshot, stop, pause, or reboot instances, or connect to them through VNC

Comment: VNC = Virtual Network Connection for a terminal emulation at a local computer for a communication to a server

3.) **Images**: View images and instance snapshots created by project users + any images that are publicly available

- Create, edit, and delete images and launch instances from images and snapshots

4.) **Key Pairs**: View, create, edit, import, and delete key pairs

Def.: "Key pairs" comprises only the public key of an OpenSSH key pair for accessing a VM during login. The private key part is secret, also to the admin.

6.10.2.2 Volume Tab

Comment: "Volume" means virtual hard drive.

1.) **Volumes**: View, create, edit and delete own volumes

2.) **Backups**: View, create, edit and delete own backups

3.) **Snapshots**: View, create, edit and delete own volume snapshots

4.) **Volume Groups**: View, create, edit, and delete a group of volumes

Def.: A volume group is used in Cinder for managing a set of volumes in the same way.

Example: Snapshots and backups for disaster recovery can be accomplished for a whole volume group. Without defining a volume group, admins cannot easily provide crash-consistent data-protection across multiple volumes.

6.10.2.3 Network Tab

1.) **Network Topology**: View the own virtual network topology

2.) **Networks**: Create and manage public and private vLANs for the own project's or domain's virtual network topology

3.) **Routers**: Create and manage virtual routers for the own project or domain

4.) **Security Groups**: View, create, edit and delete security groups and security group rules

Def.: All packet traffic that is inbound to and outbound from a VM is checked by Neutron whether it matches the firewall rules of Linux Bridge. Only data frames and packets that match the rules can pass. In case of mismatch, Neutron blocks ports, port ranges, or traffic types for that VM.

Def.: An OpenStack security group is a set of vNICs from VMs with the same firewall rules. A vNIC from a VM can belong to one or more security groups.

5.) **Floating IPs**: Allocate dynamically an IP address to or release it from a project

6.10.2.4 Object Store Tab

- **Containers**: Create and manage containers and objects

Comment: So far, containers are only weakly supported by Horizon. Physical machines are not supported at all.

6.10.3 List of Tabs and Sub-tabs for the Admin GUI

- The following list is from https://docs.openstack.org/horizon/latest/user/log-in.html

6.10.3.1 Overview Tab

- **Overview**: Display basic reports about the cloud

6.10.3.2 Compute Tab

1.) **Hypervisors**: Display a hypervisor summary

2.) **Host Aggregates**: View, create and edit "host aggregates"

Comment: For host aggregates, read this subchaper.

- also display the list of "availability zones"

Def.: Availability zones do not share the same hardware with each other, in order to provide for better resilience against hardware faults.

- The computer centers that implement a cloud can be distributed around the world
- Each computer center defines an own availability zone
- Each availability zone has independent access to Internet, power, and cooling infrastructure to ensure uninterrupted access to data and services
- An admin can subdivide a cloud into availability zones which appear to the user as disjunct subsets of VMs
- Each VM can only be in one availability zone at any time
- The user can see in which availability zone his VM is located
- Only the VMs of his availability zone(s) can be accessed by him
- The advantages of availability zones are:
  - There is a full isolation between them, such that natural desasters and power outages do not tear down the whole cloud
  - User data can be redundantly stored in different availability zones which increases the cloud's reliability significantly
- Technially, the availability zone is managed by metadata attached to so-called "host aggregates"

Def.: An admin can further subdivide an availability zone into non-disjunct subsets which are called host aggregates. Host aggregates are only visible to admins. A VM can belong to multiple host aggregates inside of the same availability zone.

- The advantages of host aggregates are:

- The admin can assign the same public/private OpenSSH key pair to all VMs of a host aggregate by one Horizon command
- The admin can migrate all VMs of a host aggregate by one command
- Nova can allocate all VMs of a host aggregate to the same physical server

3.) **Instances**: View, pause, resume, suspend, migrate, soft or hard reboot and delete VMs
   - view the log of a VM
   - access a VM through VNC

Def.: VNC = Virtual Network Computing. This is a software that displays the windows of a remote computer on the own computer, and that forwards the own mouse and keyboard information to the remote computer. This allows to work at a remote computer while sitting at the own computer.

4.) **Flavors**: View, create, edit and view extra specifications for "flavors" and delete flavors

Def.: A flavor is the size of a VM w.r.t to number of CPUs, cores per CPU, main memory, disk space and number of vNICs.

5.) Images: View, create, edit properties for images and delete custom images

6.10.3.3 Volume Tab

1.) **Volumes**: View, create, manage and delete volumes

2.) **Snapshots**: View, manage, and delete volume snapshots

3.) **Volume Types**: View, create, manage and delete volume types

6.10.3.4 Network Tab

1.) **Networks**: View, create, edit properties for virtual networks and delete them

2.) **Routers**: View, create, edit properties for virtual routers and delete them

3.) **Floating IPs**: Allocate dynamically an IP address to a project or remove it

6.10.3.5 System Tab

1.) **Defaults**: View default "quota values"

Def.: Quotas are coded in Nova-Compute and define the maximum allowable size and number of resources

Example: Display the max. main memory, max. disk space, max. number of vCPUs per CPU, max. number of CPUs per VM, ...

2.) **Metadata Definitions**: Import namespace and view its metadata information

Def.: A namespace is generally a method to limit the validity scope of identifiers ("names") to specific areas inside of a system. Namespaces allow to reuse the same name (identifier) in different contexts without creating confusion, because the namespaces of the same name are different.

Example: Linux has namespaces for processes and for network interfaces. They are used in Linux control groups (cgroups).

Example: In OpenStack, Neutron uses namespaces to restrict identifiers to a certain router or to a specific protocol.

- Metadata are generally used to explain the semantics of data

Example: In OpenStack, VM images can have metadata that explain the exact image type. This helps Nova to schedule the image after it is launched as a VM. For that purpose, Glance has an own metadata definition service.

3.) System Information: Provides the following tabs to display information about services

- Services: Display a list of all installed services
- Compute Services: Display a list of all Nova services installed on compute nodes

Comment: Compute service = Nova. Useful command to find out which of the 5 processes of Nova are running on which compute node.

- Block Storage Services: Display a list of all Block Storage services

Comment: Block Storage service = Cinder. Useful command to find out which process components of Cinder are running.

4.) Network Agents: Display all network agents Neutron has in the cloud

Example: Neutron-dhcp-agent, neutron-L3-agent, neutron-metering-agent, neutron-LBaaS-agent, ...

Comment: LBaaS = Load Balancing as a Service.

6.10.4 Horizon Projects (Tenants) And User Authorization

- A Horizon project is identical to the "tenant" notion we used previously
- Properties of tenants/projects are:
  - Each user has a "role" in the project he is member of
  - For each user role, partly or full access rights are given by Keystone
    - full access rights means to create, read, update and delete (CRUD) a project file in Cinder or a project entry in Trove or a project object in Swift or Glance
    - full access rights means also to call in the project API requests of all services
    - finally, full access rights means to use all virtual network equipment the project has, such as vNICs, virt. switches and routers
    - Admins have also full access rights, but for all projects

6.10.5 Role-Based Access-Control

- The role each user has in a project is the prerequisite for the Role-Based Access-Control the user has in OpenStack (RBAC)

Def.: RBAC means that each user role implies an own set of access rights as user authorization.

- If a user is member of multiple projects, then he has a set of roles with individual access rights per project

6.10.6 Extensions to Horizon

- Horizon allows for user applications and new services to “register“ an own GUI as dashboard that becomes part of Horizon after registration
- This means that a new GUI can be “hooked“ into Horizon‘s Python code in order to extend it
- Extensions to Horizon are alleviated by Horizon‘s “code templates“
- Additionally, new panels or whole panel groups can be added to Horizon without the need to modify Horizon‘s default settings
- Furthermore, Horizon allows also for “pluggable settings“
- Pluggable settings mean that new Horizon configurations can be stored in separate files
- Those files are read by Horizon at startup time and extend its default settings
- Finally, Horizon code is well structured because its web pages consist of many panels and not of one big panel, i.e. Horizon is modularly structured
- This allows also to augment and customize Horizon easily

6.10.7 Abstract Service API of Horizon

- Horizon has a set of methods to communicate with the most important OpenStack services without knowing their concrete APIs
- These methods form sets of “abstract APIs“ which are independant of new API releases of that services

⇒ If some service has a new API release the consequence is not that a new Horizon release must accompany it because its abstract API remains the same

6.10.8 Horizon GUI Terminology

- Horizon can present multiple GUIs to the user which serve as dashboards for different services and user applications
- Horizon‘s GUIs follow a special terminology that describe a software hierarchy:

1.) The top-level of the hierarchy are the dashboards
   - Each dashboard has its URL and must be registered at Horizon

Example: Dashboard.py is the base class that implements the base GUI of Horizon.

   - Each dashboard comprises multiple panel groups

2.) A panel group contains multiple panels which establish together a drop down menu
   - a panel group has click-able tabs as elements
   - a panel group is configered in dashboard.py

3.) Panels are the main components of each web page of Horizon
- each panel is implemented via an own Python class

⇒ No Horizon class is overwhelmingly big

- each panel code is located in an own directory and has a standardized structure
- all panels are configured in the joint file dashboard/panel/panel.py

4.) A tab group is a set of tabs within a panel
- a tab group is configered in tabs.py

5.) Each tab has an own internal data set

6.) A workflow is a sequence of internal steps in Horizon's control flow that finally enable Horizon to accept new input from the user

7.) A single workflow step results typically in one internal action

Example: LinkAction, FilterAction or DeleteAction are individual workflow steps.

- a SQL database remembers which actions have to be taken as steps in a workflow
- a SQL DB private to Horizon can be engaged or the general DB of the whole cloud can be used for that purposse

8.) Tables are used by Horizon as internal data structures for storing information and for presenting parts of them to the user

Example: A LinkAction enters a new object into a table, a FilterAction selects what will be displayed to the user from the table, and a DeleteAction removes an object from the table.

- The tables for workflow steps are also stored in the SQL DB

9.) Finally, Views correlate and connect a table with a panel
- Views are responsible for which part of the table is displayed to the user via the panel
- Views are configured in view.py
- view.py is based on utils.py which is a Django class

Comment: Django is the standard Python framework for web services.

6.11 General OpenStack Terminology

❑ Beside Horizon, also whole OpenStack has its own terminology:

1.) "Drivers" play an important role in OpenStack, but OpenStack drivers are s.th. different as usual
- Outside of OpenStack, the term driver pertains to "device driver" and is part of an operating system

Def.: OpenStack Driver: In an OpenStack service or in a component of it, an OpenStack driver written in Python is used to run a host-OS shell-command or to start any other ancillary software that may be not in Python.

Example: There is an OpenStack AMQP driver for the RabbitMQ message broker. The latter is written in the Erlang language.

Example: There are about 55 OpenStack drivers for 55 different volume types from various commercial vendors which are Python proxies or wrappers for the real device drivers written in C for Linux as OS.

2.) "Managers" are also important for OpenStack

Example: For the SQL DB of an OpenStack service, a manager is responsible for updating entries. Components or services which want to change entries in the database of an other service or components of it must engage the manager of that database to do so.

Example: Services that deal with Cinder volumes, must call methods of the Cinder VolumeManager instead of changing fields in the Cinder database directly. This allows to keep all code related to Cinder volumes inside of Cinder.

- **The concept of managers is generalized in OpenStack as follows:**

Def.: OpenStack Manager: Rather than attaching Python methods to Python objects, services should call manager methods that act on these objects.

Comment: This schema extends the concept of object-oriented programming.

3.) Furthermore, OpenStack APIs and API components are complete host OS processes rather than application programming interfaces only

Comment: API stands normally for Application Programming Interface

Example: The Nova-API component is a manager that listens for RPCs and for REST calls.

Def.: An OpenStack API is an own software component running as host OS process with various internal functions for implementing the common sense API for a service. Typically, an API component contains a manager for its internal functions.

- **Additionally, OpenStack used the term "node" and "instance" in a different sense as usual**

Def.: Each physical server is called a node. Each VM is called instance.

  - **Each node can host multiple VMs (=instances)**

- **Finally, the use of synchronous RPCs and asynchronous notifications are major concepts of OpenStack**
  - **If a remote method for a Python object must be called by a service, this must be accomplished via a RPC to that service that hosts a manager for the needed method**

6.12 The Nova Service

- **Nova is one of the first and most important services in OpenStack**
- **Nova has therefore about 200 REST API calls for various functionalities**

- Beside scheduling VMs and containers to physical servers and apart from providing ephemeral disk storage inside of a VM, Nova implements the concepts of availability zones and of “compute cells“

Comment: For Nova compute cells, see next chapter.

- By these two concepts, OpenStack becomes scalable from small to large numbers of VMs
- Because of availability zones and compute cells, OpenStack becomes also resiliant against failures of individual computer-center locations a CSP has for its OpenStack cloud

6.12.1 Nova Compute Cells

- Small OpenStack installations are possible in one Nova compute cell only
- For large OpenStack clouds however, the single private SQL DB of Nova becomes a bottleneck, as well as the single message bus of that AMQP message broker that is responsible for Nova
- Both issues can be solved by means of Nova compute cells
- In each compute cell, there is an own Nova database and an own RabbitMQ message broker that provides for an own message bus with queue
- Furthermore, each compute cell establishes an own namespace for OpenStack identifiers

⇒ **Because of Nova compute cells, the name of a VM is not longer enough information to find it**

- Instead, the respective Nova DB that stores metadata for a VM, as well as the message bus that is responsible for a VM must be given as additional address information
- If multiple Nova compute cells are present in the same big cloud, then a “global“ Nova SQL DB is needed to store “global“ Nova data in parallel to local data in local Nova DBs

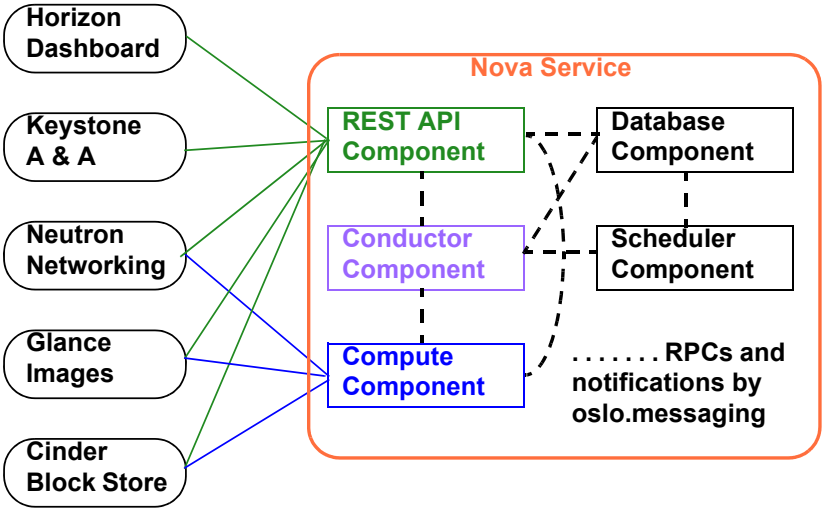
Example: Global Nova data are metadata for VM_types i.e. flavors, VM_projects, VM_type_extra_specs, general quotas, project_user_quotas, quota_classes, quota_usages, security_groups, security_group_rules, security_group_default_rules, provider_fw_rules, RSA key_pairs and migration network tags.

Example: Local Nova data are metadata for VMs, VM_info_caches, VM_extra, VM_metadata, VM_system_metadata, VM_faults, VM_actions, VM_events, VM_id_mappings, VM pci_devices, VM block_device_mappings and VM vNICs.

- The global Nova SQL database is called “API” database, while each local database is called “cell1 DB“, “cell2 DB“, “cell3 DB“, ...
- VMs that failed to start are stored locally in a “cell0 DB“ of every cell for later debugging purposes

6.12.2 Nova Internal Setup

- Nova consists of the following 5 host OS processes as components: API, scheduler, conductor, compute and database:



- Nova-API component: receives REST requests from other services, as well as RPCs from other components of Nova and forwards them via oslo.messaging to those Nova components which can process them

- Nova-Scheduler: decides which physical server gets which instance
  - The scheduler is a complex component that even has an own API beside the main Nova API
    - the scheduler API is called "Placement API"
    - the scheduler takes for its placement decision into account, for example, which flavor the VM to be allocated has and how far it will be away from the user with respect to network latency
- Nova-Compute: runs on each compute node while all other components are executed by the central controller node
  - triggers the local KVM/QEMU of the compute node to create, launch and emulate a new VM or to decommission a running VM
  - triggers also KVM/QEMU to create and emulate a virtual disk inside of the new VM
- Nova-Database: stores the flavor and server allocation of every scheduled VM
  - is a standard open-source SQL database such as MySQL
- Nova-Conductor: the Nova-conductor is the central counterpart for Nova-compute-components in the compute nodes
  - it is the manager of the local Nova database, i.e. it acts as a proxy for SQL queries from a Nova-compute component
  - it handles notification requests that need coordination between other Nova components
  - it handles Python object conversions

6.12.3 Parallel Processing Inside of Nova

- With exception of the Nova database, the API, the scheduler and the conductor can exist also as multiple parallel processes on one or more physical servers acting as controller nodes, in order to increase Nova's throughput
- This ensures scalability from small to large clouds
- There are as many compute components in the cloud as compute nodes exist, regardless whether they are specialized or non-specialized

6.12.4 Communication Between Nova Components

- Nova components communicate with each other via RPCs and by notifications that are part of oslo.messaging

Example: Nova has about 50 different notifications.

- Inside of each Nova component exists a RPC manager that listens for remote procedure calls from other components
- Each RPC manager is connected via a message bus to a RabbitMQ broker for AMQP
- Nova RPCs are implemented via AMQP and have thus message queues
- More precise: the publish/subscribe feature of AMQP is used to implement Nova RPCs
- Because of that, Nova RPCs do not block, i.e. they are asynchronous
- Normally, OpenStack RPCs are synchronous, i.e. blocking

- Furthermore, the RPC client also does not know where the server of the call is located, i.e. which oslo.messaging ID it has, because AMQP handles and forwards the RPC call

⇒ Nova RPC servers are anonymous to Nova RPC clients

- Furthermore, Nova RPC client calls are load-balanced between Nova RPC servers, i.e. between parallel processes of the same Nova component
- Each of the Nova component processes has also a CLI for a direct manual control of it

6.13 The Neutron Service

- Beside Nova, Neutron is the 2nd most important OpenStack service
- Its API has a complexity comparable to all APIs together Nova components have, e.g.
- Neutron has many tasks which are implemented by the help of ancillary software, plug-ins and OpenStack drivers and real device drivers

6.13.1 Neutron Task List

- A list of Neutron tasks follows that is needed for the creation and management of the virtual network equipment in a cloud:
  - tasks on ISO layer 2:
    - bridging between two VMs which communicate with each other in the same node via TCP/IP and "Linux bridge"

Comment: Linux bridge is a kernel module that connects two NICs or vNICs. A Linux bridge also allows to plug a 1st VM via its vNIC into the vNIC of a 2nd VM that has already an Internet connection by means of a 3rd vNIC. Then, the 1st VM can use the Internet through the 2nd VM as intermediate stage.

- switching between multiple VMs which communicate with each other in the same node via TCP/IP and "Open vSwitch"

Comment: Open vSwitch is a Linux kernel module that emulates a physical switch in software.

- for vLAN creation and vLAN ID tagging to isolate different tenant vLANs from each other that reside on the same physical LAN
- for the provisioning of MAC addresses to vNICs of VMs

Lemma: it is Nova not Neutron which is required to plug a vNIC of a VM into a particular Neutron vLAN, because the vNIC is emulated by KVM/QEMU which in turn is controlled by Nova.

- tasks on ISO layer 3:
  - for the provisioning of IP addresses and other network parameters, such as the gateway's IP address which the VMs must use in case of DHCP

Comment: DCHP = Dynamic Host Configuration Protocol.

- for routing IP packets from vNICs to physical NICs via virtual network equipment
- for the creation of Virtual Extensible LANs (VXLANs) to increase the number of available vLANs in very large OpenStack clouds
- for Network Address Translation to replace the private IP address of a VM by a public IP address that is valid for the Internet (=source Network Address Translation, NAT). Source NAT is accomplished by the L3 agent of Neutron.

Comment: For each cloud, there is only a limited set of public IP addresses which is typically much smaller than the whole amount of VMs the cloud has. This is why public IP addresses are "floating" between VMs. Floating means that the private IP address of a VM is translated temporarily into a public IP address by means of "source NAT", and after a while, an other private IP address of a VM is translated temporarily into the same public IP address.

Comment: NAT = Network Address Translation. There exists source NAT and destination NAT. By means of source NAT, the IP v4 address of the computer that starts a TCP/IP transfer is changed into a public IP v4 address. In destination NAT, the IP v4 address of the computer that is target of a TCP/IP transfer is changed into a public IP v4 address. Both address rewritings are made because of shortage of IP v4 addresses.

- for booting a VM via a specific vNIC and a private or the public network
- for monitoring IP traffic via Netflow

Comment: Netflow is a software that instruments a router to collect IP addresses and layer 4 port addresses and other L3 and L4 information about routed packets with respect to the traffic load.

- for tunneling through the Internet to a private subnet via VPN

Comment: VPN = Virtual Private Network is a software that allows to connect the own computer to a remote private network via the Internet as if the own computer were directly connected to that private network.

- tasks on ISO layer 4:
  - plug and unplug virtual ports

Def.: Whenever a vNIC of a VM, a switch or a router is connected to an OpenStack vLAN, then that connection is called a "port". OpenStack ports are not identical to Linux ports,

because the latter are associated with a TCP/UDP port number, and because one can allocate Internet IP addresses to OpenStack ports.

- tasks on ISO layer 7:
  - for establishing OpenStack security groups. They are the basis for firewalls.
  - for creating a Security Fire Wall as a service (FWaaS)
  - for establishing a Load-Balancer for data traffic as a service (LBaaS)

6.13.2 Neutron Software-Defined Networks

- The aforementioned functionalities allow Neutron also to implement virtual Software-Defined Networks (SDNs), either for ISO layer 2 or for layers 2 and 3 combined
- In many computing centers, real SDNs are the classical way for configuring physical communication equipment such as real switches and routers
- The fact that Neutron supports virtual SDNs is important for these computing centers because its allows them to continue with their well-established SDN methods and software, also in the virtual world

Def.: Each virtual SDN consists of a "data plane" which is responsible for data transmission and of a „control plane" which is responsible for managing the underlying virtual hardware. Each data plane is technically a set of Neutron vLANs coupled by virtual switches and optionally also virtual routers.

6.13.3 Arbitrary Network Topologies by means of Neutron

- In addition to virtual SDNs, whole connected sets of virtual networks (vLANs) with arbitrary topologies can be created and managed by Neutron that include also virtual routers
- Virtual L3 routers are needed for large clouds or for large projects where L2 switching is not enough
- Neutron uses Horizon to create and manage those virtual networks via a GUI

6.13.4 Neutron Components for ISO Layer 2 Operation

- If Neutron has only virtual ISO layer-2 network equipment to create and manage then the following Neutron component is sufficient for compute nodes:

1.) A software plugin, such as the Modular Layer 2 plugin ML2 from Neutron or some other plugin from ancillary software

- If compute nodes exists that are specicalized in networking then they must have additional components:

2.) A DHCP agent for dynamic IP address allocation

3.) A Neutron agent for administering metadata for the vLANs of these compute nodes

Example: the various vLANs and thus vLAN addresses a compute node may have are such metadata.

- Finally, all compute nodes need the common Linux network utilities which are present in every Linux host-OS distribution, such as Linux bridge e.g.

6.13.5 Neutron Components for ISO layer 3 Operation

- If Neutron has additionally ISO layer 3 virtual network equipment then each compute node needs a so-called OpenStack L3 agent

6.13.6 Further Neutron Components

- Furthermore, there are Neutron components that are mandatory for the central controller node only:
  - a SQL database (SQL DB) to permanently store the metadata for Neutron compute nodes and for some of Neutron's plug-ins
    - This can be a DB which is private to Neutron, or it can be the general DB of the cloud
  - a Neutron-server
    - it accepts REST API calls from other services and forwards them to the appropriate Neutron component in the controller node, or to a compute node, or to a compute node specialized in networking
    - additionally, it implements basic L2 and L3 features, such as the allocation of a private IP address to an OpenStack port
    - it is also the manager for the SQL DB if existent
- Optionally, numerous other agents, plug-ins and drivers can be engaged as well

- The agents are Neutron-internal components and devoted to manage OpenStack drivers, device drivers and plug-ins
- Ancillary device drivers and plug-ins are software taken from external, i.e. they are non-proprietary OpenStack codes

*Def.: A Neutron plug-in implements the so-called **virtual-to-physical transition** by means of a device driver. A plug-in can have multiple device drivers. Each device driver controls a specific piece of physical hardware, such as a switch or an Ethernet card, for example.*

Lemma: Only one plug-in can be used at any time in Neutron to avoid conflicts between similar plug-ins that want to command the same device driver and thus the same hardware at the same time: Either the ML2 plug-in or the plug-ins mentioned in the example below can be used alternatively.

Example: Neutron comes with agents, plug-ins and OpenStack drivers for Open vSwitch and Linux bridge. Additionally, there are plug-ins and physical device drivers for real Cisco switches, for real NEC OpenFlow products, for the VMware NSX product, and for products from many other commercial network vendors.

- All Neutron components for both, the controller and the compute nodes, are host OS processes that are communciating internally via RPCs over a message bus which is provided by oslo.messaging
- The L2 plugin and, if needed, also the L3 agent are executed by all compute nodes

6.13.7 Types of Data to be Exchanged in OpenStack

- In general, multiple data streams are exchanged inside of OpenStack

Example: Between the individual components of Neutron, data exchange is needed to make the components to cooperate.

- This data exchange is implemented in accordance with the exact data type that has to be transferred
- In total, there are 5 data types for exchange in OpenStack: 1.) public data, 2.) private data, 3.) management data, 4.) REST data and 5.) intra-service data

1.) Public-data
  - this data exchange is caused by the Internet, for example by requesting a service from the cloud from external

2.) Private-data
  - this is implemented by the data plane of a Neutron SDN that couples all VMs of the same project
    - the control plane consists of one or more Neutron vLANs
    - In OpenStack terminology, the data plane is also called Guest Network because it couples multiple guest OSes together

3.) Management-data
  - this is implemented by the control plane for the Neutron SDN data plane

    - the control plane configures the virtual switches and routers that implement the data plane in order to manage these devices
    - the control plane consists of one or more Neutron vLANs

4.) REST-data
  - REST transmissions are created by cloud users and projects that call OpenStack services via their REST APIs
  - this functionality is implemented by the so-called "API network"
    - the API network is typically another set of Neutron vLANs that is connected to the Internet
    - for security reasons, access to the API network is possible from outside via https only
  - if the API network of a project couples more than one VM then the public IP addresses of the VMs are dynamically allocated by source NAT

5.) Intra-service-data
  - this is implemented by one or more oslo message bus(es)
  - typically, one message bus connects the components of the same OpenStack service

6.13.7.1 Example of Data-Type Exchanges

- A subsequent block diagram illustrates the 5 data types that are exchanged in OpenStack, under the assumption of the following use case:

1.) There is an own node for the Horizon dashboard that is devoted to the sysadmin

2.) Glance and Cinder are running on the OpenStack controller node

3.) There is a virtual SDN and a specialized compute node as well, called the SDN service node
   • the latter is responsible for the control plane of the virtual SDN

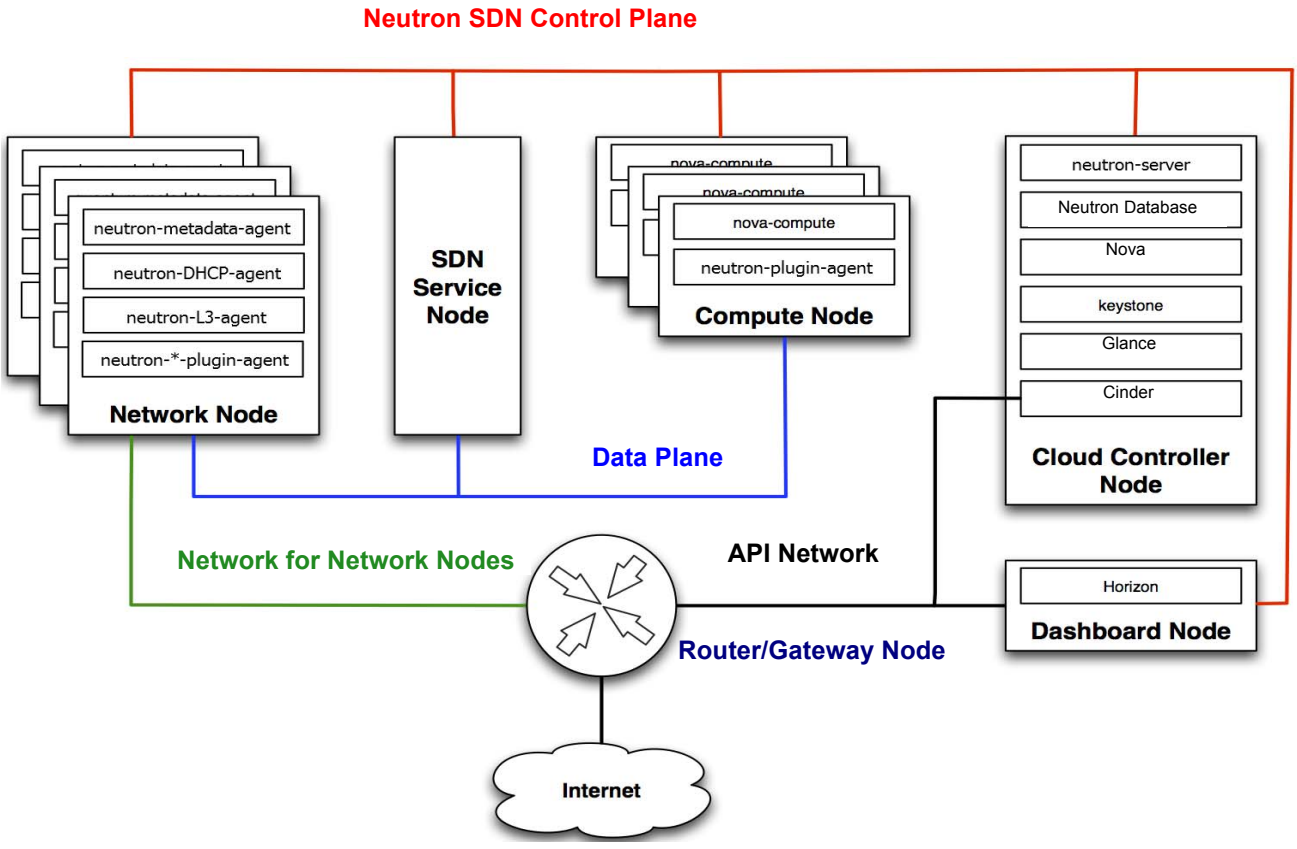
4.) There is one compute node which is specialized as router and gateway to the Internet

5.) Several other compute nodes exist and are specialized as Neutron network nodes in order to improve the bandwidth and connectivity to the Internet
   • each network node implements an own OpenStack cell and an OpenStack security group
   • each of the network nodes has its own vLAN that is connected to the central router/gateway node

- The API network is also connected to the router/gateway node
- The vLANs of the network nodes and of the API network are source NAT-translated to public IP addresses in the router/gateway node

Comment: The intra-service-data exchange via oslo message bus(es) is not shown in the block diagram below for reasons of simplicity.

- Under these assumptions, the following block diagram results:



6.13.7.2 Example of a Neutron Network Topology

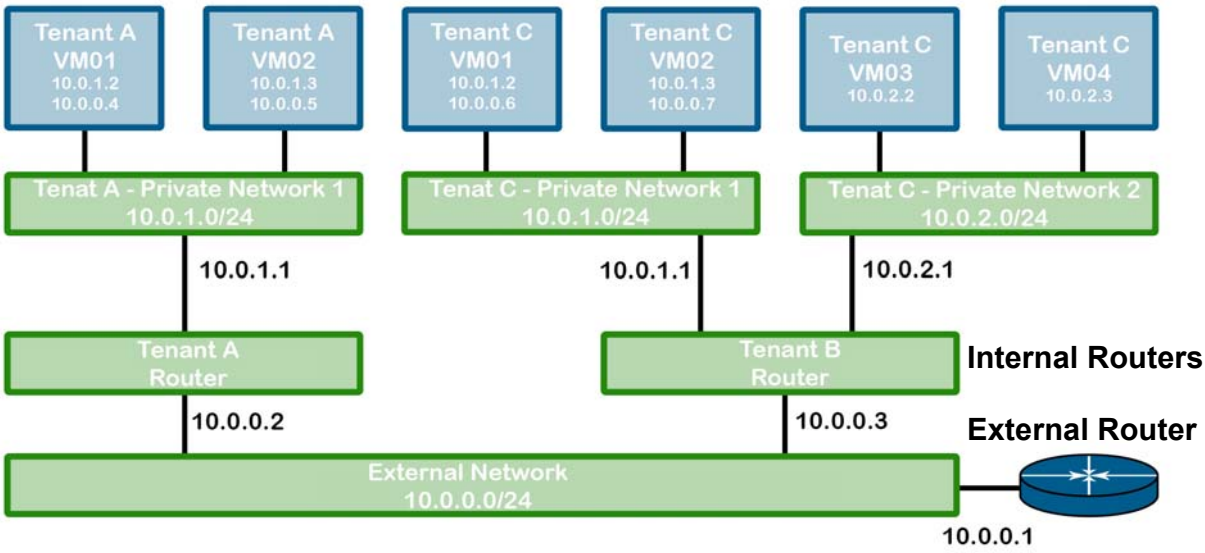
- In the next example, the following use case and vLAN topology is assumed:

1.) There is a project A (tenant A) which has 2 VMs, and a project C with 4 VMs
   - the 2 VMs of A are communicating via one Neutron vLAN with each other
   - no OpenStack port of A has a public IP address
   - the 4 VMs of project C (tenant C) are communicating pairwise via 2 Neutron vLANs with each other
   - again, no OpenStack port of C has a public IP address

2.) The vLAN of project A and one of the two project C-vLANs have the same private IP dresses
   - however, this does not disturb because projects A and C are fully isolated from each other

3.) All 3 private networks are coupled via 2 Neutron routers (tenant RA and tenant RB) to one external physical LAN that is in turn connected to an external Internet router

4.) The internal Neutron routers perform source NAT and thus translate the private-network IP-addresses into a public IP address



6.14 The Keystone Service

- Typically, Keystone is the first service that has to be installed in an OpenStack cloud because all other services depend on it
- Its main task is to authenicate users and to know and check their access rights
- Its second task is to authenticate and authorize OpenStack services when they are called
- Beside that, Keystone also maintains a list of all services which are installed in the respective cloud
  - this list is the so-called service catalog

Comment: The service catalog complements the list Murano has. Murano knows all user applications installed in the cloud.

  - Each service is registered in the service catalog with its name and with its so-called "endpoints" which are basically TCP port addresses for access from internal users, admin and public

Comment: More about "endpoints" see below.

- Last but not least, Keystone is responsible to secure all REST requests against misuse from hackers by issuing and validating user certificates that have to accompany each REST request of a cloud user
  - Securing REST requests, implies that Keystone can also revoke a user certificate if needed
  - The user certificates used by Keystone are so-called Fernet tokens

Comment: See the chapter below about Fernet tokens.

6.14.1 Keystone Terminology

- Keystone uses an architectural model for data privacy, as well as for authentification and authorization that comes with an own terminology
- The terminology uses phrases such as "Endpoint" or "Token" and is outlined as follows:
  - Endpoint: each service can be accessed by an URI that must contain a host-OS TCP/UDP port-number
    - this URI is called endpoint because it is sufficient to make an API REST request, provided that a valid token is presented
  - Token: Tokens are data structures sent by the REST caller to the callee which are digitally signed and which contain -among other fields- an encrypted message
    - so-called Fernet tokens are used by Keystone which are created by the Fernet algorithm
    - Fernet tokens are generated for both, user and service authentification and authorization and are typically <250 bytes long
    - since each token must be validated by Keystone, REST requests with Fernet tokes are not quick
    - however, because of their rel. short length, they are much faster than X.509 PKI certificates

Comment: PKI = Public Key Infrastructure.

  - Credential: a credential is either a Fernet token or a username/password pair

- a credential is associated either with a user or with user and project (=user/project pair)

- **Role**: a role is metadata associated with user/project pairs for RBAC

Comment: RBAC = Role-based access-control.

- **Extras**: extras are key-value metadata associated with a user/project pair
- **User**: each cloud user has an account credential and at least one project credential
  - each user is associated to one or more projects or "domains"

Comment: For domains see below.

- **Project**: each project is an organizational unit in OpenStack which is also known as tenant and which contains one or more users
- **Domain**: a domain is a 2nd organizational unit in OpenStack which contains users, "groups" and projects together
- **Group**: a group is a 3rd organizational unit in OpenStack and denotes a collection of users which are associated to one or more projects or domains
- **Rule**: a rule describes a set of requirements for so-called Keystone "triggering actions"
  - rules are stored in a JSON "policy file"
  - Horizon has a configuration file that contains the name and location of the Keystone policy file

6.14.2 Securing API Requests

- All API calls of OpenStack services are open to the world, even in private clouds, if the cloud's API network is declared to be a public subnet of the Internet

- To prevent from global misuse, several mechanism are therefore integrated in Keystone:

1.) Login into the cloud may be accomplished by a RSA key which is much safer than by username/password

2.) Every user and each service must get a Fernet token from Keystone for authentication

3.) The Fernet token implies also distinct access rights the user or the calling service has for each project individually and according to RBAC (= authorization)

4.) Every Fernet token expires quickly in typically 1 hour

- Several other protection measures exist as well which are not listed here
- All measures together make OpenStack clouds safe against Hacker attacks

6.14.3 Fernet Algorithm

- Beside its task for authentication and authorization a Fernet Token has, a Fernet algorithm provides for data encryption
- What has to be encrypted in OpenStack are metadata from a user or from a service required for their authentication and authorization (="A&A")
- This metadata is packed into a short string, and then encrypted and sent to Keystone
- Keystone decrypts the message with the A&A metadata und checks it for validity

- The encryption and decryption of the short A&A message is the purpose of the Fernet algorithm

6.14.3.1 Symmetric Data Encryption in Keystone

- It is the goal of any data encryption to secure data during Internet transit and for permanent disk storage in order to maintain data privacy
- Typically, this is accomplished by a public and a private encryption key
- However, for encryption due to the Fernet algorithm there is only a single private key for each user and for each user/project pair which encrypts and decrypts at the same time
- This is why Fernet encryption belongs to the class of symmetric encryptions
  - for Fernet encrytion, a so-called “Fernet key“ is needed
- A Fernet key contains the private encryption/decryption key and a digital signature for the whole Fernet data structure
  - The goal of the digital signature is to ensure that an encrypted message is from a proved issuer and not from a hacker
- Furthermore, Keystone keeps a repository for the permanent storage of its Fernet keys
- A Fernet key is the base64 url encoding of the following two fields: Signing-key (128 bits) + Encryption/Decryption-key (128 bits)

Comment: Base64url encoding is the same as base64 encoding, but the signs + and / are replaced by - und _ and the = padding at the end is replaced by %3d.

- Finally, a Fernet token is the base64url encoding of the following 5 fields:
  - a Version (8 bits)
  - a Timestamp (64 bits)
  - an “Initialization Vector“ (128 bits)

Comment: For Initialization Vector see below.

  - a “HMAC“ (256 bits)

Comment: For HMAC see below.

  - the encrypted message of variable length which must be a multiple of 128 bits
- A full specification of Fernet tokens can be found under: https://github.com/fernet/spec/blob/master/Spec.md

6.14.3.2 The Fernet Algorithm for a Fernet Token

- The Fernet algorithm is built on top of standard cryptographic algorithms as follows:

1.) Fernet uses for message encryption the “Advanced Encryption Standard“ (AES)

Comment: AES is the US encryption standard with an 128-bit encryption key. AES requires additionally an 128-bit initialization vector that is randomly selected.

  - AES encodes in the message individual blocks of 128 data bits each by means of an 128-

bit encryption key which is the first part of the Fernet key

- the initialization vector must be chosen uniquely for every token
- Fernet uses urandom() to create the initialization vector

Comment: /dev/urandom is a pseudo-random number-generator in Linux.

2.) Additionally, Fernet uses a "hash-based message-authentication code" (= HMAC) by means of the 128-bit signing key which is the 2nd part of the Fernet key

- the advantage of HMAC is that it decides about both, the authenticity of a data block and its integrity, i.e. about being not corrupted

Example: Beside its usage in the Fernet algorithm, HMAC is also used in the IPsec and TLS protocols.

- the hash function for HMAC is established with the help of "SHA256"

Comment: SHA256 is a standardized hash function that operates on 256-bit blocks which consist of 128 data bits and of 128 bits for a signing key.

3.) Each AES message must be sent together with the HMAC hash

4.) Keystone as the receiver, that has the Fernet key in its repository, has to re-compute the hash value out of the received data

5.) If the received hash value match with the hash value which is re-computed by the receiver then the data block is considered to be authentic

6.) Finally, the Fernet algorithm uses the "MessagePack" format for encoding the message

Comment: MessagePack is a compact format for binary data exchange and stands in contrast to the JSON format which is coded in ASCII and which is thus big of size, but human readable. In a MessagePack, small integers are encoded together into one single byte, and a short string needs only one extra byte as metadata in addition to the string itself.

6.14.3.3 Creating a Fernet Token

❑ The steps for creating a Fernet token are as follows:

1.) Bundle authentication and authorization metadata of a user or a service in a short string as message

2.) Convert the message into MessagePack format

3.) Encrypt the MessagePack block-wise by means of the encryption key and AES

4.) Sign the result block-wise by means of the signing key and HMAC

5.) Create the Fernet token out of the encrypted data blocks, their signing and the other fields in the Fernet token and send it

6.14.3.4 Using a Fernet Token

❑ The steps for using a Fernet token are described subsequently:

1.) The caller of an API requests, i.e. an user or a service has to add the Fernet token into the header of each REST request before transmission

2.) After reception, Keystone accesses the Fernet key in its key repository and decodes the message
   - in case that there are multiple Keystone nodes, the key repository must be shared between them

3.) Keystone checks the matching of the received and the re-computed signature

4.) Tf positive, the callee is allowed to execute the request and to answer with a REST response

6.14.3.5 Checking the Fernet Token

- The Fernet token is checked by Keystone as follows:

1.) It decodes the base64url encoding of the token

2.) It tests the token version whether it is compatible and not too old

3.) It checks whether the timestamp is not too far in the past because the token is valid only for 1 hour

4.) It computes the HMAC from the other token fields and from the signing-key

5.) It checks that the computed HMAC matches with the received HMAC field in the token

6.) It decrypts the metadata using AES 128 in CBC mode with the initialization vector field in the token and the encryption-key

Comment: CBC mode is one of the modes of AES 128.

7.) It unpads the decrypted MessagePack format, i.e. it removes padding bytes

8.) It decodes the MessagePack format, yielding the message in plain text

9.) It checks whether the authentification and authorization in the plain text message matches with the roles and rules it has stored about every user and all user/project pairs

10.)If positive, then it grants the called service to execute the request

- The listed steps above are a time-consuming process which is why Keystone is not a quick service

6.14.3.6 Example Interplay Between Caller, Callee and Horizon

- The use case in the example below is assumed to be as follows:

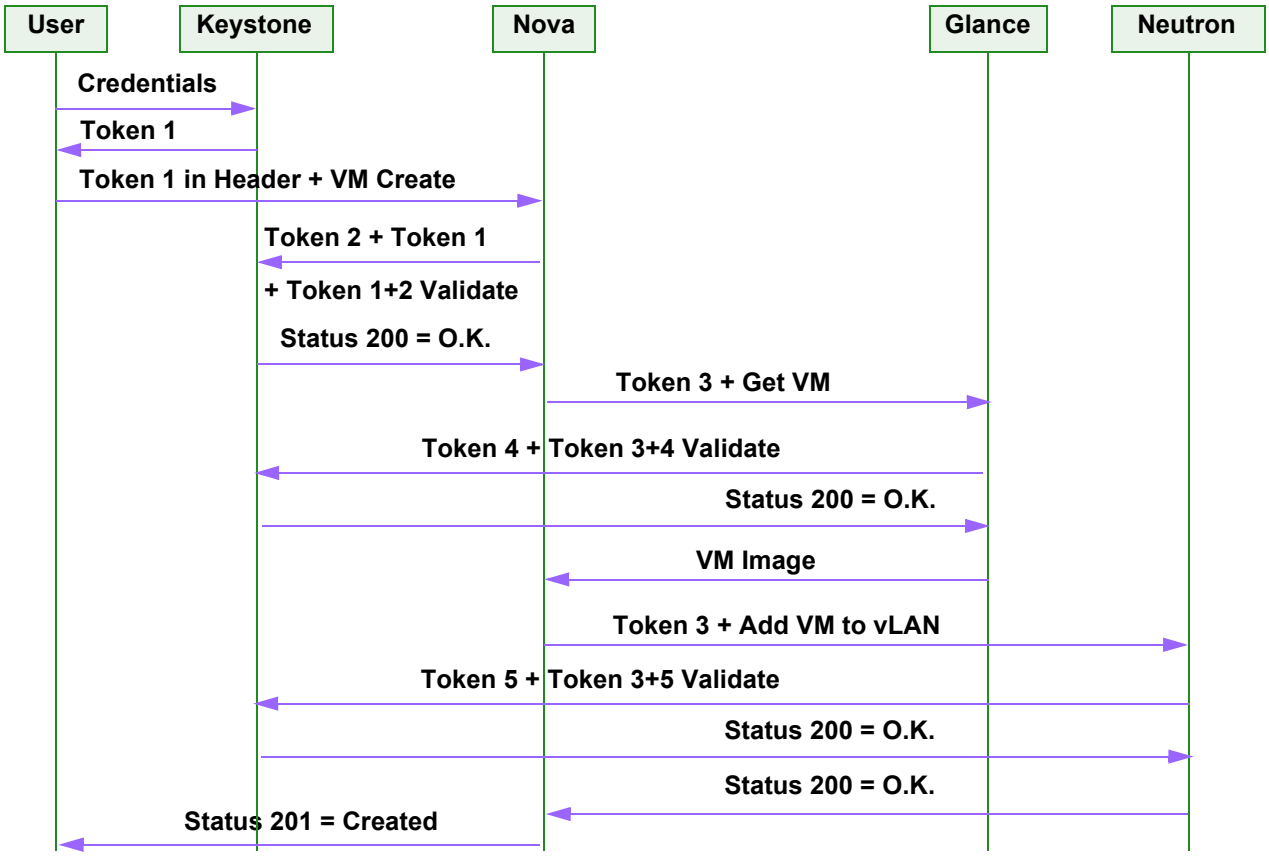
1.) A user logins with a credential at Keystone via a proper REST request

2.) He gets a Fernet token as response

3.) He wants to create and start a VM in his project by means of Nova via a REST request

4.) He gets a “created“ response with http status code 201 which is fine

- The time/space diagram of this use case is shown in one page below



6.14.3.7 Advantages of Fernet Tokens

- Fernet Tokens have the following advantages:

1.) They are at least 14 times shorter than X.509 PKI-based tokens which were previously used

⇒ REST requests with Fernet Tokens are much more efficient than X.509 certificates

2.) There is no need to store Fernet tokens permanently in the keystone database, instead to store the Fernet key is sufficient
   - as long as each Keystone node shares the same key repository, an equivalent Fernet token can be re-created again everywhere in the cloud and at anytime
   - this stands in strong contrast to UUID tokens, which were also used before, because an integral part of the UUID is time and date of the creation
   - a Fernet token also has a time and date, but it is not part of the message und thus not important

⇒ Only the private key of a user and of user/project pairs have to be stored permanently as a Fernet key
   - after expiration, a token can be deleted
   - this prevents the keystone database from overflow

6.14.3.8 Disadvantages of Fernet Tokens

- X.509 certificates have 2048 bits as signing key and are thus much safer

6.14.4 Token Revocation

- Keystone can also revoke an issued token
- Furthermore, its API allows to list revoked tokens and to list those OpenStack services that cache tokens
- Finally, it can ask other services to remove revoked tokens from their cache
- Tokens are not deleted if revoked, instead they are added to a Keystone-internal list of revoked tokens

6.14.5 Keystone REST Request Examples

- As a first example, we have the following use case:
  - An admin wants to login into Keystone without Horizon and to get a token back
  - For that purpose, the following shell script could be used (at least in principle):

```
curl -i \
  -H "Content-Type: application/json" \
  -d '
     {"auth": {
         "identity": {
           "methods": ["password"],
```

```
          "password": {
            "user": {
              "name": "admin",
              "domain": { "id": "default" },
              "password": "MySecretAdminPwd"
            }
           }
         }
       }
     }' \
  "http://myCloudController.de:13000/v3/auth/tokens" ; echo
```

- The script above assumes that Keystone is installed at a controller node named “myCloudController.de“
- Furthermore, it assumes for better illustration only, that the REST request and the token are transmitted in JSON format
- However, the request always has to be in base64ulr-encoded MessagePack-format
- As a 2nd example, we have the following use case:
  - An admin who has 999888777666 as own certificate wants to ask Keystone to validate a user certificate which is 887665443383838

Comment: The tokens used in the example are purely symbolic. Fernet tokens have up to 250 bytes.

- The curl command below could do that (at least in principle):

```
curl -H "X-Auth-Token:999888777666" \
   "http://localhost:35357/v2.0/tokens/887665443383838"
```


Comment: For admins, there is an own port number in Keystone which is 35357.

- For better illustration, the REST response from Keystone is again written as a JSON object, although it is in base64ulr-encoded MessagePack format
- The response contains the following information for a user of name "Antony ExampleUser":

```
{
    "access":{
        "token":{
            "expires":"2018-02-05T00:00:00",
            "id":"887665443383838",
            "tenant":{
                "id":"1",
                "name":"My great Foobar Project"
            }
        },
        "user":{
            "name":"Antony ExampleUser",
            "tenantName":"My great Foobar Project",
            "id":"1",
            "roles":[
                {
                    "serviceId":"1",
                    "id":"3",
                    "name":"Member"
                }
```

```
            ],
            "tenantId":"1"
        }
    }
}
```

Comment: The term “Foobar“ is widely used in informatics. It acts a placeholder for any possible string. It means “this and that which I do not know yet“. In German language: “Dingsbums“.

- The JSON response does not only validate the token but tells the admin also that Mr. Antony ExampleUser is member of "My great Foobar Project"

6.14.6 Internal Keystone Setup

- Keystone comprises one server, many modules and a few drivers
- The central server in the controller node provides authentication and authorization (A&A)
- Furthermore, there are about 24 modules

Example: One of the 24 modules is an API manager that extracts the Fernet token from the REST header and which sends it to the central server for A&A. An other module provides the service catalog of Keystone.

- Modules are connected to the Keystone server by the Python “Web Server Gateway Interface (WSGI)“ and not by oslo.messaging

Comment: WSGI is a standard interface between a web server and a web application.

- Drivers are used for accessing information in external repositories

Example: There is a driver for the Keystone SQL database and for an LDAP database

Comment: LDAP = Light-weight Directory Access Protocol. LDAP is standard software for identity systems that provides for authentification of users and their authorization.

6.15 The Cinder Service

- Cinder is the block storage service of Openstack and has five tasks:

1.) It allows to create a virtual hard drive that is block-oriented and delivers permanent storage
   - permanent storage is possible in Cinder because it is implemented by physical storage subsystems comprising storage servers and disks

2.) Cinder attaches the virtual hard drive to a VM
   - the Cinder virtual hard drive is an additional storage for the VM, beside its virtual hard drive KVM/QEMU has created
   - the Cinder virtual hard drive remains after the VM is terminated

⇒ In contrast to the VM‘s virt. harddrive, a Cinder virt. harddrive is persistent

3.) Cinder allows to create a snapshot of a virtual hard drive

4.) Additionally, it can make a backup of a virtual hard drive or of a whole group of them

5.) **Finally, it can encrypt a complete virtual hard drive**

Comment: All Cinder tasks will be explained in detail later.

6.15.1 Cinder Terminology

Def.: A distributed storage is a virtual hard drive or a virtual file system that is scattered over some or many physical hard drives. The distributed storage is implemented via a storage protocol.

Def.: A storage server is a physical computer with at least one physical hard drive that implements the storage protocol.

Def.: A storage protocol controls one or more physical hard drives, either directly or indirectly via a storage server.

Def.: A volume is a virtual hard drive Cinder has created.

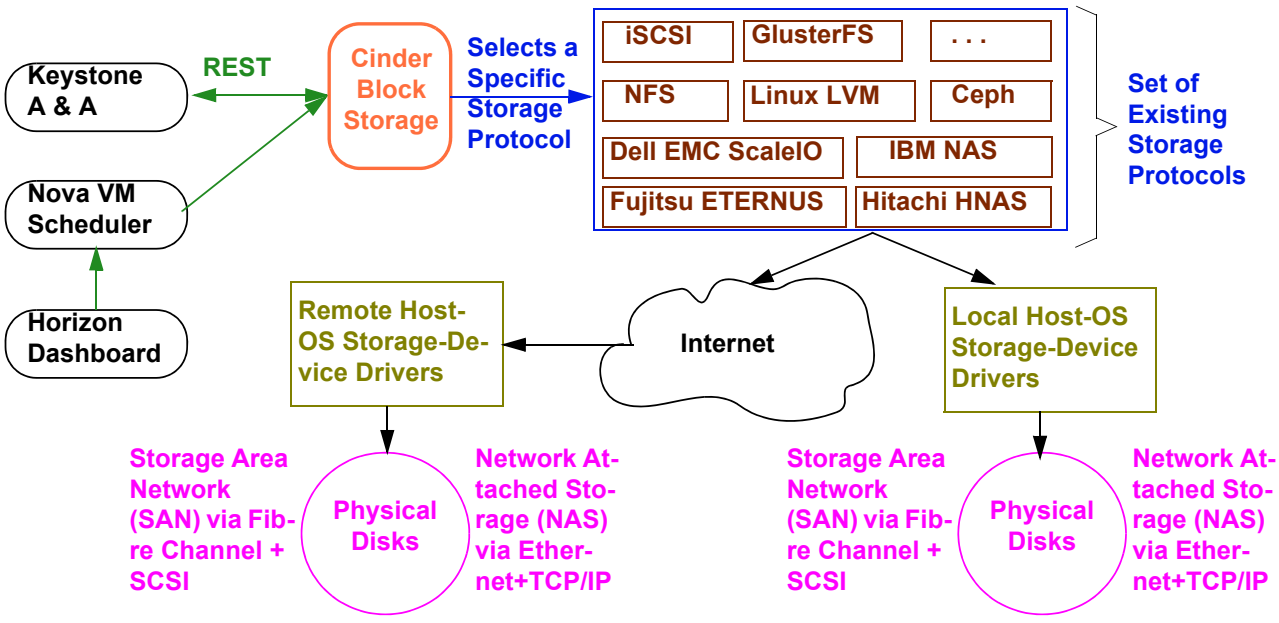
Def.: A volume type is a symbolic name a user or admin can invent. Each volume type is implemented by a Cinder volume driver. Many volumes of the same type can easily be created by specifying that volume type as stencil.

Def.: A snapshot is a copy of a volume at a specific point in time.

Def.: A Cinder driver (= storage backend driver) is a Python proxy for a storage back end. Several dozens of drivers exist in Cinder.

Def.: A storage back end is a specific physical server and protocol that a Cinder driver uses for persistent storage, such as iSCSI, NFS or GlusterFS for example.

6.15.2 Cinder's External Setup



- **Horizon acts upon a user input into its GUI or CLI**

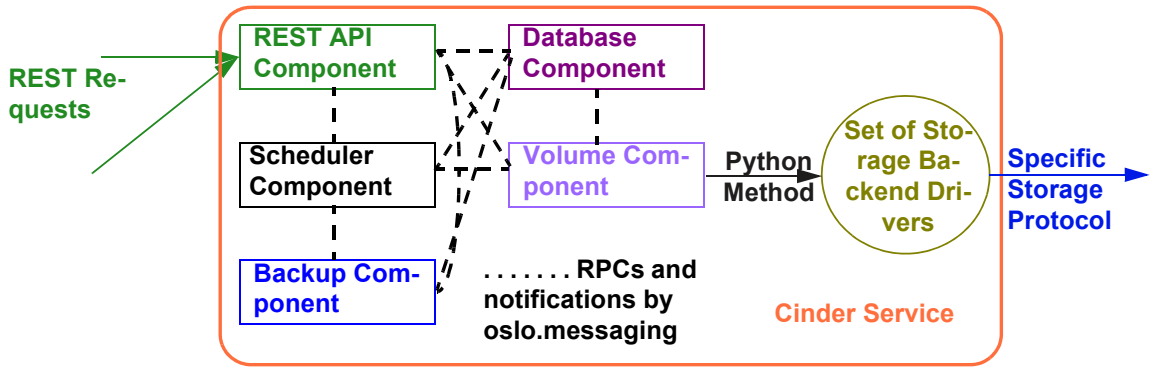
- Nova in turn acts upon a REST request from Horizon
- Finally, Cinder acts upon a REST request from Nova
- Cinder communicates in the cloud with Nova and Keystone
- It communicates externally via one or more storage protocols with physical hard drives or with storage servers containing physical hard drives
- The physical hard drives are controlled by local or by remote host-OS device-drivers which are residing in physical storage servers
- Typical for a data and computing center is that multiple physical disks are coupled to an array, either as a Storage Area Network (SAN) or as a Network Attached Storage (NAS)
- The default storage protocol for Cinder is “LVM“

Def.: LVM (= Logical Volume Manager) is part of the Linux kernel, and it allows to create a logical volume that spans across multiple physical disks.

- LVM implements a local volume group named “cinder-volumes”
- Interestingly, the size of a logical volume can be extended, even after volume creation
- This is possible because it does not really exist
- Consequently, also the size of a Cinder volume can be extended after creation
- Since the KVM/QEMU virtual volumes inside of VMs are block devices, Cinder can optionally also manage them, although being not persistent after VM shutdown
- Furthermore, Cinder supports RBAC for its volumes

- Finally, Cinder quota-controls check default-limits for projects automatically, such as:
  - the max. number of volumes that can be created
  - the max. number of snapshots that can be created
  - the total number of GBytes allowed
    - the last quota is shared between snapshots and volumes

6.15.3 Cinder‘s Internal Setup



- Each Cinder component is an own process in the host OS of the same physical storage server called “node“

- Each component can be run on the same node, or if needed, components can be run on more nodes in parallel to increase throughput
- Cinder consists of:
  - a SQL database which is shared by all Cinder components of a node
    - Cinder nodes in different cells may have different SQL databases
  - an API component which receives http requests, and converts and forwards them to other Cinder components
    - this component is implemented as a Web Service Gateway Interface (WSGI)
  - a scheduler component which decides which local or remote storage server has to host which virt. volume
    - scheduling is either round-robin or by selective filtering a proper storage server, according to its capacity, availability zone, volume type and capability, for example
  - a volume component which manages Cinder volumes, including encryption
  - a backup component which manages backups of Cinder volumes
  - a set of storage backend drivers which are python proxis to non-Python storage protocols which act as ancillary software to Cinder

6.15.4 Cinder API

- Cinder establishes an so-called abstract API for the creation and management of a virtual hard drive that is independant of the concrete storage protocol, i.e. of SAN or NAS, independant of device drivers and physical disks, and of any other hardware and software detail

- This is remarkable because there exist many storage protocols, device drivers and physical devices with highly varying properties
- This feature is accomplished by means of the Cinder volume component and its storage backend drivers
- Cinder‘s abstract API does not implement a distributed and reliable file system by itself, neither allows it to read from the volume or to write to it
- Only software ancillary to Cinder can do so

Example: If the chosen storage protocol is a distributed and reliable file system, such as “Ceph“, the created volume will be also distributed and reliable without notice to the user.

- Additionally, a typical Cinder volume has 1 TB, while a typical KVM/QEMU virtual hard drive has only about 100 GB

Comment: KVM/QEMU virtual hard drives are typically small, in order not to overcommit the local disk of the physial server that hosts the VM.

6.15.4.1 Volume Groups

- Furthermore, the Cinder API supports also “volume groups“ by means of Horizon‘s GUI
- This allows to handle sets of volumes collectively in the same way as a single volume, including snapshots and backups, which is very convenient for admins

6.15.4.2 Storage QoS

- Also for each volume or volume group, a Storage Quality-of-Service can be specified (=Storage QoS)
- Cinder allows to specify the following Storage QoS:
  - The maximum bytes per second as the volume‘s data throughput
  - The maximum bytes per second and per GByte of the volume’s capacity as an indirect measure for throughpu
    - this is an artifical metric that assumes that it is important to have more throughput if the volume is bigger
  - The maximum allowed Input/Output operations (= reads and writes) per second
  - The maximum allowed Input/Output operations per second and per GByte of Cinder volume capacity
    - this is also an artifical metric that assumes that the number of reads and writes should increase with a bigger volume

6.15.4.3 Volume Access

- A volume can be addressed uniquely even in a large cloud by its volume ID which is a UUID that is assigned to a volume by Cinder at its creation time
- Alternatively, a volume can also be accessed by a human-readable name that may be not unique

- Finally, a newly launched VM can boot from a Cinder volume because of the persistence of the latter

6.15.4.4 Volume Moves

- It is also possible to move a volume from one storage back end to another

Comment: Volume moves are needed in case of decommissioning a physical disk for example.

- However, this implies a real transfer of data because the physical storage is changed
- Cinder can also import non-Cinder storage and it can add metadata about it in its database so that it becomes a Cinder volume

⇒ **External data can be moved into Cinder**

6.15.4.5 Volume Scheduling

- For a given volume type, a collection of criteria can be specified by admin or user as a list of keyword/value pairs
- These keyword/value pairs are referenced by the Cinder scheduler when it decides which Cinder back end(s) have to host which volumes
- It implies also that Cinder drivers inform the scheduler about the physical capabilities of their disk drives by means of keyword/value pairs
- Both sets of keyword/value pairs can then be matched by the scheduler for its decision

6.15.4.6 Attachment of a Virtual Hard Drive to a VM

- Each Cinder virtual hard drive must be attached to exactly one VM or to one user

⇒ **It is not possible to share files in Cinder**

Comment: Sharing files is accomplished by the Manila service if required.

- However, a Cinder virtual hard drive can be detached from one VM and re-attached to another VM without data loss
- This allows for a very high-speed data-transfer between VMs, because no data is transferred physically
- By the REST API, it is also possible to transfer a volume from one user to another which implies also no physical data transfer

6.15.4.7 Accessing Records in Cinder Volumes

- If a VM wants to access a record in a freshly created Cinder volume then the first step is to attach the volume to the VM

Example: A volume would be attached as /dev/sdc1 in Linux if iSCSI is used as storage protocol. sd stands for SCSI device.

- The 2nd step is to format and to partition the volume, if not yet done
- The formatting and partitioning is accomplished in the guest OS of the VM

Example: Linux cfdisk can be used for that.

- The 3rd step is to create a file system on the volume if not yet done

Example: Linux mkfs can be used for that. Numerous file systems are thus possible, such as ext4 or ntfs.

- The fourth step is to mount the volume to a directory

Example: Mount /dev/disk/by-uuid/<volume ID> /myDirectory would work.

- Then one can access the volume in the language of his choice

Example: In the C/C++ languages, the use of open() in /myDirectory opens an existing file or creates a new one, and readf() and printf() can read and write records in that file, exactly as in a physical file.

6.15.5 Cinder Snapshots

Def.: A snapshot is a read-only copy of a Cinder volume at a given point-in-time

- For a snapshot, it is irrelevant whether the volume is attached to a VM or to a user
- This snapshot can then serve as the data source for a new Cinder volume, for example, by setting in Horizon the “source-valid“ keyword in the “Extra Settings“ of the new volume

6.15.6 Cinder Backups

- Cinder offers self-service backup and restore for projects

- Self-service means that no admin is needed

Def.: In contrast to a snapshot, a backup is not only a copy of a volume, but it contains also metadata to restore the contents of the copied volume to any other volume of enough space, even if it is of another type.

- Also in contrast to snapshots, backups are stored in a dedicated repository which is implememented not by Cinder but by Swift

6.15.7 Cinder Volume Encryption

- For a given volume type, one can specify an encryption for it by a set of keyword/value pairs

Example: Valid keyword/value pairs would be cipher='aes-xts-plain64', key_size='256'.

Comment: aes = American Encryption Standard.

- Beside aes-xts-plain64 also aes-cbc-essiv is supported; Beside key_size='256' also 512 is supported. Please note, setting the keyword/value pair to 512 results in a 256 bits encryption key.
- Non-admin users need the "creator role" in RBAC to create encrypted volumes and to store their encryption key in the so-called "Barbican" service

Comment: Barbican is the key-storing service of OpenStack.

- Volume encryption can be performed either by Nova via Horizon with the setting of the control-location parameter to ":= front-end", or by Cinder via Horizon with the setting of "control-location := back-end"
- If Nova encryts the volume then any data from or to the volume that is transmitted over a Neutron network is already encrypted, as well as the data on the volume
- If Cinder encryts then only the data on the volume itself is encrypted, but not its transmission

6.16 The Swift Service

- Swift is a complex OpenStack service, however with an easy API to the user
- Swift can store permanently and reliably big sets of data, together with metadata about it
- Typically, Swift manages TBs to PBs of virt. disk space

Comment: TB = Terabyte, PB = Petabyte.

- It can thus also be used for backup and archiving purposes in Big Data applications
- Swift's capacity scales arbitrarily by adding new Swift nodes which are physical storage servers
- In contrast to KVM/QEMU and Cinder blockstore, data is accessible from anywhere in the Internet, not only from a specific VM or user
- Via its API, Swift can be called directly from applications in projects (tenants)

- Also in contrast to Cinder, the Swift API is tiny and allows also for reading and writing data as well
- Only 16 REST requests are needed for the creation, reading, updating and deletion of objects and their metadata and for object management
- Because there exists no standard for objects, the Swift API does not compete with any other ancillary software or file IO method of a programming language
- Additionally, Swift ensures high reliability in data storage by means of two methods:
  - a user-defined number of replicas (= data copies) which are placed on disjoint storage nodes
    - The default value of the number of replicas is three which means one original has two copies
  - an "erasure code technique" which is used to restore lost parts of objects
- Normally, only one of the two methods is chosen by the user in a "storage policy file", but it is possible to combine both methods for extreme reliability
- Replica generation and erasure code technique are fully transparent to the user

Comment: Transparent means that after having set the method in the storage policy file, the user is not involved any more and does not see which policy was set.

- Erasure code is much more efficient with respect to required disk space than replicas

Comment: See subsequent chapters about replicas and erasure code.

6.16.1 Manifest Objects

- A normal Swift object cannot be larger than 5 Gbyte by default
- However, an object of arbitrary size can be composed out of any number of 5 Gbyte objects (or of any number of smaller objects)
- The result is called a "manifest object"
- Swift can handle an "infinite" amount of normal and manifest objects in the same cloud
- This number is only limited by free physical disk space, but not by Swift

6.16.2 Swift Terminology

Def.: A Swift account is similar to a project (tenant), but with a restricted scope. It shows only so-called Swift containers and metadata of them. It is the top level in the Swift data hierarchy.

Comment: Please, do not confuse a Swift account with a user account in Keystone, and also not do not confuse a Swift container with a docker or LXC container.

Def.: A Swift container organizes and stores objects in Swift. It is similar to a root-level directory in an operating system, but cannot be nested into sub-directories. It is the medium level in the Swift data hierarchy.

Def.: A Swift object is a large binary sequence of data (= blob), as well as metadata for it. Each Swift object resides in a Swift container. It is the low level in the Swift data hierarchy.

Comment: Blob = binary large object.

Def.: The expiration date of a Swift object is an optional flag to automatically delete objects in the future.

Def.: The versioning of Swift objects is an optional flag for an object storage container to version all objects within that container.

Def.: The hash value of an object is an unique ID to access it. The hash determines a path in the so-called "Swift ring". This path maps the Swift object to a physical disk somewhere in the cloud and allows thus to access the object's location quickly.

6.16.3 Swift Object Replication

- **If object replication is chosen by the user as redundancy method then replicas of the storage object are automatically created and distributed onto the cloud**
- **This is accomplished decentrally by the "local object replicator" which is in that node where the original object is stored**
- **The local object replicator pushes copies of local records and files to remote disks in the cloud**
- **If a replicator detects that a remote node has failed, then it chooses an alternative node**
- **In addition to object replicators, there are database replicators, which replicate databases for user accounts and containers**
- **The physical placement of replicas onto remote nodes is also handled by a hash ring**

Comment: See subsequent chapter about Swift hash ring.

6.16.4 Asynchonous Eventual Consistency for Replicas

- **In Swift, the method of "asynchronous eventual consistency" is used for replicas**
- **In general, eventual consistency is employed in distributed computing systems to achieve high availability or high throughput by creating and comparing replicas from original data**
- **Asynchonous eventual consistency means that the writing of the original data can already terminate while not all replicas are fully written**
- **In asynchronous eventual consistency, sooner or later all replicas will have the same value as the original, provided that in the meantime no update is made to the original**
- **A system that has finally achieved consistency between original and all copies is called "replica converged"**
- **Eventual consistency in general is only a weaker guarantee to the user because it is difficult to determine the exact point in time when the replica converged state is reached**
- **However, retrieval of a replica is not possible before replica convergence is obtained, in order to get only such REST response values that are consistent with the original data**

6.16.5 Erasure Codes

- Erasure codes are an error correction method for the case that parts of a message are lost during transmission, rather than parts are received wrongly
- By erasure code protection, a message of n symbols is expanded to k>n symbols, and k symbols are sent instead of n
- If $n \leq l < k$ symbols remain at the receiver after data loss then the message can still be fully restored, where the value of l depends on the specific erasure code that was engaged
- "Optimal erasure codes" have the property that any combination of l=n symbols out of k sent symbols are sufficient to fully restore the original message

Example: Reed–Solomon codes were the first optimal erasure codes.

- However, this comes along with a high computational complexity of $O(n^2)$ or at least O(nlogn) for calculating the k symbols and restoring n symbols out of it after data loss
- Near-optimal erasure codes require l>n symbols to recover the message but have only linear time complexity O(n)

6.16.5.1 Data Storage as Use Case for Erasure Codes

- If not data transmission but data storage is the use case for an erasure codes, then a storable object is decomposed into n fragments which are erasure-code expanded to k>n fragments and subsequently distributed to multiple storage nodes
- If some storage nodes that contain fragments fail then the object can still be recovered

- Because of the much lower disk space consumption compared to replicas, erasure code protection is preferred to replicas
- If erasure code protection is chosen as redundancy method then each storage object is automatically split into erasure-code protected-fragments and is distributed by means of the Swift hash ring
- For speed reasons, the algorithm of erasure-code protection is performed in an Swift-external C library and not in Python

6.16.6 Swift Data Hierarchy

- The Swift data hierarchy has three levels and consists of 1.) a Swift account that has all containers of a project, 2.) of Swift containers that store all objects, and 3.) of objects that store all data
- Every object reflects this three-level hierarchy in its URI:

Example: https://<mySwiftNode>:13808/v1/<account ID>/<container name>/<object path and name> would be a valid Swift URI. 13808 is the public Swift port that is exposed to the Internet.

- Accounts, containers and objects are explained in detail in the following subchapters

6.16.6.1 Level of Swift Accounts

- A Swift account defines a namespace for container names

- Any number of containers can be defined within one Swift account
- In an account, one can list, create and delete containers and create, update, show and delete their metadata

Example: Account metadata are the number and names of containers in that account, or the number of bytes stored.

- Access to an account is normally regulated by RBAC
- However, a user with a proper role in RBAC can allocate also an "Access Control List" (ACL) to the Swift account that bypasses the keystone RBAC mechanism
- Remarkably, such an ACL may grant access rights for a user to a Swift account which are not checked by Keystone
- Account ACLs are stored in the account as metadata

6.16.6.2 Level of Swift Containers

- A Swift container defines a namespace for object names
- In a Swift container, one can list objects, create, update, show and delete them and do the same for container metadata
- Similar to an account ACL, also container ACLs are possible
- Their scope is the container where the ACL metadata is stored
- The container ACL pertains to all objects of that container
- An optional object versioning also takes place at the container level

6.16.6.3 Level of Swift Objects

- All types of data can be stored in a Swift object, also those with block orientation
- The number of objects is unlimited
- Because of that, Swift could be a competitor to Cinder or Glance
- However, it is not recommended to have Swift as store for a relational database, for example, because SQL databases are speed-optimized for true block devices only
- It is also not recommended to store images in Swift without Glance, because the latter provides also for an image catalogue
- In case of an object download by means of a GET request, an auto-extraction of archive files is accomplished automatically by Swift
- ACLs for objects do not exist
- Furthermore, a time-limited GET to an object is possible
- Finally, any scheduled object-deletion takes place at this hierarchy level

6.16.7 Internal Swift Setup

- Swift consists of multiple components, and each of it is running as an own process in a host OS
- Because of the three-level hierarchy of Swift, its internal set-up is also structured in 3 layers, which are account layer, container layer and object layer
- Additionally, there exists a proxy server in Swift

6.16.7.1 Account Layer Components

- Account layer components comprise:
  - an account server: the account server is a software component in Swift that allows to list, create, update and delete object containers and their metadata
    - additionally, it stores object container information in account databases
  - account databases: account databases are multiple SQLite databases that contain Swift accounts and metadata for accounts
  - an account auditor: the account auditor checks for missing replicas and for incorrect or corrupted objects in a specified Swift account by means of queries in its account databases
  - an account replicator: the account replicator copies account databases for reliability reasons
  - an account reaper: the account reaper is a Swift process that scans for account databases which were marked by the account server for deletion and deletes them

6.16.7.2 Container Layer Components

- Container layer components consist of:
  - a container server: the container server is a Swift component that allows to list, create, update and delete containers and their metadata inside of accounts
  - container databases: container databases are multiple SQLite databases that store containers and container metadata
    - The container server accesses these databases

  - a container auditor: the container auditor checks for missing replicas of objects or for erasure code fragments of objects or incorrect objects in the container
    - It works via queries to the SQL databases of Swift
  - a container replicator: the container replicator copies container databases

6.16.7.3 Object Layer Components

- Object layer components have:
  - an object server: the object server is a Swift component that is responsible for storing, retrieving and deletion of objects on local disks which are part of a Swift container
  - an object auditor: the object auditor reads objects for an object server and verifies that the MD5 hash, the MD5 size, and all metadata are consistent of each object

Comment: MD5 is a method to create a hash function. It can serve, for example, to check whether a download was presumably correct.

  - an object replicator: the object replicator copies objects due to a replica policy
  - an object reconstructor: the object reconstructor implements erasure code protection if this replica policy was chosen

6.16.7.4 Proxy Server

- There is at least one proxy server in Swift that looks-up the location of account, container and objects and routes any incoming request to the proper local servers
- The proxy server manages also the public Swift API

- In case of erasure code as policy, the proxy server is additionally responsible for encoding, decoding and streaming object data during upload and download
- Additionally, multiple Swift proxy servers can exist in an availability zone to increase Swift's throughput and reliability
- Finally, Swift proxies in each availability zone should first write to and read from local disks in the own availability zone before streaming to and from other zones

6.16.8 Reliability Concepts of Swift

- Swift is a distributed architecture with no central point of control and thus also with no central point of failure
- Should one or more storage nodes fail, Swift restores their content either by replicas or by the erasure code technique, depending which policy was specified by the user
- Furthermore, retrieving of stored objects is based on so-called hash rings which are available on each storage node
- Additionally, Swift engages so-called manifest objects, the swift proxies, and two redundancy techniques to enhance reliability

6.16.8.1 Manifest Objects and Reliability

Def.: A manifest object consists of logically concatenated objects as segments

- Interestingly, a manifest object contains no object data, but only metainformation, i.e. pointers to the storage locations of the respective object

- There are two types of manifest objects: static and dynamic
- A static manifest object does not allow to add or remove segments after creation
- A dynamic manifest object can provide for that functionality
- Static manifest objects are additionally secured by a MD5 hash
- As a consequence, very high data integrity can be assured
- This is not the case for dynamic manifest objects because they can change over time which would require the hash value to change for Petabytes of data, for example
- Additionally, each static manifest object contains an ordered list of the segment names in JSON format it consists of
- A dynamic manifest object instead, contains only the name of the container where the segment objects are stored
- As a consequence, all of its segments must be stored in the same container for a dynamic manifest object
- Furthermore, Swift concatenates automatically all segments of a manifest object, in case of a GET request to it
- There is no limit in the number of segments
- The "Content-Length" field in the response header of the GET request shows the accumulated segment sizes
- Please note: the upload of manifest objects to Swift and the download from Swift can take a very long time if segments or objects are large

- Because of that and the eventual consistency of replicas, an uploaded segment might not appear in the container list until much later

6.16.8.2 Swift Proxies and Reliability

- Swift frequently communicates with its server components that are hosting objects
- Thus, even a small cloud generates megabytes per second of traffic caused by Swift
- However, Swift-internal communication is unencrypted, so Neutron networks with privacy are required
- The Swift proxy is stateless, which means that one can add more proxies in parallel to allow for more throughput and reliability
- Finally, between Swift proxies, HTTP load-balancing methods are optionally possible

6.16.9 Replica Policy and Reliability

- One should configure Swift with a sufficient number of availability zones to have for Swift's data reliability the quorum needed for replicas and/or erasure code fragments
- Only if the necessary quorum is configureed a successful REST response is possible

Example: With three replicas configured, the recommended number of availability zones would be five.

- Because of eventual consistency, sooner or later, all replicas will have the same content as the original data

6.16.10 Object Upload with Erasure Code Policy

- If erasure code policy is enabled, then data upload to Swift is made via PUT and has the following steps:

1.) The Swift proxy server streams-in one part of an object of user-configurable size, and buffers that part

2.) The proxy server calls an external library for erasure code in order to split and encode that object part with high speed into smaller erasure code fragments

3.) The proxy streams-out the erasure code fragments to storage nodes with the help of the hash rings

4.) The proxy repeats steps 1.) to 3.) until the object is fully uploaded

5.) The client is notified of upload completion when a quorum of l>1 nodes is met

6.16.11 Object Download with Erasure Code Policy

- Under erasure code policy, data download from Swift to the client is performed by GET and looks like this:

1.) The proxy server makes by means of the hash rings simultaneous requests to the k nodes where the object is stored

2.) As soon as the proxy has the l fragments it needs, it calls the object reconstructor which calls in turn the external library for erasure code to decode and re-compose the object part

3.) The proxy streams the object part back to the client

4.) The proxy repeats steps 1.) to 3.) until the object is fully downloaded

6.16.12 Load Balancing

- Load-balancing is made between projects and user applications that require Swift
- It is performed by means of Swift proxies that are distributed across multiple avaliability zones

Example: A HAProxy is one implementation for providing load balancing and high availability.

Comment: A HAProxy is open-source ancillary-software that provides for a high-availability load-balancer and proxy server for TCP/HTTP-based applications. A HAProxy can spread simultaneous requests across multiple servers.

6.16.13 Hash Rings

- Object distribution and retrieving is accomplished by means of multiple hash rings which work for both, erasure code and replicas

- A single hash ring is a data structure for the mapping of a subset of stored Swift objects onto their physical locations
- Therefore, the hash ring is also called “path hash map”
- Because of the 3-level hierarchy, there are at least three hash rings: 1.) for mapping accounts to physical disks, 2.) for mapping account containers to disks, and 3.) for mapping container objects to disks
- Typically, a hash-ring data-structure is so big that it must be stored in a file

⇒ **In every cloud, Swift has at least 3 hash ring files**

- If both data protection methods (erasure code and replicas) are engaged simultanerously, or if multiple storage policies are defined within one method, more than 3 hash rings are needed for mapping objects to disks
- Each hash ring is composed of 2 tables and one integer value as data structure elements
- The 1st table in the hash ring is the “**object-to-partition-to-device table**“

6.16.13.1 Object-to-partition-to-device Table (1)

- This table contains two mappings: 1.) from object to partitions and 2.) from partitions to physical devices

6.16.13.2 From Object To Partitions

- For Swift storage, each object is firstly decomposed into multiple "partitions" which are constituting parts of the object

6.16.13.3 From Partitions To Physical Devices

- Secondly, objects are mapped onto physical disks
- This is managed by the 1st table in the hash ring, together with the "ring partition number" which is the integer value in the hash ring

Comment: For ring partition number, see next chapter.

6.16.13.4 The socket Address Table

- The 2nd table in the hash ring contains mainly the socket address under which each local server component is listening to a REST request

Comment: A socket address consists of IP address and port number.

6.16.13.5 The Ring Partition Number

- Each object in a container is composed into a set of partitions that are addressed by numbers

⇒ The ring partition number is used to select a specific partition in the object

- The ring partition number has a length of L>0 bits

- This number is also used in the 1st table as column index
- All objects in the same container must consist of the same number of partitions
- The ring partition number is computed by a Swift Proxy as follows:

1.) The URI to the object, not the object itself, is hashed by MD5

2.) Then, the L last significant bits (LSBs) from the MD5 hash are taken as ring partition number
   - selecting only the L LSBs is equal to a 2nd hashing operation on top of MD5

- Using L LSBs allows to have $p=2^L$ partitions per object per container

6.16.13.6 The Disk Numbers

- Swift disks are numbered by Swift without gap from 0 to d-1 in order to address them
- If $d = p = 2^L$ holds, then the number of Swift disks matches exactly the number of object partitions
- If $d < p$ holds, then more than one partition is stored on every disk, and disks are modulo-wise addressed, in order to map all partitions to disks
  - The term "ring" in hash ring results from the modulo-wise addressing
  - In order to get an equal distribution of partitions to disks, d should be a power of 2
- If $d > p$ holds (i.e. the number of disks is bigger than the number of partitions), then not all of Swift disks are required to store the object

6.16.13.7 Data-to-Partition-to-Device Table (2)

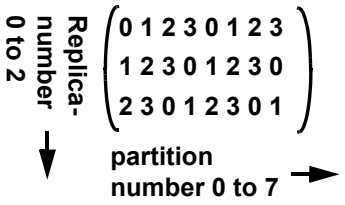
- The data-to-partition-to-device table (=1st table) is a 2D-table and has therefore a row index j and a column index i
- In case of an object with replica protection, the row index j is the replica number r of the object
  - typical is: $0 \leq r \leq 2$, where 0 is the original data, and 1, 2 are the two replicas
- In case of an object with erasure code protection, the row index j is the fragment number f of the erasure-coded object
- For both cases, the column index i is the ring partition number
- Each entry (=element) in the 1st table is a disk address
- The calculation of elements in the 1st table happens modulo-wise and is as follows:

1.) The first entry (i=0) in row j is the successor disk address of the first-entry (i=0) of row j-1 and obtained by an increment modulo d

2.) All following elements in row j with i>0 are successor disk addresses of i=0 and also obtained by an increment modulo d
  - Mathematically spoken, each row contains a cyclic left-shift permutation of the L bits of the disk number
  - This cyclic left-shift is equal to a "perfect shuffle permutation" due to number base d
  - The perfect shuffle permutation is known to deliver a very good mixing of elements

  - By successive cyclic shifts, the natural sequence of disk numbers 0,1, ..., d-1,d is transformed into a "ring" of numbers

Example: Given are an original data object and two replicas ($0 \leq i \leq 2$), together with d=4 disks (0-3). Then the disk numbers in the 1st table of a Swift ring are as follows:



- The cyclic left-shift permutation provides for an equal distribution of replicas or erasure code fragments across disks, provided that d is a power of 2
- The cyclic left-shift permutation enables also that disks are storing different subsets of partitions which is good for reliability in case of a disk failure
- Finally, it also is possible that Swift has other permutations than perfect shuffle for its 1st table

6.16.13.8 2nd Table In the Hash Ring

- The 2nd table in the hash ring is a list (=1D table) with the disk number as row index and no column index

- The elements (=entries) in the 2nd table are the socket address(es) of a Swift server component, together with some metadata needed for load balancing
- This allows the user to access via the Internet the proper Swift server component that is responsible for managing the user data to be retrieved or written

6.16.13.9 The Swift-Ring-Builder

- Each hash-ring data-structure is composed automatically by a tool called "Swift-ring-builder" component
- The Swift-ring-builder is also responsible for the allocation of disk numbers to physical disks which happens as follows:
  - The swift-ring-builder takes into account the regions the cloud has, the avaliability zones in each region, the partitions in each availability zone, the server components in each partition, and finally the specified storage policies in Swift
- As it is the case for the ring data structures, there is at least one ring builder, respectively for objects, containers and accounts which results in a minimum of three ring builders per Swift service

6.17 The Glance Service

- Glance lists, uploads, downloads, updates, deletes, suspends and resumes VM images and manages their metadata

Example: Metadata are the VM image ID and size, the image status and the project members that share an VM image.

- Additionally, Glance can import and export alien VM images from or to other an OpenStack cloud via the Internet
  - Alien images can be in another format, such as „qcow2", for example, and are converted by Glance to the own storage back-end format

Comment: qcow2 is a file format which is used by QEMU to store VM images.

- Finally, Glance can create a JSON schemes for an image and for metadata of so-called "Glance tasks"

Comment: See chapter below about Glance tasks.

- All operations are controlled by RBAC
- Glance images are used by Nova to start a VM and are using Cinder to create a bootable disk for the VM
- In large clouds, multiple Glance services can be run in parallel for better scalability and reliability

6.17.1 Glance Internal Setup

- Glance consists of the following host OS processes:
  - API component: The API component accepts REST requests and forwards them to the proper Glance component which is responsible for it

- **Database component: The database component stores and retrieves metadata for Glance**
- **Image data backend: The image data backend stores the VM images**
- **Metadata definition component: The metadata definition component allows the user to add metadata as keyword/value pairs in form of ASCII strings**
- **Replication component: The replication component copies VM images out of the image data backend which are needed for those instances of Glance which are running in parallel and ensures their consistency**
- **Addtional auditor, updater, and reaper components: The auditor, updater, and reaper components check VM images for consistency, and update and delete images if needed**
- **Task executor component: The task executor component handles "Glance tasks"**

6.17.2 Glance Tasks

- **Glance tasks are long running operations, such as an image export/import, or an image consistency check, an image upload/download or an image format conversion**
- **A Glance task can be run as a batch job in parallel to ongoing REST requests created by Nova or Cinder, in order decouple it from them**
- **A Glance task is a JSON data structure defined by a schema as described in the API documentation**

Example: The JSON task structure contains as keywords some CLI commands of Glance, such as "import_from" or "export to".

- **Each task has also metadata such as "created_at", "expires_at" and "pending", "processing", and "success" and "failure" as status**

6.18 The Trove Service

- **The Trove service manages various Database Management Software (DBMS) and the datastores created out of them**

Comment: For a definition of the term "datastore" see next chapter.

- **A Trove datastore cannot be not shared among VMs, but multiple users and projects inside of the same VM can access a single datastore**

Example: Oracle is the most famous DBMS. It creates a relational database for SQL queries.

6.18.1 Trove Terminology

Def.: database management software (DBMS) is also known as database engine. The DBMS is a commercial or OpenSource software for creating and managing a database. There are DBMSes for relational and non-relational databases.

Def.: A database is a set of files managed by a DBMS that contain user data.

Def.: A database VM is a compute VM that is specialized by a DBMS, and that contains a database for user data.

Def.: A Trove datastore is a database VM with "Trove guest-agent" and a "guest-agent configuration-file", together with a DBMS and user data.

Comment: For Trove guest-agent and Trove guest-agent configuration-file, see later chapters.

6.18.2 Supported DBMSes

- **In a Trove datastore, at least the following DBMSes are more or less supported by Trove and can thus be installed:**
  - **Cassandra, Couchbase, Couchdb, Db2, Mariadb, Mongodb, Mysql, Percona, Postgresql, Pxc, Redis and Vertica**

6.18.3 Classification of Trove Datastores

- **There are 3 classes of Trove datastores, depending on the used DBMS:**
  - **experimental: this class has only basic functionality which was tested for Trove**
  - **technical preview: this class has nearly the full functionality, but not all is tested**
  - **stable: this is the class of database VMs where all functions the DBMS are available and thoroughly tested**
    - **only datastores from the stable class are production-ready**

Example: Mysql as DBMS inside of a database VM is classified by Trove as stable datastore.

Comment: Mysql is an open source DBMS for relational databases.

Example: Cassandra and MongoDB are classified as technical preview datastores.

Comment: Cassandra is an open source DBMS for non-relational databases which are distributed across multiple servers.

Comment: MongoDB is an other open source DBMS for non-relational databases which are distributed across multiple servers. In contrast to Cassandra, it uses JSON-like documents with schemas, and it is especially suited for text documents.

6.18.4 Storage Locations

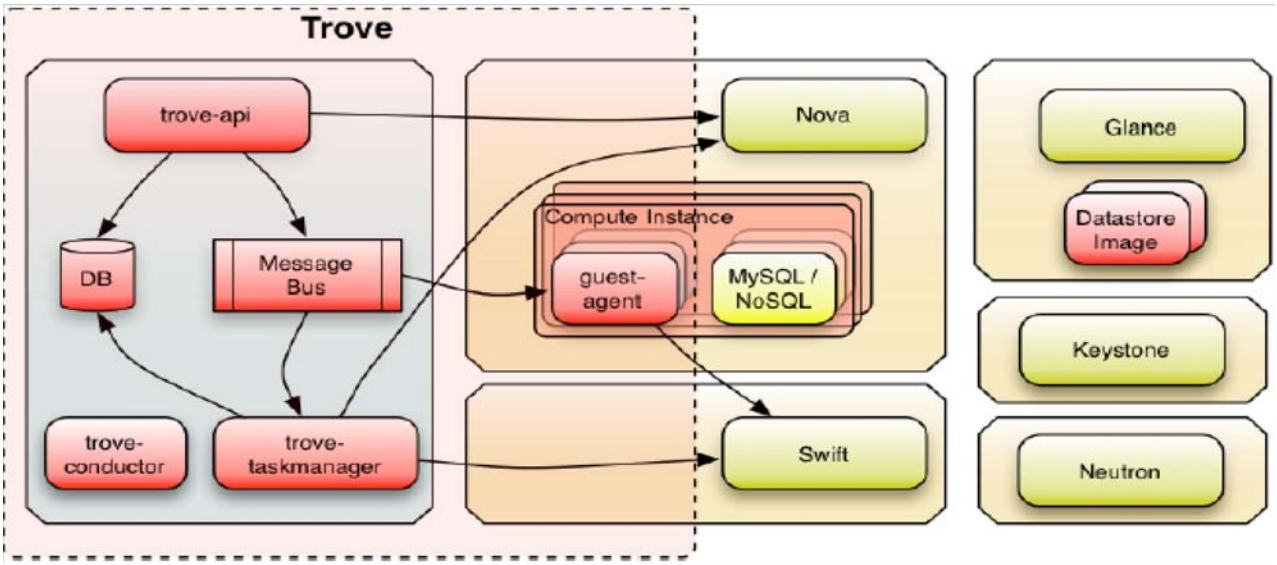
- **VM images for Trove datastores are also stored in Glance, as all image types**
- **However, backups of Trove-database VM-images are stored in Swift**
- **Furthermore, database data are stored in the respective file system of the datastore**

6.18.5 Trove Configuration Groups

- **One can configure a whole set of databases of the same DBMS type in the same way by using a Trove configuration group**
- **A Trove configuration group allows to set configuration parameters collectively for the whole group**

6.18.6 Trove Internal Setup

- **Trove has the subsequent components:**



1.) **API component:** The API component runs either on a specialized VM (=Trove node) or on the central cloud controller-node

- it is implemented as a so-called "WSGI Router" and forwards incoming REST requests to the proper Trove component which is responsible for the request

Comment: The Web Server Gateway Interface (WSGI) is a calling convention for web servers. A WSGI Router is a WSGI-based application containing a list of URI routes. The WSGI router takes the incoming

URI and decides which component will handle the request. Therefore, it parses the URI and extracts out of it the input parameters for the target component to which it forwards the request.

2.) **Taskmanager:** The taskmanager runs also either on a specialized Trove node or on the controller node

- it creates and launches a compute VM by means of Nova for a later installation of a datastore on it
- it launches an already created datastore VM image by means of Nova and Glance
- it performs operations on the launched VM after it is specialized as a datastore VM
- it manages the lifecycle of the datastore VM from creation to termination

3.) **Guest agent:** The guest agent must be run on every datastore VM

- it contains a local datastore manager as sub component
  - this is the OpenStack driver for the DBMS
  - for each supported DBMS, an own datastore manager exists and must be installed
- operations on the datastore's data are performed by SQL, or by by non-SQL queries or by the CLI of the DBMS, but not by the guest-agent itself

4.) **Conductor:** The conductor runs as well either on a specialized Trove node or on the controller node

- it receives messages via oslo.messaging from all datastore VMs to update status information of the VMs in the Trove infrastructure database

Example: NEW, BUILDING or ACTIVE are typical status information of a datastore VM.

- it provides a “heartbeat” that signals periodically to the Trove infrastructure database the ACTIVE status of the datastore VM
- it provides an “update_backup” that signals periodically to the Trove infrastructure database the status of the backup of the datastore data, including its metadata, but excluding the VM itself

Example: Size, type and checksum of the datastore data are typical metadata of a backup.

5.) **Infrastructure database**: The infrastructure database runs again either on a specialized Trove node or on the controller node
- it contains a catalog of all database VMs the Trove node/controller has
- it keeps encyrption keys for inter-component communciation
- it holds metadata about status

6.18.7 Minimum-Requirements for a Database-VM

- The specialization of a VM as Trove node requires that the VM‘s flavor matches the minimum requirements of the respective DBMS with respect to main memory, disk space and number of cores
- Each database VM must contain as minimum software the following:
  - a Trove guest-agent, a Trove guest-agent configuration-file, a DBMS, and sooner or later also database data
  - since each database VM is a specialized compute VM, it must contain also a host OS with

KVM/QEMU, Neutron-Linux Bridge and Nova-Compute

6.18.8 Trove Guest-Agent Configuration

- In the first place, the Trove guest-agent configuration contains parameters for the task manager and the DBMS
- Secondly, the guest-agent configuration contains also:
  - datastore-specific options
  - options for backups in Swift
  - options for oslo.messaging
  - set-up for the log files
- Finally, by means of the guest-agent configuration a "root" user is added to the DBMS
  - in Trove, the access rights of the root user and of other users are called “grants“
  - Interestingly, grants do not comply with the RBAC system of OpenStack because the DBMS is third-party software

6.18.9 Save Inter-Component Communication

- As for any other OpenStack-internal communication, oslo.messaging is used between Trove components
- This is accomplished as ususal via the oslo message bus

- However, Trove allows to encrypt its inter-component communciation by means of "oslo_messaging.rpc security"
- This function provides for the following:
  - It creates an individual encryption key for the communication between the API component and the taskmanager
  - Additionally, there are individual encryption keys in each Trove node for the communication with other nodes

6.18.10 Creation of Datastores

- By means of Nova, Trove can create a VM with a specific flavor and it can launch it as a database VM (=Trove node) in order to establish a datastore
- As soon as a DBMS is installed in the Trove node, it can also provide for multiple databases of the same type in that VM

⇒ Datastores with multiple databases are possible

- Both, users and admins have the grants to install a DBMS on the VM
- They can also create in the cloud as many datastores as wanted
- It is also possible that different DBMSes are installed in the same VM
- Each database VM with a specific DBMS (or more of them) is stored as image template in Glance for later launches, when the same datastore type is needed a second time

- Each image template in Glance has a name and an ID to which Trove and the user can refer for later launching

6.18.11 The Trove API

- The Trove API stays always the same because the API is only responsible for the creation and management of databases but not of their usage
- The API does not change regardless whether the DB is SQL-based or not, or whether the datastore and is localized on one node or distributed, or which DBMS is used
- The API allows to create and manage the following items:
  - database VMs
  - replicas and backups of database VMs
  - databases and whole datastores
  - configuration groups
- In the following, these functions are explained

6.18.11.1 Create and Manage Database VMs

- Trove allows for a database VM to create it, to list all created database VMs, to attach the database VM to a configuration group, to update its name and version, to list its configuration and to delete it

- Additionally, Trove can list the flavor of the database VM, resize the VM, restart it and manage its log files

6.18.11.2 Create and Manage Replicas of Database VMs

- It is frequently avoided to make changes in a production datastore because this implies an unwanted down time of the store
- Instead, a replica (copy) of the datastore is made and changes are accomplished there
- If everything works after the change, then the replica is declared to become the new datastore and replaces the old one
- This reduces the down time of the datastore to zero

6.18.11.3 Create and Manage Backups of Database VMs

- The API allows also to create, list and delete backups of database VMs

6.18.11.4 Create and Manage Databases

- Trove allows to create a database by means of a DBMS, to list all database in detail, to enable and to disable themand publish them
- Furthermore, Trove can add a root user and normal users to the database by configuring it

6.18.11.5 Manage Datastores

- It is possible in Trove to list all datastores and to show their configuration

6.18.11.6 Create and Manage Configuration Groups

- Trove allows to create, list, delete and update configuration groups

6.19 The Ceilometer Service

- Ceilometer provides as service the collection of OpenStack-internal **usage-data**

Example: Usage-data are VM uptime, elapsed CPU time, consumed disk space, performed number of IO operations per VM and per day etc.

- Additionally, Ceilometer provides for the collection and storage of OpenStack-internal **events**

Example: An event is that a VM was launched or decommissioned.

- Finally, Ceilometer provides for the collection and storage of OpenStack-internal **alarms**

Example: An alarm is when a VM uses > 90% of its main memory.

- A component of Ceilometer can be run in every node (compute, controller, ...)

- **However, Ceilometer is not a typical OpenStack service because it does not have an API**
- **Instead, it communicates with fully-fletched OpenStack services via notifications that are sent by means of oslo.messaging**

6.19.1 Ceilometer Terminology

Def.: Metering is the process of collecting data about resource consumptions with respect to type, points-in-time, users, etc. The result of metering is a set of "tickets" a.k.a. Ceilometer samples.

Comment: A.k.a. = also known as.

Def.: A meter is a counter variable that is defined by a user or admin. It tracks a resource in order to collect consumption data by counting accesses to that resource. One resource can have multiple meters.

Example: VMs, Swift or Cinder volumes, Glance images, Neutron networks and switches are resources being metered.

Def.: An event is a state-change of a resource after something interesting has occurred. Each event results in a data structure sent via oslo-messaging that comprises an event_type string, a message_id, a timestamp and a sequence of key-value pairs that describe the event's traits.

Example: An event may happen if the flavor of a VM or its public IP address has changed.

Def.: Alarms allow to automatically monitor OpenStack resources and to react in case of malfuntioning. A user or admin can define a set of Ceilometer rules that create an alarm action if they are broken. An alarm rule is a set of meters or events whose values are combined by AND and OR. Alarm rules are evaluated periodically every minute. Alarm actions are HTTP POST requests being sent to an endpoint, together with a description of the alarm in JSON key/value pairs.

Example: The following Ceilometer CLI command sends a POST request to the URI http://example.org/notify if at least one out of two alarms is risen.

```
ceilometer alarm-combination-create --name meta \
  --alarm_ids ALARM_ID1 \
  --alarm_ids ALARM_ID2 \
  --operator or \
  --alarm-action 'http://example.org/notify'
```

6.19.2 Types of Meters

- **Three types of meters exist:**

1.) Cumulative: In a cumulative meter, the value can only increase over time

Example: The number of performed disk I/O operations in a VM during a day can be measured by a cumulative meter.

2.) **Gauge:** In a gauge meter, the value is either constant or fluctuating

Example: The number of public IP addresses a VM has may be a constant meter value.

Example: The number of Swift objects, or the number of image launches per day can be measured by a fluctuating meter.

3.) **Delta:** In a delta meter, only the incremental changes of the meter value per time unit are recorded

Example: The effective bandwidth of a NIC is growing or decreasing over time, and the bandwidth changes can be captured by a delta meter.

6.19.3 Ceilometer Components

- Ceilometer has the following components:

1.) **Polling agent**: the polling agent has the following tasks:

- polling agents are the preferred method in Ceilometer for data collection
- it polls any OpenStack API or any OpenStack tool at a regular time interval
- it can be configured to run either on any (specialized) compute node or on the central controller node(s)
  - if it is run on a compute node then the polling agent serves as decentral metering process
  - if it is run on the controller node then the polling agent serves as metering process for resources that are not bound to a (specialized) compute VM

Example: Swift objects or Heat scripts can exist in the controller node independantly of compute VMs.

Comment: For Heat scripts see the next main chapter.

2.) **Notification agent**:

- it runs on the central controller node(s)
- it listens to notifications sent via oslo.messaging from ceilometer-polling agents
- it sorts event by their type
- it composes alarms, events and metering values to data structures which are called Ceilometer samples
- it forwards these data structures to “Gnocchi“ or to other OpenStack services such as “aodh“ via oslo.messaging for post-processing

Comment: Gnocchi is an OpenStack-service with API for efficient storage and statistical analysis of Ceilometer data.

Comment: Aodh is the alarm service of OpenStack. It triggers actions based on defined rules for meters, events or alarms which are collected by Ceilometer or Gnocchi.

3.) **Push agent**:

- it instruments an OpenStack service on a compute VM or on a specialized VM by adding push agent code to it
- the push agent is the only posibility for services to get measured which do not expose the required data via an API or via oslo.messaging to ceilometer
- Push agents are not the preferred data collection method in Ceilometer

- Multiple agents can run in parallel on compute nodes and controller node(s) to scale Ceilometer from small to large clouds

6.19.4 Ceilometer Operations

- Ceilometer operations are to collect usage data, events and alarms and to forward them to so-called endpoints

6.19.4.1 Collecting Usage Data

- Ceilometer is part of the so-called “billing“ work flow
- It implements “metering“ which is the first step in the billing work flow

Comment: The 2nd step for billing is called “rating“ which means setting a price for a consumed resource. The 3rd step is billing which is writing an invoice to a customer for the resources he has consumed.

- The collected consumption data may be sent to Gnocchi

6.19.4.2 Collecting Events

- Collected events are sent to the aodh alarm service for postprocessing or to mySQL or PostgresSQL for logging

Comment: PostgresSQL is a DBMS for object-relational databases and supports SQL, but with many extensions.

6.19.4.3 Collecting Alarms

- Collected alarms are sent to the aodh service or to “ElasticSearch“ for postprocessing, or to mySQL, PostgresSQL, MongoDB for logging, or to an URI via a POST request

Comment: ElasticSearch is an OpenStack service that can search all kinds of documents. Its features are available through a JSON and a Java API.

6.19.5 Supported Hypervisors

- If a polling agent wants to collect usage data, events or alarms from a (specialized) compute node then it must cooperate with the hypervisor of the VM that implements the node
- Supported hypervisors are Hyper-V, XEN and ESXi, as well as all those others which can be accessed by libvirt, such as KVM/QEMU for example

6.19.6 Other Supported Tools

- The notification agent is also able to receive notifications from two external metering services for networks which are OpenDaylight and OpenContrail

Comment: OpenDaylight is a platform that allows users to write apps that work with a wide variety of hardware and protocols in the context of Software Defined Networking (SDN).

Comment: OpenContrail is a large collection of external software for OpenStack.

- OpenDaylight and OpenContrail are ancillary software to OpenStack

6.20 The Heat Service

- Heat is the OpenStack service for orchestrating other OpenStack services

6.20.1 Orchestrating

Def.: Orchestrating is the program-controlled combining of web services by a central controller. An orchestration controller initiates actions at other web services and can check for their fullfilment and correctness. Typically, orchestrating is a centralized way for automating the deployment and management of software, including their configuration.

- By orchestrating, multiple existing work flows for cloud administration can be flexibly coupled
- This is accomplished by program-controlled REST requests to OpenStack services and by executing "Chef", "Puppet" or "Ansible" scripts which are embedded in Heat

Comment: Chef, Puppet or Ansible are widespread scripting languages that are specialized in software installation, configuration and management.

- The Heat service is the orchestration controller for all other OpenStack services
- It allows to automate:

1.) **Typical admin workflows** such as:
   - conditional launching and decommissioning of VMs

   - conditional deploying and removing of application packages
     - the deploying and removing of software packages typically imply that a specific sequence is obeyed how the components of the package have to be installed or deleted
     - this can be ensured by Heat
     - deploying or removing of packages implies also that proper hardware resources are available
     - this can be checked by Heat before the installation begins

2.) **The installation of new OpenStack services in an existing cloud**
   - this allows for an automatic adaption of the software infrastructure of a cloud to new requirements users or projects have

3.) **Cloud monitoring and conditional intervening in case of events and alarms**

4.) **Restoring of cloud functionalities after errors**

5.) **The installation of new OpenStack services on new hardware**
   - this is accomplished in the proper sequence so that services are running and can cooperate
   - this feature allows also to create a new OpenStack cloud in a new computer cluster by means of an existing OpenStack cloud that runs a "Heat stack"

Comment: A Heat stack is a script in Heat.

- Furthermore, admins can customise Heat stacks by installing Python plugins in their code

- In essence, cloud admins are significantly unburdened by Heat orchestration
- Additionally, also users can profit as well from orchestration by writing their own Heat stacks

6.20.2 The native Heat API

- The native Heat API deals with resources, resource types, Heat stacks, stack output, stack snapshots, stack actions, stack events, HOTs and project software-configurations
- These items and terms are explained subsequently

6.20.3 AWS CloudFormation Templates

- Beside its native API, Heat has a 2nd API for stacks written in Amazon‘s AWS “CloudFormation“ batch-processing language
- This language is called “AWS CloudFormation template language“

Def.: An AWS CloudFormation template is a parametrized text file that contains everything needed to carry out a specific orchestration process. As soon as the parameters in it are set, the text file is ready for execution.

- CloudFormation templates are useful for orchestration because they can model and create the virtual infrastructure of a whole AWS cloud
- In OpenStack, CloudFormation templates are called “cfn templates“

6.20.4 Heat Orchestration Templates (HOTs)

- Beside cfn templates, Heat has also its own template format called “Heat Orchestration Template (HOT)“
- HOTs are not compatible with cfn templates
- HOTs are expressed in JSON or yaml format

Comment: Yaml is similar to JSON. However, in contrast to JSON objects yaml objects are not directly executable by Python. Furthermore, in yaml the JSON brackets are replaced by line indentation for better readability. Finally, yaml is preferred to JSON because of its simplicity, readability and clearity. It is closer to human speaking than JSON and much closer to that than XML.

- Compared to cfn templates, they are more restricted in functionality, but there is the goal to surpass cfn templates
- A HOT describes the software infrastructure with respect to “OpenStack resources“ that a user application needs in a cloud

Example: Typical OpenStack resources are compute VMs, floating IPs, Neutron ports, Cinder volumes, security groups, projects, user accounts and OpenStack services.

- A list of OpenStack resources is under: https://docs.openstack.org/heat/latest/template_guide/openstack.html
- From a practical point-of-view, a HOT is a named file in JSON or yaml format that contains all information to make a user application work

6.20.4.1 The HOT language

- **Each HOT is written in the HOT language which is the counterpart to the AWS-CloudFormation template-language**

Def.: The HOT language is a collection of predefined keyword/value pairs in JSON or yaml format, together with control structures such as IF, REPEAT, NOT, AND, OR, in which also shell commands can be included, as well as Chef, Puppet or Ansible scripts.

- **Shell commands, and Chef, Puppet or Ansible scripts must be stored in separate files and are included into the template by means of the "get_file" function of the HOT language**

Example: For the inclusion of a shell script in a HOT:

```
get_file: http://ExampleOpenStackComputeNode.de/myShellScript.sh
```

Def.: Each file that is included into a HOT via the "get_file" function is called plugin.

- **The specification of the HOT language is under: https://docs.openstack.org/heat/latest/template_guide/hot_spec.html#hot-spec**

6.20.4.2 HOT Features

- **First, HOT describes the relations between OpenStack resources**

Example: A HOT can command that a given volume is attached to a given VM, or that a given database user is added to a specific user group, or that a given keystone user is added to a given Neutron security group.

- **Additionally, a HOT ensures that the installation of a software package happens in that sequence that is needed for the packet components to interact properly**

Example: For many packages, a database component must be installed first before other components because the latter need a database.

- **Finally, a HOT allows not only for resource instantiations in the proper sequence but also for "higher functions"**

Example: The creation and launching of a non-stop VM that has high availability, the auto-scaling of the number cloud VMs, or so-called nested stacks are higher HOT functions.

Def.: Nested stack means that inside of a parent stack are one or more daughter stacks located. This provides for the analogon of subroutine or macro calls, but on the template level. Nested stacks can be referenced by UUIDs or by names. The concept of nesting stacks is recursive, i.e. a daughter stack can be a parent stack as well.

6.20.4.3 Differences between a Heat Stack and HOT

Def.: A Heat stack is a single resource instantiation or a sequence of it inside of a HOT. A HOT can comprise many Heat stacks. HOT is the superset of Heat stacks.

- Since every HOT is written in the HOT language, each resource instantiation may be also accomplished in the HOT language itself
- In a Heat stack, an instance of a specific resource type can be created analogously to a variable with name in a programming language which is derived from a variable type

Example: A security group in Neutron is a Heat resource type. The creation of a specific security group that is referenced to by a name and used in an existing project is an instantiation of the security-group resource-type.

- Conversely, a HOT is a named file in JSON or yaml format in which individual Heat resources are instantiated
- Each HOT has an URI for referencing it, while each Heat stack has an UUID or a name as a reference to it
- A resource instantiation is referenced to either via an URI or via an UUID or via a name

Example: A service resource instantiation is a concrete REST request to a given service, and with a UUID in its request body.

Example: A Heat stack resource instantiation is a daughter stack inside of a nested stack which is referenced to via UUID or name.

Example: A VM resource instantiation in a HOT is a VM which is derived from an image in Glance. The instantiation is described in yaml.

- In the subsequent example for a HOT, a Heat stack comprises everything except the first two lines

```
-- Here begins the HOT
heat_template_version: 2018-03-02
description: Simple HOT to deploy a compute VM
-- from here on begins the Heat stack
resources:                                  -- instantiates an OpenStack resource
  MyVM:                                     -- name of the compute VM that has to
                                            -- be created by instantiation
    type: OS::Nova::Server                  -- OpenStack resource type for a
                                            -- compute VM. OS = OpenStack name
                                            -- space
    properties:                             -- specifies the properties of
                                            -- the instantiated VM resource type
      key_name: MyKeyEnvironmentVariable    -- specifies name of OpenSSH key-pair
                                            -- to be used in the compute VM
                                            -- via an environment variable
      image: MyImageTypeEnvironmentVar      -- Image to be used for the compute VM
      flavor: MyFlavorEnvironmentVar        -- specifies flavor of the compute VM
```

6.20.5 Heat Setup

- Heat has the following components:

1.) **Heat command-line client**:

• this is a component that provides for a CLI that allows manual control over Heat

- it communicates with the Heat-api-cfn component in case of cfn templates

2.) **Heat-api-cfn component**:

- this is the cfn API that implements some compatibility with AWS Clouds
  - it processes cfn API requests by sending them to the Heat engine via oslo.messaging over RPCs
- OpenStack developers should use the native API of Heat and not cfn templates

3.) **Heat-api component**:

- this is the native API of Heat
  - it processes native API requests by sending them to the Heat engine via oslo.messaging over RPCs
- **Heat engine**:
- this is the core of Heat
  - it orchestrates the execution of HOTs
  - it returns REST responses back to the caller
  - it returns also events via oslo.messaging
  - it returns finally the output of Heat stacks

6.20.6 Heat Terminology

Def.: The output section of a HOT defines Heat-stack output-values that are available to the user after the stack has been executed.

Example: IP addresses of launched VMs, or URIs of deployed applications inside of a VM are some Heat stack output.

Def.: A Heat stack snapshot summarizes all resources of a Heat stack in a file with JSON or yaml format.

Def.: A Heat stack action does s.th. and can be called by UUID or name.

Example: SUSPEND and RESUME are Heat stack actions.

Def.: Heat can monitor if an event is risen during the execution of a Heat stack. Subsequently, a REST request can be sent as a consequence. This is called Heat stack event. The corresponding REST response contains the UUID of the event as reference to it.

Def.: Heat can set and manage the software configuration for a project (tenant). This includes the deployment of user applications and tools and is called project software-configuration.

Part III - Amazon Web Service AWS

7 Overview

- The Amazon Web Service AWS started in 2006 with the “Simple Storage Service“ (S3)
- A short time later followed “Elastic Compute Cloud“ (EC2) and “SimpleDB“, thereby providing storage, compute, and database in a bundle to the user
- Meanwhile, SimpleDB was replaced by AWS by “DynamoDB“
- In 2018, AWS had more than 90 services including beta versions that were then not yet in production
- A list of AWS function groups is below
- Each function group comprises a set of services

Function Group	Function Group	Function Group	Function Group	Function Group
Compute	Storage	Database	Migration	Networking and Content Delivery
Developer Tools	Management Tools	Media Services	Security, Identity and Compliance	Data Analytics
Machine Learning	Mobile Devices	Augmented and Virtual Reality	Application Integration	Customer Engagement
Business Productivity	Desktop and Application Streaming	Internet of Things	Game Development	-

Example: The Compute function group comprises 12 services, the Game Development function group comprises 2 services.

- This means, AWS is a whole “Informatics Universe“ which it is not needed to leave since it covers “everything“
- Additionally, in 2017 AWS had a market share of 34% among all CSPs and was still reigning supreme since its start with more than 17 bill. $ revenues
- Finally, AWS uses frequently the terms “simple“ and “elastic“ to indicate that a service has a REST API and is scalable

7.1 Using AWS Services

- In general, managing and using AWS services is easy compared to OpenStack because of the many service-specific consoles that exist and because of the easy-to-read documention about services and consoles
- This documentation exists in abundance and is mostly but not always consistent
- Web pages about AWS are typically enriched with hyperlinks to examples
- This stands in contrast to OpenStack where it can happen that “hackers“ but not professional authors are writing manuals about their code
- However, there are also counter-examples to this rule: the manuals of DynamoDB, for instance, are, from my personal point of view, inconsistent and may lead to confusion

- Furthermore, the many service-specific AWS consoles present themselves to the user in a simple “click-here-click-there-style“ which is easy to comprehend and to use
- On the other hand, only few is known about the internal “making“ of AWS which leads to speculations about it
- Furthermore, source codes of AWS services are typically secret
- Finally, data privacy does not really exist in AWS for their customers in case that organizations such as the U.S. NSA secret service are asking for information
- The reason for that are U.S. laws about the obligations U.S. companies have towards U.S. secret services
- This may hold even in the Frankfurt region of AWS where German laws apply, at least on the territory of Germany

7.2 Calling AWS Services

- AWS services can be accessed via REST requests to specific URIs
- These requests are authenticated or anonymous
- Authenticated means that user credentials are needed to create a signature that is carried in the REST request for improved data privacy and security
- Additionally, a service can be called by language-specific “AWS Software Development Kits“ (SDKs)
- Furthermore, a service can be called manually via a AWS CLI

- **Finally, a service can be called via a service-specific console which is accessible via the master "AWS management console"**

Comment: an AWS console is a web page with a simple and intuitive GUI. There is also a master console which is the login console and that is named AWS management console.

- **Furthermore, some older AWS services can also be called via SOAP over https, but this is depricated**
- **The calling possibilities mentioned above are explained in more detail subsequently**

7.2.1 Calling via an SDK

- **The SDK is a wrapper that uses the REST requests of the respective service API to implement the call**
- **Currently supported languages and thus also SDKs exist for C, C++, Java, JavaScript, Python, PHP, MS .NET, Ruby and Go**
- **To use REST requests or SOAP calls directly is discouraged by AWS**
- **Instead, AWS says that calling via an SDK is the preferred method**
- **The argument for that is that the SDK automatically authenticates the requests by using access keys provided by the user in order to improve data security and privacy**

7.2.2 Regional Endpoints For Calls

- **AWS subdivides the world in so-called regions**

- **There are more than 16 regions**

Example: The Frankfurt region covers Germany and beyond.

- **An region is a large area where regional endpoints exist**

Def.: An endpoint is an URI that is the entry point to an AWS service. In a specific AWS region, regional endpoints should be used by the customer to reduce response latency.

- **Most Amazon Web Services have a regional endpoint for REST requests**

Example: https://dynamodb.Frankfurt.amazonaws.com is the endpoint in the Frankfurt region to DynamoDB.

7.3 AWS Accounts and Users

- **An AWS account is the administrative unit that is charged money for the consumed services and IT resources**
- **Accounts can be owned by companies, institutions or private people**
- **If a company or institution owns an AWS account, it can grant access to this account for its employees which act in turn as AWS customers**
- **A customer always belongs to an account and has access rights to AWS objects and services that are determined and limited by the account owner only and not by AWS**

Def.: An AWS object is an IT resource in AWS.

Def.: An AWS service is the counterpart to an OpenStack service.

- However, better access rights for customers may imply higher costs for the account
- Multiple users for an account can be united by the owner to a user group
- Access rights can be given collectively to a user group which simplifies the administration of users in the account
- Furthermore, every AWS object a user creates is under the full control of the account, i.e. the account owner is also the object owner
- Finally, There exists the notion of "*root user*" in AWS which is the account owner

7.4 AWS Account IDs

- Each AWS account is automatically assigned two unique IDs: an account ID and a "canonical user ID"
- The account ID is a 12-digit decimal number and is used for "Amazon Resource Names (ARNs)"

Comment: For Amazon Resource Names (ARNs) see the chaper below.

- The canonical user ID is a 64-digit hex string (=256 bits) and identifies the private person, company or institution who is owner of the account

7.5 AWS Resource Names (ARNs)

- Furthermore, every AWS object has a string as unique identifier which is the AWS Resource Name (ARN)
- The general syntax of an ARN string is:
  - AWS:<AWS service name>:<AWS region>:<AWS account ID>:<resourcetype>:<resource>
  - or AWS:<AWS service name>:<AWS region>:<AWS account ID>:<resourcetype>/<resource>

Example: the single-character string "i" indicates a VM as resourcetype in an ARN string, "ami" denotes an image file, "vol" a volume, "db" a relational database and "vpc" a virtual private cloud.

Example: MyEnvironment/MyApp, mysql_db, or ExamplePicture.jpg are possible resource parts in an ARN string.

7.6 AWS Identity and Access Management (IAM)

- The administrative service that is supporting authentification and authorization (AA) of users and user groups is called „AWS Identity and Access Management" (IAM)
- Basically, IAM has to deal with three types of access keys:

1.) one type for each user called "IAM User Access Key"

2.) and another type for user applications and external participants called "Temporary IAM Credentials"

3.) In addition to these two key types comes a third key type, which is called "Account Access Keys"

- The Account Access Keys are comparable to the superpin each smartphone has and devoted for the root only, i.e. for the account owner

- Please note that all three key types are independant of the account ID and the canonical user ID and exist in parallel to them

⇒ Each AWS account has 5 highly confidential data serving as IDs and keys

- The AWS keys mentioned above are explained subsequently in detail

7.6.1 Account Access Keys

- An account access key grants full control over the account
- Furthermore, it allows to authenticate every REST request issued by a user of that account by means of calculating a signature out of it

Comment: in AWS, every REST request must be authenticated by a signature. This is comparable to what Keystone does in OpenStack.

- The account access key is a alphanumeric string of 20 characters

- Optionally, 40 characters are also possible which is then named "secret account access key"
- The secret account access key can also be used to compute the signature for a REST request
- Finally, both key flavors can also be employed to uniquely identify an account, i.e. the account itself and its two access keys to it are synonymous to each other
- The same holds for the account ID which identifies also uniquely an account

7.6.2 IAM User Access Key

- Every AWS user or user group has its own IAM user access key that grants or denies access permissions to AWS objects
- The access rights are regulated by "policies" and maintained and enforced by AWS IAM

7.6.3 Temporary IAM Credentials

- In addition to account access key, secret account access key and IAM user access key, also temporary IAM credentials exist which are valid for a prescribed time interval only
- Temporary IAM credential comprise a pair of "IAM security token" and "IAM access key"

Comment: Please do not confuse the account access key with the IAM access key, although both terms sound similar.

- The time interval for which the IAM security token and IAM access key are valid is specified by the user of an account
- The user specifies also what is allowed and what is denied for the temporary credential pair
- This way, a user can give his application software the permission to access AWS services and IT resources via an SDK or REST request
- Furthermore, the temporary credential pair can be used to compute the signature of a REST request as well
- However, it is not recommended to do so for two reasons:

1.) the SDK already provides for the application software the REST authentication

2.) the temporary credential pair cannot be directly used to authenticate a REST request, instead, the user must calculate the signature for the request out of the pair
   - the signature has a validity interval of 15 minutes
     - this is less than the 1 hour validity of Keytone signatures
   - the signature is mainly based on a SHA256 hash of the request body, among other input such as the date when the request was issued

- Furthermore, any user can give a temporary credential pair to external people who are federated with the AWS user and that are outside of AWS in order to cooperate with them
   - Outside of AWS, there is no SDK that could provide for federated users a REST authenti-

cation automatically

7.7 Data Protection and AA in REST Requests

- Each REST header has the following fields pertaining to data protection and AA:

1.) Content-MD5 hash:

- the Content-MD5 is a field that is base64 encoded and contains a 128-bits MD5 hash of the request body according to RFC 1864
- this field is used to verify at the receiver that the received data is the same as the sent data
- it is kind of some bad CRC

Comment: CRC = cyclic redundancy check is a good method to know at a receiver side whether the sent data is corrupted during transmission. Hashing the data at the sender is a not so good method for that because it needs much more bits.

2.) x-amz-content-sha256:

- this field contains the signature of the request and contains a 256-bits hash
- when a service receives an authenticated request, it compares the self-computed signature with the signature received in the request header
  - it continues only if both match

3.) x-amz-security-token:

- this field carries the “IAM security token“ in case of a temporary IAM credential
- in case of a “DevPay“ service it must always contain a credential

8 Description of Selected AWS Services

- What follows is a description of several subjectively selected AWS services that are considered to be interesting and important

8.1 Simple Storage Service“ (S3)

- S3 is a store for file objects, not for general AWS objects

Def.: An S3 object is a file and can have also user-defined metadata about the file.

- When the object is viewed, first AWS-specific metadata is visible, such as the URI to it and its ID, as well as user-specific metadata in the form of keyword=value pairs
- Since only HTTP requests of up to 4 KB header length are supported by the Internet, the amount of metadata is restricted in each request and response
- Furthermore, AWS claims to have an object survival rate (endurance) of 99.999999999% and an accessibility rate of 99.99% per year
- This means concrete that from 100 GB of data, 1 byte is lost every year and that S3 is not accessible for 52 minutes per year which is both not so small as expected
- Furthermore, it is claimed by AWS that the storage capacity S3 has is “infinite“ and that each user can upload an “infinite“ number of objects
- However, every object is capped to 5TB to limit upload time
- This is similar to OpenStack Swift

- Finally, each object in S3 can optionally be encrypted by S3
- This option is called “server-side encryption“ (SSE)
- The encryption keys are either managed by S3 or by another AWS service called “KMS“
- With SSE, encryption and decryption is done automatically for every PUT and GET REST request

8.2 Buckets

- Buckets are sets of file objects in S3
- Each S3 file object must be contained as an element in a bucket
- Each bucket has an owner which is the AWS account
- The account owner is charged money for the S3 service on basis of the number and sizes of the buckets he has
- Furthermore, the name of an file object must be unique only within the bucket but not between buckets
- Additionally, buckets are also involved in access authorization
- Finally, buckets are the units where log files for reports are working

8.2.1 Creation of a Bucket

- The creation of a bucket is a simple clicking process in the S3 console:

1.) the user must click “create bucket“ and give it a name

2.) he must choose the region where the bucket will be stored
- currently exist about 16 regions worldwide
- it is clear that even if “Frankfurt region“ is selected, German data protection laws do not really hold because buckets are automatically replicated
- NSA has access to replicated buckets outside of Germany and inside of Germany

Example: Poland belongs to the Frankfurt region but has less stringent data protection laws.

- bucket replication is said to be performed for the sake of “data endurance“

3.) he must click “create“ and has finished the process of bucket creation

- The upload of file objects is in the same simple click style
- Further operations that are possible from the S3 console are copy, paste, delete, undelete, rename and download

8.2.2 Folders in a Bucket

- Within a bucket, folders and subfolders can be created and named to establish a directory tree

⇒ A file object is typically in a folder or subfolder which in turn is part of a bucket

8.2.3 Retrieving Objects in a Bucket

- The general URL syntax for an S3 object is: *http://<bucket name>.s3.amazonaws.com/{<folder name>/<subfolder name>/ .../}<file name>{<automatic version ID>}*

8.2.4 Automatic Versioning of Objects

- A bucket can be configured to add automatically a unique version number to a new object when it is uploaded to S3 if an object of the same name exists already there
- AWS strives to convince users to store as many versions of objects as possible because each object will be charged for
- They say that it is good for the user to have many versions in order “to avoid inadvertant deletion“

8.2.5 Website Hosting

- A bucket can be configured to host a website with static web pages
- Each browser can get that web pages directly from the bucket without needing an extra web server in AWS

8.2.6 Additional AA for Buckets

- In addition to the account access key, the IAM user access key and the temporary IAM credential pair, bucket access can be extra protected by so-called "multifactor authentication (MFA)"
- MFA is implemented by configuring a so-called "bucket access policy"
- In addition to bucket access policies, there exist also "bucket-access control-lists"

8.2.6.1 Bucket-Access Control-Lists (Bucket ACLs)

- Bucket ACLs are lists formulated in an S3–specific XML schema
- Each ACL contains the permissions a grantee gets for the bucket
- The following permissions exist:
  - *read*: Allows to list the objects in the bucket
  - *write*: Allows to create, write, and delete an object in the bucket
  - *read-acp*: permission to read the bucket ACL
  - *write-acp*: permission to write the bucket ACL
  - *full-control*: includes all four previous permissions
- Interestingly, there is no permission to read an object
- The grantee is either an user, or it is an account, or it is a group of users
- Each REST request to the bucket must have a signature to authenticate the requester

- The granting instance is the bucket owner who has also created the bucket

Example: A bucket owner with canonical user ID gives the write permission to an alien user that does not belong to the own account, so that the alien user can transfer objects to the own bucket. This implies that the PUT request is signed.

8.2.6.2 Bucket Access Policies

- A bucket access policy is formulated in a S3-specific JSON format
- Bucket policies are limited to 20 KB in size
- In a policy, the Amazon Resource Name (ARN) is used to identify the bucket and the subresource for which grants are given or denied
- There are nearly as many policy permissions as there are operations on a bucket in the S3 REST API

⇒ Nearly all S3 bucket operations can be selectivly granted or denied to a grantee by an access policy

- The grantee can be either an IAM user of the AWS account who is bucket owner, or an alien AWS account, or an IAM user of an alien AWS account, or another AWS service, or the grantee can be everybody
- The grantee is called "principal"
- There are no permissions for operations on file objects in the bucket access policy

Example: A bucket access policy gives the GET permission to everybody because the bucket contains a static website. This implies that the GET request is not signed but stems from anonymous, i.e. from everybody.

- A granting instance does not have to identify itself for a bucket access policy because only the owner of a bucket can define a policy for it
- There is some overlap between bucket-access control-lists and bucket access policies which make S3 access protection weird

8.2.7 Other Access Policies and ACLs

- Beside bucket access policies and ACLs, in S3 also "user access policies" and "object-access control-lists" are possible
- Object ACLs use the same XML schema as bucket ACLs
- User policies are expressed by the same JSON format as bucket policies
- These four mechanisms together are the AWS counterpart to OpenStack's RBAC
- Object-access control-lists and user access policies will be explained as follows

8.2.7.1 Object-Access Control-Lists

- Object-access control-lists extend the idea of bucket ACLs to individual objects in it
- There are only 3 permissions for objects which are *write*, *read_acp* and *full_control*, which have the same meaning for objects as already explained for buckets

- The object permissions can be applied by the bucket owner only to the objects he has created
- They do not pertain to objects created by others, and which are transferred to this bucket
- The grantee is the same as for bucket ACLs as well as the granting instance
- Each REST request to the object must have a signature to authenticate the requester

8.2.7.2 User Access Policies

- While bucket access policies were formulated from the view point of a bucket, user access policies are formulated from the view of users and pertain to objects in buckets
- A user access policy gives grants to individual users for specific objects
- There are practically as many permissions for user access policies as there are operations on objects in the S3 REST API

⇒ About every S3 object operation can be granted or denied to a user by a user access policy

- There are no permissions for operations on buckets in the user access policy
- Furthermore, each REST request to the object must have a signature to authenticate the requester
- Additionally, a user access policy is employed to distinguish admins from normal users

- Finally, user access policies are stored as metadata for an account and managed by IAM
- Because of that, user access policies are not proprietary to S3 but are available to all AWS services, thus allowing to grant or deny the calls in the many AWS REST APIs
- The grantee can be either an IAM user of the AWS account who is bucket owner, or an alien AWS account, or an IAM user of an alien AWS account, or another AWS service
- Anonymous permissions to everybody do not exist here, because each policy is always attached to a specific user which is checked by IAM
- As a consequence, each REST request to the object must have a signature to authenticate the requester

8.2.7.3 Personal Opinion

- To check all four authorization mechanisms that are possible in S3 for every REST request is a time-consuming process

⇒ **Already because of that, S3 requests are slow by construction**

- The four AWS mechanisms overlap in part and are thus by far not as clear as RBAC
- Since the mechanism overlap, it is possible that the authorization checks are delivering inconsistent or even contradictory results which is bad

8.2.7.4 Entity Tag for Data Protection

- Each REST response header has an “entity tag“ (etag) for data protection
- With this tag, the sender can know whether the receiver has got correct data because the etag is contained in the REST response that is transmitted back to the sender
- The etag is a hash function of the data part of the object without its metadata
- The etag is in some cases a MD5 hash, in other cases another hash function, depending on which REST call was made

8.3 Eventual Consistency and Simultaneous Writing

- Because of file object replication within the same region which is big, it takes some time until an object write is fully effective (= Eventual Consistency)
- The reason is that a new value must be propagated to every replica
- Therefore, a read immediately after a write may result in no data or stale data
- Additionally, it is possible to simultaneously write different file objects under the same name into the same bucket
- A later read will return the object with the latest time stamp, and the others are lost
- Of course, a user cannot know whether his file has had the latest time stamp
- This means he cannot be sure whether his data was stored in case of simultanous writing

- Furthermore, if a read is made immediately after a simultaneous write then an object can be returned that will never show up again because the timestamp of this object has finally propagated to all replicas and a decision was made that it did not had the latest time stamp
- These unwanted effects happen because there is no object locking in S3 and because there is aysnchronous object replication
- This behavior is similar to OpenStack Swift

8.3.1 Storage Classes

- In contrast to Swift there are “storage classes“ in S3 which differ in price, durability and accessibility
- Each object must be in one of 4 storage classes which are: “Standard“, “Standard_IA“ (for *i*nfrequent *a*ccess), “Glacier“ for low cost and very infrequent access and “Reduced Redundancy Store (RRS)“ for low reliability
- Standard_IA has an availability rate that is lower by a factor of 10 compared to the others
- RRS has a durability which is by a factor 10 Mio. lower than the others which is a lot

8.4 Calling S3

- S3 functions can be called via REST, SOAP over https, SDKs, CLI and the S3 console

8.5 S3 REST API

- The REST calls of the S3 API fall into two main categories which are operations on buckets and operations on objects

8.5.1 Operations On Buckets

- For operations on buckets, the following 4 REST calls are engaged: DELETE, GET, HEAD and PUT
- These 4 operations pertain mainly to bucket metadata
- Exceptions are DELETE bucket, GET bucket and PUT bucket which address data and define delete, read and create operations on them
- In total, there are about 52 operations on buckets alone, which illustrates the complexity of S3

8.5.2 Operations On Objects

- Operations on objects have one additional REST call compared to buckets which is POST
- Operations on the data part of objects are DELETE, GET, HEAD, PUT and POST

8.6 Elastic Compute Cloud (EC2)

- The so-called Elastic Compute Cloud (EC2) is a cloud for AWS but not for its customers

- In EC2, customers can only rent the cloud‘s VMs
- The VMs come with different specializations
- The EC2 availability is 99.99% which means that the rented VM is not reachable at most for 0.9 h per year
- VM rental prices range between 0.6 Cent and 5 $ per used hour
- Beside the pay-as-you-go business model, AWS offers also long-term rental-contracting or the bidding for VMs on online auctions
- However, for the latter exists no guarantee for nothing
- They can be interrupted by AWS within 2 minutes notification time or even reclaimed
- Since recently, AWS allows also to rent real hardware via the Internet
- There are about 16 different types of VMs that can be provisioned by EC2, either manually by means of a few mouse clicks or automatically via auto-scaling
- Both ways are very convenient for the user
- The VM types are either for general purposes or optimized in computing power, main memory size or disk space
- Each VM type has a name and a set of specific properties which would be called flavors in OpenStack
- In AWS, they are called “Amazon Machine Images (AMIs)“
- Traditionally, the hypervisor that has created a user VMs in EC2 is XEN
- Meanwhile, also KVM is responsible for creating 1 of the 16 AMIs

- The rented VM comes always with a preinstalled guest OS which is mainly Linux or Windows Server
- In case of Linux, a Java JDK is also preinstalled, together with the GNU compiler suite
- Because of that, EC2 would fall into the category of PaaS if it were a user cloud
- On the other hand, the user must allocate manually the floating IP addresses he gets with his account to one or more of his VMs which is only needed for IaaS
- The user must also specify manually the protocols, ports, and source IP ranges that can penetrate the VM‘s firewall
- VMs can have three types of storage, the VM-internal one which is ephemeral, an external block storage called “Elastic Block Store“ EBS which is comparable to Cinder in Openstack, or the external object store S3, which are both permanent
- Similar to Cinder, EBS volumes are replicated inside the same region and can have up to 1 TB of capacity
- AWS allows the combining of two or more EBS volumes in order to create a RAID
- Additionally, a DynamoDB relational database can be attached to a VM
- Although EC2 is not a user cloud, a less sophisticated counterpart to OpenStack‘s Ceilometer and “Watcher“ exists which is called “CloudWatch“
- As in OpenStack, availability zones are possible for the physical implementation of VMs
- And as in S3, EC2 is controlled via REST, or a graphical management console, or a command line, or via SDKs

- Finally, as in OpenStack, multiple VMs can be connected in a vLAN by means of VMs' vNICs
- However, virtual routers and switches do not exist, at least not for the user
- As a result, EC2 can be compared to a combination of reduced features from Horizon, Keystone, Nova, Neutron and Ceilometer in OpenStack
- On the other hand, it is kind of an all-in-one solution that may be practical and sufficient for many customers
- EC2 can be augmented to a true private cloud by means of the AWS virtual private cloud (VPC) service

8.7 DynamoDB

- DynamoDB is a noSQL database with some exceptional features:

1.) The DB allows for auto-scaling which is technically remarkable
   - it means that the same database is getting automatically faster if needed
   - it does not mean that more instantiations of the database are launched because this does not help the overloaded DB

2.) Read and write latencies are in the milli second range which is quite fast
   - With an optional in-memory caching, latencies lie even in the micro second range

3.) Solid state drives can be selected as physical implementation of DynamoDB disk drives which allow for constant and deterministic access times
   - this allows for real-time operations in DynamoDB which is also quite remarkable

4.) DynamoDB's code can be installed as executable on the computers of the customer, as it is the case with Trove from OpenStack

5.) It supports data encryption

6.) Pricing can be arranged due to data throughput and not due to consumed disk space

- If DynamoDB would be part of a private cloud, it would fit into the SaaS category since the customer gets a read-to-use sector-specific application, without the necessity to manage it
- On the other hand, DynamoDB does not support SQL queries and has thus also not the ACID properties of relational databases

Comment: ACID = Atomic + Consistent + Isolated + Durable. Atomic means that every read or write is an uninterruptable transaction. The transaction is either completely performed or not performed at all. Consistent means that the DB can change only between valid states. Intermediate states where s.th. changes over time are not visible to the user. Isolated means that concurrent execution of transaction appear to be purely sequential. Durable means that data is valid even in case of power loss, hardware crashes or software errors.

- Instead, DynamoDB has only the D property (durable) and provides for eventual consistency as S3 or Swift do

- At least, DynamoDB offers optionally also a "consistent read" as extra feature

Comment: Consistent read avoids the peculiarities S3 and Swift have in case of multiple simultaneous writes and reads.

  - However, consistent read deteriorates DynamoDB's read latency as well its throughput, but it improves the handling
  - consistent read may be temporarily not available in an AWS region if there is too much network latency between the availability zones of that region because of overloaded Internet
- The listed features above are a result of being a distributed DB which is positive for reliability but negative for user handling
- The reason for being distributed is that DynamoDB creates automatically multiple database replicas in the various availability zones of the same region

⇒ Data security is an issue in DynamoDB because even the same AWS region imply to store data in different countries with different data protection laws

8.7.1 Data Storage and Retrieval

- Data in DynamoDB is stored in a so-called "table" that has a name
- The name of the table is the name of the database
- Multiple tables, i.e. databases can be handled by DynamoDB for the same user at the same time
- Unlike to an relational DB and to sommon understanding, a DynamoDB table is not a 2D list structure that contains all entries as rows and all attributes to a row as columns

- Instead, a DynamoDB table is just a set of elements called "items"
- Each item in turn consists of a set of "attributes"
- However, there is some analogy between DynamoDB and a relational database:
  - the set of items in DynamoDB are analog to the rows of a relational database
  - the set of attributes of an item are analog to the columns of a relational database
  - this is the reason, why DynamoDB claims that it uses a 2D-"table" as data structure, although it is not a table because different items can have different numbers of attributes
    - the number of columns in a real table are always the same and depend not on the row
- Each item is identified by a unique key which comprises either one or two keyword(s)

⇒ One or two keyword(s) are needed per item as access method

- It is required that individual keyword(s) exists for all items to uniquely select a specific item
- The keyword(s) for an item are called "primary key"

Comment: For more explantions about primary keys, see nect chapter.

- Each attribute inside of an item is uniquely identified by an other keyword
- In case of writing into the DB, DynamoDb follows the "keyword=value method" in order to set an attribute in an item
- Furthermore, it is not needed that item keywords or attribute keywords have all the same style

- They can differ by name, type and number (1 or 2)
- Additionally, it is not required that an item contains more than 1 attribute
  - if it contains only 1 attribute, no attribute keyword is needed to select that attribute
- Finally, it is not required that an item contains all of its attributes at once right from the beginning
- Instead, item attributes can be added over time as needed
- It is even possible to split the attributes of an item into disjoint subsets
- As in a relational database, an attribute has a value which is represented by a variable of any possible data type
- However, an attribute is not the atomic data element of a DynamoDB table, as it would be the case in every relational database
- Instead, each attribute can consist of a tree of other attributes which may be nested with a deepness of up to 32 levels
- Only the leaves of this attribute tree are atomic data elements in the DynamoDB table
- In essence, a DynamoDB table resembles much more to a JSON or yaml document than to a classical SQL database
- This is allowed because it is a noSQL database
- As a consequence, JSON documents can be directly written into a DynamoDB table as content

8.7.2 Primary Key

- The only data types allowed for primary keys are string, number or binary
- Nesting of data types into data structures is not allowed
- If a primary key addresses an item that has only one attribute then it has the additional property of being a "partition key"
  - The reason for that property is that the primary key to such an item is hashed by DynamoDB to directly address a physical record on a disk drive in order to store quickly the single-attribute item
  - DynamoDB retrieves also that item from the disk quickly by means of a hash of the partition key
- Technicall spoken, the situation could be as follows (however, the following is speculative, because to my knowledge it is not yet disclosed by AWS):
  - If a primary key addresses an item that has two or more attributes then the first key may still have the property of being a partition key, while the second key, which is called "sort key", is needed to select one of the item's attributes
  - The reason for that could be that DynamoDB may try to store all attributes that belong to the same item, i.e. the same partition key jointly together in the same disk record, sorted according to their sort key value
  - this would improve performance singificantly
  - however, this method would have also its limits, if afterwards an attribute has to be added to an item and the number of bytes for all attributes exceeds the size of the physical record (typically 4 or 8 KB)

- DynamoDB is offering also another feature to its customers by allowing them to have more than one key to address and access the same item in a DynamoDB table
- Beside the primary key each item may have a secondary key which does the same as the primary key, but has consequences for DynamDB‘s internal data structures

8.7.3 Secondary Key

- A secondary key is not required but, if existing, can be used as an alternative to access table items
- For every secondary key, DynamoDB creates a new table, i.e. a new database in which item data from the original table is copied
- The derived table is called “secondary index“, but not secondary table which is totally confusing from my personal point of view
- The original table is called “base table“ because it contains the authoritative data and the derived table contains only copies of item attributes

Def.: A secondary index is a table that is derived from a first table called base table. The so-called secondary index (table) is organized in a different way as the base table and uses another access key than the base table.

- In contrast to a primary key for a base table, the primary key for a secondary index (table) must always comprise a partition key and a sort key
- There are two variations of secondary indices (tables): global and local

- Consequently, there are two ways to address and access the items in global or local tables which are called Global Secondary Key and Local Secondary Key

8.7.4 Global Secondary Key

- For a global secondary index (table), any two attributes that exist in all items of the base table can be chosen as primary key

8.7.5 Local Secondary Key

- For a local secondary index index (table), the partition key of the derived table and of the base table must be identical, while the sort key must exist as attribute in all items of the base table

8.7.6 Multiple Secondary Indices

- DynamoDB supports up to 5 global secondary indexes, i.e. tables which are derived from the same base table and also up to 5 local secondary indexes, i.e. tables, derived from the same base table
- This implies the same amount of global and local secondary keys

8.7.7 DynamoDB Streams

Def.: A “stream“ is a data record that is automatically created and output by DynamoDB if a change has occured in a table item.

Comment: DynamoDB is using frequently non-standard terms that are confusing, such as “tables“ which are databases, “secondary indices“ which are derived databases and “streams“ which are log files for database transactions.

- **A stream can be used as input to another AWS service called “Lambda“ which in turn triggers an action or workflow as soon as an event of interest appears in the DynamoDB stream**
- **Streams are also used for other purposes, such as for automatic updates of DynamoDB‘s replicas of databases**
- **In essence, streams are a useful extension for a database**

Much Success!