# OPEN TELEKOM CLOUD (OTC)

# TECHNICAL MIGRATION GUIDE

Version 3.1
Date 2018-08-03

# TABLE OF CONTENTS

## Change Log

| Version | Chapter | Change |
|---|---|---|
| 3.1 | 3.4.2 | Included KMS: Bring your own license / key |
| | 3.6.1 | Included encryption function and Microsoft HA-compatibility |
| | 7.3 | Included Mobile Storage Solution |
| | 12 | Included encryption function |
| 3.0 | 8 | New chapter: OTC Hybrid Solution |
| | 7.1 | New paragraph about DIS & DDS |
| 2.6 | 2.9 | Included new picture |
| | 3.1.4 | ~~four~~ five components need to be defined |
| | 3.6 | New paragraph about using IaaS services for PaaS |
| | 3.6.1 | Updated to include the possibility of reaching RDS from the internet |
| | 4.1 | Updated text to include KVM as hypervisor |
| | 4.1.4 | Updated text to include KVM as hypervisor |
| 2.5.5 | 1.2 | Updated decision process picture |
| | 2.1.1 | Updated introduction text to be more clear and concise |
| | 2.2- 2.5 | Changed headline |
| | 3.1 | New picture, changed text |
| | 3.1.1 | Updated text to include release 2.2 |
| | 3.6 | New chapter to reflect PaaS services of release 2.2 |
| | 4 | Chapter 3 split up into Chapter 3 and 4, changed headlines |
| | 5.3.3 | Updated text, more information |
| | 7.1 | Updated picture to include release 2.2 |
| | 9 | Updated text, more information and added pictures |
| | 10 | Small updates in the text to reflect changes in release 2.2 |
| | Overall | Improvements regarding language quality, clarity, and information |

# 1 OPEN TELEKOM CLOUD MIGRATION GUIDE

## 1.1 SCOPE

This document describes how to make use of the public cloud services (IaaS / Infrastructure as a Service) offerings from T-Systems' Open Telekom Cloud (OTC) and how you migrate services from other infrastructure to Open Telekom Cloud.
The document starts with an introductory chapter to give some general overview how to take advantage of scale-out cloud infrastructure and understand the advantages and SLAs that are considerably different from traditional physical or virtualized ("Enterprise Cloud") infrastructure.

Specific challenges to get virtual machines and services working well in Open Telekom Cloud are addressed next along with some considerations on security and deployment automation.

In the next chapters, we discuss several use cases where existing services are deployed into Open Telekom Cloud with increasing difficulty:

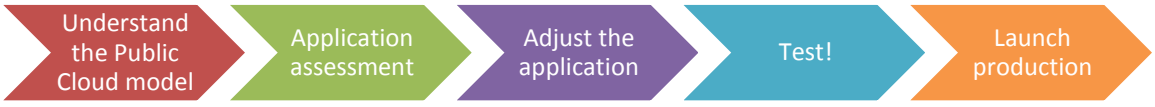
- Using Open Telekom Cloud for test and development environments
- Using Open Telekom Cloud for a web presence
- Migrating an application from another scale-out cloud platform
- Advanced topics such as the migration of large monolithic applications that require significant re-architecting to restructure the application with scalable small services is not covered in this document.

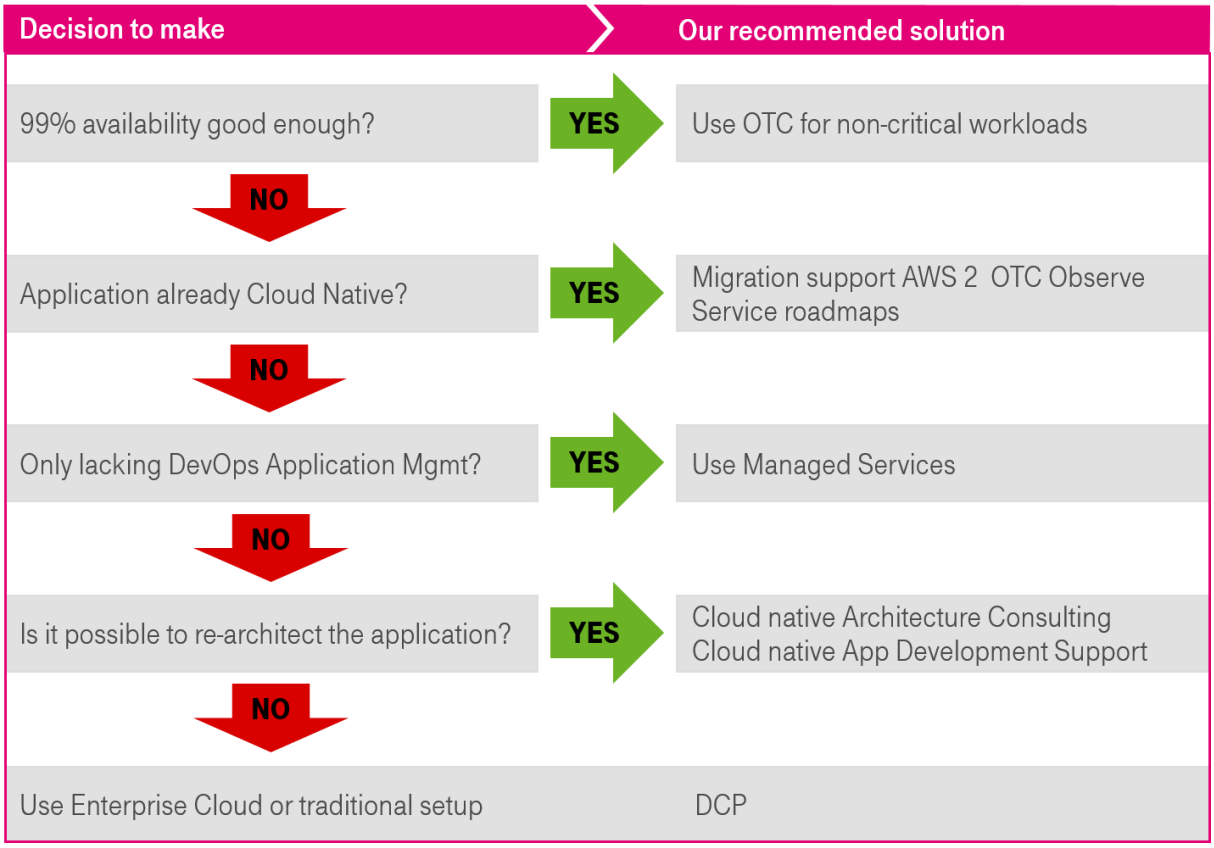
Every cloud migration project will involve several phases. The business requirements for any application and its architecture will be analyzed with respect to its suitability for public cloud and the effort to make it work well. Governance and organizational structures will be considered and potentially be adapted. There will be a negotiation and selection process with vendors, consultants and service providers. And the benefits in terms of flexibility, future-proofness and - of course - cost will be scrutinized, before a decision in favor of a migration project is taken. This document cannot provide advice on the overall but also different process structures and governance nor does it describe best practices for cloud transformation projects which also may differ case by case. But it does explicitly focus on the general technical considerations to make applications work well on Open Telekom Cloud and take advantage of its benefits.

## 1.2 MIGRATION STEPS TOWARD OPEN TELEKOM CLOUD

As most customers will be used to the traditional or enterprise cloud infrastructure model the first step of a migration towards the Open Telekom Cloud should start with a learning session to understand the Public Cloud model. There needs to be an understanding of the Public Cloud architecture and what it means for the applications which shall run on it.

The different use cases for the Open Telekom Cloud provide guidance for the next step: identifying the perfect application candidates for the Open Telekom Cloud. There needs to be an assessment of each application which covers availability requirements, scale-out design, legal/ SLA and needed transformation skills.






After the applications and the use cases are identified the application needs to be adjusted to a lower or higher degree in most cases. Certain functions can be replaced by platform services or standard scalable open source building blocks. The platform services of Open Telekom Cloud need to be chosen and customized to fit perfectly together with the application. This should result in the desired availability and resilience.

When the application is adjusted and platform services are ready a thorough testing phase must be conducted. The testing should be automated as fully as possible (even consider test driven development). Failures of instances on the platform are a key part of it (e.g. “Chaos Monkey” testing service) to achieve the required service availability of the application. Monitoring the service availability is key. Monitoring the health of the virtual machines is not as instance failure is common in the cloud. Dealing with instance failure will improve the resilience of the application.

Besides the testing of the application the migration of application data to the Open Telekom Platform needs to be taken care of. Depending on the amount of data the migration can become a rather huge challenge. “Bandwidth as a Service” is introduced with the Open Telekom Cloud to cope with part of the challenge. A shift to the public cloud brings also change to operations processes. A decision should be taken if managed service or DevOps better fits the operations model for the application and the business.

# 2 SCALE-OUT CLOUD APPLICATIONS

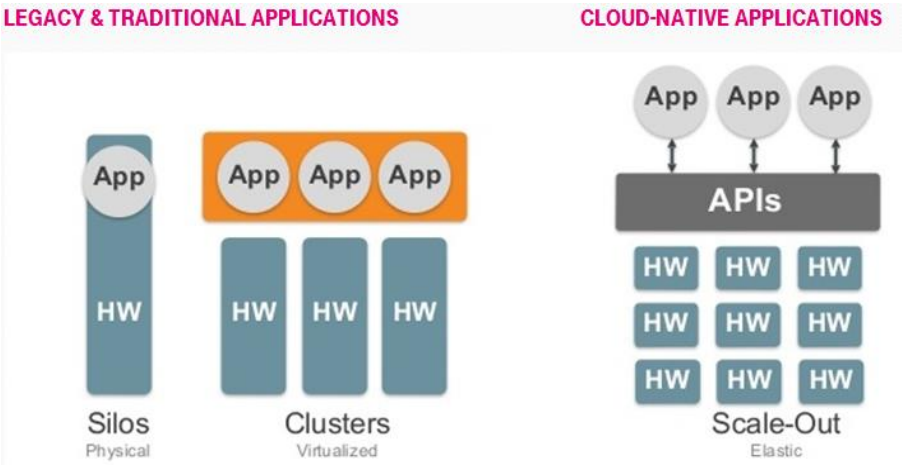
Scale-out Cloud Architecture is based on different paradigms than traditional infrastructure.

Engineers have spent decades to make infrastructure (hardware, hypervisors, operating systems, ...) more reliable. However, there are diminishing returns: Squeezing more 9's (i.e. increasing availability above 99,999%) out of the infrastructure incurs higher and higher complexity and sharply increasing cost.

On the other hand: the extension of commodity infrastructure has given it a huge economy-of-scale advantage and Scale-out clouds have turned the equation even around: What if we could build reliable services on top of only moderately reliable (read: commodity) infrastructure?

Answer: Service availability is the only thing that counts!

During the last decade, the development of paradigms that fit into the scale-out cloud architecture has emerged and been leveraged to produce very reliable services with very high flexibility at much lower cost than traditional models would have allowed.




The target of this chapter is to introduce the most relevant concepts for scale-out cloud application architecture to sketch the perfect citizen for Open Telekom Cloud. The principles are agnostic to the exact underlying cloud – they apply not only to Open Telekom Cloud, but also to other OpenStack based clouds or other scale-out clouds such as AWS.

## 2.1 ARCHITECTURE AND DESIGN PRINCIPLES

### 2.1.1 OF PETS AND CATTLE

Traditionally, servers (physical machines or VMs) are unique machines with long procurement or provisioning times. Once the machine exists, a sysadmin configures them carefully and often manually, and much effort is spent to keep the server up and running. This includes meticulous change and update management with the goal of as little outage per machine as possible.

Once the number of servers gets too big for a small team of sysadmins, additional systems and tools are used to keep an overview of the state, configuration and health of individual machines. If an important server fails during service hours, the user experience suffers directly. This is the reason why such an event is treated as an emergency in traditional IT – the responsible team is alerted to bring the failing machine back to life. This way of using a server has been compared to treating it "like a pet" by Randy Bias[1].

For a cloud application, the approach is radically different. As new VMs can be created and disposed of at near-zero cost and all configuration can be fully automated, the focus for health monitoring shifts from individual machines to service availability.

[1]See http://cloudscaling.com/blog/cloud-computing/the-history-of-pets-vs-cattle/

One VM failing doesn't have an impact on user experience anymore, because there are automatic mechanisms in place to replace it immediately. If there is more traffic than the current VMs can handle, more VMs are started and configured without any manual effort. Once the traffic spice is over, the now superfluous VMs are shut down again, automatically. This is the "cattle approach" to managing machines.

There are a few angles to look at this paradigm to make it clearer:
- Modular scale out design
- State and Statelessness
- Automation of Configuration
- Failure domains.

### 2.1.2 MODULAR SCALE-OUT DESIGN

Scale-out designs tend to break tasks and jobs in small services (micro services) that can be used by an API (often REST-based) and that can scale. If the service on one VM cannot fulfill the throughput requirements, more VMs can be started and share the load. This is a service driven architecture: services talk to underlying services to fulfill their jobs and handle failures by reconnecting after a small timeout. Some application scaling logic ensures that the health of the service (and not of the machines!) is monitored and more VMs are started if needed (autoscaling).

### 2.1.3 STATELESSNESS

When VMs come and go, they are not the place to save state. A VM can die anytime – as Netflix put it on their famous "Chaos monkey" posting: Instance Failure is frequent.

The resilience of a cloud application comes from the fact that it survives the death of a VM without losing any data – if the VM has no persistent state, none can be lost.

On 'Amazon Web Services', the root disks of most instance types (flavors) are fugitive – if you reboot, they will be unaffected again. This strongly encourages a model where no one even thinks of using the root file system of a VM as persistence layer for anything. On Open Telekom Cloud, the root disks are persistent by default to make the transition from a classical world easier, but it is a useful mindset to consider them as non-persistent.

If a VM needs to store persistent data, it needs to be written to some place that is not exclusively owned by the VM: A database, an object in the object store or (if it can't be avoided) a data disk that is attached to the VM for this purpose.

The big advantage of a stateless VM is that it can't lose persistent data on being killed, crashing or being cold migrated.

The configuration of a stateless VM needs to be stored in the image or – better – be injected from the outside on bootup. This is the topic for the next section.

## 2.2 AUTOMATING VM CONFIGURATION

Rather than creating an image that has all needed configuration preinjected already (and this way creating a need for image proliferation), a best practice in scale-out clouds is to use a small set of generic images and inject configuration at bootup.

In the Linux world, cloud-init has become the standard tool to do this configuration and customization at bootup. cloud-init reads a "YAML" configuration file (in the image) and gets additional configuration from outside configuration data sources in OpenStack environments, the meta-data server (btw: on http://169.254.169.254/ provides this metadata source.)

Cloud-init consists of dozens of modules that can set hostnames, install package updates or additional packages, resize the root partition and root file system (to fill the disk), set passwords or inject authorized ssh keys or hook up into a full blown configuration management system such as chef, puppet, saltstack or ansible. It can of course also do simple things such as injecting files or running scripts. This way, on bootup (every bootup if you have ephemeral root disks, only the first boot for persistent root disks), the generic image is configured to become a specialized VM that fulfills the task it has been started for.

## 2.3 DON'T RE-INVENT THE WHEEL

Restructuring services in ways that they scale and tolerate failures can be significant work. Fortunately, the success of the open source movement predates the emergence of scale-out cloud offerings. There are numerous good open source solutions available that can be used as building blocks. Just like the cloud uses commodity infrastructure, there are commodity building blocks available: Key-value stores, No-SQL databases, Configuration management systems, Code management and CI/CD systems, Messaging platforms, and more.

Having identified a service that is part of an application design, the following steps are recommended:

- Is there a service that the platform offers (such as object storage, RDS, ...) that fulfills the need?
- Is the SLA good enough and the lock-in effect justified? Then use it.
- Is there a company that has created software with a similar need and has it published it? Even as open source maybe? Or at least things we can learn?

## 2.4 AUTOMATION - USING THE API

Fast scaling and fast recovery don't work if the system needs manual attention to react. Even in the classic world, a good operator never does the same thing manually twice. In the cloud, the operator uses the tools to automate deployments, recovery, scaling.

A few tools are used very often for this purpose. Scale-out cloud environments have an API. You can use the API to automatically create virtual networks, virtual storage and virtual machines. The APIs are typically REST interfaces – with simple tools, you can compose the requests and parse the response from a cloud service. Chances are though that you will find higher-level abstraction tools, such as "Terraform" (abstracts AWS' CloudFormation and OpenStack's HEAT).

In addition to the API tools, you will likely use Configuration Management systems such as "Chef", "Ansible", "Puppet" or "Saltstack" to automate the deployments and maintenance of your environment.

## 2.5 UNDERSTANDING AVAILABILITY ZONES AND SLA

Some events affect not just a single host or rack, but with bad luck, a complete datacenter may lose its connectivity or power or may go down in a fire.

While a cloud-native application is in general well prepared to deal with failure, this is a challenge: If all machines providing a specific service go down at once, there will be a service interruption – hopefully a short one until new VMs have come up and can take over.

This can be avoided by using the concept of Availability Zones. The cloud setup is distributed over several, geographically distributed datacenters that are independently powered and connected to the net, so a catastrophic event is unlikely to affect both at the same time.

Applications have visibility on the availability zone. Resilient applications will replicate the persistent data across several AZs and thus survive the death of one AZ without service interruption.

It is important to understand the availability (SLA) definition of a scale-out cloud: Based on commodity components, the hardware does only provide moderate guarantees. While the cloud control plane is secured by redundancy, this is not the case for compute hosts. If they go down, all the VMs go down as well. A dying VM is a normal event on a scale-out cloud – if the application cannot deal with it, it will not achieve more than 99% availability. A scale-out application will just restart the VM somewhere else – as it is stateless, nothing got lost.

The SLA for Open Telekom components VMs ("Elastic Cloud Server" and storage ("Object Storage Service") guarantees an availability of 99,95% per month. Excused events will not be counted as outages. Excused events are: the component is available in a different Availability Zone, there is an alternative instance of the component for the customer, outage occurred

due to maintenance work, the outage was caused by the customer or a third party. An application that thus wants to provide services at 99.95% availability needs to replicate the data it needs to work across the two zones.

## 2.6 MONITORING

The focus on monitoring should not be VMs. VMs not being well is a frequent event and no reason for alarms.

The focus for cloud applications is on the service, therefore the services need to be monitored instead of VM instances. Good cloud applications have instrumentation (availability and response time / load monitoring) built into every layer and thus control the scaling as well as health reporting from the application.

Ultimately, the end user service counts – for monitoring this, a service outside of the cloud is the ultimate measurement. There are services in the net available.

## 2.7 AUTOMATIC SCALING OUT

Having the application instrumented to monitor the health of the services that compose it allows taking automatic action. When a service is unable to handle the load, new virtual machines can be started to provide additional capacity (scale-out). Likewise, if the load remains low for a certain amount of time, additional VMs can be removed again (scale-in). This mechanism also helps to recover from failed VMs – the same scale-out mechanism will work to replace failed VMs.

There is a scenario that happens very often in scale out applications: There is a load balancer that distributes the requests from another layer to a service. The Open Telekom Cloud platform offers a mechanism, where you create a custom image for VMs providing the service. The VMs are grouped in an autoscale group, and monitored with respect to response times (http or tcp) or VM metrics (CPU load etc). Policies can be defined that allow for automatic reactions when certain thresholds are exceeded for a specified amount of time. If new VMs are provisioned (scale-out), they will automatically be added to the related load balancer. Likewise, for the scale-in operation.

These autoscale groups provide a very simple and convenient way to do scaling, because most web applications do not need to be changed in any significant way to support this model of operation.

## 2.8 TESTING

In addition to the normal testing, two additional capabilities need to be validated:
- Automatic scaling
- Recovery from failed VMs

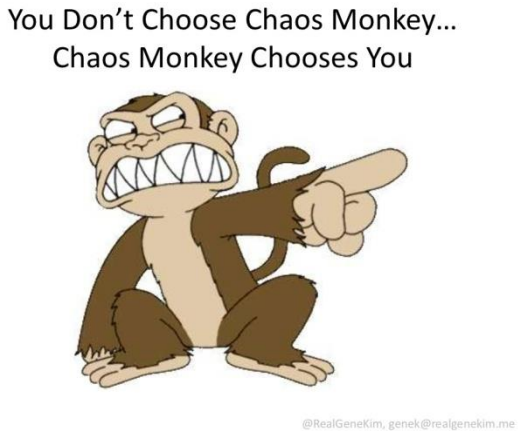
A nice tool to test the latter is the

**Chaos Monkey**

The Chaos Monkey is a famous beast, invented by Adrian Cockcroft of Netflix. Netflix realized that the only way to deal with the frequent failures of VMs in a public cloud is to consider VM failure as a normal event.

To test for this, the Chaos Monkey was written and released (as Open Source). It randomly kills VMs of an autoscale group.

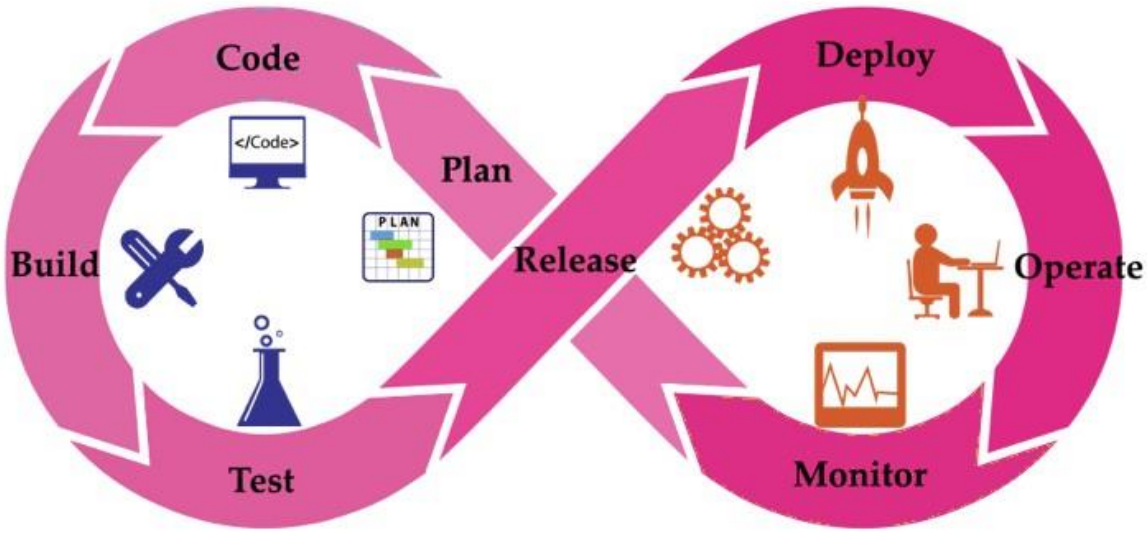
Netflix meanwhile runs this during business hours (!) on their production system (!!) to validate that the resilience of their application is at the expected level.



## 2.9 CHANGE MANAGEMENT, CI/CD AND DEVOPS

Most cloud applications are developed in an agile way. The code is constantly built (Continuous Integration - CI) and very frequently deployed into test environments (Continuous Deployment - CD). Deployments into the production environment also happen very often; weekly deployments are typical.

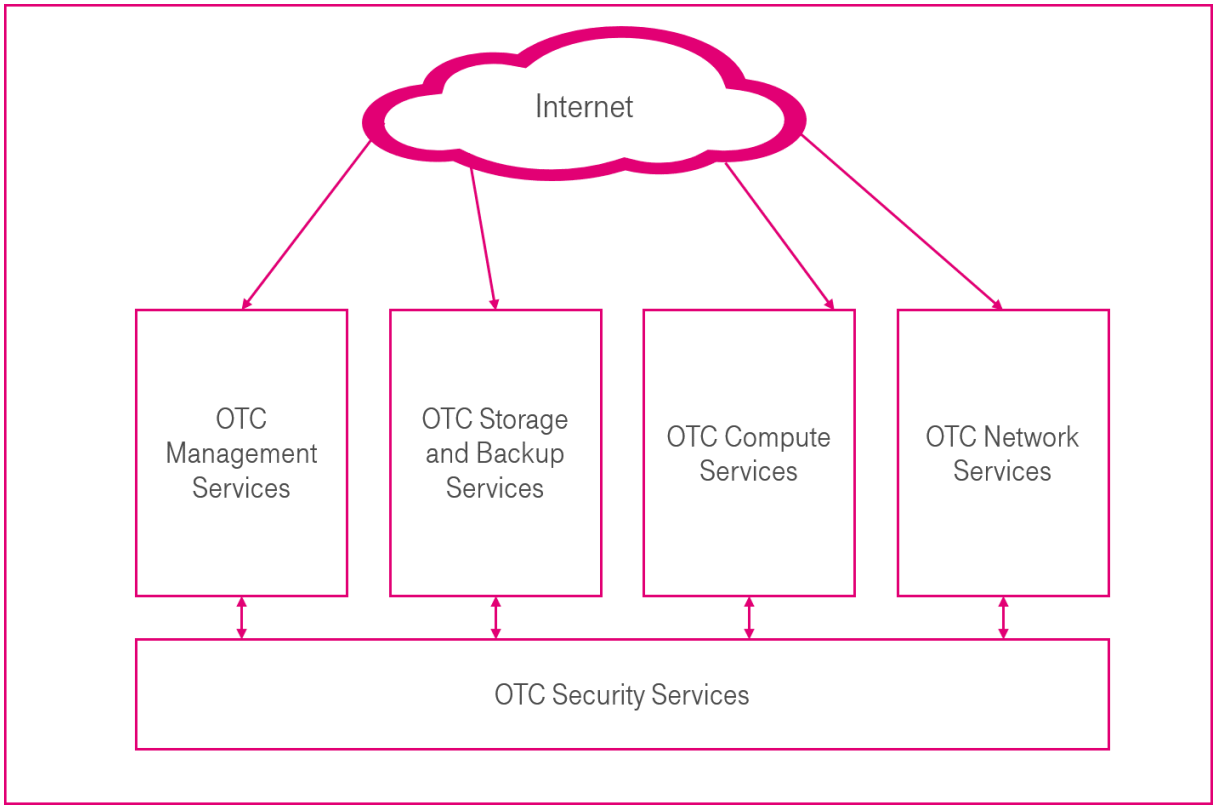
The deployment process itself is part of the engineering – the Development and Operations teams both are involved in it and have a joint responsibility to ensure that deployment and operations work. Overcoming the traditional development vs. operations silos has proven very beneficial – the close collaboration model is referred to as DevOps.




Heavyweight change management processes that try to avoid change wherever possible do not fit very well in this world. Instead the testing and validation is automated as is the deployment and change process. Should serious errors slip through nevertheless, a quick rollback action can be taken.

# 3 ARCHITECTURE OF THE OPEN TELEKOM CLOUD

The following picture provides an overview of the architecture of the Open Telekom Cloud.




The shown subsystems

- Compute Services
- Storage and Backup Services
- Network Services
- Security Services
- Management Services

will be described in this chapter. Additionally, there are PaaS services available, which are described in section 3.6.

The next chapter will describe the Image Management Service in detail. For the sake of completeness, it is still included in the architecture overview.

## 3.1 COMPUTE SERVICES

### 3.1.1 ELASTIC CLOUD SERVER

An Elastic Cloud Server (ECS) is a computing server that consists of CPUs, memory, images, and Elastic Volume Service (EVS) disks and allows on-demand allocation and elastic scaling. The ECS integrates Virtual Private Cloud (VPC), virtual firewall, and multi-data-copy capabilities to build up an efficient, reliable, and secure computing environment to ensure that your services are running stable and without any interruption.

The self-service feature of the ECS allows you to create an ECS by yourself. You are required to pick only a predefined flavor with the matching amount of vCPU-memory image specifications, and login authentication mode. Then, the ECS you requested is allocated within the minimum time required. In addition, you can modify ECS specifications based on your requirements at any time.

Basic flavors
Four basic flavors are defined. The difference between them is a fixed allocation of the ratio vCPU and RAM. Those flavors are Computing I (1:1), Computing 2 (1:2), General Purpose (1:4) and Memory Optimized (1:8).

GPU Optimized
GPU Optimized flavor enhanced the basic resources with GPUs (graphics processing unit). It is optimized for scenarios that require high-performance graphics processing, such as graphics rendering or image processing.

High Performance
High Performance flavor provides stable and ultra-high computing performance. This flavor is used e.g. when applications have ultra-high computing and high throughput requirements such as particle physics, engineering or simulations/modeling. The CPU's reserved are not shared with other users so maximum performance is delivered over the whole lifetime of the project. The high Performance flavor is available with Linux OS (CentOS, Ubuntu, openSUSE)

Large Memory
Extensive RAM resources are available and provide high memory-read bandwidth. It is well suited for in-memory and distributed cache applications such as IMDBs and SAP Hana

Disk Intensive
Provide high-network performance and uses local storage. They are purpose-built for applications that require high IOPS and fast data processing, such as distributed Hadoop computing and the processing of large volumes of concurrent data and log.

Workspace
Workspace flavor is used for virtual desktop environments and addresses standard workplace scenarios as well as desktop demands for higher graphical power (GPU).
The Workspace flavor is available with Windows (Desktop-) OS.

### 3.1.2 DEDICATED HOST (DEH)

Usually on Public Clouds hosts can be shared with several customers. On one host some ECSs are running which share the physical availability of the host, but are separated from customer to customer by using VPN (Virtual Private Networks). On a dedicated host (DeH) only one customer can run his ECSs.

When you migrate services to a DeH, you can continue to use your server software licenses used before the migration. So you can use the Bring Your Own License (BYOL) feature on the DeH to reduce costs and independently manage your ECSs.

For the migration to a DeH there is no difference to shared host.

### 3.1.3 BARE METAL SERVER

Bare Metal Server (BMS) is a physical server that is dedicated for your use. It provides high computing performance to meet requirements of key application scenarios. Bare Metal Server can be used in conjunction with other cloud services, such as Virtual Private Cloud. With Bare Metal Server, you enjoy both the stable performance provided by server hosting and high scalability offered by cloud resources.

### 3.1.4 AUTO SCALING

Auto-scaling is an event-driven engine that allows automatic, horizontal scaling of EVS instances. To enable Auto-Scaling for a specific use case, five components need to be defined:

- Auto-Scaling Configuration (Which image to use?)
- Auto-Scaling Trigger & Actions (What to do when?)
- Elastic Load Balancer (To distribute the traffic)
- Unified Load Balancer (To distribute the traffic)
- Auto-Scaling Group (The combination of the above + basic settings)

Auto-Scaling Configuration
The Auto-Scaling configuration describes which image should be deployed once the trigger is hit. This includes the image (either a private or public image) as well as the instance type of the ECS. Please note, that it does not include a network configuration yet - this is done in the Auto-Scaling Set.

Auto-Scaling Trigger & Actions
For a minimum configuration two trigger points are needed - one to scale-out in cases of high load, one to scale-in when the load lowers.
The triggers can be set on CPU, Memory and Network usage bus also be scheduled once or on a regular basis. The trigger action is then to either deploy or reduce a certain number of ECS instances or scale to a specified value.

Elastic Load Balancer
Tthe need for a Load Balancer is self-explanatory. As Auto-Scaling it the mechanism to utilize horizontal scaling the incoming load/traffic needs to be distributed to the available number of instances. To achieve exactly that effect Elastic Load Balancer is available in Open Telekom Cloud. Used in combination with the Auto Scaling Services newly deployed nodes are even automatically added to the ELB configuration as backend servers.

Unified Load Balancer
Unified Load Balance (ULB) is a service that automatically distributes incoming traffic across multiple backend servers based on a specified forwarding policy. In addition, ULB supports centralized deployment of the internal and external networks and active/standby VRRP deployment, access through VPN's, direct connections and across VPC's.

Auto-Scaling Group
The Auto-Scaling Group brings the above explained components together and defines the proper environment. This includes foremost the minimum as well as maximum number of instances the specific Auto-Scaling Group may reach. In addition, the needed networks and security groups are specified to deploy the ECS instances.

## 3.2 STORAGE AND BACKUP SERVICES

### 3.2.1 OBJECT STORAGE

The Open Telekom Cloud offers a S3-compatible Object Storage Service, where users can store file-based data. The Object Storage is accessible through a HTTP / HTTPS / S3 interface. Object Storage is the only service on the Open Telekom Cloud that can be used stand-alone and does not require a ECS instance.

It offers a wide variety of extra attributes that can be defined such as Access Control Lists, Versioning, special functions for static website hosting or black-/white lists for URLs.

OBS allows users to encrypt objects using server-side encryption so objects can be securely stored on OBS. This is useful for a higher security level due to data migration. Before uploading data to OBS, you can encrypt the data to ensure security of transfer and storage. OBS supports all encryption technologies.

The OBS provides three different storage classes:

- Standard – for frequently accessed
- Warm – for semi frequently accessed
- Cold - for rarely accessed

### 3.2.2 ELASTIC VOLUME SERVICE (BLOCK STORAGE)

Block Storage is available for the ECS as either a system or data disk. It comes in three different types of I/O and throughput characteristics: "Common I/O", "High I/O" and "Ultra-High I/O" which are backed up by SATA, SAS or SSD disks. The “Ultra-High I/O” SSD is additionally available in an optimized for latency version for the Large Memory Flavors.

For a normal ECS instance a system disk and up to 10 data disks can be attached.

### 3.2.3 VOLUME BACKUP SERVICE

Through the Volume Backup Service, a snapshot of an Elastic Volume can be created. This snapshot is crash-consistent to the current state of the ECS instance volume. From this snapshot either a new volume can be deployed or a restore of the original disk can be triggered.

## 3.3 NETWORK SERVICES

### 3.3.1 VIRTUAL PRIVATE CLOUD

The Virtual Private Cloud (VPC) is the dedicated network environment each user has on Open Telekom Cloud. It is completely isolated from other VPCs, communication between VPCs can only be done through the Internet.
A VPC contains all network definitions, Subnets, Route Tables, Security Groups, Firewall (Network ACL), Elastic (Public) IP addresses. The VPC peering feature allows communication between different VPC's. Virtual Private Network (VPN) and Direct Connect offers the connectivity to an on-premise data center.

### 3.3.2 ROUTE TABLES

A route table contain a set of rules that are used to determine where network traffic is directed. It is used e.g. for enabling ECS without an EIP access to the internet using an ECS with a Public IP and they are used for VPC Peering.

### 3.3.3 SUBNETS

As in tradition networks within Open Telekom Cloud and the VPC custom subnets can be created to separate different workloads and ECS instances from each other. With different subnets, also the creation of multi-tiers applications or DMZ scenarios are possible to be created.

A subnet is always located in one Availability Zone, communication between subnets is possible if it is allowed by the security group.

### 3.3.4 SECURITY GROUPS

A Security Group is a collection of rules which define the access control for an ECS. It is not exact like a firewall in a classical environment but quite similar. You can specify allow rules to access to the resources which belongs to the security group. In a Security Group Rule, you can define the protocol (tcp, udp, icmp, any), direction (inbound and outbound), which port or port range you want to allow to access and the source which want access. The source can be a Network segment or another security group.

### 3.3.5 FIREWALL (NACL)

A Firewall consists of one or more network access control lists (NACLs). Based on inbound and outbound network ACLs, the firewall determines whether data packets are allowed in or out of any associated subnet.

### 3.3.6 ELASTIC IP

An Elastic IP address is a public IPv4 address. This address is always connected to a dedicated bandwidth which only applies to outgoing traffic. Incoming traffic is not limited by the bandwidth.

Normally an Elastic IP is bound to a Load Balancer or an ECS.

### 3.3.7 VPC PEERING

A VPC peering connection is a network connection between two VPCs that enables you to route traffic between them using private IP addresses. A VPC peering connection can be setup between two VPCs inside a tenant or between a local VPC in the own tenant and a VPC in another tenant within the same region.

### 3.3.8 VPN

A virtual private network (VPN) establishes an encrypted communication tunnel between a remote site and a VPC. This enables the communication between the OTC and another data center or your private network.

### 3.3.9 DIRECT CONNECT

To use Direct Connect, you need to have a contract with a network provider. Then, using Direct Connect, an individual communication channel to and from the resources on Open Telekom Cloud can be established. The results are high transfer speeds, stability, and more options for greater security of the network connection. With Direct Connect, hybrid scenarios can easily be introduced.

Open Telekom Cloud's Direct Connect Service allows network connections to be built, modified and ended. Pricing is usually based on port prices. Direct Connect can handle bandwidths of between 1 Mbit and 10 Gbit. Direct Connect only supports the MPLS VPN.

Direct Connect can support your migration. With Direct Connect you can speed up data migration, it is much faster than the internet connection between your datacenter and the Open Telekom Cloud.

### 3.3.10 DOMAIN NAME SERVICES (DNS)

Open Telekom Cloud also offers a Domain Name Service. Domain Name Services (DNS) turn names of internet sites into their associated IP addresses. This principle can also be used for resources on the Open Telekom Cloud to guide users to resources.

The most common use-case occurs when Open Telekom Cloud resources or services offered there need to be offered within a corporate network. Domain names can also be managed with the DNS, increasing the ease of use of the Cloud. DNS is available via the Open Telekom Cloud console and is protected by Anti-DDoS.

## 3.4 SECURITY SERVICES

### 3.4.1 ANTI-DDOS

The Anti-DDoS service defends against "Distributed Denial-of-Service (DDoS)"-attacks initiated at ISO/OSI-layers 4 through 7.

This service protects instances from various kinds of DDoS attacks, such as CC, SYN flood, and UDP flood. The service allows the setting of desired bandwidth, thresholds and access control parameters. In addition, dedicated reports and alarming are available.

### 3.4.2 KEY MANAGEMENT SERVICE (KMS)

Key Management Service (KMS) is a secure, reliable and easy-to-use hosting service used for centrally managing and safeguarding user keys.
KMS uses a hardware security module (HSM) to protect keys, The HSM module helps you to create and control keys. All keys are protected by the root key in the HSM. KMS controls and monitors all activities on keys in logs which in turn, enables the supervision of any regulatory compliance requirements.

KMS provides central management and control capabilities of CMKs for Object Storage Service (OBS), Elastic Volume Service (EVS), Image Management Service (IMS) and user applications. It is applied in data encryption and decryption scenarios.

With the July release of 2018, KMS supports Bring Your Own License and Bring Your Own Key.

### 3.4.3 IDENTITY AND ACCESS MANAGEMENT (IAM)

Identity and Access Management (IAM) is an enterprise-level self-service cloud resource management system and provides user identity management and access control functions.

With IAM, users can manage user accounts and control the operation rights of these accounts over the resources and modules in OTC. IAM also ensures account security and reduces security risks for enterprise information by allowing users to set login verification policies, password policies and access control lists (ACL).

Additional to a standard login procedure with user and password there is a two-factor authentication available.
This works like online banking. After submitting the user and password a TAN is send to the mobile aligned to the user. This function must be activated in the Identity and Access Management.

## 3.5 MANAGEMENT SERVICES

### 3.5.1 CLOUD EYE

The Cloud Eye Service is the monitoring service of the Open Telekom Cloud. It can be used to monitor different usage parameters like CPU, RAM, Disk or network utilization of ECS instances or network elements like Elastic IPs or Elastic Load Balancer. Based upon threshold values it is also possible to configure alarming rules. Alarming is possible to either an Email address or per SMS to a mobile phone.

### 3.5.2 CLOUD TRACE SERVICE

The Cloud Trace Service (CTS) provides records of operations on cloud services resources. With CTS you can query, audit and backtrack operations. CTS provides three types of operation records:

- Operations performed on the management console
- Operations performed by invoking supported APIs
- Operations triggered by cloud services

### 3.5.3 TAG MANAGEMENT

Tag Management Service (TMS) is a service used to centrally and quickly manage your tags. It enables you to tag and categorize cloud services across regions. You can use the TMS console or APIs to access the TMS service. The information from the Tag Management can also be used to track cost per tag.

## 3.6 PAAS

In contrast to the *Compute* instances of the Open Telekom Cloud, the PaaS services already contain the necessary software to start directly with the particular offering. No additional installation is needed. With PaaS, there is for instance no need to worry about the configuration of the Server OS in a mySQL-environment. Start right away.

Even though the focus of OTC is on IaaS, some IaaS services can be used as part of PaaS implementations. Examples for this are Cloud Container Engine (CCE) or Data Warehouse Service (DWH)

### 3.6.1 RDS – RELATIONAL DATABASE SERVICE

RDS in Open Telekom Cloud is a tailor-made database service for RDB-environments based on mySQL, PostgeSQL and MS SQL. There are flavors for different requirements in CPU and RAM, starting from 1vCPU and 2 GB RAM up to higher performance systems with 8VCPUs and 64 GB RAM. Storage can be added based on the individual size and performance requirements. For mySQL and PostgreSQL databases, public internet access can be configured allowing applications running outside of OTC to utilize RDS.
The service offers an automatic-backup function as well as a wide range of management tools for analyzing and optimizing the overall performance of database operation. Encryption functionality is also included.

The RDS also supports high availability for Microsoft, mySQL and PostgreSQL also across different availability zones.

### 3.6.2 DCS – DISTRIBUTED CACHE SERVICE

Distributed Cache Service is an offering to complement the relational data base offering with Redis (NoSQL) instances. Caching the storage in the RAM via NoSQL capabilities allows fast data access, but is only persistent as long as the VM is up and running. DCS supports five different data types: string, hash, list, set and sorted set which allows the usage of DCS as a cache server for real-time analytics, high-speed transactions or message queuing.

Clusters for high availability environments can be applied by individual DCS instances.

### 3.6.3 DWS – DATA WAREHOUSE SERVICE

Data Warehouse Service integrates data from different data sources by consolidating them in a database and a data model. in nearly any sector examples of such approaches can be found, where data must be collected and evaluated based on different data sources.
The Data Warehouse Service enables data mining and also the analysis of larger data sets by uploading data from different data sources located anywhere to the Open Telekom Cloud.
The VM setup and storage offerings are similar to the Relational Database Service.

### 3.6.4 CCE – CLOUD CONTAINER ENGINE

The Cloud Container Engine based on Kubernetes offers the user a complete container service to manage all container relevant tasks like clusters, images, templates and container-capable applications as well as operation of the applications.
Working with Kubernetes requires the creation of a cluster in which users set up and manage their container applications. This also includes the virtual resources as well as their connections to each other. Own internal images can be used but also external images from Docker Registries.

For redundancy needs, the Cloud Container Engine can also be deployed across different availability zones

All Taks for setting up an environment are carried out by an easy to use drag-and-drop graphical interface

### 3.6.5 WORKSPACE SERVICE

The Workspace Service creates cloud-based desktops, which can be accessed from anywhere, based on Workspace Flavors of the *Elastic Cloud Server* offerings.
The flavors address standard (office-) desktops as well as desktops running graphics-intensive applications. Desktops can be operated (using a vGPU) on a Windows 7 or Windows 10 user interface. Licenses can be brought in.

The workstation systems can be managed and rolled out centrally. Despite this deployment from the cloud, nearly all remaining and necessary local services and data from any location and device can be accessed. The data produced while working are normally stored in the OTC-environment.

# 4 IMAGE MANAGEMENT

## 4.1 USING OWN VM-IMAGES

The Open Telekom Cloud allows the user to bring their own images and run them as virtual machines. The registration is done in two easy steps:

- uploading the image to the Object Storage Service and
- registering it in the Image Management Service.

As the underlying platform is relying on XEN and KVM as hypervisors, some additional drivers may be needed - depending on the OS type and release - to enable all functions of the platform.
Private images, regardless if created through an ECS instance or an image file, are only visible to the local user and nobody else on the platform.

### 4.1.1 UPLOAD AND REGISTRATION

In a first step, the image needs to be uploaded to the Object Storage Service. For access to the Object Storage Service please refer to the according documentation. For easier management, it is recommended to create a separate folder where the images are uploaded to. For images, larger than 5 GB multi-part uploads need to be enabled to successfully upload to the platform.

To then register the image the Image Management Service on the self-service portal needs to be used. Through the "Create Private Image" Button the dialog for registration is opened. Source can then be either an already installed ECS instance (which needs to be shut down) or an image file from Object Storage. Select the proper image file, a suitable name, the OS type & release and a size for the system disk. This disk size will then be pre-selected when a machine with this image is deployed.

Once the job is submitted, the Open Telekom Cloud will convert the uploaded image file from Object Storage and store it for further usage in the Glance service.

### 4.1.2 CREATION OF "GOLD MASTER" IMAGES

As described above it is also possible to create private images from a ECS instance. This gives the user the possibility to configure a machine to suit his needs and install all needed applications.

This ready-made ECS can then be converted to a private image that is being used as basis for further deployments. This is especially useful for the Auto-Scaling Service, where instances with a running application need to be deployed.

### 4.1.3 USING HEAT / RESOURCE TEMPLATES

Heat enables the automated deployment of cloud resources and is particularly useful for storing statements that need to be executed repeatedly. Defined system landscapes do not have to be manually configured each time.

If you migrate from another OpenStack platform to OTC you can use your HEAT Templates in the HOT-Format. An adaption to other name conventions, e.g. for flavors of ECS will be necessary.

### 4.1.4 XEN DRIVERS

While the virtualization layer provides emulated hardware, which is modeling the real hardware, better performance can be achieved by providing a higher abstraction for the interfaces. Driver specifically created for the hypervisor are called paravirtualized drivers.

Open Telekom Cloud uses a variant of Xen as hypervisor – the I/O performance is greatly enhanced by using Xen paravirtualized drivers.

For recent Linux distributions, these drivers are easily available – starting with the Linux Kernel 3.0, the so-called pv_ops Xen drivers work well. For distributions with older kernels, drivers can be compiled using the source from the uvp-tools package. There is a project in OpenBuildService, where sources (under the GNU GPL v2 license) and binaries in form of kernel module packages (KMPs/kmods) for many distributions are available.

For openSUSE (up to 42.1) and SUSE Linux Enterprise Server (SLES – up to SLES12 SP1), the pv_ops drivers are not enabled. Instead the drivers from SUSE's xen-kmp should be included in the image. This has been done for all the preloaded images.

The Windows images have the Xen drivers included as well.

In addition to Xen, all Basic flavors are available with KVM as hypervisor.

### 4.1.5 UVP MONITOR

Open Telekom Cloud provides additional functionality by inserting an agent, uvp-monitor, into the operating system. This agent feeds information to the host to allow the host to collect monitoring data. It also supports operations such as soft shutdown, snapshots and live migration.

uvp-monitor for Linux has also been released under the terms of the GNU GPL and is available in source form and as packages for various distribution in the Open Telekom Cloud project in OpenBuildService.

The preloaded images already contain uvp-monitor – both Windows and Linux images. The use of uvp-monitor on Linux is highly recommended – for Windows it is a precondition to have a supportable Operating System.

## 4.2 THE OPEN TELEKOM CLOUD API

The Open Telekom Cloud is based on OpenStack. This makes the Open Telekom Cloud compatible with the open industry standard.

The API offers the possibility to configure the infrastructure (network, storage, virtual machines, images, ...) via a programmable interface – infrastructure as code.

Open Telekom Cloud offers the standard OpenStack APIs with several enhancements that are specific to Open Telekom Cloud. Customers looking for maximum compatibility are recommended to focus on the standard APIs.

The OpenStack API (as well as the Open Telekom Cloud enhancements) is based on REST calls. The application sends an https request to the API gateway, authenticating itself in the header and encoding the request as a JSON object. The cloud executes the call and returns a JSON object again.

There are command line tools (the classic nova, neutron, glance, cinder tools as well as the modern OpenStack tool that wraps all the functionality into one tool) that can be called from the command line and be embedded into simple scripts.

To use the enhanced Open Telekom Cloud functions, there is also a specific Open Telekom Cloud tool, which can be used instead of or in combination with the pure OpenStack tools.

# 5 USING OPEN TELEKOM CLOUD FOR TEST AND DEVELOPMENT

This chapter describes how the OTC can be used for development and test of applications and services. In terms of a development strategy it provides ideas how to make use of Open Telekom Cloud for Source Code Management. It provides further suggestions for the build or transition of additional (the development team supporting) tools into Open Telekom Cloud as well as how Open Telekom Cloud could support the release build process and strategy. In terms of testing it lights up the setup and management of test environments on Open Telekom Cloud and perform certain test scenarios like functional, load, acceptance testing and others.

Applications and services will be developed due to varying reasons: e.g. one may just want to adopt or integrate existing software into its own business solutions while for someone else software development is core business. But not only the motivation to create software differs: A broad variety of software characters are getting created - from monolithic, up-scaling applications to out-scaling web applications or automated agents etc.

However independently from that software projects have always a tough timeline regarding time to production without any compromises in terms of quality. Furthermore, for every software development project an appropriate set of development tools and practices are mandatory to be successful. Limiting factors during such projects beside people and skills are often IT capacities or budget. A lifecycle of software integration usually is dividable in these segments:


This chapter focuses onto the three phases in the middle (development, build and test) because only these are Open Telekom Cloud-relevant. Each of them requires another kind of infrastructure.

In this regard, Open Telekom Cloud offers several advantages to development projects. IT resources and various cloud services are instantly usable and only for the consumption is to pay for. Thus, with a transformation of development and test projects to Open Telekom Cloud the necessity for own expensive equipment is getting decreased if not eliminated which also leads to a reduction of operational efforts. Operating own IT infrastructure comes mostly along with an investment for a three- or more years period while in contradiction to that development projects utilize these resources often only for hours, days or weeks.

This results usually in huge idle periods and the desire to share the infrastructure among certain projects. But the inflexibility of the own infrastructure in terms of scaling and adoptability often leads in such cases to several issues such as increased coordination efforts, project-across incompatible adoptions, blocking of resources etc. All of them influence somehow the cost- and time-consumption to be calculated for projects and they are even factors which can increase the risk for delay or fail of such. This also means that the time one has to spend for all that comes at the expenses of the project itself.

Open Telekom Cloud offers great options to overcome these issues:

- Compared to a traditional in-house hardware approach significant savings can be achieved by only deploying the resources on Open Telekom Cloud which are really required for the deployment and test phase - in terms of size and duration.
- Resources can be allocated very granular and thus project needs and budget can be met much better than with legacy approaches.

Beside these economic advantages Open Telekom Cloud also comes along with many operational benefits:

- The setup of complete development and test infrastructures is on Open Telekom Cloud rather a matter of minutes than weeks or months like in traditional IT development projects (A circumstance which potentiates the economic aspect even more.).
- Furthermore up- and downscaling of capacity as needed is a similar quick and simple task which even could be managed automated and rule based while up and downtimes can potentially be aligned with the productive time of the project members.

So, this chapter is going to explain a couple of best practices and ideas on how to utilize Open Telekom Cloud best for development and test scenarios. For the development part, it will provide for instance approaches on how to set up tools and processes like Source Code repositories or collaboration tools in a secure and long-lasting manner. Within the test part it enlightens aspects like automated provisioning of test environments and opportunities for test use cases like load tests, stress tests and others.

## 5.1 DEVELOPMENT

Proper development tools and facilities are key to success in each software project regardless of the type or complexity of the software itself, the amount of developers or the intended runtime of the project. A defined and practiced operations model for these tools is required like for any other IT system as well. Administrating and maintaining such tools on Open Telekom Cloud frees project member from many low-level operation tasks known from traditional approaches such as hardware setup or network configuration and thus boosts the execution of tasks on higher level. The next section describes how to provision and operate main components for development on Open Telekom Cloud.

### 5.1.1 PROVISION A VERSION CONTROL SYSTEM

Version control with a source code repository is an important task of a developer or development team. It must be available at every time and the containing data is durable and consistent to store and backup. Covering this requires resources, expertise and time from the development team which not necessarily has the competency and capacity.

Using Open Telekom Cloud to build up a version control system usually requires these (high level) steps:
- Creating an Elastic Cloud Server (ECS) instance which satisfies the compute requirements of the version control software
- Assigning an image to that instance which satisfies requirements of the version control software to the Operating System
- Configuring a dedicated EVS volume as data store for the repository content to keep this data separated from the system data
- Attaching the EVS volume to the ECS instance
- Installing and configuring the version control software on the ECS instance, for example GitHub
- Assigning a public Elastic IP (EIP) to the ECS instance to make the version control system accessible for the developers from outside

Optionally and in terms of durability and reusability these additional steps are recommended:
- Creating a private image from the ECS instance after the version control software was installed and configured.
- Setting up regular snapshots with Open Telekom Cloud's Volume Backup Service (VBS) to ensure durability and to facilitate migrations.

The advantages of this approach compared to a classic realization are amongst other things the following:
- Setup and configuration of any ECS instances, EVS volumes and EIPs is done within minutes.
- Costly and slowly in-house ramp-up of a physical or virtual system - which is probably even to procure - becomes needless.
- The Image Management Service (IMS) provides several ready to use and cloud-optimized Operating System images.
- Compute (the ECS instance) and Storage (the EVS volume) capacities are scalable on-demand and on the fly.
- ECS instance and EVS volume persist independently from each other. The EVS volume could be attached to any other ECS instance as well.
- The creation of a private image afterwards would simplify the recreation of the instance in case of a failure or it could serve as a template for further version control instances.

Because developers access the repository via the EIP the underlying infrastructure could be changed transparently, e.g. the ECS instance could be replaced.

### 5.1.2 SETUP A SYSTEM FOR ADDITIONAL TOOLS

Besides a version control system in software development projects usually several other tools are involved to manage the project. These are for instance tools for task tracking and collaboration or for analyzing the quality of software code. Many of such tools are designed as kind of web application. Like any other legacy web application, they need a system to run on as well as quite often a relational database as backend.

Using Open Telekom Cloud to initially build up a system and a database for the supporting applications of a software development project usually requires these (high level) steps:

- Creating an Elastic Cloud Server (ECS) instance which satisfies the compute requirements of the supporting software.
- Assigning an image to the instance which satisfies requirements of the supporting software to the Operating System.
- Provisioning of a Relational Database Service (RDS) instance.
- Configuring dedicated EVS volumes for the ECS instance and the Relational Database Service (RDS) instance.
- Attaching the EVS volumes separately to the ECS and the RDS instance.
- Installing and configuring the application software on the ECS instance(s).
- Assigning a public Elastic IP (EIP) to the ECS instance to make the application system accessible from outside.

Additionally, to guarantee a satisfactory level of availability two possible steps are conceivable. Either:

- Creating a private image from both provisioned and configured systems to decrease the complexity and duration of a necessary recreation in case of a failure.

Or:

- Setting up more than one ECS instance as application server - if the application follows a scale out architecture - and configuring the Elastic Load Balance (ELB) feature which distributes incoming traffic across the multiple ECS instances.

In case the development team or number of tools grows and more capacities is required the following steps can be undertaken to overcome this issue. For scale out applications:

- Creating scaling polices for the Open Telekom Cloud Auto Scaling (AS) feature to define thresholds for automated out- and in-scaling - so starting and stopping of additional ECS instances running the application within the ELB cluster

Or for scale up applications:

- Preparing a more powerful ECS instance intended to replace the old instance.
- Taking a private image from the running.
- Attaching the private image to the new ECS instance.
- Attaching the corresponding EVS volume the new instance.
- Ramping down the old instance (short downtime).
- Assigning the EIP to the new instance.

The advantages of this approach compared to a classic are similar those mentioned in section 4.3.1 plus:

- RDS decreases the complexity of managing and maintaining a dedicated database as it contains automated backup integration
- ELB and AS in combination help to automatically adopt the compute capacity to the actual need
- ELB alone helps to increase the availability level of the application and compensates failures of single ECS instances.

### 5.1.3 SETUP OF DEVELOPER SYSTEMS

Local laptops or desktops are primarily the locations where developers perform their developmental work. Here typically the IDE is installed, code is checked in, first tests are executed and so on. Although, there are scenarios where development environments hosted in Open Telekom Cloud on-demand can make sense. If a software development project uses specialized software sets for its purpose which are difficult to handle and require significant resources for installation and maintenance on local systems it can be wise to prepare and configure such development environments with all the required tools on OTC.

A typical approach to achieve this consists of the following steps:

- Deploying a ECS instance intended to represent the reference developer system.
- Installing and configuring all necessary developer software and tools on this instance.
- Taking and storing a private image from that ECS instance.
- Creating a new ECS instance deploying the taken image to this instance whenever a new developer system is required.
- Configuring the Open Telekom Cloud Virtual Private Cloud (VPC) feature if an interconnection to services such as LDAP in an on-premise environment is required.

The advantages of this approach compared to a classic realization are amongst other things the following:

- Setup and configuration of a new developer system is done easy and within minutes.
- Costly and slowly in-house ramp up of a new developer system - which is probably even to procure - becomes needless.
- Is the project over and the developer system(s) no longer needed it can be shut down and resources will be freed up with an immediate effect to costs.

- The concept of hosted desktops is not restricted to development environments. It can also be applied to other roles or functions.

## 5.2 BUILDS

Application building is a procedure of certain tasks such as compilation and packaging. In case of larger software most of the tasks are combined with several (inter-)dependencies like building internal libraries, using helper apps, writing documentation and so on. Sometimes it's even intended to build the software for different CPU architectures or operating systems. In consequence, the build task itself can take quite a long time to get finished which directly effects the flexibility of the project team. A circumstance which even weighs more if the to develop software inbuilt methods like Continuous Delivery (CD) where every new feature or bug fix leads to a new build release.

### 5.2.1 PROVISION A BUILD ENVIRONMENT

There are different build strategies imaginable which will be explained later in this chapter. But independently from that every build job requires a build system which checks out the current stage of the software's source code from the repository and builds a current version of the software.
A general approach to achieve this persists in the following steps:

- Deploying an ECS instance as build system.
- Installing and configuring all necessary build software and tools on this instance.
- Granting the build system access to the repository data to be able to check out the current stage of development.

### 5.2.2 IMPLEMENTING A BUILD DATA STORE

Each time a build is created the outcome must be stored somewhere. Within usual software development projects, the data to be stored is small at the beginning but grows during the project runtime with each build. OTC Object Storage Service (OBS) with its on-demand and flexibility attributes is an adequate opportunity here and is fully S3-compatible.
Using OTC OBS to build up a data store for the various software builds usually requires these steps:

- Provisioning of the OBS data store itself.
- Assigning an EIP to that OBS data store.
- Granting the Build system write access to that OBS data store.
- Setting up life cycle policies for automated deletion of older not needed builds.

OBS also offers a variety of options for the distribution of data to any kind of audience. In terms of a development, build and test workflow that means the builds can be distributed to the test environment directly out of the OBS data store. This requires the following actions:

- Exporting the OBS data store read only via HTTP/HTTPS or S3 protocol to the test ECS instance(s) or complete test environment.
- Restricting the access via the implementation of bucket permissions.
- The main advantage of this approach compared to a classic realization is amongst other things the following:
- The size of the OBS data store can be tightly and on-demand adopted with the growing numbers of builds which positively impacts the costs due to the pay-as-you-go nature of the OTC OBS feature.

### 5.2.3 MANAGING NIGHTLY AND ON-DEMAND BUILDS

To shrink the issue of the build duration software development teams often fall back on nightly build approaches, divide the project into smaller chunks or combine both strategies. The day after is mostly intended for testing and thus a not finished build job would hurt accordingly.
So, performing nightly build jobs requires usually these elements and steps:

- An ECS instance as configured as build host like described earlier.
- Access onto the code repository from this build host.
- An OBS data store for storing the build result.

If a monolithic build run would not finish during night:

- Down breaking the project into smaller chunks.
- Parallel building of the chunks.
- Plugging together these chunks to get a complete build.

The main advantage of this approach the following:
- Assurance that the builds is getting completed during the night. Thus, office times can be efficiently used for development and testing.

Nonetheless this approach come also along with a critical aspect: Splitting of the project can increase the complexity as afterwards someone or automate should ensure all parts are put together in a functioning manner. To overcome this problem an alternative approach would be indicated:
Empowering the ECS instance which is the build host with more compute capacity on demand that the build process can stay monolithic.

Are multiple builds to perform each day a single OTC ECS instance is probably not capable to produce all of them as fast as expected. To solve that circumstance multiple OCS ECS instances can be ramped up simply and quickly to spread the certain builds among them. This setup can even be automated by the usage of the OTC Auto Scaling (AS) feature. With Auto Scaling (AS) build hosts capacity can be - based on defined policies - increased during peaks and vice versa be decreased during inactivity to optimize costs.

## 5.3 TESTING

Testing is a key element during software development projects to achieve a satisfactory quality level for the software. Tests are a standard method to indicate and fix issues during the project phase already before releasing the software to production. And as better tests are planned and executed as more do they help to limit the by undetected issues generated costs. Usually, the following tests are conducted:

- Unit test
- Module test
- Performance test
- System Integration test
- End to End test
- Penetration test
- User Acceptance test

All tests scenarios have in common that they require IT infrastructure to run on. That implies test teams have similar issues and conditions like development teams. A bunch of enough IT resources is indicated but only claimed infrequently during the execution of the tests. And in contradiction to the infrequent utilization the requirements to the test environment are changing frequently and from project to project.

The pay-as-you-go and on-demand model of OTC offers certain added values to overcome these limitations. OTC provides great options to test teams as test infrastructure can be deployed and flexibly adopted in just minutes instead of weeks or months. Further OTC relieves the test teams from the necessity to invest in expensive in-house infrastructure and from significant operational efforts.

### 5.3.1 PROVISIONING OF SINGLE TEST INSTANCES

As described already earlier OTC ECS instances can be simply deployed by taking any public image available in OTC's Image Management Service (IMS). IMS images combine the OS and other software or configuration files pre-installed on the ECS instance. Beside that it's also possible to implement own - so called "Bring Your Own" (BYO) images.
Thus, getting a single test instance up and running usually persists in these tasks:
- Creating a ECS instance and EVS volume in suitable sizes.
- Attaching either a requirement fulfilling public IMS or BYO image to the ECS instance.
- Deploying of any by the test software presupposed tools.
- Deploying of to be tested software itself.
- Running either manually or better automated the intended test.
- Collecting and reviewing the test results.

Remark: To optimize this process in terms of efficiency for a repeating number deployments of test systems its indicated to create a private image from the first deployed and configured test instance. For all following instances only these steps would remain:

- Creating a ECS instance and EVS volume in suitable sizes.
- Attaching the former created ready to use private to the ECS instance.
- Running either manually or better automated the intended test.
- Collecting and reviewing the test results.

The issue of private IMS image based deployments is just that every time software is to upgrade or configuration is to change insight a new image is to take from an ECS instance again. Even the process of image creation can be automated as well this nonetheless forces to draw a management and maintenance concept for an increasing number of private images.

To limit the number of private images and the corresponding maintenance efforts another option is to include only core components into a private image such as the OS, libraries or mainly static applications and to leave more volatile components like the software under development out. These volatile components can be hooked up to any ECS instance later during its boot via the cloud-init method as described earlier in section 2.2.

### 5.3.2 PROVISIONING COMPLETE ENVIRONMENTS

Before the execution of any test often a complete environment must be prepared - not only a single instance as described in the section before. This includes infrastructure provisioning, deployment of a test version of the developed software and orchestration of test runs on various systems. But having enough resources to deploy the complete application stack and all corresponding systems and services is just one part of the challenge. Another one is to initialize the test environment again and again in the same manner as they should be identical between test runs to achieve comparable results.

That OTC can be operated programmatically using the OTC API or CLI tools is a valuable aspect in terms of automation of test runs on OTC. The automation of test use cases can be planned down to the operation of all components. This directly affects the efficiency of test teams in a positive way as it relieves them from manual tasks known from traditional environments. Furthermore, automation decreases the probability of errors caused by human intervention. And an aligned build process and automated test environment can help to follow Continuous Delivery approaches. Each new build could follow an automated provisioning of a test environment and an automated test execution on it.

### 5.3.3 PERFORMANCE TESTING

Functional test use cases are a useful method to prove the quality of software in general. Solely they don't allow any conclusions in regard of the performance characteristics and limits of an application or complete environment.

Typically load expectation can be divided into two categories: Either upper limits are well known and defined by the software development project or the target of the performance tests is to get out these upper limits by gradually increasing the load in such as way until the system cannot operate anymore.

#### 5.3.3.1 PERFORMANCE TESTING WITH OPEN TELEKOM CLOUD INSTANCES

One of the main difficulties in load testing is being able to generate large enough amounts of inputs to push the tested system to its limits. Typically, it implies having large amounts of IT resources to deploy the system to test, and to generate the test input, which requires further infrastructure. Since load tests generally don't run for more than a couple of hours, the OTC pay-as-you-go model nicely fits this use case.

So, to realize load testing with the help of OTC ECS instances these steps are typically indicated:

- Creating of several Elastic Cloud Server (ECS) instances.
- Assigning an IMS or BYO image to that instance which satisfies the requirements of the test suite.
- Installing and configuring the test suite on the various ECS instances.
- Implementing Auto Scaling to decrease the number of test instances and this way the generated load automatically.
- Running the load test

To limit the efforts in relation with the deployment of the test instance it's a good approach to create a private image from the first instance for the deployment of any following instance as described already earlier. Even better and to support AS would

be to decouple the test suite from the instance image and hook it up to any ECS instance later during its boot via the cloud-init method as described earlier in section 3.1.1.

#### 5.3.3.2 PERFORMANCE TESTING AN APPLICATION ON OPEN TELEKOM CLOUD

Performance testing an application running on OTC is useful to make sure that elasticity features are correctly implemented. Testing a system for network load is important to make sure that Auto Scaling and Elastic Load Balancing configurations are correct.

Auto Scaling offers many parameters and can use multiple conditions defined with OTC Cloud Eye Service (CES) to scale the number of instances up or down. These parameters and conditions influence how fast an Auto Scaling group will add or remove instances. An OTC ECS instance's post-provisioning time might also affect an application's ability to scale up quickly enough. After the initialization of the operating system running on ECS instances, additional services are initialized—such as web servers, application servers, middleware services, etc. The initialization time of these different services affects the scale-up delay, especially when additional software packages need to be pulled down from a repository. Performamce testing will provide valuable metrics on how fast additional capacity can be added into a particular system.

### 5.3.4 USER ACCEPTANCE TESTING

The idea of user acceptance tests (UAT) is to provide a current stage of the software or product to a group of friendly users to check if the project requirements and specification are met. Early user tests can further help to detect conceptual issues of the software.
Through testing of software more frequently, functional implementation errors and user interface or application flow misconceptions can be detected earlier, lowering the cost and impact of correcting them. The more often acceptance tests are conducted, the better for the project, since end users provide valuable feedback to development teams as requirements evolve. However, like any other test practice, acceptance tests require resources to run the environment where the application to be tested will be deployed. OTC with its on-demand and pay-as-you-go nature is also very adequate for UAT scenarios.

Using some of the methods described earlier in this document, OTC allows the automation of the provisioning process and of disposing environments no longer needed. Test environments can be also provided for certain times only. By deploying the acceptance test environment within OTC VPC, internal users can transparently access the application to be tested. Besides, such an application can also be integrated with other production services inside the company, like LDAP, etc.

## 5.4 CONCLUSION

Development and test tasks require various resources at various times of a software development project. Those resources are mostly limited, inflexible or insufficient or not even available in a legacy environment. They are sometimes even wasted since not all the time needed.
OTC provides a more cost-efficient and flexible approach compared to such traditional infrastructures. Instead it takes weeks or even months to get hardware, resources can be provisioned instantly then needed, scaled up as the workload grows, and release resources when they are not needed anymore.
This flexibility allows to focus onto the project, not on operating and maintaining the infrastructure. OTC also provides opportunities that were difficult to implement in legacy environments: Resources can be fully automated on OTC so that environments can be provisioned and decommissioned without human intervention.

# 6 MIGRATING WEB APPLICATIONS TO OPEN TELEKOM CLOUD

## 6.1 DESIGN OF A CLOUD-NATIVE WEB APPLICATION

A cloud native application is a highly horizontal scalable designed application which should be world completely stateless and is only configured during boot and runtime. The single instance is no longer important for the availability of the service, due to the highly-redundant design availability is measured at the outer edge of the application and not for the single instance.

For more details regarding the general design pattern please refer to Chapter "Scale-out applications" - this chapter is focused on the concrete example of a web application.

### 6.1.1 REQUIREMENTS

The following elements are needed to compose a simple 1-tier cloud-native web application:
- Public IP address
- Elastic Load Balancer with the following Listener
  - HTTP (Port 80)
  - HTTPS (Port 443)
- ECS instance with an installed web server e.g. Apache
- Suitable storage to store the webserver data (see next chapter)

Optional:
- Auto-Scaling Group for automatic provisioning / de-provisioning of webserver instances

These basic elements compose a cloud-native web application, a sample configuration will be explained in more details in the following chapters, also with increased complexity for more tiers.

### 6.1.2 DATA STORAGE

As explained in the beginning, good applications and therefore the ECS instances are designed stateless. This brings several challenges for the configuration as well as the storage that holds the actual data that the webserver is presenting to the WWW.

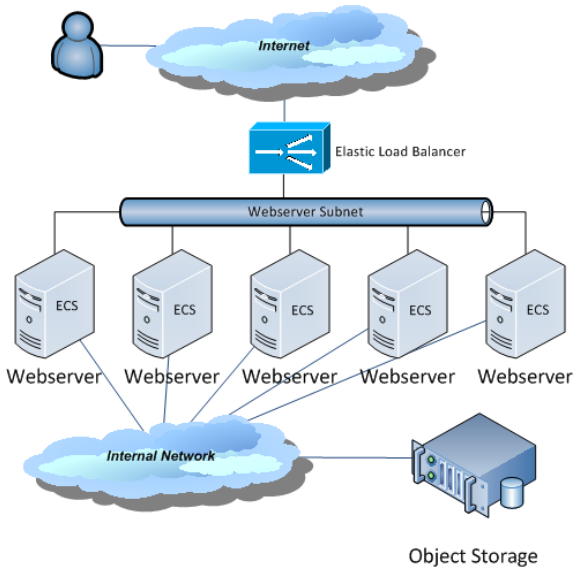
For the configuration, cloud-init is commonly used to bring configuration parameter during runtime to the ECS instance. If this is not sufficient also an archive with overlay files which will be injected into the ECS instance during deployment can be used.

To host just static website data, Object Storage is the perfect fit. By using Object Storage, the actual instance which is running the webserver is decoupled from the data that is being served. In addition, changes to the data on the Object Store are directly visible on all connected clients. Utilizing additional features on the Object Store like Versioning is bonus that is very hard to achieve by using conventional storage within a singular instance.

Finally, it is also possible to store the web site content on each host and fetch the latest version of the content during deployment of the ECS instance. This has the advantage that nearly no modifications need to be done to the local configuration but on the downside an update to the content results in a complete re-deployment of the web server infrastructure.

### 6.1.3 SAMPLE 1-TIER APPLICATION

A 1-tier web application is the easiest setup of a web server that will only deliver his own static content. First component - looking from the outside to the inside - is an Elastic Load Balancer with an Elastic IP assigned. This is the IP the web service will be reachable from the outside. The ELB needs Listeners on ports 80 and 443 assuming that standard HTTP and HTTPS should be delivered to the client.



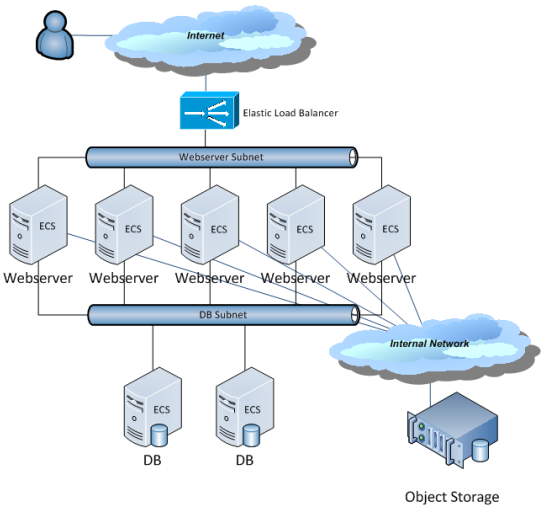

Second part are the webservers which deliver the content. This boils down to a number of ECS instances. For simplicity, we assume for the moment a static number of servers which are deployed. For dynamic setups see the next chapter on Auto-Scaling. This fixed number of ECS instances should be stateless so that on demand new instances can be easily deployed. The prerequisite for this is therefore a suitable image - most likely a private image derived from one of the official images with a small footprint and enriched with a web server instance. The configuration for the web server should come through cloud-init or with file injection during deployment. The configured instances should then be added as backend servers to the ELB and must have a security group applied that will allow incoming traffic on port 80 and 443.

Third and last part is the storage of the website content. As explained above putting this on Object Storage is a good possibility. The Object storage may be used like a local file system or only specific files may be linked into the website. If necessary adjust the ACL on the Object Storage to either allow public access or just access from the deployed webservers. If a more traditional approach is favored than the fetch of the latest content during deployment and local storage of it may be the weapon of choice.

### 6.1.4 SAMPLE 2-TIER APPLICATION

The 2-tier web application is composed basically with the same elements as the 1-tier application but behind the web layer a second database layer is connected, where the web server can store and retrieve dynamically generated data from database. Looking at the needed components from above this does not mean that the Object Storage is replaced by a database. In reality, this is more an "and" than an "xor".

Adding to the components from above, Object Storage would become something like "3-A" and the database would be "3-B". The database is then either a redundant database running on at least two hosts or is provided through the DBaaS offering RDS on Open Telekom Cloud.




## 6.2 INCREASING AVAILABILITY AND SCALABILITY

In the setup described above are all the building blocks for a cloud-native web application available yet it contains still several single points of failure or just lacks the dynamics a Public Cloud can offer. This can be mitigated by utilizing at least two geographically separated Availability Zones as well as the on-demand provisioning through Auto-Scaling.

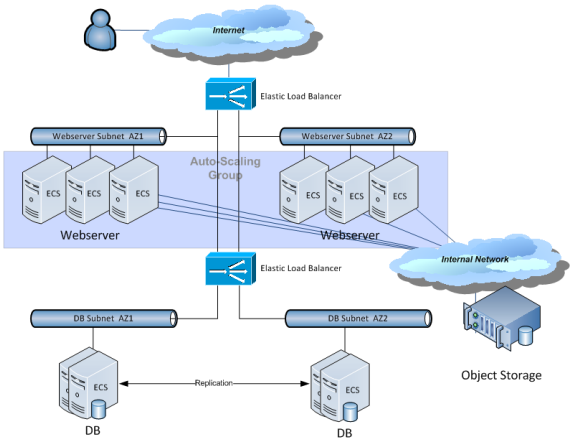
### 6.2.1 MULTIPLE AVAILABILITY ZONES

Deploying ECS instances only in one Availability Zone (AZ) is a single point of failure. The design of a cloud-native application should be aware that failure is common and be prepared for it rather than to avoid it. This does include the outage of a complete AZ and therefore spreading the instances of an application over at least two independent AZs is a mandatory prerequisite.

Per design a subnet may not span multiple AZs which means the minimum setup includes two subnets, each allocated in a different AZ. In this subnet, the web servers are placed and connected to the load balancer, which can connect to both.
This eliminates the single point of failure a single AZ would impose to the overall architecture.

### 6.2.2 AUTO-SCALING

To leverage all functions and possibilities of a public cloud also load-based dynamics need to be implemented. By implementing a static amount of ECS the application is neither protected against instance failure nor able to cope with changing load.




The Auto-Scaling Service is a self-managing system that acts between predefined upper and lower boundaries regarding the number of deployed instances based on predefined triggers.

Applying this mechanism to the use-case above means that the lower boundary should be at least "2" so that an instance in every AZ is available. The upper boundary is depending on the individual use case but should not be too limiting. The number of expected instances would also be at least 2 so that for the start both instances are available.

Second part is the definition of trigger points. At least one for scale-out and one for scale-in is needed. A typical trigger for scale-out is high CPU or Memory usage on the system, this would cause the deployment of new ECS instances. Depending on the individual setup it may be a good idea to set the number of ECS instances to be provisioned on scale out to a larger number than the number to be de-provisioned on scale-in. In other words, the cloud would quickly add more resources in case of high load and then slowly reduce in case of low load. For certain scenario, another trigger to add is a time-based scaling action to e.g. scale-out to a fixed number of instances at 8 a.m. to be prepared for the steep load increase once business hours start.

Third part is connecting this to the used load balancer so that newly deployed instances are automatically added to the pool of available backend servers for that specific load balancer or be removed vice-versa.

## 6.3 SAMPLE MIGRATION PLAN

This part will describe a general migration strategy from an on-premise web application to Open Telekom Cloud. This will only outline the general steps for a migration and needs to be adapted to the individual cases. All design patterns described above should be applied to the individual use case to maximize the benefits from the possibilities of a hyper-scale public cloud like Open Telekom Cloud.

### 6.3.1 ANALYSIS

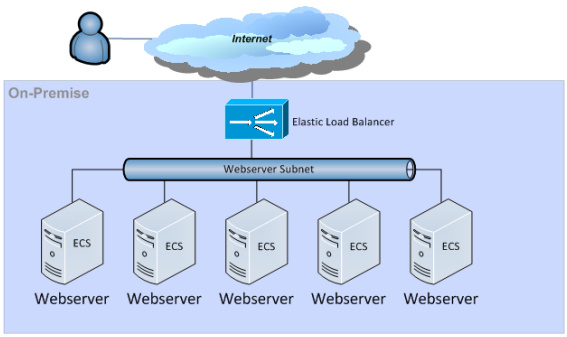
The existing application needs to be carefully examined, in how far it is already compliant to the criteria outlined above regarding a cloud-native application.

Mandatory

- Separation of application and content
- Use of load balancers

Optional

- Distribution of applications to different instances
- Use of different, geographically indecent location



### 6.3.2 PREPARATION

The two mandatory points from above need to be fulfilled to be prepared for the actual migration step in the hybrid phase.

The preparation on the cloud involves usually the setup of the network infrastructure, the creation of golden images that contain the needed applications and configurations and the preparation of a static environment with one or two instances.

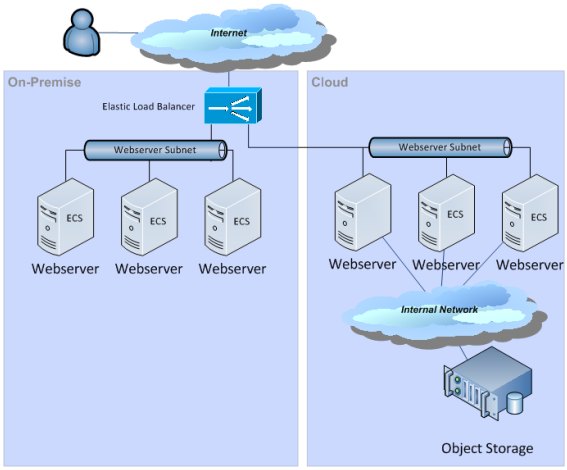
The needed data needs also be migrated to the cloud and suitable mechanism to keep the data in sync need to be put in place.

Functionality of the web application on Open Telekom Cloud as well as seamless provisioning of new instances needs then to be tested and verified. Once that is given, the next step is to bring both environments together.

### 6.3.3 HYBRID PHASE

In the beginning, it may be useful to start with a static configuration without additional Load Balancer on the cloud side to minimize the error sources. The available instance(s) on the cloud should then be integrated into an on-premise Load Balancer setup to act as additional resources together with the existing, on-premise servers.

Once it is verified, that the server(s) on the cloud are performing well, on-premise servers may be switched off in a ramp-down phase and load is successively moved over to the cloud. To ensure availably and performance, the usage of different AZs as well as the Auto-Scaling functionality is highly recommended to leverage the benefits of Open Telekom Cloud.



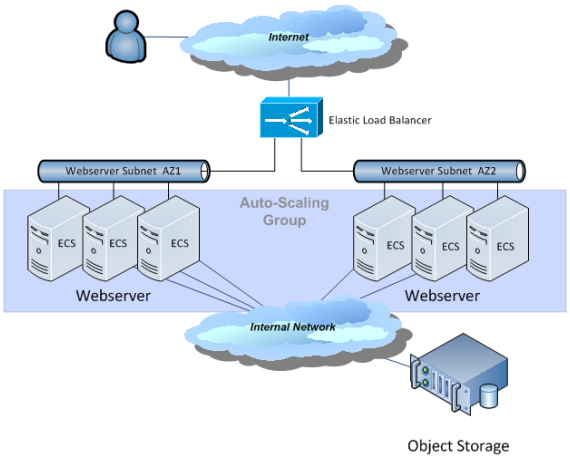

### 6.3.4 CUT-OVER TO OPEN TELEKOM CLOUD

Once the load is transferred to the Open Telekom Cloud, the DNS for the web application may be switched to the new public IP on Open Telekom Cloud and the on-premise Load Balancer can be switched off, once traffic is completely routed over the cloud instances.

## 6.4 CONCLUSION

To design and deploy a good cloud-native web application the use of the outlined design patterns is mandatory to ensure the desired availability on the outer edge of the application respectively the cloud. Following the paradigm of utilizing Auto-Scaling and Elastic Load Balancers the concepts can be easily adopted to suit multi-tiered applications.

As it is not always possible to start with a green field approach and existing servers should be migrated to the cloud, a step-by-step hybrid approach offers a smooth transition and the chance to adopt the new design models while moving into the cloud. Nevertheless, a tailoring of the method described is needed to reflect the individual use cases.



# 7 MIGRATING APPLICATIONS FROM OTHER SCALE-OUT CLOUD IAAS TO OTC

Applications developed on and for other scale-out clouds should already be prepared to deal with the dynamic nature of scale-out cloud environments. They are thus good candidates for a migration to Open Telekom Cloud – if the advantages (geography, jurisdiction, compliance, price, OpenStack APIs, ...) are relevant to the user.

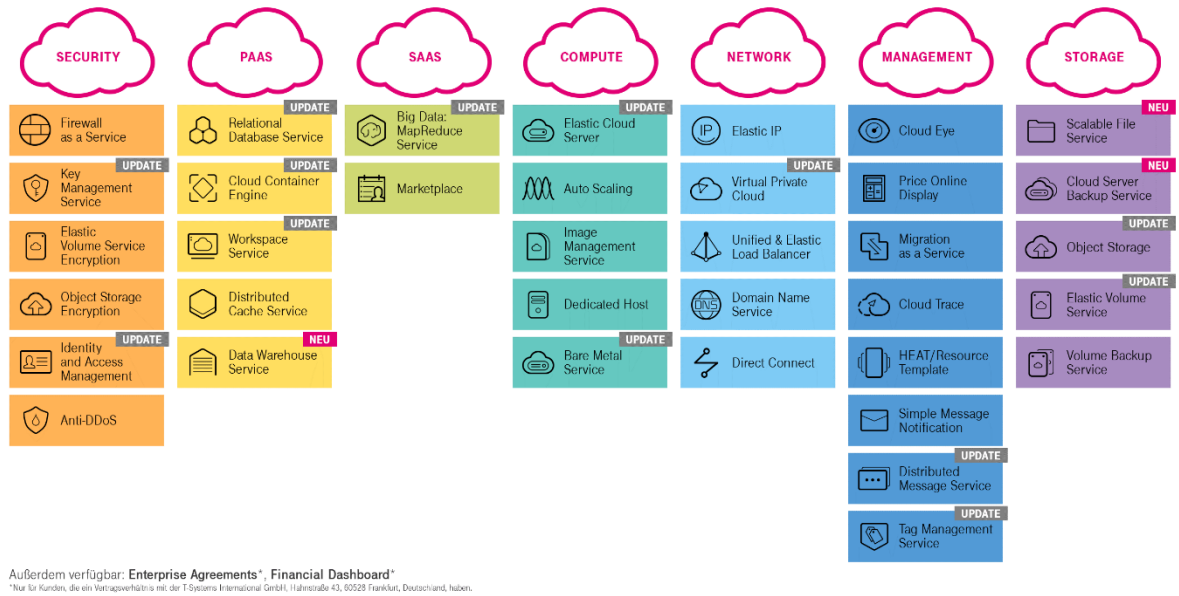
With the larger architectural challenges out of the way, a migration from another scale-out cloud to Open Telekom Cloud becomes a relatively straight-forward project that can be well planned and understood. A lot of the needed work will be on the details of configuring things to take optimal advantage of the underlying infrastructure. For those details, the description of VM migration in chapter 3 is a good reference.

This chapter will describe some of the general considerations and challenges that may be relevant for migrating an application to the Open Telekom Cloud.

## 7.1 SERVICES ON OTC

In the beginning, Open Telekom Cloud launched as a cloud focused on Infrastructure services. Higher level services (platform services or complete PaaS platforms as well as Software Services) were introduced in release 2.0 and have been further developed in the releases in November 2017 and March 2018. Still, Open Telekom Cloud will not attempt to rival AWS in the richness of platform services, but instead focus on the most used and best standardized components. The tradeoff between flexibility and convenience shifts the more exotic the platform services become.

The Open Telekom Cloud provides currently several IaaS, PaaS and SaaS services: The latest detailed descriptions of the services can be found at https://docs.otc.t-systems.com




The new release in March 2018 introduced two new services. The Data Ingestion Service (DIS) ensures quick availability of large data volumes in the cloud by streaming data automatically to the OTC. This can be configured to real-time or batch wise processing. The second new feature is the Document Database Service (DDS), where users can monitor and scale database entries. Furthermore, the service facilitates managing differently structured information in a single database.

## 7.2 APIS

The concepts behind AWS and OpenStack are similar. This does not only have the advantage of supporting applications with the same architecture, but even is displayed at API level. When OpenStack launched, some of the functionality was offered with AWS APIs in addition or sometimes even instead of the then still quickly evolving OpenStack APIs.
Most of the OpenStack APIs are rather mature these days and the AWS compatibility has been defocused. Still the similarity in concepts will make the porting of scripts using AWS API calls to scripts using OTC API calls straight forward. The Open Telekom Cloud command line tool (which uses some of the enhanced function beyond the pure OpenStack APIs) has been designed to resemble the AWS tools to ease migration.

## 7.3 DATA MIGRATION CHALLENGE

Depending on the amount of persistent data that is stored in the cloud, the migration of data can become a serious challenge. Determining the amount of data and assuming a few 100MBit/s bandwidth, a rough estimation on the time needed to copy data over can be done. Is a cut-over for one night possible?

With the July 2018 update, the Mobile Storage Solution has joined the OTC as a supporting offering. With this service, vast data volumes can be transferred to the OTC. Instead of copying data over networks, data carriers will be brought to the data center, enabling the transfer of data volumes in the high TB / PB range.

T-Systems is working on other solutions to support customers to resolve this challenge. There are plans to provide temporary high-bandwidth links (Bandwidth-as-a-Service). There are also concepts for using boxes with many (encrypted) disks to transport the data – like the snowball offering from AWS.

Ultimately, there may be possibilities to avoid the need to migrate all data at once, see below notes on Step-wise approaches.

## 7.4 MIGRATION PROCESS

The migration from other cloud providers to Open Telekom Cloud will have roughly this sequence:

- Analyze if any special service from other cloud providers is not (yet) available or very different in Open Telekom Cloud
- Determine the changes needed (if any) e.g. due to implementing a service based on open source tools and decide whether the effort is worth the advantages
- Create those services (if needed)
- Create the deployment automation for OpenStack (or use some libraries that allow you to abstract the differences away, so your automation works for both cloud types). During that step, you preferably base on the preloaded images in Open Telekom Cloud, so you don't have to deal with drivers yourself.
- Test the application with test data ...
- Do incremental data synchronization to work with real data?
- Decide on a flag time and switch over.

Any of these steps can be done with external advice, should the amount of in-house experience, skills or available time prove insufficient. T-Systems can offer advice on migration projects out of the T-Systems “Systems Integration” unit and thus offer end to end responsibility for the success of the project.

## 7.5 STEP-WISE APPROACHES

It is possible to migrate application landscapes one-by-one. Public clouds are well connected to the internet, communication through encrypted tunnels (VPN) can be used if the applications need to talk to each other while living in different clouds. Whether this scenario is desirable depends on the amount of data that needs to be transferred. Data traffic is expensive on AWS.

A possibility worth considering is providing new customers in the new cloud (Open Telekom Cloud), while old ones are lazily migrated over. This way, no single point in time where all migration is done needs to be defined. This will be done once the old application is close to empty.

Another possibility worth mentioning is that there may be components in the application landscape that don't fit the scale-out cloud model so well. They can be migrated to other platforms from T-Systems (such as vCloud or DCP) in the same datacenters, so a good connectivity and service can be achieved. This may be an attractive option for applications with large monolithic databases, e.g. from Oracle.

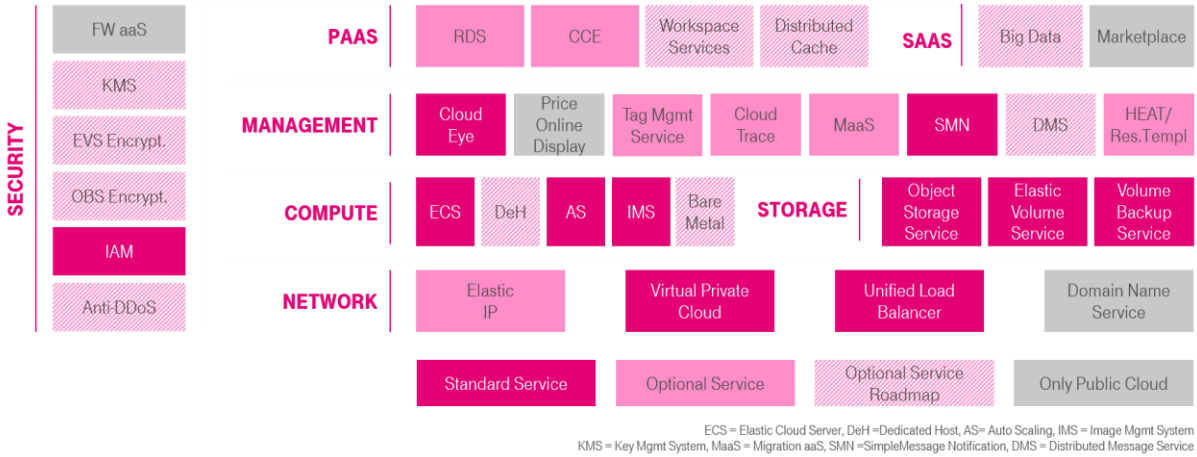
# 8 OTC HYBRID SOLUTION

Companies that want to leverage the advantages of public cloud but also need the additional security of a private cloud can now benefit from the Open Telekom Cloud Hybrid Solution. This innovative concept provides a consistent user experience and seamless integration between the private and public cloud.

This is achieved through establishing a private OTC on-premise for the customer. There are four configurations available:

- S Small size with approximately 170 to 1000 physical cores
- M Medium size configurations with approximately 700 to 4000 physical cores
- L Large size configurations with approximately 1400 to 8000 physical cores
- XL Scaling up is no problem

Each configuration has minimum requirements and can be scaled up in half-racks. The following services are available for the private OTC instances:




Per default, the private and public OTC are connected via the internet. Private Line Access Service via Ethernet Connect or IP VPN is available. The on-premise appliance is built on OTC software and hardware stack managed by TSI and the operations and processes follow established OTC public cloud standards. This provides flexibility to shift workload from private to public and back without reintegration of applications, which enables a unified DevOps environment.

# 9 DATA MIGRATION AS A SERVICE (MAAS)

A common challenge when moving applications is the transfer of the allocated real data. With MaaS, the Open Telekom Cloud provides a tool for moving data from object stores or content from relational databases to the Open Telekom Cloud. Data is encrypted and compressed prior to transfer. MaaS also provides a service that monitors the progress of the transmission and checks the data for consistency. A notification service informs users about the status and progress of the transfer.

**Migration Process**
Migration as a Service consists of three parts: Object Data Migration, Image Migration and Database Migration. This allows you to use data from object storage, server instances or VMs, and complete mySQL databases. Users start the migration service via the management console of the Open Telekom Cloud. The transfer is via https. Object Storage can be successfully transferred even when the session is broken or stopped; the transfer is automatically resumed at the breakpoint. Migration as a service also supports data compression, encryption, and consistency checks.

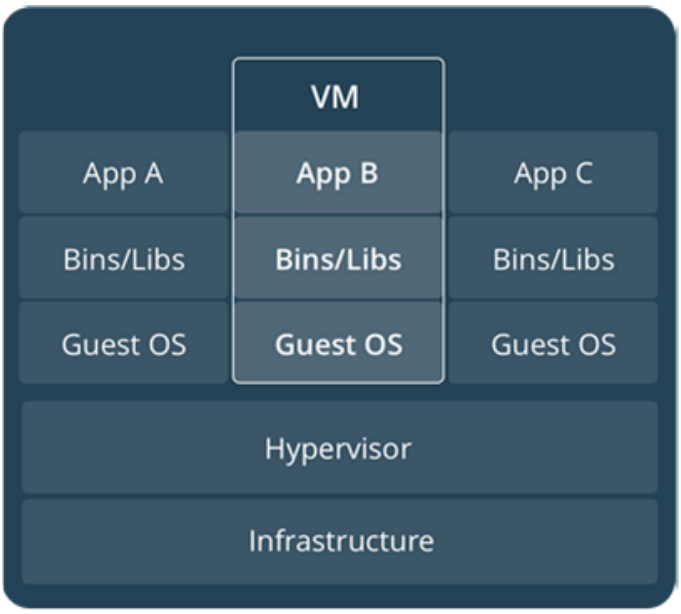
The Object Storage Migration Service is an automatic process. It is only necessary to use the MaaS console to create a migration task, that is, configure the data source and destination parameters, including the AK (access key) and SK (secret access key), name of the bucket where the source data is located, and the folders or files to be migrated in the bucket. After the configuration is complete, the migration tasks will start.

# 10 MIGRATING DOCKER CONTAINER FROM OTHER SCALE-OUT CLOUD IAAS TO OTC

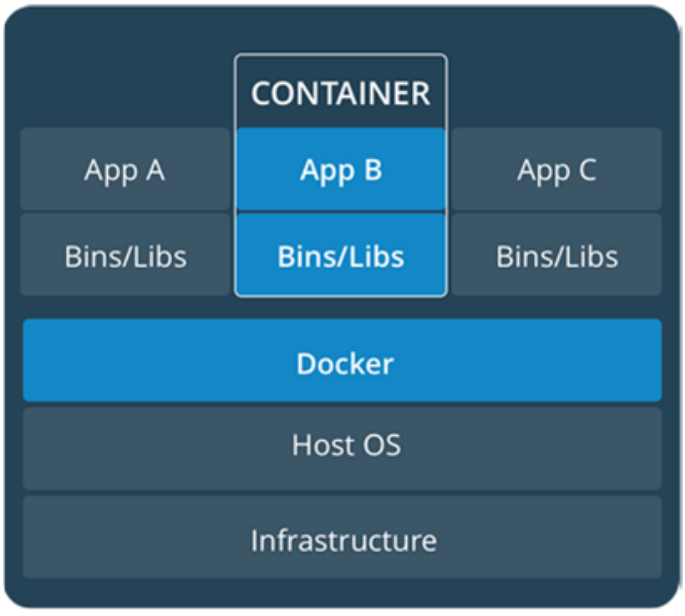
A Cloud native Application is based on some concepts which should be considered in the architecture:

- IP Addresses and Ports
- Virtual Machines
- Configuration
- Dependencies and Communication of Services
- Machine resources like CPU, Memory, RAM etc.

Container Technology like Docker will abstract from the IaaS Layer.




With the help of Docker, all the mentioned Concepts will be controlled centrally through a Controller, which will manage Containers within an IaaS Infrastructure.



Docker Containers on a node share a single kernel. Every dependency and information, needed by the App is stored within an container image. This image is described by a Docker File. It is executable on many Operation Systems without the need of changing anything.

The CCE (Cloud Container Engine) of the OTC is based on Docker. This is the reason why the takeover of a Docker App is very easy.

**Migration Process**
The migration from the source cloud service provider to Open Telekom Cloud will have roughly this sequence:

- Commit a copy of the container to the local cache
- Push the container to the docker private registry of OTC
- Copy the Volume
- Connect the Application Container to the Volume
- Start the application container

# 11 MIGRATING APPLICATIONS FROM DESKTOP TO WORKPLACE SERVICE OF OTC

Based on Workspace Flavors, Workplace Service creates cloud-based desktops. An access client is added to the virtual machine that delivers access to a virtual desktop based on the Windows 7 or Windows 10 design. Access to the desktop, including its data, is possible via any end-device – even mobile. Depending on the Workspace Flavor, it is possible to provision more than just standard desktops; even graphically demanding applications can be run on the Windows 7 or Windows 10 desktop by using a vGPU.

**Migration Process**

Workspace provides the same configurations (including vCPUs, GPUs, memory, and disks) as those of conventional desktops and Windows OSs.

A software client (SC) runs on PCs running Windows or Mac OS and is used for logging in to and accessing Workspace.

For a migration to Cloud Workplace Service these steps are mostly essential:

- Applying for Workspace Service
- Creating Workspace Desktops
- Managing Policies
- Check/setup licensing
- Configuring the WSUS (Optional)
- Install Applications on the Workplace Desktop

# 12 MIGRATING RELATIONAL DATABASE SERVICES FROM A SCALE-OUT CLOUD TO OTC

Open Telekom Cloud's Relational Database Service (RDS) is a tailor-made database service. RDS offers ten Flavors running the relational database software mySQL 5.6.x, PostgreSQL 9.5 or MS SQL (2014). RDS features a Monitor Service which can cover over 20 parameters such as capacity utilization of the database or the speed of running scripts. It also helps optimize the performance of the database.

Furthermore, RDS offers automatic backups and Point in Time Recovery stretching back the last 35 days. The database is scalable (up and down) in relation to the resources used (CPU and memory). RDS supports high availability. A standby database can mirror the primary database and up to five read replicas can also be added to the database cluster. Read replicas ensure increased security and speed when data needs to be read, for example, when users read an article in an online shop. Encryption functionality is also included. Currently, you can only access RDS DB instances from an ECS in the same Virtual Private Cloud (VPC) in an internal network.

**Migration Process**

Data cannot be exported or imported between heterogeneous databases due to different data formats. However, if the data formats are compatible, you can import table data to MySQL or PostgreSQL from Oracle or SQL Server theoretically. Generally, third-party software is required for data replication to export and import between heterogeneous databases. For example, use a third-party tool to export table records from Oracle or SQL server in .txt format and then use Load statements to import the exported table records to MySQL or PostgreSQL.

Before the migration of homogeneous database, stop any application using the source database. On the source database, you need a dump tool to save the data. Transfer the dump to the ECS. Use a dump tool to import the data in the RDS.

# 13 FURTHER READING

- https://d0.awsstatic.com/whitepapers/cloud-migration-main.pdf
- https://d36cz9buwru1tt.cloudfront.net/CloudMigration-scenario-backend-processing.pdf
- https://d36cz9buwru1tt.cloudfront.net/CloudMigration-scenario-wep-app.pdf
- https://d36cz9buwru1tt.cloudfront.net/CloudMigration-scenario-batch-apps.pdf
- http://arxiv.org/ftp/arxiv/papers/1002/1002.3492.pdf
- https://cloud.telekom.de/en/infrastructure/open-telekom-cloud/specifications/#navigation-product-subnavi
- https://docs.otc.t-systems.com/

The **order portal** for Open Telekom Cloud can be found at
https://cloud.telekom.de/infrastruktur/open-telekom-cloud/

# 14 TERMS AND ABBREVIATIONS

| Term/Abbreviation | Explanation |
|---|---|
| API | Application Programming Interface |
| AS | Auto Scaling |
| AWS | Amazon Web Services |
| AZ | Availability Zone |
| BYO | Bring Your Own |
| BYOL | Bring Your Own License |
| CCE | Cloud Container Engine |
| CI/CD | Continuous Integration / Continuous Deployment |
| CLI | Command Line Interface |
| DBaaS | Data Base as a Service |
| DCP | Dynamic Computing Platform |
| DDOS | Distributed Denial of Service |
| DeH | Dedicated Host, a host that is dedicated to one customer |
| DevOps | Clipped compound of “Development” and “Operations” |
| DMZ | Demilitarized Zone |
| DNS | Domain Name Service, Domain names are translated to IP-Addresses |
| ECS | Elastic Cloud Service |
| EIP | Elastic IP |
| ELB | Elastic Load Balancer |
| EVS | Elastic Volume Service |
| HEAT | Heat: a service to orchestrate multiple composite cloud applications using templates |
| HOT | HEAT Orchestration Template |
| IaaS | Infrastructure as a Service |
| IMS | Image Management Service |
| JSON | JavaScript Object Notation |
| KMS | Key Management System |
| MaaS | Migration as a Service |
| MPLS | Multiprotocol Label Switching |
| OBS | Object Storage Service |
| OS | Operating System |
| OTC | Open Telekom Cloud |
| RDS | Relational Database Service |
| REST | REpresentational State Transfer |
| SAS | Serial Attached SCSI |
| SATA | Serial AT Attachment |
| SLA | Service Level Agreement |
| SQL | Structured Query Language |
| SSD | Solid State Disk |
| VBS | Volume Backup Service |
| VM | Virtual Machine |
| VPC | Virtual Private Cloud |
| VPN | Virtual Private Network |
| XEN | A hypervisor, which was developed under the GPL license |
| YAML | YAML Ain't Markup Language |