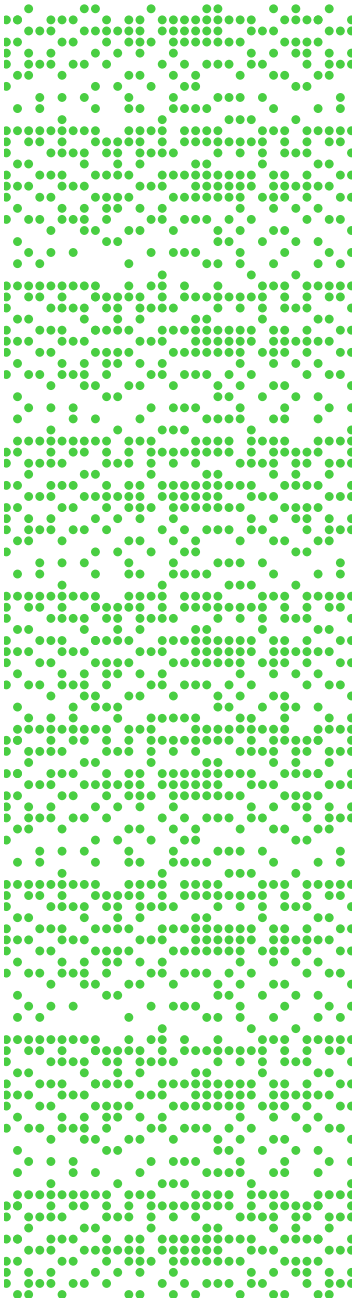
routz

EXCEEDING NETWORK AMBITIONS



WHITE PAPER
AUTOMATION AND ORCHESTRATION

(VISION TO DATE NOVEMBER 2019)

ROUTZ.NL

INDEX

PREFACE

Automation and orchestration help businesses …
Businesses require highly available, scalable and fast provisioning of services. IT organizations should therefore provide services on customer demand, and the services they provide should automatically adapt to changes and faults in the infrastructure. These requirements can be met by automating the IT processes for service provisioning and adaption.

… but they are not easily implemented
At Routz, we want to help organizations on their journey to automated provisioning and adaption of their services. With this white paper we share our knowledge and experience regarding automation and orchestration.

Familiarize yourself with Routz' vision …
This white paper describes Routz' vision on automation and orchestration. It outlines an architecture for automation and orchestration that facilitates the automated provisioning of on-demand services and the automatic adaption of these services. It shows the 'why', the 'what' and the 'how' of automation and orchestration tooling.

… and get a head start with our best practices
In this white paper, we also provide guidelines for the deployment of an automated IT infrastructure. We focus on network infrastructures, but will also refer, where relevant, to computing and storage infrastructures. The contents of this paper are based on both desktop research and the hands-on experience we acquired during customer projects.

1

INTRODUCTION

INTRODUCTION

The challenge: keeping up with business demands

One of the biggest challenges IT organizations face nowadays is keeping their systems and processes agile enough to be able to keep up with the rapidly changing needs of the business. The business requires fast provisioning of services on customer demand. It also requires services that automatically adapt to changes and faults in the IT infrastructure. In short: the business requires highly available, scalable and fast provisioning of services. As a result, the business pushes more application workloads to public clouds, which are more agile but also more costly in the long run. Another downside is that a lot of these services, especially end-to-end services, are not cloud-ready and cannot be reached from the public cloud. To address these issues, more and more organizations opt for building their own private agile IT infrastructures.

The answer: network functions virtualization ...

IT organizations are moving from infrastructures based on bare metal devices to network functions virtualization (NFV) infrastructures based on NFV devices. This facilitates easier and faster delivery of services. Services can be delivered on demand - within hours instead of weeks. With the use of automation and orchestration tooling this time can even be shortened to minutes. Automation and orchestration tooling bypasses human intervention and automates the entire process for service provisioning. Customers can start the service provisioning by selecting the service on the service provider's portal. Within minutes, the service will automatically be provisioned and added to the Operating Support System (OSS) and Business Support System (BSS) environments (e.g. for monitoring or billing purposes).

... and automation and orchestration

The definition of automation is setting up a single task to run on its own - automating one thing. This single task can be anything from launching a web server, stopping or starting a service, to configuring a VLAN. Orchestration utilizes multiple automated tasks to automatically execute a larger workflow or process. These workflows and processes can consist of multiple automated tasks and can involve multiple systems.

An example: Figure 1 shows a service request initiated on the customer's portal. A service request can also be initiated by another customer application (such as an OSS or BSS application). The service request is initiated by a customer selecting the required service. The orchestration process will translate this service request to a larger workflow or process consisting of several individual automation tasks. The automated process will execute these individual tasks.

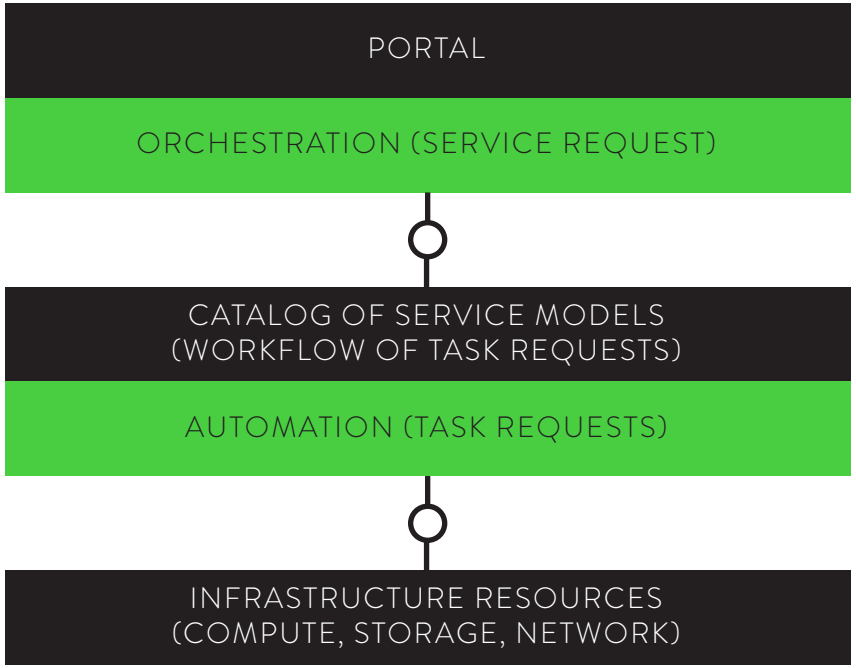


Figure 1: overview customer service request

Automation as a subset of orchestration

Automation can be seen as a subset of orchestration. The table below shows some characteristics of automation and orchestration.

- What to do
- Perform a task without manual intervention
- Eliminate human intervention
- Reduce IT costs
- Automate anything - deployment, configuration, tests, and so on
- Focus is on how to make devices do task-oriented 'human work'
- Improve repetitive work, with high confidence in the outcome

AUTOMATION

- When to do
- Coordinate, organize, or builda process or workflow that encompasses multiple automated tasks
- Deliver new features faster
- Enable and empower continuous integration and continuous delivery
- Brings together or integrates different technologies and tools
- Provides the ability to coordinate informed decision making, formalize and automate responsive actions

ORCHESTRATION

Automation offers the following advantages compared to traditional methods of management:

- Improved efficiency
  By automating functions of infrastructure devices, humans are no longer required to perform time-consuming tasks.

- Reduced likelihood of human error
  Manually performed tasks are prone to human error, and a miscalculation or incorrect entry can have significant consequences for the stability of the IT infrastructure. Setting up a task for automation means it only needs to be entered correctly once.

- Lower operational costs
  This benefit ensues from the previous two items. By eliminating certain manual tasks related to device provisioning and management, businesses can operate with greater speed and agility. For example, automated provisioning might save a network engineer from having to travel to a new branch office to establish network connectivity -- thus enabling employees at that site to commence work faster.

Orchestration offers the following advantages compared to traditional methods of management and to automation:

- Faster introduction of services
  New services can be introduced relatively fast, as they can be automated into one workflow without having to separately configure multiple infrastructure devices with the required configuration for the requested service (for example: VLANs, interfaces, routing).

- Easier troubleshooting, more consistent configuration
  The configuration can be standardized across the IT infrastructure based on pre-defined configuration templates. This simplifies troubleshooting and provides a consistent overall configuration.

- No vendor lock-in
  Using market or international standards eliminates the risk of vendor lock-in.

- Central point of control for compliancy
  Compliancy can be achieved through the implementation of a central point of control where the condition of the IT infrastructure is audited by the authorized bodies.

- Programmability
  Programming will eliminate human intervention and human errors and will increase availability. The infrastructure can be programmed through an open Application Program Interface (API).

2
ARCHITECTURE

ARCHITECTURE

In this chapter we describe the Routz architecture for automation and orchestration, and give our general advice for creating your own architecture.

Architectures encompass several domains …

An architecture for orchestration encompasses several technology domains and consists mainly of a dedicated controller that manages the infrastructure components, like compute, storage and network devices, of each domain. Figure 2 shows a high-level overview of the automation and orchestration architecture as recommended by Routz. It clearly shows the orchestration layer, which provides the orchestration workflows and automation tasks for service provisioning. It also shows the application layer providing the applications that initiate the required service requests.

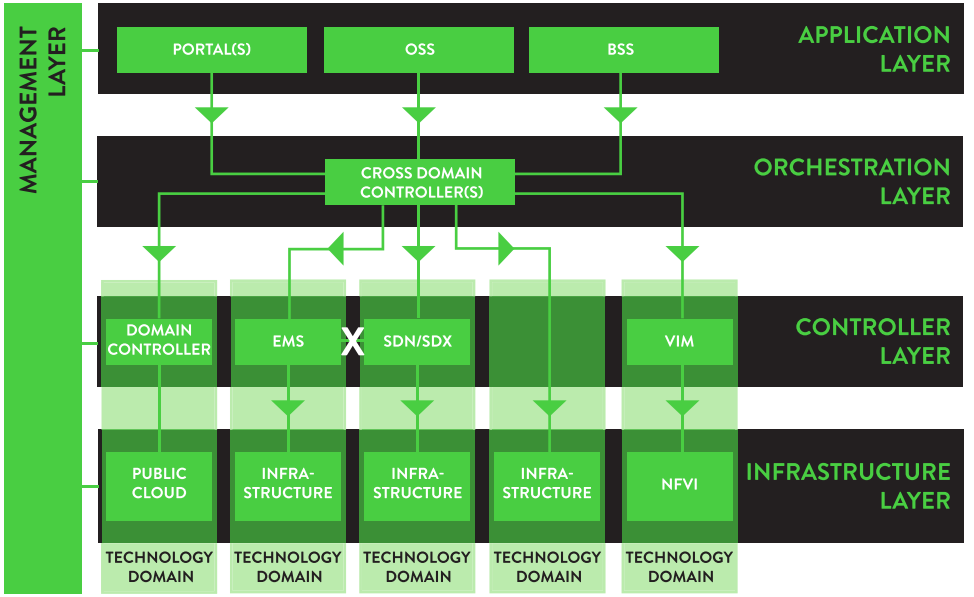


Figure 2: Routz architecture for automation and orchestration

… that each consist of two layers

Each of the technology domains consist of an infrastructure layer and a controller layer. The infrastructure layer consists of compute, storage, or network devices, or a combination thereof. The controller layer consists of

a controller device, a management system or EMS, or a software-defined controller, used for central management of the infrastructure components of the domain. The controller device has a northbound interface which enables access by management and orchestration tooling. The controller device has a southbound interface for accessing the infrastructure devices of the technology domain.

It is possible that a technology domain does not support a controller in the controller layer, for example an infrastructure layer based on a standard classic network infrastructure. In that case, the orchestration tooling should provide the controller function and should directly access the devices of the infrastructure layer. In short: the architecture in figure 2 supports the automation and orchestration of any technology domain – but in some cases, this requires self-designed orchestration tooling.

Domain types

The architecture shows examples of technology domains. IT organizations can facilitate fewer or more of these domains. The most common ones are:

- The NFVI domain
  NFV defines standards for compute, storage, and networking resources that are used to build virtualized network functions. NFV Infrastructure (NFVI) is a key component of the NFV architecture that describes the hardware and software components on which virtual networks are built. The NFVI domain has its own virtual infrastructure manager (VIM).

  NFVI leverages the economies of scale of the IT industry. It is based on widely available and low-cost, standardized computing components. The NFVI works with servers – virtual, bare metal, or other – and the software, hypervisors, virtual machines, containers, and virtual infrastructure managers to thus enable the physical and virtual layer of the network to function. An IT organization can consist of more than one NFVI domain (e.g. vCenter, OpenStack).

- The data center domain
  This is a network infrastructure that consists of an overlay network (e.g. NSX, Contrail, Nuage) combined with an underlay network, or of only an underlay network. A software-defined network (SDN) controller is part of the overlay network.
  The underlay network could be a fabric (e.g. Cisco ACI, Arista, Juniper) or a classic network deployment (e.g. fabric-patch, loop-free layer-2). An SDN controller should be part of the fabric network. Because of the complex configuration for the day-1 deployment of the fabric, Routz advises to use an SDN controller for this task. The management of the classic data center infrastructure is performed by a central management system. An IT organization can facilitate more than one data center domain.

- The WAN domain
  This domain is used to extend the private customer network to other locations. WAN infrastructures can consist of multiple data planes and can use an SD-WAN controller for configuration and orchestration purposes.

- The firewall domain
  The firewall domain consists of multiple firewall devices. These are managed by a dedicated element manager system (EMS).

- The load balancer domain
  This domain consists of multiple load balancing devices. The load balancer devices are managed by a dedicated EMS.

- The public cloud domain
  Provides the ISP cloud services. By accessing the cloud domain controller, customers should be able to automatically provision services in the public cloud.

- The campus domain
  Consists of a network infrastructure that facilitates network connectivity for wired end user devices. The controller layer mostly consists of a central management system. Some vendors support an SDN controller for the campus domain.

- The WLAN domain
  Consists of a network infrastructure that facilitates network connectivity for wireless end user devices. The controller layer mainly consists of a dedicated EMS.

Our best practices

Avoid integrated deployments of controller devices ...

We often see integrated deployments of controller devices belonging to different technology domains (see the white cross in figure 2). This is mostly done to assign all the network configuration to a network engineer. For example, the security engineer is responsible for the security domain. However, the configuration of the firewalls of the security domain requires network configuration. This can be done through the SDN controller of the data center domain, if the EMS controller of the security domain is integrated with that SDN controller. But even then, the security policies still need to be set by the security engineer through the EMS controller. This means both controllers are still needed for configuration of the firewall service. Integration of controllers will result in dependencies, due to life cycle management (LCM) and software updates.

... instead, deploy cross-domain controllers

Our advice is to deploy a cross-domain controller(s) and use orchestration workflows for cross-domain configurations. The cross-domain controller provides an abstraction layer and facilitates automation and orchestration tooling covering multiple technology domains. The cross-domain controllers facilitate automatic provisioning of services, rendering network specialists and security specialists redundant for this provisioning.

Create a service catalog for standard service deployment

With regard to automation and orchestration, Routz recommends creating a service catalog that specifies the standard service deployment models. The service catalog accommodates centralized service management, end-to-end service visibility, standardization and clear service offering. It also reduces IT service delivery time and costs, and increases user satisfaction.
A standard service model, as shown in figure 3, determines the orchestration workflow tasks for provisioning and the technology domains to be involved. Organizations that want to start with service deployment should first of all specify the use cases for automated service provisioning. Next, they should define the service deployment models based on these cases. These deployment models determine the required compute, storage and network functions for each of the technology domains and determine the capabilities for each of these functions.

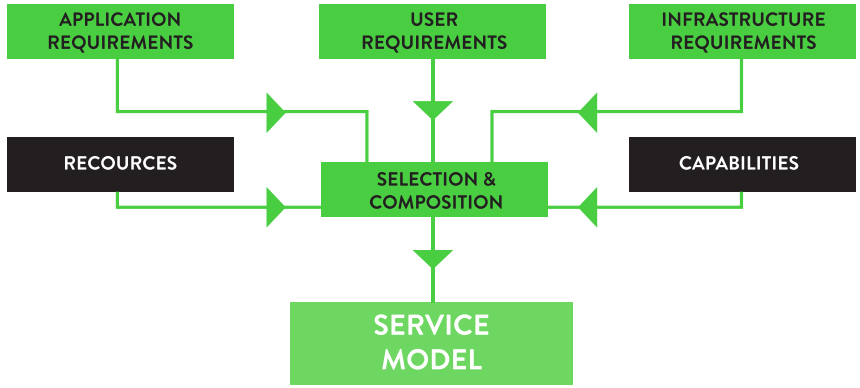


Figure 3: creation of a service model

3

DEPLOYMENT

DEPLOYMENT

However useful they may be, automation and orchestration are not the answer to every problem. This chapter provides guidelines for deciding when to use them, and for how to approach their use.

Decision tree for automation and orchestration

Figure 4 is Routz' decision tree for automation and orchestration. It applies to new service demands that are not yet operational within the network infrastructure, and to existing services that already are operational.

Use cases and suitability

The decision tree shows three main use cases regarding a service demand for automation and orchestration. The service demand should hit at least one of these use cases. If not, the use case is not suitable for automation and orchestration.

- Auto-healing use case – based on automatic recovery (no manual intervention) of faults in the network infrastructure. OSS and BSS will continuously gather monitoring and logging information and will automatically act on faults and malfunctions of the network infrastructure. OSS and BSS use analysis methods like artificial intelligence and machine learning for the automated recovery process;
- Auto-scaling use case – based on automatic recovery (no manual intervention) of resource issues in the network infrastructure. OSS and BSS will continuously gather monitoring and logging information and will automatically act on resource issues of the network infrastructure. OSS and BSS use analysis methods like artificial intelligence and machine learning for the automated resource scaling process;
- Auto-provisioning use case – based on repeatable provisioning of work flows.

Benefits and suitability

The decision tree also shows benefits of automation and orchestration. The service demand should hit at least one of these benefits to be suitable for automation and orchestration.

- Increase availability
  High availability is a main principle for a network infrastructure. Availability can be increased by eliminating certain manual tasks.

- Increase customer experience
  Satisfactory customer experience is dependent on the prevention of disruption in service and/or outages. Most of the auto-healing and auto-scaling functions will improve customer experience.
- Reduce time to deliver
  Businesses nowadays require fast, on-demand provisioning of services. By automating functions on network devices, humans no longer have to perform time-consuming tasks.
- Reduce costs
  OPEX and CAPEX saving is the main business driver.

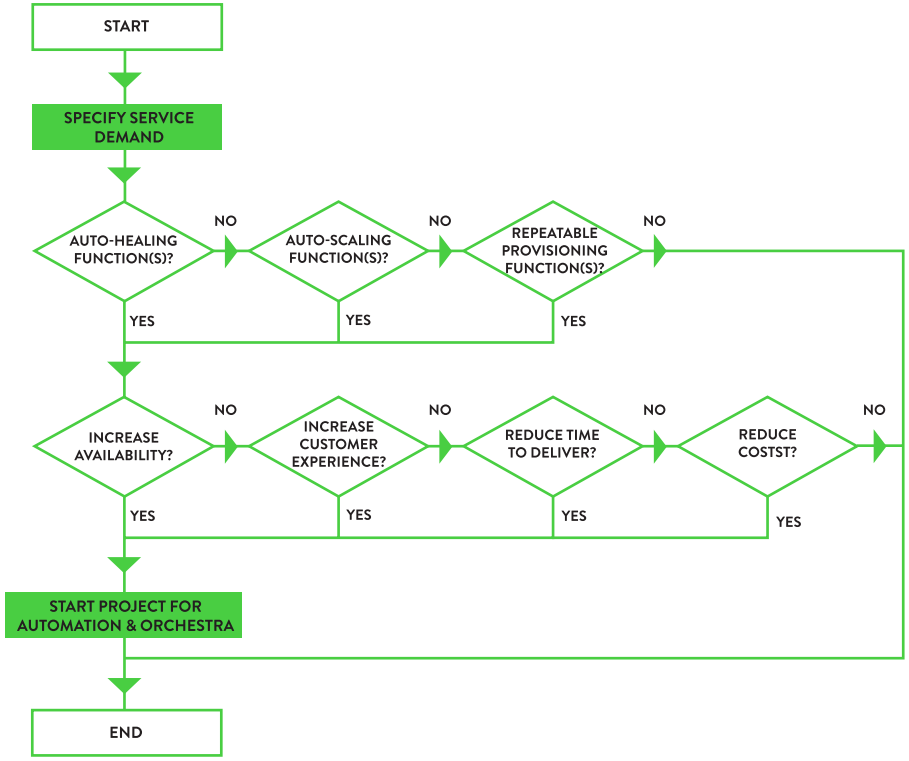


Figure 4: decision tree for automation and orchestration

Service demands to service model

If a service demand hits one of the use cases and one of the benefits, a project should be started to cover the automation and orchestration of that service. Service demands can be auto-healing or auto-scaling, both resulting in a configuration adjustment of the capabilities of a service model. A service demand can also be an auto-provisioning demand:

- A demand for an end-to-end service, consisting of the provisioning of several consecutive service models;
- A demand for the provisioning of one service model;
- A demand for one configuration task.

Figure 5 shows a high-level flow chart for creating a service model and a high-level flow chart for creating an automation task. These charts are based on an existing network infrastructure, consisting of one or more technology domains. If you want to create an automating task, you should be familiar with the detailed features of the technology domain. Only then can you adequately specify the configurations required for deployment of the technology domain, and select an individual configuration task to be automated.

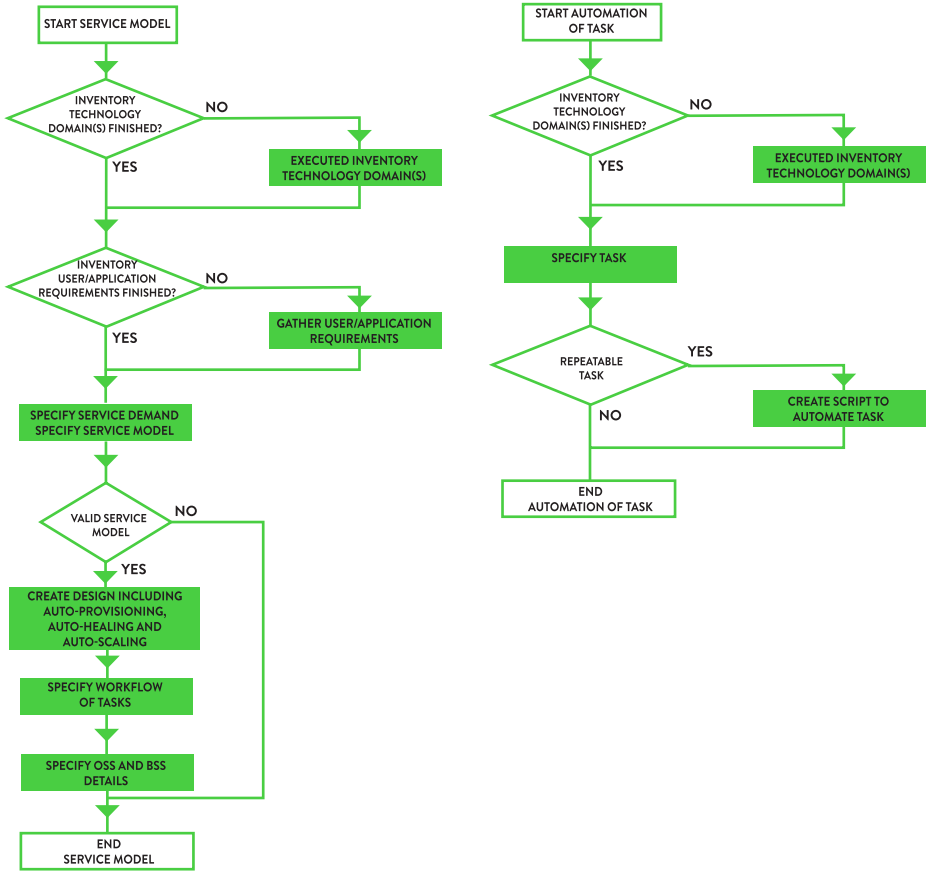


Figure 5: creation of service model and automation task

Requirements for service models

If you want to create a service model, you should be familiar with the detailed features of the technology domains and with the user and application requirements. Once you are, you should be able to specify a service demand and service model. Next, check the decision flow chart (figure 4) to decide whether the service model is suitable for automation and orchestration. If it is, some actions must be undertaken in order to specify the service model in greater detail. These actions are shown in the flow chart of figure 5 and represent the 'start project' as shown in figure 4.

Deliverables of service models

Creating the service model results in the following deliverables:

- A workflow of configuration tasks for auto-provisioning of the service model. Each of the configurations tasks of the workflow must be translated into a configuration script.
- A configuration task for adjustment of capabilities, for auto-healing and auto-scaling of the service model.
- The application requirements for the OSS and BSS regarding auto-healing and auto-scaling. These requirements are mainly based on the analyses of the monitoring and logging information.

Next step: orchestration tooling

The service models and automation tasks provide the requirements for the orchestration tooling that is to be used for automation and orchestration. Selecting the orchestration tooling that is to be used is challenging for IT organizations. Nowadays, many vendors are working on the development of their orchestration tooling. As yet, there is not one single orchestration tool that meets the requirements for cross domain orchestration, in the sense that it supports all, or even a significant subset of technology domains. It can even be hard to just find an orchestration tool that supports specific customer requirements for a single technology domain.

A choice has to be made. There are essentially four options:

- select a vendor orchestration tool that does not yet provide all the required features, but does provide some or most of the required features,
- start orchestration with existing tooling (OSS and BSS and existing controller systems of the controller layer),
- use a combination of both,
- start developing your own orchestration tooling.

Road map to the selection of tooling

This is the proposed road map for selecting automation and orchestration tooling:

1. Create scripts for repeatable tasks (right flow chart of figure 5).
2. Create service models consisting of a workflow of repeatable tasks (left flow chart of figure 5).
   Our advice: start small. Not with an end-to-end service model covering multiple technology domains, but with a simple service model that covers only one technology domain.
3. Create a process for auto-provisioning of the service model by using existing tooling and start a Proof of Concept (PoC) for this process.
4. Create a process for auto-healing and auto-scaling of the service model by using existing tooling and start a PoC for this process.
5. Carry out continuous vendor research, market investigation, for orchestration tooling. Confirm the features of the vendor orchestration tooling with the experience you gained with earlier road map actions.
6. Start the RFI/RFP phase for orchestration tooling. The information, outcomes and experiences of the previous actions will result in an RFI/RFP phase for selecting the orchestration tooling.

4

PRINCIPLES

PRINCIPLES

In this chapter we describe the seven design principles as recommended by Routz, specifically regarding automation and orchestration.

Principle 1: think big, start small

The basis for this principle lies in the current availability of vendor orchestration tooling. Several vendors are working hard to develop a mature orchestration tool that covers most of the technology domains and facilitates orchestration for end-to-end service provisioning. However, such a tool is not available at this time.

That is why our advice is to still think big, to still aim for end-to-end provisioning, but to start small, with orchestration for one domain. That way, you increase your chance of success and minimize the costs in case of failure. And either way, you will draw valuable lessons from the experience.

Principle 2: avoid integration

This principle states that the integration of (software-defined) controllers for different technical domains should always be avoided. This will prevent dependencies between controllers and between technical domains. Changes like LCM and software updates of the devices or controllers of one domain could have a major impact on the interaction with the controller of another domain. In the worst case scenario, it would kill the interaction. Furthermore, integration leads to more complex situations, which are harder to understand and are more likely to lead to errors.

Principle 3: provide provisioning at the edge of the network

Provisioning should take place at the edge of the network. This prevents any configuration in the core of the network and creates a simpler situation for automation and orchestration.

This principle is based on the Software-Defined Network (SDN) technology, which consists of a switched infrastructure with a centralized SDN controller providing the forwarding instructions. This SDN controller facilitates automation and orchestration tooling and is based on programmability. Programmability works best in software, and because the x86 platforms are

becoming more powerful, network intelligence tends to be provisioned on the x86 platform these days, meaning at the edge of the data center network.

This principle works best for the NFVI domain. However, it can also be used for other domains:

- For an MPLS network, most of the intelligence is facilitated on the PE routers, e.g. provisioning for L2VPNs and L3VPNs is achieved on these routers;
- For a campus network, the intelligence for network access is facilitated on the access switches at the edge of the network.

Principle 4: use standardized service deployment models
Service models should be based on pre-defined configuration templates. This simplifies troubleshooting and provides a consistent overall configuration. The standard service models should be based on market standards or international standards, in order to remove the risk of vendor lock-in.

Principle 5: aim for full automation
Service models should be fully automated, and manual configuration should be avoided. Manual configuration is time-consuming and increases the risk of human errors. Therefore, all the individual workflows of a service model should support automated provisioning.

Principle 6: create modularity in programming
Scripts and programs for the provisioning of service models should be modular and standardized. That way, they can be used for programming other and future service models. In other words: do not create a program for each individual service model. Instead, create re-usable modular scripts that can be shared to program multiple service models.

Principle 7: provide a trusted source
Orchestration is dependent on a service catalog that specifies the service models. The configuration parameters of a service model are specified in a database (CMDB). This database must be accurate and up to date and should only be adjusted by an authorized service.

Our advice is to start with a greenfield database when starting the auto-provisioning process, and to only allow orchestration tooling to update the database. The result will be a trusted source that does not contain any polluted information.

5
USE CASES

USE CASES

With this chapter we want to help you specify your service models, by showing examples of use cases for automation and orchestration. We cover the tenant service model use case, and a number of more general use cases.

The tenant service model use case

Suitable for public and private cloud

These days, IT organizations follow the principle of 'cloud, unless' - meaning that the preferred provisioning of their services takes place through the cloud, either a public or private one. In order to do this, they require the flexibility to move resources from the public to the private cloud and vice versa. To facilitate a simple moving process, a similar service model should be supported for both the public and private cloud.

This use case is based on the service model of a public cloud provider.

Figure 6 shows a tenant service model that can be used for the private cloud (and for a public cloud as well, with some minor changes regarding implementation of individual functions). This service model is based on the virtual private cloud (vPC) service model of service providers. It provides an isolated network infrastructure (a tenant) for a specific customer. The service model can be duplicated for other customers. It can be useful, possibly with some minor adjustments, for automatic deployments by IT organizations.

Aggregation and security

The tenant service model is connected with the corporate network. The aggregation function is covered by a virtual router. This router can provide the security policies based on access control lists (ACLs). The router can, when required, provide network address translation (NAT). NAT facilitates overlapping IP network ranges for all the tenants. If higher security demands are required, the virtual router can be replaced by a virtual firewall.

The tenant service model consists of one or more network zones. A network zone can be assigned to a specific customer function, and each network zone contains several applications. These can be protected by micro segmentation (e.g. by host-based ACLs or by security groups).

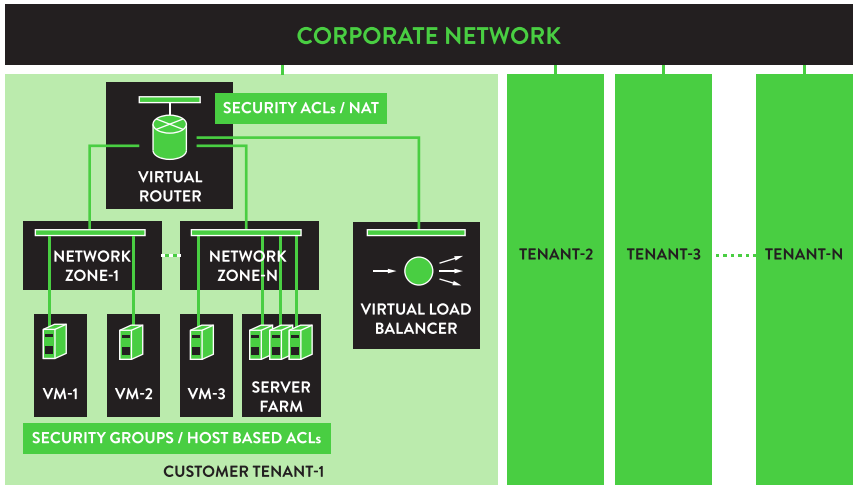


Figure 6: service model customer tenant

The tenant service model shown in figure 6 supports server farm provisioning. A virtual load balancer is logically connected with the tenant and supports server farm services for all the network zones. The model can be provisioned as a whole, but each of the services can also be provisioned individually. If a customer wants to provision a server farm, orchestration should check whether or not the required server farm is already in place and whether or not orchestration should provision these services.

The tenant service model consists of a workflow of individual automation tasks. The table below shows examples of these individual automation tasks.

AUTOMATION TASK	DESCRIPTION
Virtual router provisioning	Deployment of a virtual router instance. It depends on the customer infrastructure whether this is a virtual router instance of the overlay network or a virtual router of the underlay network.
Interconnect VLAN provisioning	Deployment of an interconnect VLAN for logical connectivity with the corporate network. It depends on the customer infrastructure whether this will be a configuration in the underlay or overlay network or in both.
Network zone VLAN provisioning, including automatic VLAN assignment	Deployment of the VLAN for a network zone. It depends on the customer infrastructure whether this will be a configuration in the underlay or overlay network or in both. Automatic VLAN assignment requires integration with VLAN database application (e.g. IPAM).
Subnetwork provisioning, including automatic IP assignment	Providing the IP configuration for the VLANs. Automatic IP assignment requires integration with IPAM.
Virtual load balancer provisioning	Deployment of a virtual load balancer instance. It depends on the customer infrastructure whether this will be a virtual context on a load balancer appliance or a virtual load balancer in the overlay network. In the latter case, this would result in the execution of an orchestration task instead of an automation task, because there is a workflow of automation tasks that must be executed (like VM, load balancer application, subnet configuration).
Virtual machine provisioning	Deployment of a VM including IP connectivity. For the VM provisioning, there is a choice of flavors (e.g. memory capacity, storage capacity, number of CPUs).

AUTOMATION TASK	DESCRIPTION
Server farm provisioning	Deployment of a server farm based on a configuration of the virtual load balancer. Several VMs will be assigned to the server farm and a VIP will be assigned for accessibility.
Provisioning of ACLs	Deployment of the security policies based on ACLs. The security policies determine the accessibility of applications of a network zone by applications of other security zones (e.g. other network zone, corporate network, another tenant).
Provisioning of NAT	Deployment of NAT rules. NAT rules are mandatory in case of overlapping IP address range assignment for different tenants.
Server farm provisioning	Deployment of a server farm based on a configuration of the virtual load balancer. Several VMs will be assigned to the server farm and a VIP will be assigned for accessibility.
Provisioning of ACLs	Deployment of the security policies based on ACLs. The security policies determine the accessibility of applications of a network zone by applications of other security zones (e.g. other network zone, corporate network, another tenant).
Provisioning of NAT	Deployment of NAT rules. NAT rules are mandatory in case of overlapping IP address range assignment for different tenants.
Provisioning of security group	Deployment of security groups and assignment of applications with a security group. The security group function supports micro segmentation, which can prevent applications assigned to the same network zone from reaching each other.

AUTOMATION TASK	DESCRIPTION
Provisioning of host-based ACLs	Deployment of host-based ACLs. This security function also supports micro-segmentation. The advantage of this deployment is that it provides micro segmentation as near to the edge of the network as possible. The policy will be automatically removed in case of removal of the VM or application.
Provisioning of PAAS	Deployment of a VM including runtime service.
Provisioning of SAAS	Deployment of a VM including an application.

General use cases

Below is a list of general use cases. This list is far from complete, but forms a good starting point for exploring possibilities for automation and orchestration. The list covers both compute, storage and networking use cases. We have distinguished use cases for auto-provisioning, auto-healing and auto-scaling.

Auto-provisioning

Auto-provisioning is based on the provisioning of a service on demand.

The service can be initiated by a customer by selecting and pressing the button for this service on a portal. The service will then be automatically provisioned, and the required OSS and BSS will be automatically updated, for example for monitoring and billing purposes.

The service can also be initiated by a customer service request in the ITSM application. In this case, the service request needs to be approved by one or more authorized managers. After approval, the service will be automatically provisioned, including the update of the OSS and BSS.

The table below shows an overview of use cases to be used for automatic service provisioning.

USE CASE	AUTO-PROVISIONING TASK
Server provisioning	Add/remove a VM (IAAS). Add/remove a VM + OS + middleware + runtime (PAAS).
Storage provisioning	Add/remove storage (IAAS).
Network provisioning	Add/remove a network (IAAS), e.g. virtual private cloud (VPC).
Security provisioning	Add/remove a security policy.
Application provisioning	Add/remove an application (SAAS).
End-to-end service provisioning	Provisioning of multiple services. E.g. Telco provider for customer service (L3VPN + tenant + B2B application).
Zero touch provisioning for bare metal devices	Ordering of bare metal device (manual action / time-consuming). Install bare metal device (manual action). From there, orchestration process can start for on-boarding device (fully automated).
WAN provisioning	Add/remove customer location, L2VPN, L3VPN, IPsec VPN.
Load balancer provisioning	Add/remove virtual load balancer.
Virtual firewall provisioning	Add/remove virtual firewall.

Auto-healing

Auto-healing is based on the automatic adjustment of services after the detection of failures in the network infrastructure. The adjustments are based on an analysis (e.g. artificial intelligence and machine learning) of the monitoring and logging information.

The table below shows an overview of use cases determined for automatic service healing.

USE CASE	AUTO-HEALING TASK
Faulty computing device	Move VMs to backup computing device.
Faulty VM	Restart VM.
Faulty storage device	Move logical storage to backup storage device.
Network congestion	Reroute traffic over backup data plane.
Security problem	Add or adjust security policy. Block traffic.

Auto-scaling

Auto-scaling provides the automatic adjustment of services after the detection of capacity issues in the network infrastructure. The adjustments are based on an analysis (e.g. artificial intelligence and machine learning) of monitoring and logging information.

The table below shows an overview of use cases determined for automatic service scaling.

USE CASE	AUTO-SCALING TASK
Computing resource	Add/remove computing resource (CPU, memory). Move VM to other computing device in case resource of current device reaches limits. Clear a bare metal server of VMs by moving VMs to other servers.
Storage resource	Add/remove logical storage. Move logical storage to other storage device.
Network resource	Adding hardware resources that need zero-touch provisioning. Scaling-in/scaling-out of network services.
Bandwidth resource	Adjust allowed bandwidth for specific application. Adjust QoS settings. Reroute traffic
Daytime resource allocation	Adjust QoS settings. Add/remove resources. Power-down or power-up devices.
Hardware device	Power-off/power-on device when not in use and when needed again.

6 IMPACT

IMPACT

This chapter describes the impact that automation and orchestration have on the (sub)systems of any IT organization, and discusses the roles of the people responsible for designing, building and running phases of an automation and orchestration implementation.

Siloed systems are the problem ...

All IT organizations share one major problem: siloed (sub)systems. A siloed (sub)system is any system (in this context a system is a group of devices forming one ecosystem) that is unable to operate with any other system. Silos create an environment of individual and separate systems within an IT organization. The technology domains as shown in figure 2 represent siloed systems. What's more, each of these technology domains can have their own siloed (sub)systems based on e.g. compute, storage and networking functions. Siloed systems prevent fast provisioning of services on customer demand and fast automatic adaptation to changes and faults in the IT infrastructure - especially for cross-silo services.

... and abstraction layers are the solution

The previous chapters discussed a technological solution that can deal with siloed systems. This solution is essentially based on adding abstraction layers. It is a way of hiding the working details of a subsystem, allowing the separation of concerns to facilitate interoperability and platform independence. The architecture as discussed before shows two deployment scenarios for an abstraction layer:

- A domain controller acting as an abstraction layer for one technology domain. The domain controller provides one central API for the services of one technology domain.
- A cross-domain controller acting as an abstraction layer for multiple technology domains. The cross-domain controller provides one central API for the services of multiple technology domains.

Software that supports the overall business goals ...

Each of these abstraction layers are modularized and organized in a way that is comprehensible to business teams, while still containing enough levels of detail for the technical audience. This approach prevents communication mismatches

and puts the focus on the real aims of the business. Agile principles ensure that both business and technical focuses are considered in the development stage, which in turn leads to software that supports the overall business goals more adequately. An abstraction layer ensures that these benefits are applied to technical capabilities, grouping them in modules that support business capabilities.

... will impact people and processes

The automation and orchestration of services covering multiple siloed systems will impact both people and processes of an IT organization:

- People need to be assigned to cross-functional teams. This will facilitate direct communication for people responsible for separate siloed systems and prevent communication mismatches. The people responsible for service provisioning must posses software skills to support the programmability of the services.
- Processes should align with the DevOps approach of development and operations. This approach enables people of development and operations to work together in an agile way.

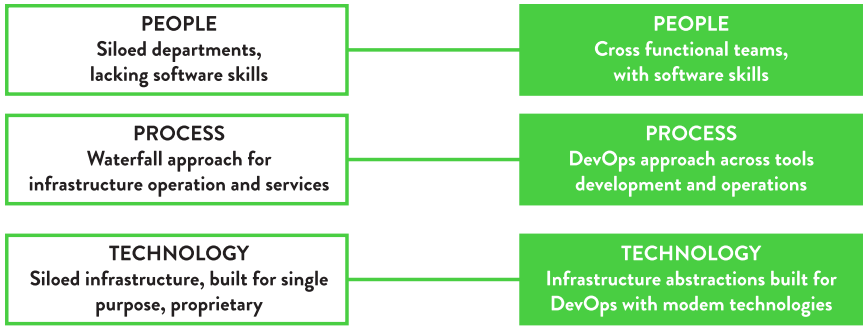


Figure 7: current (left) and target (right) situation for IT organization

Roles of people responsible

Below, we will delve slightly deeper into the roles of people responsible for designing, building and running phases of any automation and orchestration implementation.

Business consultant

The business consultant advises organizations on how to best use automation and orchestration to achieve their business objectives. He provides advice on the road map for automation and orchestration, and regarding new services and business requirements.

Solution architect

The solution architect translates the business requirements to a technical solution. This technical solution can consist of the following components:

- technical domains to be readied for leveraging the service,
- orchestration tooling to be readied for leveraging the service,
- service models,
- processes for auto-provisioning, auto-healing, auto-scaling,
- and processes for monitoring and analysis.

Service developer

The service developer translates the solution design to a technical design that is ready for deployment in the infrastructure. The service developer provides the detailed technical design documents for the services. Together with operations, the service developer makes sure that the service is ready for production, using the DevOps approach of development and operations.

The service developer must possess software and programmer skills to create the scripts for automating these configuration tasks:

- scripts for automating tasks for the technology domains,
- scripts for auto-provisioning of service models,
- scripts for auto-healing of service models,
- and scripts for auto-scaling of service models.

To be able to realize service integration, the service developer should also possess knowledge of the following components:

- technology domains,
- OSS applications for e.g. monitoring and analysis,
- BSS applications for e.g. CRM, billing, on-demand provisioning,
- and orchestration tools.

Operations engineer OSS and BSS

The operation engineer provisions services in the infrastructure and manages service quality.

Together with development, the operations engineer will make the service ready for production, using the DevOps approach of development and operations.

The operations engineer should possess software and programmer skills to validate and troubleshoot the programmable scripts. For troubleshooting and provisioning purposes, the operations engineer should possess knowledge of the technology domains, OSS applications, BSS applications and orchestration tools.

Technology domain engineer

In an automated and orchestrated infrastructure, the OSS provides the monitoring and analysis functions for the infrastructure. In most cases, the OSS provides operations with the troubleshooting information it needs to solve the incident. For complex technical incidents, deep technical troubleshooting is required, however. This requires troubleshooting of the infrastructure on CLI level.

This requires deep technical knowledge. A technical domain engineer, technical expert for the technology domain, is required for troubleshooting these complex technical incidents. In most cases, the technology domain engineer is an external specialist.

7

DEVELOPMENT, TESTING, ACCEPTANCE AND PRODUCTION: DTAP

DEVELOPMENT, TESTING, ACCEPTANCE AND PRODUCTION: DTAP

In this chapter, we discuss the DTAP process for automation and orchestration. DTAP stands for Development, Testing, Acceptance and Production. We explain our DTAP architecture and share our best practices regarding the DTAP process.

DTAP helps control changes ...

Auto-provisioning, auto-healing, and auto-scaling processes should implement changes in a production environment with no disturbances - or with only minor, controlled disturbances. IT organizations that want to be in control of changes should therefore commit to a DTAP process. This process ensures that no disturbances occur in the production environment, or, if any, only minor, controlled, disturbances.

... in four steps

1. The program or component is developed on a development system of the development environment. The development environment is the environment in which changes to software are developed, for instance an individual developer's workstation. The development environment might not have testing capabilities.
2. Once the developer feels the product is ready, it is copied to a testing environment, where its functionality is verified. This environment should be standardized, and in close alignment with the target production environment.
3. If the test is successful, the product is copied to an acceptance test environment. During the acceptance test, the customer checks whether the product meets his expectations. The acceptance test environment is an environment for testing that perfectly resembles a production environment, with the same tooling, the same service models and the same IP address assignment.
4. Once the customer accepts the product, it is deployed to a production environment, making it available to all users of the system.

The figure below shows a high-level overview of the Routz DTAP architecture.

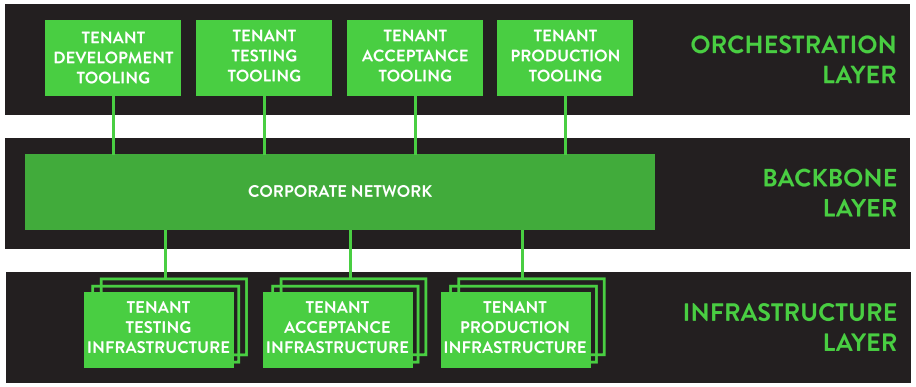


Figure 8: Routz architecture for DTAP

The Routz DTAP architecture consists of isolated functional building blocks, so-called tenants. Each tenant has an interface to the corporate backbone, and provides an isolated availability domain. The corporate backbone provides end-to-end IP connectivity for the tenants. In this architecture, we distinguish seven separate tenants:

- The Tenant development tooling provides the tooling for software development.
- The Tenant testing tooling provides the tooling for testing the process for auto-provisioning, auto-healing and auto-scaling of services of the testing environment.
- The Tenant acceptance tooling provides the tooling for testing the process for auto-provisioning, auto-healing and auto-scaling of the service of the acceptance environment.
- The Tenant production tooling provides the tooling for auto-provisioning, auto-healing and auto-scaling of services in the production environment.
- The Tenant testing infrastructure provides the test infrastructure for the deployment of the customer service models. It provides an isolated environment for testing the deployment process.
- The Tenant acceptance infrastructure provides the acceptance infrastructure for the deployment of customer service models. It provides an isolated environment for testing the deployment process and particularly the production deployment process.
- The Tenant production infrastructure provides the production infrastructure for the deployment of customer production services.

Orchestration layer tenants may be shared …

The tenants of the orchestration layer can be shared. Tooling of these tenants is used for execution and testing of the auto-provisioning, the auto-healing and the auto-scaling process for all the tenants of the infrastructure layer. Due to this sharing feature, the DTAP architecture contains just four separated tenants for the orchestration layer. Depending on the specifics of their situation, IT organizations can decide to provide extended isolation for the orchestration layer. However, for most customer situations, the four orchestration tenants as shown in the DTAP architecture will meet customer DTAP requirements.

… whereas infrastructure layer tenants may not

The tenants of the infrastructure layer are not shared; each tenant is dedicated to a separated customer or service. The infrastructure layer can consist of multiple infrastructure tenants. The infrastructure layer does not consist of isolated tenants for development, since the development environment does not require testing capabilities for the infrastructure.

Our best practices

Use dedicated tooling and greenfield databases

To conform to the ‘think big, start small’ principle, most IT organizations start with a PoC phase for automation and orchestration and focus on a minor selection of use cases. This phase requires three tenants to be deployed - namely the ones for development tooling, for testing tooling and for testing infrastructure. Our advice is to start with a greenfield database for this PoC phase and to use dedicated tooling. You should prevent any dependency with existing databases and tooling and prevent these existing processes from impacting your PoC.

If the results of the PoC are satisfactory, you may decide to deploy the tested services in the production environment. You should then provide the tenants with acceptance for production. At this stage, we also advise you to use separated tooling and databases for acceptance and production – and, with regard to the databases, to start with a greenfield database.

Of course, this will increase costs. You might be tempted to use shared tooling and shared database(s). However, this could have an impact on the availability for production, as incidents in the acceptance environment could force incidents in the production environment.

Use separated hardware devices for production tenants …
Every tenant is an isolated environment, each with their own compute, storage and network resources. This isolation can be achieved logically or physically. Physical isolation is achieved by using separate hardware appliances for each tenant. Logical isolation is based on Network Functions Virtualization (NFV). NFV is the process of decoupling the network functions from proprietary hardware devices, so they can run in software on standardized hardware. In this situation, hardware devices can be shared by multiple tenants.

Our advice is to use separated hardware devices for the production environment. If the production environment were to share hardware devices with, for example, the acceptance environment, changes in the acceptance environment could decrease the availability for the production environment. This is to be avoided: changes within a development, testing or acceptance tenant should not have any impact on a production tenant.

… except for the IP transport service
For most of their compute, storage and network services, IT organizations can cope with an architecture based on separated hardware devices for production. However, in the case of the IP transport service, this would lead to much higher costs. The IP transport service (shown in figure 8 as corporate backbone) provides end-to-end IP connectivity for end users and services. It can consist of datacenter switching service(s), campus switching service(s) and WAN routing services (including Internet and third-party connections). It would be very costly to build an identical IP transport service for both production and acceptance (including development and testing). In most cases, we advise you to use shared hardware for the IP transport services, shared by all tenants.

… and even then
Some changes might still require acceptance testing on separated hardware. This is especially valid for software upgrades for hardware devices. It can be very helpful to test upgrades on separated hardware before executing the upgrade on the shared hardware device of the production IP transport service. Therefore, we suggest you facilitate a small subset of the IP transport services based on separated hardware, and use this subset for these critical acceptance tests.

CONTACT: INFO@ROUTZ.NL
TEL: 020-5651800


Praecellenti
The Network Academy