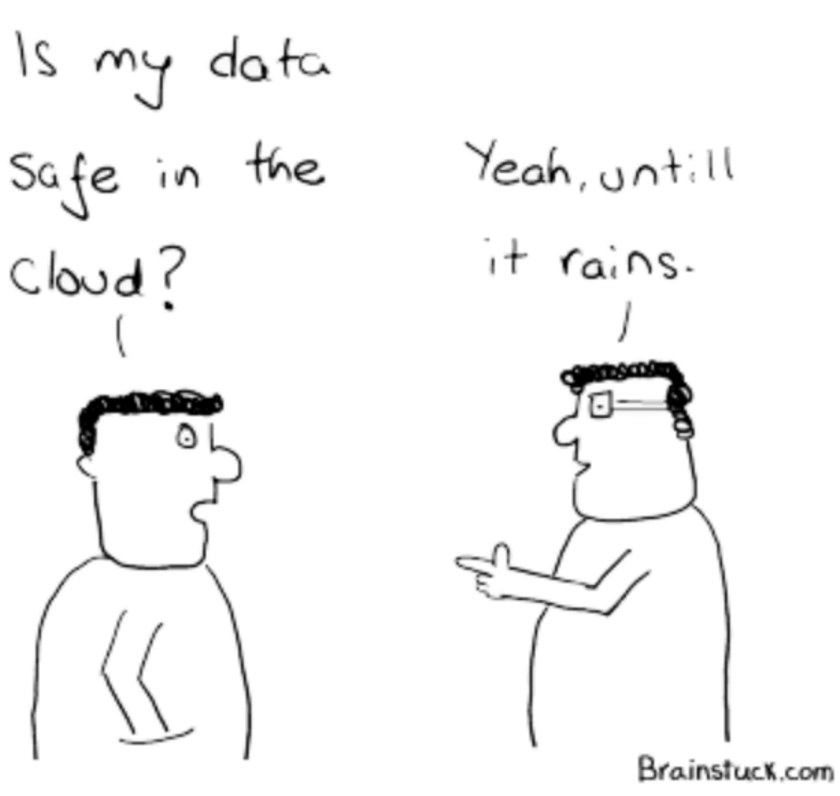
DHBW Informatik Wahlpflicht
Frühjahr 2022 - Cloud Computing
Chapter 2 IaaS

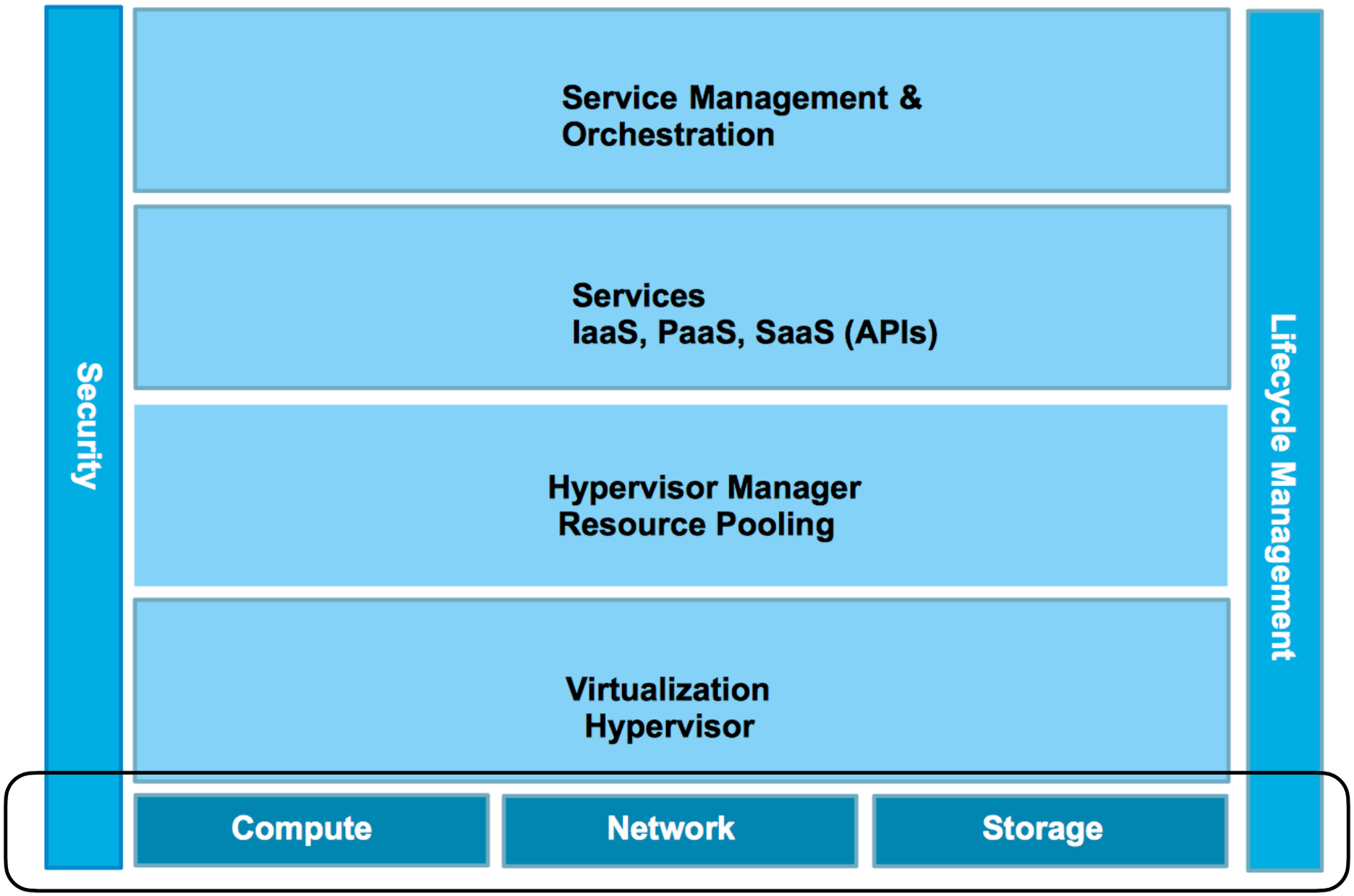
Juergen Schneider



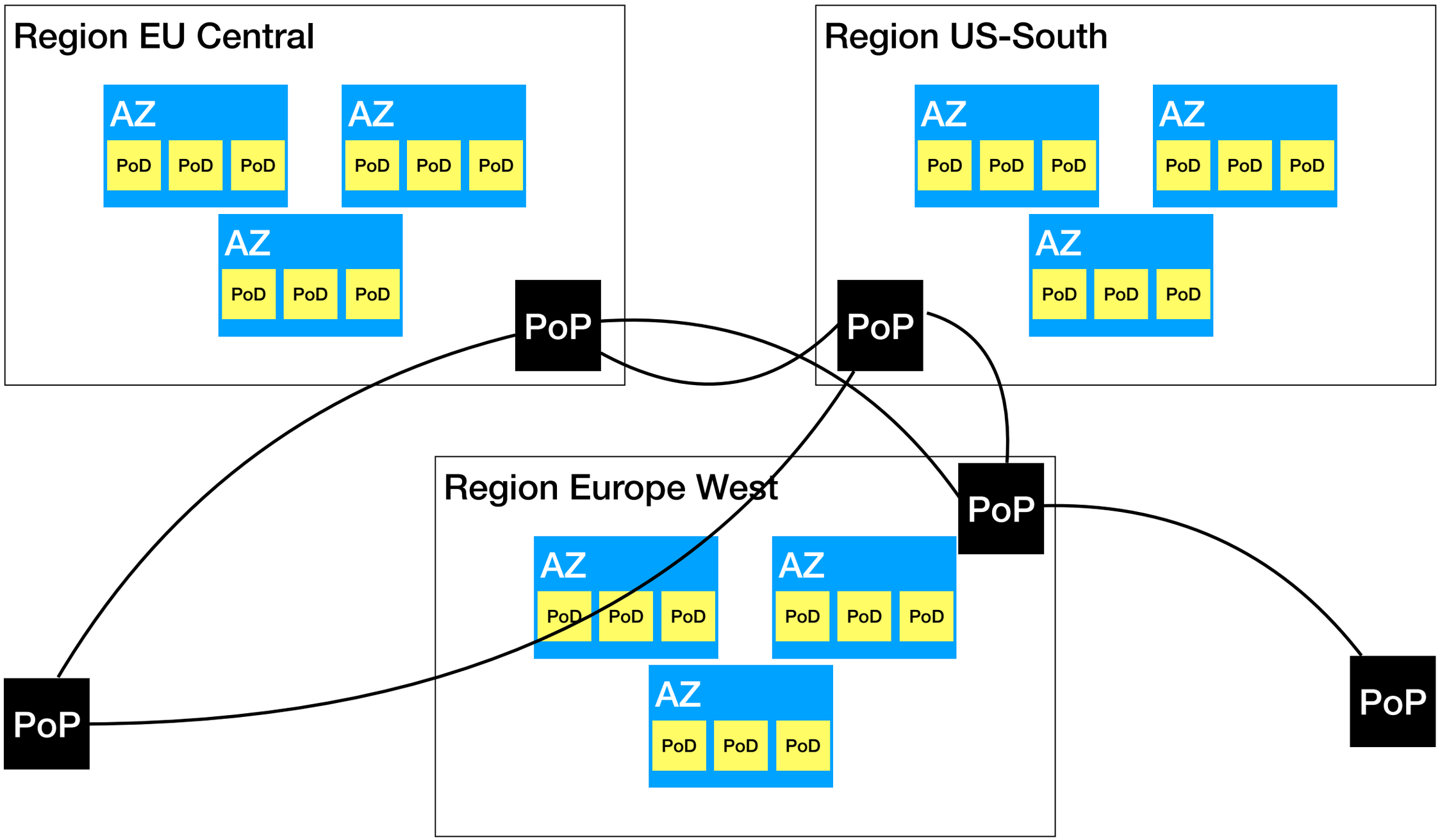
Agenda

- Cloud Data Center physical Layout
- Virtualization and Virtualization Management
- Open Stack
- Automation (Terraform and friends)

Technology Layering



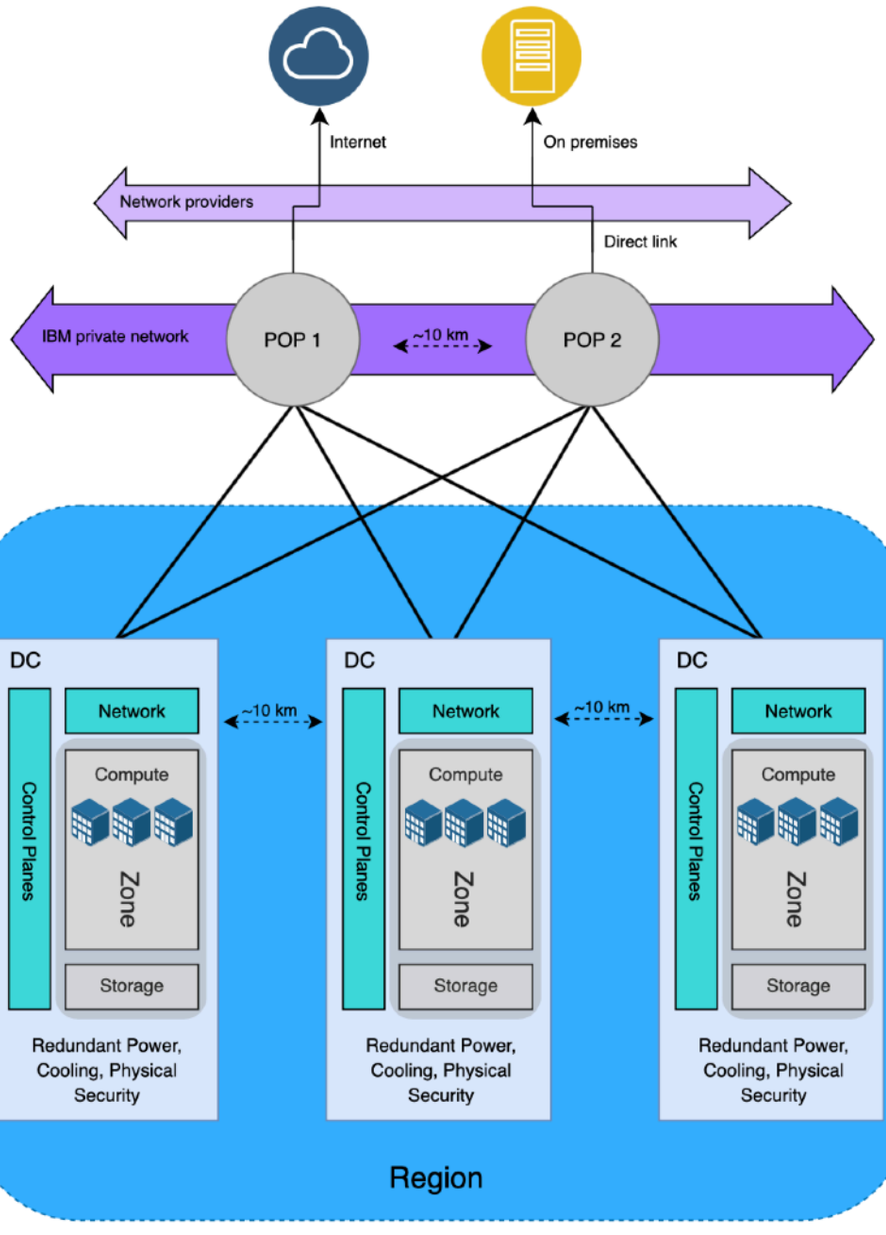
Cloud Data Centers
Physical Layout



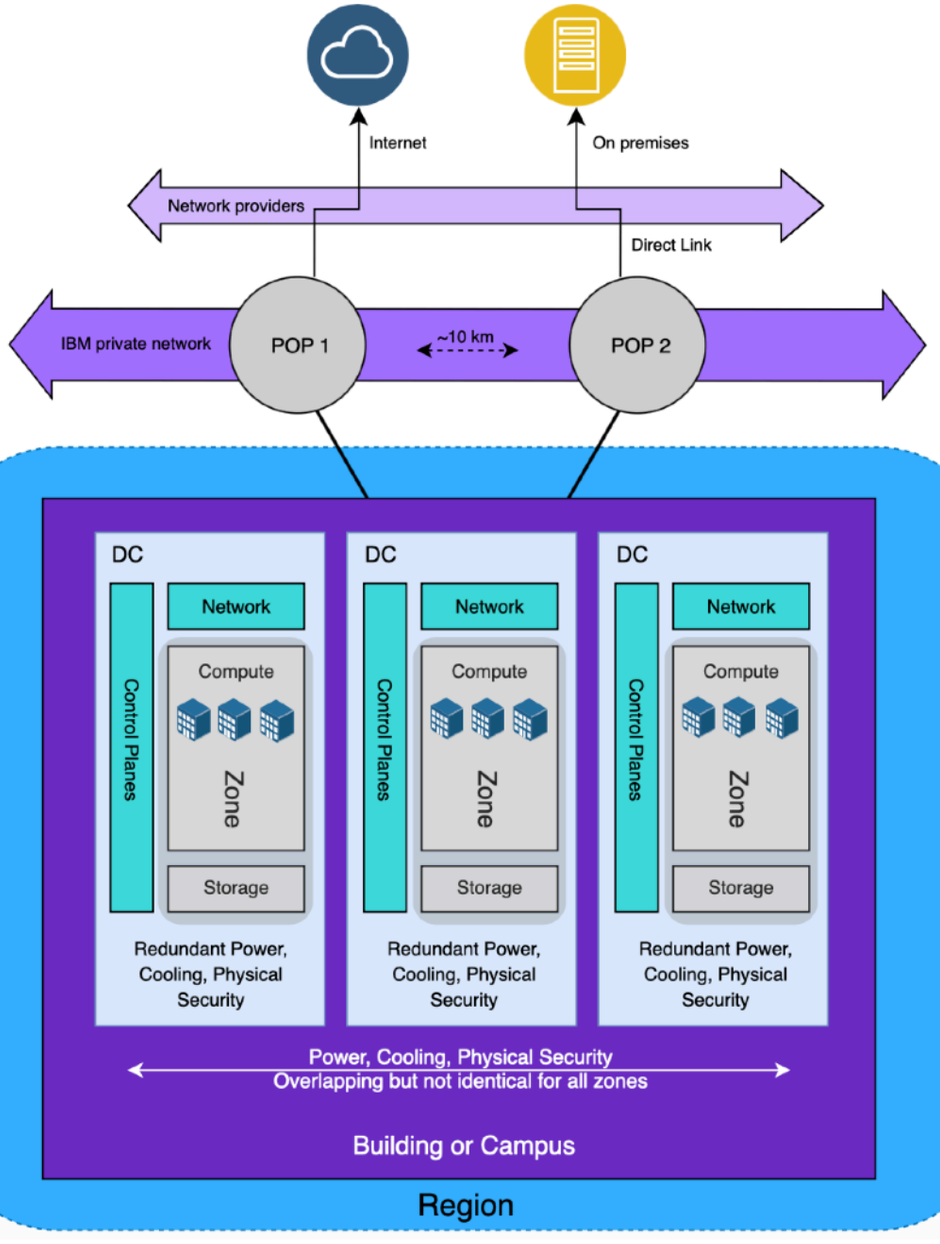
- Regions = important Business Region (low latency) (typical Availability 99.99% = 365/10000; about 53min downtime im Jahr)
- AZ = Availability Zone (typically a single DC building)
- PoD = Point of Delivery (typically a floor with Servers)
- PoP = Point of Presence (Network Entry of all sorts to all Cloud Data Centers, each PoP is a entry to all Data Centers, high speed links)

The IBM Picture

Multi AZ Region



Multi AZ Campus/Building



https://cloud.ibm.com/docs/overview?topic=overview-locations

Microsoft Azure Data Centers

https://azure.microsoft.com/en-us/global-infrastructure/regions/

Azure regions

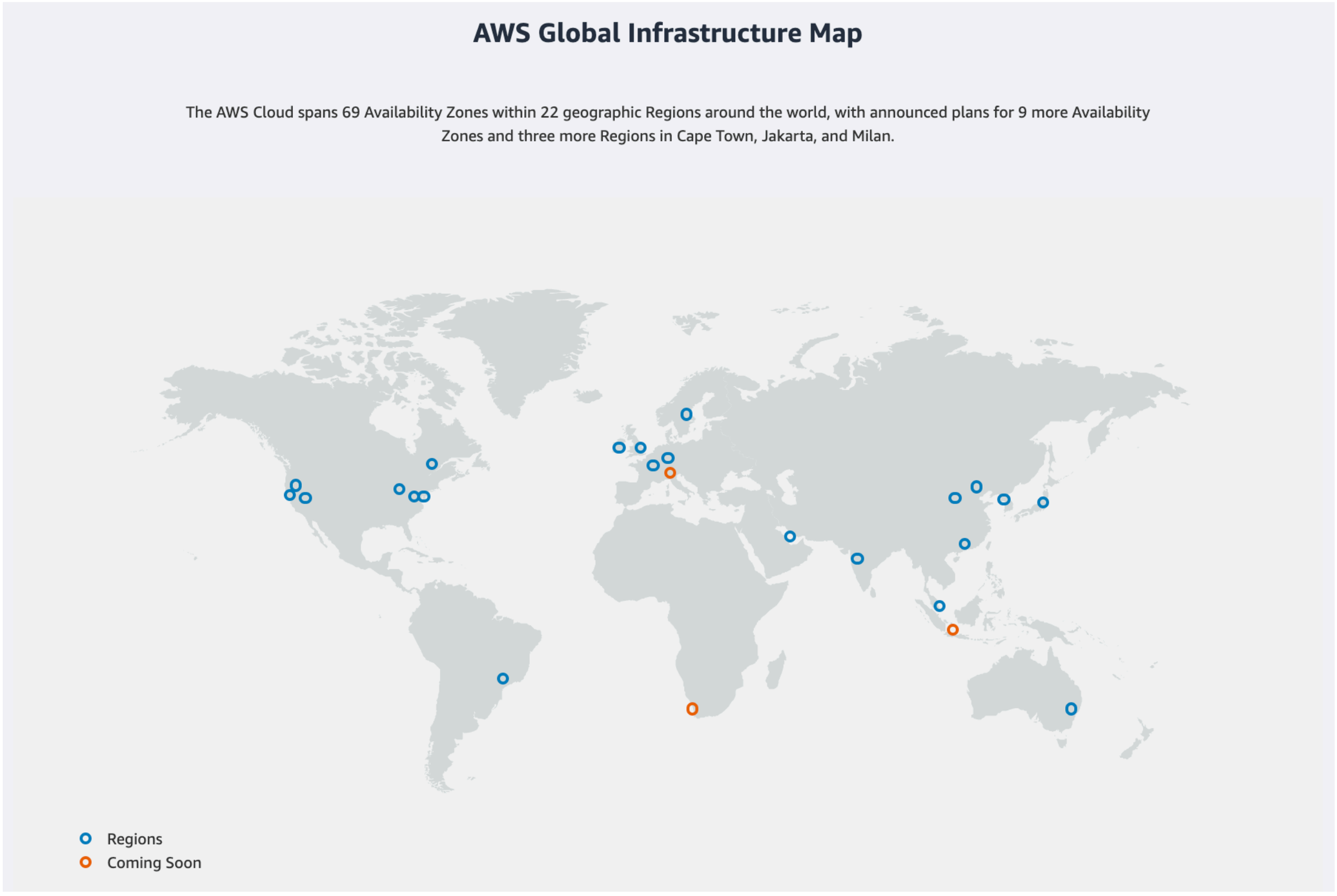
Azure has more global regions than any other cloud provider—offering the scale needed to bring applications closer to users around the world, preserving data residency, and offering comprehensive compliance and resiliency options for customers.

54 regions worldwide

140 available in 140 countries

Amazon Cloud DCs

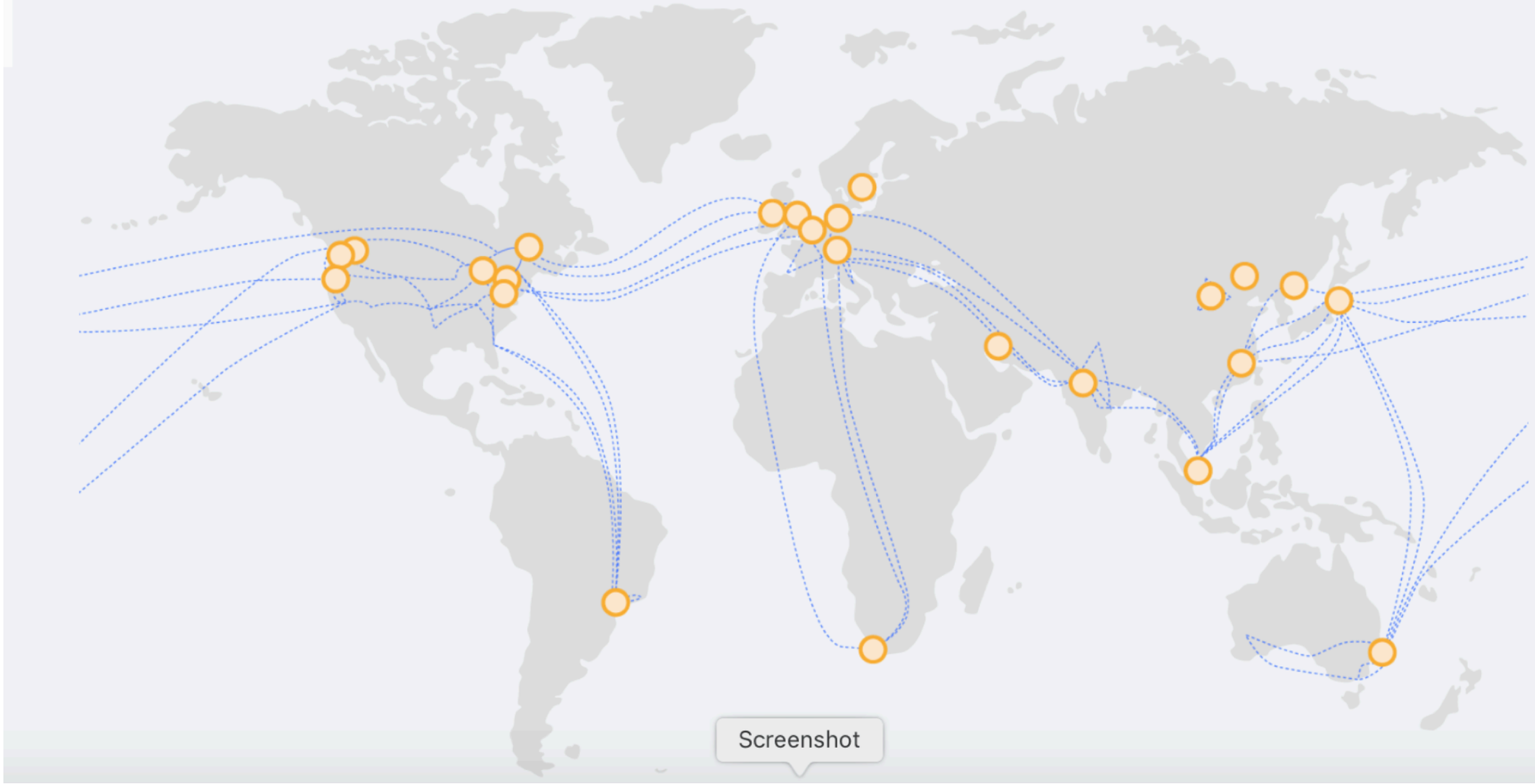
https://aws.amazon.com/about-aws/global-infrastructure/
https://www.infrastructure.aws



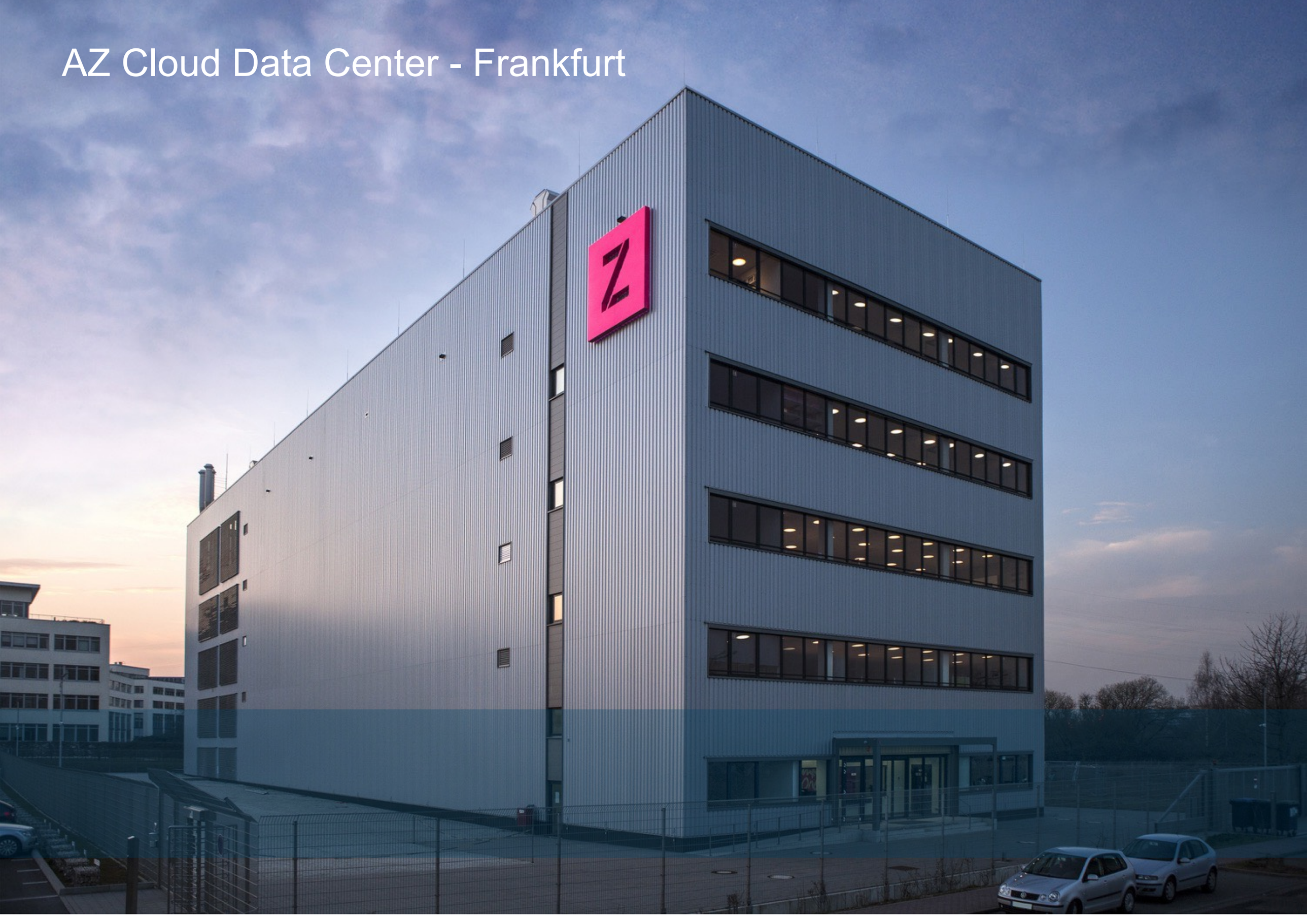
Amazon Global Network

Global Infrastructure

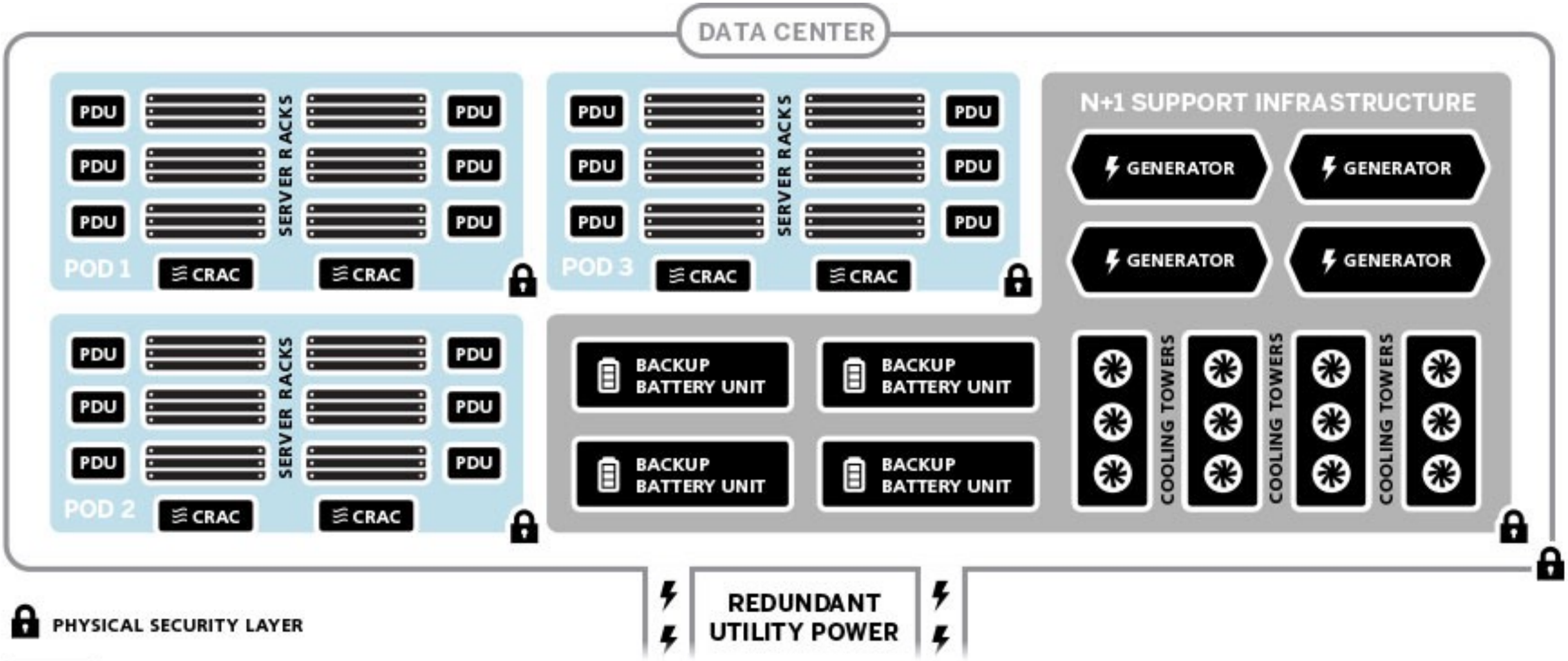
Every data center, AZ, and AWS Region is interconnected via a purpose-built, highly available, and low-latency private global network infrastructure. The network is built on a global, fully redundant, parallel 100 GbE metro fiber network that is linked via trans-oceanic cables across the Atlantic, Pacific, and Indian Oceans, as well as the Mediterranean, Red Sea, and South China Seas.



AZ Cloud Data Center - Frankfurt



IBM Datacenter Architecture

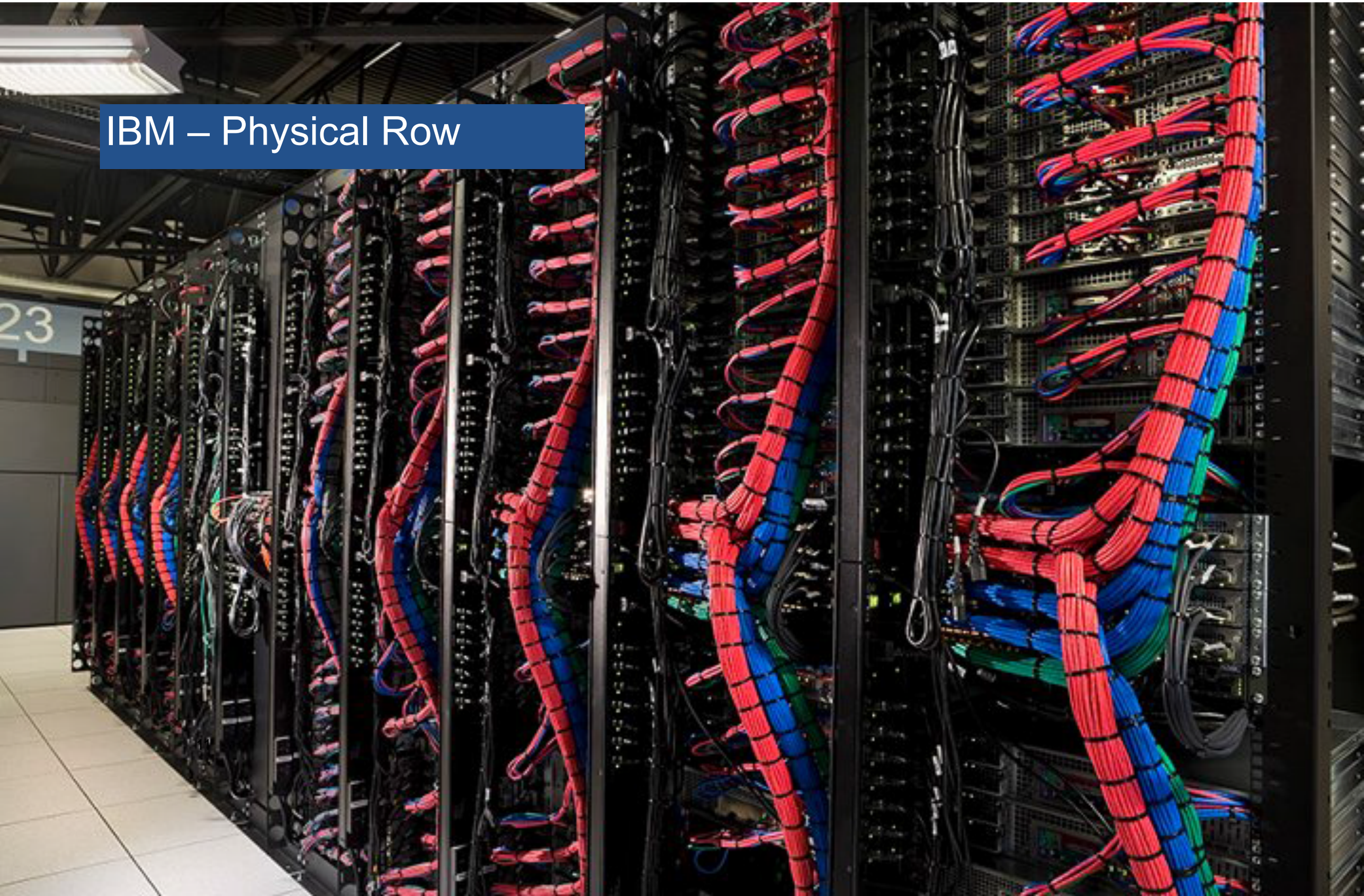


N+1 Redundancy = One more than normally needed

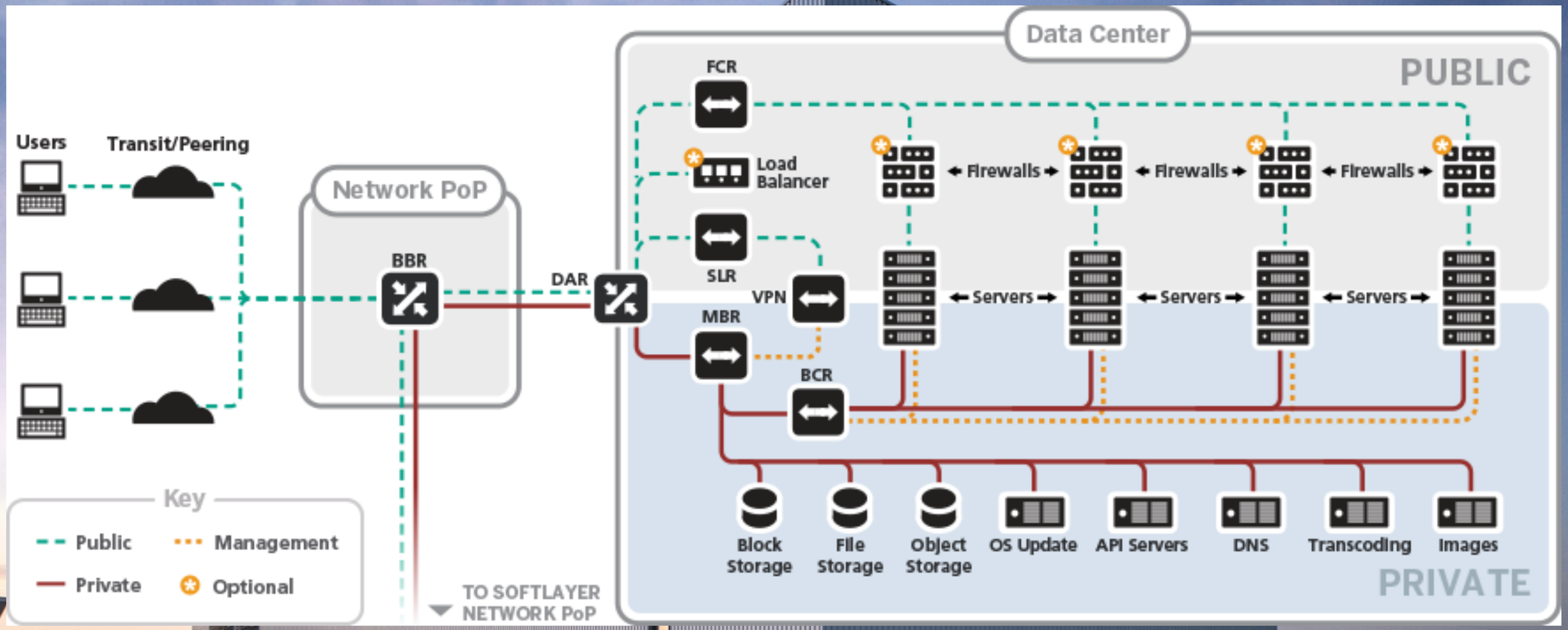
- 4 Cooling Towers for 3 Pods (so one cooling Tower could fail, but not 2)
- 4 Generators for 3 Pods (after an power outage one additional Generator could fail)
- 4 Backup Batterie Units (one could fail….)

2N for Power

IBM – Physical Row



IBM Cloud Data Center Network Layout

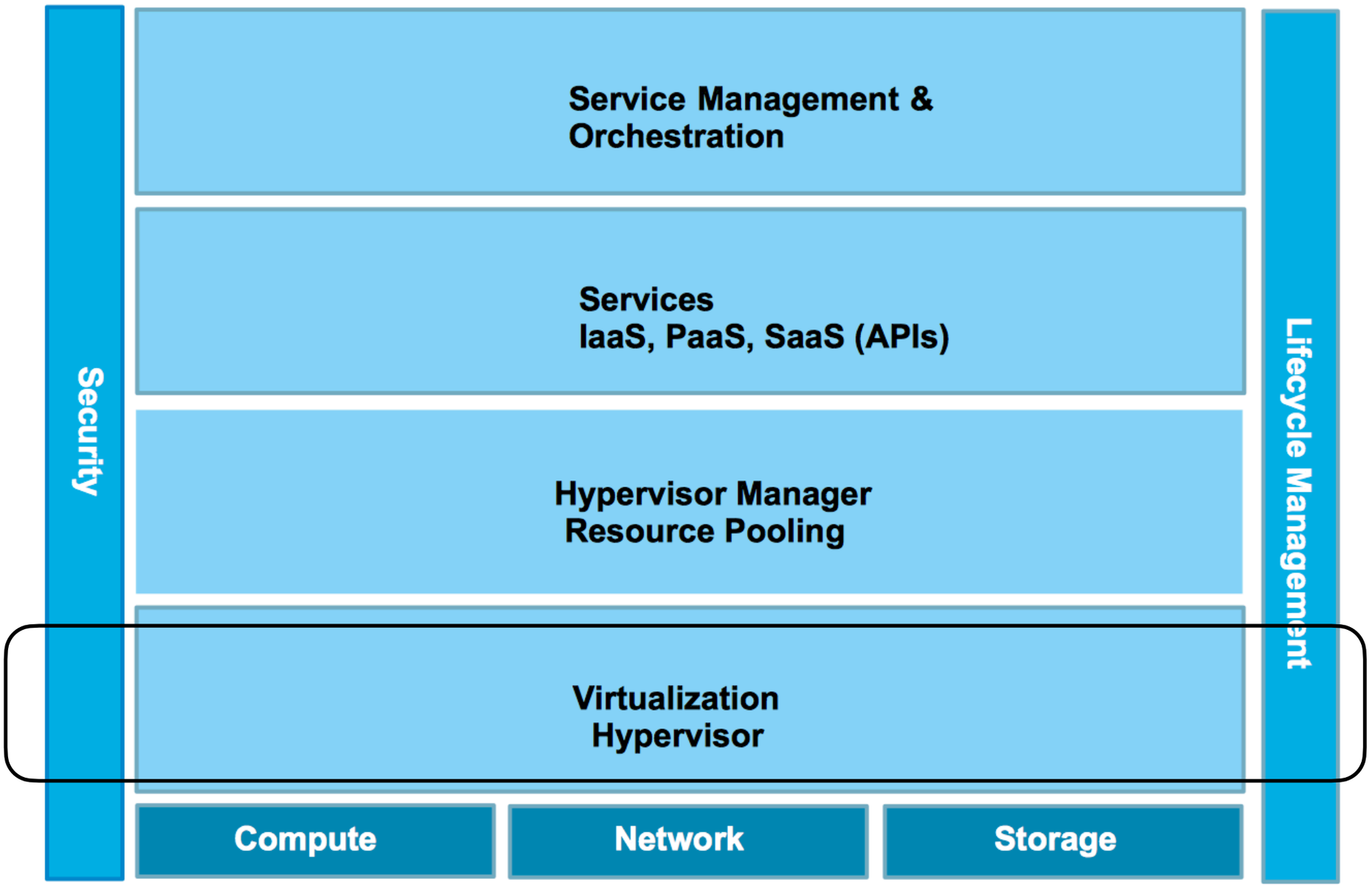


- 3 types of networks (public, private, management)
- Network Access to one POD is controlled by a FCR (Front Customer Router) / BCR (Backend Customer Router) pair

Security - Current State of the Art

- Internet (HTTPS) TLS 1.2, or VPN Server
- Data
  - Storage Encryption AES 265 bit, Keys stored in a vault (Hardware Security Module HSM)
  - BYOK (bring your own key) a key owned by the consumer to wrap the data key
- IAM (Integrated Access Management)
  - Authentication and Authorization for your own service and the used cloud services
- Security and Event Management
  - Monitor traffics, logins, Application logs to scan for potential problems
  - Vulnerability Scanning
- Physical Access to the AZs
  - Protected thru video surveillance, proximity card and biometrics, audits, visitor logs etc.

Technologie Layering



What is Virtualization ?

VIR|TU|ELL (1) auf Deutsch

Herkunft:

- ~~französisch *virtuel*, lateinisch *virtus*: Tüchtigkeit; Mannhaftigkeit; Tugend~~
- englisch *virtual*: nicht in Wirklichkeit vorhanden, aber echt erscheinend

Virtualization Gives Users Idealized Resources

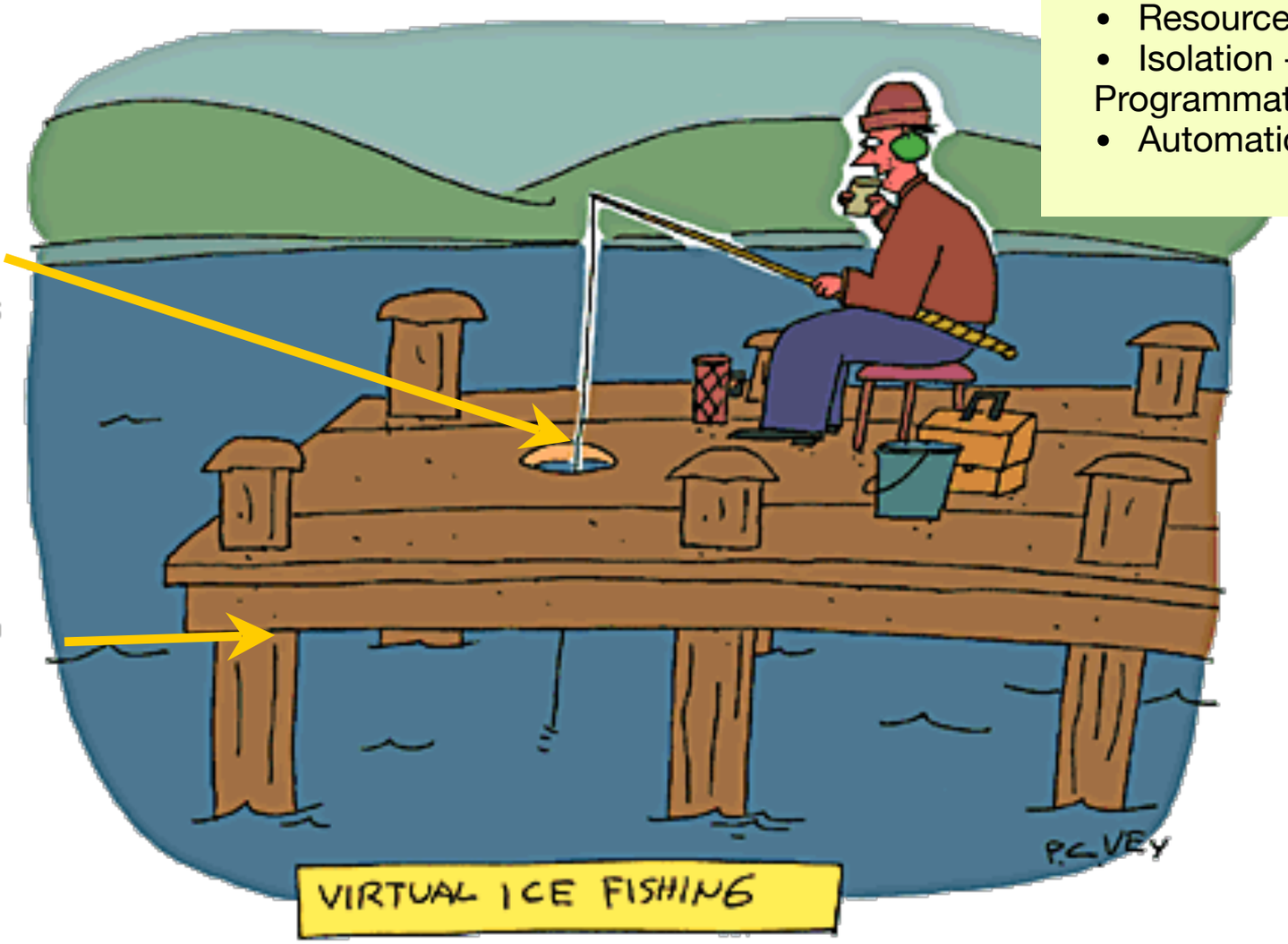
Virtualization can provide users with the experience they want

Efficient usage of IT Resources
- Resource Utilization
- Isolation -> critical for Cloud Multi Tenancy

Programmatic Access to the IT Resource
- Automation of deployment and operation

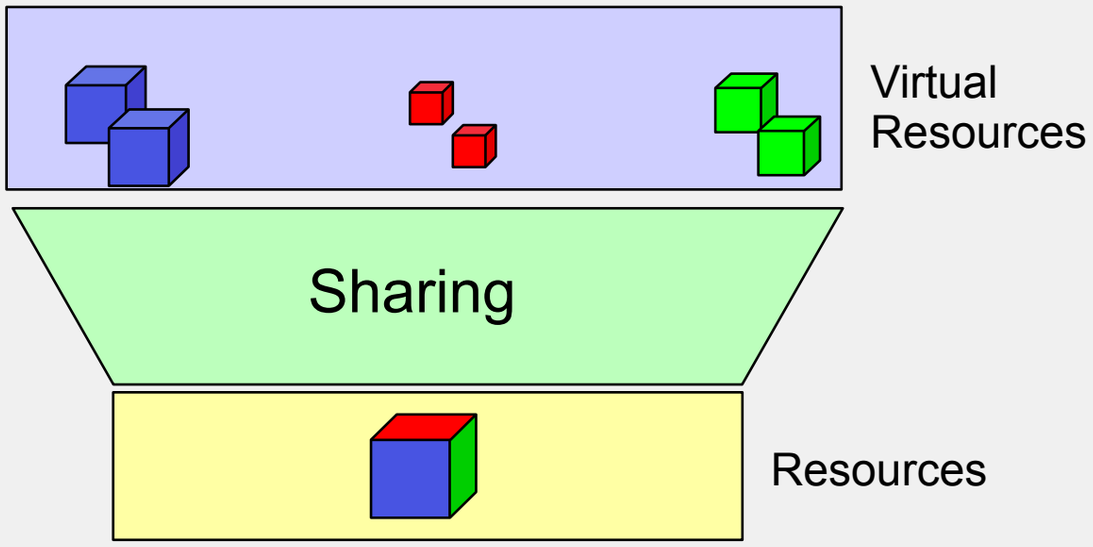
Architected Ice Fishing Interface

Virtual Ice
Has better RAS than real ice



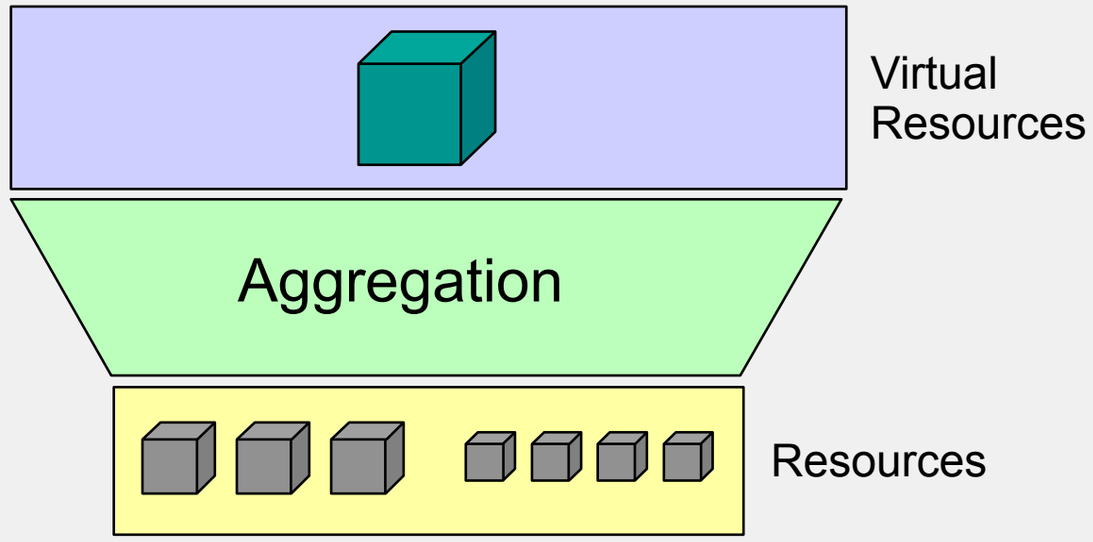
Gives you the illusion that you work with something that is more convenient or better than the reality

Virtualization Functions and Benefits



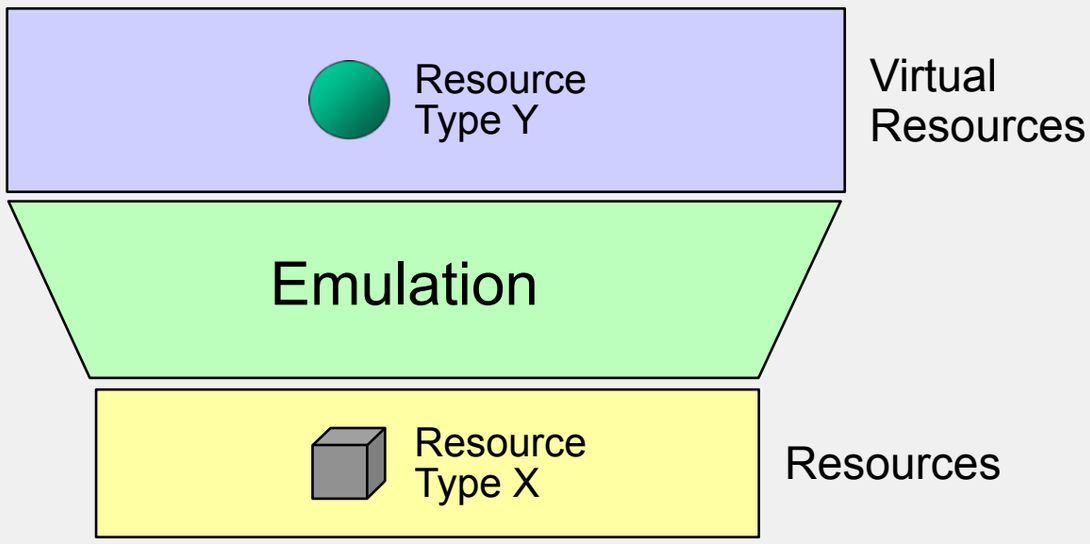
Examples: VMs, Container, LPARs, vCPU, virtual memory, virtual disks, VLAN

Benefits: Resource utilization, flexibility, isolation



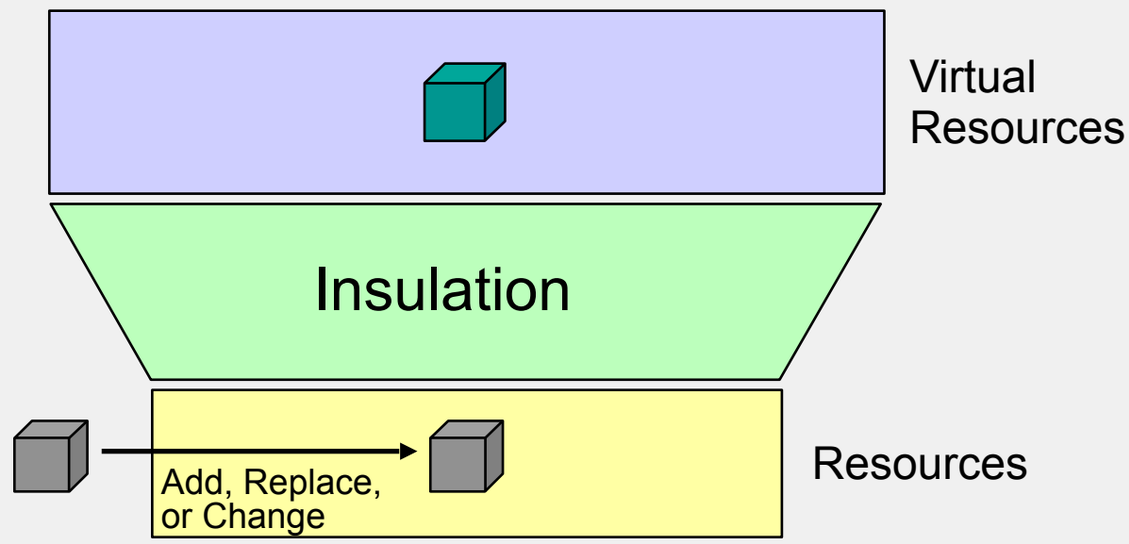
Examples: Virtual disks

Benefits: Resource utilization, flexibility, isolation



Examples: Arch. emulators, iSCSI, virtual tape

Benefits: Compatibility, software investment protection, interoperability, flexibility



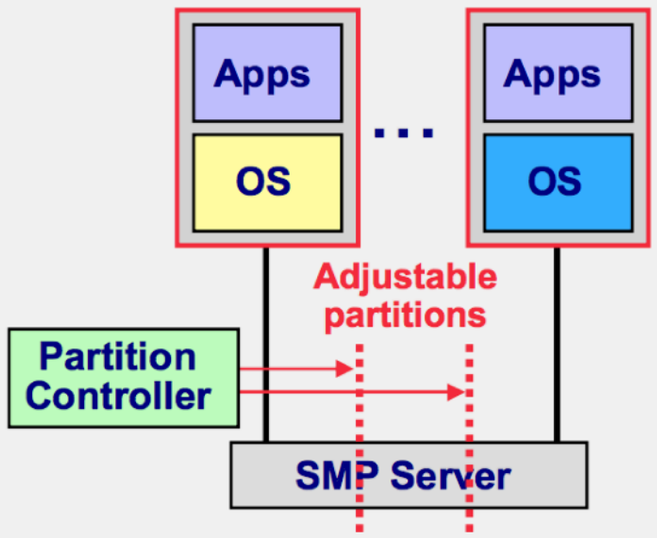
Examples: Spare CPU subst.

Benefits: Continuous availability, flexibility, software investment protection

Server Virtualization (late 80's and on)

Server Virtualization Approaches

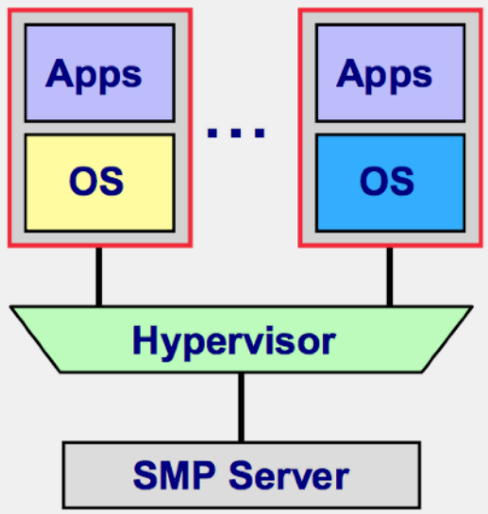
Hardware Partitioning



Physical partitioning
Sun Domains, HP nPartitions

Logical partitioning
pSeries LPAR, HP vPartitions

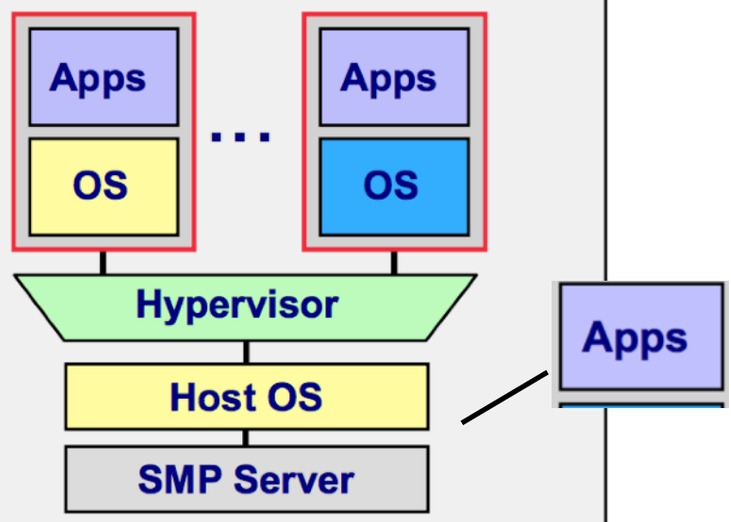
Hypervisor: Type 1



Hypervisor software/firmware runs directly on server

Examples
- Xen Server
- KVM
- HyperV
- VMWare ESXi

Hypervisor: Type 2



Hypervisor software runs on a host operating system

Examples
- Virtual Box
- KVM
- HyperKit (Mac)
- Windows Virtual PC
- VMWare Workstation

- Hardware partitioning subdivides a server into fractions, each of which can run an OS
- Hypervisors use a thin layer of code to achieve fine-grained, dynamic resource sharing
- Type 1 hypervisors with high efficiency and availability will become dominant for servers
- Type 2 hypervisors will be mainly for clients where host OS integration is desirable

Statements in the 80's

https://www.quora.com/Whats-the-difference-between-an-LPAR-and-a-Virtual-Machine

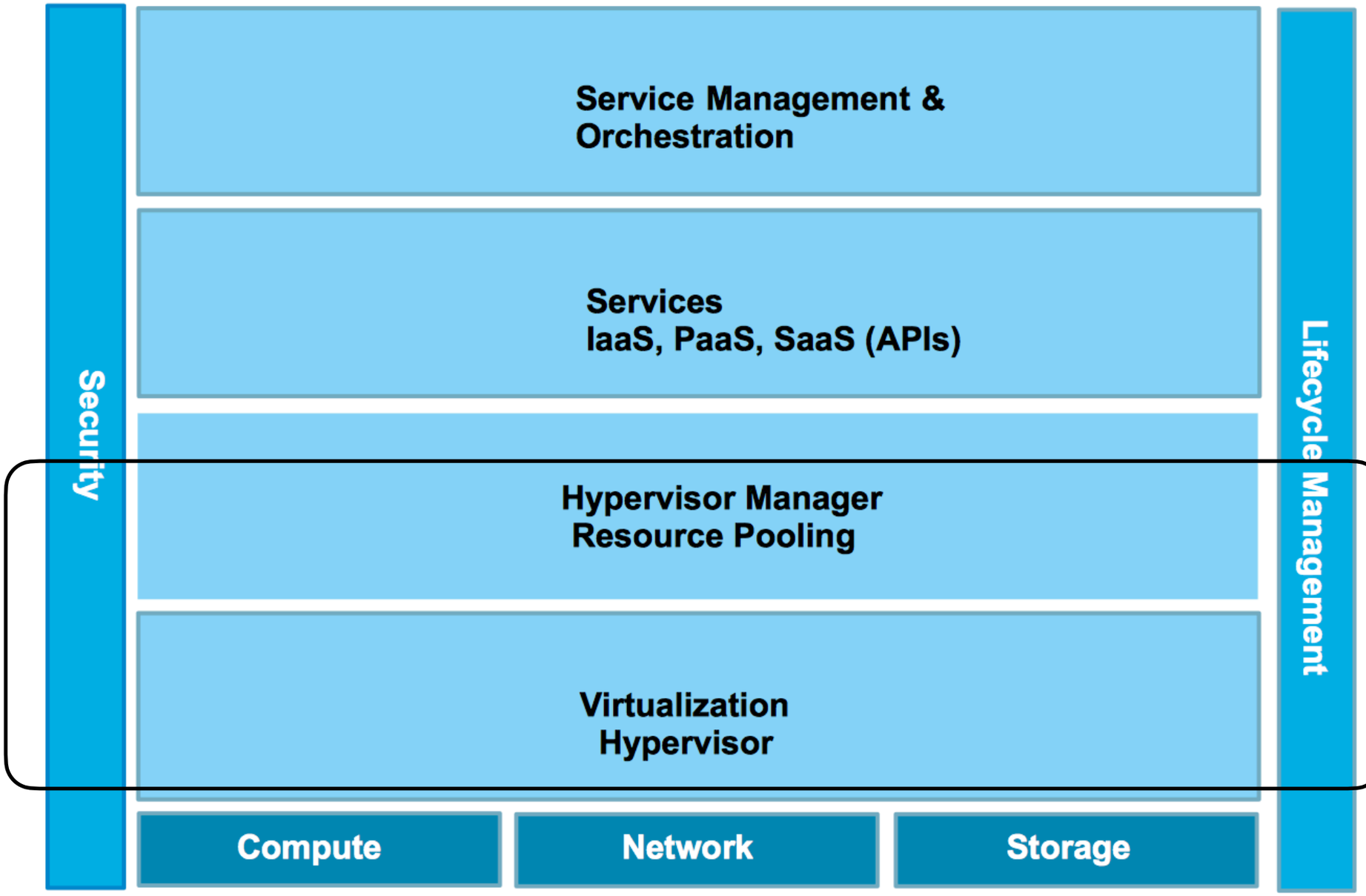
Type of virtualization

- **Partitioning**
    - HW based separation of physical HW resources, all OS runs unchanged directly on that subset (not really a VM)
- **paravirtualization (PV)**
    - Guest OS has been changed (is aware) that it is virtualized.
    - The guests are modified to use a special hypercall ABI, instead of certain architectural features. (XEN)
    - No need of Virtualisation Support in CPUs like Intel VT-x or AMD-V
- **Full virtualization**
    - CPUs that support virtualization make it possible to run unmodified guests, including proprietary operating systems (such as Microsoft Windows). This is known as hardware-assisted virtualization.
- https://en.wikipedia.org/wiki/Xen

Hypervisors

- Type 1
  - **KVM**, The open-source KVM (or Kernel-Based Virtual Machine) is a Linux-based type-1 hypervisor that can be added to most Linux operating systems including Ubuntu, SUSE, and Red Hat Enterprise Linux. It supports most common Linux operating systems, Solaris and ,Windows. Most distributions that offer KVM offer additional management tools on top such as Red Hat's Virtual Machine Manager
  - **Xen** is a type-1 bare-metal hypervisor. Just as Red Hat Enterprise Virtualization uses KVM, Citrix uses Xen in the commercial XenServer. In 2007 Citrix bought XenSource, Inc, who supported Xen. Today, the Xen open source projects and community are at Xen.org. Today, XenServer is a commercial tier-1 hypervisor solution from Citrix, offered in 4 editions.
  - **Hyper-V** is one of the top 3 Tier-1 hypervisors (First released with Windows Server, Hyper-V has now been greatly enhanced with Windows Server 2012 Hyper-V)
  - The leader in the Tier-1 hypervisors is **VMware** with their vSphere/ESXi product – available in a free edition and 5 commercial editions. VMware led the market in developing innovative features such as memory overcommitment, vMotion, Storage vMotion, Fault Tolerance, and more. Previously, VMware called their free hypervisor "Free ESXi" as ESXi Server is what is loaded directly on the physical server. However, VMware calls the "suite" of features "vSphere", available in various editions.
- Type 2
  - **VMware Workstation** has three major use cases: for running multiple different operating systems or versions of one OS on one desktop, for developers that need sandbox environments and snapshots, or for labs and demonstration purposes.
  - **Windows Virtual PC** is the latest Microsoft's version of this hypervisor technology, Windows Virtual PC and runs only on Windows 7 and supports only Windows operating systems running on it.
  - **VirtualBox** hypervisor technology provides reasonable performance and features if you want to virtualize on a budget. Despite being a free, hosted product with a very small footprint, VirtualBox shares many features with VMware vSphere and Microsoft Hyper-V.
- Please note:
  - Consider the hypervisor as 'done'. Critical is the management of the VMs (often include pools of physical Servers, and the network and storage part to be able to boot the OS)

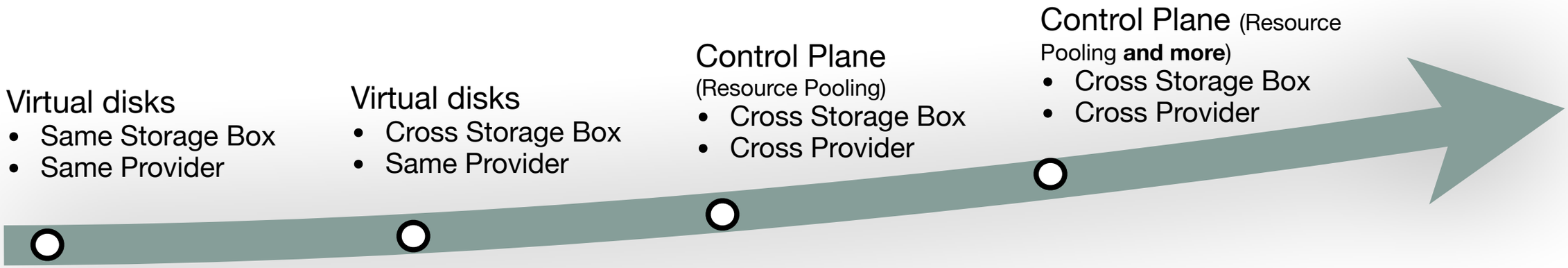
Technologie Layering



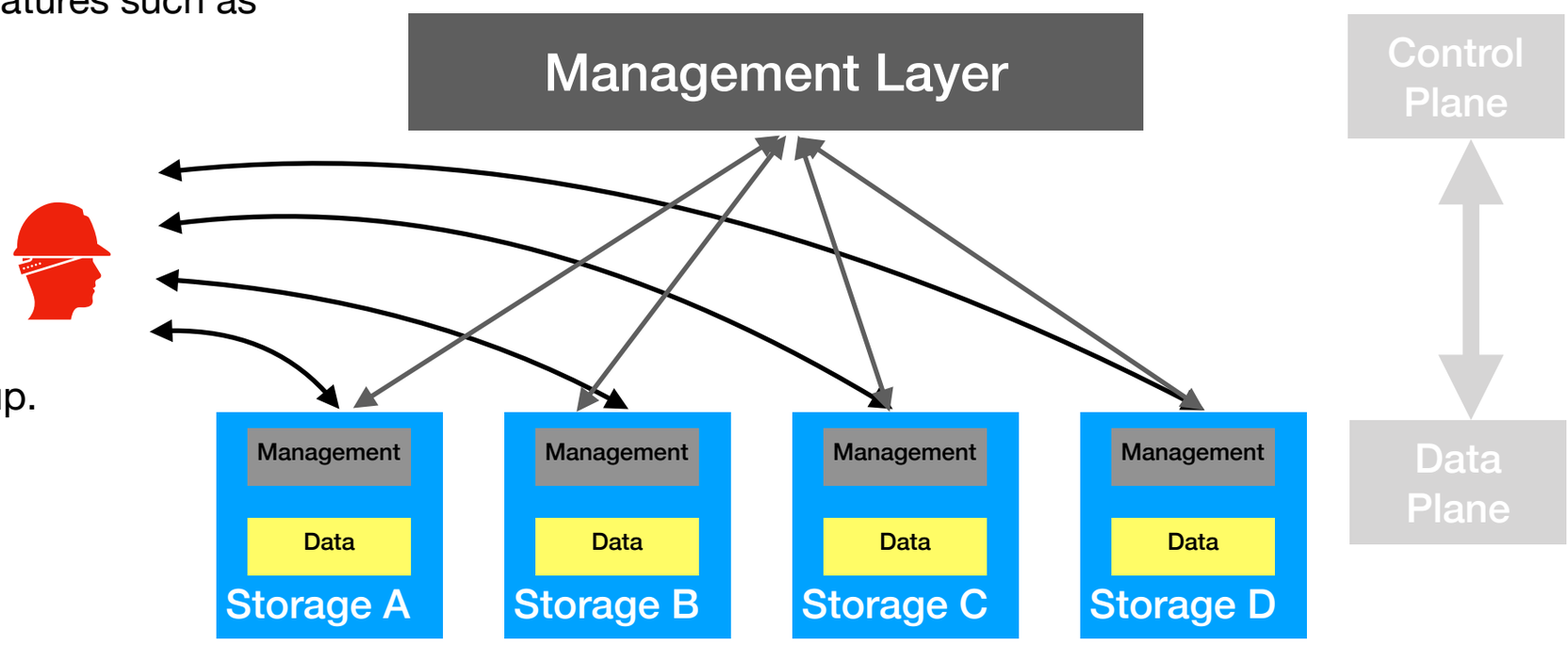
Software Defined…. (is needed)

- Why ? (Automation, Automation, Automation). (Remember the MAPE Loop)
- So what is meant by Software defined:
    - **Server, Storage and Network (Resources) can be fully controlled and defined by Software.**
        - Software-defined includes
            - Virtualized resources (they already have an Software Interface)
            - Physical Resources as long as they have a Software based Interface
                - Resources can be modelled as “Objects” or Templates including Resource Topologies (A Topology could mean an articulation of dependencies or groupings among resources)
                - Such modelled resources can typically be started/stopped (moved etc) and you can monitor the health and usage
- **Logically the next level of abstraction (for example resource pooling across physical entities)**
- A Marketing term for the next set of Management Features (further abstracting from the underlying vendor specific implementations)

Software Defined Storage



- **Storage Virtualization** (according : https://searchstorage.techtarget.com/definition/storage-virtualization)
    - Storage virtualization allows the pooling of physical storage from multiple storage devices into what appears to be a single storage device -- or pool of available storage capacity -- that is managed from a central console.
- **Software-defined storage** typically includes a form of storage virtualization **to separate the storage hardware (and providers) from the software that manages it.** (picture below implementing things in the dark grey layer)
- This Management Software (**Control Plane**) may also provide
    - policy management for features such as
        - data deduplication,
        - replication,
        - thin provisioning,
        - snapshots and backup.
        - Object Storage



And how is Network doing ???

- The Network HW dilemma:
    - Historically, the best networks — those which are the most reliable, have the highest availability and offer the fastest performance, etc. — **are those built with custom silicon (ASICs) and purpose-built hardware**. (ASIC = Application specific integrated circuit)
    - A **technology shift to software** can finally take place. It's this shift that underlies all SDN, NFV and NV technologies.
        - **NV**: ensures that the network can integrate with and support the demands of virtualized architectures, particularly those with **multi-tenancy** requirements. Examples vNIC, vLan, vSwitch (e.g. Open Stack Security Group)…
        - **SDN**: **separates the network's control (brains) and forwarding (muscle) planes and provides a centralized view of the distributed network for more efficient orchestration and automation of network services.** (sometimes called overlay network). In cloud different customers want to manage their own network as they would do on-prem. The market message is 'bring you own IP' .. of course those virtual IP resources and functions are then mapped to the cloud network infrastructure
        - **NFV**: focuses on optimizing the network services themselves. NFV decouples the network functions, such as DNS, caching, etc., from proprietary hardware appliances, so they can run in software to accelerate service innovation and provisioning, particularly within service provider environments.
    - Cloud Computing requires (hardware independent) APIs to (re-)configure start/stop and monitor Server IPs LBs, DNS, Firewall supporting multi-tenancy separation

https://www.sdxcentral.com/networking/sdn/definitions/what-is-overlay-networking/
https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html

Virtualization and Software Defined towards IaaS

- How to use all the virtualization and SDx capabilities in order to provide an IaaS Service
    - Deployment Automation (**Orchestrate all the various hypervisor types)**
        - A Server needs
            - (CPU, Memory) = Server Hypervisor
            - Storage = Storage Hypervisors (SDS)
            - IP Addresses, Firewalls = Network Hypervisors (SDN)
            - Resource Pooling (get a v-entity across physical entities)
            - Images
- Service Orientation
    - Self Service Portal (User Interface, GUI and API)
    - Security and Identity Management
    - Common API northbound to service abstraction (Templates, Offerings, Accounting Billing)

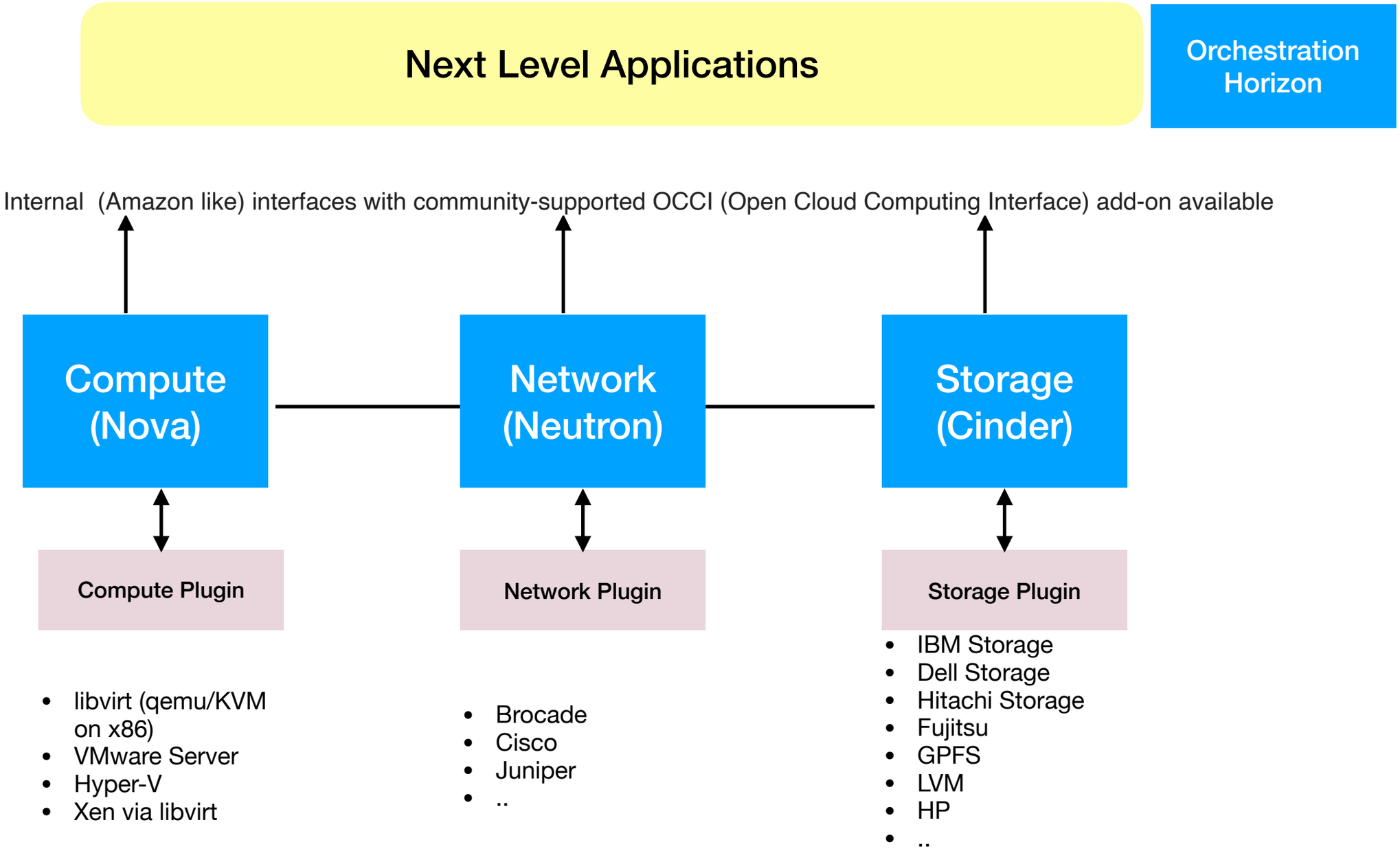
OpenStack

Open software to manage compute, network & storage resources

https://www.openstack.org

https://en.wikipedia.org/wiki/OpenStack

Open Stack is basically an API Layer

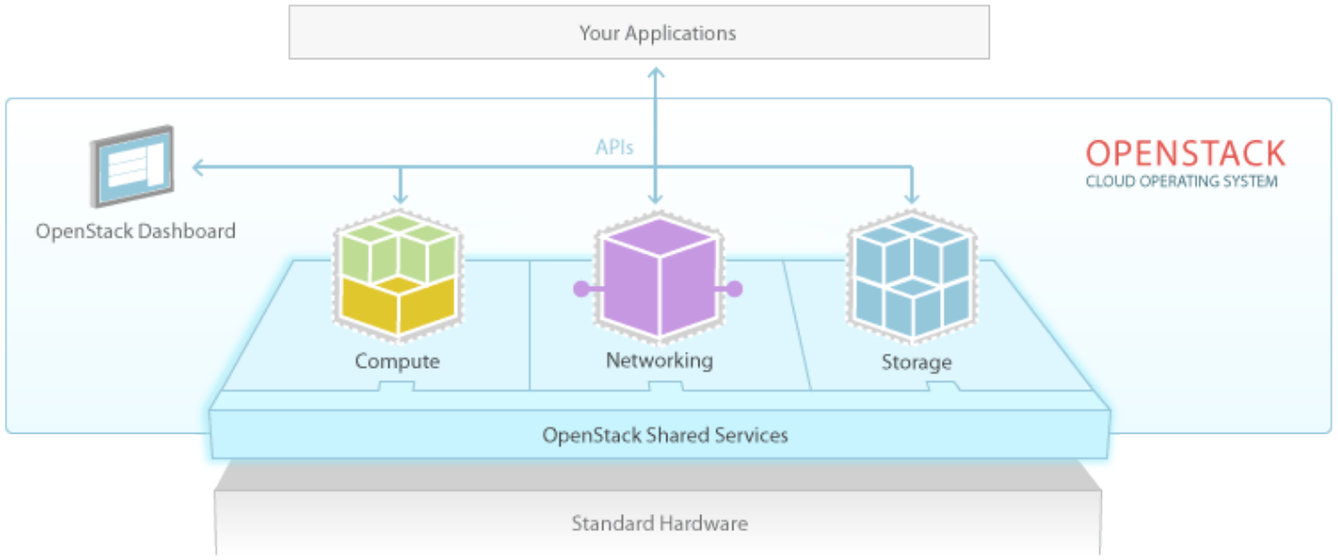


https://docs.openstack.org/api-ref/compute/?expanded=#create-server
https://docs.openstack.org/api-ref/network
https://docs.openstack.org/api-ref/block-storage/v3/?expanded=show-all-extra-specifications-for-volume-type-detail,create-a-volume-detail#volumes-volumes

Open Stack High Level Overview

Developed by an open community, OpenStack is able to control large pools of compute, storage & networking resources throughout a datacenter

https://wiki.openstack.org/wiki/ProjectTypes



OpenStack Compute (core)
Provision and manage large networks of virtual machines

OpenStack Network (core)
Network dynamically with a pluggable, scalable and API-driven system for managing networks and IP addresses.

OpenStack Storage (core)
Create petabytes of secure, reliable object or block storage using standard HW.

OpenStack Dashboard (core)
Enables administrators and users to access & provision cloud-based resources through a self-service portal.

OpenStack Shared Services

OpenStack Image service
Catalog and manage massive libraries of server images

OpenStack Identity service
Unified authentication across all OpenStack projects and integrates with existing authentication systems.

OpenStack Telemetry service
Aggregated usage and performance data across the services deployed in an OpenStack cloud

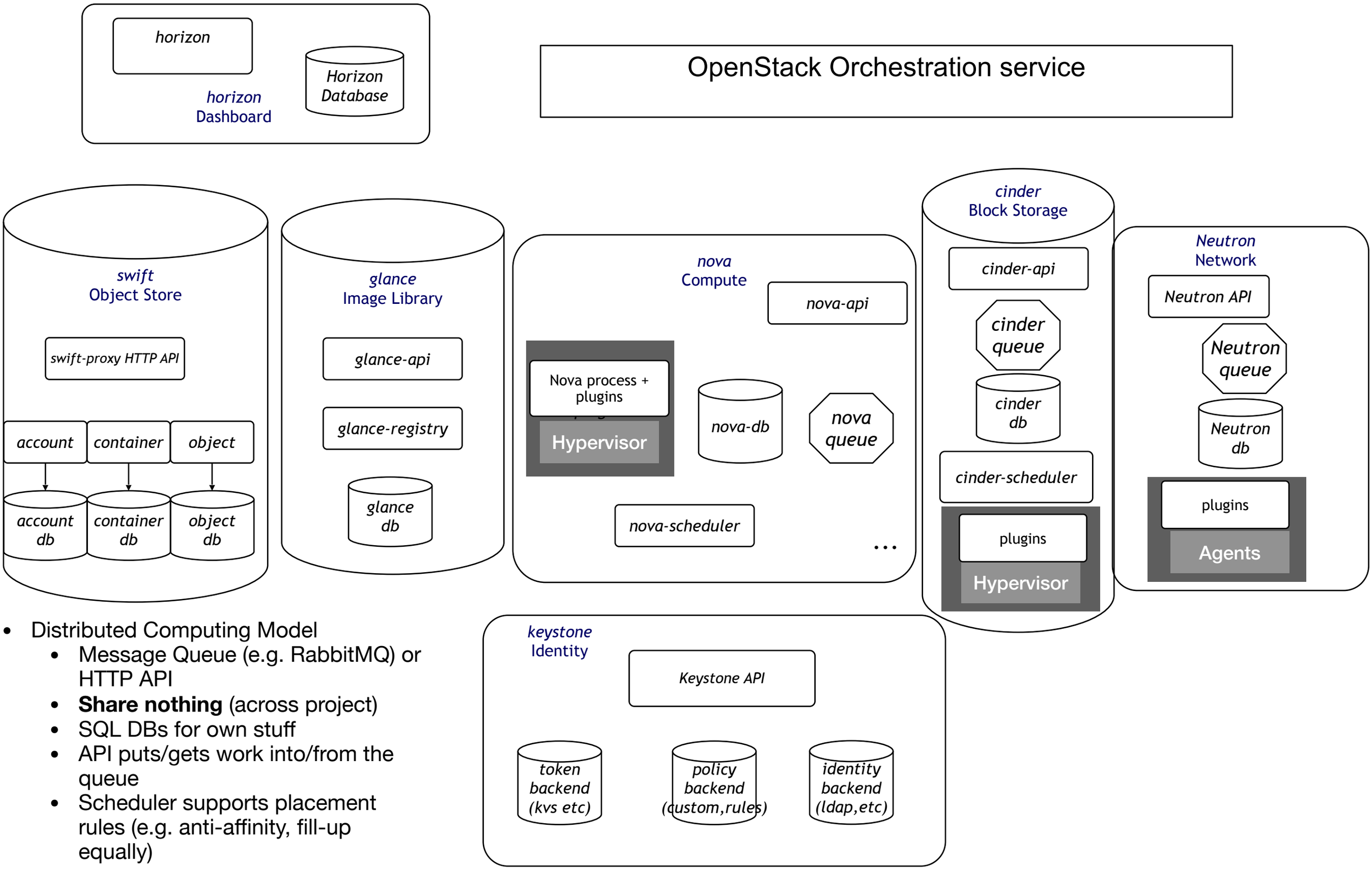
OpenStack Orchestration service
Template-driven engine that allows application developers to describe and automate the deployment of infrastructure

OpenStack Database service
Quickly and easily utilize the features of a relational database without the burden of handling complex administrative tasks

There are many more projects

- Integrated projects (core and shared) are those which will always be part of a release
- Official Open Stack are governed by the Open Stack Technical Audience

Open Stack Logical Architecture



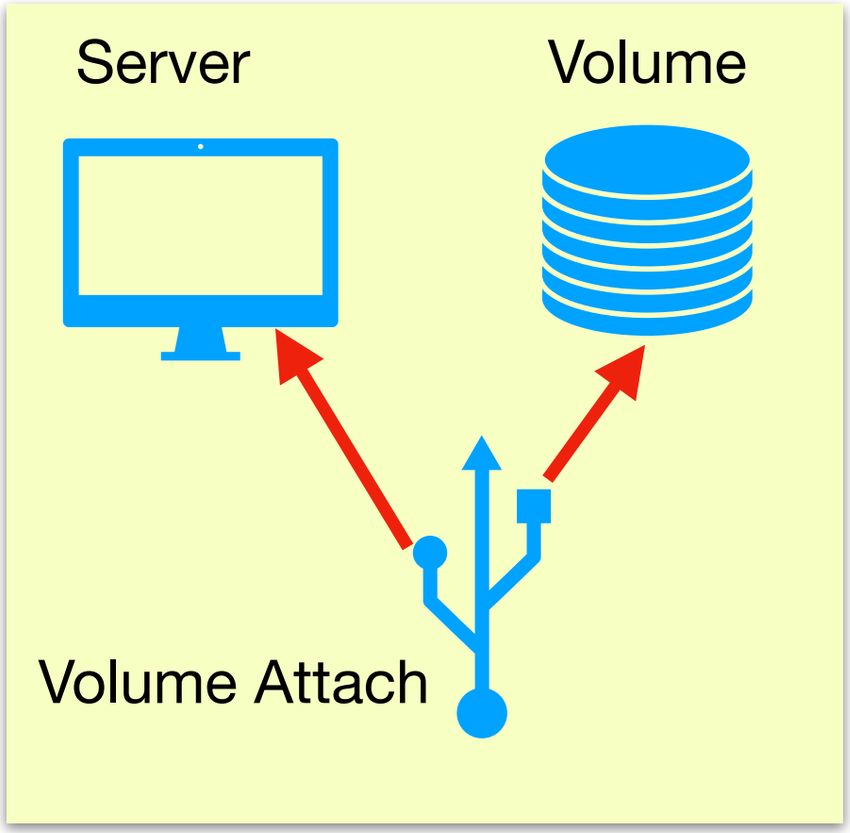
- Distributed Computing Model
  - Message Queue (e.g. RabbitMQ) or HTTP API
  - **Share nothing** (across project)
  - SQL DBs for own stuff
  - API puts/gets work into/from the queue
  - Scheduler supports placement rules (e.g. anti-affinity, fill-up equally)

Model ‘workloads’ with Open Stack

- Hot and Heat
  - Heat the ‘orchestration’ engine and Hot (**H**eat **o**rchestration **T**emplate) the template (YAML) to describe
    - the Open Stack resources and their configuration options
    - the dependencies among those (e.g. Network and Storage to a certain Server)
    - Compositions of resources belonging to one workload
- Companies working on a graphical representation of such resource topologies

A HoT (Template)

```yaml
heat_template_version: "2018-08-31"
description: "version 2017-09-01 created by HOT Generator at Sun, 14 Feb 2021 13:26:13 GMT."
resources:
  Server_1:
    type: "OS::Nova::Server"
    properties:
      security_groups:
        - "d86a0c82-7741-4f9a-a9e3-236355da2ec6"
      networks:
        - network: "c4eafee4-e84d-45ca-92a6-238302b75083"
      name: mytest
      flavor: "m1.small"
      image: "5cce79e9-14b3-47d6-95d8-95c62e99bef6"
      key_name: cloudbwkey2
  Volume_1:
    type: "OS::Cinder::Volume"
    properties:
      name: myVolume1
      size: 3
      volume_type: "4954db20-bddf-4f4d-bfa5-c5dd246efe45"
  VolumeAttachment_1:
    type: "OS::Cinder::VolumeAttachment"
    properties:
      instance_uuid: {get_resource: Server_1}
      volume_id: {get_resource: Volume_1}
    depends_on:
      - Server_1
      - Volume_1
```



Logical Dependencies:

- Volume Attach (which is modelled as Resource) needs the Server and the Volume up and running
- Volume and Server can be started and stopped independently
- Server and Volume cannot be stopped before the volume attach is down (detached)
- Dependencies can be expressed using properties in different objects (= resource). E.g. instance_uuid) or explicitly using the depends_on property..

Logical Grouping:

- The whole YAML (stack) can be started or stopped

Open Stack Release Cycles

https://releases.openstack.org/

Series	Status	Initial Release Date	Next Phase	EOL Date
Wallaby	Development	2021-04-14 *estimated* (schedule)	Maintained *estimated 2021-04-14*	
Victoria	Maintained	2020-10-14	Extended Maintenance *estimated 2022-04-18*	
Ussuri	Maintained	2020-05-13	Extended Maintenance *estimated 2021-11-12*	
Train	Maintained	2019-10-16	Extended Maintenance *estimated 2021-05-12*	
Stein	Extended Maintenance (see note below)	2019-04-10	Unmaintained *TBD*	
Rocky	Extended Maintenance (see note below)	2018-08-30	Unmaintained *TBD*	
Queens	Extended Maintenance (see note below)	2018-02-28	Unmaintained *TBD*	
Pike	Extended Maintenance (see note below)	2017-08-30	Unmaintained *TBD*	
Ocata	Extended Maintenance (see note below)	2017-02-22	Unmaintained *estimated 2020-06-04*	
Newton	End Of Life	2016-10-06		2017-10-25
Mitaka	End Of Life	2016-04-07		2017-04-10
Liberty	End Of Life	2015-10-15		2016-11-17
Kilo	End Of Life	2015-04-30		2016-05-02
Juno	End Of Life	2014-10-16		2015-12-07
Icehouse	End Of Life	2014-04-17		2015-07-02
Havana	End Of Life	2013-10-17		2014-09-30
Grizzly	End Of Life	2013-04-04		2014-03-29
Folsom	End Of Life	2012-09-27		2013-11-19

Deliverable Rocky (implied Projects)

- Rocky
  - Service Projects
    - barbican
    - blazar
    - cinder
    - congress
    - cyborg
    - designate
    - freezer
    - glance
    - heat
    - horizon
    - ironic
    - keystone
    - manila
    - masakari
    - mistral
    - monasca-api
    - murano
    - neutron
    - nova
    - octavia
    - qinling
    - sahara
    - searchlight
    - senlin
    - solum
    - storlets
    - swift
    - trove
    - vitrage
    - watcher
    - zaqar
  - Service Client Projects
    - python-aodhclient
    - python-blazarclient
    - python-brick-cinderclient-ext
    - python-cinderclient
    - python-congressclient
    - python-cyborgclient
    - python-freezerclient
    - python-glanceclient
    - python-heatclient
    - python-ironic-inspector-client
    - python-ironicclient
    - python-keystoneclient

Service Projects

Release Summary

Deliverable	Earliest Version	Most Recent Version	Stable Status	Notes
barbican	7.0.0.0b1	7.0.0.0b2	Development	
blazar	2.0.0.0b1	2.0.0.0b2	Development	
cinder	13.0.0.0b1	13.0.0.0b2	Development	
congress	8.0.0.0b1	8.0.0.0b2	Development	
cyborg	1.0.0.0b1	1.0.0.0b2	Development	
designate	7.0.0.0b1	7.0.0.0b2	Development	
freezer	7.0.0.0b1	7.0.0.0b1	Development	
glance	17.0.0.0b1	17.0.0.0b2	Development	
heat	11.0.0.0b1	11.0.0.0b2	Development	
horizon	14.0.0.0b1	14.0.0.0b2	Development	
ironic	11.0.0	11.0.0	Development	
keystone	14.0.0.0b1	14.0.0.0b2	Development	
manila	7.0.0.0b1	7.0.0.0b2	Development	
masakari	6.0.0.0b1	6.0.0.0b2	Development	
mistral	7.0.0.0b1	7.0.0.0b2	Development	
monasca-api	2.6.0	2.6.0	Development	
murano	6.0.0.0b1	6.0.0.0b1	Development	
neutron	13.0.0.0b1	13.0.0.0b2	Development	
nova	18.0.0.0b1	18.0.0.0b2	Development	release notes
octavia	3.0.0.0b1	3.0.0.0b2	Development	release notes
qinling	1.0.0.0b1	1.0.0.0b2	Development	
sahara	9.0.0.0b1	9.0.0.0b2	Development	
searchlight	5.0.0.0b2	5.0.0.0b2	Development	
senlin	6.0.0.0b1	6.0.0.0b2	Development	
solum	5.6.0	5.6.0	Development	
storlets	2.0.0.0b1	2.0.0.0b2	Development	
swift	2.18.0	2.18.0	Development	
trove	10.0.0.0b1	10.0.0.0b1	Development	
vitrage	3.0.0	3.0.0	Development	
watcher	1.9.0	1.10.0	Development	
zaqar	7.0.0.0b1	7.0.0.0b1	Development	

Get you hands on bwCloud (using Openstack)

Manual Setup of a Linux Server in our IaaS bwCloud

- See : /Übungen/Deploy a virtual Server in the bwCloud

- Remarks:
  - It is almost all supported via a GUI (Horizon), the critical part left is to build and use SSH keys in order to be able to remotely access your system
  - Let’s see how the Security Groups work

First we can demo this case ……

Terraform (Scripting Infrastructure)

A different approach to Open Stack for managing infrastructure

HashiCorp

Terraform

Desired

Resource Configuration Properties

Including properties from other resources

Plan the next step to reach the desired state

(Considering operational dependencies)

Current

Resource Configuration Properties

Currently monitored

Including properties from other resources

Get the current state

Run activate or deactivate

Get current state (Active, inactive, unknown, taint)	Activate / Deactivate

A managed **Resource** can be anything which has a Software Interface to

- activate / deactivate
- get the current state

No additional modelling of the resource. This means that every resource has a different set of properties. The needed configuration represent the resource

Terraform (Scripting Infrastructure)

https://www.terraform.io/docs/index.html

- Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently.

- Infrastructure as Code

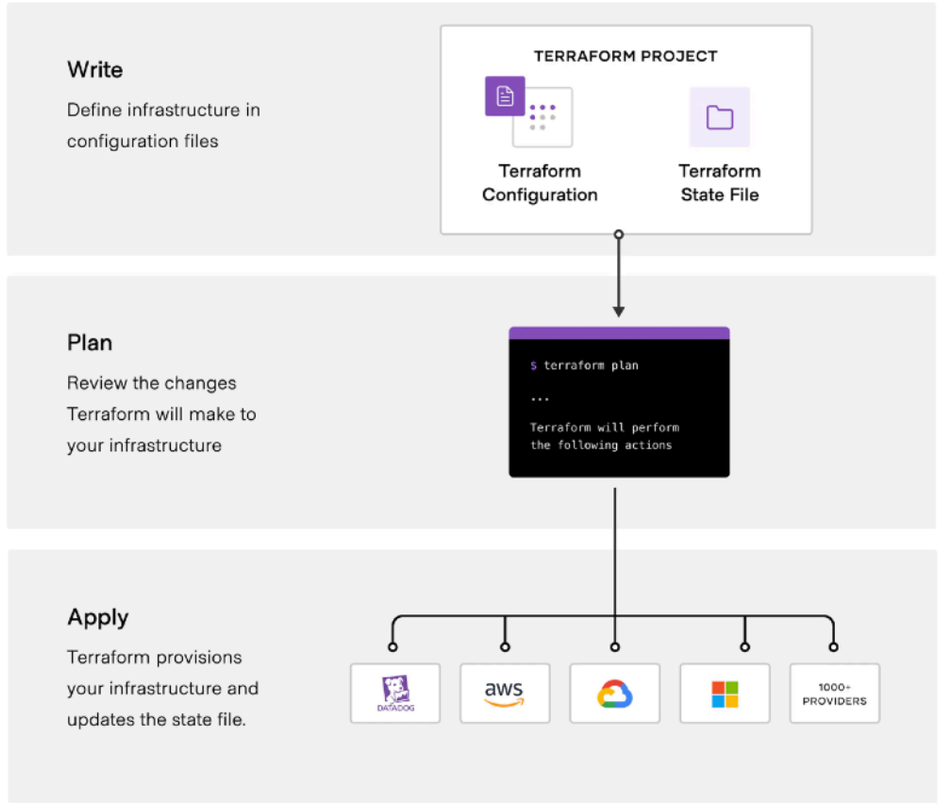
    - Infrastructure is described (declarative) using a high-level configuration syntax. This allows a blueprint of your datacenter to be versioned and treated as you would any other code **(the desired state)**

- Terraform has a **"planning"** step where it generates an ***execution*** *plan*. The execution plan shows what Terraform will do **to reach the desired state from the actually state.**

- Terraform builds a graph of all your resources, and **parallelizes the creation** and modification of any non-dependent resources. Because of this, Terraform builds infrastructure as efficiently as possible, and operators get insight into **dependencies** in their infrastructure.

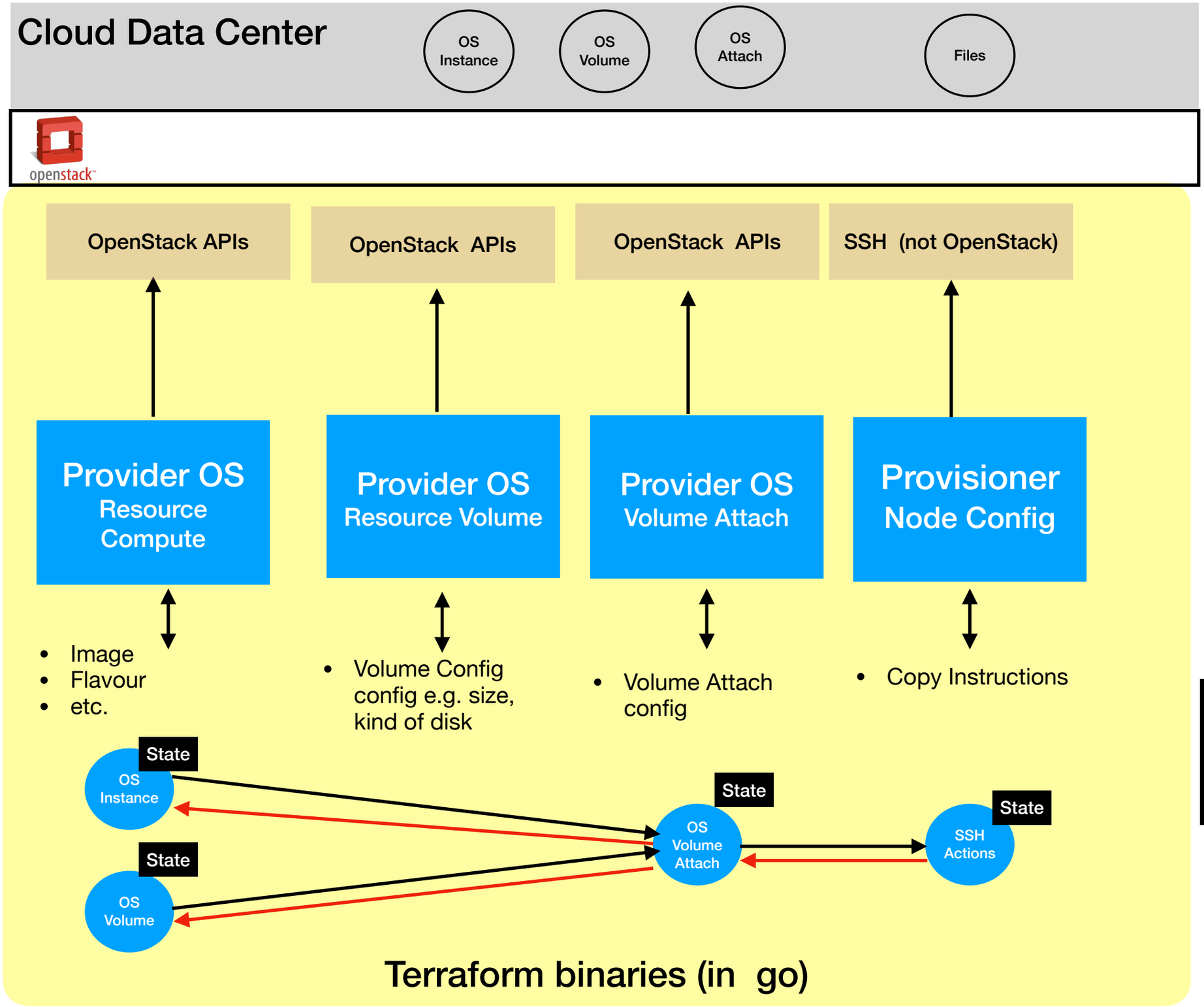
Terraform (the many providers)

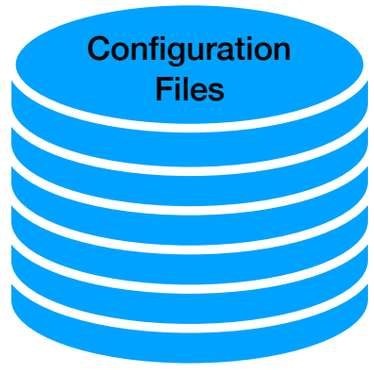
- Terraform does not change the data model of the managed infrastructure
- It uses what is available and provide a configuration syntax
- It assumes a IT infrastructure resource (modelled as objects) can be started, stopped and queried
- There are many providers (code often come from the managed infrastructure resource vendor)



> 1000 Providers
https://registry.terraform.io/browse/providers
https://registry.terraform.io/providers/hashicorp/aws/latest/docs
https://registry.terraform.io/providers/terraform-provider-openstack/openstack/latest/docs

Terraform (Configure the Infrastructure beyond a single resource)

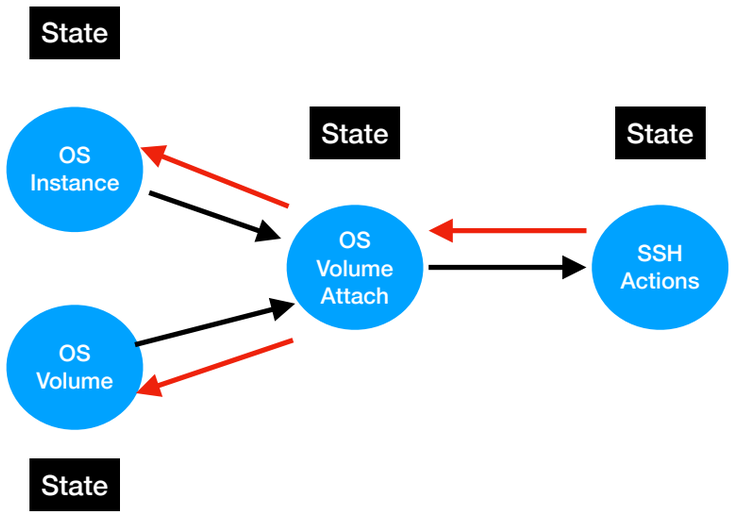




abc.tf config (desired State)

Terraform.tfstate
(current state)

Terraform (Dependency and State Management)



Dependencies expressed

- **Implicit** : a dependent resources requires a property of a supporter resources (e.g. IP address)
- **Explicit** : via the key word depends_on = [resource1, resource2]

Apply Infrastructure (Activate Resources)

Destroy Infrastructure (Delete Resources)

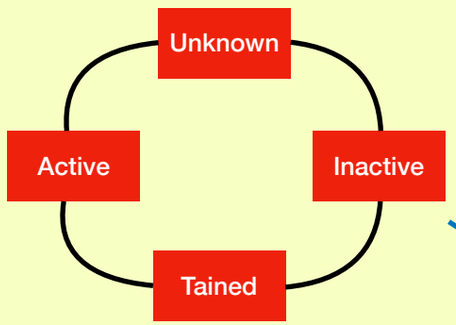
Local Computer (managed from)

Infrastructure as code = desired State

All .tf files of a directory

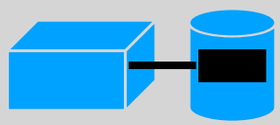
Data Query (For large selections)

Terraform Representation of current state



Infrastructure (managed to)

(real) infrastructure current state



Synch

$ terraform refresh

Triggers (E.g. changes on local file)

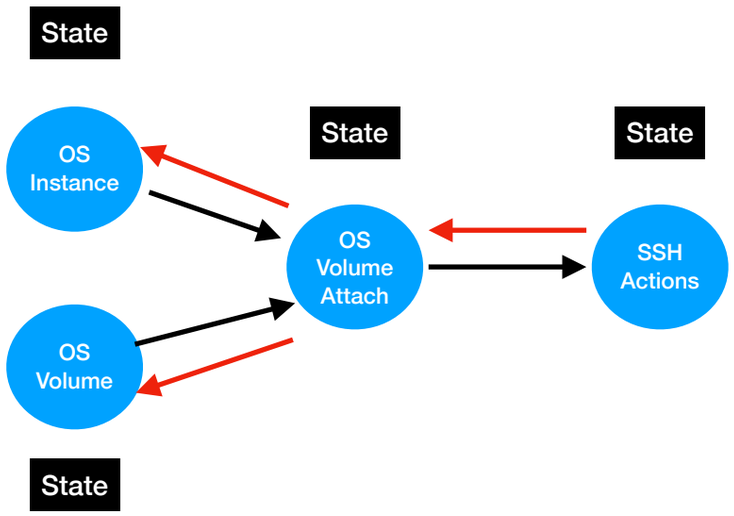
$ terraform apply

Build and propose a plan how to reach the desired state based on the currently refreshed current state

- Activate a resource because
  - Resource is defined but currently not active or not existing in the real infrastructure
  - It has been changed in the desired config in a way that it must be restarted. (can not be modified in place)
  - A trigger has changed that would require a reactivation
- Activate all selected resources according their dependency
- Change a resource, because this change can be done without a reactivation (e.g. rename of a volume name)
- Destroy a resource because the resource has been removed from the desired configuration. This could also mean to destroy all dependent resource. (e.g. destroy a volume would also destroy the attachment resource)
- A **$ terraform destroy** would destroy the whole configuration.
- A **$ terraform init** is our initial command before the first activation. It initialise the local setting

Terraform Example

Basic Ubuntu 22.04 Image with a secondary disk volume attached, formatted and mounted



Hint: /bwcloud/BaseUbunutu

```
#-------------------------------------------------------------------------
# Create a basic Ubuntu.. attention security group default is not
sufficient to even allow ssh
# as during boot additional packages are loaded
#
-------------------------------------------------------------------------
resource "openstack_compute_instance_v2" "myUbuntu" {
  name            = "myUbuntu"
  image_id        = "2e170e26-3749-4326-b716-437379c9ba93"
  flavor_name       = "m1.small"
  key_pair        = "cloudkey"
  security_groups = ["default","OpenDNS"]

  metadata = {
    this = "if this work, that would be wonderful"
  }
  network {
    name = "public-belwue"
  }
}
```

```
# -------------------------------------------------------------------------
#  Get an additional volume (beside the Ubuntu Image)
# -------------------------------------------------------------------------
resource "openstack_blockstorage_volume_v2" "volume_1" {
  name = "mySecondDataVolume"
  size = 1
}
```

```
# -------------------------------------------------------------------------
#  Attach the volume if both the volume and the system Instance is
available
#  We use an implicit dependency to have this attachment done
-------------------------------------------------------------------------
resource "openstack_compute_volume_attach_v2" "attach-1" {
  instance_id = "${openstack_compute_instance_v2.myUbuntu.id}"
  volume_id   = "${openstack_blockstorage_volume_v2.volume_1.id}"
}
```

```
# -------------------------------------------------------------------------
# Access this volume format it and mount it to /mydata/
# we use a implicit dependency as the attach returned device name
(dev/vdb)
# -------------------------------------------------------------------------
resource "null_resource" "volumeformating" {
 connection {
   type = "ssh"
    host = openstack_compute_instance_v2.myUbuntu.access_ip_v4
    user = "ubuntu"
    port = 22
    private_key = file("/Users/juergenschneider/Documents/
webpages/DHBW/Cloud2022/InternalStuff/BWCloud/cloud.key")
  }
  provisioner "remote-exec" {
      inline = [
      "sudo mkfs -t ext4 $
{openstack_compute_volume_attach_v2.attach-1.device}",
      "sudo mkdir /mydata",
      "sudo mount $
{openstack_compute_volume_attach_v2.attach-1.device} /mydata",
      "sudo chmod ugo+rwx /mydata",
    ]
 }
}
```

Get your Hands on Terraform

Automated Setup of a Linux Server in our IaaS bwCloud

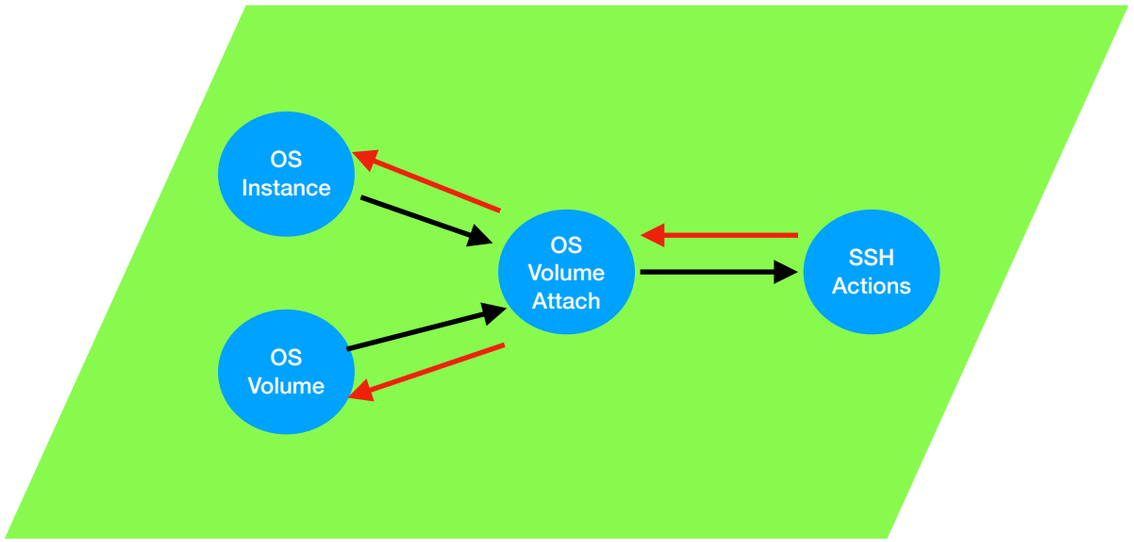
See : /Übungen/Terraform and bwCloud

Or

Automated Setup of Linux Server in a Amazon or Azure Cloud

See : /Übungen/Terraform and Amazon or /Übungen/Terraform and AWS

Recap: Terraform Infrastructure as Code



Reale Infrastructure

Desired State

Current State

Terraform Recap ..

- Terraform has two type of resources
  - A **full resource** completely managed by a provider software
    - Knows what to do to reach the desired state
    - Knows what to do when destroyed
    - E.g. https://registry.terraform.io/providers/hashicorp/aws/latest/docs
  - A null_resource to be used when a SSH based connection is needed for file transfer or CLI commands
    - Execute the file copy or commands for activation
    - Is a no-op for the destroy request (has no undo logic, e.g if you destroy only the server, this resource may still be active)
    - A null-resource can execute commands on the target system but also on the local system.
      - A local system command itself can be a Ansible Python script doing magic things on the remote system
- All the Terraform logic (written in Go) is a single binary.
  - All provider logic is installed during the **$ terraform init.** The needed Providers are specified in the .tf file
- A **$ terraform apply** will collect all .tf files of the acting folder and all included subfolders

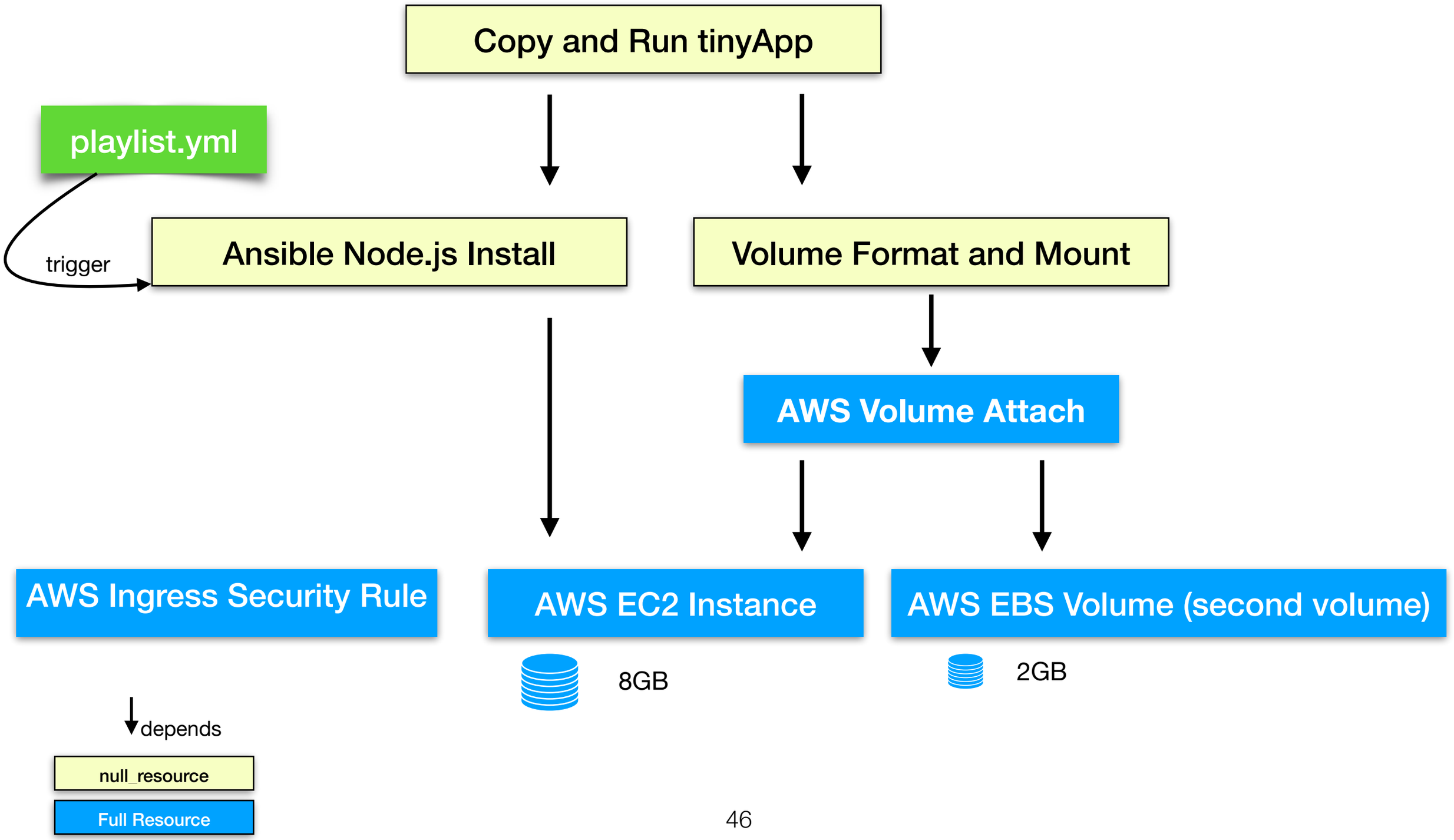
Terraform Recap ..

- Dependencies between Terraform Resources are:
  - **implicit**, if the dependent resource requires a property of the supporting resource
  - **explicit,** if explicitly declared **depends_on** in a depending resource.
    - Makes only sense in null_resources, in other cases rethink ;-)
  - This dependency is considered for activation (apply) and deactivation (destroy)
- When is a Plan Change ?
  - Potential plan changes are discovered during **$ terraform apply, plan**
  - A full resource provider :
    - can query the target infrastructure to explore changes
    - detects if the .tf configuration has changed (e.g new property)
  - A null_resource can define triggers to reflect a change
    - A trigger can be a hash of file content (which is used in in the null_resource)
  - Changes may require a re-activation or an update in place (full resource only)

Terraform Recap ..

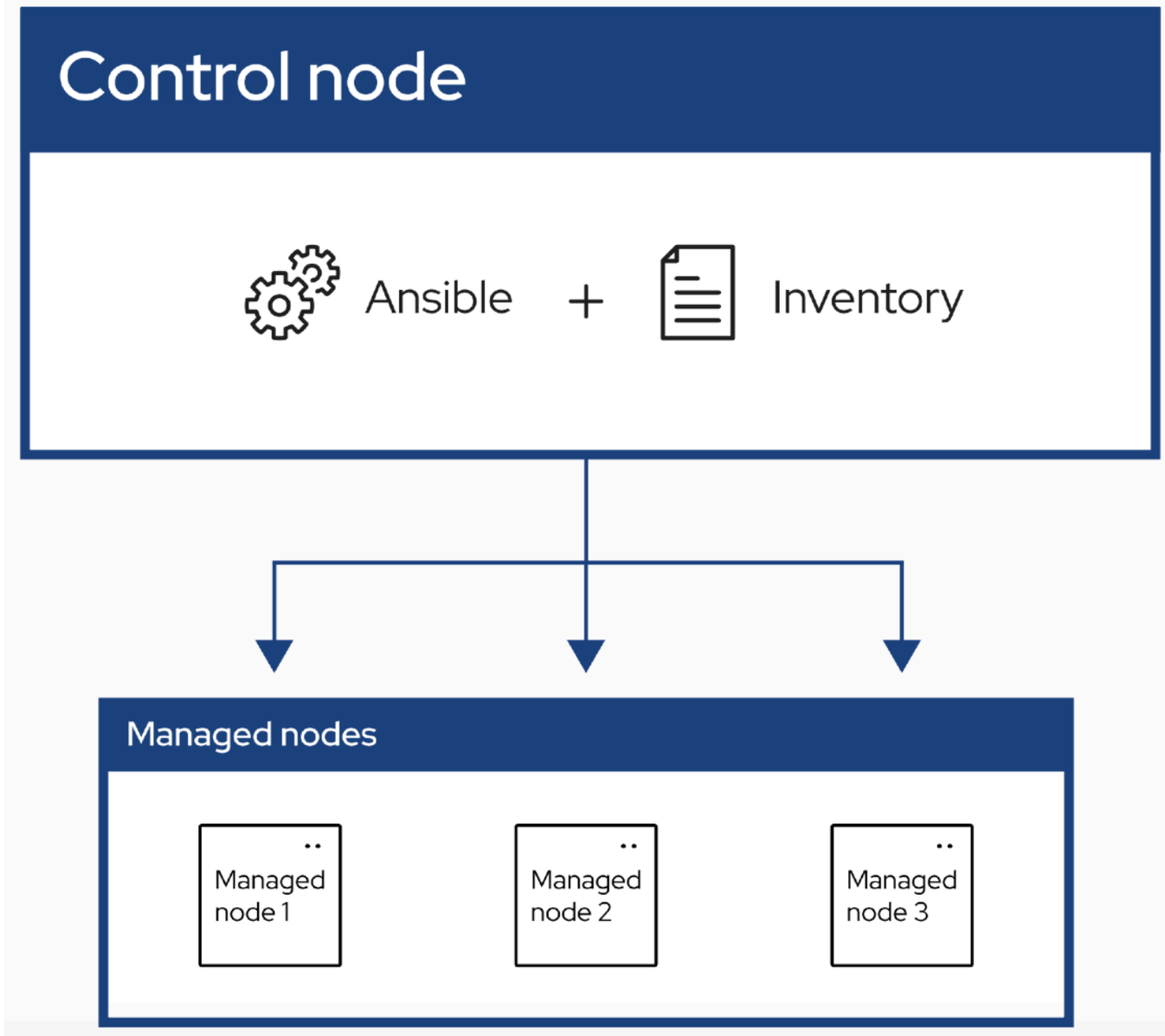
- A third resource is the **data source** resource.
  - Data Source are supported by providers
  - Data source is a coding tool to retrieve complex properties by filtering the right property name needed for a real resource.
    - e.g. get the best AMI image out of 700 possibilities
    - e.g. the property needed is generated by the target infrastructure but I have a simple name. (VPC security group)
- Often used **variables** can be set once and referenced by the variable name.
- For all this we have a complete example in our Nextcloud Lösungen/AWSCloud or Lösungen/Azure or Lösungen/bwCloud

Node.js Web App Deployment



Ansible - Introduction

https://docs.ansible.com/ansible/latest/getting_started/index.html



Control Node
A system on which Ansible is installed.

Inventory
A list of managed nodes that are logically organized. You create an inventory on the control node to describe host deployments to Ansible.

Managed node
A remote system, or host, that Ansible controls.

Agent-less architecture
Low maintenance overhead by avoiding the installation of additional software across IT infrastructure.

Simplicity
Automation playbooks use straightforward YAML syntax for code that reads like documentation. *Ansible is also decentralized, using SSH with existing OS credentials to access to remote machines.*

Scalability and flexibility
Easily and quickly scale the systems you automate through a modular design that supports a large range of operating systems, cloud platforms, and network devices.

Idempotence and predictability
When the system is in the state your playbook describes Ansible does not change anything, even if the playbook runs multiple times.

Ansible Concepts

Playbooks or Roles

- They contain Plays (which are the basic unit of Ansible execution). Roles are reusable assets which can run in different playbooks.
- Playbooks or Roles are written in YAML and are easy to read, write, share and understand.

Plays

- The main context for Ansible execution. The Play contains variables, ordered lists of tasks and can be run repeatedly. It basically consists of an implicit loop over the mapped hosts and tasks and defines how to iterate over them.

Tasks

- The definition of an ‘action’ to be applied to the managed host.

Handlers

- A special form of a Task, that only executes when notified by a previous task which resulted in a ‘changed’ status.

Modules

- The code or binaries that Ansible copies to and executes on each managed node (when needed) to accomplish the action defined in each Task.
- Ansible modules are grouped in collections.

Plugins

- Pieces of code that expand Ansible’s core capabilities. Plugins can control how you connect to a managed node (connection plugins), manipulate data (filter plugins) and even control what is displayed in the console (callback plugins).

Collections

- A format in which Ansible content is distributed that can contain playbooks, roles, modules, and plugins. You can install and use collections through Ansible Galaxy.

Ansible Example 1

In this example, the first play targets the web servers; the second play targets the database servers.

```
---
- name: Update web servers
  hosts: webservers
  remote_user: root

  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest

  - name: Write the apache config file
    ansible.builtin.template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

- name: Update db servers
  hosts: databases
  remote_user: root

  tasks:
  - name: Ensure postgresql is at the latest version
    ansible.builtin.yum:
      name: postgresql
      state: latest

  - name: Ensure that postgresql is started
    ansible.builtin.service:
      name: postgresql
      state: started
```

Desired state and ‘idempotency’

Most Ansible modules check whether the desired final state has already been achieved, and exit without performing any actions if that state has been achieved, so that repeating the task does not change the final state. Modules that behave this way are often called ‘idempotent.’ Whether you run a playbook once, or multiple times, the outcome should be the same.

Ansible Example 2-1

```
##################################################
# DO Community Playbooks: Node.js
##################################################
---
- hosts: all
  become: true
  gather_facts: no
  tasks:
    - name: Wait 600 seconds for target connection to become reachable/usable
      ansible.builtin.wait_for_connection:
    - name: Install GPG
      tags: nodejs, install, setup
      apt:
        name: gnupg
        update_cache: yes
        state: present

  - name: Install the gpg key for nodejs LTS
    apt_key:
      url: "https://deb.nodesource.com/gpgkey/nodesource.gpg.key"
      state: present
```

Ansible Example 2-2

```
- name: Install the nodejs LTS repos
  apt_repository:
    repo: "deb https://deb.nodesource.com/node_18.x focal main"
    state: present
    update_cache: yes

- name: Install NodeJS
  tags: nodesjs, install
  apt:
    name: nodejs
    state: latest
- name: Install PM2
  ansible.builtin.command: npm install -g pm2

- name: Install WebServer
  ansible.builtin.command: npm install
  args:
    chdir: /home/ubuntu/work/proxy/

- name: Run WebServer
  ansible.builtin.command: pm2 start proxy.js -f
  args:
    chdir: /home/ubuntu/work/proxy/
```

VM Images

- VM Image —> Systemabbild
- Images are the data needed to activate and run a VM
- Data is typically the content of the primary (bootable disk).
    - **Image Template** = A Image w/o instance data (golden master) from which you create (instantiate) new VMs. Templates can be generated from snapshots. Template Clones are just copies of templates
    - **(Image Snapshot)** = A point in time copy of running VM volume(s)
        - normally no meta data and not registered in in a image registry
        - is a backup of a individual running VM volume and restored back to rollback
        - but can also be used to generate a image template again
            - In the bwCloud you can create a snapshot and re-instantiate it (run the complete dialog with new IP, Security, etc,)
            - AWS UI —> create image from snapshot
- Meta Data is needed to describe the content and the deployment requirements of a Image Template
- Could include Middleware and Application software
- Could reflect even a bare metal systems
- Is a full ‘asset’ in Virtualization Management
    - Lifecycle Management (e.g how to apply important security patches to 700+ potential images)

VM Image Data

- Meta Data visible in the image registry (Openstack Glance)
  e.g. Standard .OVF (Open Virtualization Format, DMTF based)
    - The location and format of all disk image**s** needed (the format of the disk file**s** depends on the provider e.g VMWare)
    - General Description for a better Selection in the Image Registry
    - Operating System (Type and Version) Architecture
    - Description how to use it
    - Potential Licenses Agreements
    - Required Hardware Capacity (vCPU, Memory, Disk, Network (# of vNIC etc)

- OVA (Open Virtualization Archive, DMTF) is an OVF file packaged together with all of its supporting files (disk images, etc.) as TAR. (Unix Tape Archive format)

- Disk File Formats. (Not a complete list)

    - VDI – VirtualBox's internal disk image format
    - VMDK – One of the most common formats. VMWare's products use various versions and variations of VMDK disk images.
    - OSI Disk Images for optical Disks
    - AMI ARI. Amazon
    - VHD Azure
    - raw (.img, .raw, etc.) – Without compression, disk images can be very large, but sometimes converting to raw disk images might make sense as an intermediate step or in certain scenarios for better performance at the cost of space.

- Open Stack Glance (Repository for the Image Meta Data) https://docs.openstack.org/glance/rocky/user/formats.html
    - All of above with a priority order of implementations (a certain focus on AWS)
        - Disk Format all above and aki (Amazon kernel image), ami (Amazon machine image,) ari (Amazon ramdisk image) and ISO
        - OVF is supported in some way
        - https://specs.openstack.org/openstack/glance-specs/specs/mitaka/implemented/ovf-lite.html

HashiCorp Packer

- Used to pack/build Images by
  - **deploying a base image (template)**
  - applying software updates
  - registering and storing the data as a new image

```
{
  "variables": {
    "aws_access_key": "",
    "aws_secret_key": ""
  },
  "builders": [{
    "type": "amazon-ebs",
    "access_key": "{{user `aws_access_key`}}",
    "secret_key": "{{user `aws_secret_key`}}",
    "region": "eu-central-1",
    "source_ami_filter": {
      "filters": {
        "virtualization-type": "hvm",
        "name": "ubuntu/images/*ubuntu-xenial-16.04-amd64-server-*",
        "root-device-type": "ebs"
      },
      "owners": ["099720109477"],
      "most_recent": true
    },
    "instance_type": "t2.micro",
    "ssh_username": "ubuntu",
    "ami_name": "node-js-Spiele {{timestamp}}"
  }],
```

- Update and rebuild golden Master (Security Patches)
- Golden Master with Software (e.g. LAMP) —> PaaS

HashiCorp Packer

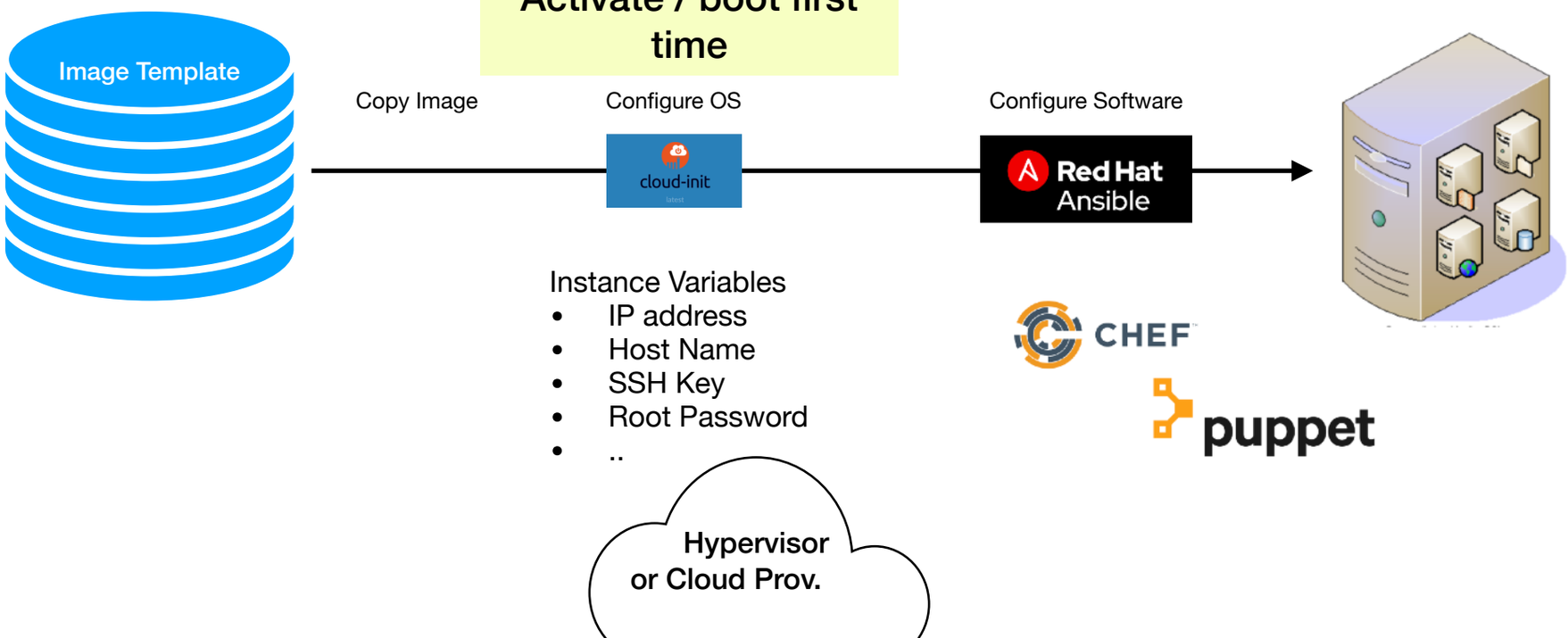
- Used to pack/build Images by
  - deploying a base image
  - **applying software updates**
  - **registering and storing the data as a new image**


```
"provisioners": [
  {
  "type": "shell",
   "inline": [
    "sleep 40",
    "mkdir work",
    "chmod ugo+rwx work"
    ]
  },
  {
   "type": "file",
   "source": "/Users/juergenschneider/Documents/webpages/DHBW/Cloud2019/InternalStuff/SpieleApp/",
   "destination": "/home/ubuntu/work/"
  },
  {
  "type": "shell",
  "inline": [
     "sudo apt-get install curl",
     "curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -",
     "sudo apt-get --assume-yes install nodejs",
     "sudo npm install pm2 -g",
     "cd /home/ubuntu/work",
     "sudo npm install"
   ]
}]
```

The problems with Images ??

- Images are large as they contain the full OS stack (including a software stack)
    - It may take a long time to copy (from a Master Image) during deployment (optimization can be applied, but at the end there is a copy process)
- A Image may contain all potential software which only a subset gets configured, for a particular deployment.
- Images are fully deployed (Docker images are incremental)
- Proliferation of too many different images (different OS and OS version and the software versions) was perceived as a management challenge
- Too many (not standardised) formats made conversion necessary
- Problems are OK and understood.. —> Images work

Cloud-Init Bootstrap



https://cloudinit.readthedocs.io/en/latest/index.html

- Cloud-init is the **industry standard** multi-distribution method for **cross-platform cloud instance initialization**. (no windows support)

- It is supported across all major public cloud providers, provisioning systems for private cloud infrastructure, and bare-metal installations.

- **Cloud instances are initialized from a disk image and instance data in various steps during boot**:
  - Metadata (the data you give defining the Server Instance, e.g Network Security Keys)
  - User data (optional, typically a set of actions given by the user, see example next page)
  - Vendor data (optional, additional set of actions given by the cloud provider)

Cloud-Init und Terraform

Terraform transfers the User Data file to the cloud instance during deployment

- Specify ssh keys, users, groups and potential installation package in a yml file

```
# ------------------------------------------------------------------------------------------------------
#  Define a cloud-init YAML to install docker (docker.io) and docker compose (docker-compose)
# ------------------------------------------------------------------------------------------------------
data "template_file" "user_data" {
  template = file("./cloud_init.yaml")
}
# ------------------------------------------------------------------------------
# Create a Docker Host
# ------------------------------------------------------------------------------
resource "openstack_compute_instance_v2" "Docker" {
  name            = "Docker"
  image_id        = "2e170e26-3749-4326-b716-437379c9ba93"
  flavor_name     = "m1.small"
  key_pair        = "cloudkey"
  security_groups = [openstack_networking_secgroup_v2.DockerSec.id]
  user_data       =  data.template_file.user_data.rendered
  metadata = {
    this = "if this work, that would be wonderful using cloud Init"
  }
}
```

Parts of a .tf file

Refer to cloud_init.yaml as part of the user data property of the Openstack Instance

```
groups:
  - dhbw

# Add user juergen in group dhbw and docker.
users:
  - default
  - name: juergen
    gecos: dhbw Cloud2023
    shell: /bin/bash
    primary_group: dhbw
    sudo: ALL=(ALL) NOPASSWD:ALL
    groups: users, admin, dhbw, docker
    lock_passwd: false
    ssh_authorized_keys:
      - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAAB
```

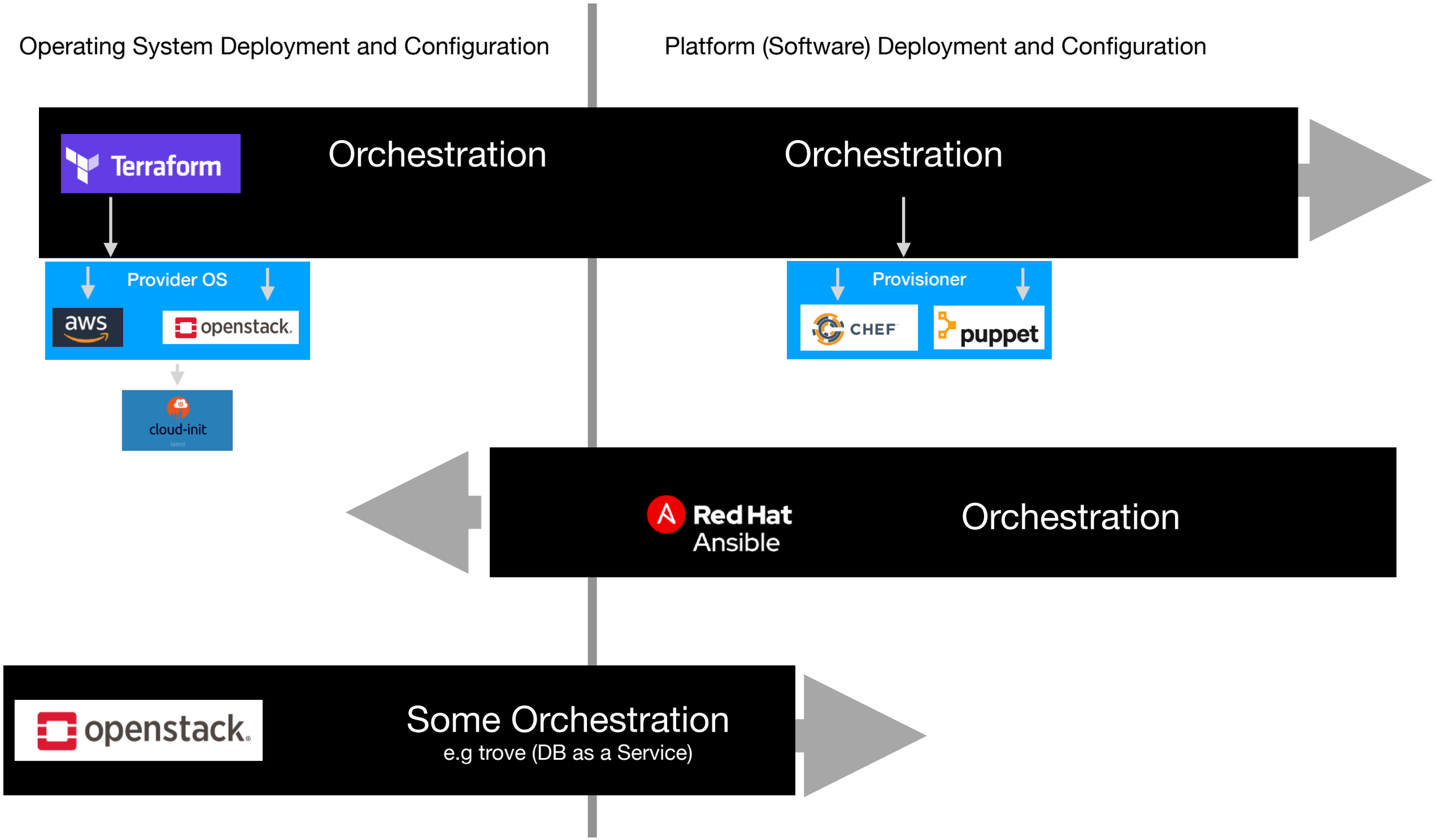
Cloud-init.yml as User Data to define

- juergen as user of the group dhbw

The problem : Terraform continues with the next resource, while those user-data processes continue to run

Cloud Deployment Tooling

How the different tooling relate (overlap)

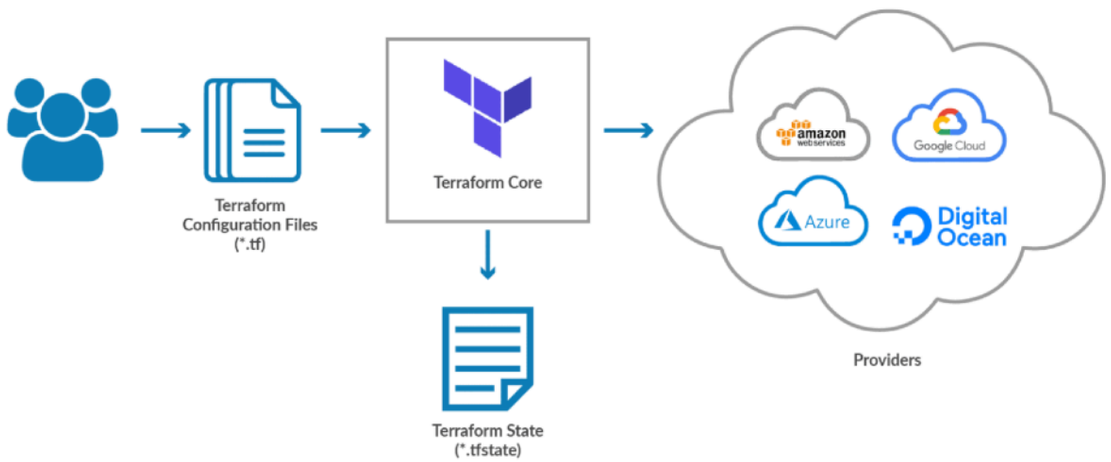


Terraform and Ansible

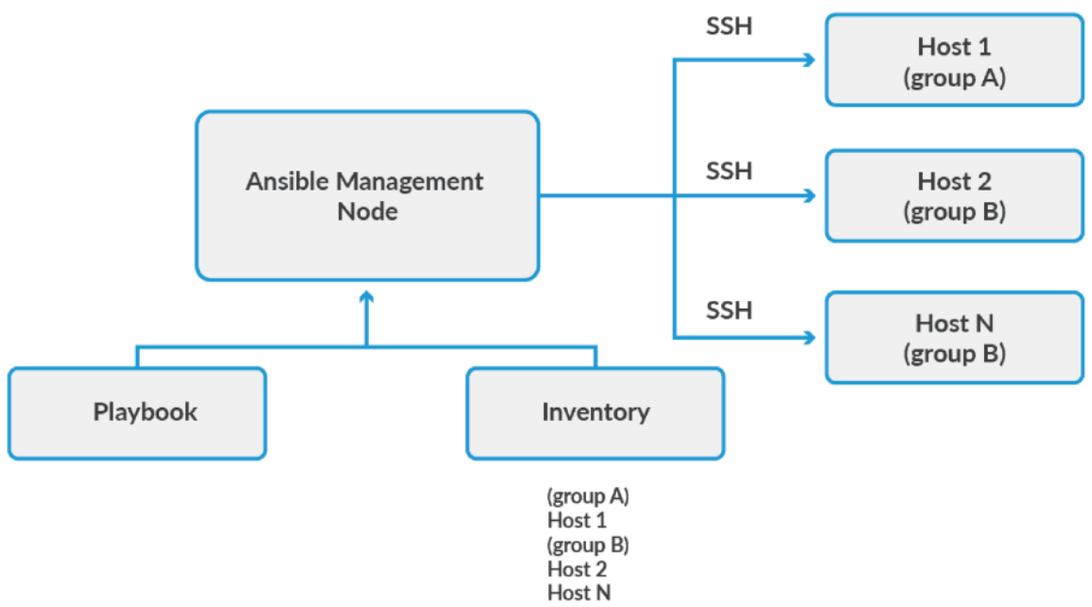
https://k21academy.com/ansible/terraform-vs-ansible/

https://www.ansible.com/integrations/cloud/amazon-web-services
https://docs.ansible.com/ansible/latest/collections/amazon/aws/ec2_instance_module.html#examples

Terraform : OS Deployment, Configuration



Ansible : Software Installation and Configuration



Declarative Approach : Resources as Objects with Properties (States)
Desired State drives next actions based on changes in the configurations (.tf) and the environments (if that can be detected)
Providers for certain resources provide the necessary interfaces (e.g. Openstack API) often as REST
SSH is used only for simple CLI execution

Ansible is **agentless** and doesn't run on target nodes. It makes **connections using SSH** or other authentication methods. It installs various **Python modules** on the target using JSON. These modules are simple instructions that run on the target. These modules are executed and removed once their job is done
Procedural and Declarative (Playbooks) Approach - no general state and dependency management Management

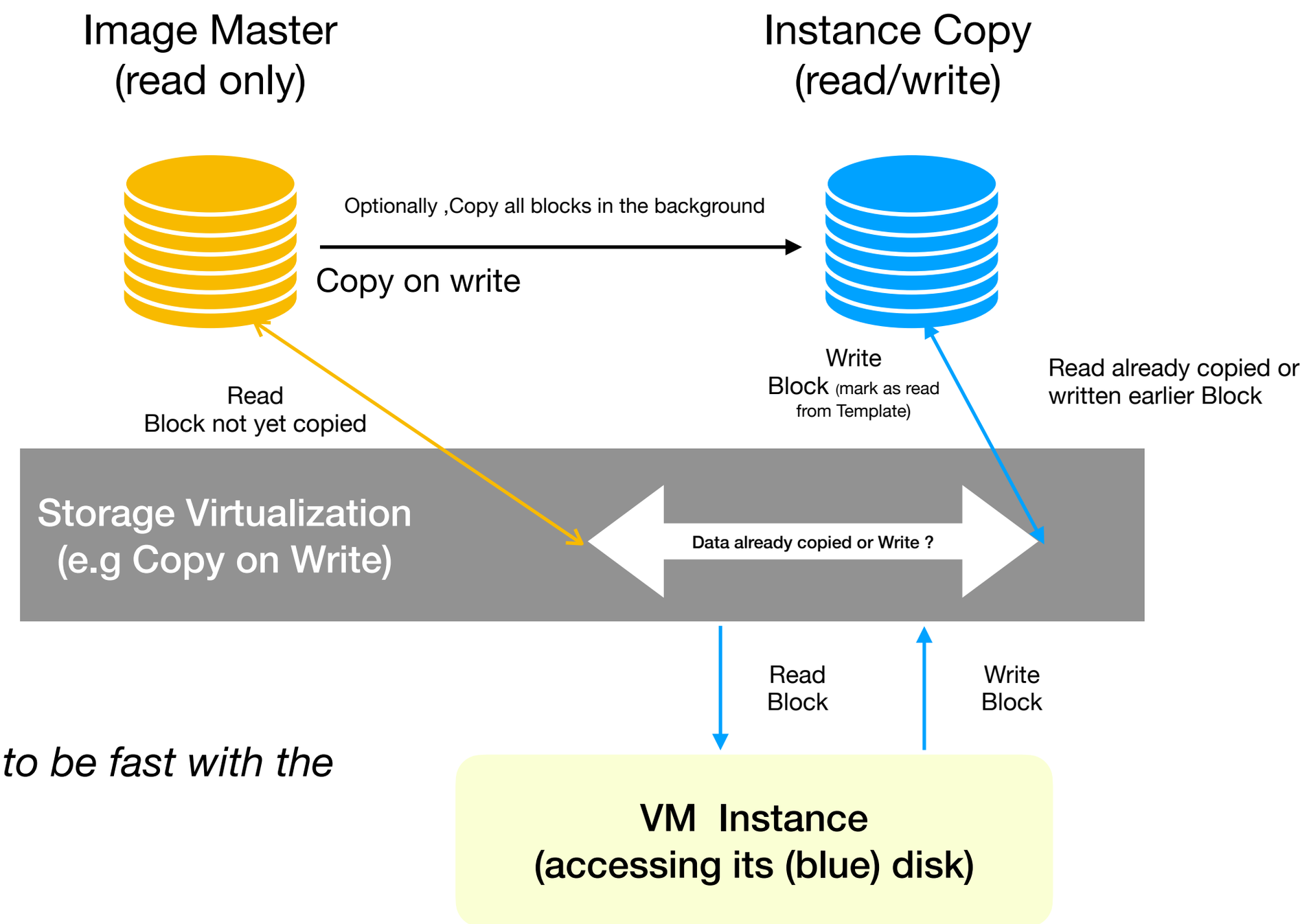
```
provisioner "local-exec" {command = "ANSIBLE_HOST_KEY_CHECKING=False ansible-playbook -u
{var.user} -i '${self.ipv4_address},' --private-key ${var.ssh_private_key} playbook.yml"}
```

https://www.cprime.com/resources/blog/terraform-and-ansible-tutorial-integrating-terraform-managed-instances-with-ansible-control-nodes/

Deployment Automation Considerations

- A flow of activities (not exhaustive)
    - Estimated the ‘best host’ (e.g. Anti-Affinity rules)
    - Get Network connectivity (e.g. internal, external IP)
    - Get Storage (Block Storage, get a fitting storage device)
        - Thin provisioning for additional volumes (only really requested blocks are allocated)
    - Copy the Image (e.g from Swift into the bootable disk)
        - Performance critical, caching, Storage Optimisations (e.g. use the master image for reads and have a local copy for writes (run the full copy in background))
    - Establish a virtual Server (CPU, Memory) via the server hypervisor
    - Boot the server including the instantiation of the server (cloud init)
    - OS Level configurations (inclusive platform configuration, we discussed earlier)

Storage Virtualization to optimize initial deployment



It is all about to be fast with the first ping

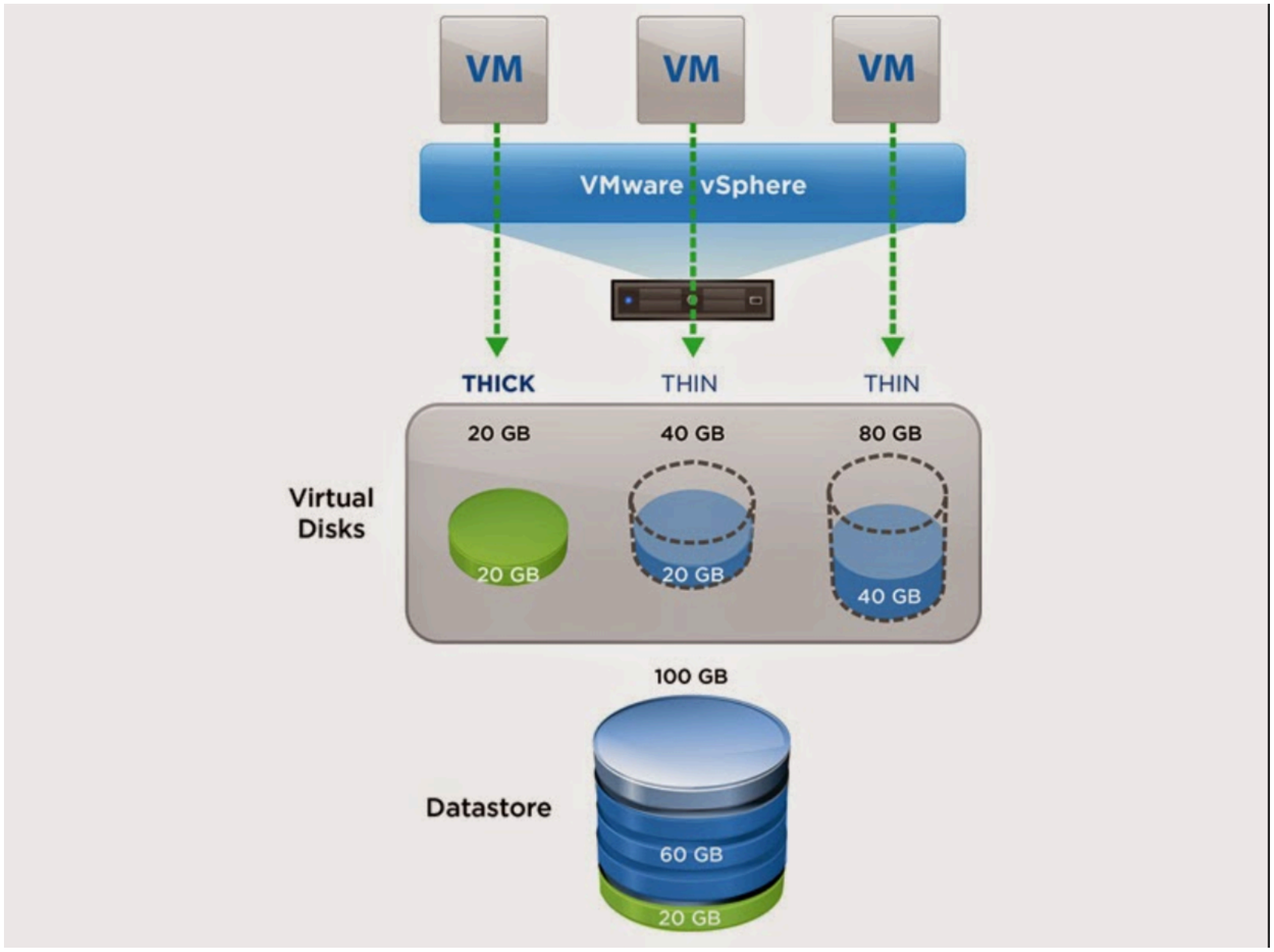
Over-allocate Storage and Thin Provisioning

Thin Provisioning

- in a shared-storage environment, provides a method for optimizing utilization of available storage. It relies on on-demand allocation of blocks of data versus the traditional method of allocating all the blocks in advance.

Over-allocation or over-subscription

- is a mechanism that allows a server to view more storage capacity than has been physically reserved on the storage array itself.
- Physical storage capacity on the array is only dedicated when data is actually written by the application, not when the storage volume is initially allocated.



In other words **thin** means you get your blocks when you need them, **over-subscription** means hopefully not all VM are requesting the full storage demand, in that case you don't get what you think you deserve

Open Cloud Architecture

Creativity and Innovation thru collaboration among different groups and individuals

CONWAYS LAW:

Organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations.

Melvin Conway (1967)

https://en.wikipedia.org/wiki/Melvin_Conway

Referenzen

- Open Stack
    - https://docs.openstack.org/queens/
    - https://releases.openstack.org/
    - https://wiki.openstack.org/wiki/BasicDesignTenets
    - https://www.slideshare.net/mirantis/open-stack-architecture-overviewmeetup662013?qid=b85f3161-3de3-48da-a78a-18e05d6dfddf&v=&b=&from_search=8
    - https://cloudify.co/2015/12/26/openstack-docker-kubernetes-hybrid-cloud-nfv-orchestration-mano-etsi-cloudify.html
    - https://docs.openstack.org/heat/pike/template_guide/hello_world.html
- Images
    - https://spin.atomicobject.com/2013/06/03/ovf-virtual-machine/
    - https://support.citrix.com/article/CTX121652
- Virtualization
    - ftp://public.dhe.ibm.com/s390/zos/vse/pdf3/techconf2006/G51.pdf
    - http://www.virtualizationsoftware.com/top-5-enterprise-type-1-hypervisors/
    - https://vapour-apps.com/what-is-hypervisor/
    - https://en.wikipedia.org/wiki/Software-defined_storage
    - https://www.sdxcentral.com/sdn/definitions/why-sdn-software-defined-networking-or-nfv-network-functions-virtualization-now/
- Softlayer
    - https://www-05.ibm.com/de/services/softlayer/Softlayer_ePDF.pdf
    - https://www.softlayer.com/sites/default/files/assets/page/softlayer-platform-description.pdf