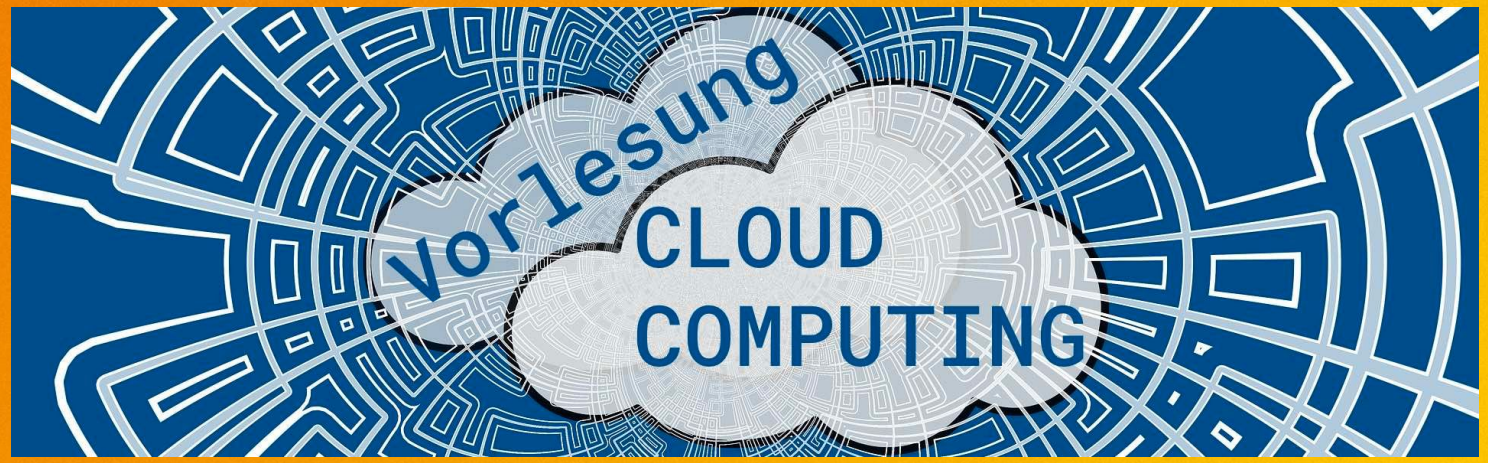
qaware.de

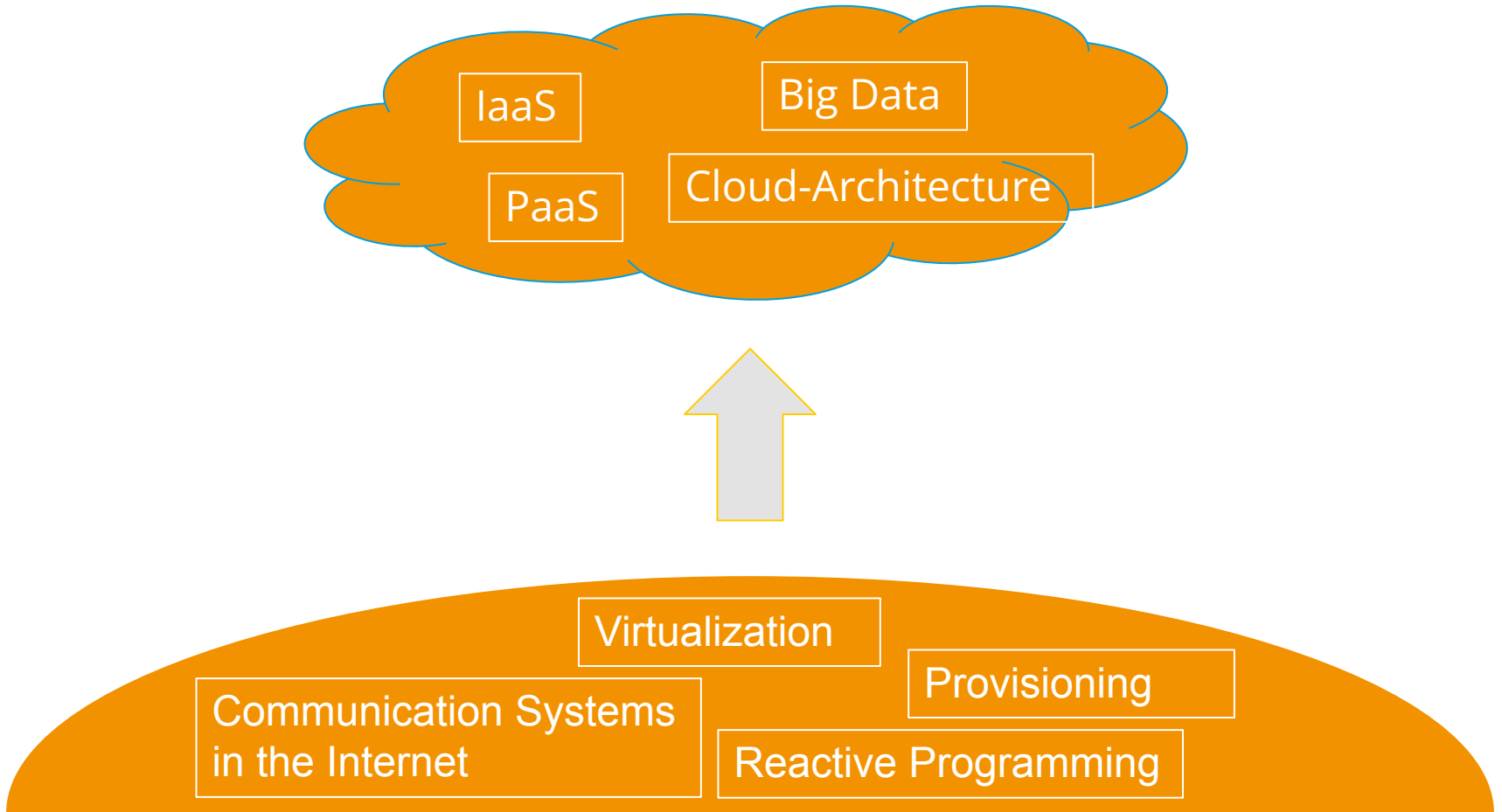
Hochschule **Rosenheim**
University of Applied Sciences

QA|WARE
SOFTWARE ENGINEERING

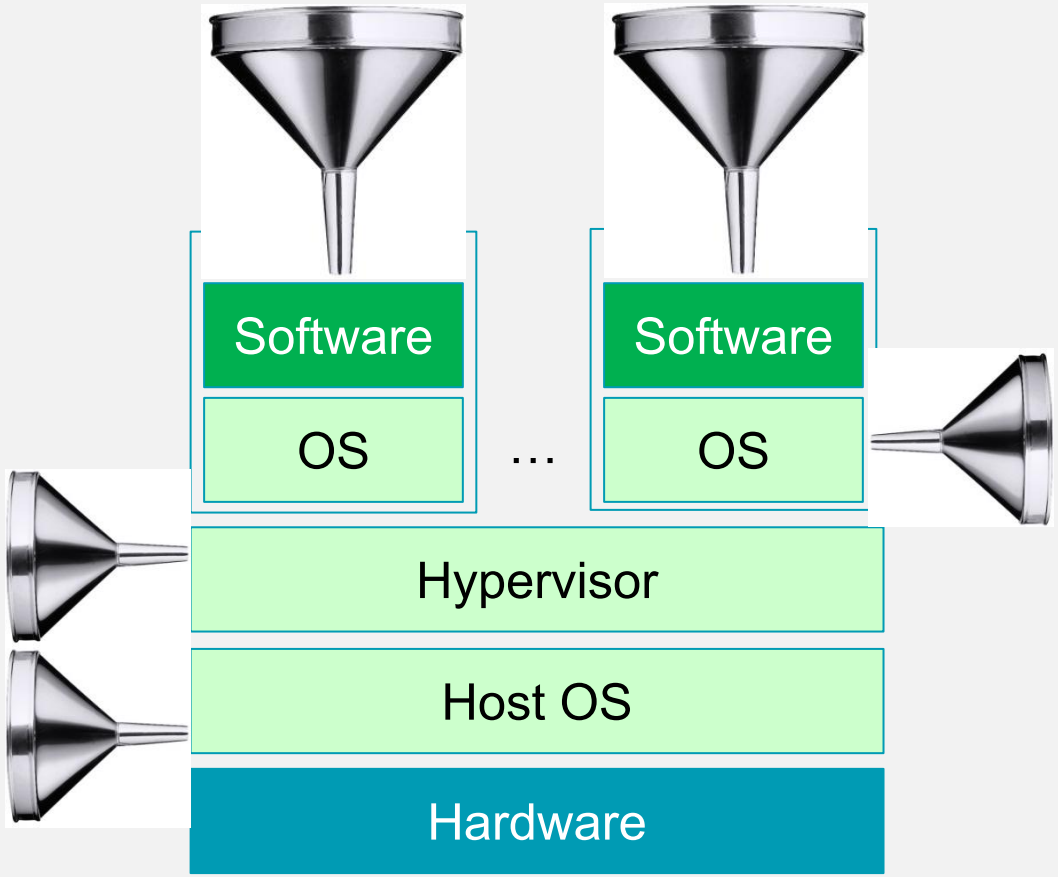
Lecture 5:
Infrastructure-as-a-Service



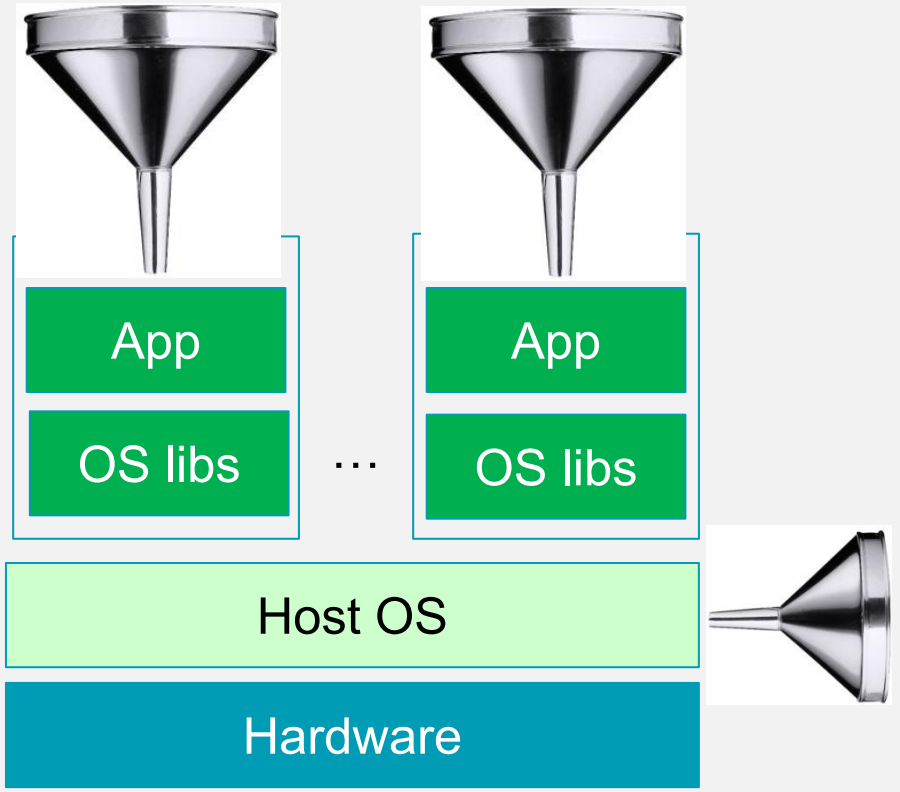
As of today we are in the cloud



Provisioning: How does software get into the boxes?

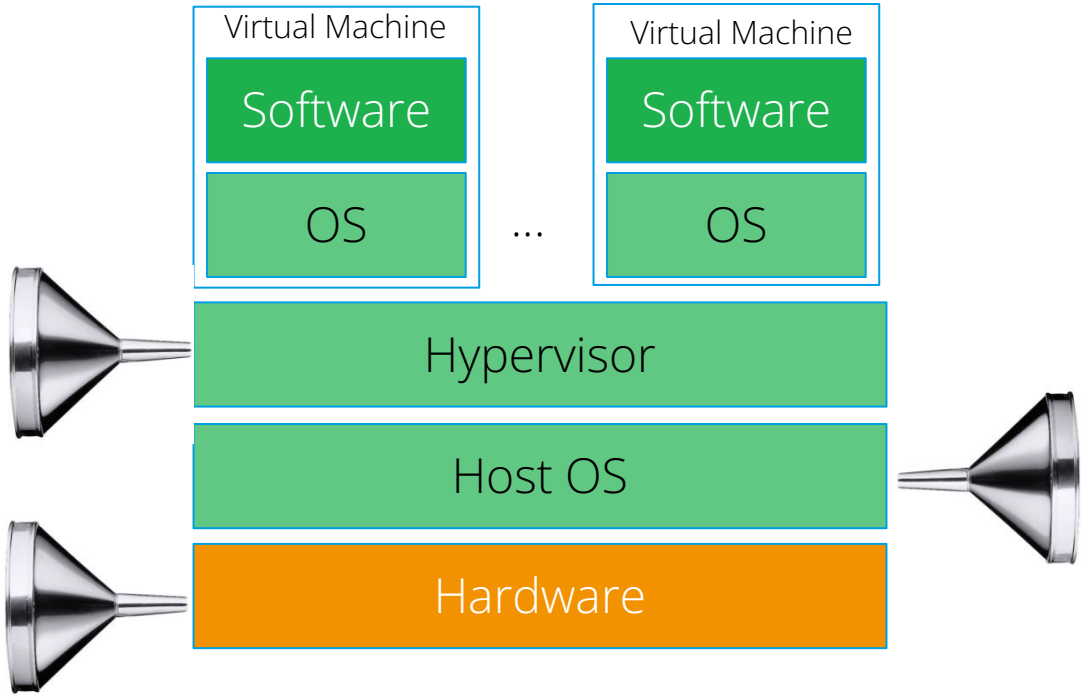


Hardware Virtualization

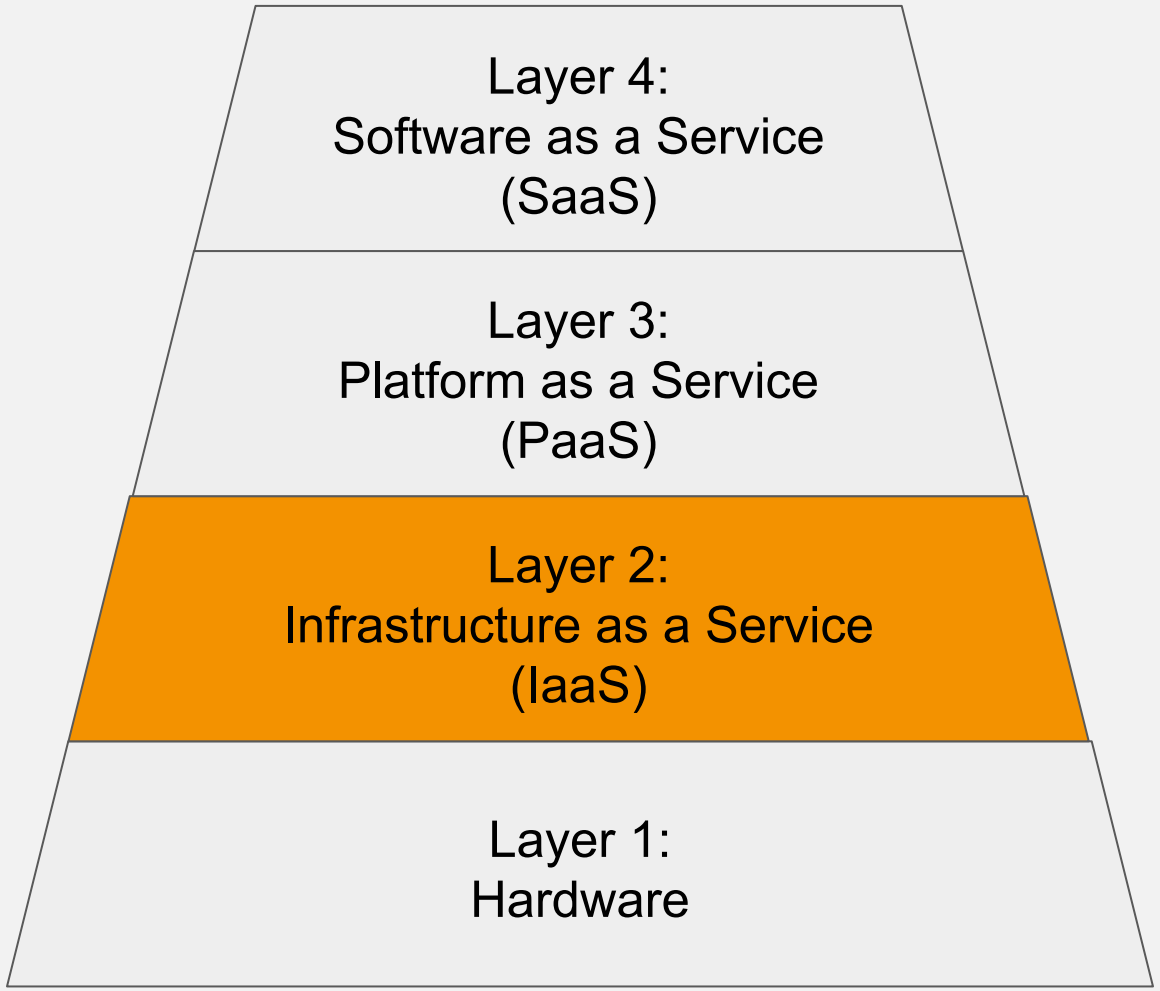


OS Virtualization

In this lecture: How to get infrastructure/hardware?



The layered model of cloud computing: From metal to application.



- Customizable software services
- CaaS: Component as a service (e.g. Google Charts)
- BaaS: Backend as a Service
- Transparent Updates

Target Group: Users

- Platform services
- Application programming interfaces (APIs)
- Abstraction of technical infrastructure

Target group: Developers

- Elasticity
- Virtual resourcepools
- Technical infrastructure: Machines, Servers (DNS, DHCP, LB, NAS, …)

Target group: Operations

- Computers
- Network
- Storage

https://www.youtube.com/watch?v=M988_fsOSWo

Introduction: Infrastructure-as-a-Service

Time2System during the last century: > 1 Jahr.

http://de.wikipedia.org/wiki/Gro%C3%9Frechner

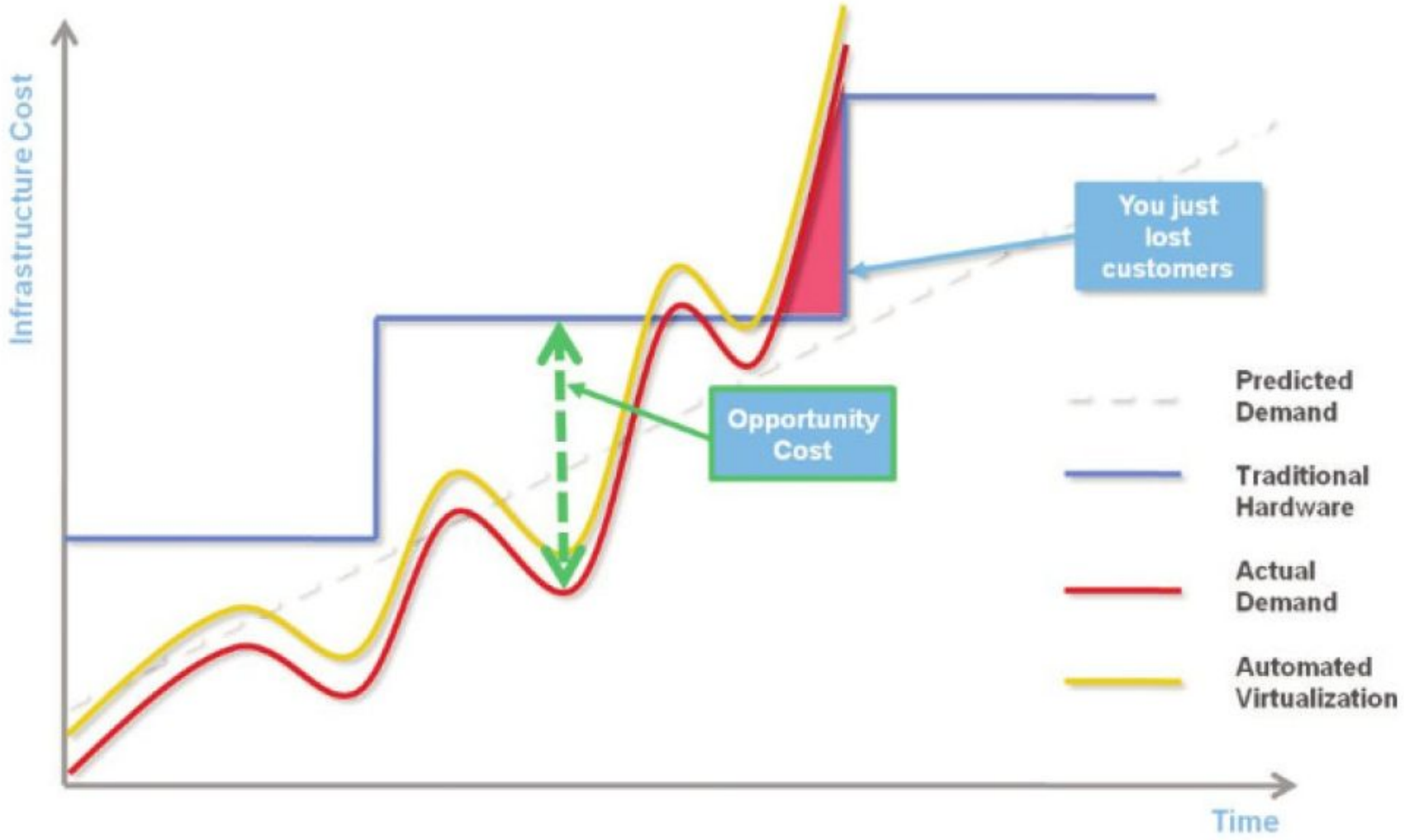
Time2System in the Cloud-Era: In real time.

The **Slashdot effect**, also known as **slashdotting**, occurs when a popular website links to a smaller website, causing a massive increase in traffic. This overloads the smaller site, causing it to slow down or even temporarily become unavailable. Typically, less robust sites are unable to cope with the huge increase in traffic and become unavailable – common causes are lack of sufficient data bandwidth, servers that fail to cope with the high number of requests, and traffic quotas. Sites that are maintained on shared hosting services often fail when confronted with the Slashdot effect. This has the same effect as a denial-of-service attack, albeit accidentally. The name stems from the huge influx of web traffic which would result from the technology news site *Slashdot* linking to websites. The term **flash crowd** is a more generic term.[1]

The original circumstances have changed, as flash crowds from *Slashdot* were reported in 2005 to be diminishing due to competition from similar sites,[2] and the general adoption of elastically scalable cloud hosting platforms.

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

Classical operations are expensive in times of dynamic demand



Source: Amazon Web Services

Definition IaaS

> IaaS refers to a business model that, contrary to the traditional purchasing of computing infrastructure, provides for renting and releasing it as needed.

Properties of an IaaS-Cloud:

- **Resource-Pools**: Availability of seemingly unlimited resources, that process requests in a distributed manner
- **Elasticity**: Dynamic allocation of additional resources based on demand
- **Pay-as-you-go Modell**: Only pay for what you actually use

Resource-Types in an IaaS-Cloud:

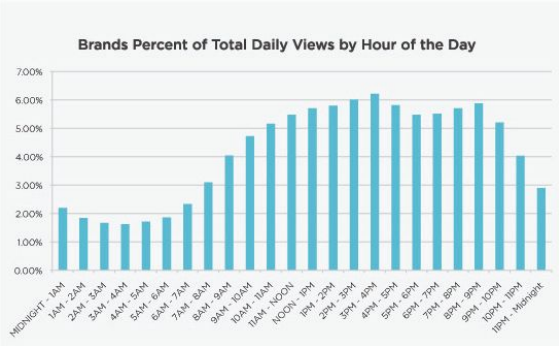
- **Compute**: Compute-Nodes with CPU & RAM
- **Storage**: Storage-capacity via mountable filesystems, block storage or database services.
- **Network**: Network and network-services like DNS, DHCP, VPN, CDN and Load Balancer.

Infrastructure-services in an IaaS-Cloud:

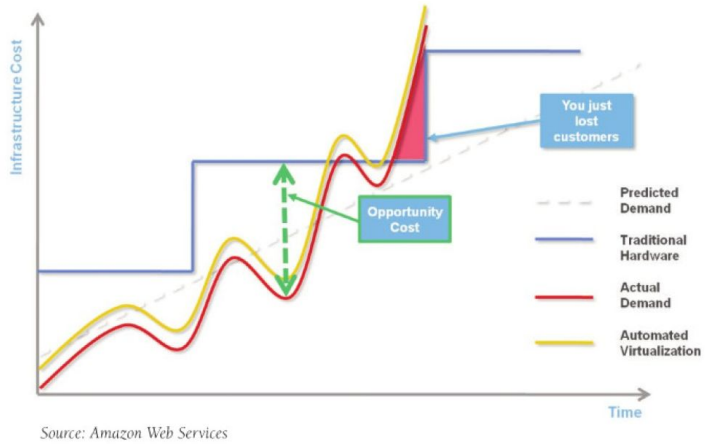
- **Monitoring**
- **Resource-Management**

Scalability: Effects

- **Daily and seasonal effects:** midday peak, prime-time peak, weekend peak, Christmas, Valentine's Day, Mother's Day, etc. (predictable load peaks)

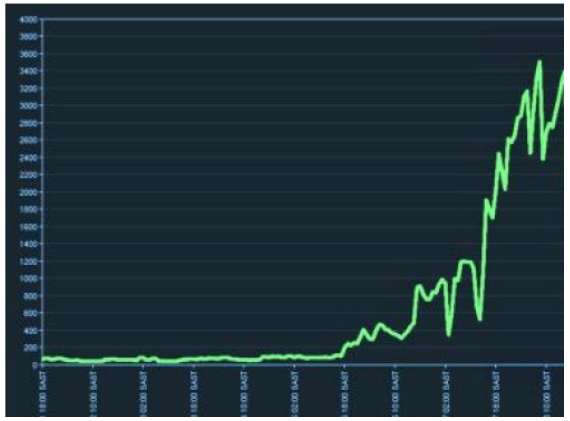


- Continuous growth

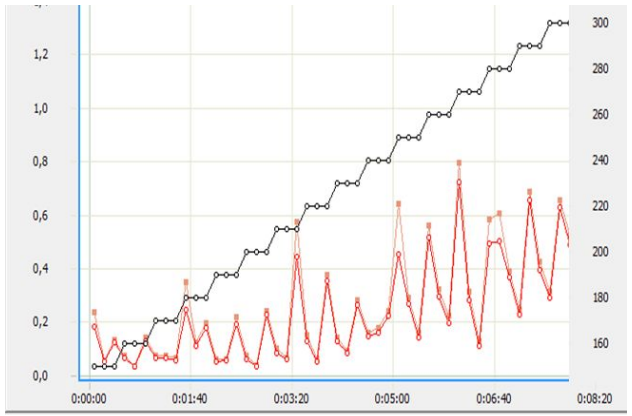


Source: Amazon Web Services

- **Special effects**: z.B. Slashdot-Effekt (unpredictable load peaks)



- **Temporary Platforms**: Projects, Tests, Batch…



Kinds of elasticity

Demand elasticity: Allocated resources increase/decrease with demand.

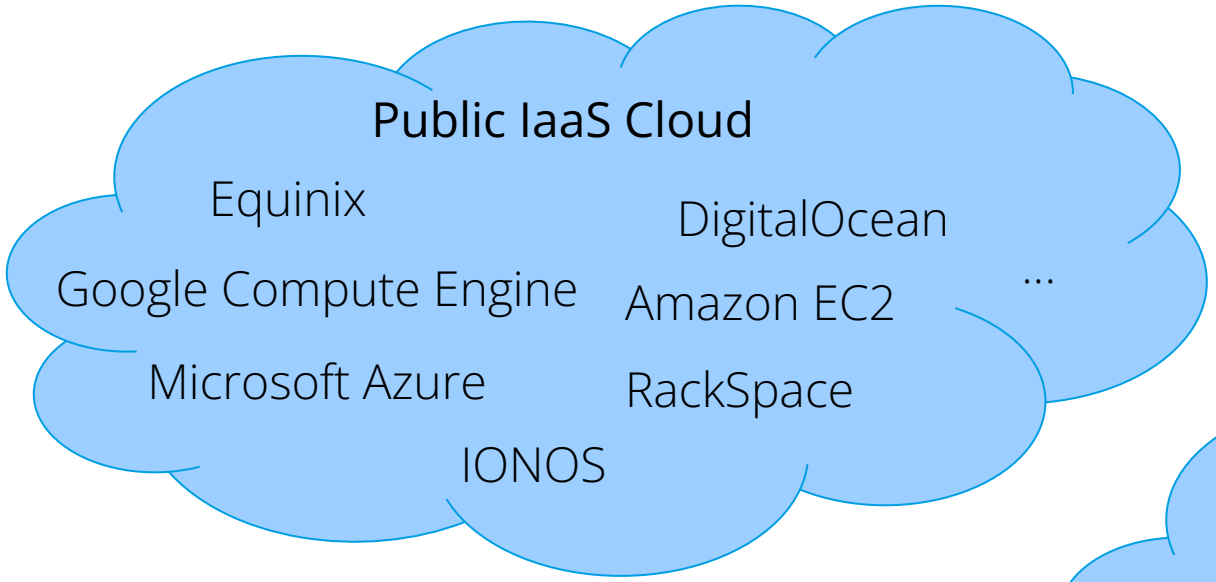
- **Pseudo-elasticity:** Quick setup. Short cancellation period.
- **Real-time elasticity:** Allocation and release of resources within seconds. Automated process with manual triggers or on schedule.
- **Self-adaptive elasticity:** Automatic allocation and release of resources in real-time based on rules and metrics.

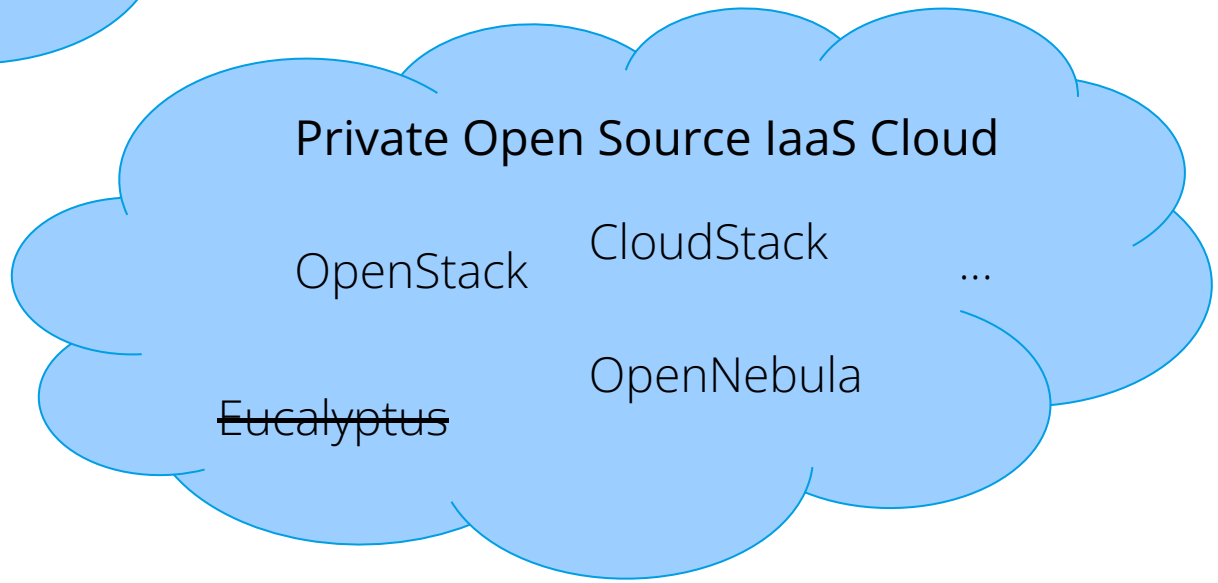
Supply elasticity: Allocated resources increase/decrease with supply.

- This is typical behavior of a grid: all available machines are allocated.
  Variants are also available where one can bid for free resources.

Income elasticity: Allocated resources increase/decrease with income or budget.

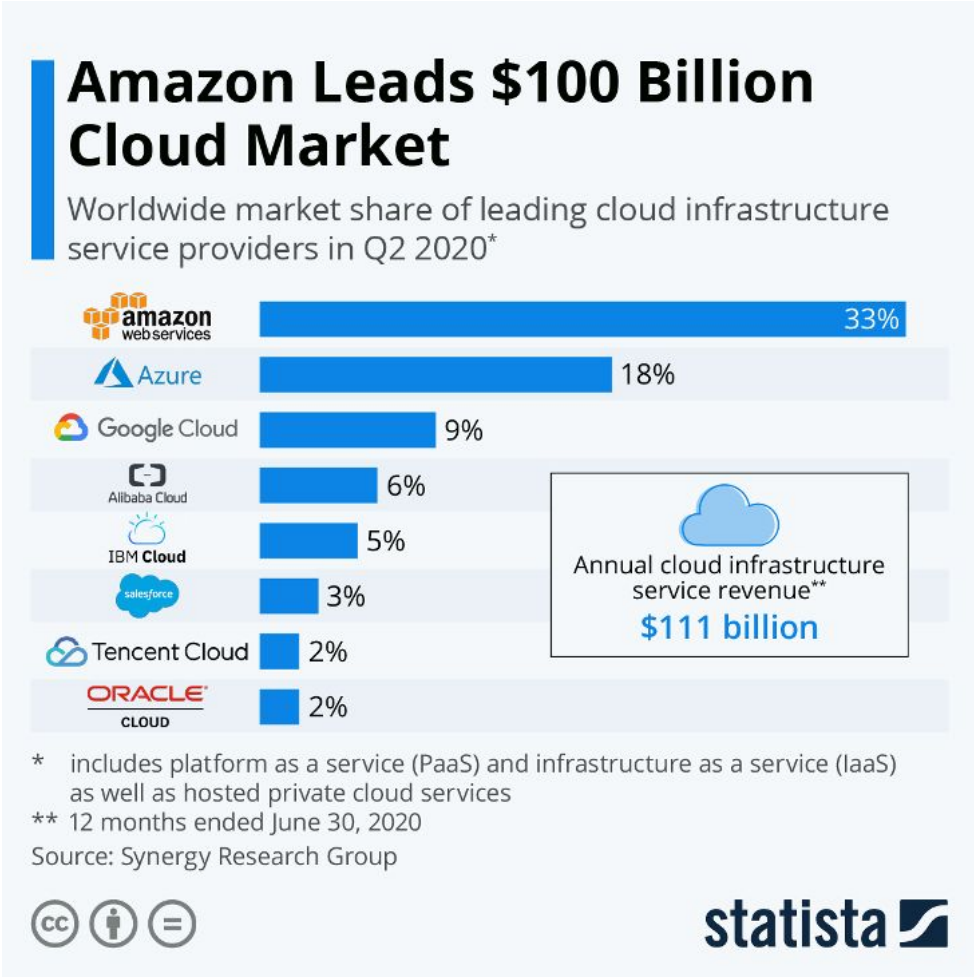
There are many cloud providers.





The IaaS market 2020

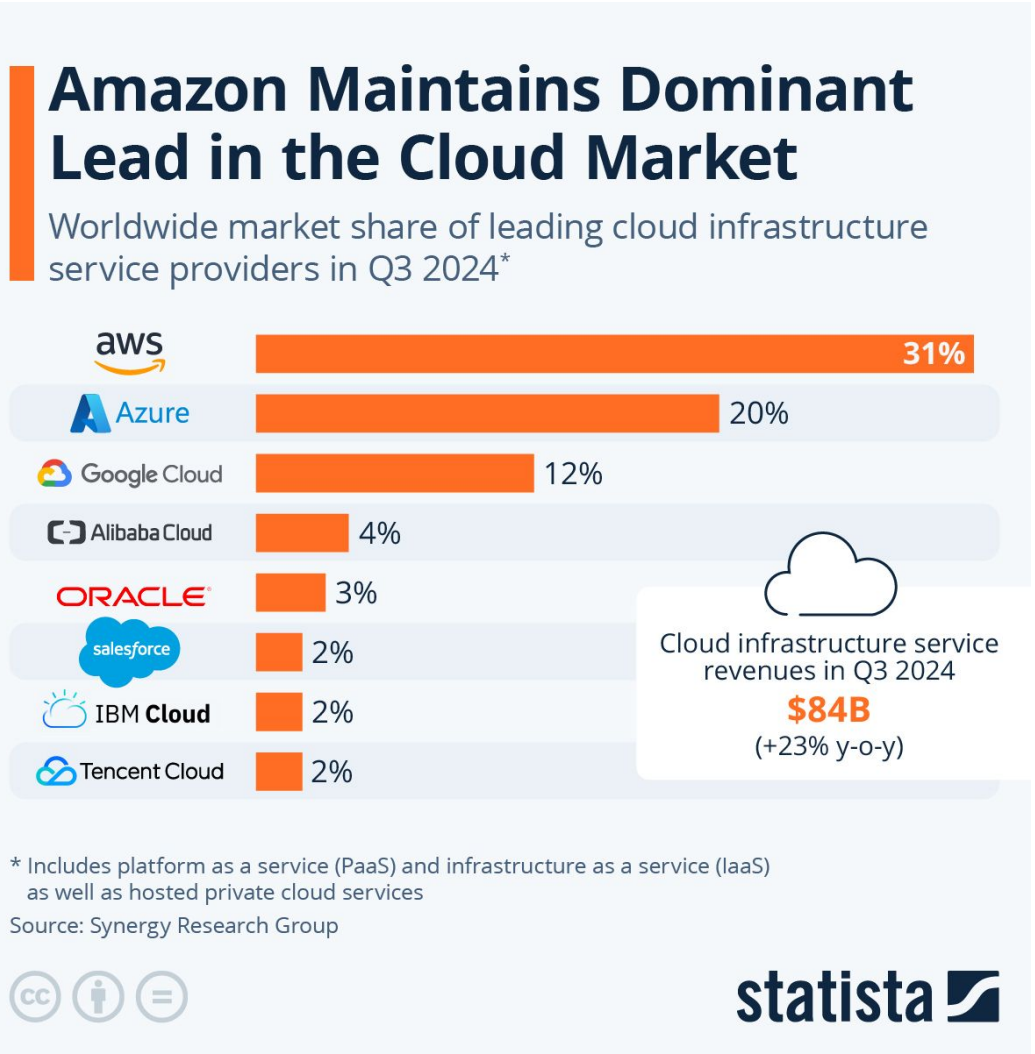
Figure 1. Magic Quadrant for Cloud Infrastructure and Platform Services



2020 Magic Quadrant for Cloud Infrastructure & Platform Services
https://pages.awscloud.com/GLOBAL-multi-DL-gartner-mq-cips-2020-learn.html

The IaaS market 2024

Figure 1: Magic Quadrant for Strategic Cloud Platform Services



12/2023 Magic Quadrant for Cloud Infrastructure & Platform Services
https://www.gartner.com/technology/media-products/reprints/amazon/1-2FTPMD88-DEU.html?trk=44f67619-4f3b-42e8-93b9-32ad8a123845&sc_channel=el

Criteria when selecting an IaaS cloud

- Supported cloud variants (Private Cloud, Public Cloud, Hybrid Cloud, etc.)
- Reliability / Availability
- Security and data protection
- Predictable and stable performance
- Pricing model: Fixed and flexible costs
- Scalability: Limits, automation, and response times
- Lock-in of data and applications: Open APIs
- Liability
- Support

Service Level Agreement

Service Level Objective

An SLO is a specific measurable target that defines a key aspect of the service's performance. It is essentially the goal or objective that the service provider aims to achieve and can include metrics like uptime, response time, or error rates.

Service Level Agreement

An SLA is a formal contract between a service provider and a customer that outlines the expected level of service, including the SLOs, and specifies the responsibilities of both parties.

Availability-classes:

Availability %	Downtime per Year	Downtime per Month	Downtime per Week
99.9% (three nines)	8.76 hours	43.2 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
99.99% (four nines)	52.6 minutes	4.32 minutes	1.01 minutes
99.999% (five nines)	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% (six nines)	31.5 seconds	2.59 seconds	.0605 seconds

Example: Amazon S3 (Storage)

Service Commitment

AWS will use commercially reasonable efforts to make Amazon S3 available with a Monthly Uptime Percentage (defined below) of at least 99.9% during any monthly billing cycle (the "Service Commitment"). In the event Amazon S3 does not meet the Service Commitment, you will be eligible to receive a Service Credit as described below.

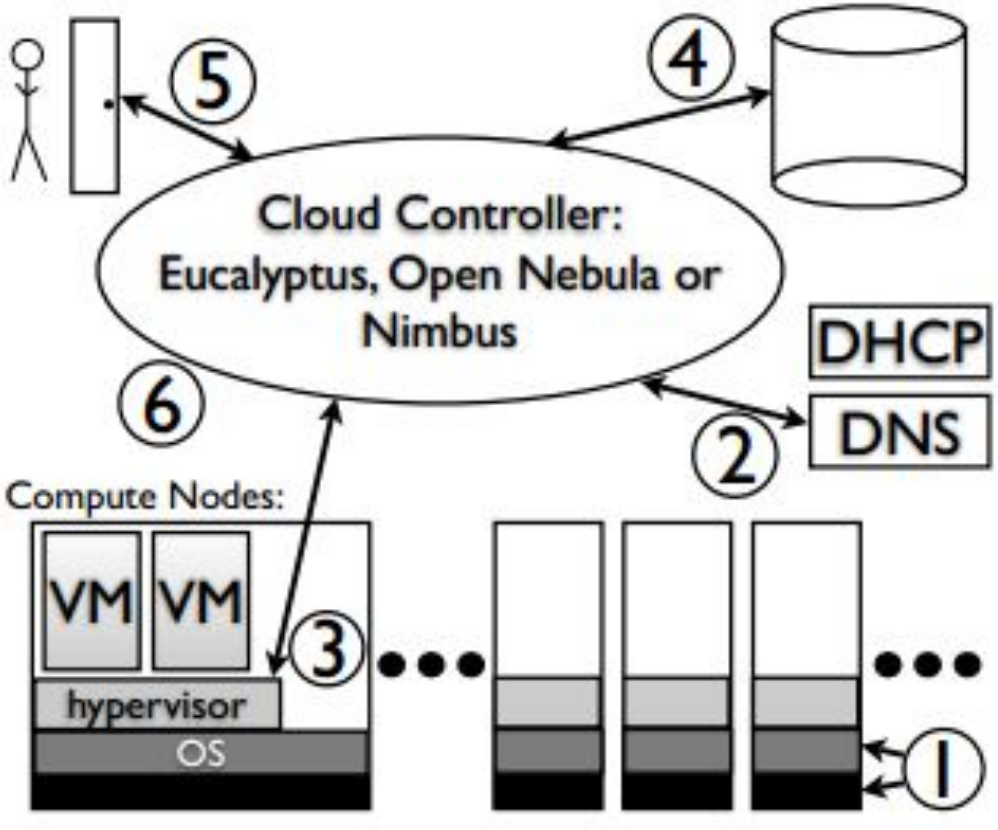
Monthly Uptime Percentage	Service Credit Percentage
Equal to or greater than 99% but less than 99.9%	10%
less than 99%	25%

Security Aspects of an IaaS-Cloud.

- Confidentiality of data and data communication: data encryption, VPNs
- Traceability of data: compliance with national laws (e.g., EU data protection regulation, US Patriot Act) through geographical data storage
- Firewalls and strong authentication methods
- Backup of VMs, storage, and databases
- Certifications: ISO 27001, TÜV IT

Architecture of an IaaS-Cloud

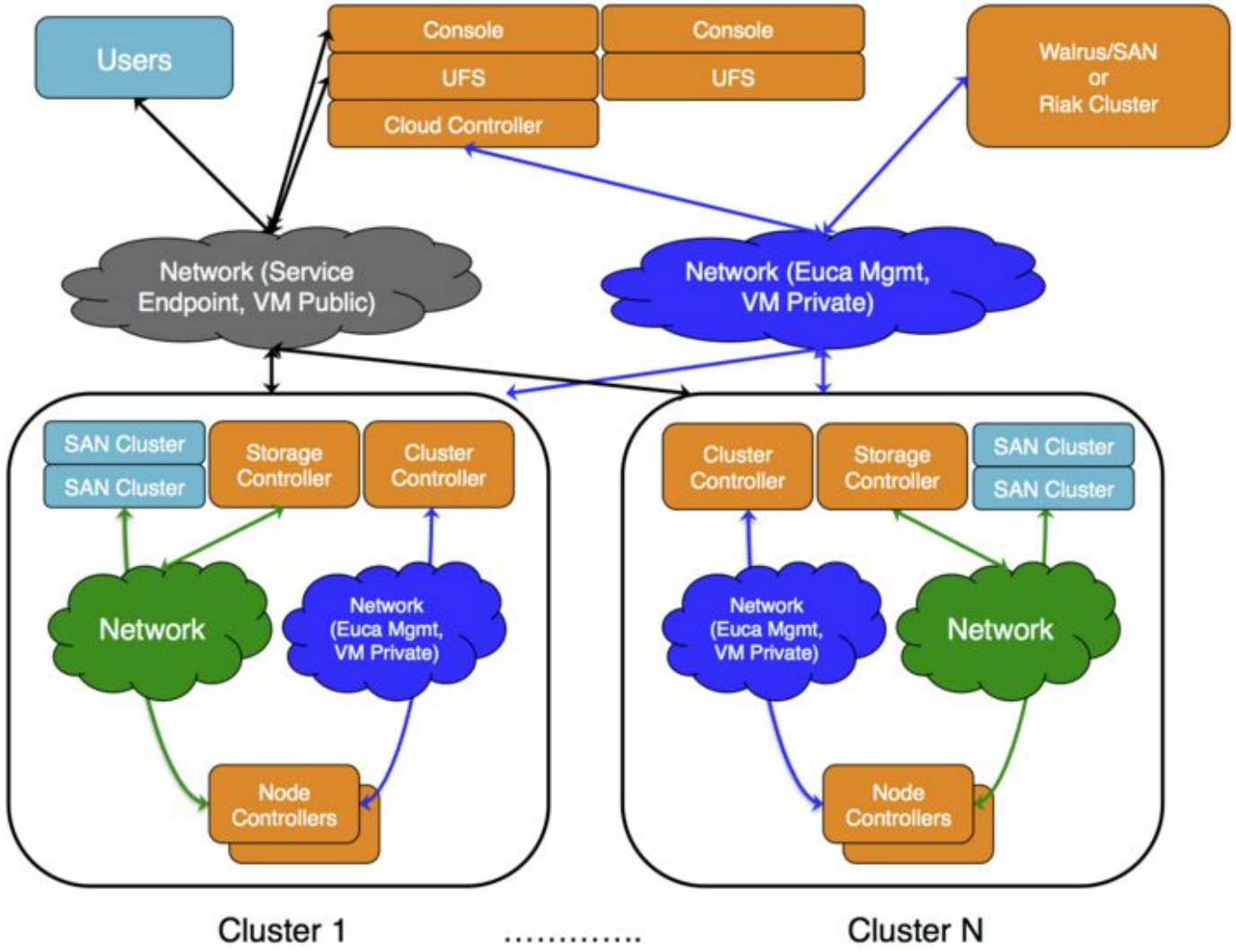
An IaaS reference architecture.



1. Hardware and OS
2. Virtual network and network services
3. Virtualization
4. Storage and Image management
5. Management interface for admins and users
6. Cloud Controller for tenant specific management of Cloud-Ressourcen

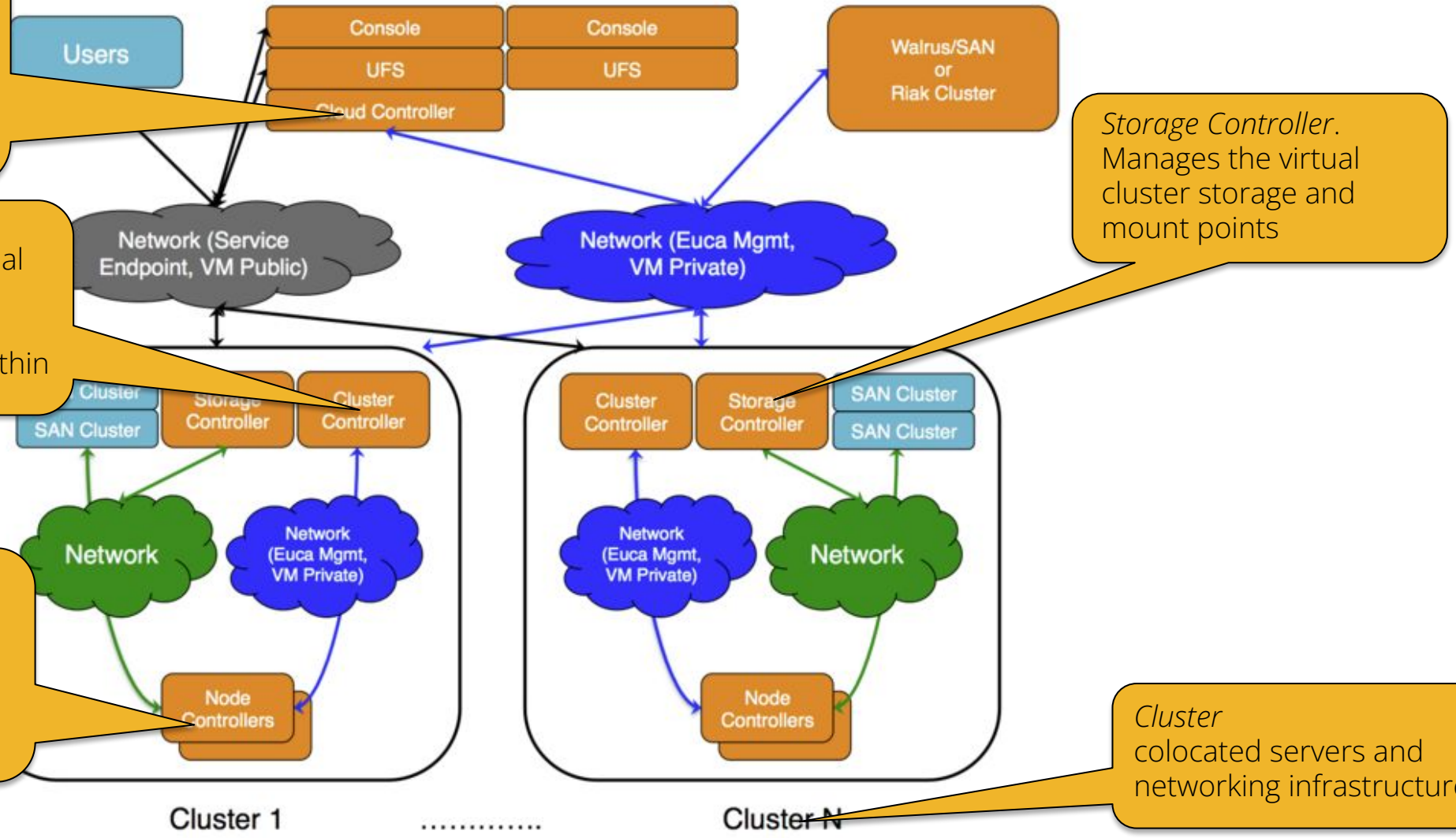
Peter Sempolinski and Douglas Thain,
"A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus",
IEEE International Conference on Cloud Computing Technology and Science, 2010.

Internal architecture of an IaaS-Cloud based on Eucalyptus.



Internal architecture of an IaaS-Cloud based on Eucalyptus.

Cloud Controller
Central control unit of the cloud.
Aggregates information from the CCs and distributes and manages all virtual resources.
Offers a central storage service (Walrus/S3), among other things, for virtual images.



Cluster Controller.
Controls the distribution of virtual images on nodes based on NC (Node Controller) information.
Manages the private network within the cluster.

Storage Controller.
Manages the virtual cluster storage and mount points

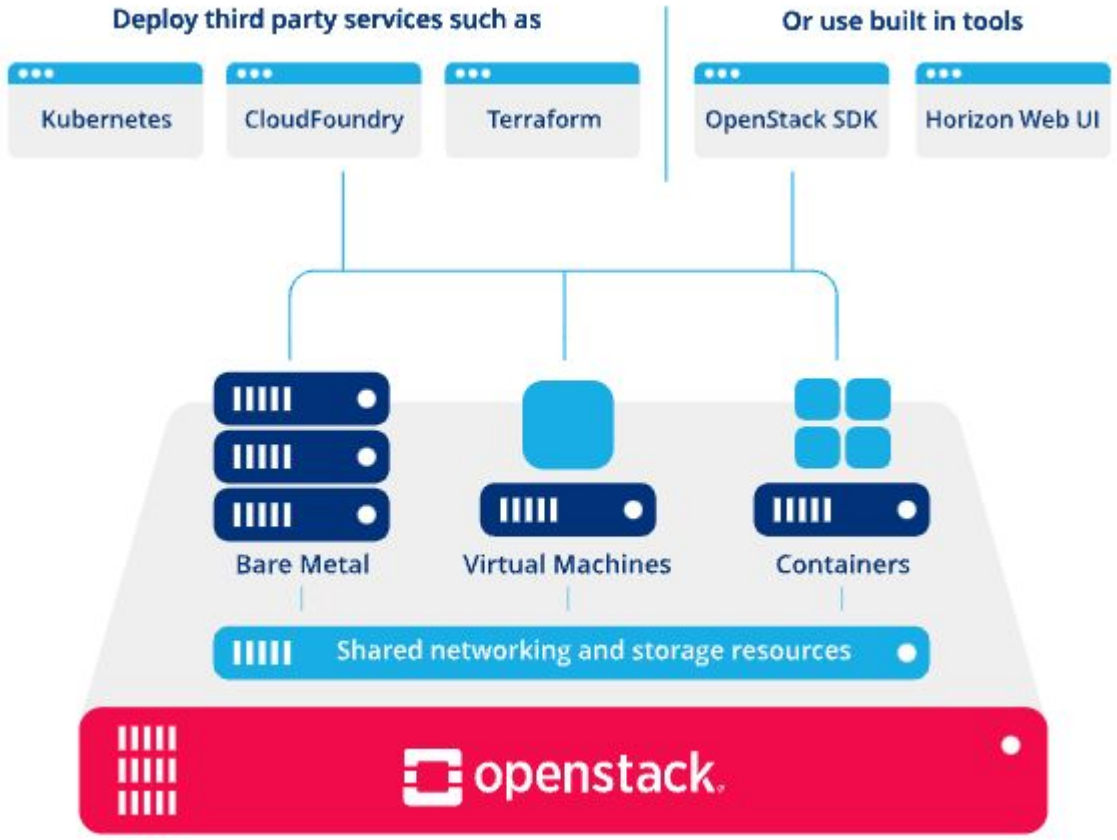
Node Controller.
Each hypervisor (cloud node) has a node controller.
Provides information on the available resources.

Cluster
colocated servers and networking infrastructure

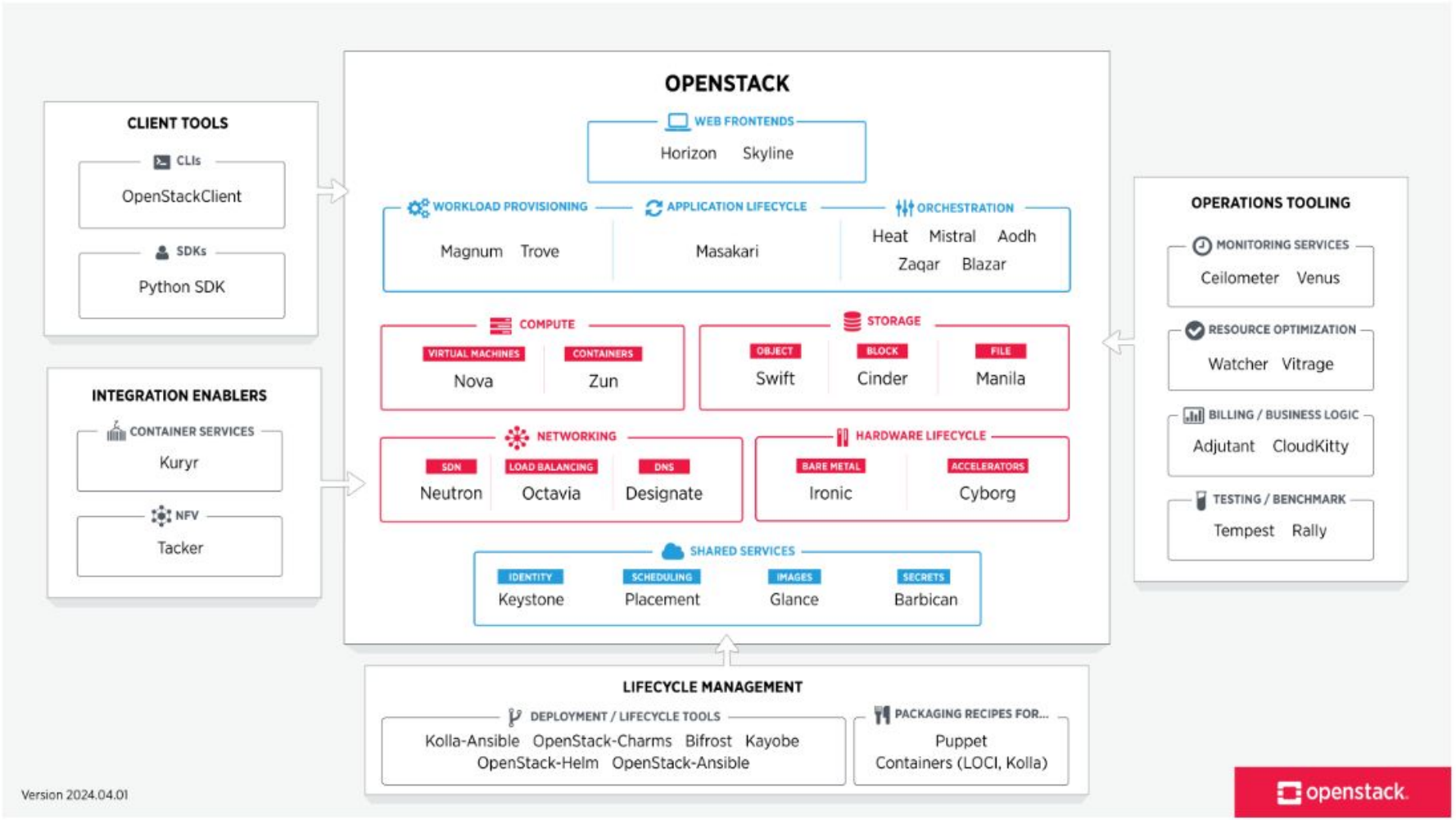
IaaS with OpenStack

OpenStack: the de-facto standard for Open-Source Private IaaS Clouds.

- The open-source project was significantly initiated by RackSpace and NASA.
- The first full release was in October 2010.
- Licensed under the Apache License.
- A wide range of traditional IT players (SAP, IBM, vmWare, HP, Oracle, Cisco) are part of the OpenStack community.
- Designed more as a framework than a finished system for IaaS clouds.

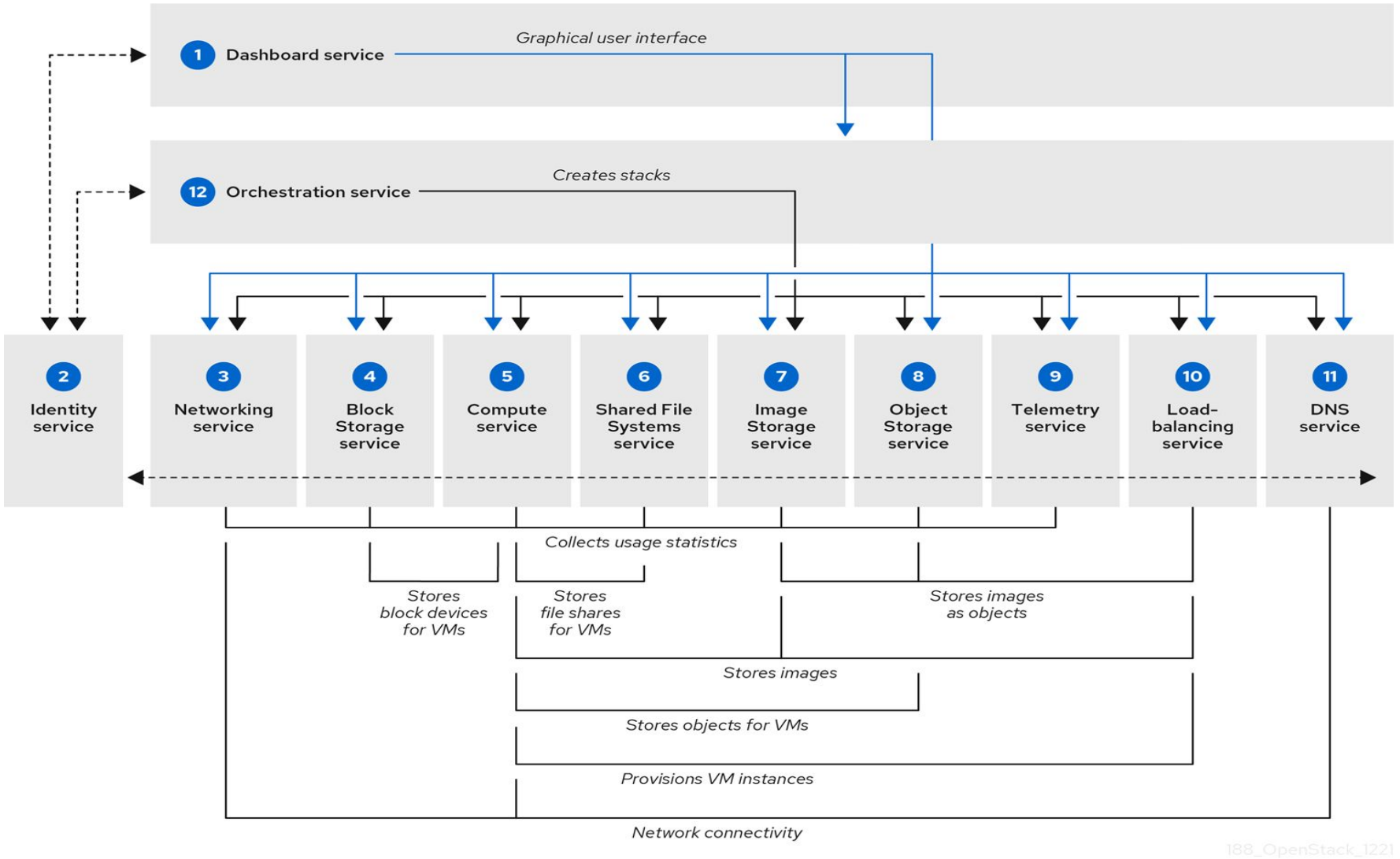


The OpenStack components.



https://www.openstack.org/software/

Core-Components working together in OpenStack.

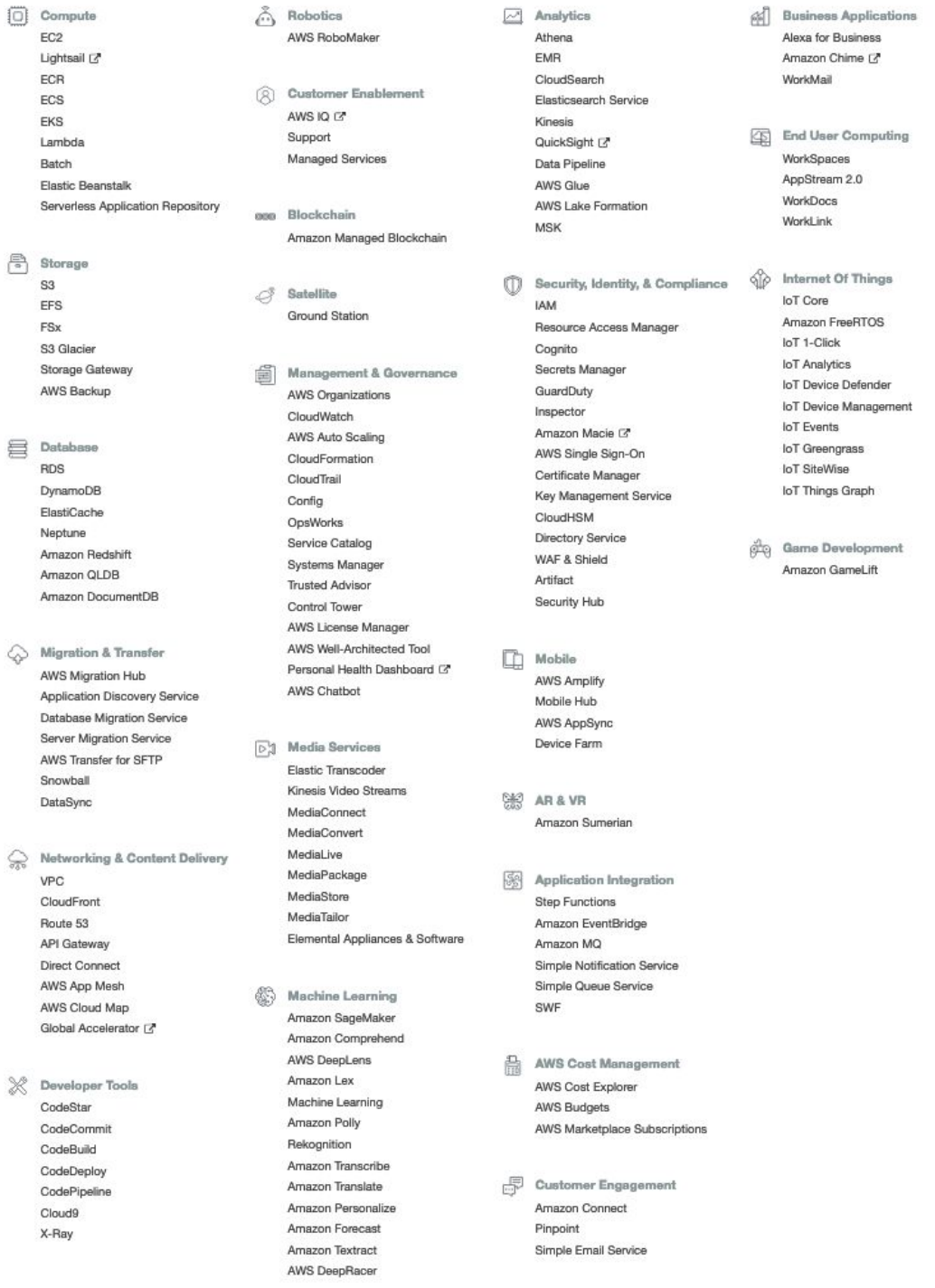


https://docs.redhat.com/en/documentation/red_hat_openstack_platform/17.1/html/introduction_to_red_hat_openstack_platform/assembly_rhosp-software#assembly_rhosp-software

IaaS with Amazon EC2

Besides Amazon EC2 IaaS Cloud AWS offers many more IaaS-components, PaaS, Serverless-, und SaaS-services.

Product listing: https://aws.amazon.com/de/products/



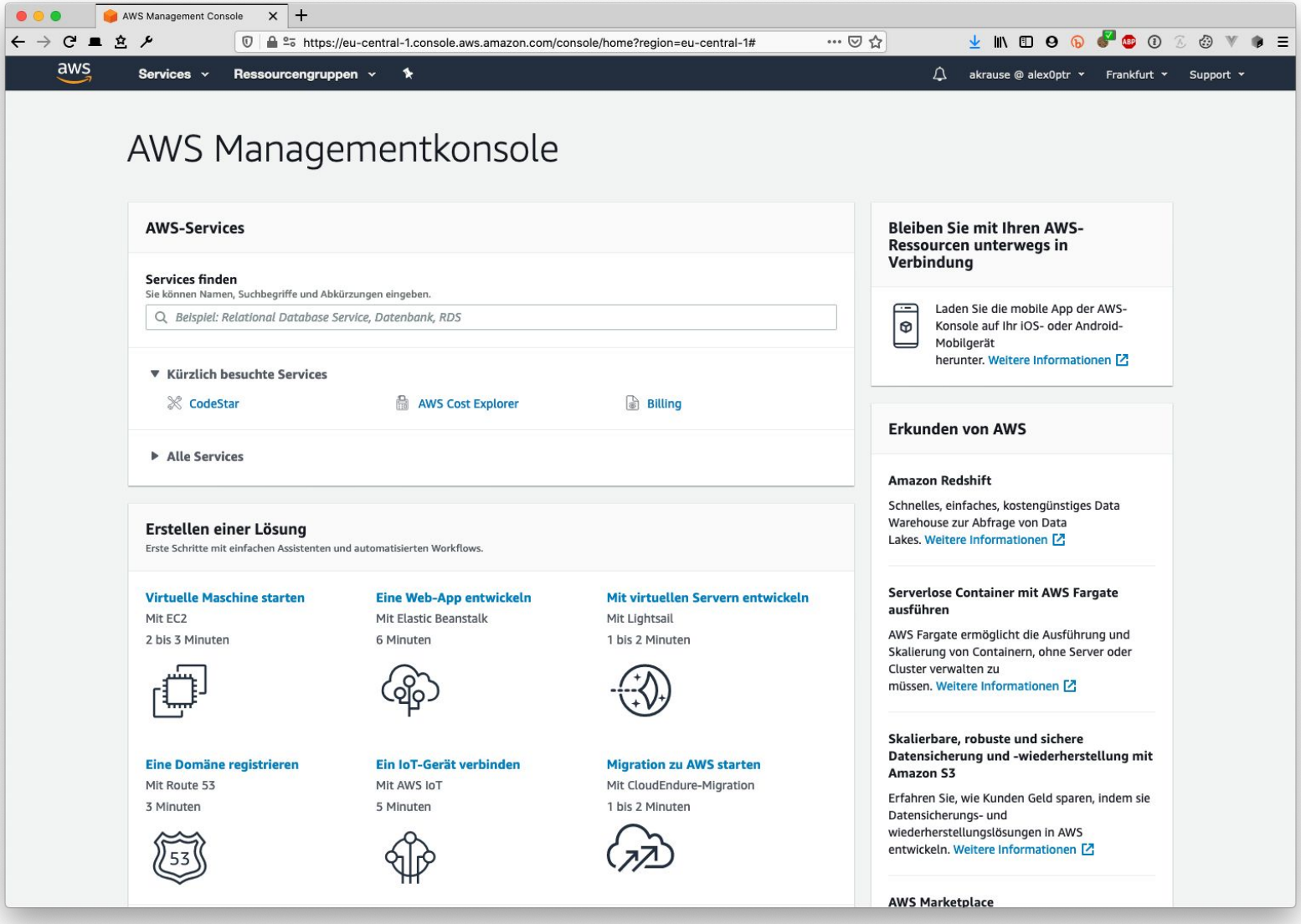
Global distribution of AWS

- 24 Regions
- split into 77 Availability Zones
- 216 Points of Presence

Example:

- eu-central-1 (Frankfurt)
- since 2014
- 3 Availability Zones

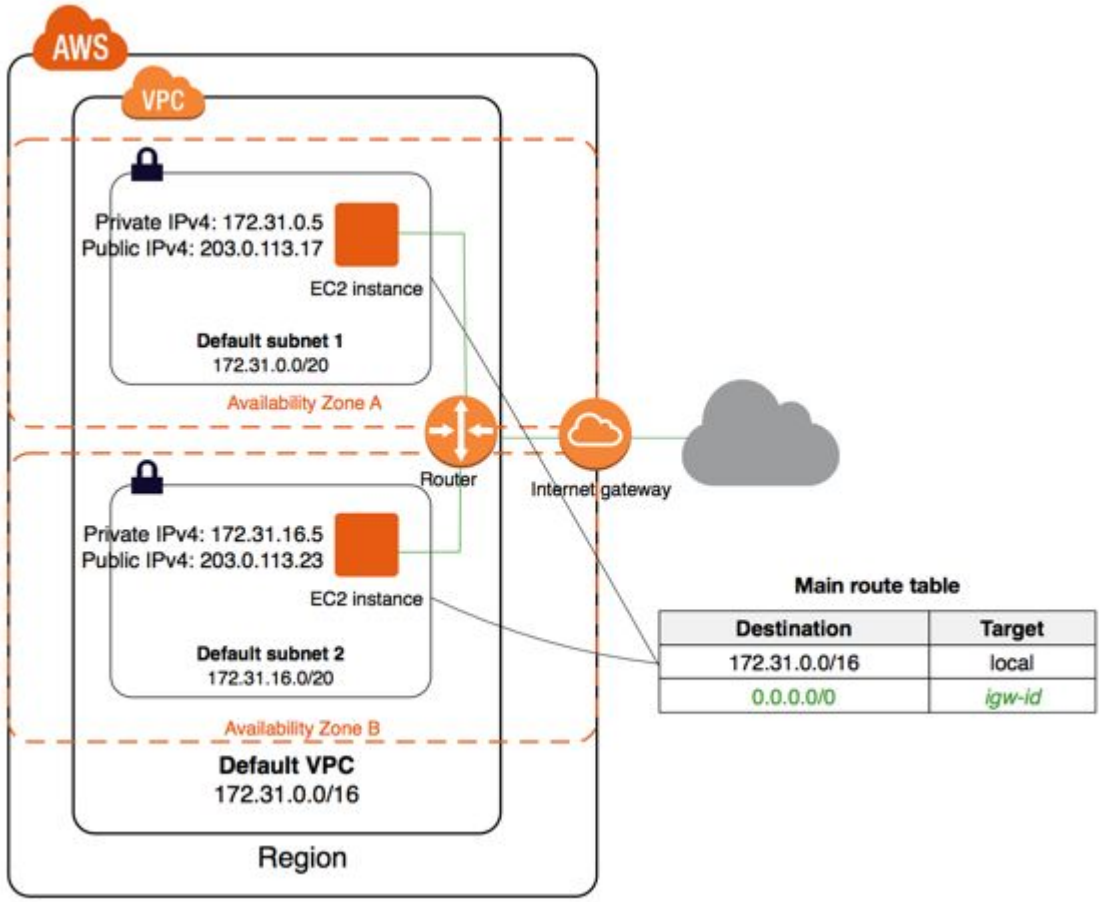
The AWS Management console allows to control all services



Amazon VPC (Virtual Private Cloud)

- Software Defined Network is an enforced prerequisite for any IaaS architecture on AWS
- Allows to provision:
  - VPCs and subnets
  - Network interfaces
  - Security Groups
  - Routing tables
  - Internet Gateways
  - NAT
  - DHCP
  - DNS
  - Elastic IP addresses
  - …

Preconfigured Standard Network:



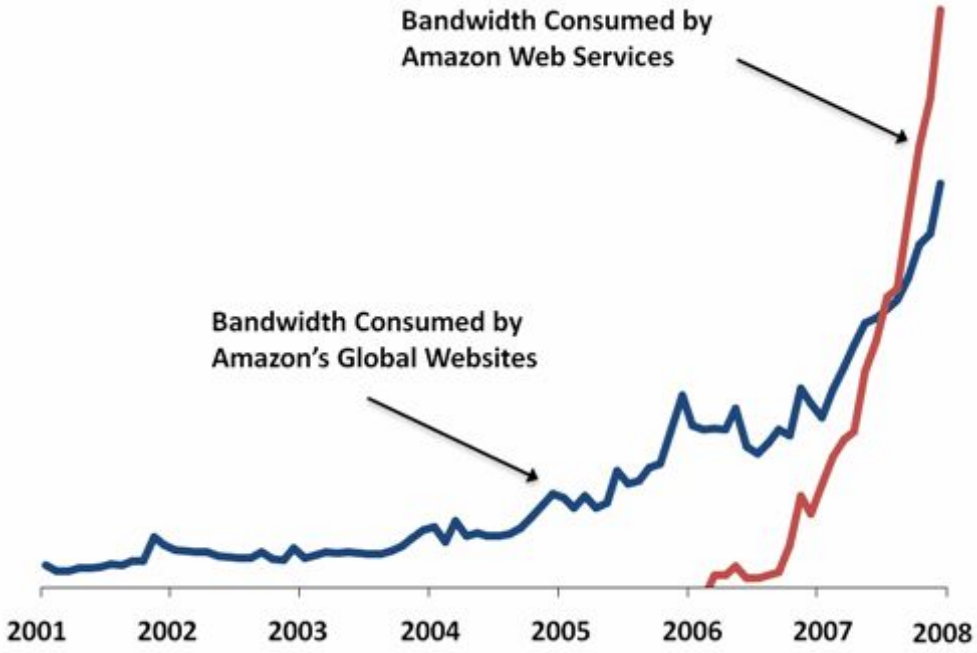
Amazon EC2 (Elastic Compute Cloud)

Amazon also offers an IaaS cloud as part of AWS (Amazon Web Services).

History:

- Started within Amazon in 2001
- Public beta launched on August 25, 2006
- By mid-2007, more bandwidth was consumed in the cloud by third parties than by Amazon's websites
- Production-ready as of October 23, 2008
- Approximately $12 billion investment in infrastructure from 2005 to 2012
- 2015: 1.5 to 2 million servers in 10 global data centers.

On-demand, reserved, and spot instances in various sizes and with different hardware: AWS EC2 Instance Types, as well as various storage and network services.

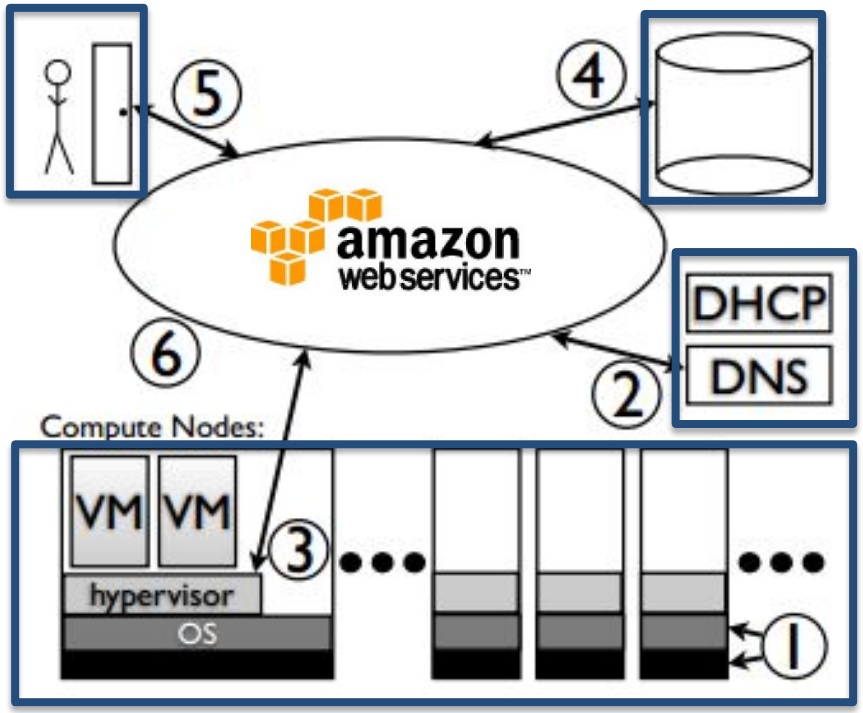


http://aws.typepad.com/aws/2008/05/lots-of-bits.html

Architecture of Amazon EC2.

- AWS Management Console
- Webservice-API

- EBS (Elastic Block Store)
- S3 (Simple Storage Service)



- VPC (Virtual Private Cloud)
- Route 53
- Elastic Load Balancer
- CloudFront CDN

- EC2-Nodes with Xen- or HVM-Virtualisierung
- Monitoring via CloudWatch
- AutoScaling based on CloudWatch-Metrics

EC2 Metadata Service

- Offered by the hypervisor it is available on every EC2-Instance under http://169.254.169.254/latest/meta-data

Allows an instance to query data:

- about its environment
- about itself
- user-defined metadata

Enables the implementation of advanced security and automation mechanisms:

- Instance profile via tokens to call AWS services, including automatic token exchange
- Short-lived management of SSH keys (EC2 Instance Connect)
- Tagging instance metrics in CloudWatch, etc.
- Provisioning via Cloud Init

Demonstration (5 Min): https://www.youtube.com/watch?v=tPQsI8n6er0

cloud-init

cloud-init

“Cloud-init is the defacto multi-distribution package that handles early initialization of a cloud instance.”

- since 2008
- Init System for the Cloud
- Initially only AWS and Ubuntu
- Now the de-facto standard across all cloud environments
- Configuration via user data from the metadata service

cloud-init - configuration options

Shell / Bash Skript:

```
#!/bin/sh
echo "Hello cloud-init!"
```

Templated Shell / Bash Skript:

```
## template: jinja
#!/bin/bash
{% if v1.region == 'us-east-2' -%}
echo 'Installing custom proxies for {{ v1.region }}'
sudo apt-get install my-xtra-fast-stack
{%- endif %}
```

Cloud-Config:

```
#cloud-config
packages:
  - cowsay
users:
  - default
  - name: app
    groups: docker
write_files:
  - content: nVc+Xj7rPhMqb...
    encoding: b64
    owner: app:app
    path: /home/app/application.yml
    permissions: '0655'
```

EC2 AutoScaling

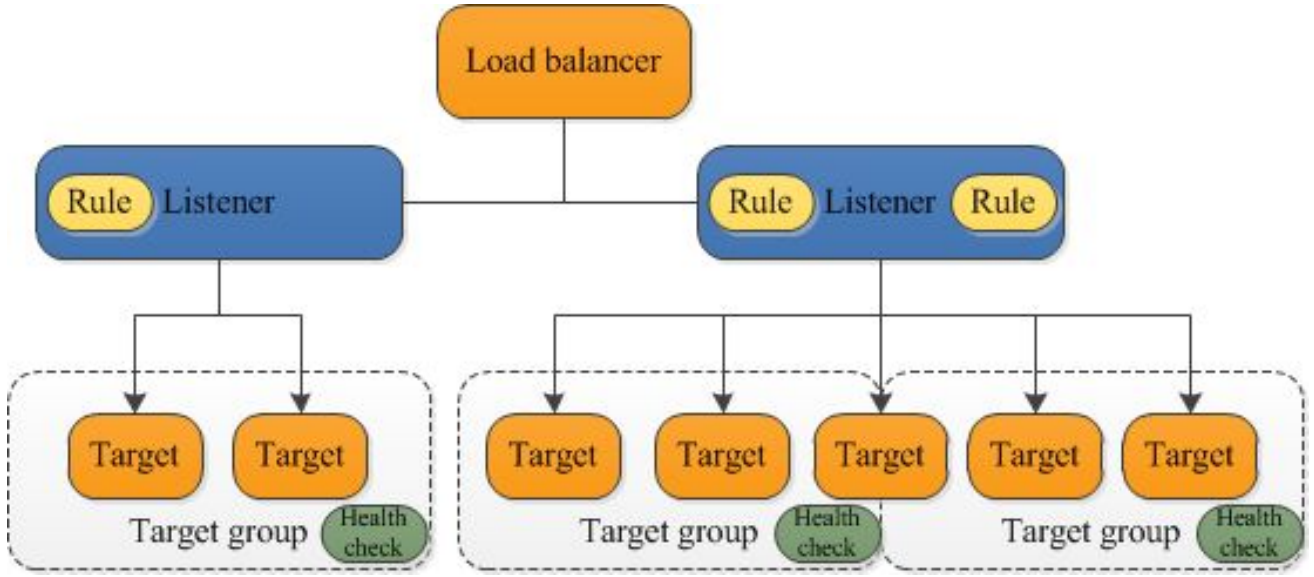
EC2 AutoScaling enables organizing instances into **groups** as a logical unit. For example, the group can be used to control the number of instances, allowing new ones to be created automatically if some fail.

EC2 AutoScaling groups use **launch templates** to create new instances or to replace them in a rolling manner. Launch templates define instance parameters such as AMI ID, instance type, security group, or block device mappings.

Scaling options allow automatic scaling of instances based on conditions, such as CPU load, scheduled timing, or predictive scaling.

Elastic Load Balancing

- Accepts public traffic and distributes it across instances.
- Monitors the functionality of instances/applications (health check) and only routes requests/connections to "healthy" targets.
- Different variants:
  - Application Load Balancer - Layer 7
  - Network Load Balancer - Layer 4
  - Classic Load Balancer - Legacy
- Supports TLS, integration with AutoScaling, etc.

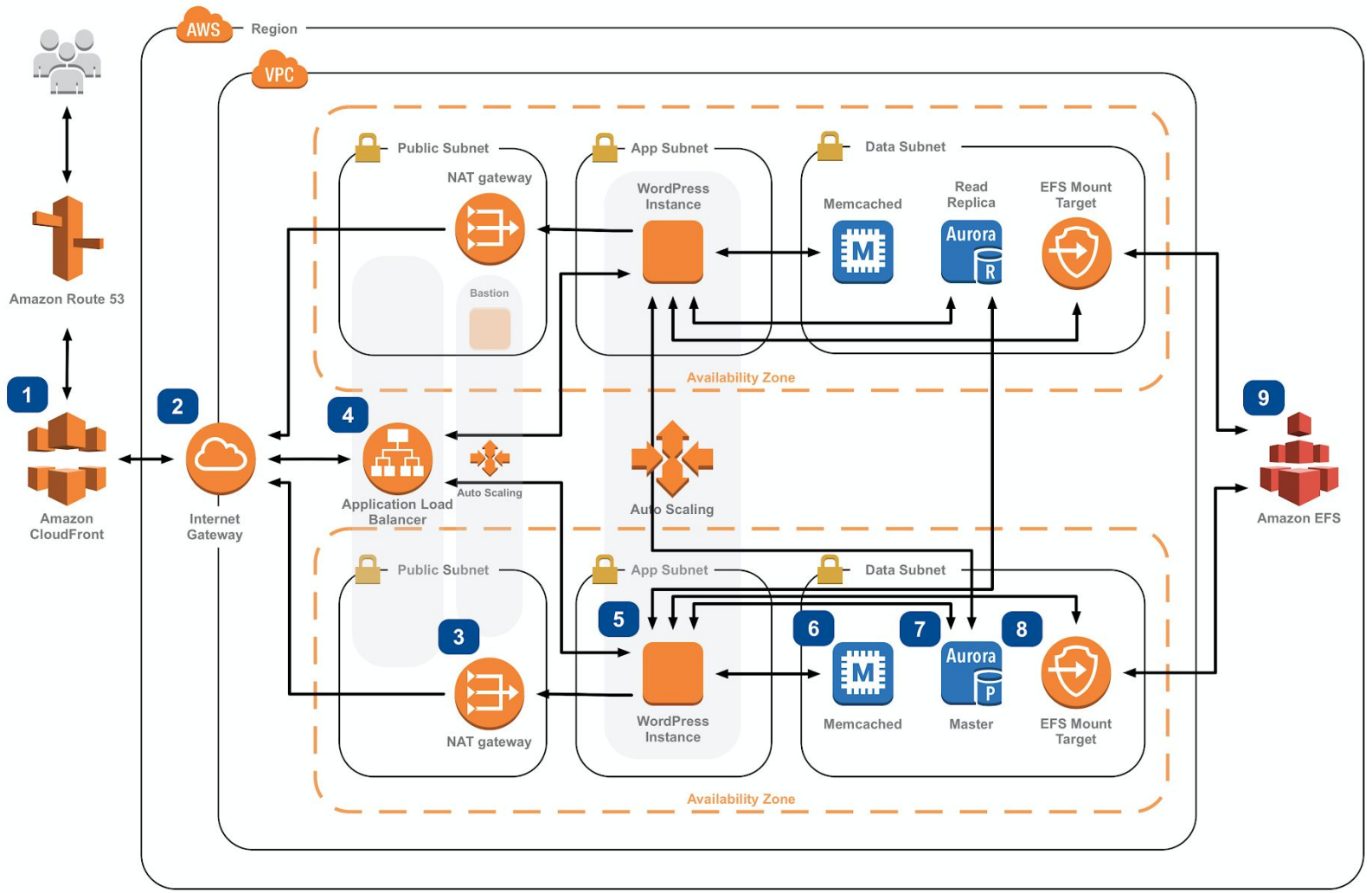


Example: Application Load Balancer

WordPress Hosting

How to run WordPress on AWS

WordPress is one of the world's most popular web publishing platforms, being used to publish 27% of all websites, from personal blogs to some of the biggest news sites. This reference architecture simplifies the complexity of deploying a scalable and highly available WordPress site on AWS.



1. Static and dynamic content is delivered by **Amazon CloudFront**.
2. An **Internet gateway** allows communication between instances in your VPC and the Internet.
3. **NAT gateways** in each public subnet enable Amazon EC2 instances in private subnets (App & Data) to access the Internet.
4. Use an **Application Load Balancer** to distribute web traffic across an Auto Scaling Group of Amazon EC2 instances in multiple AZs.
5. Run your WordPress site using an **Auto Scaling group** of **Amazon EC2 instances**. Install the latest versions of WordPress, Apache web server, PHP 7, and OPcache and build an Amazon Machine Image that will be used by the Auto Scaling group launch configuration to launch new instances in the Auto Scaling group.
6. If database access patterns are read-heavy, consider using a WordPress plugin that takes advantage of a caching layer like **Amazon ElastiCache** (**Memcached**) in front of the database layer to cache frequently accessed data.
7. Simplify your database administration by running your database layer in **Amazon RDS** using either Aurora or MySQL.
8. Amazon EC2 instances access shared WordPress data in an Amazon EFS file system using **Mount Targets** in each AZ in your VPC.
9. Use **Amazon EFS**, a simple, highly available, and scalable network file system so WordPress instances have access to your shared, unstructured WordPress data, like php files, config, themes, plugins, etc.

amazon web services

AWS Reference Architectures

https://github.com/aws-samples/aws-refarch-wordpress

Security aspects of AWS

Certified according to ISO 27001 / C5 (recommendation from BSI) and many other standards: AWS Compliance Programs
European data centers and offices are subject to EU data protection regulations.
Amazon is also subject to the US Patriot Act and the CLOUD Act.

AWS offers services and products to meet security and compliance requirements:

- **Identity and Access Management**: IAM, Single Sign-On, Cognito…
- **Detection**: GuardDuty, Config, CloudTrail…
- **Infrastructure protection**: Shield, Web Application Firewall, Firewall Manager
- **Data protection**: KMS, CloudHSM, Macie…
- **Incident response**: Detective, CloudEndure Disaster Recovery
- **Compliance**: Artifact

Infrastructure as Code

Infrastructure as Code

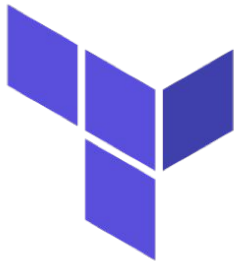
Provisioning and managing entire data centers—not just individual virtual machines
Distinction from configuration management (e.g., Ansible):

- Explicit creation and destruction of the infrastructure of a (virtual) data center
- Immutable infrastructure, instead of continuously modifying existing resources
- Typically declarative rather than imperative
- First introduced for the cloud in 2010 with AWS CloudFormation

Advantages:

- Versioning of the data center, enabling easy staging and rollbacks
- Accelerated delivery of infrastructure changes
- Consistency across environments
- Provides security and auditability of infrastructure in code
- Reusable and modular
- Enables collaboration through code management

Infrastructure as Code with Terraform



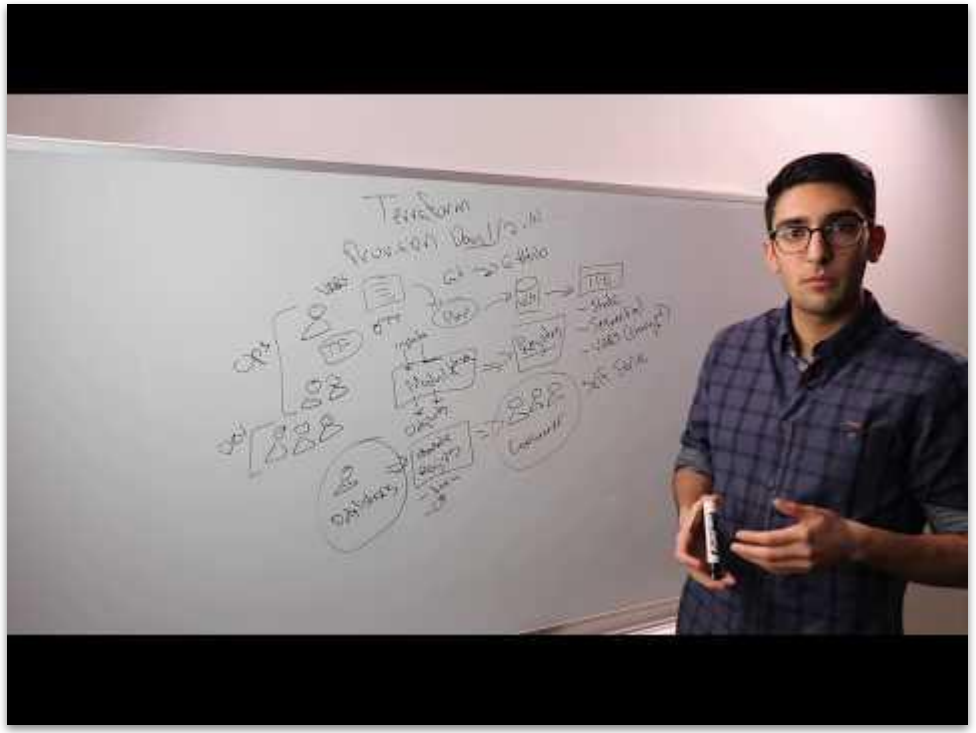
HashiCorp Terraform

"Write, Plan, and Create Infrastructure as Code"

- Since 2014, open source by Hashicorp
- Supports around 40 cloud providers
- Also integrates with database systems, monitoring, and infrastructure software such as Kubernetes
- Wide selection of plugins and reusable modules
- Declarative configuration language
- Commercial extensions available

Terraform Basics

- **Write**: Describe the target state using a domain-specific language, HCL (HashiCorp Configuration Language)
- **Plan** (terraform plan): Determine the current state. Plan necessary changes (ordered and parallelized according to dependencies, minimizing disruptions as much as possible)
- **Apply** (terraform apply): Idempotent creation of the target state. The state (.tfstate file) is usually stored in remote storage (e.g., S3, HTTP, etc.)



Video Pause at 06m:30s, then up to 10m18s:
https://www.youtube.com/watch?v=h970ZBgKINg

Core entities of a Terraform-configuration

Resource: managed infrastructure component

```
resource "aws_instance" "web" {
  ami           = "ami-408c7f28"
  instance_type = "t1.micro"
}
```

Resources have arguments and attributes. Further they define dependencies between each other, forming a directed graph.

Provider: integration to the infrastructure/software provider e.g. AWS

Alicloud	Archive	AWS
Azure	Bitbucket	CenturyLinkCloud
Chef	Circonus	Cloudflare
CloudScale.ch	CloudStack	Cobbler
Consul	Datadog	DigitalOcean
DNS	DNSMadeEasy	DNSimple

```
provider "aws" {
  access_key = "${var.aws_access_key}"
  secret_key = "${var.aws_secret_key}"
  region     = "us-east-1"
}
```

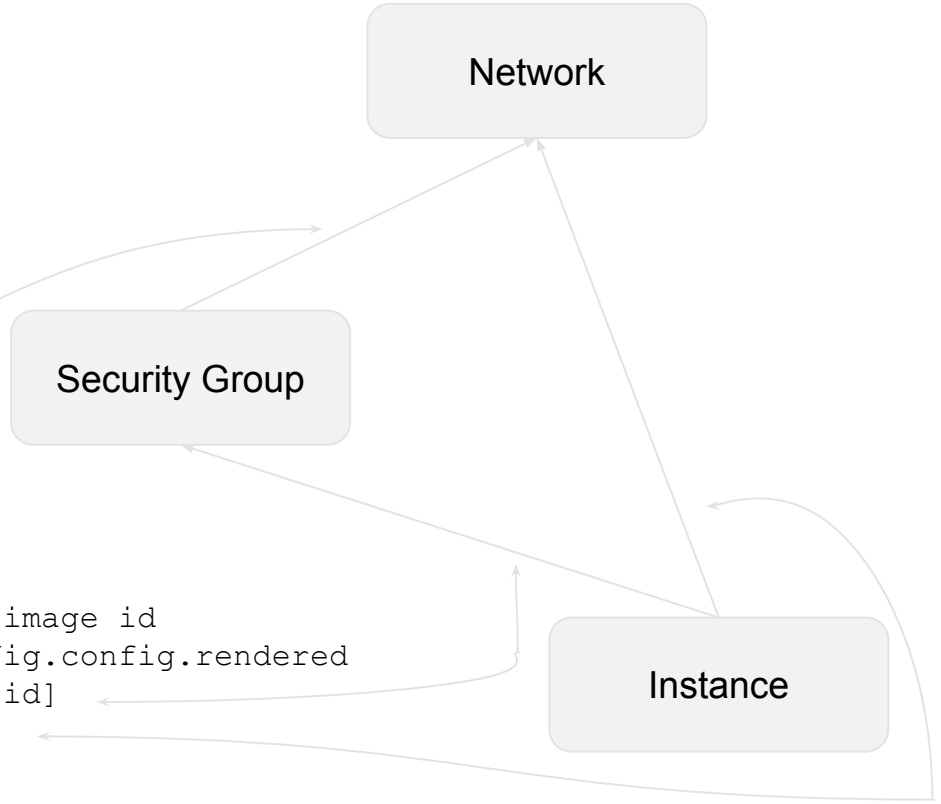
Nomad	NS1	Null
1&1	OpenStack	OpenTelekomCloud
OpsGenie	Oracle Public Cloud	Oracle Cloud Platform
OVH	Packet	PagerDuty
Palo Alto Networks	PostgreSQL	PowerDNS
ProfitBricks	RabbitMQ	Rancher
Random	Rundeck	Scaleway
SoftLayer	StatusCake	Spotinst
Template	Terraform	Terraform Enterprise
TLS	Triton	UltraDNS
Vault	VMware vCloud Director	VMware NSX-T

Provisioner: execute custom actions in the graph locally or remotely. Use only when **absolutely necessary**, in cases the provider does not fulfill your needs.

```
resource "aws_instance" "web" {
  # ...

  provisioner "local-exec" {
    command = "echo ${self.private_ip} > file.txt"
  }
}
```

Example Hashicorp Configuration Language

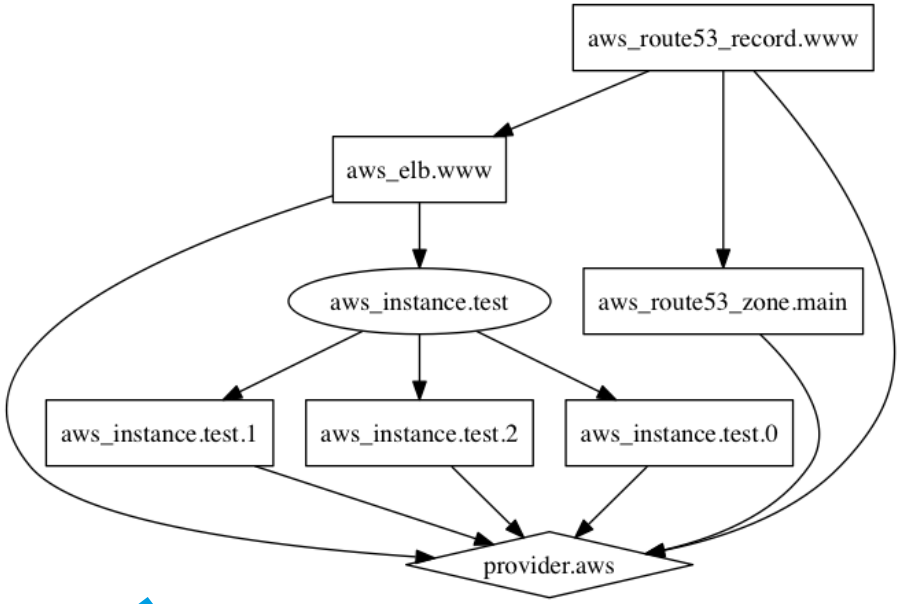
```
module vpc {
  source  = "terraform-aws-modules/vpc/aws"
  version = "2.18.0"
  name    = local.env
  # <shortened>
}

resource aws security group bastion {
  name        = "${local.env}-bastion"
  description = "For Bastion Hosts"
  vpc id      = module.vpc.vpc_id
  # <shortened>
}

resource aws instance bastion {
  instance type          = "t3.nano"
  ami                    = data.aws ami.amazon linux 2.image id
  user data base64       = data.template cloudinit config.config.rendered
  vpc security group ids = [aws security group.bastion.id]
  subnet id              = module.vpc.public_subnets[0]
  # <shortened>
}
```

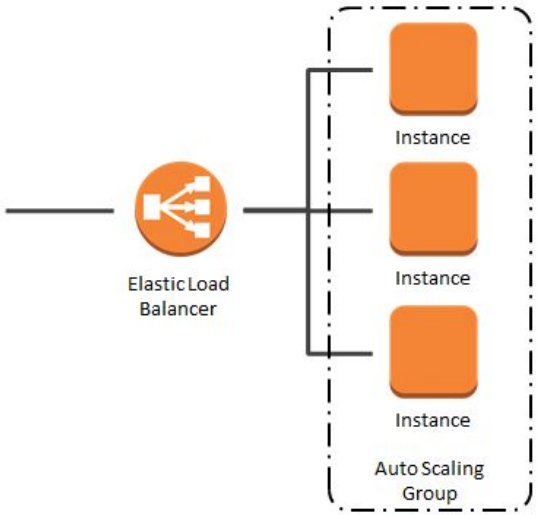


Workflow

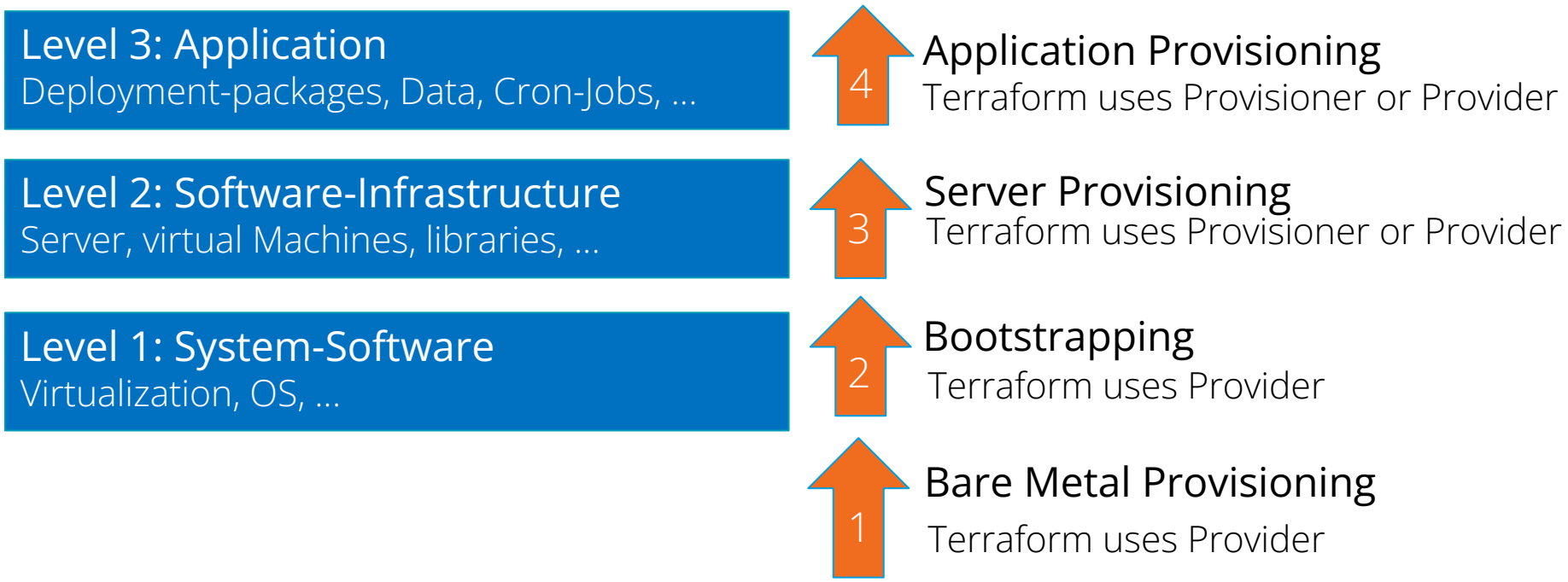
```
resource aws instance bastion {
  instance type           = "t3.nano"
  ami                     = data.aws ami.amazon linux 2.image id
  user data base64        = data.template cloudinit config.config.rendered
  vpc security group ids  = [aws security group.bastion.id]
  subnet id               = module.vpc.public_subnets[0]
  # <shortened>
}
```

```
terraform/
├── main.tf
└── terraform.tfvars
```





Terraform Deployment Levels



Example: Provider

```
provider aws {
  version = "2.56.0"
  region  = "eu-central-1"
 }
```

Example: Data

```
data aws ami amazon linux_2 {
  most_recent = true

  owners = ["amazon"]

  filter {
    name   = "name"
    values = ["amzn2-ami-hvm*"]
  }

  filter {
    name   = "root-device-type"
    values = ["ebs"]
  }

  filter {
    name   = "architecture"
    values = ["x86_64"]
  }
}
```

Example: Resources

```
resource aws instance bastion {
  instance type          = "t3.nano"
  ami                    = data.aws ami.amazon linux 2.image id
  user data base64       = data.template cloudinit config.config.rendered
  vpc security group ids = [aws security group.bastion.id]
  subnet id              = module.vpc.public_subnets[0]
  tags = merge(local.standard tags, {
    "Name" = "${local.env}-bastion"
    }
  )
 }
```

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance

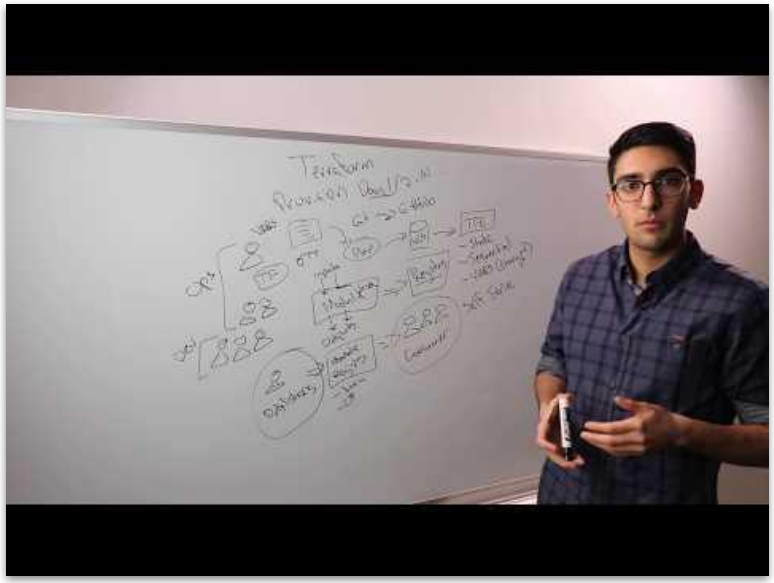
Example: Output

```
output "bastion" {
  description = "Information about the bastion host."
  value = {
    instance         = aws_instance.bastion.id
    dns name         = aws_route53 record.bastion.name
    key_push_policy = aws_iam_policy.allow_bastion_ssh_access.arn
  }
}
```

Collaboration, State and Workspaces

- Terraform is stateful and usually stores this state in the cloud; local storage is possible but not recommended. This allows, for example, the removal of resources in the code to also logically remove the cloud resource. In this case, the reference in the state represents the difference from the target state.
- Remote state locking prevents the application of changes in parallel.
- The state may contain sensitive secrets, such as passwords and certificates.
- Multiple states (workspaces) for the same configuration enable:
  - easy staging, with one workspace per environment
  - completely independent development

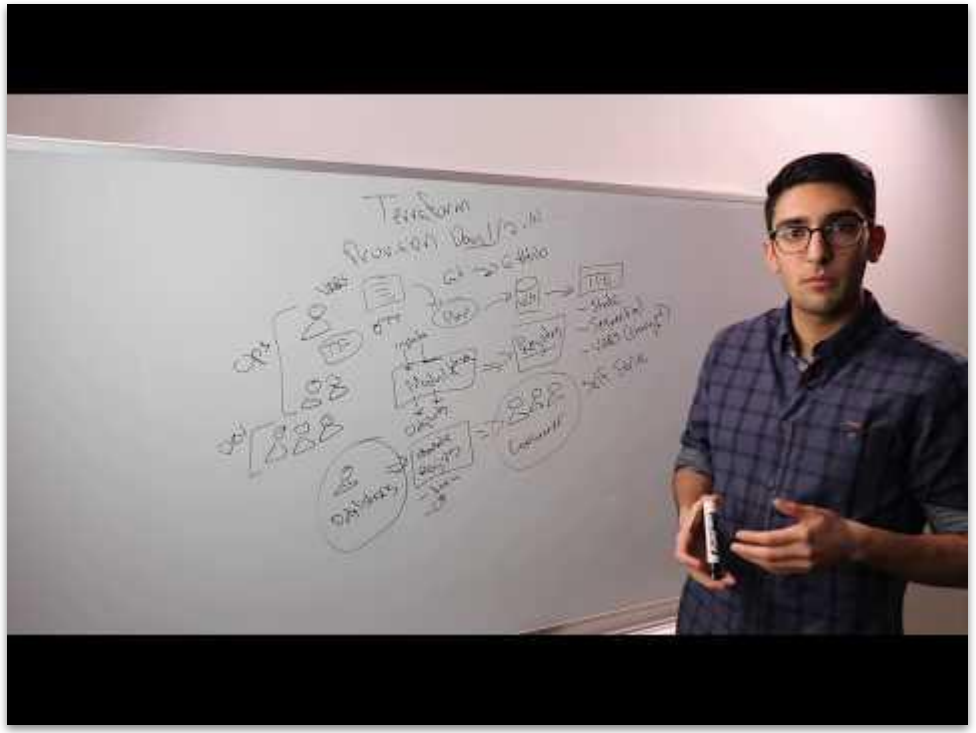
```
terraform {
  backend s3 {
    bucket         = "prj-aws-dev-tfstate-storage"
    key            = "prj-main"
    dynamodb_table = "prj-tfstate-lock"
    region         = "eu-central-1"
  }
}
```



Video ab 10m18s, bis 13m:15s:
https://www.youtube.com/watch?v=h970ZBgKINg

Modules: Reusable abstractions

- akin to a library in the programming language of your choice
- contains complete, “final” Terraform declaration
- contains multiple resource definitions
- “Inputs” are the parameters of the module resource
- “Output” are the attributes of the module resource
- versioned
- published via https://registry.terraform.io or a Git repository



Video ab 13m53s, bis Ende:
https://www.youtube.com/watch?v=h970ZBgKINg

Example: Module

```
module vpc {
  source  = "terraform-aws-modules/vpc/aws"
  version = "2.18.0"
  name    = local.env
  cidr    = "10.0.0.0/16"
  azs     = data.aws availability zones.azs.names
  private subnets        = [ "10.0.0.0/19", "10.0.32.0/19", "10.0.64.0/19" ]
  public subnets         = [ "10.0.96.0/21", "10.0.104.0/21", "10.0.112.0/21" ]
  enable nat gateway     =  true
  single nat gateway     = ! local. config.ha nat gateways
  one nat gateway per az = local. config.ha_nat_gateways
  enable dns hostnames   =  true
  enable dns support     =  true
  tags                   = merge(local.standard_tags, map( "kubernetes.io/cluster/${local.env}", "shared" ))

  enable s3 endpoint =  true
  enable ecr dkr endpoint              =  true
  ecr dkr endpoint private dns enabled =  true
  ecr_dkr_endpoint_security_group_ids  = [aws_security_group.vpc_endpoints.id]

  private subnet tags = {
    "kubernetes.io/role/internal-elb" =  "1"
    "kubernetes.io/role/elb"          =  "1"
  }
 }
```

https://registry.terraform.io/modules/terraform-aws-modules/vpc/aws/latest

Example: Directory structure

For getting your hands dirty

https://github.com/brikis98/terraform-up-and-running-code