Summary of Cloud Computing at University of Bristol 2018 / 2019*

Florian Bauer

January 18, 2019

*This is just a simple summary. I am not responsible for the provided content or anything which belongs to this. If there are any questions please contact me at bauerflorian13@gmail.com .

Contents

Lecture 01: Introduction

Comparison of the internet and electricity network

- starts with everyone has his own (electricity/ computationally power)
- connection between every single users grows
- ends in an all connected world with only a few big services provided by a small number of providers (computationally power goes from the device of the endusers to the cloud, electricity comes from big providers)

Normal Failure

- cloud data centre with 99.999% survival rate
- 500000 server, probability of 100% of the servers are still running after 3 years is 1%.
- **solution**: modular data centres, *servers in container boxes*

Essential Characteristics of Cloud Computing

This definition belongs to NIST's characteristics of Cloud Computing

- **On-demand self service**
- **Broad network access**
- **Ressource pooling**
- **Rapid elasticity**
- **Measured service**

A common stratification: *aaS

Everything as a Service.

- **SaaS**: *Software as a Service*, for instance: everyone
- **PaaS**: *Platform as a Service*, for instance: *Google App Engine, Amazon Appstream*
- **IaaS**: *Infrastructure as a Service*, for instance: *Amazon EC2, S3, Google Compute Engine*

A small number of companies providing IaaS/PaaS s services. Convergence to an oligopoly of less than five providers seems certain.

Lecture 02: Coursework

Just a few informations about the coursework and programming project. May be hopefully not important for the exam...

L03: Economics of Cloud

The basic Economics

- **Cap**ital **Ex**penditure: *Capex*
- **Op**erating **Ex**panditure: *Opex*
- Capex vs Opex: *Why buy a cow if all you need is the milk?*

A typical warehouse scale computer

- *pizzabox* in a *refrigerator* is a server rack
- multiple server racks together are a cluster
- see Figure 1

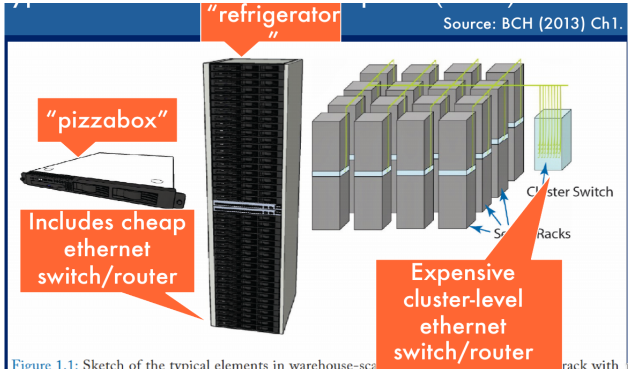

Figure 1: WSC - Warehouse-scale Computer

Energy & Power Efficiency

- cooling cost are around 42%
- optimizing the cooling efficiency will lower the overall costs massivley

Resume

- there is a lot going on *under the hood of a WSC* (WSC = **Warehouse-scale Computer**)
- *prod>>dev*: The innovations are made by and in companies not universitys

L05: *aaS

Definiton see **??**

Why Xaas or *aaS

- avoiding of **Undiffertiated Heavy Lifting**
- the cloud is an ideal environment providing *scale, low cost, automation via Infrastructure-as-Code*

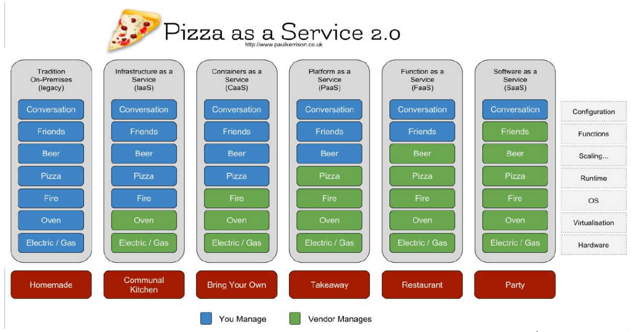

Figure 2: Pizza as a Service Example for *aaS

Structure of AWS Cloud

- **Availability Zones**: cluster of independent data centres, enables **fault isolation** and **high availability**
- **Regions**: entirely independent clouds, consists of a least two *AZ*s, interconnection on global backbone, different regions have different costings

Which Region should I choose?

- **Data souvereignty and compilance**: where to store user data?
- **Proximity of users to data**: where are the most of my users? -¿ lowest latency
- **Services and feature availability**: services and features may vary
- **Cost effectiveness**: each region has different costs (Europe and US are the cheapest)

High Availability & Fault Tolerance

High Availability:

- minimise service downtime by using redundant components
- require components in at least two AZs
- IaaS may have HA, PaaS usually will have HA

Fault Tolerance

- ensure no service disruption by using active-active architecture
- requires service components in at least three AZs
- Iaas is unlikely to offer FT, PaaS some offers FT

AWS Storage options

- Elastic Block Storage: SSDs, Magnetic, NAS, Use: OS, Apps
- S3: durable object storage, very cheap and big
- Instance Storage: on-host storage, very fast, caching
- Elastic File Store: shared storage across AZs

IaaS vs PaaS

- IaaS mainly used by SysAdmins, PaaS mainly used by Developers
- IaaS provides e.g. *VMs,Storage Services,Networking*, PaaS provides e.g. hosted databases, *App deployment and managment env.,test suites*
- IaaS lower cloud costs, PaaS lower human costs

L07: Virtualisation, Containers and Container Orchestration

Virtualisation Basics

- server hardware should be hidden from the user, → user sees only guest OS in a VM and not the host OS
- Amazon offers different VMs (*AMIs*) with Linux or Windows
- VMs are created and run by the *Virtual Machine Monitor (VMM)* aka the **hypervisor**
- VMs can stopped, copied, paused and resumed, which enables **server consolidation**: compress VMs to freeup servers

Types of Virtualisation

Have a look at Figure 3

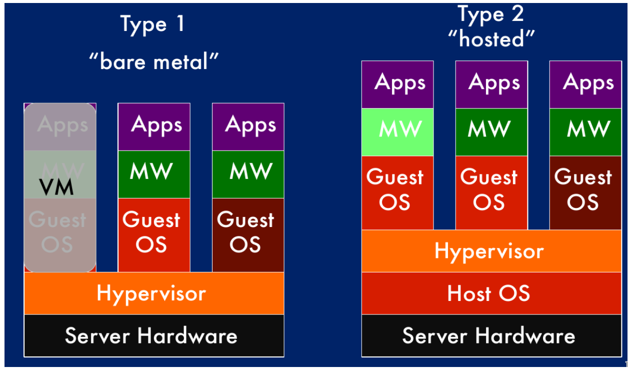

Figure 3: The two different virtualisation types

Xen is an example for Type 1 VMs.

- **Full virtualisation**: complete simulation of underlying guest machine hardware
- **Paravirtualisation**: guest OS can make Syscalls via the hypervisor's API, hypervisor does not simulate hardware

Containerisation: Docker

- package and run application in lightweight, isolated environment
- Docker runs user processes in a super-isolated execution mode
- *operating system level virtualisation* with shared kernel

- Advantage: No need to boot a whole VM
- Disadvantage: Potentially more insecure than complete virtualisation

Docker Objects

- **Images**: read only template with instructions how to create a Docker Container
- **Container**: runnable instance of an image, but ephemeral → all changes not mounted to persistend storage will be lost

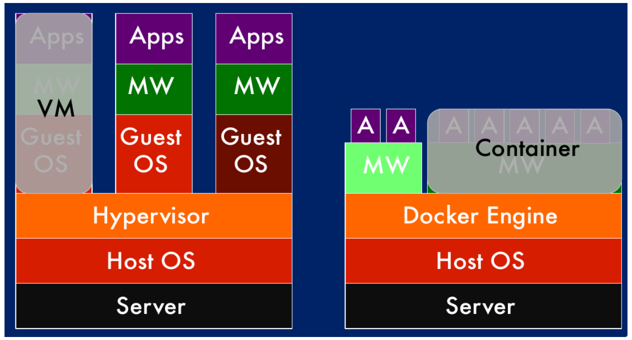

Figure 4: VMs vs Docker architecture schema

Container Orchestration: Kubernetes

Motivation

- To run containers at scale needs managment tools
- **(Horizontal) Auto-scaling on demand**
- **Fault Tolerance**
- **Manage Accessibility from the web**
- **update/rollback without downtime**

Featues of Kubernetes

- **Automated scaling**
- **Self healing**
- **Horizontal scaling**
- **Service discovery and Load Balancing**
- **Automated Rollbacks/Rollouts**

Kubernetes Components

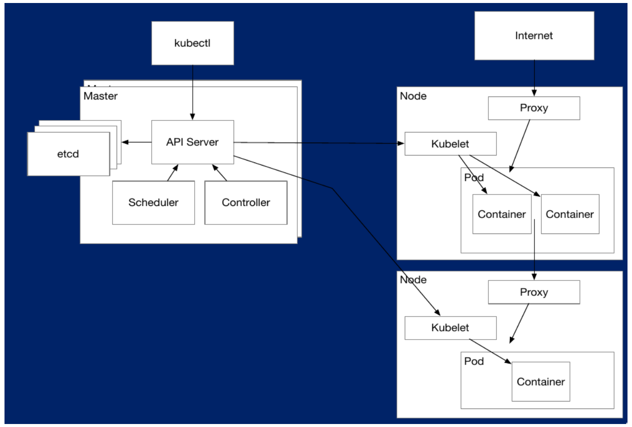

Figure 5: Components of the Kubernetes architecture

- **Master**: manages the cluster state, subcomponents: **API Server**, **Controller**, **Scheduler**, writes to *etcd*
- **Nodes**: run work in pods, **Pods** are the scheduling unit, **Kubelet** is the agent to communicates with master, **Kube-proxy** is the network agent
- **Kubeclt**: local cli to controll cluster
- **Etcd**: distributed key-value store
- **Deployments**: **Replica Sets**, balances the number of running and scheduled pods; deployments provide update to Pods or ReplicaSets
- **Services**: groupings of pods which can be referred by a name, Unique IP and DNS name; Pods in Services are load balanced

L09: Serverless

Definiton: *The essence of the serverless trend is the* ***absence*** *of the server concept during software development.*

Abstractions of App Deployment

- *More Abstraction*: more control and trust to given platform
- *Less Abstraction*:more undifferentiated heavy lifting

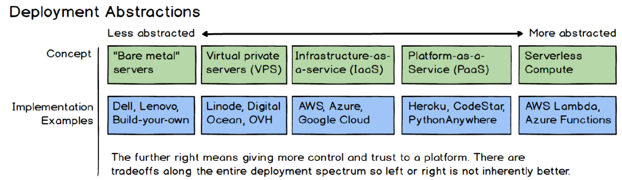

Figure 6: Deployment abstractions: More vs less abstraction

The four pillars of serverless

- No server managment
- Flexible Scaling
- High Availability
- Never Pay for Idle

Serverless FaaS: AWS Lambda

- Triggered by an event
- typically invoked in a few ms (warm start)
- Cold start issue: code that hasn't been used for a while takes longer to start

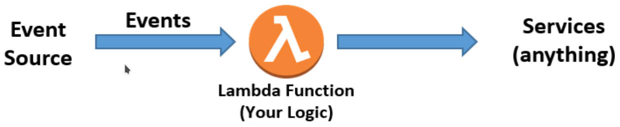

Figure 7: AWS Lambda: Event Triggers

The four stumbling blocks of serverless

- Performance Limitations
- Vendor Lock-in
- Monitoring and Debugging
- Security and Privacy

Serverless usecases

- Event-driven data processing (resize uploaded images)
- Serverless webapplication (simple 3-tier app)
- Mobile and IoT Apps (Airbnb smart home)
- Application Ecosystem (Alexa Skill)
- Event Workflow (image recognition and processing)

L11: Scalable Systems

The Scale Cube

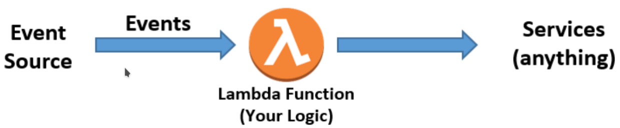

Figure 8: AWS Lambda: Event Triggers

- x-axis: **Horizontal Duplication**, unbiased cloning of services and data
- y-axis: **split by function or service**:refers to isolation (making different services)
- z-axis: **partitioning the domain of incoming requests**:data-partitioning, split relevant to client (example: All customers from A-F are together processed, all customers from G-M, etc)

Software architectures

- set of structures needed to reason about the system
- might be implicit

Architectural Components and Patterns for scalable systems

- **Decoupled Components**: allows independent scalability of components; mechanisms to decouple:
  - load balancers
  - message queues
  - message topics
  - service registry
- **Load Balancers**: distributing requests, hiding the server from client access, manage availability (HA),session affinity/sticky sessions
- **Session affinity/sticky sessions**:cookies managed by load balancer(duration based), cookies managed by application cookie
- **LB Algorithms**:(Weighted) Round Robin, Least connections
- **Message Topics**:messages are immeditaley pushed to subscribers, decouple producers and subscribers, concurrent processing
- **Message Queues**: Asynchronous: queue it now but run it later; seperates application logic; introduces latency
- **Service Registries**: resolve addresses for names, Leader voting (*Byzantine Generel*)
- **Automation**: autoscaler as sclaing can not be done manually (Metrics are CPU, RAM, Memory)
- Architectural Patterns: Service oriented architectures; APIs are cloud requirement

L13: MapReduce and GFS/HDFS

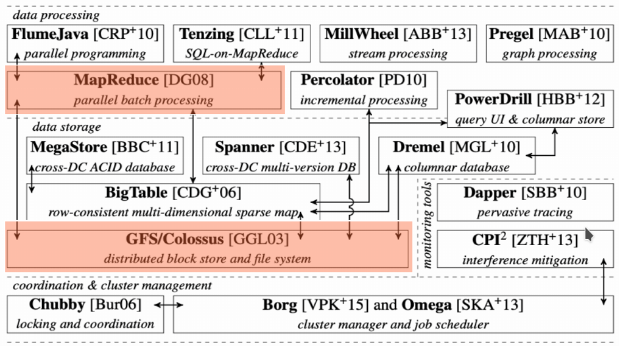

Figure 9: The Google Technology Stack

MapReduce: Basics

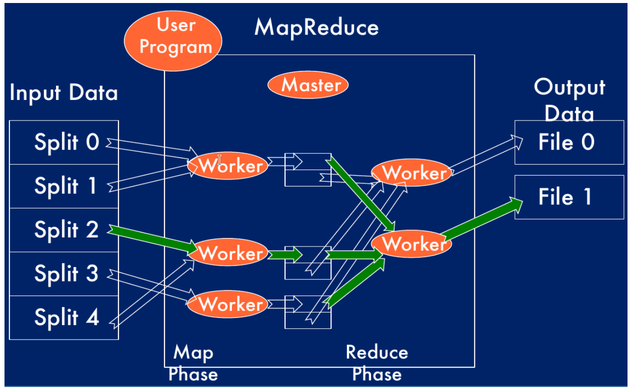

Figure 10: The MapReduce Technology

- we have some input data
- *Map phase*: master process assigns worker processes their part of the data, the data is than processed
- *Reduce phase*: other worker processes collect the processed data and reduce them

AS the master pings the worker and a failure would be noticed really fast. This can now be handled by assigning other processes the task of the failed process.

GFS - Google File System

GFS Objects

- TODO

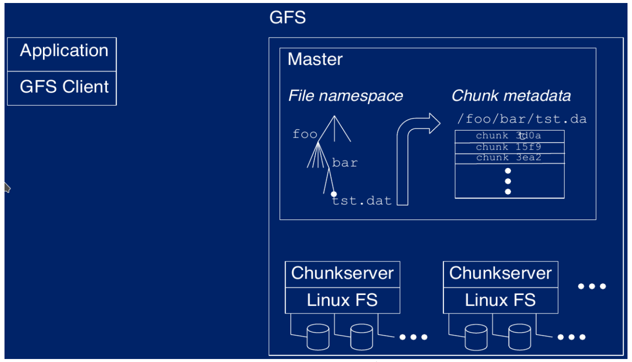

Figure 11: The GFS Architecture

TODO

L14: CAP, Paxos, BGP

-

L15: The Hadoop Ecosystem

L16: Spark and In-Memory Methods

L17: NoSQL

L18: Graph Databases

L19: NewSQL & Event Stream Processing

L20: Cloud Security

L21: DevOp

Todo...