Methods of Cloud Computing

Platforms / Platform-as-a-Service

Complex and Distributed Systems
Faculty IV
Technische Universität Berlin

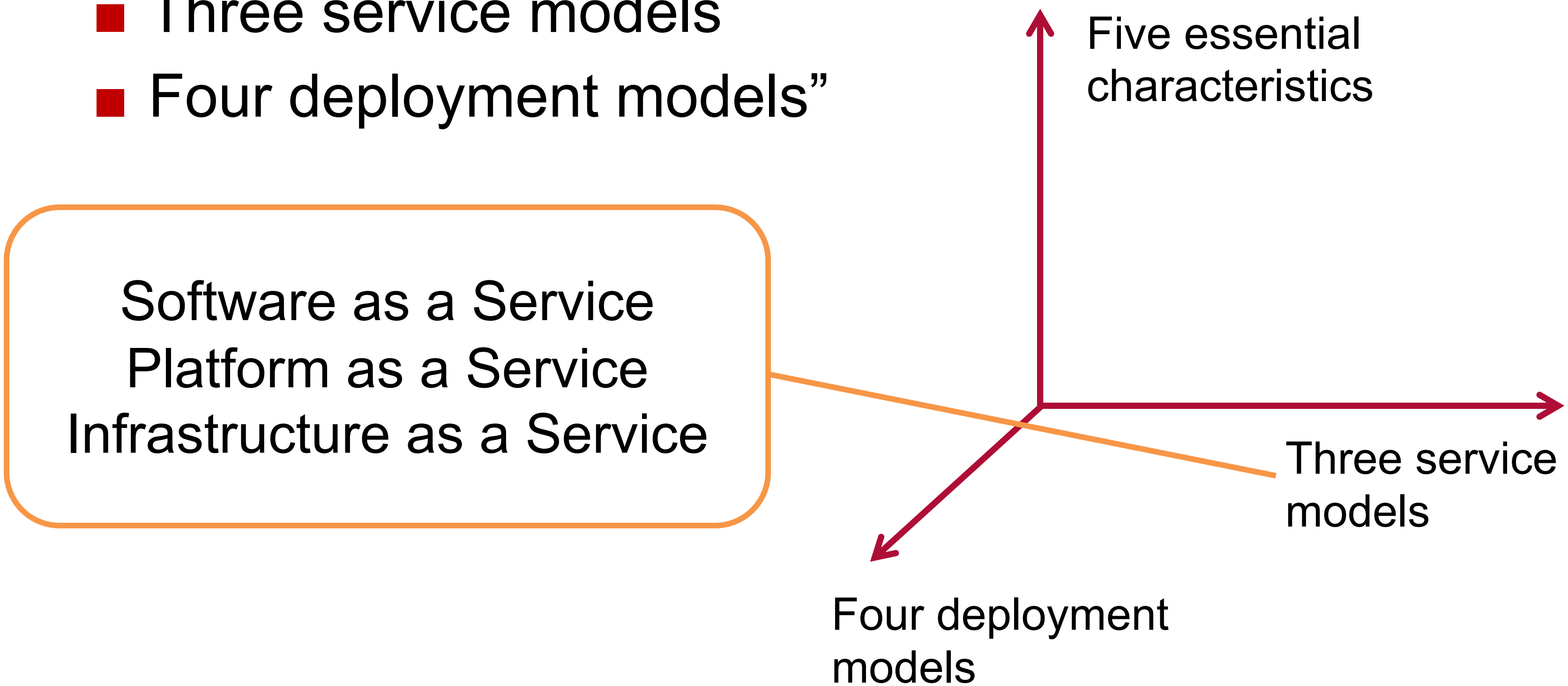
Operating Systems and Middleware
Hasso-Plattner-Institut
Universität Potsdam

Overview

- **Intro**
  - Cost of Scalability
  - Platforms vs Infrastructure
  - Abstraction Levels of Platforms
- Platforms
  - Azure
  - Amazon EMR and SageMaker
- Serverless Computing
  - Concept, FaaS, BaaS
  - Serverless Architectures and Implications
  - Examples

Dimensions of Cloud Computing (NIST)

- NIST: “Cloud model is composed of
  - Five essential characteristics
  - Three service models
  - Four deployment models”



Overview

- Intro
  - **Cost of Scalability**
  - Platforms vs Infrastructure
  - Abstraction Levels of Platforms
- Platforms
  - Azure
  - Amazon EMR and SageMaker
- Serverless Computing
  - Concept, FaaS, BaaS
  - Serverless Architectures and Implications
  - Examples

Cost of Scalability

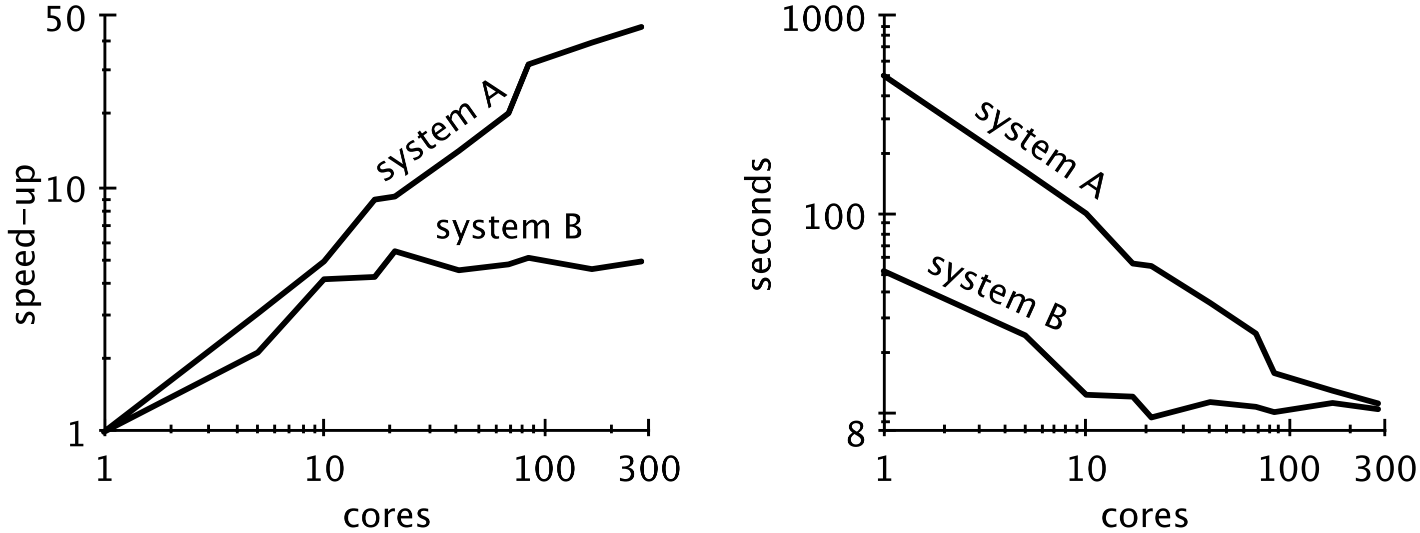
- Easiest way to reach good scalability: Have a terrible baseline speed
- When does distributed processing actually increase net performance?

Scalability! But at what COST?

Frank McSherry	Michael Isard	Derek G. Murray
Unaffiliated	Unaffiliated*	Unaffiliated†

- COST: Configuration that Outperforms a Single Thread
- Paper from 2015 [1]

COST of Scalability: Illustrating Example



- System A scales better than System B
- Actually System B is the optimized version of System A
- Scalability is worse because of better performance!

COST of Scalability: PageRank Speed

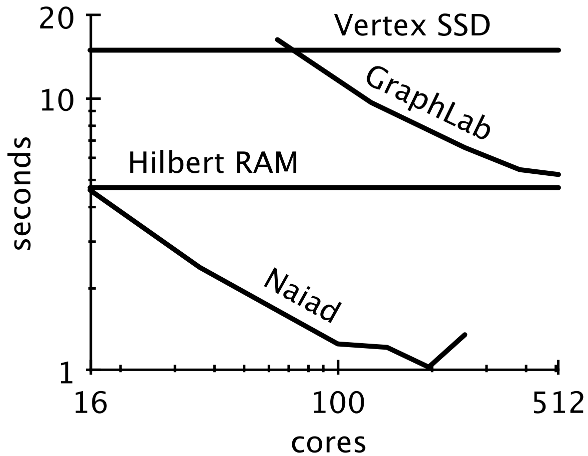
- Speed of a single-threaded implementation is compared to speed from literature on two datasets:

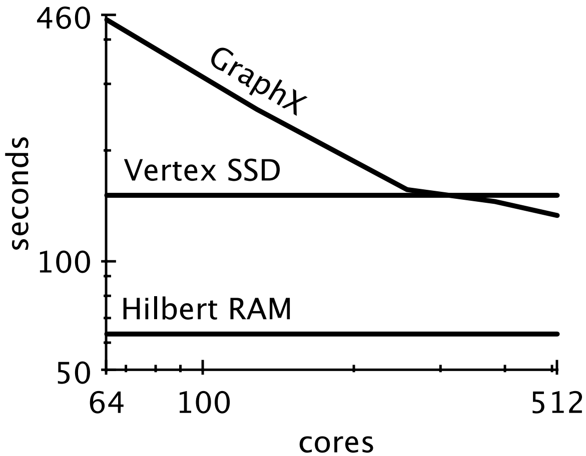
name	twitter_rv [13]	uk-2007-05 [5, 6]
nodes	41,652,230	105,896,555
edges	1,468,365,182	3,738,733,648
size	5.76GB	14.72GB

- Single threaded implementation outperforms all systems on 20 iterations of PageRank

scalable system	cores	twitter	uk-2007-05
GraphChi [12]	2	3160s	6972s
Stratosphere [8]	16	2250s	-
X-Stream [21]	16	1488s	-
Spark [10]	128	857s	1759s
Giraph [10]	128	596s	1235s
GraphLab [10]	128	249s	833s
GraphX [10]	128	419s	462s
Single thread (SSD)	1	300s	651s
Single thread (RAM)	1	275s	-

COST of Scalability: Scaling measurements





Naiad has COST of 16 Cores, GraphX unbounded COST

→ When to use scalable systems is an important decision

Also: Amdahl's Law, Gustafson's Law

COST of Flink (1/4)

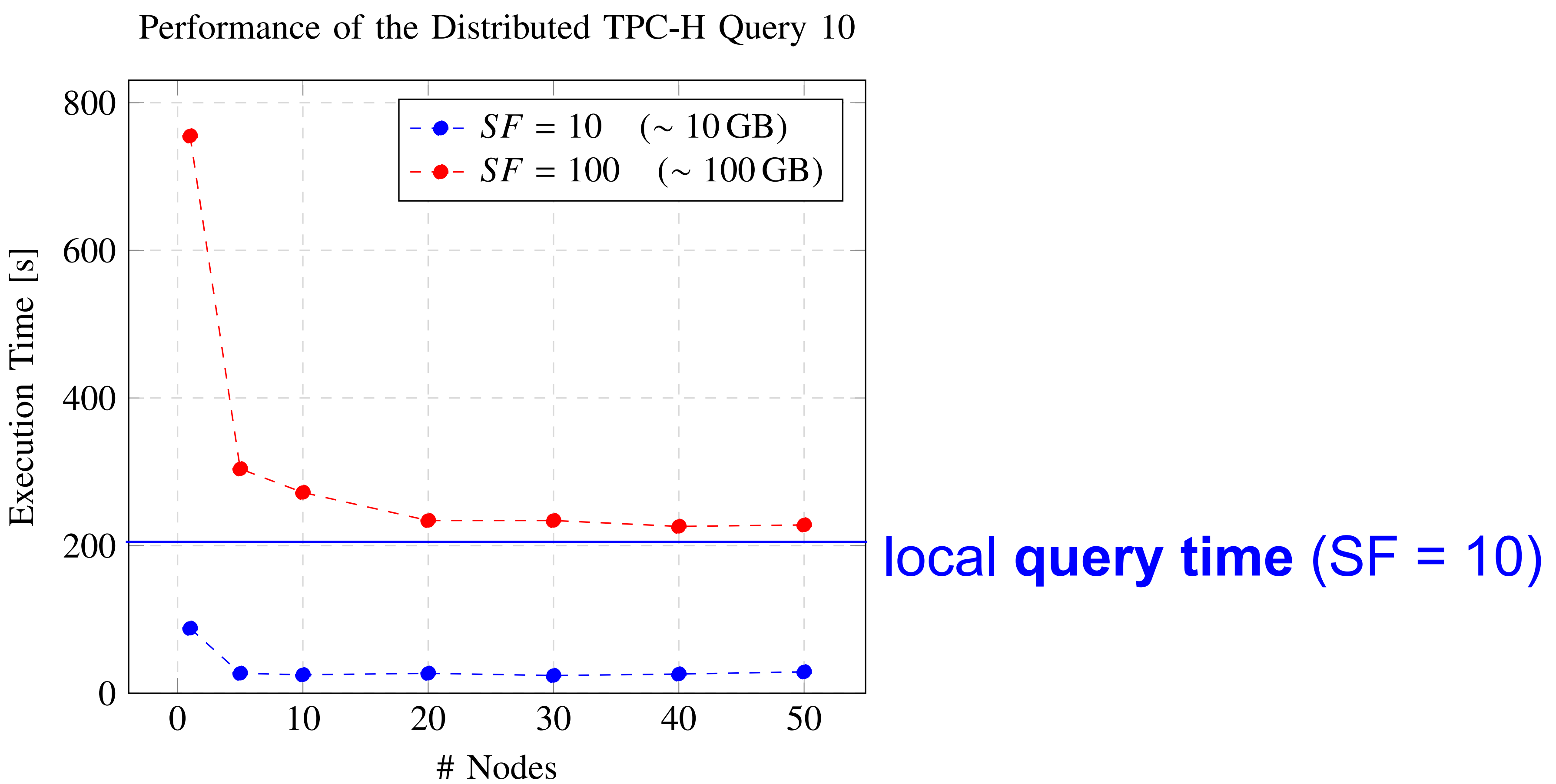
- Bachelor thesis at CIT and a paper in 2016 [6]

When to Use a Distributed Dataflow Engine: Evaluating the Performance of Apache Flink

Ilya Verbitskiy*, Lauritz Thamsen†, Odej Kao†
Technische Universität Berlin
*ilya.verbitskiy@campus.tu-berlin.de

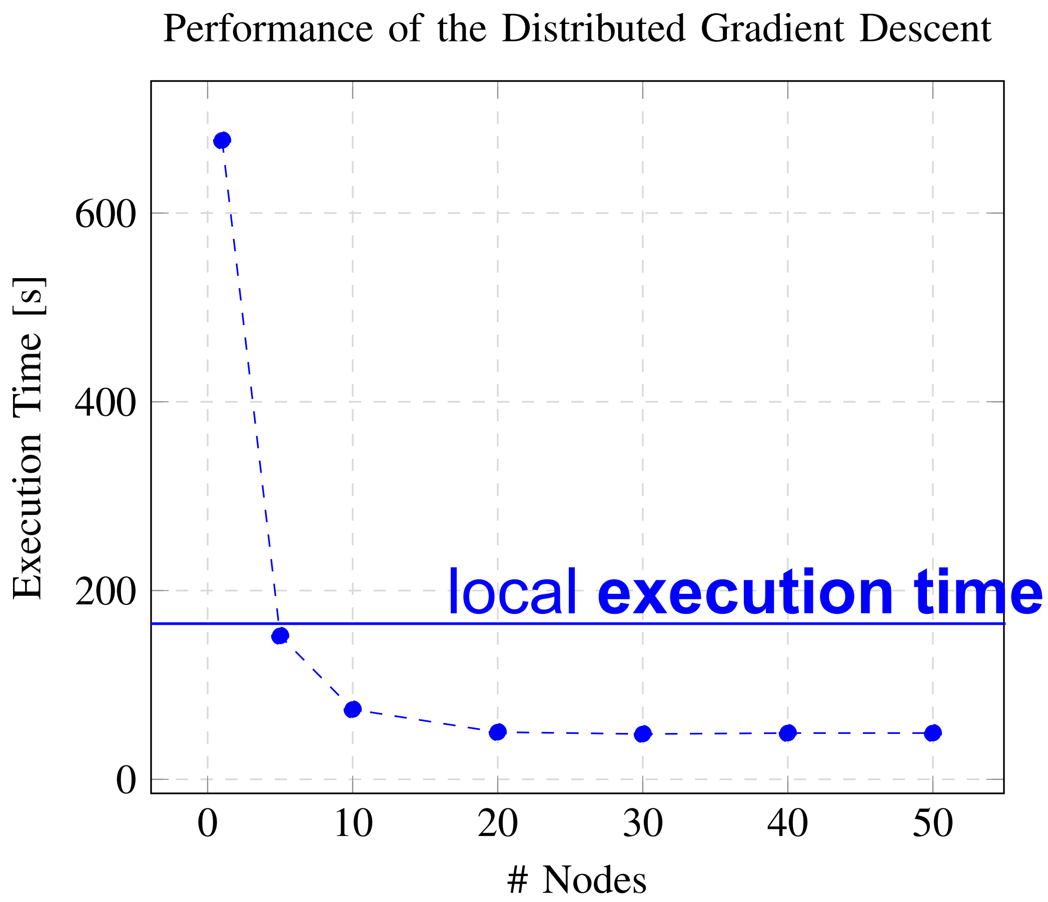
- 50 nodes commodity cluster (Intel Xeon X3450, 4 cores, 16 GB RAM) vs. Macbook (Intel Core i5 5257U, 2 cores, 8 GB RAM)

COST of Flink (2/4)



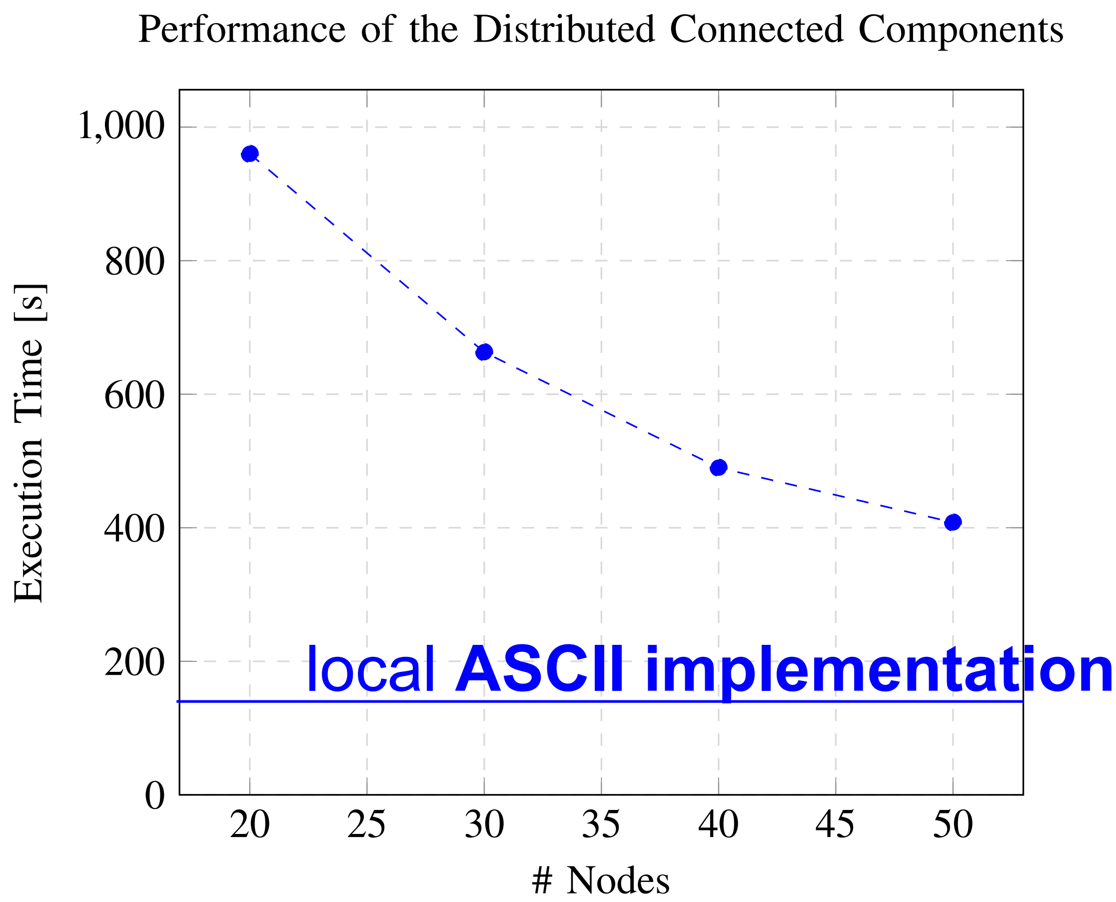
- Even single-node Flink outperforms the single thread

COST of Flink (3/4)



- Flink quickly outperforms the local implementations

COST of Flink (4/4)



- Not even 50 nodes outperform single-threaded impl.

Overview

- Intro
  - Cost of Scalability
  - **Platforms vs Infrastructure**
  - Abstraction Levels of Platforms
- Platforms
  - Azure
  - Amazon EMR and SageMaker
- Serverless Computing
  - Concept, FaaS, BaaS
  - Serverless Architectures and Implications
  - Examples

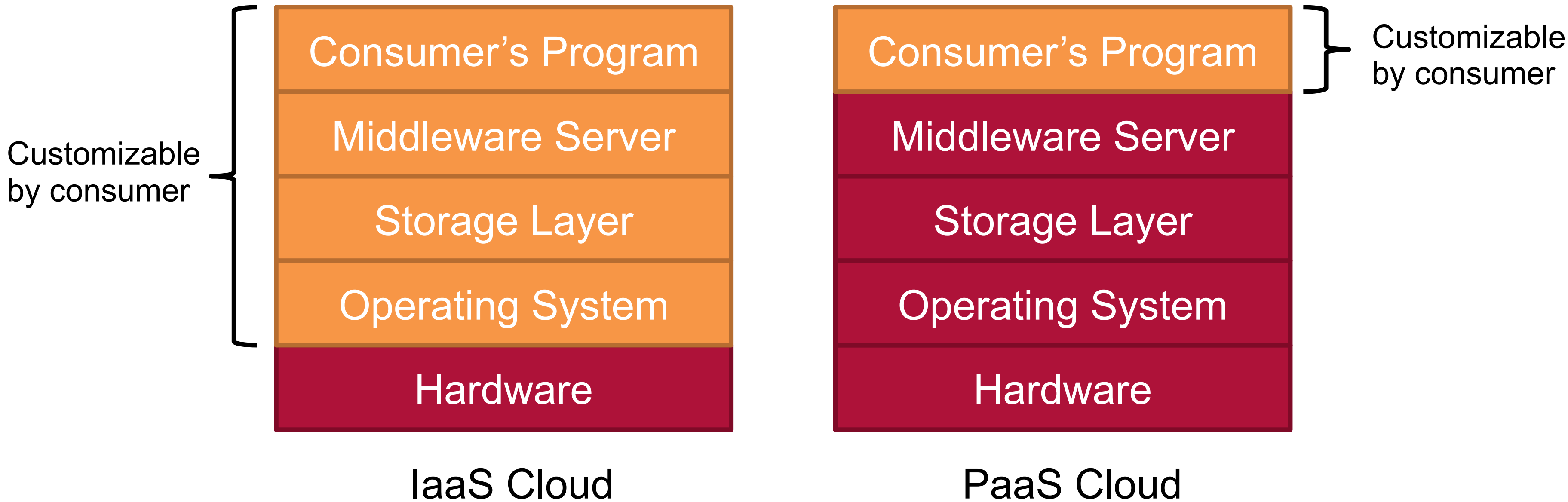
Recap: PaaS Clouds

- Services offered by PaaS clouds (according to NIST[2])
  - Programming languages
  - Libraries
  - Services
  - Tools

- Characteristics of services
  - Consumer can deploy custom applications on PaaS cloud using the provider's application model
  - Consumer does not directly control the operating system, storage, and deployed runtime

IaaS vs. PaaS (1/2)

- PaaS offers higher abstraction level compared to IaaS
  - Less development/maintenance effort
  - Less flexibility, high provider dependence



IaaS vs. PaaS (2/2)

- Higher abstraction level enables other pricing models
- Service usage can be charged
  - by time
  - per query (e.g., for database services)
  - per message (e.g., for message queues)
  - per CPU usage (e.g., for request-triggered applications)
  - …
- Enables interesting scenarios for consumers
  - Deployed applications can have very low operational costs until they become popular

Overview

- Intro
  - Cost of Scalability
  - Platforms vs Infrastructure
  - **Abstraction Levels of Platforms**
- Platforms
  - Azure
  - Amazon EMR and SageMaker
- Serverless Computing
  - Concept, FaaS, BaaS
  - Serverless Architectures and Implications
  - Examples

PaaS Abstraction Levels

- Many levels of abstractions in PaaS
  - Execution environments (like on Heroku)
  - Databases
  - Distributed processing
  - Domain-specific workbenches (like SageMaker)
  - Complete services (like Rekognition)*

- A lot of new offerings are quite high-level PaaS

*almost SaaS, but mostly used by developers

PaaS Value Proposition

1. Maintenance
   - Hardware maintenance
   - OS patches
   - Middleware updates
2. Availability
   - Application/service will be available despite maintenance/outages
3. Scalability
   - Application/service will scale to thousands of concurrent users

Overview

- Intro
  - Cost of Scalability
  - Platforms vs Infrastructure
  - Abstraction Levels of Platforms
- **Platforms**
  - Azure
  - Amazon EMR and SageMaker
- Serverless Computing
  - Concept, FaaS, BaaS
  - Serverless Architectures and Implications
  - Examples

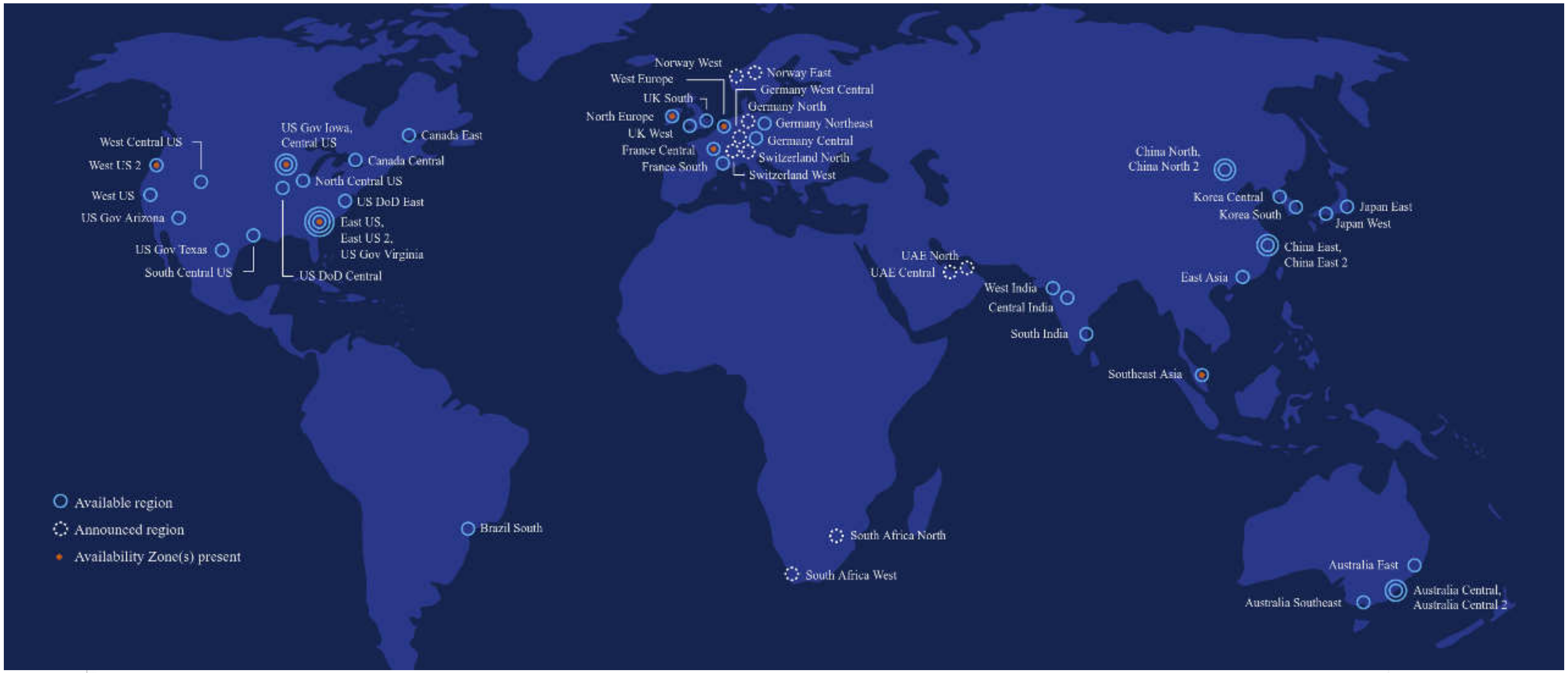
Overview

- Intro
  - Cost of Scalability
  - Platforms vs Infrastructure
  - Abstraction Levels of Platforms
- Platforms
  - **Azure**
  - Amazon EMR and SageMaker
- Serverless Computing
  - Concept, FaaS, BaaS
  - Serverless Architectures and Implications
  - Examples

Microsoft Azure

- Microsoft‘s cloud computing platform
- Launched in 2010 as PaaS solution
  - Targeted at scalable, reliable web applications
- Initially, platform was composed of three components
  - Microsoft Azure: Compute and storage services
  - SQL Azure: Cloud-based DBMS
  - Azure AppFabric: Tools to bridge gap between local and cloud-hosted applications

Geographic Distribution of Azure Data Centers



- Microsoft calls each geographic location a *region*
  - Within a region, customers can create *affinity groups* to deploy services as closely together as possible

Microsoft Azure Affinity Groups

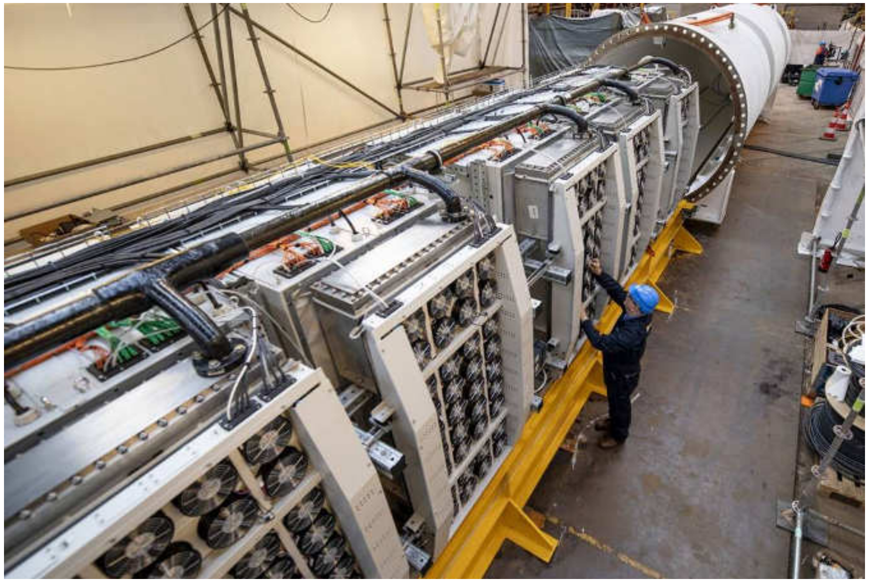
- Some data centers are built from shipping containers
  - Each container can contain up to 2500 servers

Images from [3]

- Affinity group: Logical grouping of components
  - Azure tries to deploy components as closely as possible
  - For example within the same container, if possible

Microsoft Azure Affinity Groups

- Project Natick tries to go further in global distribution with underwater data centers



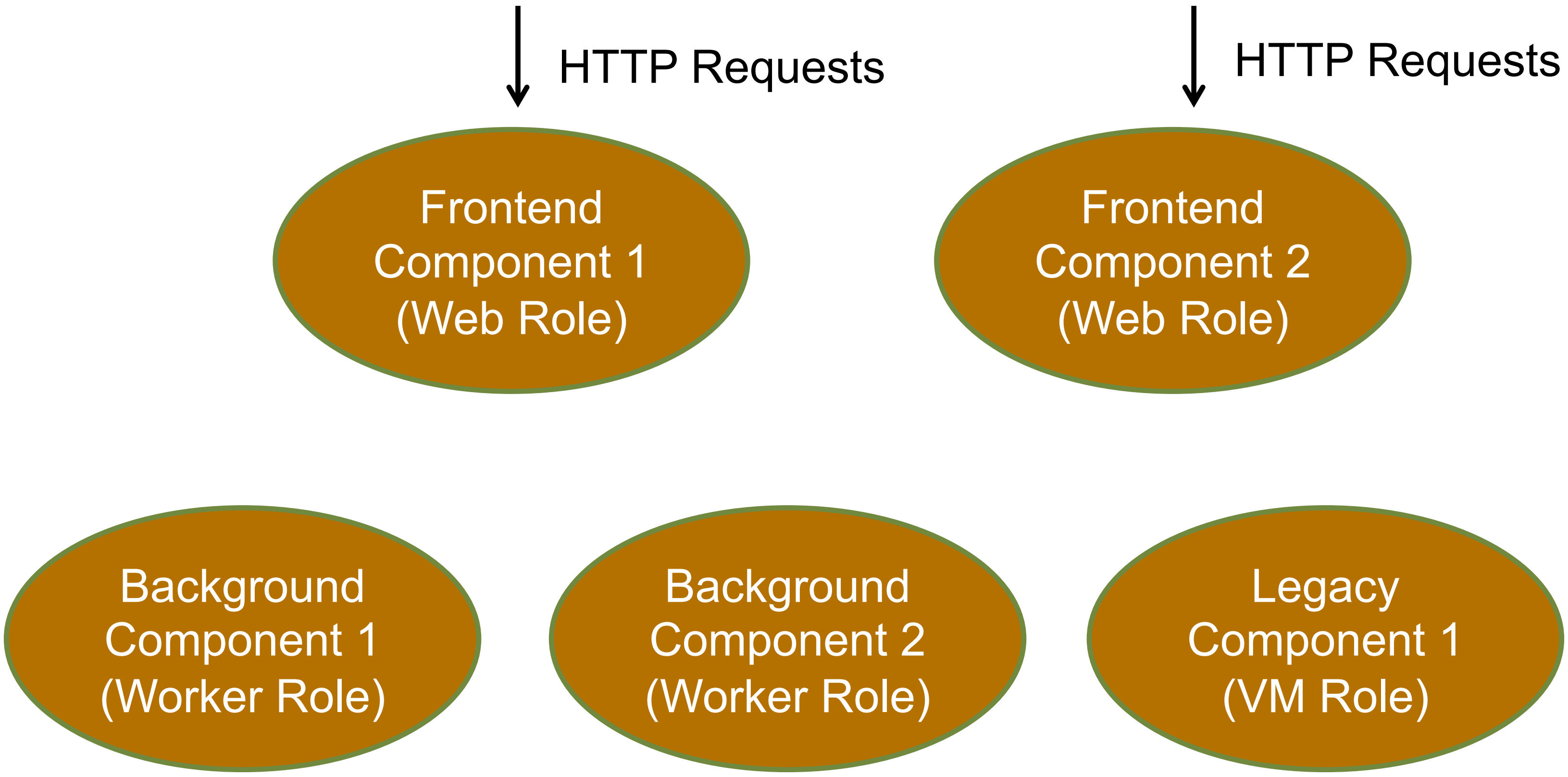
Microsoft Azure Programming Model (1/4)

- Azure applications separated into logical components
  - Each component as assigned to a so-called role
  - Multiple instances of the same component might exist

- Azure programming model offers three roles
  - Web role: Components facing the outer world
    - Accepting requests via HTTP, e.g. Web sites/services
  - Worker role: Components doing background tasks
  - VM role: Legacy components
    - Component which cannot be converted to Azure's programming model

Microsoft Azure Programming Model (2/4)

- Example of an Microsoft Azure application according to the programming model



Microsoft Azure Programming Model (3/4)

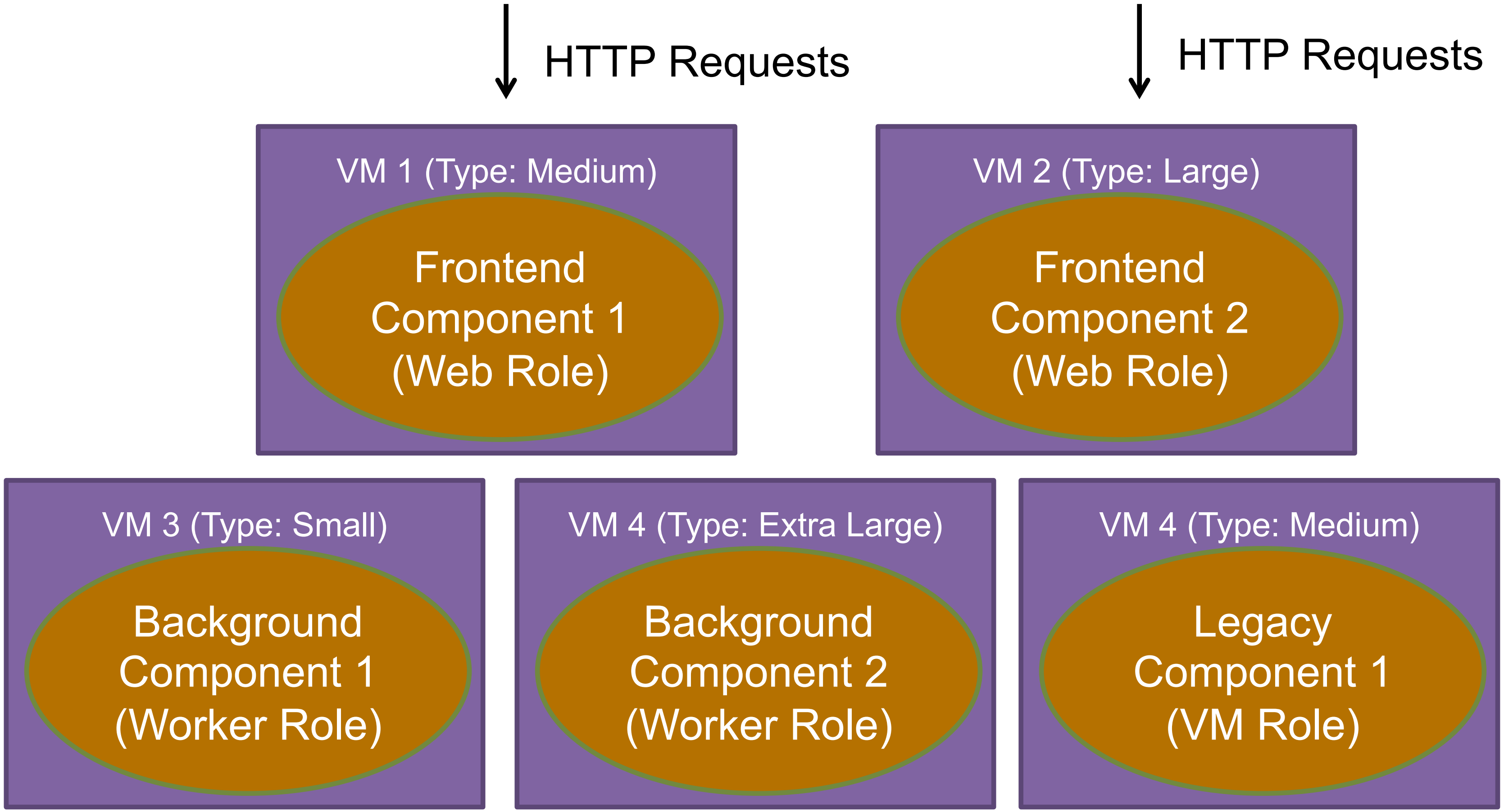
- Each instance of a component is executed in a separate virtual machine
  - Improves maintainability and isolation
  - Customer can choose between different types of VMs
  - VM usage is billed by the hour (like IaaS model)

Virtual Machine Size	CPU Cores	Memory	Price per Hour
Extra Small	Shared	768 MB	USD 0.02
Small	1	1.75 GB	USD 0.12
Medium	2	3.5 GB	USD 0.24
Large	4	7 GB	USD 0.48
Extra Large	8	14 GB	USD 0.96

Overview of VM types and pricing from https://azure.microsoft.com/de-de/pricing/ /

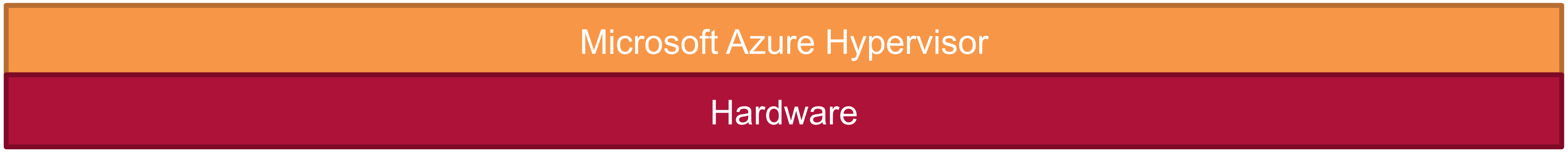
Microsoft Azure Programming Model (4/4)

- Example application with virtual machine isolation



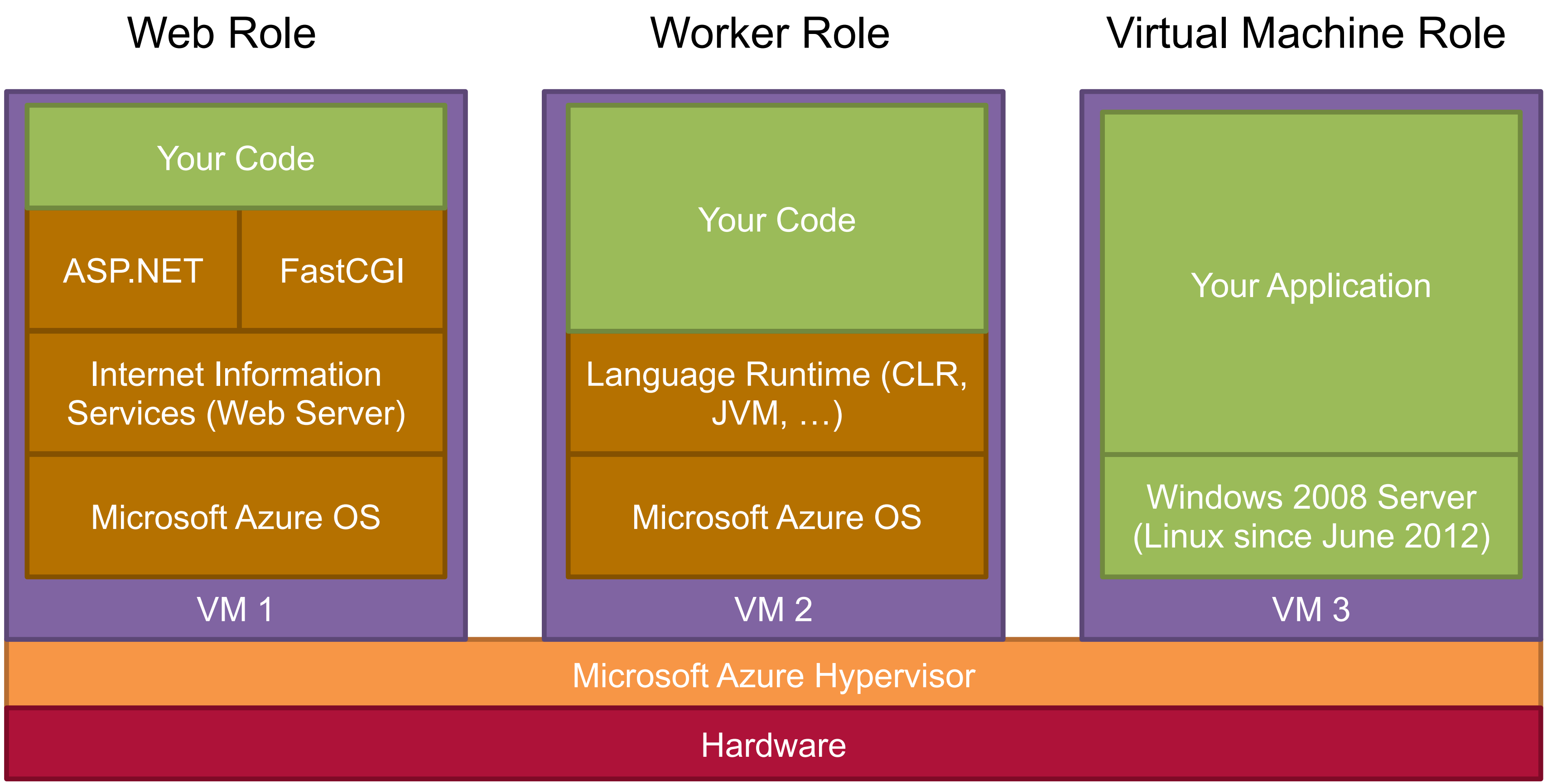
Why Are There Different Roles (1/2)?

- Different roles offer different levels of abstraction
- Some components are the same across all roles
  - Hardware
    - Commodity servers (physical specs are not disclosed)
    - MS tries to keep it as homogeneous as possible
  - Microsoft Azure Hypervisor
    - Few details known, but it is not Hyper-V
    - Relies on homogeneous HW to improve performance
    - Support for 2nd gen. hardware virtualization (NPT, EPT)
    - Only executes signed and authorized components



Why Are There Different Roles (2/2)?

- Different roles offer different levels of abstraction



Supported Programming Languages (1/2)

- Azure is basically a standard Windows environment
  - Every programming language for Windows Server also runs on Microsoft Azure
  - However, there are differences in the levels of integration

- Languages supported in Web Role
  - .NET
  - Node.js (through IIS extension)
  - Every language supporting the FastCGI interface
    - PHP, Ruby, …

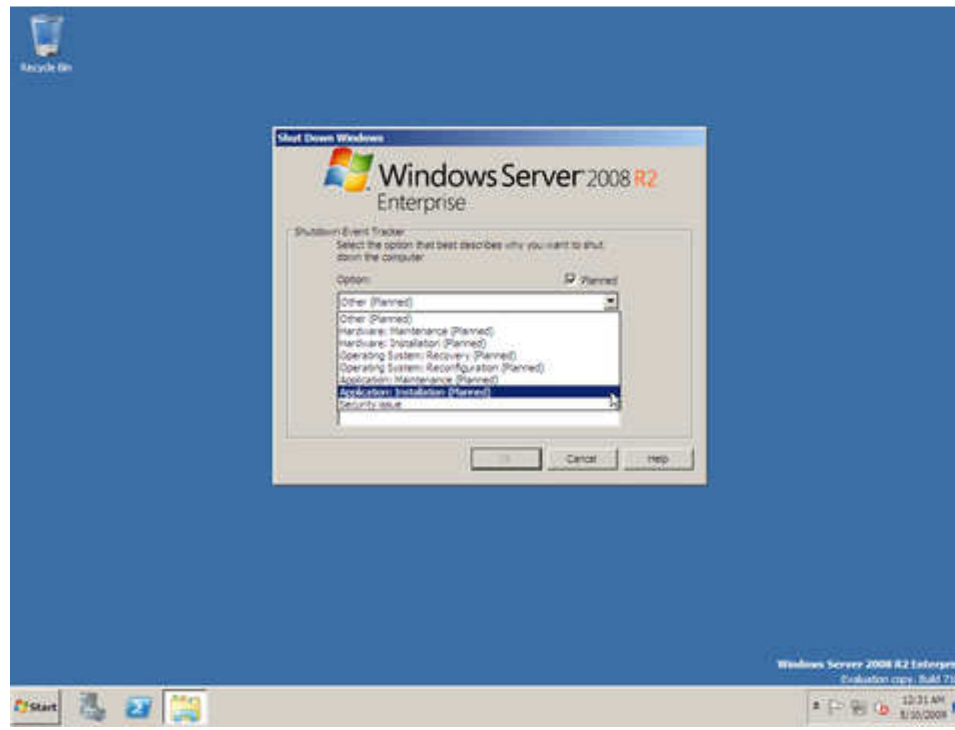
Supported Programming Languages (2/2)

- Languages supported in Worker Role
  - .NET
  - Possibility to run arbitrary Windows binaries
    - C++, Java, …
    - Java support for Worker Role
  - Also possible to bundle additional software with code
    - Example: Create Worker Role with Java code
    - Bundle Tomcat application server
    - Java application server on Azure with similar behavior to standard Web Role

Entry Points into Microsoft Azure Components (1/3)

- VM Role entry point: Precompiled VM image

- Azure is agnostic of guest OS
  - No automatic patches or updates
  - Basically IaaS abstraction
  - Intended for legacy application only



Entry Points into Microsoft Azure Components (2/3)

- Worker Role entry point: Archive with intermediate code
  - Code must implement particular interface

```
namespace WorkerRole1
{
    public class WorkerRole : RoleEntryPoint
    {

        public override void Run()
        {
            // Code to be executed during role‘s life time
        }

    }
}
```

Entry Points into Microsoft Azure Components (3/3)

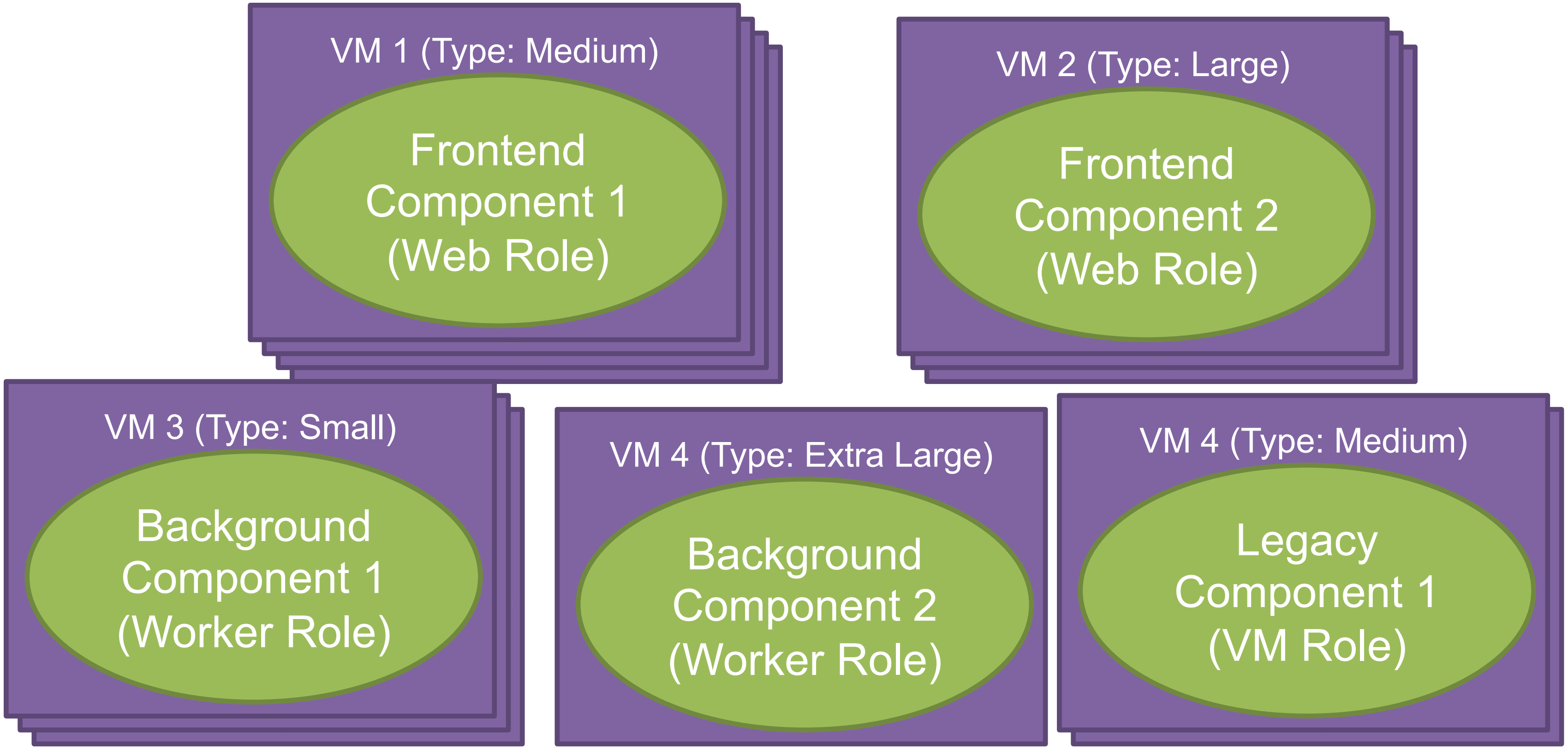
- Web Role entry point: Archive with web code
  - Either ASP.NET code or code compliant with FastCGI

```
@{
    // Some .NET code to be evaluated when page is rendered
    ViewBag.Title = "About Us";
}

<h2>Your heading</h2>
<p>
     The content of your website.
</p>
```

How To Achieve Scalability (1/2)?

- Azure's answer: Run many instances of each role



How To Achieve Scalability (2/2)?

- Load balancers distribute requests between instances
  - For Web Roles: HTTP load balancer for requests
    - Round-robin distribution
  - For Worker Roles: Dependant on communication scheme
- Prerequisite: Instances of roles must be stateless
- Questions:
  - Microsoft Azure does support stateful applications
  - Where does the state go?

Storing State in Microsoft Azure

- Azure offers distinct services to store state
  - Microsoft Azure Storage
    - Table
    - Blob
    - Queue
    - Drives
  - SQL Azure
- Idea: Storing state reliably is complex
  - Most customers don‘t care how it is done
  - Offer different persistent storage facilities as a service
  - Use standard interfaces to services (SOAP/REST)

SQL Azure

- Relational database service
  - Based on MS SQL Server
- Limited scalability compared to Azure Table storage
  - Size of databases limited to 50 GB
  - Replication used to ensure availability
- Consistency: Strong consistency
  - ACID-compliant transactions
- Pricing: Billing per GB per month

Microsoft Azure Platform Maintenance

- How to deal with OS patches, middleware updates, …?
- Azure's approach
  1. Bring up new instance of role on new HW / patched OS image
  2. Redeploy customer's code
  3. Register new instance with the load balancer
  4. Kill outdated instance
  5. Continue until all old instances have been replaced

- Prerequisite: OS and user code can be separated (not true for VM Role)

Three Rules of the Azure Programming Model[6]

1. An Azure application is built from one or more roles
   - Decoupled application from a logical perspective
   - Assign role depending on component's characteristics
2. An Azure application runs multiple instances of each role
   - Key to scalability and availability
   - Allows Microsoft to silently update and restart instances
3. An Azure application behaves correctly when any role instance fails
   - Instances are expected to be stateless

Overview

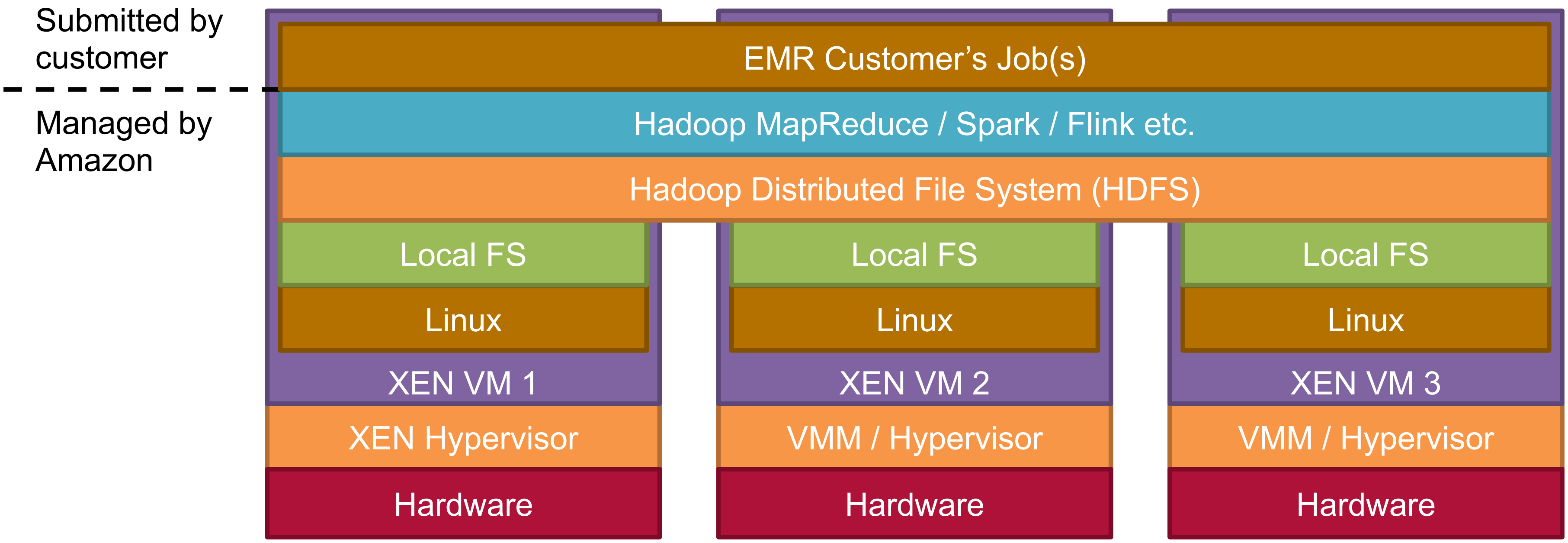
- Intro
  - Cost of Scalability
  - Platforms vs Infrastructure
  - Abstraction Levels of Platforms
- Platforms
  - Azure
  - **Amazon EMR and SageMaker**
- Serverless Computing
  - Concept, FaaS, BaaS
  - Serverless Architectures and Implications
  - Examples

Amazon EMR

- Cloud service for data-intensive applications
  - Introduced in 2009 as part of AWS
- Started with Hadoop MapReduce on EC2 integrated with S3 storage
  - No setup/configuration effort for the EC2 resources
  - Follows pay-per-use model (billed by the second)
- Today also supports many processing systems (e.g. Spark and Flink) and storage services (e.g. Amazon DynamoDB)

Software Stack of Amazon EMR

- Amazon runs preconfigured systems on EC2
  - User submits/monitors jobs through web interface
  - Dedicated VMs per user
  - No direct access to the VMs (in contrast to IaaS)



Job Processing Cycle on Amazon EMR (1/2)

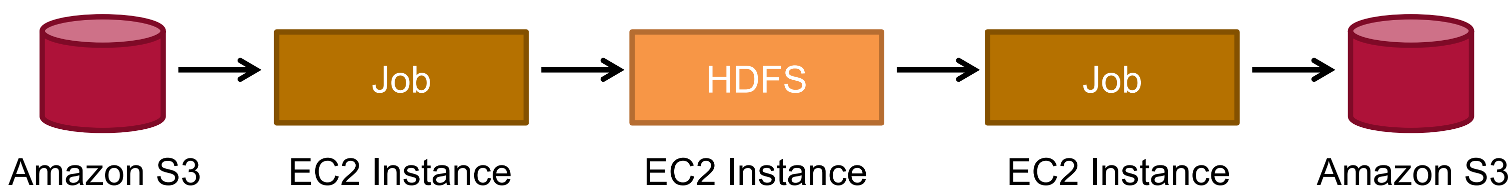
1. Customer submits job through web interface
   - Job specification contains
     - Location of input data on Amazon S3
     - Framework, job code, user libraries, parameters, …
     - Number of virtual machines to run the job on
     - Type of virtual machines to run the job on
     - Designated output location on Amazon S3
2. Requested virtual machines are started on EC2
   - Pricing starts to apply
   - Boot of EC2 instances with preconfigured systems and start of these
   - Worker nodes automatically contact master

Job Processing Cycle on Amazon EMR (2/2)

3. Job runs on rented VMs
   - Input read from Amazon S3
   - Customer can monitor execution progress through web interface
   - Output saved to Amazon S3
4. After job completion, EC2 instances are automatically shut down
   - Job completed after all distributed tasks have finished
   - All VMs must remain allocated until that point in time
     - Intermediate data might be required for potential recovery
   - Shutdown also marks end of billing period

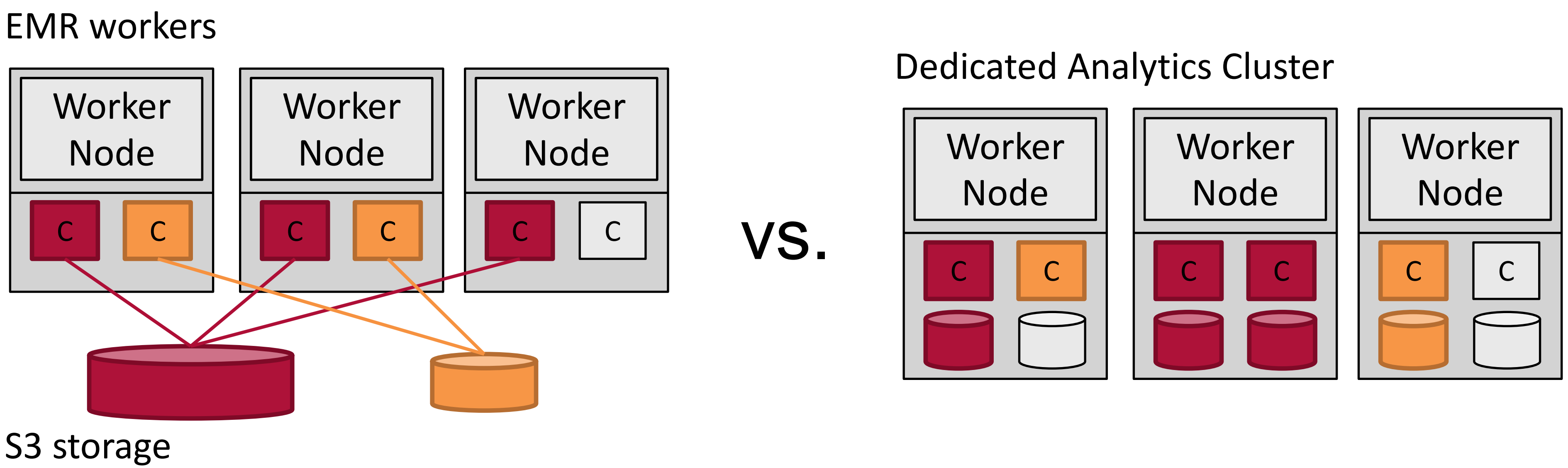
Executing Sequences of Jobs

- Deployment model requires to store initial input/final output on Amazon S3
  - Violates principle to keep data local to computation, so increased effort to move data to/from virtual machines
  - Yet storage nodes can be independently scaled
- EMR also allows to execute sequences of jobs
  - Initial input/final output still stored on Amazon S3
  - Intermediate results between jobs stored in HDFS
    - Data kept at least local between two e.g. MR jobs



Data Stored in S3

- Data stored in S3 in-between jobs
  - Pro: independent number of storage/compute nodes and less expensive than a permanent HDFS cluster
  - Con: Data locality
  - Thus, good if you only run jobs every now and then



EMR and Elasticity (1/2)

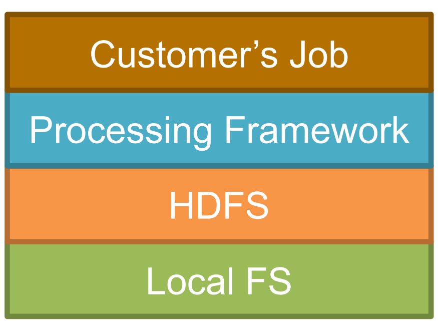
- EMR allows customers to adjust number of virtual machines while job is running
  - Customers can monitor job and respond to unexpected changes in workload
- Problem: VMs to be removed might store intermediate results in HDFS
  - HDFS expects node loss as result of hardware failure
    - If too many nodes are removed at the same time, replication mechanism cannot catch up

    → Risk that job execution fails, initial input must be re-read from Amazon S3 (expensive!)

EMR and Elasticity (2/2)

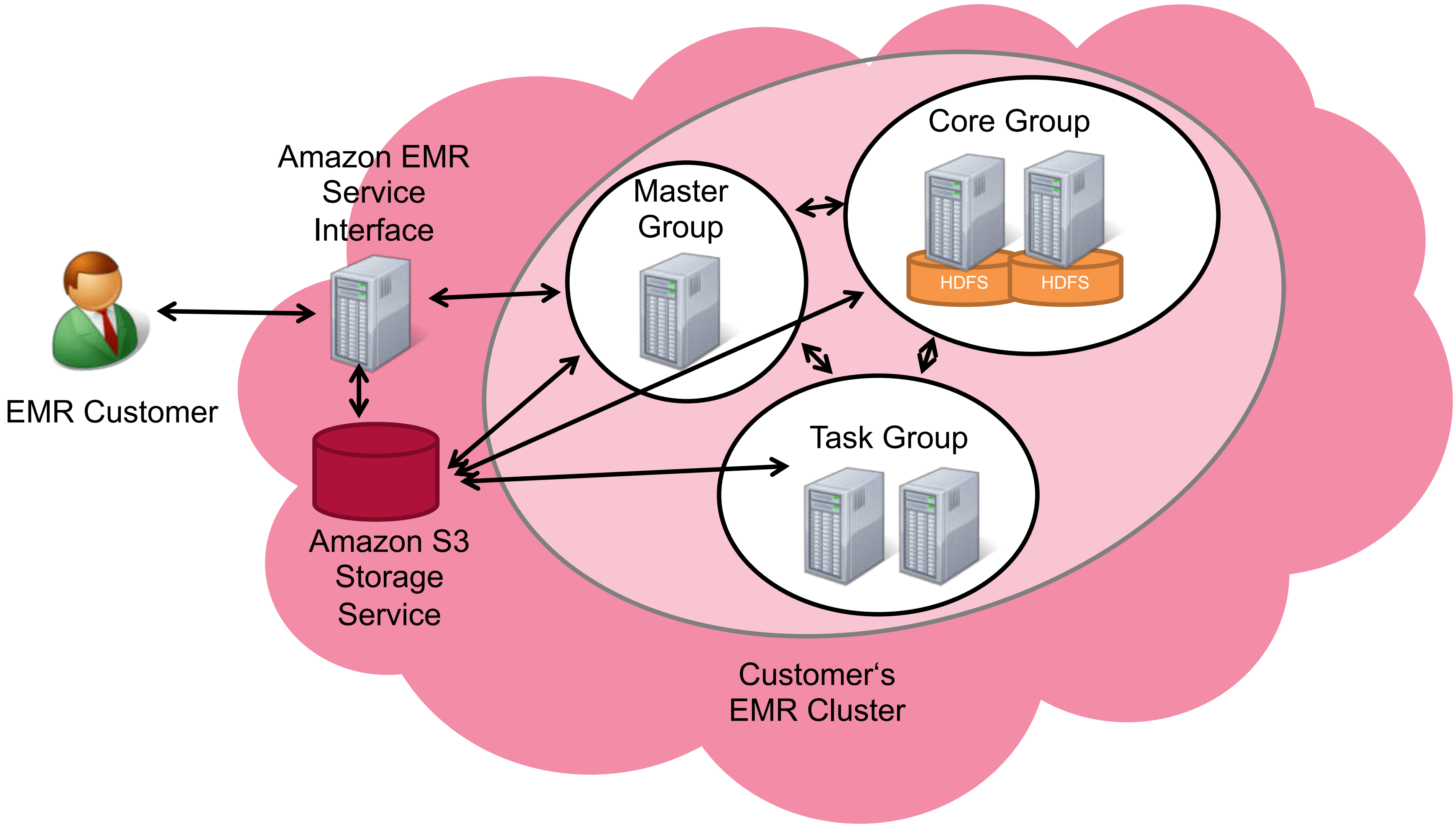
- Solution: EMR separates VMs into three groups
  1. Master group: Contains only VM running master
  2. Core group: VMs run processing and HDFS worker
     - Size of core group can only be increased, not decreased
  3. Task group: VMs run only processing worker, no HDFS
     - VMs store no local data between two MR jobs
     - Intermediate data is transferred to VMs of core group
     - Size of task group can be increased/decreased

Software stack excerpt of a core group VM



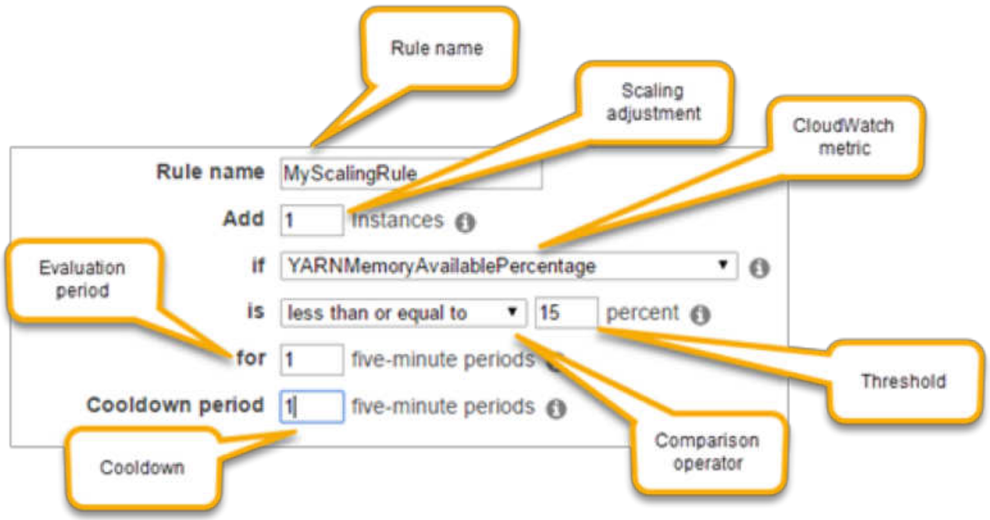
Software stack excerpt of a task group VM

EMR Architectural Summary



Auto-Scaling with EMR

- EMR allows you to set rules for dynamically scaling of core and task nodes based on metrics
  - e.g. scale-up number of workers when less than 15% of memory is available for a five minute period



https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-automatic-scaling.html

EMR Pricing (Jan 2019, EU, Frankfurt)

- Amazon adds surcharge on regular EC2 usage fees

Instance Type	Amazon EC2 Price	Amazon EMR Price
m5.xlarge	0.23 USD	0.048 USD
m5.24xlarge	5.520 USD	0.270 USD
c5.18xlarge	3.492 USD	0.270 USD
r5.24xlarge	7.296 USD	0.270 USD

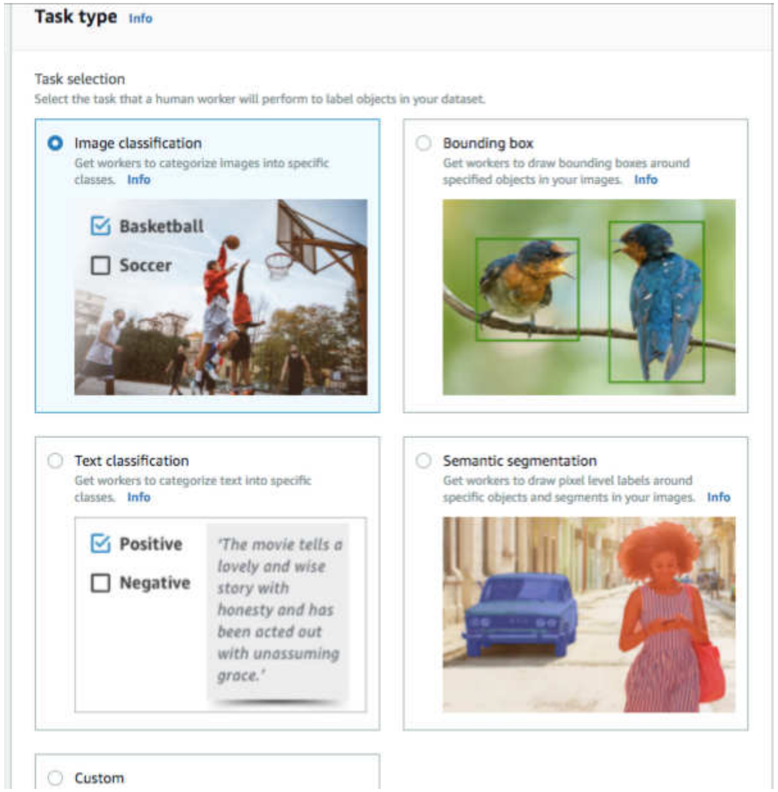
- In addition, the regular AWS usage fees apply
  - S3 storage/access fee
  - Fee for Internet traffic
  - …

Amazon SageMaker

- Domain-specific service for machine learning tasks, started in late 2017
- Facilitates the steps of a machine learning workflow
  - Labelling data
  - Building
  - Training
  - Deployment

Amazon SageMaker: Labelling Data

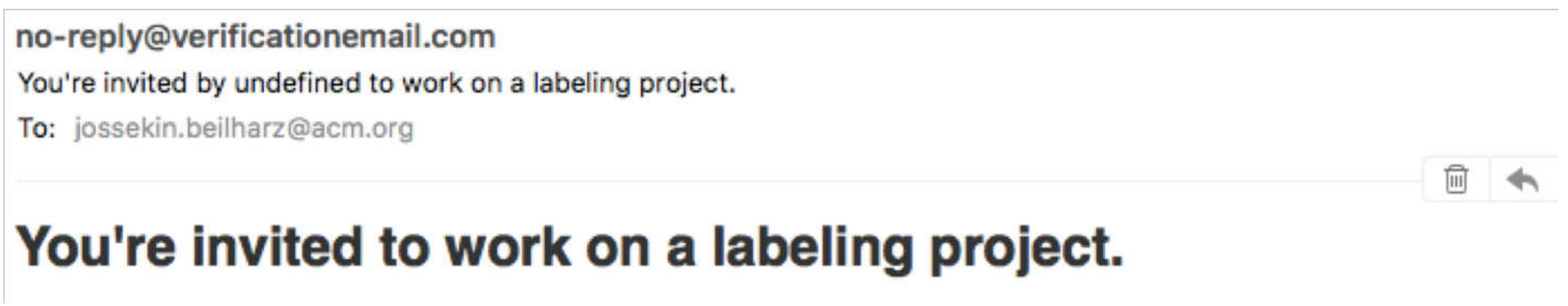
- Data labelling by humans
- Mechanical turk workers via amazon or self-recruited
- Common labelling types have a predefined task type
- Active learning for faster labelling



Predefined task types for labelling images

Amazon SageMaker: Labelling Data Process (1/2)

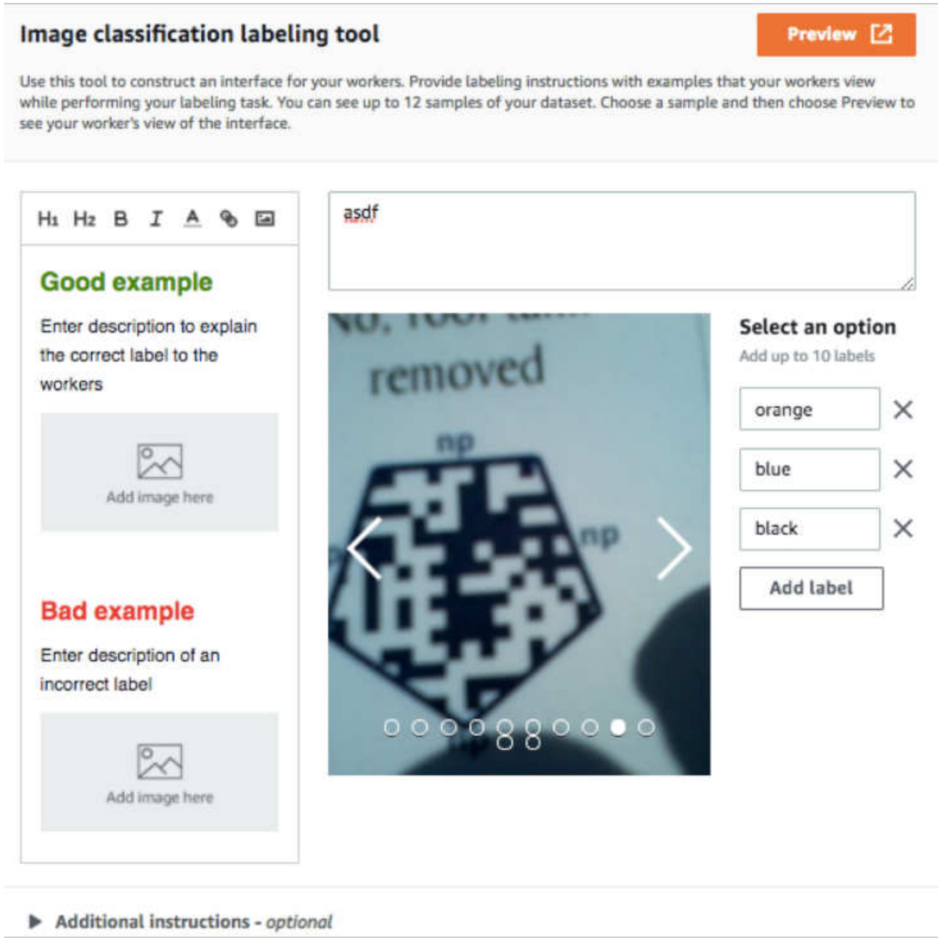
1. Data is uploaded to s3
2. Labelling Job is submitted, consisting of
   - Instructions to human workers
   - Reference to unlabeled data
   - Definition of Labels, if fixed
   - Email addresses of workers, if self-recruited



Amazon SageMaker: Labelling Data Process (2/2)

3. Human workers label data via web interface

4. Labelling can be accelerated via active learning
   - Active learning used to automatically select which data need to be labelled manually



Amazon SageMaker: Training Jobs

- Options to build models:
  - Often Jupyter Notebooks
  - Simpler problems: via CLI, JSON, REST
- There are many built-in algorithms for tasks like image classification, text classification, and clustering

```
In [ ]: # create the Amazon SageMaker training job
        sagemaker = boto3.client(service_name='sagemaker')
        sagemaker.create_training_job(**training_params)

        # confirm that the training job has started
        status = sagemaker.describe_training_job(TrainingJobName=job_name)['TrainingJobStatus']
        print('Training job current status: {}'.format(status))

        try:
            # wait for the job to finish and report the ending status
            sagemaker.get_waiter('training_job_completed_or_stopped').wait(TrainingJobName=job_name)
            training_info = sagemaker.describe_training_job(TrainingJobName=job_name)
            status = training_info['TrainingJobStatus']
            print("Training job ended with status: " + status)
```

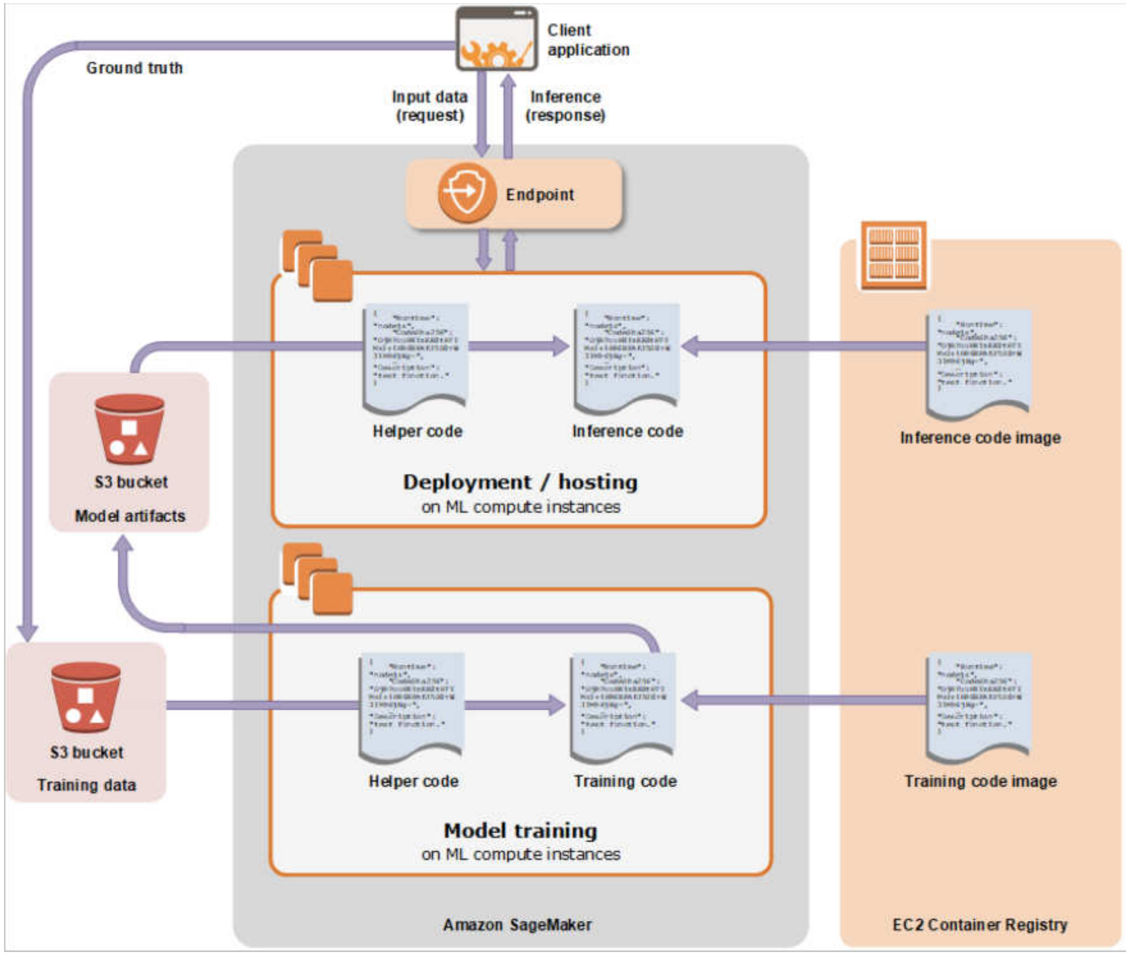
Amazon SageMaker: Training Jobs

- SageMaker integrates popular ML-libraries like MXNet and Tensorflow

- Alternatively customers can build docker containers to
  - Use specific unsupported versions
  - Use a ML-framework that’s not provided by SageMaker
  - Use unsupported additions to a framework

Amazon SageMaker: Deployment

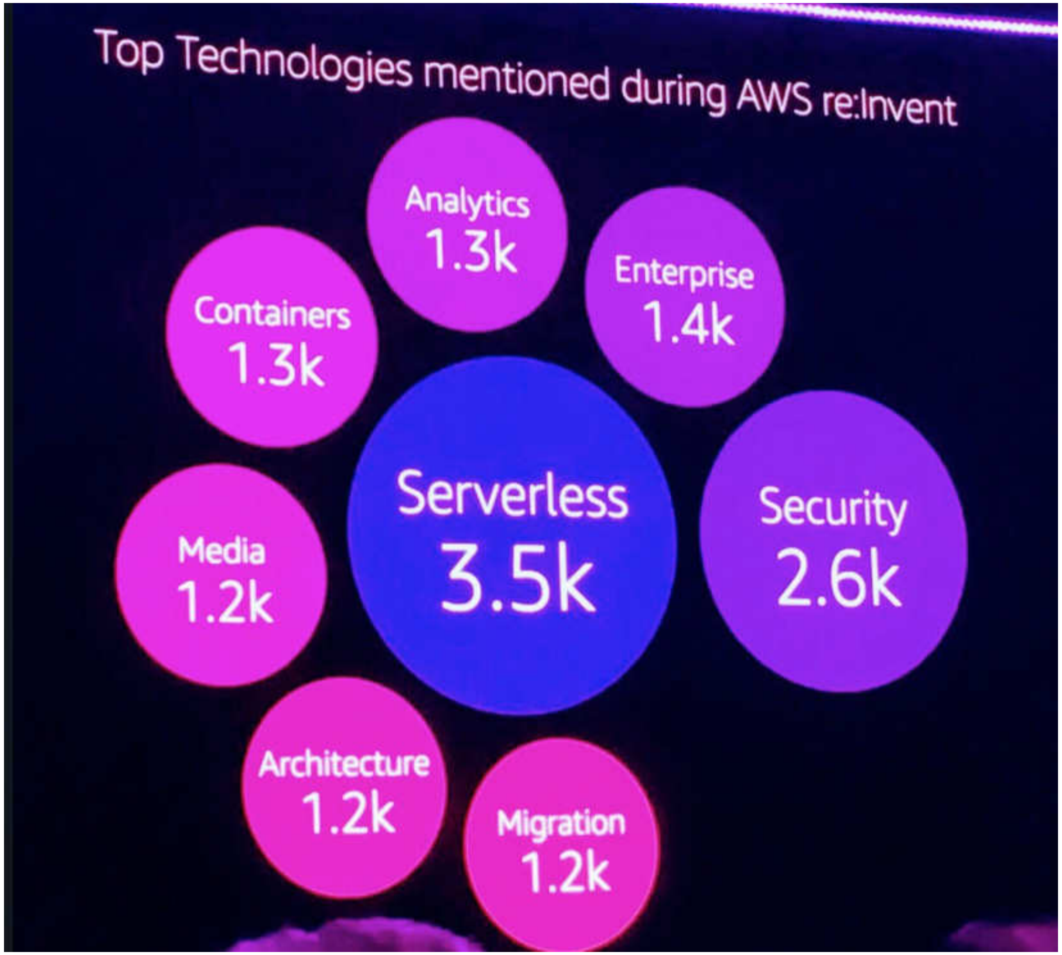
- Trained models can be deployed as
  - Web endpoints
  - Batch transform
- Autoscaling deployed models based on
  - Target metric
  - Scaling policy
  - Integraded with CloudWatch



Overview

- Intro
  - Cost of Scalability
  - Platforms vs Infrastructure
  - Abstraction Levels of Platforms
- Platforms
  - Azure
  - Amazon EMR and SageMaker
- **Serverless Computing**
  - Concept, FaaS, BaaS
  - Serverless Architectures and Implications
  - Examples

AWS re:Invent 2018



Overview

- Intro
  - Cost of Scalability
  - Platforms vs Infrastructure
  - Abstraction Levels of Platforms
- Platforms
  - Azure
  - Amazon EMR and SageMaker
- Serverless Computing
  - **Concept, FaaS, BaaS**
  - Serverless Architectures and Implications
  - Examples

Serverless Computing: Concept, FaaS, BaaS[11]

- Serverless computing is a cloud execution model
  - Provider runs the server-side and dynamically manages the resources
- Enabled by high-level PaaS offerings:
  - Backend as a Service
    - Use cloud database, authentication, etc. as backend to a rich client app
    - Often used for mobile apps
    - Examples: Firebase, Parse, AWS Cognito, ...
  - Function as a Service
    - Execute own code in event-triggered, stateless, and ephemeral compute containers
    - Examples: AWS Lambda, Google Cloud Functions, …

Serverless and PaaS

- Serverless Computing is an execution model that is enabled by Platform as a Service offerings

- Serverless uses
  - Platform services that don't live long
  - Platform services where you don't think about scaling
    - Serverless scaling is automatically managed, transparent, and fine-grained

Overview

- Intro
  - Cost of Scalability
  - Platforms vs Infrastructure
  - Abstraction Levels of Platforms
- Platforms
  - Azure
  - Amazon EMR and SageMaker
- Serverless Computing
  - Concept, FaaS, BaaS
  - **Serverless Architectures and Implications**
  - Examples

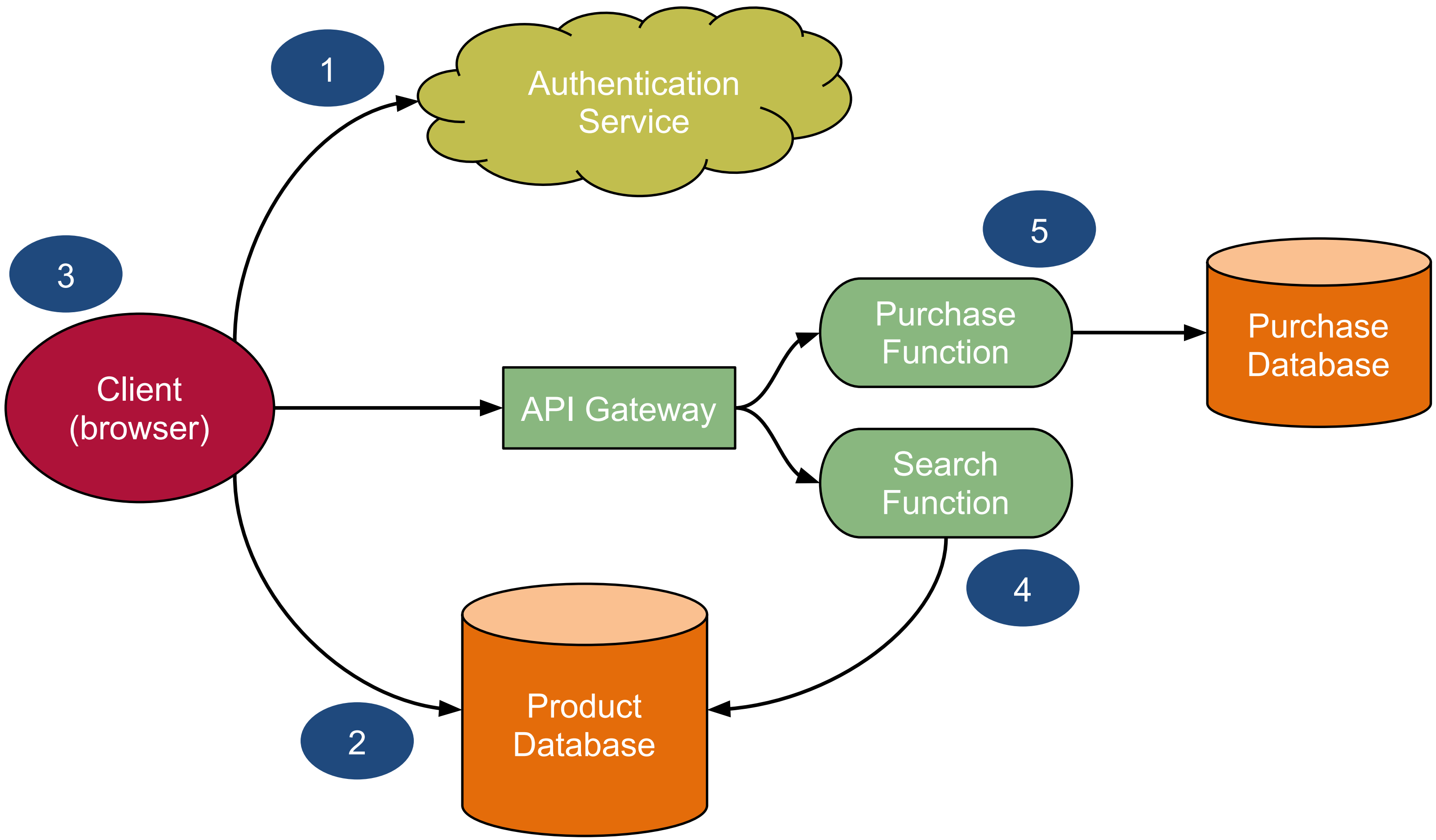
Serverless Architectures: Illustrating Examples (1/3)

A webshop as a traditional three-tier web application:

[11]

Serverless Architectures: Illustrating Examples (2/3)

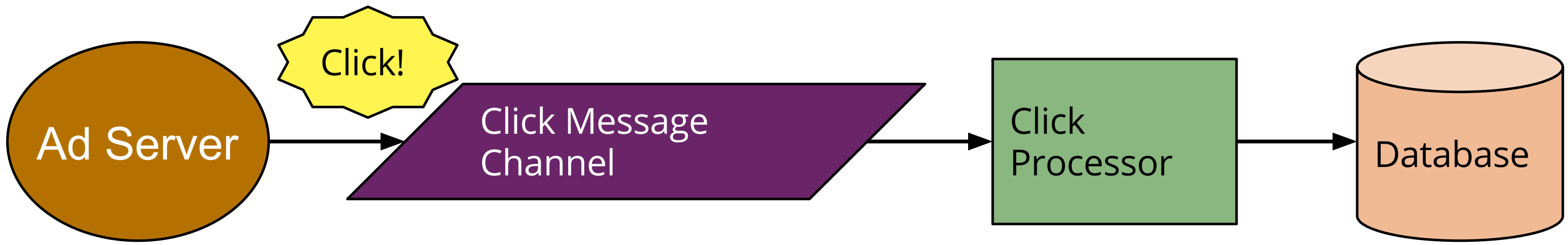
A possible serverless architecture for the webshop:



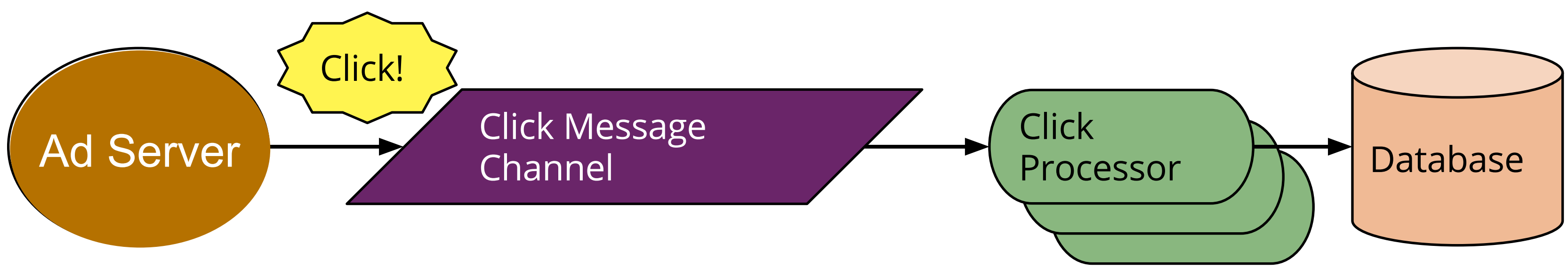
[11]

Serverless Architectures: Illustrating Examples (3/3)

Tracking ad clicks as an example of backend data-processing:



A possible serverless architecture for the backend service:



Function as a Service

- Motivation: Run backend code without managing server resources and/or long-running applications
- Architecture: Stateless, ephemeral, event-triggered
- Automatic horizontal scaling

- Deployment is uploading code and artifacts like for many PaaS services
- No restrictions regarding languages and frameworks

Function Triggers

- Functions are triggered by events
  - File changes in file storage (like S3)
  - Messaging system (like Kafka)
  - Database events
  - Scheduled tasks
  - HTTP endpoint (possibly via an API gateway)

Function as a Service: Restrictions

- Managing State
  - FaaS often described as stateless
  - While state can be kept in execution container, no guarantees when container will disappear
  - Persistent state should be externalized (e.g. in a database, file/object store, or in distributed memory)
  - View FaaS as node in shared-nothing distributed system
- Timing
  - Restricted execution duration of functions (i.e. 5 min)
    - Forces a clear separation of coordination and execution
  - Startup latencies vary (milliseconds to seconds)

API Gateways

- BaaS service to offer (often RESTful) APIs
- Configured mapping between routes and endpoints
- Offer many default tasks for API endpoints
  - Rate limiting
  - Authorization
  - Monitoring
  - Version management
- Often used in conjunction with FaaS to handle incoming requests
- Well-suited to microservice architectures

Serverless Architecture Implications (1/2)

- Less Ops to worry about at the expense of flexibility
- Increasing vendor lock-in with all higher PaaS abstractions
- Coordination vs. execution split is forced
- Change of structure of cloud expenses
  - From paying for reserved resources to actually utilized capacities
  - True compartmentalization, modularization in production is possible

Serverless Architecture Implications (2/2)

- Arbitraging different charging models, i.e.:
  - No functions for data transfer
  - Session state within authentication service
  - IoT Gateways for publish-subscribe models
- Concentrate on the code that creates real business value, lean product development
- Simpler experimentation
  - Cloud service implements version management
  - API Gateway can redirect requests to named versions

When to use Serverless Architectures

- Good:
  - Less application code, reduced development effort
  - Possibly less costs
  - Cloud operator probably better at implementing default tasks than application developer
- Bad:
  - Monitoring becomes more complex
  - Security needs to be handled at all the services
  - Additional complexity through managing the different cloud services

Overview

- Intro
  - Cost of Scalability
  - Platforms vs Infrastructure
  - Abstraction Levels of Platforms
- Platforms
  - Azure
  - Amazon EMR and SageMaker
- Serverless Computing
  - Concept, FaaS, BaaS
  - Serverless Architectures and Implications
  - **Examples**

Serverless Landscape

amazon web services™

Microsoft Azure

spotinst

Google Cloud Platform

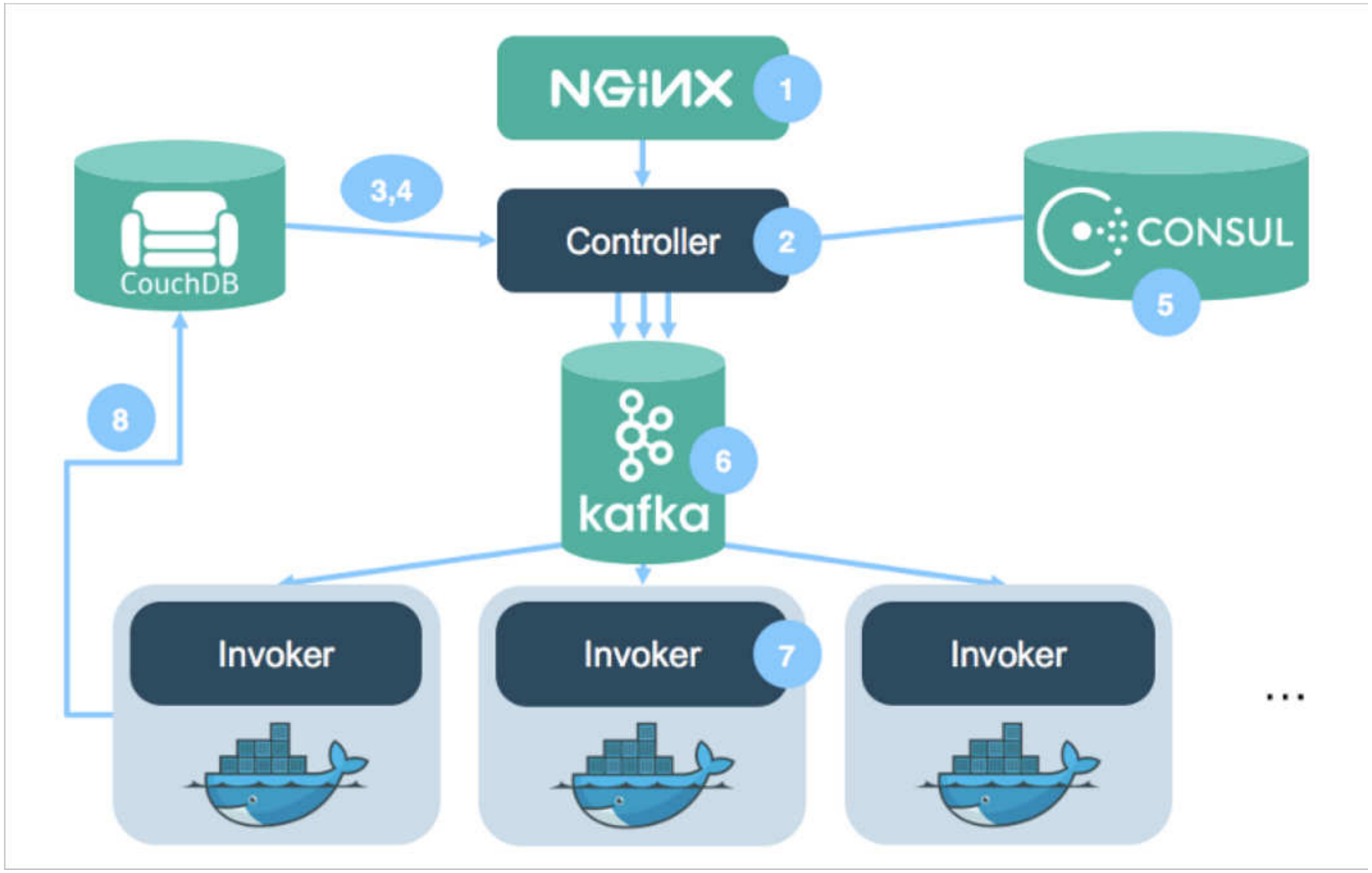
CLOUDFLARE®

fn

APACHE OpenWhisk

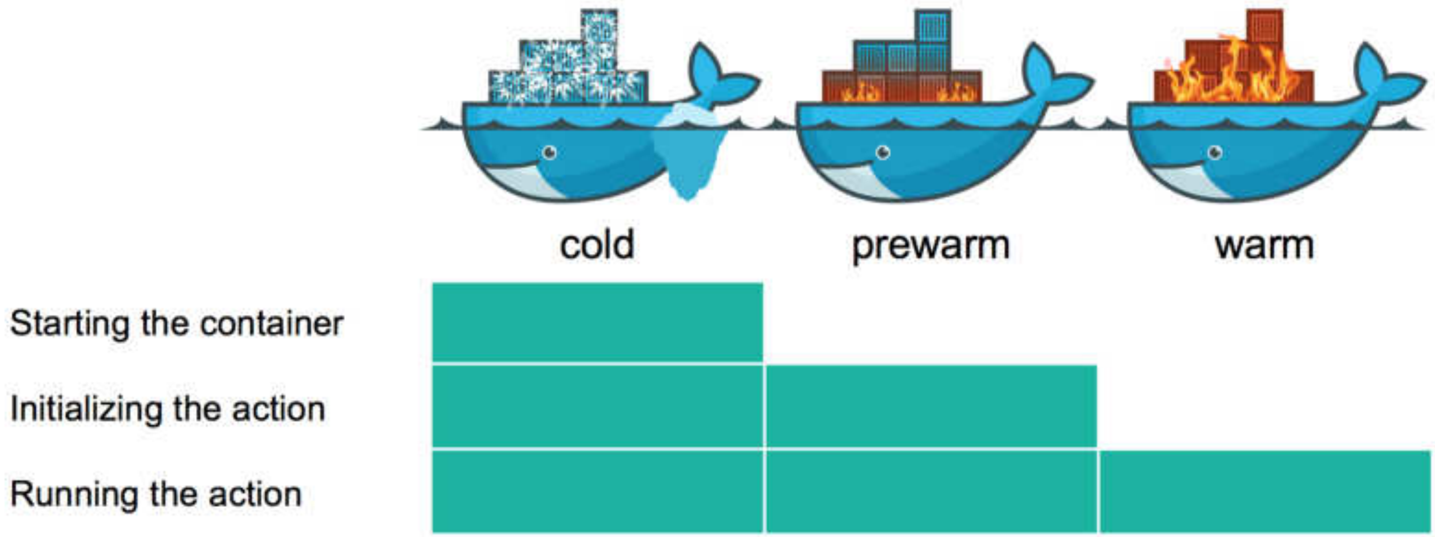
Kubeless

Apache OpenWhisk



[13]

Apache OpenWhisk Invoker



[13]

FaaS Performance Comparison[9][10]

Peeking Behind the Curtains of Serverless Platforms

Liang Wang [1], Mengyuan Li [2], Yinqian Zhang [2], Thomas Ristenpart[3], Michael Swift[1]

[1]UW-Madison, [2]Ohio State University, [3]Cornell Tech

Abstract

Serverless computing is an emerging paradigm in which an application's resource provisioning and scaling are managed by third-party services. Examples include AWS Lambda, Azure Functions, and Google Cloud Functions. Behind these services' easy-to-use APIs are opaque, complex infrastructure and management ecosystems. Taking on the viewpoint of a serverless customer, we conduct the largest measurement study to date, launching more than 50,000 function instances

Serverless computing originated as a design pattern for handling low duty-cycle workloads, such as processing in response to infrequent changes to files stored on the cloud. Now it is used as a simple programming model for a variety of applications [14, 22, 42]. Hiding resource management from tenants enables this programming model, but the resulting opacity hinders adoption for many potential users, who have expressed concerns about: security in terms of the quality of isolation, DDoS resistance, and more [23, 35, 37, 40]; the need to

Evaluation of Production Serverless Computing Environments

Hyungro Lee, Kumar Satyam and Geoffrey C. Fox
School of Informatics, Computing and Engineering
Indiana University Bloomington
{lee212, ksatyam, gcf}@indiana.edu

***Abstract*—Serverless computing provides a small runtime container to execute lines of codes without infrastructure management which is similar to Platform as a Service (PaaS) but a functional level. Amazon started the event-driven compute named Lambda functions in 2014 with a 25 concurrent limitation, but it now supports at least a thousand of concurrent invocation to process event messages generated by resources like databases, storage and system logs. Other providers, i.e., Google, Microsoft, and IBM offer a dynamic scaling manager to handle parallel requests of stateless functions in which additional containers are provisioning**

Microsoft Azure already have the per-second billing, but it even costs every second whether a program runs or not.

Serverless is a miss-leading terminology because it runs on a physical server but it succeeded in emphasizing no infrastructure configuration along with the preparation of computing environments. Fox et al [1] defines serverless computing among other existing solutions, such as Function-as-a-Service (FaaS) and Event-Driven Computing, and we see produc-

FaaS Performance Comparison[9][10]

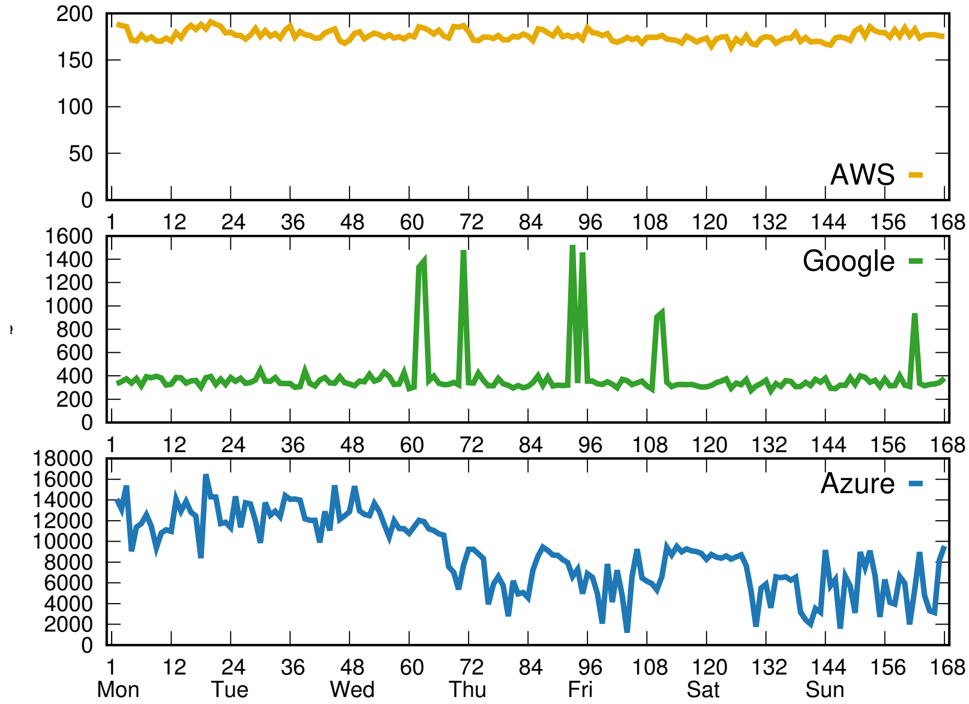


Figure 8: Coldstart latency (in ms) over 168 hours. All the measurements were started at right after midnight on a Sunday. Each data point is the median of all coldstart latencies collected in a given hour. For clarity, the y-axes use different ranges for each service.

FaaS Performance Comparison[9][10]

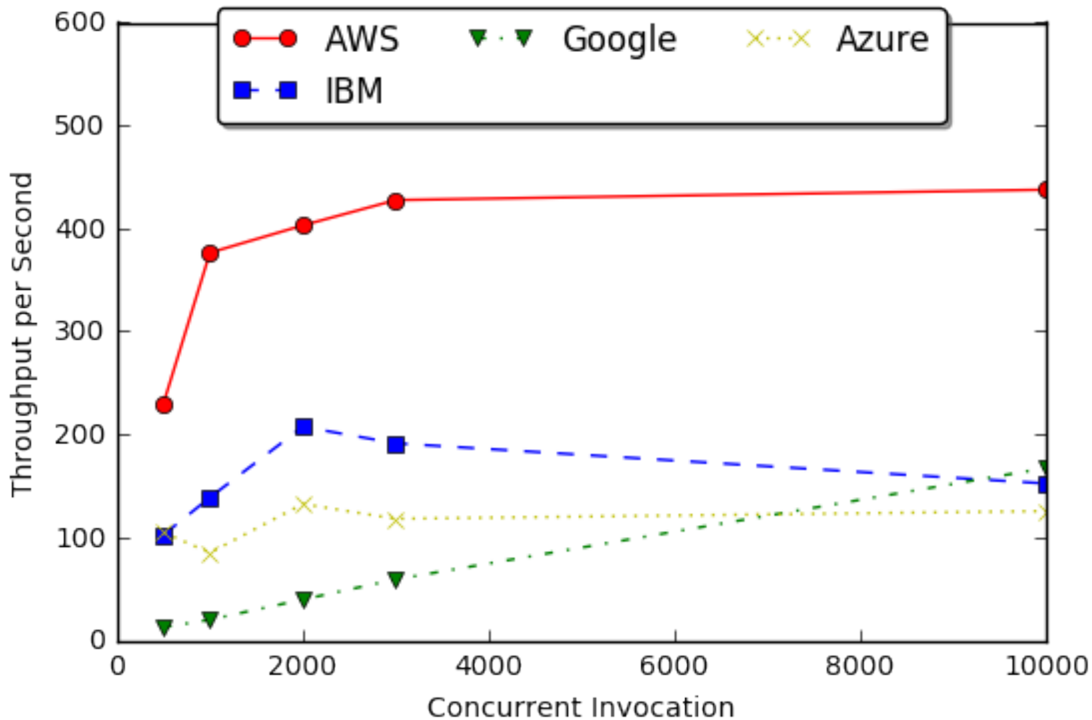


Fig. 1: Function Throughput on Concurrent Invocations

FaaS Performance Comparison[9][10]

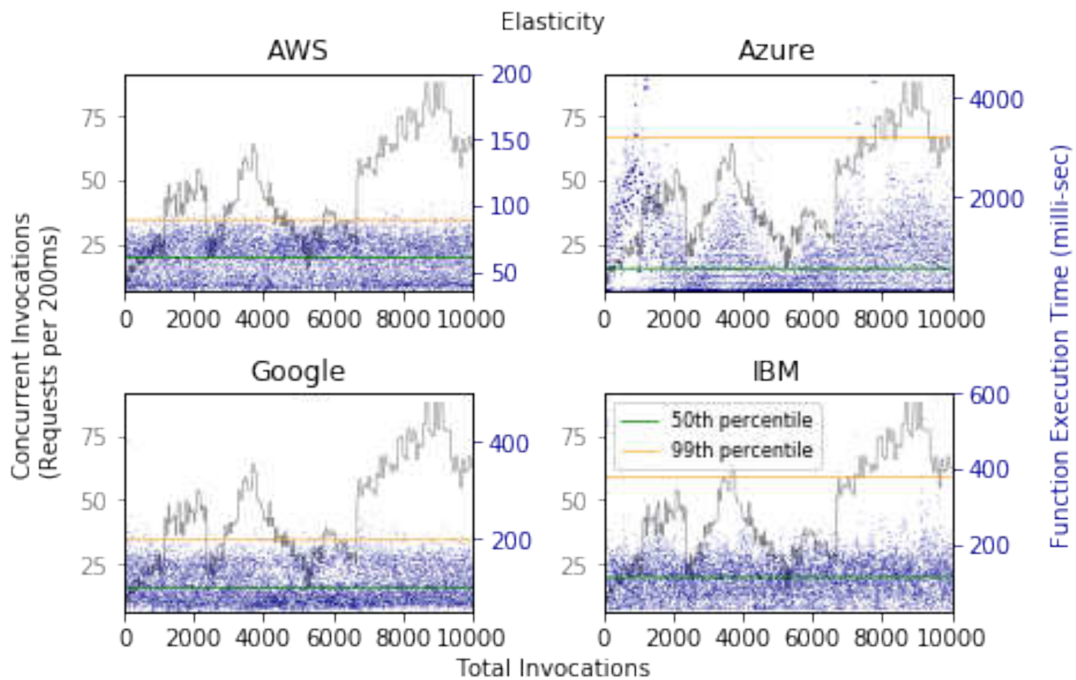


Fig. 5: Response Time for Dynamic Workload

FaaS Performance Comparison[9][10]

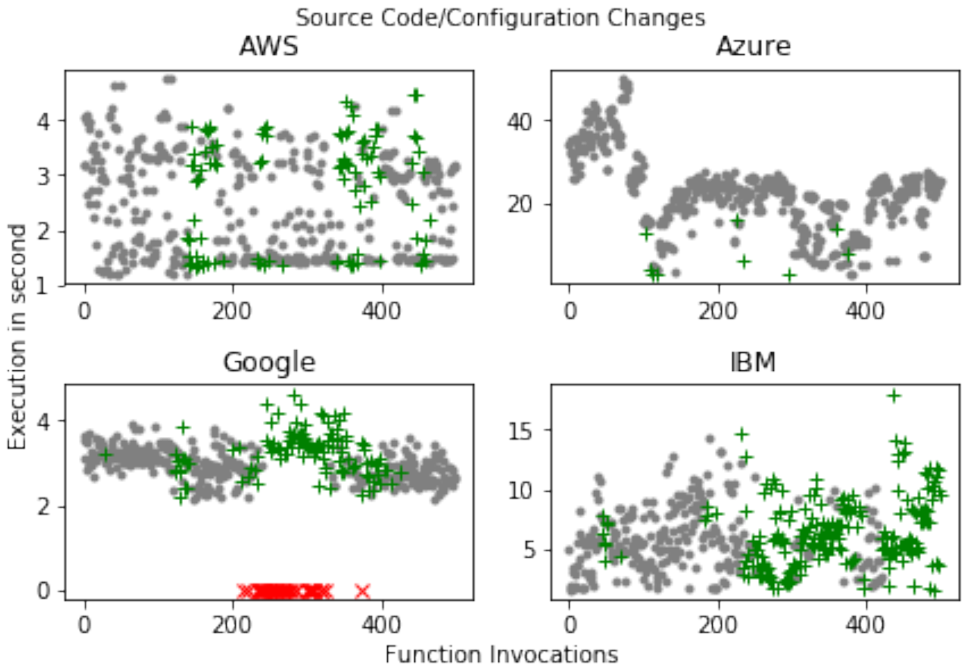


Fig. 6: Function Behavior over CD/CI

(gray dot: existing instances, green +: new instances, red x: failed instances)

Serverless Usage[7]

The Journal of Systems and Software 149 (2019) 340–359

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A mixed-method empirical study of Function-as-a-Service software development in industrial practice

Philipp Leitner[a,*], Erik Wittern[b], Josef Spillner[c], Waldemar Hummer[b]

[a] Software Engineering Division, Chalmers/University of Gothenburg, Sweden
[b] IBM Research, Yorktown Heights, New York, USA
[c] Service Prototyping Lab, Zurich University of Applied Sciences, Switzerland

ARTICLE INFO

Article history:
Received 22 June 2018
Revised 13 November 2018
Accepted 16 December 2018
Available online 18 December 2018

Keywords:

ABSTRACT

Function-as-a-Service (FaaS) describes cloud computing services that make infrastructure components transparent to application developers, thus falling in the larger group of "serverless" computing models. When using FaaS offerings, such as AWS Lambda, developers provide atomic and short-running code for their functions, and FaaS providers execute and horizontally scale them on-demand. Currently, there is no systematic research on how developers use serverless, what types of applications lend themselves to this model, or what architectural styles and practices FaaS-based applications are based on. We present

Serverless Usage[7]

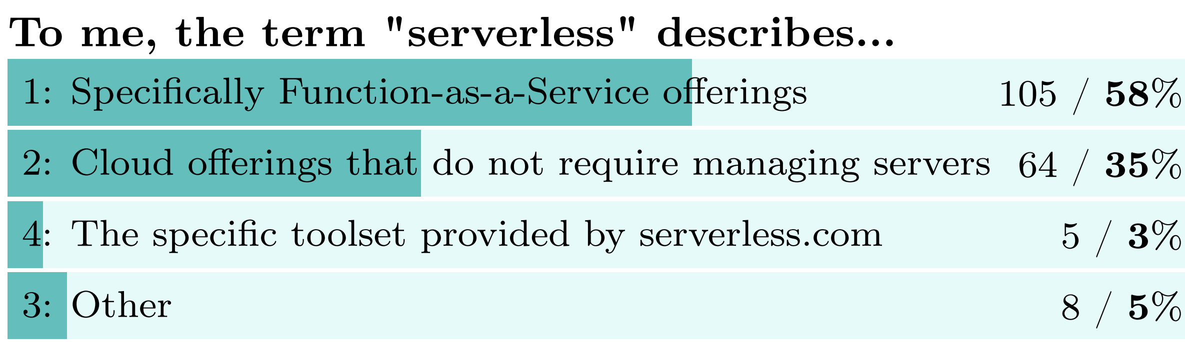


Fig. 5. Survey respondent's definition of the term "serverless".

Serverless Usage[7]

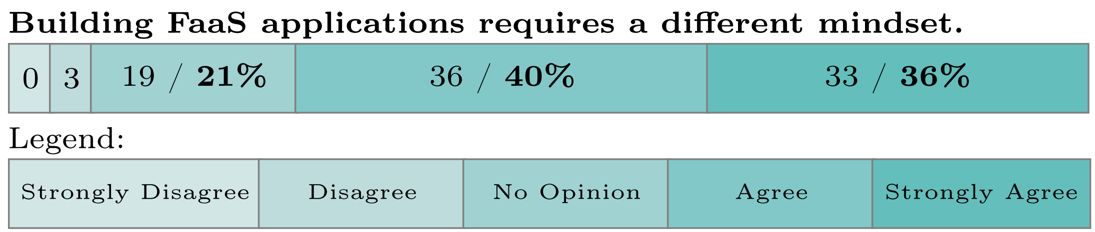


Fig. 6. Mental model for developing FaaS applications. Values without percentage sign refer to absolute numbers of responses.

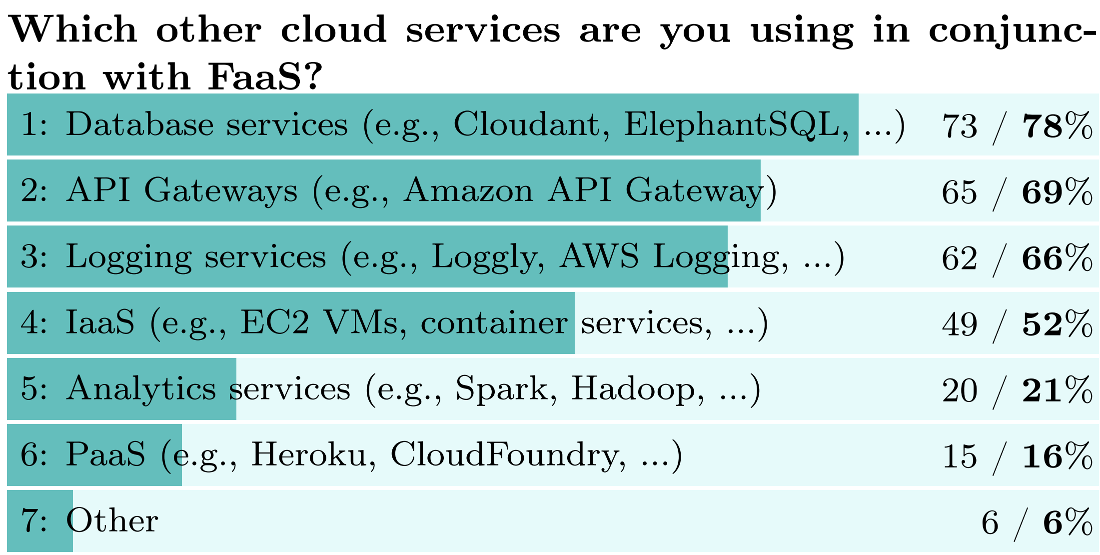


Fig. 7. Cloud services used in conjunction with FaaS.

Serverless Usage[7]

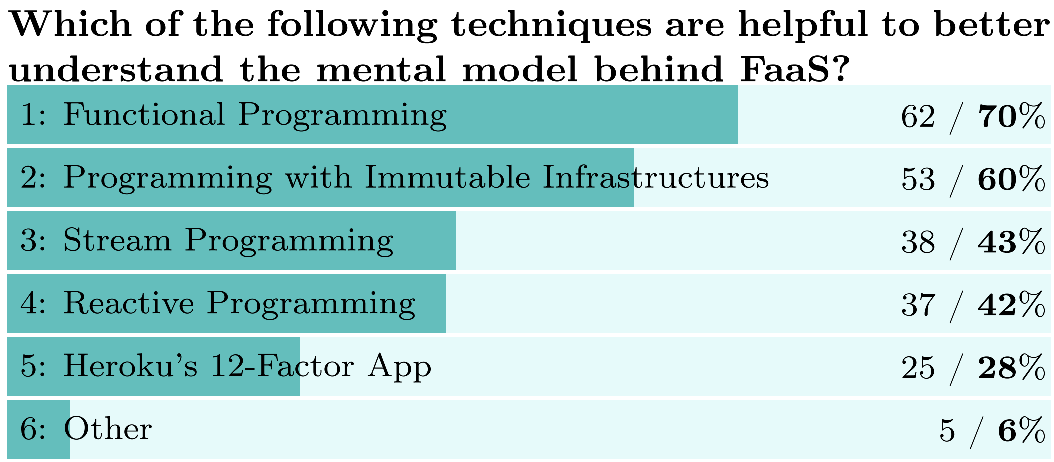


Fig. 9. Helpful techniques to better understand FaaS.

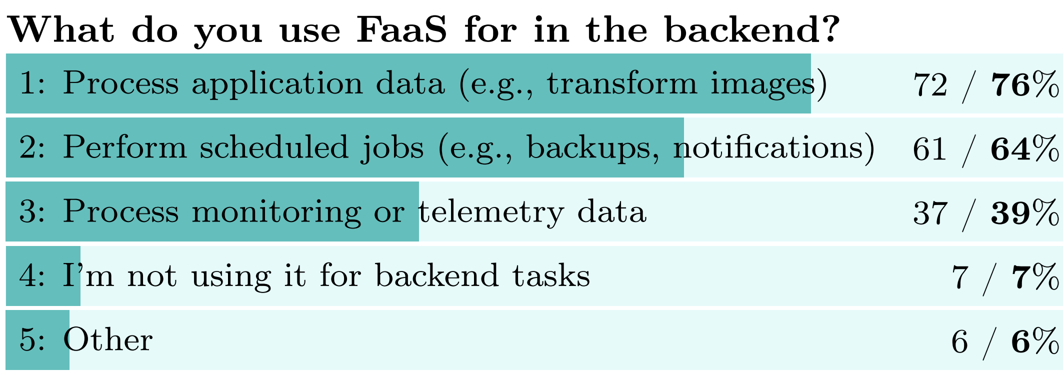


Fig. 11. Usage of FaaS in the backend.

Serverless Usage[7]

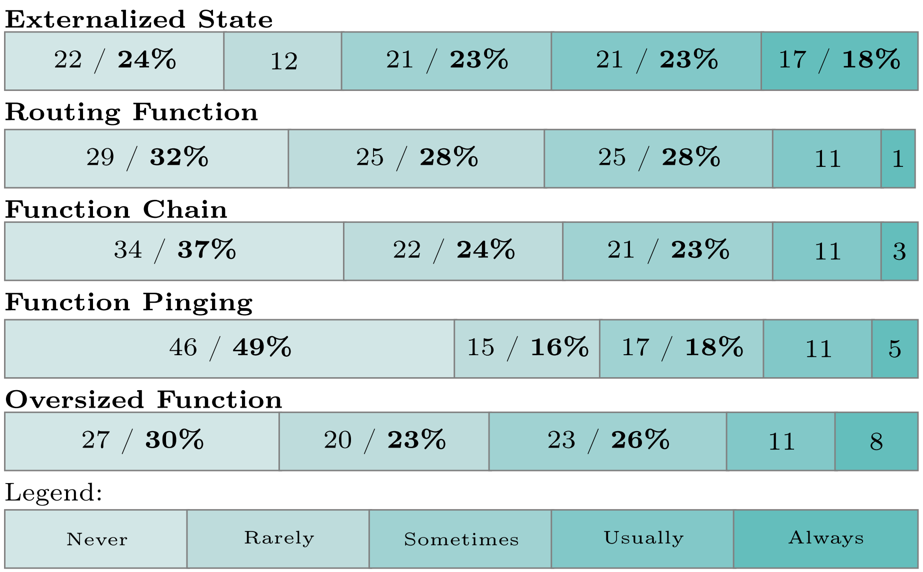


Fig. 15. Prevalence of FaaS application patterns in practice. Values without percentage sign refer to absolute numbers of responses.

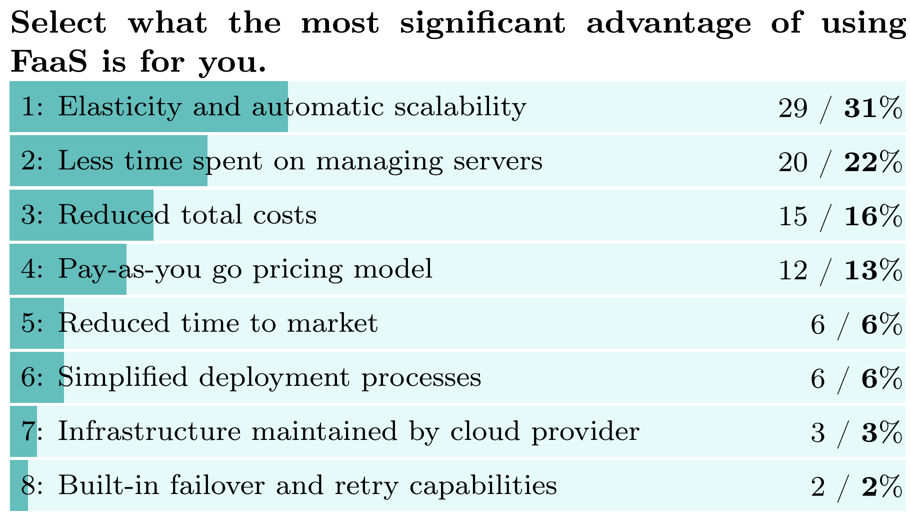


Fig. 18. Significant advantages when working with FaaS services.

Serverless Usage[7]

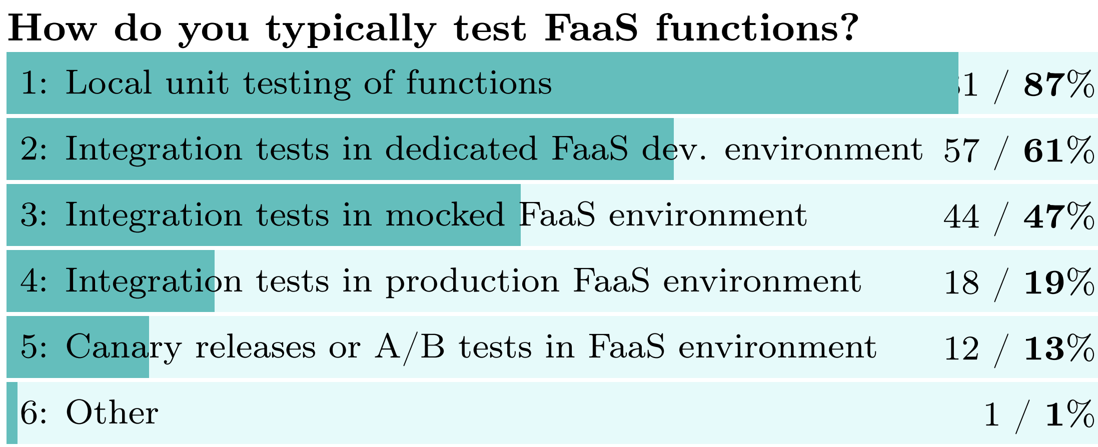


Fig. 17. Testing approaches for FaaS functions.

Serverless Usage[7]

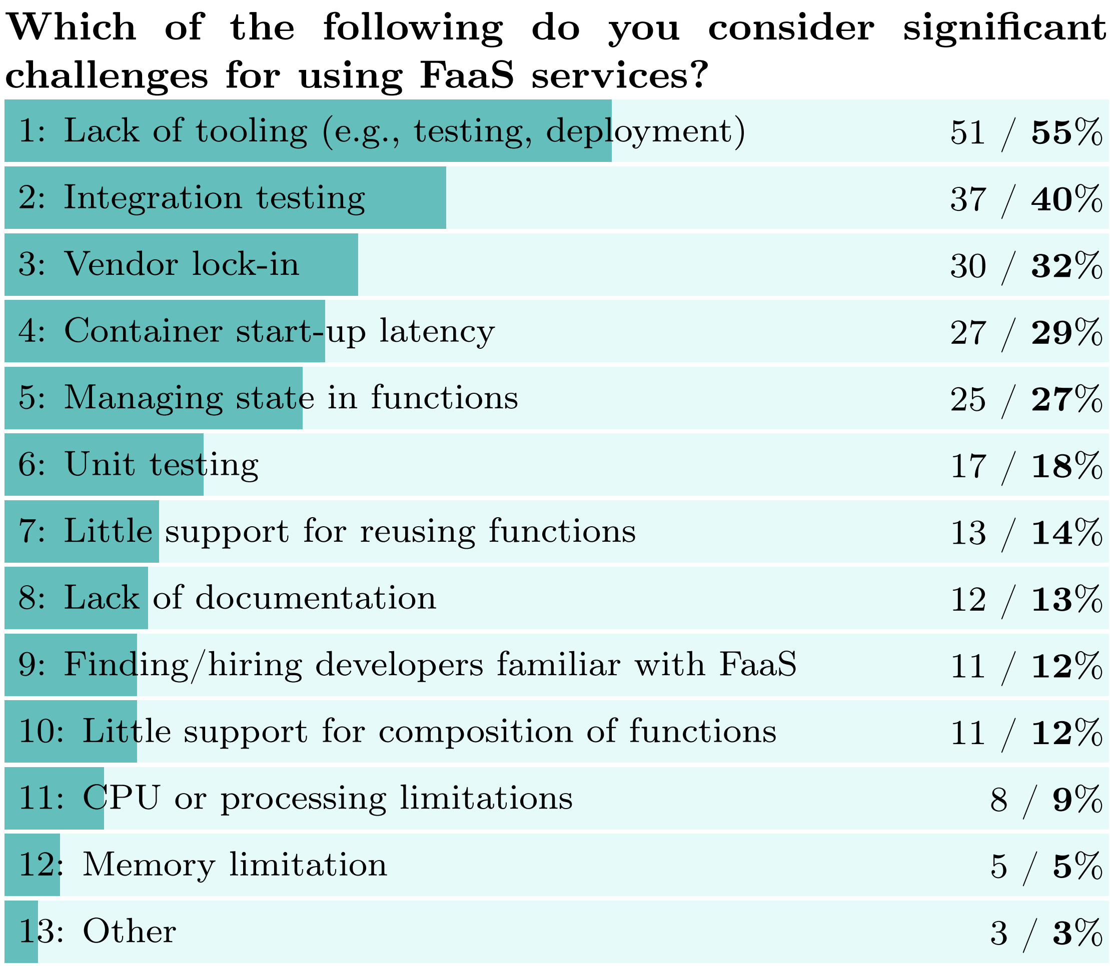


Fig. 19. Significant challenges when working with FaaS services.

Serverless Usage[7]

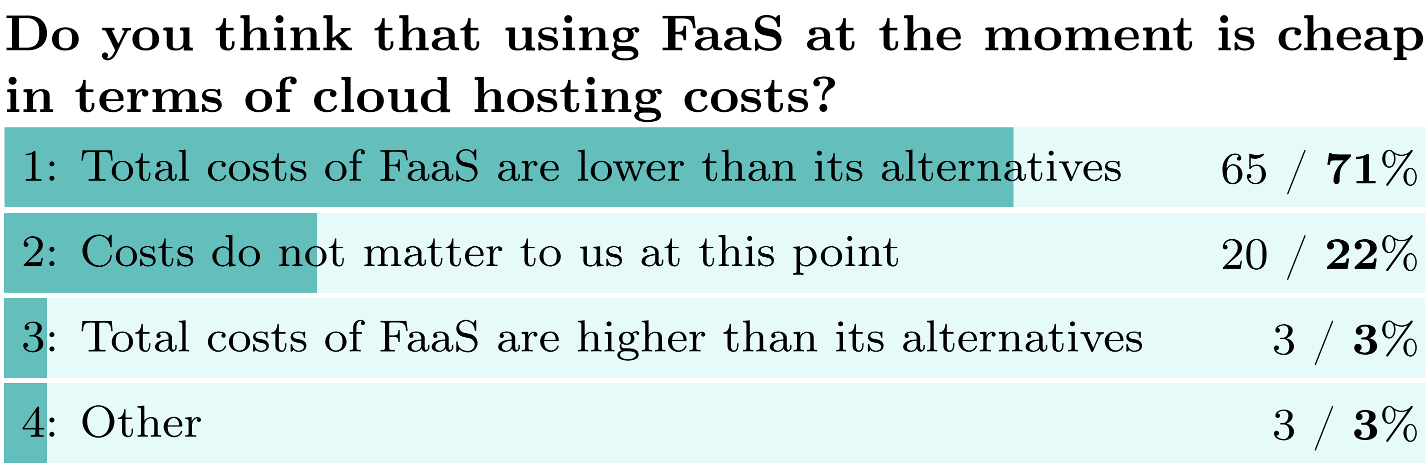


Fig. 20. Perceived costs for using FaaS versus alternatives.

Overview

- Intro
  - Cost of Scalability
  - Platforms vs Infrastructure
  - Abstraction Levels of Platforms
- Platforms
  - Azure
  - Amazon EMR and SageMaker
- Serverless Computing
  - Concept, FaaS, BaaS
  - Serverless Architectures and Implications
  - Examples

Summary

- Managed platforms allow developers to focus more on their applications and less on operations
  - PaaS: in addition to the infrastructure, the entire runtime is managed as well, yet users typically still allocate and scale resources
  - FaaS: users only write and upload their functions, provider manages allocation and scaling of resources
- Platforms available for all kinds of use cases: e.g. long running Web application (e.g. Azure Platform), processing of events (e.g. AWS Lambda), distributed data processing jobs (e.g. AWS EMR)
- New cloud architecture / execution model: Serverless

References

[1] McSherry, Frank, Michael Isard, and Derek G. Murray. "Scalability! but at what cost?." Proceedings of the 15th USENIX conference on Hot Topics in Operating Systems 2015.

[2] P. Mell, T. Grance: “The NIST Definition of Cloud Computing”, Technical Report, National Institute of Standards and Technology, 2011, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[3] I. Fried: “Inside one of the world's largest data centers”, CNET, 2009, http://news.cnet.com/8301-13860_3-10371840-56.html

[4] J. Haridas, N. Nilakantan, B. Calder: “Windows Azure Table - Programming Table Storage”, 2009, http://go.microsoft.com/fwlink/?LinkId=153401&clcid=0x409

[5] B. Calder: “Windows Azure Queue - Programming Queue Storage”, 2008, http://go.microsoft.com/fwlink/?LinkId=153402&clcid=0x409

[6] D. Chappell: “The Windows Azure Programming Model”, 2010, http://go.microsoft.com/?linkid=9751501&clcid=0x409

[7] Leitner, Philipp, et al. "A mixed-method empirical study of Function-as-a-Service software development in industrial practice." Journal of Systems and Software (2018)

[8] Adzic, Gojko, and Robert Chatley. "Serverless computing: economic and architectural impact." Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering

[9] Lee, Hyungro, Kumar Satyam, and Geoffrey Fox. "Evaluation of production serverless computing environments." 2018 IEEE 11th International Conference on Cloud Computing (CLOUD)

[10] Wang, Liang, et al. "Peeking behind the curtains of serverless platforms." 2018 USENIX Annual Technical Conference

[11] Roberts, Mike "Serverless Architectures", https://martinfowler.com/articles/serverless.html

[12] Gojko Adzic "Designing for the Serverless Age", https://gojko.net/2017/10/05/serverless-design-gotocph.html

[13] Markus Thömmes “Uncovering the magic: How serverless platforms really work!”, https://medium.com/openwhisk/uncovering-the-magic-how-serverless-platforms-really-work-3cb127b05f71