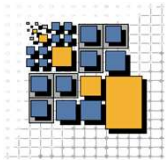


DSG-DSAM-M 2024/25:
- Cloud Computing – Part B -
- Google App Engine / FaaS -

Dr. Andreas Schönberger

with contributions of Johannes Manner

Distributed Systems Group
Faculty Information Systems and Applied Computer Science
University of Bamberg

Google Cloud Offerings

- Google App Engine (PaaS) is part of Google's comprehensive cloud platform:
  - Compute Engine (IaaS)
  - Container Engine (IaaS)
  - Kubernetes Engine (IaaS)
  - Cloud Storage (PaaS)
  - BigQuery (PaaS)
  - Cloud SQL (IaaS/PaaS)
  - Cloud Functions (FaaS)
  - …
- Common Characteristic
  - Free use for entry-level scenarios (quota-limited)
  - Payed use for high-volume services

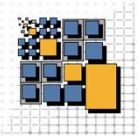
© Google

GAE :=
Google App Engine

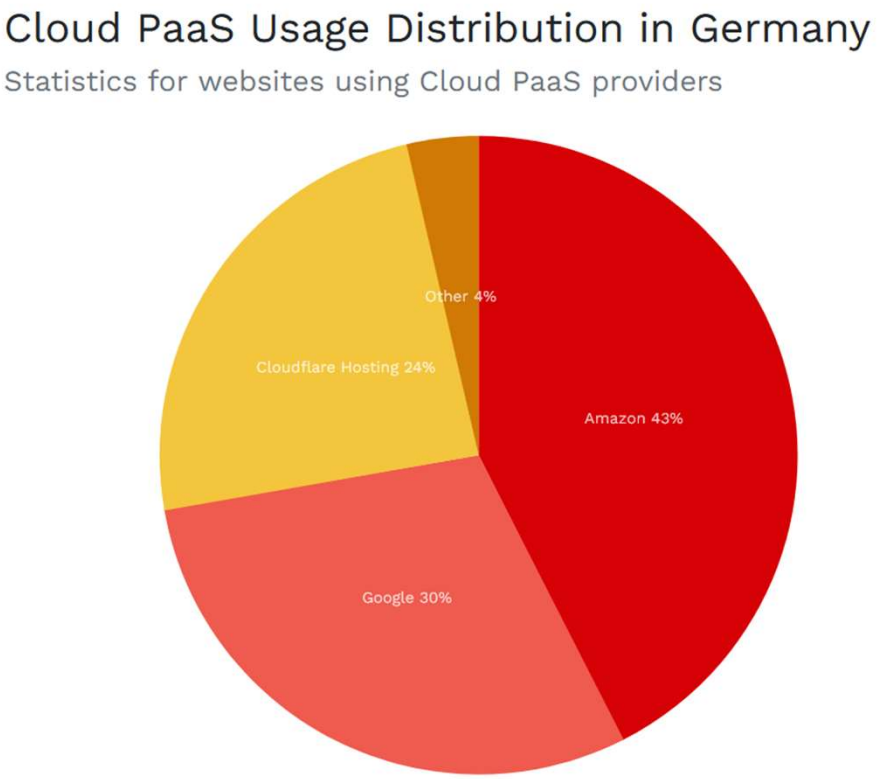
By the way, are Gmail, Maps, Search, YouTube SaaS services?

https://cloudplatform.googleblog.com/2013/12/an-ode-to-sharkon.html

Is Google a Relevant Paas Player?



2023 Figures of builtwith.com:



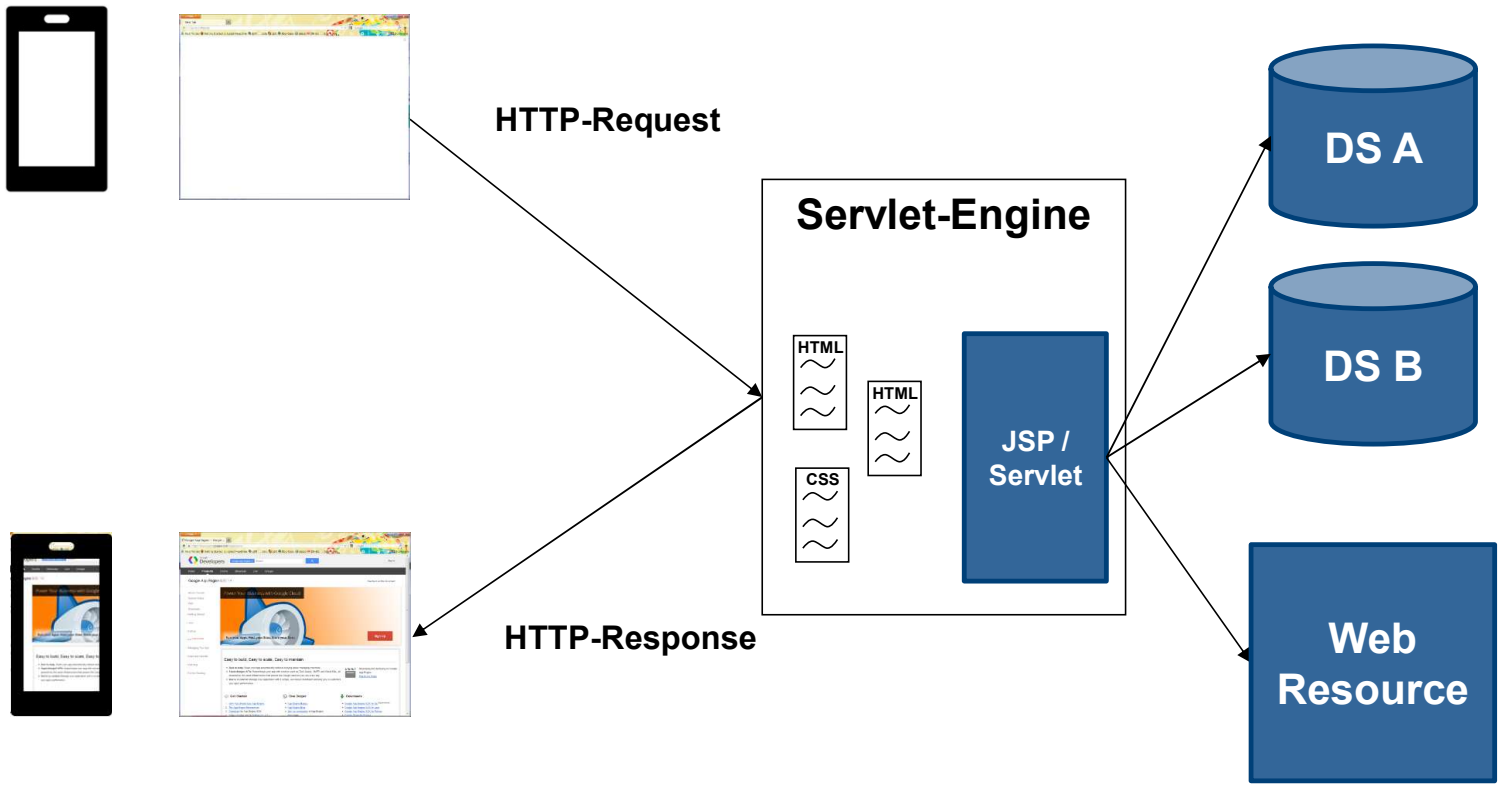
Beware!
Google here does not necessarily mean Google App Engine

src.: https://trends.builtwith.com/hosting/cloud-paas/country/Germany

App Engine – Application Scenario

GAE is made for running web applications!

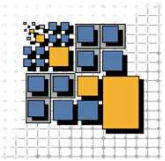
Automatic scaling and loadbalancing



Servlets are just one of multiple environments

Watch out for matching the use case:
https://cloud.google.com/appengine/docs/standard/how-requests-are-handled

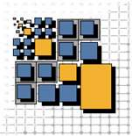
Among others, max. processing times, size limitations etc.



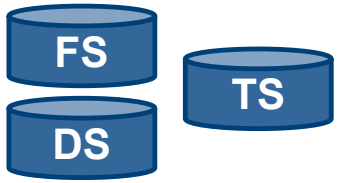
Google App Engine

Distributed Systems Group
Faculty Information Systems and Applied Computer Science
University of Bamberg

Google App Engine – Big Picture

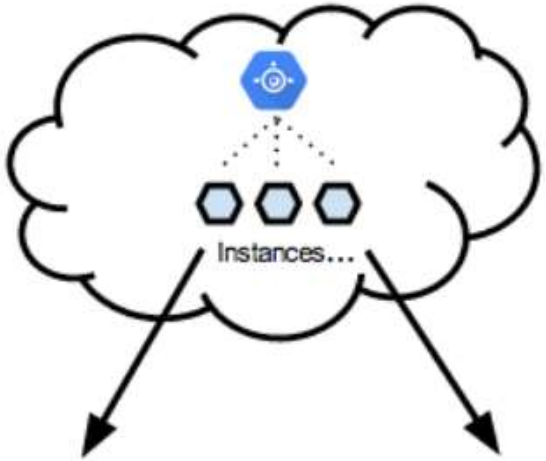


- Four different languages/runtimes on offer
  (and all are subject to some sandbox limitations!)
  - **Java**
  - Python
  - Go
  - Ruby
  - Node, PHP, more languages via flexible environment
- Several data services available
  - **Firestore**
  - Cloud Storage
  - Cloud SQL
- App Engine Services, quotas & limits



Firestore is also available in datastore mode for backwards compatibility.

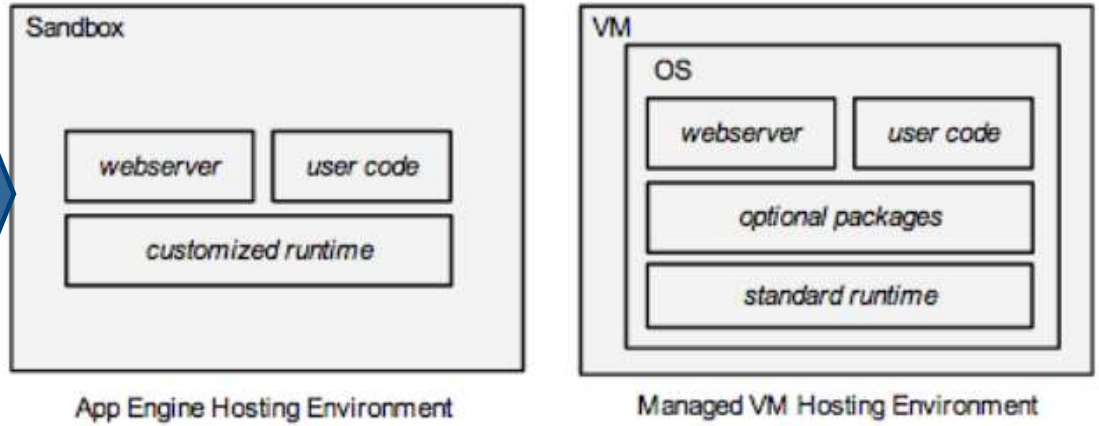
2014: Google's extended offerings



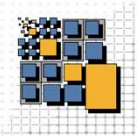
Flexible Environment ~~(Managed VM)~~;

formally supported since **2017**, thereby adding several new languages (Node, C#, custom)

This course!



Standard PaaS vs. Flexible Environment



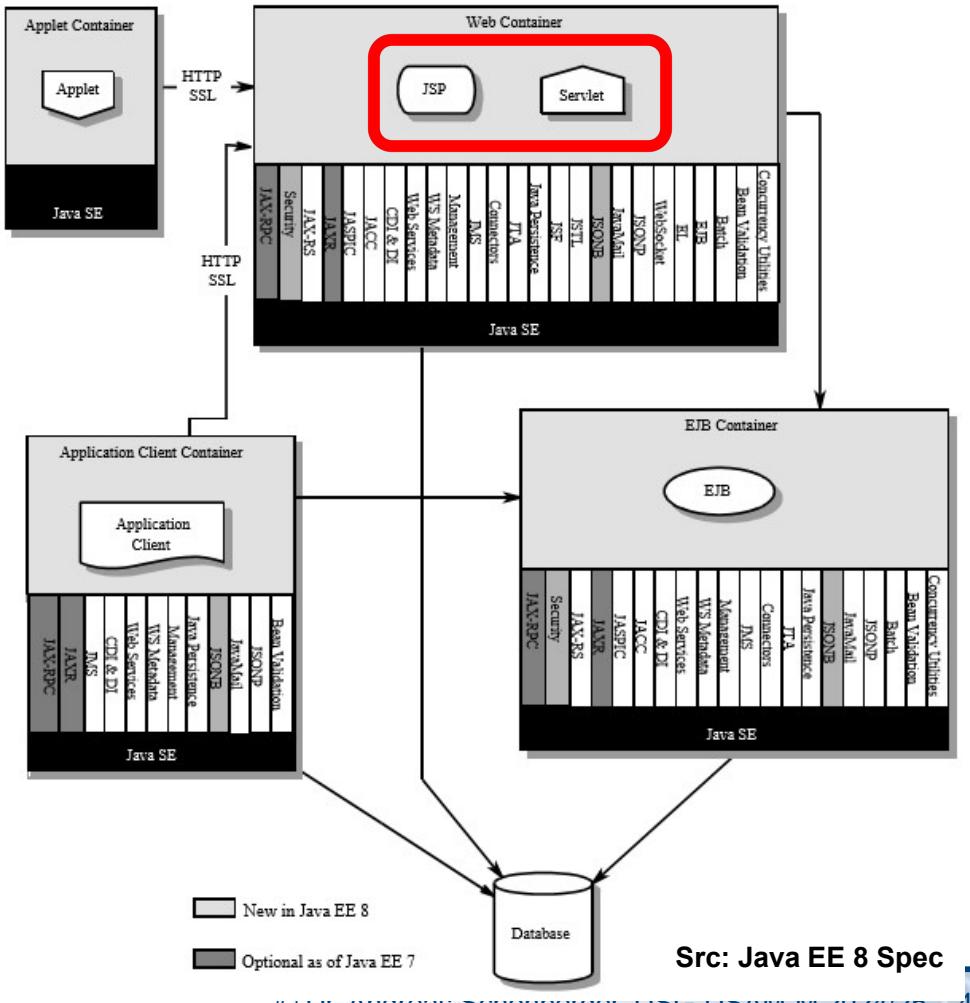
This course!

Feature	Standard Environment (App Engine hosting)	Flexible Environment ~~(Managed VM)~~ hosting
Instance startup time	(Milli)seconds	Minutes
Background threads	Yes, with restrictions	Yes
Background processes	No	Yes
SSH debugging	No	Yes
Scaling	Manual, Basic, Automatic	Manual, Automatic
Scale to zero	Yes	Minimum 1 instance
Writing to local disk	Write to /tmp OR no access	Yes, ephemeral
Customizable serving stack	No	Yes
WebSockets	No	Yes
Supports installing third party libraries	Yes (prev. No)	Yes
Pricing	Based on Instance hours	Uses Compute Engine Pricing for each VM

https://cloud.google.com/appengine/docs/the-appengine-environments#compare_high-level_features

GAE Java Runtime – **Previously**

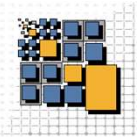
Google documentation:
“App Engine runs your web application using a Java 8 JVM. App Engine invokes your app's servlet classes to handle requests and prepare responses in this environment.”
→ **Hey, that’s Java EE! … the Servlet part of it.**



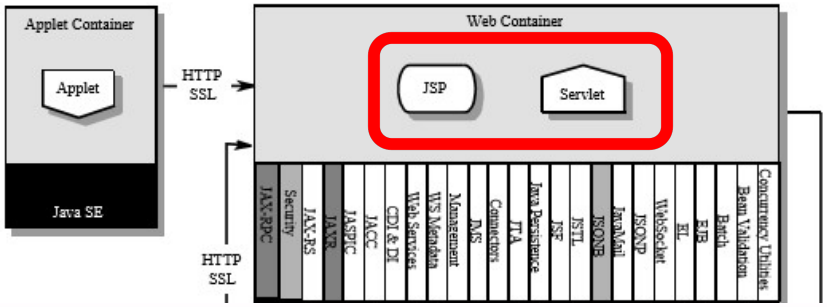
Src: Java EE 8 Spec

Java 11 with Non-Servlet Environments available now

GAE Java Runtime - **Previously**



Google documentation:
“App Engine runs your Java web application using a Java 7 JVM in a safe "sandboxed" environment. App Engine invokes your app's servlet



Warning: The App Engine SDK no longer supports Java 6. Applications that use Java 6 need to be migrated to Java 7. Existing applications that use the Java 6 runtime are still supported, but this support will be removed in a future release.

Src: google.com, 2015

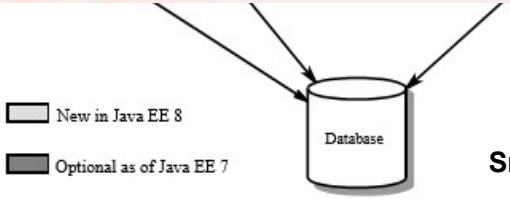
Warning: Support for Java 6 has been deprecated and is going to be removed. See the Java 6 Runtime Support Turndown document for details and timetable.

Src: google.com, 2016

Warning: The Java 7 runtime is deprecated. Google will continue to support the Java 7 runtime in accordance with the terms of service. You should migrate to the Java 8 runtime at your earliest convenience.

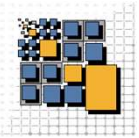
Src: google.com, 2017

Java 8 beta available as of Dec. 2017



Src: Java EE 8 Spec

Initial GAE Servlet Focus



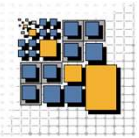
- Reuse of defined programming contract (Servlet spec.)
  - Well-known API classes (HttpServlet, Filter, HttpSession)
  - Standard application layout (WAR, web.xml)
  - Standard programming limitations
    (Actually no file system access)
- **In theory**, Servlet-based artifacts can be reused
  - Existing web applications
  - Existing Servlet-based technologies
    - JSP
    - JSF
  - Existing web frameworks
    - Wicket
    - MyFaces
    - …

https://jcp.org/en/jsr/detail?id=340

http://www.jcp.org/en/jsr/detail?id=154

There used to be a "Will-it-play" web site, but it has been taken offline

GAE Java Runtime - **Now**



- Provide an executable jar with:
  - includes main-class that starts a web server
  - web-server needs to be exposed on PORT variable
  - Includes an app.yaml as deployment descriptor

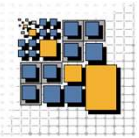
→ Less assumptions about environment, basically Java 17/21

→ Less frequent adaptations to environment changes

→ Less constraints on deployment model

→ More flexibility in choosing the framework you want

GAE follows an industry trend: less Java EE / Jakarta EE, more Spring



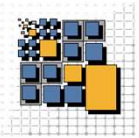
cf. Chapter 3 slide 14

What is (early) Jakarta EE?	What is Spring?
• First enterprise specification extending Java SE	• Enterprise framework, ideas based on J2EE, Java EE (selected specifications from the EE umbrella)
• Formerly J2EE (1999-2006) and Java EE (2006-2019)	• First release in 2003 under Apache 2.0 license, written by Rod Johnson
• Full-blown application servers (e.g. Glassfish, Wildfly) and servlet containers	• Servlet container like Tomcat
• A lot of configuration effort has to be done, e.g. web.xml	• Annotation based configuration and via properties and profiles
• EJB (Java EE specification including e.g. concurrency, security) based programming model (needs an app server to work)	• POJO based programming model (framework does the work, e.g. concurrency, security etc.)

Both ecosystems try to support developers writing enterprise code, but Spring's passion is to make enterprise coding easier and more transparent.

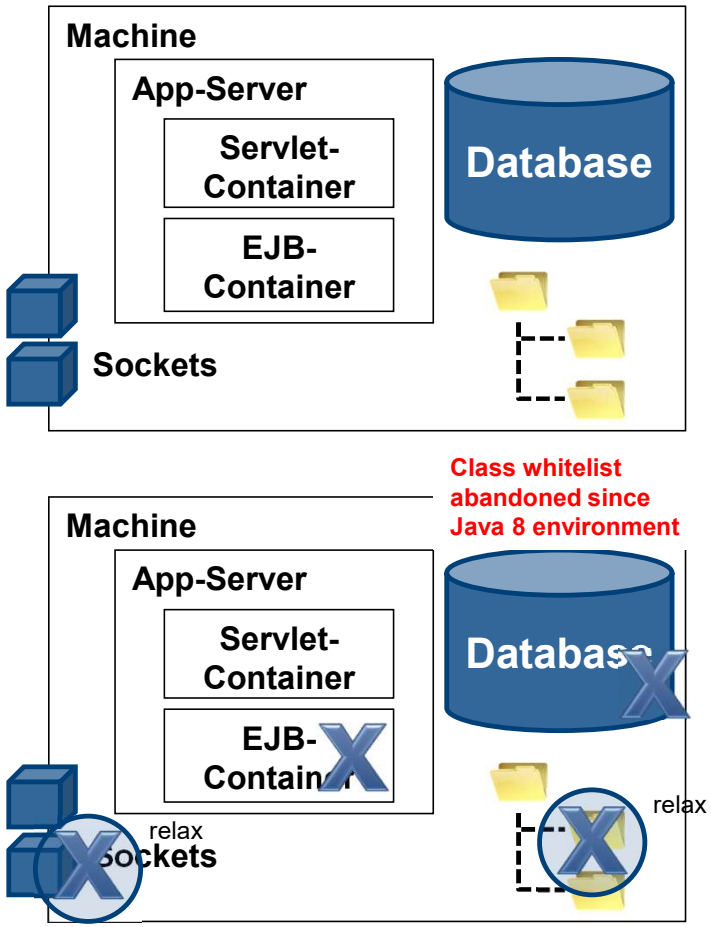
https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html#overview

GAE Java Runtime = Servlet Container?

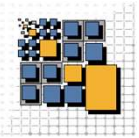


- Traditionally (before GAE), servlet containers such as (an embedded) Jetty or Tomcat were hosted with (more or less) full control over the environment:
  - You code the servlets
  - You still control a lot of the host's resources (file system, sockets etc.)
- With GAE, your access is more limited …
  because
  - Sandbox approach
  - Standard management procedures

  → However: 2nd Gen. Relaxations



GAE Java Runtime = Java Web Server Environment

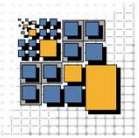


In essence, the GAE Java Runtime is a managed environment for Java-centric Web Servers:

- The application is clustered by default
  → multiple web servers serve your app
  - No sticky sessions
  - Single-Threaded servlets per default
- Modification of standard request/response headers
- Limitations on **processing time and request/response data size**
- GAE deployment descriptors
- Asynchronous request execution via cloud tasks
- Warmstart procedures (`GET /_ah/warmup`)
- Cloud ecosystem for service management (logging, routing etc.)

"cloudified", if you want

Request Header Modifications



- The following headers are removed from the request:
  - Accept-Encoding
  - Connection
  - Keep-Alive
  - Proxy-Authorization
  - Trailer
  - Transfer-Encoding
- App Engine adds some headers, e.g.
  - X-AppEngine-Country
  - X-AppEngine-Region
  - X-AppEngine-City
  - X-AppEngine-CityLatLong

Similar modifications for response headers that relate to the interaction between client and server.

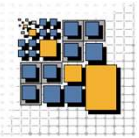
See
https://cloud.google.com/appengine/docs/standard/reference/request-headers

https://cloud.google.com/appengine/docs/standard/how-requests-are-handled

This is COOL, but puts portability at risk!

→ Is striving for portability a resonable goal when coding for GAE?

ServICE Scaling Options



Scaling options to tailor scaling behavior

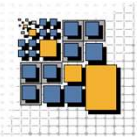
Note: Goal should be to respond in milliseconds

→ Be prepared for `DeadlineExceededException`

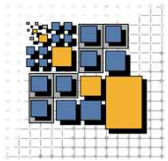
https://cloud.google.com/appengine/docs/standard/java11/how-instances-are-managed#scaling_types

Feature	Automatic Scaling	Basic Scaling	Manual Scaling
Goal	Minimum Latency (rather frontend)	Minimum Cost (rather backend)	Specific number of instances
Scaling concept	Preemptive instantiation based on KPIs, e.g., request rate and latencies	Per request instantiation	Fixed number of instances
Maximum Request Timeout	10 minutes (env. dependencies; prev. 60 seconds)	24 hours (prev. 60 minutes)	Same as basic
Background threads	No	Yes	Yes
Instance addressability	Instances are anonymous	Per instance, version and service	Same as basic

GAE Application Logging



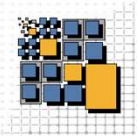
- Standard request log:
  Automatically collected log of the app requests, acessible via Google‘s Cloud Logging
- Application-specific log
  - Just use java.util.logging or Logback/SLF4J
  - Use the typical log levels
  - You COULD write to stdout and stderr, but please use a logger
- Note:
  Google offers Debugging and Profiling for the Cloud, but logs sometimes are the main source for chasing misbehaving applications



Firestore

Distributed Systems Group
Faculty Information Systems and Applied Computer Science
University of Bamberg

Google Firestore

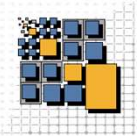


- Firestore is a **schemaless** object/document store
- Automatic distribution across storage nodes
  - The developer defines relationships between objects
  - GAE Datastore distributes data automatically
- Firestore's key promises
  - Response time depends on result set size, not the database size
  - Queries are always fast
  - Automatic data distribution
- Offered in two modes, Google recommends:
  - *"Use Firestore in Datastore mode for new server projects."*
  - *"Use Firestore in Native mode for new mobile and web apps."*

Remember that Google Cloud Storage and Cloud SQL are available as alternatives!

Remember that the "cloud" uses lots of standardized machines to serve your app … and you take care that your app is distributable!

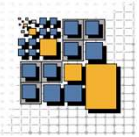
Comparison of Firestore Modes



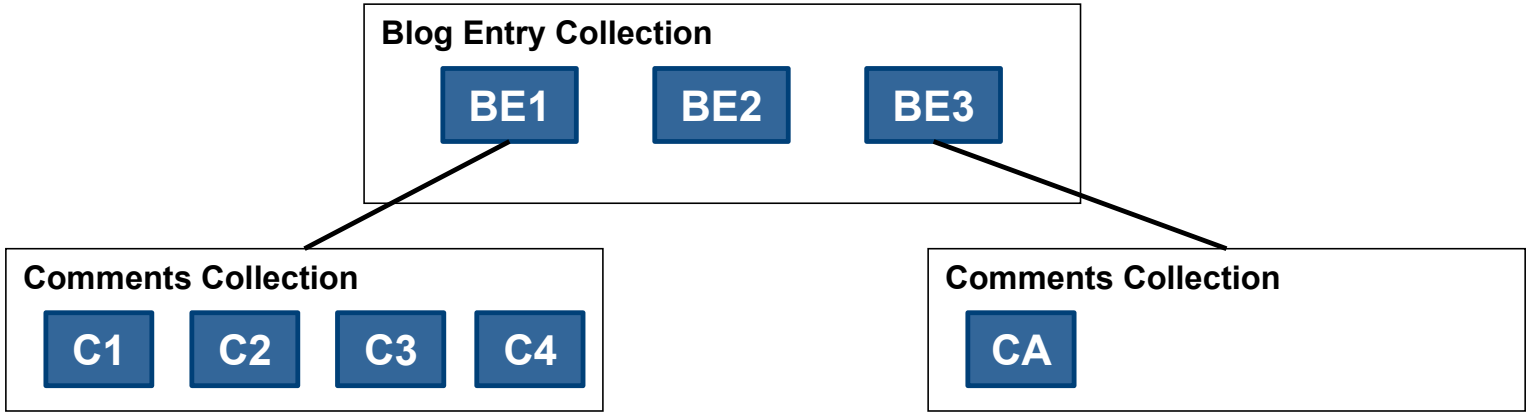
	Firestore Native Mode	Firestore Datastore Mode
Data model	Document database organized into documents and collections.	Entities organized into kinds and entity groups.
Storage Layer	New storage layer that is always strongly consistent	New storage layer that is always strongly consistent
Queries and transactions	• Strongly consistent queries across the entire database • Up to 500 documents per transaction across any number of collections. • **Limitation**: No projection queries.	• Removes the previous consistency limitations of Datastore • Strongly consistent queries across the entire database • Transactions can access any number of entity groups
Real-time updates	Supports the ability to *listen* to a document or a set of documents for real-time updates. While listening to a document or set of documents, your clients are notified of any data changes and sent the newest set of data.	Not supported
Offline data persistence	The mobile and web client libraries support offline data persistence.	Not supported
Performance	Automatically scales to millions of concurrent clients. Max 10,000 writes per second.	Automatically scales to millions of writes per second.

https://cloud.google.com/datastore/docs/firestore-or-datastore

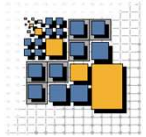
Firestore Native - Essentials



- Organized as *Collections* of **schemaless** *Documents* where each *Document* can have *Collections*
- Strongly consistent access model
- Comprehensive shallow reads
  → one complete document without sub-collections
- Realtime and offline capabilities
  (in particular for mobile use cases)



Firestore Native - Documents



- Documents represent data as Maps or POJOs
- By and large primitive types and Strings
- Denormalized Data Expected

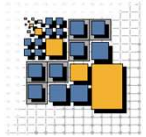
Blog Entry Collection

```
BE1
title : "a"
author :
"andreas"
```

```
BE2
title : "b"
author :
"andreas"
```

- Document Limits
  - 1 MB in size
  - 40 thousand indexed fields
  - 1 write per second sustained write rate
- Billing per document access
- Rule of thumb: One document per controller

Firestore Native – CRUD I



Getting hold of a firestore reference

```
FirestoreOptions firestoreOptions =
    FirestoreOptions.getDefaultInstance().toBuilder()
        .setProjectId(projectId)
        .setCredentials(GoogleCredentials.getApplicationDefault())
        .build();
Firestore db = firestoreOptions.getService();
```

Create

```
DocumentReference docRef = db.collection("blogentries").document("firestore");
DocumentReference docRef = db.collection("blogentries").document("firestore").collection("comments")…
```

→ implicitly creates collection / document reference if not existent

```
Map<String, Object> data = new HashMap<>();
data.put("title", "Firestore");
data.put("meta", "concept, data modeling");
ApiFuture<WriteResult> result = docRef.set(data);
```

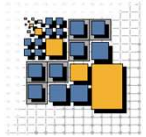
→ replaces existing data

→ yields an ApiFuture; there is some work to be done behind the scenes

→ adding a POJO is possible as well

Quickstart Snippets: https://github.com/googleapis/java-firestore/tree/main/samples/snippets

Firestore Native – CRUD II



- Update

```
docRef.update("title", "Firestore Native");
```

OR

```
Map<String, Object> data = new HashMap<>();
data.put("title", "Firestore");
docRef.update(data)
```

- Read

```
ApiFuture<QuerySnapshot> query = db.collection("blogentries").get();
QuerySnapshot querySnapshot = query.get(); // essentially two gets
List<QueryDocumentSnapshot> docs = querySnapshot.getDocuments(); // then iterate
System.out.println(document.getData().get("title"));
//Similarly
db.collection("blogentries").document("firestore").listCollections();
```

- Delete

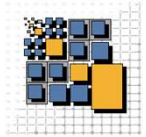
```
ApiFuture<WriteResult> res = db.collection("blogentries").document("firestore").delete();
```

→ note: sub-collections need to be deleted manually!

OR

```
Map<String, Object> upd = new HashMap<>();
upd.put("meta", FieldValue.delete());
ApiFuture<WriteResult> res = docRef.update(upd);
```

Firestore Native – Indices and Queries



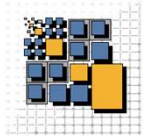
- Firestore writes an index on every field (also for nested fields)

author	documentid
andreas	firestore
andreas	appengine
johannes	lambda

rating	documentid
5	lambda
4	firestore
4	appengine

- Basic query concept
  - Find starting point in index
  - Gather adjacent documents in one go!
- Queries designed to be fast irrespective of data set size
  - One inequality filter max.
  - No Joins and No subquery filtering
- No partial fetches of documents

Firestore Native – Query Example



Simple Query

```
CollectionReference entries = db.collection("blogentries");
Query query = entries.whereEqualTo("title", "firestore");
// get query results
ApiFuture<QuerySnapshot> querySnapshot = query.get();
// then iterate
for (DocumentSnapshot document : querySnapshot.get().getDocuments()) { .. }
```

Compound Query

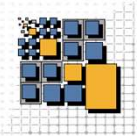
```
Query compQ = entries.whereEqualTo("author", "andreas").whereGreaterThan("rating", 5);
```

→ Note: One range filter max. per compound query

Collection Group Query

```
Query jComs = db.collectionGroup("comments").whereEqualTo("author", "johannes");
ApiFuture<QuerySnapshot> querySnapshot = jComs.get();
for (DocumentSnapshot document : querySnapshot.get().getDocuments()) {…}
```

Firestore Native - Transactions



- Firestore offers two „sorts“ of transactions
  - read and write operations on one or more documents
    → called *transactions* in Google documentation

https://cloud.google.com/firestore/docs/manage-data/transactions

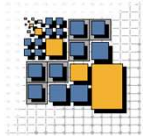
```
ApiFuture<Void> ftTx = db.runTransaction(transaction -> {
  Map map = transaction.get(docRef).getData();
  rating = map.getDouble("rating");
  transaction.update(docRef, "rating", rating * 1.1);
  return null;
});
```

  - set of write operations on one or more documents
    → called *batched write* in Google documentation

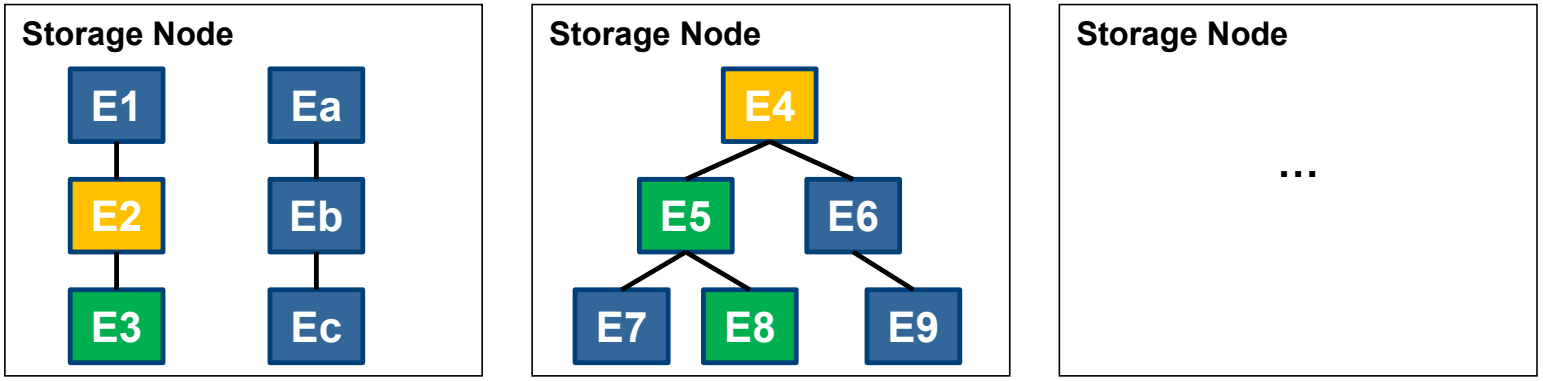
```
WriteBatch batch = db.batch();
batch.set(..); batch.update(..); batch.delete(..)
ApiFuture<List<WriteResult>> future = batch.commit();
```

- Liabilities
  - Read operations must come before write operations
  - Maximum of 500 affected documents

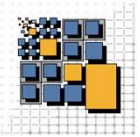
Fire Store in Datastore Mode



- Before Firestore, Google offered Datastore
- Firestore now sold as Datastore successor with
  → Firestore in *Datastore Mode*
- Automatic distribution across storage nodes
  - The developer sets up parent-child relationships
  - GAE Datastore distributes trees automatically



Firestore-Datastore Entities



GAE Datastore Entity != POJO Entity

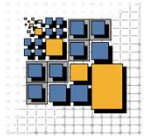
Fundamental difference to JPA entities

```
Key blogKey = datastore.newKeyFactory()
    .setKind("Blogentry")
    .newKey("firestore");
Entity blogEntry = Entity.newBuilder(blogKey)
    .set("title", "Firestore in datastore mode")
    .set("author", "andreas")
    .build();
datastore.put(blogEntry) // replaces existing entry if need be
datastore.add(blogEntry) // requires entry to be new
Datastore.update(blogEntry) // updates a previously retrieved / modified entry
```

A Datastore Entity exists of

- Its Kind
- Properties
- Key

Datastore Entity Keys



- An Entity Key is defined by
  - the entity's kind (say "Blogentry")
  - an identifier

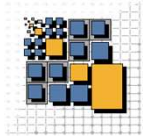
    `.newKey("firestore");`
  - An optional ancestor path consisting of *Kind-ID* pairs (locates the entity within the Datastore hierarchy)

    → could be Entry:E1/Comment:E2/Reply:E3

And a namespace if need be

- The Datastore entity hierarchy relies on keys
  - Entities without ancestors are root entities (tree roots)
  - All entities with the same root entity as ancestor make up an entity group (tree)

Datastore Entity Properties



- Set of data types restricted since firestore introduction
  - Mostly primitive / text types
  - 1 MB limitation
- Remember that GAE Datastore is schemaless!

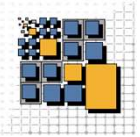
Admissible Property Types: https://cloud.google.com/datastore/docs/concepts/entities#properties_and_value_types

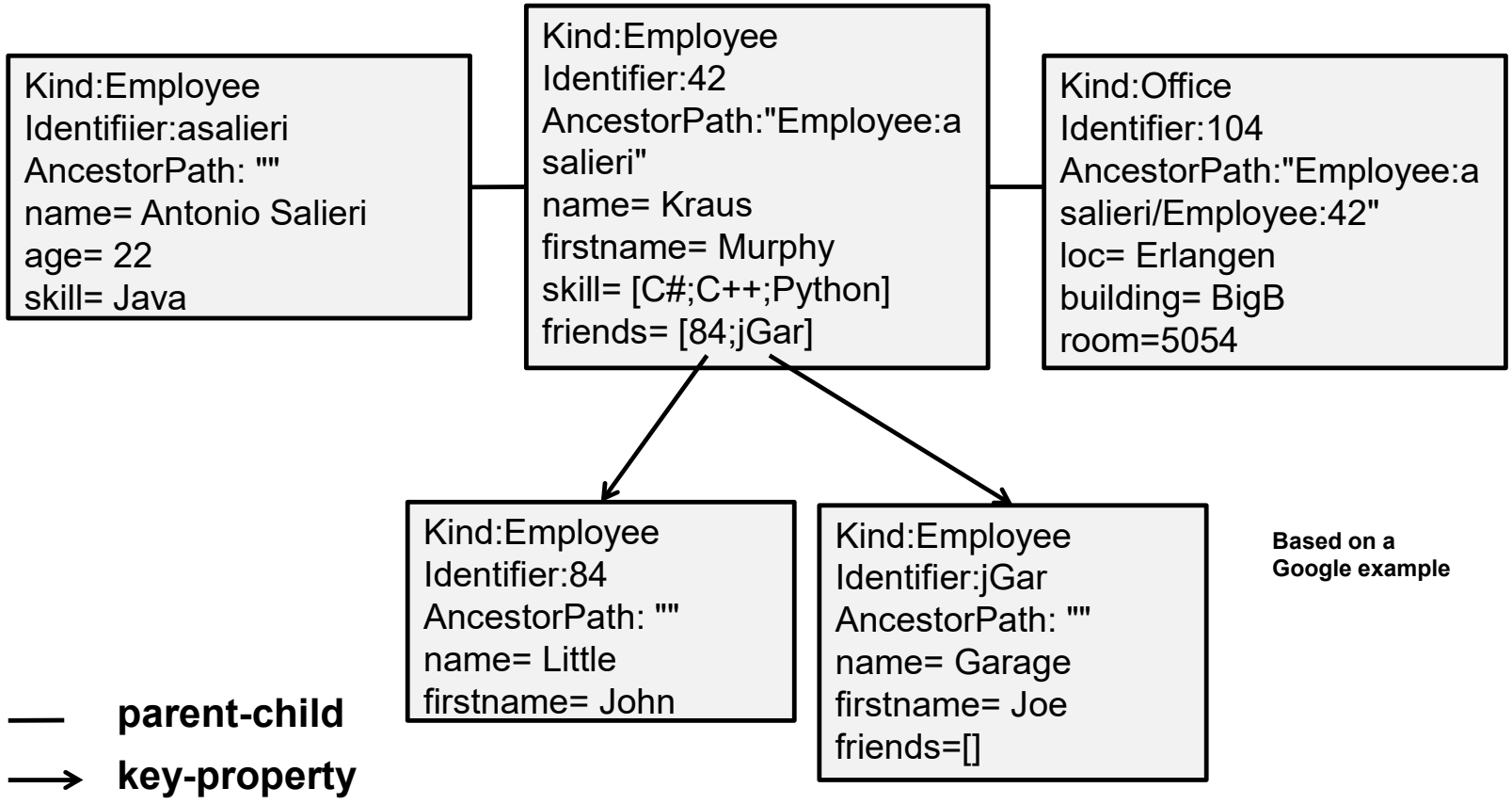
```
Entity blogEntry = Entity.newBuilder(blogKey)
    .set("title", "Firestore in datastore mode")
    .set("rating", 3)
    .build();
```

```
Entity blogEntry2 = Entity.newBuilder(blogKey2)
    .set("title", "Firestore in native mode")
    .set("rating", "GREAT")
    .build();
```

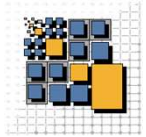
This code is fully acceptable

Sample Datastore Entities Visualized



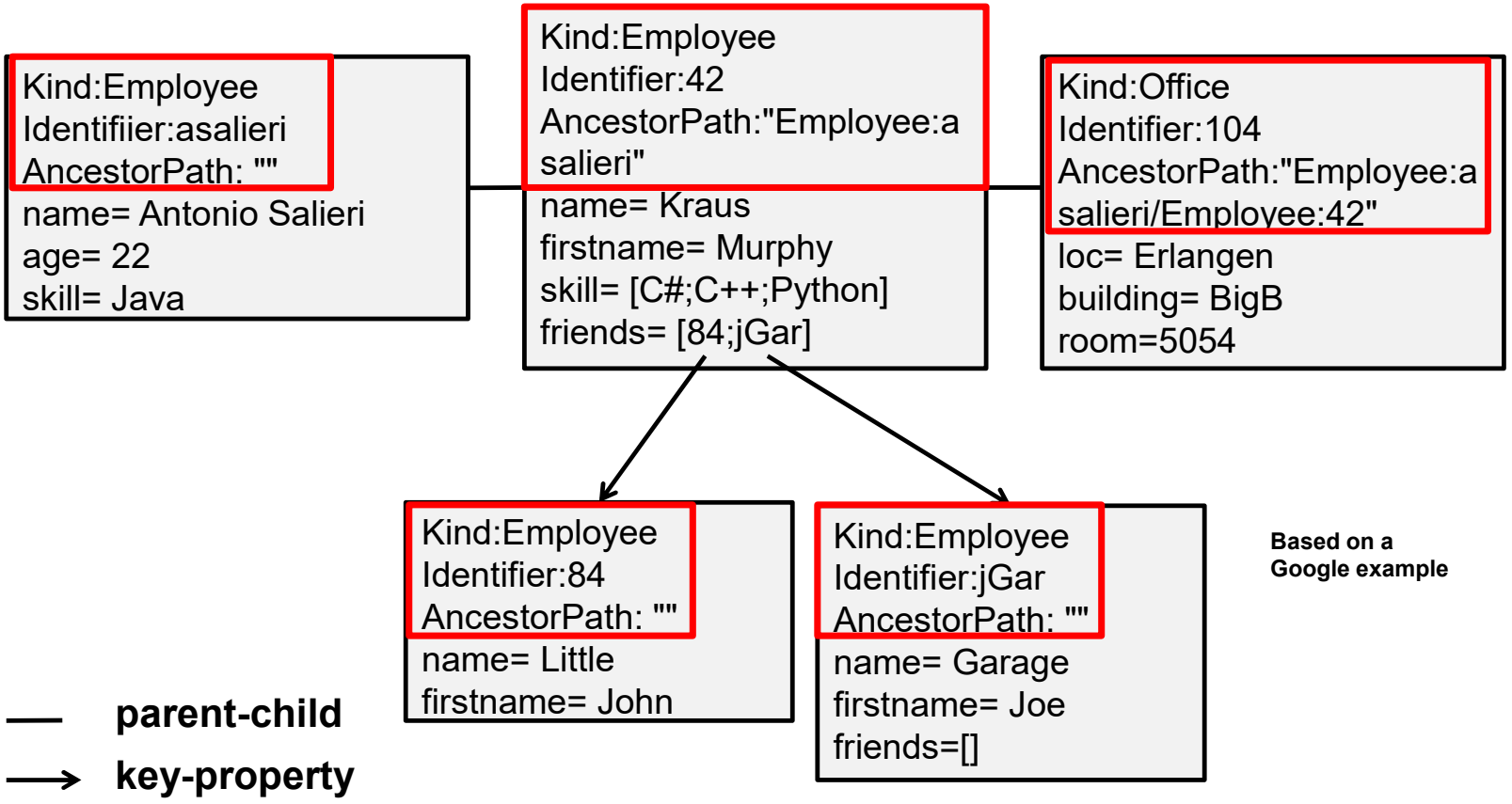


Sample Datastore Entities Visualized



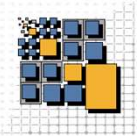
What is the key?

namespace component of a key left out here!

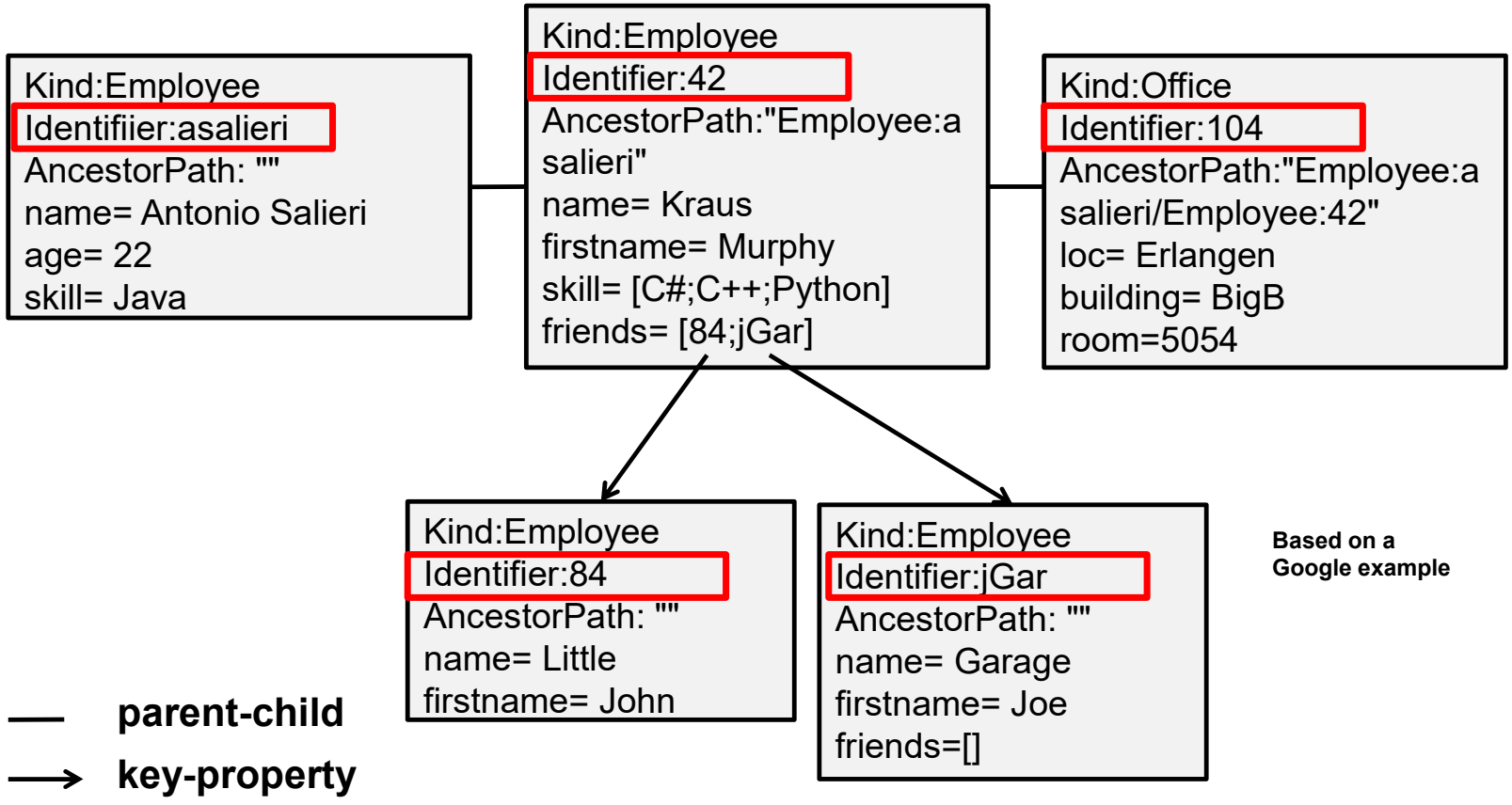


Based on a Google example

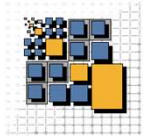
Sample Datastore Entities Visualized



Different identifiers may have different types (String, long)

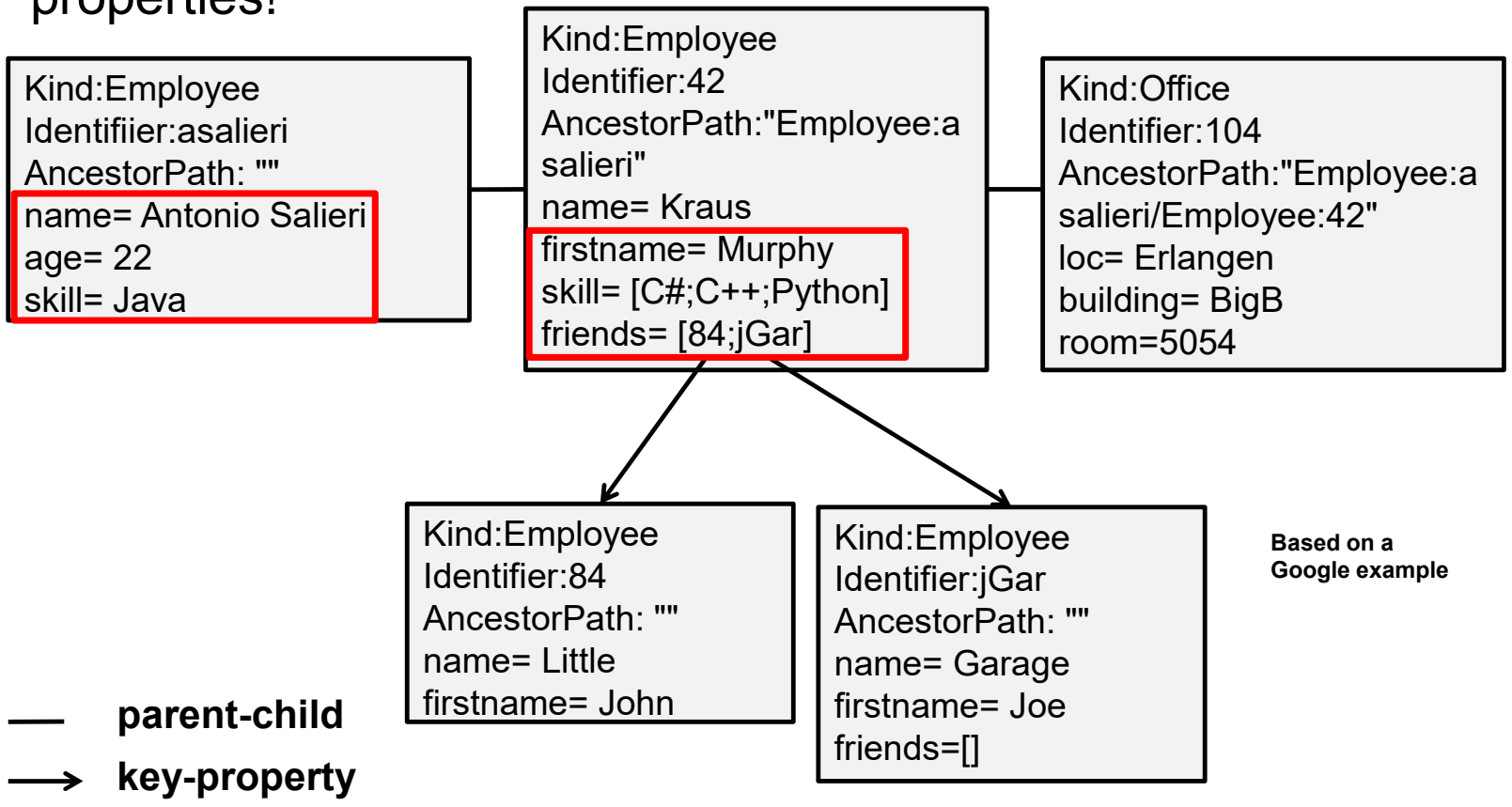


Sample Datastore Entities Visualized



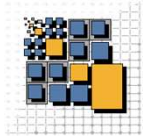
Different entities of the same kind may have different properties!

Google datastore is schemaless!



Based on a Google example

Sample Datastore Entities Visualized



Entity groups may span entities of different kinds!

An entity group is not a class!

Group 1

Kind:Employee
Identifiier:asalieri
AncestorPath: ""
name= Antonio Salieri
age= 22
skill= Java

Kind:Employee
Identifier:42
AncestorPath:"Employee:a salieri"
name= Kraus
firstname= Murphy
skill= [C#;C++;Python]
friends= [84;jGar]

Kind:Office
Identifier:104
AncestorPath:"Employee:a salieri/Employee:42"
loc= Erlangen
building= BigB
room=5054

Group 2

Kind:Employee
Identifier:84
AncestorPath: ""
name= Little
firstname= John

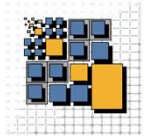
Group 3

Kind:Employee
Identifier:jGar
AncestorPath: ""
name= Garage
firstname= Joe
friends=[]

parent-child

key-property

Setting up Parent-Child Relationships



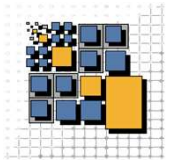
```
KeyFactory kfact = datastore.newKeyFactory()
    .addAncestors(
     PathElement.of("Employee", "asalieri"), PathElement.of("Employee", 42))
    .setKind("Office");
Key officeKey = kfact.newKey("homeOffice");

// then create actual office

Entity office = Entity.newBuilder(officeKey)
    .set("category", "home")
    .build();
```

→ Basis for automatic data distribution

→ Usable for scoping queries

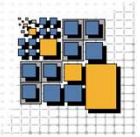


GAE Services

The Ecosystem is Dead – Long Live the Ecosystem!

Distributed Systems Group
Faculty Information Systems and Applied Computer Science
University of Bamberg

App Engine Services



Overview of legacy bundled services

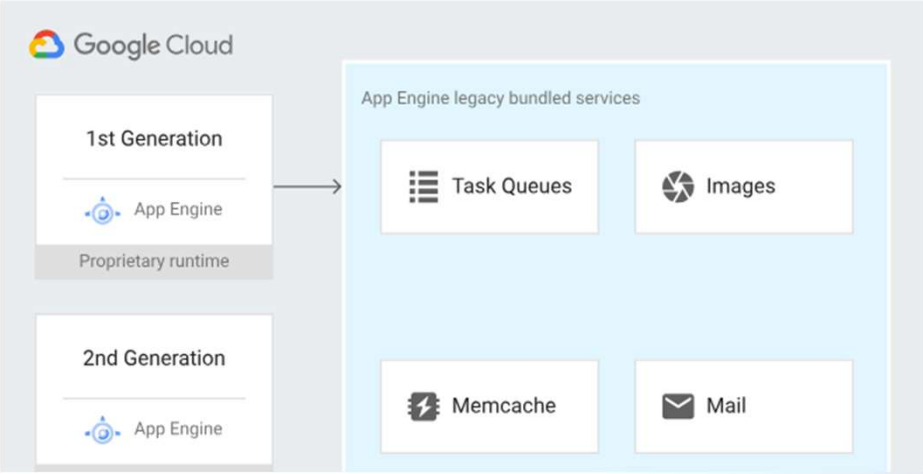
Send feedback

https://cloud.google.com/appengine/docs/standard/bundled-services-overview

https://cloud.google.com/appengine/docs/standard/java/javadoc/

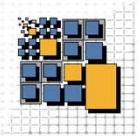
Historically, the App Engine standard environment provided several scalable, proprietary services for app development. These services were bundled with first-generation runtimes (also called App Engine *legacy runtimes*), which include Python 2, Java 8, Go 1.11, and PHP 5.5. Because several of the legacy runtimes are no longer maintained by their respective open-source communities, as an App Engine developer, you may face hard choices on whether to invest time and effort in migrating to a newer runtime or else incur growing costs to retain your app on a legacy runtime.

Recognizing this challenge, Google Cloud is committed to providing you with a more incremental migration path to newer runtimes. To reduce runtime migration complexity, Google Cloud now supports a set of App Engine legacy bundled services and their associated APIs on second-generation runtimes, which include Python 3, Java 11+, Go 1.12+, and PHP 7+. Your app can call legacy bundled services APIs for second-generation runtimes through language-idiomatic libraries.



Used to provide very useful services from log facilities and caching to storage and mail

Quotas Describe Anybody's Free Limits



- Datastore

Resource	Default limit
Stored Data (billable)	1 GB free; no maximum. Beyond free quota, billing rates apply.
Number of Indexes	200
Entity Reads	50,000 free; no maximum. Beyond free quota, billing rates apply.
Entity Writes	20,000 free; no maximum. Beyond free quota, billing rates apply.
Entity Deletes	20,000 free; no maximum. Beyond free quota, billing rates apply.
Small Operations	Unlimited

https://developers.google.com/appengine/docs/quotas

Just a selection of important and interesting figures

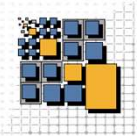
Roughly the same for 5 years!

- Mail service

Resource	Default daily limit	Maximum rate
Recipients emailed	100 messages	8 messages/minute
Admins emailed	5,000 mails	24 mails/minute
Message body data sent	60 MB	340 KB/minute
Attachments sent	2,000 attachments	8 attachments/minute
Attachment data sent	100 MB	10 MB/minute

Unchanged from 2012-2015!
Significant reduction of Mail API Calls and messages sent in 2016!
Another reduction from 4900 max rate to 32 max rate in 2017!
No change from 2017-2024

Quotas II



- Networking

> ★ **Note:** Outgoing bandwidth is a billable resource, which includes HTTP/HTTPS requests. The first 1 GB of outgoing bandwidth per day is free. There is no daily limit or maximum quota for both outgoing or incoming bandwidth.

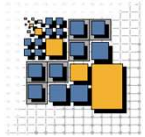
Unchanged for 6 years!
Max limits removed.

Remember:

➔ GAE Services more and more moved to general cloud services

Prices tend to be slightly falling or to be at least stable

How Much Can You Expect?



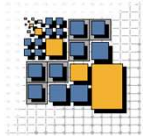
Google says (not online anymore):

“An application on a free account can use up to 1 GB of storage and up to 5 million page views a month.”

★ **Tip:** The maximum per-minute quotas accommodate high traffic levels, enough to handle a spike in traffic from your site getting mentioned in news stories. If you believe a particular quota does not meet this requirement, submit feedback in the issue tracker. Note that filing feedback is not a request for increasing your quota, but it will help us understand which quota is potentially too low for general use cases.

If you're expecting extremely high traffic levels, or for some reason your app requires particularly high quotas (for example, because of a significant product launch or large load tests), we recommend that you sign up for a support package.

Is that all?

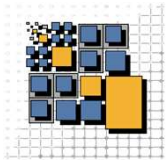


No, not by far:

- Lots of development facilities not discussed
  - Development Console
  - Admin Console
  - Logs
  - Analytics …

Some pointers for the interested:

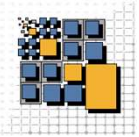
- https://cloud.google.com/appengine/docs
- https://cloud.google.com/blog/



FaaS

Distributed Systems Group
Faculty Information Systems and Applied Computer Science
University of Bamberg

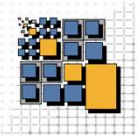
Agenda



- What is Serverless?
- Function as a Service (FaaS)
- Traditional 3-tier vs. FaaS Architecture
- Benefits and Drawbacks

- Hands-On – AWS Lambda

What is Serverless?



Informal Def.:

"Serverless denotes a cloud computing style where the user does not care about concrete server instances."

(of course there are machines/servers).

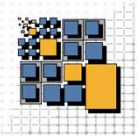
- FaaS (Lambda, Cloud Functions)
- Existing PaaS platforms, e.g., App Engine
- Storage, Messaging

Was a real hype

https://cloud.google.com/serverless/?hl=en

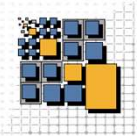
- Why the hype about Serverless?
- What's new about this trend?
  (GAE (2008) is not so new)
  - As you know, GAE automatically scales instances
  - No operational tasks for developers

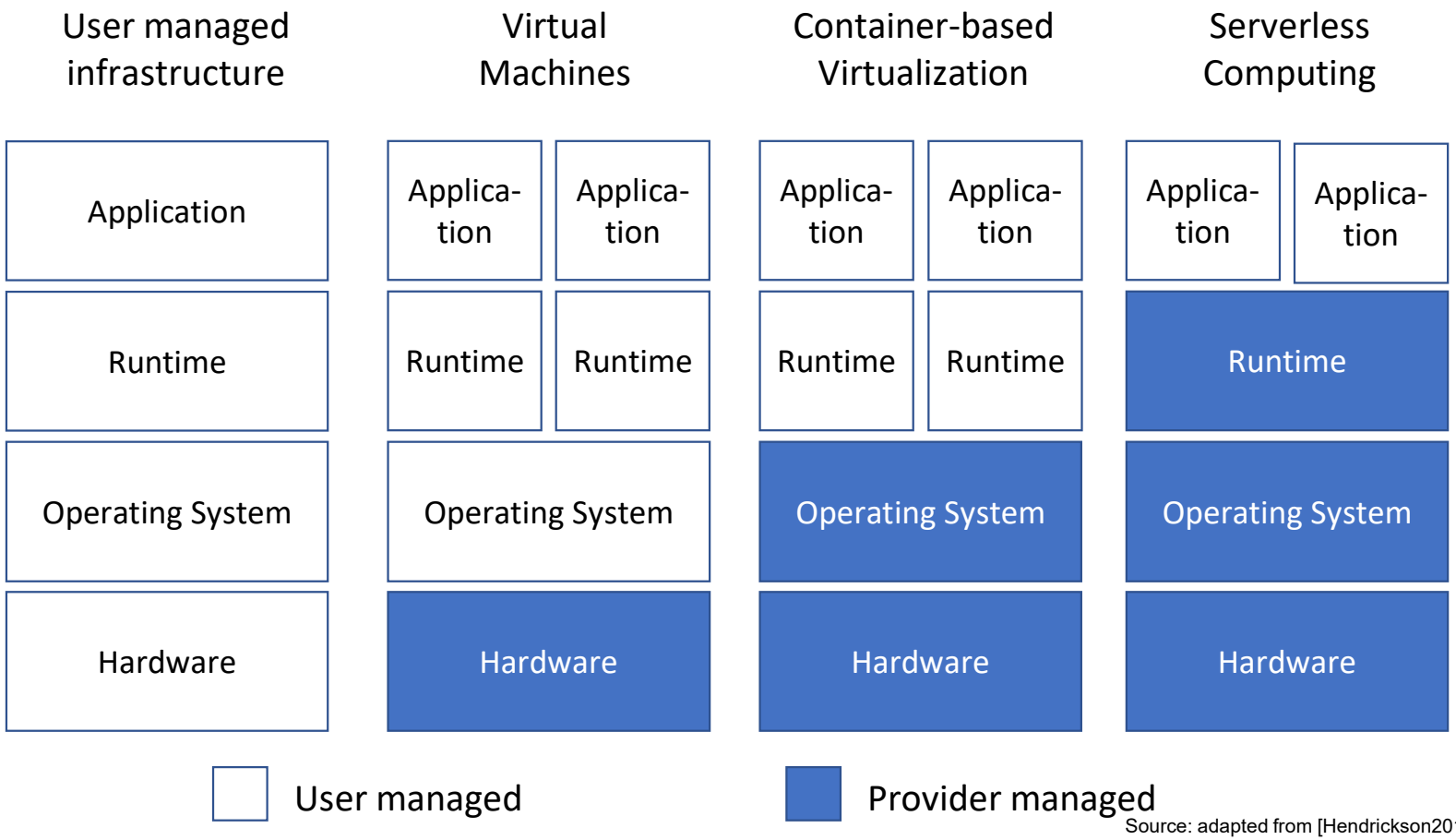
Cloud Computing – Recap [Mell2011]



- Essential Characteristics
  - On-demand self-service
  - Broad network access
  - Resource pooling
  - Rapid elasticity
  - Measured service
- Service Models
  - Software as a Service (SaaS)
  - **Function as a Service (FaaS) (Main part of Serverless | A new service model?) [Savage2018]**
  - Platform as a Service (PaaS)
  - Infrastructure as a Service (IaaS)

Serverless in the Compute Stack …





Source: adapted from [Hendrickson2016]

… and Classical Categorization

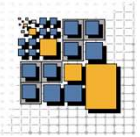
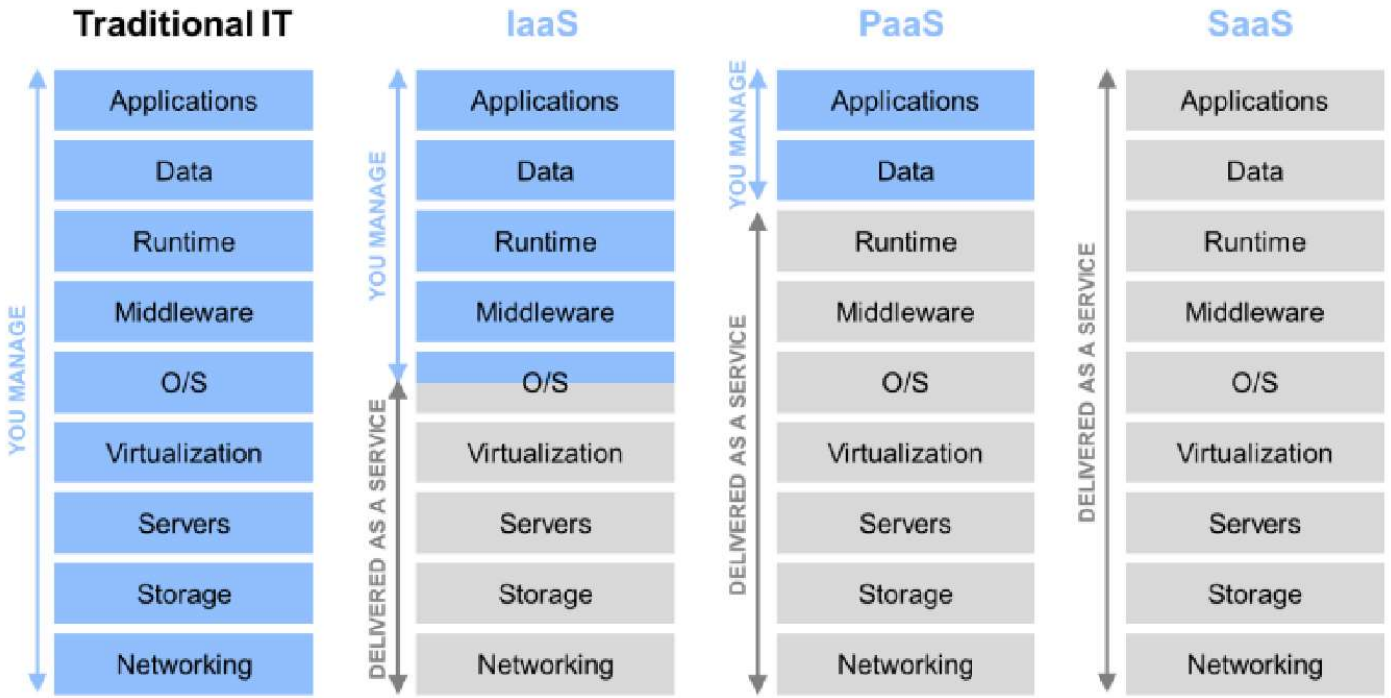


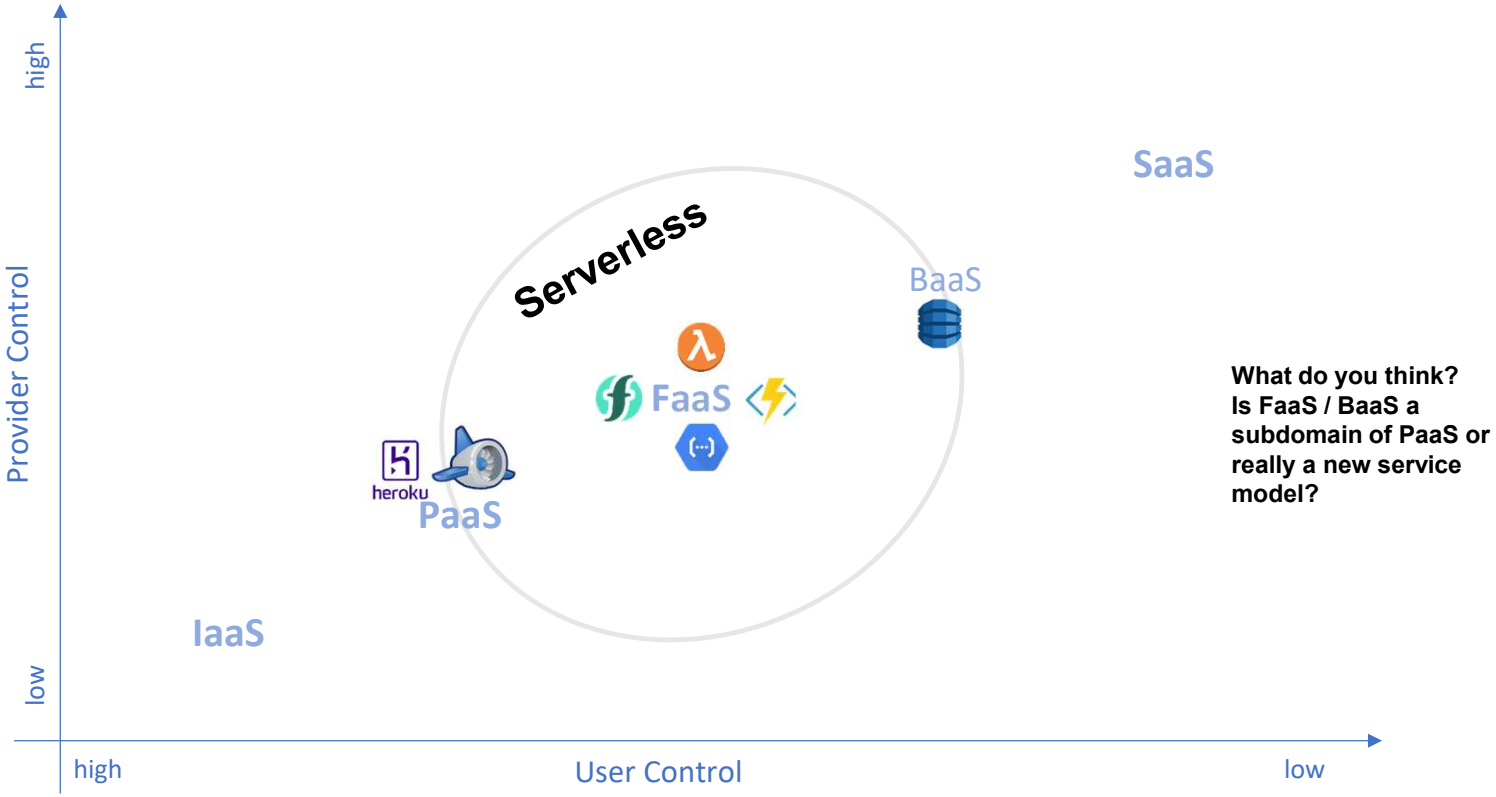
FIG. 17: CAPTURING CLOUD BENEFITS

cf. Cloud Part A slide 2



Source:
Microsoft Cloud Computing Whitepapers,
"The Economics of the Cloud", 2010

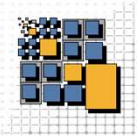
FaaS / BaaS as new Service Models?



What do you think?
Is FaaS / BaaS a subdomain of PaaS or really a new service model?

Source: adapted from [Eyk2017]

Why the hype about Serverless?



- Function as a Service (FaaS) caused this hype since 2014 (AWS Lambda release)

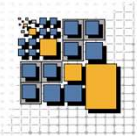
  Remark: Serverless and FaaS sometimes used as synonyms
- FaaS is a neat tool for hybrid architectures

The marketing machine works!

Let‘s talk about FaaS… ☺

- FaaS providers offer a fully managed runtime environment for executing functions in the cloud – easy to use without getting started with a web framework
- Promise of FaaS: Users don’t care about any nitpicky server configurations
- Endless scalability enabled by statelessness of functions
- Proceeds the trend to more fine-grained architectures started with microservices

Function as a Service (FaaS) [Eyk2017]

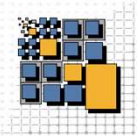


- FaaS is an event-driven computing model
- FaaS is a cloud service model, where a provider
  - provides a managed execution environment
  - abstracts (nearly) all operational tasks (gain of control)
  - facilitates auto-scaling of short-lived, context-unaware cloud functions
  - facilitates scaling to zero (no idle running functions, unique for FaaS?)
  - enables a pay-per-use cost model
    (most granular billing option in present cloud technologies)
- FaaS is a cloud service model, where a user
  - writes single-threaded, performant, stateless cloud functions
  - specifies only a few configuration parameters (e.g. RAM, timeout)

IMO, a PaaS focusing on functions.

Do phi{Faas/PaaS} and check correctness of statements

FaaS: Cloud Function Execution



- Example: File upload in a S3 bucket, event processing
- Functions are executed in lightweight containers
- Cold starts are/were a major problem for FaaS

Hey guys, what's wrong with pooling?

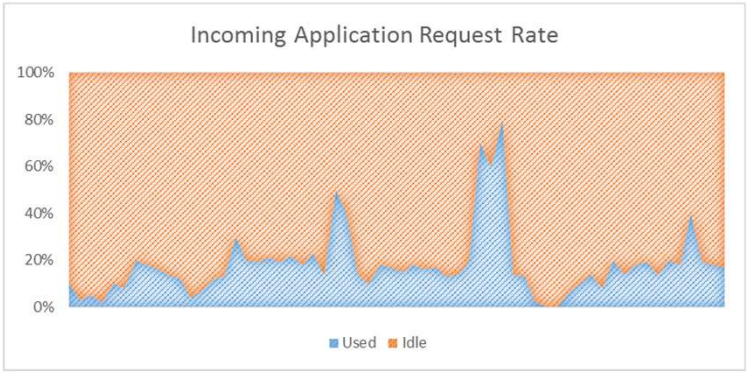
Source: https://www.youtube.com/watch?v=eOBq__h4OJ4

- Typical triggers
  - Http based triggers (e.g. API Gateway)
  - CRON triggers (e.g. Cloud Watch)
  - Queuing/ Notification triggers (e.g. SNS, SQS)
  - Database triggers (e.g. DynamoDb, S3)

FaaS: Powerful Use Cases

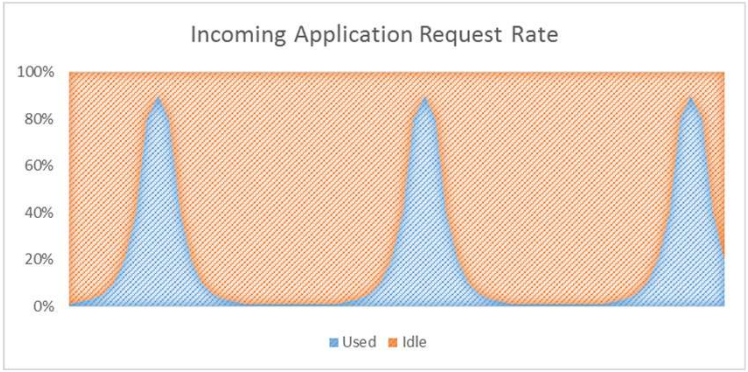
Solving especially under-/overprovisioning problem

- Bursty Workloads

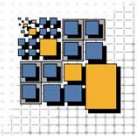


You can do this with IaaS as well. FaaS makes it harder to get it wrong ;-)

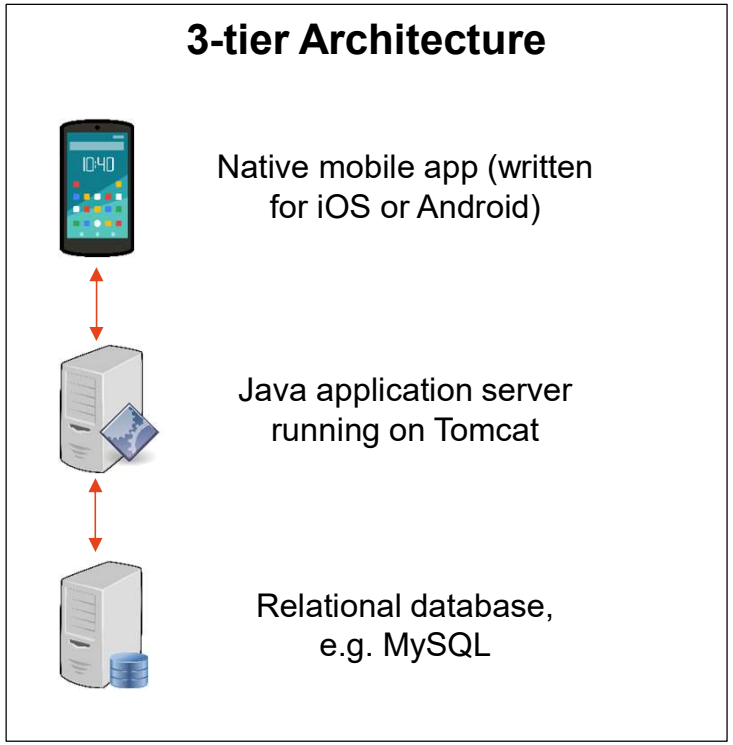
- Periodic Workloads

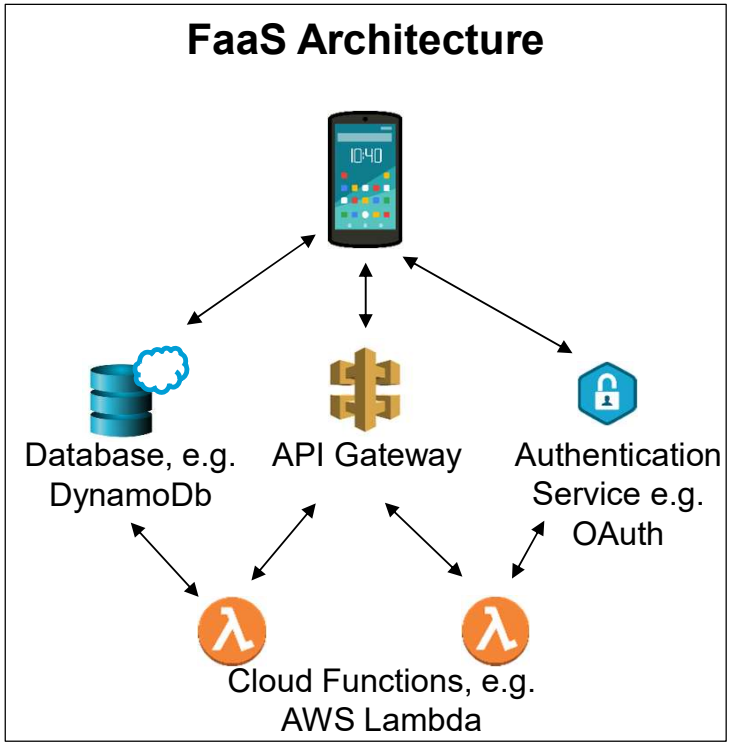


3-tier Architecture vs. FaaS Architecture



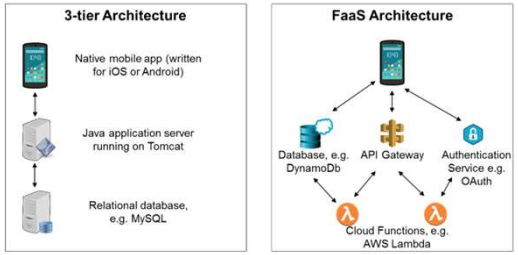
Scope: Application with a mobile user interface, an authentication mechanism and some gameplay logic





Source: adapted from [Roberts2017]

Migration Benefits



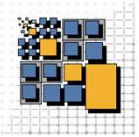
- Expertise in configuring, deploying and operating Java application and database server required (Ops)
- Also operating the host systems (security patches, licensing etc.) (Ops)
- Thinking about security, scalability, availability etc. by ourselves (DevOps)

Seems more complicated at first glance, but

- hardly any operational tasks are done by the service providers
- the code is focused on core business
- scaling, security etc. are service inherent
- components are more decoupled and easier to change

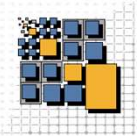
?

FaaS Benefits



- Scaling On-Demand and to zero
- Calculation Model (pay per use)
- Reduced Labor Cost
- Time to Market
- DevOps
- Rich Provider Ecosystems
- Quality settings per function, e.g., security
  (each cloud functions can be treated separately)

FaaS: Challenges / Liabilities



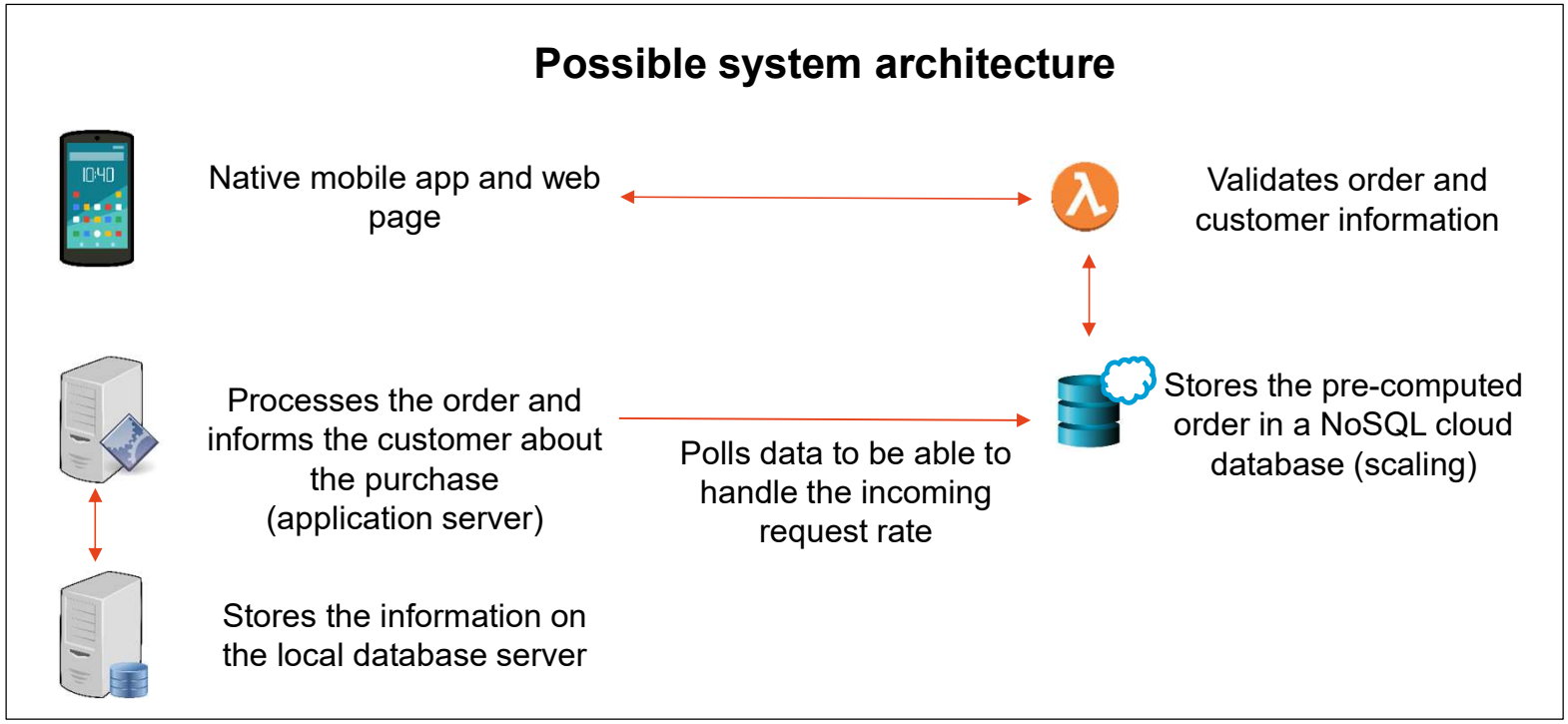
- Provider Limits (i.e. timeout and memory setting)
- Testing (i.e. Integration Testing)
- Cold Starts (i.e. for time critical applications)
- Vendor Lock-In
- Loss of Control (HW/SW beneath)
- Network Latency within a FaaS Application

Think of retest in case of env changes!

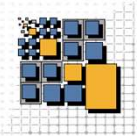
Hybrid Applications

- Use Case: Event organizer – Madonna Concert

  Due to the high expected traffic, the reactive component is implemented as cloud function

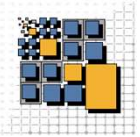


Hybrid Applications



- Hybrid Applications come in handy when individual services have diverging scaling / env. requirements
- Always consider the scaling properties of your traditional (non-Serverless) application part and the added cloud part
- Decoupling the two parts of your application via messaging or a database is the best practice to avoid DoS attacking yourself

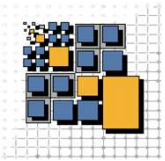
Hands-On: FaaS Provider



- Language matrix

AWS Lambda	Java	Node.js	C#, Python, Go, PowerShell, Ruby, Custom Runtimes
Google Cloud Functions	Java	Node.js	Python, Go, .NET, Ruby, PHP
Microsoft Azure Functions	Java	Node.js	C#, F#, Python, TypeScript, PowerShell
IBM Cloud Functions (OpenWhisk)	Java	Node.js	Swift, Go, PHP, Python, any language via Docker container

Last update on 12/7/2022



Hands-On

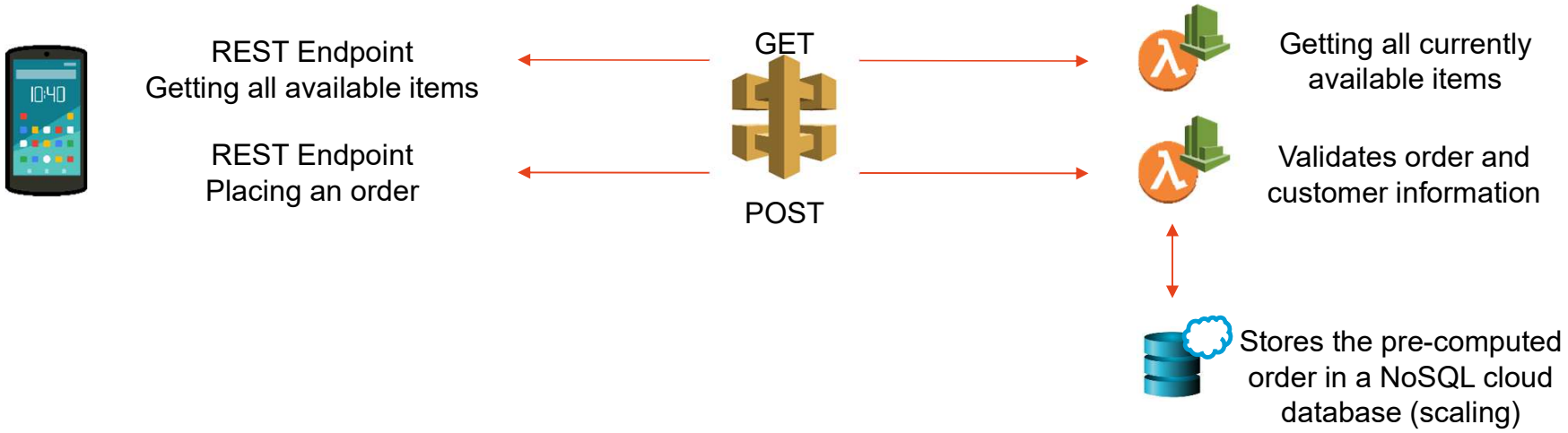
AWS Lambda Functions written in Java, enabled by Eclipse Plugin

https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/

Distributed Systems Group
Faculty Information Systems and Applied Computer Science
University of Bamberg

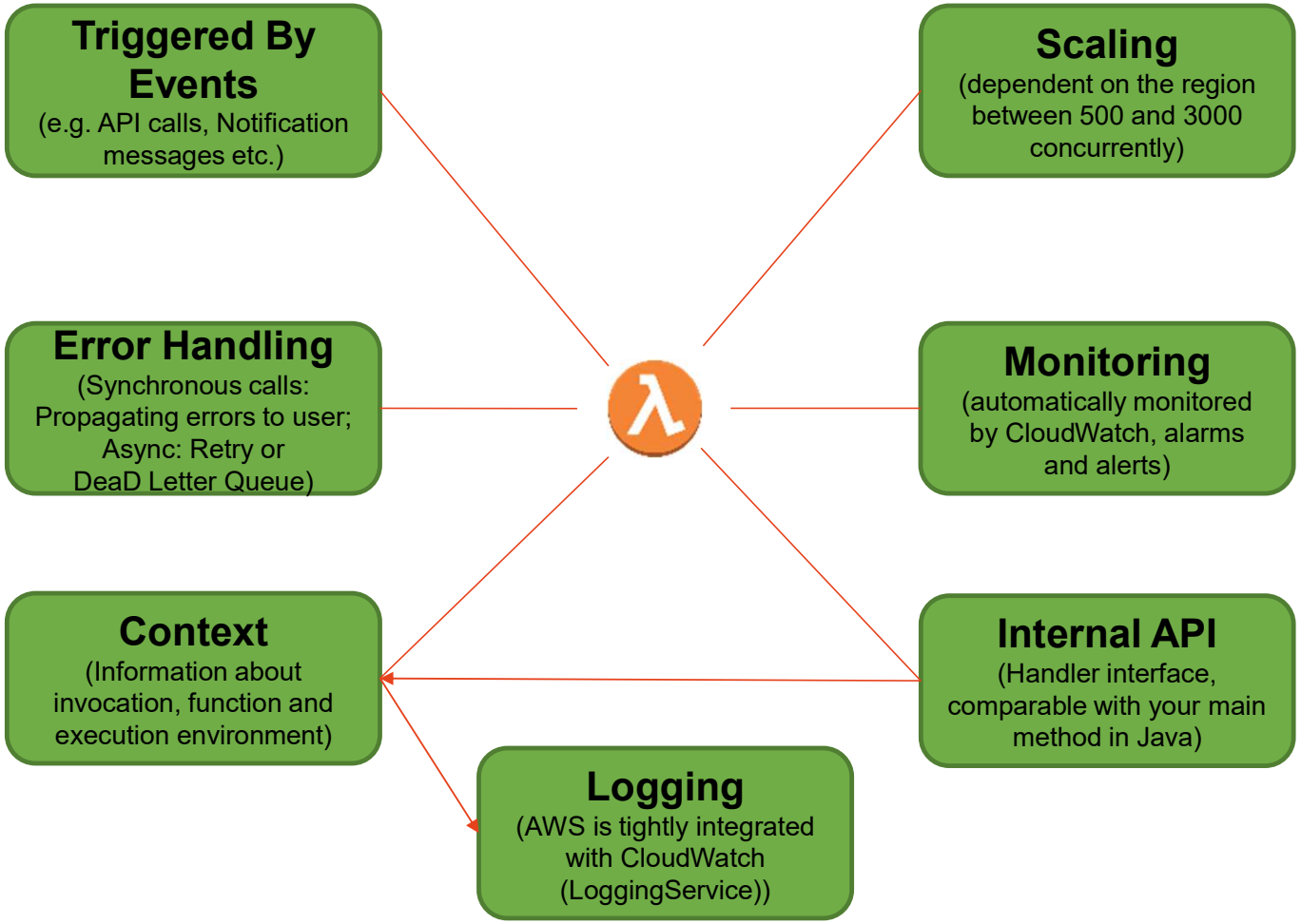
Watch out for SDK changes: https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/home.html

Scenario

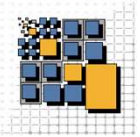


- Lambda is automatically integrated with CloudWatch, where all the logs are stored
- The scenario and all required steps are also described here: https://github.com/johannes-manner/faas-demo

Environment of a single Function



AWS Lambda – Function template



```
public class AllItemsHandler
             implements RequestHandler<Object, String> {

    @Override
    public String handleRequest(Object input, Context context) {
        context.getLogger().log("Input: " + input);
        return "Hello from Lambda!";
    }
}
```

Handler as a SAM (Single Abstract Method) Interface

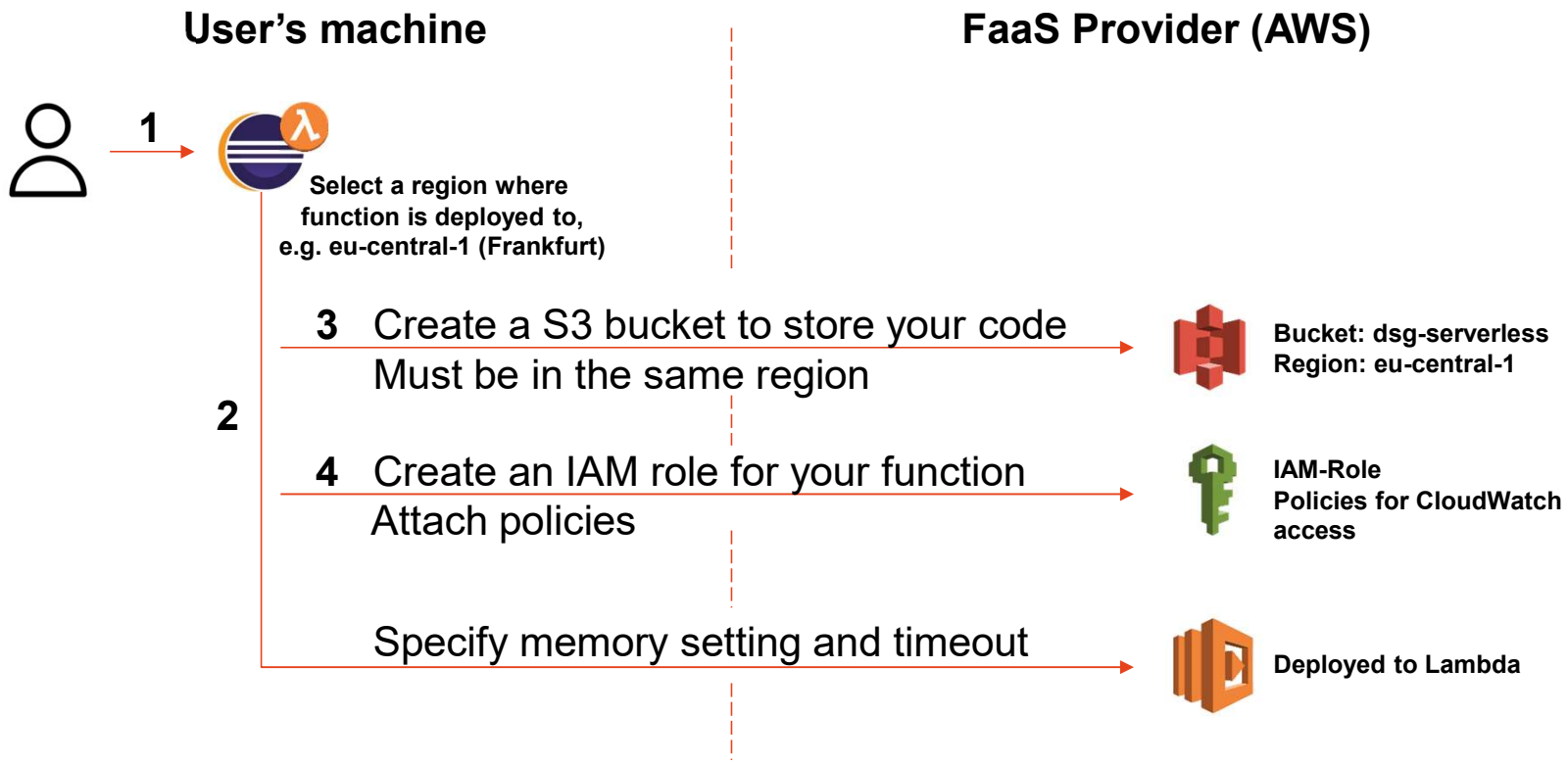
Input type of your Lambda function. Pre-integrated classes like SNSEvent or self-written POJOs are possible here

Output type of your Lambda function.

Context Object with a lot of environmental support and information (e.g. logger, function name, log group etc.)

CloudWatch creates for each function a *log group*, where the *log streams* (correspond with the a single function container) are stored. Log streams include logged messages.

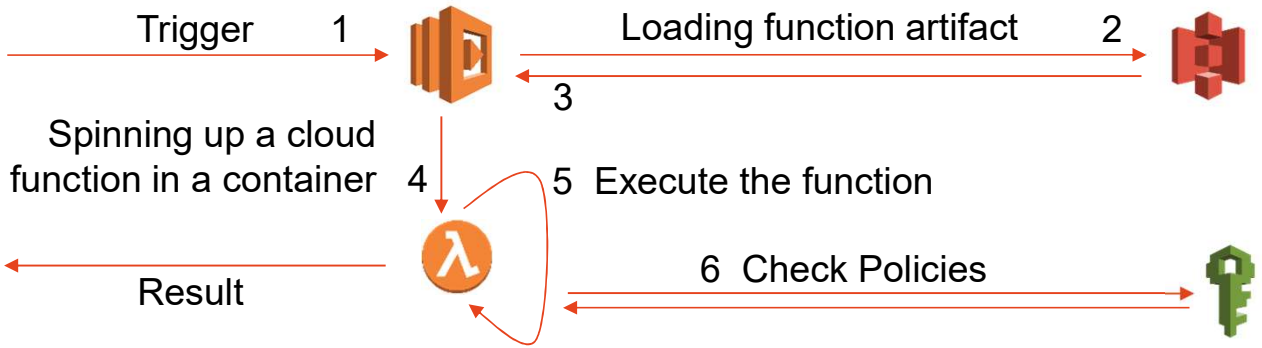
Getting all available items – Implementation and Deployment



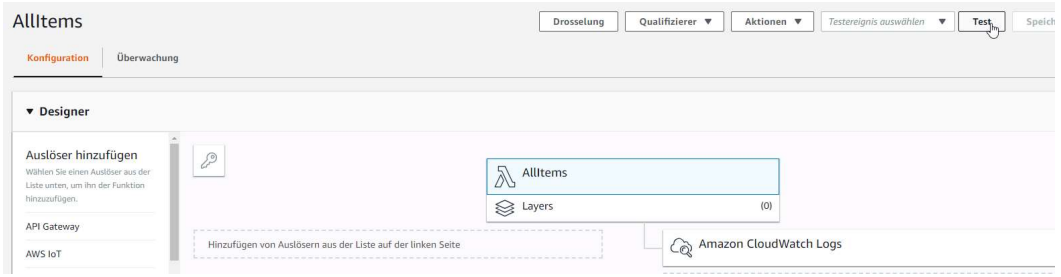
Further Information on the GitHub Page: *Get all items cloud Function - Doing all steps manually*

Invoking your function

- Sketch – Logical flow on AWS

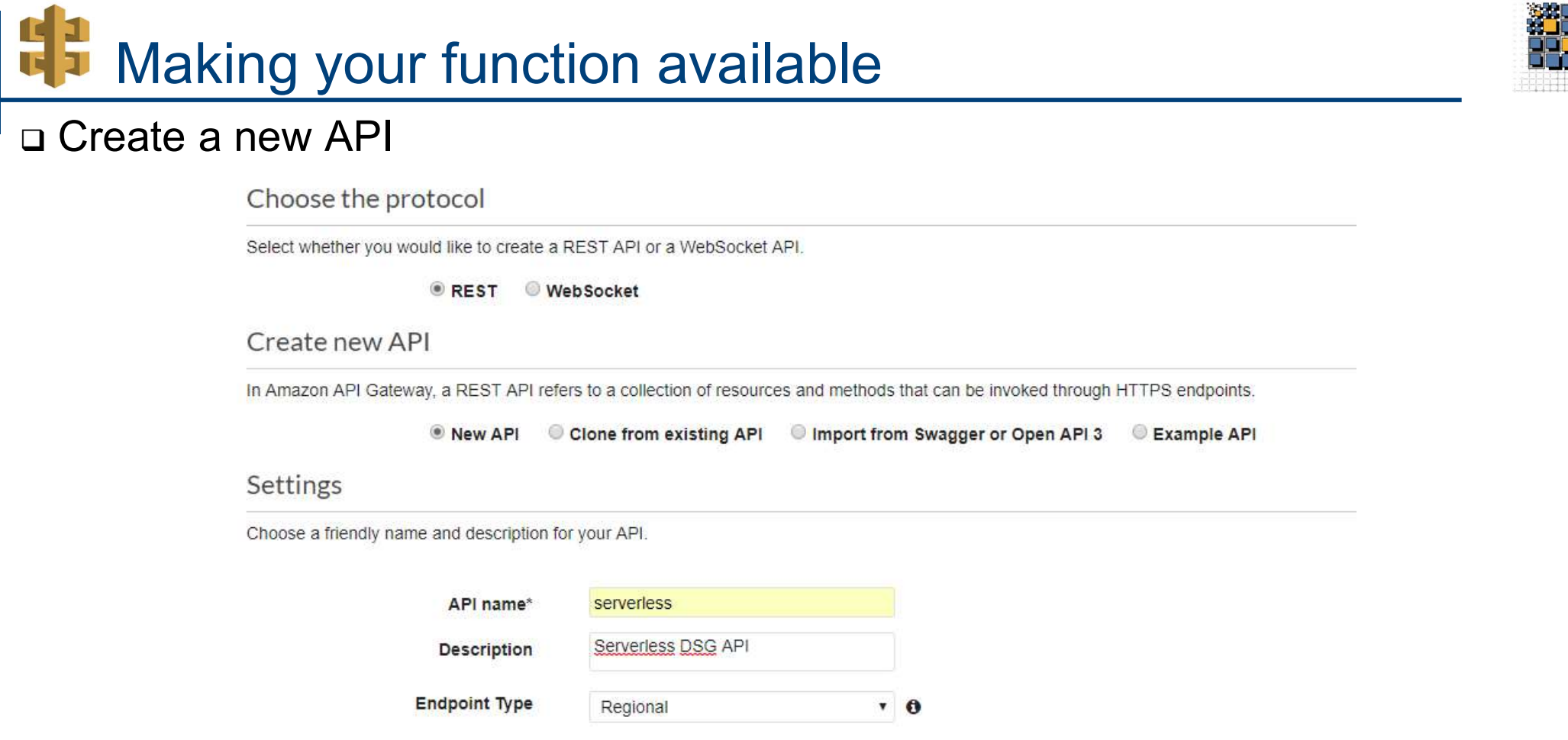


- Go to the AWS console to your function
- Invoke the function with any test content
- Number of concurrent function invocations is unbounded. AWS limits the number per account to 1000 concurrent function running at the same time.

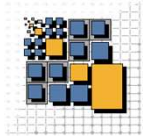


Making your function available

- Create a new API

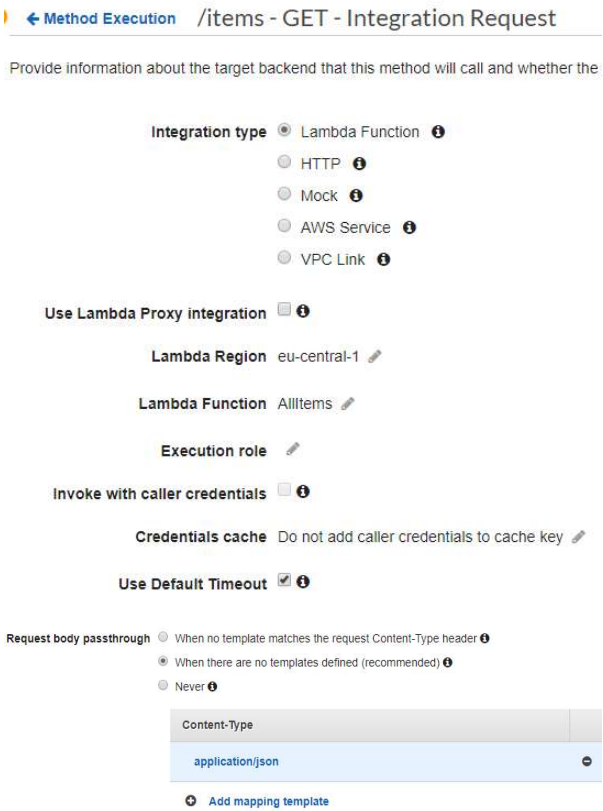


Making your function available

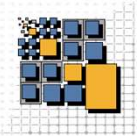


- Actions -> Create Resource :
  items and enable CORS
  (comparable to @Path in JAX-RS)
- Click on your new resource *items*
- Action -> Create Method
- Select GET
  (comparable to @GET in JAX-RS)
- Click on Integration Request
- Click on MappingTemplate
  Controls the mapping of the request data
  Stores querystring parameters in a key:value list

```
#set($allParams = $input.params())
#set($params = $allParams.get("querystring")){
  #foreach($paramName in $params.keySet())
  "$paramName" : "$util.escapeJavaScript($params.get($paramName))"
  #if($foreach.hasNext),#end
  #end
}
```



Deploy your API

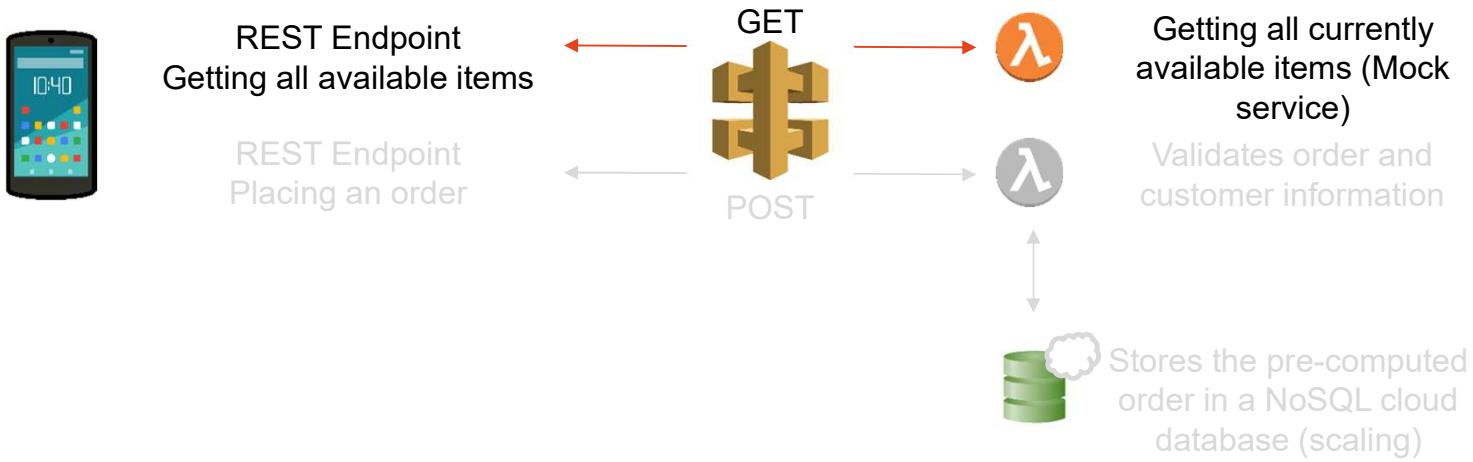


- Action -> Deploy API
- Create New Stage, use “dev” as stage name
- Deploy

- Test your API Gateway and your function
- Open a REST interaction tool like INSOMNIA (https://insomnia.rest/)
- Creating a GET request

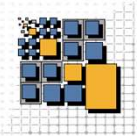
  https://####.execute-api.eu-central-1.amazonaws.com/dev/items?maxPrice=4000

Scenario – so far



- Next steps:
  - Setting up DynamoDb
  - Implementing the second integration via POST

Implementation process so far . . .



Situation

- Working a lot with the AWS user interface

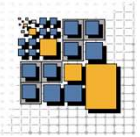
Problem

- Configuring all services is tedious and error prone
  -> no reproducibility of your application architecture

Solution

- Infrastructure automation (in our use case via CloudFormation)

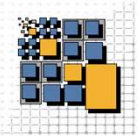
Cloud Formation



- CloudFormation is an “Infrastructure as Code” service
- Modelling your application in a .json or .yaml template
- CloudFormation automatically creates or updates the resources specified in the template
- Benefits:
  - Version control of your infrastructure (easier to detect changes over time)
  - Automate your deployment and configuration (reproducibility, less errors)

Source: https://aws.amazon.com/cloudformation/

Setting up DynamoDB via CloudFormation



1. **Specify your infrastructure**
   Example is written in .yaml format (.json is also possible)

   Line 01, 02 are metainfos
   04 OrderTable is the logical ID of our table
   05 Type of our resource
   06-16 configuration of your DynamoDB table

2. Create your application stack via web user interface or CLI:
   ```
   > aws --region eu-central-1 cloudformation create-stack --stack-name serverless-stack --template-body file://cloudFormation.yaml
   ```

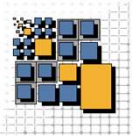
   --region <name>: where your cloudformation stack is deployed to and also your resources

Source: https://docs.aws.amazon.com/cli/latest/reference/cloudformation/create-stack.html

cloudFormation.yaml

```
01 AWSTemplateFormatVersion: '2010-09-09'
02 Description: Creates a dynamoDb instance
03 Resources:
04   OrderTable:
05     Type: AWS::DynamoDB::Table
06     Properties:
07       AttributeDefinitions:
08         - AttributeName: ID
09           AttributeType: S
10       KeySchema:
11         - AttributeName: ID
12           KeyType: HASH
13       ProvisionedThroughput:
14         ReadCapacityUnits: 5
15         WriteCapacityUnits: 5
16       TableName: Order
```

Setting up a cloud function via CloudFormation



1. Prepare your function

- Upload the function source to a S3 bucket.
- The S3 bucket and the function deployment have to be in the same region

2. Specify your function configuration

02&13 Each Lambda Function needs a IAM Role for accessing other services

05-07 The code artifact is located within the same region in a S3 bucket

09-10 The two basic configuration parameters on most platforms

11 Entry for the FaaS Platform (comparable with main-method)

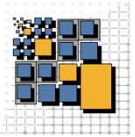
12 Selected your source code's runtime

14-16 Configuration via environment variables. Also visible in the web UI

cloudFormation.yaml

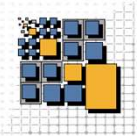
```
01 StoreOrderFunction:
02     DependsOn: LambdaExecutionRole
03     Type: AWS::Lambda::Function
04     Properties:
05       Code:
06         S3Bucket: !Ref 'DSGBucket'
07         S3Key: StoreOrder.jar
08       FunctionName: DSG-StoreOrder
09       MemorySize: 512
10       Timeout: 20
11       Handler: de….StoreOrderHandler
12       Runtime: java8
13       Role: !GetAtt 'LambdaExecRole.Arn'
14       Environment:
15         Variables:
16           REGION: !Ref 'AWS::Region'
```

Setup via CloudFormation



```
"Resources": {
  "OrderTable": {
    "Type": "AWS::DynamoDB::Table", …
  },
  "LambdaExecutionRole": {
    "Type": "AWS::IAM::Role", …
  },
  "AllItemsFunction": {
    "DependsOn": "LambdaExecutionRole", …
    "Type": "AWS::Lambda::Function",
  },
  "StoreOrderFunction": {
    "DependsOn": "LambdaExecutionRole",
    "Type": "AWS::Lambda::Function", …
  },
  "ItemApi": {
    "Type": "AWS::ApiGateway::RestApi", …
  },
  "ItemResource": {
    "DependsOn": "ItemApi",
    "Type": "AWS::ApiGateway::Resource", …
  },
  "LambdaPermissionAllItemsFunction": {
    "Type": "AWS::Lambda::Permission", …
  },
  "LambdaPermissionStoreOrderFunction": {
    "Type": "AWS::Lambda::Permission", …
  },
  "ItemGet": {
    "DependsOn": ["…"],
    "Type": "AWS::ApiGateway::Method", …
  },
  "ItemPost": {
    "DependsOn": ["…"],
    "Type": "AWS::ApiGateway::Method", …
  },
  "ItemApiDeployment": {
    "DependsOn": ["..."],
    "Type": "AWS::ApiGateway::Deployment", …
  }
}
```

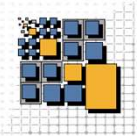
Summary



- Cloud Functions are a fast way to get scalable services up and running
- “Infrastructure as code” offerings help to automate most of the ops tasks
- Focusing only on the business logic saves time and money

- BUT: Hybrid architectures should be designed carefully. Think about the different scaling properties!

Literature



- [Eyk2017] E. van Eyk et al. “The SPEC Cloud Group’s Research Vision on FaaS and Serverless Architectures,” in Proceedings of the 2nd International Workshop on Serverless Computing, Las Vegas, Nevada, 2017, pp. 1–4.
- [Hendrickson2016] S. Hendrickson et al., “Serverless Computation with openLambda,” in Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing, Denver, CO, 2016, pp. 33–39.
- [Mell2011] P. M. Mell and T. Grance, “The NIST definition of cloud computing,” National Institute of Standards and Technology, 2011.
- [Roberts2017] M. Roberts and J. Chapin, What Is Serverless? O’Reilly Media, 2017.
- [Savage2018] N. Savage, “Going Serverless,” Communications of the ACM, vol. 61, no. 2, pp. 15–16, 2018.