
GAE — Google App Engine

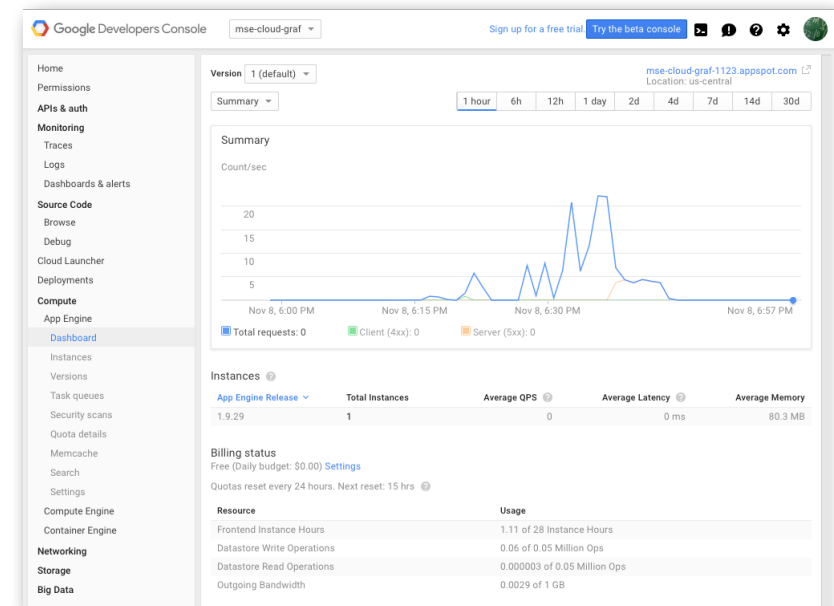
Prof. Dr. Marcel Graf

Google App Engine

Introduction

- Google App Engine is a PaaS for building scalable web applications and mobile backends.
- Makes it easy to deploy a web application:
 - The client (developer) supplies the application's program code.
 - Google's platform is responsible for running it on its servers and scaling it automatically.
 - The platform offers built-in services and APIs such as NoSQL datastore, memcache and user authentication.
- There are some restrictions regarding the supported application types.
 - The application cannot access everything on the server.
 - The application must limit its processing time.
- Launched in April 2008 (preview)
 - In production since September 2011
- Supported programming languages and runtimes:

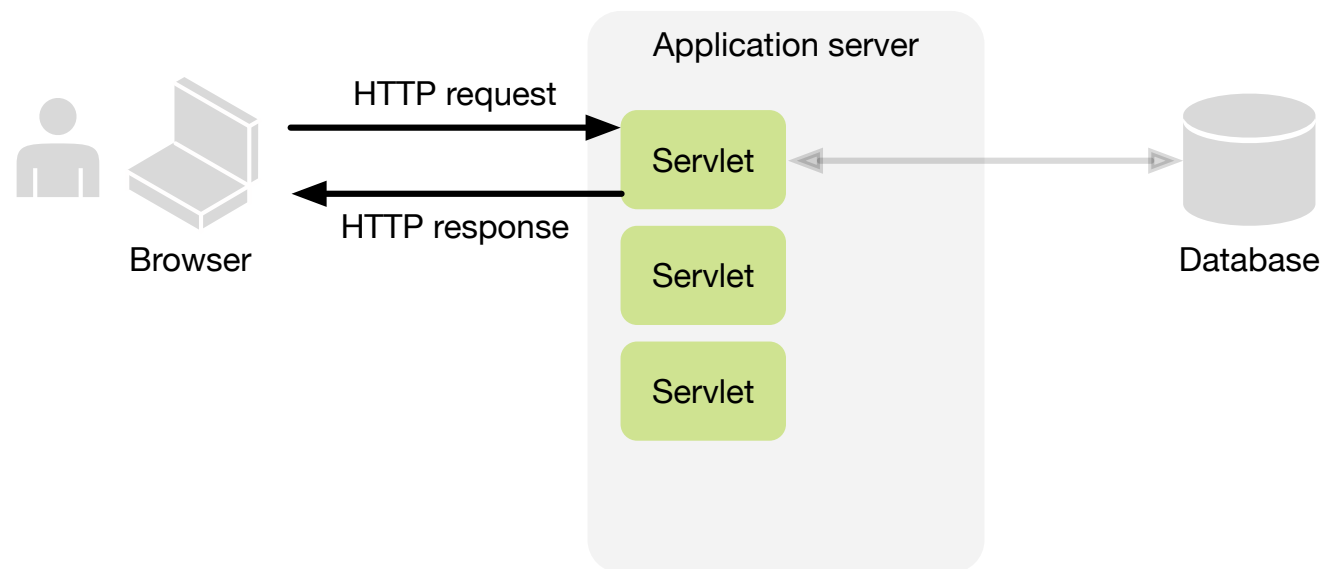
■ Go	■ Node.js
■ PHP	■ .NET
■ Java	■ Ruby
■ Python	■ ...



Platform as a Service

Reminder: Web applications in Java — *Servlets*

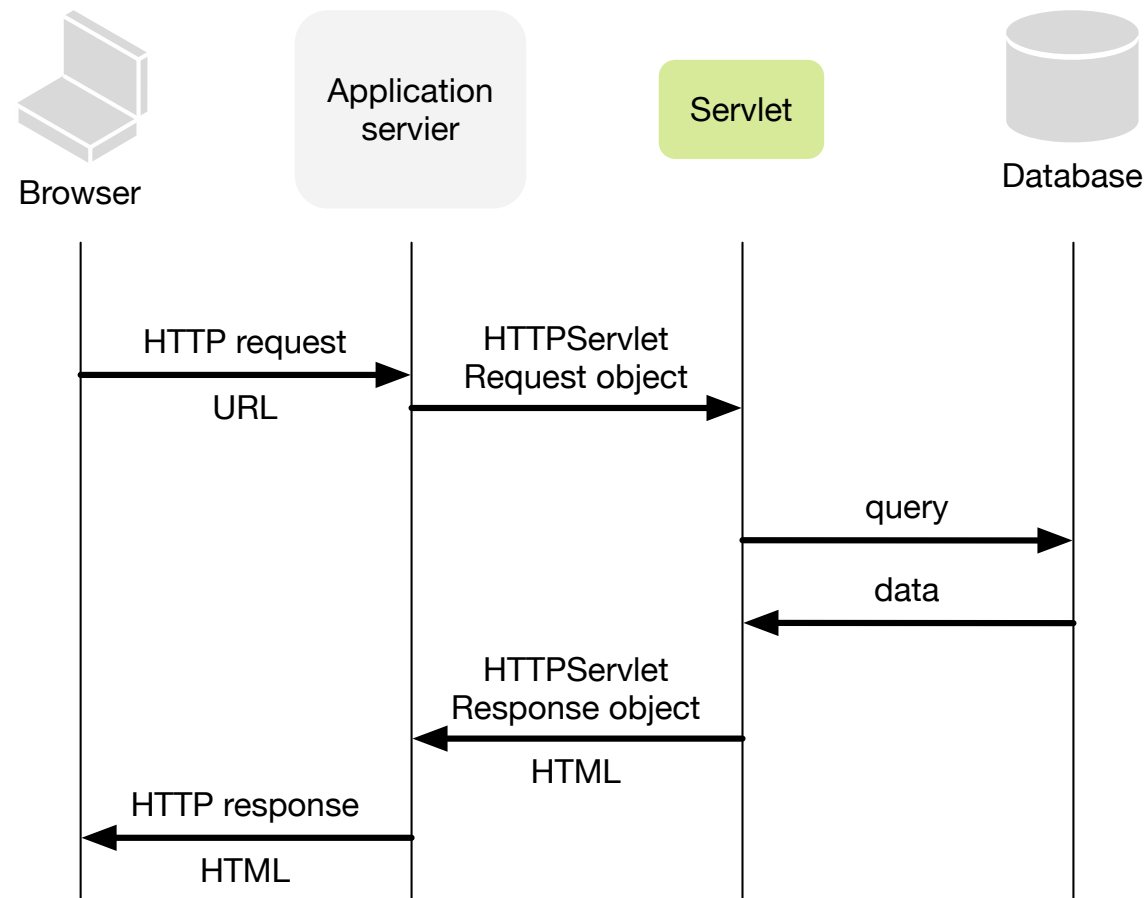
- A *servlet* runs inside an application server
 - The server calls the *servlet* to handle an HTTP request
 - The task of the *servlet* is to create the response



Platform as a Service

Reminder: Web applications in Java — *Servlets*

- Complete picture of the processing of an HTTP request



Platform as a Service

Reminder: Web applications in Java — *Servlets*

- Example implementation of a *servlet*

```
package com.example;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.*;

@SuppressWarnings("serial")
public class SampleServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        resp.setContentType("text/plain");
        PrintWriter pw = resp.getWriter();
        pw.println("Hello, world");
    }
}
```

Google App Engine

Important properties

- Automatic scaling
 - Able to support a large number of users.
 - Resources are added automatically.
 - The application does not need to specify in advance how much resources it will need.
- Google App Engine is a cloud service:
 - On-demand self-service
 - Pay as you go
 - Only pay for the resources *used*, not the resources *reserved*.
- Conceptually App Engine can be divided into three pieces:
 - Application instances
 - Data storage services
 - Scalable complementary services

Google App Engine

Standard Environment vs. Flexible Environment

- In the process of overhauling App Engine's Runtime Management component Google introduced in 2016 a new offering complementing the existing Standard Environment.

■ Standard Environment



- Deploys the application in containers in a sandboxed environment

- Limited choice of languages
- Pricing based on instance hours
- Simple to use but with limitations

■ Flexible Environment (General Availability 2017-03)



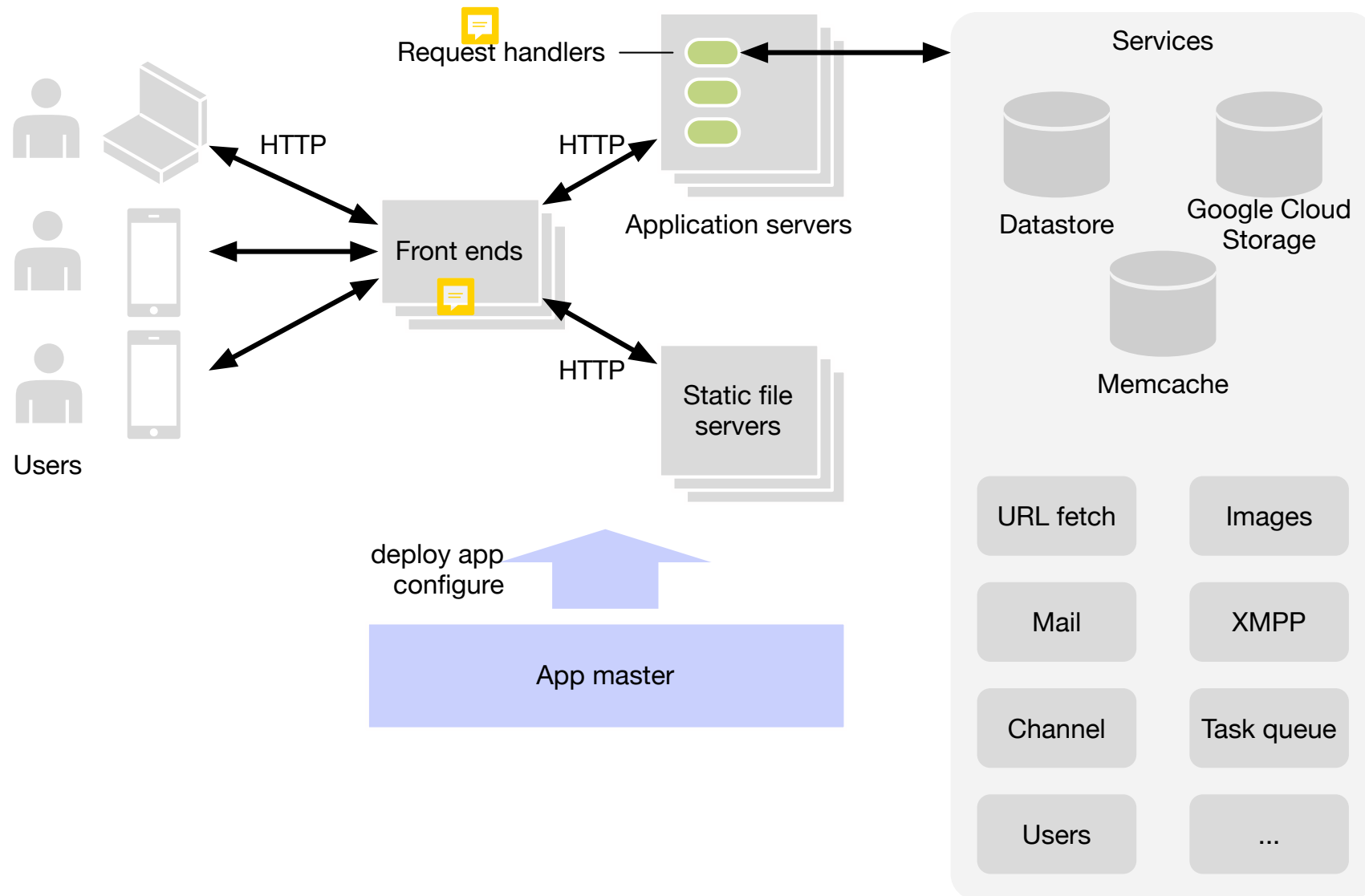
- Deploys the application in standard Google Compute Engine Virtual Machines
- Wider choice of languages
- Pricing based on vCPU, memory and persistent disks
- More flexibility but not as simple

- In this lecture we focus on the Standard Environment

	Standard environment	Flexible environment
Python	Python 2.7	Python 2.7, 3.6
Java	Java 7, 8	Java 8
PHP	PHP 5.5	PHP 5.6, 7
Go	Go 1.6	Go 1.8
Node	-	Node.js
Ruby	-	Ruby
.NET	-	.NET
more	-	Custom runtimes


Google App Engine

Architecture




Google App Engine

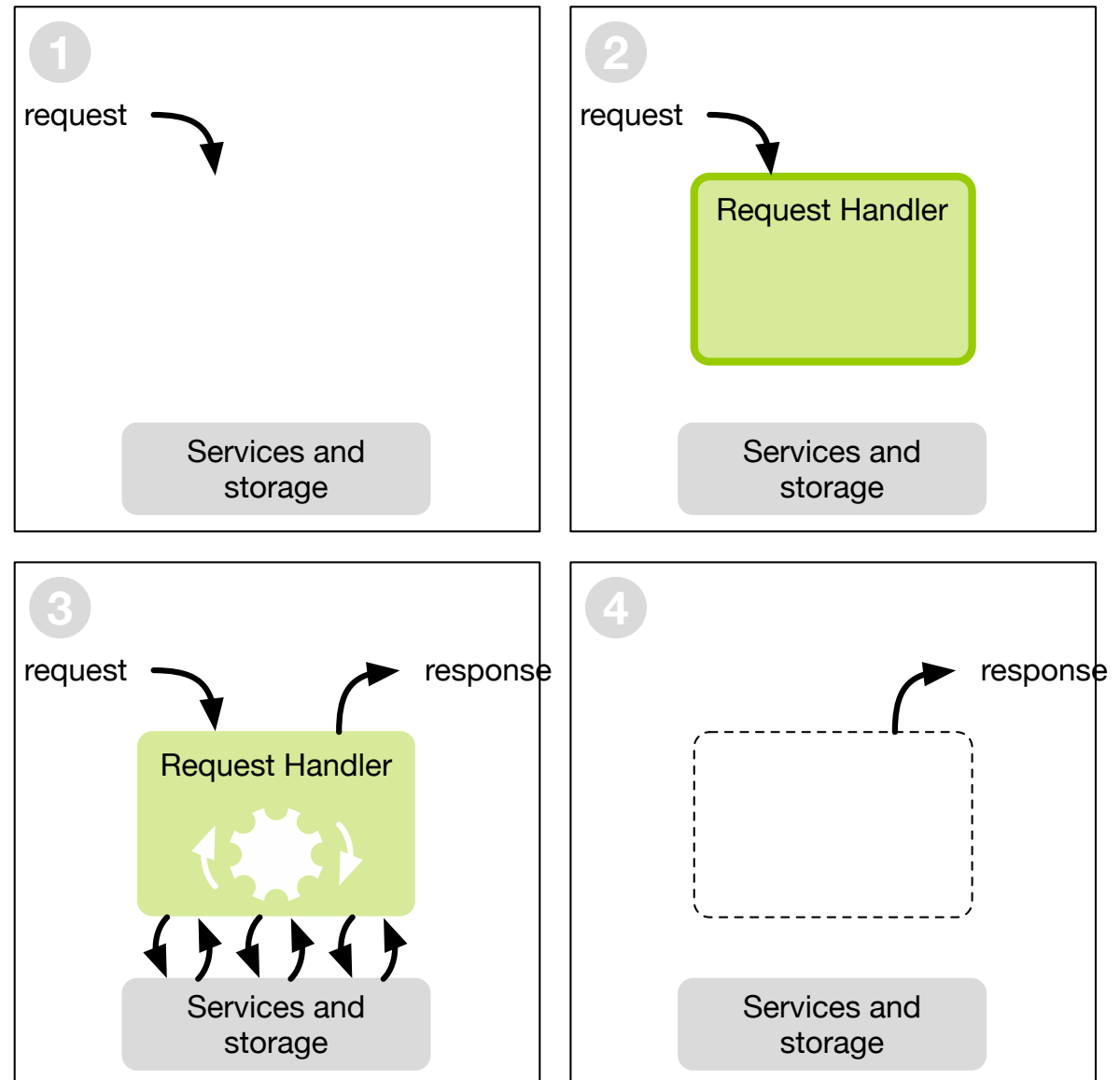
Programming model — Java

- The client/developer writes servlets.
 - App Engine implements the Java Servlet Specification.
- The platform offers data storage services.
 - Database accessible via a direct API or an ORM like Java Data Objects (JDO) or Java Persistence API (JPA).
 - File/Object Store
- The developer can use web application frameworks.
 - Java Server Pages (JSP)
 - Java Server Faces (JSF)
 - Google Web Toolkit (GWT)
 - Spring MVC
 - Play
 - ...
- The platform offers other APIs for
 - Asynchronous tasks
 - Caching
 - Notifications
 - Image processing
 - ...
- In principle the developer can use other languages that compile to Java Virtual Machine byte code, like
 - Scala
 - Ruby (JRuby)
 - Kotlin
 - ...

Google App Engine

Request Handler abstraction

- A *Request Handler* is an abstraction: it is a piece of code responsible for creating a response to an HTTP request.
 - In the case of Java it's a *Servlet*.
- A *Request Handler* has a life cycle:
 - (1) A request arrives.
 - (2) The *Request Handler* is created. It receives the request.
 - (3) The *Request Handler* creates the response. To do this it can use the App Engine services and the storage.
 - (4) When the response has been sent App Engine is free to remove the *Request Handler* from memory.
- **The *Request Handler* is stateless.** (except manual scaling) 

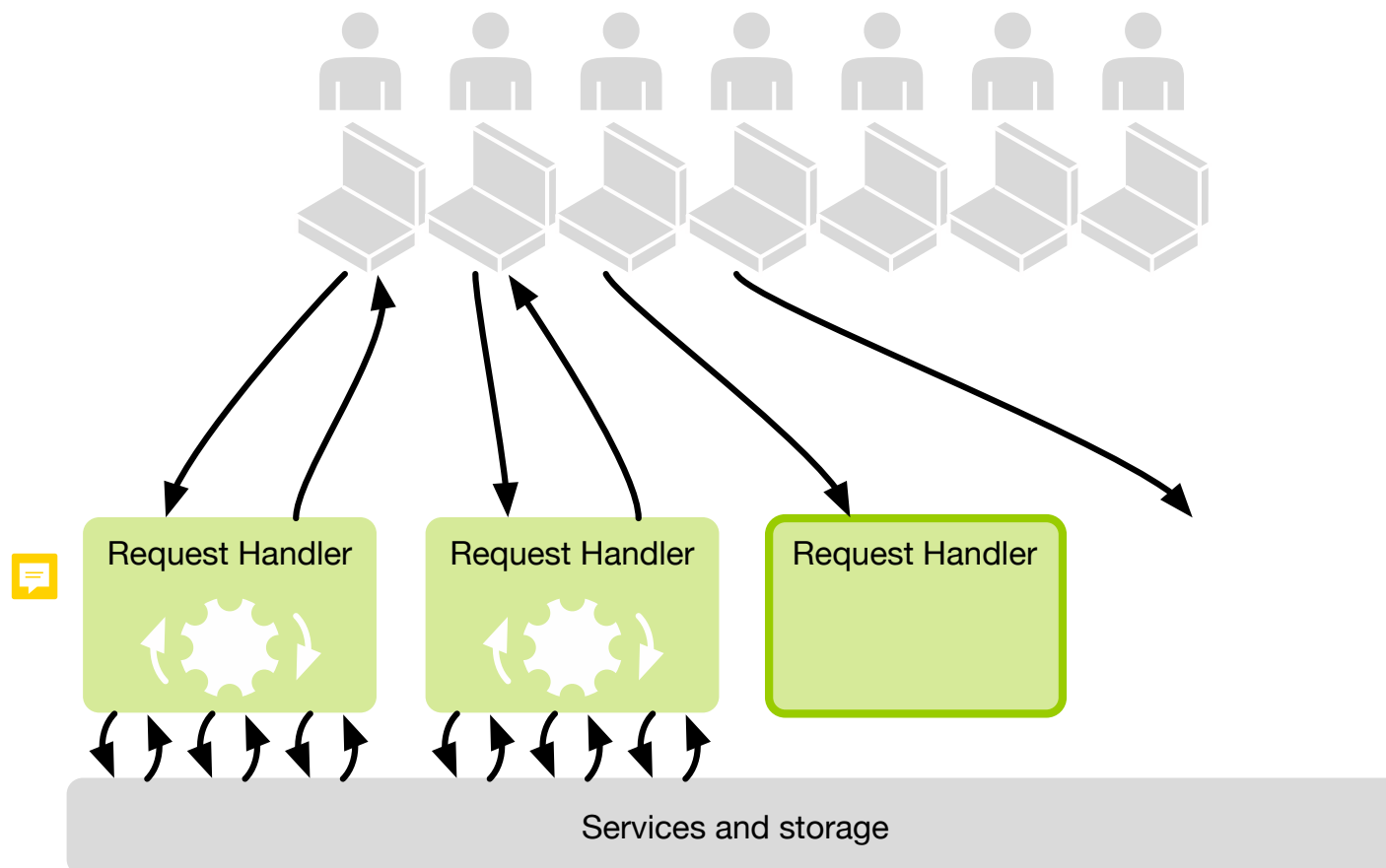


Source: Dan Sanderson, Programming Google App Engine, O'Reilly

Google App Engine

Request Handler abstraction

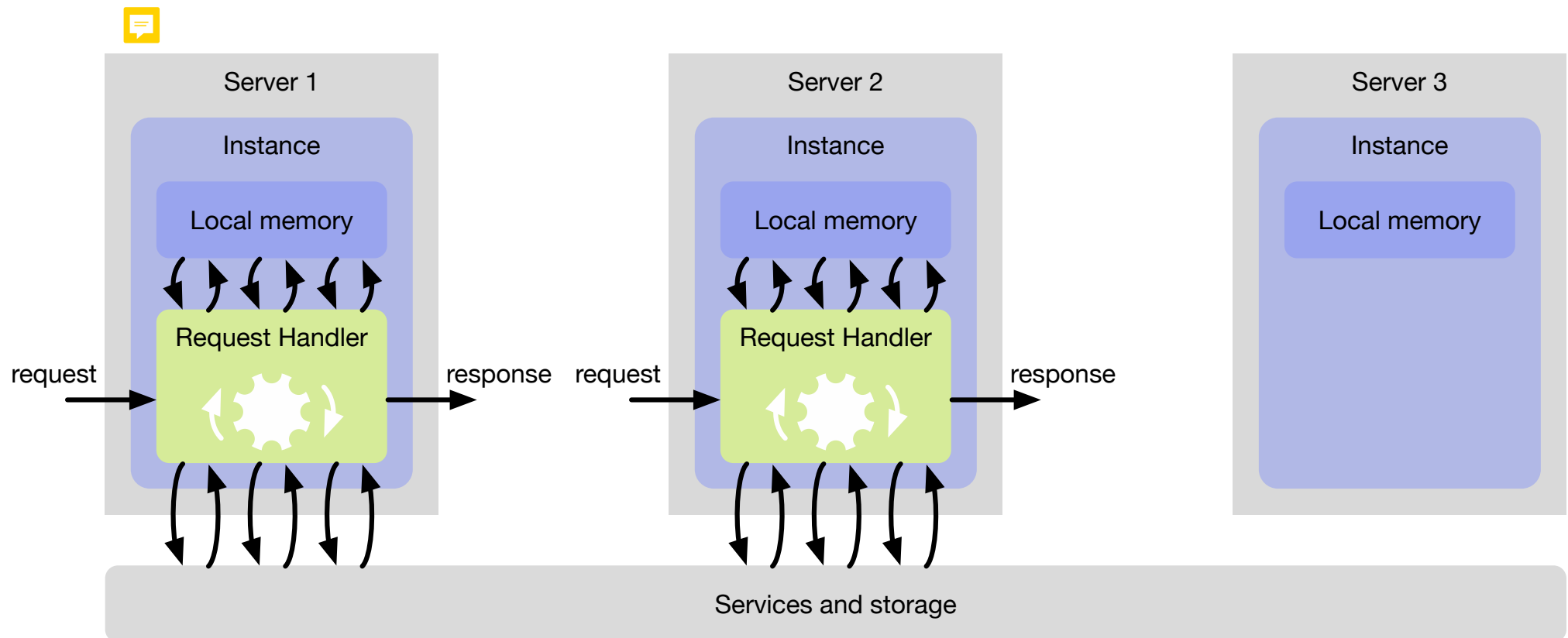
- When the number of requests increases App Engine allocates additional *Request Handlers*. All *Request Handlers* process the requests in parallel.



Google App Engine

Request Handler and *Instance* abstractions

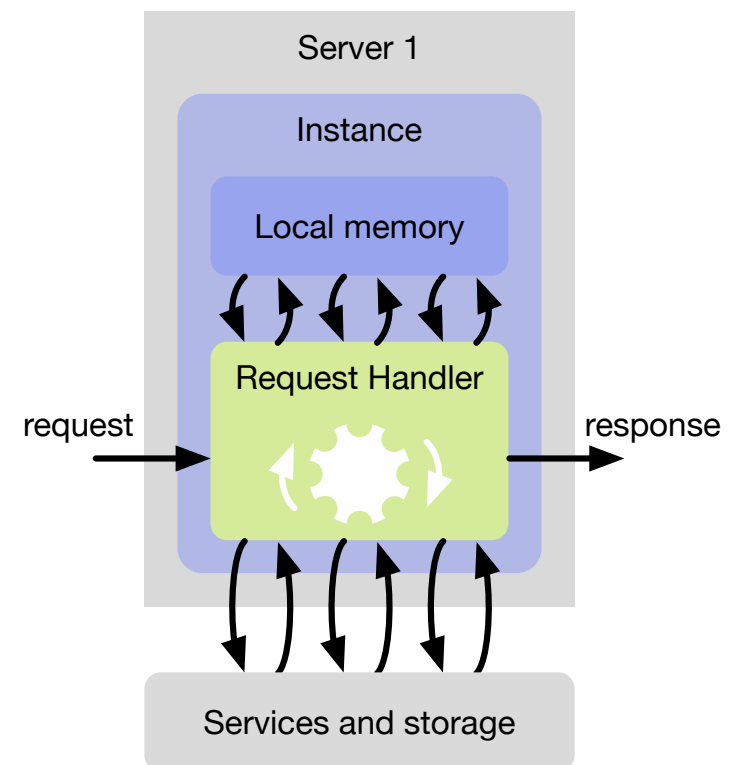
- Where does a *Request Handler* live? Inside an instance!



Google App Engine

Request Handler and *Instance* abstractions

- The abstraction of a *Request Handler* may be satisfying, but instantiating and destroying a program for each request would not make sense in practice.
 - The initialization of a program is expensive.
 - Especially the initialization of local memory.
- Instances are containers in which the *Request Handlers* live.
 - An instance keeps the initialized local memory that will be reused for subsequent requests.
- At any point in time an application has a certain number of instances allocated to process requests.
 - The front-end distributes the requests among the available instances.
- If necessary App Engine allocates new instances.
- If an instance is not used during a certain time interval (idle), App Engine frees the instance.



Google App Engine

Request Handler and Instance abstractions

- Instances are visible in the console

Instances ?									
	QPS last minute	Latency last minute	Requests	Errors	Memory	Start Time	Logs	Availability	
✓	1.112	0 ms	0	0	81 MB	Just now	View	Dynamic	Shutdown
✓	0.78	0 ms	0	0	64.8 MB	Just now	View	Dynamic	Shutdown
✓	0.988	0 ms	0	0	56.5 MB	Just now	View	Dynamic	Shutdown

- Active instances are billed to the client by instance hours.
- Depending on the scaling type the developer is able to change the parameters of the instance allocation algorithm in the console.

Performance

Frontend Instance Class: F1 (600MHz, 128MB)

Adjusting your Frontend Instance Class will affect all frontend versions of your application. Your frontends will have more memory and processing power, but also consume frontend instance hours at an increased rate, which will lead to increased costs. For example an F2 consumes instance hours at twice the rate of an F1. [Learn more.](#)

Max Idle Instances: (Automatic)

The Idle Instances slider allows you to control the number of idle instances available to the default version of your application at any given time. Idle Instances are pre-loaded with your application code, so when a new Instance is needed, it can serve traffic immediately. You will not be charged for idle instances over the specified maximum. A smaller number of idle Instances means your application costs less to run, but may encounter more startup latency during load spikes. [Learn more.](#)

Min 1 50 100 500 Automatic Max

Min Pending Latency: (Automatic)

The Pending Latency slider controls how long requests spend in the pending queue before being served by an Instance of the default version of your application. If the minimum pending latency is high App Engine will allow requests to wait rather than start new Instances to process them. This can reduce the number of instance hours your application uses, but can result in more user-visible latency. [Learn more.](#)

Min Automatic 500ms 1s 7.5s 15s Max

Save Settings

Google App Engine

Scaling

- When the developer uploads an application, he specifies the *scaling type* in a configuration file. The scaling type controls how instances are created.
- **Automatic Scaling** — With automatic scaling GAE decides when to create and delete instances using predictive algorithms based on request rate, response latencies, and other application metrics. The developer has only indirect control by tweaking certain parameters. The application has to be stateless. This is the default scaling type.
- **Manual Scaling** — With manual scaling instances run continuously, allowing the application to perform complex initialization and rely on the state of its memory over time (stateful). The developer specifies the number of instances to instantiate.
- **Basic Scaling** — With basic scaling GAE will create an instance when the application receives a request. The instance will be turned down when the app becomes idle. The application has to be stateless. The developer has direct control by specifying two parameters: the maximum number of instances and the idle timeout to delete instances.

Google App Engine

Tuning of Automatic Scaling

- With fluctuating traffic load scaling always has to make a tradeoff between happy users (low response times) and the cost of running instances. When traffic increases or decreases the scaling algorithm has also to decide how quickly to create or release instances.
- With **Automatic Scaling** the developer can do some limited tuning of the scaling algorithm (in the application's deployment descriptor):
 - `<max-concurrent-requests>` The number of concurrent requests an automatic scaling instance can accept before the scheduler spawns a new instance (default: 8, maximum: 80).
 - `<max-idle-instances>` The maximum number of idle instances that App Engine should maintain. Can be set to "automatic".
 - `<max-pending-latency>` The maximum amount of time that App Engine should allow a request to wait in the pending queue before starting a new instance to handle it. Default: 30 ms.
 - `<min-pending-latency>` The minimum amount of time that App Engine should allow a request to wait in the pending queue before starting a new instance to handle it.
 - `<min-idle-instances>` The minimum number of idle instances that App Engine should maintain.

Google App Engine

Deployment units: Applications, Services, Versions, Instances

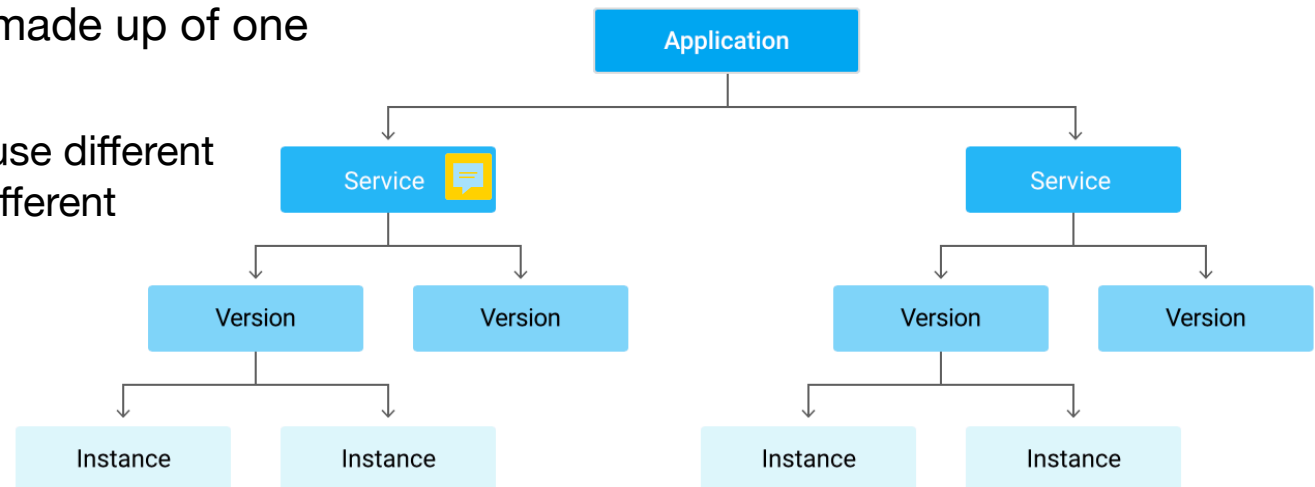
- Conceptually an Application is made up of one or more *Services*.

- Services can be configured to use different runtimes and to operate with different performance settings.
- Enables developer to factor a large application into logical components.
- A deployed Service behaves like a microservice.
- If the developer does not specify a Service explicitly, App Engine creates a *default* Service.

- Different Services can be rolled out in different *Versions*.

- Independent life-cycles

- GAE runs a particular Service Version in one or more *Instances*.



Google App Engine

Instance classes

- Each application has an *instance class* which determines its compute resources and pricing.
- F instances are used in automatic scaling, B instances are used in manual and basic scaling.

Instance class	Memory limit	CPU limit	Cost per hour per instance
B1	128 MB	600 MHz	\$0.05
B2	256 MB	1.2 GHz	\$0.10
B4	512 MB	2.4 GHz	\$0.20
B4_1G	1024 MB	2.4 GHz	\$0.30
B8	1024 MB	4.8 GHz	\$0.40
F1	128 MB	600 MHz	\$0.05
F2	256 MB	1.2 GHz	\$0.10
F4	512 MB	2.4 GHz	\$0.20
F4_1G	1024 MB	2.4 GHz	\$0.30

Google App Engine

Isolation of different applications: the *sandbox* 

- App Engine cannot work with *any* Java application.
 - App Engine needs to distribute HTTP requests to several servers. A server may run several applications.
 - Applications written by different App Engine clients must not interfere between one another.
- Compared to a generic Java application, GAE applications run in a more restricted environment: the *sandbox*.
 - For Java 7 Google modified the Java runtime environment and removed some classes and methods. The allowed classes and methods were listed in the *JRE Whitelist*.
 - App cannot write to the filesystem.
 - App cannot open a socket or access another host directly.
 - Make system calls.
 - For Java 8 these restrictions do no longer apply.
 - Additionally the web server runtime is wrapped in custom-built Google containers.




Google App Engine

Data storage services

- App Engine offers mainly three services for data storage:
 - **Google Cloud Datastore:** Persistent data storage in a NoSQL database
 - **Google Cloud SQL:** Persistent data storage in a relational database that is MySQL-compatible
 - **Google Cloud Storage:** Persistent object (= file) storage
- Additionally the application is able to use caching of database requests by using **Memcache**, an in-memory data caching service

Google App Engine

Data storage services

	Google Cloud Datastore	Google Cloud SQL	Google Cloud Storage
Data model	NoSQL	SQL	File/object
Tenancy	Multi-tenant 	Single-tenant 	Multi-tenant
Scheduled downtime	No	Yes	No
Pricing — Data transfer	Pay per GB transferred in or out	Pay per GB transferred out (in is free) 	Pay per GB transferred in or out
Pricing — Data storage	Pay per GB stored per month	Pay per instance running (instance minutes) and per GB capacity reserved per month	Pay per GB stored per month

Google App Engine

The *Datastore*

- The **Datastore** is a data storage **service** (Datastore-as-a-Service)
 - It is not a database dedicated to one client.
 - It is a gigantic scalable database that contains the data from **all** clients.
 - **Multi-tenant** model
 - Runs on a cluster of servers (distributed architecture). Behind the scenes Google manages the distribution and replication of data and load balancing of database queries automatically.
 - **No scheduled downtime** with 99.95% uptime SLA
 - The application calls simply an API,
 - without caring where the data is stored;
 - without caring about the capacity of the database: virtually unlimited.
- Datastore has a NoSQL data model different from the relational (SQL) model.
 - No need to declare a schema before writing data
 - Aggregate-oriented data model
- Transaction guarantees are different from SQL databases.
 - Relaxed consistency guarantees

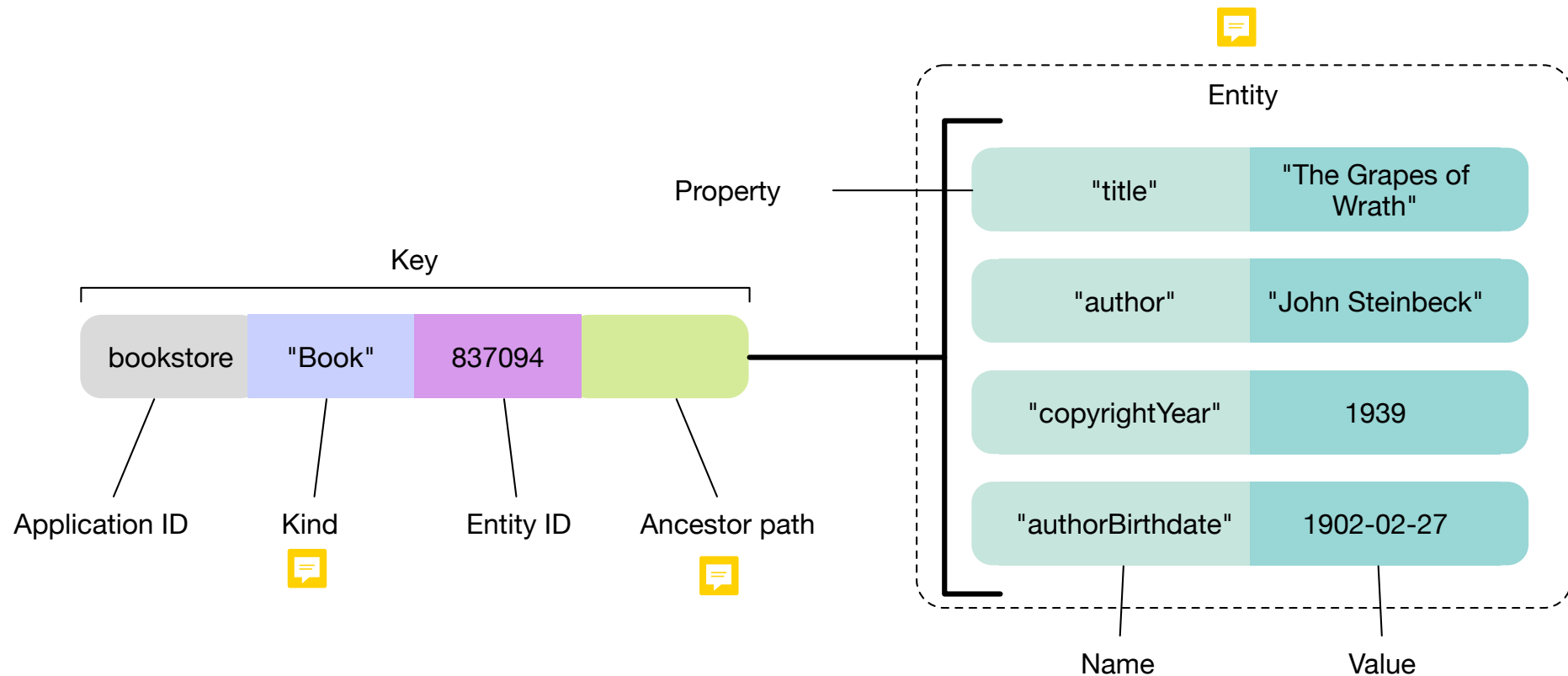
Google App Engine

The *Datastore* — Data model

- The data model of the Datastore is organized along three major concepts:
 - **Entity:** An object in the Datastore containing several properties. Each object has a unique key.
 - **Kind:** The "type" of an entity, for example an e-commerce application would have the kinds *client*, *article*, *order*.
 - Used in queries to find the stored entities.
 - Does not imply the presence of properties.
 - **Property:** A property contains one piece of data of an entity, for example customer name, article price.
 - An entity may have several properties.
 - Each property has a name and at least one value.

Google App Engine

The *Datastore* — Data example



Google App Engine

The *Datastore* — Entity key

- The keys of the entities in the datastore have to be unique — even across App Engine clients.
- A key is composed of several parts
 - The **application identifier**
 - Ensures that there are never collisions with keys from other applications.
 - Is automatically managed by App Engine. Cannot be modified by the application.
 - The entity **kind**
 - Contributes to the uniqueness of the key.
 - Is used by the Datastore in queries
 - The **entity identifier**. This part may be ...
 - specified by the application. In this case this part is called *key name*.
 - automatically generated by the Datastore. In this case this part is called *ID*.
 - (Optional) The **ancestor path**: a path composed of the entity's ancestors
 - Allows to locate the entity in a hierarchical structure.
- **NB:** When the key of an entity has been created it can no longer be changed!

Google App Engine

The *Datastore* — Corresponding terms

- Approximative correspondence of terms used in object-oriented programming, relational databases and the Datastore. Be careful, the concepts are not equivalent!



Object-oriented	Relational	Datastore
Class	Table	Kind
Object	Row	Entity
Attribute	Column	Property

Google App Engine

The *Datastore* — Low-level API

```
// ...
import com.google.appengine.api.datastore.DatastoreService;
import com.google.appengine.api.datastore.DatastoreServiceFactory;
import com.google.appengine.api.datastore.Entity;

// ...
DatastoreService ds = DatastoreServiceFactory.getDatastoreService();

Entity book = new Entity("Book");

book.setProperty("title", "The Grapes of Wrath");
book.setProperty("author", "John Steinbeck");
book.setProperty("copyrightYear", 1939);
Date authorBirthdate =
    new GregorianCalendar(1902, Calendar.FEBRUARY, 27).getTime();
book.setProperty("authorBirthdate", authorBirthdate);

ds.put(book);

// ...
```



Google App Engine

The *Datastore* — High-level API — Java Persistence API (JPA)

A class that we want to store
in the Datastore:

```
import java.util.Date;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity(name = "Book")
public class Book {
    @Id
    private String isbn;

    private String title;
    private String author;
    private int    copyrightYear;
    private Date   authorBirthdate;

    // ... constructors, accessors
}
```

The code that will store this class:

```
import myapp.Book; // our data class

// ...

EntityManager em = null;
try {
    em = emf.createEntityManager();
    Book book = new Book();
    book.setTitle("The Grapes of Wrath");
    // ...
    em.persist(book);
} finally {
    if (em != null)
        em.close();
}
```

Google App Engine

Quotas



- Like any other cloud service, Google bills clients for the resources they use,
 - But with *automatic scaling* the allocation of resources is automatic as a function of user demand.
 - The client does not have direct control.
 - With resource allocation that is principally unpredictable costs would be unpredictable as well.
- Google adopted a billing model that is similar to prepaid contracts for mobile networks.
 - The client fixes a budget at the beginning.
 - When the budget is exhausted the service stops functioning.
- Quota model:
 - At the beginning of each day (midnight Pacific time) maxima are fixed for the usage of resources during the next 24 hours: the quotas.
 - If usage remains below the quota, everything is fine.
 - When usage exceeds the quota, the service is suspended. Service resumes the next day.
 - Users may receive HTTP error 500.
 - A call to a service by the application may raise an exception.
- There are also per-minute quotas to limit explosive resource usage.

Google App Engine

Free usage

- Google offers a free usage of App Engine.
 - In this case the daily quotas are the ones listed in the left column
 - Notably 28 instance hours, 1 GB traffic in and out and 5 GB storage
- By paying a client is able to increase the quotas (middle column).
- NB: The free usage is not limited in time, so it is possible to host small projects indefinitely for free.

	FREE LIMIT PER DAY	PRICE ABOVE FREE LIMIT (PER UNIT)	PRICE UNIT
Instances	28 instance hours	\$0.05	Instance per hour
Cloud Datastore (NoSQL) - Operations	50k reads, 20k write, 20k deletes	\$0.06 reads, \$0.18 writes, \$0.02 deletes	Per 100k entities for each category
Cloud Datastore (NoSQL) - Storage	1 GB	\$0.18	GB / month
Network Traffic (Outgoing)	1 GB	\$0.12	GB
Network Traffic (Incoming)	1 GB	FREE	-
Cloud Storage	5 GB	\$0.026	GB / month
Memcache - Dedicated Pool	Free Usage of Shared Pool, no free quota for Dedicated Pool	\$0.06 (Free usage of Shared Pool)	GB / hour
Search - Searches	1000 basic operations, 100 searches	\$ 0.50	10k searches
Search - Indexing Documents	0.01 GB	\$2.00	GB
Email API	100 recipients	Contact Sales	-
Logs API	100 MB	\$0.12	GB
Task Queue	5 GB	\$0.026	GB / month
SSL Virtual IPs	-	\$39.00	Virtual IP / month

Google App Engine

Service Level Agreement

*[...] the Covered Service will provide a **Monthly Uptime Percentage** to Customer of at least **99.95%** (the "Service Level Objective" or "SLO"). If Google does not meet the SLO, and if Customer meets its obligations under this SLA, Customer will be eligible to receive the Financial Credits described below. [...]*

Monthly Uptime Percentage	Percentage of monthly bill for Covered Service which does not meet SLO that will be credited to future monthly bills of Customer
99.00% – < 99.95%	10%
95.00% – < 99.00%	25%
< 95.00%	50%

- **"Monthly Uptime Percentage"** means total number of minutes in a month, minus the number of minutes of **Downtime** suffered from all **Downtime Periods** in a month, divided by the total number of minutes in a month.
- **"Downtime"** means more than a ten percent Error Rate.
- **"Downtime Period"** means, for an Application, a period of five consecutive minutes of Downtime. Intermittent Downtime for a period of less than five minutes will not be counted towards any Downtime Periods.
- **Customer Must Request Financial Credit:** In order to receive any of the Financial Credits described above, Customer must notify Google technical support within thirty days

Google App Engine

Services summary

- **Datastore:** The Datastore is a schemaless object datastore, with a query engine and atomic transactions.
- **Blobstore API:** The Blobstore API allows your app to serve data objects, called blobs, that are much larger than the size allowed for objects in the Datastore service.
- **Capabilities API:** With the Capabilities API, your application can detect outages and scheduled downtime for specific API capabilities.
- **Channel API:** The Channel API creates a persistent connection between your application and Google servers, allowing your application to send messages to JavaScript clients in real time without the use of polling.
- **Images API:** The Images API can resize, rotate, flip, and crop images. It can also enhance photographs using a predefined algorithm.
- **Mail API:** The Mail API sends email messages on behalf of the app's administrators, and on behalf of users with Google Accounts. Apps can receive email at various addresses.
- **Memcache:** Memcache is a distributed in-memory data cache to be used in front of or in place of robust persistent storage for some tasks.
- **Namespace:** The Namespaces feature allows you to implement multitenancy in your applications.
- **OAuth:** OAuth is a protocol that allows a user to grant a third party limited permission to access a web application on her behalf, without sharing her credentials (username and password) with the third party.
- **Task Queue API:** The Task Queue API allows applications to perform work initiated by a user request, outside of that request.
- **URL Fetch API:** The URL Fetch API allows applications to fetch resources and communicate with other hosts over the Internet using HTTP and HTTPS requests.
- **Users API:** The Users API allows applications to authenticate users who have Google Accounts or accounts on your own Google Apps domain.
- **XMPP API:** The XMPP API allows applications to send and receive instant messages to and from users of XMPP-compatible instant message services.

Google App Engine

Logging

- For debugging and monitoring an application developers are encouraged to use the `java.util.Logger` library.
 - Log messages are saved by App Engine.
 - The developer can look them up in the console,
 - or download them to the local machine.
- The standard output (`System.out`) and error (`System.err`) streams are also captured and saved by App Engine.
 - Appear in the logs as INFO / WARNING messages.

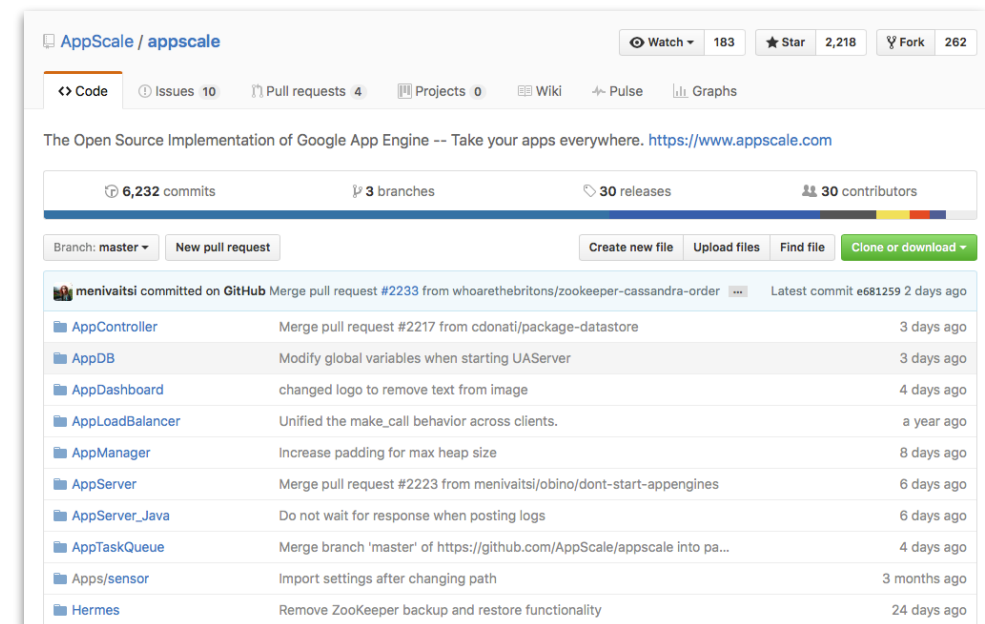
AppScale

An Open Source clone of Google App Engine



- AppScale is an open-source cloud computing platform that automatically deploys and scales unmodified Google App Engine applications over public and private cloud systems and on-premise clusters.
- AppScale is supported and maintained by AppScale Systems, in conjunction with Google.
- Started in 2009 as a research project at U Santa Barbara, 2012 foundation of commercial company
- Available under Apache license v. 2.0 and BSD license
- <https://github.com/AppScale/appscale>

APPSCALE SYSTEMS



Snapchat

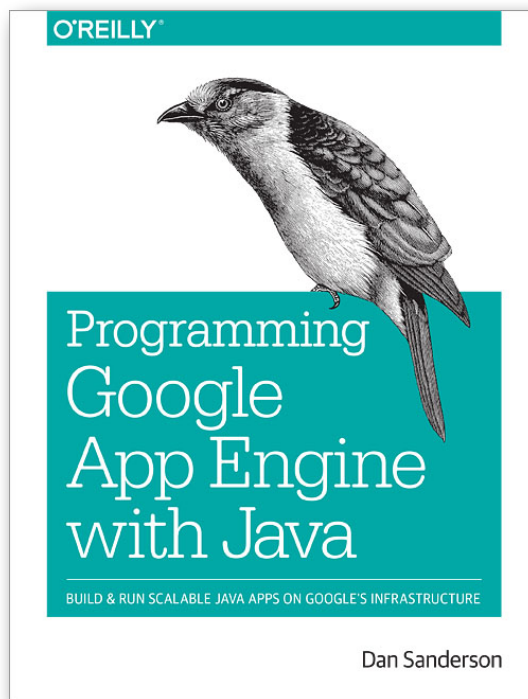
A Google App Engine case study

- 2011 At U Stanford three students work on a class project to design a photo sharing app. New idea: the photos disappear after being viewed
- 2011-09 First public release of Snapchat, built on Google App Engine
- 2012-01 3'000 monthly users
- 2012-05 30'000 monthly users, 1 million snaps a day
- 2012-11 20 million snaps a day
- 2012-12 Introduction of videos
- 2013-09 Snapchat rejects Facebook offer for \$3 billion (six employees)
- 2014 400 million snaps a day
- 2015-05 2 billion videos a day
- 2015-11 6 billion videos a day
- 2017-03 Snapchat goes public
 - \$28 billion valuation, 2'000 employees
 - Snapchat is Google App Engine's biggest customer, spending at least \$400 million a year



Google App Engine

References



Dan Sanderson
Programming Google App Engine with Java

2015-06
O'Reilly Media



Dan Sanderson
Programming Google App Engine with Python

2015-06
O'Reilly Media



Mohsin Shafique Hijazee
Mastering Google App Engine

2015-10
Packt Publishing