Mobile Computing and Internet of Things
Web Services & Cloud Computing

Pelle Jakovits

Jakovits@ut.ee
Announcements

(Homework 3 due end of week - Friday)

• Try to communicate issues early

• This Wednesday – Mobile Assignment
  – 16 pts – work in pairs
  – Exact details will be published to courses website

• Next week we start working with IoT
  – We’ll hand out IoT hardware kits during lectures and labs.
  – Due to limited no. of kits you need to share it with a co-student!
  – IoT assignments will also be solved in pairs
Outline

• Web Services
  – SOAP, REST, OpenAPI

• Cloud Computing
  – Cloud computing models
  – NoSQL and Cloud Storage services

• Cloud services for Mobile applications
Web services

• Software which provides access to resources over the internet using standardized web protocols (HTTP, HTTPS)

  “Loosely coupled, standard-based reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols” [Sleeper, 2001]

• **Google Translate** – submit text to be translated
• **Message board** – submit message to be posted on a „board“
• **shrinkpictures.com** – submit images to be resized
Web service characteristics

• **Interoperability** of heterogeneous services
  – Platform neutrality
  – Programming language independence

• **WSDL** - Web Services Description Language
  – Standard for describing web services, resources and supported operations

• **UDDI** - Universal Description, Discovery and Integration
  – XML-based standard for describing, publishing and discovering web services
HyperText Transfer Protocol

- Application layer protocol for the World Wide Web
- Follows the Request-Response model
- Servers serve a set of resources (web page, image, etc.)
- Clients request an operation to be performed on resources
- HTTP operations:
  - GET – Download a resource
  - POST – Update/modify a resource
  - PUT – Create a new (sub) resource
  - DELETE – delete a resource
  - HEAD, TRACE, OPTIONS, CONNECT, PATCH
SOAP - Simple Object Access Protocol

• Messaging protocol (1998) for exchanging structured data
• Platform and language independent
• Messages are formatted as XML documents
  – Issues with the size of messages
  – Not well suitable for binary data
  – Parsing XML is somewhat slow
• Relies on HTTP
  – but can be used with other protocols
**RESTful - Representational state transfer**

- A set of rules (2000) how to use the **HTTP** (or other) protocol for communicating and exchanging data with web services
- Provides a simplistic and uniform interface for web services
- URI identifies a resource
  - E.g. [http://api.example.com/users/693](http://api.example.com/users/693)
- Specifies which operations are allowed on which resources
  - POST, PUT, GET, DELETE, PATCH
- **Statelessness** – No session information stored in server
  - Session on client side, Data in a database
- **Cacheability** - responses must be defined as cacheable or non-cacheable
SOAP vs REST: Query resource value

Listing 2.1: SOAP-based request example

GET / HTTP/1.1
Host: www.example.com
Content-Type: application/soap+xml; charset=UTF-8
Content-Length: {length}

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <!-- Header information here -->
  </env:Header>
  <env:Body>
    <m:getProductDetail xmlns:m="http://www.example.com/">
      <productID>21</productID>
    </m:getProductDetail>
  </env:Body>
</env:Envelope>

Listing 2.2: REST-based request example

GET /product/21
Host: www.example.com
Content-Type: application/x-www-form-urlencoded

Chang, C. (2013), Service-Oriented Mobile Social Network in Proximity, PhD Dissertation, Monash University, Australia.
SOAP vs REST

- **SOAP**: A set of user defined operations described in a standard manner
- **REST**: A set of standard operations applied on a set of custom resources and services
OpenAPI

• A specification for describing, consuming and visualizing RESTful web services
• Originally known as the Swagger Specification
• Users create a web service template in the OpenAPI specification:
  – Resources, their endpoints and descriptions
  – Supported HTTP operations
  – Allowed inputs and expected outputs
openapi: 3.0.0
info:
  title: Sample API
  version: 0.1.9
servers:
  - url: http://api.example.com/v1
    description: Server description, e.g. Main (production)
paths:
  /users:
    get:
      summary: Returns a list of users.
      responses:
        '200':  # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
OpenAPI generator

• Can generate both client and server code from the OpenAPI specification
  – Clients: 30+ languages
  – Servers: 15 languages

• Generated server code includes:
  – full server side (e.g. Flask),
  – Docker file to deploy it,
  – Swagger web interface for live API documentation and testing

• User only needs to implement the content of the REST methods (PUT, GET, POST, DELETE) and additional features (e.g. DB connection, Auth, integrations)
Generating API server side

Ada, C# (ASP.NET Core, NancyFx), C++ (Pistache, Restbed, Qt5 QHTTPEngine),
Erlang, F# (Giraffe), Go (net/http, Gin, Echo), Haskell (Servant, Yesod),
Java (MSF4J, Spring, Undertow, JAX-RS: CDI, CXF, Inflector, Jersey, RestEasy, Play Framework, PKMST, Vert.x),
Kotlin (Spring Boot, Ktor, Vertx),
PHP (Laravel, Lumen, Mezzio (fka Zend Expressive), Slim, Silex, Symfony),
Python (FastAPI, Flask),
NodeJS,
Ruby (Sinatra, Rails5),
Rust (rust-server),
Scala (Akka, Finch, Lagom, Play, Scalatra)
Generating API client side

**ActionScript, Ada, Apex, Bash, C, C#** (.net 2.0, 3.5 or later, .NET Standard 1.3 - 2.0, .NET Core 2.0, .NET 5.0. Libraries: RestSharp, HttpClient),

**C++** (Arduino, cpp-restsdk, Qt5, Tizen, Unreal Engine 4),

**Clojure, Crystal, Dart, Elixir, Elm, Eiffel, Erlang, Go, Groovy, Haskell**

**Java** (Apache HttpClient, Jersey1.x, Jersey2.x, OkHttp, Retrofit1.x, Retrofit2.x, Feign, RestTemplate, RESTEasy, Vertx, Google API Client Library for Java, Rest-assured, Spring 5 Web Client, MicroProfile Rest Client),

**k6, Kotlin, Lua, Nim, Node.js/JavaScript** (ES5, ES6, AngularJS with Google Closure Compiler annotations, Flow types, Apollo GraphQL DataStore),

**Objective-C, OCaml, Perl, PHP, PowerShell, Python, R, Ruby, Rust**

**Scala** (akka, http4s, scalaz, sttp, swagger-async-httpclient), **Swift**

**Typescript** (AngularJS, Angular (2.x - 11.x), Aurelia, Axios, Fetch, Inversify, jQuery, Nestjs, Node, redux-query, Rxjs)
### OpenAPI Generator Online 5.0.0-beta2

This is an online openapi generator server. You can find out more at [https://github.com/OpenAPITools/openapi-generator](https://github.com/OpenAPITools/openapi-generator).

**API SERVER:**

- [x] http://api.openapi-generator.tech/

**CLIENTS**

<table>
<thead>
<tr>
<th>Method</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/api/gen/clients</td>
<td>Gets languages supported by the client generator</td>
</tr>
<tr>
<td>GET</td>
<td>/api/gen/clients/{language}</td>
<td>Returns options for a client library</td>
</tr>
<tr>
<td>POST</td>
<td>/api/gen/clients/{language}</td>
<td>Generates a client library</td>
</tr>
<tr>
<td>GET</td>
<td>/api/gen/download/{fileId}</td>
<td>Downloads a pre-generated file</td>
</tr>
</tbody>
</table>

**SERVERS**

<table>
<thead>
<tr>
<th>Method</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/api/gen/servers</td>
<td>Gets languages supported by the server generator</td>
</tr>
<tr>
<td>GET</td>
<td>/api/gen/servers/{framework}</td>
<td>Returns options for a server framework</td>
</tr>
<tr>
<td>POST</td>
<td>/api/gen/servers/{framework}</td>
<td>Generates a server library</td>
</tr>
</tbody>
</table>
Cloud Computing

• Computing as a utility
  – Utility services e.g. water, electricity, gas etc
  – Consumers pay based on their usage

• Cloud Computing characteristics
  – Illusion of infinite resources
  – No up-front cost, Fine-grained billing

• Perfect for hosting web services which need to be accessed by a large number of users across the world

Gartner: “Cloud computing is a style of computing where massively scalable IT-related capabilities are provided ‘as a service’ across the Internet to multiple external customers”

Everything as a Service (XaaS)
Cloud Models

http://nolegendhere.blogspot.com.ee/2012/06/presentation-4-5-7.html
Infrastructure as a Service

- Provides access to Virtual Machines on-demand, in real-time
- To deploy applications in IaaS, need to choose and set up:
  - Virtual machines
  - Software environment
- User is responsible for:
  - System administration, backups
  - Monitoring, log analysis
  - Managing software updates
  - Stability & scalability of the software environment
Platform as a Service - PaaS

- Complete platform for hosting applications in Cloud
- Underlying infrastructure & software is managed for you
- Businesses can deploy web-based applications on-demand
- Eliminates the complexity of selecting, purchasing, configuring, and managing hardware and software
- Dramatically decreases upfront costs
- Built-in scalability of deployed software
- Integrated with cloud services and databases
- **Google App Engine**, AWS BeanStalk
Google App Engine

- PaaS for developing and hosting web applications in Google-managed data centers
- Easy to build, maintain, and scale applications
- No servers to maintain or configure by yourself
- Upload & Go (Git push and go)
- Supported languages: Python, Java, PHP, Go, ...
- Integration with all other Google Cloud services and APIs
- Persistent storage with queries, sorting, and transactions
- App Engine distributes user requests across multiple servers and scales servers to meet dynamic traffic demands
- Application runs within its own secure, sandboxed and reliable environment
App Engine Environment
Integration with Cloud services
PaaS Advantages

- Do not have to manage low level resources and services
- Many services ready to use in a plug-in fashion without any configuration or setup
- Provider handles most of the non-functional requirements of your applications
- Scaling is automatically managed by the platform
- Easier and more agile application deployment
  - Simplifies prototyping and launching software startup apps/services
- Platform provider has the best knowledge to optimize the services running on the underlying hardware
Function as a Service (FaaS)

- Often also referred to as **Serverless**
- Each deployed "application" is a single Function
- Functions are independent from each other
  - Scaled, managed and billed separately
  - Can be written in different languages

**Event driven execution** based on triggers and preconditions:
- **Trigger Event:** New image uploaded to S3
- **Precondition:** File size is larger than 10 MB
- **Execute:** `Resize_image(filePath)`

- Functions can be composed into larger applications
- Only pay per function execution time, not for idle runtime!
Current FaaS platforms

- Apache OpenWhisk
- AWS Lambda
- OpenLambda
- Azure Functions
- fission
- Red-Hat
- Google Functions
- Kubernetes
FaaS in Public Clouds

• **AWS Lambda**
  – Run code in AWS without managing infrastructure or software
  – Java, Go, PowerShell, Node.js, C#, Python, and Ruby code
  – Pricing is based on number of **requests** and **GB-Sec "Memory-Duration"**
    – Free: 1M **requests** a month. After: $0.20 per 1M
    – Free: 400,000 **GB-Sec.** After: $0.000017 per 1 **GB-Sec**

• **IBM BlueMix Cloud Function**
  – Based on OpenWhisk - Open Source Serverless cloud platform
  – Event, trigger & rule based execution
  – Supports any language*
    – Free: 400,000 **GB-Sec.** After: $0.000017 per 1 **GB-Sec**
FaaS Example

- We have a message board
- We want users to be able to write messages, which are entered into databases and displayed to users
- We create the server side as FaaS functions (e.g. IBM cloud)
- Each REST endpoint and method is an individual FaaS function
Function to add message into DB

```python
import sys
from cloudant.client import Cloudant

def addDocToDB(new_doc, username, apikey):
    databaseName = "labdb1"
    client = Cloudant.iam(username, apikey, connect=True)
    myDatabase = client[databaseName]
    return myDatabase.create_document(new_doc)

def main(request):
    new_doc = {"message": message, "user": user}
    modified_doc = addDocToDB(new_doc, request["username"], request["apikey"])

    return {
        "headers": {
            'Content-Type': 'text/html'
        },
        "statusCode": 201,
        "body": '<html><body><h3>Message added to the database</h3></body></html>'
    }
```
from cloudant.client import Cloudant

def addDocToDB(new_doc, username, apikey):
    databaseName = "labdb1"
    client = Cloudant.iam(username, apikey, connect=True)
    myDatabase = client[databaseName]
    return myDatabase.create_document(new_doc)

def main(input):
    new_doc = { 'message': input["message"], 'user': input["user"]}

    modified_doc = addDocToDB(new_doc, input["username"], input["apikey"])

    return {
        "headers": {
            'Content-Type': 'text/html'
        },
        "statusCode": 201,
        "body": '<html><body><h3>Message added! </h3></body></html>'
    }
Accessing function over REST

Web Action

- **Enable as Web Action**: Allow your Cloud Functions actions to handle HTTP events. Web Actions allow to control the response data and type by using a set of URL extensions, such as .json or .html. Learn more about [Web Actions](#). **Note**: The Web Action URL below requires to return a dict object that contains a body property.

- **Raw HTTP handling**: When enabled your Action receives requests in plain text instead of a JSON body.

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Auth</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANY</td>
<td>Public</td>
<td><a href="https://eu-gb.functions.appdomain.cloud/api/v1/web/jakovits%40ut.ee_dev/default/pelleform">https://eu-gb.functions.appdomain.cloud/api/v1/web/jakovits%40ut.ee_dev/default/pelleform</a></td>
</tr>
</tbody>
</table>

REST API

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Auth</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>API-KEY</td>
<td><a href="https://eu-gb.functions.cloud.ibm.com/api/v1/namespaces/jakovits%40ut.ee_dev/actions/pelleform">https://eu-gb.functions.cloud.ibm.com/api/v1/namespaces/jakovits%40ut.ee_dev/actions/pelleform</a></td>
</tr>
</tbody>
</table>
import sys

def main(dict):
    form = ''
    <html>
        <body>
            <form id="form_1" method="post" action="https://eu-gb.functions.appspot.com/v1/web/jakovits%40ut.ee_dev/default/pelleform">
                User: <input id="user" name="user" size="30" type="text" value=""/> <br />
                Message: <textarea rows="10" cols="30" id="message" name="message" type="text" value=""></textarea>
                <input id="saveForm" type="submit" name="submit" value="Send message" />
            </form>
        </body>
    </html>
    ''

    return {
        "headers": {
            'Content-Type': 'text/html'
        },
        "statusCode": 201,
        "body": form
    }
Extensions

• Function 2 also prings out latest X messages
• Add a function 3, which is triggered when a new message is added to database
  – For example:
    • Sends a notification a new message has been written
    • Compute word and character count of the message
    • etc.
Event based FaaS execution

• FaaS functions are not running in the background
• Functions are executed when they are triggered
• Example events for triggering FaaS functions:
  – REST event (GET, PUT, DELETE) against function endpoint
  – New image uploaded to cloud storage
  – New record added to client database
  – Record modified in the database
    • Temperature value higher than 100°C! -> Send email
  – New message arrived in message queue
Advantages of Serverless/FaaS

• Very simple and "cheap" to scale
• Rapid prototyping
• Easy to modify serverless functions
• Pay only for the execution time, not for idle computation time
• Can create applications by composing functions written in different languages
Cloud Model complexity

- **Own Hardware**
  - Applications
  - Data
  - Runtime
  - Middleware
  - Operating System
  - Virtualization
  - Servers
  - Storage
  - Networking

- **IaaS**
  - Applications
  - Data
  - Runtime
  - Middleware
  - Operating System
  - Virtualization
  - Servers
  - Storage
  - Networking

- **PaaS**
  - Applications
  - Data
  - Runtime
  - Middleware
  - Operating System
  - Virtualization
  - Servers
  - Storage
  - Networking

- **SaaS**
  - Applications
  - Data
  - Runtime
  - Middleware
  - Operating System
  - Virtualization
  - Servers
  - Storage
  - Networking

Cloud provider managed
Cloud Data storage services

• Provide a Managed storage services
  – STtorage as a Service (STaaS)
• Provider takes care of installing, deploying, configuring, scaling, sharding, backups, etc.
• Different pricing options depending on object/file access frequency, expected latency, duration of storage
• Types:
  1. Bucket/Blob Storage
  2. Managed SQL storage services
  3. Managed NoSQL storage services
  4. Managed File System services
Object / Bucket / Blob Storage

- Follow the Key-value non-relational model
- Storing large amount of unstructured data
  - No schemas
  - Images, Videos, Log files, backup files, etc.
- Amazon S3, Azure Blob storage, Google Cloud Storage, IBM Cloud Object Storage
- May have different modes for storage:
  - Real-time vs Low-frequency vs Archived
AWS S3

• Amazon Simple Storage Service (Amazon S3)
  • Divides data into buckets
    – A logical unit of storage - unique web folder
    – Bucket can contain virtually unlimited folders and files
    – 100 buckets per user
  • Storage classes
    – S3 Standard
    – Intelligent-Tiering (Automatic storage class selection, min 30 days)
    – Standard-Infrequent Access (Less frequent, but rapid access, min 30 days)
    – One Zone-Infrequent Access (Single availability Zone, min 30 days)
    – S3 Glacier (data archiving, access in minutes to hours, min 90 days)
    – S3 Glacier Deep Archive (access in 12 hours, min 180 days)
Managed Cloud SQL storage

• Two main types:
  – Simple SQL server on demand
  – Fully managed, sharded SQL cluster on demand

• Amazon (RDS) - service for dynamically deploying and scaling SQL servers (Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and SQL Server)

• IBM Db2 – IBM own engine from 83.
  – SQL, BigSQL (Hadoop), Data Warehouse, Analytics

• Google Cloud SQL - Managed MySQL, PostgreSQL, or SQL Server
Managed Cloud NoSQL storage

• Non-Relational database models:
  – Key-Value
    • AWS DynamoDB, Google Cloud Datastore
  – Document DB
    • AWS DocumentDB, IBM Cloudant (CouchDB), Google Cloud Firestore
  – Column oriented
    • AWS Managed Apache Cassandra Service, Google BigQuery
  – Graph DB
    • AWS Neptune, IBM Graph
The Key-value Model

- Data stored as key-value pairs
- The value is an opaque blob to the database
- Examples: Dynamo, Riak, Apache Ignite, ArangoDB, Berkeley DB, Couchbase
- Horizontal scalability
  - Data with the same Key stored close to each other
  - Suitable for cloud computing
- Flexible schema-less models suitable for unstructured data
- Queries: Get, Put and Delete (REST)
- Fetching data by key can be very fast
Key-Value Model: Key design

- AWS S3 "keys":
  - https://s3.Region.amazonaws.com/bucket-name/KeyName

- Keys can be complex:
  - employee:1:firstName = "Martin"
  - employee:2:firstName = "John"
  - payment:1:1:amount = "10000"
  - payment:1:1:date = "01/12/2019"
  - payment:2:1:amount = "5000"
  - payment:2:1:date = "01/12/2019"
The Document-oriented Model

• Data is also stored as key-value pairs
  – Value is a „document“ and has further structure
• No strict schema
  – Expectation is that documents contains also the schema
  – Not enforced in any way
• Query data based on document structure
• Examples: CouchDB, MongoDB, ArangoDB, BaseX, Clusterpoint, Couchbase
Example JSON document

```json
{
    "name": "Asheville Veedub Volkswagon Repair & Restoration",
    "address": "851 Charlotte Hwy",
    "attributes": {
        "BusinessAcceptsCreditCards": "True",
        "ByAppointmentOnly": "False",
        "GoodForKids": "True"
    },
    "business_id": "0KwutFa520HgPLWtFv02EQ",
    "categories": "Auto Repair, Automotive",
    "city": "Fairview",
    "is_open": 1,
    "latitude": 35.5431561,
    "longitude": -82.4419299,
    "neighborhood": "",
    "postal_code": "28730",
    "review_count": 7,
    "stars": 4.5,
    "state": "NC"
}
```

- Aggregates are described in JSON using map and array data structures
The Column Family Model

• Data stored in large sparse tabular structures
• Columns are grouped into column families
  – Column family is a meaningful group of columns
  – Similar concept as a table in relational database
• A record can be thought of as a two-level map
  – Column Family -> Column -> Value
• New columns can be added at any time
• Examples: BigTable, Cassandra, HBase, Accumulo
## Column Family Example

### Static column families

<table>
<thead>
<tr>
<th>_id</th>
<th>names</th>
<th>contacts</th>
<th>messages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>username</td>
<td>firstname</td>
<td>lastname</td>
</tr>
<tr>
<td>a001</td>
<td>jsmith01</td>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>b014</td>
<td>pauljones</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Dynamic column family

- **a001**
  - **Names**
    - username: jsmith
    - firstname: John
    - lastname: Smith
  - **Contacts**
    - phone: 5 550 001
    - email: jsmith@example.com
  - **Messages**
    - item1: Message1
    - item2: Message2
    - ... (multiple entries)
    - itemN: MessageN

- **b014**
  - **Names**
    - username: pauljones
  - **Contacts**
  - **Messages**
    - item1: new Message
Why use Relational databases?

• Highly available and highly consistent
  – Old, tried and tested technologies
  – Well optimized for performance
• Strict schema means less mistakes
  – Easier to control the quality of data
  – Easier to avoid missing data, wrong data types, etc.
• Very good performance as long as data fits into a single machine
When use Non-Relational?

• When volume of data grows too large
  – Easier and cheaper to scale
• When the schema is not fixed
  – Store a query unstructured data
  – Prototyping: Build applications without pre-defining schema
  – Updating schemas is very simple
• When using structured documents (JSON, XML)
  – Similar schema inside documents, but can vary greatly
  – Nested sub-structures
• Many open source options
Firebase

• A collection of managed Google cloud services designed specifically for Mobile Applications
  – Provides services for Authentication, Realtime Database, Performance Monitoring, Cloud Storage, Analytics, AdWords etc.
  – https://firebase.google.com/docs/

• Firebase ~= Google Cloud
  – Can convert Firebase project into Google Cloud project or vice versa

• Compared to Vanilla GC, Firebase provides:
  – Additional SDKs for mobile clients
  – Additional integration with the Firebase CLI
  – Means to configure security rules to control access to data through SDKs
Firebase Cloud Messaging

- Push Notification as a cloud service
- Goal is to avoid polling in mobile applications

- Free service that allows you to send data from your servers to your apps/users
  - FCM replaced Google Cloud Messaging (GCM)
  - GCM is now deprecated
Types of messages

• Notification messages
  – Handled by the FCM SDK automatically
  – Have a predefined set of user-visible keys
  – Optional data payload of custom key-value pairs
  – Background apps receive the notification payload in the notification tray
    • Only handles the data payload when the user taps on the notification
  – Foreground app receives a message object with both payloads available

• Data messages
  – Handled by the client app
  – Data messages have only custom key-value pairs
Working with FCM - direct message

1. Request for Registration
2. Reply with Registration Token
5. Send Message

Mobile App

3. Send Registration Token

4. Send message {RegId, msg}

https://firebase.google.com/docs/cloud-messaging/
Subscribe to topics

1. Subscribe to a topic
2. Confirmation
4. Send Message

Mobile App

FCM

3. Send message {topic, msg}

https://firebase.google.com/docs/cloud-messaging/
Firebase Test Lab

- Cloud-based app testing infrastructure
- Test your app on various devices and configuration
  - Both Android and iOS
- Can be run on real devices deployed in Google data center
- Can integrate with deployment workflows
- Robo test - analyzes the structure of your app's UI and then explores its simulating user activities

Physical devices

- Nexus 7 (2013) ASUS
  - 19
  - 21
- Nexus 9 HTC
  - 21
- LG G4 LG
  - 22
- Nexus 5 LG
  - 19
  - 21
  - 22
  - 23
- HTC One (M8) HTC
  - 19
- LG G3 LG
  - 19
- Nexus 4 LG
  - 19
  - 22
- Moto E Motorola
  - 19
Firebase Cloud Messaging (FCM)

• A free cross-platform messaging solution
  – Can send messages to Android, iOS, etc.

• Handles queuing of messages and delivery to apps
  – Application does not need to be running in the background
  – Can wake up application when new message arrives

• FCM also supports sending messages to multiple phones, to devices subscribed to topics etc.

• FCM also supports sending messages to the application server from the mobile

• Uses Extensible Messaging and Presence Protocol (XMPP)
Conclusions

• Web Services enable standardized way for applications and services to communicate

• The rise of Cloud Computing has brought Everything as a Service:
  – Infrastructure, Platforms, Functions, Storage, Messaging, ...

• Specialised cloud service bundles for Mobile applications
  – Designed to provide all background cloud services required by typical Mobile Apps

• Push notification as a service allows for asynchronous communication from cloud to apps
  – Even if Apps are sleeping or turned off

• Testing mobile apps in Cloud can also be done on real devices