Fakulteta za informacijske študije
Faculty of information studies

Data Processing in The Cloud – JS Approach

1259. sredin seminar / Ljubljana 20. jan. 2016

Uroš Mesojedec

Faculty of Information Studies in Novo mesto

uros.mesojedec@fis.unm.si

uros@mesojedec.net

Introduction

- Cloud computing is next logical step to utility computing
- „Cloudification“
  - Existing processes
    - Adapt / Rewrite
  - New processes
    - Best practices, future proofing
- Cloud Rank
  - Set of tests and measurements for cloudification potential of process
  - (including existing software)

Cloud

- Two rules of thumb:
  - If you have to buy new hardware
  - If you are dependent on the client
  - … then this is not real cloud
- Cloud solution:
  - Use and integrate existing services for consumption on any device

Cloud == Web platform

- IPv4 → IPv6
- HTTP → HTTPS
- (X)HTML → HTML5
- Server → Services
- Client → Devices („Things“)
- Network → Utility

Web as Development Platform

- Web solved cross-platform
- Web should be the platform developers invest first / most
- … because it is the best (easiest) way to great solutions
- Ecosystem
  - Cloud Services (AWS, GCP, Azure…)
  - Rich clients (evergreen browsers, smart mobile devices…)
  - Expanding environment (Internet of Things)
- Developers
  - Server code
  - Client code

JavaScript (JS) is everywhere

- It is underlying base for the Web (platform)

- Some JS manifestations
  - important part of HTML5
  - JSON
  - Libraries ecosystem (maybe the biggest!)
  - Databases (MongoDB)
  - Server (Node.js)
  - Platform (asm.js)

Rule of Least Power

- The **rule of least power** in programming:
  - Design principle that suggests choosing the least powerful [computer] language suitable for a given purpose.

- Alternatively:
  - Given a choice among computer languages, the less procedural, more descriptive the **language** one chooses, the more one can do with the **data** stored in that language.

source: http://www.w3.org/2001/tag/doc/leastPower.html

- Atwood‘s „Law“, A corollary to the Principle of Least Power:
  - Any application that *can* be written in JavaScript, *will* eventually be written in JavaScript.

Jeff Atwood, 2007: http://blog.codinghorror.com/

JavaScript

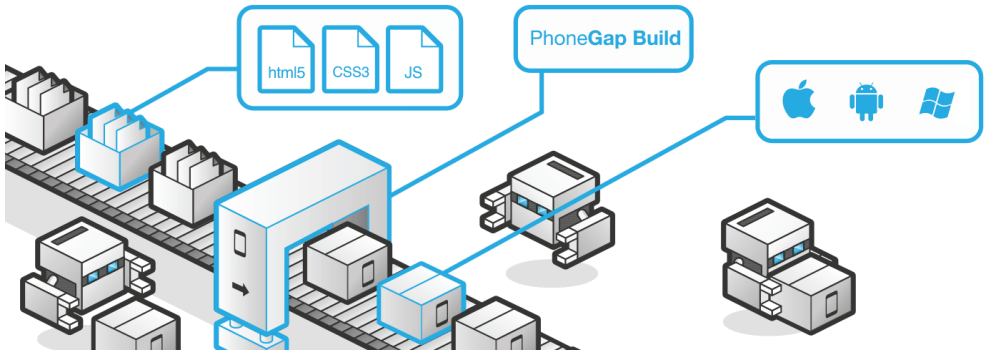
- JavaScript (JS)
  - Prototype-based with first-class functions
  - Multi-paradigm language, liberal at programming styles (object-oriented || imperative || functional)
- JS !== Java.
  - Unrelated with very different semantics
  - Some naming, syntactic, and standard library similarities (marketing-driven at the time)
  - JS syntax derived from C, semantics and design influenced by Self and Scheme
- JS is the only language for web browsers
- Speed of JS engines → JS is a feasible compilation target

JavaScript Milestones

- 1995 – Brendan Eich (Netscape) – Navigator LiveScript, rebranded "**JavaScript**"
- 1996 – Microsoft connects **Server** (IIS) and Client (IE) with "JScript"
- 1997 – **ECMAScript** standard
- 1998 – Microsoft XMLHTTP for Outlook Web Access → **AJAX** is born
- 2001 – Douglas Crockford (State Software) – **JSON**
- 2006 – John Resig – **jQuery**
- 2008 – Perf. breakthrough – Google V8, Apple Nitro, Mozilla TraceMonkey
- 2009 – Ryan Dahl (Joyent) – Google V8 + event loop + I/O → **Node.js** is born
- 2011 – Nitobi PhoneGap, Adobe acquisition → **Apache Cordova** with cloud compiler
- 2012 – Microsoft **TypeScript**, later adopted by Google's **Angular 2**
- 2013 – Jordan Walke (Facebook) – **React**
- 2013 – Mozilla **Asm.js**
- 2015 – Brendan Eich announces **WebAssembly**

Quick intermission – Apache Cordova

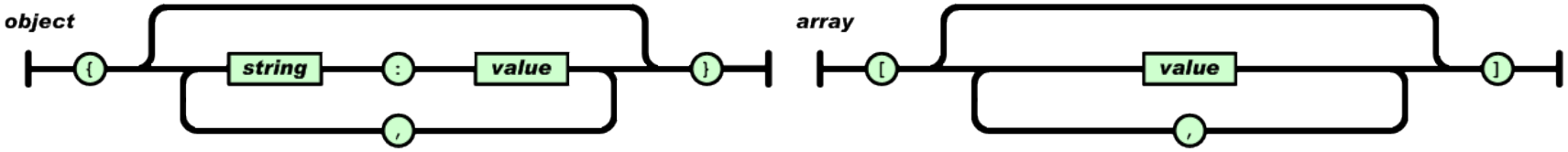
- Cloud compiler
  - PhoneGap Build
    https://build.phonegap.com/
    - Compiler as a service (no investment in devices and dev kits)
    - JS as completely viable cross-platform mobile development language



source: https://build.phonegap.com/

JSON (JavaScript Object Notation)

- Object serialization
- Lightweight, human readable, key-value pairs
- Standard
  - Douglas Crockford, 2006: RFC 7159 and ECMA-404
- Identical to JS objects
  - Can be evaluated (without parsing) to JS structures



source: http://www.json.org/

JSON Example

- Wine & Cheese Network (source: https://gist.github.com/maxkfranz/)

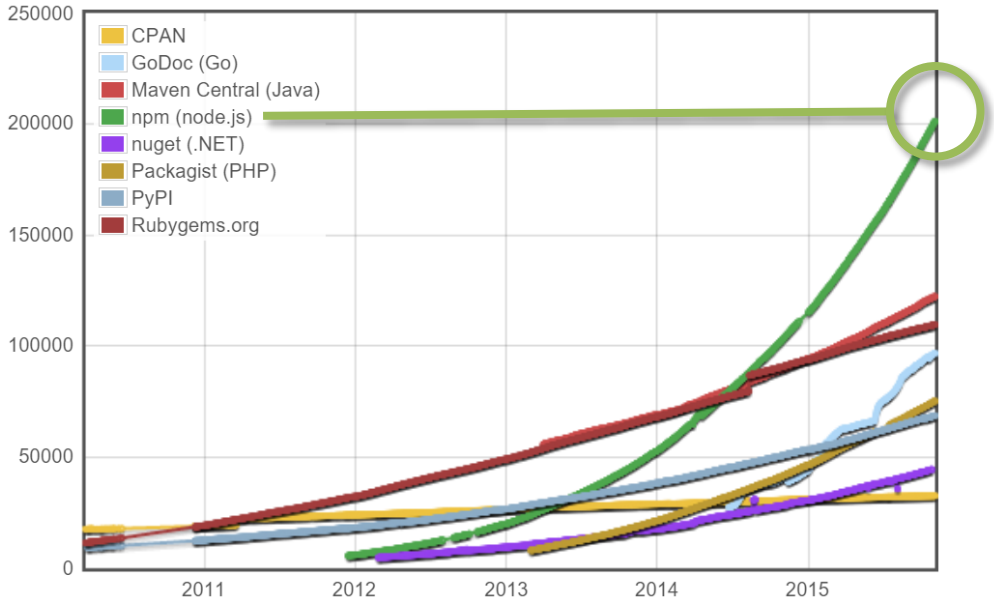
```
{
  "format_version": "1.0",
  "generated_by": "cytoscape-3.2.0",
  "data": {
    "name": "WineCheeseNetwork"
  },
  "elements": {

    "nodes": [
       {
        "data": {
          "id": "430",
          "name": "Aarauer Bierdeckel",
          "Strength": 5,
          "selected": false,
          "Milk": "Raw cow's milk",
          "Synonym": "Kuentener",
          "Quality": 90,
          "Type": "Semi-soft",
          "NodeType": "Cheese",
          "Country": "Switzerland"
        }
      },
      {
        "data": { ... }
      },
      { ... }
    ],

    "edges": [
       {
        "data": {
          "id": "1763",
          "source": "430",
          "target": "429",
          "SUID": 1763,
          "name": "Aarauer Bierdeckel (cc) Bergues",
          "interaction": "cc"
       },
       {
        "data": { ... }
       },
       { ... }
    ]

  }

}
```

Vibrant Ecosystem (.js is the new .com)

- Vibrant ecosystem (just some examples)
  - MVC Frameworks
    - Backbone.js
    - Capuccino
    - Knockout
    - AngularJS
    - Ember.js
  - Client
    - Polymer
    - Bootstrap
    - React.js
    - EXT.js
    - D3.js
    - Cordova
  - Server, services
    - Node.js
    - MongoDB (NoSQL, JSON/BSON store)
    - Microsoft Azure Cloud (hosted MongoDB, Node.js…)
    - Google Cloud (JS automation, JSON APIs, JS client libs…)

Module Counts

http://www.modulecounts.com



OK, Great! But High Performance?

- JS is Turing complete
- We have high performance JS engines
- Why not compile other languages to JS?
- Use best optimizing subset of JS
  - Asm.js
    - It‘s still just JS
    - Can be compiled ahead of time and highly optimised
    - C/C++ (and other codebases) → Emscripten → JS (Asm.js subset)

- Show me the numbers!
  - Currently approx. 2× slower than optimised compiled C/C++

    source: F. Khan et al.: *Using JavaScript and WebCL for Numerical Computations: A Comparative Study of Native and Web Technologies* (Splash 2014, http://goo.gl/HFR3Q3)
  - Improvements are inevitable

- WebAssembly (announced 2015 by B. Eich)
  - Assembler for the Web
  - Industry cooperation

High Performance

- Asm.js and further optimisations
  - WebAssembly with wide industry support
- WebGL and WebCL
  - GPU visualisation & processing
- WebWorkers API
  - Simplified Map/Reduce (e.g. parallel.js)
- SIMD support coming (SIMD.js)
  - Emscripten will use SIMD.js

Some Impressive Demos

- High performance heatmaps (*WebGL*)
  - http://codeflow.org/entries/2013/feb/04/high-performance-js-heatmaps/
- Chrome Experiments
  - https://www.chromeexperiments.com/
  - http://david.li/vortexspheres/
- D3.js Demos (*visualisations*)
  - Directed Graph Editor: http://bl.ocks.org/rkirsling/5001347
  - Force-Directed Graph: http://bl.ocks.org/mbostock/4062045
  - Collatz Graph: https://www.jasondavies.com/collatz-graph/
- BananaBread (Asm.js *showcase*)
  - Hi-Perf 3D Game: https://developer.mozilla.org/ms/demos/detail/bananabread
- Map/Reduce & Parallel processing (*WebWorkers/child processes*)
  - Parallel.js https://adambom.github.io/parallel.js/
  - JS Map/Reduce: http://jcla1.com/blog/javascript-mapreduce/

Demo time

- Demonstration

Conclusion

- JS is (more than) a language for the cloud
- JS spans data models, frameworks, clients, servers, application automation…
- JS is high performance
- JS already has probably the richest ecosystem with exponential growth trend
- JS can coexist with existing systems
- JS is also (high performance) cloud platform