

Ein Blick über den Tellerrand: Was steckt hinter Node.js

Stefan Schuster, IRIAN Solutions GmbH

Über mich

- Stefan Schuster
- Bei IRIAN Solutions seit 2007
- Spezialgebiete:
 - JavaScript (CoffeeScript, TypeScript)
 - Client (Dojo)
 - Server (Node.js)
 - HTML5
 - CSS
 - Canvas
 - ...
- Beispiel: mind42.com (komplett in CoffeeScript)

Agenda

- Grundlagen
- Asynchrones Programmieren
- Ökosystem
- Beispiele

Grundlagen

JavaScript als Sprache

- Vorstellung: JavaScript == Webseite
- Man muss trennen
 - JavaScript als API

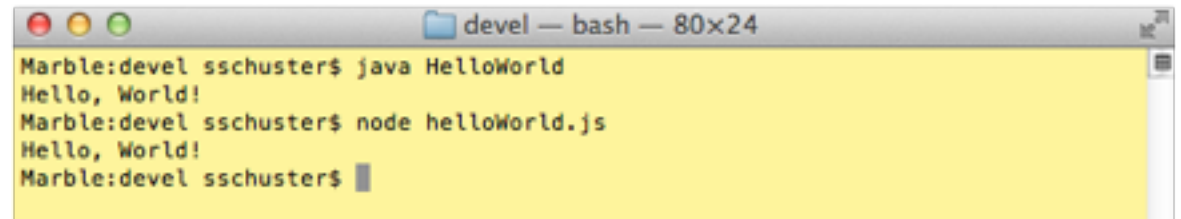
```
document.addEventListener("click", function() {
    console.log("click");
});
document.getElementById("main").style.display = "none";
document.title = "Hello, World!";
```

- JavaScript als Sprache

```
function Dog() {
    this.name = "Timo";
}
Mammal.prototype = new Mammal();
var dog = new Dog();
```

JavaScript am Server

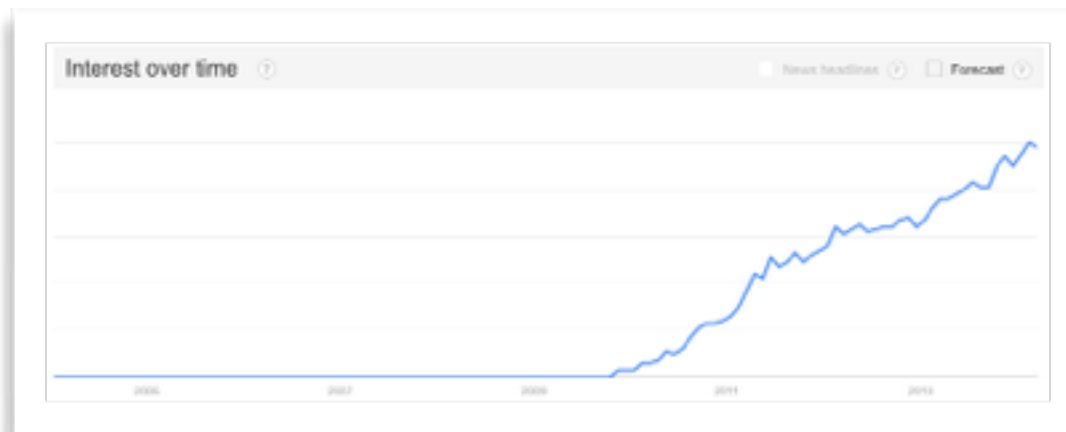
- Was bedeutet JavaScript am Server nun?
 - JavaScript Runtime Environment
 - Headless
 - Keine Browser APIs
 - Alles, was mit JS programmierbar ist, ist machbar
 - Spezielle APIs je nach Runtime Environment
 - Wie Java VM



```
devel — bash — 80x24
Marble:devel sschuster$ java HelloWorld
Hello, World!
Marble:devel sschuster$ node helloWorld.js
Hello, World!
Marble:devel sschuster$
```

Node.js

- Open Source JavaScript Runtime Environment
 - Gestartet in 2009
 - Basiert auf Google V8
 - 1.0 am Horizont
 - Bereits in Produktion bewährt
 - Stark wachsende Community



Node.js API

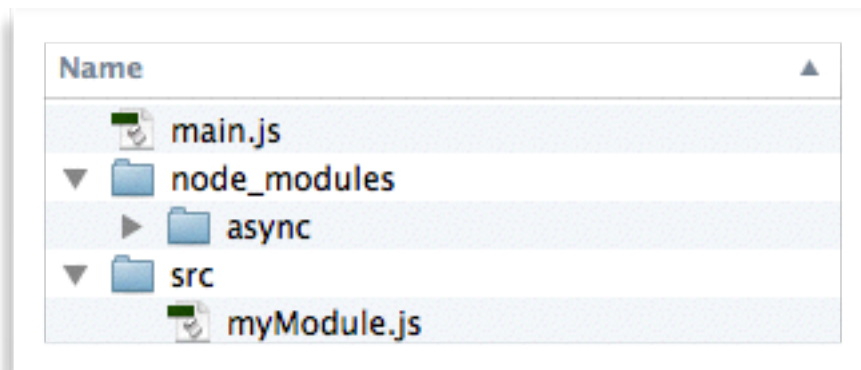
- Anstelle von DOM APIs, Server-relevante APIs
 - Dateisystem
 - Netzwerk
 - TCP/UDP
 - HTTP/HTTPS
 - DNS
 - Crypto
 - OS
 - Prozesse
 - Plattform

CommonJS

- Spezifikation eines Modulsystems
- Synchrones API
- Implementiert in node.js
- Hauptsächlich verwendet am Server

```
main.js:  var fs = require("fs");  
          var async = require("async");  
          var myModule = require("../src/myModule");
```

Verzeichnisstruktur:



AMD

- Spezifikation eines Modulsystems
- Asynchrones API
- Implementiert von z.B. require.js / Dojo
- Hauptsächlich verwendet am Client (funktioniert aber auch am Server)

```
declare(function() {  
    var mammal = function() { ... }  
    mammal.prototype...
```

Mammal.js:

```
    return mammal;  
});
```

```
main.js: require(["jQuery", "./Mammal"], function(jQuery, Mammal) {  
    var mammal = new Mammal();  
});
```

Node.js: Hello World

- Einfacher Web-Server, der “Hello, World!” zurückgibt
- Basierend auf node.js HTTP API

```
var http = require("http");

http.createServer(function(req, res) {
  res.end("Hello, World!");
}).listen(8080);

console.log("Server running at http://localhost:8080/");
```

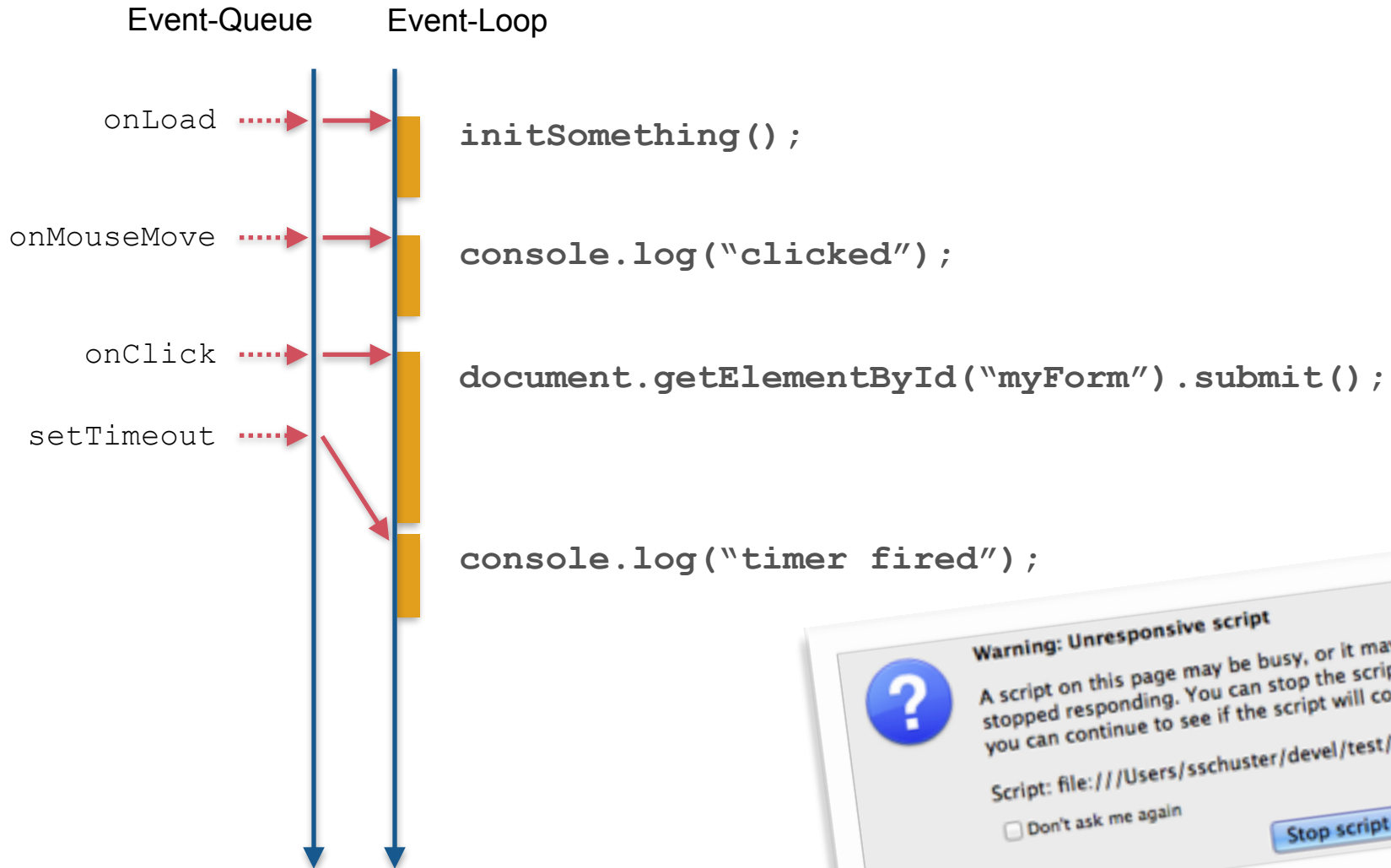
- Vergleichbar mit sehr einfachem Servlet
- Reicht für die Erstellung einfacher Apps
 - Runtime Environment ✓
 - Modulsystem ✓
 - “Servlet” API ✓

Asynchrones Programmieren

JavaScript = Event-Driven

- JavaScript hat meist nur kurze “Ausführungseinheiten”
- Diese werden durch Events angestoßen
 - onLoad
 - onClick
 - ...
- Event-Loop verwaltet Events und Ausführung
- Zwischen Events “idle”

Event-Loop = Event Queue



Node.js Event-Loop

- Selbes Konzept in Node.js
- Konsequenzen
 - Single-Threaded
 - Event-Driven
 - Kein klassisches, blockierendes Programmieren möglich

Hello World - Revisited

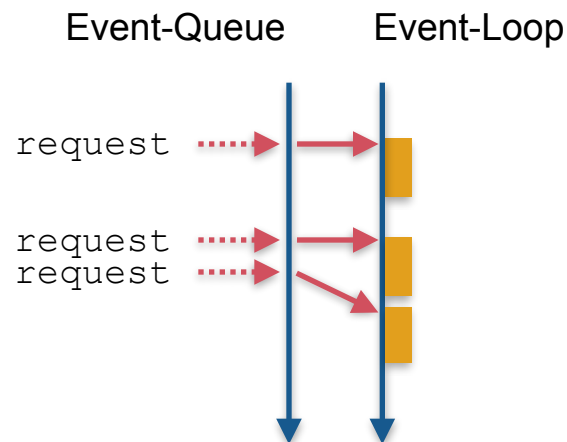
Event-Handler wird registriert

```
var http = require("http");
```

“Ausführungseinheit”

```
http.createServer(function(req, res) {  
  res.end("Hello, World!");  
}).listen(8080);
```

```
console.log("Server running at http://localhost:8080/");
```



Node.js - Pro/Contra

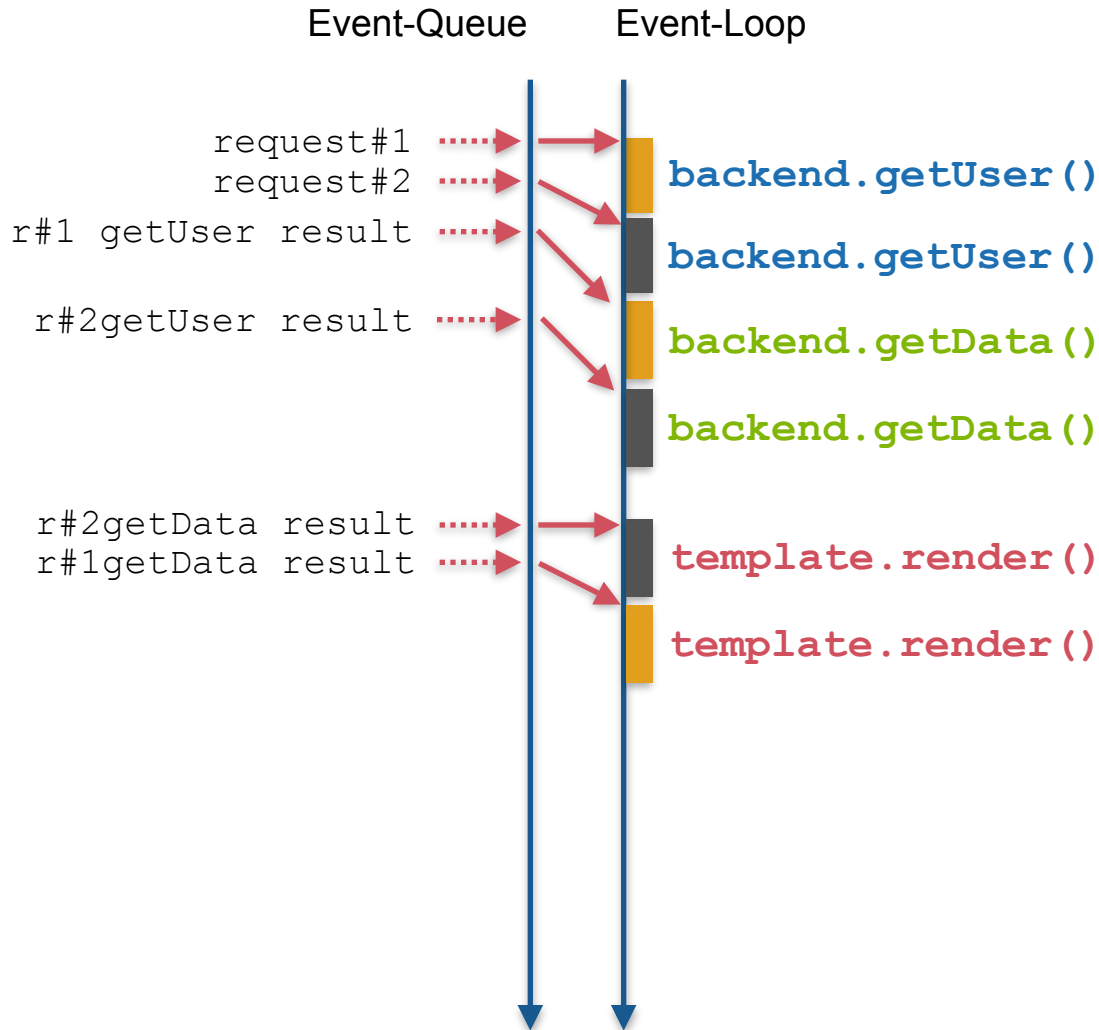
- Vorteile
 - Kein Overhead
 - 1 Prozess, 1 Thread für alles
 - Kein Thread pro Connection
 - Kann sehr viele gleichzeitige Connections halten
 - Überraschend schnell
- Nachteile
 - Gewöhnungsbedürftig / Komplex
 - “Manuelles” managen des “Schedulings”

Async Webapp mit Node.js?

```
var http = require("http");
var backend = require("./backend");
var template = require("./template");

http.createServer(function(req, res) {
  backend.getUser(req.headers, function(err, user) {
    //Todo: Error checking
    backend.getData(user, function(err, data) {
      //Todo: Error checking
      var html = template.render({ user:user, data:data });
      res.end(html);
    });
  });
}).listen(8080);
```

Async Webapp: Event-Loop



Async Webapp: Leserlicher

```
http.createServer(function(req, res) {
  var user, data;

  async.series([
    function(cb) {
      backend.getUser(req.headers, function(err, fetchedUser) {
        if (!err) user = fetchedUser;
        cb(err);
      })
    },
    function(cb) {
      backend.getData(user, function(err, fetchedData) {
        if (!err) data = fetchedData;
        cb(err)
      }
    )
  ], function(err) {
    if (err) {
      res.statusCode = 500;
      res.end("Server Error");
    }
    else {
      res.end(template.render({ user:user, data:data }));
    }
  });
}).listen(8080);
```

Node.js: Skalierbarkeit

- Single-Threaded
 - Limit ist, was dieser eine Thread = 1 CPU Core schafft
 - Überraschend viel
 - In der Praxis wird viel Zeit mit warten auf DB, ... verbracht - Hier spielt Node.js seine Vorteile aus
- Skalierungsoptionen
 - Node.js Cluster Modul
 - Mehrere Instanzen
 - Aufsplitten in mehrere Prozesse

Vert.x

- Open Source Async JVM Runtime Environment
 - Gestartet in 2011
 - Polyglot: JavaScript, Java, Ruby, Groovy, Python
 - Asynchrones Programmiermodell (unter anderem...)

```
var vertx = require("vertx");
var console = require("vertx/console");

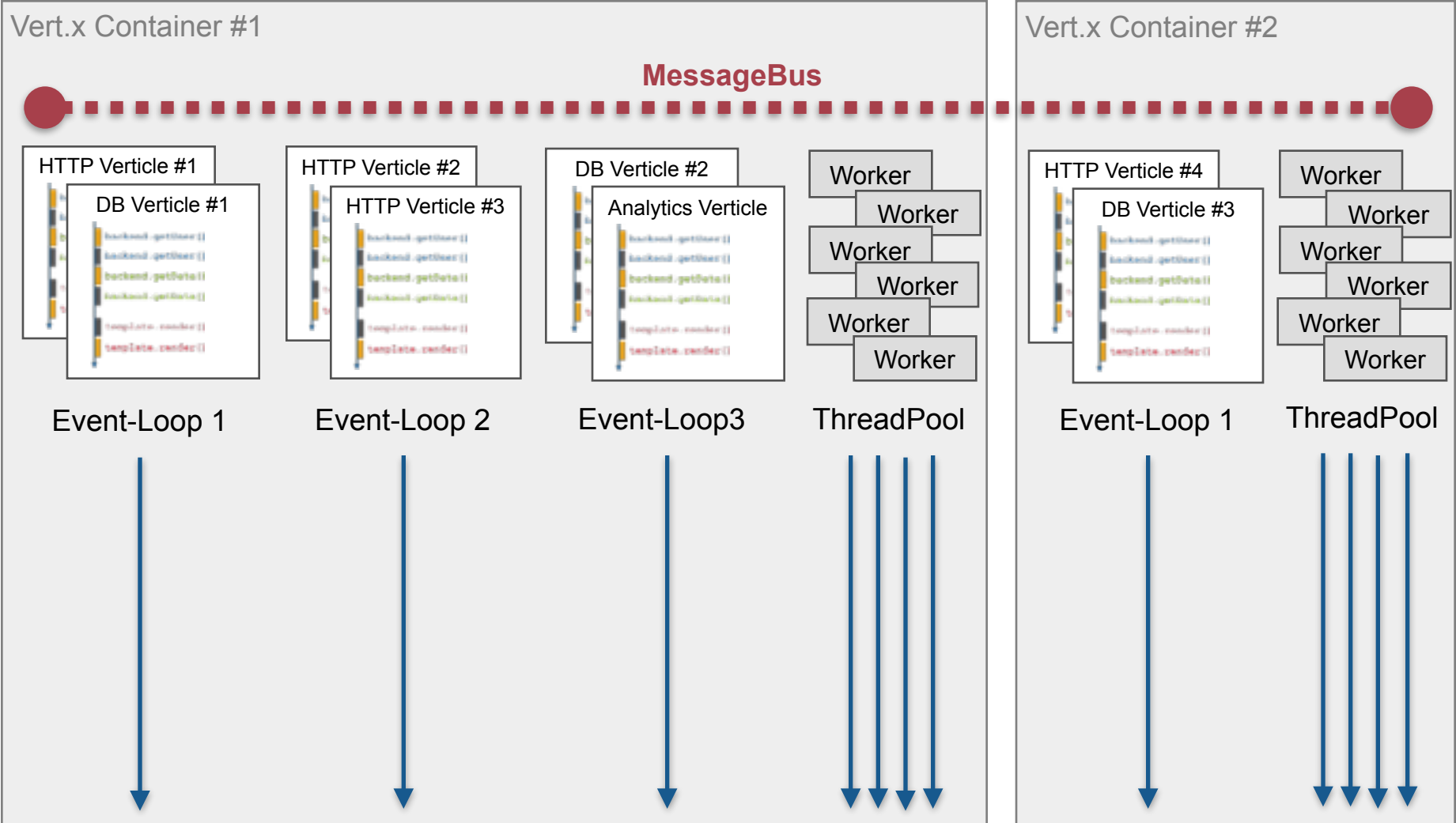
vertx.createHttpServer().requestHandler(function(req) {
    req.response.end("Hello, World!");
}).listen(8080);

console.log("Server running at http://localhost:8080/");
```

Vert.x - Event-Loop

- Vert.x mehr als nur eine Event-Loop
 - Mehrere Event-Loops (1 pro Core)
 - Mehrere “Verticles” per Event-Loop
 - Thread-Pool für Blocking Tasks
 - Message Bus
 - Cluster-fähig (durch Message Bus)
- Spannendes, noch recht junges Projekt

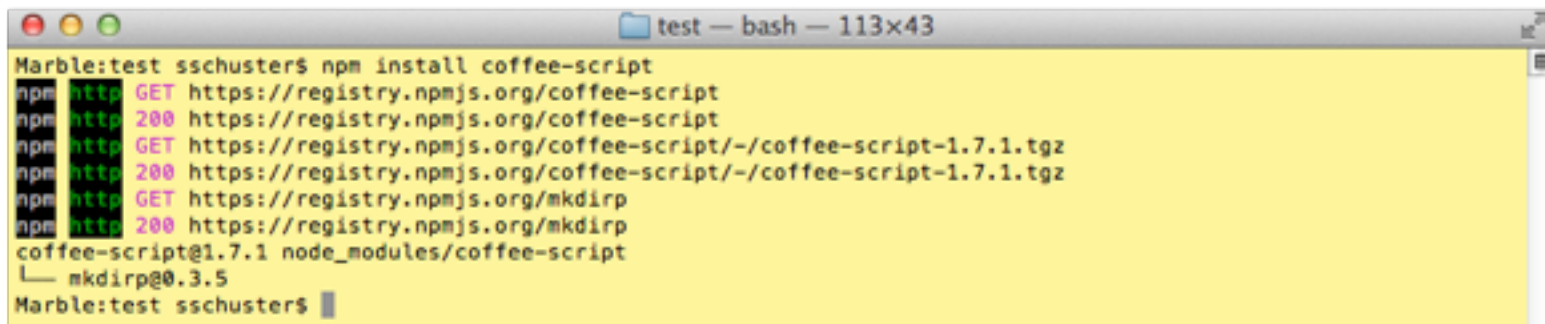
Vert.x - Container



Ökosystem

NPM

- Node Package Management System
 - Vergleichbar mit Maven
 - Teil von Node.js
 - > 60.000 Module
 - Gute Versionsverwaltung von Dependencies
 - package.json



```
test — bash — 113x43
Marble:test sschuster$ npm install coffee-script
npm http GET https://registry.npmjs.org/coffee-script
npm http 200 https://registry.npmjs.org/coffee-script
npm http GET https://registry.npmjs.org/coffee-script/-/coffee-script-1.7.1.tgz
npm http 200 https://registry.npmjs.org/coffee-script/-/coffee-script-1.7.1.tgz
npm http GET https://registry.npmjs.org/mkdirp
npm http 200 https://registry.npmjs.org/mkdirp
coffee-script@1.7.1 node_modules/coffee-script
└─ mkdirp@0.3.5
Marble:test sschuster$
```

package.json

```
{  
  "name": "mind42",  
  "version": "2.3.2",  
  "private": true,  
  "dependencies": {  
    "coffee-script": "1.6.3",  
    "express": "3.4.0",  
    "stylus": "0.38.0",  
    "ect": "0.5.2",  
    "async": "0.2.9",  
    "when": "2.5.1",  
    "underscore": "1.5.2",  
    "pg": "2.6.2",  
    "couchbase": "1.2.0",  
    "amqpplib": "0.1.0",  
    "moment": "2.2.1",  
    "node-uuid": "1.4.1",  
    "validator": "1.5.1",  
    "nodemailer": "0.5.2",  
    "libxmljs": "0.8.1",  
    ...  
  }  
}
```

NPM Registry



Most Depended Upon

- 5315 **underscore**
- 4442 **async**
- 3685 **request**
- 2551 **commander**
- 2518 **express**
- 2485 **optimist**
- 2298 **lodash**
- 2040 **coffee-script**
- 1898 **colors**
- 1428 **mkdirp**
- **More...**

Most Starred

- 223 **express**
- 116 **async**
- 100 **request**
- 94 **grunt**
- 87 **socket.io**
- 82 **mocha**
- 72 **lodash**
- 69 **underscore**
- 58 **mongoose**
- 56 **redis**
- **More...**

Pakete

- Web-Frameworks
 - express, stylus, ejs, ...
- Datenbank Treiber
 - pg, mysql, redis, couchbase, ...
- Libraries / Utilities
 - async, underscore, node-uuid, ...
- Datenformate
 - Excel, CSV, ZIP, ...

Pakete

- Netzwerk
 - socket.io, amqp, ...
- Wrapper
 - Imagemagick,
- Testing
 - mocha, chai, ...
- Rendering
 - node-canvas, ...

Beispiele

Simple CRUD Webapp

- Kleines, praktisches Beispiel
 - Express
 - Templating
 - Async DB
- <https://github.com/sschuster/javalandDemo>

DEMO

package.json

```
{  
  "name": "javalanddemo",  
  "version": "0.0.1",  
  "private": true,  
  "dependencies": {  
    "express": "3.5.0",  
    "ejs": "0.8.5",  
    "async": "0.2.10",  
    "sqlite3": "~2.1.5",  
    "sequelize": "1.7.0"  
  }  
}
```

main.js

```
var path = require("path");
var express = require("express");
var server = require("./src/server");

var app = express();
app.set("views", path.join(__dirname, "src/views"));
app.use(express.urlencoded());
app.use(app.router);
app.use(express.static(path.join(__dirname, "public")));
app.use(express.errorHandler());

server.init(app);

var server = app.listen(8080, function() {
  console.log("Listening on port 8080");
});
```

persistence.js

```
var path = require("path");
var Sequelize = require("sequelize");

var persistence = new Sequelize("javalanddemo", "username", "password", {
  dialect: "sqlite",
  storage: "./javalanddemo.sqlite"
});

var Todo = persistence.import(path.join(__dirname, "/model/todo"));
persistence.sync();

module.exports = {
  Todo: Todo
};
```

todo.js

```
module.exports = function(sequelize, DataTypes) {  
  return sequelize.define("Todo", {  
    description: { type: DataTypes.STRING, allowNull: false, validate: { notEmpty: true }  
  },  
    done: { type: DataTypes.BOOLEAN, allowNull: false, defaultValue: false }  
  }, {  
    tableName: "todos"  
  });  
};
```

server.js - GET - /

```
app.get("/", function(req, res, next) {
  var todos;

  async.series([
    function(callback) {
      persistence.Todo.findAll({
        order: "createdAt ASC"
      }).complete(function(err, fetchedTodos) {
        if (!err) {
          todos = fetchedTodos;
        }
        callback(err);
      });
    }
  ], function(err) {
    if (err) {
      next(err);
    }
    else {
      res.render("todos.ejs", {
        todos: todos
      });
    }
  });
});
```

todo.ejs

```
<body>
  <h1>Todos:</h1>
  <% for(var i=0, todo; (todo = todos[i]); i++) { %>
    <div class="todo">
      <form class="edit" method="post" action="/edit/<%= todo.id %>">
        <input type="text" name="description" value="<%= todo.description %>">
        <input type="checkbox" name="done" value="true" <%= todo.done ? "checked" : "" %>>
        <input type="submit" value="Save">
      </form>
      <form class="delete" method="post" action="/delete/<%= todo.id %>">
        <input type="submit" value="Delete">
      </form>
    </div>
  <% } %>

  <div class="todo new">
    <form class="create" method="post" action="/create">
      <input type="text" name="description" size="50">
      <input type="submit" value="Create">
    </form>
  </div>
</body>
```

server.js - POST - /create

```
app.post("/create", function(req, res, next) {
  async.series([
    function(callback) {
      persistence.Todo.create({
        description: req.body.description
      }).complete(callback);
    }
  ], function(err) {
    if (err) {
      next(new Error("Creating todo failed"));
    }
    else {
      res.redirect("/");
    }
  });
});
```


server.js - POST - /edit

```
app.post("/edit/:todoId", function(req, res, next) {
  var todo;

  async.series([
    function(callback) {
      persistence.TODO.find(req.params.todoId).complete(function(err, fetchedTodo) {
        if (!err) {
          todo = fetchedTodo;
        }
        callback(err);
      });
    },
    function(callback) {
      todo.description = req.body.description;
      todo.done = (req.body.done && req.body.done == "true") ? true : false;
      todo.save().complete(callback);
    }
  ], function(err) {
    if (err) {
      next(new Error("Updating todo failed"));
    }
    else {
      res.redirect("/");
    }
  });
});
```

server.js - POST - /delete

```
app.post("/delete/:todoId", function(req, res, next) {
  var todo;

  async.series([
    function(callback) {
      persistence.TODO.find(req.params.todoId).complete(function(err, fetchedTodo) {
        if (!err) {
          todo = fetchedTodo;
        }
        callback(err);
      });
    },
    function(callback) {
      todo.destroy().complete(callback);
    }
  ], function(err) {
    if (err) {
      next(new Error("Deleting todo failed"));
    }
    else {
      res.redirect("/");
    }
  });
});
```

**Vielen Dank für Ihre
Aufmerksamkeit!**

Fragen?