

Web-basierte Systeme – Übung

o6: Node.js, MongoDB und Aufgabe 5

Wintersemester 2023

Arne Vogel



Lehrstuhl für Verteilte Systeme
und Betriebssysteme



Friedrich-Alexander-Universität
Technische Fakultät

Node.js

- Überblick

- Non-blocking I/O

- Module

- Events

- Express

MongoDB

- mongoose

Virtualisierung & Docker

- OpenStack

Aufgabe 5

Node.js

Node.js

- Überblick

- Non-blocking I/O

- Module

- Events

- Express

MongoDB

- mongoose

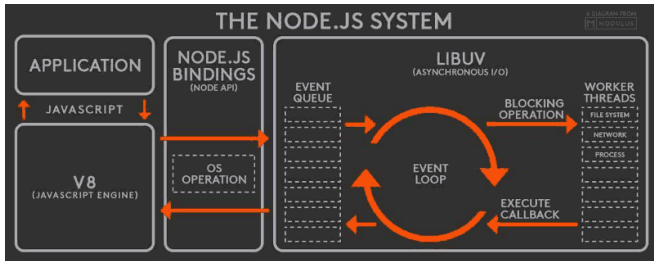
Virtualisierung & Docker

- OpenStack

Aufgabe 5

- Node.js ist eine **serverseitige** Laufzeitumgebung für JavaScript
- **Node.js = V8 + Event Loop + Async. I/O API**
- **npm**: Packetmanager für Node.js
- Alle I/O Operationen **non-blocking** möglich

Node.js: Event Loop



- **Ein Thread** führt JS des Entwicklers + Event Loop aus
- Anfragen werden **direkt vom OS** oder von **Worker Threads** bearbeitet

Quelle: <https://www.timcosta.io/the-node-js-event-loop/>

Node.js: non-blocking I/O (Übersicht)

- Alle I/O Operation haben **synchrone** und **asynchrone** Versionen
 - Synchron: Direkter Aufruf mit Argumenten
 - Asynchron: Zusätzliches **Callback** Argument
- Beispiel aus der Node.js API:
 - `fs.readFileSync(path[, options])`
 - `fs.readFile(path[, options], callback)`
- Callback Funktionen
 - Konvention: Error Objekt ist **immer** der erste Parameter
 - Bsp. für `readFile`: `function(err, data) {...}`
 - Je nach API Call mehr/andere Parameter möglich!
- **Promises**-basierte API
 - Gleiche API-Aufrufe, aber Promise wird returned
 - z.B. `fs.promises`

Node.js: non-blocking I/O (Beispiel)

Synchrones Lesen einer Datei:

```
1 const fs = require('fs');
2 const data = fs.readFileSync('file.md'); // blocks here
3 console.log(data);
4 moreWork(); //will run after console.log
```

Asynchrones Lesen einer Datei:

```
1 const fs = require('fs');
2 fs.readFile('file.md', (err, data) => { // does NOT block here
3   if (err) throw err;
4   console.log(data);
5 });
6 moreWork(); //will run before console.log
7
```

Node.js Module

- Node.js bietet viele **Module**, welche unterschiedliche Funktionen implementieren, Beispiele (built-in):
 - Datei- und Netzwerkzugriff: fs und net
 - Netzwerkprotokolle: http, dns, ...
 - Kryptographie: crypto, tls
 - Kompression: zlib
 - ...
- Quasi **Libraries** für Node.js
- Hunderttausende zusätzliche Pakete in **npm**
- Module werden mit dem Schlüsselwort `require` eingebunden

```
1 var net = require('net');
2 var client = new net.Socket();
3 client.connect(1337, '127.0.0.1', function() {
4     client.write('Hello, server! Love, Client.');
```

```
5 });
```

Eigene Node.js Module

- Module **exportieren** JavaScript Objekte
- Zum Beispiel Funktionen:

```
1 // bar.js
2 module.exports.bar = function () {
3     console.log('bar!');
4 }
```

- Eigene Module können dann auch mit `require` verwendet werden:

```
1 // app.js
2 var lib = require('./bar.js');
3 lib.bar();
```

Events

- Mit Node.js lassen sich **Event-getriebene** (event-driven) Anwendungen bauen
- **Events** werden von **Emittern** ausgesendet
- **Listener** sind Funktionen, die bei Events ausgeführt werden
- Dazu müssen Listener mit der `.on()` Methode **registriert** werden

```
1  const EventEmitter = require('events');
2  class MyEmitter extends EventEmitter {}
3
4  const myEmitter = new MyEmitter();
5  myEmitter.on('event', () => {
6    console.log('an event occurred!');
7  });
8  myEmitter.emit('event');
9
```

Node.js Events: Beispiel Webserver

- Server hört auf Port 8081
- GET Requests werden mit dem String Hello World! beantwortet
- Alle anderen Requests werden ignoriert
- Dokumentation: <https://nodejs.org/api/http.html>

```
1 var http = require('http');
2 var server = http.createServer().listen(8081);
3
4 server.on('request', function (request, response){ //Behandlung des Events 'request'
5     if (request.method==='GET'){
6         response.writeHead(200, {'Content-Type': 'text/html'});
7         response.end("Hello World!\n");
8     }
9 });
```

Node.js: Express

- Meistgenutztes Web-Framework für Node.js
- Bereits als Abhängigkeit des Servers vorgegeben
 - Installierbar mit `npm install` (siehe Übungsblatt)
- Antworten auf HTTP Requests werden über **Routen** definiert
- Das `app`-Objekt stellt `.get()`, `.post()`, ... zur Verfügung

```
1 var express = require('express');
2 var app = express();
3
4 // GET method route
5 app.get('/', function (req, res) {
6   res.send('GET request to the homepage')
7 })
8
9 // POST method route
10 app.post('/', function (req, res) {
11   res.send('POST request to the homepage')
12 })
```

- Pfade der Route können natürlich angepasst werden

```
1 app.get('/', function (req, res) {
2   res.send('root')
3 })
4
5 app.get('/about', function (req, res) {
6   res.send('about')
7 })
8
9 app.get('/users/:userId/', (req, res) => {
10   res.send(req.params) // /user/vogel => req.params: { "userId": "vogel" }
11 })
```

- Dabei sind auch reguläre Ausdrücke möglich:
 - /ab?cd → acd und abcd
 - /ab+cd → abcd, abbcd, abbbcd, ...
 - /ab*cd → abcd, abxcd, abRANDOMcd, ab123cd, ...
 - ...

- Zur Erhöhung der Übersichtlichkeit können die Definitionen von Routen in externe JavaScript-Dateien **ausgelagert** werden:

```
1  const index = require('./routes/index.js');
2  const service = require('./routes/service.js');
3
4
5  app.use('/', index);
6  app.use('/service', service);
```

- Das NPM Paket `express-ws` bietet Endpunkte von WebSockets für Express

```
1 ■ var router = express.Router();
2
3 router.ws('/ws', function(ws, req) {
4   ws.on('message', function(msg) {
5     ws.send("message received!");
6   });
7 });
8
9 app.use("/", router);
```

Node.js: Umgebungsvariablen

- Mit **Umgebungsvariablen** kann man einfach Parameter an Node.js Anwendungen weitergeben
- Die Variable wird in der Bash gesetzt:

```
1 $ TESTVAR="hello" nodejs app.js
```

- Und kann in Node.js ausgelesen werden:

```
1 const testvar = process.env.TESTVAR;
```

- Auch Standardwerte können definiert werden:

```
1 const testvar = process.env.TESTVAR || 'default_value';
```

MongoDB

Node.js

Überblick

Non-blocking I/O

Module

Events

Express

MongoDB

mongoose

Virtualisierung & Docker

OpenStack

Aufgabe 5

- MongoDB ist eine quelloffene NoSQL-Datenbank
 - Abgeleitet vom engl. humongous, „gigantisch“
 - NoSQL: **nicht** relational!
 - Keine Festlegung auf Tabellenschemata
 - Stattdessen werden Daten als **Dokumente** abgelegt
- Ablage von **JSON-ähnlichen Objekten**
 - Erweiterung von JSON → Binary JSON (BSON)
 - Vorteile: geringere Speicherbelegung, schneller zu parsen
- Bekannt für **hohe Skalierbarkeit**
- Oft eingesetzt in Webanwendungen
 - MEAN-Stack: MongoDB, Express, Angular, Node.js
 - MERN-Stack: MongoDB, Express, React, Node.js

- **Mongoose** ist ein ODM Tool für MongoDB
- ODM: Object Document Mapper
 - Objekt im Code → Dokument in Datenbank
- **Schema**: Mapping von Objekten zu MongoDB
- Schemas werden zu **Models** kompiliert
- Models bieten verschiedene Methoden:
 - Neues Dokument erzeugen: `new Model()`
 - Neues Dokument speichern: `Model.save()`
 - Alle Vorkommen finden: `Model.find()`
 - Einzelnen Eintrag löschen; `Model.deleteOne()`
 - Einzelnen Eintrag ändern: `Model.updateOne()`

MongoDB: mongoose ODM (Beispiel)

```
1  const mongoose = require('mongoose');
2  mongoose.connect('mongodb://localhost:27017/test', {useNewUrlParser: true});
3
4  //new schema
5  var catSchema = new mongoose.Schema({
6    name: String
7  });
8
9
10 //new model:
11 const Cat = mongoose.model('Cat', catSchema);
12
13 //create new cats
14 const cat1 = new Cat({ name: 'Simon' });
15 const cat2 = new Cat({ name: 'Garfield' });
16
17 //save cats
18 cat1.save();
19 cat2.save().then(function(){
20   console.log("cat2 saved!");
21   Cat.find({}, function(err, cats){ //{} is an empty Conditions Object
22     console.log(cats)
23     mongoose.connection.close();
24     console.log("connection closed");
25   });
26 });
```

Virtualisierung & Docker

Node.js

Überblick

Non-blocking I/O

Module

Events

Express

MongoDB

mongoose

Virtualisierung & Docker

OpenStack

Aufgabe 5

■ Schaffung virtueller Ressourcen auf Basis von physischen

- Analog zu multitasking Betriebssystemen
 - Virtual Machine Monitor (VMM), **Hypervisor**
- Virtuelle Maschine (VM)

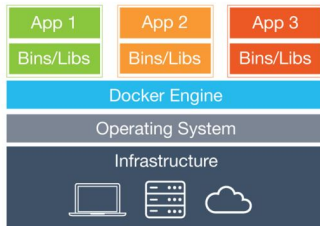
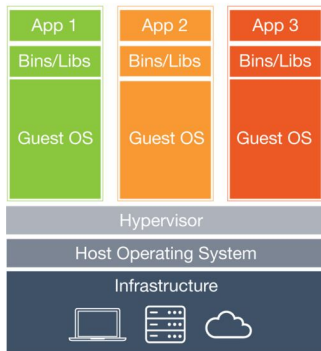
■ Ziele von Virtualisierung

- Bessere Ausnutzung existierender Ressourcen
- Erhöhung von Verlässlichkeit und Sicherheit
- Höhere Skalierbarkeit von Systemen
- Zentralisierung von Systemadministration
- Betrieb von Altsystemen ohne alte Hardware

■ **Problem:** Hoher Ressourcenbedarf → Containervirtualisierung

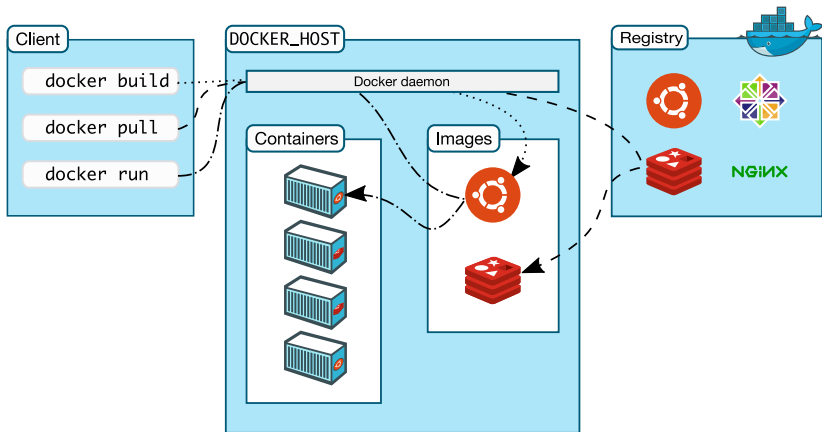
- Gleiche Ziele wie normale Virtualisierung
- Es werden **nur einige** Betriebsmittel isoliert
- Bspw. gemeinsame Nutzung des Betriebssystemkerns
- Bekannteste Implementierung: **Docker**
- **Problem:** Schwache Isolation ggü. normaler Virtualisierung

Virtualisierung vs. Docker



Quelle: <https://www.stratoscale.com/blog/compute/linux-containers-benefits-and-market-trends/>

Docker: Interface



Quelle: <https://docs.docker.com/engine/docker-overview/>

Docker als Deployment Tool

- Durch die **Kapselung** von Anwendung und Bibliotheken eignet sich Docker sehr gut als **Deployment Tool**
- Anwendungen können mit allen Abhängigkeiten gebündelt als Dockercontainer vertrieben werden
- Anwendungen mit **mehreren Komponentent** beinhalten dann mehrere Dockercontainer
- Die Kommunikation zwischen diesen kann mit `docker-compose` konfiguriert werden

- Mit `docker-compose` können Anwendungen bestehend aus **mehreren Docker Containern** definiert und ausgeführt werden
- Konfiguration über YAML-Datei `docker-compose.yml`:

```
1 services:
2   mongodb:
3     restart: always
4     image: mongo:latest
5     ports:
6       - "27017:27017" # this is dangerous! Dont do this in a production system
7
8   node:
9     restart: always
10    build: ./chat_server
11    environment:
12      - DB_URL= mongodb://mongodb/chatDB
13    links:
14      - mongodb
15    command: npm start
```

- Wichtige Befehle: `docker-compose [up|down|kill|logs]`
- `docker-compose` bereits auf VMs vorinstalliert

- **OpenStack** ist eine quelloffene Software, die eine Architektur für das sogenannte Cloud-Computing zur Verfügung stellt
- U.a. schnelles erstellen von **virtuellen Maschinen** (VMs)
- Das i4 betreibt eine angepasste Version unter <https://i4cloud1.cs.fau.de/>
- Ablauf für Aufgabe 5:
 - Start einer VM über das Webinterface
 - Zugriff per ssh
 - Kopieren von Dateien mit scp oder rsync

OpenStack: Webinterface

openstack web-sys_steschmi goltzsch Sign Out

Project

- Compute
- Overview
- Instances
- Volumes
- Images
- Access & Security
- Network
- Orchestration
- StudentCloud

Instances

Filter Filter [+ Launch Instance](#) [Soft Reboot Instances](#) [Terminate Instances](#)

<input type="checkbox"/>	Instance Name	Image Name	IP Address	Size	Key Pair	Status	Task	Power State	Uptime	Actions
<input type="checkbox"/>	WebSys	(not found)	192.168.0.11 134.169.47.199	balanced-small 1GB RAM 1 VCPU 10.0GB Disk	stefan	Active	None	Running	1 week, 1 day	Create Snapshot More

Displaying 1 item

The screenshot displays the OpenStack web interface. At the top left is the OpenStack logo. To its right is a dropdown menu showing the current project name 'web-sys_steschmi'. Further right is a user profile dropdown for 'goltzsch' and a 'Sign Out' link. On the left side, there is a navigation sidebar with a 'Project' dropdown menu. Under 'Project', there are links for 'Compute', 'Network', 'Orchestration', and 'StudentCloud'. Below these are 'Info', 'Quota usage', 'Running Projects' (highlighted with a red bar), and 'Requested Projects'. The main content area is titled 'Project Details: web-sys_steschmi' and contains three tabs: 'Overview' (selected), 'Members', and 'Quota'. The 'Project Overview' section lists the following details:

- Name:** web-sys_steschmi
- Description:** Backend für die Übungsaufgaben für Web-sys
- Start Date:** Feb. 13, 2018, midnight
- End Date:** Dec. 31, 2019, midnight

The screenshot shows the OpenStack web interface. At the top left is the OpenStack logo. To its right is a dropdown menu showing the current project: 'web-sys_steschmi'. Further right is a user profile dropdown for 'goltzsch' and a 'Sign Out' button.

The main content area is titled 'Project Details: web-sys_steschmi'. It has three tabs: 'Overview', 'Members', and 'Quota'. The 'Quota' tab is currently selected.

On the left side, there is a navigation menu under the heading 'Project'. The menu items are: 'Compute', 'Network', 'Orchestration', 'StudentCloud', 'Info', 'Quota usage', 'Running Projects', and 'Requested Projects'. 'Running Projects' is highlighted with a red vertical bar.

The 'Project Quota' section contains the following text: 'These quota values describe the allocatable resources by the project. If more quota is needed, please contact one of the project supervisors.'

The quota values are listed as follows:

- Subnets: 2
- Networks: 2
- Floating IPs: 2
- Volume Storage (GB): 25
- RAM (MB): 8192
- Snapshots: 4
- Instances: 2
- Volumes: 2
- Router: 2
- Cores: 4

Aufgabe 5

Node.js

Überblick

Non-blocking I/O

Module

Events

Express

MongoDB

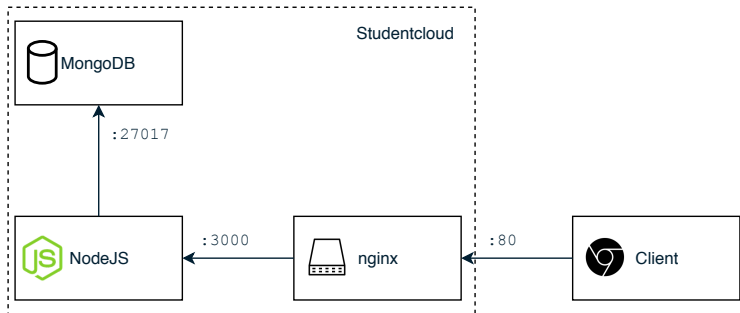
mongoose

Virtualisierung & Docker

OpenStack

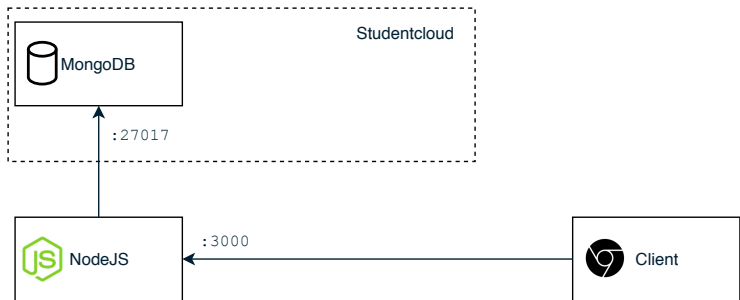
Aufgabe 5

Aufgabe 5: Endgültiges Deployment



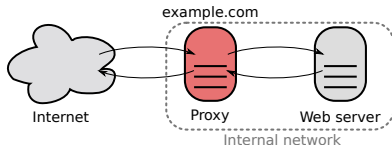
- Implementierung des Chatserver, der bisher vorgegeben war
- Komponenten: Node.js (mit Express), nginx, MongoDB
- Jede Komponente läuft in **Docker Container**
- Docker Container laufen auf **Virtueller Maschine (VM)**
- Eine VM pro Team in der **OpenStack**

Aufgabe 5: Deployment während Entwicklung



- Während der Entwicklung muss Servercode laufend angepasst werden
- Nur die MongoDB soll in der OpenStack ausgeführt werden

- nginx ist eine bekannte offene Webserver-Implementierung
- Hier: Einsatz als **Reverse Proxy**
- Gründe für einen Reverse Proxy:
 - Verstecken der Existenz/Charakteristik des eigentlichen Servers
 - Einsatz von Caching zur Entlastung der eigentlichen Server
 - Lastverteilung auf mehrere Server ohne Konfiguration des Clients
 - Hinzufügen von TLS Verschlüsselung
 - HTTP Authentifizierung
 - ...



Quelle: https://de.wikipedia.org/wiki/Reverse_Proxy

- <http://expressjs.com/en/guide/routing.html>
- <https://www.npmjs.com/package/express-ws>
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Proxy_servers_and_tunneling

Promise

```
1  const myPromise = new Promise((resolve, reject) => {
2    setTimeout(() => {
3      resolve('foo');
4    }, 300);
5  });
6  myPromise
7    .then(handleResolvedA)
8    .then(handleResolvedB)
9    .then(handleResolvedC)
10   .catch(handleRejectedAny);
```
