Vorlesungsreihe
Entwicklung webbasierter Anwendungen

Serverfarmen, C10M-Ansätze mit NodeJS, Express und Nuxt

Prof. Dr.-Ing. Thomas Wiedemann
email: wiedem@informatik.htw-dresden.de

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)
Fachbereich Informatik/Mathematik

Gliederung

- Aufbau und Konfiguration von **Serverfarmen und High-Performance-Servern**
  - Notwendigkeit und historische Entwicklung des Load Balancing
- Diskussion zu C10K und C10M
- **NodeJS** und Erweiterungen (**MEAN-Stack**)
  - NodeJS-Grundlagen und Beispiele
  - Das **Express Framework**
  - Das **Vue**-orientierte Serverframework **Nuxt**

Entwicklung von Serverfarmen und des Load Balancing

Typische Hardwareanforderung beim Aufbau von Webanwendungen

- zu Beginn steht (bei Startups?) das Preis/Leistungsverhältnis im Fokus
- bei Erfolg waren / sind dann schnelle Anpassungen an die höheren Zahlen der User-Requests notwendig
- **Probleme:**
  - **einfache PC-basierte Server können trotz Hauptspeicherausbaus nur in Grenzen in der Antwortkapazität erweitert werden**
  - **stärkere Workstations oder Supercomputer stellen wiederum spezielle Softwareanforderungen oder unterstützen nicht alle Optionen der Anwendungen**

Häufige (und heute typische) Lösung

- **Aufbau von größeren Rechnerclustern auf PC-Hardwarebasis und meist Linux-Betriebssystemen (Serverblades)**
- **Die Anfragen müssen dann möglichst gleichmässig auf die Rechner verteilt werden (= Load Balancing)**

Allgemeine Anforderungen an das Load Balancing

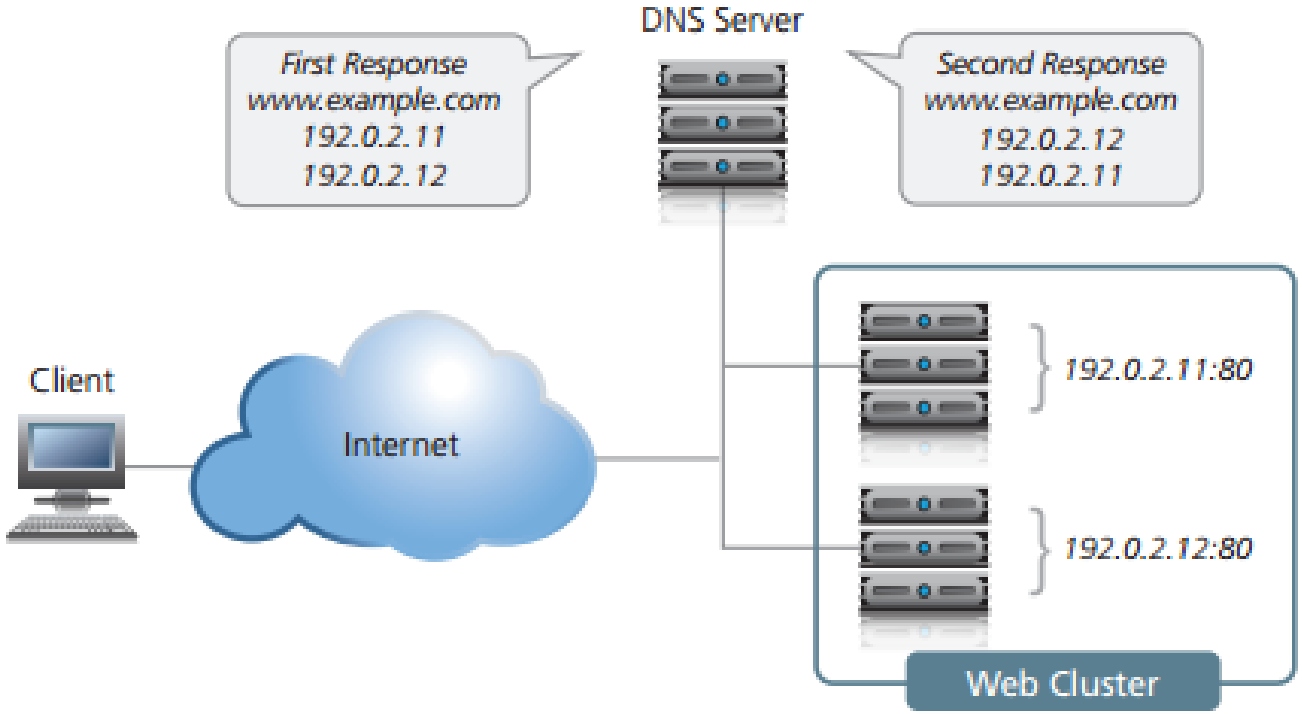
Im Rahmen der Lastverteilung sind folgende Anforderungen zu erfüllen:

- **Hochverfügbarkeit (highly available)**
  - auch bei Ausfall einzelner Server muss die Gesamtfunktionalität der Serverfarm erhalten bleiben
- **Skalierbarkeit (scalable)**
  - Neue Anforderungen an die Leistung müssen einfach und ohne Beeinträchtigung des vorhandenen Systems umgesetzt werden können
- **Voraussagbares, konsistentes Verhalten (predictable)**
  - Das Verhalten der Serverfarm muss aus Sicht der Webapplikationen ohne Unterschiede zu einem Einzelsystem funktionieren und konsistente Ergebnisse bei mehrfachen Aufrufen bringen.

Erste Optionen des Load Balancing

Load Distribution unter Nutzung der DNS-Server

- DNS-Round Robin (pro Aufruf werden mehrere Adressen vom DNS-Server zurückgegeben, dies ergibt ein eher zufälliger Verteilen, kein Balancing)
- funktioniert für kleinere Serverfarmen auch heute noch ganz gut, Probleme können auftreten bei long-term-Sessions und Serverausfällen

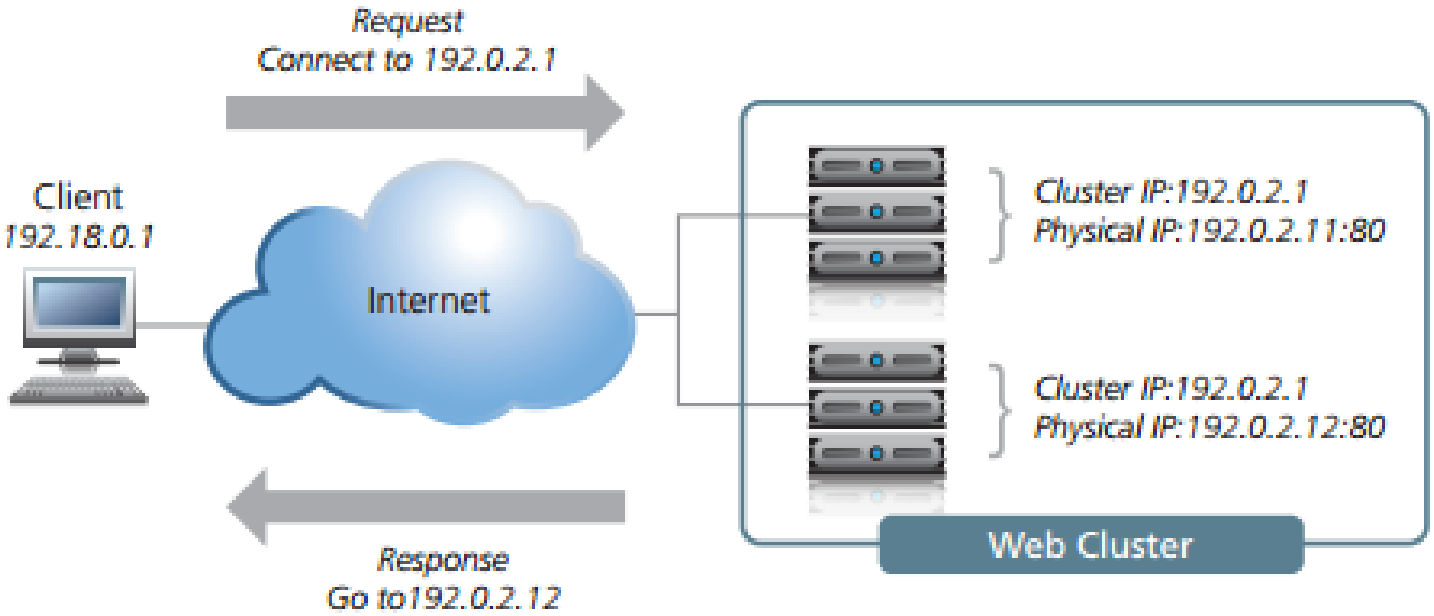


Quelle: http://www.f5.com/pdf/white-papers/

Optionen des Load Balancing

Load Balancing auf Softwarebasis

- Erster Client-Aufruf der Anwendung erfolgt über eine zentrale Cluster-IP,
- der Cluster entscheidet dann in der Anwendungssoftware (oder auf OS-Niveau), welcher Server die geringste Auslastung hat und initiiert einen REDIRECT auf diesen Server.
- Bei geringer Serverzahl (<10) sehr gut, dann anwachsende Probleme mit dem Serverabgleich.

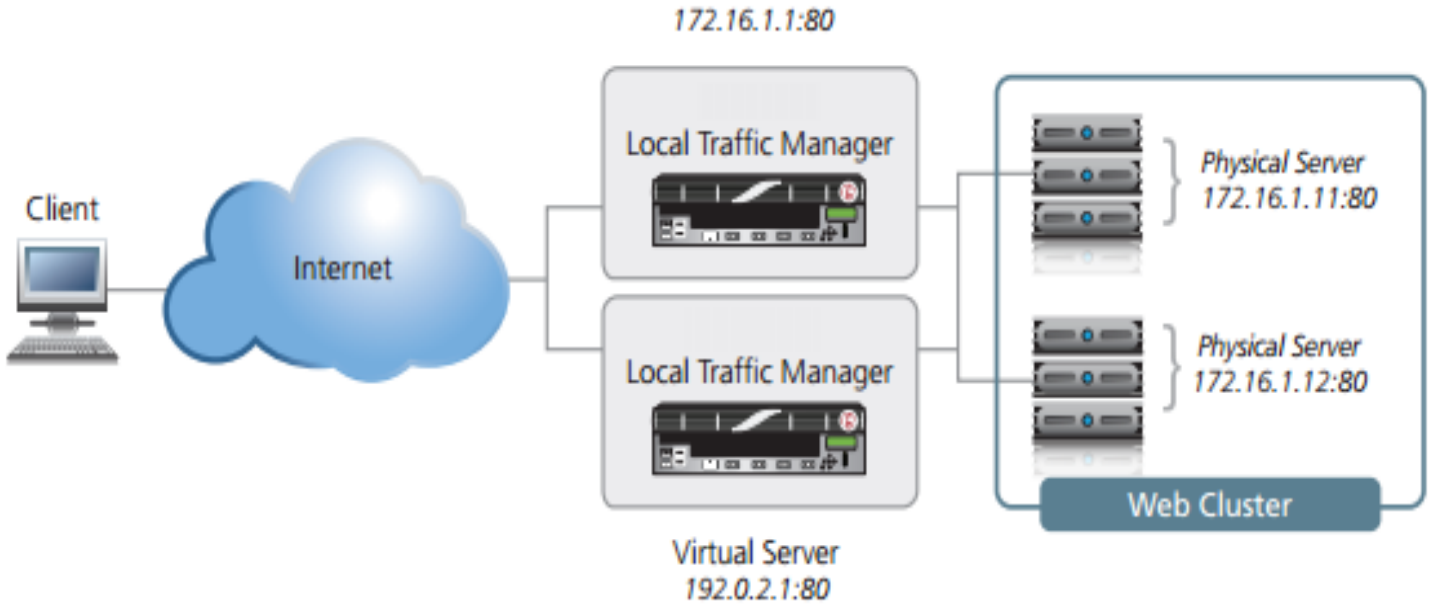


Quelle: http://www.f5.com/pdf/white-papers/

Optionen des Load Balancing

Netzwerk-Infrastruktur-basiertes Load Balancing

- Lastverteilung durch die Netzwerkrouter (spezielle Hardware >Performance)
- mit bidirektionalem Network Address Translation (NAT)
- Verteilung der Anfragen erfolgt durch ständiges Messen der Auslastung und Antwortzeiten und kann auch unterschiedliche Hardwareausbauten in der Serverfarm unterstützen



Quelle: http://www.f5.com/pdf/white-papers/

Aktuelle Entwicklungen

Weiterentwicklung des reinen Load Balancings zum Application Delivery Controller (ADN) mit folgenden Zusatzfunktionalitäten:

- **Health Monitoring:** Gesamtüberwachung der Serverfarm auf Funktionalität und Leistung
- **Dynamic provisioning** : Automatisches Verwalten der verfügbaren Rechnerresourcen (Erweiterung bei Last / Abschaltung zwecks Energiesparen)
- **Intelligentes Caching und Kompression** in Abhängigkeit von den Inhalten
- einheitliche **Authentifizierung** auf dem ADN (spart verteilte Auth. auf den Applikationsservern selbst
- **Location - basiertes Verteilen** der Anfragen (Suche nach dem netzwerkstechnisch "räumlich nächsten" Serverpool
- Bereitstellung von **Content Delivery Networks (CDN) (**vgl. Goggle-Jquery-Bereitstellung oder Ablage hochfrequentierter Presse-Seiten)
- Integration weiterer Services (wie Antiviruskontrolle etc.) in die Hardwareebene

Allgemeine Probleme beim Load Balancing

- Probleme mit Long-term http-Connections :
  - Bei Verwendung von long term Sessions oder größeren Datenoperationen (Download per ftp oder http) müssen die folgenden Requests wieder auf dem gleichem Server erfolgen, damit die Daten ab der aktuellen Stelle verfügbar sind
  - Auch bei permanenten Server-Datenobjekten (DLL oder Java Enterprise) muss eine konstante Zuordnung zum Server erfolgen !
- **Mögliche Lösung:** Speicherung der Quell-IP im Load Balancer und konstante Zuordnung des Clients zum Hosts (dies kann aber wiederum zu Problemen mit Clients hinter einem Proxy geben)

- Beim Ausfällen von Serverhardware muss dies erkannt und ggf. eine erneute Aussendung des Request erfolgen

Beispiele für aktuelle Serverfarmen

(Schätzungen nach https://www.topdatacenters.com/blog/list-of-top-data-center-countries/) – ca. 550 Data-centers weltweit

- **Google Data Center**
  - ca. 13 weltweit verteilte Data Center mit ca. 900,000 Servern
    (Verbrauch ca. **300 Megawatt** = 0.01% der Welt-Elektroenergie
    = ausreichend für ca. 200,000 Haushalte)
  - In der Regel werden neue Datacenter in der Nähe von Wasserkraftwerken (billige Elektroenergie + Kühlung) gebaut -> Colorado / Finnland
- **Microsoft Data Center**
  - ca. 1.000.000 Server weltweit verteilt (europäischer Node in Dublin Irland)
- **Amazon Data Center**
  - 450,000 Server, davon ca. 40.000 für Cloud-Nutzer (siehe Folgeseiten)

Andere Big-Player (Facebook, Domain-Hoster) verfügen über ähnliche Serverfarmen.

Verteilungen der aktuellen Serverfarmen

Stand 11.2019

Germany (195)

- **top cities Frankfurth Munich**

The Netherlands (100)

Singapore (34)

Sweden (50)

Norway (29)

United Arab Emirates (9)

United Kingdom (264)

China (79)

Brazil (45)

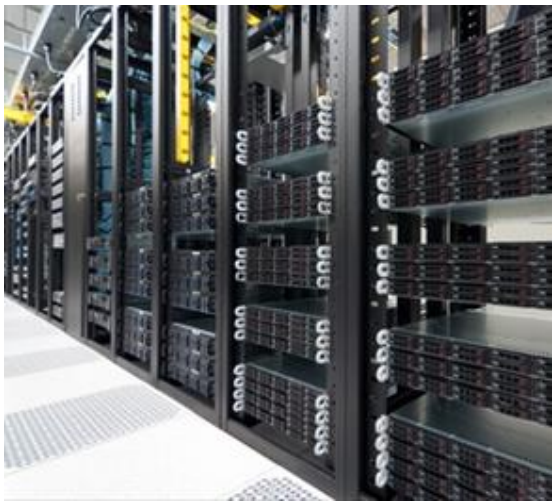
Achtung: USA nicht gelistet, müsste noch einmal ca. 500 Center haben (?)

USA: https://www.topdatacenters.com/blog/the-largest-data-centers-in-the-usa/

Trends bei aktuellen Serverfarmen

Trends **nach** https://www.topdatacenters.com/blog/data-center-trends-facts-statistics/

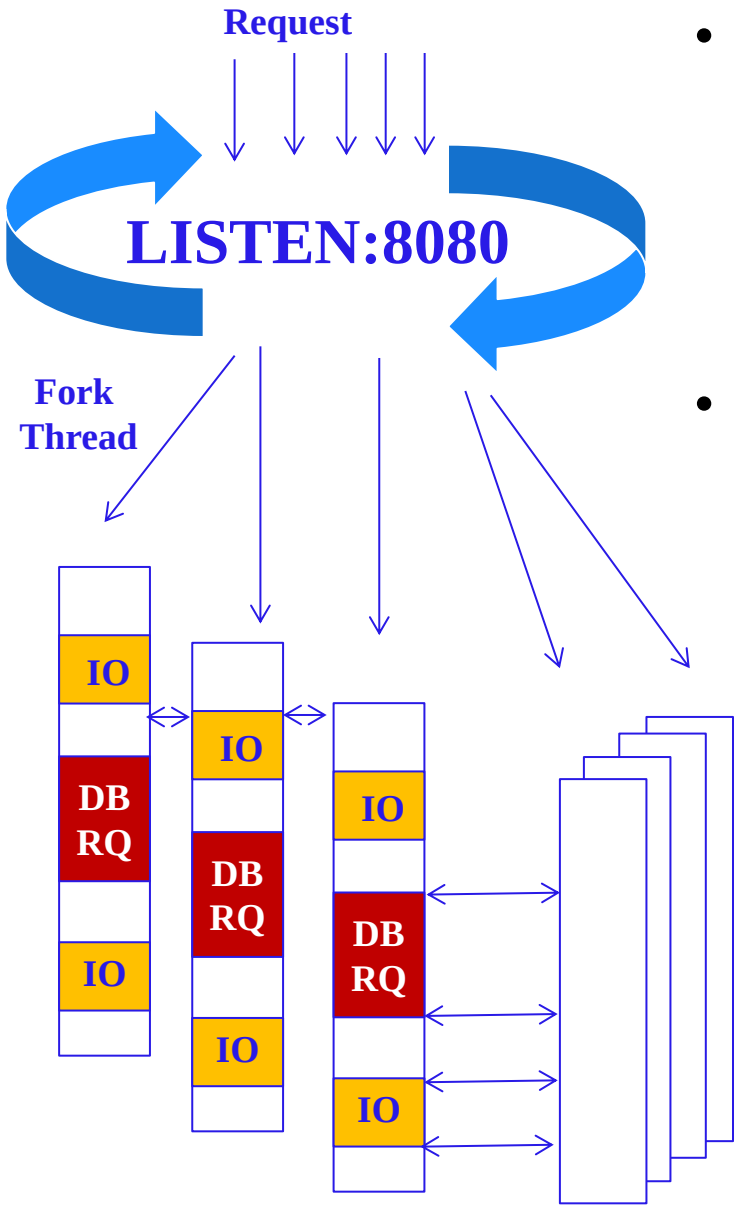
2019-2020 Data Center Trends



- AI will continue to grow 34% Year over year into 2020
- Disaster recovery service demand grew 23% in 2019
- The average tier 1-2 data center uses **24k miles of network cable (= 45.000 km)**
- **Service cost for on server about 500 ….800$/ month**
- 2019 saw 562 Hyperscale data centers existing
- AI is reducing the cooling costs which make up 40% of Data Center costs
- 28% of cloud spending is focused on private cloud hosting
- 2020-2022 10% of IT organizations will be using serverless computing

- **10/2019 : Apple to Spend 10B (10 Mrd $) on US Data Centers Over 5 Years**
  - https://www.topdatacenters.com/blog/apple-to-spend-10b-on-us-data-centers-over-5-years/
- **Weitere Trends: https://www.vxchnge.com/blog/facts-about-data-centers**

Entwicklung neuer Strategien - Kritik der bisherigen Server



- Bei bisherigen Servern (wie z.B. dem Apache-HTTP-Server) läuft eine relativ kleine LISTEN-Schleife und generiert für jeden URL-Request jeweils einen NEUEN (schwergewichtigen) Thread im Betriebssystem
- OS-Threads sind
  - relativ aufwändig bzgl. Speicheranforderungen und auch **langsam** beim Umschalten (meist ist ein **Kontext-Switching** im Prozessor mit jeweiligem Speicher-Sichern auf dem Stack erforderlich…)
  - In der Default-Installation ist z.B. beim Apache eine **max. Anzahl von 200 Threads** eingestellt
  - Das Problem hat sich verschärft durch neue mobile Anwendungen (Whatsapp etc.) mit sehr **VIELEN KLEINEN Anfragen**

Aktuelle Hardwareleistung vs. Serveranforderungen

- Die nur ca. 200 parallelen Anfragen pro Zeiteinheit stehen in starken Kontrast zur aktuellen Leistungsfähigkeit der Hardware:

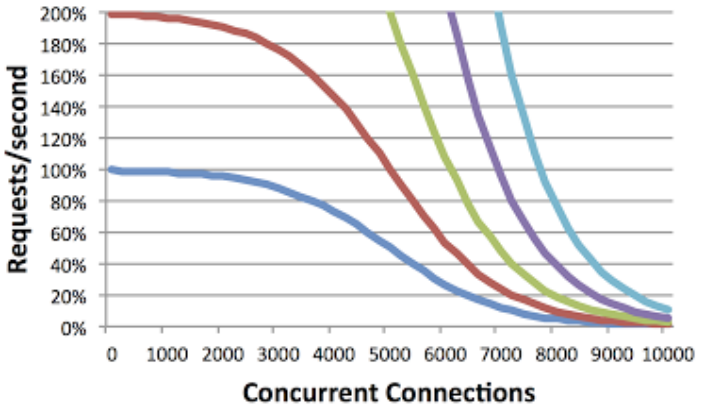
- Ein „besserer“ Hardwareserver mit 8 Cores, 3 GHz, 64 GByte RAM und 10-Gbit Ethernet erlaubt theoretisch :
  - 10 Gbits/s Übertragungsrate und damit 10 Millionen Pakete /s, bei nur einigen Paketen (mit 0.5 KByte) pro Connections wären damit
  - **1 Million parallele Connections möglich**

- Entsprechende Diskussionen zu diesem Thema sind unter den Stichworten **„C10K-Problem“** (http://www.webcitation.org/6ICibHuyd)

- oder aktuell analog zu „C10M-Problem“ (10 Million connections) finden – siehe http://c10m.robertgraham.com/p/blog-page.html

Skalierbarkeit der Serverleistung

- Die bereits diskutierte OS-Thread-Abhängigkeit von Apache verhindert auch eine Skalierbarkeit durch bessere Hardware:
- Bei entsprechend starker Belastung bricht die Anzahl möglicher paralleler Verbindungen zusammen - Ursachen: ältere Unix- und Win-OS erlaubten nur ca. 5000 bis 7000 Interrupts pro /s und entspr.viele Handler



Quelle: c10m.robertgraham.com/p/blog-page.html

- **1. Lösung: Optimierung der OS** in Richtung größerer Interrupt, Puffer- und Handler-bereiche ➔ Erreichung von 10.000 Connections (C10K), aber danach trotzdem Stagnation
- **2. Lösung (und Erkenntnis) :** „The Secret To 10 Million Concurrent Connections **-The Kernel Is The Problem**, Not The Solution “ –

see http://highscalability.com/blog/2013/5/13/the-secret-to-10-million-concurrent-connections-the-kernel-i.html

➔ **Auslagerung** der performancekritischen Codeteile **AUS DEM SERVER** (d.h. Management der 10 Mio. Connections durch eigenen Code !!!

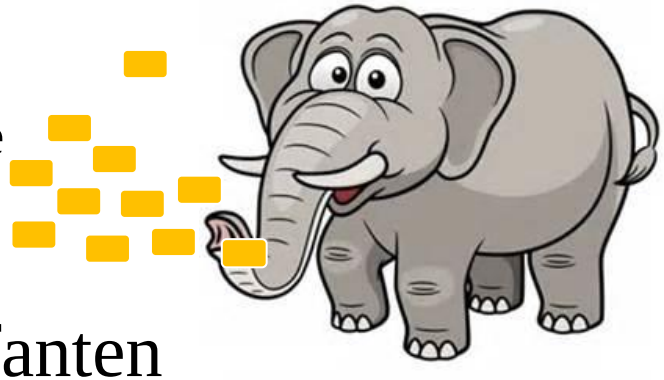
Bessere Lösungen ?

Vergleich mit der Natur …

Traditionelle Server (wie Apache) sind

- Langsam und benötigen eine große Menge an Ressourcen (Futter)
- designt für große Datenpakete

Sie sind ineffizient
für die vielen
kleinen Datenpakete
mobiler Apps



analog zu Elefanten

und sie
stören sich
selbst bei
der
Arbeit…

Bessere Ideen ?

Wir benötigen KLEINE, kooperative und und hoch-effiziente Arbeitstiere wie

Bienen

Diese sind

- sehr gut organisiert
- arbeiten sehr strukturiert
- benötigen sehr wenig Ressourcen

Wie können wir dies in die IT übertragen ?

Merke : "**The Kernel Is The Problem**, Not The Solution !"

Solution: ➔ Don't use the **Kernel for context switching,**
but make your own – better solution !!!

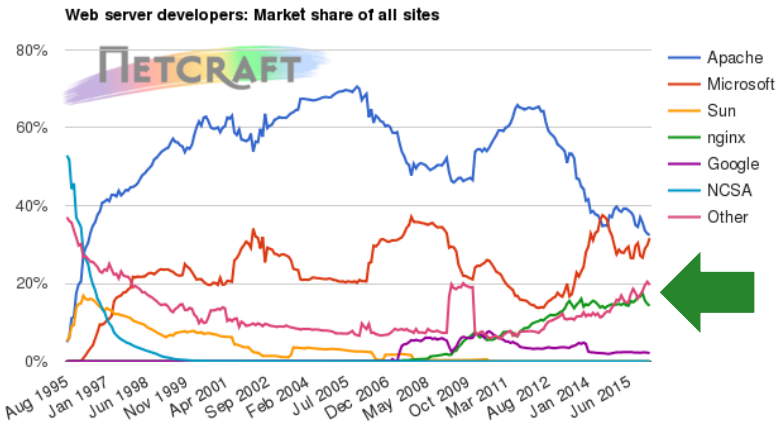
Good news : Dieser Job ist bereits erledigt !

➔ Durch die Nutzung von JavaScript in der sehr effektiven Multitasking-Umgebung von Googles V8-Engine entstand **Node.js** ***(see next pages)***

➔ 2. Option: russische Entwickler implementierten die

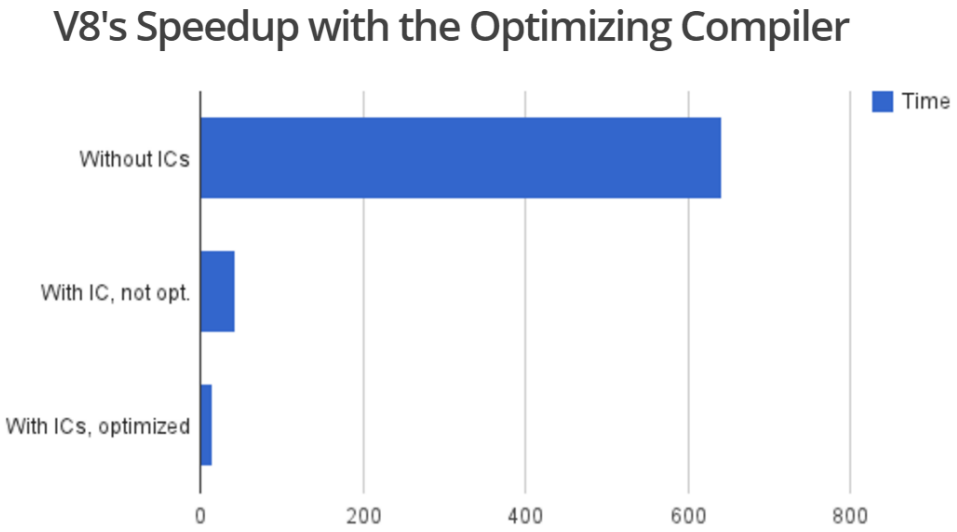
Nginx ("engine x") http://nginx.org/

- ➔ High-performance server mit sehr resource foot print (about 10.000 requests with only 3 Mbytes RAM !!!)
- ➔ sehr modular / weit konfigurierbar
- ➔ Stark anwachsende Nutzung
- ➔ Auch benutzt durch Wordpress / Netflix …



Serverlösungen mit NodeJS

- NodeJS-basiert auf V8 JavaScript Engine von Google, welche auch im Chrome-Browser eingesetzt wird
- Die V8-Engine wurde über die Jahre stark bzgl. Performance optimiert und erreicht bei optimal geschrie-benen JS-Code die Performance vom C++-Code !!! (bzgl. Details siehe https://developers.google.com/v8/



Quelle: http://v8-io12.appspot.com/#68

Historische Entwicklung von NodeJS

- **Erste Version 2009 durch Ryan Dahl in der Fa. Joyent mit Fokus auf Cloud-Computing**
- **package manager (npm) zur einfacheren Installation von Bibl. 2011**
- Anfang 2015 wurde Node.js Foundation gegründet (nach einem Intermezzo mit io.js wegen Differenzen bzgl. Weiterentwicklung)
- Offizielle Webpage: https://nodejs.org/ (MIT Lizenz)

Installation of NodeJS

- **Download und Installation von NodeJS von**
  https://nodejs.org/
- **Node hat ein commandline interface - REPL (Read / Eval / Print / Loop), welches interaktive Tests erlaubt :**

```
node
> 1 + 3
 4
```

- **erlaubt sind die meisten bekannten JavaScript-Befehle**

```
>console.log("Hello World!");
```

- **.help für help**
- **.exit oder 2 x CTRL+C or CTRL+D beendet REPL**

- **Details zu REPL unter**
  https://www.tutorialspoint.com/nodejs/nodejs_repl_terminal.htm

Serverlösungen mit NodeJS – Einfacher HTTP-Server

- NodeJS beinhaltet bereits alle Komponenten für einen HTTP-Server

```
var http = require('http');
var server = http.createServer(function(request, response)
 {   console.log('Got Request Headers: ');
      console.log(request.headers);
      response.writeHead(200, {  'Content-Type': 'text/plain' });
      response.write('Hello World');
      response.end();
}).listen(8080);
```

Besonderheiten:

- **NodeJS arbeitet mit einer Event-driven architecture und Nicht-blockierenden I/O- (non-blocking I/O API), damit ist kein langsames Betriebssystem-Thread-Multitasking notwendig und alle diskutierten Nachteile herkömmlicher Server entfallen (➔ C10K … C10M-Connections)**

Serverlösungen mit NodeJS – HTTP-Server mit Pfadmapping

- **Ein etwas besserer HTTP-Server mit Pfadmapping und Fehlerkontrolle**

```
var http = require('http');
var fs = require('fs'); /
var url = require('url');
http.createServer( function (request, response) {// Create a server
var pathname = url.parse(request.url).pathname; // Parse the request
console.log("Request for " + pathname + " received.");
fs.readFile(pathname.substr(1), function (err, data) {// Read the requested file
    if (err) {    console.log(err);      // HTTP Status: 404 : NOT FOUND
                response.writeHead(404, {'Content-Type': 'text/html'});
    }else {        //Page found        // HTTP Status: 200 : OK
            response.writeHead(200, {'Content-Type': 'text/html'});
            response.write(data.toString()); // Write the content of the file
    }  response.end(); // // Send the response body
  });
}).listen(8081);
```

Serverlösungen mit NodeJS – Streaming files over HTTP

- Mit zusätzlichen File-Read-Funktionen können auch größere Dateien gestreamt werden :

```
require('http').createServer(function(req, res) {
    res.writeHead(200, {'Content-Type': 'video/mp4'});
    var rs = fs.createReadStream('test.mp4');
    rs.pipe(res);
}).listen(8080);
```

Besonderheiten:

- **Die Übertragung auf die obige Art wird vom Browser auch bei noch vollständiger Übertragung verarbeitet und die Wiedergabe sollte damit sofort beginnen**

NodeJS – multicore – servers

- NodeJS ist ein single-threaded Programm – falls mehrere Prozessoren vorhanden sind, können auch mehrere Instanzen gestartet werden
  (Example from https://nodejs.org/api/cluster.html)

```
const cluster = require('cluster');
const http = require('http');
const numCPUs = require('os').cpus().length;
if (cluster.isMaster) {
  // Fork workers.
  for (var i = 0; i < numCPUs; i++) {    console.log("Start cluster  " +  i ) ;
    cluster.fork();    }
  cluster.on('exit', (worker, code, signal) => {
    console.log(`worker ${worker.process.pid} died`);
  });
} else {
  // Workers can share any TCP connection
  // In this case it is an HTTP server
  http.createServer((req, res) => {
    res.writeHead(200);
    res.end('hello world\n');
  }).listen(8000);
}
```

Das NodeJS – express framework

- Der Programmierlevel von NodeJS entspricht etwa einer C++-Entwicklung eines HTTP-Server – ist also sehr aufwändig für Standard-Webanwendungen.
- Bei mehreren Pfadmappings müsste für den Pfad eine entsprechende Funktion geschrieben und eine größere Anzahl if/switch generiert werden
- Aus diesem Grund entstanden schnell zusätzliche Frameworks, welche diese Aufgaben effizient und komfortabel übernehmen
- Das bekannteste Framework ist **Express.js (ebenfalls mit JS entwickelt)**

Express

- **wurde von Douglas Christopher Wilson entwickelt und ist momentan in der Version 4.16.x verfügbar.**
- **ermöglicht die Einrichtung von Middleware, um auf HTTP-Anfragen zu antworten.**
- **Definiert eine Routing-Tabelle, die verwendet wird, um verschiedene Aktionen basierend auf HTTP-Methoden und URL auszuführen.**
- **Ermöglicht die dynamische Darstellung von HTML-Seiten auf Grundlage der Übergabe von Argumenten aus Vorlagen (Templates) -> EJS – API**

Das NodeJS – Express framework

- Express läuft HINTER dem NodeJS-Server und verbindet diesen mit zusätzlichen Funktionen, z.B. auch zur Realisierung komplexerer Middleware-Aufgaben.

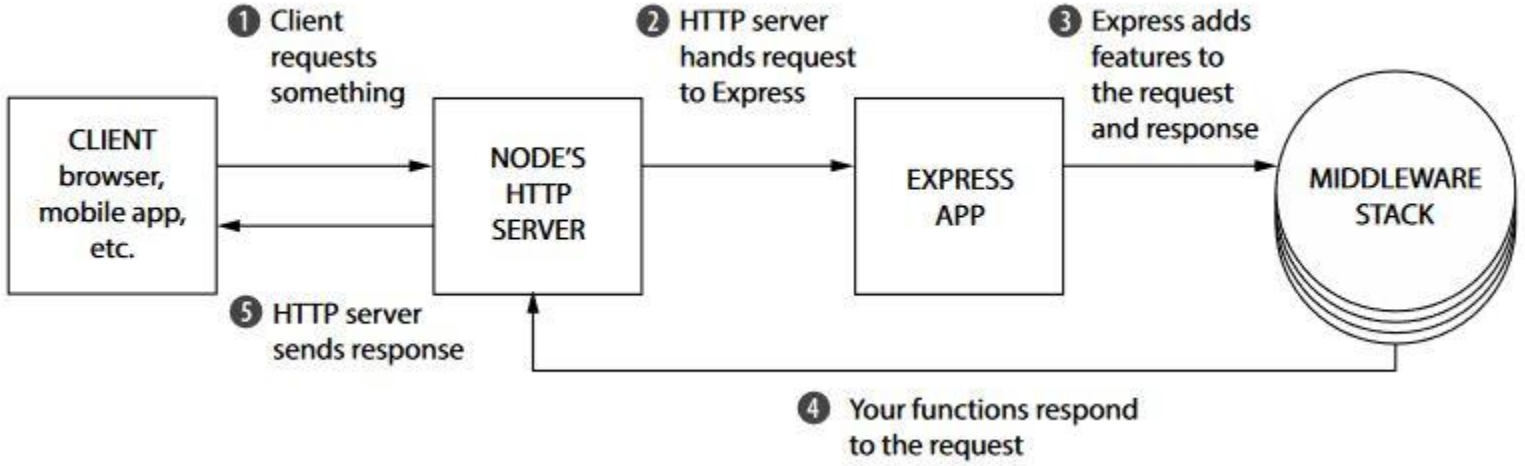


Abb.-Quelle: Evan M. Hahn: Express in Action - Writing, building, and testing Node.js applications, Manning Publications, 2016

- Weitere Quellen und Referenzen zu Express:
  - **http://expressjs.com**
  - **tutorialspoint.com**
  - **http://www.embeddedjs.com/**
  - **https://blog.xervo.io/nodejs-and-express-create-rest-api**

Das NodeJS – Express framework - Installation

- **Installation läuft über npm.**

```
$ npm install express
```

- **Nun kann Express importiert werden, indem man die NodeJS-require-Funktion aufruft.**

```
var express = require(`express`);
```

- **Instanziiert und gestartet wird Express analog zum NodeHS-http-Server**

```
var app = express();
http.createServer(app).listen(3000);

// dieser Server ist online, stellt aber noch keine Funktionalität bereit !
```

Das NodeJS – Express framework – „Hallo Welt“ - Beispiel

- „Hallo Welt“ – Beispiel mit Express.

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {   // mappt die root zur Funktion !
  res.send('Hello World!');
});

app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
});
```

- Der Server ist über Port 3000 empfangsbereit.
- Mit app.get wird ein **Routing** von der URL (hier /) zu einem JS-Code definiert !
- Der Code wird in einer Datei abgespeichert app.js und dann ausgeführt:

```
$ node app.js
```

- dann erreichbar unter http://localhost:3000

The NodeJS – express framework – Routings

- Es können beliebig viele Routings angegeben werden :

```
var express = require('express');  var app = express();

app.get('/', function (req, res) {   // Routing to  /
   res.send('Hello World from root '); })

app.get('/data', function (req, res) {   // Routing to  /data
   res.send('Hello World from data'); })
  //   usw.    …
app.use(function(request, response) {  // Default  ->  hier  Fehlerbehandlung
response.status(404).send(“Seite nicht gefunden!");
});
var server = app.listen(8081, function () {
   var host = server.address().address
   var port = server.address().port
      console.log("Example app listening at http://%s:%s", host, port)
}) //   Achtung: Die Routings werden sequentiell geprüft und
    //   bei Erfolg wird die Suche beendet, ansonsten kommt die  „404“ … !
```

Das NodeJS – Express framework – Routing-Methoden

- Neben dem gezeigten app.get()-methode existieren weitere Routing-Methoden:
  - **app.all()** **// Kann alle Requests behandeln**
  - **app.get()** **// Behandelt nur GET-Requests**
  - **app.post()** **// Behandelt nur POST-Requests**
  - **app.put()** **// Behandelt nur PUT-Requests**

- **Zusätzlich werden auch noch spezielle HTTP-Methoden und Spezialrequests unterstützt**
  copy, lock, mkcol, move, purge, propfind, proppatch, unlock, report, mkactivity, checkout, merge, m-search, notify, subscribe, unsubscribe, patch, search und connect, post, head, put, options, delete, trace, connect.

- **Beispiel für POST (siehe auch komplettes Post-Form-beispiel auf Folgeseite):**

```
// POST method route
app.post('/', function (req, res) {
res.send('POST request zur Webseite');
});
```

The NodeJS – express framework - Arbeit mit POST-Daten

- Daten aus einem Standard-HTML-Formular

```
<form action = "http://127.0.0.1:8081/process_get" method = "GET">
   First Name: <input type = "text" name = "first_name">  <br>
   Last Name: <input type = "text" name = "last_name">
   <input type = "submit" value = "Submit">    </form>
```

- können mittels **rec.query** – analysiert werden :

```
var express = require('express');
var app = express();
app.get('/process_get', function (req, res) {
   var  response = " The submitted name is " +  req.query.first_name + "  " +
req.query.last_name ;
 res.send( response    );
})
```

- Andere Operationen, wie File uploads, cookie und session management analog !
- see : https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm

The NodeJS – express framework - Serving static files

- Die Bereitstellung statischer File und das Logging werden ebenfalls unterstützt:

```
var express = require('express');
var app = express();
app.use(function(request, response, next) {  // Logging the requests
console.log("In comes a " + request.method + " to " + request.url);
next();   });

app.use(express.static('files'));  // serve  static files form dir „files“
app.use(express.static(‘public'));  // serve  static files form dir „public“

var server = app.listen(8082, function () {
   var host = server.address().address;    var port = server.address().port;
   console.log("Example app listening at http://%s:%s", host, port)
})
```

- **Achtung: Der Inhalt des Verzeichnisse wird in die root gemappt ! So ist es sinnvoll bei größeren Datenmengen Unterverzeichnisse zu verwenden !**

Express – Weitere Antwort-Methoden

- Neben der bereits gezeigten send()-Methode bietet Express auch noch weitere Methoden zum Beantworten an:

Methode	Beschreibung
res.download()	Gibt eine Eingabeaufforderung zum Herunterladen einer Datei aus.
res.end()	Beendet den Antwortprozess
res.json()	Sendet eine JSON-Antwort
res.jsonp()	Sendet eine JSON-Antwort mit JSONP-Unterstützung (wrappped payload)
res.redirect()	Leitet die Anforderung um
res.sendFile()	Sendet eine Datei als Oktett-Stream
res.send()	Sendet eine Text-Antwort

Beispiele:

res.download('/Beispieldatei-12345.pdf'); // bereitstellung File

res.redirect('http://htw-dresden.de'); // Umleitung zu HTW-HP

Die EJS-Template-Erweiterung

- Mit den bislang gezeigten Beispielen war die gesamte Anwendungslogik immer noch in der JS-Datei konzentriert.
- Modernere Webapp-Konzepte orientieren auf einen Template-Ansatz, welcher den HTML-Code von der Geschäftslogik trennt (vgl. PHP- Templates – VL PHP2)

- **EJS („Effective JS“) ist eine von vielen Template-Engine für NodeJS / Express**

Funktionen der Template-Engine

- **Effektivere Darstellungslogik innerhalb von HTML-Docs:**
  - **Ausgabe von Variablen**
  - **Farbgestaltung**
  - **Anzeigen von Formularen**
- **Geschäftslogik:**
  - **Verbindung zu Datenbanken und Abruf von Datenbankdaten**
  - **Rechenoperationen**
  - **Formulararbeit**
  - **Datensicherung**

Referenzen: http://ejs.co/

Die EJS-Erweiterung – Installation und Initialisierung

- **Die Installation erfolgt über die Konsole mit:**

  $ npm install ejs

- **Nach der Installation muss Express nur noch mitgeteilt werden, dass es EJS verwenden soll. Dies ist sehr einfach und lässt sich mit einer einzigen kurzen Codezeile umsetzen:**

  app.set(„view engine“, „ejs“);

- **Damit wird Express mitgeteilt, daß bei einer Dateierweiterung = „EJS“ die Template-Engine verwendet werden soll !**
- **Dabei wird ein spezieller Views-Ordner verwendet, in welchem die *.EJS-Dateien ablegen werden !**
- **-> notwendiger Unterordner „views“ mit HTML-dateien mit Endung .ejs**

Die EJS-Erweiterung - Statisches Anwendungsbeispiel

- Im nachfolgenden Beispiel wird die Ausgabe mit EJS vorbereitet und die Datei Beispiel.ejs als Template für die Ausgabe definiert:

```
var express = require(„express“);
var app = express();
app.set(„view engine“, „ejs“); //Konfiguration der Template-Engine EJS

app.get(„/“, function(req, res){
   res.render(“Beispiel.ejs“);
});
var server = app.listen(3000, function(){
  console.log(„Der Server ist nun betriebsbereit“);
});
```

- **Ablage des Programms als app.js und starten des Servers in der Konsole:**

  $ node app.js

- **Anzeige auf Konsole nun: „Der Server ist nun betriebsbereit“ und bei Requst wird die HMTL-Datei „beispiel.ejs“ ausgeliefert (= noch static file)**

Die EJS-Erweiterung - Dynamisches Anwendungsbeispiel

- sinnvoller wird das Beispiel mit der Übergabe von Daten an das Template

```
var express = require(„express“);
var app = express();
app.set(„view engine“, „ejs“); //Konfiguration der Template-Engine EJS
app.get(„/“, function(req, res){
  res.render(„Beispiel.ejs“, {   „title“: „Ich bin die Titelvariable!“        });
});
var server = app.listen(3000, function(){
  console.log(„Der Server ist nun betriebsbereit“);
});
```

- **Für die Anzeige von Titel muss das Template entsprechend um <% %>-Tags erweitert werden:**
- <!DOCTYPE html>
- <html> <head>
- **<title><%= title %></title> //Zugriff auf die Variable**
- </head><body> <h1>Welt</h1> </body>
- </html>

Die EJS-Erweiterung - Tag-Typen

- Details sind unter http://ejs.co/ und http://www.embeddedjs.com/ abrufbar

Die wichtigsten Tag-Types:

- **<%** 'Scriptlet' tag, for control-flow, no output
- **<%=** Outputs the value into the template (HTML escaped)
- **<%-** Outputs the unescaped value into the template
- **<%#** Comment tag, no execution, no output
- **<%%** Outputs a literal '<%'
- **%>** Plain ending tag
- **-%>** Trim-mode ('newline slurp') tag, trims following newline

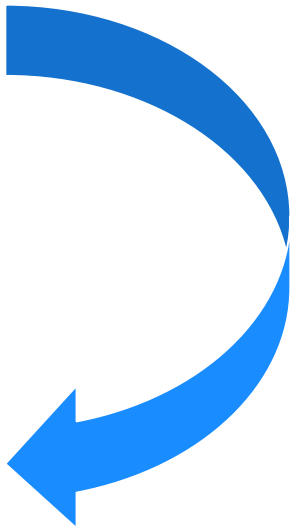
Die EJS-Erweiterung - 2. Anwendungsbeispiel

- Komplexere Daten mit logischen Abfragen im HTML-Template

```
… app.set("view engine", "ejs");
app.get("/", function(req, res){
res.render("bio2s.ejs", {       name: "Tom Mustermann",
                                birthyear: 1992,
                                career: "Node-Developer"      });
});
```

- **HTML-Template :**

```
<!DOCTYPE html>
<html>  <head></head><body>
Hi <%= name %>!
You were born in <%= birthyear %>, so that means you are
 <%= (new Date()).getFullYear() - birthyear %> years old.   <p><p>
<% if (career) { -%>
        <%= career %> is a cool career!
<% } else { -%>     Haven't started a career yet? That's also cool.   <% } -%>
</body>  </html>
```



Express – API´s

- Express erlaubt auch API´s zur Bereitstellung von Schnittstellen (analog WS)
  Detail unter **http://expressjs.com/de/api.html**

```
var express = require('express');
var app = express();
var Angebot = [
  { author : 'Dan Brown', text : "Sakrileg"},
  { author : 'J.K Rowling', text : "Harry Potter und der Feuerkelch"},
  { author : 'Simon Becket', text : "Kalte Asche"},
  { author : 'Richard David Precht', text : "Wer bin ich und wenn ja wie viele?"}
];
app.get('/', function(req, res) {
  res.json(Angebot);   // Bereitstellung als JSON-Daten
});

var server = app.listen(3000, function(){
  console.log(„Der Server ist nun betriebsbereit“);
});
```

Express – API´s mit Filterfunktion über URL

- Express erlaubt auch API´s zur Bereitstellung von Schnittstellen (analog WS)
  Detail unter http://expressjs.com/de/api.html

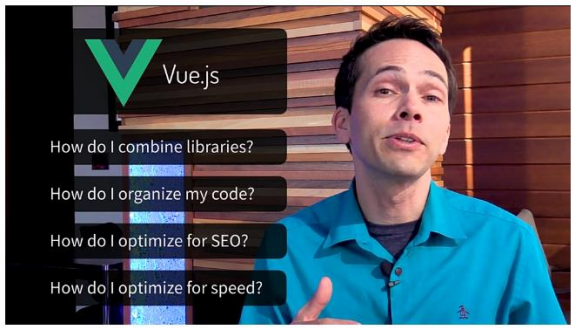
```
…
app.get('/Angebot/:id', function(req, res) {
  if(Angebot.length <= req.params.id || req.params.id < 0) {

    res.statusCode = 404;
    return res.send('Error 404: Keine Angebote gefunden.');
  }
var Hilfsvariable = Angebot[req.params.id];
  res.json(Hilfsvariable);
});
….
```

- Damit kann dann das 2. Angebot abgerufen werden mit

  http://localhost:3000/Angebot/2

ergibt als Antwort den 2. Datensatz :
{"author":"Simon Becket","text":"Kalte Asche"}

Das Vue-orientierte Serverframework **Nuxt**



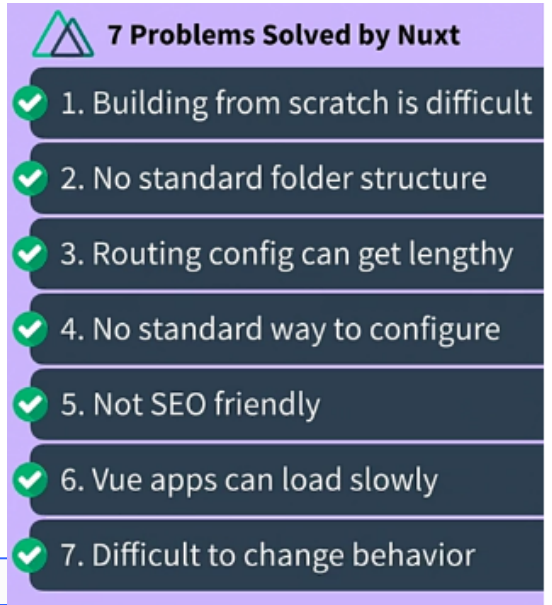
Video produced by Vue Mastery, download their free Nuxt Cheat Sheet.

- Bei einer Nutzung des JS-Frameworks Vue.js für die Frontend-Entwicklung im Browser ist adäquat zu Express (welches natürlich auch mit Vue verwendet kann) noch das Vue-spezifischere Serverframework **Nuxt** verfügbar
- Homepage: https://nuxtjs.org/ (sehr gute Einführung mit Video-auf HP! =>)
- 2016 entwickelt durch die Brüder Chopin (Frankreich)
- wieder sehr gute Dokus ! https://nuxtjs.org/guide

Nuxt.js includes the following to create a rich web application (RIA) development:

- Vue 2 (see lecture on Vue 2)
- Vue Router (for front-side routing !)
- Vuex (included only when using the store option)
- Vue Server Renderer (excluded when using mode: 'spa')
- Vue Meta (Manage HTML metadata)

A total of only 57kB min+gzip (60kB with Vuex) !



Das Vue-orientierte Serverframework **Nuxt – Main Topics**

Main ideas / topics of **Nuxt** from https://nuxtjs.org/guide :

- **Server Rendered (Universal SSR)**
  - generates HTML-content already on the server and delivers it to the client browser
  - views are then "rehydrated" to full SPA functionality
- **Single Page Applications (SPA)**
  - one file set “simulates” a full Web application with dozens of URLs
  - If you need static hosting for your applications, you can simply use SPA mode using **nuxt --spa**. It generates a set of static HTM/CSS/JS – files, which can be uploaded to an traditional web server. There is no need of Node.js in this case !
- **Static Generated (Pre Rendering)**
  - When building your application, it will generate the HTML for every one of your routes and store it in a file. This improves readability by search engines -> better SEO !
- **Testing** (End-to-End Testing with the AVA-framework)
- **Code analysis and code management** (with ESLint and Prettier)

Actual trends in Vue.js and Nuxt:

- Usage of TypeScript (correct var-Typing, better OO-Support -> VL TS)
- if interested, please visit “Vue.js-workshop” in next semester (after Vue.js-conf)

Zusammenfassung zu NodeJS und Express

Die neuen Framework im NodeJS-Umfeld sind auf dem Weg zu einem Full-Stack-Entwicklungssystem bestehend aus :

- **NodeJS als Serverlösung (N)**
- **Express als Middleware-Framework (E)**
- **Datenbanken wie die dokumentenorientierte DB MongoDB (M)**
- **und Browser-Frontends wie Angular (A)**

Die obige Kombination wird immer häufiger als **MEAN-Stack** bezeichnet und deckt alle wesentlichen Komponenten für die zukünftige Webentwicklung ab.

In ähnlicher Weise kombiniert auch das Nuxt-Framework auf der Basis von Vue einen NodeJS-Stack mit der Vue.js-Infrastruktur.

Kombiniert mit der exzellenten Performance von NodeJS sind sehr effektive Webserverlösungen möglich und könnten in der Zukunft den alten XAMP-Stack (mit Apache / PHP und MySQL) ablösen !

Bei neuen Entwicklungen sollte unbedingt der aktuelle Stand geprüft werden!