

Maturaarbeit

Performance Vergleich von Webservern & -browsern

Melvin Suter (AME Klasse 26 - 27.02.2019)

Betreuungsperson Michel Hauswirth

Inhaltsverzeichnis

Teil 1 - Einleitung

1	Thematik & Leitfragen.....	2
1.1	Leitfragen.....	2
1.2	Verwendungsbereich	2
1.3	Begriffs-Definition.....	2
1.3.1	Runtime	3
1.3.2	Response-Time	3

Teil 2 - Hardware & Software

2	Hardware.....	5
2.1	VM-Hosts	5
2.1.1	Oer Provisioning.....	6
2.2	VM Ressourcen	6
3	Software	7
3.1	OS.....	7
3.2	Webserver.....	7
3.2.1	Apache	7
3.2.2	nginx	7
3.3	Datenbank.....	8
3.4	Webbrowser	8
4	Übersicht	8

Teil 3 - Installation

5	OS Installation.....	11
5.1	dk-matura-01	11
5.1.1	Grund Konfiguration	12
5.1.2	Swap File und RAM	14
5.1.3	PHP Installation.....	15

5.1.4	Software.....	15
5.2	dk-matura-02	15
5.3	dk-matura-03	16
5.3.1	SSH.....	17
5.3.2	Swap File und RAM	18
5.3.3	PHP Instalaltion.....	18
5.3.4	Software.....	18
6	Software Installation.....	19
6.1	Datenbank.....	19
6.2	Webserver.....	19
6.2.1	dk-matura-01	19
6.2.2	dk-matura-02 & dk-matura-03	20
 Teil 4 - Runtime Script		
7	Grundsatz	23
8	Code	23
lib/Helper	23
ExportHelper.php	24
TestHelper.php	25
lib/Models	26
Test.php	26
TestSerie.php	27
Timer.php	28
lib/Tests	28
ExampleTest.php	28
lib/App.php	30
lib/boot.php	30
lib/config.php	30
public/index.php	30
9	Tests	31
9.1	Math	31

9.2	Loop	32
9.3	DB.....	33
9.4	FileSystem	34
9.5	BigFile.....	35
9.6	URL Call.....	36
10	Resultat	37
10.1	dk-matura-01	37
10.2	dk-matura-02	41
10.3	dk-matura-03	43
10.4	Vergleich.....	45
10.4.1	Datenbank.....	46
10.4.2	File System	47
10.4.3	URL Call	47

Teil 5 - Browser Script

11	Grundsatz	49
12	Code	49
12.1	index.html	49
12.2	jquery-3.3.1.min.js.....	50
12.3	sciprt.js	50
12.3.1	Loop Test.....	51
12.3.2	Console Test.....	51
12.3.3	reqres Call Test	52
12.3.4	Self Call Test	52
12.3.5	DOM Test	52
12.4	test.txt	52
13	Resultat	53
13.1	Daten.....	53
13.1.1	Chrome	53
13.1.2	Firefox	53
13.1.3	Safari	53

13.2 Interpretation	53
13.2.1 Loop & DOM	53
13.2.2 Console	54
13.2.3 Reqres & Self.....	54
13.3 Fazit.....	55

Teil 6 - Fazit

14 Resultate	57
14.1.1 Gibt es einen effektiv messbaren Unterschied der Zeiten?.....	57
14.1.2 Gibt es einen Zusammenhang zwischen Script-Runtime und Response-Time?	57
14.1.3 Hängt die Durchschnitts-Laufzeit von den minimalen und maximalen Zeiten ab?	57
15 Arbeitstechnik.....	58
15.1 Probleme.....	58
15.2 Erkenntnisse	58
16 Persönlich	59

Teil 7 - Anhang

17 Quellen	61
18 Anhang.....	62

TEIL 1

Einleitung

1 Thematik & Leitfragen

Meine Maturaarbeit behandelt das Thema, welcher Webserver die besten Zeiten mit einem selbst programmierten Benchmark-Scripts erreicht. Hauptsächlich interessieren mich hier die Runtime & Reponse-Time. Folgend sind meine 3 Leitfragen.

1.1 Leitfragen

Gibt es einen effektiv messbaren Unterschied der Zeiten?

Dies ist die primäre Leitfrage meiner Arbeit. Hier geht es darum, ob mit dem Unterschied in den gemessenen Zeiten effektiv ein Rückschluss zu führen ist, welcher Webserver die bessere Performance liefert.

Gibt es einen Zusammenhang zwischen Script-Runtime und Response-Time?

Unter dieser Leitfrage werde ich die beiden Messdaten untersuchen und vergleichen.

Hängt die Durchschnitts-Laufzeit von den Minimalen und Maximalen Zeiten ab?

Mit dieser Leitfrage möchte ich herausfinden, ob es einen Zusammenhang zwischen durchschnittlicher Performance & Peaks gibt.

1.2 Verwendungsbereich

Diese Arbeit ist unter der Annahme geschrieben, dass der Leser Grundkenntnisse im Bereich Informatik hat und Fachbegriffe, sowie gängige Verfahren kennt und versteht. Die wichtigsten Begriffe sind im Glossar genauer definiert.

1.3 Begriffs-Definition

Ein komplettes Glossar ist, hinsichtlich des Verwendungsbereiches, meiner Meinung nach nicht notwendig. Fachbegriffe sollten bekannt sein, oder können nachgelesen werden. Eine Ausnahme bilden die beiden Begriffe "Runtime" und "Response-Time", da diese vom Aufbau des Tests beeinflusst werden. Deshalb folgend die Definitionen.

Zum besseren Verständnis wurde eine Grafik dazu erstellt.

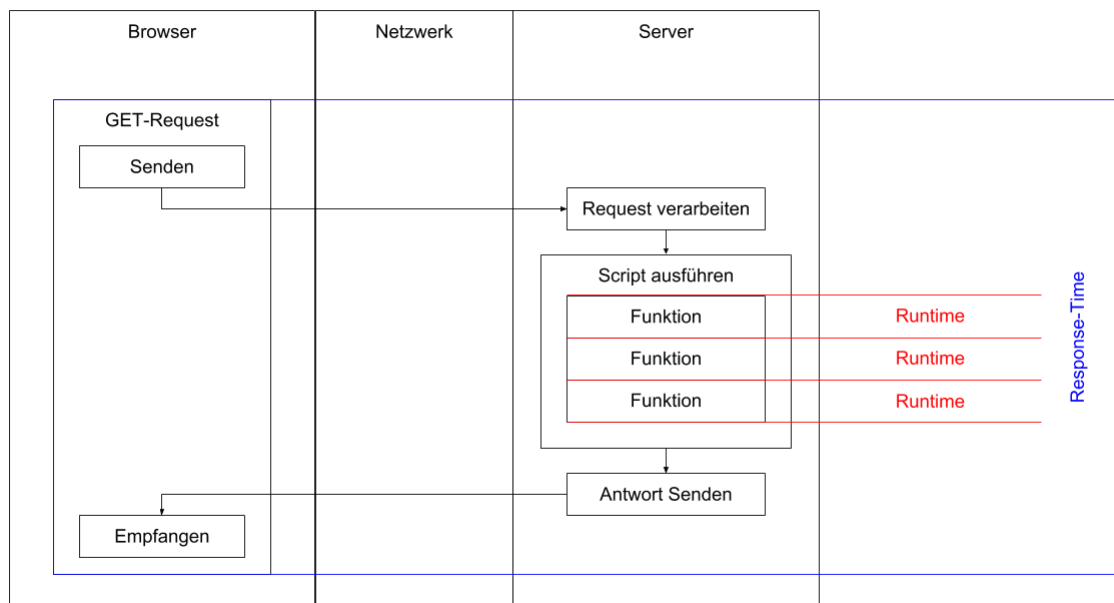


Abbildung 1 Runtime & Response-Time

1.3.1 Runtime

Als Runtime oder Laufzeit wird die Zeit gemessen, die eine Software benötigt, um eine Funktion auszuführen.

1.3.2 Response-Time

Die Response-Time oder auch Antwortzeit ist die Zeit, die der Webserver benötigt um einen Request zu empfangen, zu verarbeiten und die Antwort zu senden.

TEIL 2

Hardware & Software

2 Hardware

2.1 VM-Hosts

Da mehrere Webserver verwendet werden, ist es unabdingbar diese zu virtualisieren. Die Verwendung einer Software wie "VirtualBox" auf einer normalen Workstation ist jedoch nicht zu empfehlen, da die Leistung der VMs somit abhängig von der Auslastung der Workstation wäre. Deshalb werden in dieser Arbeit die Webserver im Homelab des Autors eingerichtet. Hier die Eck-Daten der VM-Hosts:

Komponente	Spezifikation
Name	dk-host-01
Model	IBM iDataPlex dx360 M3
OS	ESX 6.5.0
CPU	2x Intel Xeon L5520 2.27 GHz 8C/16T
RAM	32GB
Disk	500GB SSD
Netzwerk	2x 1 Gbit/s

Komponente	Spezifikation
Name	dk-host-02
Model	IBM iDataPlex dx360 M3
OS	ESX 6.5.0
CPU	2x Intel Xeon L5520 2.27 GHz 8C/16T
RAM	24GB
Disk	1TB HDD
Netzwerk	2x 1 Gbit/s

Nun gilt es noch einen der beiden Hosts auszuwählen. Mit einem kurzen Blick auf die bereits installierten VMs sollte dies keine schwierige Entscheidung sein:









Name ↑	Provisioned Space	Memory Size	CPUs
 dk-backup-01	44.12 GB	4 GB	2
 dk-proxy-01	17.61 GB	1.5 GB	1
 dk-puppet-01	18.11 GB	2 GB	4
 dk-remote-01	18.11 GB	2 GB	2
 dk-web-01	51.71 GB	1.5 GB	2
 dk-web-02	25.11 GB	1 GB	1
 dk-ws-01	121.22 GB	6 GB	6
 dk-ws-02	36.19 GB	4 GB	2

Abbildung 2 VMs auf dk-host-01






Name ↑	Provisioned Space	Memory Size	CPUs
 dk-dc-01	43.11 GB	3 GB	4
 dk-glt-01	102.11 GB	6 GB	4
 dk-monitor-01	27.11 GB	2 GB	3
 dk-pihole-01	11.11 GB	1 GB	1
 dk-vcenter-01	329.7 GB	10 GB	2

Abbildung 3 VMs auf dk-host-02

Wie zu sehen ist, hat der Host dk-host-02 bereits 22GB von 24GB RAM verwendet, jedoch nur 14 von 16 vCPU's. Der Host dk-host-01 hingegen hat nur 22GB von 32GB RAM verwendet, jedoch mit 20 von 16 vCPU's diese bereits "over provisioned". Da es jedoch weniger problematisch ist vCPU's "over provisioned" zu haben, als RAM, werden die Webserver auf dk-host-01 installiert. Zudem kommt hier zu Hilfe, dass auf dk-host-01 nicht so wichtige VMs laufen wie auf dem anderen Host. Somit kann auch ein Server während den Tests deaktiviert werden.

2.1.1 Over Provisioning

Der Verständlichkeit halber von Over Provisioning hier eine kurze Erklärung:

Wen man einer VM mehr logische Ressourcen zuspricht, als physisch auf dem Host vorhanden sind, nennt man die Over Provisioning. Dies hat bei verschiedenen Ressourcen unterschiedliche Auswirkungen. Bei vCPUs macht dies nicht viel aus, da, sehr einfach erklärt, dies dazu führt, dass pro Cycle mehr Aktionen für eine VM durchgeführt werden. Somit ist ein over provisioned CPU kein grossers Hindernis für den Host.

Bei over provisioned RAM muss der Host auf dem lokalen Datenträger ein Swap File (eine Auslagerungs Datei) anlegen. Da eine Harddisk oder auch eine SSD viel langsamer und durch die virtuellen HDDs der VMs auch starker ausgelastet sind als normaler RAM, verlangsamt dies das gesamte System enorm.

2.2 VM Ressourcen

Nun stellt sich noch die Frage, wieviele Ressourcen man den Test-VMs zuweisen soll. Dies kann einen sehr grossen Einfluss auf die Test-Resultate haben. Mehr Ressourcen bedeutet (meist) auch schnellere Runtime. Doch muss man aufpassen, da es Webserver gibt, die mit mehr Ressourcen schneller sind als seine Konkurrenten, mit weniger jedoch langsamer. Da ich persönlich oft sehr kleine Cloud-Instanzen verwende und dies für einen grossen Teil der VPS und "privaten" Webservern zutrifft, wird mit wenig Ressourcen vorgegangen. Folgend die genauen Spezifikationen:

Komponente	Spezifikation
CPU	1 vCPU auf 1 Socket
RAM	512MB
Disk	10GB Thin-Provisioned
Netzwerk	1 Adapter in "VM Network"

3 Software

3.1 OS

Schlussendlich stellt sich hier noch die Frage, welche Software zu verwenden ist. Wenn man die OS Statistiken (von w3techs (w3techs, Usage of operating systems for websites, kein Datum)) für Webserver zur Hand nimmt erkennt man, dass Unix einen Marktanteil von über 65% hat. (w3techs (w3techs, Usage statistics and market share of Unix for websites, kein Datum) unterteilt Unix in Linux, BSD, etc. und Unknown. Der Einfachheit halber wird Unknown zu Linux dazugezählt, da dies grösstenteils der Fall sein wird.) Deshalb wird auch in dieser Arbeit eine Linux-Distribution verwendet. Durch die guten persönlichen Erfahrungen des Autors mit CentOS und weil es von RHEL, der Linux Distribution Red Hat mit Enterprise-Support abstammt, wird es für die Test-Maschinen verwendet. Zudem wird noch eine weitere virtuelle Maschine installiert mit der Distribution "Ubuntu", da sie laut w3techs (w3techs, Usage statistics and market share of Linux for websites, kein Datum) den grössten Marktanteil (38.2%) unter den Linux-Distributionen besitzt.

3.2 Webserver

3.2.1 Apache

Apache ist der wohl bekannteste Webserver. Er überzeugt durch eine vergleichsweise einfache Konfiguration, mit sehr viel Möglichkeiten. Er wird von kleinen bis grossen Webservern verwendet und besitzt einen Marktanteil von 44.5%. (w3techs, Usage of web servers for websites, kein Datum)

Persönlich habe ich gute Erfahrungen mit Apache gesammelt und verwende ihn momentan auf meinen drei Webservern.

3.2.2 nginx

Nginx ist im Vergleich zu Apache ein Webserver für eher fortgeschrittene Benutzer. Er besitzt eine etwas komplexere Konfiguration, was für viele neue User abschreckend wirkt, weshalb er mit ca. 40% Marktanteil hinter Apache liegt.

Ich habe bereits mit nginx als Webserver gearbeitet, verwende ihn zur Zeit jedoch nur als Reverse-Proxy.

3.3 Datenbank

Ein wichtiger Teil jedes Webapps ist die Datenbank. Da ich bisher nur Erfahrung mit MySQL und MariaDB gemacht habe, werde ich auch die beiden verwenden. MySQL gehört zu den Top 3 Datenbank Engines. (StackOverflow, Developer Survey Results 2018, kein Datum) (DB-Engines, kein Datum) MariaDB ist ein direkter Open Source Ableger von MySQL und gilt als gleich schnell, wenn nicht sogar schneller.

3.4 Webbrowser

Als Webbrowser wird curl, Firefox, Chrome und Safari verwendet. Curl dient dem Ausführen des Script- Runtime Tests, was von einem Server im Homelab durchgeführt wird. Für Chrome, Firefox und Safari wird das MacBook des Autors verwendet.

4 Übersicht

Um die nachfolgende Installation verständlicher zu machen, wurde eine grafische Übersicht über die Installtion erstellt. Die gleichen Komponenten wie im dk-matura-01 aufgezeigt, sind auch in dk-matura-02 und dk-matura-03 vorhanden.

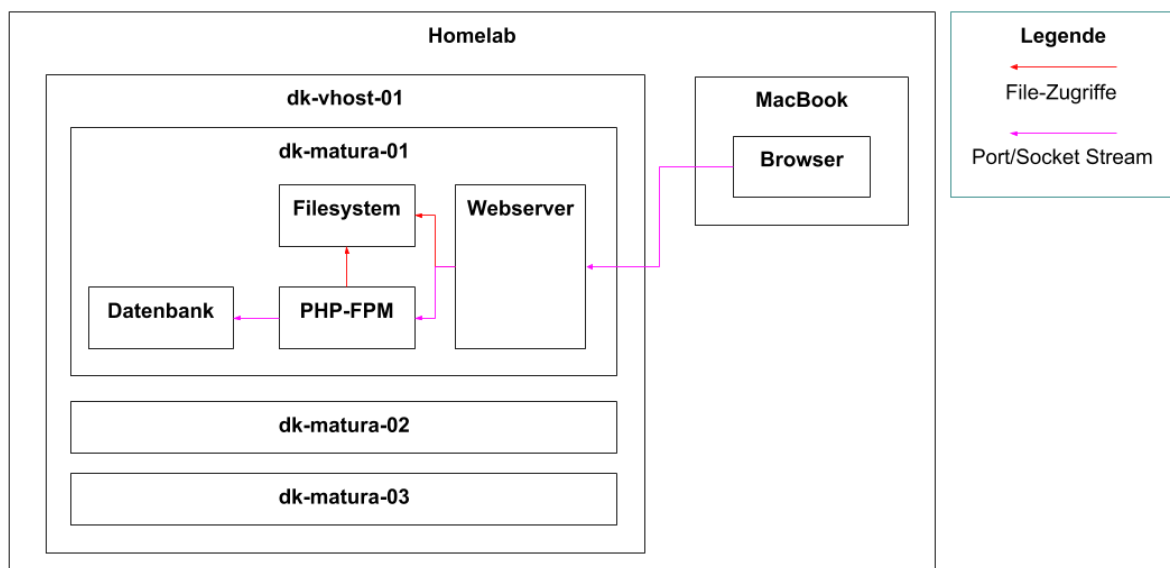


Abbildung 4 Übersicht Installation

Hier eine kurze Aufstellung, was die Pfeile bedeuten und in welcher Reihenfolge diese zu lesen sind:

1. Der Browser öffnet einen Request über einen TCP-Stream auf Port 80 auf den Webserver
2. Der Webbrowser liest im Filesystem die aufgerufene Datei aus
3. Der Webbrowser übergibt die gelesene Datei über einen Socket-Strem dem PHP-FPM
4. PHP-FPM kommuniziert falls nötig über einen Socket-Stream mit der Datenbank
5. PHP-FPM antwortet über den vom Webserver geöffneten Socket-Stream mit dem berechneten Content

6. Der Webserver antwortet dem Browser über den TCP-Stream mit dem generierten Content

TEIL 3

Installation

5 OS Installation

5.1 dk-matura-01

Die erste VM die in Angriff genommen wird, hat folgende Eckdaten:

Komponente	Spezifikation
Hostname	dk-remote-01.home.damon.id
IP	192.168.1.60
OS	CentOS 7
Webserver	nginx
Datenbank	MariaDB
PHP Version	7.2

Man beginnt erstmals mit der Erstellung der Virtuall-Machine. Dazu wird im vCenter (der zentralen Steuerung der VM-Hosts) die VM angelegt. Folgend die Einstellungen:

Ready to complete
Click Finish to start creation.

Provisioning type	Create a new virtual machine
Virtual machine name	dk-matura-01
Folder	Datacenter
Host	192.168.1.21
Datastore	datastore1 (2)
Guest OS name	CentOS 7 (64-bit)
Virtualization Based Security	Disabled
CPUs	1
Memory	1 GB
NICs	1
NIC 1 network	VM Network
NIC 1 type	VMXNET 3
SCSI controller 1	VMware Paravirtual
Create hard disk 1	New virtual disk
Capacity	10 GB
Datastore	datastore1 (2)
VM storage policy	Datastore Default
Virtual device node	SCSI(0:0)
Mode	Dependent

Abbildung 5 Konfigurations-Übersicht für dk-matura-01

Hier gilt es zu bemerken, dass 1 GB RAM zugewiesen sind. Dies ist der Fall, da CentOS 7 minimal seit Neuem eine Mindestanforderung von 1 GB RAM hat. Zudem ist der VM das ISO CentOS7_minimal_1804.iso angehängt. (Dies wurde bereits heruntergeladen und auf dem NAS abgelegt.)

Die Erst-Installation ist mit Hilfe der VMware Remote Console vorzunehmen. Dabei wird erstmals alles auf den Standard-Einstellungen belassen, ausser dem Tastatur-Layout und der Zeitzone.

5.1.1 Grund Konfiguration

Nun gilt es die VM zu konfigurieren. Im ersten Schritt sollte man den Hostnamen anpassen, den Netzwerkanschluss vorbereiten und SSH konfigurieren. Als erstes der Hostnamen, dazu gibt man folgendes in der Kommandozeile ein:

```
# hostname dk-matura-01.home.damon.id
# echo "dk-matura-01.home.damon.id" > /etc/hostname
# echo "127.0.0.1    dk-matura-01.home.damon.id" >> /etc/hosts
# echo "127.0.0.1    dk-matura-01" >> /etc/hosts
```

Der Command `hostname dk-ma...` ändert den Hostnamen in der Laufzeit. Mit dem Command `echo "dk-ma..." > /etc/hostname` ändert sich der Inhalt dieser Datei zum neuen Hostname, damit dieser beim nächsten Startvorgang ebenfalls angenommen wird. Zuletzt wird der Datei `/etc/hosts` zwei Zeilen hinzugefügt, damit bei der DNS Abfrage auf `dk-remote-01` und dessen FQDN `dk-remote-01.home.damon.id` die IP Adresse `127.0.0.1` (also localhost) zurückgegeben wird.

Als nächstes kommt das Netzwerk dran. Dazu editiert man die `ifcfg` Datei des Network-Adapters mit folgendem Befehl:

```
# vi /etc/sysconfig/network-scripts/ifcfg-ens192
```

Darin setzt man folgende Einstellungen:

```
BOOTPROTO=static
IPADDR=192.168.1.60
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
DNS1=192.168.1.52
DNS2=192.168.1.1
ONBOOT=yes
ZONE=public
```

Die Einstellung `BOOTPROTO` definiert wie die Netzwerkkarte konfiguriert ist. Den Servern wird eine statisch IP zugeteilt, deshalb ist die Option hier auf `static`. Die Parameter `IPADDR` und `NETMASK` definieren die IP Adresse `192.168.1.60/24` für das Interface. Als `Gateway` und `sekundären DNS` wird der Router (`192.168.1.1`) angegeben. Als `primären DNS` dient der lokale PiHole DNS Server. Zuletzt wird mit `ONBOOT=YES` definiert, dass die Netzwerkkarte beim Startvorgang des Servers aktiviert werden soll und mit `ZONE=public` in welcher Firewall-Zone sie sich befindet.

Nun muss das Interface mit `ifup ens192` noch gestartet werden.

Als nächstes gilt es SSH für den Fernzugriff einzurichten. Dazu führt man folgende Commands aus:

```
# ssh-keygen -t rsa -b 4096 -C "default" -P '' -f ~/.ssh/root_key
# cat ~/.ssh/root_key.pub > ~/.ssh/authorized_keys
# cat ~/.ssh/root_key
-----BEGIN RSA PRIVATE KEY-----
...
-----END RSA PRIVATE KEY-----
# sed -i -e 's/SELINUX=.*SELINUX=disabled/' /etc/selinux/config
# setenforce 0
#
# firewall-cmd --zone=public --add-port=2121/tcp --permanent
# firewall-cmd --reload
#
# sed -i 's/#Port 22/Port 2121/' /etc/ssh/sshd_config
# sed -i 's/PasswordAuthentication yes/ PasswordAuthentication no/' /etc/ssh/sshd_config
# service sshd restart
```

`ssh-keygen` generiert einen SSH Key, mit `rsa` und einer Keylänge von `4096`, dieser wird mit `cat ...key.pub > ...authorized_keys` für den aktuellen User (`root`) aktiviert und anschliessend mit `cat ...key` ausgegeben. Die Ausgabe sollte gesichert werden.

Anschliessend wird mit dem `sed` Befehl SELinux für den nächsten Neustart deaktiviert und mit `setenforce 0` geschieht das gleiche für die aktuelle Laufzeit. SELinux ist ein Sicherheitsfeature der RHEL basierenden Linux Distributionen. Es dient zum Auditing, der Ressourcen und Zugriffskontrolle. In der Kombination mit einem Webserver kann sich dies jedoch oft als sehr störend auszeichnen. Um weiteren, unnötigen Problemen aus dem Weg zu gehen, wird die Funktion hier deaktiviert.

Nun wird der Port 2121 in der Firewall freigegeben und die Regel angewendet.

Schlussendlich wird der SSH Port mit einem weiteren `sed` Befehl von 22 auf 2121 geändert und die Passwort Authentifizierung deaktiviert. Dies wird aus Sicherheitsgründen gemacht, da Passwort-Authentifizierung im Vergleich zu einem SSH-Key sehr unsicher ist.

Damit funktioniert der Zugriff jedoch noch nicht. Auf dem Domain Controller (dk-dc-01) muss nun noch ein A Record für diesen Server erstellt werden:




 dk-matura-01	Host (A)	192.168.1.60
 dk-matura-02	Host (A)	192.168.1.61
 dk-matura-03	Host (A)	192.168.1.62

Abbildung 6 DNS Record auf dk-dc-01

Weil die anderen beiden Server ebenfalls einen solchen Record benötigen werden, werden diese auch gleich erstellt.

Schlussendlich muss nur noch die SSH Verbindung im SSH Client konfiguriert werden. Hier wird für all meine Remote-Zugriffe Apache Guacamole verwendet. Hier die Settings dazu:

EDIT CONNECTION

Name: dk-matura-01

Location: SSH

Protocol: SSH

CONCURRENCY LIMITS

Maximum number of connections: 5

Maximum number of connections per user: 5

Abbildung 7 Guacamole Settings 1

Die maximalen gleichzeitig Verbindungen wird auf 5 gestellt, da es vorkommen kann, dass Guacamole eine Session nicht sauber beendet und deshalb eine "zweite" Verbindung aufgebaut wird. Folgend wird der Host, Port, Username und SSH Key definiert, sowie einiger Display-Einstellungen, die vom Autor bevorzugt werden.

PARAMETERS

Network

Hostname: dk-matura-01

Port: 2121

Authentication

Username: root

Password:

Private key: -----BEGIN RSA PRIVATE KEY-----

Passphrase:

Display

Color scheme: White on black

Font name: Roboto Mono

Font size: 10

Read-only:

Abbildung 8 Guacamole Settings 2

5.1.2 Swap File und RAM

Die VM ist zur Zeit mit 1GB RAM konfiguriert. Dies sollte auf 512MB geändert werden. Damit dabei nicht Probleme entstehen, muss zuvor ein Swapfile erstellt werden. Diese Datei dient als Auslagerungsdatei, wenn die RAM voll sind. Die Erstellung lässt sich mit folgenden Befehlen machen.

```

# dd if=/dev/zero of=/swapfile bs=1M count=2048
# chown root:root /swapfile
# chmod 0600 /swapfile
# mkswap /swapfile
# swapon /swapfile
# echo /swapfile none swap sw 0 0 >> /etc/fstab
# swapoff /dev/dm-1
# sed -i 's;/dev/mapper/centos-swap swap;#/dev/mapper/centos-swap swap;' /etc/fstab
# swapon -s
Filename Type.      Size.    Used. Priority
/swapfile file      2097148 0        -2

```

Dieses Code-Snippet erstellt eine leere Datei mit einer Grösse von 2GB, setzt die korrekten Rechte, bereitet sie vor und aktiviert sie. Schlussendlich wird noch die System-Default Swap-Partition deaktiviert und die aktuelle Config ausgegeben.

Nun muss die VM nur noch heruntergefahren, im vCenter angepasst und wieder gestartet werden.



Abbildung 9 dk-matura-01 RAM

5.1.3 PHP Installation

Da auf beiden Maschinen die gleiche PHP Version benötigt wird, wird diese bereits jetzt installiert:

```
# yum install -y epel-release yum-utils http://rpms.remirepo.net/enterprise/remi-release-7.rpm
# yum-config-manager --enable remi-php72
# yum update -y
# yum install -y php php-fpm php-pdo php-pdo_mysql
```

Als erstes muss das **EPEL** (Extra Packages for Enterprise Linux) und das **remi-php72** Repo aktiviert werden, damit **yum** überhaupt ein PHP Packet findet und dies dann auch die richtige Version hat. Für die Aktivierung von **remi-php72** wird das Packet **yum-utils** benötigt, deshalb wird dies mit **EPEL** zusammen installiert. Anschliessend muss **yum** mit Hilfe des **update** Commands die Repo-Liste und alle installierten Pakete aktualisieren und php ist bereits zur Installation.

5.1.4 Software

Nun installiert und konfiguriert man noch ein paar von den häufig verwendeten Tools:

```
# yum install -y wget vim git open-vm-tools
# git config --global user.name "administrator"
# git config --global user.email "gitlab@damon.ch "
# git config --global color.ui true
# git config --global format.pretty oneline
# git config --global push.default simple
# ssh-keygen -t rsa -b 4096 -C "matura" -P '' -f ~/.ssh/gitlab
# cat ~/.ssh/gitlab.pub
ssh-rsa AAAA... == matura
```

Das wichtigste hier ist wohl **git**, da der Source-Code auf dem lokalen Gitlab gespeichert ist. **open-vm-tools** wird benötigt für einen optimalen Betrieb auf ESX.

5.2 dk-matura-02

Die Installation vom zweiten Server ist sehr viel einfacher. Man kann einfach den bestehenden Server im vCenter klonen und anpassen. Zuerst einmal die Spezifikationen:

Komponente	Spezifikation
Hostname	dk-remote-02.home.damon.id
IP	192.168.1.61
OS	CentOS 7
Webserver	Apache
Datenbank	MariaDB
PHP Version	7.2

Um diese VM zu erstellen, wählt man die VM dk-matura-01 aus und klonst sie.

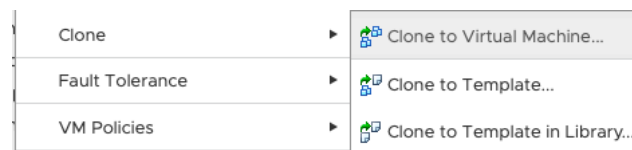


Abbildung 10 VM Klonen

Nach dem dieser Vorgang abgeschlossen ist, kann man die VM starten und den Hostnamen und die IP ändern. Um dies einfacher zu machen, kann man die VM dk-matura-01 zuerst herunterfahren, die neue VM starten und dann per SSH darauf zugreifen.

```
# hostname dk-matura-02.home.damon.id
# echo "dk-matura-02.home.damon.id" > /etc/hostname
# echo "127.0.0.1 dk-matura-02.home.damon.id" >> /etc/hosts
# echo "127.0.0.1 dk-matura-02" >> /etc/hosts
# sed -i 's/IPADDR=192.168.1.60/IPADDR=192.168.1.61/' /etc/sysconfig/network-scripts/ifcfg-ens192
# reboot
```

5.3 dk-matura-03

Komponente	Spezifikation
Hostname	dk-remote-03.home.damon.id
IP	192.168.1.62
OS	Ubuntu 18.10
Webserver	Apache
Datenbank	MySQL
PHP Version	7.2

Für die VM dk-matura-03 wird die Distribution Ubuntu verwendet. Somit kann die VM nicht einfach von einer Vorhanden geklont werden. Als erstes erstelle man im vCenter wieder eine VM und mountet das Ubuntu ISO.

Provisioning type	Create a new virtual machine
Virtual machine name	dk-matura-03
Folder	Datacenter
Host	192.168.1.21
Datastore	datastore1 (2)
Guest OS name	Ubuntu Linux (64-bit)
Virtualization Based Security	Disabled
CPUs	1
Memory	1 GB
NICs	1
NIC 1 network	VM Network
NIC 1 type	VMXNET 3
SCSI controller 1	LSI Logic Parallel
Create hard disk 1	New virtual disk
Capacity	10 GB
Datastore	datastore1 (2)
VM storage policy	Datastore Default
Virtual device node	SCSI(0:0)
Mode	Dependent

Abbildung 11 Konfigurations-Übersicht für dk-matura-03

Durch den Installer wird bereits die IP gesetzt und der Hostname vergeben. Von den Grundeinstellungen bleibt also nur noch SSH übrig. Jedoch stellt sich dies schon als kleine Herausforderung dar, da Ubuntu während der Installation die Möglichkeit bietet ein Tastaturlayout auszuwählen, diese Einstellung jedoch keine Auswirkung hat. Deshalb muss zuerst `console-data` installiert und die Schweizer Tastatur aktiviert werden:

```
# sudo apt-get install console-data
# sudo loadkeys sg-latin1
```

5.3.1 SSH

Hier stellt man wie bei den beiden VMs zuvor auf die Key-Authentifizierung um und wechselt auf Port 2121. Was hier noch speziell dazu kommt ist, dass das Login als `root` manuell aktiviert werden muss, da Ubuntu dies Standardmässig deaktiviert.

```
# sudo su
# ssh-keygen -t rsa -b 4096 -C "default" -P '' -f ~/.ssh/root_key
# cat ~/.ssh/root_key.pub > ~/.ssh/authorized_keys
# cat ~/.ssh/root_key
-----BEGIN RSA PRIVATE KEY-----
...
-----END RSA PRIVATE KEY-----
# passwd root
# passwd -u root
# sed -i 's/#Port 22/Port 2121/' /etc/ssh/sshd_config
# sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config
# sed -i 's/#PubkeyAuthentication yes/PubkeyAuthentication yes/' /etc/ssh/sshd_config
# sed -i 's/#PasswordAuthentication yes/PasswordAuthentication yes/' /etc/ssh/sshd_config
# service sshd restart
```

Im Vergleich zur Konfiguration von CentOS gibt es hier nur paar wenige Änderungen:

1. Zu Beginn muss `sudo su` ausgeführt werden, um als root zu arbeiten
2. Dem User `root` muss ein Passwort gesetzt & dann aktiviert werden (mit `passwd`)
3. Die Einstellungen `PermitRootLogin` und `PubkeyAuthentication` müssen gesetzt werden

5.3.2 Swap File und RAM

Ubuntu erstellt bereits von sich aus ein Swapfile mit einer Grösse von 2GB, somit muss ich lediglich die VM herunterfahren, im vCenter auf 512MB RAM stellen und wieder starten.

5.3.3 PHP Instalation

Wie bei der CentOS VM wird auch hier PHP 7.2 installiert:

```
# apt-get update
# apt-get upgrade
# apt-get install -y php php-fpm php7.2-mysql
```

Hier werden alle Updates und PHP Installiert.

5.3.4 Software

Nun installiert und konfiguriert man noch ein paar häufig verwendete Tools, analog zu den vorherigen VMs:

```
# apt-get install -y wget vim git open-vm-tools
# git config --global user.name "administrator"
# git config --global user.email "gitlab@damon.ch "
# git config --global color.ui true
# git config --global format.pretty oneline
# git config --global push.default simple
# ssh-keygen -t rsa -b 4096 -C "matura" -P '' -f ~/.ssh/gitlab
# cat ~/.ssh/gitlab.pub
ssh-rsa AAAA... == matura
# printf "\nHost dk-git-01.home.damon.id\nUser git\nIdentityFile\n~/.ssh/gitlab\nIdentitiesOnly yes" >> ~/.ssh/config
```

6 Software Installation

Nun geht es um die Installation der eigentlichen Software.

6.1 Datenbank

Als erstes installiert man MariaDB auf CentOS:

```
# yum install -y mariadb-server
# systemctl enable mariadb
# service mariadb start
```

Und MySQL auf Ubuntu:

```
# apt-get install -y mysql-server
# systemctl enable mysql
# service mysql start
# service mysql start
# mysql -u root
mysql # GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' IDENTIFIED BY '';
mysql # FLUSH PRIVILEGES;
```

6.2 Webserver

6.2.1 dk-matura-01

Jetzt geht es ums Herzstück des Ganzen, die Webserver. Im ersten Schritt installiert und konfiguriert man **nginx** auf **dk-matura-01**.

```
# yum install -y nginx
# systemctl enable nginx
# service nginx start
# sed -i 's;listen = 127.0.0.1:9000;listen = /run/php-fpm/php-fpm.sock;' /etc/php-fpm.d/www.conf
# sed -i 's;user = apache; user = nginx;' /etc/php-fpm.d/www.conf
# sed -i 's;group = apache; group = nginx;' /etc/php-fpm.d/www.conf
# systemctl enable php-fpm
# service php-fpm start
# vim /etc/nginx/conf.d/httpd.conf
# vim /etc/nginx/nginx.conf
# firewall-cmd --zone=public --add-port=80/tcp --permanent
# firewall-cmd --reload
# sed -i 's/memory_limit = 128M/memory_limit = 1G/' /etc/php.ini
# sed -i 's/max_execution_time = 30/max_execution_time = 3600/' /etc/php.ini
# sed -i 's;;listen.owner = nobody/listen.owner = nginx/' /etc/php-fpm.d/www.conf
# sed -i 's;;listen.group = nobody/listen.group = nginx/' /etc/php-fpm.d/www.conf
# sed -i 's;;listen.mode/listen.mode/' /etc/php-fpm.d/www.conf
# service php-fpm restart
```


Die in der Mitte geöffnete `httpd.conf` Datei sieht wie folgt aus:

```
server {
    listen      80 default_server;
    listen      [::]:80 default_server;
    server_name dk-matura-01.home.damon.id;
    root        /var/www/default/public;
    index       index.php index.html;
    location / {
        try_files $uri $uri/ =404;
    }
    location ~ /\.php$ {
        include fastcgi.conf;
        fastcgi_read_timeout 3600;
        fastcgi_pass unix:/run/php-fpm/php-fpm.sock;
    }
}
```

In der `nginx.conf` Datei muss noch der `server` Bereich gelöscht werden und mit `service nginx restart` der Dienst neu gestartet werden.

Somit ist der Webserver bereit. Fehlt nur noch der Inhalt. Für das Deployment kann lediglich der Source-Code vom gitlab heruntergeladen werden. Dies geschieht mit folgendem Befehl:

```
# mkdir /var/www/default
# cd /var/www/default
# git clone git@dk-git-01.home.damon.id:root/matura.git .
```

Wenn sich etwas im Code ändert, kann das mit `git pull origin master` aktualisiert werden.

6.2.2 dk-matura-02 & dk-matura-03

Da die beiden Server `dk-matura-02` und `dk-matura-03` beide Apache als Webserver verwenden, werden diese gleichzeitig eingerichtet. Als erstes muss der Webserver installiert werden:

```
CentOS # yum install -y httpd
CentOS # systemctl enable httpd
CentOS # service httpd start

Ubuntu # apt-get install -y apache2
Ubuntu # systemctl enable apache2
Ubuntu # service apache2 start
```

```
# mkdir /run/php-fpm/
# mkdir -p /var/www/default/public
# sed -i 's;listen = 127.0.0.1:9000;listen = /run/php-fpm/php-fpm.sock;' /etc/php-fpm.d/www.conf
# sed -i 's;listen.owner = nobody;listen.owner = apache/' /etc/php-fpm.d/www.conf
# sed -i 's;listen.group = nobody;listen.group = apache/' /etc/php-fpm.d/www.conf
# sed -i 's;listen.mode;listen.mode/' /etc/php-fpm.d/www.conf
# sed -i 's/memory_limit = 128M/memory_limit = 1G/' /etc/php.ini
# sed -i 's/max_execution_time = 30/max_execution_time = 3600/' /etc/php.ini
# systemctl enable php-fpm
# service php-fpm start
# firewall-cmd --zone=public --add-port=80/tcp --permanent
# firewall-cmd --reload
# vim /etc/httpd/sites-enabled/000-default.conf
```

Nun gilt es noch die Config-Dateien anzupassen. Die Pfade und einige Einstellungen sind unterschiedlich, deshalb zuerst die Commands für Ubuntu und anschliessend CentOS:

```
<VirtualHost *:80>
    ServerAdmin admin@home.damon.id
    DocumentRoot /var/www/default/public
    ServerName dk-matura-03.home.damon.id
    Timeout 3600

    <FilesMatch \.php$>
        SetHandler "proxy:unix:/run/php-fpm/php-fpm.sock|fcgi://localhost/"
    </FilesMatch>

    ErrorLog /var/log/apache_error.log
    CustomLog /var/log/apache_access.log combined
</VirtualHost>
```

Die auf der letzten Zeile beider Snippets geöffnete Datei sieht sowohl bei Ubuntu wie auch CentOS gleich aus:

Nach einem Neustart des Dienstes httpd bzw. apache2 ist der Webserver soweit bereit. Wie bereits beim Server zuvor, muss jedoch noch die Software vom Gitlab mit dem Command `clone` heruntergeladen werden:

```
# cd /var/www/default
# rm -rf public
# git clone git@dk-git-01.home.damon.id:root/matura.git .
```

TEIL 4

Runtime Script

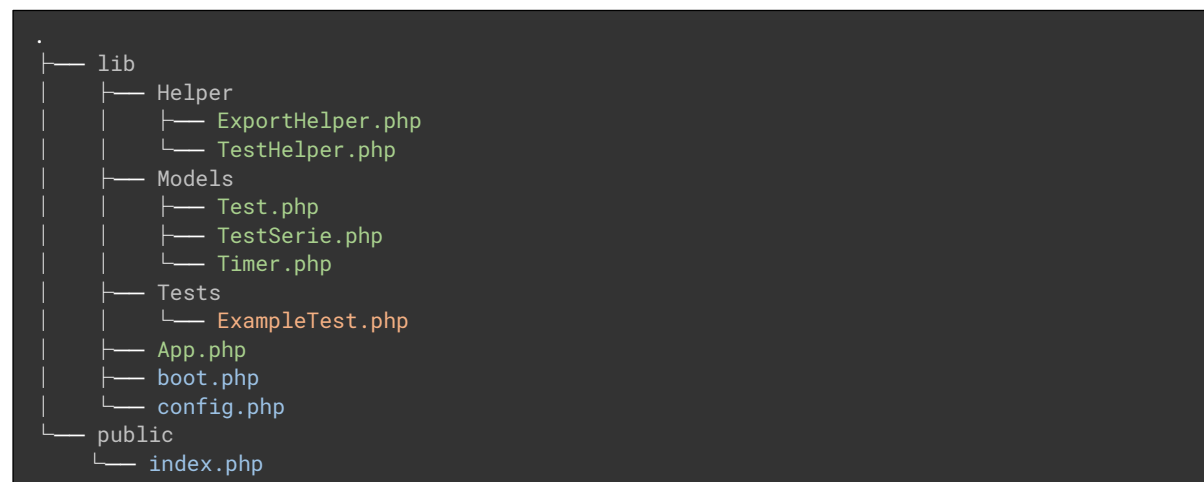
7 Grundsatz

Die Grundidee des Projektes ist folgende:

Es werden mehrere Tests definiert, diese werden je einmal ausgeführt, was eine Test Serie bildet. Solche Test Serien werden mehrere generiert, was dann einen auswertbaren Test-Satz bilden sollte.

8 Code

Dar Code lässt sich am einfachsten an Hand der Verzeichnis-Struktur und der Dateien erklären:



Folgend wird auf die einzelnen Ordner & Dateien eingegangen und in reduzierter Form aufgezeigt um auf gewisse Spezialitäten und Code-Blöcke einzugehen und zu erklären. Den vollständigen Code kann im Anhang besichtigt werden.

lib/Helper

Dieser Ordner enthält Klassen die hauptsächlich zur Unterstützung dienen und hilfreiche Funktionen definieren.

ExportHelper.php

Die Klasse ExportHelper enthält Funktionen um das generierte Array mit den gemessenen Resultaten in eine verwertbare Form zu bringen. Hier ist die Konvertierung in eine CSV-Datei, in HTML oder in ein JSON-Objekt implementiert.

```
public static function resultToCSV($results)
{
    (...)
}

public static function resultToHTML($results)
{
    (...)
}

public static function resultToJSON($results)
{
    (...)
}

public static function microtimeToTime($microtime)
{
    (...)
}

public static function microtimeToSec($microtime, $asString = true)
{
    (...)
}
```

Mit einem Blick auf den Code sieht man, dass zuerst die 3 Funktionen für die Konvertierung definiert werden. Anschliessend kommen noch zwei kleinere Funktionen, die die gemessene Zeit von `microtime` umformatieren in ein leserliches Format.

TestHelper.php

Diese Klasse dient als Hilfe für die Ausführung der Test-Serien.

```
private static $series;
public static $tests;

/*
 * Loads all Tests into local array
 */
public static function loadTests()
{
    (...)
}

/*
 * Runs one Test Serie
 */
public function runTestSerie(){}

/*
 * Calculates averages and returns as a php array
 *
 * @return array Results
 */
public static function result()
{
    (...)
}
```

Die Funktion `loadTests` geht durch alle Files im Ordner `Tests` und lädt den Klassennamen samt `Namespace` als String in das lokale Array `$tests`.

Als nächstes wird die Funktion `runTestSerie` definiert. Obwohl es eine sehr einfache und kleine Funktion ist, ist sie einer der Wichtigsten im ganzen Code. Sie führt (wie der Name schon sagt) eine Test-Serie durch und hält den Zeitpunkt des Startes fest. Das Ganze wird anschliessend im Array `$series` gesichert.

Die letzte Funktion dieser Klasse iteriert durch alle Test-Serien und darin enthaltenen Test und berechnet den Durchschnitt. Dies wird am Ende als Array zurückgegeben.

lib/Models

In diesem Verzeichnis liegen alle Klassen im Namespace `Models`. Von diesen Klassen werden Objekte erstellt oder dienen als Parent für weitere Klassen.

Test.php

Diese Klasse dient als Parent für alle Test-Cases. Die Klasse hat folgenden Aufbau:

```
private $state;
public $timer;
public $name;

/*
 * Creates Timer
 */
public function __construct()
{
    (...)
}

/*
 * Needs to be overwritten by Tests, with test-function
 */
public function test(){}

/*
 * Runs the test and stops time
 *
 * @throws Exception if test has been executed before
 */
public function runTest()
{
    (...)
}
```

Die erste Funktion `__construct()` wird beim Erstellen eines Objektes aus dieser Klasse ausgeführt. Darin wird für die Variabel `$timer` ein neues Objekt vom Typ `Timer` erstellt.

Die nächste wichtige Funktion ist `runTest()`. Diese prüft, ob ein Test bereits ausgeführt wurde. Falls ja, wird eine `Exception` mit dem Text `"Test already executed!"` ausgeworfen, da dies in einem normalen Durchlauf nie der Fall sein sollte. Wenn der Test noch nicht ausgeführt wurde, wird der Timer gestartet, die Funktion `test()` aufgerufen, der Timer wieder gestoppt und der Status (`$state`) aktualisiert.

Erwähnenswert ist die Variable `$name` die am Anfang definiert wird. Der damit definierte String wird als Titel beim Generieren der Resultate verwendet.

TestSerie.php

Diese Klasse wird verwendet, um alle Tests einmal durchzuführen und die Resultate zu speichern.

```
private $tests;

/*
 * Creates objects from tests-array in TestHelper */
public function __construct()
{
    (...)
}

/*
 * Cycles through tests and executes test
 */
public function runTests(){}

/*
 * Returns all Test Results with a grand-total as a php array
 *
 * @return array Test results + grand-total
 */
public function result()
{
    (...)
}
```

Im "Constructor" wird in dieser Klasse durch `TestHelper::$tests` iteriert und aus jeder sich darin befindlichen Klasse ein neues Objekt erstellt und in das lokale Array `$tests` gepushed.

Die Funktion `runTests()` geht durch das lokale Array `$tests` und führt für jedes Objekt `runTest()` aus.

Als letztes wird die Funktion `result()` definiert. Diese berechnet ein Total und gibt anschliessend die Resultate und das Total als Array zurück.

Timer.php

Auch wenn die Klasse `Timer` aus lediglich drei Einzeiler-Funktionen und zwei Variablen besteht, ist sie doch eines der Herzstücke des ganzen Projektes. Sie misst die Runtime der Tests.

```
private $startTime;
private $endtime;

/*
 * Starts the timer/Set's startTimer
 */
public function start()
{
    (...)
}

/*
 * Stops the timer/Set's endTime
 */
public function stop(){}

/*
 * Calculates and returns the stopped Time
 *
 * @param integer $decimals if set rounds result with those decimals; if not set returns
unrounded result
 * @return float Stopped Time
 */
public function result($decimals = false)
{
    (...)
}
```

Die Funktionen sollten selbsterklärend sein.

lib/Tests

In diesem Ordner werden alle Tests definiert. Zurzeit ist lediglich ein Beispiel enthalten.

ExampleTest.php

Dient lediglich als Beispiel.

```
class ExampleTest extends Test
{
    public $name = "ExampleTest";

    public function test()
    {
        (...)
    }
}
```

Hier wird eine Klasse definiert, basierend auf dem Model `Test`. Da alle wichtigen Funktionen und Variablen bereits im Parent definiert sind, muss hier lediglich die Variable `$name` und die Funktion `test()` überschrieben werden.

lib/App.php

Die in dieser Datei definierte Klasse dient als Startpunkt des Projektes und enthält lediglich eine statische Funktion `run()`:

```
public function run()
{
    \Helper\TestHelper::loadTests();

    for($i = 0 ; $i < CONFIG_SERIES ; $i++)
        \Helper\TestHelper::runTestSerie();

    $method = (...) $_GET['m'] (...);
    switch($method)
    {
        (...)
    }
}
```

Hier ist ein bisschen mehr Code dargestellt um die Aufgabe dieser Funktion besser zeigen zu können. Als erstes werden alle Tests im Ordner `lib/test` geladen. Anschliessend werden Test-Serien gestartet, entsprechend der Anzahl definiert in `config.php`. Schlussendlich wird überprüft ob und welcher Wert im `GET` Parameter `m` übergeben wird und dementsprechend das Resultat generiert.

lib/boot.php

Diese Datei besteht nur aus `require` Commands und lädt alle benötigten Dateien.

lib/config.php

In der Config-Datei werden nur `define` Befehle abgesetzt und damit alle Einstellungen definiert, wie zum Beispiel die Anzahl durchzuführenden Test-Serien.

public/index.php

Die `Index` Datei dient als Einstiegspunkt für den Webserver. Darin wird lediglich die `boot.php` Datei geladen und anschliessend das Projekt gestartet.

```
require(__DIR__.'../lib/boot.php');
App::run();
```

9 Tests

Das Grundgerüst steht schon mal. Jetzt gilt es noch die Tests zu definieren.

9.1 Math

Als erster Test wird etwas verwendet, das sehr häufig in Bechmark Scripts anzutreffen ist: Math-Functions. In diesem Test-Case werden diverse mathematische Funktionen definiert und ausgeführt.

```
public $name = "Math Test";
public $functions = [
    'abs' => 1,
    'acos' => 1,
    'acosh' => 1,
    (...)
    'srand' => 1,
    'tan' => 1,
    'tanh' => 1,
];

public function test()
{
    for($i = 0; $i < 1000; $i++)
        foreach($this->functions as $func => $numParam)
        {
            $params = $numParam > 0 ? explode(',', substr(str_repeat("1,", $numParam), 0, -1))
: [];
            call_user_func_array($func, $params);
        }
}
```

Dieser Test sieht komplizierter aus als er ist. Im ersten Schritt wird das Array `$functions` definiert das aus allen mathematischen Funktionen als String im `Key` definiert mit der Anzahl an Parameter als `Value`. Das sind hier 46 Stück. Anschliessend in der Funktion `test()` wird durch dieses 1000 mal Array iteriert. Nun kommt die komplizierteste Zeile im Test. Diese besteht einfach aus ein paar verschachtelten Funktionen, deren Wirkungen und Outputs hier aufgezeigt werden.

Für das folgende Beispiel wird davon ausgegangen, dass es sich um das `KeyValuePair` Element `'min' => 2` handelt und `$numParam` somit `2` entspricht.

Es beginnt zuinnerst mit `str_repeat("1,", $numParam)` wiederholt den String im ersten Parameter `n`-mal. Das gibt den Output: `"1,1,"`

Im nächsten Schritt wird mit `substr(..., 0, -1)` davon ein Substring herausgelöst vom Anfang bis zum 2. letzten Zeichen. Was den Output `"1,1"` generiert.

Dieser String wird der Funktion `explode` an 2. Stelle übergeben mit dem ersten Parameter auf `,` gesetzt. Dadurch wird ein Array generiert, das wie folgt aussieht: `[1,1]`

Vor alledem steht noch die Abfrage `$numParam > 0 ? ... : []` was zur Folge hat, dass bei einer Funktion mit keinem Parameter nicht diese Prozedur ausgeführt, sondern direkt ein leeres Array erstellt wird.

Dieses zuvor generierte Array wird nun als Parameter der in `$func` gespeicherten Funktion übergeben. In diesem Beispiel würde das heissen, dass folgender Command ausgeführt wird: `min(1,1)`

9.2 Loop

Dieser Test ist zwar sehr einfach, doch ist er in den meisten Benchmark Scripts wieder zu finden. Es geht hier um die Geschwindigkeit der Loop Funktionen `for` und `foreach`.

```
public $name = "Loop Test";

public function test()
{
    $arr = [];
    for($i = 0; $i < 10000; $i++)
        array_push($arr,$i);

    foreach($arr as $el)
        unset($arr[array_search($el,$arr)]);
}
```

Der Aufbau ist sehr einfach. Es wird ein leeres Array definiert, das anschliessend mit einer `for` Schleife von 0 bis 10'000 gefüllt wird. Dieses nun gefüllte Array wird gleich darauf in einer `foreach` Schleife wieder geleert.

9.3 DB

Der DB Test dient dem Vergleich der verschiedenen Datenbank Server und besteht aus folgenden Commands:

```
public $name = "DB Test";

public function test()
{
    $pdo = new \PDO('mysql:host=localhost', 'root', '');
    $pdo->query("CREATE DATABASE benchmark_test;");

    $pdo = new \PDO('mysql:host=localhost;dbname=benchmark_test', 'root', '');
    $pdo->exec("CREATE TABLE data_test(id INT(10) AUTO_INCREMENT PRIMARY KEY, key_col
    VARCHAR(255), value_col VARCHAR(255) );");
    for($i = 0; $i < 1000; $i++)
        $pdo->exec("INSERT INTO data_test(key_col,value_col) VALUES('$i','$i');");

    foreach($pdo->query("SELECT * FROM data_test;") as $row)
    {
        $q = $pdo->query("SELECT * FROM data_test WHERE id = ".$row['id']);
        $id = !$q ? -1 : $q->fetch()['id'];
        $pdo->exec("DELETE FROM data_test WHERE id = $id");
    }

    $pdo = new \PDO('mysql:host=localhost', 'root', '');
    $pdo->exec("DROP DATABASE benchmark_test;");
    $pdo = null;
}
```

Trotz einem wirren Aussehen der Funktion, ist es ziemlich simpel. Zuerst wird eine Verbindung, ohne eine Datenbank auszuwählen, aufgebaut und eine Datenbank mit dem Namen `benchmark_test` erstellt. Anschliessend wird eine neue Verbindung aufgebaut zu dieser Datenbank. Nun wird eine neue Tabelle `data_test` erstellt und in einer `for` Schleife mit 1000 Einträgen gefüllt. Diese Einträge werden mit einem `select` Befehl zuerst alle und gleich darauf nochmals einzeln die Spalte `id` ausgelesen. Mit dieser `id` wird der Eintrag, gefolgt von der ganzen Datenbank, gelöscht.

Was merkwürdig scheinen kann ist, dass `PDO` mit `\PDO` aufgerufen wird. Dies liegt daran, dass die Klassen im `Namespace` des jeweiligen Ordners liegen.

9.4 FileSystem

Der File System Test besteht aus vielen Zugriffen auf eine Datei.

```
public $name = "File System Test";

public function test()
{
    $path = realpath(__DIR__.'../../public/test.txt');
    file_put_contents($path, "-");

    for($i = 0; $i < 1000; $i++)
    {
        $content = file_get_contents($path);
        $content .= str_repeat($i, $i);
        file_put_contents($path, $content);
    }
}
```

Dieser Test-Case ist folgendermassen aufgebaut: Zuerst wird der Inhalt der Datei `public/test.txt` auf `"-"` gesetzt, damit kann sichergestellt werden, dass die Datei keine Grösse hat. (Bzw. nur ein paar Bytes) Somit beginnt der eigentliche Test. Diese Datei wird nun 1000 mal ausgelesen und mit einem String ergänzt, der aus n-mal dem n-ten Ziffer besteht. Somit sieht die Datei folgendermassen aus: `"1223334444..."`

Für diesen Test musste noch ein `.gitignore` im Verzeichnis `public` erstellt werden, damit `test.txt` nicht mit gesynced wird, da sich der Inhalt konstant ändert.

9.5 BigFile

Der Test Big File ist dem File System Test sehr ähnlich, aber unterscheidet sich in einem wesentlichen Punkt. Er arbeitet nicht mit einer Datei von einigen Kilobytes, sondern mit einer Grösse von ca. 150 MB. Dafür wird dies auch nur 10 mal durchgeführt.

```
public $name = "Big Files Test";

public function test()
{
    $path = realpath(__DIR__.'../../public/test.txt');
    file_put_contents($path, "-");

    for($i = 0; $i < 10; $i++)
    {
        $content = file_get_contents($path);
        $content = str_repeat($i, 100*1000000);
        file_put_contents($path, $content);
    }
}
```

Im Vergleich zu vorher hat sich geändert, dass die Variable `$content` durch den neuen String ersetzt und nicht ergänzt wird, und die Länge des Strings nicht abhängig von der Iteration ist.

9.6 URL Call

Nun kommen wir zum grossen Finale. Der URL Call Test soll die effektive Antwortzeit des Webserver

```
public $name = "URL Call test";

public function test()
{
    $currenturl = "http://$_SERVER[HTTP_HOST]$_SERVER[REQUEST_URI]/";

    for($i = 0; $i < 1000; $i++)
        file_get_contents("$currenturl/test.txt?i=$i");
}
```

testen. Da dies von den Webserver selber kommt, ist sogar die Netzwerk-Latency umgangen.

Die Funktion ist sehr kurz und lässt sich auch so schnell erklären. Als erstes wird der Hostname und URL des Servers bestimmt und anschliessend 1000 mal aufgerufen. Hier wird die Datei `test.txt` geöffnet. Der `GET` Parameter `i=$i` dient lediglich zur Vorbeugung, damit die Datei nicht gecached wird.

10 Resultat

Schlussendlich das eigentlichen Thema dieser Arbeit: Der Vergleich.

Es wurden jeweils pro Server zwei Test-Sätze an 50 Serien generiert.

10.1 dk-matura-01

Nachfolgend die durchschnittlichen Ausführzeiten der beiden Sätze:

Total	Big Files Test	DB Test	File System Test	Loop Test	Math Test	URL Call test
34867.711	11002.751	7076.579	6637.755	76.826	41.555	10032.244
24167.611	9948.625	3278.268	7347.344	74.578	40.778	3478.019

Wenn man den Verlauf über die Zeit grafisch darstellt sieht das für das Total folgendermassen aus:

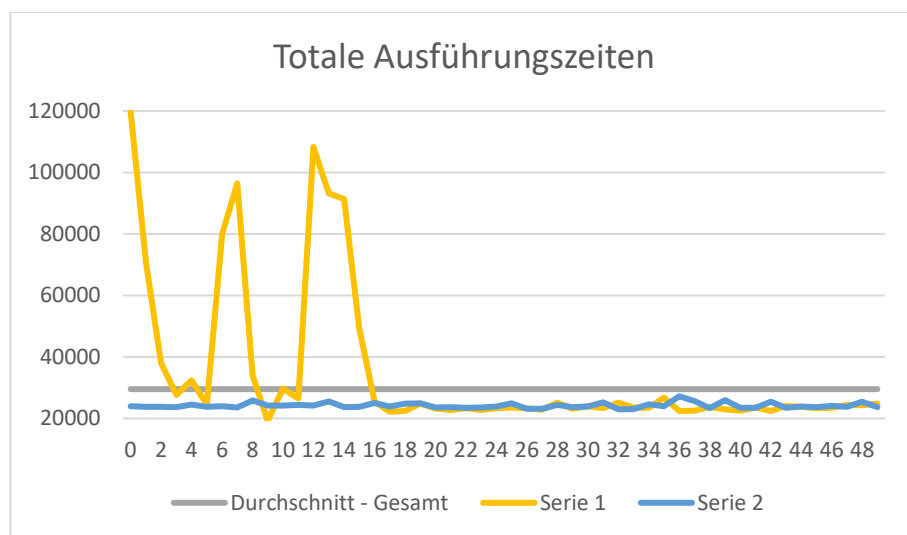


Abbildung 12 dk-matura-01 Totale Ausführungszeiten

Da bei den einzelnen Tests im zeitlichen Verlauf die genauen Zeiten nicht interessant sind, wurden sie auf einer Skala von 0 bis 100 heruntergerechnet, wobei 0 die minimale und 100 die maximale Zeit beträgt. Hier sind die beiden Test-Sätze mit dieser Skala dargestellt. Es wurden jeweils drei Tests zusammen dargestellt. Zur Nachverfolgung, der Excel Command sieht folgendermassen aus, wobei die Spalte B von Zeile 3 bis 52 alle Datensätze des aktuellen Testes enthalten und B3 der zu berechnende Datensatz ist:

```
= (B3-MIN(B$3:B$52)) / (MAX(B$3:B$52)-MIN(B$3:B$52)) * 100
```

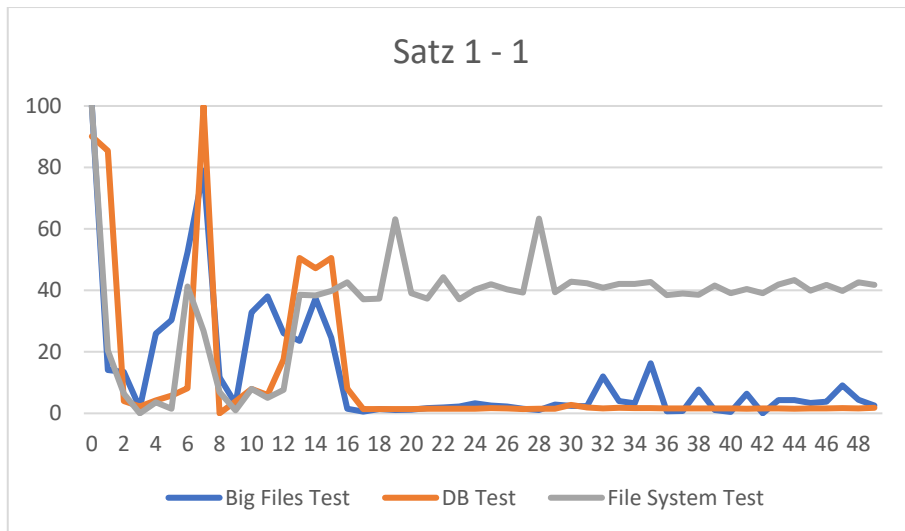


Abbildung 13 dk-matura-01 Erster Durchlauf, Big Files, DB & File System Test

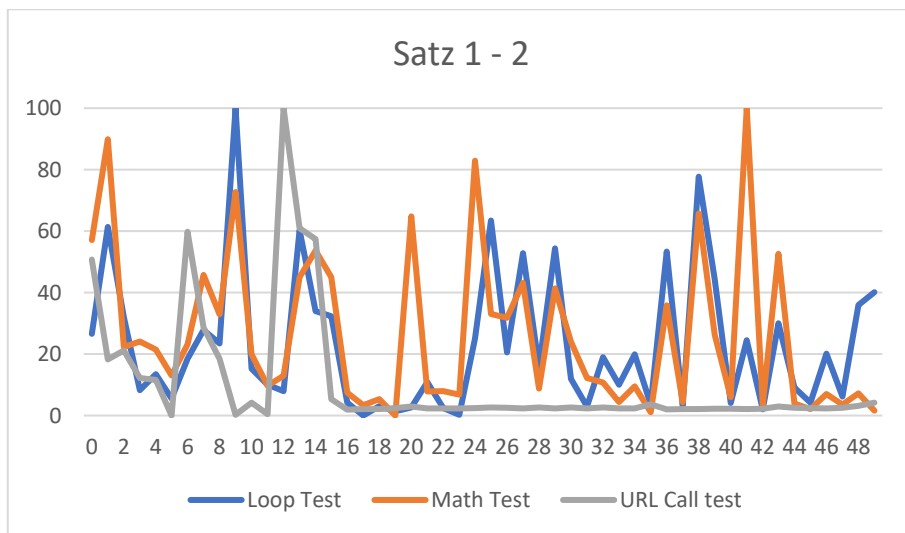


Abbildung 14 dk-matura-01 Erster Durchlauf, Loop, Math & URL Call Test

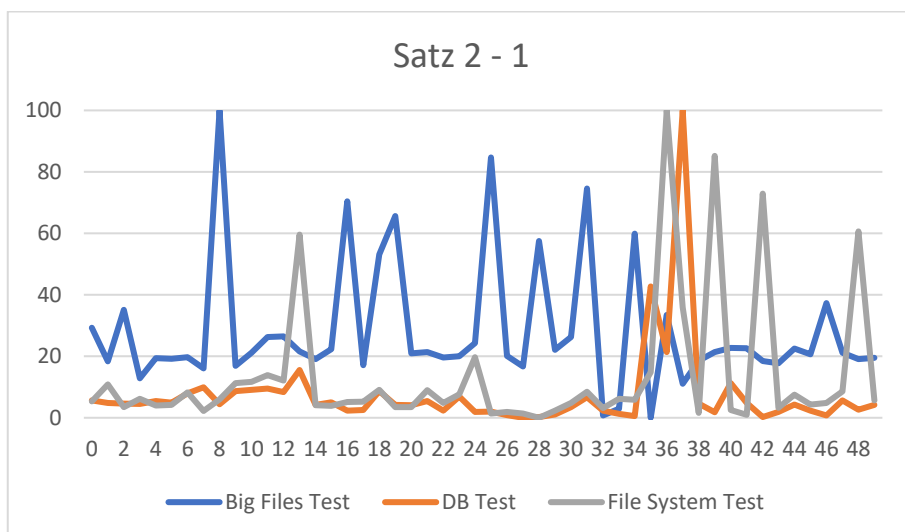


Abbildung 15 dk-matura-01 Zweiter Durchlauf, Big Files, DB & File System Test

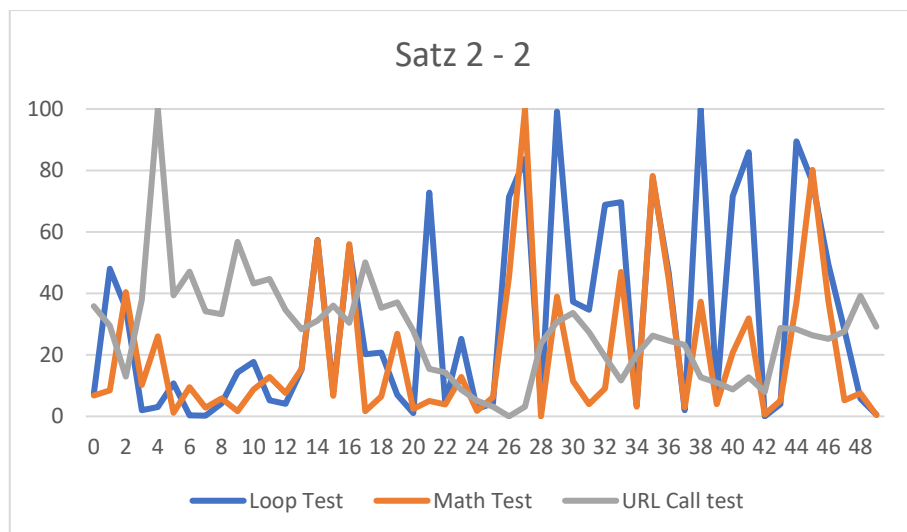


Abbildung 16 dk-matura-01 Zweiter Durchlauf, Loop, Math & URL Call Test

Was lässt sich in diesen Grafiken nun rauslesen und reininterpretieren?

Als erstes fällt auf, dass sowohl das Total, wie auch der Datenbank und "Big File" Test am Anfang grosse Ausschläge machen. Der Zusammenhang zwischen den Tests und dem Total liegt wohl darin, dass diese beiden Tests sehr grosse Zeiten generieren und dadurch einen starken Einfluss auf das Total haben. Auch zwischen den beiden Test-Cases ist der Zusammenhang nicht weiter verwunderlich, da beide viele Dateizugriffe auf grössere Daten verursachen, was sehr stark abhängig vom System und der Harddisk/SSH ist. Vermutlich war das System gerade anderweitig ausgelastet.

Interessanter wird es nun, wenn wir den "File System Test" dazu nehmen. Sobald der Big File Test schneller wird, verlangsamt sich der File System Test. Dieser Effekt, in umgekehrter Reihenfolge, lässt sich, zumindest schwach, auch im zweiten Satz wiedererkennen. Ein Phänomen, welches äusserst interessant scheint, jedoch keine Begründung offensichtlich genannt werden könnte.

Wenn man nun die Stabilität dieser drei Tests betrachtet und die grobe Änderung in der Ausführzeit missachtet fällt auf, dass die Datenbank übers Ganze die grösste Stabilität bietet, gefolgt vom File System Test (also kleinen Files) mit einem nahen Schlusslicht, den grossen Dateien. Dies ist ein Merkmal, dass für die Entwicklung sehr wichtig sein kann. Dies beweist nämlich, dass die Verwendung einer Datenbank die beste Option ist und man im Falle von File-Zugriffen diese möglichst klein, dafür häufiger halten soll.

Kommen wir nun noch zu den zweiten Grafiken. Es ist sehr deutlich zu sehen, dass ein direkter Zusammenhang zwischen dem Loop und dem Math Test besteht. Dies war zu erwarten, da beide auf PHP Funktionen basieren. Das Ganze sieht zwar äusserst instabil aus, das täuscht jedoch nur durch die 0-100 Skala. Wenn man auf die effektiven Daten zurückgreift und die minimale und maximale Ausführzeit betrachten ist zu erkennen, dass der Unterschied jeweils bei ca. 40ms liegt. Bei den

anderen Tests handelt es sich um hunderte bis tausende von Millisekunden. Zu bemerken ist, dass sich die starke Auslastung am Anfang des ersten Satzes kaum abzeichnet.

Schauen wir uns doch noch den URL Call Test Case an. Er ist, ähnlich den anderen beiden auf PHP Funktionen basierenden Tests, eher stabil. Auch hier ist im Normalfall nur von einigen zehn bis hundert Millisekunden Unterschied zu sprechen. Jedoch war dieser extrem beeinflussbar von der grossen Belastung zu Beginn des ersten Sets. Dies zeigt auf, dass die Response Time des Servers extrem von dessen Auslastung abhängt.

10.2 dk-matura-02

Nun dasselbe nochmals, einfach für den Server dk-matura-02.

Total	Big Files Test	DB Test	File System Test	Loop Test	Math Test	URL Call test
19052.36	7559.49	2929.786	5497.447	66.94829	35.95863	2962.731
18574.41	7297.326	2943.352	5311.696	66.44755	35.03839	2920.546

Auch hier wieder dieselbe Grafik wie zuvor:

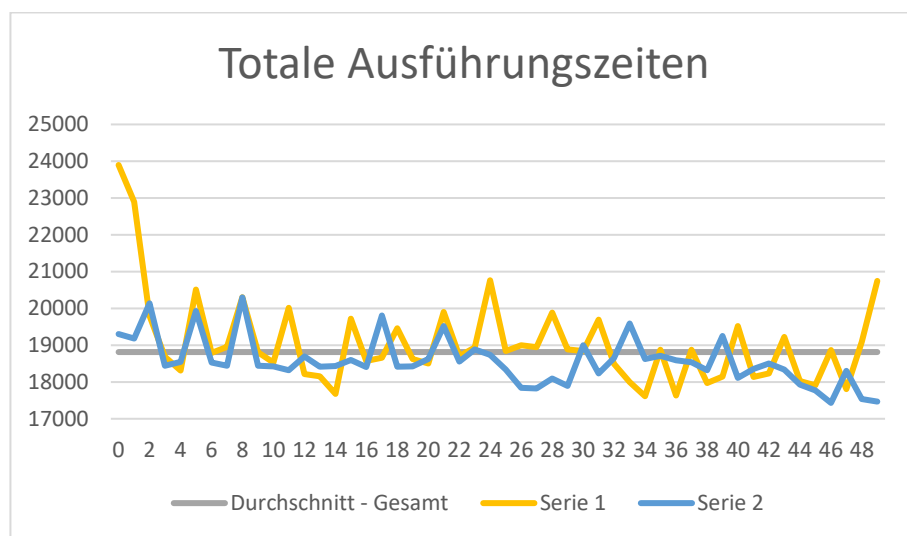


Abbildung 17 dk-matura-02 Totale Ausführungszeiten

Und ebenfalls der Tests:

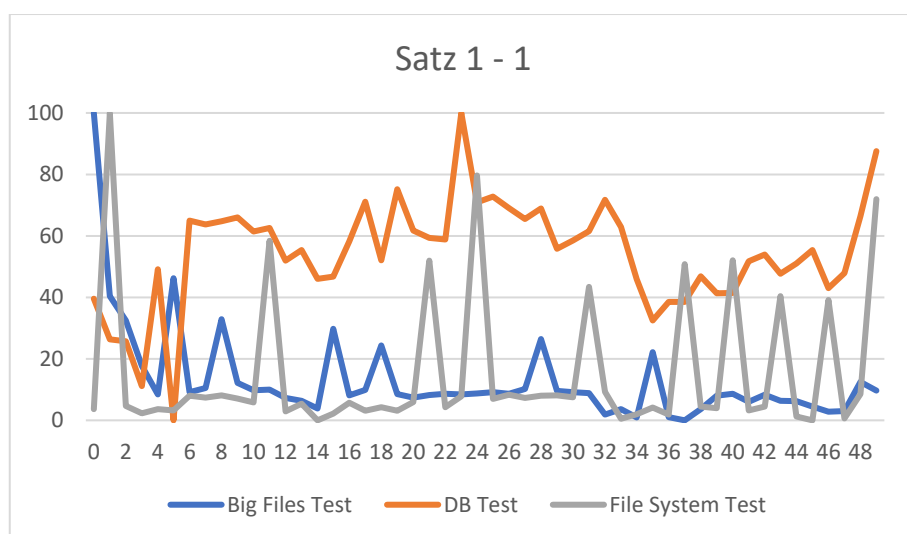


Abbildung 18 dk-matura-02 Erster Durchlauf, Big Files, DB & File System Test

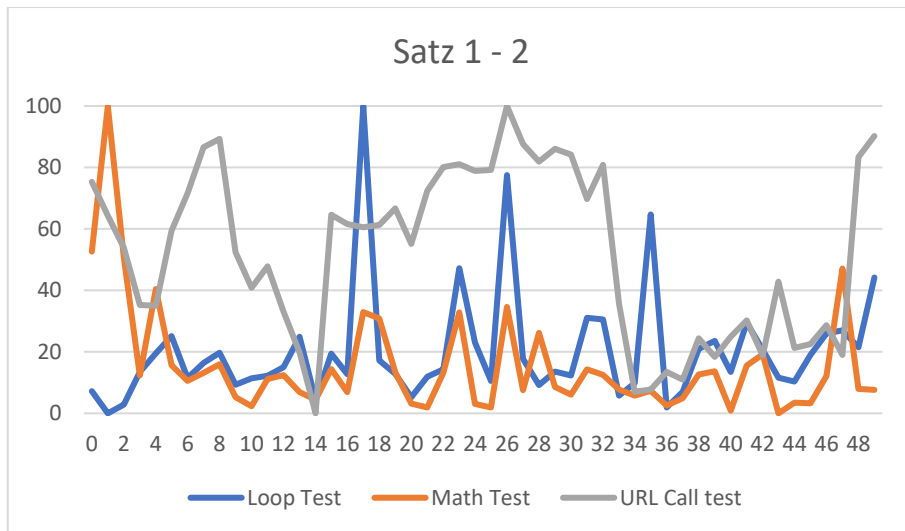


Abbildung 19 dk-matura-02 Erster Durchlauf, Loop, Math & URL Call Test

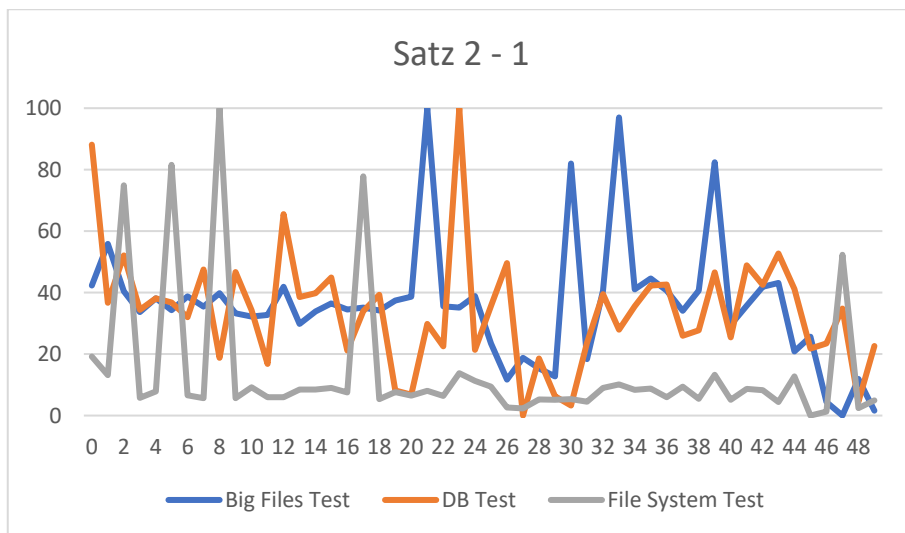


Abbildung 20 dk-matura-02 Zweiter Durchlauf, Big Files, DB & File System Test

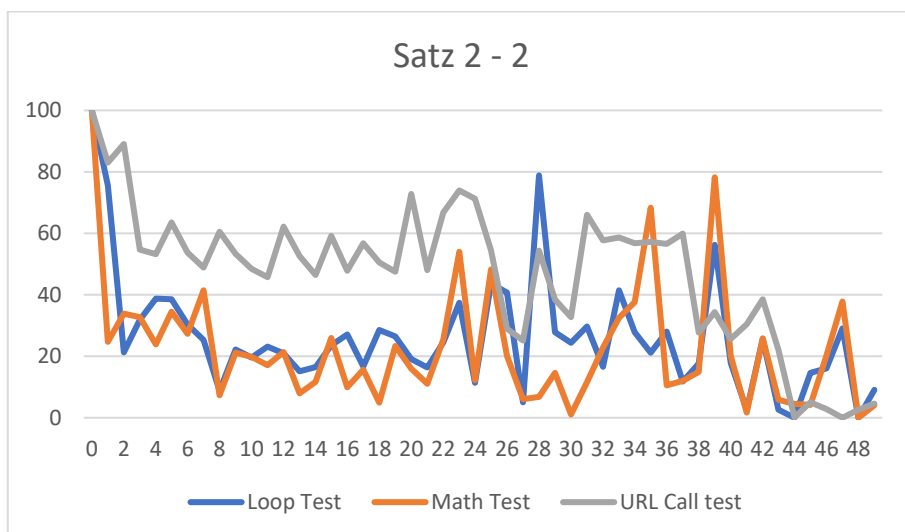


Abbildung 21 dk-matura-02 Zweiter Durchlauf, Big Files, DB & File System Test

Bereits bei einem kurzen Blick auf das Total fällt auf, dass hier stabilere Zahlen vorhanden sind. Dies hat zur Folge, dass beim direkten Vergleich zwar die Daten nicht "bereinigt" werden müssen, jedoch innerhalb der Testserien ein Unterschied schwerer zu erkennen ist. Dass es keinen solchen "Spike" gegeben hat, ist wohl komplett dem Zufall zu zuschreiben.

Bei den File und DB Tests lassen sich in sich selber keine grossen Feststellungen machen. Einzig die Annahme vom dk-matura-01, dass sich kleine und grosse Files gegensätzlich verhalten, kann man vor allem beim zweiten Test-Set schwach erkennen.

Ebenso bei den Tests Loop, Math und URL Call sehen die Resultate dem vorherigen Server ähnlich.

10.3 dk-matura-03

Und zum Schluss dasselbe mit der Ubuntu Maschine:

Total	Big Files Test	DB Test	File System Test	Loop Test	Math Test	URL Call test
50738.34	9604.388	27748.45	10105.74	69.14797	40.9298	3169.685
24195.49	7327.404	7708.21	5985.922	67.9733	35.17417	3070.803

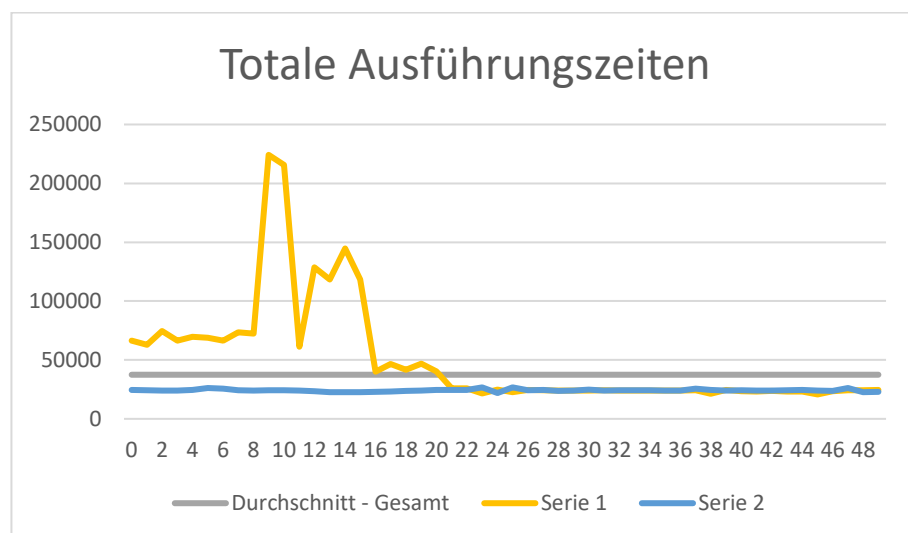


Abbildung 22 dk-matura-03 Totale Ausführungszeiten

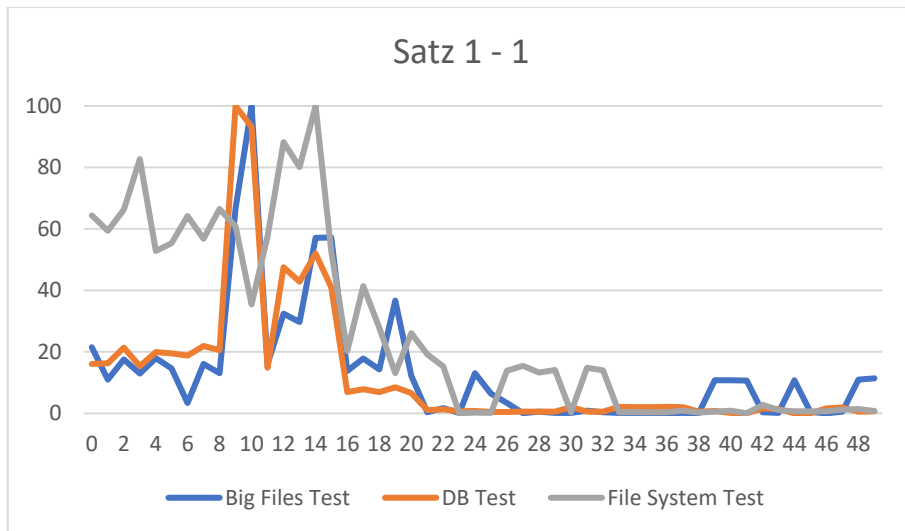


Abbildung 23 dk-matura-03 Erster Durchlauf, Big Files, DB & File System Test

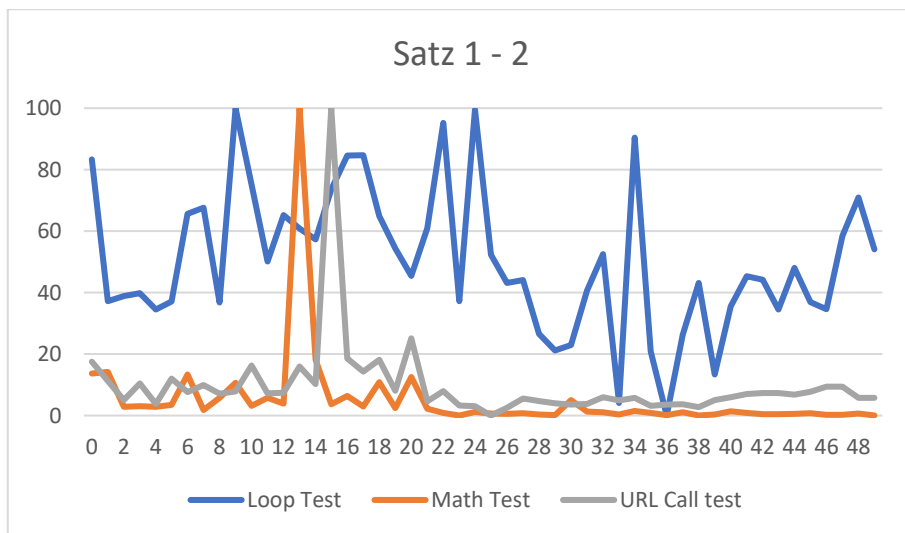


Abbildung 24 dk-matura-03 Erster Durchlauf, Loop, Math & URL Call Test

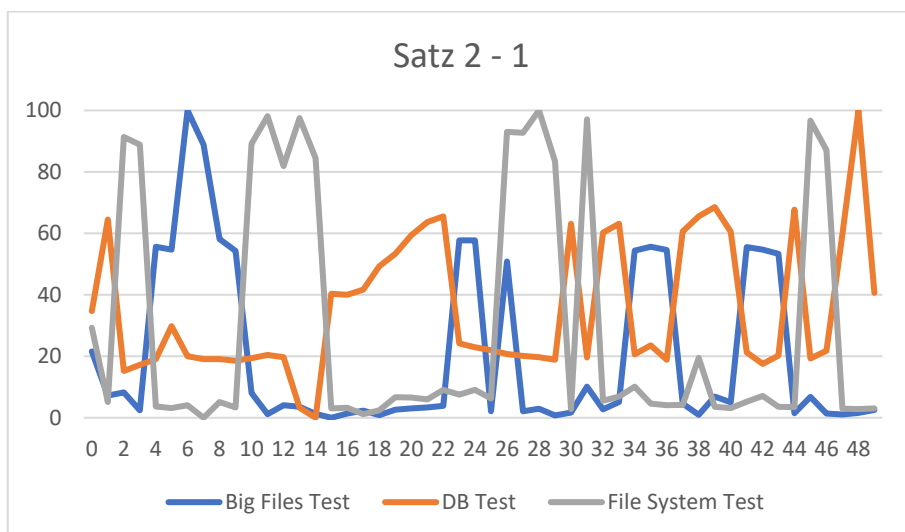


Abbildung 25 dk-matura-03 Zweiter Durchlauf, Big Files, DB & File System Test

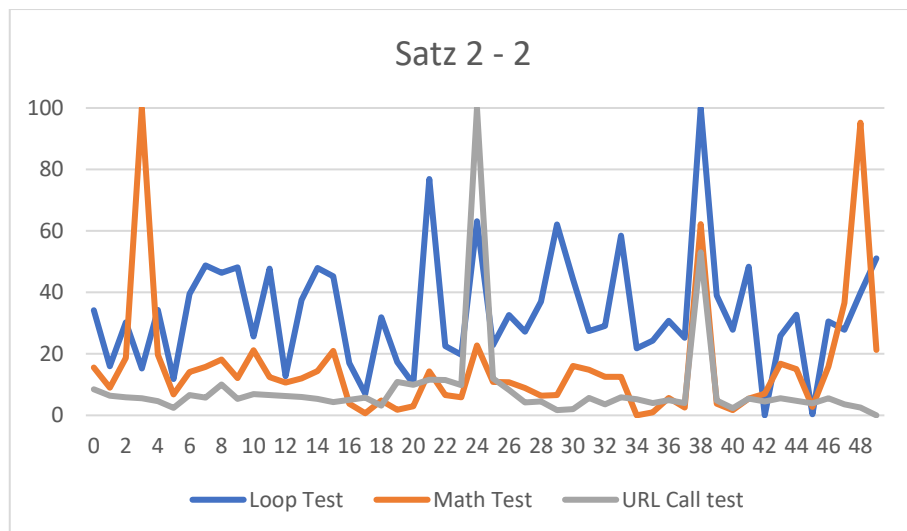


Abbildung 26 dk-matura-03 Zweiter Durchlauf, Loop, Math & URL Call Test

Das Erste, das einem förmlich ins Auge sticht, sind die extremen Spikes am Anfang des "Total" Diagrammes. Wenn man seinen Blick nun auf das nächste Diagramm schwenkt fällt auf, dass die gleiche Form dieser Extreme ebenfalls wieder zu erkennen ist. Jedoch sind diese starken Ausprägungen nur im ersten Test-Set zu sehen, der Zweite sieht sehr stabil aus. (Dies liegt natürlich daran, dass die extremen Maximalwerte die Grafik verfälschen. Doch auch wenn man die Achsenbeschriftung auf den Bereich 0-50'000 setzt, sieht der zweite Test-Satz ähnlich stabil und unspektakulär aus.)

Wenn man etwas genauer auf die zweite und vierte Grafik schaut ist zu erkennen, dass sich, wie bereits bei den CentOS Maschinen, die Tests "Big File" und "File System" gegensätzlich verhalten. Die Datenbank ist im ersten Test-Satz, nach anfänglichem starkem Schwanken, sehr stabil. Im zweiten Set sieht sie eher instabil aus, jedoch trägt der Schein hier ein wenig.

Interessant ist hier in der dritten und fünften Grafik, dass der Loop und der Math Test nicht mehr so nahe beieinanderliegen wie zuvor und eher unabhängig zu sein scheinen.

Dafür ist bei dieser Maschine der URL Call Test, abgesehen von ein paar wenigen Spikes, sehr stabil.

10.4 Vergleich

Vergleichen wir nun mal die beiden CentOS und die Ubuntu Maschinen. Zuerst muss man sich nochmals vor Auge führen, was denn der Unterschied ist.

Messpunkt	dk-matura-01	dk-matura-02	dk-matura-03
Distribution	CentOS	CentOS	Ubuntu
Webserver	nginx	Apache	Apache
Datenbank	MariaDB	MariaDB	MySQL

Sowohl die Distribution wie auch die Datenbank sind bei den ersten beiden Servern identisch. Somit ist zwischen diesen beiden Servern eher ein kleinerer Unterschied zu erwarten und nahezu keiner in Punkte Datenbank und File System. Der grösste Unterschied dürfte sich wohl zwischen dk-matura-01 und dk-matura-03 zeigen, da diese nichts identisch haben. Aber ist dem nun wirklich so?

Es werden folgend jeweils zuerst den Durchschnitt aller Test Serien und anschliessend den Durchschnitt, das Minimum und das Maximum der Mittleren 80% gezeigt. Dieses "Abschneiden" der oberen und unteren 10% verhindert, dass extreme Ausschläge den Vergleich zu stark verfälschen. Somit haben wir 80 Test-Serien in 2 Test-Sätzen, die zeitlich versetzt je über eine Zeit von ca. 20-30 Minuten aufgenommen wurden.

10.4.1 Datenbank

Als erstes fassen wir die Datenbank ins Auge. Jeder Test enthält ca. 1000 Abfragen. Also werden mehr als 80'000 SQL Queries verrechnet, was durchaus eine repräsentative Aussagekraft hat.

Messpunkt	dk-matura-01	dk-matura-02	dk-matura-03
DB Runtime Durchschnitt	5177.42 ms	2936.50 ms	17728.33 ms
DB Runtime Durchschnitt der Mittleren 80%	3364.56 ms	2936.58 ms	10197.34 ms
DB Runtime Minimum der Mittleren 80%	3183.23 ms	2885.82 ms	6590.43 ms
DB Runtime Maximum der Mittleren 80%	5047.74 ms	2977.82 ms	38695.75 ms

Die Unterschiede sind in diesem Test massiv. Wie zu sehen ist, verliert der dk-matura-03 in höchstem Masse. In der Mitte befindet sich dk-matura-01, der sich relativ gut geschlagen hat und der klare Sieger ist dk-matura-02. Was kann man nun daraus schliessen?

Ganz klar scheint, dass die Distribution einen Einfluss auf die Datenbank hat. Ob die Niederlage vom dk-matura-03 nun Ubuntu oder MySQL zuzuschreiben ist, ist ohne weitere Tests nicht feststellbar. Jedoch lässt sich ganz klar sagen, dass CentOS-MariaDB eine bessere Kombination als Ubuntu-MySQL darstellen und am besten mit Apache harmonisieren.

10.4.2 File System

Kommen wir zum File System. Hier werden die beiden Tests "File System Test" und "Big File Test" analysiert.

Messpunkt	dk-matura-01	dk-matura-02	dk-matura-03
File System Durchschnitt	6992.55 ms	5404.57 ms	8045.83 ms
File System Durchschnitt der Mittleren 80%	7068.85 ms	5270.33 ms	6921.56 ms
File System Minimum der Mittleren 80%	5915.27 ms	5128.17 ms	5212.80 ms
File System Maximum der Mittleren 80%	7390.41 ms	6431.71 ms	15889.04 ms

Messpunkt	dk-matura-01	dk-matura-02	dk-matura-03
Big File Durchschnitt	10475.69 ms	7428.40 ms	8465.90 ms
Big File Durchschnitt der Mittleren 80%	9908.20 ms	7302.85 ms	7634.84 ms
Big File Minimum der Mittleren 80%	9100.76 ms	6952.79 ms	6355.74 ms
Big File Maximum der Mittleren 80%	12892.83 ms	8082.99 ms	10579.48 ms

Dieser Test liefert ein äusserst unerwartetes Resultat. Da beim Zugriff von PHP auf das Datei System die Webengine eigentlich keinen direkten Einfluss hat. Trotzdem scheint der dk-matura-01 dem dk-matura-02, zumindest im Big File Test, ganz klar unterlegen zu sein. Dies führt zu folgenden Erkenntnissen: Sowohl Ubuntu wie auch CentOS scheinen im Großen und Ganzen in Sachen File-Zugriff gleichauf zu sein, sind jedoch beide von starken Schwankungen/Spikes betroffen.

10.4.3 URL Call

Schauen wir uns zu guter Letzt noch die Resultate des "URL Call Test" an.

Messpunkt	dk-matura-01	dk-matura-02	dk-matura-03
File System Durchschnitt	6755.13 ms	2941.64 ms	3120.24 ms
File System Durchschnitt der Mittleren 80%	3671.20 ms	2948.02 ms	3057.03 ms
File System Minimum der Mittleren 80%	3232.46 ms	2680.42 ms	2961.92 ms
File System Maximum der Mittleren 80%	10740.53 ms	3187.98 ms	3226.10 ms

Dieser Test hat vermutlich die stärkste Aussagekraft im direkten Bezug auf die Webengine und zeigt (mehr oder weniger direkt) die Response-Time des Servers auf. Das Resultat entspricht hier definitiv meiner Erwartungen, bzw. Hoffnungen. Apache hat hier nginx ganz klar übertroffen, dies ist auch, was man generell von Hobby Webserver Betreibern hört im Bezug auf kleinere Webserver mit wenig Leistung. Nginx ist nicht für so kleine Systeme ausgelegt.

Auch im Bezug auf die Distribution entspricht es meinen Hoffnungen, da CentOS vorne liegt, wenn auch nur um ganz wenig.

TEIL 5

Browser Script

11 Grundsatz

Die Grundidee des (Teil-)Projektes ist folgende:

Es werden mehrere Tests definiert, diese werden je einmal ausgeführt, was eine Test-Serie bildet. Solche Test-Serien werden mehrere generiert, welche dann ein auswertbares Resultat bilden sollte. Der Einfachheit halber ist dies als Teilprojekt im **GIT** Projekt des Runtime Scripts untergeordnet, im Verzeichnis `public/client`.

12 Code

Das Projekt besteht aus lediglich vier Dateien, auf deren Funktion hier eingegangen wird.

12.1 index.html

Beginnen wir mit der **Index** Datei. Diese dient als Einstiegspunkt für den Server und enthält die Struktur der Webkomponente. Auch hier wieder der grobe Aufbau:

```
<html>
  <head>
    <title>Client Test</title>
    <script src="jquery-3.3.1.min.js"></script>
    <script src="script.js"></script>
    <style>
      (...)
    </style>
  </head>
  <body>
    <pre id="json-output"></pre>
    <table id="results"></table>
    <div id="hidden-dom"></div>
  </body>
</html>
```

Der Aufbau ist der einer klassischen HTML Datei. Im **head** Tag werden die beiden Scripts, die im gleichen Ordner liegen, eingebunden wie auch einige Styles definiert, die die Ausgabe leserlicher machen.

Im **body** Tag werden lediglich drei Elemente erstellt, der Rest geschieht über JavaScript. Das Erste ist ein **pre** Element mit der ID `json-output`. Darin werden die Resultate als **json-string** ausgegeben, die mit der Website `json-csv.com` (JSON CSV, kein Datum) in ein CSV, bzw. eine Excel-Datei umgewandelt werden. Das nächste Element ist eine Tabelle mit der ID `results` in der die Resultate gleich in einer übersichtlichen Variante dargestellt werden. Das letzte Element, ein **div**, ist versteckt und dient lediglich einem Test.

12.2 jquery-3.3.1.min.js

Diese Datei ist eine **Dependency** namens jQuery (jQuery, kein Datum) und ist im **minified-js** Format vorliegend. Mit Hilfe dieser Library können gewisse Funktionalitäten, wie zum Beispiel **AJAX Calls** und Änderungen im **Document**, einfacher implementiert werden.

12.3 sciprt.js

Diese JavaScript ist vom Autor geschrieben worden und enthält den eigentlichen Test. Folgend ein grober Überblick:

```
$(document).ready(function(){
    /**
     * Define Tests
     */
    function consoleTest(){}
    function loopTest(){}
    function reqresCallTest(){}
    (...)

    /**
     * Define Variables & Constants
     */
    var results = [];
    const tests = [
        {'name': 'Loop Test', 'callback' : loopTest},
        {'name': 'Console Test', 'callback' : consoleTest},
        {'name': 'reqres Call Test', 'callback' : reqresCallTest},
        {'name': 'Self Call Test', 'callback' : selfCallTest},
        {'name': 'DOM Test', 'callback' : domTest},
    ];
    (...)

    /**
     * Define Stopwatch
     */
    function stopwatch(callback){
        var start = new Date();
        callback();
        var end = new Date();
        return (end-start);
    };

    /**
     * Running Test Seties
     */
    (...)
    for(testID in tests)
    {
        var test = tests[testID];
        var result = stopwatch(test.callback);
        (...)
    }
    (...)
    $('#json-output').html(JSON.stringify(results));
});
```

Da die Datei ca. 160 Zeilen enthält ist es schwierig sie hier korrekt aufzuzeigen, deshalb wurde versucht einige wichtige Anhaltspunkte und die grobe Struktur abzubilden.

Beginnen wir gleich zu Oberst. Das ganze Script ist in einem `ready` Event auf dem `document` verpackt. Dies dient dazu, dass der Code erst ausgeführt wird, wenn die komplette Datei geladen und die Webseite bereit ist. Gleich zu Beginn werden die Test `Callbacks` definiert. Auf die einzelnen Tests wird in den nächsten Unterkapiteln eingegangen.

Im nächsten Schritt werden diverse globale Variablen und Konstanten definiert, die im Verlauf des Scriptes benötigt werden, darunter auch `results`, die die Resultate der Testserien enthält, wie auch `tests`. In der Konstante `tests`, die selber ein Array ist, sind mehrere Objekte definiert, die jeweils ein Property `name` und `callback` haben. Die Eigenschaft `callback` enthält die Funktion, die für den Test ausgeführt werden soll, `name` ist selbsterklärend denke ich.

Nun kommen wir bereits zum Herzstück des Projektes, die `stopwatch` Funktion. Sie misst die Zeit, die ein Test zum Auführen benötigt. Dies geschieht, indem vor und nach dem Aufruf der Callback-Funktion ein `Date` Objekt erstellt und anschliessend die Differenz berechnet wird.

Somit kommen wir schon zum Schlussteil, dem eigentlichen Ausführen der Tests. Da dieser Teil aus über 60 Zeilen besteht, jedoch nicht soviel macht, ist es folgend in Worten zusammen gefasst. (Der komplette Code kann im Anhang betrachtet werden.)

Es beginnt mit dem Erstellen der Tabellen-Zeilen für die Überschrift und den Durchschnitt. Darauf folgt eine grosse `for` Schleife, die sich durch die Anzahl Serien zählt. Beim ersten Durchgang wird die Tabellen-Überschrift erstellt und das `Array` für den Durchschnitt initialisiert. Bei allen weiteren Durchgängen wird eine Testserie ausgeführt. Als erstes wird eine Tabellen-Zeile vorbereitet, die nach einer weiteren `for` Schleife in die Tabelle eingefügt wird. In diesem Loop (den wir im Ausschnitt oben auch sehen) wird der Callback des Tests der Funktion `stopwatch` übergeben und darin ausgeführt. Das Resultat wird im `results` Array gespeichert, in der HTML Tabelle hinzugefügt und in den Durchschnitt verrechnet.

Zum Schluss werden noch einige Daten bereinigt und verrechnet und zu einem `json-string` umformatiert im `pre` Element dargestellt.

12.3.1 Loop Test

Der Loop Test ist dem gleichbenannten Test im Runtime Script sehr ähnlich. Er füllt in einer `for` Schleife ein Array mit 100'000 Zahlen, durch welches anschliessend iteriert und diese wieder löscht.

12.3.2 Console Test

In diesem Test-Case werden 1000 Zahlen in die Entwickler-Konsole geschrieben.

12.3.3 reqres Call Test

```
function reqresCallTest(){
  const result = $.ajax({
    url: "https://reqres.in/api/users/1",
    type: 'GET',
    async: false
  });
  return result;
}

function reqresCallTest(){
  for(var i = 0; i < 100; i++)
  {
    reqresCall();
  }
}
```

Hier wird die Geschwindigkeit eines **AJAX Calls** getestet. Um ein möglichst neutrales Resultat zu erhalten, wird eine externe API verwendet. Um genau zu sein, wurde die API von REQ|RES (REQ|RES, kein Datum) die als Backend für Frontend-Tests dient ausgewählt.

AJAX Calls sind asynchron und führen normalerweise nach Abschluss des Aufrufes eine Callback Funktion aus. Dies nützt mir hier natürlich nichts, da eine asynchrone Funktion mit meinem linearen Messungsverfahren nicht funktioniert. Deshalb hat der **AJAX Call** explizit die Option **async** auf **false** gesetzt. Da bereits eine Anfrage mehrere hunderte Millisekunden dauert, wird nur einen einzigen gestartet.

12.3.4 Self Call Test

Der Self Call Test sieht gleich aus wie der reqres Call Test mit lediglich zwei Unterschieden. Der aufgerufene URL ist `window.location.href+"test.txt"` was dazu führt, dass die Datei test.txt im selben Ordner abgefragt wird. Der zweite Unterschied ist, dass der gesamte Inhalt in einer **for** Schleife sitzt, die zehn Calls ausführt, da diese nicht solange wie beim reqres Test benötigen. Dieser Test zielt darauf ab herauszufinden, welcher Webserver die beste Antwortzeit auf den **AJAX Call** hat und ob eine besondere Beziehung zwischen einer Browser-Server Kombination zu sehen ist.

12.3.5 DOM Test

In diesem Test wird die Zeile `$('#hidden-dom').append('<div class="test item">'+i+'</div>')` 1000 mal ausgeführt, was ein neues **div** Element erstellt und dem versteckten Container anhängt.

12.4 test.txt

Diese Datei enthält lediglich ein Beispiel String wird beim Self Call Test benötigt.

13 Resultat

13.1 Daten

Bei diesem Projekt gibt es sehr viele Rohdaten, da mit jedem Browser das Script auf jedem Server ausgeführt wurde. Deshalb hier nur eine stark zusammengefasste Version der Durchschnitts-Resultate. Es wurden jeweils, zeitlich distanziert, zwei Datensätze generiert, welche jeweils aus 100 Testserien bestehen.

13.1.1 Chrome

Server	Loop	Console	Reqres	Self	DOM	Total
dk-matura-01	34.70 ms	39.36 ms	222.58 ms	180.57 ms	59.53 ms	536.73 ms
dk-matura-02	34.43 ms	39.88 ms	220.71 ms	194.57 ms	58.67 ms	548.25 ms
dk-matura-03	38.33 ms	41.68 ms	215.53 ms	72.10 ms	56.66 ms	424.29 ms

13.1.2 Firefox

Server	Loop	Console	Reqres	Self	DOM	Total
dk-matura-01	41.73 ms	31.17 ms	215.96 ms	131.79 ms	34.12 ms	454.75 ms
dk-matura-02	38.82 ms	29.13 ms	211.96 ms	134.74 ms	32.11 ms	446.75 ms
dk-matura-03	41.08 ms	31.36 ms	210.98 ms	34.72 ms	32.48 ms	350.61 ms

13.1.3 Safari

Server	Loop	Console	Reqres	Self	DOM	Total
dk-matura-01	66.21 ms	2.13 ms	208.20 ms	18.98 ms	29.39 ms	324.91 ms
dk-matura-02	79.43 ms	2.09 ms	211.73 ms	19.31 ms	30.67 ms	343.22 ms
dk-matura-03	56.06 ms	1.77 ms	211.36 ms	15.94 ms	28.67 ms	313.80 ms

13.2 Interpretation

Für die Interpretation dieser Daten werden die Tests jeweils in sinnvollen Gruppen aufgeteilt.

13.2.1 Loop & DOM

Diese beiden Tests konzentrieren sich auf die effektive Geschwindigkeit des Frontend's und dienen zur Analyse der JavaScript-Runtime. Die mit diesem Script gemessenen Resultate sind äusserst

interessant. Es scheint so, als ob Chrome, sehr nah gefolgt von Firefox und mit Safari als Schlusslicht, bei reinem JavaScript die Nase vorne hätte. Benötigt man also Rechenpower für JS Kalkulationen, ist einer der beiden Browser sicherlich die beste Wahl. Wenn man jedoch ein Frontend hat, das sehr viele Aktionen im DOM aufruft (also ein komplexes GUI zum Beispiel) scheint dies gerade Umgekehrt zu sein. Da ist Safari ein bisschen besser unterwegs als Firefox und ist fast doppelt so schnell wie Chrome.

13.2.2 Console

Dieser Test ist für die meisten Enduser nicht weiter interessant, da eine korrekt programmierte Website keine Console-Interaktionen enthalten sollte. Für die Web-Developer, wozu ich mich wenigstens im Hobby Bereich auch zähle, kann dieser Test sehr interessant werden. Die Tests wurden mit geschlossener Console durchgeführt.

Was sicherlich als Erstes ins Auge sticht ist die extreme Geschwindigkeit von Safari. Mit meinem aktuellen Wissensstand lässt sich dieses Phänomen nicht mit Gewissheit erklären, jedoch kann eine Vermutung aufgestellt werden. Aus Neugierde wurde der Test nochmals durchgeführt, jedoch mit offener Konsole, dabei ist folgendes aufgefallen: Safari hat noch nicht alle Elemente in der Console dargestellt, wenn der Test bereits abgeschlossen ist. Bei den anderen Browsern ist dies nicht der Fall. Das deutet auf eine asynchrone Funktionalität der Safari-Konsole hin, was zu solch einer extrem schnellen Ausführzeit bei geschlossener Console führen könnte. Kurz erklärt; Wenn der Befehl `console.log` aufgerufen wird, muss Firefox & Chrome zuerst das Element im GUI erstellen und das Script wartet dies ab, egal ob die Konsole geöffnet ist oder nicht. Safari hingegen erstellt diese GUI Elemente erst beim Öffnen/bei offener Konsole und dies höchst wahrscheinlich auch asynchron.

Firefox und Chrome sind in etwa gleichauf.

13.2.3 Regres & Self

Diese beiden Tests zeigen wie gut die Browser jeweils mit einer API zusammenarbeiten können. Der Regres Test hat nur einen sehr geringen Unterschied in der Runtime zwischen den Browsern und lässt somit keine eindeutigen Funde nachweisen. Der Self Test hingegen zeigt uns ein wenig mehr auf. Er widerspricht jedoch den Runtime-Script Tests. Dieser hat die Konstellation dk-matura-02 als Beste hervorgehoben, welche in diesem Test der langsamste war. Die Kombination dk-matura-03 war dafür beim vorherigen Projekt der Langsamste, zeigt hier jedoch die beste Geschwindigkeit auf.

Nun kommen wir aber noch zum auffallendsten Punkt: Die Geschwindigkeit nimmt mit dem Lauf der Zeit (also in der Reihenfolge, in der ich die Tests gemacht habe, Chrome Firefox Safari) enorm ab. Dies lässt sich auf den Cache der Webserver zurückführen.

Im Endeffekt muss auch beim Self Test erwähnt werden, dass wir nur beim ersten Test im Chrome eine starke Differenz sehen zwischen dk-matura-03 und den anderen Beiden, was bei einem genaueren Blick in die Zeiten auf eine grosse Auslastung am Server oder lokal zu führen ist. Wenn man die Daten ein wenig bearbeitet und filtert, muss man auch hier erkennen, dass nur von wenigen Millisekunden Unterschied zu sprechen ist.

13.3 Fazit

Fassen wir zuerst die Resultate nochmals kurz zusammen

1. Bei einer grossen Rechenkapazität wählt man am besten Chrome oder Firefox
2. Bei einem komplexen GUI/vielen DOM Aktivitäten hat Safari die Nase vorn
3. Bei der Frontend-Entwicklung mit starkem Console gebrauch kann Safari besser geeignet sein
4. Die **AJAX Calls** liefern keine schlüssigen Resultate

Würde ich mit diesen Daten eine Empfehlung wagen? - Ja:

Verwendet den Browser, den ihr mögt und am besten euren Bedürfnissen entspricht. All diese Unterschiede sind im Bereich von einigen Millisekunden. Die Browser sind heute bereits so fortschrittlich, dass der Unterschied sehr klein ist und jeder Browser hat seine Stärken und Schwächen.

TEIL 6

Fazit

14 Resultate

Als erster Teil in meinem Fazit möchte ich auf die Resultate eingehen.

Das Projekt "Runtime Script" hat mich in meiner Annahme bestätigt, dass Apache, MariaDB & CentOS für VPS und ähnlich kleine VMs die beste Wahl ist. Jedoch bin ich ein bisschen ins Schwanken geraten, als das Projekt "Browser Script" dagegengesprochen hat. Im Endeffekt muss ich wohl eingestehen, dass es einen guten Grund gibt, weshalb auch heute immer noch stark debattiert wird, welcher Webserver, Browser, Datenbank, etc. der Beste ist. Es gibt so viele Einflüsse, die darauf einwirken und jedes System ist einzigartig, so dass es nie eine eindeutige Wahl geben wird.

Für mich persönlich hat diese Arbeit jedoch bestätigt, dass ich zukünftig weiterhin auf CentOS & Apache setzen werde.

Nun möchte ich noch die 3 Leitfragen versuchen beantworten:

14.1.1 Gibt es einen effektiv messbaren Unterschied der Zeiten?

Diese Leitfrage kann ich weder mit Ja noch mit Nein beantworten. Es hat bei allen Tests zeitliche Unterschiede gegeben. Bei manchen grössere, bei einigen sehr kleine. Schlussendlich muss auch hinterfragt werden, ob diese Unterschiede wirklich von der zu messender Instanz verursacht wurde oder nicht noch von anderen externen Einflüssen abhängig ist, was sich bei einigen Tests auch gezeigt hat.

14.1.2 Gibt es einen Zusammenhang zwischen Script-Runtime und Response-Time?

Wenn ich nur das Projekt "Runtime Script" betrachte, welches mit dem URL Call die Response-Time misst, würde ich die Frage mit Ja beantworten. Wo die sonstige Runtime hoch war, war auch die Response Time hoch und umgekehrt. Jedoch hat das Projekt "Browser Script" dies wieder in Frage gestellt, da die Resultate gegensätzlich waren. Um nun dies als widerlegt oder bestätigt zu kennzeichnen, wären weitere Tests und Analysen notwendig, die den Rahmen dieser Arbeit sprengen würden.

14.1.3 Hängt die Durchschnitts-Laufzeit von den minimalen und maximalen Zeiten ab?

Diese Frage ist die Einzige die ich mit gutem Gewissen mit Ja beantworten kann. Wir haben beim Projekt "Runtime Script" gesehen, dass die minimale und maximale Zeit sich meist im nahen Umfeld der Durchschnitts-Laufzeit befand.

15 Arbeitstechnik

Kommen wir nun zur Arbeitstechnik. Im Großen und Ganzen bin ich zufrieden mit der Arbeit. Es gab ein paar wenige Punkte, die ich jedoch festhalten möchte.

15.1 Probleme

Eines der Probleme auf die ich gestossen bin, war die Wahl des Textbearbeitungs-Programms. Zu Beginn hatte ich mit TexMaker, einem Editor für Latex, gearbeitet. Ich bin weiterhin überzeugt, dass Latex in vielen Fällen Word überlegen ist. Doch bin ich noch nicht genügend darin bewandert, um eine komplette Arbeit mit Diagrammen, Tabellen und vor allem Code-Blöcken zu schreiben. Speziell das Code-Highlighting hat mir in Latex Schwierigkeiten bereitet. Deshalb musste ich mitten in der Arbeit auf Word umsteigen, da mir die Zeit zur Ausarbeitung aller Probleme einfach gefehlt hätte. Dies hat mich einiges an Zeit gekostet.

Ansonsten hatte ich nur einige kleinere Probleme bei der Installation der 3 VMs, die ich jedoch mit ein wenig Logs durchforschen und der bereits mit Linux gesammelten Erfahrung, beseitigen konnte. Trotz allem hatte mich das sicher mindestens einen Abend gekostet.

15.2 Erkenntnisse

Eine der wichtigsten Erkenntnisse ist, dass ich mit der Wahl des Themas gleichzeitig was sehr Einfaches aber auch Komplexes ausgewählt habe. Nun, da ich diese Arbeit abgeschlossen habe, muss ich sagen, dass die Wahl von lediglich einem der beiden Projekte mir die Möglichkeit gegeben hätte diese mehr in die Tiefe zu bearbeiten und möglicherweise eine definitivere Antwort zu finden.

Eine freudige Erkenntnis durfte ich bei der Installation der Linux Maschinen und der Realisation der Projekte machen. Ich befasse mich mit dem Betriebssystem Linux nun seit ca. einem Jahr und hätte mich immer noch als blutigen Anfänger bezeichnet. Mit dem Programmieren habe ich zum Beginn meiner Lehre begonnen, vor ca. 8 Jahren. Auch da hätte ich mich noch als starken Anfänger bezeichnet, da ich meist kleinere Projekte entwickelt habe und dies bisher auch nur einmal in einem professionellen Umfeld. Doch durfte ich feststellen, dass ich beides mit nur wenig Hilfe und Recherche aus dem Internet durchführen konnte, was für mich ein grosser Erfolg war.

Die letzte Erkenntnis ist, dass das Dokumentieren vom Code sich als eine äusserst schwierige Sache entpuppte. Man muss beachten, dass man die Abläufe und Zusammenhängen zwischen den Funktionen und Objekten klar darstellt, ohne zu viele Details miteinzubeziehen, die den Leser verwirren könnten.

16 Persönlich

Für mich persönlich war die Arbeit, trotz dem unschlüssigen Resultat ein Erfolg. Ich durfte einiges dazulernen und meinen Fortschritt über die vergangenen Monate und Jahre erkennen. Vom Zeitmanagement her hätte ich ein wenig früher beginnen sollen, etwas dass ich in noch fast jeder Arbeit erwähnt habe. Die Resultate haben die Wahl meiner primären Distribution (CentOS) verstärkt und bewegen mich dazu, Firefox nochmals eine Chance zu geben, da ich bisher fast ausschliesslich Chrome verwendet habe.

Im Grossen und Ganzen bin ich zufrieden mit meiner Arbeit und möchte mit diesem Satz meine Arbeit nun auch abschliessen.

TEIL 7

Anhang

17 Quellen

- [1] w3techs, «Usage of operating systems for websites,» [Online]. Verfügbar: https://w3techs.com/technologies/overview/operating_system/all. [Zugriff am 08 01 2019].
- [2] w3techs, «Usage statistics and market share of Unix for websites,» [Online]. Verfügbar: <https://w3techs.com/technologies/details/os-unix/all/all>. [Zugriff am 08 01 2019].
- [3] w3techs, «Usage statistics and market share of Linux for websites,» [Online]. Verfügbar: <https://w3techs.com/technologies/details/os-linux/all/all>. [Zugriff am 08 01 2019].
- [4] w3techs, «Usage of web servers for websites,» [Online]. Verfügbar: https://w3techs.com/technologies/overview/web_server/all. [Zugriff am 08 01 2019].
- [5] StackOverflow, «Developer Survey Results 2018,» [Online]. Verfügbar: <https://insights.stackoverflow.com/survey/2018>. [Zugriff am 08 01 2019].
- [6] DB-Engines, «DB-Engines Ranking,» [Online]. Verfügbar: <https://db-engines.com/de/ranking>. [Zugriff am 08 01 2019].
- [7] «JSON CSV,» [Online]. Verfügbar: <https://json-csv.com>. [Zugriff am 21 12 2018].
- [8] jQuery, «jQuery,» [Online]. Verfügbar: <https://jquery.com/>. [Zugriff am 21 12 2018].
- [9] «REQ|RES,» [Online]. Verfügbar: <https://reqres.in/>. [Zugriff am 21 12 2018].
- [10] StackOverflow, „Increasing 504 timeout error,“ [Online]. Verfügbar: <https://stackoverflow.com/questions/10806928/increasing-504-timeout-error>. [Zugriff am 16 12 2018].
- [11] StackOverflow, „I can't start MySQL on linux - Error mysqld_safe,“ [Online]. Verfügbar: <https://stackoverflow.com/questions/19117487/i-cant-start-mysql-on-linux-error-mysqld-safe>. [Zugriff am 16 12 2018].
- [12] Apache, „core - Apache HTTP Server Version 2.4,“ [Online]. Verfügbar: <https://httpd.apache.org/docs/2.4/de/mod/core.html>. [Zugriff am 16 12 2018].
- [13] StackOverflow, „MySQL root access from all hosts,“ [Online]. Verfügbar: <https://stackoverflow.com/questions/11223235/mysql-root-access-from-all-hosts>. [Zugriff am 16 12 2018].
- [14] StackOverflow, „MySQL Error #1133 - Can't find any matching row in the user table,“ [Online]. Verfügbar: <https://stackoverflow.com/questions/12877458/mysql-error-1133-cant-find-any-matching-row-in-the-user-table>. [Zugriff am 16 12 2018].
- [15] StackOverflow, „How to get time difference between two timestamps in seconds with jQuery?,“ [Online]. Verfügbar: <https://stackoverflow.com/questions/21516809/how-to-get-time-difference-between-two-timestamps-in-seconds-with-jquery>. [Zugriff am 20 12 2018].

- [16] StackOverflow, „Push array items into another array,“ [Online]. Verfügbar: <https://stackoverflow.com/questions/4156101/push-array-items-into-another-array>. [Zugriff am 20 12 2018].
- [17] StackOverflow, „How do I encode a JavaScript object as JSON?,“ [Online]. Verfügbar: <https://stackoverflow.com/questions/10919965/how-do-i-encode-a-javascript-object-as-json>. [Zugriff am 20 12 2018].
- [18] StackOverflow, „<https://stackoverflow.com/questions/6685249/jquery-performing-synchronous-ajax-requests>,“ [Online]. Verfügbar: jQuery: Performing synchronous AJAX requests . [Zugriff am 20 12 2018].

18 Anhang

Da der Anhang eine grössere Anzahl Seiten beinhaltet, ist dies Digital als CD beigefügt. Ebenfalls kann dies Digital auf GitHub unter folgendem Link eingesehen werden:

<https://github.com/melvin-suter/Maturaarbeit>