Abschlussprüfung Winter 2018

Fachinformatiker Systemintegration

Integrationstest-Framework für Linux Pakete

Erstellung eines Frameworks zum Testen von Linux-Paketinstallationen mit Docker

Nürnberg 08.12.2017

Herr Karl Jonathan Kawohl
Identnummer: 2578976
Prüflingsnummer: 23322
E-Mail: john@owncloud.com,
Telefon: +49 151 637 49 502

Ausbildungsbetrieb:
ownCloud GmbH, Rathsbergstraße 17, 90411 Nürnberg
Projektbetreuer: Jürgen Weigert
E-Mail: jw@owncloud.com,
Telefon: +49 179 206 96 77

Store, Share, Work!

1. Einleitung:

Dieses Kapitel befasst sich mit der Vorstellung des Ausbildungsbetriebes und des Auftraggebers. Zusätzlich dazu werden Projektziele, Notwendigkeit und zeitlicher Ablauf des Projektes dargestellt. Fachspezifische Begriffe und Abkürzungen werden bei erstmaligem Gebrauch durch kursive Formatierung kenntlich gemacht und können im Glossar - siehe Anhang A - nachgeschlagen werden

1.1 Projektumfeld

Die ownCloud GmbH mit Sitz in Nürnberg, ist ein Open Source Software Hersteller. ownCloud entwickelt und vertreibt eine Vielzahl an Produkten für Enterprise *File Sync & Share*. Über 400 Unternehmen Weltweit setzen ownCloud in ihrem Betrieb ein. Und über 10 Millionen End-Nutzer weltweit nutzen den ownCloud *Server*. ownCloud hat zur Zeit 62 Mitarbeiter die teilweise auch von ihrem Homeoffice aus arbeiten. In Nürnberg arbeiten etwa 35 Mitarbeiter. Das Projekt wurde in den Räumen der ownCloud GmbH umgesetzt.

1.2 Projektziele

Die ownCloud GmbH stellt im Wesentlichen zwei Produkte her:
Den ownCloud-Server, und den ownCloud-*Client*.
Die ownCloud Software dient zum Austausch und gemeinsamen Bearbeiten von Dateien. Die Produkte ownCloud-Server und ownCloud-Client werden für 28 verschiedene Linux-Distributionen bzw. Distributionsversionen als *Linux-Pakete* bereitgestellt. Diese Pakete werden mit dem Open Build Service[1] hergestellt und ausgeliefert. Funktionstests der Software erfolgen über Unit-Tests während der Entwicklung und Integrationstests erfolgen nur auf einzelnen, ausgewählten Linux-Distributionen per Hand. Um die Funktionsfähigkeit der ownCloud Linux Pakete auch vollständig und automatisiert zu testen, soll ein Testframework entwickelt werden.

1.3 Projektbegründung

Das händische Testen und Überprüfen einer Paketinstallation ist sehr zeitaufwändig und verursacht somit auch hohe Kosten. Um Entwicklerressourcen effektiver einsetzen zu können, ist eine Testumgebung notwendig, die sowohl lokal ausgeführt als auch vollautomatisiert auf einem Server gestartet werden kann. Durch die automatisierte Überprüfung soll sehr viel Zeit gespart werden.

1.4 Projektabgrenzung

Das Projekt ist in mehrere Phasen untergliedert, von denen. Das Projekt wird von einer Person durchgeführt. Einige Handlungsschritte lassen sich parallelisieren, sodass am Ende des Projekts eine umfangreiche erste Testsuite zur Verfügung steht.

[1] (openSuse, 2017)

1.5 Projektanalyse

Dieses Kapitel befasst sich mit der Projektanalyse und Planung der einzelnen Projektphasen. Die vorhandenen Gegebenheiten werden in einer IST-Analyse aufgezeigt. Die zu erreichenden Ziele und eingesetzte Hardware in einem SOLL-Konzept dargestellt. Es wird zusätzlich eine Wirtschaftlichkeitsbetrachtung in Form einer Kosten-Nutzen-Analyse durchgeführt.

2.1 Zeitlicher Ablauf der Projektphasen

Es stehen insgesamt 38 Stunden für die Durchführung des Projektes zur Verfügung. Die zeitliche Planung wird in nachstehender Tabelle zusammengefasst. Die Dauer des Projektes erstreckt sich vom 04.12.17 bis 08.12.17.

Tätigkeit	Datum	Zeit
Planungsphase		
Vorgespräch mit Projektbetreuer	04.12.2017	1,5 h
Ist-Analyse: Analyse der vorhandenen Struktur	04.12.2017	2 h
Soll-Planung: Festlegung der Technologie	04.12.2015	4 h
Erstes Prototyping mit ubuntu:16.04 Container und ownCloud production Paketen	05.12.2015	8 h
Realisierung		
Scripting für ownCloud Server Production & Latest 9.1 : Ubuntu& Debian	06.12.2017	2h
Scripting für ownCloud Server Production & Latest 9.1 : openSuse & CentOS	06.12.2017	6h
Scripting für ownCloud Client für alle Distros	07.12.2017	4h
Qualitätssicherung		
Testlauf, Überprüfung und Fehlerbehebung && Code Anpassungen	07.12.2017	2 h
Abschlussphase		
Präsentation im Unternehmen	07.12.2017	1,5 h
Erstellung der Projektdokumentation	04.-08.12.2017	7 h
Gesamt		**38 h**

2.2 Ist Analyse der momentane Test-Situation für Linux-Pakete

Die ownCloud Produkte ownCloud-Client und ownCloud-Server werden unter anderem als Linuxpakete für viele verschiedene Distributionen mittels des open-build-service bereitgestellt. Zur Zeit werden die Linuxpakete nur händisch getestet: Dies ist ein aufwendiger Prozess: Der Entwickler/Paketierer stellt die Pakete bereit. Dann fährt er eine Virtuelle Maschine mit einer Linux Distribution hoch, installiert alle benötigten

Abhängigkeiten: Für den Server ist, dass, das so genannte LAMP-Stack. Also Linux (Betriebssystem), Apache (Webserver), MYSQL (Relationale Datenbank) und PHP.

Der ownCloud Server ist eine PHP Anwendung und benötigt folgende PHP Bibliotheken und Erweiterungen für eine Standard installation: Siehe Anhang

Diese php Erweiterungen haben auf unterschiedlichen Linux Distributionen unterschiedliche Namen und sind auch versionsabhängig.
Der ownCloud Server benötigt zum Zeitpunkt der Projekterstellung mindestens php5.6.
Hier beginnt die Komplexität:
Moderne Distributionen wie Ubuntu:16.04 werden von den Distributionsherstellern bereits mit php7.0 in den Standardpaketquellen ausgeliefert. Etwas „ältere“ Distributionen wie CentOS 7 haben nur php 5.4 in ihren Standardpaketquellen. Um jetzt mindestens php 5.6 zu installieren ist es nötig eine externe Paketquelle anzumelden, um dann die entsprechende PHP Version zu installieren und zu aktivieren.
Durch die unterschiedliche Paketbenennung in den unterschiedlichen Distributionen sowie die teilweise Anmeldung von Paketquellen von Drittanbietern, steigt die Komplexität der Installation. Der ownCloud-Server wird aus Sicherheitsbedenken nur mit einer minimalen Konfiguration paketiert und installiert nicht alle Abhängigkeiten automatisch mit.
Ein Server Paket bei dem alles an Abhängigkeiten mitgeliefert würde, hat das Potential zum Beispiel eine bestehende Wordpress Instanz in die Knie zu zwingen, wenn sich durch das mitgelieferte PHP die PHP Version ungewollt ändert.

Der ownCloud Client hingegen wird als Paket mit allen nötigen Abhängigkeiten ausgeliefert, um einen einheitlichen Stand auf allen Distributionen erzielen zu können. D.h. die Installlation des ownCloud Client Paketes installiert automatisch die Pakete mit die zur kompletten Installation benötigt werden. Insbesondere QT5.6.2. Die Sicherheitsbedenken kommen hier nicht zum Tragen, da der Client nicht mit dem Webserver auf dem System interagiert auf dem er installiert ist.

2.3 Soll-Konzept

Um die Funktionalität der Pakete auf den verschiedenen Linux-Distributionen zu überprüfen und um Regressionstests auf den vielen verschiedenen Linux-Distributionen durchzuführen sowie das homogene Verhalten auf allen von der ownCloud Gmbh unterstützten Linux- Distributionen zu überprüfen, soll ein Integrations-Testframework geschrieben werden, dass die Installation der einzelnen Pakete jeweils testet. Hierzu ist eine Testmatrix zu entwickeln, jeweils für Server und Client. Das Framework soll in *Shell*-Scripten geschrieben werden, um den Entwicklern einen einfachen Einstieg zu liefern. Als Ausgangspunkt für weitere Tests soll die Paketinstallation mindestens getestet werden auf:

1. Ubuntu 16.04 (Momentan beliebteste Distribution)
2. Debian 9
3. OpenSuse Leap 42.3
4. CentOS 7

Zu Testen sind für den Server die letzten zwei Releases der jeweiligen Major Versionen: (momentan) ownCloud-Server latest 9.1. und production sowie die letzte Version des ownCloud-Clients 2.3.3 und der aktuelle rc1: 2.4.0. Um möglichst schnell, auch lokal, testen zu können sind die Tests mittels Docker-Virtualisierung auszuführen. Dieses ergibt eine Anzahl von 16 Testpunkten. Als Grundlage dienen dafür die offiziellen Docker-Images welche auf https://hub.docker.com/ vorzufinden sind und von den jeweiligen Projekten gepflegt werden.
Als Testsuite wird eine Liste von sehr einfachen Tests verwendet, die als Shell Script funktionieren und als Grundlage für die Entwickler dienen, das Framework zu erlernen und um weitere Tests zu erweitern.
Ein Beispiel für einen Server Test ist die Überprüfung der http-Verbindung. Es wird erwartet die Login-Seite geliefert zu bekommen. Die Überprüfung der ownCloud-Clients soll durch überprüfen der Versionsnummer nach der Installation erfolgen.

Da es sich bei dem ownCloud-Client nicht um einen Webserver handelt, auf den über das Netzwerk zugegriffen werden kann, sondern um eine Desktop Applikation die mittels X-Server eine grafische Repräsentation erfährt entfällt ein Test auf eine Login-Seite.
Hier ist nach Möglichkeiten zu suchen wie Docker der Zugriff auf den X-Server[2] erlaubt werden kann, um auch optische Überprüfungen der Darstellung des ownCloud-Clients vornehmen zu können. An zusätzlichen Beispielen für ownCloud-Server und ownCloud-Client soll gezeigt werden wie die Matrix zu erweitern ist.

In Folgender Tabelle werden die zu testenden ownCloud Client und Server Versionen sowie die entsprechende Linux Distributionen als Matrix dargestellt. In diese Matrix sollten jeweils die Testergebnisse eingetragen werden können.

[2] (Wikipedia, 2017)

Linux Distribution/ ownCloud Version	ubuntu	CentOS	debian	openSUSE
ownCloud Server production				
ownCloud Server latest 9.1				
ownCloud Client production				
ownCloud Client testing				

2.4 Kosten–Nutzen Analyse

Beispiel: Müsste der Entwickler obige Virtuelle Maschinen nebst Abhängigkeiten selbst erstellen, pflegen und damit entwickeln wäre dies, verglichen mit der beabsichtigten Lösung mit der Docker sehr Zeitaufwändig, nur bedingt migrierbar, hätte einen hohen Ressourcenverbrauch (insbesondere Speicherplatz) und Änderungen währen relativ schwerfällig in den jeweils laufenden Maschinen vorzunehmen.

In der angestrebten Lösung kann jeder Test einzeln, bzw. auch alle Tests sequentiell auf jedem Rechner vorgenommen werden.

Die jeweiligen Distributionen werden jedes Mal aufs Neue hochgefahren und es werden dann auch jeweils die neuesten Paketversionen der Abhängigkeiten installiert. Dies ermöglicht es über die Installation der jeweiligen ownCloud Pakete hinaus festzustellen, wenn sich Abhängigkeiten ändern bzw. in Abhängigkeit installierte Pakete zu Installationsfehlern führen.

Es ist schwer ermittelbar wie hoch die Kosten bei falscher, bzw. nicht ausreichender Paketierung für das Unternehmen ownCloud sind. Durch die überaus komplexe

Datenlage, deren Erfassung auch nur in Teilen bereits mehr als eine Mann-Woche in Anspruch nehmen dürfte, entscheidet sich der Projektverantwortliche hier anstatt für eine quantitative für eine qualitative Bewertung des Kosten-Nutzen Aufwands:

Ungetestete Linux-Pakete bergen bei Nichtbeachtung der ownCloud Dokumentation [3]ein Potential an möglichen Fehlerursachen angefangen bei falsch konfigurierten *Apache vhosts*, über fehlende Abhängigkeiten bzw. falsch oder mit veralteten Versionen installierte Abhängigkeiten.
Eine direkte Bezifferung von dadurch entstehenden Folgekosten z.B. für den Support ist nur schwer ermittelbar, da die jeweiligen Kosten ganz von dem Setup bei dem jeweiligen Kunden abhängen.
Ein standardisiertes Verfahren, bei dem der verantwortliche Entwickler auch lokal und schnell Änderungen an den Linuxpaketen testen kann, erhöht die Agilität und die Entwicklungsgeschwindigkeit.

3. Umsetzung

In diesem Kapitel wird die Arbeitsumgebung, die benötigte Technologie sowie die angestrebte Architektur vorgestellt.
Alle Tests wurden auf einem Mac Book Air mit einem Intel Core i5 Prozessor mit 8GB Ram durchgeführt und mit Shell Scripten geschrieben und getestet. Zur Überprüfung wurde der Code unter https://github.com/Kawohl/abschlussprojekt mit einem Thinkpad mit openSuse:42.3 ausgecheckt und ausgeführt. Zum Vergleich des Ressourcenverbrauchs wurde außerdem eine virtuelle Maschine mit Ubuntu 16.04 Server erstellt und getestet.

3.1 Technische Grundlagen: Docker[4]

Docker ist eine Virtualisierungstechnologie die im Wesentlichen auf Technologien wie CGroups[5] und Namespaces[6] aufbaut, hierbei werden kleine gekapselte Virtualisierungen erstellt, die den Kernel des Hosts nutzen dabei aber auch komplette Betriebssysteme wiederspiegeln können.
Der wesentliche Vorteil zu Virtuellen Maschinen besteht in der Ausführung zur Laufzeit des Hosts, d.h. es muss kein zusätzliches System gebootet werden.
Vorausgesetzt das Image eines Betriebssystems oder eines Dienstes ist bereits auf der Festplatte des Hostrechners, kann der Dienst bzw. das Betriebssystem ohne Verzögerung gestartet werden.
Der gestartete Container nutzt dafür den Kernel des Host und legt die Konfiguration des zu startenden Betriebssystems beziehungsweise des Dienstes dazu.

[3] (owncloud.org, 2017)
[4] (docker.com, 2017.)
[5] (Wikipedia, 2017)
[6] (Wikipedia, 2017)

Durch das schnelle starten, drängt sich normalerweise eine weitere Abstrahierung verschiedener Dienste geradezu auf. Die Kapselung in Container schafft eine weitere Abstrahierungsebene, und lässt so gerade auch ein verbessertes Debugging bzw. eine gezielte Fehlersuche in Deployments zu.
In diesem Projekt geht es darum sich die Eigenschaften der Docker-Technologie: portabel, ressourcensparend und vollständig scriptbar, zu eigen zu machen und darauf aufbauend ein Testframework für die Linux-Paketinstallation vorzunehmen. Docker bietet auf https://hub.docker.com/explore/ die docker-images mit den Betriebssystemen bzw. services an. Die einzelnen Images werden von den *Maintainern* der jeweiligen Projekte hochgeladen.

3.2 Docker Syntax

Docker startet Prozesse in Isolierten Containern. Ein Container ist ein Prozess welcher auf einem host läuft. Wenn ein Administrator docker run ausführt, führt er einen Dienst aus, der isoliert vom Host ein eigenes Netzwerk, ein eigenes Dateisystem, und eigene Prozesse ausführt. docker run werden dann Parameter übergeben.

```
docker run -ti -v
$(pwd)/desktop/centos7/testing/install.sh:/install.sh -v \
$(pwd)/logs/:/logs centos:7 sh install.sh
```

Obige Zeile lässt sich so segmentieren und lesen

- „docker run“ startet den container
- „-ti“ startet tty
- „-v“ definiert ein Volume vom Host das in den Container gemountet wird
- EINGABE: „$(pwd)/server/ubuntu16.04/latestproduction/install.sh:/install.sh“ mountet das script install.sh in den Container
- AUSGABE: „-v $(pwd)/logs/:logs“ mountet das Verzeichnis /logs in den container.
- „ubuntu:16.04“ ist das Ubuntu Image um den Container zu erstellen.
- „sh install.sh“ der Befehl der ausgeführt wird sobald der Container gestartet wird

Mit diesem Befehl startet docker einen Container mit ubuntu 16.04 als Distribution und führt das Script install.sh aus. Es gibt zwei Ausgaben: Einmal die Shell während das Script läuft, zum anderen die Logfiles in denen nur relevante und parsbare Informationen stehen.

3.3 Installationsscripte Server :

Nachdem wir jetzt oben gesehen haben wie der Start eines Containers aussehen könnte möchte ich nun exemplarisch zwei Installationsscripte darstellen:
Mit diesen Scripten installiere ich zunächst Abhängigkeiten (Mit Gelb Markiert), Konfiguriere und starte die Datenbank (Mit Türkis markiert) und installiere ownCloud (Rosa markiert):

Installation auf ubuntu:16.04

```
#!/bin/bash
#update package lists
apt update

#install dependencies
apt install -y apache2 mariadb-server libapache2-mod-php7.0 \
    php7.0-gd php7.0-json php7.0-mysql jq php7.0-curl \
    php7.0-intl php7.0-mcrypt php-imagick \
    php7.0-zip php7.0-xml php7.0-mbstring wget vim sudo lynx </dev/null
#start apache web server
service apache2 start

#add signing key for owncloud repository
wget -nv
https://download.owncloud.org/download/repositories/9.1/Ubuntu_16.04/Releas
e.key -O Release.key

apt-key add - < Release.key

#add owncloud repo to sources list
echo 'deb
http://download.owncloud.org/download/repositories/9.1/Ubuntu_16.04/ /' >
/etc/apt/sources.list.d/owncloud.list

#update sources lists
apt update
#install owncloud
apt install -y owncloud-files
#add apache configuration
cat <<EOT > /etc/apache2/sites-available/owncloud.conf
Alias /owncloud "/var/www/owncloud/"

<Directory /var/www/owncloud/>
  Options +FollowSymlinks
  AllowOverride All

 <IfModule mod_dav.c>
  Dav off
 </IfModule>

 SetEnv HOME /var/www/owncloud
```

```
 SetEnv HTTP_HOME /var/www/owncloud

</Directory>
EOT

ln -s /etc/apache2/sites-available/owncloud.conf /etc/apache2/sites-
enabled/owncloud.conf
a2enmod rewrite
#restart apache
service apache2 restart
#start mysql
service mysql start
sleep 3
#create owncloud database user and database
mysql -u root -e "create database owncloud"
mysql -u root -e "create user 'owncloouduser'@localhost identified by
'admin'"
mysql -u root -e "GRANT ALL PRIVILEGES ON owncloud. * TO
'owncloouduser'@'localhost'"

#install owncloud

sudo -u www-data php /var/www/owncloud/occ maintenance:install --database
"mysql" --database-name "owncloud" --database-user "owncloouduser" --
database-pass "admin" --admin-user "admin" --admin-pass "admin"

#check if install worked (lynx is a text webbrowser, jq parses json output,
/etc/os-release outputs the distro)

. /etc/os-release
(echo "SUCCESS:$(lynx --dump localhost/owncloud/status.php| jq -r
.versionstring ) installed! System $PRETTY_NAME" || echo "FAIL:
Installation failed! System $PRETTY_NAME") >> /logs/server.install.log 2>&1
```

Wie man hier sieht, erfolgt die Installation nur mit den von der Distribution zur Verfügung stehenden Paketquellen sowie dem ownCloud Repository. Services wie Apache Webserver und mysql lassen sich wie gewohnt starten.
Die Namensgebung der Linuxpakete scheint konsistent zu sein. ownCloud bietet mit dem Befehl occ die Möglichkeit sich zu installieren und die nötigen Informationen als Parameter zu übergeben.
Die Installation muss dann als Webserver-User ausgeführt werden.

```
sudo -u www-data php /var/www/owncloud/occ maintenance:install
```

Diese Zeile initiiert den Installationsprozess.

```
--database "mysql" --database-name "owncloud" --database-user
"owncloduser" --database-pass "admin" --admin-user "admin" --
admin-pass "admin"
```

Sind die Parameter, die für die Installation übergeben werden müssen. [7]
Hierzu ist es notwendig, sudo zu installieren um den Befehl als WebServer-User ausführen zu können. Docker Container bzw. Prozesse in ihnen laufen normalerweise immer als root.
Der Output bei einer Erfolgreichen Installation in die Datei /logs/server.install.log sieht folgendermaßen aus:

```
SUCCESS:9.1.7 installed! System Ubuntu 16.04.3 LTS
```

Installation auf CentOS:7

```
#! /bin/bash
#update repo list
yum update -y
#install dependencies
yum install -y sudo
yum install -y  epel-release
yum -y install yum-utils
#add third party repo // needed for php7
rpm -Uvh https://mirror.webtatic.com/yum/el7/webtatic-release.rpm
yum install  -y curl  sudo    wget mariadb-server mariadb mariadb-server
php70w php70w-dom php70w-mbstring php70w-gd php70w-pdo php70w-json php70w-
xml php70w-zip jq php70w-curl php70w-mcrypt php70w-pear php70w-mysql lynx
setroubleshoot-server \
yum install httpd
#import ownCloud signing key
rpm --import
https://download.owncloud.org/download/repositories/production/CentOS_7/rep
odata/repomd.xml.key
#import ownCloud repo
wget
http://download.owncloud.org/download/repositories/production/CentOS_7/ce:s
table.repo -O /etc/yum.repos.d/ce:stable.repo
#update and clean repo cache
yum clean expire-cache
#install owncloud
yum install -y owncloud-files

#apache configuration
mkdir /etc/httpd/sites-available
mkdir /etc/httpd/sites-enabled
```

[7] (owncloud.org, 2017)

```
echo "IncludeOptional sites-enabled/*.conf" >> /etc/httpd/conf/httpd.conf

#start apache
httpd -k start

sleep 2
#start of mysql service by hand
/usr/libexec/mariadb-prepare-db-dir mariadb.service
/usr/bin/mysqld_safe &


#more apache config
cat <<EOT > /etc/httpd/sites-available/owncloud.conf
Alias /owncloud "/var/www/html/owncloud/"

<Directory /var/www/html/owncloud/>
  Options +FollowSymlinks
  AllowOverride All

 <IfModule mod_dav.c>
  Dav off
 </IfModule>

 SetEnv HOME /var/www/html/owncloud/
 SetEnv HTTP_HOME /var/www/html/owncloud/

</Directory>
EOT
ln -s /etc/httpd/sites-available/owncloud.conf /etc/httpd/sites-
enabled/owncloud.conf
#restart apache
httpd -k restart
sleep 3

#create database owncloud database, database user
mysql -u root -e "create database owncloud"
mysql -u root -e "create user 'owncloduser'@localhost identified by
'admin'"
mysql -u root -e "GRANT ALL PRIVILEGES ON owncloud. * TO
'owncloduser'@'localhost'"

#install owncloud
sudo -u apache php /var/www/html/owncloud/occ maintenance:install --
database "mysql" --database-name "owncloud" --database-user "owncloduser"
--database-pass "admin" --admin-user "admin" --admin-pass "admin"
. /etc/os-release

(echo "SUCCESS:$(lynx --dump localhost/owncloud/status.php| jq -r
.versionstring ) installed! System $PRETTY_NAME" || echo "FAIL:
Installation failed! System $PRETTY_NAME") >> /logs/server.install.log 2>&1
```

Der Output bei einer erfolgreichen Installation sieht folgendermaßen aus:

```
SUCCESS:10.0.4 installed! System CentOS Linux 7
```

Hier ist zu sehen, dass die Installation etwas aufwendiger ist. Es muss zusätzlich ein Third-Party Repository angemeldet werden, da CentOS7 von Haus aus nur mit PHP5.4 ausgeliefert wird und ownCloud mindestens PHP5.6 benötigt. Außerdem muss der mysql Prozess von Hand gestartet werden, da das CentOS7 Image ohne init.d bzw. systemd ausgeliefert wird. Das Installieren von Third-Party Paketquellen ist durchaus als problematisch anzusehen:
Einige Maintainer sind zwar bekannt und durchaus anerkannt, es ist jedoch nahezu unmöglich für diese Paketquellen offiziellen Support von einem Unternehmen zu bekommen. Durch die unbekannte Herkunft der Pakete öffnen sich für Unternehmen die diese einsetzen möchten Risikofelder, die in einem Unternehmenskontext zu beachten sind. Stichworte sind hier Haftung und Compliance.

3.4 Installationsscripte Desktop

ownCloud liefert den ownCloud Client mit allen Dependencies um einen komplett konsistenten Zustand auf allen unterstützten Betriebssystemen zu haben. Der ownCloud Client wird mit Hilfe des *QT Frameworks* für nahezu alle Desktop Plattformen gebaut, vom Windows Desktop über Apple OSX bis hin zu den zahllosen Linuxdistributionen. Dadurch, das ownCloud für den ownCloud Client alle benötigten Abhängigkeiten in einem Riesenpaket mitliefert sinkt der initiale Konfigurationsaufwand immens:
So reicht für die Installation auf einem ubuntu:16.04 ein einfaches:
(Hier nur die ownCloud Installation gelb hinterlegt)

```
#!/bin/bash
#update repolists
apt-get update
#add ownCloud repo
echo 'deb
http://download.opensuse.org/repositories/isv:/ownCloud:/desktop/Ubuntu_16.
04/ /' > /etc/apt/sources.list.d/owncloud-client.list
#update repolists
sudo apt-get update
#install ownCloud Client.
apt-get --allow-unauthenticated install -y owncloud-client
#get Distro Info
. /etc/os-release
#run ownCloud-Client and output.
(owncloudcmd --version | grep -q "$expected_client_testing_version" && echo
"SUCCESS: version $(owncloudcmd --version | head -1) installed! System:
$PRETTY_NAME" || echo "FAIL: ownCloud not installed\! ") >>
/logs/desktop.install.log 2>&1
```

Dieses gilt auch für die anderen Distributionen. Ein einfaches Hinzufügen des ownCloud Repositorys sowie die Installation des Clients zieht automatisch alle Abhängigkeiten mit, die benötigt werden. Zur Überprüfung ob der ownCloud Client installiert wurde, reicht dies aus.

. Mithilfe von `. /etc/os-release` und der Variable `$PRETTY_NAME wird` die Distribution ausgegeben.

Mit der Zeile:

```
(owncloudcmd --version | grep -q "ownCloud version" && echo "SUCCESS:
version $(owncloudcmd --version | head -1) installed! System: $PRETTY_NAME"
|| echo "FAIL: ownCloud not installed\! ") >> /logs/desktop.install.log
2>&1
```

Wird die Installation des Clients getestet. Es wird owncloud in der Kommandozeile ausgeführt `owncloudcmd` und die Version ausgegeben `–version`. Die Version wird mit Hilfe von `grep` und den Operatoren && bzw. || wird ein entsprechender Output an das Lofile angehängt.
Verschiedene Distributionen und Desktop Anwendungen haben jedoch auch verschiedene grafische Anpassungen. Dies müsste visuell getestet werden.
Mit den üblichen Docker Einstellungen ist dieses nicht möglich: Docker wird der Zugriff auf den XServer des Hosts verweigert.
Bei der Recherche ist eine Möglichkeit aufgefallen:
Es ist theoretisch möglich dem Container Zugriff auf den X-Server des Hosts zu geben. So könnten die Befehle um einen ownCloud ubuntu client auf einem openSuse host zu starten heißen[8]:

```
XSOCK=/tmp/.X11-unix
XAUTH=/tmp/.docker.xauth
xauth nlist :0 | sed -e 's/^..../ffff/' | xauth -f $XAUTH
nmerge -
docker run -ti -v $XSOCK:$XSOCK -v $XAUTH:$XAUTH -e
XAUTHORITY=$XAUTH
$(pwd)/desktop/ubuntu16.04/testing/install.sh:/install.sh -v
$(pwd)/logs/:/logs ubuntu:16.04 sh -c "install.sh; export
DISPLAY=:0.0; owncloud"
```

Hier ergeben sich wieder weitere Probleme die erledigt werden müssen: der ownCloud Client erfährt keine geeignete Repräsentation, weil die Fonts die das Gast System braucht vom Host nicht zur Verfügung gestellt werden. Hiermit steigt zum

[8] (Rehm, 2017)

einem wieder die Komplexität, zum anderen müssten für weitere Tests visuelle Tests vorgenommen werden
Es ist also möglich die GUI des ownCloud Clients auf dem Host System anzuzeigen, allerdings gibt es hier weitere Dinge wie die Fonts zu beachten.
Eine Automatisierung ist ohne weitere Technologie machbar und würde den Rahmen dieses Projektes überschreiten.
Als mögliche Frameworks für UI Tests böte sich zum Beispiel openQA[9] an. In Absprache mit dem Projektbetreuer, wurde die Recherche in diese Richtung daher eingestellt.

3.5 Abschliessende Bemerkungen Scripting/Paketinstallation

Je nach ausgewählter Distribution, ist die Installation der Pakete und ihrer Abhängigkeiten sehr einfach: alle nötigen aktuellen Pakete werden von den Standard Repositories der Linux Distribution bereitgestellt und die Benennung der Pakete erfolgt konsistent so dass es einfach ist sie zu finden. (Debian/Ubuntu)
Oder die Installation ist etwas komplexer: Aktuelle Versionen benötigter Pakete stehen in den Standard Repositories der Distributionen nicht zur Verfügung und Third Party Repositories müssen angemeldet werden. (CentOS).
Teilweise stellt einen auch die nicht-konsistente Benennung von Paketen zunächst vor Rätsel. (OpenSuse).
So heißen so gut wie alle PHP7-Pakete etwa so: php7-$modulname, Das Apache modul heißt: apache2-mod_php7 (Der Unterstrich bildet die Inkonsistenz.)

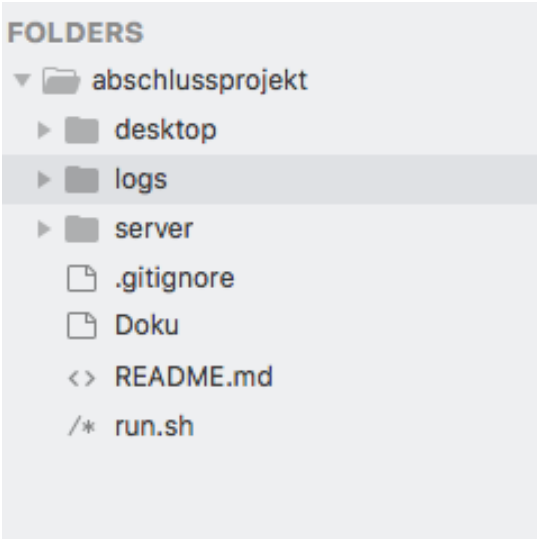
Das Scripting für Ubuntu und Debian, war demnach sehr schnell erledigt, CentOS und openSuse brauchten deutlich länger.

3.6 Struktur / Architektur

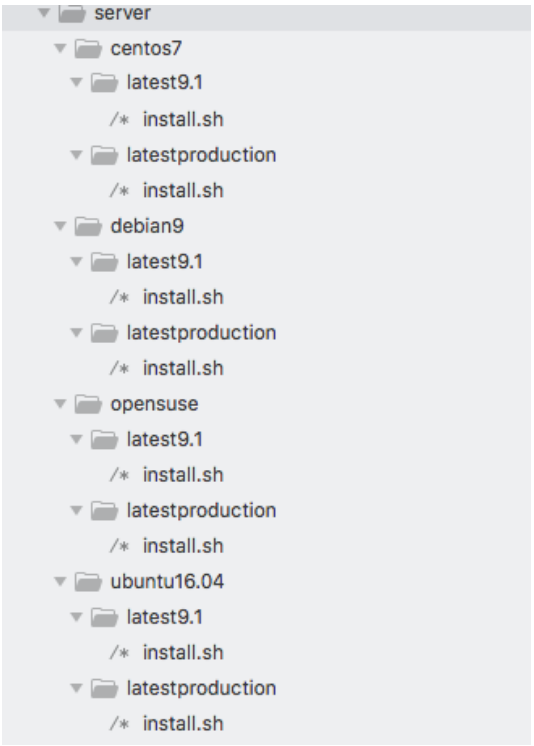
Das Projekt wurde mit Hilfe der Versionsverwaltung git erstellt und auf https://github.com/Kawohl/abschlussprojekt gehostet.

[9] (https://openqa.opensuse.org/, 2017)

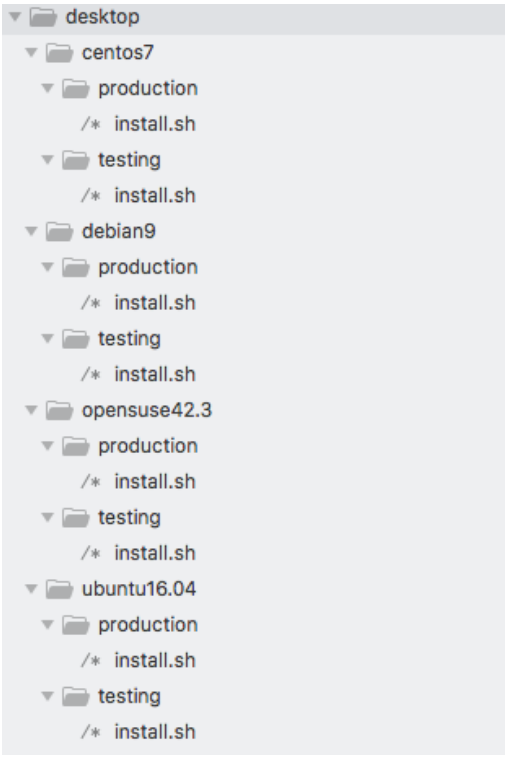
Um die in 2.3 dargestellte Testmatrix darzustellen wurde der Code folgendermaßen aufgebaut.



Im Desktop Ordner liegen alle Scripte für die Installation des Desktop Clients jeweils für production und testing, im Server Ordner alle Scripte zur Installation der ownCloud Server für production und latest 9.1.
Im Logs Ordner liegt jeweils ein Logfile für Server und Client. Für den Server ergibt sich folgende Struktur:



Für den ownCloud Client existiert die gleiche Struktur.



Die Konsistenz im Aufbau und Namensgebung soll es späteren Weiterentwicklungen ermöglichen konsequent weiter darauf aufzubauen und die Fehlererkennung vereinfachen.

3.6 Automatisierung:

Um alle Tests sequentiell abfolgen zu lassen wurde ein run.sh Script in dem Wurzelordner des Frameworks erstellt. Initial wurde das Script für die Server Installation auf Ubuntu 16.04 erstellt.
Um daraus eine komplette Testsuite zu erstellen wurde das run.sh script mit den Dockerbefehlen für alle geforderten Distributionen und Releases erstellt.

```
docker run –ti –rm=true -v
$(pwd)/server/ubuntu16.04/latestproduction/install.sh:/install.sh -v
$(pwd)/logs/:/logs ubuntu:16.04 sh install.sh
```

Dieses diente als Ausgangbasis für alle weiteren Scripte. Die Scripte werden sequentiell in der run.sh ausgeführt.
Die Redundanz von Strings in den Scripten ist dadurch zu erklären, dass es zukünftigen Mitentwicklern möglichst einfach gemacht werden sollte das Projekt fortzuführen und weiterzuentwickeln.
Außerdem lassen sich so einzelne Testpunkte hinzufügen und anpassen ohne weitere Systeme zu beachten.

Zu Beginn eines jeden Testlauf wird das aktuelle Datum und die Uhrzeit angezeigt. Dieses Script startet sequentiell für jeden Test einen dedizierten docker container. Nach Beendigung des install.sh Scriptes wird jeder Container wieder beendet und im Anschluss wird der Container vom Filesystem entfernt.
Der Output der Tests wird an jeweils das Server und Client Logfile geschickt. Nach dem der letzte Container beendet wurde wird in der Kommandozeile der Inhalt der beiden Logfiles angezeigt.

3.7 Vergleich mit dem Testframework Cucumber[10]

Cucumber führt bereits während des Programmierens Unit-Tests durch und es stehen Implementationen für verschiedenste Programmiersprachen zur Verfügung.. In diesem Projekt wird weniger die Programmierung des Produkts an sich getestet. Hier liegt das Augenmerk darauf, ob das Produkt richtig paketiert wurde, bzw. eine Installation per Programme mittels eines Paketmanagement Systems erfolgreich ist. Das Augenmerk liegt auf Integrationstests.

4. Fazit und Ausblick

Das Projekt wurde erfolgreich abgeschlossen: Es lässt sich mittels eines Testframeworks auch lokal überprüfen, ob die Installation des ownCloud Clients bzw. Servers erfolgreich war.
Die Tests wurden soweit für ownCloud Server für die Release Channels Production und latest 9.1 implementiert sowie für den ownCloud Desktop Client für die Release Channels Production und Testing.
Ein ubuntu Docker Container startet vollautomatisch innerhalb von weniger als einer Sekunde den Installationsprozess. Die Ubuntu VM auf Virtualbox braucht alleine schon 20 Sekunden für das booten. Der Festplattenplatz für die minimale Ubuntu VM gebootet beträgt schon 2.7 GB. Die des docker images liegt bei 123 MB. Die Durchschnittliche Memory Usage des Docker Containers liegt bei etwa 75 MB Die Ubuntu VM hat mindestens 1 GB als empfohlenen Ram.
Multipliziert man dieses mit 16 (Anzahl der Testpunkte) so wird ersichtlich weshalb die docker Testumgebung auch lokal und nicht nur auf einem großen dedizierten Testsystem sinnvoll ist. Die Ressourceneinsparung ist beträchtlich.
Das Ziel ein Testframework zu erstellen, das auch lokal ausführbar ist wurde erreicht. Im Rahmen des Projektes wurde festgestellt, dass das Testing der GUI des Desktop Clients mittels Docker einen größeren Aufwand birgt, es muss ein GUI Testing Framework eingebunden werden, da Shell dafür nicht geeignet ist. Als mögliche Option bietet sich z.b. openqa[11] an. Open QA lässt Maus und Tastatureingaben in der automatisiert testbar machen.

[10] (Cucumber, 2017)
[11] (https://openqa.opensuse.org/, 2017)

Künftige Weiterentwicklung:
Durch die modulare Struktur kann das Framework leicht ergänzt und erweitert werden. Mit den zwei dedizierten Logfiles ist die Basis geschaffen um zum Beispiel eine Webseite zu erstellen, die den Inhalt parst und eine entsprechende und Anwenderfreundliche Rückmeldung gibt ob die Installation des jeweiligen Pakets erfolgreich war. Mit den zu ergänzenden Tests kann dann auch noch festgestellt werden ob die Installation auch mit von anderen Drittanbieterpaketen kompatibel ist.
Die Tests können einfach um weitere Distributionen und Versionen erweitert werden.

Literaturverzeichnis

Cucumber. 2017. *Cucumber.io.* 08.. 12. Zugriff am 08.. 12. 2017. https://cucumber.io/docs.

docker.com. 2017. *https://www.docker.com.* 08.. 12. Zugriff am 08.. 12. 2017. https://www.docker.com.

https://openqa.opensuse.org/. 2017. *openqa.opensuse.org/.* 08.. 12. Zugriff am 08.. 12. 2017. https://openqa.opensuse.org/.

openbuildservice.org. 2017. *Open Build Service.* openSuse. 4.. 12. Zugriff am 4.. 12. 2017. http://openbuildservice.org/.

opensuse.org. 2017. *http://openbuildservice.org/.* opensuse. 04.. 12. Zugriff am 04.. 12. 2017. http://openbuildservice.org/.

owncloud.org. 2017. *doc.owncloud.org.* 08.. 12. Zugriff am 08.. 12. 2017. https://doc.owncloud.org/server/latest/admin_manual/configuration/server/occ_command.html?highlight=occ.

—. 2017. *doc.owncloud.org.* 08.. 12. Zugriff am 08.. 12. 2017. https://doc.owncloud.org/server/latest/admin_manual/installation/.

Rehm, Fabio. 2014. *http://fabiorehm.com/blog/2014/09/11/running-gui-apps-with-docker/.* 09.. 11. Zugriff am 08.. 12. 2017. http://fabiorehm.com/blog/2014/09/11/running-gui-apps-with-docker/.

Wikipedia. 2017. *https://en.wikipedia.org/.* Wikipedia. 08. 12. Zugriff am 08. 12. 2017. https://en.wikipedia.org/wiki/Cgroups.

—. 2017. *Wikipedia.* Wikipedia. 08.. 12. Zugriff am 08.. 12. 2017. https://en.wikipedia.org/wiki/Cgroups.

—. 2017. *Wikipedia.* 08. 12. https://en.wikipedia.org/wiki/Namespace.

—. 2017. *Wikipedia.* 08. 12. https://en.wikipedia.org/wiki/X_Window_System.

Anhang A Glossar:

Apache Vhost: Der Begriff *virtueller Host* bezieht sich auf die Praxis, mehr als ein Webangebot (z.B. www.company1.com und www.company2.com) auf einer einzigen Maschine zu betreiben.

Client: Die Software welche typischerweise auf dem Desktop installiert wird um Dateien vom Server zu synchronisieren.

File Sync & Share: Software zum Synchronisieren und Tauschen, bzw. Teilen von Dateien.

JSON: Die JavaScript Object Notation, kurz JSON ist ein kompaktes Datenformat in einer einfach lesbaren Textform zum Zweck des Datenaustausches zwischen Anwendungen.

Linux Paket: Sofware die mittels eines Paketverwaltungsmanagement-Tools installiert wird. Typischerweise haben Linux Distributionen von Haus aus ein Paketmanagement-Tool mit an Bord.

Maintainer: bezeichnet im Bereich der freien Sofware einen Hauptentwickler, der über die Aufnahme von Änderungen entscheidet und für einen bestimmten Fachbereich die Verantwortung trägt.

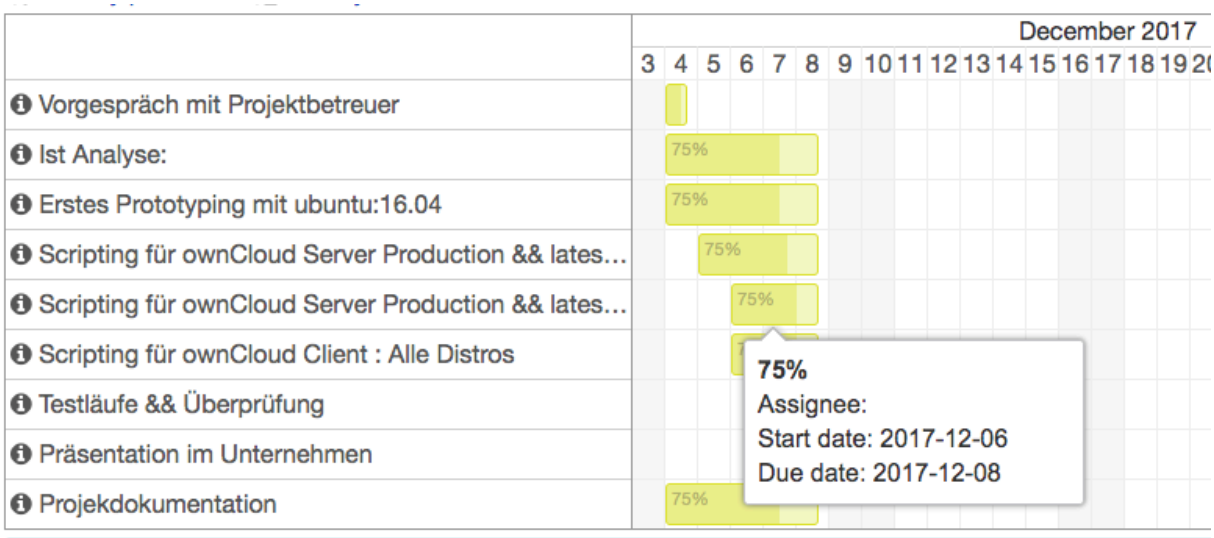
Paketabhängigkeiten: Kurz Abhängigkeiten: Software die von anderen Softwareteilen und Bibliotheken in der Installation abhängt. Diese Abhängigkeiten werden normalerweise durch Paketverwaltungstools verwaltet.

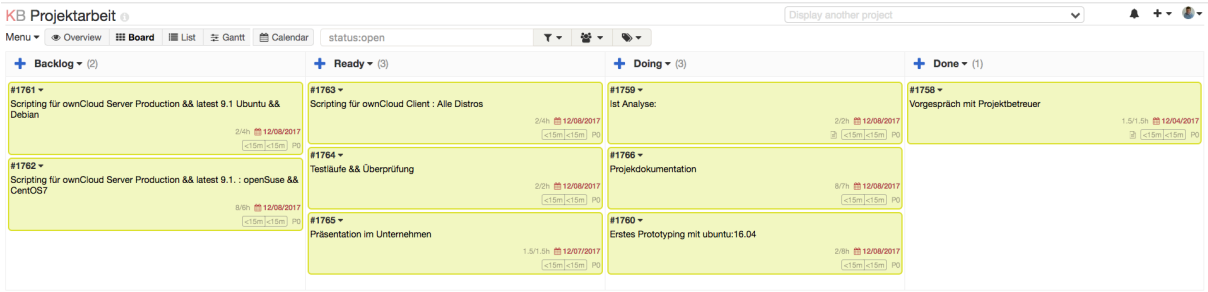
ownCloud Server: Server Software zum Speichern, Teilen und Freigeben von Dateien.

Shell: Die Benutzer Schnittstelle auf unixoiden Betriebssystemen: Der Benutzer kann in einer Eingabe Zeile Kommandos eintippen die der Computer sogleich ausführt.

QT Framework: Framework zur Entwicklung von CrossPlattform Apllikationen mit GUI. D.h. Mit einer Entwicklungsumgebung können Applikationen für OSX, Windows Linux u.a. geschrieben werden.

Anhang B Kanboard && Gantt Diagramm





Anhang C Benötigte PHP Module

Ctype	Zur Überprüfung von Schriftzeichen
cURL	Für Aspekte der HTTP user authentication
DOM	XML Dokumente über die DOM API bearbeiten
GD	Bilddokumente erstellen und manipulieren, für eine Vielzahl von Formaten z.B. GIF, PNG, JPEG, WBMP, und XPM.
iconv	Iconv ermöglicht es zwischen ISO und UTF8 zu wechseln.
intl	Für Nicht-ASCII Schriftzeichen
JSON	Mit JSON zu interagieren.
libxml	Dies is notwending damit die _DOM_, _libxml_, _SimpleXML_, and _XMLWriter_ Erweiterungen funktionieren . Benötigt wird außerdem libxml2, version 2.7.0 oder höher.
Multibyte String	Mit multibyte encoding schemata arbeiten.
OpenSSL	Symmetrische und asymmetrische Verschlüsselung und Entschlüsselung, PBKDF2, PKCS7, PKCS12, X509 und andere Verschlüsselungsmechanismen.

PDO	Für die Funktionsfähigkeit von pdo_msql .
Phar	Arbeiten mit PHP Archiven.
POSIX	Kommunikation mit der UNIX POSIX Funktionalität.
SimpleXML	XML Dateien als Objekte
XMLWriter	Um streams oder dateien aus XML daten zu generieren.
Zip	IP komprimierte Archive und Dateien lesen und zu schreiben.
Zlib	Zum Lesen und Schreiben von gzip (.gz) komprimierten Dateien.
pdo_mysql	Um mit MySQL & MariaDB zu arbeiten.

Anhang D Projektantrag

Winterprüfung 2017

Ausbildungsberuf
Fachinformatiker/-in Systemintegration

Prüfungsbezirk
Nürnberg Fachinformatiker-NB 829 (AP T2V1)

Herr Karl Jonathan Kawohl Identnummer: 2578976
Prüflingsnummer: 23322
E-Mail: john@owncloud.com, Telefon: 015163749502

Ausbildungsbetrieb: ownCloud GmbH Projektbetreuer: Herr Jürgen Weigert
E-Mail: jw@owncloud.com, Telefon: 01792069677

Thema der Projektarbeit

Entwicklung eines Integrationstest-Frameworks für Linux Pakete mit Hilfe von Docker.

1. Thema der Projektarbeit

Entwicklung eines Integrationstest-Frameworks für Linux Pakete mit Hilfe von Docker.

2 .Geplanter Bearbeitungszeitraum

Beginn: 03.12.2017
Ende: 08.12.2017

3. Projektbeschreibung

IST-Analyse:

Die ownCloud GmbH stellt im Wesentlichen zwei Produkte her: Den ownCloud-Server, und den ownCloud-Client. Die ownCloud Software dient zum Austausch und gemeinsamen Bearbeiten von Dateien.
Die Produkte ownCloud-Server und ownCloud-Client werden für 28 verschiedene Linux-Distributionen bzw. Distributionsversionen als Linux-Pakete bereitgestellt. Diese Pakete werden mit dem Open Build Service hergestellt und ausgeliefert. Funktionstests der Software erfolgen über Unit-Tests während der Entwicklung und Integrationstests erfolgen nur auf einzelnen, ausgewählten Linux-Distributionen.

SOLL-Analyse:

Um die Funktionalität der Pakete auf den verschiedenen Linux-Distributionen zu überprüfen und um Regressionstests auf den vielen verschiedenen Linux-Distributionen durchzuführen sowie das homogene Verhalten auf allen von der ownCloud Gmbh unterstützten Linux- Distributionen zu überprüfen, soll ein Integrations-Testframework geschrieben werden, dass die einzelnen Pakete jeweils testet. Hierzu ist eine Testmatrix zu entwickeln, jeweils für Server und Client. Das Framework soll in Shell-Scripten geschrieben werden, um den Entwicklern einen

einfachen Einstieg zu liefern. Als Ausgangspunkt für weitere Tests soll mindestens getestet werden auf:

1. Ubuntu 16.04 (Momentan beliebteste Distribution)
2. Debian 9
3. OpenSuse Leap 42.3
4. CentOS 7

Zu testen sind für den Server die letzten zwei Releases der jeweiligen Major Versionen: (momentan) ownCloud-Server 9.1.6 und 10.0.3 sowie die letzten beiden Versionen des ownCloud-Clients 2.3.3 und 2.4.0.

Um möglichst schnell, auch lokal, testen zu können sind die Tests mittels Docker-Virtualisierung auszuführen.

Dieses ergibt eine Anzahl von 16 Testpunkten. Als Grundlage dienen dafür die offiziellen Docker-Images welche auf https://hub.docker.com/ vorzufinden sind und von den jeweiligen Projekten gepflegt werden.

Als Testsuite wird eine Liste von sehr einfachen Tests verwendet, die als Shell Script funktionieren und als Grundlage für die Entwickler dienen, das Framework zu erlernen und um weitere Tests zu erweitern. Ein Beispiel für einen Server Test ist die Überprüfung der
http-Verbindung. Es wird erwartet die Login-Seite geliefert zu bekommen.

Die Überprüfung der ownCloud-Clients soll durch überprüfen der Versionsnummer nach der Installation erfolgen. Außerdem soll ein statischer Test durchgeführt werden, der überprüft welche Abhängigkeiten mit dem ownCloud-Client Paket installiert werden. Da es sich bei dem ownCloud-Client nicht um einen Webserver handelt, auf den über das Netzwerk zugegriffen werden kann, sondern um eine Desktop Applikation die mittels X-Server eine grafische Repräsentation erfährt entfällt ein Test auf eine Login-Seite. Hier ist nach Möglichkeiten zu suchen wie Docker der Zugriff auf den X-Server erlaubt werden kann, um auch optische Überprüfungen der Darstellung des ownCloud-Clients vornehmen zu können.
An zusätzlichen Beispielen für ownCloud-Server und ownCloud-Client soll gezeigt werden wie die Matrix zu erweitern ist.

4. Projektumfeld

Das Projekt wird von mir, betreut durch die Firma ownCloud GmbH in den Geschäftsräumen der ownCloud GmbH durchgeführt. Den Source Code stelle ich nach Absprache mit meinem Betreuer unter AGPL3 Lizenz öffentlich auf github.com

5. Projektphasen mit Zeitplanung

Geplante Projektplanung: 4h

- IST Analyse 1h
- SOLL Analyse 1h
- Kostenkalkulation 1h
- Projektablaufplan 1h

Geplante Projektrealisierung: 23 h

- Umsetzung:
- Initiales Scripting: für ein ubuntu 16.04(Docker-Container herunterladen und starten, Paket und Abhängigkeiten installieren. Docker-Container stoppen, Docker-Container

löschen.)
6h

- Scripting für die anderen 3 Betriebssysteme: (Docker-Container herunterladen und starten, Pakete und Abhängigkeiten installieren, Docker-Container stoppen, Docker- Container löschen.)

4h

- Testpunkte und Tests definieren und scripten für die letzten beiden ownCloud-Server Versionen der Releases ownCloud 9.1 und 10.0

5h

- Evaluation von verschiedenen Möglichen Tests (client ist keine Webapplikation, GUI muss überprüft werden)

4h

- Testpunkte und Tests definieren und scripten entsprechend der vorherigen Evaluierung. 3h

- Interne Präsentation für Entwickler und Administratoren 1h

Geplante Projektnacharbeiten: 8 h davon:

- Mögliche Probleme beheben 2h
- Projektdokumentation erstellen 6h

Gesamt: 35 h

6. Dokumentation zur Projektarbeit

Die Dokumentation wird enthalten:

1. Einführung in Docker und Docker Grundlagen-befehle
2. Vergleich bereits entwickelter Testframeworks z.B. Cucumber
1. Eine Übersicht der Testmatrix
2. Die einzelnen Test-cases
3. Eine Einführung zur weiteren Entwicklung für die Entwickler
4. Eine Prototypische Skizzierung einer späteren möglichen Automatisierung

5. Abschliessende Bemerkungen.

6. **Anlagen**

keine

1. **Präsentationsmittel**

- Präsentationsprojektor
- Handout gedruckt

1. **Hinweis!**

Ich bestätige, dass der Projektantrag dem Ausbildungsbetrieb vorgelegt und vom Ausbildenden genehmigt wurde. Der Projektantrag enthält keine Betriebsgeheimnisse. Soweit diese für die Antragstellung notwendig sind, wurden nach Rücksprache mit dem Ausbildenden die entsprechenden Stellen unkenntlich gemacht.

Mit dem Absenden des Projektantrages bestätige ich weiterhin, dass der Antrag eigenständig von mir angefertigt wurde. Ferner sichere ich zu, dass im Projektantrag personenbezogene Daten (d. h. Daten über die eine Person identifizierbar oder bestimmbar ist) nur verwendet werden, wenn die betroffene Person hierin eingewilligt hat.

Bei meiner ersten Anmeldung im Online-Portal wurde ich darauf hingewiesen, dass meine Arbeit bei Täuschungshandlungen bzw. Ordnungsverstößen mit „null“ Punkten bewertet werden kann. Ich bin weiter darüber aufgeklärt worden, dass dies auch dann gilt, wenn festgestellt wird, dass meine Arbeit im Ganzen oder zu Teilen mit der eines anderen Prüfungsteilnehmers übereinstimmt. Es ist mir bewusst, dass Kontrollen durchgeführt werden.