Universität Hamburg
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften

Clustertools

am Arbeitsbereich Wissenschaftliches Rechnen (DKRZ)

Kim Johannes Zülsdorff
6338392

Betreuer: Michael Kuhn und Julian Martin Kunkel

20.09.2013

Abstract

In dieser Dokumentation werden die Programme Docker und Lmod untersucht. Dazu wird jeweils eine Einführung geschildert, sowie eine kleine Anleitung der Installation (evtl. für unterschiedliche Systeme). Desweiteren werden die Handhabung und Probleme, die dabei auftreten, untersucht. Zu Docker werden einige Beispiele dokumentiert. Bei Lmod wird in dieser Ausarbeitung vorausgesetzt, dass dem Leser bereits bekannt ist, was Enviroment Modules sind. Abschließend wird aufgelistet, welche Vorteile die Nutzung von Lmod mit sich bringt.

Inhaltsverzeichnis

1 Docker **4**

1.1 Einführung in Docker . . . 4
1.1.1 Funktionsweise von Docker . . . 4
1.1.2 Erstellen eines Containers . . . 5
1.2 Ubuntu Precise 12.04 (LTS) (64-bit) . . . 6
1.2.1 Installation . . . 6
1.2.2 Probleme . . . 6
1.3 openSUSE 12.3 . . . 7
1.3.1 Installation . . . 7
1.3.2 Probleme . . . 7
1.4 Windows 7 Ultimate 64bit . . . 7
1.4.1 Installation . . . 7
1.4.2 Probleme . . . 9
1.5 Beispiele . . . 9
1.5.1 Hello World . . . 9
1.5.2 Hello World Daemon . . . 10
1.5.3 Python Web App . . . 11
1.5.4 Node.js Web App . . . 13
1.5.5 CouchDB Service . . . 15
1.5.6 Building an Image with MongoDB . . . 16
1.5.7 PostgreSQL Service . . . 18
1.5.8 Redis Service . . . 19
1.5.9 SSH Daemon Service . . . 21

2 Lmod **24**

2.1 Einführung in Lmod . . . 24
2.2 Installation . . . 25
2.3 Moduldateien in Lmod . . . 26
2.4 Anweisungen in Lmod . . . 27
2.4.1 Befehle im Terminal . . . 27
2.4.2 Befehle innerhalb eines Moduls . . . 29
2.5 Besonderheiten von Lmod . . . 30

3 Quellen **32**

3.1 Quellen für Docker . . . 32
3.2 Quellen für Lmod . . . 33

1 Docker

Docker befindet sich noch mitten in der Entwicklungsphase, weshalb es durchaus möglich ist, dass das ein oder andere bereits veraltet ist[1].

In diesem Bericht wird auf die Installation, die Probleme und die Handhabung von Docker auf verschiedenen Betriebsystemen eingegangen.

Alle Installationen wurden auf der virtuellen Maschine VirtualBox von Oracle getestet. Als Hauptspeicher haben alle Systeme 512MB zur Verfügung gestellt bekommen und ansonsten wurden alle Standardeinstellungen beibehalten.

Docker ist für Ubuntu ausgelegt, dennoch wurde im Rahmen dieser Arbeit getestet, inwiefern Docker auch auf Windowssystemen funktioniert.

Im Folgenden wird die Begründung der Wahl der Betriebsysteme erläutert:
Es wurden verschiedene Distributionen von Linux verwendet, um auf allen Betriebssystem die Lauffähigkeit zu testen. Zudem ist die offizielle Dokumentation bisher sehr eingeschränkt und deshalb umso schwieriger andere Distributionen zu verwenden, als vorgesehen.
Windows 7 ist momentan (wenn auch nicht das neueste, aber) ein sehr viel gebrauchtes und bewährtes Betriebsystem, deshalb wurde ebenfalls versucht Docker hier zum Laufen zu bringen. Tiefer getestet wurde hier aber nichts, da es bei der Ausführung wahrscheinlich keine Unterschiede zwischen den Betriebssystemen gibt.

1.1 Einführung in Docker

1.1.1 Funktionsweise von Docker

Der Name lässt es schon vermuten: Docker spielt auf die Container an[2]. Container sind dafür da, alles mögliche zu transportieren und das mit den verschiedensten Transportmitteln[3]. Mit Docker ist es nicht anders[4]. Man kann in einem virtuellen Container verschiedene Software-Komponenten unterbringen und diese auf unterschiedlichsten Hardwarekomponenten laufen lassen, ohne die Hardware entsprechend konfigurieren zu müssen[5]. So spielt es zum Beispiel keine Rolle, ob man den Container von einem Ubuntu-System, einer Cloud, einer VM, oder in einem Cluster aus verwendet[6].

[1] vgl. Unbekannter Autor (2013): Learn what Docker is all about. <http://www.docker.io/learn_more/> (Stand: 2013) (Zugriff: 2013-09-16).
[2] vgl. Unbekannter Autor (2013): Why Docker. <http://www.slideshare.net/fullscreen/dotCloud/why-docker/1> (Stand: 2013) (Zugriff: 2013-09-16).
[3] vgl. ebd.
[4] vgl. ebd.
[5] vgl. ebd.
[6] vgl. ebd.

In jedem Container ist die Software von allem anderen abgekapselt. So wird unter anderem verhindert, dass es Probleme mit den Abhängigkeiten beziehungsweise mit verschiedenen Versionen gibt[7]. Jede Version bekommt ihren eigenen Container, wo sie ausgeführt beziehungsweise modifiziert werden kann[8].

Docker arbeitet möglichst speicherarm[9]. So verwendet Docker im Gegensatz zu virtuellen Maschinen kein Betriebssystem (vgl. ebd.). Wenn man Programme kopiert, so wird nur das Programm kopiert, nicht aber die Bibliotheken[10]. Werden Änderungen vorgenommen, so speichert Docker die Änderung und hängt sie an das bereits vorhandene Programm an[11].

Um es dem Benutzer leichter zu machen, kann bei Änderung am Dateisystem ein neuer Container erstellt werden, ohne das der Benutzer eine neue Konfiguration vornehmen muss[12].

1.1.2 Erstellen eines Containers

Zum Erstellen eines Containers wird zunächst ein Dockerfile angelegt[13]. Mit einem Dockerfile erstellt man einen Container automatisch. Ohne Dockerfile geht es auch, dann muss man aber einen Container manuell basteln[14]. Dann bildet man zusammen mit dem Quellcode und den zugehörigen Ordnern von einem System aus, in dem Docker installiert ist, einen Container[15]. Diesen kann man mit dem Befehl `build` anschließend erstellen[16]. Dieser Container beinhaltet, abgekapselt von allem anderen und unabhängig von der Hardware, das Programm[17]. Diesen Container pusht man nun, wodurch ein Image erstellt wird[18]. Ein Zugriff auf diesen Container ist nun von jedem System aus möglich[19]. Es gibt eine search-Funktion und eine pull-Funktion[20]. Mit der search-Funktion kann man nach vorhandenen Containern suchen und mit der pull-Funktion holt man diese auf sein System[21]. Wenn man ein Container auf sein System gezogen (`pull`) hat, so kann man es dort via `run` ausführen[22]. Auf einem System können sich beliebig viele Container befinden[23].

[7] vgl. Unbekannter Autor (2013): Why Docker. <http://www.slideshare.net/fullscreen/dotCloud/why-docker/1> (Stand: 2013) (Zugriff: 2013-09-16).
[8] vgl. ebd.
[9] vgl. Unbekannter Autor (2013): The whole story - Docker, The linux container engine. <http://www.docker.io/the_whole_story/> (Stand: 2013) (Zugriff: 2013-09-16).
[10] vgl. ebd.
[11] vgl. ebd
[12] vgl. ebd.
[13] vgl. Unbekannter Autor (2013): The whole story - Docker, The linux container engine. <http://www.docker.io/the_whole_story/> (Stand: 2013) (Zugriff: 2013-09-16).
[14] vgl. ebd.
[15] vgl. ebd.
[16] vgl. ebd.
[17] vgl. ebd.
[18] vgl. ebd.
[19] vgl. ebd.
[20] vgl. ebd.
[21] vgl. ebd.
[22] vgl. ebd.
[23] vgl. ebd.

1.2 Ubuntu Precise 12.04 (LTS) (64-bit)

1.2.1 Installation

Als Erstes ruft man das Terminal auf (im Dash `Terminal` eingeben). Dort gibt man Folgendes in den Terminal ein[24]:

```
sudo apt-get update && sudo apt-get install
linux-image-generic-lts-raring
```

Dadurch wird der Kernel von der Version 3.2 auf die Version 3.8 aktualisiert. Dieser Vorgang kann je nach Internetleitung einige Minuten dauern. Danach muss ein reboot ausgeführt werden[25]:

```
sudo reboot
```

Jetzt benötigt man die PPA, dazu gibt man Folgendes in das Terminal ein[26]:

```
sudo apt-get install python-software-properties && sudo
add-apt-repository ppa:dotcloud/lxc-docker
```

Hier muss zwischendurch eine Bestätigung mit `Enter` erfolgen. Danach führt man ein update aus[27].

```
sudo apt-get update
```

Nun kommt die eigentliche Installation über den Befehl[28]:

```
sudo apt-get install lxc-docker
```

Dann lädt man sich den Basis 'ubuntu' Container herunter[29]:

```
docker run -i -t ubuntu /bin/bash
```

Und als Letztes gibt man noch `exit` ein[30].

1.2.2 Probleme

Beim Ausführen des 'Hello World!'-Beispiels erscheint die Warnung:

```
Warning: Docker detected local DNS server on resolv.conf. Using
default external servers: [8.8.8.8 8.8.4.4]
```

[24] vgl. Unbekannter Autor (2013): Requirements and Installation on Ubuntu Linux - Docker Documentation. <http://docs.docker.io/en/latest/installation/ubuntulinux/> (Stand: 2013) (Zugriff: 2013-07-26).
[25] vgl. ebd.
[26] vgl. ebd.
[27] vgl. ebd.
[28] vgl. ebd.
[29] vgl. ebd.
[30] vgl. ebd.

Ob dieses Problem nun besteht, weil es ein echtes Problem gibt oder lediglich daran liegt, dass eine virtuelle Maschine genutzt wird, ist bisher unklar.

Einige Beispiele konnten nicht erfolgreich abgeschlossen werden. Das kann darauf zurückgeführt werden, dass Docker einerseits ständigen Aktualisierungen unterliegt und circa im Wochentakt neue Versionen erscheinen, aber eventuell liegt es auch daran, dass Docker sehr anfällig gegenüber kleinen Fehlern ist. Schreibt man beispielsweise beim Befehl `touch Dockerfile`, Dockerfile klein, so funktioniert er nicht oder wenn man den Punkt am Ende eines build-Befehls weglässt, so wird dieser nicht ausgeführt.

1.3 openSUSE 12.3

1.3.1 Installation

Eine Installation mit einer anderen Distribution als die, die empfohlen wurde, scheint nicht ohne Weiteres mit Docker verwendbar zu sein.

1.3.2 Probleme

Die offizielle Installationsanleitung ist für Debian geschrieben, also nicht für openSUSE. Das hat zur Folge, dass alle Befehle, die zur Installation nötig sind und auf der Website beschrieben werden, auf openSUSE nicht funktionieren.

1.4 Windows 7 Ultimate 64bit

Im Vordergrund steht in diesem Bericht Ubuntu, deshalb werden hier nur kurz die Installation auf einem Windows-System sowie ein kurzer Test, ob es tatsächlich funktioniert, beschrieben. Näher wird aber nicht drauf eingegangen, da die Funktionsweise dieselbe ist, wie auf einem Ubuntu-System. Lediglich die Installationarten unterscheiden sich.

1.4.1 Installation

Windows wird nicht unterstützt, deshalb kann es passieren, wenn einige Pfade geändert werden, dass die Installation nicht genau so funktioniert, wie hier beschrieben[31]. Als Erstes braucht man Vagrant[32]. Dafür geht man auf http://downloads.vagrantup.com, klickt die neueste Version an und lädt den 2. Link (msi) herunter[33]. Danach installiert man Vagrant, in dem man die heruntergeladene Datei öffnet und den Installationsanweisungen folgt[34]. Dann muss man noch git mit ssh

[31] vgl. Unbekannter Autor (2013): Requirements and Installation on Ubuntu Linux - Docker Documentation. <http://docs.docker.io/en/latest/installation/windows/> (Stand: 2013) (Zugriff: 2013-07-26).
[32] vgl. ebd.
[33] vgl. ebd.
[34] vgl. ebd.

installieren[35]. Dazu geht man auf http://git-scm.com/download/win und lädt die Datei herunter[36]. Bei der Installation folgt man den Installationsanweisungen, aber statt Use Git Bash only, wählt man Run Git from the Windows Command Prompt[37]. Nun öffnet man die Kommandokonsole (`Windows-Taste + R`, dort `cmd` eintippen und `Enter` drücken).

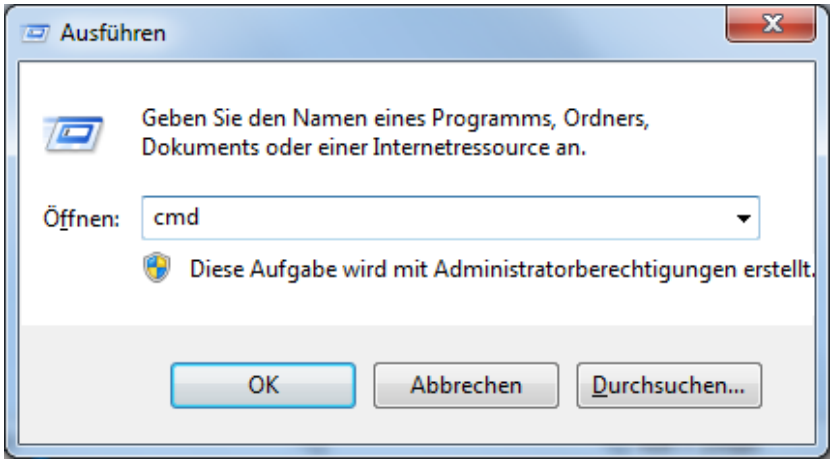

Abbildung 1: Kommandokonsole aufrufen

Jetzt muss das Ubuntu image heruntergeladen und gestartet werden[38]:

```
git clone https://github.com/dotcloud/docker.git
cd docker
vagrant up
vagrant ssh
```

Unter Umständen erscheint nun die Fehlermeldung[39]:

```
`ssh` executeable not found in any directories in the %PATH%
SSH client installed? Try installing Cygwin, MinGW or Git, a
contain an SSH client. Or use the PuTTY SSH client with the
authentication information shown below: [...]
```

Wenn also die PATH-Variablen noch nicht gesetzt sind, wird dies wie folgt vorgenommen[40]:

```
set PATH=%PATH%;C:\Program Files (x86)\Git\bin
```

Jetzt braucht man die IP und den Port des Ubuntu-servers[41]:

[35] vgl. Unbekannter Autor (2013): Requirements and Installation on Ubuntu Linux - Docker Documentation. <http://docs.docker.io/en/latest/installation/windows/> (Stand: 2013) (Zugriff: 2013-07-26).
[36] vgl. ebd.
[37] vgl. ebd.
[38] vgl. ebd.
[39] vgl. ebd.
[40] vgl. ebd.
[41] vgl. ebd.

`vagrant ssh-config`

Und gibt sich noch die root-Rechte[42]:

`sudo su`

Jetzt ist die Windows-Installation abgeschlossen, als Letztes kann man noch ein Hello-World! Programm starten[43]:

`docker run busybox echo Hello World!`

Bei der ersten Ausführung werden eventuell noch images geladen, aber letztendlich erscheint `Hello World!` auf der Konsole[44]. Mit `exit` (eventuell muss man es mehrmals eintippen) beendet man den Server (und die Konsole) wieder[45].

1.4.2 Probleme

Docker ist für Ubuntu ausgelegt, deshalb ist es umständlich, Docker auf einem Windows-System zu installieren. Deshalb ist es vonnöten, auf dem Windowssystem einen virtuellen Ubuntuserver zu simulieren.

1.5 Beispiele

Im Folgenden werden einige Beispiele gezeigt. Die Beispiele sind den Beispielen der Seite http://docs.docker.io/en/latest/[46] nachempfunden, wobei an einigen Stellen Änderungen vorgenommen wurden. Zunächst werden einige Beispiele vorgestellt, die erfolgreich abgeschlossen wurden und darauf einige Beispiele, die ohne Erfolg ausgeführt wurden. Um die öffentlichen Beispiele mit den hier bearbeiteten besser vergleichen, beziehungsweise nachgucken zu können, wurden die Originalüberschriften beibehalten.

1.5.1 Hello World

Als Erstes wird hier das Standard-Beispiel demonstriert: Die Ausgabe von Hello World![47].

Zunächst ruft man den Docker daemon auf, das macht man über `sudo docker -d &`[48]. Um jetzt das Programm zu starten, tippt man `sudo docker pull base` ein, um das base image

[42] vgl. Unbekannter Autor (2013): Requirements and Installation on Ubuntu Linux - Docker Documentation. <http://docs.docker.io/en/latest/installation/windows/> (Stand: 2013) (Zugriff: 2013-07-26).
[43] vgl. ebd.
[44] vgl. ebd.
[45] vgl. ebd.
[46] siehe Unbekannter Autor (2013): Docker Documentation - Docker Documentation. <http://docs.docker.io/en/latest/> (Stand: 2013) (Zugriff: 2013-09-16).
[47] vgl. Unbekannter Autor (2013): Hello world example - Docker Documentation. <http://docs.docker.io/en/latest/examples/hello_world/> (Stand: 2013) (Zugriff: 2013-09-16).
[48] vgl. ebd.

herunterzuladen und dann `docker run base /bin/echo Hello World!`, um Hello World! auf dem Terminal ausgeben zu lassen[49].

Erläuterung der Befehle[50]:

1. **docker run**: startet einen Befehl auf einem neuen Container
2. **base**: definiert das Image, wo der Befehl ausgeführt wird
3. **/bin/echo**: ist der Befehl, den wir im Container laufen lassen
4. **Hello World!**: ist der Parameter für den Echo-Befehl

Nach einem Neustart des Systems kann man das Beispiel ausführen, ohne das image wieder herunterladen zu müssen oder daemon zu starten[51].

1.5.2 Hello World Daemon

Im nächsten Beispiel wird ebenfalls hello world ausgegeben - allerdings im Sekundentakt[52].

```
CONTAINER_ID=$(sudo docker run -d ubuntu /bin/sh -c "while
true; do echo hello world; sleep 1; done")
```
[53]

Erläuterung der Befehle[54]:

1. **docker run -d**: startet einen Befehl auf einem neuen Container als daemon
2. **ubuntu**: definiert das Image, wo der Befehl ausgeführt wird
3. **/bin/sh -c**: ist der Befehl, den wir im Container laufen lassen
4. **while true; do echo hello world; sleep 1; done**: ist ein Mini-Script, dass beliebig lang im Sekundentakt Hello World zurück gibt.
5. **$CONTAINER_ID**: In dieser Variable wird die ID des Containers gespeichert

Der Befehl

```
sudo docker logs $CONTAINER_ID
```

listet alle bisherigen Ausgaben auf[55]

1. **docker logs**: Gibt alle logs des Containers zurück

[49] vgl. Unbekannter Autor (2013): Hello world example - Docker Documentation. <http://docs.docker.io/en/latest/examples/hello_world/> (Stand: 2013) (Zugriff: 2013-09-16).
[50] vgl. ebd.
[51] vgl. ebd.
[52] vgl. Unbekannter Autor (2013): Hello world daemon example - Docker Documentation. <http://docs.docker.io/en/latest/examples/hello_world_daemon/> (Stand: 2013) (Zugriff: 2013-09-16).
[53] vgl. ebd.
[54] vgl. ebd.
[55] vgl. ebd.

2. **$CONTAINER_ID**: ID des Containers

```
sudo docker attach $CONTAINER_ID
```

Normalerweise laufen die Ausgaben im Hintergrund ab. Dieser Befehl zeigt sie an. Nun wird im Sekundentakt auf der Konsole hello world ausgegeben[56].

1. **docker attach**: Holt den Hintergrundsprozess des Containers in den Vordergrund
2. **$CONTAINER_ID**: ID des Containers

Nun soll der Container wieder gestoppt werden. In diesem Vorgang werden alle laufenden Prozesse aufgerufen, der Container gestoppt und dann wieder alle laufenden Prozesse aufgerufen, um sicher zu gehen, dass der Container auch wirklich geschlossen wurde.

Die Zeile

```
sudo docker ps
```

listet alle laufenden Container auf[57]

```
sudo docker stop $CONTAINER_ID
```

1. **docker stop**: Stoppt einen Container
2. **$CONTAINER_ID**: ID des Containers

[58]

```
sudo docker ps
```

Jetzt sollte der Container nicht mehr angezeigt werden[59].

1.5.3 Python Web App

In diesem Beispiel wird ein bereits vorhandener Container verwendet. An diesem Änderungen vorgenommen und der dann als neuer Container wieder hochgeladen wird[60].

Der Befehl

```
sudo docker pull shykes/pybuilder
URL=http://github.com/shykes/helloflask/archive/master.tar.gz
```

[56] vgl. Unbekannter Autor (2013): Hello world daemon example - Docker Documentation. <http://docs.docker.io/en/latest/examples/hello_world_daemon/> (Stand: 2013) (Zugriff: 2013-09-16).
[57] vgl. ebd.
[58] vgl. ebd.
[59] vgl. ebd.
[60] vgl. Unbekannter Autor (2013): Python Web app example - Docker Documentation. <http://docs.docker.io/en/latest/examples/python_web_app/> (Stand: 2013) (Zugriff: 2013-09-16).

speichert eine Variable namens $URL, die auf einen so genannten Tarball von einer helloflask-WebApp zeigt[61].

```
BUILD_JOB=$(sudo docker run -d -t shykes/pybuilder:latest
/usr/local/bin/buildapp $URL)
```

Hier wird nun der neue Container mit der ID $BUILD_JOB erstellt. Für diesen Container wird der Befehl buildapp aus dem Container "shykes/pybuilder" verwendet. Als Parameter wird die Variable $URL genutzt[62].

```
BUILD_IMG=$(sudo docker commit $BUILD_JOB _/builds/github.com/
shykes/helloflask/master)
```

Mit commit wird ein neuer Container erstellt. Der erste Parameter ($BUILD_JOB) ist die ID des neuen Containers, der gespeichert werden soll, der zweite Parameter gibt den Namen an. Die ID des neuen Containers wird in $BUILD_IMG gespeichert[63]. `WEB_WORKER=$(sudo docker run -d -p 5000 $BUILD_IMG /usr/local/bin/runapp)`

Erläuterung der Befehle[64]:

1. **-p 5000**: Teilt dem Hostsystem mit, dass die Kommunikation über den Port 5000 geschieht
2. **$BUILD_IMG**: Der Container, in dem der Befehl ausgeführt wird
3. **/usr/local/bin/runapp**: Der Befehl, der die App startet
4. **$WEB_WORKER**: ID des neuen Containers

Im Folgenden wird ein Test zur Prüfung der Funktionsweise beschrieben[65]:

```
sudo docker logs $WEB_WORKER
```

Nun sollte "Running on http://0.0.0.0:5000/" auf der Konsole erscheinen[66].

Der Befehl

```
WEB_PORT=$(sudo docker port $WEB_WORKER 5000)
```

speichert den privaten Port des Containers in $WEB_PORT[67].

```
curl http://127.0.0.1:$WEB_PORT
```

Wenn nun alles geklappt hat, erscheint auf der Konsole "Hello world!"[68].

[61] vgl. Unbekannter Autor (2013): Python Web app example - Docker Documentation. <http://docs.docker.io/en/latest/examples/python_web_app/> (Stand: 2013) (Zugriff: 2013-09-16).
[62] vgl. ebd.
[63] vgl. ebd.
[64] vgl ebd.
[65] vgl. ebd.
[66] vgl. ebd.
[67] vgl. ebd.
[68] vgl. ebd.

1.5.4 Node.js Web App

Hier wird wieder mit Hilfe eines vorhandenen Containers ein neuer erstellt[69]. Allerdings wird hier eine 'Hello Word'-Web Application mit Node.js mit Hilfe von CentOS erstellt[70]. Dafür werden drei Dateien benötigt: eine Dockerfile, um den Container zu erstellen, package.json, das die App beschreibt und deren Abhängigkeiten und eine index.js, welche die Web Aplication definiert[71].

Quellcode von package.json[72]:

```
{
"name": "docker-centos-hello",
"private": true,
"version": "0.0.1",
"description": "Node.js Hello World app on CentOS using
docker",
"author": "Daniel Gasienica <daniel@gasienica.ch>",
"dependencies": {
"express": "3.2.4"
}
}
```

Quellcode von index.js[73]:

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

// Constants
var PORT = 8080;

// App
var app = express();
app.get('/', function (req, res) {
res.send('Hello World\n');
});

app.listen(PORT)
console.log('Running on http://localhost:' + PORT);
```

Um eine leere Dockerfile zu erstellen, geht man in den Ordner, in dem der Container erstellt werden soll und tippt dort Folgendes ein[74]:

[69] vgl. Unbekannter Autor (2013): Running a Node.js app on CentOS - Docker Documentation. <http://docs.docker.io/en/latest/examples/nodejs_web_app/> (Stand: 2013) (Zugriff: 2013-09-16).
[70] vgl. ebd.
[71] vgl. ebd.
[72] vgl. ebd.
[73] vgl. ebd.
[74] vgl. ebd.

```
touch Dockerfile
```

Nun kann man die Datei öffnen und einige Codezeilen einfügen[75].

```
FROM centos:6.4
```

Dies definiert den Container, der als Basis verwendet wird[76].

```
# Enable EPEL for Node.js
RUN rpm -Uvh http://download.fedoraproject.org/pub/epel/6/i386/
epel-release-6-8.noarch.rpm
# Install Node.js and npm
RUN yum install -y npm-1.2.17-5.el6
```

Mit diesen Befehlen wird Node.js und npm für den CentOS-Container installiert. Node.js wird zum Starten der App benötigt. npm erstellt die Abhängigkeiten[77].

Der Befehl

```
# Bundle app source
ADD . /src
```

packt den Quellcode in den Container[78].

Der Befehl

```
# Install app dependencies
RUN cd /src; npm install
```

installiert die Abhängigkeiten der App, wobei npm verwendet wird[79].

```
EXPOSE 8080
```

Die App benötigt den Port 8080, damit eine Mitteilung an den Container möglich ist[80].

```
CMD ["node", "/src/index.js"]
```

CMD definiert die Laufzeit der App. In diesem Fall node und den Pfad zur App[81].

Nun wird der Container gebildet:

```
sudo docker build -t <Benutzername>/centos-node-hello .
```

Das -t -Flag erleichtert die spätere Lokalisation des Containers. Wichtig ist, dass man nicht den Punkt am Ende der Codezeile vergisst. Der Benutzername muss mindestens vier Zeichen lang sein[82].

---

[75] vgl. Unbekannter Autor (2013): Running a Node.js app on CentOS - Docker Documentation. <http://docs.docker.io/en/latest/examples/nodejs_web_app/> (Stand: 2013) (Zugriff: 2013-09-16).
[76] vgl. ebd.
[77] vgl. ebd.
[78] vgl. ebd.
[79] vgl. ebd.
[80] vgl. ebd.
[81] vgl. ebd.
[82] vgl. ebd.

Der Befehl

```
CONTAINER_ID=$(sudo docker run -d <Benutzername>/centos-node-hello)
```

startet den Container[83].

```
sudo docker logs $CONTAINER_ID
```

Nun sollte Folgendes als Rückgabe der App erscheinen[84]:

```
Running on http://localhost:8080
```

Um den Container nun zu testen, benötigt man zunächst den Port. Dazu gebe man Folgendes ein[85]:

```
sudo docker ps
```

Die Zahl vor 8080 ist der Port, den man braucht. Hier sei der Port 49160. Nun kann man die App über curl aufrufen[86]:

```
curl -i localhost:49160
```

Nun sollte das Ergebnis der App auf der Konsole zu sehen sein[87].

1.5.5 CouchDB Service

Bei folgendem Beispiel sollen zwei Datenvolumen sich dieselben Daten teilen - realisiert mit CouchDB[88].

```
COUCH1=$(sudo docker run -d -v /var/lib/couchdb
shykes/couchdb:2013-05-03)
```

Hier wird /var/lib/couchdb als erstes Datenvolumen erstellt[89].

Wenn Docker über localhost erreichbar ist, so reicht es aus, Folgendes zu schreiben[90]:

```
HOST=localhost
```

Sollte es mit localhost nicht möglich sein, so kann über `inspect` die öffentliche IP ermittelt werden.

Mit folgendem Code kann man die URL erstellen[91]:

[83] vgl. Unbekannter Autor (2013): Running a Node.js app on CentOS - Docker Documentation. <http://docs.docker.io/en/latest/examples/nodejs_web_app/> (Stand: 2013) (Zugriff: 2013-09-16).
[84] vgl. ebd.
[85] vgl. ebd.
[86] vgl. ebd.
[87] vgl. ebd.
[88] vgl. Unbekannter Autor (2013): Sharing data between 2 couchdb databases - Docker Documentation. <http://docs.docker.io/en/latest/examples/couchdb_data_volumes/> (Stand: 2013) (Zugriff: 2013-09-16).
[89] vgl. ebd.
[90] vgl. ebd.
[91] vgl. ebd.

```
URL="http://$HOST:$(sudo docker port $COUCH1 5984)/_utils/"
echo $URL
```

Damit wird die URL angezeigt, sodass man sie über einen Browser aufrufen kann[92].

```
COUCH2=$(sudo docker run -d -volumes-from $COUCH1
shykes/couchdb:2013-05-03)
```

Hierdurch wird ein zweites Volumen erstellt, allerdings mit den gleichen Daten von $COUCH1[93].

Um auf die zweite Datenbank über den Browser zugreifen zu können, gibt man Folgendes in die Konsole ein[94]:

```
HOST=localhost
URL="http://$HOST:$(sudo docker port $COUCH2 5984)/_utils/"
echo $URL
```

1.5.6 Building an Image with MongoDB

In diesem Beispiel soll wieder über ein Dockerfile ein Container erstellt werden[95]. Dieser lädt ein Image herunter und installiert die Software auf Ubuntu[96].

Zunächst erstellt man das Dockerfile[97]:

```
touch Dockfile
```

In diese Datei wird nun Folgendes hineingeschrieben[98]:

```
FROM ubuntu:latest
```

Dies definiert ubuntu als Basis-Image[99].

Die Zeilen

```
RUN apt-key adv -keyserver hkp://keyserver.ubuntu.com:80 -recv
7F0CEB10
RUN echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart
dist 10gen' | tee /etc/apt/sources.list.d/10gen.list
```

[92] vgl. Unbekannter Autor (2013): Sharing data between 2 couchdb databases - Docker Documentation. <http://docs.docker.io/en/latest/examples/couchdb_data_volumes/> (Stand: 2013) (Zugriff: 2013-09-16).
[93] vgl. ebd.
[94] vgl. ebd.
[95] vgl. Unbekannter Autor (2013): Building a Docker Image with MongoDB - Docker Documentation. <http://docs.docker.io/en/latest/examples/mongodb/> (Stand: 2013) (Zugriff: 2013-09-16).
[96] vgl. ebd.
[97] vgl. ebd.
[98] vgl. ebd.
[99] vgl. ebd.

fügen das offizielle 10gen zu der 'sources list' hinzu[100].

```
RUN dpkg-divert -local -rename -add /sbin/initctl
RUN ln -s /bin/true /sbin/initctl
```

Hier wird /sbin/initctl zu /bin/true geändert, damit Ubuntu denkt, das alles arbeitet[101].

Der Befehl

```
RUN apt-get update
RUN apt-get install mongodb-10gen
```

installiert MongoDB (vergl. ebenda).

Mit

```
RUN mkdir -p /data/db
```

erstellt den Standard-Ordner für MongoDB. Dadurch wird verhindert, dass man eine zusätzliche Konfigurationsdatei erstellen muss[102].

Die Zeile

```
EXPOSE 27017
```

legt den Standardport für MongoDB fest[103].

Damit ist die Dockerfile fertig und der Container kann über die Konsole erstellt werden[104]:

```
sudo docker build -t <Benutzername>/mongodb .
```

```
MONGO_ID=$(sudo docker run -d <Benutzername>/mongodb mongod
--noprealloc --smallfiles)
```

startet den Container[105].

```
sudo docker ps
```

Hier kann man den Port nachschauen, den der Container hat, in diesem Beispiel sei er 49158[106].

```
mongo -port 49158
```

Nun kann man die MongoDB verwenden[107].

[100] vgl. Unbekannter Autor (2013): Building a Docker Image with MongoDB - Docker Documentation. <http://docs.docker.io/en/latest/examples/mongodb/> (Stand: 2013) (Zugriff: 2013-09-16).
[101] vgl. ebd.
[102] vgl. ebd.
[103] vgl. ebd.
[104] vgl. ebd.
[105] vgl. ebd.
[106] vgl. ebd.
[107] vgl. ebd.

1.5.7 PostgreSQL Service

In diesem Beispiel wird gezeigt, wie PostgreSQL auf Docker installiert wird[108].

Zunächst öffnet man eine interaktive shell im Container[109]:

```
CONTAINER_ID=$(sudo docker run -i -t ubuntu /bin/bash)
```

Dann aktualisiert man die Abhängigkeiten[110]:

```
apt-get update
```

Die Befehle

```
apt-get install python-software-properties
apt-get install software-properties-common
```

installieren die Python-Software[111].

```
add-apt-repository ppa:pitti/postgresql
apt-get update
```

fügt Pitti's PostgreSQL-Ordner hinzu[112].

Nun wird PostgreSQL installiert[113]:

```
apt-get -y install postgresql-9.2 postgresql-client-9.2
postgresql-contrib-9.2
```

Nun wird der Benutzer docker hinzugefügt, der unter anderem Datenbanken und andere Benutzer erstellen kann[114]:

```
sudo -u postgres createuser -P -d -r -s docker
```

```
sudo -u postgres createdb -0 docker docker
```

erstellt die Datenbank docker vom Benutzer docker[115].

Damit remote-Verbindungen zur Datenbank möglich sind, muss Folgendes in der Datei `/etc/postgresql/9.2/main/pg_hba.conf` stehen[116]:

```
host all all 0.0.0.0/0 md5
```

[108] vgl. Unbekannter Autor (2013): PostgreSQL service How-To - Docker Documentation. <http://docs.docker.io/en/latest/examples/postgresql_service/> (Stand: 2013) (Zugriff: 2013-09-16).
[109] vgl. ebd.
[110] vgl. ebd.
[111] vgl. ebd.
[112] vgl. ebd.
[113] vgl. ebd.
[114] vgl. ebd.
[115] vgl. ebd.
[116] vgl. ebd.

Zusätzlich muss der Kommentar von listen_addresses in `/etc/postgresql/9.2/main/pg_hba.conf` wieder entfernt werden, sodass der Code wieder ausgeführt wird[117].

Mit der folgenden Zeile wird ein Image erstellt[118]:

```
sudo docker commit $CONTAINER_ID <Benutzername>/postgresql
```

Nun wird der PestgreSQL-Server über Docker gestartet[119]:

```
CONTAINER=$(sudo docker run -d -p 5432 -t <your
username>/postgresql /bin/su postgres -c '/usr/lib/postgresql/
9.2/bin/postgres -D /var/lib/postgresql/9.2/main -c
config_file=/etc/postgresql/9.2/main/postgresql.conf')
```

Um eine Verbindung mit dem Server zu bekommen, braucht man zunächst die IP-Adresse. Dafür wird wieder `inspect` genutzt, diesmal aber mit weiteren Parametern, sodass die IP-Adresse direkt zurückgegeben wird[120]:

```
CONTAINER_IP=$(sudo docker inspect $CONTAINER | grep IPAddress
| awk 'print $2' | tr -d ',"')
```

Nun kann man sich mit folgendem Code verbinden[121]:

```
psql -h $CONTAINER_IP -p 5432 -d docker -U docker -W
```

Jetzt ist es möglich mit dem Server zu arbeiten und zum Beispiel Datenbanken zu erstellen[122].

1.5.8 Redis Service

Nun zu einigen Beispielen, die ohne Erfolg bearbeitet wurden:

Im kommenden Beispiel wird die Funktionsweise in Kombination mit einem Redis-Service erläutert.[123].

Der Befehl

```
CONTAINER_ID=$(sudo docker run -i -t ubuntu /bin/bash)
```

startet den Container[124],

der Befehl

[117] vgl. Unbekannter Autor (2013): PostgreSQL service How-To - Docker Documentation. <http://docs.docker.io/en/latest/examples/postgresql_service/> (Stand: 2013) (Zugriff: 2013-09-16).
[118] vgl. ebd.
[119] vgl. ebd.
[120] vgl. ebd.
[121] vgl. ebd.
[122] vgl. ebd.
[123] vgl. Unbekannter Autor (2013): Running a Redis service - Docker Documentation. <http://docs.docker.io/en/latest/examples/running_redis_service/> (Stand: 2013) (Zugriff: 2013-09-16).
[124] vgl. ebd.

```
apt-get update
apt-get install redis-server
exit
```

installiert den Redis-Server und geht wieder raus[125].

Mit

```
sudo docker commit $CONTAINER_ID <Benutzername>/redis
```

wird der Container gespeichert[126].

Der Befehl

```
SERVICE_ID=$(sudo docker run -d -p 6379 <Benutzername>/redis
/usr/bin/redis-server)
```

startet den Service[127].

Nun gibt es zwei Tests[128]:

Beim ersten Test wird eine Verbindung zum Container mit redis-cli hergestellt[129]:

```
sudo docker inspect $SERVICE_ID
```

Mit diesem Befehl kann man die IP-Adresse des Containers ausfindig machen[130].

```
redis-cli -h <IP-Adresse> -p 6379
```

```
set docker awesome
```

```
get docker
```

Nun erscheint "awesome" auf der Konsole[131].

```
exit
```

Beim zweiten Test wird eine Verbindung mit dem Host-Betriebssystem mit redis-cli hergestellt[132].

Der Befehl

```
sudo docker port $SERVICE_ID 6379
```

holt den externen Port des Containers[133].

---

[125] vgl. Unbekannter Autor (2013): Running a Redis service - Docker Documentation. <http://docs.docker.io/en/latest/examples/running_redis_service/> (Stand: 2013) (Zugriff: 2013-09-16).
[126] vgl. ebd.
[127] vgl. ebd.
[128] vgl. ebd.
[129] vgl. ebd.
[130] vgl. ebd.
[131] vgl. ebd.
[132] vgl. ebd.
[133] vgl. ebd.

Folgende Zeile

```
ip addr show
```

holt die IP-Adresse des Host's[134].

```
redis-cli -h <Host-IP-Adresse> -p <Externer Port>
```

```
set docker awesome
```

```
get docker
```

Nun erscheint wieder "awesome" auf der Konsole[135].

```
exit
```

1.5.9 SSH Daemon Service

Dieses Beispiel soll zeigen, wie ein SSH-Service erstellt werden kann und wie auf ihn zugegriffen wird[136].

Der Befehl

```
sudo docker pull ubuntu
```

lädt das Ubuntu-Image herunter[137].

```
sudo docker run -i -t base /bin/bash
```

Das -i steht interactive und -t für terminal[138]. /bin/bash wird ausgeführt, um eine Kommandozeile zu bekommen[139].

Der Befehl

```
apt-get update
```

```
apt-get install openssh-server
```

installiert den OpenSSH-Server[140].

Die Zeile

```
which sshd
```

gibt einen Pfad zurück (/usr/sbin/sshd)[141].

[134] vgl. Unbekannter Autor (2013): Running a Redis service - Docker Documentation. <http://docs.docker.io/en/latest/examples/running_redis_service/> (Stand: 2013) (Zugriff: 2013-09-16).
[135] vgl. ebd.
[136] vgl. Unbekannter Autor (2013): Running an SSH service - Docker Documentation. <http://docs.docker.io/en/latest/examples/running_ssh_service/> (Stand: 2013) (Zugriff: 2013-09-16).
[137] vgl. ebd.
[138] vgl. ebd.
[139] vgl. ebd.
[140] vgl. ebd.
[141] vgl. ebd.

Der Befehl

```
mkdir /var/run/sshd
```

erstellt einen neuen Ordner[142].

```
/usr/sbin/sshd
```

```
passwd
```

Nun wird das Passwort des Servers gesetzt[143].

```
exit
```

Nun wird das neue Image erstellt[144]:

```
sudo docker ps
```

```
sudo docker commit <ID> dhrp/sshd
```

Mit

```
sudo docker run -d dhrp/sshd /usr/sbin/sshd -D
```

wird der Server gestartet[145].

Der Befehl

```
sudo docker ps
```

prüft, ob es tatsächlich läuft[146].

Die Zeile

```
sudo docker stop <ID>
```

stoppt es wieder[147].

Der Befehl

```
ID=$(sudo docker run -d -p 22 dhrp/sshd /usr/sbin/sshd -D)
```

startet den Server und öffnet den Port 22[148].

```
sudo docker port $ID 22
```

An dieser Stelle wird ein Port ausgegeben, an dieser Stelle sei es 49153[149].

---

[142] vgl. Unbekannter Autor (2013): Running an SSH service - Docker Documentation. <http://docs.docker.io/en/latest/examples/running_ssh_service/> (Stand: 2013) (Zugriff: 2013-09-16).
[143] vgl. ebd.
[144] vgl. ebd.
[145] vgl. ebd.
[146] vgl. ebd.
[147] vgl. ebd.
[148] vgl. ebd.
[149] vgl. ebd.

Nun verbindet man sich über die öffentliche IP-Adresse des Hosts[150]:

```
hostname
ifconfig
ssh root@192.168.33.10 -p 49153
```

Hier wird das Passwort abgefragt, das vorhin eingegeben wurde[151].

[150] vgl. Unbekannter Autor (2013): Running an SSH service - Docker Documentation. <http://docs.docker.io/en/latest/examples/running_ssh_service/> (Stand: 2013) (Zugriff: 2013-09-16).
[151] vgl. ebd.

2 Lmod

2.1 Einführung in Lmod

Diese Einführung bezieht sich auf den Artikel "Lmod – Alternative Environment Modules" von Jeff Layton (2013). Diesem zufolge haben sich bei Enviroment Modules zwei Methoden durchgesetzt, die Versionen der Compiler und MPI-Bibliotheken zu verwalten: das Flat Naming Scheme und das Hierarchical Naming Scheme[152]. Flat Naming Scheme habe den Nachteil, dass im Namen zwar ein Compiler und eine MPI-Bibliothek stünden, der Nutzer aber nicht sicher sein könne, dass die Anwendung tatsächlich diese verwende[153]. Sollte man jetzt einen neueren Compiler verwenden, die alte MPI-Bibliothek aber behalten wollen, so müsse man dennoch alles entfernen und neu laden[154]. Mache man hier irgendwo einen kleinen Fehler, so könne es schnell zu mysteriösen, schwer zu behebenden Problemen führen[155]. Im Hierarchical Naming Scheme bestünde das Problem, dass, wenn man einen Compiler geladen habe, man nur die darin enthaltenen Bibliotheken sehen könne. Aber nicht andere zur Verfügung stehenden Compiler, geschweige denn Bibliotheken, die sich in Unterordnern der anderen Compiler befänden[156]. Dadurch, dass man die Compiler nicht sehen könne, könne man sie auch nicht laden[157]. Diese Probleme versuche Lmod mit einem neuen Aufbau zu beheben - welches dem Hierarchical Naming Scheme recht ähnlich sei - näheres dazu in Moduldateien in Lmod[158].

Jeff Layton zufolge bringt Lmod alle Funktionalitäten, die TCL/C hat, ebenfalls mit[159]. Lmod sei damit in der Lage, alle Module, die in TCL geschrieben wurden, ebenfalls auszuführen[160]. Das "L" in Lmod steht für Lua, da dies die Primärsprache von Lmod ist.

Lmod sei in der Lage, sich bereits geladene Module zu merken, auch wenn sie wieder entfernt würden und andere Module geladen würden[161]. Wenn man nun einen vorher bereits geladenen Compiler wieder lade, so lade Lmod automatisch auch alle dazugehörigen Bibliotheken[162].

[152] vgl. Layton, Jeff (2013): Lmod – Alternative Environment Modules. <http://www.admin-magazine.com/HPC/Articles/Lmod-Alternative-Environment-Modules> (Stand: 2013) (Zugriff: 2013-09-13).
[153] vgl. ebd.
[154] vgl. ebd.
[155] vgl. ebd.
[156] vgl. ebd.
[157] vgl. ebd.
[158] vgl. ebd.
[159] vgl. ebd.
[160] vgl. ebd.
[161] vgl. ebd.
[162] vgl. ebd.

2.2 Installation

Nach Jeff Layton gibt es bei Lmod einige Vorraussetzungen, um installiert werden zu können: zwei Lua-Bibliotheken (luaposix und LuaFileSystem)[163]. Wenn man eine ältere Version als Lua 5.2 verwendet, braucht man zusätzlich Lua BitOp, ab der Version 5.2 ist dies aber nicht mehr erforderlich. Hier wird alles unter /usr/local installiert; man kann natürlich auch andere Pfade verwenden[164].

Um Lua zu installieren und die Bibliotheken einzubinden, führt man folgenden Code aus, nachdem man alles heruntergeladen hat[165]:

```
./configure –prefix=/usr/local

make

make install

cd /usr/local/bin

ls –s

./lua
```

Nun fügt man Lua der Umgebungsvariable hinzu, damit Lmod Lua finden kann[166]:

```
PATH=$PATH:/usr/local/bin/
```

Um nun Lmod zu installieren, führe man folgenden Code aus[167]:

```
./configure –prefix=/usr/local

make

make install

ls –s

cd lmod

ls –s

cd lmod

ls –s
```

Um nun Lmod für Benutzer ausführbar zu machen, müsse man noch eine kleine Konfiguration vornehmen[168]:

[163] vgl. Layton, Jeff (2013): Lmod – Alternative Environment Modules. <http://www.admin-magazine.com/HPC/Articles/Lmod-Alternative-Environment-Modules> (Stand: 2013) (Zugriff: 2013-09-13).
[164] vgl. ebd.
[165] vgl. ebd.
[166] vgl. ebd.
[167] vgl. ebd.
[168] vgl. ebd.

```
ln -s /usr/local/lmod/lmod/init/profile /etc/profile.d/modules.sh
ln - s /usr/local/lmod/init/cshrc /etc/profile.d/modules.csh
```

Nun müsse noch Folgendes in *.bashrc* ausgeführt werden, um das Initialisierungsskript ausführen zu können:

```
. /etc/profile.d/modules.sh
```

Jedesmal wenn man sich nun in die shell einlogge, würde modules.sh ausgeführt[169].

2.3 Moduldateien in Lmod

Jeff Layton zufolge werden alle Anwendungen über Compiler ausgeführt[170]. Hierfür sei der Compiler im "Core"[171]. Man benötige ebenfalls MPI-Bibliotheken[172]. Die Kombination aus beidem bilde den Compiler, da viele Anwendungen beides vorraussetzen würden[173]. Daraus habe Lmod eine Hierarchie entwickelt, die wie folgt aussieht[174]:

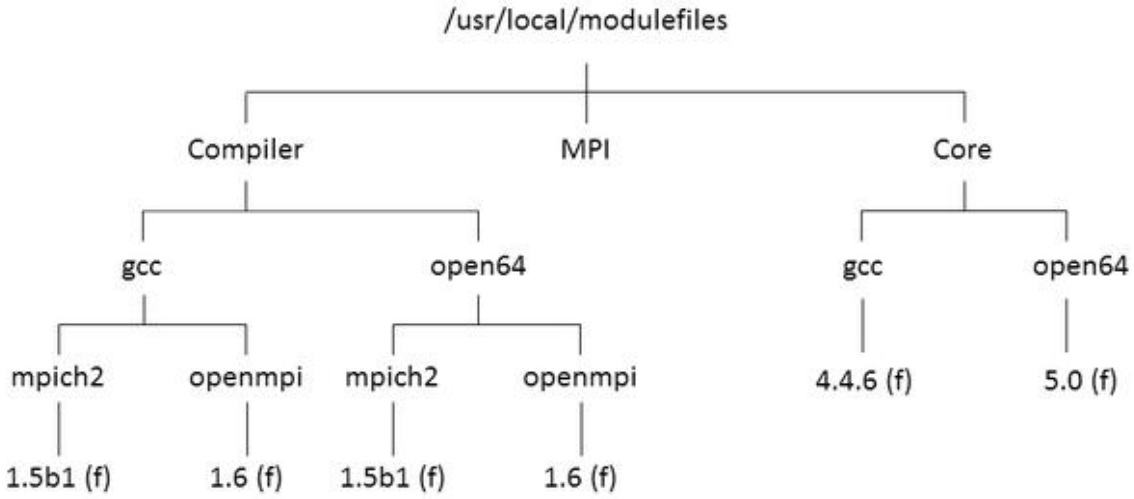

Abbildung 2: Moduldateiaufbau in Lmod

In diesem Beispiel verwendet der Anwender die beiden Compiler `gcc 4.4.6` und `open64 5.0` und die beiden MPI-Bibliotheken `mpich2-1.5b1` und `Open MPI 1.6`.

[169] vgl. http://www.admin-magazine.com/HPC/Articles/Lmod-Alternative-Environment-Modules
[170] vgl. Layton, Jeff (2013): Lmod – Alternative Environment Modules. <http://www.admin-magazine.com/HPC/Articles/Lmod-Alternative-Environment-Modules> (Stand: 2013) (Zugriff: 2013-09-13).
[171] vgl. ebd.
[172] vgl. ebd.
[173] vgl. ebd.
[174] vgl. ebd.

`(f)` markiert Dateien; unmarkierte Namen sind hier lediglich Ordner. In diesem Beispiel gibt es zwei MPI-Bibliotheken und zwei Compiler, also vier Möglichkeiten (`2*2`), die Anwendung auszuführen - mit den jeweiligen Konfigurationen[175].

Laut TACC wurde diese Hierarchie eingeführt, um Module, die weder Compiler noch MPI-Bibliotheken benöntigen in Core abzulegen, Module die einen Compiler benötigen, werden in Compiler gespeichert und Module, die MPI-Bibliotheken brauchen, werden im Ordner MPI abgelegt[176].

2.4 Anweisungen in Lmod

Im Folgenden werden einige Anweisungen erläutert, die für den Gebrauch von Lmod wichtig sind. Zuerst wird auf Ausführungen im Terminal eingegangen, danach auf Anweisungen in Modulen. Wie bereits erwähnt, kann man sowohl TCL als auch lua verwenden. Hat eine Datei keine Endung, so geht Lmod davon aus, dass es sich um eine TCL-Datei handelt, schreibt man jedoch in lua, so ist die Endung `.lua` erforderlich. TCL-Datein werden von Lmod automatisch in lua übersetzt, eine eigene Änderung von TCL zu lua aufgrund von Performance sei laut TACC nicht erforderlich, da der Übersetzer sehr effizient sei[177].

2.4.1 Befehle im Terminal

Alle Anweisungen werden nach dem Schema `module [Anweisung]` ausgeführt.

`help`
listet die wichtigsten Befehle auf. Optional kann man (beliebig viele) Module als Parameter angeben, zu denen man sich die Hilfe anzeigen lassen kann. In einem Modul werden diese über `help([[]])` deklariert.

`use [modulepath]`
fügt `[modulepath]` der Variablen MODULEPATH hinzu. Sobald der `MODULEPATH` geändert wird, werden alle Module entfernt und die zu ladenden Module, sofern möglich, geladen.

`unuse [modulepath]`
entfernt `[modulepath]` von der Variable MODULEPATH.

`echo $MODULEPATH`
zeigt den aktuellen modulepath an, es können beliebig viele Pfade hinzugefügt werden.

`spider`
listet alle Module auf, die gerade verfügbar sind. Zusätzlich prüft der Befehl alle modulefiles auf korrekte Syntax.

[175] vgl. http://www.admin-magazine.com/HPC/Articles/Lmod-Alternative-Environment-Modules

[176] vgl. Unbekannt (2013): Lmod – System Administrator Guide. <http://www.tacc.utexas.edu/tacc-projects/lmod/system-administrator-guide/module-hierarchy> (Stand: 2013) (Zugriff: 2013-11-23).

[177] vgl. Unbekannt (2013): Lmod – System Administrator Guide. <http://www.tacc.utexas.edu/tacc-projects/lmod/system-administrator-guide/tcl-vs-lua-modulefiles> (Stand: 2013) (Zugriff: 2013-11-23).

`spider [modul]`
listet alle verfügbaren Versionen auf. Es können beliebig viele Module als Argument angegeben werden.

`avail`
zeigt alle Module an, die vorhanden sind.
Man beachte, dass vorhanden ungleich verfügbar ist.

`list`
zeigt die gerade geladenen Module an.

`load [modul]`
lädt ein Modul. Es können beliebig viele Module als Argument angegeben werden. Über `load [modul]/[version]` kann man die Version spezifizieren, die man geladen haben möchte. Gibt man keine Version an, so wird die default-Version geladen (wird bei `module avail` mit einem `(D)` gekennzeichnet und ist standardmäßig die alphabetisch letzte Version, so ist zum Beispiel bei den beiden Versionen 9 und 11 Version 9 die default-Version, da $9 > 1$). Mit dem Befehl `ln -s [string] default` wird das Modul `[string]` als default festgelegt.
Sollten im `MODULEPATH` mehrere Ordner angegeben sein, auf den der `load`-Befehl zutrifft, so wird der erste Ordner gewählt, der in `MODULEPATH` steht.
Sollte eine bestimmte Version eines Moduls bereits geladen sein und man lädt eine andere Version eines Moduls, so wird die alte Version durch die neue ersetzt. Es läuft hierbei genau wie beim swap ab.

`try-load [modul]`
funktioniert genauso wie load, nur das keine Fehlermeldung ausgegeben wird, sollte das Modul nicht ladbar sein.

`unload [modul]`
entfernt ein Modul. Es können beliebig viele Module als Argument angegeben werden.

`swap [modul1] [modul2]`
entfernt [modul1] und lädt [modul2].

`purge`
entfernt alle Module.

`refresh`
lädt alle bisher geladenen Module neu.

`whatis [modul]`
gibt aus, wofür ein Modul gebraucht wird. Man setzt diese Variable innerhalb eines Moduls so: whatis("Beschreibung des Moduls").

`keyword [string]`
durchsucht alle Namen und whatis nach dem String und zeigt diese an. Mehrere Argumente sind zulässig. Alle Argumente sind mit einem oder verknüpft. Große Buchstaben kann man nicht finden.

`save`
sichert die geladenen Module. Als Argument kann man optional einen Namen für den Save angeben. Wird kein Argument angegeben, wird 'default' genommen.

`restore`
stellt gesicherte, geladene Module aus einem save wieder her. Als Argument kann man optional den Namen des save angeben. Alle zuvor (nicht) geladenen Module gehen verloren! Wird kein Argument angegeben, wird 'default' genommen.

`savelist`
listet alle save's auf.

`show [modul]`
gibt die Befehle eines Moduls aus. Mehrere Argumente sind angebbar.

`tablelist:`
gibt alle geladenen Module als Lua-Tabelle aus.

Spezielle Notationen für spider, list, avail und keyword:
`^[string]` Findet alle Module, die mit [string] beginnen.
`[string]` Findet alle Module, die [string] im Namen enthalten.
`[string]$` Findet alle Module, die mit [string] enden.

2.4.2 Befehle innerhalb eines Moduls

`prepend_path("PATH", [string])`
fügt vor die PATH-Variable `[string]` ein.

`append_path("PATH", [string])`
fügt hinter die PATH-Variable `[string]` ein.

`setenv("NAME", [string])`
weist dem Namen den Wert `[string]` zu.

`whatis([string])`
legt den Text fest, der bei der Eingabe von `module whatis` erscheint.

`help( [ [ [string] ] ] )`
gibt `[string]` aus, wenn `help` aufgerufen wird.

`load([string])`
lädt ein modulefile. Es können beliebig viele angegeben werden, die mit Kommata getrennt werden.

`family([string])`
Ein Modul kann einer Familie zugeordnet werden. Es kann immer nur ein Modul einer Familie zur Zeit geladen werden.

`prereq([string])`
Das Modul kann nur geladen werden, wenn das Modul mit dem Namen `[string]` bereits geladen wurde. Es können beliebig viele Module mit Kommata getrennt angegeben werden.

`conflict([string])`
Das Modul kann nur geladen werden, wenn das Modul mit dem Namen `[string]` nicht geladen wurde. Es können beliebig viele Module mit Kommata getrennt angegeben werden.

2.5 Besonderheiten von Lmod

Damit Lmod weiß, welcher Compiler mit zugehöriger Bibliothek verwendet werden soll, muss man dies nach Jeff Layton über die Variable `MODULEPATH` in der Moduldatei festlegen[178]. Siehe Bild 2, wo der Compiler gcc und die MPI-Bibliothek mpich2-1.5b1 verwendet werden[179]:

```
» set MODULEFILE_COMPILER modulefiles/Compiler/gcc/4.4.6/mpich2/1.5b1
» prepend-path MODULEPATH /usr/local/$MODULEFILE_COMPILER
```

Tippe man `module avail` ein, so erscheine normalerweise eine Nachricht wie "No Modulefiles Currently Loaded"[180]. Wenn man Lmod verwende, so erscheine eine Liste mit allen Compilern, die verfügbar seien, beziehungsweise diejenigen, die unter "Core" stünden, siehe Bild 2[181]. Sollten Compiler geladen sein, so würde zusätzlich angezeigt, welche MPI-Bibliotheken für den jeweiligen Compiler zur Verfügung stünden[182].

Laut Jeff Layton ist es eine sehr hilfreiche Funktion von Lmod, dass, wenn man den Compiler wechselt, es den alten Compiler automatisch entfernt, den neuen korrekt lädt und zudem die passende MPI-Bibliothek dazu lädt[183].

Pakete könnten mit Lmod geladen und wieder entfernt werden, außerdem würden folgende Shells unterstützt: bash, ksh, csh, tcsh und zsh[184]. Es sei ebenfalls für Perl und Python verfügbar[185].

Außerdem kann man nach Jeff Layton aktuelle Module in einer Datei speichern, sodass man sie beim nächsten Login wieder laden kann[186].

[178] vgl. Layton, Jeff (2013): Lmod – Alternative Environment Modules. <http://www.admin-magazine.com/HPC/Articles/Lmod-Alternative-Environment-Modules> (Stand: 2013) (Zugriff: 2013-09-13).
[179] vgl. ebd.
[180] vgl. ebd.
[181] vgl. ebd.
[182] vgl. ebd.
[183] vgl. ebd.
[184] vgl. Unbekannter Autor (2013): Lmod: Environmental Modules System. <http://www.tacc.utexas.edu/tacc-projects/lmod> (Stand: 2013) (Zugriff: 2013-09-12).
[185] vgl. ebd.
[186] vgl. Layton, Jeff (2013): Lmod – Alternative Environment Modules. <http://www.admin-magazine.com/HPC/Articles/Lmod-Alternative-Environment-Modules> (Stand: 2013) (Zugriff: 2013-09-13).

Lmod könne mit Lua, TCL und auch beidem gleichzeitig arbeiten[187]. Um bereits geschriebene TCL-Module nutzen zu können, müsse man nur einige wenige Zusätze schreiben (zum Beispiel den `MODULEPATH`)[188].

Lmod erlaube es, verschlüsselte Abhängigkeiten in die Module selbst zu integrieren[189].

Lmod ließe nur gültige Änderungen zu[190]. Sollte man etwas tun wollen, das zu einer nicht funktionierenden Version führe, so könne man dies mit Lmod nicht tun[191]. Außerdem informiere Lmod einen darüber, welche Version die richtige ist[192].

Für Administratoren sei es unter anderem hilfreich, dass aufgelistet wird, welche Benutzer welche Versionen wann verwendet hätten, was hilfreich sei, um zum Beispiel herauszufinden, welche Software installiert, aktualisiert, oder optimiert werden sollte[193].

Lmod messe die Effektivität der Programme, sodass eine eventuelle Optimierung erleichert werde[194].

Ein weiterer, nicht offensichtlicher Vorteil: Lmod werde stets weiterentwickelt, um neue Features verfügbar zu machen[195].

[187] vgl. ebd.
[188] vgl.ebd.
[189] vgl. Layton, Jeff (2013): Lmod – Alternative Environment Modules. <http://www.admin-magazine.com/HPC/Articles/Lmod-Alternative-Environment-Modules> (Stand: 2013) (Zugriff: 2013-09-13).
[190] vgl. ebd.
[191] vgl. ebd.
[192] vgl. Layton, Jeff (2013): Lmod – Alternative Environment Modules. <http://www.admin-magazine.com/HPC/Articles/Lmod-Alternative-Environment-Modules> (Stand: 2013) (Zugriff: 2013-09-13).
[193] vgl. Dubrow, Aaron (2013): Lmod: The SSecret Sauce"Behind Module Management at TACC. <http://www.tacc.utexas.edu/news/feature-stories/2012/lmod> (Stand: 2013) (Zugriff: 2013-09-12).
[194] vgl. ebd.
[195] vgl. Dubrow, Aaron (2013): Lmod: The SSecret Sauce"Behind Module Management at TACC. <http://www.tacc.utexas.edu/news/feature-stories/2012/lmod> (Stand: 2013) (Zugriff: 2013-09-12).

3 Quellen

3.1 Quellen für Docker

Unbekannter Autor (2013): Building a Docker Image with MongoDB - Docker Documentation. <http://docs.docker.io/en/latest/examples/mongodb/> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): Getting started - Docker, The linux engine. <http://www.docker.io/gettingstarted/#> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): Hello world daemon example - Docker Documentation. <http://docs.docker.io/en/latest/examples/hello_world_daemon/> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): Hello world example - Docker Documentation. <http://docs.docker.io/en/latest/examples/hello_world/> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): It's easy to start using Docker. <http://www.docker.io/gettingstarted/> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): Learn what Docker is all about. <http://www.docker.io/learn_more/> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): PostgreSQL service How-To - Docker Documentation. <http://docs.docker.io/en/latest/examples/postgresql_service/> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): Python Web app example - Docker Documentation. <http://docs.docker.io/en/latest/examples/python_web_app/> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): Requirements and Installation on Ubuntu Linux - Docker Documentation. <http://docs.docker.io/en/latest/installation/ubuntulinux/> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): Requirements and Installation on Ubuntu Linux - Docker Documentation. <http://docs.docker.io/en/latest/installation/windows/> (Stand: 2013) (Zugriff: 2013-07-26).

Unbekannter Autor (2013): Running a Node.js app on CentOS - Docker Documentation. <http://docs.docker.io/en/latest/examples/nodejs_web_app/> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): Running a Redis service - Docker Documentation. <http://docs.docker.io/en/latest/examples/running_redis_service/> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): Running an SSH service - Docker Documentation. <http://docs.docker.io/en/latest/examples/running_ssh_service/> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): Running the Examples - Docker Documentation. <http://docs.docker.io/en/latest/examples/running_examples/> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): Sharing data between 2 couchdb databases - Docker Documentation. <http://docs.docker.io/en/latest/examples/couchdb_data_volumes/> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): The whole story - Docker, The linux container engine. <http://www.docker.io/the_whole_story/> (Stand: 2013) (Zugriff: 2013-09-16).

Unbekannter Autor (2013): Why Docker. <http://www.slideshare.net/fullscreen/dotCloud/why-docker/1> (Stand: 2013) (Zugriff: 2013-09-16).

3.2 Quellen für Lmod

Dubrow, Aaron (2013): Lmod: The SSecret Sauce"Behind Module Management at TACC. <http://www.tacc.utexas.edu/news/feature-stories/2012/lmod> (Stand: 2013) (Zugriff: 2013-09-12).

Layton, Jeff (2013): Environment Modules – A Great Tool for Clusters. <http://www.admin-magazine.com/HPC/Articles/Environment-Modules> (Stand: 2013) (Zugriff: 2013-09-12).

Layton, Jeff (2013): Lmod – Alternative Environment Modules. <http://www.admin-magazine.com/HPC/Articles/Lmod-Alternative-Environment-Modules> (Stand: 2013) (Zugriff: 2013-09-13).

Unbekannter Autor (2013): Lmod: Environmental Modules System. <http://www.tacc.utexas.edu/tacc-projects/lmod> (Stand: 2013) (Zugriff: 2013-09-12).