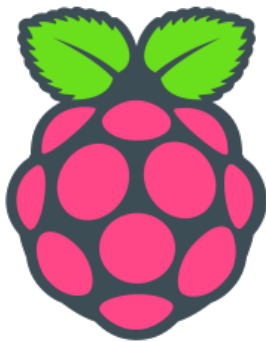


Eine Dokumentation von:

Jens Tucholski

US-FI 29 Gruppe B

Raspberry Pi Labor 2



Raspberry Pi



SMTP



Chia-Blockchain

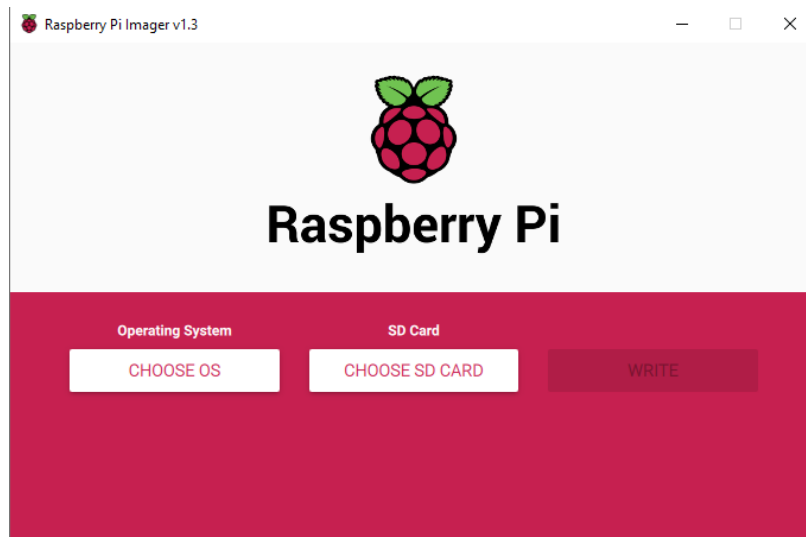
Datum: 30.01.2021

Inhaltsverzeichnis

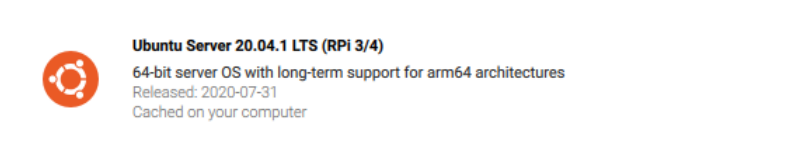
Schritt 1: Vorbereitung und Installation eines Betriebssystems.....	3
Schritt 2: Konfiguration.....	4
Root und SSH:.....	4
Wlan einrichten:.....	4
Updates, DE-Tastaturlayout, Zeitzone, IP und SSH Verbindung.....	5
Root Anmeldung per SSH.....	6
Schritt 3: Chia-Blockchain.....	7
Voraussetzungen schaffen mit Python, Python-Virtuelle- Umgebung + Pakete.....	7
Chia download, Chia installation.....	7
Python-Virtuelle-Umgebung, Chia richtig ausführen.....	8
Chia Kommandos Überblick.....	11
Plots erstellen mit Windows.....	12
Plots einbinden und Farmer starten.....	14
Schritt 4: SMTP.....	16
E-Mail Account erstellen.....	16
Python Skript mit smtpip und ssl.....	17
Crontab.....	20
SMTP E-Mails.....	22
Fazit:.....	24
Quellen:.....	24

Schritt 1: Vorbereitung und Installation eines Betriebssystems

1. Die SD-Karte des Raspberry Pi's komplett formatieren.
2. Mit dem Raspberry Pi Imager ein Betriebssystem installieren.



Ich Empfehle hier Ubuntu-Server 20.04.1 LTS 64-Bit



3. SD-Karte auswählen und die Installation starten.
4. Nach der Installation kann man die SD-Karte in seinem Raspberry stecken, zusätzlich einen Monitor und Tastatur verbinden und mit Strom versorgen.
5. Nachdem das Betriebssystem hochgefahren ist, muss man sich zunächst mit dem Login: ubuntu und dem selbigen Passwort, also "ubuntu" einloggen.
6. Jetzt wird man dazu aufgefordert ein neues Passwort zu vergeben, was wir auch direkt umsetzen. (**Achtung! US-Tastaturlayout**)

Nachdem wir diese Punkte alle erledigt haben, befinden wir uns nun auf unserem Betriebssystem Ubuntu-Server 20.04.1 LTS.

Schritt 2: Konfiguration

Root und SSH:

1. Nun vergeben wir dem User "root" ein Passwort, damit wir uns mit diesem zukünftig verbinden können.

Folgendes müssen wir dazu eingeben: ***sudo passwd root***

Nach Aufforderung vergeben wir also ein Passwort.

(Achtung! US-Tastaturlayout)

2. Jetzt verändern wir die sshd_config Datei um uns im späteren Verlauf auch per SSH als "root" anmelden zu können. Dies tun wir, indem wir mit dem Editor nano folgenden Pfad zur Datei aufrufen:

sudo nano /etc/ssh/sshd_config

3. In dieser Datei suchen wir folgenden Eintrag wie abgebildet und geben dort ein : ***PermitRootLogin yes***

```
# Authentication:
#LoginGraceTime 2m
#PermitRootLogin prohibit-password
PermitRootLogin yes
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10
```

4. Mit Strg + X, anschließend Y und Enter speichern wir dies.

Wlan einrichten:

1. Um Wlan zu aktivieren, müssen wir wieder mit dem Editor nano eine Datei konfigurieren. Dazu führen wir folgenden Befehl aus um diese Datei zu öffnen:

sudo nano /etc/netplan/50-cloud-init.yaml

2. In dieser Datei geben wir wie auf nachfolgendem Bild zu erkennen ist, folgendes ein. Zu beachten ist, dass es pro Leerzeichen-Block 4 Leertastenanschläge sind. Diese müssen stimmen und es darf kein Tab benutzt werden.

```
GNU nano 4.8
# This file is generated from information provided by the datasource. Changes
# to it will not persist across an instance reboot. To disable cloud-init's
# network configuration capabilities, write a file
# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:
# network: {config: disabled}
network:
  ethernets:
    eth0:
      dhcp4: true
      optional: true
  version: 2
  wifis:
    wlan0:
      optional: true
      access-points:
        "Dein-Wlan-Name":
          password: "Dein-Wlan-Passwort"
      dhcp4: true
```

3. Danach wieder mit Strg + X, Y bestätigen und Enter.
4. Nun müssen wir den Raspberry Pi neu starten:

sudo shutdown -r 0

Updates, DE-Tastaturlayout, Zeitzone, IP und SSH Verbindung

1. Nachdem wir uns als root eingeloggt haben, führen wir als erstes ein Update und Upgrade mit folgendem Befehl durch:

apt-get update; apt-get upgrade -y

(Manchmal kommt in diesem Schritt eine Fehlermeldung, das liegt daran, dass sich im Hintergrund wichtige Prozesse nun mit dem Internet verbinden und sich updaten. Hier hilft es entweder geduldig zu warten oder man führt einen weiteren Neustart aus. Dieser hat den Zweck, dass wenn man dies tut man eine Meldung bekommt wie lange diese Hintergrundprozesse noch geupdatet werden. Wie man sich auch entscheidet, können danach die Befehle fürs Update und Upgrade ausgeführt werden.)

2. Nachdem man den vermutlich etwas längeren Prozess des Updates/Upgrades vollzogen hat, ändern wir nun das US-Tastaturlayout ins deutsche, dies geschieht über die folgenden Befehle: ***dpkg-reconfigure keyboard-configuration***

3. Jetzt wählen wir wie abgebildet mit den Pfeiltasten aus, durchlaufen die nächsten Schritte indem wir unsere Vorlieben auswählen und mit Enter jedes mal bestätigen.



4. Jetzt teilen wir Ubuntu unsere Zeitzone mit, indem wir für z.B. Berlin folgenden Befehl benutzen:

timedatectl set-timezone Europe/Berlin

5. Als letzten Punkt möchten wir unsere IP-Adresse erfahren, um uns erfolgreich über ein anderes System auf dem Raspberry Pi einloggen zu können. Am einfachsten finden wir diese durch das nächste Kommando heraus und suchen dort die Kategorie "wlan0" mit dem Eintrag "inet". Die Zahlenkombination vor dem Slash, ist unsere Lokale IP-Adresse des Pi's die wir benötigen: ***ip a***

```
3: wlan0: <BROADCAST,MULTICAST>  
    link/ether dc:a6:32:46:8b:8e  
    inet 192.168.2.134/24  
    valid_lft forever  
    preferred_lft forever
```

Nun haben wir ersteinmal alle Schritte unternommen, um uns Erfolgreich von Monitor und Tastatur die an dem Raspberry Pi hängen zu verabschieden. Wir sind nun in der Lage, uns per SSH z.B. über die Windows PowerShell zu verbinden.

Root Anmeldung per SSH

In unserem Fall, verbinden wir uns über ein Windows 10 Pro System per PowerShell über SSH mit unserem Pi. Dazu rufen wir die PowerShell auf und geben dort folgendes ein:

ssh root@192.168.2.134 (Die IP-Adresse muss natürlich mit eurer ersetzt werden)

Anschließend muss das vorherig vergebene Passwort eingegeben werden und die Frage nach dem "Fingerprint" bestätigen wir auch durch ein "yes".

```
Windows PowerShell  
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.  
Lernen Sie das neue plattformübergreifende PowerShell kennen - https://aka.ms/powershell  
PS C:\Users\derde> ssh root@192.168.2.134
```

Schritt 3: Chia-Blockchain

Voraussetzungen schaffen mit Python, Python-Virtuelle-Umgebung + Pakete

1. Normalerweise ist Python Standardmäßig auf einer Linux Distribution vorinstalliert. Da dies aber absolut nötig ist um Chia erfolgreich installieren können, geben wir den Installationsbefehl ein. Sollte Python vorhanden und auf dem aktuellsten Stand sein, wird nichts weiter installiert. Sollte Python aber aus irgendeinem Grund nicht vorhanden sein, wird die neuste Version heruntergeladen und anschließend installiert.

apt-get install python3 -y

2. Als nächstes besorgen wir uns die neuste Version der Virtuellen Umgebung von Python samt allen benötigten Pakete mit diesem Befehl:

apt-get install python3-venv python3-distutils git -y

Wurde dies installiert geben wir noch diese Kommandos ein:

apt-get install -y build-essential python3-dev

Hiermit haben wir nun alle benötigten Python Pakete installiert und auf den neusten Stand gebracht. Dies versetzt uns nun in die Lage den nächsten Punkt auszuführen.

Chia download, Chia installation

1. Wir laden uns nun die aktuellste Version von Chia-Blockchain über die Seite github.com per git herunter.

git clone <https://github.com/Chia-Network/chia-blockchain.git>

2. Ist der Download beendet, wurde nun im Wurzelverzeichnis ein neues Verzeichnis Namens "chia-blockchain" mit den Installationsdateien angelegt. Um dort hin zu gelangen geben wir folgendes ein: ***cd /chia-blockchain***

3. Befinden wir uns nun im Verzeichnis /chia-blockchain, finden wir dort eine Skriptdatei Namens "install.sh", die den Installationsprozess startet. Damit wir diesen Prozess starten können, führen wir folgenden Befehl aus:

sh install.sh

Geschafft, nun sind alle Voraussetzungen erfüllt um mit Chia arbeiten zu können.

Python-Virtuelle-Umgebung, Chia richtig ausführen

1. Chia können wir nur über die Virtuelle Umgebung von Python ausführen, deshalb starten wir diese. Dazu müssen wir die Datei "activate" im Verzeichnis /chia-blockchain aufrufen. Da wir uns aktuell noch in diesem Verzeichnis befinden geben wir einfach folgenden Befehl ein. (**Achtung! Die beiden Punkte müssen mit eingegeben werden, die wiederum durch ein Leerzeichen getrennt sind**)

./activate

Wir erkennen, ob es geklappt hat, indem vor unserem Prompt nun ein (venv) steht. Dies bedeutet "Virtual-Environment".

```
root@ubuntu:~/chia-blockchain# ./activate
(venv) root@ubuntu:~/chia-blockchain#
```

Um aus dieser Umgebung nun wieder herauszukommen, genügt es wenn man einfach ***deactivate*** eingibt. Dies kann von überall aus erfolgen.

2. Wir bleiben aber ersteinmal in dieser Umgebung. Um in dieser Umgebung nun Befehle für Chia-Blockchain ausführen zu können, gibt es nun ein neues Schlüsselwort Namens "chia". Dieses benutzen wir auch direkt um zu überprüfen, ob auch wirklich die aktuellste Version installiert wurde. Wir geben folgendes ein: ***chia init***

Nun wird angezeigt, dass wir die neuste Version besitzen oder ob es eine neue Version gibt, die wir migrieren können.

3. Außerdem erhalten wir den Hinweis, dass keine "Keys" gefunden wurden, also erstellen wir uns nun welche. Diese "Keys" sind sehr wichtig und sollten daher sicher gespeichert und aufbewahrt werden. Am besten ist es, wenn man sich diese Digital auf einem Speichermedium ohne Internetzugang speichert und außerdem auf ein Blatt Papier korrekt notiert und gut aufbewahrt. Diese "Keys" enthalten einen sogenannten "mnemonic", dies ist eine Kombination aus zufälligen 24 Wörtern die in einer bestimmten Reihenfolge generiert wurden. Diese 24 Wörter sollten nicht weiter gegeben werden, da diese den Zugang zu eurer Wallet gewähren. Es ist quasi der Zugang zu eurem Account. Wir geben also folgendes ein:

chia keys generate

```
(venv) root@ubuntu:~/chia-blockchain# chia keys generate
Generating private key.
Added private key with public key fingerprint 2818420004 and mnemonic
raccoon profit meat marine rural purpose short roast fatigue capable execute adult soul fossil estate mosquito snow fortune hurdle home tuition stamp mesh dismiss
Setting the xch destination address for coinbase fees reward to txch105x186z3mjwek2jfa99344jdxepaf8ultgndwjamnyzqy7xmmpuqjql09q
Setting the xch destination address for coinbase reward to txch105x186z3mjwek2jfa99344jdxepaf8ultgndwjamnyzqy7xmmpuqjql09q
(venv) root@ubuntu:~/chia-blockchain#
```

4. Nun sind wir auch in der Lage unsere anderen "Keys" anzeigen zu lassen. Dies tun wir mit: *chia keys show*

```
Fingerprint: 2818420004
Master public key (m): b2acb9898471cdb90320db569268210310c0d9e980ca81f15cdc05b622c0ac2005f941c7456f1251422daaadad52ad0
Master private key (m): 47d28dcef74de695b4856cb22740046785a19c2a7190169451c93937e7887eff
Farmer public key (m/12381/8444/0/0): 9457eb91ad3540543bd09aeb1b4df1979647f348bf2fcae8def262a836cf5b6c25d956b9a2663d2184c7d0b8cb307b
Pool public key (m/12381/8444/1/0): 8e64bc07af438b0ae22d8406ad1e14084499e36b585c5fc62d2a06163253e364cdc250f6bae2f8d2474464b24dea2368
First wallet key (m/12381/8444/2/0): 97f2872a8e4af55935b402df6a6298cabe9778ad1f30cd70973382c08c0670bacb326029c4f17a1a0fa310e90e045e1
First wallet address: txch105x186z3mjwek2jfa99344jdxepaf8ultgndwjamnyzqy7xmmpuqjql09q
Mnemonic seed (24 secret words):
raccoon profit meat marine rural purpose short roast fatigue capable execute adult soul fossil estate mosquito snow fortune hurdle home tuition stamp mesh dismiss
(venv) root@ubuntu:~/chia-blockchain#
```

Hier ist nun eine Auflistung aller "Keys" zu sehen. Die Wichtigsten davon sind:

mnemonic seed = Zugang zum Account

fingerprint = Eine einmalige Zahlenkombination aus allen Keys

farmer public key = Ein einmaliger Key der auf deinen Farmer verweist. (Wird benötigt für Plots)

pool public key = Ein einmaliger Key der auf deinen "Pool" verweist. Damit können sich Harvester konnekten. (Wird benötigt für Plots)

first wallet address = Die Haupt Wallet-Adresse, mit der alle anderen Wallet-Adressen von euch Verknüpft werden.

5. Mit dem Befehl: ***chia start node***

wird die Full-Node gestartet und diese beginnt auch direkt sich mit dem Netzwerk zu synchronisieren. Dazu Verbindet sich die Node im Idealfall mit mehreren Peers, führt einen Abgleich durch und lädt sich so Stück für Stück Teile der Gesamten Blockchain herunter. Das Ziel dieser Full-Node ist es also irgendwann auf dem aktuellsten Stand zu sein um so selbst Teil des Netzwerks zu sein und "Anderen" die Blockchain zu Verfügung zu stellen. Da sich das Projekt Chia allerdings noch in der sogenannten Beta Phase befindet, kommt es hier noch Teilweise zu Abbrüchen und sonstigen Fehlern. Diese werden aber Regelmäßig von den Entwicklern behoben und Updates werden durchgeführt. Dieser Prozess kann also etwas länger dauern, vor allem wenn viele das Netzwerk nutzen, denn desto länger wird die Blockchain. Dieser Prozess nimmt also nur beim ersten mal viel Zeit in Anspruch, hat man diese Hürde allerdings geschafft hält die Node sich immer aktuell solange eine Internetverbindung besteht und die Node aktiviert ist.

6. Damit wir einen Überblick erhalten wie weit unsere Full-Node synchronisiert ist, geben wir folgendes ein: ***chia show -s***

Leider befinden wir uns gerade in der Version 1.0b23 und es gibt einige Probleme mit der Full-Node und der Synchronisation. Aber Grundsätzlich erhält man so einen sehr guten Überblick. Version 1.0b22 lief sehr gut mit dem Raspberry Pi. In der nächsten Version wird vermutlich auch alles wieder Ordnungsgemäß funktionieren. Davon sollte man sich nicht abschrecken lassen, da wie gesagt sich dieses Projekt noch in der Beta Phase befindet.

Nichts desto trotz, hat man bis hier alle Schritte unternommen um mit dem Plotten anfangen zu können.

Chia Kommandos Überblick

`chia show -h` = Zeigt ein paar nützlichen Hilfen an

`chia keys show` = Führt alle "Keys" auf

`chia start/stop node` = Startet bzw. stoppt die Full-Node

`chia start/stop wallet` = Startet bzw. stoppt die Wallet

`chia start/stop harvester` = Startet bzw. stoppt den Harvester

`chia start/stop farmer` = Startet bzw. stoppt den Farmer/Node/Harvester/Wallet

`chia start -r node/wallet/harvester/farmer` = Startet einen Dienst neu

`chia wallet -wp 9256` = Zeigt einem seine Wallet

`chia wallet show (new)` = Zeigt die Wallet

`chia plots add -d "path to plot"` = Speichert einen zusätzlichen Plotpfad

`cat ~/.chia/"Version"/log/debug.log` = Zeigt einem die Log Datei

`./chia-blockchain/activate` = Aktiviert die Python Umgebung

`deactivate` = Deaktiviert die Python Umgebung

Theoretisch ist es auch möglich Plots mit dem Raspberry Pi zu erstellen, aber da würde ich von abraten, da dies ein sehr sehr langwieriger Prozess ist. Ein Plot der größe k32 was die Mindestgröße ist die verlangt wird, das sind ca. 101GB FinalPlot würde mehrere Tage dauern. Deshalb führe ich den Befehl hier nicht an, dieser kommt aber im nächsten Abschnitt für Windows.

Plots erstellen mit Windows

Um Plots über unseren Windows Pc für unseren Raspberry Pi herstellen zu können, benötigen wir auch hier alle Chia Dateien. Allerdings benötigen wir hier keine Blockchain, Wallet oder sonst etwas. In dem Prozess des Plottens benötigen wir auch keinen Internetzugang diesen benötigen wir lediglich einmalig beim Downloaden des Programms.

1. Dazu geben wir in einem Browser unserer Wahl folgende Adresse ein:

<https://download.chia.net/beta-1.23-win64/ChiaSetup-0.1.23.exe>

sollte die Version nicht mehr Verfügbar sein, hilft folgende Seite und wir müssen dort auf Link für Windows klicken:

github.com/Chia-Network/chia-blockchain/wiki/INSTALL

Windows

Install the Windows installer - [Chia Blockchain Windows](#)

2. Nach dem Download installieren wir das Programm durch ein doppelklick. Sobald die GUI erscheint beenden wir diese Allerdings, da wir diese nicht benötigen und auch nicht wollen. Sobald das Programm versucht die Full-Node zu starten und wir uns mit unserem Pi im selben Netzwerk befinden gibt es einen Konflikt mit UPNP. Deshalb Deaktivieren wir dieses auch als erstes, damit kein Konflikt entstehen kann.
3. Dazu rufen wir folgenden Pfad auf um die Datei config.yaml zu editieren:

C:\Users\Benutzername\.chia\Version\config\config.yaml

Und wählen nun einen Editor aus, es reicht der einfach Windows Editor.

4. Nun suchen wir folgenden Eintrag und ändern diesen von **true** zu **false**

```
full_node:
  database_path: db/blockchain_v23.db
  enable_upnp: false
  farmer_peer:
    host: localhost
    port: 8443
```

5. Jetzt kann dieser Konflikt nicht mehr entstehen und wir können unseren ersten Plot anfangen zu erstellen. Dazu bewegen wir uns ersteinmal in unserem Ordner mit:

```
cd C:\Users\Benutzername\AppData\Local\chia-blockchain\Version\resources\app.asar.unpacked\daemon\
```

```
PS C:\Users\derde> cd C:\Users\derde\AppData\Local\chia-blockchain\app-0.1.23\resources\app.asar.unpacked\daemon\
PS C:\Users\derde\AppData\Local\chia-blockchain\app-0.1.23\resources\app.asar.unpacked\daemon>
```

6. Nun können wir nachfolgenden Befehl nach unseren System/Vorlieben anpassen und mit dem Plotten loslegen. Bei mir würde das ganze so aussehen:

```
./chia plots create -k 32 -n 4 -b 4650 -f  
b3e8aa2a44a045b80f4bf9dd23f21f6eb8a49bbaa289203bb9a31faadc91399ae5ba56  
8fc4e70b2c97c66f28a63c5f78 -p  
b4825bdedde2bed3199ea58a985f1329edb079b0bb853a16e500a7fb4a1ff311b8d70b  
8b0fb667a62478985b6caba008 -t C:\Users\derde\OneDrive\Desktop\tempo\ -d F:\  
plots\ -r 4 -u 128 -e
```

./chia plots create = ist der Befehl zum starten des Plottens

-k32 = ist die gröÙe des Plottes

-n + Zahl = Gibt an wie viele Plots nacheinander erstellt werden sollen. **-n 4** bedeutet also es werden 4 Plots infolge erstellt.

- b + Zahl = Steht für den Arbeitsspeicher der verwendet werden darf

-f = Hier müsst ihr euren Farmer Public Key eingeben

-p = Hier müsst ihr euren Pool Public Key eingeben

-t = Wo die Temprären Dateien abgelegt werden sollen

-d = Wo der Finale Plot gespeichert werden soll

-r + Zahl = Wie viele Threads beim Plottvorgang zur Verfügung gestellt werden

-u + Zahl = Anzahl an Buckets

-e = Reduziert den Ram verbrauch benötigt aber auch 12% mehr Speicherplatz

Plots einbinden und Farmer starten

1. Ich gehe davon aus, dass der Standard Weg gewählt wurde. Also zum Plotten eine NVME genutzt wurde und der fertige Plott sich auch einer externen HDD befindet. Natürlich gibt es noch viele andere Varianten, aber hier führe ich dieses Beispiel an, da dieser der komplizierteste ist.

Zuerst schließen wir die externe HDD an dem Raspberry Pi an.

2. Nun müssen wir die Festplatte ins System mounten, das bedeutet, dass das wir der Festplatte einen Pfad zuweisen können um so auf den Inhalt zugreifen zu können. Dazu erstellen wir uns zunächst ein Verzeichnis Namens Disk1 im "home" Verzeichnis:

mkdir /home/Disk1

3. Jetzt überprüfen wir ob unsere Festplatte erkannt wurde und welchen Bezeichnung diese bekommen hat:

fdisk -l

```
Disk /dev/sda: 5.47 TiB, 6001175125504 bytes, 11721045167 sectors
Disk model: Expansion Desk
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disklabel type: gpt
Disk identifier: C0E08E40-2E7C-41B4-89B2-41C292EFD781

Device      Start      End      Sectors  Size Type
/dev/sda1    34        262177   262144   128M Microsoft reserved
/dev/sda2  264192 11721043967 11720779776 5.5T Microsoft basic data

Partition 1 does not start on physical sector boundary.
root@ubuntu:~#
```

4. In meinem Fall ist es sda mit der Partition 2 also sda2. Mit diesem Wissen, können wir die Platte nun mounten:

mount /dev/sda2 /home/Disk1

```
root@ubuntu:~# mount /dev/sda2 /home/Disk1
root@ubuntu:~# ls /home
Disk1  ubuntu
root@ubuntu:~#
```

5. Wurde die Festplatte erfolgreich eingebunden, erscheint das Verzeichnis Disk1 nun mit einem grünen Hintergrund. Und in diesem Verzeichnis sehen wir auch direkt unsere erstellten Plots:

ls /home/Disk1/plots

```
root@ubuntu:~# ls /home/Disk1/plots/
plot-k32-2021-01-13-11-09-559bb7339c17aa40db0210c13171d244a6e1c939ee54b897be882e9909e8976b.plot
plot-k32-2021-01-13-20-40-b2ddfec8e8ee8bb9177f6c4f1d57750db1399af9ad896f44a1e1507d08782b.plot
plot-k32-2021-01-16-23-37-4441c24087ba9e98b0622425f98bd07424995a1066a3c3611c063ae073d116.plot
plot-k32-2021-01-16-23-38-c6d93b0b795ac327b73730c771a93f6d5291f6ecf11b780c566a8a5bdaa0a0.plot
plot-k32-2021-01-17-06-29-4ae481296798c175dc1494944e8dc310fcccbe040723479d5a68065bfccab09.plot
plot-k32-2021-01-17-06-41-ae92525db5d15b25b396db903eecca89ee7cc1927d1a199cb5bec1a4c666be.plot
plot-k32-2021-01-17-14-05-2ec760a6e6b97e55e266a29fcdad0762142d144e035d65b81da4c06209ebf738.plot
plot-k32-2021-01-17-14-05-f48a94f689868d0c70369224e0d61a3b663206c8d553536dae2ebe7611d834d8.plot
plot-k32-2021-01-17-21-32-940eefce0078a50ddad10831ab2a4213587c2499406fa0238e02012badda9.plot
plot-k32-2021-01-17-21-41-e23b8b08d8704bad0078640bcfcdecad7f9b535ee07fe0828598879ee44841.plot
plot-k32-2021-01-18-16-43-4b3d95181715c2e4da53ab07807f203a023144a9d1d2fd1c1a743a6f6e41c.plot
plot-k32-2021-01-18-16-44-885e99de0f8ea9b07d88b404b071e326b0de7de6d3d2b87f6d9df28989a2b.plot
plot-k32-2021-01-18-23-46-30d387de3f4a8a397d52e955b78480d16e2746475f594e43de6fa1f369e4d.plot
plot-k32-2021-01-18-23-56-e1f7e99dc32b7d35b089320c724079dd435236b36eb38ea8893e2a62e79aa.plot
plot-k32-2021-01-19-06-55-43e441e44a3ae172dd7f3d0b33eac481f716d1c275f91e13eadd83beed3e8.plot
plot-k32-2021-01-19-07-04-4c57e1a045cadf0bcbce72aed811b782f016763b5bad8cbdd121d0ef1474a.plot
plot-k32-2021-01-19-16-57-25173fb8339cda9e36912dc34dec7d1767d67639933bfc7e3f3ceae673b7e1c.plot
plot-k32-2021-01-19-16-57-35347b66c8f0a4526f36067936c255d767d8e61af85feb3e8ce7b12bf773eb.plot
plot-k32-2021-01-20-00-14-9d58ab67a03c826bdf56155f88ef1d9dd48e9ed88729162f7b194ae2547af09.plot
plot-k32-2021-01-20-00-20-aa445706b70841c78404110174963c13acab1cd3d2a758405cdad8a3549f2c8.plot
plot-k32-2021-01-20-06-57-704affcd5007ab9a59f366b077c2e0921ce590061f4470a89a0e07718f413.plot
plot-k32-2021-01-20-07-11-fce31ba31f2e014fcaec908545ba7f7d2c6f6f930f9d6e63c61689a1b5e802fb.plot
plot-k32-2021-01-20-17-04-50243cd18de041ae8170ea5de6af7d7af0951748753b96ae9559f10074eac1f.plot
plot-k32-2021-01-20-17-05-285c2fa32c5502ee3943c93a63d6f8602b3f166136110d471d37ae4dc588eef.plot
plot-k32-2021-01-20-23-55-2c3dc2919b5e1076628302734624c523c3f4f3515a50dbd53349d941759b3ee.plot
plot-k32-2021-01-20-23-58-4b57a06e3668fa705998336e781697046d1eac2f0b4c049f354b14415f639.plot
plot-k32-2021-01-21-07-00-50d8ee01e01b3a3090950416118110f1e203a41c0f7a80847145ef9e295310.plot
plot-k32-2021-01-21-07-05-39469ec0f95e1d104c4c16627d20887426110c7f32b3229f32ad1b0c420a2f.plot
plot-k32-2021-01-21-14-15-b4b7d8d8ad166bcbf1fa7f6db5a8f919f6170a28363f2567738945686a70d8a.plot
plot-k32-2021-01-21-22-14-8249728006317e2533ab8ec7c57f6d1301f0a26f109ef30bb86701be29ce164.plot
plot-k32-2021-01-21-22-15-cb4f360825996b6aa1f0c01e2928c682cd990f6b0034852b1276247300811e.plot
plot-k32-2021-01-22-04-26-c053c07f999140a1e12d9f260394c636a4177ef5c7e408f4ada70b30b.plot
plot-k32-2021-01-22-04-25-314007f4ca1dcdeed6cd3291f273f814138b5ac90206a3955e0a796b1f7fab.plot
plot-k32-2021-01-22-10-37-bcebbf777a988ebc3cea9f91d0c7ce9952510e2db3d9fa30469131325e23a81.plot
plot-k32-2021-01-22-10-37-7688f55dd10ecd9f9630eaf9f59c3e9f365e888d844f1ba3cc398f91bea45d7.plot
plot-k32-2021-01-22-16-50-df0d45827de333dc8385e80049b6597c42bcfe67662efe40211019a1ba8dd2a.plot
plot-k32-2021-01-22-16-51-110dd79e2041bea10a0514dc6e0070782147a018c9c7aae594dfe4da492.plot
plot-k32-2021-01-22-25-54-2a4652443f2b0d317a082f1708ecd0b31a6eae2563305d7c08d19d1537b36a.plot
plot-k32-2021-01-22-25-54-51d31385513096fda0f69032b7aa2382945311a53cde06b0422123a57a698e.plot
plot-k32-2021-01-23-06-13-104f402fff6b0941831ac33a401e32386617ed0228fab146fbd0b15976e72.plot
plot-k32-2021-01-23-06-13-e468b6ef820f7de17576373047a61e60f6e6a4dbf8475960805b024fac5734c6.plot
plot-k32-2021-01-23-12-29-b387c0284cb7115310ca61d4037f66e7f02ff665f74be0145be51fa69f291fb.plot
plot-k32-2021-01-23-12-31-bddcf8e22525e6fc73a2f0b5e97c1372a2399b0609ed19c304a4e91e102c.plot
plot-k32-2021-01-23-18-52-6d9504e8ee38ef8f83b9accd4505158b3a8c0e64e993b4f691acdfc040.plot
plot-k32-2021-01-23-18-53-6d92b033d3da3ec31eb476f7fee2be350098cab23b88862ef7a0f0ee3ddff74.plot
plot-k32-2021-01-24-12-55-322eb8f592c2c1dff0e4454fcb053c3af36f122f7decabae68677f7ec57d01cd.plot
plot-k32-2021-01-24-12-56-1043b7f70dd42ee067fa9cde5029eef210e809abae9a7d126a5cb131a9b2ccf4.plot
plot-k32-2021-01-24-19-24-a6e7a0f4012c0068f93a5a633840149611c949f40513rcf44e05026482dca.plot
plot-k32-2021-01-24-19-28-676d2ae0fda9f0c2124e0eb005a74e07f0b042682e040052a84c51051e1.plot
plot-k32-2021-01-25-01-44-b96b840dd710c61e37c591c5817b5bd0fcd703798720c2335bccc851a043cf7.plot
plot-k32-2021-01-25-01-50-13df280c9b0f0ca01e92c793b54268f64942137243a6691cf901a11088ad.plot
plot-k32-2021-01-25-07-46-dbe362491476007e77e7881fb11cdd1adce6327a89042f885bca7becb6f612.plot
plot-k32-2021-01-25-07-55-ae9178c04cf3003a449c040f98837d42435f4557e1fca578d5709a0aeeeb6.plot
plot-k32-2021-01-25-15-35-3c59fb01d72d0a0eae307c0b0d84f0654f00651f9622e0132aa1d7ef831d68.plot
plot-k32-2021-01-25-22-15-1b55907d5d40847fcd7737591b672e17acc5921840e21fba3898551e7e80053.plot
```

6. Nun müssen wir Chia noch mitteilen wo sich unsere neuen Plots befinden, dies tun wir indem wir den vorher gelernt Befehl dafür verwenden:

chia plots add -d /home/Disk1/plots/

```
root@ubuntu:~# cd chia-blockchain/
root@ubuntu:~/chia-blockchain# ./activate
(venv) root@ubuntu:~/chia-blockchain# chia plots add -d /home/Disk1/plots/
(venv) root@ubuntu:~/chia-blockchain#
```

Nun haben wir Erfolgreich unsere erstellten Plots eingebunden und mit Chia verbunden.

Geben wir nun chia start farmer ein und ist unsere Full-Node synchronisiert, so startet auch der farming Prozess und wir werden mit txch für erfolgreiche Challenges belohnt.

Natürlich gibt es hier noch viel mehr Möglichkeiten, doch dies alles würde den Rahmen für diese Dokumentation völlig sprengen.

Schritt 4: SMTP

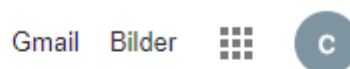
E-Mail Account erstellen

1. Um SMTP nutzen zu können, benötigen wir entweder einen E-Mail Account bei einem Provider der SMTP unterstützt oder aber wir hosten unseren eigenen E-Mail Server, welches aber nicht wirklich für unsere Anwendung zu empfehlen ist da so ein paar kleine Kosten auf uns zu kommen, wir uns viel mit dem Thema Sicherheit beschäftigen müssten und unsere E-Mails dann von den meisten Anbietern automatisch im Spamordner gelegt werden. Aus diesen Gründen ist es besser einfach einen Account bei einem Anbieter zu erstellen.

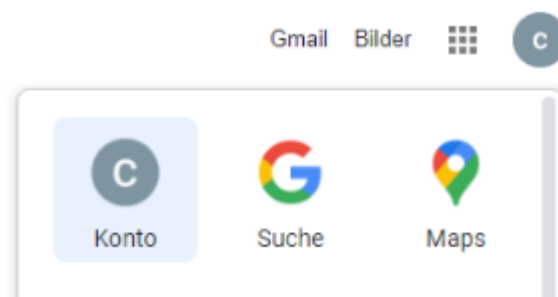
In unserem Fall habe ich extra einen Account bei Google erstellt.

2. In dem Google Account müssen wir in der Verwaltung "Apps mit niedriger Sicherheit zulassen" dies tun wir wie folgt:

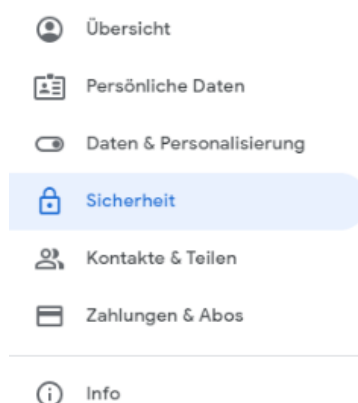
1.





2.



3.



4. **Zugriff durch weniger sichere Apps**
Ihr Konto ist angreifbar, weil Sie Apps und Geräten, die weniger sichere Anmeldetechnologien verwenden, Zugriff darauf gewähren. Damit die Sicherheit Ihres Kontos gewährleistet ist, DEAKTIVIERT Google diese Einstellung automatisch, wenn sie nicht verwendet wird.

An
[Zugriff deaktivieren \(empfohlen\)](#)
5. **Weniger sichere Apps zulassen: AN** 

Das war es schon, wir haben für unsern Google Account alles eingerichtet.

Python Skript mit smtpip und ssl

Wir haben nun die Möglichkeit unser Python Skript direkt auf unserem Raspberry Pi mit einem Editor z.B. nano zu erstellen oder aber mit einer Entwicklungsumgebung auf dem Windows System wo wir anschließend das fertige Skript per SCP an den Pi senden. In diesem Beispiel haben wir uns für letzteres entschieden.

1. Wir öffnen einen Editor oder Entwicklungsumgebung, in diesem Fall benutzen wir Visual Studio Code. Und fügen wie auf dem Bild zu sehen den Inhalt ein. Dort ist auch eine Erklärung zu dem ganzen hinterlegt.



status_mail_version_2.py X

C:\> Users > derde > OneDrive > Desktop > chia_send_mail > status_mail_version_2.py

```
1 import smtplib, ssl # Importiert die Bibliothek smtplib und die Verschlüsselung ssl
2 import os # Wählt das Betriebssystem aus
3
4 # Hier wird ein SMTP Server erstellt, genauer gesagt SMTPLib. Welcher in der Lage sein wird über den erstellten Google Account
5 # eine E-Mail an den eingetragenen Empfängeradresse zu senden.
6 # Sinn und zweck des ganzen Programms ist es, das wir wir eine Verbindung zwischen Chia-Blockchain auf dem Raspberry Pi und dem SMTP
7 # herstellen. Wir leiten mit Python, eine Chia abfrage weiter an eine Log Datei. Dann findet ein Abgleich statt. Hat es eine Veränderung
8 # mit der vorherigen Datei gegeben oder existierte vorher noch keine Datei, so wird entweder eine neue Datei mit der Abfrage erstellt
9 # oder aber die neue Datei mit der alten ersetzt. Ist dem so, so wird der Inhalt dieser neuen log Datei in eine E-Mail gepackt und an dem
10 # Empfänger gesendet.
11 # Findet keine Veränderung statt, so wird auch nichts überschrieben und auch keine E-mail gesendet.
12 # Ich habe den Code so geschrieben, dass auch ein kleiner positiver Nebeneffekt eintritt. Sollte es z.B. bei der Chia Abfrage zu einem Fehler
13 # bekommen so bekommen wir auch darüber eine Nachricht per E-Mail gesandt und auch Meldung darüber, ob wieder alles funktioniert.
14 # Das liegt daran, dass bei einer Veränderung, das immer die E-Mail mit dem neuen Inhalt versendet wird, nicht der alte.
15 # In Verbindung mit crontab auf Linux gibt so etliche Möglichkeiten die sich dadurch ergeben.
16
17
18
19 if os.path.isdir('/home/sendmail'):
20     if os.path.isfile('/home/sendmail/status_1.txt') == False: # Ist die Datei status_1.txt schon vorhanden wird die Datei geöffnet
21         os.system('touch /home/sendmail/status_1.txt') # Ist die Datei status_1.txt noch nicht vorhanden, so wird diese vor dem öffnen erstellt.
22 else:
23     os.system('mkdir /home/sendmail')
24     os.system('touch /home/sendmail/status_1.txt')
25
26 show_list_1 = []
27 with open('/home/sendmail/status_1.txt', 'r') as show_file_1:
28     if os.stat('/home/sendmail/status_1.txt').st_size != 0:
29         for show_line in show_file_1:
30             show_list_1 += [show_line.strip()]
31
32 show_list_2 = []
33
34 os.system('chia wallet show > /home/sendmail/status_2.txt') # Die Ausgabe der Chia Abfrage wird in die Datei status_2.txt geleitet
35
36 with open('/home/sendmail/status_2.txt', 'r') as show_file_2:
37     for show_line in show_file_2:
38         show_list_2 += [show_line.strip()]
39
40 if len(show_list_2) > 0:
41     del show_list_2[0]
42 if len(show_list_1) > 0:
43     del show_list_1[0]
44
45 switch = True
46
47 show_file_1 = open('/home/sendmail/status_1.txt', 'r')
48 show_file_2 = open('/home/sendmail/status_2.txt', 'r')
49
50 if len(show_list_1) > 0:
51     a = 0
52     for i in show_list_2:
53         if i != show_list_1[a]:
54             status2 = show_file_2.read() # In der Variable status2 wird der Inhalt der Datei status_2.txt gespeichert
55             port = 587 # Portnummer für den SMTP
56             smtp_server = "smtp.gmail.com" # Googles smtp Server
57             sender_email = "DEINE-SENDER-EMAIL" # Sender Adresse
58             receiver_email = "DEINE-EMPFÄNGER-EMAIL" # Empfänger Adresse
59             pw = open('/root/google_pw/pw.txt', 'r') # Öffnet die Datei, die das Passwort zum Google Account enthält. Diese ist im root Verzeichnis hinterlegt
60             password = pw.read() # Speichert den Inhalt der Passwort Datei
```

```

61     SUBJECT = "Chia-Status" # Betreff
62     TEXT = status2 # Inhalt von Status2
63     message = 'Subject: {}\n\n{}'.format(SUBJECT, TEXT) # Hier wird in der Variable message die Reihenfolge und das Format der Mail gespeichert.
64     context = ssl.create_default_context() # Verschlüsselung
65     with smtplib.SMTP(smtp_server, port) as server: # Hier wird der SMTP Server erstellt und nutzt die Angegebene Portnummer
66         # anschließend startet die Verschlüsselung, es wird sich in den Google Account eingeloggt
67         # dann wird eine Mail erstellt und an den Empfänger gesendet.
68         server.ehlo()
69         server.starttls(context=context)
70         server.ehlo()
71         server.login(sender_email, password)
72         server.sendmail(sender_email, receiver_email, message)
73     show_file_1.close() # Die Datei status_1.txt wird wieder geschlossen
74     show_file_2.close() # Die Datei status_2.txt wird wieder geschlossen
75     os.system('mv /home/sendmail/status_2.txt /home/sendmail/status_1.txt') # Die Datei status_2.txt wird Unbenannt und ersetzt dadurch status_1.txt
76     switch = False
77     break
78 else:
79     a += 1
80
81 else:
82     status2 = show_file_2.read() # In der Variable status2 wird der Inhalt der Datei status_2.txt gespeichert
83     port = 587 # Portnummer für den SMTP
84     smtp_server = "smtp.gmail.com" # Googles smtp Server
85     sender_email = "DEINE-SENDER-EMAIL" # Sender Adresse
86     receiver_email = "DEINE-EMPFÄNGER-EMAIL" # Empfänger Adresse
87     pw = open('/root/google_pw/pw.txt', 'r') # Öffnet die Datei, die das Passwort zum Google Account enthält. Diese ist im root Verzeichnis hinterlegt
88     password = pw.read() # Speichert den Inhalt der Passwort Datei
89     SUBJECT = "Chia-Status" # Betreff
90     TEXT = status2 # Inhalt von Status2
91     message = 'Subject: {}\n\n{}'.format(SUBJECT, TEXT) # Hier wird in der Variable message die Reihenfolge und das Format der Mail gespeichert.
92     context = ssl.create_default_context() # Verschlüsselung
93     with smtplib.SMTP(smtp_server, port) as server: # Hier wird der SMTP Server erstellt und nutzt die Angegebene Portnummer
94         # anschließend startet die Verschlüsselung, es wird sich in den Google Account eingeloggt
95         # dann wird eine Mail erstellt und an den Empfänger gesendet.
96         server.ehlo()
97         server.starttls(context=context)
98         server.ehlo()
99         server.login(sender_email, password)
100        server.sendmail(sender_email, receiver_email, message)
101    show_file_1.close() # Die Datei status_1.txt wird wieder geschlossen
102    show_file_2.close() # Die Datei status_2.txt wird wieder geschlossen
103    os.system('mv /home/sendmail/status_2.txt /home/sendmail/status_1.txt') # Die Datei status_2.txt wird Unbenannt und ersetzt dadurch status_1.txt
104    switch = False
105
106 if switch == True:
107     show_file_1.close()
108     show_file_2.close()
109     os.system('rm /home/sendmail/status_2.txt') # Die Datei status_2.txt wird gelöscht

```

https://github.com/Crowdie88/chia_send_mail

- Wir senden diese Datei nun per SCP an den Raspberry Pi. In meinem Fall sieht das ganze folgendermaßen aus, in eurem Fall müsst ihr natürlich die Parameter ändern.

scp C:\Users\derde\OneDrive\Desktop\send_email\status_mail.py root@192.168.2.134:/home

```

PS C:\Users\derde> scp C:\Users\derde\OneDrive\Desktop\send_email\status_mail.py root@192.168.2.134:/home
root@192.168.2.134's password:
status_mail.py                                100% 4339   423.7KB/s   00:00
PS C:\Users\derde>

```

Crontab

Crontab ist das letzte Bindeglied, mit dem wir alle vorherigen Anstrengungen einen Automatismus verleihen können. Hiermit stellen wir sicher, dass die Chia Full-Node immer aktiviert ist, sodass der Farmer und der Harvester immer UpToDate sind auch wenn der Raspberry Pi mal unerwartet Neustarten sollte. Nur so können wir sicherstellen, dass wir das Maximum aus unserer Farm herausholen können. Außerdem stellt Crontab sicher, dass unser Python Skript samt SMTP Server uns über unsere Einnahmen und Fehler Informiert.

Auch hier sind nur einige Beispiele angegeben, hier sind der Fantasie keine Grenzen gesetzt. Für diese Dokumentation allerdings ist das zu viel und würde auch hier den Rahmen sprengen.

1. Crontab ist auf jeder Linux Distribution Heutzutage schon vorinstalliert, deshalb können wir direkt mit der Konfiguration starten. Dazu geben wir folgendes ein:

crontab -e

Nun wird eine Entscheidung von uns erwartet. Crontab wird in unserem Fall für den User root angelegt. Da zuvor noch kein Crontab verwendet wurde, müssen wir nun eine Möglichkeit auswählen. In unserem Fall wird es Punkt 1 sein mit nano.

```
root@ubuntu:~/chia-blockchain# crontab -e
no crontab for root - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /bin/nano      <---- easiest
 2. /usr/bin/vim.basic
 3. /usr/bin/vim.tiny
 4. /bin/ed

Choose 1-4 [1]:
```

2. Nun können wir Crontab editieren wie im nächsten Bild zu sehen, die Erklärungen dazu folgen darunter:

```
root@ubuntu: ~/chia-blockchain
GNU nano 4.8
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
*/10 * * * * . /chia-blockchain/activate; /usr/bin/python3 /home/status_mail.py; deactivate >> /home/log_sendmail.txt 2>&1
@reboot mount /dev/sda2 /home/Disk1; sleep 120; . /chia-blockchain/activate; chia start farmer >> /home/log_chia.txt 2>&1
```

Die erste Zeile Bedeutet folgendes:

Alle 10 Minuten soll die Python Entwicklungsumgebung aus dem Verzeichnis /chia-blockchain geöffnet werden. Danach öffne durch Python3 aus dem Verzeichnis /usr/bin/ das Skript status_mail.py im Verzeichnis /home. Anschließend deaktiviere die Entwicklungsumgebung wieder. Nachdem dies geschehen ist, leite falls vorhanden, eine Fehlerausgabe in die Datei log_sendmail.txt im Verzeichnis /home um.

Die zweite Zeile Bedeutet:

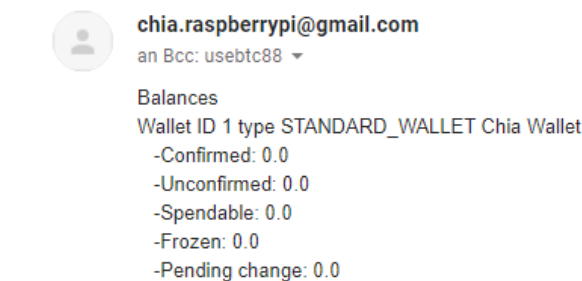
Nach einem hochfahren, binde die 2. Partition der Festplatte Namens sda in das Verzeichnis /home/Disk1 ein. Danach versetze dich 120 Sekunden in den Ruhemodus. Als nächstes öffnest du die Python Entwicklungsumgebung aus dem Verzeichnis /chia-blockchain heraus. Nun gebe Chia den Befehl "start farmer". Gibt es eine Fehlermeldungs Ausgabe, leite diese in die Datei log_chia.txt um die sich im Verzeichnis /home befindet.

Mit Crontab lassen sich also ziemlich mächtige Automatismen bilden. Man sollte allerdings immer aufpassen, dass man die Zeiten gut timed und man immer Absolute Pfade angibt.

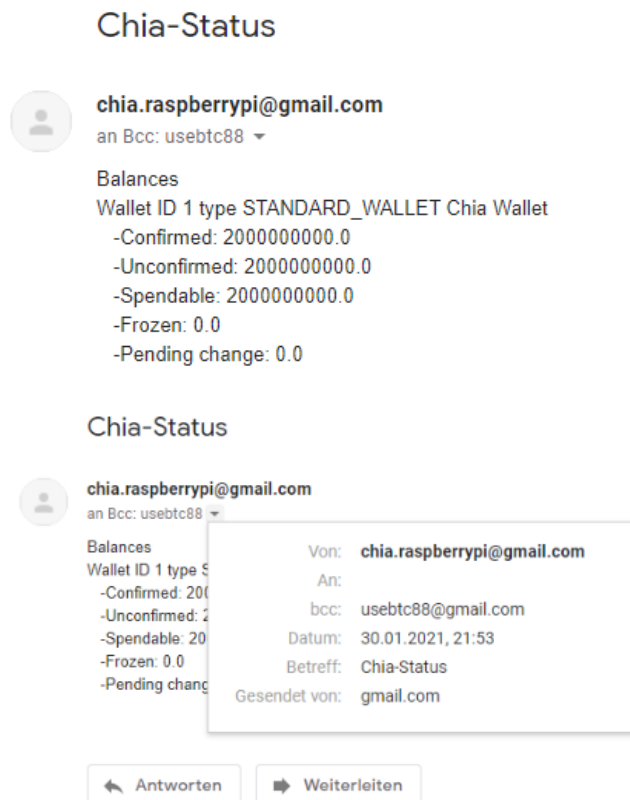
SMTP E-Mails

Hat man alles Ordnungsgemäß erstellt, fängt der E-Mail Server auch schon an Mails zu versenden. Diese sehen dann wie folgt aus:

Erste Nachricht Absender ohne Coins:



Zweite Nachricht Absender mit Coins und Betreff:



Dritte Nachricht Absender Fehlermeldung:

Chia-Status



chia.raspberrypi@gmail.com

an Bcc: usebtc88 ▾

Connection error. Check if wallet is running at 9256



Antworten



Weiterleiten

Vierte Nachricht Empfänger:

Chia-Status Posteingang x



chia.raspberrypi@gmail.com

an ▾

Balances

Wallet ID 1 type STANDARD_WALLET Chia Wallet

-Confirmed: 2000000000.0

-Unconfirmed: 2000000000.0

-Spendable: 2000000000.0

-Frozen: 0.0

-Pending change: 0.0

Es funktioniert also genau wie gewünscht. Die Kombination aus SMTP Python und der Linuxumgebung auf dem Raspberry Pi harmonieren und funktionieren ausgezeichnet.

Fazit:

Es lassen sich mit dem Raspberry Pi in Kombination mit Servern erstaunliche Dinge vollbringen. Gerade in dieser Dokumentation lässt sich meiner Meinung nach gut nachvollziehen was möglich ist. Man kann eine Mining(Farming) Farm betreiben, die sozusagen völlig autonom arbeiten kann, wenn man diese denn vorab vernünftig konfiguriert.

Leider lässt sich nicht alles und vorallem nicht bis ins kleinste Detail festhalten was möglich ist, da man dafür ein weitaus größeren Zeitaufwand mitbringen muss.

Quellen:

www.chia.net

www.github.com

www.google.de

[*keybase\(chia_network.public\)*](#)

wiki.ubuntuusers.de

www.github.com/Crowdie88