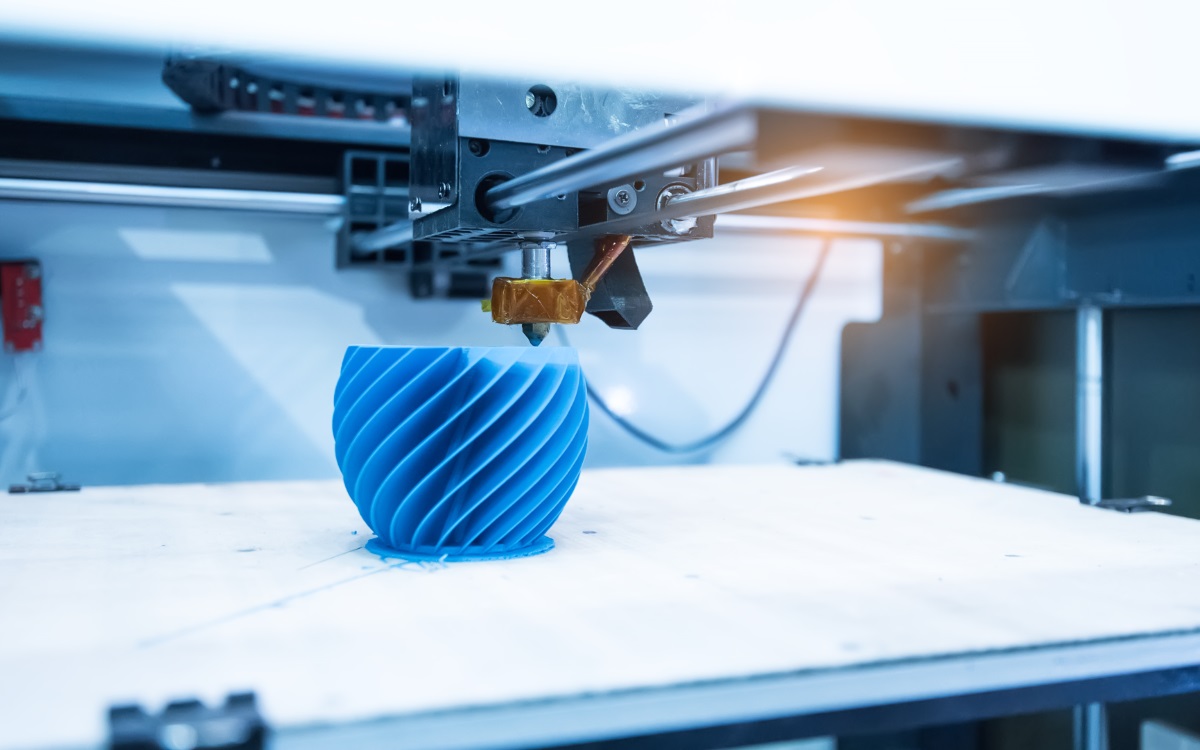
Cloud Computing – Teil 2

- Prof. Dr. Kasche –

**Dokumentation**

Thema: Skript zum outsourcen eines Slicing-Prozess (3D)



Kevin Lempert

Jonas G. Renz

- PIA18 -

Inhaltsverzeichnis

[1 Kurzinfo 3](#_Toc55546022)

[2 Bedienungsanleitung 4](#_Toc55546023)

[2.1 Anleitung 4](#_Toc55546024)

[2.2 Parameter 5](#_Toc55546025)

[3 Projektdokumentation 7](#_Toc55546026)

[3.1 User Stories / Story-Board 7](#_Toc55546027)

[3.2 KanBan-Board 9](#_Toc55546028)

[3.3 MVP 11](#_Toc55546029)

[4 Systemdokumentation 12](#_Toc55546030)

[4.1 PAP 12](#_Toc55546031)

[5 Schnittstellendokumentation 14](#_Toc55546032)

[5.1 AWS 14](#_Toc55546033)

[5.2 Python SDK 14](#_Toc55546034)

[6 Testdokumentation 15](#_Toc55546035)

[7 Restaufgaben 16](#_Toc55546036)

[8 Quellcode 17](#_Toc55546037)

1. Kurzinfo

|  |  |  |
| --- | --- | --- |
| **Autor:** |  | **Kevin Lempert Jonas G. Renz** |
| **Hochschuleinrichtung:** |  | **Duale Hochschule Gera-Eisenach (Campus Gera)** |
| **Version:** |  | **1.0** |
| **Programmiersprache:** |  | **Python 3.x** |
| **Datum:** |  | **06.11.2020** |
| **Systemaufruf:** |  | **cloudslice.py** |
| **Beschreibung:** |  | Dieses Skript dient zum Outsourcen für Slicing-Prozesse in die Cloud. Mithilfe von AWS und Slic3r können so Slicing-Prozesse Ressourcen sparend und schnell den GCODE für 3D Drucker in AWS EC2 berechnen. |

1. Bedienungsanleitung
   1. Anleitung

Es muss für das Skript eine bestehende EC2 Instanz auf Amazon AWS existieren. Diese Instanz muss über das Betriebssystem Linux verfügen. Wenn eine keine Instanz existiert muss eine EC2 Instanz auf AWS initialisiert werden.

Schritt: Installieren des Programms Slic3r auf EC2-Instanz

$ sudo apt-get install slic3r

Schritt: Konfiguration in der CONFIG-Sektion des Skriptes vornehmen

2.1 Schritt: ID der Instanz eintragen

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

2.2 Schritt: Benutzername und den Pfad des Private-Key eintragen

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Schritt: Python-Skript ausführen

$ py cloudslice.py

Schritt: Wizard mit abgefragten Parametern befüllen

Schritt: fertige GCODE-Datei liegt im angegebenen Verzeichnis

* 1. Parameter

|  |  |  |
| --- | --- | --- |
| Parameter | Variable(n) | Erklärung |
| --bed-temperature <t> | t = ganze Zahl, z.B.: 60 | Hier soll die Druckbett-temperatur angegeben werden. Maßeinheit: [°C] |
| --cooling |  | Lüfter- und Kühlungssteuerung aktivieren |
| --filament-diameter <d> | d = ganze Zahl, 1.75 oder 3 | Durchmesser des Filaments  Maßeinheit: [mm] Default: 3 |
| --fill-density <d> | d = ganze Zahl, z.B.: 50 | Dichte der Füllung Maßeinheit: [%] Default: 40 Bereich: 0-100 |
| --layer-height <h> | h = Kommazahl, z.B.: 0.2 | Hier wird die Layer-Höhe des Modells angegeben. Maßeinheit: [mm] Default: 0.4 |
| --nozzle-diameter <d> | d = Kommazahl, > 0.4 | Durchmesser der Düse Maßeinheit: [mm] Default: 0.5 |
| --output <o> | o = Name der Datei | Dateiname, in die gcode ausgegeben werden soll Wenn ein Verzeichnis für diese Option angegeben wird, wird die Ausgabe unter diesem Verzeichnis gespeichert. |
| --retract-length <l> | l = ganze Zahl, z.B.: 36 | Länge des Rückzugs beim Anhalten der Extrusion Maßeinheit: [mm] |
| --scale <s> | s = Kommazahl, z.B.: 1.3  1 = Orginalgröße | Hier soll ein Skalierungsfaktor für das Model angegeben werden. |
| --support-material |  | Erzeugen von Hilfsmaterial für Überhänge |
| --temperature <t> | t = ganze Zahl, z.B.: 215 | Temperatur Hotend  0 zum Deaktivieren  Maßeinheit: [°C] Default: 200 |

1. Projektdokumentation
   1. User Stories / Story-Board

Ein Bild, das drinnen, Screenshot, sitzend, Monitor enthält.

Automatisch generierte Beschreibung

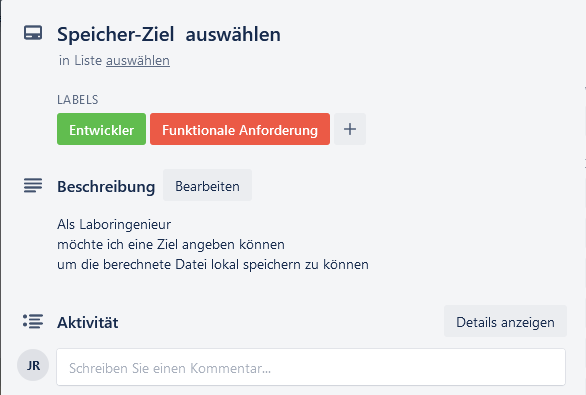
Das Story Board besteht aus einzelnen Karten, in dem der Kunde seine Anforderungen äußert.



Hier ist eine Karte aus dem Bereich „anzeigen“. Die Karte enthält eine nichtfunktionale Anforderung, die von den Entwicklern eingesetzt wurde. **Nicht-funktionale Anforderungen** hingegen sind meist unspezifisch für ein Produkt. Beispiele für nicht-funktionale Anforderungen sind User Stories enthalten immer den gleichen Text:

Als …  
möchte ich …  
um …

In der nächsten Karte ist eine funktionale Anforderung zu erkennen. **Funktionale Anforderungen** sind die Anforderungen, deren Umsetzung direkt der Zweckbestimmung des Produkts dienen. Sie sind spezifisch für dieses Produkt.



* 1. KanBan-Board

Ziel des Kanban-Boards ist es, Projektabläufe und Aufgaben zu visualisieren – in übersichtlichen Spalten mit einzelnen Einträgen, die der Reihe nach abgearbeitet werden können.

So sieht das Team auf einen Blick, was zu tun ist und in welchen Bereichen Probleme gelöst werden müssen. Dabei kann das Kanban Board auf viele verschiedene Weisen genutzt und gestaltet werden – um es optimal anzupassen und die Ergebnisse zu verbessern.

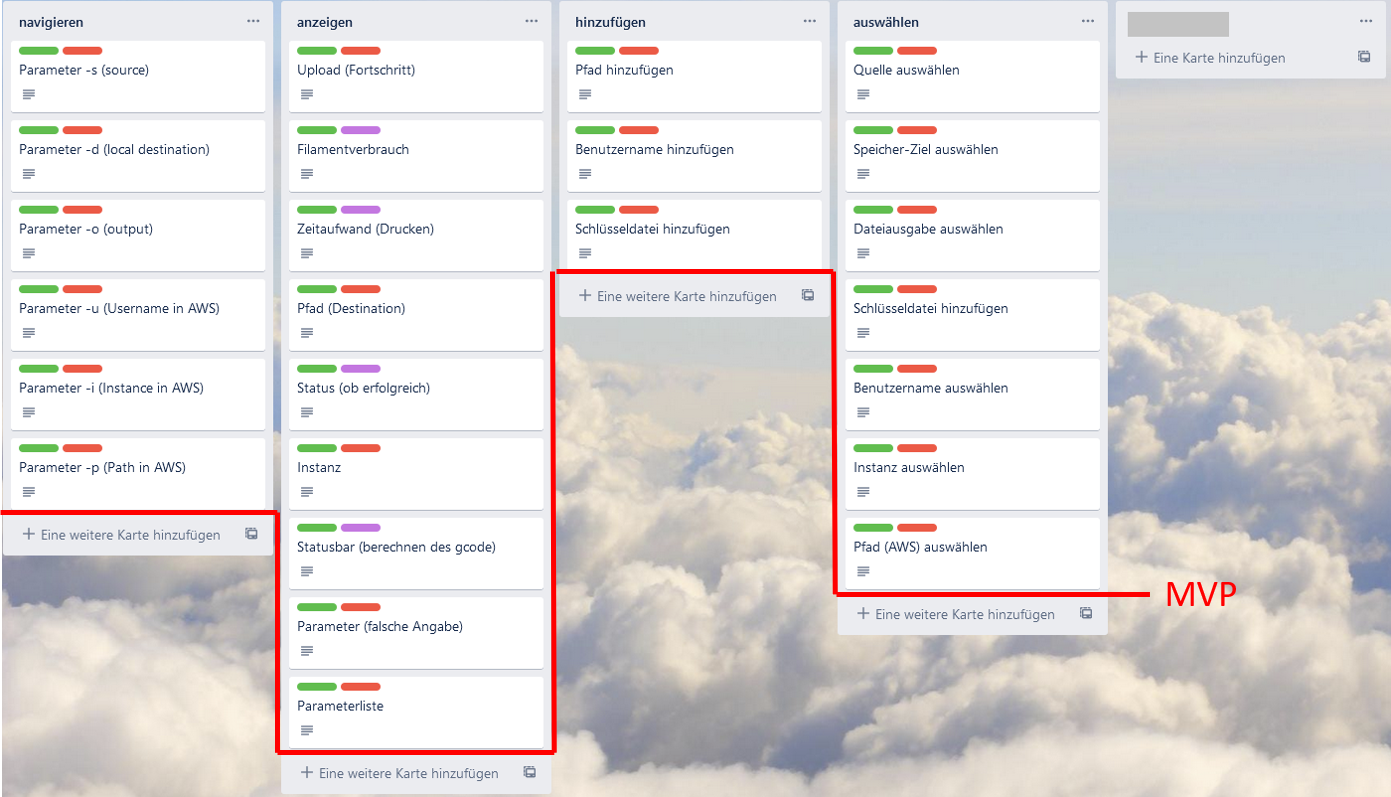
Hier ist ein kleiner Ausschnitt aus dem KanBan-Board zu sehen. Zur Übersichtlichkeit wurde aus der frühen Projektphase ein Foto übernommen.

Ein Bild, das Screenshot, drinnen, Monitor, Computer enthält.

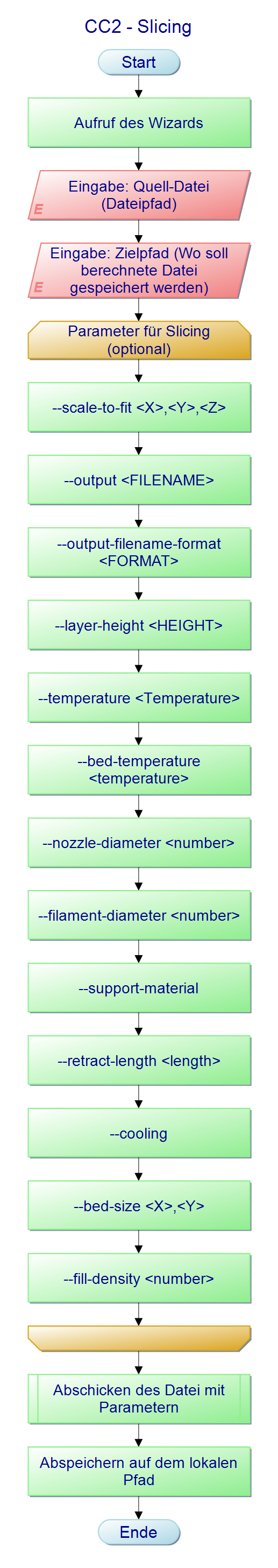
Automatisch generierte Beschreibung

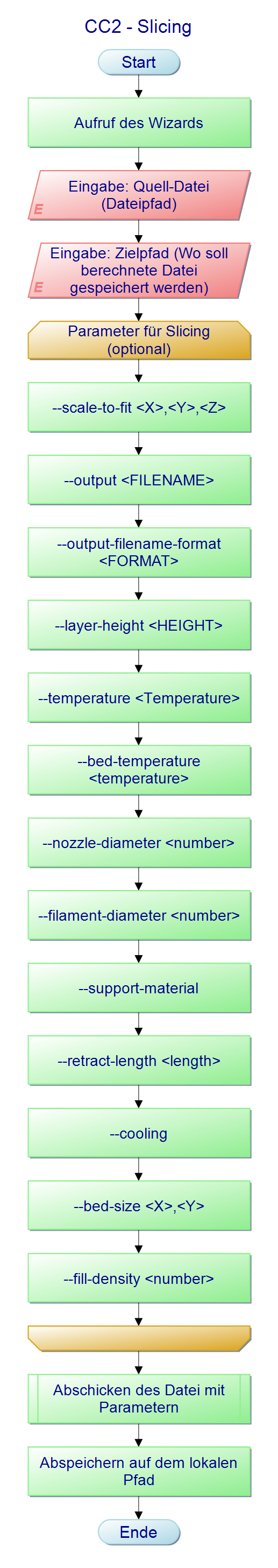
* 1. MVP

Ein Minimum Viable Product (MVP) ist die erste minimal funktionsfähige Iteration eines [Produkts](https://de.wikipedia.org/wiki/Produkt_(Wirtschaft)), das entwickelt werden muss, um mit minimalem Aufwand den Kunden-, Markt- oder Funktionsbedarf zu decken und handlungsrelevantes Feedback zu gewährleisten. Das MVP sieht ist aus dem Story-Board zu entnehmen.



1. Systemdokumentation
   1. PAP





1. Schnittstellendokumentation
   1. AWS

**Amazon Web Services** (**AWS**) ist ein US-amerikanischer Cloud-Computing-Anbieter, der 2006 als Tochterunternehmen des Online-Versandhändlers Amazon.com gegründet wurde. Zahlreiche populäre Dienste wie beispielsweise Dropbox, Netflix, Foursquare oder Reddit greifen auf die Dienste von Amazon Web Services zurück. 2017 stufte Gartner AWS als führenden internationalen Anbieter im Cloud Computing ein.

Der Web-Service Amazon Elastic Compute Cloud (Amazon EC2) stellt sichere, skalierbare Rechenkapazitäten in der Cloud bereit. Der Service ist darauf ausgelegt, Cloud Computing für Entwickler zu erleichtern. Amazon EC2 bietet die breiteste und tiefste Computerplattform mit einer Auswahl an Prozessor, Speicher, Netzwerk, Betriebssystem und Kaufmodell.

* 1. Python SDK

Amazon stellt für Python-Skripte ein SDK zur Verfügung. Das SDK nennt sich Boto3. Boto3 erleichtert die Integration von Python-Anwendung, ‑Bibliothek oder Scripts in AWS-Services wie Amazon S3, Amazon EC2, Amazon DynamoDB und andere.

Boto3 hat zwei getrennte Ebenen von APIs. Die Client-APIs (oder API auf niedriger Ebene) bieten eine 1-zu-1-Zuordnung der zugrunde liegenden HTTP API-Vorgänge. Ressourcen-APIs verbergen explizite Netzwerkaufrufe und stellen stattdessen Ressourcenobjekte und Collections bereit, um auf Attribute zuzugreifen und Aktionen durchzuführen.

1. Testdokumentation

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abb. 1: Testablauf eines Slicing-Vorgangs mit CloudSlice

1. Restaufgaben

Restaufgaben stehen in den Post-Releases des KanBan-Boards. Diese Liste ist Links des Boards zu sehen.



1. Quellcode

"""

##################################################

SCRIPT INFORMATION

##################################################

## Author: Kevin Lempert, Jonas Renz

## Copyright: Copyright 2020, CloudSlice

## Version: 1.0.0

##################################################

STARTING GUIDE

##################################################

1. init EC2 instance (Linux)

2. install slic3r

> sudo apt-get install slic3r

3. configurate the script (CONFIG section)

> instance\_id = Instance ID

> cUser = Username

> cKey = Private Key

(optional)

> u\_rPath = AWS upload location (STL)

> d\_rPath = AWS download location (GCODE)

##################################################

DEV INFORMATION

##################################################

AWS States:

0 : pending

16 : running

32 : shutting-down

48 : terminated

64 : stopping

80 : stopped

scp:

scp -i .\key\Key\_1.pem file.stl ubuntu@[instancedns]:/home/ubuntu/file.stl

ssh:

ssh -i .\key\Key\_1.pem ubuntu@[instancedns]

slice:

slic3r file.stl --output file.gcode [options]

"""

from time import sleep

from datetime import datetime

import sys

import boto3

from botocore.exceptions import ClientError

import os

import paramiko

import logging

#-------------------------------------------------------------------------------

# CONFIG

#-------------------------------------------------------------------------------

# AWS instance

instance\_id = ['i-0dc8a2dbfad73a683']

# Credentials

cUser = 'ubuntu'

cKey = './key/Key\_1.pem'

# upload/download Path AWS

u\_rPath = '/home/ubuntu/temp.stl'

d\_rPath = '/home/ubuntu/temp.gcode'

# Print parameter list for wizard

# add new parameters here

# Input text wizard : cmd slic3r

cmddict = {

"> Layer height ( 0.2 - 1 ) " : "--layer-height",

"> Temperature ( 100-250 ) " : "--temperature",

"> Bed temperature ( 0-100 ) " : "--bed-temperature",

"> Cooling ( true/false ) " : "--cooling",

"> Support ( true/false ) " : "--support-material",

"> Fill Density ( 0-100 ) " : "--fill-density",

"> Filament Diameter ( 1.75/3 ) " : "--filament-diameter",

"> Nozzle Diameter ( 0.4-2 ) " : "--nozzle-diameter",

"> Retract Length ( 0-10 ) " : "--retract-length",

"> Scale ( 1 = orginalsize ) " : "--scale"

}

#-------------------------------------------------------------------------------

# GLOBAL

#-------------------------------------------------------------------------------

# List for commands

cmdlist = list()

# init boto3

ec2 = boto3.resource('ec2')

# Logging Configuration

logging.basicConfig(

level=logging.INFO,

format="%(asctime)s [%(levelname)s] %(message)s",

handlers=[

logging.FileHandler("debug.log"),

logging.StreamHandler()

]

)

#-------------------------------------------------------------------------------

# MAIN

#-------------------------------------------------------------------------------

# main function

def main():

header()

startwizard()

stime\_ssh = 20

# First State Check

if( getinstancesstate() == 80 ):

logging.info("Start instance...")

startinstance()

elif(getinstancesstate() == 16):

logging.info("Instance already running...")

stime\_ssh = 5

else:

logging.info("Wait for instance...")

while True:

sleep(2)

if(getinstancesstate() == 16 or getinstancesstate() == 80):

if(getinstancesstate() == 80):

startinstance()

break

# Wait for instance start

while True:

sleep(3)

if(getinstancesstate() == 16):

logging.info("Wait for SSH server...")

sleep(stime\_ssh)

break

# Get instance DNS

server = getinstancesDNS()

logging.info("Server Adress: " + server)

if( getinstancesstate() == 16 ):

# SCP upload:

logging.info("Upload file...")

putSCP(server , u\_lPath, u\_rPath)

logging.info("File upload complete!")

# SSH command:

command = buildcommand()

logging.info("Start slice process...")

makeSSH(server, command)

logging.info("Slice process complete!")

# SCP download:

logging.info("Download GCODE...")

getSCP(server, d\_lPath, d\_rPath)

logging.info("GCODE download complete! File Path: " + d\_lPath)

# Stop instance

logging.info("Stop instance...")

stopinstance()

else:

logging.error("Error! Instance is stopped!")

#-------------------------------------------------------------------------------

# WIZARD

#-------------------------------------------------------------------------------

# wizard for print parameters

def startwizard():

global u\_lPath

global d\_lPath

logging.info("Starting WIZARD!")

print(">>> File parameters")

while(True):

u\_lPath = input(" Path input file (path/filename.stl): ")

if( os.path.isfile(u\_lPath) ):

break

else:

logging.error("File " + u\_lPath + " doesnt exist!")

d\_lPath = input(" Path output file (path/filename.gcode): ")

print("\n>>> Print parameters (false=default)")

for key, item in cmddict.items():

value = input(" " + key + ": ")

if(value.lower() in "true"):

cmdlist.append( item )

elif(value.lower() not in "false"):

cmdlist.append( item + " " + value )

print()

# build slic3r command

def buildcommand():

command = 'slic3r ' + u\_rPath + ' --output ' + d\_rPath

for value in cmdlist:

command = command + " " + value

return command

#-------------------------------------------------------------------------------

# FUNCTIONS

#-------------------------------------------------------------------------------

#==================================AWS==================================

# start instance

def startinstance():

try:

response = ec2.instances.filter(InstanceIds=instance\_id).start()

except ClientError as e:

logging.error(e)

# stop instance

def stopinstance():

try:

response = ec2.instances.filter(InstanceIds=instance\_id).stop()

except ClientError as e:

logging.error(e)

# get instance state

def getinstancesstate():

instances = ec2.instances.filter(InstanceIds=instance\_id)

for instance in instances:

return(instance.state.get("Code"))

# get DNS from instance

def getinstancesDNS():

instances = ec2.instances.filter(InstanceIds=instance\_id)

for instance in instances:

return(instance.public\_dns\_name)

#==================================SCP==================================

# SCP - upload

def putSCP(server, lPath, rPath):

ssh = paramiko.SSHClient()

ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

ssh.connect(server, username=cUser, key\_filename=cKey )

sftp = ssh.open\_sftp()

sftp.put(lPath, rPath)

sftp.close()

ssh.close()

# SCP - download

def getSCP(server, lPath, rPath):

ssh = paramiko.SSHClient()

ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

ssh.connect(server, username=cUser, key\_filename=cKey )

sftp = ssh.open\_sftp()

sftp.get(rPath,lPath)

sftp.close()

ssh.close()

#==================================SSH==================================

# start SSH connection

def makeSSH(server, command):

ssh = paramiko.SSHClient()

ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

ssh.connect(server, username=cUser, key\_filename=cKey )

stdin, stdout, stderr = ssh.exec\_command(command)

lines = stdout.readlines()

logging.debug(str(lines))

ssh.close()

#-------------------------------------------------------------------------------

# PRINT-INFOS

#-------------------------------------------------------------------------------

# Print CMD header

def header():

print(" \_\_\_\_\_ \_ \_ \_\_\_\_\_ \_ \_ ")

print(" / \_\_\_\_| | | |/ \_\_\_\_| (\_) ")

print("| | | | \_\_\_ \_ \_ \_\_| | (\_\_\_ | |\_ \_\_\_ \_\_\_ ")

print("| | | |/ \_ \| | | |/ \_` |\\_\_\_ \| | |/ \_\_/ \_ \ ")

print("| |\_\_\_\_| | (\_) | |\_| | (\_| |\_\_\_\_) | | | (\_| \_\_/ ")

print(" \\_\_\_\_\_|\_|\\_\_\_/ \\_\_,\_|\\_\_,\_|\_\_\_\_\_/|\_|\_|\\_\_\_\\_\_\_| ")

print(" Created by Jonas R. & Kevin L. ")

print("")

#-------------------------------------------------------------------------------

# START

#-------------------------------------------------------------------------------

# start point

if \_\_name\_\_ == '\_\_main\_\_':

main()