Aktor: Server herunterfahren (Shutdown)

Ein simples Aktorscript in Python, um den Server herunterzufahren - zum Beispiel für eine Maintenance.

Konzept und Installation

Grundsätzlich gibt es eine einzelne `main.py`. Dieses Aktorscript basiert auf dem Python Process Template. Das Script führt `sudo /sbin/shutdown -h now` aus und verhindert, dass dieser Status nach dem Neustart erhalten bleibt. Dazu arbeiten wir mit einem Lock File `shutdown.lock`, welches im Verzeichnis des Scripts (`/opt/fabinfra/adapters/shutdown/`) abgelegt oder gelöscht wird - je nach Aktion.

Berechtigungen anpassen

Wir erlauben dem Nutzer `bffh` das Ausführen des Befehls `shutdown` per `sudo`.

```
sudo echo "bffh ALL=NOPASSWD: /sbin/shutdown" > /etc/sudoers.d/bffh
```

Script files

```
mkdir -p /opt/fabinfra/adapters/shutdown/
vim /opt/fabinfra/adapters/shutdown/main.py
```

```
import argparse
import psutil
import subprocess
import os

'''
This actor scripts shuts down the server, if no shutdown.lock file is existent (pressing "USE" in the client". The lock file is needed because otherwise the server will ALWAYS shutdown as long as the state of the actor was not set back. So we trigger only if the lock file was removed. The removal of the lock file is done in the client by "GIVEBACK"
'''

file_path = os.path.join(os.path.dirname(__file__), "shutdown.lock")
def on_free(args, actor_name):
```

```
    if os.path.exists(file_path):
        os.remove(file_path)

def on_use(args, actor_name, user_id):
    try:
        with open(file_path, 'x') as file:
            file.write("DO NOT DELETE")
            cmd = "sudo /sbin/shutdown -h now"
            try:
                proc = subprocess.Popen(cmd, shell=True, stdin=subprocess.PIPE, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
            except OSError as e:
                raise OSError("{0}\nCommand failed: errno={1} {2}".format(' '.join(cmd), e.errno, e.strerror))

    except FileExistsError:
        print("The file '{}' already exists".format(file_path))

def on_tocheck(args, actor_name, user_id):
    print("To Check")

def on_blocked(args, actor_name, user_id):
    print("Blocked")

def on_disabled(args, actor_name):
    print("Disabled")

def on_reserve(args, actor_name, user_id):
    print("Reversed")

def on_raw(args, actor_name, data):
    print("Raw")



def main(args):
    new_state = args.state
    if new_state == "free":
        on_free(args, args.name)
    elif new_state == "inuse":
        on_use(args, args.name, args.userid)
    elif new_state == "tocheck":
```

```
        on_tocheck(args, args.name, args.userid)
    elif new_state == "blocked":
        on_blocked(args, args.name, args.userid)
    elif new_state == "disabled":
        on_disabled(args, args.name)
    elif new_state == "reserved":
        on_reserve(args, args.name, args.userid)
    elif new_state == "raw":
        on_raw(args, args.name, args.data)
    else:
        print("Process actor called with unknown state %s" % new_state)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("name", help="name of this actor as configured in bffh.dhall")

    subparsers = parser.add_subparsers(required=True, dest="state")

    parser_free = subparsers.add_parser("free")

    parser_inuse = subparsers.add_parser("inuse")
    parser_inuse.add_argument("userid", help="The user that is now using the machine")

    parser_tocheck = subparsers.add_parser("tocheck")
    parser_tocheck.add_argument("userid", help="The user that should go check the machine")

    parser_blocked = subparsers.add_parser("blocked")
    parser_blocked.add_argument("userid", help="The user that marked the machine as blocked")

    parser_disabled = subparsers.add_parser("disabled")

    parser_reserved = subparsers.add_parser("reserved")
    parser_reserved.add_argument("userid", help="The user that reserved the machine")

    parser_raw = subparsers.add_parser("raw")
    parser_raw.add_argument("data", help="Raw data for for this actor")

    args = parser.parse_args()
    main(args)
```

```
chown -R bbfh:bffh /opt/fabinfra/adapters/shutdown/
```

Das Script manuell testen

```
#USE
/usr/bin/python3 /opt/fabinfra/adapters/shutdown/main.py state inuse 1

#GIVEBACK
/usr/bin/python3 /opt/fabinfra/adapters/shutdown/main.py state free
```

bffh.dhall Snippet

```
    shutdown =
    {
      module = "Process",
       params =
       {
          cmd = "/usr/bin/python3",
          args = "/opt/fabinfra/adapters/shutdow/main.py",
       }
    },
```

FabAccess Config Generator Snippet

```
vim /opt/fabinfra/fabaccess-config-generator/actors.ini
```

```
[shutdown]
#that script is so simple we do not need a special venv for it!
module = Process
param_cmd = "/usr/bin/python3"
param_args = "/opt/fabinfra/adapters/shutdown/main.py"
```

Version #5
Erstellt: 29 November 2024 15:26:11 von Mario Voigt (Stadtfabrikanten e.V.)
Zuletzt aktualisiert: 8 Dezember 2024 01:42:23 von Mario Voigt (Stadtfabrikanten e.V.)