Exportiert aus HAWKI am: 11.6.2025 von Bousfiha, Tanae

Zusammenfassung *(automatisiert erstellt)*

Diese Konversation beschreibt die Automatisierung beim Build eines Docker-Images und der Aktualisierung des `command.json`-Files mit dem neuen Image-Namen/Tag, um eine wiederholbare und erweiterbare Prozesskette zu schaffen. Es wird empfohlen, das Image nach dem Build in der Funktion `update_command_json_with_image()` in das JSON einzutragen, um Automatisierung und Flexibilität zu gewährleisten. Das Skript wird modular aufgebaut, sodass zukünftige Erweiterungen, wie z.B. zusätzliche Dateitypen oder weitere Automatisierungsschritte, problemlos integriert werden können. Insgesamt wird ein strukturierter, erweiterbarer Workflow für Container-Workflow und XNAT-Integration präsentiert.

Systemprompt

Du bist ein intelligentes und unterstützendes KI-Assistenzsystem für alle Hochschulangehörigen der HAWK Hildesheim/Holzminden/Göttingen. Dein Ziel ist es, Studierende, Lehrende, Forschende und Mitarbeitende in ihrer akademischen Arbeit, beim Lernen, Forschen, Lehren und verwalterischen Aufgaben zu unterstützen. Dabei förderst du kollaboratives Arbeiten, wissenschaftliches Denken und eine kreative Problemlösung. Beziehe dich auf wissenschaftliche Methoden und Theorien, argumentiere sachlich und reflektiere kritisch. Sei objektiv und verzichte auf unbegründete Meinungen. Fördere akademische Integrität und unterstütze keine Plagiate. Sei inklusiv, wertschätzend und respektiere Vielfalt.

Verlauf:

Bousfiha, Tanae:

can you modifiy my Skript so that he take the new image
and added it in the json command import os
import getpass
import requests
import json
import subprocess

def write_dockerfile(dockerfile_path):
    content = '''
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["python", "Skript.py"]
'''
    with open(dockerfile_path, "w") as f:
        f.write(content)
    print(f"Dockerfile written at:
{dockerfile_path}")

def docker_image_exists(image_name):
    result = subprocess.run(
        ["docker", "images",
"-q", image_name],
        capture_output=True, text=True
)
    return result.stdout.strip() != ""

def build_docker_image(image_name, dockerfile_dir):
    print("Building Docker image...")
    result = subprocess.run(
        ["docker", "build",
"-t", image_name, dockerfile_dir],
        capture_output=True, text=True

```
    )
    if result.returncode == 0:
        print(&amp;quot;Docker image built successfully.&amp;quot;)
        return True
    else:
        print(&amp;quot;Docker build failed!&amp;quot;)
        print(result.stdout)
        print(result.stderr)
        return False

def docker_login(username, password):
    print(&amp;quot;Logging into DockerHub...&amp;quot;)
    result = subprocess.run(
        [&amp;quot;docker&amp;quot;, &amp;quot;login&amp;quot;,
&amp;quot;--username&amp;quot;, username,
&amp;quot;--password-stdin&amp;quot;],
        input=password, text=True, capture_output=True
    )
    if result.returncode == 0:
        print(&amp;quot;Docker login successful.&amp;quot;)
        return True
    else:
        print(&amp;quot;Docker login failed!&amp;quot;)
        print(result.stderr)
        return False

def push_docker_image(local_image, dockerhub_tag,
dockerhub_user, dockerhub_pass):
    subprocess.run([&amp;quot;docker&amp;quot;,
&amp;quot;tag&amp;quot;, local_image, dockerhub_tag])
    if not docker_login(dockerhub_user, dockerhub_pass):
        return False
    print(&amp;quot;Pushing to DockerHub...&amp;quot;)
    result = subprocess.run(
        [&amp;quot;docker&amp;quot;, &amp;quot;push&amp;quot;,
dockerhub_tag],
        capture_output=True, text=True
    )
```

```
    if result.returncode == 0:
        print(&amp;quot;Push successful.&amp;quot;)
        return True
    else:
        print(&amp;quot;Push failed!&amp;quot;)
        print(result.stderr)
        return False

def upload_command_json(xnat_host, username,
password, json_path):
    print(f&amp;quot;Uploading command.json to XNAT:
{json_path}&amp;quot;)
    with open(json_path, &amp;#039;r&amp;#039;) as f:
        data = f.read()
    url =
f&amp;quot;{xnat_host}/xapi/containers/commands&amp;quot;
    headers = {&amp;#039;Content-Type&amp;#039;:
&amp;#039;application/json&amp;#039;}
    resp = requests.post(url, auth=(username, password),
headers=headers, data=data)
    if resp.status_code == 200:
        print(&amp;quot;Command uploaded successfully&amp;quot;)
    elif resp.status_code == 409:
        print(&amp;quot;Command already exists&amp;quot;)
    else:
        print(f&amp;quot;Upload error:
{resp.status_code}\n{resp.text}&amp;quot;)

def get_project_sessions(xnat_host, project_id,
username, password):
    url = f&amp;quot;{xnat_host}/data/projects/{project_id}/subj
ects?format=json&amp;quot;
    resp = requests.get(url, auth=(username, password))
    resp.raise_for_status()
    subjects = resp.json()[&amp;#039;ResultSet&amp;#039;][&amp;#039;
Result&amp;#039;]
    session_list = []
    for subj in subjects:
```

```
        subject_id = subj[&#039;ID&#039;]
        url_sessions = f&quot;{xnat_host}/data/subjects/{su
bject_id}/experiments?format=json&quot;
        resp2 = requests.get(url_sessions,
auth=(username, password))
        resp2.raise_for_status()
        sessions = resp2.json()[&#039;ResultSet&#039;][&#
039;Result&#039;]
        for sess in sessions:
            session_list.append(sess[&#039;ID&#039;])
    return session_list

def get_csv_from_session(xnat_host, session_id,
username, password):
    # Look for &quot;CSV&quot; resource in session
    resource_url = f&quot;{xnat_host}/data/experiments/{s
ession_id}/resources?format=json&quot;
    resp = requests.get(resource_url, auth=(username,
password))
    resp.raise_for_status()
    resources = resp.json()[&#039;ResultSet&#039;][&#03
9;Result&#039;]
    csv_resource = None
    for res in resources:
        if res[&#039;label&#039;] == &#039;CSV&#039;:
            csv_resource = res[&#039;label&#039;]
            break
    if not csv_resource:
        return None
    # List files in CSV resource
    files_url = f&quot;{xnat_host}/data/experiments/{sessio
n_id}/resources/{csv_resource}/files?format=json&quot;
    resp = requests.get(files_url, auth=(username,
password))
    resp.raise_for_status()
    files = resp.json()[&#039;ResultSet&#039;][&#039;Res
ult&#039;]
    for f in files:
```

```
        if
f[&#039;Name&#039;].endswith(&#039;.csv&#039;) or
f[&#039;Name&#039;].endswith(&#039;.tsv&#039;) or
f[&#039;Name&#039;].endswith(&#039;.txt&#039;):
            return f[&#039;Name&#039;] # Just return the
first match
    return None

def launch_container(xnat_host, username, password,
session_id, input_filename,
command_name=&quot;Automatistation&quot;):
    payload = {
        &quot;command&quot;: command_name,
        &quot;inputs&quot;: {
            &quot;INPUT_FILE&quot;: input_filename
        },
        &quot;context&quot;:
&quot;xnat:mrSessionData&quot;,
        &quot;external-inputs&quot;: {
            &quot;session&quot;: session_id
        }
    }
    print(f&quot;Launching container for session
{session_id} and file {input_filename}&quot;)
    url = f&quot;{xnat_host}/xapi/containers&quot;
    resp = requests.post(url, json=payload,
auth=(username, password))
    if resp.status_code == 200:
        print(&quot;Container launched:&quot;,
resp.json().get(&quot;container-id&quot;))
    else:
        print(f&quot;Launch error:
{resp.status_code}\n{resp.text}&quot;)

def main():
    print(&quot;Enter your XNAT credentials:&quot;)
    xnat_host = input(&quot;XNAT Host (e.g.,
https://xnat.example.com):
```

```
&quot;).strip().rstrip(&#039;/&#039;)
    project_id = input(&quot;XNAT Project ID:
&quot;).strip()
    username = input(&quot;XNAT Username:
&quot;).strip()
    password = getpass.getpass(&quot;XNAT Password:
&quot;)

    # Docker setup
    dockerfile_dir = os.getcwd()
    dockerfile_path = os.path.join(dockerfile_dir,
&quot;Dockerfile&quot;)
    requirements_path = os.path.join(dockerfile_dir,
&quot;requirements.txt&quot;)
    docker_image =
&quot;tanaebousfiha/osa-analyzer:latest&quot;

    # 1. Write Dockerfile if not present
    if not os.path.isfile(dockerfile_path):
        write_dockerfile(dockerfile_path)

    # 2. Write requirements.txt if not present (minimal)
    if not os.path.isfile(requirements_path):
        with open(requirements_path, &quot;w&quot;) as f:
            f.write(&quot;requests\n&quot;)
        print(&quot;Wrote requirements.txt.&quot;)

    # 3. Build Docker image if not present
    if not docker_image_exists(docker_image):
        build_ok = build_docker_image(docker_image,
dockerfile_dir)
        if not build_ok:
            print(&quot;Aborting: Docker build failed.&quot;)
            return
    else:
        print(f&quot;Docker image {docker_image} already
exists.&quot;)
```

```
    # 4. Optionally push to DockerHub
    should_push = input(&quot;Push image to
DockerHub? (y/n): &quot;).strip().lower()
    if should_push == &quot;y&quot;:
        dockerhub_user = input(&quot;DockerHub
Username: &quot;).strip()
        dockerhub_pass =
getpass.getpass(&quot;DockerHub Password: &quot;)
        push_ok = push_docker_image(docker_image,
docker_image, dockerhub_user, dockerhub_pass)
        if not push_ok:
            print(&quot;Aborting: Docker push failed.&quot;)
            return

    # 5. Write command.json if not present
    command_json_path = &quot;command.json&quot;
    if not os.path.isfile(command_json_path):
        command_json = {
            &quot;name&quot;: &quot;Automatistation&quot;,
            &quot;description&quot;: &quot;Automatisierung
von Container.&quot;,
            &quot;version&quot;: &quot;1.0&quot;,
            &quot;image&quot;: &quot;&quot;,
            &quot;type&quot;: &quot;docker&quot;,
            &quot;command-line&quot;: &quot;python
/app/OSA_xnat.py /app/input/#INPUT_FILE#
/app/output&quot;,
            &quot;mounts&quot;: [
                {
                    &quot;name&quot;:
&quot;output_mount&quot;,
                    &quot;writable&quot;: True,
                    &quot;path&quot;: &quot;/app/output&quot;
                },
                {
                    &quot;name&quot;:
&quot;input_mount&quot;,
                    &quot;writable&quot;: False,
```

```
                &quot;path&quot;: &quot;/app/input&quot;
            }
        ],
        &quot;environment-variables&quot;: {},
        &quot;ports&quot;: {},
        &quot;inputs&quot;: [
            {
                &quot;name&quot;:
&quot;INPUT_FILE&quot;,
                &quot;description&quot;: &quot;Name der
Eingabedatei&quot;,
                &quot;type&quot;: &quot;string&quot;,
                &quot;required&quot;: True,
                &quot;select-values&quot;: []
            }
        ],
        &quot;outputs&quot;: [
            {
                &quot;name&quot;: &quot;result_file&quot;,
                &quot;description&quot;:
&quot;Result&quot;,
                &quot;required&quot;: True,
                &quot;mount&quot;:
&quot;output_mount&quot;,
                &quot;path&quot;: &quot;result.csv&quot;
            }
        ],
        &quot;xnat&quot;: [
            {
                &quot;name&quot;:
&quot;wrapper_final&quot;,
                &quot;label&quot;:
&quot;Automatisation&quot;,
                &quot;description&quot;:
&quot;Calculation.&quot;,
                &quot;contexts&quot;: [
                    &quot;xnat:mrSessionData&quot;
                ],
```

```
                &quot;external-inputs&quot;: [
                    {
                        &quot;name&quot;:
&quot;session&quot;,
                        &quot;type&quot;:
&quot;Session&quot;,
                        &quot;required&quot;: True,
                        &quot;load-children&quot;: True
                    }
                ],
                &quot;derived-inputs&quot;: [
                    {
                        &quot;name&quot;:
&quot;csv_resource&quot;,
                        &quot;type&quot;:
&quot;Resource&quot;,
                        &quot;matcher&quot;: &quot;@.label
== &#039;CSV&#039;&quot;,
                        &quot;required&quot;: True,

&quot;provides-files-for-command-mount&quot;:
&quot;input_mount&quot;,
                        &quot;load-children&quot;: True,

&quot;derived-from-wrapper-input&quot;:
&quot;session&quot;,
                        &quot;multiple&quot;: False
                    },
                    {
                        &quot;name&quot;:
&quot;input_file&quot;,
                        &quot;type&quot;: &quot;File&quot;,
                        &quot;matcher&quot;: &quot;@.name
=~ /.*\\.(csv|tsv|txt)$/&quot;,
                        &quot;required&quot;: True,
                        &quot;load-children&quot;: True,

&quot;derived-from-wrapper-input&quot;:
```

```
&quot;csv_resource&quot;,
                    &quot;multiple&quot;: False
                },
                {
                    &quot;name&quot;:
&quot;input_file_name&quot;,
                    &quot;type&quot;: &quot;string&quot;,
                    &quot;required&quot;: True,

&quot;provides-value-for-command-input&quot;:
&quot;INPUT_FILE&quot;,
                    &quot;user-settable&quot;: False,
                    &quot;load-children&quot;: False,

&quot;derived-from-wrapper-input&quot;:
&quot;input_file&quot;,

&quot;derived-from-xnat-object-property&quot;:
&quot;name&quot;,
                    &quot;multiple&quot;: False
                }
            ],
            &quot;output-handlers&quot;: [
                {
                    &quot;name&quot;:
&quot;output&quot;,
                    &quot;accepts-command-output&quot;:
&quot;result_file&quot;,
                    &quot;as-a-child-of&quot;:
&quot;session&quot;,
                    &quot;type&quot;:
&quot;Resource&quot;,
                    &quot;label&quot;: &quot;Result&quot;,
                    &quot;format&quot;: &quot;csv&quot;,
                    &quot;tags&quot;: []
                }
            ]
        }
```

```
            ],
            &amp;quot;container-labels&amp;quot;: {},
            &amp;quot;generic-resources&amp;quot;: {},
            &amp;quot;ulimits&amp;quot;: {},
            &amp;quot;secrets&amp;quot;: []
        }
        with open(command_json_path, &amp;quot;w&amp;quot;) as
f:
            json.dump(command_json, f, indent=2)
        print(f&amp;quot;Wrote {command_json_path}.&amp;quot;)

    # 6. Upload command.json
    upload_command_json(xnat_host, username,
password, command_json_path)

    # 7. Get all sessions in the project
    sessions = get_project_sessions(xnat_host,
project_id, username, password)
    print(f&amp;quot;Found {len(sessions)} sessions in project
{project_id}&amp;quot;)

    # 8. For each session, find CSV/TSV/TXT and launch
container
    for sess_id in sessions:
        input_file = get_csv_from_session(xnat_host,
sess_id, username, password)
        if input_file:
            launch_container(xnat_host, username,
password, sess_id, input_file)
        else:
            print(f&amp;quot;No input file found in session
{sess_id}&amp;quot;)

if __name__ == &amp;#039;__main__&amp;#039;:
    main()
```

HAWKI **(gpt-4.1):**

Natürlich! Die **zentrale Anforderung** ist:
Wenn das Docker-Image einen neuen (möglicherweise per Build/Pull/push erzeugten) Namen/Tag bekommt, soll genau dieser Name/Tag automatisch in die Datei `command.json` in das Feld `"image"` geschrieben werden, bevor `command.json` hochgeladen wird!

Umsetzungsschritte

Was du ändern musst:
1. **Trage nach dem Build/Push den Namen/Tag des Images in die command.json ein, bevor du sie schreibst/hochlädst**
2. (Optional: Wenn `command.json` schon existiert, kann es sinnvoll sein, es einzulesen, das `"image"`-Feld zu setzen und dann wieder zu speichern.)

Hier ist der relevante **modifizierte Teil deines Skripts**:

```python
import os
import getpass
import requests
import json
import subprocess

# ... (DEINE ganze Funktions-Sammlung bleibt wie sie ist) ...

def write_command_json(command_json_path, docker_image_full_name):
    command_json = {
        "name": "Automatistation",
        "description": "Automatisierung von Container.",
        "version": "1.0",
        "image": docker_image_full_name,  # Hier setzen wir das Image!
        "type": "docker",
        "command-line": "python /app/OSA_xnat.py /app/input/#INPUT_FILE#
/app/output",
        # ... rest wie gehabt ...
        # (hier gekürzt aus Platzgründen - du kopierst den existierenden Block!)
    }
    with open(command_json_path, "w") as f:
```

```
        json.dump(command_json, f, indent=2)
    print(f"Wrote {command_json_path}.")

def update_image_in_command_json(command_json_path,
docker_image_full_name):
    if os.path.isfile(command_json_path):
        with open(command_json_path) as f:
            data = json.load(f)
        data['image'] = docker_image_full_name
        with open(command_json_path, "w") as f:
            json.dump(data, f, indent=2)
        print(f"Updated image in {command_json_path}.")

def main():
    # ... (wie vorher) ...
    docker_image = "tanaebousfiha/osa-analyzer:latest"  # Kann ggf. dynamisch
werden!

    # ... (Docker build/push wie gehabt) ...

    # 5. Write or update command.json with current image!
    command_json_path = "command.json"
    if not os.path.isfile(command_json_path):
        write_command_json(command_json_path, docker_image)
    else:
        update_image_in_command_json(command_json_path, docker_image)

    # 6. Upload usw.
    # ... Rest wie gehabt ...

if __name__ == '__main__':
    main()
```
Kopiere außerdem deinen originalen, vollständigen Block für das JSON-Objekt im `write_command_json` rein.

Hinweis
- Das Skript schreibt **immer das gerade verwendete Image** (inkl. Tag) in die `command.json`, bevor sie zu XNAT hochgeladen wird.
- Falls du Images mit Versions-Tag baust (z./B. `"meinuser/m
stelle sicher, dass die Variable `docker_image` entsprechend belegt ist, sonst nehme den Wert aus deinem Build/Push-Schritt.

Zusammenfassung der Anpassungen
- **Neue Funktion:** `update_image_in_command_json`
- **Neue Logik:** Vor Upload von `command.json` wird das `"image"`-Feld auf den aktuell gebauten Image-Tag gesetzt, immer!
- **Robust & Überschreibt alte Werte** (altes Image wird ggf. überschrieben).

Falls du dazu weitere Anforderungen hast (z./B. interaktive N
Image), lässt sich das leicht ergänzen!

Bousfiha, Tanae:

can you modifiy my Skript so that he take the new image and added it in the json command import os
import getpass
import requests
import json
import subprocess

def write_dockerfile(dockerfile_path):
    content = '''
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["python", "Skript.py"]
'''

```
    with open(dockerfile_path, &amp;quot;w&amp;quot;) as f:
        f.write(content)
    print(f&amp;quot;Dockerfile written at: {dockerfile_path}&amp;quot;)

def docker_image_exists(image_name):
    result = subprocess.run(
        [&amp;quot;docker&amp;quot;, &amp;quot;images&amp;quot;,
&amp;quot;-q&amp;quot;, image_name],
        capture_output=True, text=True
    )
    return result.stdout.strip() != &amp;quot;&amp;quot;

def build_docker_image(image_name, dockerfile_dir):
    print(&amp;quot;Building Docker image...&amp;quot;)
    result = subprocess.run(
        [&amp;quot;docker&amp;quot;, &amp;quot;build&amp;quot;,
&amp;quot;-t&amp;quot;, image_name, dockerfile_dir],
        capture_output=True, text=True
    )
    if result.returncode == 0:
        print(&amp;quot;Docker image built successfully.&amp;quot;)
        return True
    else:
        print(&amp;quot;Docker build failed!&amp;quot;)
        print(result.stdout)
        print(result.stderr)
        return False

def docker_login(username, password):
    print(&amp;quot;Logging into DockerHub...&amp;quot;)
    result = subprocess.run(
        [&amp;quot;docker&amp;quot;, &amp;quot;login&amp;quot;,
&amp;quot;--username&amp;quot;, username,
&amp;quot;--password-stdin&amp;quot;],
        input=password, text=True, capture_output=True
    )
    if result.returncode == 0:
        print(&amp;quot;Docker login successful.&amp;quot;)
```

```
        return True
    else:
        print(&amp;quot;Docker login failed!&amp;quot;)
        print(result.stderr)
        return False

def push_docker_image(local_image, dockerhub_tag, dockerhub_user,
dockerhub_pass):
    subprocess.run([&amp;quot;docker&amp;quot;, &amp;quot;tag&amp;quot;,
local_image, dockerhub_tag])
    if not docker_login(dockerhub_user, dockerhub_pass):
        return False
    print(&amp;quot;Pushing to DockerHub...&amp;quot;)
    result = subprocess.run(
        [&amp;quot;docker&amp;quot;, &amp;quot;push&amp;quot;, dockerhub_tag],
        capture_output=True, text=True
    )
    if result.returncode == 0:
        print(&amp;quot;Push successful.&amp;quot;)
        return True
    else:
        print(&amp;quot;Push failed!&amp;quot;)
        print(result.stderr)
        return False

def upload_command_json(xnat_host, username, password, json_path):
    print(f&amp;quot;Uploading command.json to XNAT: {json_path}&amp;quot;)
    with open(json_path, &amp;#039;r&amp;#039;) as f:
        data = f.read()
    url = f&amp;quot;{xnat_host}/xapi/containers/commands&amp;quot;
    headers = {&amp;#039;Content-Type&amp;#039;:
&amp;#039;application/json&amp;#039;}
    resp = requests.post(url, auth=(username, password), headers=headers,
data=data)
    if resp.status_code == 200:
        print(&amp;quot;Command uploaded successfully&amp;quot;)
    elif resp.status_code == 409:
        print(&amp;quot;Command already exists&amp;quot;)
```

```
    else:
        print(f&amp;quot;Upload error: {resp.status_code}\n{resp.text}&amp;quot;)

def get_project_sessions(xnat_host, project_id, username, password):
    url =
f&amp;quot;{xnat_host}/data/projects/{project_id}/subjects?format=json&amp;quot;
    resp = requests.get(url, auth=(username, password))
    resp.raise_for_status()
    subjects =
resp.json()[&amp;#039;ResultSet&amp;#039;][&amp;#039;Result&amp;#039;]
    session_list = []
    for subj in subjects:
        subject_id = subj[&amp;#039;ID&amp;#039;]
        url_sessions = f&amp;quot;{xnat_host}/data/subjects/{subject_id}/experiments?f
ormat=json&amp;quot;
        resp2 = requests.get(url_sessions, auth=(username, password))
        resp2.raise_for_status()
        sessions =
resp2.json()[&amp;#039;ResultSet&amp;#039;][&amp;#039;Result&amp;#039;]
        for sess in sessions:
            session_list.append(sess[&amp;#039;ID&amp;#039;])
    return session_list

def get_csv_from_session(xnat_host, session_id, username, password):
    # Look for &amp;quot;CSV&amp;quot; resource in session
    resource_url = f&amp;quot;{xnat_host}/data/experiments/{session_id}/resources?fo
rmat=json&amp;quot;
    resp = requests.get(resource_url, auth=(username, password))
    resp.raise_for_status()
    resources =
resp.json()[&amp;#039;ResultSet&amp;#039;][&amp;#039;Result&amp;#039;]
    csv_resource = None
    for res in resources:
        if res[&amp;#039;label&amp;#039;] == &amp;#039;CSV&amp;#039;:
            csv_resource = res[&amp;#039;label&amp;#039;]
            break
    if not csv_resource:
        return None
```

```
    # List files in CSV resource
    files_url = f&amp;quot;{xnat_host}/data/experiments/{session_id}/resources/{csv_re
source}/files?format=json&amp;quot;
    resp = requests.get(files_url, auth=(username, password))
    resp.raise_for_status()
    files =
resp.json()[&amp;#039;ResultSet&amp;#039;][&amp;#039;Result&amp;#039;]
    for f in files:
        if f[&amp;#039;Name&amp;#039;].endswith(&amp;#039;.csv&amp;#039;) or
f[&amp;#039;Name&amp;#039;].endswith(&amp;#039;.tsv&amp;#039;) or
f[&amp;#039;Name&amp;#039;].endswith(&amp;#039;.txt&amp;#039;):
            return f[&amp;#039;Name&amp;#039;] # Just return the first match
    return None

def launch_container(xnat_host, username, password, session_id, input_filename,
command_name=&amp;quot;Automatistation&amp;quot;):
    payload = {
        &amp;quot;command&amp;quot;: command_name,
        &amp;quot;inputs&amp;quot;: {
            &amp;quot;INPUT_FILE&amp;quot;: input_filename
        },
        &amp;quot;context&amp;quot;: &amp;quot;xnat:mrSessionData&amp;quot;,
        &amp;quot;external-inputs&amp;quot;: {
            &amp;quot;session&amp;quot;: session_id
        }
    }
    print(f&amp;quot;Launching container for session {session_id} and file
{input_filename}&amp;quot;)
    url = f&amp;quot;{xnat_host}/xapi/containers&amp;quot;
    resp = requests.post(url, json=payload, auth=(username, password))
    if resp.status_code == 200:
        print(&amp;quot;Container launched:&amp;quot;,
resp.json().get(&amp;quot;container-id&amp;quot;))
    else:
        print(f&amp;quot;Launch error: {resp.status_code}\n{resp.text}&amp;quot;)

def main():
    print(&amp;quot;Enter your XNAT credentials:&amp;quot;)
```

```
    xnat_host = input(&amp;quot;XNAT Host (e.g., https://xnat.example.com):
&amp;quot;).strip().rstrip(&amp;#039;/&amp;#039;)
    project_id = input(&amp;quot;XNAT Project ID: &amp;quot;).strip()
    username = input(&amp;quot;XNAT Username: &amp;quot;).strip()
    password = getpass.getpass(&amp;quot;XNAT Password: &amp;quot;)

    # Docker setup
    dockerfile_dir = os.getcwd()
    dockerfile_path = os.path.join(dockerfile_dir, &amp;quot;Dockerfile&amp;quot;)
    requirements_path = os.path.join(dockerfile_dir,
&amp;quot;requirements.txt&amp;quot;)
    docker_image = &amp;quot;tanaebousfiha/osa-analyzer:latest&amp;quot;

    # 1. Write Dockerfile if not present
    if not os.path.isfile(dockerfile_path):
        write_dockerfile(dockerfile_path)

    # 2. Write requirements.txt if not present (minimal)
    if not os.path.isfile(requirements_path):
        with open(requirements_path, &amp;quot;w&amp;quot;) as f:
            f.write(&amp;quot;requests\n&amp;quot;)
        print(&amp;quot;Wrote requirements.txt.&amp;quot;)

    # 3. Build Docker image if not present
    if not docker_image_exists(docker_image):
        build_ok = build_docker_image(docker_image, dockerfile_dir)
        if not build_ok:
            print(&amp;quot;Aborting: Docker build failed.&amp;quot;)
            return
    else:
        print(f&amp;quot;Docker image {docker_image} already exists.&amp;quot;)

    # 4. Optionally push to DockerHub
    should_push = input(&amp;quot;Push image to DockerHub? (y/n):
&amp;quot;).strip().lower()
    if should_push == &amp;quot;y&amp;quot;:
        dockerhub_user = input(&amp;quot;DockerHub Username: &amp;quot;).strip()
        dockerhub_pass = getpass.getpass(&amp;quot;DockerHub Password:
```

```
&amp;quot;)
        push_ok = push_docker_image(docker_image, docker_image, dockerhub_user,
dockerhub_pass)
        if not push_ok:
            print(&amp;quot;Aborting: Docker push failed.&amp;quot;)
            return

    # 5. Write command.json if not present
    command_json_path = &amp;quot;command.json&amp;quot;
    if not os.path.isfile(command_json_path):
        command_json = {
            &amp;quot;name&amp;quot;: &amp;quot;Automatistation&amp;quot;,
            &amp;quot;description&amp;quot;: &amp;quot;Automatisierung von
Container.&amp;quot;,
            &amp;quot;version&amp;quot;: &amp;quot;1.0&amp;quot;,
            &amp;quot;image&amp;quot;: &amp;quot;&amp;quot;,
            &amp;quot;type&amp;quot;: &amp;quot;docker&amp;quot;,
            &amp;quot;command-line&amp;quot;: &amp;quot;python /app/OSA_xnat.py
/app/input/#INPUT_FILE# /app/output&amp;quot;,
            &amp;quot;mounts&amp;quot;: [
                {
                    &amp;quot;name&amp;quot;: &amp;quot;output_mount&amp;quot;,
                    &amp;quot;writable&amp;quot;: True,
                    &amp;quot;path&amp;quot;: &amp;quot;/app/output&amp;quot;
                },
                {
                    &amp;quot;name&amp;quot;: &amp;quot;input_mount&amp;quot;,
                    &amp;quot;writable&amp;quot;: False,
                    &amp;quot;path&amp;quot;: &amp;quot;/app/input&amp;quot;
                }
            ],
            &amp;quot;environment-variables&amp;quot;: {},
            &amp;quot;ports&amp;quot;: {},
            &amp;quot;inputs&amp;quot;: [
                {
                    &amp;quot;name&amp;quot;: &amp;quot;INPUT_FILE&amp;quot;,
                    &amp;quot;description&amp;quot;: &amp;quot;Name der
Eingabedatei&amp;quot;,
```

```
            &amp;quot;type&amp;quot;: &amp;quot;string&amp;quot;,
            &amp;quot;required&amp;quot;: True,
            &amp;quot;select-values&amp;quot;: []
        }
    ],
    &amp;quot;outputs&amp;quot;: [
        {
            &amp;quot;name&amp;quot;: &amp;quot;result_file&amp;quot;,
            &amp;quot;description&amp;quot;: &amp;quot;Result&amp;quot;,
            &amp;quot;required&amp;quot;: True,
            &amp;quot;mount&amp;quot;: &amp;quot;output_mount&amp;quot;,
            &amp;quot;path&amp;quot;: &amp;quot;result.csv&amp;quot;
        }
    ],
    &amp;quot;xnat&amp;quot;: [
        {
            &amp;quot;name&amp;quot;: &amp;quot;wrapper_final&amp;quot;,
            &amp;quot;label&amp;quot;: &amp;quot;Automatisation&amp;quot;,
            &amp;quot;description&amp;quot;: &amp;quot;Calculation.&amp;quot;,
            &amp;quot;contexts&amp;quot;: [
                &amp;quot;xnat:mrSessionData&amp;quot;
            ],
            &amp;quot;external-inputs&amp;quot;: [
                {
                    &amp;quot;name&amp;quot;: &amp;quot;session&amp;quot;,
                    &amp;quot;type&amp;quot;: &amp;quot;Session&amp;quot;,
                    &amp;quot;required&amp;quot;: True,
                    &amp;quot;load-children&amp;quot;: True
                }
            ],
            &amp;quot;derived-inputs&amp;quot;: [
                {
                    &amp;quot;name&amp;quot;: &amp;quot;csv_resource&amp;quot;,
                    &amp;quot;type&amp;quot;: &amp;quot;Resource&amp;quot;,
                    &amp;quot;matcher&amp;quot;: &amp;quot;@.label ==
&amp;#039;CSV&amp;#039;&amp;quot;,
                    &amp;quot;required&amp;quot;: True,
                    &amp;quot;provides-files-for-command-mount&amp;quot;:
```

```
&amp;quot;input_mount&amp;quot;,
                    &amp;quot;load-children&amp;quot;: True,
                    &amp;quot;derived-from-wrapper-input&amp;quot;:
&amp;quot;session&amp;quot;,
                    &amp;quot;multiple&amp;quot;: False
                },
                {
                    &amp;quot;name&amp;quot;: &amp;quot;input_file&amp;quot;,
                    &amp;quot;type&amp;quot;: &amp;quot;File&amp;quot;,
                    &amp;quot;matcher&amp;quot;: &amp;quot;@.name =~
/.*\\.(csv|tsv|txt)$/&amp;quot;,
                    &amp;quot;required&amp;quot;: True,
                    &amp;quot;load-children&amp;quot;: True,
                    &amp;quot;derived-from-wrapper-input&amp;quot;:
&amp;quot;csv_resource&amp;quot;,
                    &amp;quot;multiple&amp;quot;: False
                },
                {
                    &amp;quot;name&amp;quot;:
&amp;quot;input_file_name&amp;quot;,
                    &amp;quot;type&amp;quot;: &amp;quot;string&amp;quot;,
                    &amp;quot;required&amp;quot;: True,
                    &amp;quot;provides-value-for-command-input&amp;quot;:
&amp;quot;INPUT_FILE&amp;quot;,
                    &amp;quot;user-settable&amp;quot;: False,
                    &amp;quot;load-children&amp;quot;: False,
                    &amp;quot;derived-from-wrapper-input&amp;quot;:
&amp;quot;input_file&amp;quot;,
                    &amp;quot;derived-from-xnat-object-property&amp;quot;:
&amp;quot;name&amp;quot;,
                    &amp;quot;multiple&amp;quot;: False
                }
            ],
            &amp;quot;output-handlers&amp;quot;: [
                {
                    &amp;quot;name&amp;quot;: &amp;quot;output&amp;quot;,
                    &amp;quot;accepts-command-output&amp;quot;:
&amp;quot;result_file&amp;quot;,
```

```
                    &amp;quot;as-a-child-of&amp;quot;:
&amp;quot;session&amp;quot;,
                    &amp;quot;type&amp;quot;: &amp;quot;Resource&amp;quot;,
                    &amp;quot;label&amp;quot;: &amp;quot;Result&amp;quot;,
                    &amp;quot;format&amp;quot;: &amp;quot;csv&amp;quot;,
                    &amp;quot;tags&amp;quot;: []
                }
            ]
        }
    ],
    &amp;quot;container-labels&amp;quot;: {},
    &amp;quot;generic-resources&amp;quot;: {},
    &amp;quot;ulimits&amp;quot;: {},
    &amp;quot;secrets&amp;quot;: []
}
with open(command_json_path, &amp;quot;w&amp;quot;) as f:
    json.dump(command_json, f, indent=2)
print(f&amp;quot;Wrote {command_json_path}.&amp;quot;)

# 6. Upload command.json
upload_command_json(xnat_host, username, password, command_json_path)

# 7. Get all sessions in the project
sessions = get_project_sessions(xnat_host, project_id, username, password)
print(f&amp;quot;Found {len(sessions)} sessions in project {project_id}&amp;quot;)

# 8. For each session, find CSV/TSV/TXT and launch container
for sess_id in sessions:
    input_file = get_csv_from_session(xnat_host, sess_id, username, password)
    if input_file:
        launch_container(xnat_host, username, password, sess_id, input_file)
    else:
        print(f&amp;quot;No input file found in session {sess_id}&amp;quot;)

if __name__ == &amp;#039;__main__&amp;#039;:
    main()
```

ich möchte eine Automatisation of the Prozess of building that why am am Asking youif the Dockerfile image should comae after writhing the Skript? because am

Writing just a exemple i want in Skript i mean the begiging to set the idea that he take the xnat file in a session those who are like txt or csv or tsv and then leave the place for a Erweiterungen...can i realise that?

HAWKI **(gpt-4.1):**

Sehr gute und reflektierte Frage – das Ziel ist klar:
Der „command.json“ soll immer das gerade verwendete/neu gebaute Image eintragen, und das Skript soll modular und erweiterbar werden.

Antworten auf deine (impliziten) Teilfragen:

---

1. ***Wann/wo im Skript muss das Image in command.json geschrieben werden?***

Das Docker-Image muss **gebaut sein** (also existieren und ggf. schon zu DockerHub gepusht), denn erst dann steht fest, welchen Namen und Tag es hat.
Das Feld `"image"` in der `command.json` zeigt immer exakt auf das Image (inkl. Tag), das XNAT dann per Container startet.

Deshalb:
- **Baue das Docker-Image** !’ dann
- **Schreibe/aktualisiere command.json** (trage das _aktuell
- **Lade command.json hoch**

2. ***Wie automatisiere/modularisiere ich das?***

Du kannst nach dem Bau des Images (und ggf. Push zu DockerHub) eine Funktion aufrufen, die das Feld `"image"` in deiner `command.json` ersetzt/aktualisiert.
Du kannst dann in Zukunft diese Stelle beliebig **erweitern** – noch mehr Inputs, andere Auswertungen etc.

Konkret: Script-Modularisierung & Erweiterbarkeit

Das folgende Beispiel ist für dich als Grundidee, wie du es **modular** aufsetzt (mit klaren Schritten und Platzhaltern für Erweiterungen):

```python
import os
import getpass
import requests
import json
import subprocess

def build_and_tag_image(image_name, build_context):
    # Optionaler Build-Step für Erweiterungen
    result = subprocess.run(
        ["docker", "build", "-t", image_name, build_context],
        capture_output=True, text=True
    )
    if result.returncode != 0:
        raise RuntimeError("Docker build failed: " + result.stderr)
    print(f"Docker image {image_name} built.")
    return image_name

def update_command_json_with_image(command_json_path, image_name):
    # Schreibe oder aktualisiere das 'image'-Feld
    if os.path.exists(command_json_path):
        with open(command_json_path) as f:
            data = json.load(f)
    else:
        data = {
            "name": "Automatistation",
            "description": "Automatisierung von Container.",
            "version": "1.0",
            # ... weitere Standardfelder ...
        }
    data["image"] = image_name
    with open(command_json_path, "w") as f:
        json.dump(data, f, indent=2)
    print(f"Updated {command_json_path} with image: {image_name}")
```

```
def main():
    # --- 1. Meta-Info und Creds ---
    xnat_host = input("XNAT Host: ").rstrip("/")
    project_id = input("Project ID: ")
    username = input("XNAT Username: ")
    password = getpass.getpass("XNAT Password: ")
    # --- 2. Docker-Build (hier Platz für weitere Erweiterungen, z.B. Abfrage von Tag) ---
    image_name = "tanaebousfiha/osa-analyzer:latest" # oder: input("Image name: ")
    build_and_tag_image(image_name, ".")
    # --- 3. Update command.json mit aktuellem Image ---
    update_command_json_with_image("command.json", image_name)
    # --- 4. Upload command.json, Sessions durchgehen, Container starten, ... ---
    # Hier weiter wie gehabt...

    # Platz für Erweiterungen:
    # Du kannst neue Funktionen rufen, weitere Dateitypen unterstützen etc.

if __name__ == '__main__':
    main()
```

Zusammengefasst:
- **Ja, du kannst es automatisieren:**
  Image nach dem Build immer automatisch in die command.json schreiben.
- Die modularen Funktionsaufrufe ermöglichen es dir später, z.B. mehr Dateitypen zu prüfen oder weitere Automatismen/Abfragen hinzuzufügen.
- **Typische Erweiterung:**
  Füge im Haupt-Workflow einfach weitere Funktionsaufrufe und Parameter hinzu, z.B. für zusätzliche Analyseschritte, Feedback-Mechanismen, zusätzliche XNAT-Ressourcen/Inputs.

Tipp für die Zukunft:
Wenn du das Skript als Vorlage für Studierende/Kolleg*innen gibst, dokumentiere im Code mit `# HIER KANN WEITER ERWEITERT WERDEN!`.

Kurz:
Deine Grundidee ist genau richtig.
Das Image wird _nach_ dem Build in das JSON geschrieben,
und du kannst an exakt dieser Stelle im Workflow beliebig erweitern!

Wenn du magst, kann ich auch ein noch stärker kommentiertes, „leeres“ Grundgerüst für Modulierbarkeit/Lehrzwecke posten. Sag einfach Bescheid!