

Zusammenfassung *(automatisiert erstellt)*

Die Konversation dreht sich um die Automatisierung des Container-Launchs in XNAT per API. Der Nutzer entdeckt, dass die bisher verwendeten Endpunkte (wie ``/xapi/container-launcher/launch``) nicht unterstützt werden, was zu einem 404-Fehler führt. Es wird klargestellt, dass die verfügbaren APIs stattdessen Endpunkte wie ``/projects/{project}/commands/{commandId}/wrappers/{wrapperName}/launch`` bieten, um Container-Wrapper direkt zu starten. Ein Beispiel-Python-Skript wird angepasst, um diese Endpunkte korrekt zu nutzen. Wichtig sind korrekte URLs, Payload-Struktur und Identifikatoren. Ein zentraler Tipp ist, die tatsächlichen Inputs für den Wrapper über die Launch-UI im Browser zu ermitteln.

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:

tanae@mi0082n:~/dockerfile/xnat-osa/Ziel1_osaDocker

Container\$ python3 WRAPPERCODE.py

XNAT Username:tanae

XNAT Password:

Project ID:tanae_osa_Predct

Session ID:XNAT_E00428

Path to the Python script:Bonjour.py

What is the name of the command in XNAT: fillerinjektion

What is the description of the command in XNAT: how many ml has the patient filler injected

Dockerfile written to ./Dockerfile

Building Docker image

'fillerinjektion:latest'...

Docker image 'fillerinjektion:latest' built successfully.

JSON file created at command.json

Uploading command to

<https://xnat-dev.gwdg.de/xapi/commands>

Command created successfully.

Wrapper existiert bereits: 101

Wrapper 'fillerinjektion_wrapper' wurde global aktiviert.

Wrapper 'fillerinjektion_wrapper' wurde im

Projekt 'tanae_osa_Predct' aktiviert.

Launching container with payload: {

"command": 80,

"wrapper": 101,

"inputs": {

"session": "XNAT_E00428"

}

}

Failed to launch container: 404 - <!doctype

html><html

lang="en"><head><title>HTTP

Status 404 – Not Found</title><style

```

type="text/css">body
{font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b
{color:white;background-color:#525D76;} h1
{font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;}
p {font-size:12px;} a {color:black;} .line {height:1px;backg
round-color:#525D76;border:none;}</style></head
<><body><h1>HTTP Status 404 – Not
Found</h1><hr class="line"
/><p><b>Type</b> Status
Report</p><p><b>Description</b>
The origin server did not find a current representation for
the target resource or is not willing to disclose that one
exists.</p><hr class="line"
/><h3>Apache Tomcat/9.0.43
(Debian)</h3></body></html>

```

HAWKI (gpt-4.1):

Deine Fehlermeldung `Failed to launch container: 404` zeigt, dass beim Versuch, den XNAT-Container zu starten, ein HTTP-Fehler 404 ("Not Found") zurückgegeben wurde. Das bedeutet, dass die angeforderte Ressource oder API-URL auf dem Server `https://xnat-dev.gwdg.de` **nicht gefunden wurde**.

Hier sind mögliche Ursachen und Lösungsansätze, die speziell auf XNAT und das Container Service (XCS) eingehen:

1. Falsche oder veraltete API-Endpunkte

Der 404-Fehler tritt häufig auf, wenn der API-Endpunkt falsch ist oder nicht (mehr) existiert. Prüfe daher:

- **URL:** Passt die verwendete API-URL (`/xapi/containers` o.ä.) zur XNAT-Server-Version und ist sie korrekt?
- **API-Referenz:** [XNAT Container Service API Doku](https://wiki.xnat.org/documentation/how-to-use-xnat-s-container-service/container-service-api)

Beispiel für einen typischen Endpunkt (je nach XNAT-Version):

...

POST /xapi/containers

// oder ggf.

POST /xapi/projects/{projectID}/wrappers/{wrapperID}/launch

...

Vergleiche unbedingt mit der [XNAT-Instanz deiner Einrichtung](<https://xnat-dev.gwdg.de/xapi/swagger-ui.html>) die verfügbaren Endpunkte!

2. Fehlende Ressourcen (Command, Wrapper, Session usw.)

- **Existieren die angegebenen Command/Wrapper IDs noch?** (Hier: `command: 80`, `wrapper: 101`)
- **Ist die Session-Nummer korrekt und im richtigen Projekt vorhanden?** (`XNAT_E00428` unter `tanae_osa_Predct`)
- **Hast du die nötigen Rechte im Projekt?**

3. Fehlerhafte Payload/Struktur

- Prüfe, ob die an die API gesendeten Daten (Payload/JSON) die erwarteten Felder haben und nicht fehlerhaft sind.

4. XNAT-/Container Service Plugin installiert und aktiviert

- Ist das XNAT Container Service Plugin vollständig installiert und aktiviert?
- Sind alle Registry-Services erreichbar?

5. API-Authentifizierung/Session

- Eventuell ist deine Authentifizierung abgelaufen oder es fehlen Berechtigungen.

ToDo – Schrittweise Fehleranalyse

1. **API-Endpunkt prüfen:**

Logge dich ein und besuche [<https://xnat-dev.gwdg.de/xapi/swagger-ui.html>](https://xnat-dev.gwdg.de/xapi/swagger-ui.html)

Suche nach den Container-API-Routen, die du nutzen solltest.

2. **Vorhandene Commands und Wrapper anzeigen:**

Hole dir testweise die aktuelle Liste:

...

GET /xapi/commands

GET /xapi/wrappers

...

So siehst du, was tatsächlich existiert.

3. **Session und Projekt prüfen:**

Stelle sicher, dass die Session-ID zur ausgewählten Projekt-ID passt und du Rechte hast.

4. **Payload validieren:**

Prüfe, ob das JSON exakt dem in der Swagger-UI beschriebenen Schema entspricht.

5. **Logs auf XNAT-Server prüfen:**

Wenn Zugriff, können die XNAT-Server-Logs mehr Details zum Fehler enthalten.

Womit kann ich konkret helfen?

Bitte beantworte/prüfe:

- Welche genaue Request-URL verwendet dein Wrapper/Script für das Starten des Containers?

(Ersetze sensible Daten beim Teilen)

- Welches XNAT Container Service-Plugin und welche Version laufen auf dem Server?
- Hast du Zugriff auf die System-Logs für mehr Details?

Gerne kannst du den Abschnitt zeigen, wo in deinem Skript der Container "gestartet" wird (API-Aufruf). Dann kann ich noch gezielter Feedback zu deiner Implementation geben.

****Bitte berücksichtige aus Datenschutz- und Projektgründen, keine vertraulichen Zugangsdaten öffentlich zu posten.****

****Weiterführende Links:****

- [XNAT Container Service Quickstart](https://wiki.xnat.org/xnat-tools/container-service.html)
- [XNAT API Doku (Swagger)](https://xnat-dev.gwdg.de/xapi/swagger-ui.html)

Melde dich gerne mit weiteren Details! Dann kann ich tiefergehend auf deinen konkreten Fall eingehen.

Bousfiha, Tanae:

container-rest-api
Container Rest Api

GET

/container/name/{name}

Get Containers by database name

GET

/containers

Get all Containers

POST

/containers

Get paginated containers per request

GET

/containers/version

Check XNAT Version compatibility.

GET

/containers/{containerId}/logSince/{file}

Get Container logs

GET

/containers/{containerId}/logs

Get Container logs

GET

/containers/{containerId}/logs/{file}

Get Container logs

GET

/containers/{id}

Get Containers by database ID

DELETE

/containers/{id}

Get Container by container server ID

POST

/containers/{id}/finalize

Finalize Container

POST

/containers/{id}/kill

Kill Container

GET

/projects/{project}/containers

Get all Containers by project

GET

/projects/{project}/containers/name/{name}

Get Containers by name

POST

/projects/{project}/containers/{id}/kill

Kill Container

HAWKI (gpt-4.1):

Danke für diese Übersicht der Endpunkte der Container REST API auf deinem XNAT-Server. Das gibt wichtige Hinweise!

Analyse – Warum tritt der 404-Fehler auf?

Du gettest/spielst ein Wrapper/Command über den Endpunkt

****`/xapi/commands`****

und versuchst vermutlich danach, mit

****`/xapi/containers`****

deinen Container zu starten.

Dein Output und das von dir gezeigte API-Schema unterscheiden sich aber

****grundsätzlich****.

Der von dir gepostete API-Auszug beschreibt Endpunkte, die alle mit

`**`/containers`**` oder `**`/projects/{project}/containers`**` beginnen,
aber **KEINE** ``/xapi/commands`` oder ``/xapi/wrappers`` Einträge enthalten.

Das weist darauf hin:

- **Die Command/Wrapper-API ist nicht verfügbar oder anders implementiert!** Du verwendest vermutlich einen Code, der für das „XNAT Container Service Plugin“ geschrieben wurde, was von neueren „XNAT Container Dispatchers“ oder „XNAT Docker Service“-Versionen abweichen kann.
- **Die REST-API deines XNAT bietet direktes Management von Containern, aber (so wie gepostet) keinen Endpunkt zum „Starten eines Wrappers/Commands“** (wie z./B. ``/xapi/projects/{project}/wrappers/{wrapper}/launch``).

Fazit:

Du versuchst, einen Endpunkt zu verwenden, den dein XNAT-System NICHT unterstützt!

Daher kommt der 404-Fehler.

Was tun? Nächste Schritte:

1. Passende API-Endpunkte nutzen

Wenn du einen Container direkt starten willst, musst du laut Doku einen der folgenden POST-Endpunkte verwenden:

- **POST** ``/containers``
(Paginated/filtered Containerliste, vermutlich nicht zum Starten eines neuen Containers.)
- **POST** ``/projects/{project}/containers/{id}/kill``
(Beenden eines Containers.)

Aber: Es gibt KEINEN Start-Container-POST-Endpunkt in dieser Auflistung. Das ist ungewöhnlich. Üblicherweise gäbe es z./B.

`**`POST /projects/{project}/commands/{commandID}/wrappers/{wrapperID}/launch`**`

(Das ist aber meist nur bei XNAT „Container Service Plugin“ vorhanden.)

****!'** Prüfe in deiner API-UI, ob es einen expliziten Endpunkt z Containers gibt!******

Gehe im XNAT-Backend auf `/xapi/swagger-ui.html` oder `/swagger-ui.html` und sieh nach!

Es sollte einen Endpunkt wie „launch“/„run“ o./ä. geben.

2. XNAT-Version und Plugins prüfen

- Welches ***Container-Plugin*** ist installiert?

(XNAT-Admin oder Hosting/IT kann das prüfen: System-Adm

- Welche Version?

- Wird evtl. nur das „Legacy REST API“-Plugin genutzt, das Commands/Wrapper nicht unterstützt?

3. Dein Ziel genauer spezifizieren

- ****Willst du „Commands/Wrapper“ (als strukturierte Interface-Vorlagen) verwenden****

! Du brauchst das passende Plugin und musst ggf. mit Admin
- ****Oder willst du einfach ein Docker-Image im Projekt auf Daten loslassen?****

Dann solltest du mit den verfügbaren `/containers`-Endpunkten arbeiten.

Muster für direkten Container-Start (wenn vom Server unterstützt)

Prüfe, ob folgender Aufruf in deiner Swagger-UI existiert:

```http

POST /projects/{project}/commands/{command}/wrappers/{wrapper}/launch

```

Falls ****nein****: Du kannst keine Wrapper über API starten.

Alternativ musst du evtl. andere, projektspezifische Methoden oder Plugins nutzen.

Empfehlungen:

1. **Prüfe sorgfältig, welche Container-Plugins installiert sind.**

! Nur das „XNAT Container Service Plugin“ bietet die von d
Command/Wrapper-Workflow-API.

2. **Vergleiche in der Swagger-UI** (oder über `/swagger.json`) **die tatsächlich vorhandenen Endpunkte** mit deinem Skript.

3. **Passe dein Skript an die konkret vorhandene Container-API an**

– evtl. musst du deinen Container „manuell“ über das XNAT-Webinterface oder durch ein anderes API-Skript starten.

Hilfe für die Anpassung

Wenn du zeigst, **wie aktuell dein Python-Skript/Wrapper den Container starten möchte** (Post-URL, Payload),
kann ich dir helfen, diesen Teil an die tatsächlich unterstützten Endpunkte anzupassen!

Du kannst dazu z./B. die konkrete Fehlermeldung & relevante
(ohne Passwörter oder sensible Userdaten).

Zusammengefasst:

- Der 404 entsteht durch einen nicht (mehr) vorhandenen API-Endpunkt
(Commands/Wrapper-Start).

- Prüfe und passe die verwendeten Endpunkte an die installierte XNAT-Version und das Container-Plugin deiner Instanz an!

Melde dich mit Details, dann kann ich dich gezielter unterstütz
Python-Code für einen Container-Start, wenn der Endpunkt existiert.

Bousfiha, Tanae:

hier is die complete liste
command-configuration-rest-api
Command Configuration Rest Api

GET

/commands/{commandId}/wrappers/{wrapperName}/config
Get (site)

POST

/commands/{commandId}/wrappers/{wrapperName}/config
Configure (site)

DELETE

/commands/{commandId}/wrappers/{wrapperName}/config
Delete (site)

PUT

/commands/{commandId}/wrappers/{wrapperName}/disabled
Disable (site)

GET

/commands/{commandId}/wrappers/{wrapperName}/enabled
Is Enabled (site)

PUT

/commands/{commandId}/wrappers/{wrapperName}/enabled
Enable (site)

GET

/projects/{project}/commands/{commandId}/wrappers/{wrapperName}/config

Get (project)

POST

/projects/{project}/commands/{commandId}/wrappers/{wrapperName}/config
Configure (project)

DELETE

/projects/{project}/commands/{commandId}/wrappers/{wrapperName}/config
Delete (project)

PUT

/projects/{project}/commands/{commandId}/wrappers/{wrapperName}/disabled
Disable (project)

GET

/projects/{project}/commands/{commandId}/wrappers/{wrapperName}/enabled
Is Enabled (project)

PUT

/projects/{project}/commands/{commandId}/wrappers/{wrapperName}/enabled
Enable (project)

GET

/projects/{project}/wrappers/{wrapperId}/config
Get (project)

POST

/projects/{project}/wrappers/{wrapperId}/config
Configure (project)

DELETE

/projects/{project}/wrappers/{wrapperId}/config

Delete (project)

PUT

/projects/{project}/wrappers/{wrapperId}/disabled

Disable (project)

GET

/projects/{project}/wrappers/{wrapperId}/enabled

Is Enabled (project)

PUT

/projects/{project}/wrappers/{wrapperId}/enabled

Enable (project)

GET

/wrappers/{wrapperId}/config

Get (site)

POST

/wrappers/{wrapperId}/config

Configure (site)

DELETE

/wrappers/{wrapperId}/config

Delete (site)

PUT

/wrappers/{wrapperId}/disabled

Disable (site)

GET

/wrappers/{wrapperId}/enabled

Is Enabled (site)

PUT

/wrappers/{wrapperId}/enabled

Enable (site)

command-event-mapping-rest-api

Command Event Mapping Rest Api

GET

/commandeventmapping

Get all Command-Event Mappings

POST

/commandeventmapping

Create a Command-Event Mapping

GET

/commandeventmapping/{id}

Get a Command-Event-Mapping

DELETE

/commandeventmapping/{id}

Delete a Command-Event Mapping

POST

/commandeventmapping/{id}/convert

Convert a Command-Event Automation item to an Event Service Subscription

PUT

/commandeventmapping/{id}/disable

Disable a Command-Event Mapping

PUT

/commandeventmapping/{id}/enable

Enable a Command-Event Mapping

command-rest-api

Command Rest Api

GET

/commands

Get all Commands

POST

/commands

Create a Command

GET

/commands/available

Get Commands available in given project context and XSIType

GET

/commands/available/site

Get Commands sitewide with given XSIType

POST

/commands/{commandId}/wrappers/{wrapperId}

Update a Command Wrapper

GET

/commands/{id}

Get a Command by ID

POST

/commands/{id}

Update a Command

DELETE

/commands/{id}

Delete a Command

POST

/commands/{id}/wrappers

Create a Command Wrapper

GET

/projects/{project}/commands/available

Get Commands available in given project context and XSIType

DELETE

/wrappers/{wrapperId}

Delete a Command Wrapper

container-rest-api

Container Rest Api

GET

/container/name/{name}

Get Containers by database name

GET

/containers

Get all Containers

POST

/containers

Get paginated containers per request

GET

/containers/version

Check XNAT Version compatibility.

GET

/containers/{containerId}/logSince/{file}

Get Container logs

GET

/containers/{containerId}/logs

Get Container logs

GET

/containers/{containerId}/logs/{file}

Get Container logs

GET

/containers/{id}

Get Containers by database ID

DELETE

/containers/{id}

Get Container by container server ID

POST

/containers/{id}/finalize

Finalize Container

POST

/containers/{id}/kill

Kill Container

GET

/projects/{project}/containers

Get all Containers by project

GET

/projects/{project}/containers/name/{name}

Get Containers by name

POST

/projects/{project}/containers/{id}/kill

Kill Container

docker-rest-api

Docker Rest Api

GET

/docker/hubs

Get Docker Hubs

POST

/docker/hubs

Create new Docker Hub

GET

/docker/hubs/{id}

Get Docker Hub by ID

POST

/docker/hubs/{id}

Update Docker Hub by ID

DELETE

/docker/hubs/{id}

Delete Docker Hub by ID

GET

/docker/hubs/{id}/ping

Ping Docker Hub by ID

POST

/docker/hubs/{id}/pull

Pull image from Docker Hub by ID

GET

/docker/hubs/{name}

Get Docker Hub by Name

DELETE

/docker/hubs/{name}

Delete Docker Hub by Name

GET

/docker/hubs/{name}/ping

Ping Docker Hub by Name

POST

/docker/hubs/{name}/pull

Pull image from Docker Hub by Name

GET

/docker/image-summaries

Get summary list of images and commands.

GET

/docker/images

Get list of images.

POST

/docker/images/save

Save Commands from labels

GET

/docker/images/{id}

Get Docker image

DELETE

/docker/images/{id}

Delete Docker image

POST

/docker/pull

Pull image from default Docker Hub

GET

/docker/server

Docker server

POST

/docker/server

Set Docker server configuration

GET

/docker/server/ping

Ping docker server.

HAWKI (gpt-4.1):

Danke für die vollständige Endpoint-Übersicht deiner XNAT-Container- und Command-REST-API.

****Damit ist die Ursache deines 404-Fehlers und die korrekte Vorgehensweise jetzt eindeutig klärbar.****

Zentrale Beobachtung

****Es gibt keinen API-Endpunkt zum STARTEN eines Wrappers oder eines Commands.****

- Die von deinem Skript verwendeten Routen wie `/xapi/commands`, `/xapi/wrappers` oder `/commands/{id}/wrappers/{wrapperID}/launch` tauchen hier nicht auf und sind auf deinem System UNTERSTÜTZT NICHT.

****Was ist verfügbar?****

- Du kannst Commands (und Wrapper) anlegen und konfigurieren (siehe `POST /commands`, `POST /commands/{id}/wrappers`) und sie in Events einhängen (Automation),
- Aber den eigentlichen "Launch" eines Command Wrappers / einer Command Configuration gibt es als API-Aufruf nicht.
- Die Endpunkte unter `/containers` betreffen nur bereits laufende oder beendete Container – nicht das Starten neuer Container-Tasks basierend auf einem Command/Wrapper.

Was bedeutet das für dich?

Du ****kannst mit der aktuell verfügbaren API****:

- Container-Metadaten abfragen,
 - existierende Wrapper und Commands verwalten,
 - Container stoppen/finalisieren,
 - Docker-Hub-Interaktionen durchführen,
- aber du ****kannst keinen "Container-Task" (Command/Wrapper-Ausführung) per API starten****.

Warum ist das so?

- Bei vielen (älteren oder systemnahen) XNAT-Setups werden Commands/Wrapper-Tasks noch immer ausschließlich
 - ****über das XNAT-Webinterface**** gestartet (grafisch über "Aktionen" im Session-Interface),
 - ****oder automatisch durch Event-Mappings**** (z.B. beim Upload neuer Sessions, Trigger durch "Command Event Mapping").
- Ein expliziter ****API-Call zum Starten ("launch wrapper/command")**** ist nur vorhanden, wenn das ****XNAT Container Service Plugin**** in bestimmter Version installiert ist.
- Auf deiner Instanz ist offensichtlich (noch) kein solcher "Launch"-Endpoint eingerichtet oder aktiviert.

Wie kannst du dennoch einen Container auf Daten laufen lassen?

Es gibt zwei Wege:

1. Über das XNAT-Webinterface

- Im Experiment/Session-Menü findest du unter "Aktionen" ! "ausführen" (oder ähnlich).
- Dort kannst du den Command/Wrapper per Hand starten und die Eingabeparameter angeben.

2. Automatischer/Workflow-basierter Start (Event Mapping)

- Du kannst über den Endpunkt:

...

POST /commandeventmapping

...

ein Command-Event-Mapping anlegen. Sobald ein Event (z.B. eine Session erstellt oder bearbeitet) eintritt,

wird dein Wrapper/Command ****automatisch**** auf die neuen Daten angewendet und der Container ausgeführt.

Weitere Informationen:

- [XNAT Command Event Mappings](<https://wiki.xnat.org/container-service/container-event-mapping>)
- [XNAT Command- und Container-Workflow-Übersicht](<https://wiki.xnat.org/container-service>)

Zum API-basierten "Triggern über ein Event":

Du kannst ein Command Event Mapping programmatisch anlegen!

Dafür brauchst du das richtige Event (etwa "neue MR-Session im Projekt"), verknüpfst es mit deinem Command/Wrapper, und XNAT erledigt den Rest automatisch im Hintergrund.

Was ist nicht möglich?

- Du ****kannst mit dieser API keinen Command-/Wrapper-Job direkt per HTTP-Request starten**** (kein klassisches "launch").
- Dein Skript müsste an diese Realität angepasst oder dein XNAT/CS-Plugin

upgedatet werden.

Empfehlung: So kannst du weiter verfahren

1. ****Kläre, was du möchtest:****

- Möchtest du das Starten weiterhin automatisieren, z.B. nach dem Upload?
 - !' Nutze Event-Mapping.
- Möchtest du manuell Container-Tasks triggern?
 - !' Über das Webinterface, ggf. mit GUI-Automatisierung.

2. ****Skript anpassen:****

- Entferne API-Aufrufe, die einen "Start"-Endpunkt verwenden.
- Implementiere bei Bedarf das Anlegen von Event-Mappings.

3. ****Langfristig:****

- Prüfe mit deinem XNAT-Admin, ob das "XNAT Container Service Plugin" in aktueller Version installiert werden kann, damit ein `launch`-Endpoint existiert.

Zusammengefasste Antwort

****Dein 404 entsteht, weil der von dir angesprochene Start-Endpoint ("launch container/command/wrapper") in deiner XNAT-Installation nicht existiert. Du kannst daher nur über Webinterface oder Event Mapping Container ausführen, aber nicht per API-Call.****

Wenn du möchtest, kann ich dir ein Beispiel geben, wie du ein Command Event Mapping via API anlegst oder wie das manuelle Starten per UI funktioniert.

****Melde dich gerne mit deinem bevorzugten Workflow – dann bekommst du passgenaue Hinweise/Codebeispiele dafür.****

Bousfiha, Tanae:

i ahve found

launch-rest-api
Launch Rest Api

GET

/commands/{commandId}/wrappers/{wrapperName}/bulklaunch

Get Bulk Launch UI for wrapper

POST

/commands/{commandId}/wrappers/{wrapperName}/bulklaunch

Resolve a command from the variable values in the request body, and launch it

GET

/commands/{commandId}/wrappers/{wrapperName}/launch

Get Launch UI for wrapper

POST

/commands/{commandId}/wrappers/{wrapperName}/launch

Resolve a command from the variable values in the query params, and launch it

POST

/commands/{commandId}/wrappers/{wrapperName}/root/{rootElement}/bulklaunch

Resolve a command from the variable values in the request body, and launch it

POST

/commands/{commandId}/wrappers/{wrapperName}/root/{rootElement}/launch

Resolve a command from the variable values in the query params, and launch it

GET

/projects/{project}/commands/{commandId}/wrappers/{wrapperName}/bulklaunch

Get Bulk Launch UI for wrapper

POST

/projects/{project}/commands/{commandId}/wrappers/{wrapperName}/bulklaunch

Resolve a command from the variable values in the request body, and launch it

GET

/projects/{project}/commands/{commandId}/wrappers/{wrapperName}/launch

Get Launch UI for wrapper

POST

/projects/{project}/commands/{commandId}/wrappers/{wrapperName}/launch

Resolve a command from the variable values in the request body, and launch it

POST

/projects/{project}/commands/{commandId}/wrappers/{wrapperName}/root/{rootElement}/bulklaunch

Resolve a command from the variable values in the request body, and launch it

POST

/projects/{project}/commands/{commandId}/wrappers/{wrapperName}/root/{rootElement}/launch

Resolve a command from the variable values in the query params, and launch it
POST

/projects/{project}/wrapper/{wrapperId}/launch

Resolve a command from the variable values in the query params, and launch it
POST

/projects/{project}/wrapper/{wrapperId}/root/{rootElement}/launch

Resolve a command from the variable values in the query params, and launch it
GET

/projects/{project}/wrappers/{wrapperId}/bulklaunch

Get Bulk Launch UI for wrapper

POST

/projects/{project}/wrappers/{wrapperId}/bulklaunch

Resolve a command from the variable values in the request body, and launch it
GET

/projects/{project}/wrappers/{wrapperId}/launch

Get Launch UI for wrapper

POST

/projects/{project}/wrappers/{wrapperId}/launch

Resolve a command from the variable values in the request body, and launch it
POST

/projects/{project}/wrappers/{wrapperId}/root/{rootElement}/bulklaunch

Resolve a command from the variable values in the request body, and launch it
POST

/projects/{project}/wrappers/{wrapperId}/root/{rootElement}/launch

Resolve a command from the variable values in the request body, and launch it
GET

/wrappers/{wrapperId}/bulklaunch

Get Bulk Launch UI for wrapper

POST

/wrappers/{wrapperId}/bulklaunch

Resolve a command from the variable values in the request body, and launch it
GET

/wrappers/{wrapperId}/launch

Get Launch UI for wrapper

POST

/wrappers/{wrapperId}/launch

Resolve a command from the variable values in the query params, and launch it
POST

/wrappers/{wrapperId}/root/{rootElement}/bulklaunch

Resolve a command from the variable values in the request body, and launch it
POST

/wrappers/{wrapperId}/root/{rootElement}/launch

Resolve a command from the variable values in the query params, and launch it

HAWKI (gpt-4.1):

Sehr gut, dieser **launch-rest-api**-Abschnitt ist der entscheidende Teil für dein Anliegen!

Du hast jetzt genau die Endpoints gefunden, mit denen du einen Command/Wrapper in XNAT **direkt per API starten** kannst.

Was bedeutet das für dich?

! **Du kannst deinen Container/Command per POST an einen ausführen!**

Jetzt kommt es nur noch darauf an, dass du:

- die richtige URL verwendest
- die richtige Payload sendest

Beispiel: Direktes Starten eines Wrappers mit Command im Projekt

POST

``plaintext

/projects/{project}/commands/{commandId}/wrappers/{wrapperName}/launch

``

- `{project}` = Projekt-ID (z.B. `tanae_osa_Predct`)

- `{commandId}` = deine Command-ID (z.B. `80`)

- `{wrapperName}` = Wrapper-Name (z.B. `fillerinjektion_wrapper`)

Body:

Als Content-Type wird meistens `application/json` verwendet.

****Minimalbeispiel für den Payload**** (siehe [XNAT-Doku](https://wiki.xnat.org/container-service/api-guide) - diese Felder können je nach Wrapper abweichen):

```
```json
{
 "inputs": {
 "session": "XNAT_E00428"
 }
}
```
```

Je nach Command kann es sein, dass weitere/nur andere Inputs Pflicht sind!

Beispielaufruf (Python, requests)

```
```python
import requests

url = "https://xnat-dev.gwdg.de/xapi/projects/tanae_osa_Predct/commands/80/wrappers/fillerinjektion_wrapper/launch"
headers = {
 "Authorization": "Bearer <DEIN_TOKEN_ODER_BASIC_AUTH>",
 "Content-Type": "application/json",
}
data = {
 "inputs": {
 "session": "XNAT_E00428"
 }
}

resp = requests.post(url, json=data, headers=headers, verify=True)
print(resp.status_code)
print(resp.text)
```
```

****Hinweis:****

- Wenn du Basic Auth nutzt, verwende stattdessen `auth=(user, pass)` im requests-Aufruf.
- `` ersetzen!

Typische Fehlerquellen und Hinweise

- ****404:**** !' URL falsch, oder ID/Name stimmt nicht exakt.
- ****400:**** !' Payload oder Parameter nicht korrekt.
- ****403/401:**** !' Rechte/Authentifizierung fehlen.
- ****Command-/Wrapper-Name oder ID:****
Nutze exakt den Namen/ID wie über `/commands`-GET und `/wrappers`-GET ermittelt (ohne Tippfehler oder Groß/Kleinschreibung zu ändern).
- ****Pflichtfelder in inputs:****
Jede Command-Definition legt andere Inputschlüssel fest – im Zweifel einmal die "Launch UI" im Webinterface öffnen und schauen, welche Felder gebraucht werden.

Zusammenfassung für dich

- Mit

...

POST

/projects/tanae_osa_Predct/commands/80/wrappers/fillerinjektion_wrapper/launch
...

und einem passenden JSON-Body kannst du deinen Workflow programmgesteuert starten.

- Die `"inputs"`-Schlüssel müssen zum Command/Wrapper passen!
- Bei Fehlern gibt die Fehlermeldung meist Details zu fehlenden/unerwarteten Parametern.

****Wenn du möchtest, poste gerne dein aktuelles Python-Skript oder die genaue**

Fehlermeldung samt Request-URL und Body (ohne Userdaten), um deine Anfrage gemeinsam im Detail debuggen zu können!**

So kannst du beliebig viele Sessions/Experimente automatisiert durch deinen Docker/Wrapper schicken.

****Literatur****

- [XNAT Container Service API Guide (offizielle Doku)](<https://wiki.xnat.org/container-service/api-guide>)

- [XNAT-Benutzerforum](https://groups.google.com/forum/#!forum/xnat_discussion)

Melde dich, wenn du Hilfe für die genaue "inputs"-Konfiguration brauchst!

Bousfiha, Tanae:

```
def run_container_in_xnat(xnat_host, command_id, wrapper_id, session_id,
xnat_user, xnat_password):
    url = f"{xnat_host}/xapi/container-launcher/launch"
    headers = {'Content-Type': 'application/json'}
    payload = {
        'command': command_id,
        'wrapper': wrapper_id,
        'inputs': {
            'session': session_id
        }
    }
    print("Launching container with payload:", json.dumps(payload,
indent=2))
    response = requests.post(url, auth=(xnat_user, xnat_password), headers=headers,
json=payload, verify=False)
    if response.status_code in [200, 201]:
        print("Container launched successfully.")
    else:
        print(f"Failed to launch container: {response.status_code} -
{response.text}")
```

hier ist meine Abschnitt

HAWKI (gpt-4.1):

Danke, dass du deinen Code gepostet hast! Dein Ansatz ist gut, aber die von dir verwendete URL und die Struktur des Payloads entsprechen nicht den Anforderungen der gefundenen Launch-API deiner XNAT-Instanz.

****Was ist im Code falsch?****

- ****URL****:

Du verwendest

```
```python
f"{xnat_host}/xapi/container-launcher/launch"
```
```

aber laut deiner API gibt es keine `/xapi/container-launcher/launch`-Route.

Korrekt wäre z.B.:

...

```
{xnat_host}/xapi/projects/{project}/commands/{commandId}/wrappers/{wrapperName}
}/launch
```

...

oder, wenn du Wrapper-ID statt Name verwenden willst:

...

```
{xnat_host}/xapi/projects/{project}/wrappers/{wrapperId}/launch
```

...

- ****Payload****:

Die Launch-API verwendet (gemäß API-Dokumentation) meist einen Payload nach folgendem Format:

```
```json
```

```
{
```

```
 "inputs": {
```

```
 ... # Schlüssel/Werte für die Eingabeparameter des Wrappers/Commands, z.B.
```

```
"session": "XNAT_E00428"
```

```
 }
```

```
}
```

```
```
```

Du brauchst nicht `"command"` und `"wrapper"` als separate Felder im Body, da diese aus der URL abgeleitet werden!

KORREKTE VERSION

1. URL anpassen

Nehmen wir an:

```
- `xnat_host = "https://xnat-dev.gwdg.de/xapi"`  
- `project = "tanae_osa_Predct"`  
- `command_id = 80`  
- `wrapper_name = "fillerinjektion_wrapper"`
```

Dann:

```
```python  
url = f"{xnat_host}/projects/{project}/commands/{command_id}/wrappers/{wrapper_name}/launch"
```
```

2. Payload/Body anpassen

```
```python  
payload = {
 "inputs": {
 "session": session_id
 # eventuell weitere erforderliche Felder
 }
}
```
```

3. Beispiel: Korrigierte Funktion

```
```python  
import requests
import json

def run_container_in_xnat(
 xnat_host, project, command_id, wrapper_name, session_id,
 xnat_user, xnat_password
):
 url = f"{xnat_host}/projects/{project}/commands/{command_id}/wrappers/{wrapper_name}/launch"
```

```

headers = {"Content-Type": "application/json"}
payload = {
 "inputs": {
 "session": session_id
 # ggf. weitere Schlüssel je nach Wrapper
 }
}
print("Launching container with payload:", json.dumps(payload, indent=2))
response = requests.post(
 url,
 auth=(xnat_user, xnat_password),
 headers=headers,
 json=payload,
 verify=False
)
if response.status_code in [200, 201]:
 print("Container launched successfully.")
 print(response.json())
else:
 print(f"Failed to launch container: {response.status_code} - {response.text}")

```

# Beispielaufruf:

```

run_container_in_xnat(
"https://xnat-dev.gwdg.de/xapi",
"tanae_osa_Predct",
80,
"fillerinjektion_wrapper",
"XNAT_E00428",
"USERNAME",
"PASSWORD"
)
...

```

---

### \*\*Tipp:\*\*

Wenn Fehler wie 400 (Bad Request) kommen, prüfe:

- Stimmen die "inputs"-Keys mit dem im Wrapper/Command geforderten Parametern

exakt überein? (Im Zweifel im XNAT-Webinterface am gleichen Wrapper testen und Felder vergleichen!)

- Sind Projektnamen, Wrappernamen und Command-IDs exakt?
- Verwendest du `/xapi/` oder `/data/` im Pfad? Bleibe konsistent!

---

Wenn du nicht sicher bist, wie die Inputs heißen sollen:

Öffne die "Launch UI" im XNAT-Webinterface für diesen Wrapper, gib beliebige Beispielwerte ein und schaue dir die Netzwerk-Requests im Browser-Entwicklertool an – so findest du das geforderte JSON.

---

**\*\*Melde dich gerne, falls du weitere Anpassung oder Hinweise zur Fehlersuche brauchst!\*\***