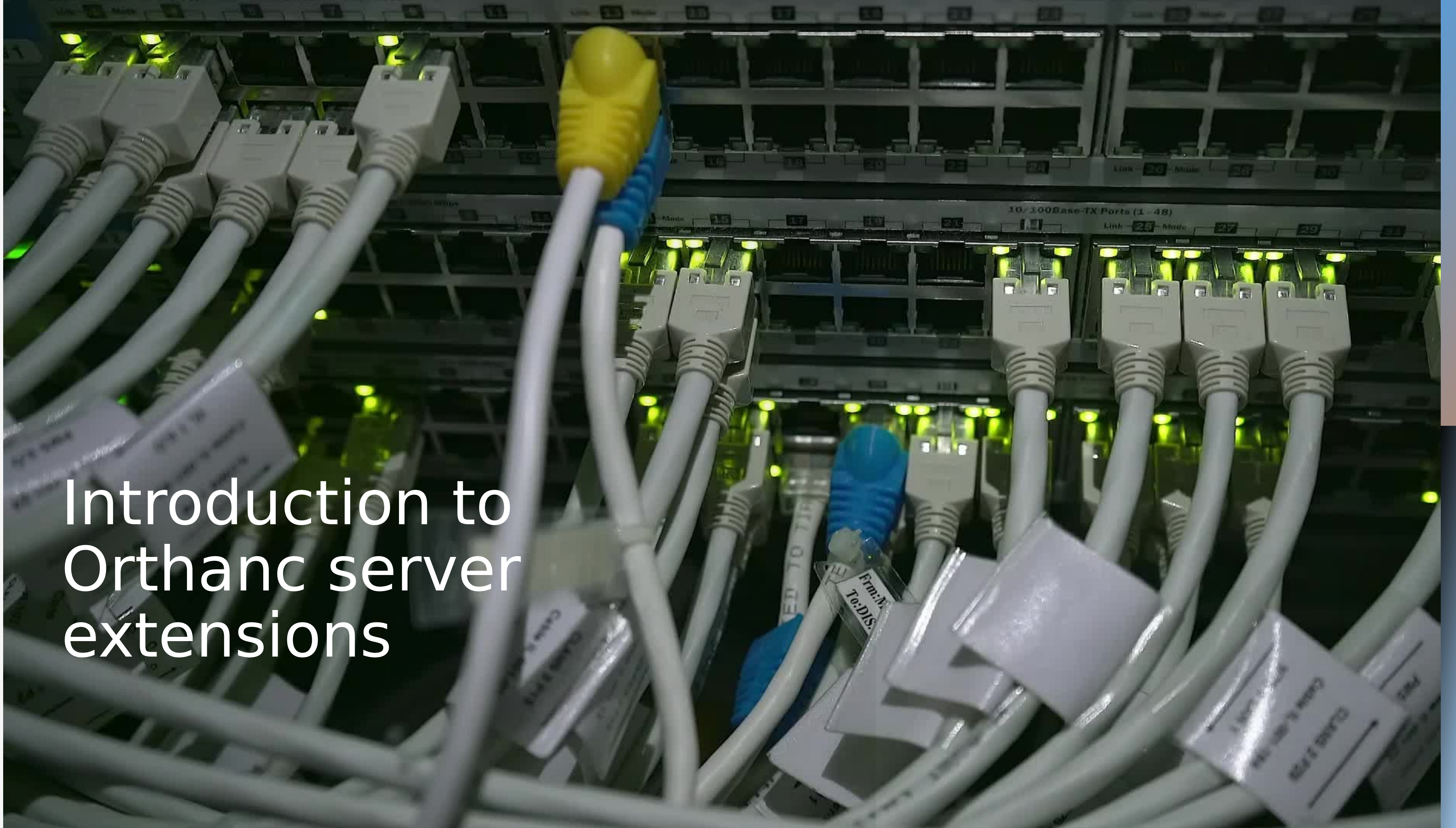


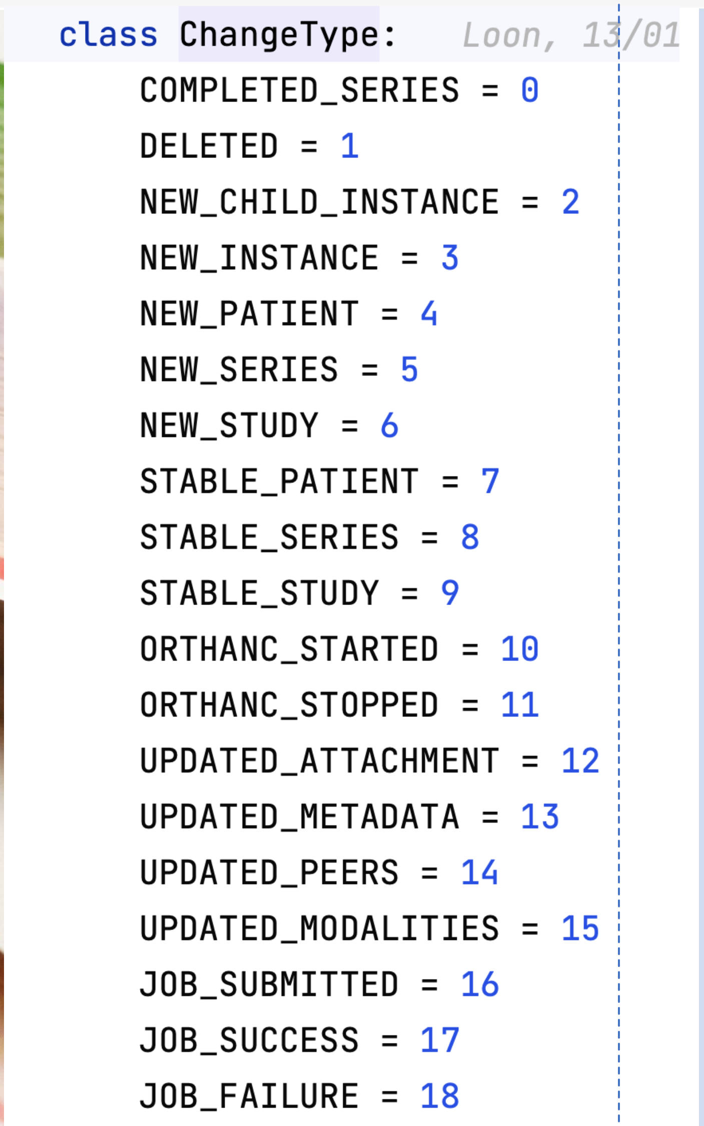
Introduction to Orthanc server extensions

Orthanc Python Plugin Basics

Handle Orthanc (new DICOM, C-STORE request) **events** with Python callback functions

```
def OnChange(changeType, level, resource):
    if changeType == orthanc.ChangeType.ORTHANC_STARTED:
        ...

orthanc.RegisterOnChangeCallback(OnChange)
```

Great for routing, filtering, tag morphing



```
class ChangeType:    Loon, 13/01
    COMPLETED_SERIES = 0
    DELETED = 1
    NEW_CHILD_INSTANCE = 2
    NEW_INSTANCE = 3
    NEW_PATIENT = 4
    NEW_SERIES = 5
    NEW_STUDY = 6
    STABLE_PATIENT = 7
    STABLE_SERIES = 8
    STABLE_STUDY = 9
    ORTHANC_STARTED = 10
    ORTHANC_STOPPED = 11
    UPDATED_ATTACHMENT = 12
    UPDATED_METADATA = 13
    UPDATED_PEERS = 14
    UPDATED_MODALITIES = 15
    JOB_SUBMITTED = 16
    JOB_SUCCESS = 17
    JOB_FAILURE = 18
```

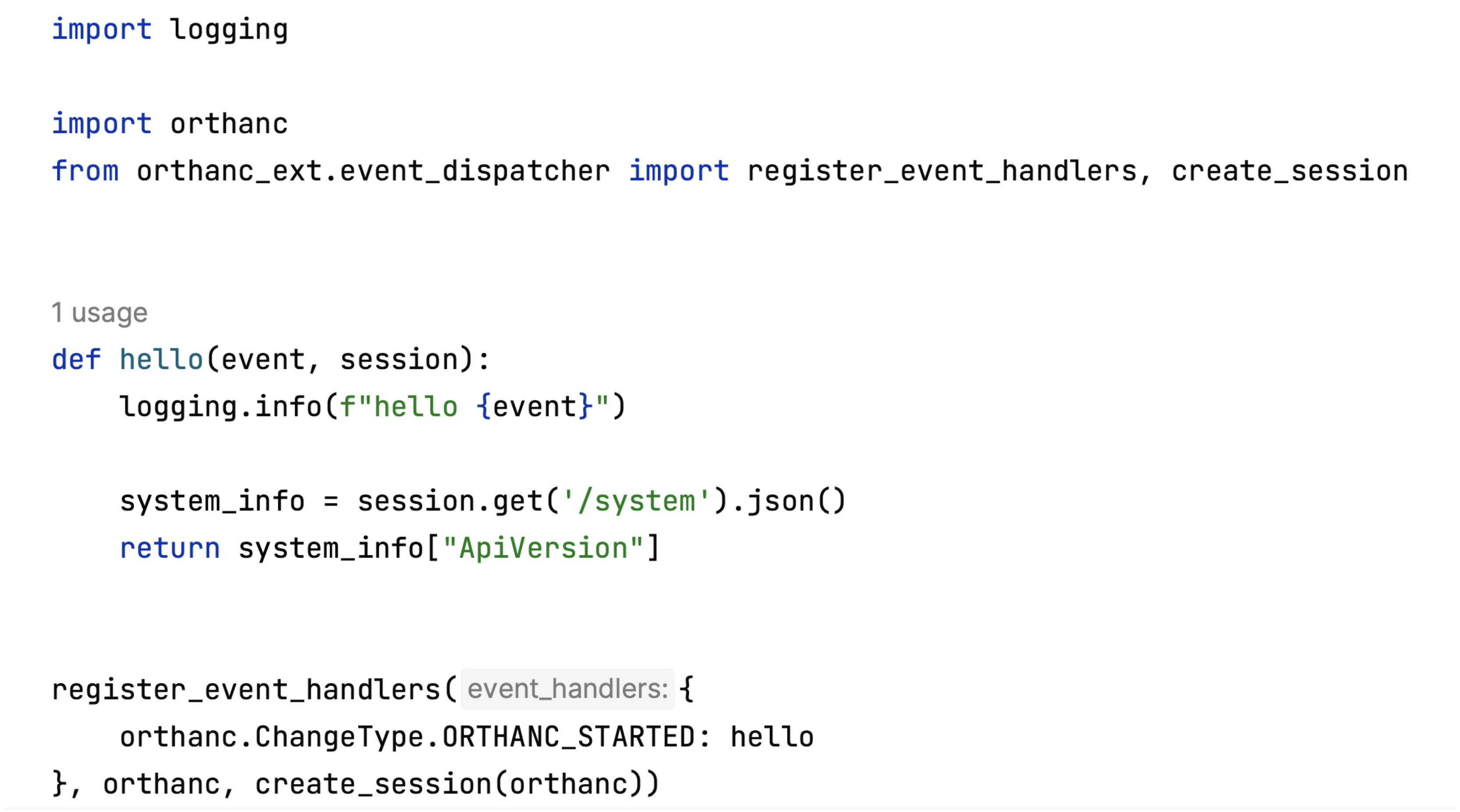
Orthanc Python Plugin
Drawbacks

- The `orthanc` Python module is only available at Orthanc runtime
- Can become a bit tedious to develop
    - Make a change
    - Restart Orthanc
    - Trigger the event
    - Check the logs
- Not so easy to write unit tests, need to mock the `orthanc` module everywhere

Orthanc-server-extensions
Design goals

- Faster development loop
- Support for (unit) testing
- Includes recipes for common production requirements
  - HTTP client with TLS and Authentication support
  - Logging
  - De/re-identification
  - Job retries

Hello world

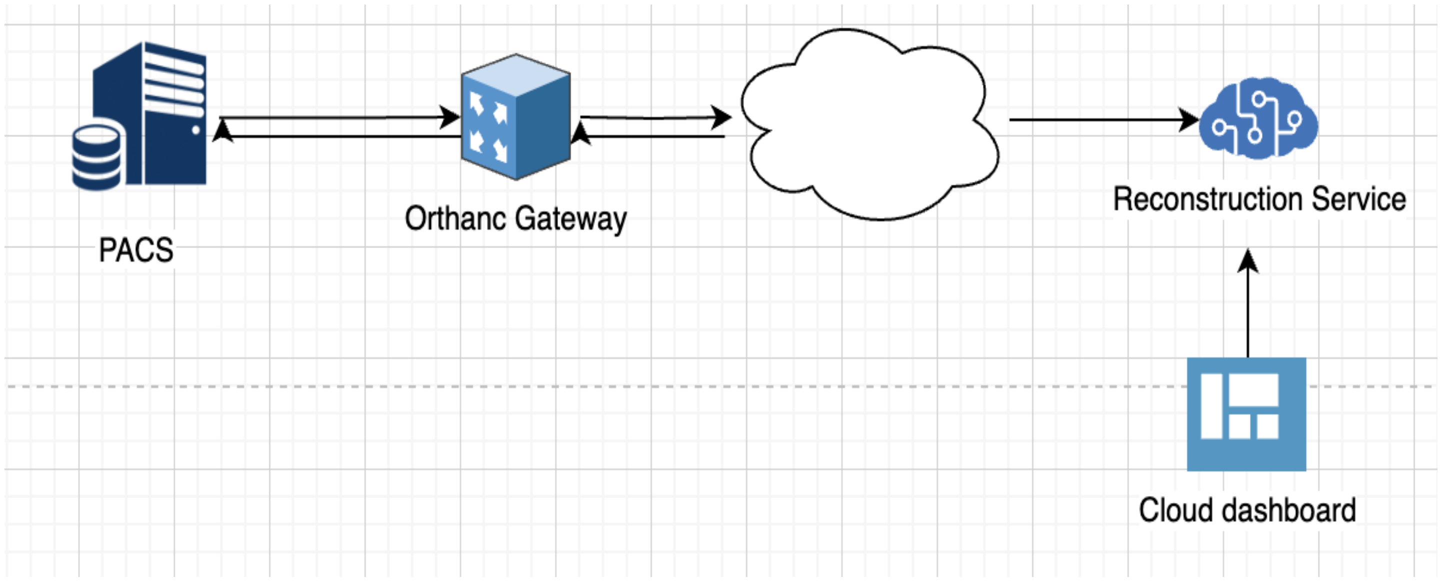


```
import logging

import orthanc
from orthanc_ext.event_dispatcher import register_event_handlers, create_session


1 usage
def hello(event, session):
    logging.info(f"hello {event}")

    system_info = session.get('/system').json()
    return system_info["ApiVersion"]


register_event_handlers( event_handlers: {
    orthanc.ChangeType.ORTHANC_STARTED: hello
}, orthanc, create_session(orthanc))
```

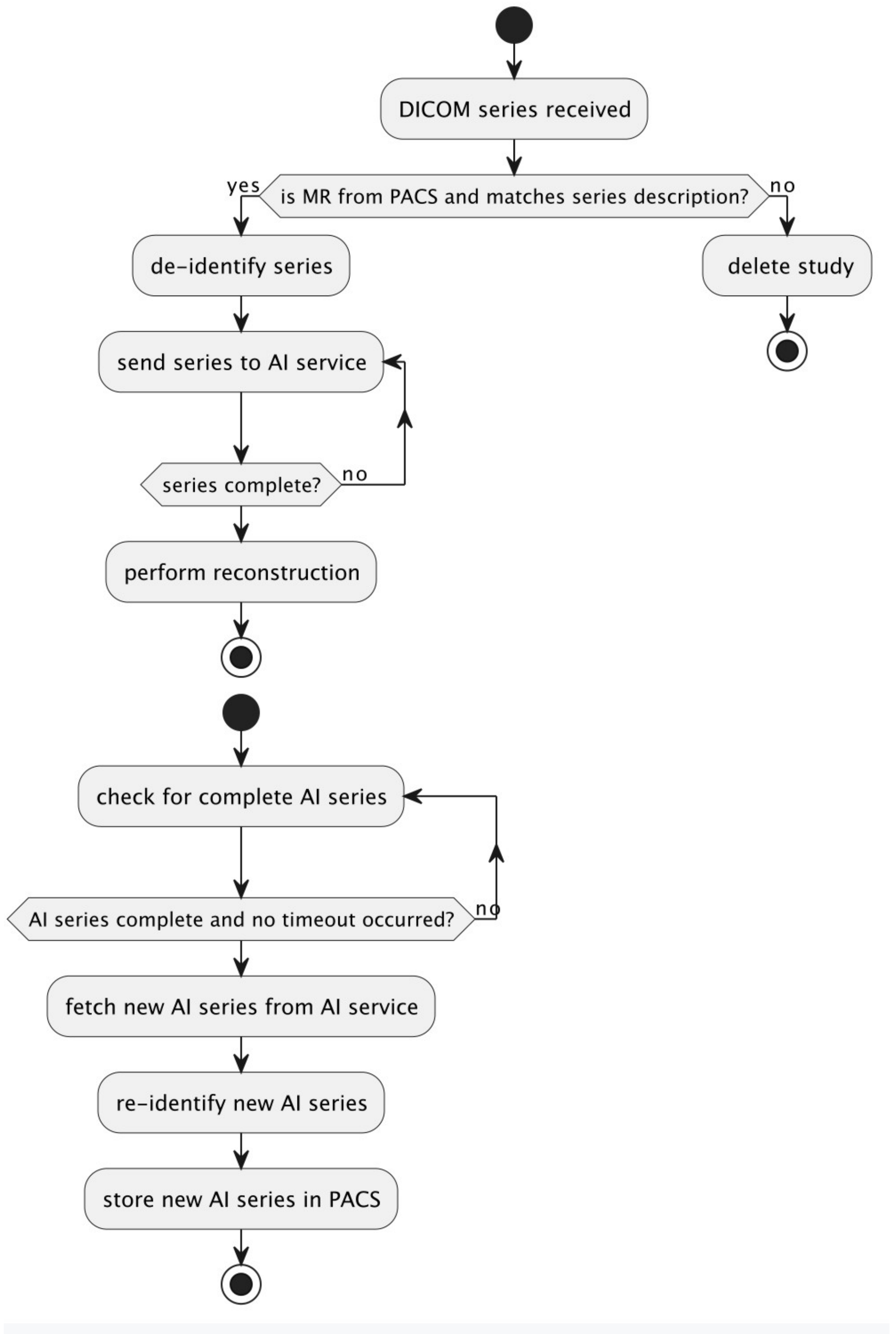
Hello world: unit test

```
@respx.mock
def test_hello_world():
    orthanc = OrthancApiHandler()
    client = create_internal_client('https://localhost:8043')

    respx.get('https://localhost:8043/system').respond(
        status_code: 200, json={
            "ApiVersion": 18,
            "DicomAet": "ORTHANC",
            "DicomPort": 4242,
            "HttpPort": 8042,
            "IsHttpServerSecure": True})

    event_dispatcher.register_event_handlers(
        event_handlers: {orthanc.ChangeType.ORTHANC_STARTED: hello}, orthanc, client)
    sync_result = orthanc.on_change(
        orthanc.ChangeType.ORTHANC_STARTED, orthanc.ResourceType.NONE, resource_id: None)
    assert sync_result == [18]
```

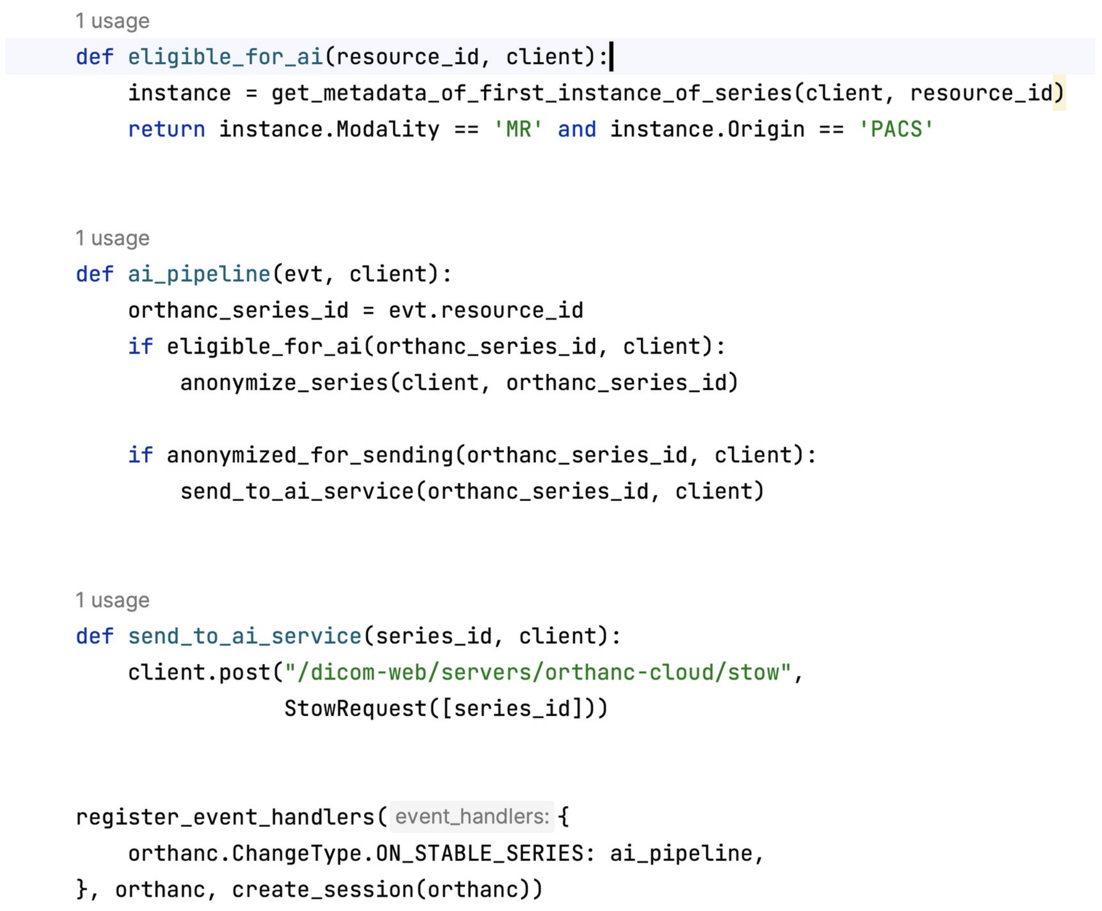
Common AI deployment



Common DICOM AI workflow scenario

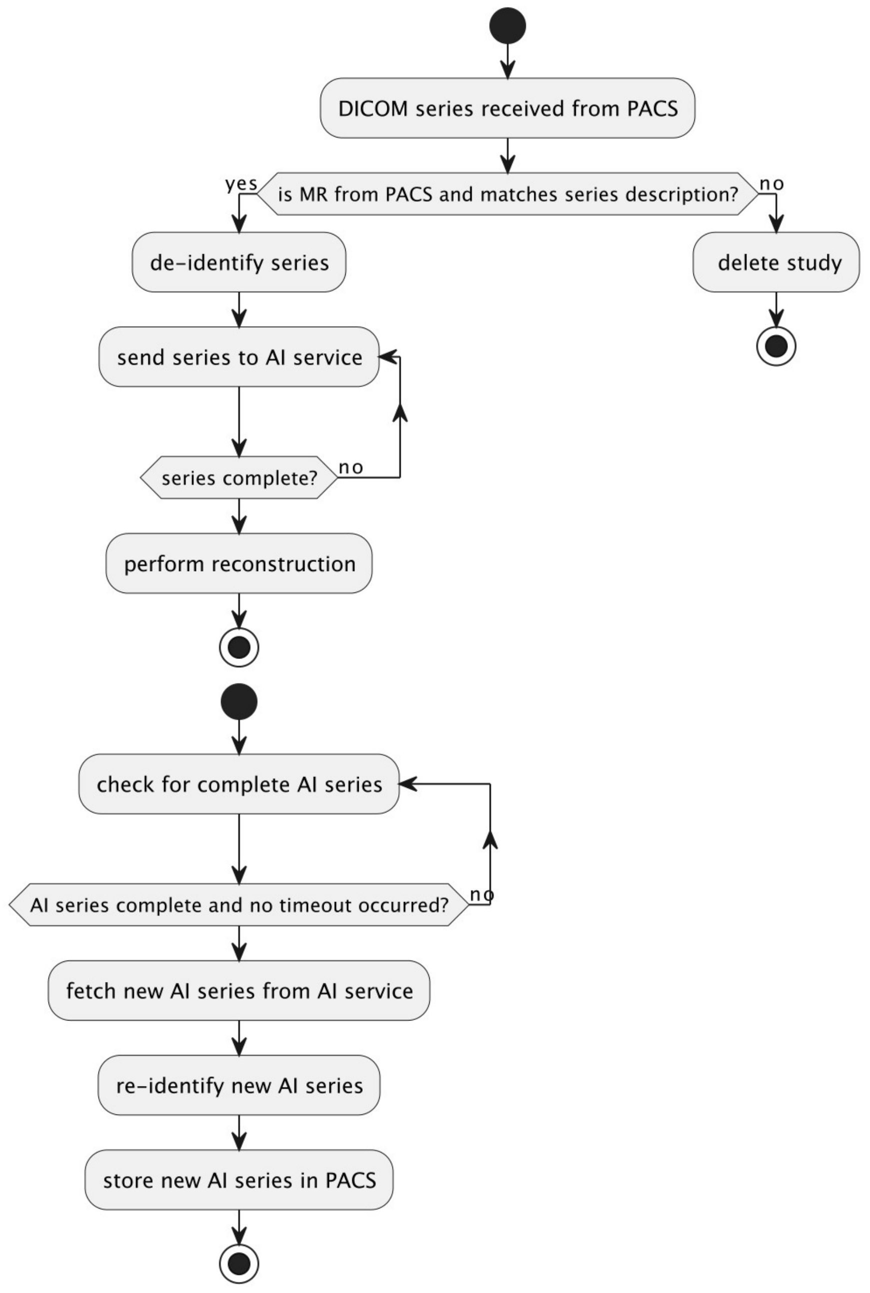

DICOM series received
is MR from PACS and matches series description?
yes
no
de-identify series
delete study
send series to AI service
series complete?
no
perform reconstruction
check for complete AI series
AI series complete and no timeout occurred?
no
fetch new AI series from AI service
re-identify new AI series
store new AI series in PACS

AI workflow



```
1 usage
def eligible_for_ai(resource_id, client):
    instance = get_metadata_of_first_instance_of_series(client, resource_id)
    return instance.Modality == 'MR' and instance.Origin == 'PACS'


1 usage
def ai_pipeline(evt, client):
    orthanc_series_id = evt.resource_id
    if eligible_for_ai(orthanc_series_id, client):
        anonymize_series(client, orthanc_series_id)

    if anonymized_for_sending(orthanc_series_id, client):
        send_to_ai_service(orthanc_series_id, client)


1 usage
def send_to_ai_service(series_id, client):
    client.post("/dicom-web/servers/orthanc-cloud/stow",
                StowRequest([series_id]))


register_event_handlers( event_handlers: {
    orthanc.ChangeType.ON_STABLE_SERIES: ai_pipeline,
}, orthanc, create_session(orthanc))
```

Common DICOM AI workflow scenario (refined)


DICOM series received from PACS
is MR from PACS and matches series description?
yes
no
de-identify series
delete study
send series to AI service
series complete?
no
perform reconstruction
check for complete AI series
AI series complete and no timeout occurred?
no
fetch new AI series from AI service
re-identify new AI series
store new AI series in PACS

Latest release

- Supports cloud workflow
- Process async functions with `asyncio`
- publish CloudEvents directly to NATS, RabbitMQ and Kafka
    - Events can be handled by external software
    - Logic can be implemented independently of orthanc
    - Begets all the advantages of event platforms
        - dynamically scale deployments
        - Implement retries

Future directions

- Higher level events to simplify workflows
- Performance evaluation of the asyncio implementation

References

Python plugin

- https://github.com/walkIT-nl/orthanc-server-extensions/
- https://orthanc-server-extensions.readthedocs.io/en/latest/

Python plugin

- https://book.orthanc-server.com/plugins/python.html