Guide for Installing RACOON JIP on Ubuntu

1. Install NVIDIA GPU driver:
    a. `sudo lshw -C display`
    Use this command to check if there exists a GPU in your system. It generates a list where your GPU should be included
    b. `sudo apt update`
    c. `sudo apt upgrade -y`
    d. `sudo apt install nvidia-driver-450-server -y`
    (Watch out that you install the correct version 450)
    e. `sudo systemctl mask sleep.target suspend.target hibernate.target hybrid-sleep.target`
    f. Test the NVIDIA driver with the command: `nvidia-smi`
    You should now see a table that includes your gpu. If the command is not working, a reboot may be necessary.
2. Prepare Platform Installation
    a. Get Kaapana: `git clone https://github.com/kaapana/kaapana.git`
    b. Checkout the RACOON branch: `git checkout feature/racoon-platform`
    c. Navigate to the dir: `kaapana/platforms/racoon-platform/platform-installation`
    d. Edit the file `install_racoon.sh`
    e. Adapt the following variables:
        - `FAST_DATA_DIR` and `SLOW_DATA_DIR`
        just choose two arbitrary directories. E.g. I have used for testing:
        `/home/user/simon/racoon/fast` and `/home/user/simon/racoon/slow`
        - Set `DEV_MODE` to `true`
        - Set `GPU_SUPPORT` to `true`
    f. Make both installation files executable:
        - `chmod +x server_installation.sh`
        - `chmod +x install_racoon.sh`
3. Start the server installation:
    a. `sudo ./server_installation.sh`
    b. Follow the instructions and confirm that there is „no proxy“ with `yes`
    c. Activate GPU support: `sudo ./server_installation.sh -gpu`
    d. Reboot your machine
4. Install the RACOON JIP platform
    a. Run without sudo: `./install_racoon.sh`
    b. Enter Username: *kaapana*
    c. Enter Password: *EjsH53fXznKMtVFfwXxs*
    d. GPU: `yes`
    e. Server Domain: Either a correct server domain, or for local testing: `localhost`
    f. Wait until the installation finishes
    g. `watch microk8s.kubectl get pods --all-namespaces`
    Wait until all pods have status `running` or `completed`
5. Access the platform:
    a. Navigate inside your browser to https://localhost This will open the Kaapana startpage
    b. Use the following login credentials:
        - Username: *kaapana*
        - Passwort: *kaapana*
6. Install our TUDA workflows to the platform:
    a. Copy all contents from `workflows/airflow-components` to `FAST_DATA_DIR/workflows/`
    `FAST_DATA_DIR` is the directory you have chosen in step 2e

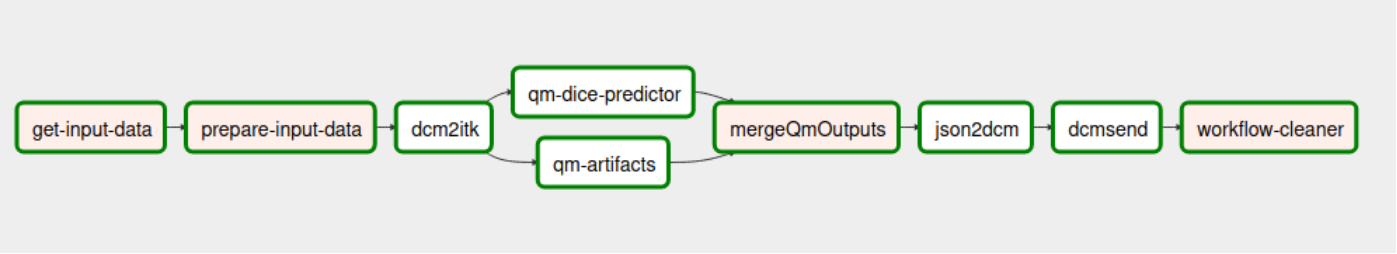
b. The following operators rely on a pre-built docker container. Currently they are uploaded to Dockerhub. In the future this may change when the docker container have to be provided inside the RACOON nodes.
The processing container that belongs to an operator is denoted in brackets behind the filename:
- *Dcm2ItkOperator.py* (/processing-containers/dcm2itk_converter)
- *DcmSr2JsonOperator.py* (/processing-containers/jsonDcmSr_tools)
- *Json2DcmSrOperator.py* (/processing-containers/jsonDcmSr_tools)
- *QmArtifactsOperator.py* (/processing-containers/qm_artifacts)
- *QmDicePredictorOperator.py* (/processing-containers/qm_dicePredictor)

For testing, you do not have to make any changes and the Operators will automatically pull the pre-built containers from Dockerhub. For LIVE deployment it is necessary to change the parameter image inside these operators to the location where the containers were pushed to on the server.

7. Send test-data to the platform:
   a. Install DCMTK: apt get install dcmtk
   b. Send DICOM files to the platform:
   dcmsend -v <ip-address of server> 11113 --scan-directories --call <aetitle of images, used for filtering> --scan-pattern '*' --recurse <data-dir-of-DICOM images>
      - <ip-address of server> is the server address that was also used during the installation in step 4e. For the local host this should be localhost
      - <aetitle of images, used for filtering> is a name tag that can later be used to search for data
      - <data-dir-of-DICOM images> is either a DICOM file or a directory with multiple DICOM files (e.g. multiple CT slices)
8. Run the TUDA Workflows:
   a. After logging into the platform there is a navigation menu on the right side with different platform components. Workflows can be started over the Meta-Dashboard (Just confirm the dialog)
   b. On the Meta-Dashboard you can filter for specific criteria by clicking on the graphs or searching for custom tags on the top. Workflows can be selected from the large dropdown menu and will run on the currently active filtered data.

TUDA Workflow Overview

tuda_calc_quality_measures



Description

This workflow only respects passed DICOM-SEG files. For each of these segmentations it searches the PACS for corresponding CT images and calculates the quality metrics for each of these pairs. The JSON results are then stored in DICOM-SR (Structure Report) files and sent back to the PACS. The SRs have the same patient name, study-uid to make them assignable to the correct segmentation. This series description is „*JSON Embedding – Single QM*“ which indicates that the embedded quality metrics belong to a single CT-SEG pair.

When is the workflow executed?

This workflow should be executed whenever new segmentations arrive at the platform or an automated segmentation is generated

Where does this workflow run?

On Local JIP Nodes

tuda_aggregate-quality-measures

Description

This workflow does not need any input data. It automatically searches the PACS for all DICOM-SR files with series description „*JSON Embedding – Single QM*“. Afterwards the QM-JSON metrics from all obtained DICOM-SR are extracted and aggregated (either by their segmentation date or all together as one). The aggregated QM-JSON is then again embedded as one or multiple DICOM-SR and sent to PACS. The metadata for these DICOM-SR can be set inside the code to make them identifiable for Mint as „Statistics Data“ that should be forwarded to Central. The study description of these DICOM-SR is „*JSON Embedding – Aggregated QM*“ to allow searching only aggregated DICOM-SRs later on.

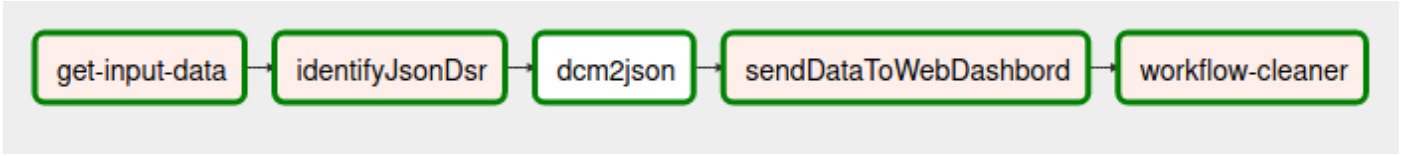
When is the workflow executed?

This workflow should be regularly triggered by a cron job. The frequency is determined by the aggregation paradigm. If the aggregation happens based on segmentation date, running the job only every few days is sufficient. If all data is aggregated into one DICOM-SR file regardless of segmentation dates, the workflow should run every 24h to get a new datapoint each day.

Where does this workflow run?

On Local JIP Nodes

tuda_receive-quality-measures



Description

This workflow automatically checks its input data for DICOM-SR files with series description „*JSON Embedding – Aggregated QM*“. They indicate that the encoded QM-JSONs contain aggregated QM data. In the next step the QM data is extracted and automatically send to the dashboard via a HTTP Post request. The dashboard hostname, port and API key have to be set inside the workflow.

When is the workflow executed?

On all incoming data. DICOM-SRs are will be sorted out and further processed.

Where does this workflow run?

On the RACOON Central JIP node

TUDA Workflow Components (Operators)

dag_tud_calc_quality_measures

LocalGetInputDataOperator

This is not a TUDA operator. It is currently only used to get data into the workflow and should later be replaced by an operator that automatically receives incoming data

PrepareInputDataOperator

If an input_operator is passed, this operator works on the output of this given operator. Otherwise it obtains its data from PACS based on the config passed by the Meta-Dashboard

- Case *input_operator*: Filter the incoming data for DICOM-SEG and load corresponding CT images from PACS
- Case *config*: Extract DICOM-SEG files form the config and load corresponding CT images from PACS

In both cases a side output is a *reference_meta.json* file that contains meta info from the segmentations. This reference file will be later used during creation of the QM-DICOM-SRs to assign them to the right segmentations.

NOTE: The outputs of this operator do not adhere to the batch-directory structure of JIP. This means the outputs are only compatible with the Dcm2ItkOperator that is based on these outputs!

Dcm2ItkOperator

This operator transforms DICOM image and segmentations to nii.gz files

NOTE: The outputs of this operator do not adhere to the batch-directory structure of JIP. This means the outputs are only compatible with the subsequent CalcQualityMetricsOperator!

CalcQualityMetricsOperator

Calculate the quality metrics on the given input images and outputs a metrics JSON file with the results. Currently we have two different operators of this type: QmDicePredictorOperator and QmArtifactsOperator. They are both used inside the workflow in parallel.

NOTE: The outputs of this operator do not adhere to the batch-directory structure of JIP. This means the outputs are only compatible with the subsequent MergeQualityMetricsOperator!

MergeQualityMetricsOperator

This operator takes as input a list of CalcQualityMetricsOperators. It then collects the JSON metric files of all these QM Operators and merges them into a single JSON file.

NOTE: The outputs of this operator do not adhere to the batch-directory structure of JIP. This means the outputs are only compatible with the subsequent Json2DcmSrOperator!

Json2DcmSrOperator

Store the resulting QM Json file from the previous operator into a DICOM-SR for each of the Image-Segmentation-DICOM pairs. These DICOM-SR have the series description „*JSON Embedding – Single QM*“

There are two different *levels* that can be set as a parameter:

- *batch*: Here a reference_meta.json file is required that is generated by the PrepareInputDataOperator. This reference file will be used to set the study and patient meta information according to the original segmentation DICOM. This allows to assign the DICOM-SR to the segmentation.
- *element*: Here no reference_meta.json file is required and will be ignored. The QM JSON file from the previous operator will be directly encoded into a DICOM-SR file. The meta data of this DICOM-SR have to be set as parameters of this operator. (The element level will be later used when aggregated QM data which does not relate to a single patient will be stored into a DICOM-SR)

NOTE: The outputs of this operator do not adhere to the batch-directory structure of JIP. The DICOM-SRs are stored in a operator output directory next to the batch folder.

DcmSendOperator

This is not a TUDA operator. It is used to send the previously generated DICOM-SR files to the PACS.

LocalWorkflowCleaner

This is not a TUDA operator. It is clean up all temporary data created by all operators of a workflow

dag_tud_aggregate_quality_measures

GetAllQmDsrOperator

Loads all DICOM-SR files from PACS that have the „*Single QM*“ inside their series description which indicates that this is DICOM-SR contains QM Data from a single image-segmentation-pair which is not aggregated.

NOTE: The outputs of this operator adhere to the batch-directory structure of JIP

DcmSr2JsonOperator

Extracts all QM JSONs which are embedded in the DICOM-SRs from the previous operator

NOTE: The outputs of this operator adhere to the batch-directory structure of JIP

AggregateQMDataOperator

Aggregates the data of all QM JSONs into new JSON files. There are currently two strategies for aggregation which are controlled by the parameter `aggregation_strategy`

- *all*: aggregates all QM JSONs into a single new aggregated QM JSON file
- *seg_date*: aggregates the QM JSONs according to their segmentation dates. For each occurring segmentation date one aggregated QM JSON file will be generated.
  Additionally a JSON database file is created on the first run that stores for each date the aggregated measures. On all future runs the aggregated data is compared to this database to ensure that only aggregated data from new dates or dates where the aggregations changed are forwarded.

NOTE: The outputs of this operator do not adhere to the batch-directory structure of JIP. The aggregated JSONs are stored in a operator output directory next to the batch folder.

Json2DcmSrOperator

This is the same operator as above. In this case it runs with level *element* since there are no reference_meta files available and the aggregated data is no longer assignable to a single patient.

The generated DICOM-SR will have the series description „*JSON Embedding – Aggregated QM*“ to make them differentiable from the „*Single QM*“ DICOM-SRs and indicate that their content is aggregated data.

NOTE: The outputs of this operator do not adhere to the batch-directory structure of JIP. The aggregated JSONs are stored in a operator output directory next to the batch folder.

DCMSendOperator

Same as above

LocalWorkflowCleanerOperator

Same as above

dag_tud_receive_quality_measure

LocalGetInputDataOperator

Same as for the first workflow, this operator is currently a placeholder for testing. It should be replaced with an operator that automatically passes newly received DICOMs.

IdentifyJsonDsrOperator

This operator takes all files from its previous operator and filters for DICOM-SRs with series description „*JSON Embedding – Aggregated QM*“ since these are the ones that contain the aggregated QM metrics.

NOTE: The outputs of this operator adhere to the batch-directory structure of JIP.

DcmSr2JsonOperator

Same as above the JSON data is extracted from the DICOM-SRs. The structure is the same as before, but the contained data are the aggregated values

NOTE: The outputs of this operator adhere to the batch-directory structure of JIP.

SendDataToWebDashboardOperator

The extracted QM JSONs are taken as inputs and are sent to the web dashboard. This operator also supports other send-to-dashboard actions (e.g. create and send a loss plot from NN logs) which can be controlled by the parameter `command` If the JSON files are in a different location and not passed by an input_operator, the parameter `json_dir` can be set instead and the operator will only send JSON files from this directory to the dashboard.

Valid commands are:

- sendQualityMeasures: this is the command used in this workflow. It will take a QM JSON, brings it into the correct format and sends it as a new DailyData entry to the dashboard
- createLocation: Takes a JSON (has to be in right syntax, examples are given in the code) for creating a new location on the dashboard
- createMeasure: Takes the JSON (has to be in right syntax, examples are given in the code) for creating a new measure on the dashboard
- sendPlots: Takes a JSON that represents the options of an Apache Echarts Plot and sends it to the dashboard given a date and location
- sendNNtrainingLog: Takes the JIP-nn-UNet JSON log as an input and processes it into an Apache echarts plot. With a given date and location it is then sent to the dashboard.

LocalWorkflowCleanerOperator

Same as above