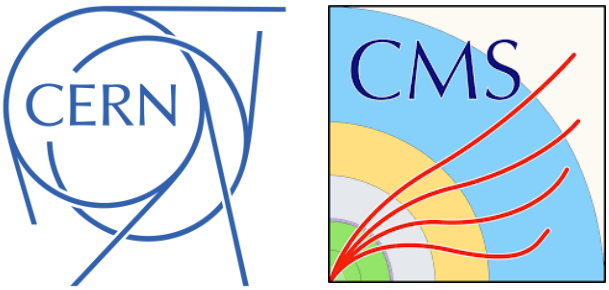

CI/CD Pipeline Optimization: Reducing Docker Image Build Time

By: Janna Almuqaisib

Supervisors:

Thanayut Seethongchuen

Dario Mapelli

Summer Student 2024

August 23rd, 202

Table of Contents

Table of Figures

Acknowledgements

I would like to express my sincere gratitude to all who I have encountered at CERN, and specifically the CRAB team for their invaluable guidance and support throughout my summer internship. Their wealth of knowledge and expertise served as an incredible resource, guiding me through complex and foreign concepts. At the same time, they provided the perfect balance of mentorship and trust, allowing me the freedom to explore various topics and develop my research skill as we tackled new challenges. This experience has been profoundly enriching, and I am deeply appreciative of the opportunities for growth and discovery that I have been provided by all who made it possible.

Introduction

What is CMS?

The Compact Muon Solenoid (CMS) is a detector at the Large Hadron Collider (LHC), designed to study the outcomes from its high-energy proton collisions. The different layers of the CMS cylindrical structure are each dedicated to measuring specific particles produced by the LHC. This detector captures particles by rapidly taking as many as 40 million three-dimensional photographs per second, and thus makes it possible to capture and project the previously fleeting particles of interest for further analysis.

What is CRAB?

CMS Remote Analysis Builder, or CRAB, is a software service which allows physicists and all those working with CMS to access its data and computing resources remotely. The CMS computing system, referred to as 'The Grid', runs 'jobs', which contain parts of datasets retrieved from the detector and translated as data access requests (tasks). This software is exploited using the command-line terminal with custom-built CRAB commands.

Some of the most important components of the CRAB client include the REST server, TaskWorker and Publisher. The REST server connects to the Oracle database which hosts the data from the detector. The TaskWorker takes the user's request submission and assigns it to many different jobs to be run on the Grid. Finally, the Publisher saves the output from these jobs in the database as separate datasets pertaining to the user. The following figure demonstrates the dependencies which exist between the CRAB components and the life cycle of a user request:

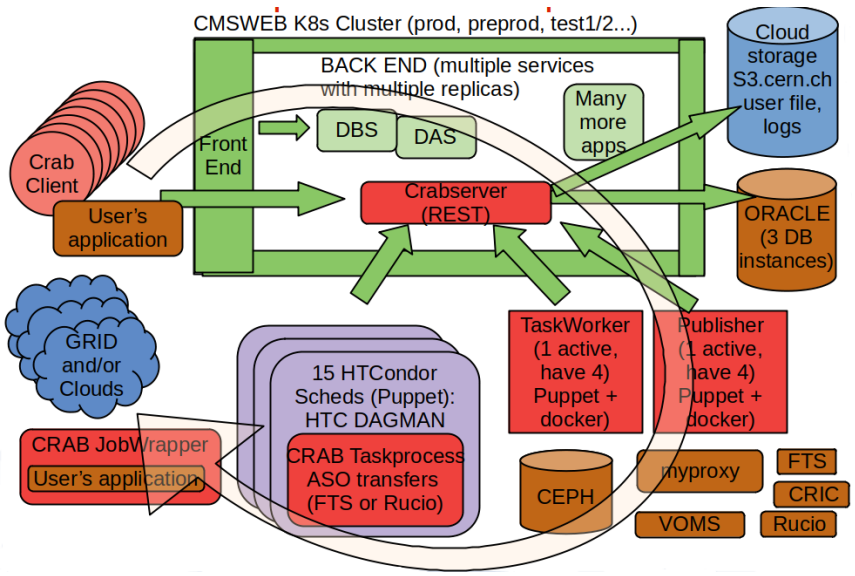

Figure 1: CRAB components and dependencies

CI/CD and Docker

Continuous Integration and Continuous Deployment (CI/CD) refers to the automation and facilitation of delivering new versions of existing software by pushing changes through a series of tests and publications, also known as a ‘pipeline’. Typically, the pipeline consists of three main stages: building, testing and deployment. Many other stages may be included to ensure security, debug individual features using unit tests, and so on.

The main advantages that this pipeline and development style present is minimizing human error, unifying deployment strategies, and exploiting the use of containers to bypass environment compatibility issues across operating systems.

Containers are features provided by the Docker platform, which allows for the creation of infrastructures and environments to streamline the delivery of software as quickly and consistently as possible. Docker operates using ‘images’, which are templates containing instructions on how to build the container. These images consist of many ‘immutable’ layers, meaning that if a lower layer is altered during the building process, all image layers on top of it must be rebuilt as well. This brings us to the problem of how much overhead can result from using Docker in use cases that necessitate frequent image updates.

The Problem

Project Structure

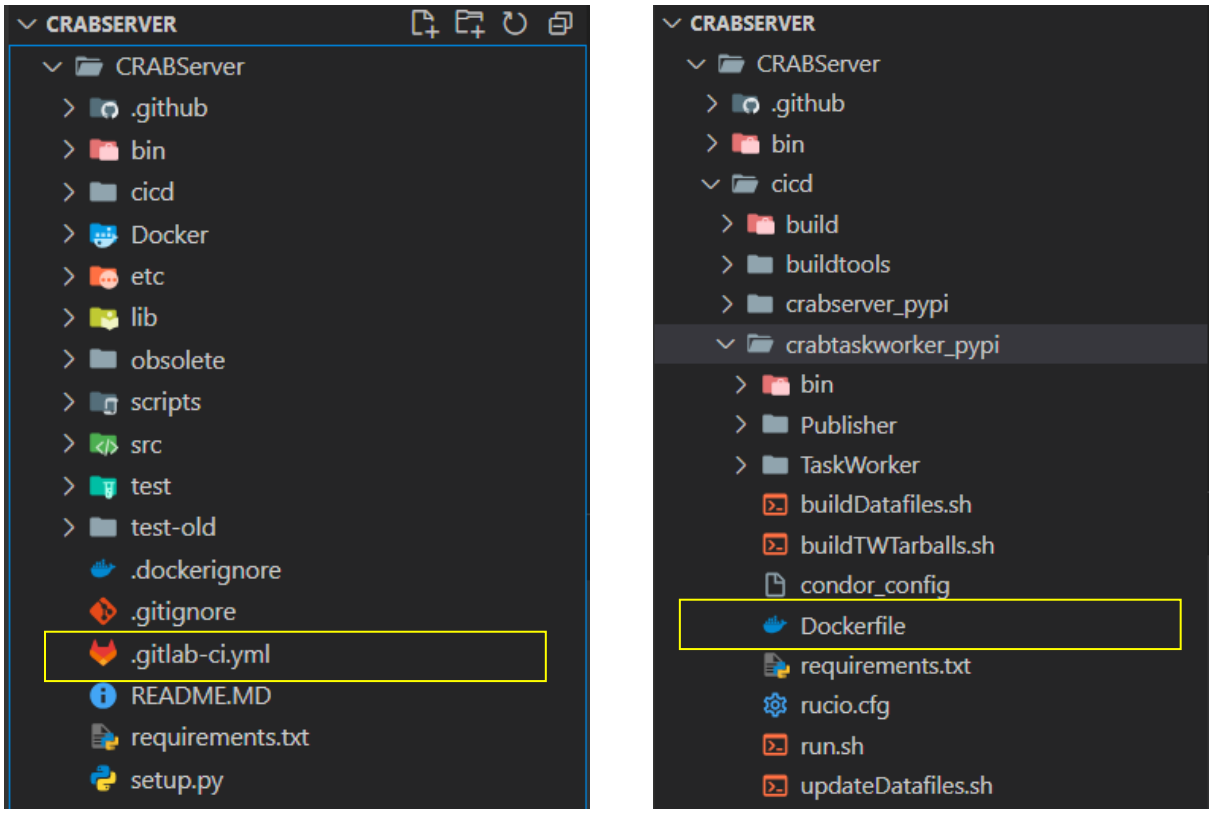

Figure 2: Project Directory Structure

As seen from the figure, the project contains a file '.gitlab-ci.yml', which defines the jobs which handle how the pipeline runs (building, deployment, and testing images). This is where the bulk of the change to how the image is being stored will be integrated.

Within the 'cicd' folder are folders 'crabserver_pypi' and 'crabtaskworker_pypi'. These folders contain the resources, scripts or otherwise, necessary for building the Docker images. The former handles a 'base' or 'primary' image and the latter builds the previously defined 'task worker' image. The Dockerfile which will require modification is the one pertaining to the 'task worker' portion of the pipeline, as this is where the file duplication was happening.

Code Diagnosis

The current CI/CD pipeline in use was discovered to have some inefficiencies that were causing the build time for each image to take significantly longer than necessary. This consumes additional space and computing resources. The following code is a portion of the aforementioned Dockerfile from which the Docker image is being built, and highlighted is a line which was creating a duplication of files. This duplication consumed around an additional 900MB of space.

```
116. ## TaskWorker
117. COPY cicd/crabtaskworker_pypi/TaskWorker/start.sh \
118.     cicd/crabtaskworker_pypi/TaskWorker/env.sh \
119.     cicd/crabtaskworker_pypi/TaskWorker/stop.sh \
120.     cicd/crabtaskworker_pypi/TaskWorker/manage.sh \
121.     cicd/crabtaskworker_pypi/updateDatafiles.sh \
122.     ${WDIR}/srv/TaskManager/
123.
124. ## publisher
125. COPY cicd/crabtaskworker_pypi/Publisher/start.sh \
126.     cicd/crabtaskworker_pypi/Publisher/env.sh \
127.     cicd/crabtaskworker_pypi/Publisher/stop.sh \
128.     cicd/crabtaskworker_pypi/Publisher/manage.sh \
129.     ${WDIR}/srv/Publisher/
130.
131. ## entrypoint
132. COPY cicd/crabtaskworker_pypi/run.sh /data
133.
134. # for debugging purpose
135. RUN echo "${USER} ALL=(ALL) NOPASSWD:ALL" > /etc/sudoers.d/01-crab3
136.
137. # make sure all /data own by running user
138. RUN chown -R 1000:1000 ${WDIR}
139.
140. USER ${USER}
141.
142. ENTRYPOINT ["tini", "--"]
143. CMD ["/data/run.sh"]
```

This command assigns ownership to the current user for all of the layers being built instead of applying them as necessary to individual layers. This creates many duplicate files which consume computing resources and memory in the registry. It became necessary to find a way to alleviate this issue through a few different suggested avenues.

Objectives

In order to resolve the problem of duplicate files and redundancies in the code which were producing more overhead, an establishment of concrete improvement metrics was necessary. This consisted of the following:

- Implement the use of a service called ‘Docker-in-Docker’, as it allowed the exploitation of a much more extensive library of tools and functions developed in compatibility with docker commands. This would replace ‘Kaniko’, the current service.
- Modularize the image-building jobs by separating the core processes, with the goal of diagnosing the source of the file duplication.
- Introduce image caching such that the cache does not consume more memory than the original image being built with duplications of some layers.
- Upon building a new image, ensure that the jobs are pulling from the image cache created during the previous build.
- Integrate these changes with the original configuration of the pipeline and ensure that none of the subsequent and dependent jobs are hindered.

Tools and Technologies

To enact the necessary modifications to this pipeline, it was necessary to make use of and understand the following technologies:

- Docker
  - Containers
  - Docker Hub
  - Docker-in-Docker
  - Docker BuildX library - caching
- Git
  - GitLab CI/CD
    - Runners
    - Pipelines
  - GitHub
    - Push Requests
    - Repository Mirroring
- Dive
  - CLI commands - inspecting images
- CRAB Virtual Machines
  - CentOS - Linux commands
- Authorization
  - LXPLUS Membership
  - VOMS Digital Certificate

Results

Project Modifications

Dockerfile

```
# install gfal
# symlink to workaround calling gfal from absolute path
COPY --from=wmcore-gfal ${WDIR}/miniconda ${WDIR}/miniconda
RUN ln -sf ${WDIR}/miniconda/bin/gfal-ls /usr/bin/gfal-ls \
    && ln -sf ${WDIR}/miniconda/bin/gfal-rm /usr/bin/gfal-rm \
    && ln -sf ${WDIR}/miniconda/bin/gfal-copy /usr/bin/gfal-copy \
WORKDIR /build

# install dependencies
COPY --from=wmcore-src /wmcore_version .
COPY cicd/crabtaskworker_pypi/requirements.txt .
RUN pip install -r requirements.txt \
    && pip install --no-deps wmcore==$(cat wmcore_version) \
    && pip cache purge

# copy htcondor config
RUN mkdir /etc/condor
COPY cicd/crabtaskworker_pypi/condor_config /etc/condor/

# install crabserver
# will replace with pip later
COPY src/python/ ${WDIR}/srv/current/lib/python/site-packages/
# copy TaskManagerRun.tar.gz
COPY --from=build-data /build/data_files/data ${WDIR}/srv/current/lib/python/site-
packages/data

# copy cern openldap config
COPY --from=cern-cc7 /etc/openldap /etc/openldap

# copy rucio config
RUN mkdir -p /opt/rucio/etc/
COPY cicd/crabtaskworker_pypi/rucio.cfg /opt/rucio/etc/

# add github repos, reuse script in crabserver_pypi
COPY cicd/crabserver_pypi/addGH.sh .
RUN bash addGH.sh

# clean up
    && install -o ${USER} -d ${WDIR}

# create working directory
RUN mkdir -p ${WDIR}/srv/tmp
## taskworker
RUN mkdir -p ${WDIR}/srv/TaskManager/current \
           ${WDIR}/srv/TaskManager/cfg \
           ${WDIR}/srv/TaskManager/logs
## publisher
RUN mkdir -p ${WDIR}/srv/Publisher/current \
           ${WDIR}/srv/Publisher/cfg \
           ${WDIR}/srv/Publisher/logs \
           ${WDIR}/srv/Publisher/PublisherFiles

# copy process executor scripts
## TaskWorker
COPY cicd/crabtaskworker_pypi/TaskWorker/start.sh \
     cicd/crabtaskworker_pypi/TaskWorker/env.sh \
     cicd/crabtaskworker_pypi/TaskWorker/stop.sh \
     cicd/crabtaskworker_pypi/TaskWorker/manage.sh \
     cicd/crabtaskworker_pypi/updateDatafiles.sh \
     ${WDIR}/srv/TaskManager/

COPY cicd/crabtaskworker_pypi/bin/crab-taskworker cicd/crabtaskworker_pypi/bin/crab-
publisher /usr/local/bin
```

```
62.
63. ## publisher
64. COPY cicd/crabtaskworker_pypi/Publisher/start.sh \
65.      cicd/crabtaskworker_pypi/Publisher/env.sh \
66.      cicd/crabtaskworker_pypi/Publisher/stop.sh \
67.      cicd/crabtaskworker_pypi/Publisher/manage.sh \
68.      ${WDIR}/srv/Publisher/
69.
70. ## entrypoint
71. COPY cicd/crabtaskworker_pypi/run.sh /data
72.
73. # for debuggin purpose
74. RUN echo "${USER} ALL=(ALL) NOPASSWD:ALL" > /etc/sudoers.d/01-crab3
75.
76. # make sure all /data own by running user
77. RUN chown -R 1000:1000 ${WDIR}
78.
79. USER ${USER}
80.
81. ENTRYPOINT ["tini", "--"]
82. CMD ["/data/run.sh"]
```

Figure 3: Dockerfile Before Modifications

```
  1. # start image
  2. FROM registry.cern.ch/cmsweb/wmagent-base:pypi-20230705
  3. SHELL ["/bin/bash", "-c"]
  4. ENV USER=crab3
  5. ENV WDIR=/data
  6.
  7. # install gfal
  8. # symlink to workaround calling gfal from absolute path
  9. COPY --chown=${USER}:${USER} --from=wmcore-gfal ${WDIR}/miniconda ${WDIR}/miniconda/
 10. RUN ln -sf ${WDIR}/miniconda/bin/gfal-ls /usr/bin/gfal-ls \
 11.     && ln -sf ${WDIR}/miniconda/bin/gfal-rm /usr/bin/gfal-rm \
 12.     && ln -sf ${WDIR}/miniconda/bin/gfal-copy /usr/bin/gfal-copy \
 13.     && ln -sf ${WDIR}/miniconda/bin/gfal-sum /usr/bin/gfal-sum
 14.
 15. # install package from debian repository
 16. # deps for openldap: libsasl2-dev python3-dev libldap-dev libssl-dev
 17. RUN apt-get update \
 18.     && apt-get install -y tini git zip voms-clients-java fd-find ripgrep libsasl2-dev
python3-dev libldap-dev libssl-dev \
 19.     && apt-get clean all
 20.
 21. # local timezone (hardcode)
 22. RUN ln -sf /usr/share/zoneinfo/Europe/Zurich /etc/localtime
 23.
 24. # prepare build
 25. RUN mkdir /build
 26. WORKDIR /build
 27.
 28. # install dependencies
 29. COPY --chown=${USER}:${USER} --from=wmcore-src /wmcore_version ./
 30. COPY --chown=${USER}:${USER} cicd/crabtaskworker_pypi/requirements.txt ./
 31. RUN pip install -r requirements.txt \
 32.     && pip install --no-deps wmcore==$(cat wmcore_version) \
 33.     && pip cache purge
 34.
 35. # copy htcondor config
 36. RUN mkdir /etc/condor
 37. COPY --chown=${USER}:${USER} cicd/crabtaskworker_pypi/condor_config /etc/condor/
 38.
 39. # install crabserver
 40. # will replace with pip later
 41. COPY --chown=${USER}:${USER} src/python/ ${WDIR}/srv/current/lib/python/site-packages/
 42. # copy TaskManagerRun.tar.gz
 43. COPY --chown=${USER}:${USER} --from=build-data /build/data_files/data
${WDIR}/srv/current/lib/python/site-packages/data/
 44.
```

```
45. # copy cern openldap config
46. COPY --chown=${USER}:${USER} --from=cern-cc7 /etc/openldap /etc/openldap/
47.
48. # copy rucio config
49. RUN mkdir -p /opt/rucio/etc/
50. COPY --chown=${USER}:${USER} cicd/crabtaskworker_pypi/rucio.cfg /opt/rucio/etc/
51.
52. # add github repos, reuse script in crabserver_pypi
53. COPY --chown=${USER}:${USER} cicd/crabserver_pypi/addGH.sh ./
54. RUN bash addGH.sh
55.
56. # clean up
57. WORKDIR ${WDIR}
58. RUN rm -rf /build
59.
60. # add new user and switch to user
61. RUN useradd -m ${USER} \
62.     && install -o ${USER} -d ${WDIR}
63.
64. # create working directory
65. RUN install -d -o ${USER} -g ${USER} ${WDIR}/srv/tmp
66. # Create directories for TaskWorker
67. RUN install -d -o ${USER} -g ${USER} ${WDIR}/srv/TaskManager/current \
68.     && install -d -o ${USER} -g ${USER} ${WDIR}/srv/TaskManager/cfg \
69.     && install -d -o ${USER} -g ${USER} ${WDIR}/srv/TaskManager/logs \
70.     # Change ownership to the running user
71.     && chown -R ${USER}:${USER} ${WDIR}/srv
72.
73. # Create directories for Publisher
74. RUN install -d -o ${USER} -g ${USER} ${WDIR}/srv/Publisher/current \
75.     && install -d -o ${USER} -g ${USER} ${WDIR}/srv/Publisher/cfg \
76.     && install -d -o ${USER} -g ${USER} ${WDIR}/srv/Publisher/logs \
77.     && install -d -o ${USER} -g ${USER} ${WDIR}/srv/Publisher/PublisherFiles \
78.     # Change ownership of parent and current directories to the running user
79.     && chown -R ${USER}:${USER} ${WDIR}/srv
80.
81. # copy process executor scripts
82. ## TaskWorker
83. COPY --chown=${USER}:${USER} cicd/crabtaskworker_pypi/TaskWorker/start.sh \
84.      cicd/crabtaskworker_pypi/TaskWorker/env.sh \
85.      cicd/crabtaskworker_pypi/TaskWorker/stop.sh \
86.      cicd/crabtaskworker_pypi/TaskWorker/manage.sh \
87.      cicd/crabtaskworker_pypi/updateDatafiles.sh \
88.      ${WDIR}/srv/TaskManager/
89.
90. COPY --chown=${USER}:${USER} cicd/crabtaskworker_pypi/bin/crab-taskworker
cicd/crabtaskworker_pypi/bin/crab-publisher /usr/local/bin/
91.
92. ## publisher
93. COPY --chown=${USER}:${USER} cicd/crabtaskworker_pypi/Publisher/start.sh \
94.      cicd/crabtaskworker_pypi/Publisher/env.sh \
95.      cicd/crabtaskworker_pypi/Publisher/stop.sh \
96.      cicd/crabtaskworker_pypi/Publisher/manage.sh \
97.      ${WDIR}/srv/Publisher/
98.
99. ## entrypoint
100. COPY --chown=${USER}:${USER} cicd/crabtaskworker_pypi/run.sh /data/
101.
102. # for debuggin purpose
103. RUN echo "${USER} ALL=(ALL) NOPASSWD:ALL" > /etc/sudoers.d/01-crab3
104.
105. USER ${USER}
106.
107. ENTRYPOINT ["tini", "--"]
108. CMD ["/data/run.sh"]
```

Figure 4: Dockerfile After Modifications (Highlighted)

GitLab Configuration File (.gitlab-ci.yml)

```yaml
---
default:
  tags:
    - crab3

variables:
  IMAGE_TAG: "${CI_COMMIT_REF_SLUG}"  # to distinct it from commit tag and final image tag

.default_rules:
  default:
    - if: $CI_COMMIT_TAG =~ /pypi-.*/         # pypi-(prod|preprod|test*)-1714418922
  release:
    - if: $CI_COMMIT_TAG =~ /v3\.[0-9]{6}.*/  # same as above

stages:
  - prepare_env
  - prepare_release
  - build_docker
  - deploy
  - run_testsuite
  - check_testsuite
  - tagging_release

get_env:
  rules:
    - !reference [.default_rules, default]
    - !reference [.default_rules, release]
  stage: prepare_env
  image:
    name: registry.cern.ch/cmscrab/buildtools
    entrypoint: [""]
  script:
    - printenv  # debug check ci env
    - cicd/gitlab/parseEnv.sh $CI_COMMIT_TAG  # create .env
  artifacts:
    paths:
      - .env
    expire_in: 1 week

set_version_name:
  rules:
    - !reference [.default_rules, release]
  stage: prepare_release
  image:
    name: registry.cern.ch/cmscrab/buildtools
    entrypoint: [""]
  script:
    - |
        echo -e "\n__version__ = \"${RELEASE_NAME:-$CI_COMMIT_TAG}\" #Automatically added
during build process" >> src/python/TaskWorker/__init__.py;
    - |
        echo -e "\n__version__ = \"${RELEASE_NAME:-$CI_COMMIT_TAG}\" #Automatically added
during build process" >> src/python/CRABInterface/__init__.py;
  cache:
    - key: $CI_PIPELINE_ID
      paths:
        - src/python/TaskWorker/__init__.py
        - src/python/CRABInterface/__init__.py
      policy: push

build_rest_image:
  rules:
    - !reference [.default_rules, default]
    - !reference [.default_rules, release]
  stage: build_docker
  image:
    name: gcr.io/kaniko-project/executor:v1.14.0-debug
```

```
 66.     entrypoint: [""]
 67.   script:
 68.     - echo "{\"auths\":{\"$CMSCRAB_REGISTRY_URL\":{\"auth\":\"$(echo -n
$CMSCRAB_REGISTRY_USER:$CMSCRAB_REGISTRY_PASSWORD | base64)\"}}}" > /kaniko/.docker/config.json
 69.     - cat /kaniko/.docker/config.json
 70.     - /kaniko/executor
 71.       --context "${CI_PROJECT_DIR}"
 72.       --dockerfile "${CI_PROJECT_DIR}/cicd/crabserver_pypi/Dockerfile"
 73.       --destination "registry.cern.ch/cmscrab/crabserver:${IMAGE_TAG}"
 74.   cache:
 75.     - key: $CI_PIPELINE_ID
 76.       paths:
 77.         - src/python/CRABInterface/__init__.py
 78.       policy: pull
 79.
 80. build_tw_image:
 81.   rules:
 82.     - !reference [.default_rules, default]
 83.     - !reference [.default_rules, release]
 84.   stage: build_docker
 85.   image:
 86.     name: gcr.io/kaniko-project/executor:v1.14.0-debug
 87.     entrypoint: [""]
 88.   script:
 89.     - echo "{\"auths\":{\"$CMSCRAB_REGISTRY_URL\":{\"auth\":\"$(echo -n
$CMSCRAB_REGISTRY_USER:$CMSCRAB_REGISTRY_PASSWORD | base64)\"}}}" > /kaniko/.docker/config.json
 90.     - cat /kaniko/.docker/config.json
 91.     - /kaniko/executor
 92.       --context "${CI_PROJECT_DIR}"
 93.       --dockerfile "${CI_PROJECT_DIR}/cicd/crabtaskworker_pypi/Dockerfile"
 94.       --destination "registry.cern.ch/cmscrab/crabtaskworker:${IMAGE_TAG}"
 95.   cache:
 96.     - key: $CI_PIPELINE_ID
 97.       paths:
 98.         - src/python/TaskWorker/__init__.py
 99.       policy: pull
100.
101. build_monit_image:
102.   rules:
103.     - !reference [.default_rules, default]
104.     - !reference [.default_rules, release]
105.   stage: build_docker
106.   needs: ["build_tw_image"]
107.   image:
108.     name: gcr.io/kaniko-project/executor:v1.14.0-debug
109.     entrypoint: [""]
110.   script:
111.     - echo "{\"auths\":{\"$CMSCRAB_REGISTRY_URL\":{\"auth\":\"$(echo -n
$CMSCRAB_REGISTRY_USER:$CMSCRAB_REGISTRY_PASSWORD | base64)\"}}}" > /kaniko/.docker/config.json
112.     - cat /kaniko/.docker/config.json
113.     - /kaniko/executor
114.       --context "${CI_PROJECT_DIR}"
115.       --dockerfile "${CI_PROJECT_DIR}/cicd/monit_pypi/Dockerfile"
116.       --destination "registry.cern.ch/cmscrab/crabtaskworker:${IMAGE_TAG}.monit"
117.       --build-arg "BASE_TAG=${IMAGE_TAG}"
118.   cache:
119.     - key: $CI_PIPELINE_ID
120.       paths:
121.         - src/python/TaskWorker/__init__.py
122.       policy: pull
123.
269.     - crane auth login -u ${CMSCRAB_REGISTRY_USER} -p ${CMSCRAB_REGISTRY_PASSWORD}
${CMSCRAB_REGISTRY_URL}
270.     - crane cp registry.cern.ch/cmscrab/crabtaskworker:${IMAGE_TAG}.monit
registry.cern.ch/cmscrab/crabtaskworker:v3.latest.monit
271.
```

Figure 5: GitLab Configuration File Before Modifications

```yaml
---
default:
  tags:
    - crab3

variables:
  IMAGE_TAG: "${CI_COMMIT_REF_SLUG}"  # to distinct it from commit tag and final image tag
  # The `DOCKER_TLS_CERTDIR` variables is needed to run Docker-in-Docker, `DOCKER_BUILDKIT` is to make sure the docker build use the new BuildKit.
  # https://docs.gitlab.com/ee/ci/docker/using_docker_build.html#docker-in-docker-with-tls-enabled-in-the-docker-executor
  # Creating a docker context is required to be able to cache to the registry using Buildkit.
  # https://docs.docker.com/build/cache/backends/
  DOCKER_TLS_CERTDIR: "/certs"
  DOCKER_BUILDKIT: 1

.default_rules:
  default:
    - if: $CI_COMMIT_TAG =~ /pypi-.*/         # pypi-(prod|preprod|test*)-1714418922
  release:
    - if: $CI_COMMIT_TAG =~ /v3\.[0-9]{6}.*/  # same as above

stages:
  - prepare_env
  - prepare_release
  - build_docker
  - deploy
  - run_testsuite
  - check_testsuite
  - tagging_release

get_env:
  rules:
    - !reference [.default_rules, default]
    - !reference [.default_rules, release]
  stage: prepare_env
  image:
    name: registry.cern.ch/cmscrab/buildtools
    entrypoint: [""]
  script:
    - printenv  # debug check ci env
    - cicd/gitlab/parseEnv.sh $CI_COMMIT_TAG  # create .env
  artifacts:
    paths:
      - .env
    expire_in: 1 week

set_version_name:
  rules:
    - !reference [.default_rules, release]
  stage: prepare_release
  image:
    name: registry.cern.ch/cmscrab/buildtools
    entrypoint: [""]
  script:
    - |
        echo -e "\n__version__ = \"${RELEASE_NAME:-$CI_COMMIT_TAG}\" #Automatically added during build process" >> src/python/TaskWorker/__init__.py;
    - |
        echo -e "\n__version__ = \"${RELEASE_NAME:-$CI_COMMIT_TAG}\" #Automatically added during build process" >> src/python/CRABInterface/__init__.py;
  cache:
    - key: $CI_PIPELINE_ID
      paths:
        - src/python/TaskWorker/__init__.py
        - src/python/CRABInterface/__init__.py
      policy: push

```

```
build_rest_image:
  rules:
    - if: $SUBMIT_STATUS_TRACKING
      when: never
    - !reference [.default_rules, default]
    - !reference [.default_rules, release]
  stage: build_docker
  image:
    name: docker:27.1.1
  services:
    - name: docker:27.1.1-dind
  before_script:
    - docker info
  script:
    - docker login -u $CMSCRAB_REGISTRY_USER -p $CMSCRAB_REGISTRY_PASSWORD $CMSCRAB_REGISTRY_URL
    - source .env
    - docker context create mycontext
    - docker buildx create mycontext --use --name mybuilder --bootstrap
    - docker buildx build --push -f "${CI_PROJECT_DIR}/cicd/crabserver_pypi/Dockerfile" --cache-to=type=registry,ref="registry.cern.ch/cmscrab/crabserver:pypi-${REST_Instance}-cache",image-manifest=true,mode=max --cache-from=type=registry,ref="registry.cern.ch/cmscrab/crabserver:pypi-${REST_Instance}-cache" -t "registry.cern.ch/cmscrab/crabserver:${IMAGE_TAG}" .
  cache:
    - key: $CI_PIPELINE_ID
      paths:
        - src/python/CRABInterface/__init__.py
      policy: pull

build_image:
  rules:
    - if: $SUBMIT_STATUS_TRACKING
      when: never
    - !reference [.default_rules, default]
    - !reference [.default_rules, release]
  stage: build_docker
  image:
    name: docker:27.1.1
  services:
    - name: docker:27.1.1-dind
  script:
    - docker login -u $CMSCRAB_REGISTRY_USER -p $CMSCRAB_REGISTRY_PASSWORD $CMSCRAB_REGISTRY_URL
    - source .env
    - docker context create mycontext
    - docker buildx create mycontext --use --name mybuilder --bootstrap
    - docker buildx build --push -f "${CI_PROJECT_DIR}/cicd/crabtaskworker_pypi/Dockerfile" --cache-to=type=registry,ref="registry.cern.ch/cmscrab/crabtaskworker:pypi-${REST_Instance}-cache",image-manifest=true,mode=max --cache-from=type=registry,ref="registry.cern.ch/cmscrab/crabtaskworker:pypi-${REST_Instance}-cache" -t "registry.cern.ch/cmscrab/crabtaskworker:${IMAGE_TAG}" .
  cache:
    - key: $CI_PIPELINE_ID
      paths:
        - src/python/TaskWorker/__init__.py
      policy: pull

build_monit_image:
  rules:
    - if: $SUBMIT_STATUS_TRACKING
      when: never
    - !reference [.default_rules, release]
  stage: build_docker
  needs: ["build_image"]
  image:
    name: docker:27.1.1
  services:
    - name: docker:27.1.1-dind
  script:
```

```
125.     - docker login -u $CMSCRAB_REGISTRY_USER -p $CMSCRAB_REGISTRY_PASSWORD
$CMSCRAB_REGISTRY_URL
126.     - source .env
127.     - docker context create mycontext
128.     - docker buildx create mycontext --use --name mybuilder --bootstrap
129.     - docker buildx build --push --build-arg="BASE_TAG=${IMAGE_TAG}" -f
"${CI_PROJECT_DIR}/cicd/monit_pypi/Dockerfile" --cache-
to=type=registry,ref="registry.cern.ch/cmscrab/crabtaskworker:pypi-${REST_Instance}-
cache",image-manifest=true,mode=max --cache-
from=type=registry,ref="registry.cern.ch/cmscrab/crabtaskworker:pypi-${REST_Instance}-cache" -t
"registry.cern.ch/cmscrab/crabtaskworker:${IMAGE_TAG}.monit" .
130.   cache:
131.     - key: $CI_PIPELINE_ID
132.       paths:
133.         - src/python/TaskWorker/__init__.py
134.       policy: pull
135.
136. build_crabtwfilebeat_image:
137.   rules:
138.     - if: $SUBMIT_STATUS_TRACKING
139.       when: never
140.     - !reference [.default_rules, release]
141.   stage: build_docker
142.   image:
143.     name: docker:27.1.1
144.   services:
145.     - name: docker:27.1.1-dind
146.   script:
147.     - docker login -u $CMSCRAB_REGISTRY_USER -p $CMSCRAB_REGISTRY_PASSWORD
$CMSCRAB_REGISTRY_URL
148.     - source .env
149.     - docker context create mycontext
150.     - docker buildx create mycontext --use --name mybuilder --bootstrap
151.     - docker buildx build --push --build-arg="BASE_TAG=${IMAGE_TAG}" -f
"${CI_PROJECT_DIR}/cicd/filebeat/Dockerfile" --cache-
to=type=registry,ref="registry.cern.ch/cmscrab/crabtwfilebeat:pypi-${REST_Instance}-
cache",image-manifest=true,mode=max --cache-
from=type=registry,ref="registry.cern.ch/cmscrab/crabtwfilebeat:pypi-${REST_Instance}-cache" -t
"registry.cern.ch/cmscrab/crabtwfilebeat:${IMAGE_TAG}" .
152.   cache:
153.     - key: $CI_PIPELINE_ID
154.       paths:
155.         - src/python/TaskWorker/__init__.py
156.       policy: pull
157.
```

Figure 6: GitLab Configuration File After Modifications (Highlighted)

Run Time Improvements

The following figure represents the average case of the running times of each of the pipelines from before and after the modifications were made. The reduction in build time may be observed through the ‘build_rest_image’ and ‘build_image’ or ’build_tw_image’ jobs. As such, the deployment jobs also demonstrate a reduction in execution time due to the presence of the cache assisting in the pulling of the image. The test suite, or testing jobs, often introduce more variability in duration and are not regarded when assessing whether there has been an improvement in the image creation process.

Job	**Before (minutes)**	**After (minutes)**

build_rest_image	2:41	0:23
build_tw_image/build_image	6:11	1:10
deploy_server	0:15	0:08
deploy_taskworker	0:54	0:26
deploy_publisher_schedd	0:53	0:11
deploy_publisher_rucio	0:53	0:10
check_test_result	124:50	140:13
task_submission_status_tracking	0:47	0:46

Figure 7: Pipeline Runtime Comparison

Challenges

The most difficult obstacle when it came to brainstorming and looking into solutions for the existing issue was the lack of experience with the tools and the development platform. As a new user of git and specifically Gitlab CI/CD, this posed a major learning curve. Similarly, working on a Windows OS was not ideal, as the virtual machines were configured to have CentOS, thus using Linux commands. Once the unfamiliarity with these tools was resolved, progress could be made in logically optimizing the pipeline.

Conclusion

The process of optimizing a CI/CD pipeline by reducing Docker image build times for the CRAB project at CERN was indeed successful. Of the aforementioned objectives, significant improvements were achieved by implementing Docker-in-Docker and introducing image caching. This resulted in substantial reductions in build and deployment times. The modifications streamlined the pipeline, reducing resource consumption and improving efficiency. Despite initial challenges due to unfamiliarity with the tools and environment, the project successfully enhanced the overall performance of the CI/CD pipeline.

Resources/ References

https://cmscrab.docs.cern.ch/technical/high-level/crab-overview.html

https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideCrab#Introduction

https://home.cern/science/experiments/cms

https://www.redhat.com/en/topics/devops/what-cicd-pipeline

https://github.com/dmwm/CRABServer/issues/8522