

Dockerfile Streamlit Explanation



```

Explain what this Dockerfile do: #
# This is AWS Content subject to the terms of the Customer Agreement
# -----
# File content:
#   Docker image of the streamlit container
FROM --platform=linux/amd64 python:3.9-slim
WORKDIR /app

RUN apt-get update -y && apt-get install -y --no-install-recommends\
    build-essential \
    curl \
    software-properties-common \
    git \
    && rm -rf /var/lib/apt/lists/*

COPY streamlit/pyproject.toml /app
COPY streamlit/.streamlit/ /app/.streamlit/
COPY shared_module/python /app/shared_module

RUN pip3 --no-cache-dir install -U pip
RUN pip3 --no-cache-dir install poetry
RUN poetry config virtualenvs.create false
RUN poetry install --only main

COPY streamlit/src/ /app/src

EXPOSE 8501

HEALTHCHECK CMD curl --fail http://localhost:8501/_stcore/health

ENTRYPOINT ["streamlit", "run", "src/Home.py", "--server.port=8501", "--server.address=0.0.0.0"]

```



This Dockerfile builds a Docker image for a **Streamlit** web application using Python 3.9. It specifically uses the slim variant of the Python image to keep the container lightweight. Here's a breakdown of each part:

1. FROM --platform=linux/amd64 python:3.9-slim:

- This sets the base image to Python 3.9-slim for the `amd64` platform, which is a minimal version of Python to keep the image smaller.

2. WORKDIR /app:

- Sets the working directory inside the container to `/app`, where the application code will be copied and executed.

3. **RUN `apt-get update -y && apt-get install -y --no-install-recommends...`:**
 - Updates the package index and installs necessary dependencies:
 - `build-essential` for compiling Python dependencies.
 - `curl` for network requests.
 - `software-properties-common` for managing software repositories.
 - `git` for version control.
 - After installation, it removes the package index files from the cache to keep the image size small.
4. **COPY `streamlit/pyproject.toml /app` and other COPY commands:**
 - These commands copy various parts of your local project into the container:
 - `pyproject.toml`: This file is used by **Poetry** (a dependency manager) to define and install dependencies.
 - `.streamlit/`: Streamlit configuration files.
 - `shared_module/python`: Shared Python module for your project.
 - `src/`: The source code of the Streamlit app.
5. **RUN `pip3 --no-cache-dir install -U pip`:**
 - Updates `pip` to the latest version while avoiding caching to keep the image small.
6. **RUN `pip3 --no-cache-dir install poetry`:**
 - Installs **Poetry**, a tool for dependency management in Python.
7. **RUN `poetry config virtualenvs.create false`:**
 - Configures Poetry not to create a virtual environment, as dependencies will be installed globally in the container.
8. **RUN `poetry install --only main`:**
 - Installs only the main dependencies specified in `pyproject.toml`, ignoring development dependencies.
9. **EXPOSE 8501:**
 - Informs Docker that the container listens on port **8501**, the default port for Streamlit applications.
10. **HEALTHCHECK CMD `curl --fail http://localhost:8501/_stcore/health`:**
 - Adds a health check to ensure the application is running by making a request to the Streamlit health endpoint (`/health`). If the health check fails, Docker marks the container as unhealthy.
11. **ENTRYPOINT [`"streamlit", "run", "src/Home.py", "--server.port=8501", "--server.address=0.0.0.0"`]:**
 - Defines the command to run the Streamlit app, specifically the `Home.py` script located in the `src/` directory. It binds the app to **port 8501** and makes it accessible from any network interface (`0.0.0.0`).

In summary, this Dockerfile creates a container for a Streamlit app by installing necessary system packages, managing Python dependencies using Poetry, and setting up the container to run the app on port 8501.



What does this build docker file do: # FAS Modified this file to not use 'latest as the IMAGE_TAG'
 # This may be the reason why the nested cloudformation stack
 # ends up being run every time. This is the stack that defines
 # the ECS objects and is the reason the stack hangs. So if
 # we want to avoid running it if possible.

```
AWS_ACCOUNT_ID=$1
AWS_REGION='us-east-1'

IMAGE_TAG='v1.0' # Use versioned tags
# ECR_REPOSITORY='public.ecr.aws/v1a3q6c0/streamlit-temp-stack:latest'
ECR_REPOSITORY='cdk-hnb659fds-container-assets-454674044397-us-east-1'
#ECR_REPOSITORY='cdk-hnb659fds-container-assets-562860900886-us-east-1'
aws ecr-public get-login-password --region us-east-1 | docker login --username AWS --password-
stdin public.ecr.aws/v1a3q6c0
docker build . --tag $IMAGE_TAG
docker tag $IMAGE_TAG
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$ECR_REPOSITORY:$IMAGE_TAG
eval $(aws ecr get-login --no-include-email)
docker push
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$ECR_REPOSITORY:$IMAGE_TAG
```



This script builds and pushes a Docker image to AWS Elastic Container Registry (ECR), but it has been modified to avoid using the `latest` tag for the Docker image, which may have previously caused issues with CloudFormation and ECS stacks. Here's a breakdown of what each part of the script does:

1. **AWS_ACCOUNT_ID and AWS_REGION:**

- `$1` is a positional argument that is passed to the script representing the AWS account ID.
- The AWS region is hardcoded to `us-east-1`.

2. **IMAGE_TAG='v1.0':**

- The script uses a versioned tag (`v1.0`) for the Docker image instead of the default `latest` tag to avoid CloudFormation/ECS issues related to image updates.

3. **ECR_REPOSITORY='cdk-hnb659fds-container-assets-454674044397-us-east-1':**

- This is the name of the Amazon ECR repository where the Docker image will be pushed. The script uses a specific repository rather than the commented-out public ECR repository (`public.ecr.aws`).

4. **aws ecr-public get-login-password --region us-east-1 | docker login --username AWS --password-stdin public.ecr.aws/v1a3q6c0:**

- This command logs in to **AWS Public ECR** (though this repository is not actively used in the script) using an authentication token retrieved from AWS. The login is performed using the Docker CLI.

5. **docker build . --tag \$IMAGE_TAG:**

- This command builds a Docker image from the current directory (`.`) and tags it with the version `v1.0` specified by `$IMAGE_TAG`.

6. **docker tag \$IMAGE_TAG**

\$AWS_ACCOUNT_ID.dkr.ecr.\$AWS_REGION.amazonaws.com/\$ECR_REPOSITORY:\$IMAGE_TAG:

- After building the Docker image, this command re-tags it with the full ECR repository name and versioned tag. This makes the image ready to be pushed to the private Amazon ECR.

7. **eval \$(aws ecr get-login --no-include-email):**

- This retrieves login credentials for the private AWS ECR repository and logs in using the Docker CLI. The `--no-include-email` flag ensures that the email address is omitted from the login process (this option is deprecated in some regions).

8. docker push

\$AWS_ACCOUNT_ID.dkr.ecr.\$AWS_REGION.amazonaws.com/\$ECR_REPOSITORY:\$IMAGE_TAG:

- This command pushes the tagged Docker image to the specified Amazon ECR repository.

Summary:

- The script builds a Docker image with a specific version tag (`v1.0`), avoiding the `latest` tag to prevent issues with CloudFormation/ECS stacks hanging.
- It logs in to both public and private AWS ECR repositories, but it ultimately pushes the image to a private ECR repository (`cdk-hnb659fds-container-assets-454674044397-us-east-1`).



This is my pyproject.toml located in the same directory as the as the build_docker.sh file which is one level below the directory where the Dockerfile is located: [build-system]

```
requires = ["poetry-core>=1.6.1"]
```

```
build-backend = "poetry.core.masonry.api"
```

```
[tool.poetry]
```

```
name = "CDS GenAI front-end"
```

```
version = "0.1.0"
```

```
description = "Front-end of CDS GenAI application"
```

```
authors = [
```

```
    "Tristan Nguyen <nnatri@amazon.com>",
```

```
    "Philipp Kaendl <philikai@amazon.de>",
```

```
]
```

```
[tool.poetry.dependencies]
```

```
python = "3.9.19"
```

```
boto3 = "^1.33.0"
```

```
langchain = "0.2.7"
```

```
langchain-community = "0.2.7"
```

```
streamlit = "1.36.0"
```

```
streamlit-chat = "~0.1.1"
```

```
st-pages = "0.5.0"
```

```
streamlit-extras = "0.4.3"
```

```
watchdog = "~3.0.0"
```

```
pyjwt = "~2.7.0"
```

```
python-dotenv = "1.0.1"
```

```
qrcode = "^7.4.2"
```

```
aiohttp = "^3.9.4"
```

```
gitpython = "^3.1.40"
```

```
pillow = "^10.3.0"
```

```
pyarrow = "^14.0.1"
```

```
tornado = "^6.4.1"
```

```
urllib3 = "1.26.18"
```

```
s3fs = "^2023.12.2"
```



It looks like your project uses **Poetry** for dependency management, and the `pyproject.toml` file defines the necessary dependencies and metadata for your project. Since the `pyproject.toml` is located one level below where the `Dockerfile` is, you'll need to ensure the Docker build process can access it correctly. Here's a breakdown of your `pyproject.toml`:

Sections Breakdown:

1. [build-system]:

- Specifies the **Poetry** version used to manage the project build process.
- Uses `poetry-core>=1.6.1` as the required dependency for building the project.

2. [tool.poetry]:

- Provides basic project metadata:
 - `name`: The project name (`CDS GenAI front-end`).
 - `version`: The version of the project (`0.1.0`).
 - `description`: Short description of the project (front-end for a CDS GenAI application).
 - `authors`: List of project authors with their email addresses.

3. [tool.poetry.dependencies]:

- Lists the Python version (`3.9.19`) and other dependencies required for your project:
 - **boto3**: AWS SDK for Python.
 - **langchain** and **langchain-community**: Libraries related to AI/LLM integrations.
 - **streamlit** and **streamlit-chat**: Streamlit and its chat module for building interactive applications.
 - **st-pages**: Extension for Streamlit apps with multi-page functionality.
 - **streamlit-extras**: Additional Streamlit utilities.
 - **watchdog**: Library for monitoring file system events.
 - **pyjwt**: For JSON Web Tokens handling.
 - **python-dotenv**: Loads environment variables from a `.env` file.
 - **qrcode**: Library to generate QR codes.
 - **aiohttp**: Asynchronous HTTP client/server framework.
 - **gitpython**: Git integration with Python.
 - **pillow**: Python Imaging Library for image processing.
 - **pyarrow**: Apache Arrow integration for memory-efficient columnar data processing.
 - **tornado**: Asynchronous web framework.
 - **urllib3**: HTTP client for Python.
 - **s3fs**: File system for S3 buckets.

Integration with Docker:

To build this Docker image using the `pyproject.toml`, you'll need to modify the `COPY` commands in your Dockerfile to reflect the relative location of the `pyproject.toml` file and the required files. For example, assuming your directory structure is something like this:

markdown

```
project/
├── Dockerfile
├── subdir/
│   ├── pyproject.toml
│   └── build_docker.sh
```

Updating the `Dockerfile`:

1. **COPY pyproject.toml to the correct location:** You can modify your Dockerfile to include the relative path to the `pyproject.toml` file.
2. **Example of Dockerfile update:** Update your Dockerfile's `COPY` commands to reflect the actual location of the `pyproject.toml`:

Dockerfile

```
COPY subdir/pyproject.toml /app
```

With this change, the `pyproject.toml` will be copied correctly into the Docker build context. Would you like assistance in updating your Dockerfile to reflect these changes?



This is my pyproject.toml file at the top level of the project which is a level above where the Dockerfile is located: [build-system]
requires = ["poetry-core>=1.6.1"]
build-backend = "poetry.core.masonry.api"

```
[tool.poetry]
name = "CDS GenAI"
version = "0.1.0"
description = "Advanced Marketer Portal powered by Generative AI"
authors = [
    "Tristan Nguyen <nnatri@amazon.com>",
    "Philipp Kaindl <philikai@amazon.de>",
]
```

```
[tool.poetry.dependencies]
python = "~3.9.0"
aws-cdk-lib = "~2.70.0"
constructs = "^10.0.0"
boto3 = "~1.27.0"
pyyaml = "^6.0.0"
sagemaker = "^2.169.0"
aws-cdk-aws-apigatewayv2-alpha = "^2.0"
aws-cdk-aws-apigatewayv2-integrations-alpha = "^2.0.0"
aws-cdk-aws-apigatewayv2-authorizers-alpha = "^2.0.0"
aiohttp = "^3.9.4"
```

```
[tool.poetry.group.dev.dependencies]
pre-commit = "^3.3.3"
black = "^24.3.0"
ruff = "^0.0.282"
isort = "^5.12.0"
```

```
[tool.poetry.group.test.dependencies]
```

```

pytest = "^6.0.0"

[tool.isort]
profile = "black"
multi_line_output = 3
include_trailing_comma = true
force_grid_wrap = 0
use_parentheses = true
ensure_newline_before_comments = true
line_length = 120
skip_gitignore = true

[tool.ruff]
select = ["E", "F", "C", "W", "R", "B", "Q"]
line-length = 120
target-version="py39"
exclude = [
    ".bzzr",
    ".direnv",
    ".eggs",
    ".git",
    ".hg",
    ".mypy_cache",
    ".nox",
    ".pants.d",
    ".pytype",
    ".ruff_cache",
    ".svn",
    ".tox",
    "__pypackages__",
    "_build",
    "build",
    "dist",
    "venv",
    ".venv",
]

[tool.black]
target-version = ['py39']
line-length = 120

```



Your `pyproject.toml` file defines the project setup and dependencies for the "CDS GenAI" project. Since it's located at the top level of the project, and the `Dockerfile` is located one level below, you will need to update your Dockerfile and any relevant commands to ensure that the build process can correctly access the `pyproject.toml` file and other dependencies. Here's a detailed breakdown of the key sections in your `pyproject.toml`:

Sections Breakdown:

1. [build-system]:

- Specifies that the project uses `poetry-core` version `>=1.6.1` as a build system.

2. [tool.poetry]:

- Defines the project's name (`CDS GenAI`), version (`0.1.0`), description, and authors.

3. [tool.poetry.dependencies]:

- Lists the dependencies for your project:
 - **python**: Version `~3.9.0` (compatible with Python 3.9.x).
 - **aws-cdk-lib**: AWS CDK library (version `~2.70.0`).
 - **constructs**: Version `^10.0.0` for creating CDK constructs.
 - **boto3**: AWS SDK for Python.
 - **pyyaml**: YAML parsing and writing.
 - **sagemaker**: AWS SageMaker SDK.
 - **aiohttp**: Asynchronous HTTP client.

4. [tool.poetry.group.dev.dependencies]:

- Lists development dependencies, such as:
 - **pre-commit**: For managing pre-commit hooks.
 - **black**: Python code formatter.
 - **ruff**: Linter.
 - **isort**: For sorting imports.

5. [tool.poetry.group.test.dependencies]:

- Specifies **pytest** for testing.

6. [tool.isort], [tool.ruff], and [tool.black]:

- Configurations for code formatting and linting tools (`isort`, `ruff`, `black`), with rules such as line length (`120`), Python version targeting (`py39`), and exclusion patterns.

Dockerfile Considerations:

Since the `pyproject.toml` is located at the top level of the project and the `Dockerfile` is one level below, you will need to update your Dockerfile to reference the correct paths. Here's how you can adjust your Dockerfile:

1. Modify the `COPY` commands to reference the correct paths:

- The `pyproject.toml` file and other dependencies need to be copied into the Docker container. Given the structure, you will reference the top-level directory from within the `Dockerfile`.

2. Example of Dockerfile adjustments:

Dockerfile

```
FROM --platform=linux/amd64 python:3.9-slim
WORKDIR /app

RUN apt-get update -y && apt-get install -y --no-install-recommends \
    build-essential \
    curl \
    software-properties-common \
    git \
    && rm -rf /var/lib/apt/lists/*

# Copy the pyproject.toml and poetry.lock files from one level up
COPY ../pyproject.toml /app
COPY ../poetry.lock /app # If you have a poetry.lock file

# Install Poetry
```



```
RUN pip3 --no-cache-dir install -U pip
RUN pip3 --no-cache-dir install poetry

# Install dependencies without creating a virtual environment
RUN poetry config virtualenvs.create false
RUN poetry install --only main

# Copy the rest of the application files
COPY ../src /app/src

EXPOSE 8501

ENTRYPOINT ["streamlit", "run", "src/Home.py", "--server.port=8501", "--server.address=0.0.0.0"]
```

In this updated Dockerfile:

- `COPY ../pyproject.toml /app` and `COPY ../poetry.lock /app`` are used to copy the project configuration files from the top-level directory (`..``).
- `poetry install --only main`` installs the production dependencies based on `pyproject.toml``.

`build_docker.sh` Adjustments:`

Your `build_docker.sh`` script might need adjustments as well, ensuring it properly reflects the directory structure when the Docker build command is issued.

```
bash
```

```
docker build -f path_to_Dockerfile/Dockerfile ..
```

This command ensures Docker is aware of the top-level directory context for copying `pyproject.toml``. Let me know if you'd like assistance with other parts of the setup!



here is the directory structure: .

```
|— CODE_OF_CONDUCT.md
|— CONTRIBUTING.md
|— InstallIssues.xlsx
|— LICENSE
|— README.md
|— app.py
|— assets
|  |— Dockerfile
|  |— demo-data
|  |  |— df_interactions.csv
|  |  |— df_item_banking.csv
|  |  |— df_item_deduplicated.csv
|  |  |— df_item_deduplicated.csv.orig
|  |  |— df_segment_data.csv
|  |  |— df_segment_data.csv.orig
|  |  |— df_segment_data_v2.csv
|  |  |— df_segment_data_v3.csv
```

```

├── df_segment_data_v4.csv
├── df_users_deduplicated.csv
├── products.json
├── lambda
│   ├── bedrock_content_generation_lambda
│   │   ├── bedrock_content_generation_lambda.py
│   │   └── model_configs
│   │       ├── ai21.j2-grande-instruct.json
│   │       ├── ai21.j2-jumbo-instruct.json
│   │       ├── amazon.titan-tg1-large.json
│   │       ├── anthropic.claude-3-haiku-20240307-v1:0.json
│   │       ├── anthropic.claude-instant-v1.json
│   │       ├── anthropic.claude-v1.json
│   │       └── anthropic.claude-v2.json
│   ├── genai_personalize_batch_segment_job
│   │   └── personalize_batch_segment_job.py
│   ├── genai_personalize_batch_segment_jobs
│   │   └── personalize_batch_segment_jobs.py
│   ├── genai_pinpoint_job
│   │   └── pinpoint_job.py
│   ├── genai_pinpoint_message
│   │   └── pinpoint_message.py
│   ├── genai_pinpoint_message_custom
│   │   └── pinpoint_message_custom.py
│   ├── genai_pinpoint_message_email
│   │   └── pinpoint_message_email.py
│   ├── genai_pinpoint_message_sms
│   │   └── pinpoint_message_sms.py
│   ├── genai_pinpoint_segment
│   │   └── pinpoint_segment.py
│   ├── genai_s3
│   │   └── s3_fetch.py
├── layers
│   ├── factory_module
│   │   └── python
│   │       ├── c2m_add_credit.py
│   │       ├── c2m_check_job_status.py
│   │       ├── c2m_check_tracking.py
│   │       ├── c2m_create_job.py
│   │       ├── c2m_delete_address_lists.py
│   │       ├── c2m_submit_job.py
│   │       ├── c2m_upload_address_list.py
│   │       ├── c2m_upload_document.py
│   │       └── factory_module.py
│   ├── langchain
│   │   └── requirements.txt
│   └── utilities
│       ├── python
│       │   └── aws_helper.py
├── shared_module
│   └── python

```

```

├── _init_.py
├── channel_states.py
├── print_response.py
├── streamlit
│   ├── build_docker.sh
│   ├── poetry.lock
│   ├── pyproject.toml
│   └── src
│       ├── Home.py
│       ├── app_pages
│       │   ├── 00_Prompt_Smith.py
│       │   ├── 01_Segment_Architect.py
│       │   ├── 02_Content_Personalizer.py
│       │   └── 03_Message_Dispatcher.py
│       ├── components
│       │   ├── _init_.py
│       │   ├── authenticate.py
│       │   ├── genai_api.py
│       │   ├── model_specs.json
│       │   ├── personalize_api.py
│       │   ├── pinpoint_api.py
│       │   ├── utils.py
│       │   └── utils_models.py
│       └── images
│           ├── background.jpg
│           ├── banner2.webp
│           ├── click2mail-header-logo.webp
│           └── render.png
├── cdk.context.json
├── cdk.json
├── config.yml
├── images
│   ├── architecture.png
│   ├── cognito_create_user.png
│   ├── cognito_mfa_setup.png
│   ├── cognito_user_pool.png
│   ├── filter_product.png
│   ├── lambda_solution_ARN.png
│   ├── marketing_content_distribution.png
│   ├── marketing_content_generator.png
│   ├── personalize_datasetgroup.png
│   ├── personalize_import_done.png
│   ├── personalize_import_interaction_data.png
│   ├── personalize_solution_ARN.png
│   ├── personalize_solution_creation.png
│   ├── personalize_solutions.png
│   ├── prompt_engineering.png
│   ├── request_model_access.png
│   ├── user_segment_batch.png
│   └── user_segment_upload.png
└── infra

```

```

├── _init_.py
├── cdsgenai_stack.py
├── constructs
│   ├── _init_.py
│   ├── cdsai_api.py
│   ├── cdsai_endpoint.py
│   ├── cdsai_personalize.py
│   ├── cdsai_pinpoint.py
│   └── llm_endpoints
│       ├── _init_.py
│       ├── config_factory
│       │   ├── _init_.py
│       │   ├── base.py
│       │   ├── jumpstart.py
│       │   ├── marketplace.py
│       │   └── tgi.py
│       ├── constants.py
│       ├── iam.py
│       └── utils.py
├── stacks
│   ├── _init_.py
│   └── streamlit.py
├── poetry.lock
├── pyproject.toml
├── requirements-dev.txt
├── requirements.txt
├── resize.sh
├── samples
│   └── roles
│       ├── cross_account_bedrock
│       │   ├── policy_document.json
│       │   └── trust_relationships.json
├── scripts
│   ├── add_cognito_user.bsh
│   ├── get_digest.bsh
│   ├── get_digest_and_build.bsh
│   ├── replaceSingleToDoubleQuotesAllFiles.bsh
│   ├── segment-request.json
│   └── update_password.bsh
├── tempDebugging
│   ├── Python Version Compatibility Issues (1).pdf
│   ├── Python Version Compatibility Issues.pdf
│   ├── git_status.txt
│   └── outputSuccessfulBuildAndErrorBuild.txt
├── tests
│   ├── _init_.py
│   └── unit
│       └── _init_.py

```



Given your directory structure, the `Dockerfile` is located in the `assets` folder, and the `pyproject.toml` file is located at the top level of the project. To make sure the `Dockerfile` can correctly access the `pyproject.toml` file and other required dependencies during the build, you'll need to adjust the paths used in the `Dockerfile` and `build_docker.sh` script accordingly.

Dockerfile Adjustments

Since your `Dockerfile` is one level below the `pyproject.toml`, you need to adjust the `COPY` commands in your `Dockerfile` to refer to the correct relative paths.

Example Dockerfile Updates:

Dockerfile

```
FROM --platform=linux/amd64 python:3.9-slim

# Set the working directory to /app inside the container
WORKDIR /app

# Install system dependencies
RUN apt-get update -y && apt-get install -y --no-install-recommends \
    build-essential \
    curl \
    software-properties-common \
    git \
    && rm -rf /var/lib/apt/lists/*

# Copy pyproject.toml and poetry.lock from the root directory (one level up)
COPY ../pyproject.toml /app
COPY ../poetry.lock /app # If you have a poetry.lock file

# Install Poetry and dependencies
RUN pip3 --no-cache-dir install -U pip
RUN pip3 --no-cache-dir install poetry
RUN poetry config virtualenvs.create false
RUN poetry install --only main

# Copy the source code from the appropriate folder
COPY ../assets/streamlit/src /app/src

# Expose the port Streamlit runs on
EXPOSE 8501

# Define the entry point to run the Streamlit application
ENTRYPOINT ["streamlit", "run", "src/Home.py", "--server.port=8501", "--server.address=0.0.0.0"]
```

Explanation:

- Relative Paths:** The `COPY ../pyproject.toml /app` refers to the `pyproject.toml` file, which is one level up from the `Dockerfile` location (`assets` directory). The same logic applies to the `poetry.lock` file.
- Install Dependencies:** Installs Poetry and uses it to install the project dependencies from the `pyproject.toml`.
- Source Code:** The `COPY ../assets/streamlit/src /app/src` command copies the Streamlit application source code into the `/app/src` directory inside the container.

`build_docker.sh` Adjustments:

Since your `Dockerfile` is in the `assets` directory, you can modify your `build_docker.sh` to specify the correct path to the `Dockerfile` and ensure the build context is set to the top-level directory (where your `pyproject.toml` is located).

```
bash

#!/bin/bash

AWS_ACCOUNT_ID=$1
AWS_REGION='us-east-1'
IMAGE_TAG='v1.0' # Use versioned tags
ECR_REPOSITORY='cdk-hnb659fds-container-assets-454674044397-us-east-1'

# Log in to ECR
aws ecr-public get-login-password --region us-east-1 | docker login --username AWS --password-stdin public.ecr.aws/vla3q6c0

# Build the Docker image, using the Dockerfile located in the 'assets' folder and the top-level directory as context
docker build -f assets/Dockerfile -t $IMAGE_TAG .

# Tag the image for ECR
docker tag $IMAGE_TAG $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$ECR_REPOSITORY:$IMAGE_TAG

# Push the image to ECR
eval $(aws ecr get-login --no-include-email --region $AWS_REGION)
docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$ECR_REPOSITORY:$IMAGE_TAG
```

Key Changes:

1. **Context for Docker Build:** The `docker build -f assets/Dockerfile -t $IMAGE_TAG .` command specifies the Dockerfile's path (`assets/Dockerfile`) and uses the top-level directory (`.`) as the build context.
2. **Dockerfile Reference:** The script now points to the correct `Dockerfile` location.

Final Notes:

With these adjustments, your Docker build process will access the `pyproject.toml` file and source code correctly, and the Docker image will be built and pushed to ECR without issues related to file paths.



But there is also a `pyproject.toml` file in the directory below the Dockerfile



Memory updated

In your case, where there are **two** `pyproject.toml` files—one located **above** and one located **below** the `Dockerfile`—you will need to clarify which `pyproject.toml` file the Docker build should reference, as this can lead to confusion.

Handling Multiple `pyproject.toml` Files

- **Top-level** `pyproject.toml`: This one likely contains the broader application dependencies and configurations.
- **Sub-directory** `pyproject.toml`: This file might handle dependencies specific to that submodule or a particular feature (e.g., the Streamlit app or other microservices).

To ensure the correct behavior during the Docker build process, you need to decide which `pyproject.toml` should be used for installing dependencies within the Docker container.

Option 1: Use the Top-Level `pyproject.toml`

If the top-level `pyproject.toml` is meant to define the global dependencies for the project, you can stick with the earlier solution, which copies the **top-level** `pyproject.toml` into the Docker image.

Dockerfile

```
# Example for using top-level pyproject.toml
COPY ../pyproject.toml /app
COPY ../poetry.lock /app # If there is a poetry.lock

# Install dependencies using Poetry
RUN poetry install --only main
```

Option 2: Use the Sub-directory `pyproject.toml`

If the `pyproject.toml` located in the sub-directory is intended to define dependencies specific to the submodule (e.g., the Streamlit app), then you would modify your `Dockerfile` to copy that file instead.

Dockerfile

```
# Example for using sub-directory pyproject.toml
COPY streamlit/pyproject.toml /app
COPY streamlit/poetry.lock /app # If there is a poetry.lock

# Install dependencies using Poetry
RUN poetry install --only main
```

Option 3: Use Both `pyproject.toml` Files

If both files serve distinct purposes (e.g., the top-level `pyproject.toml` manages global dependencies, and the sub-directory's `pyproject.toml` manages app-specific ones), you might need a two-step process:

1. Install global dependencies from the top-level `pyproject.toml`.
2. Install app-specific dependencies from the sub-directory's `pyproject.toml`.

Example of using both:

Dockerfile

```
# Copy and install global dependencies from top-level pyproject.toml
COPY ../pyproject.toml /app
COPY ../poetry.lock /app
```

```
RUN poetry install --only main

# Copy and install app-specific dependencies from sub-directory pyproject.toml
COPY streamlit/pyproject.toml /app/streamlit
COPY streamlit/poetry.lock /app/streamlit # If there is a poetry.lock
WORKDIR /app/streamlit
RUN poetry install --only main
```

Final Considerations

- **Choose based on the scope:** If the `pyproject.toml` in the sub-directory is meant only for that particular app (e.g., Streamlit), use the sub-directory's file. If it's for the entire project, use the top-level one.
- **Avoid redundancy:** Installing dependencies twice could increase the Docker image size, so make sure dependencies are not repeated across both `pyproject.toml` files unless necessary.

Let me know if you'd like further clarification or adjustments!

ChatGPT can make mistakes. Check important info.