CI/CD Pipeline for a Continuously Learning ML Project

Objective

Create an SBERT and FAISS based search bar and ensure that any change in the training data, i.e feature store, or any change in the code, i.e resulting from hyper-parameter tuning or bert/faiss model upgrade, will trigger an end-to-end training job without disruption of the service already in production

Pre-requisites

- Setup EC2 Instance
- Setup docker and docker-compose
- search engine bar up and ready

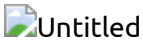
Prepare Codebase Github Repositories

Repos

- search bar repo: https://github.com/projectpro-product/sbert-search-bar.git
  - Production branch: `prod`
  - Staging branch: `stage`
- jenkins-server repo: https://github.com/kedardezyre/camille_projects.git
  - branch-name: `jenkinsserver`

Create personal access token

- Ensure that you are logged in github
- Go to https://github.com/settings/tokens
- Click on “Generate new token” and choose “classic”.

   Untitled
- Fill in some notes to describe your token
- Select the expiration length
- Check box on: *admin:org_hook, admin:public_key, admin:repo_hook, admin:ssh_signing_key, repo, user*
- Copy the token and save it in a safe place

```
# personal access token - expire on 2022-12-17
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
```

Deploy Search Bar App on EC2 Instance

Note:

- Instance type: t2.medium, ubuntu 22.04, 30GB
- Instance public IP: http://54.186.23.242/
- SSH in the instance

```
ssh -i "<path-to-pemfile.pem>" ubuntu@<instance-url>
```

- Create a folder for logs

```
mkdir logs
```

- Install apt packages: git, tree

```
sudo apt-get update
sudo apt-get install git tree
```

- Install conda

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-
x86_64.sh \
&& sudo mkdir /root/.conda \
&& bash Miniconda3-latest-Linux-x86_64.sh -b \
&& rm -f Miniconda3-latest-Linux-x86_64.sh
export PATH="/home/ubuntu/miniconda3/bin:${PATH}"
```

- Clone search bar repo and pull `repo` branch for most recent updates

```
git clone [https://github.com/projectpro-product/sbert-search-bar.git]
(https://github.com/projectpro-product/sbert-search-bar.git)
cd sbert-search-bar
git remote set-url origin https://<username>:<person-access-
token>@github.com/projectpro-product/sbert-search-bar.git
git fetch --all
git pull origin prod
```

- Create conda environment and install all requirements

```
conda create -y -n sbar-env python=3.10
source activate sbar-env
pip install -r requirements.txt
```

- Train search index and run the app in the background using nohup

```
python3 engine.py
nohup streamlit run app.py &>/dev/null &
```

Set up Jenkins Server on EC2 Instance

Note:

- Instance type: t2.large, ubuntu 22.04, 30GB
- Instance public IP: http://35.93.71.86/
- SSH in the instance

```
ssh -i "<path-to-pemfile.pem>" ubuntu@<instance-url>
```

- Install apt packages: git, tree

```
sudo apt-get update
sudo apt-get install git tree
```

- Install docker and docker-compose

```
# Installing Docker-ce
sudo apt update
sudo apt install apt-transport-https ca-certificates curl software-
properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" |
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
apt-cache policy docker-ce
sudo apt install docker-ce
```

```
sudo systemctl enable --now docker

# Adding docker-compose-plugin
sudo apt install python3-pip
pip install --upgrade pip
sudo pip install docker-compose
```

- Clone jenkins-server repo

```
git clone [https://github.com/kedardezyre/camille_projects.git]
(https://github.com/kedardezyre/camille_projects.git)
cd camille_projects
git remote set-url origin https://<username>:<person-access-
token>@[github.com/kedardezyre/camille_projects.git]
(https://github.com/kedardezyre/camille_projects.git)
git fetch jenkinsserver
git pull origin jenkinsserver
```

- Instance Jenkins on a docker container

```
cd jenkins
sudo docker-compose build
sudo docker-compose up -d
```

- Configure Jenkins server for first time
  - Copy one time log in pass

    ```
    sudo docker logs jenkins | less
    ```

  - Go to http://35.93.71.86:8080 and follow steps
- Install default plugins
- Install additional plugins
  - Github Pull Request Builder
  - publish-over-ssh
- Setup security tokens
  - Github personal access token

Jenkins Jobs

- update search bar
- data monitor job
- code monitor job

1. **Job 1: Update search bar**

    ProjectPro (3).png

   - SSH into Searchbar server, update the code or data and restart the app
   - Freestyle job
   - Trigger by either `code monitor job` or `data monitor job`
   - Bash Script

```
cd /home/ubuntu/sbert-search-bar
echo "------------BEGIN-------------">/home/ubuntu/logs/sbert-
search-bar.txt
echo "last job start: $(date)">>/home/ubuntu/logs/sbert-search-
bar.txt
export PATH="/home/ubuntu/miniconda3/bin:${PATH}"
source activate sbar-env
git checkout prod
git stash
git pull origin prod --rebase
pip3 install -r requirements.txt
python3 engine.py>>/home/ubuntu/logs/sbert-search-bar.txt
kill -9 $(pgrep streamlit) || echo "Process Not Found!"
nohup streamlit run app.py &>/dev/null &
echo "last job end: $(date)">>/home/ubuntu/logs/sbert-search-
bar.txt

echo "[info] pushing new dataset back to
github">>/home/ubuntu/logs/sbert-search-bar.txt
git add data/project_mappings.csv
git commit -m "[searchbar-server autocommit] training data
updated @ timestamp $(date +%s)"
git pull origin prod --rebase
git push origin prod
echo "------------END-------------">>/home/ubuntu/logs/sbert-
search-bar.txt
```

   - Demo how to setup update search bar job
     - Create password to log into the search bar VM
     - Configure Publish Over SSH Jenkins plugin
     - Create a freestyle project
     - Test

2. **Job 2: Data monitor**

- Runs periodically: Keeps checking whether the data currently in production is same as training data of the model currently in production
- Pipeline job
- Jenkinsfile

```
pipeline {
    agent any
    environment {
        PATH = "$WORKSPACE/miniconda/bin:$PATH"
    }
    stages {
        stage('Repos') {
            steps {
                // clean workspace
                cleanWs(
                    deleteDirs: true,
                    notFailBuild: true,
                    patterns: [[pattern: '*', type: 'INCLUDE']
                ])

                script {
                    // retrieve github personal access token from
jenkins parameters
                    def ghPersonalAccessToken =
params['GH_PERSONAL_ACCESS_TOKEN']

                    // checkout repo in prod branch
                    checkout([
                        $class: 'GitSCM',
                        branches: [[name: "origin/prod"]],
                        doGenerateSubmoduleConfigurations: false,
                        extensions: [],
                        submoduleCfg: [],
                        userRemoteConfigs: [[credentialsId: "github-
credential", url:
"https://kthouz:${ghPersonalAccessToken}@github.com/projectpro-
product/sbert-search-bar.git"]]
                    ])
                }
            }
        }
        stage("Setup ML Environment") {
            steps {
                //check if ml environment exists
                script {
                    sh '''#!/usr/bin/env bash
                    conda create -y -n sbar-data python=3.10
                    source activate sbar-data
                    pip install -r requirements.txt
                    '''
```

```
                }
            }
        }
        stage("Data Monitoring") {
            steps {
                // data monitoring will run every <T> minutes
                sh'''#!/usr/bin/env bash
                source activate sbar-data
                echo '[info] Running data monitor'
                echo '[info] Running data monitor'>log.txt
                python monitor_data.py>>log.txt
                '''
            }
        }
        stage("Update Search Bar Data") {
            steps {
                // Update data on search bar server
                script {
                    if (sh(returnStdout: true, script: 'cat log.txt |
tail -n 1').contains("DATA UPDATE REQUIRED")) {
                        echo "[info] Triggering update-searchbar job"
                        currentBuild.result = 'SUCCESS'
                        build job: 'update-searchbar', parameters: []
                    } else {
                        echo "[info] No changes in data - aborting"
                        currentBuild.result = 'ABORTED'
                    }
                }

            }
        }
        stage("Sync stage with prod") {
            steps {
                // sync stage with prod
                script {
                    if (sh(returnStdout: true, script: 'cat log.txt |
tail -n 1').contains("DATA UPDATE REQUIRED")) {
                        echo "[info] Syncing stage with prod"
                        sh '''#!/usr/bin/env bash
                        git stash
                        git checkout stage
                        git pull origin stage --rebase
                        git pull origin prod --rebase
                        git push origin stage
                        '''
                    } else {
                        echo "[info] No changes in data - aborting"
                        currentBuild.result = 'ABORTED'
                    }
                }
            }
        }
    }
}
```

- Demo how to setup data monitor as a pipeline job
    - Avoid concurrent builds
    - Link the pipeline to a Github project: https://github.com/projectpro-product/sbert-search-bar/
    - Define parameters:
        - GH_PERSONAL_ACCESS_TOKEN
    - Define build schedule
    - Link the pipeline to a jenkinsfile on Github repo
    - Test

1. **Job 3: Code monitor**

    - Triggered by PR: When there is code change, this will test and validate the code change as well as current data change against any negative impact on the search.index accuracy
    - Pipeline job
    - Triggered by new pull request or update on pull request
    - Jenkinsfile

```
pipeline {
    agent any
    environment {
        PATH = "$WORKSPACE/miniconda/bin:$PATH"
    }
    stages {
        stage('Repos') {
            // this will clean workspace and then checkout the
repo and merge the source branch into stage branch before running
the pipeline
            steps {
                cleanWs(
                    deleteDirs: true,
                    notFailBuild: true,
                    patterns: [[pattern: '*', type: 'INCLUDE']
                ])

                echo "[info] Pull Request author:
${env.ghprbPullAuthorLogin}"
                echo "[info] Pull Request Branch:
${env.ghprbSourceBranch}"
                echo "[info] Pull Request Target Branch:
${env.ghprbTargetBranch}"
                echo "[info] Current Branch: ${env.GIT_BRANCH}"
                echo "[info] Current Pull Request ID:
${env.ghprbPullId}"
                echo "[info] Current Pull Request Title:
${env.ghprbPullTitle}"
```

```
                script {
                    // retrieve github personal access token from
jenkins parameters
                    def ghPersonalAccessToken =
params['GH_PERSONAL_ACCESS_TOKEN']
                    // checkout prod branch
                    checkout([
                        $class: 'GitSCM',
                        branches: [[name: "origin/prod"]],
                        doGenerateSubmoduleConfigurations: false,
                        extensions: [],
                        submoduleCfg: [],
                        userRemoteConfigs: [[credentialsId:
"github-credentials", url:
"https://projectprodezyre:${ghPersonalAccessToken}@github.com/pro
jectpro-product/sbert-search-bar.git"]]
                    ])
                }


                sh '''#!/usr/bin/env bash
                echo '[info] Make backup of current prod data and
model'
                rm log.txt || echo "[info] log.txt not found --
skipping"
                rm -r temp_ml || echo "[info] temp_ml not found -
- skipping"
                mkdir temp_ml
                cp data/project_mappings.csv
temp_ml/project_mappings.csv
                cp output/search.index temp_ml/search.index
                '''

                script {
                    // retrieve github personal access token from
jenkins parameters
                    def ghPersonalAccessToken =
params['GH_PERSONAL_ACCESS_TOKEN']
                    // checkout branches and do necessary pre-
build merges
                    checkout([
                        $class: 'GitSCM',
                        branches: [[name:
"origin/${env.ghprbSourceBranch}"]],
                        doGenerateSubmoduleConfigurations: false,
                        extensions: [],
                        submoduleCfg: [],
                        userRemoteConfigs: [[credentialsId:
"github-credentials", url:
"https://projectprodezyre:${ghPersonalAccessToken}@github.com/pro
jectpro-product/sbert-search-bar.git"]]
                    ])
                    sh("git pull origin ${env.ghprbSourceBranch}
--rebase")
```

```
                    checkout([
                        $class: 'GitSCM',
                        branches: [[name: 'origin/stage']],
                        doGenerateSubmoduleConfigurations: false,
                        extensions: [],
                        submoduleCfg: [],
                        userRemoteConfigs: [[credentialsId:
"github-credentials", url:
"https://projectprodezyre:${ghPersonalAccessToken}@github.com/pro
jectpro-product/sbert-search-bar.git"]]
                    ])
                    sh("git checkout stage")
                    sh("git reset --hard HEAD")
                    sh("git pull origin stage --rebase")
                    sh("git merge origin/${env.ghprbSourceBranch}
--no-edit")
                    sh("git branch")
                }
            }
        }
        stage("Setup ML Environment") {
            steps {
                //check if ml environment exists and update it
                script {
                    sh '''#!/usr/bin/env bash
                    echo '[info] Setting up ML environment'
                    conda env list | grep sbar-code

                    conda create -y -n sbar-code python=3.10
                    source activate sbar-code
                    pip install -r requirements.txt
                    '''
                }
            }
        }
        stage("Functional Tests") {
            // this will run the functional unit tests
            // if the tests fail, the pipeline will stop
            steps {
                sh '''#!/usr/bin/env bash
                source activate sbar-code
                python -m unittest discover -s test -p "test*.py"
                '''
            }
        }
        stage("Build Search Index") {
            // this will build the search index
            steps {
                sh'''#!/usr/bin/env bash
                source activate sbar-code
                git branch

                echo '[info] Before running engine.py'>log.txt
```

```
                wc -l temp_ml/project_mappings.csv>>log.txt
                wc -l data/project_mappings.csv>>log.txt

                echo '[info] Running engine.py ...'
                python engine.py>>log.txt

                echo '[info] After running engine.py'>>log.txt
                wc -l temp_ml/project_mappings.csv>>log.txt
                wc -l data/project_mappings.csv>>log.txt
                '''
            }
        }
        stage("Push Changes to Stage") {
            // this will push the changes to stage if
            // there are any changes in the data or
            // changes in code base
            //
            // it will also create a PR to merge stage to prod
branch
            // the pull request will be merged automatically if
the

            steps {
            script {
                    // activate conda environment
                    sh '''#!/usr/bin/env bash
                    source activate sbar-code
                    python validate_changes.py >> log.txt
                    '''
                    // check if there are any changes in the data
                    if (sh(returnStdout: true, script: "cat
log.txt | tail -n 1").contains("CHANGE VALIDATION: SUCCESS")) {
                        // if there are changes, push the changes
to stage
                        echo "[info] Changes in data, merging to
stage"
                        sh("git add .")
                        sh("git commit -m \"[jenkins-server
autocommit] JenkinsMerge :: ${env.ghprbPullTitle}-
>#${env.ghprbPullId}\"")
                        sh("git push origin stage")
                    } else {
                        // if there are no changes, abort the
deployment
                        echo "[info] No changes in data or search
index broken by code change: aborting deployment"
                        currentBuild.result = 'ABORTED'
                    }
                }
            }
        }
        stage("Deploy Changes to Production") {
            steps {
                sh'''#!/usr/bin/env bash
```

```
                git checkout prod
                git pull origin prod --rebase
                git merge stage --no-edit
                git push origin prod

                echo "[info] ssh to the search bar server and
pull the latest changes"
                '''
                build job: 'job-update-searchbar', parameters: []
            }
        }
    }
}
```

- Demo how to setup data monitor as a pipeline job
    - Integrate Jenkins with Github via webhooks and Github Pull Request Builder
        - http://35.93.71.86:8080/ghprbhook
        - http://35.93.71.86:8080/github-webhook/
    - Avoid concurrent builds
    - Link the pipeline to a Github project:
        - https://github.com/projectpro-product/sbert-search-bar/
    - Define parameters:
        - GH_PERSONAL_ACCESS_TOKEN
    - Link the pipeline to a jenkinsfile on Github repo
        - `jenkinsfiles/code_monitor`
    - Test

Summary

Concluding Tests

1. Synthetic data
2. Code change
3. Check jenkins jobs
4. Check logs on searchbar app server
5. Check github merge reports