

Health.aiaudit.org - Documentation

1. Background and Goal

The evaluation platform is part of a larger project to develop an ML assessment platform. The platform is developed by [FG-AI4H](#) and consists of multiple work streams namely, data storage, data cataloging, reporting, annotation and our core package.

The platform is based on the open-source project [EvalAi](#) by CloudCV. EvalAi offers ML benchmarking in the form of challenges, offered by organizations for everyone to participate. Our goal is to adapt EvalAi's technology, and provide a model evaluation platform to be integrated into a larger project. A mockup of our project, including user stories and wireframes can be found [here](#).

Currently (05/21), we have deployed the production environment [here](#). We can already register new users, create text-based challenges and run evaluations. In total we have two instances running on AWS, one demo and the production mentioned above. Ideally we will deploy another staging for development in the next week and will discontinue the previous demo one.

The code for our project is on a private Github found [here](#). We have a master, staging and some sub branches, but mostly work on the staging branch. We also have a Kaban style [overview](#) of all our current tasks.

2. Our Team

- ★ Luis Oala: Project Lead
- ★ [Elora Schörverth](#): Development Lead
- ★ Steffen Vogler: Development
- ★ Alixandro Werneck: Project Management
- ★ Dominik Schneider: Development
- ★ Danny Xie-Lie: Evaluation Scripts

3. Evalai

3.1 Structure

EvalAi is a Django-based project and is deployed using Docker. The full documentation can be found [here](#). In general the structure of the project is the following:

```
├── apps
│   ├── challenges
│   ├── hosts
│   ├── participants
│   ├── jobs
│   ├── web
│   ├── accounts
│   └── base
├── docker
│   ├── dev
│   └── prod
├── frontend
│   └── src
│       ├── css
│       ├── fonts
│       ├── js
│       └── views
├── scripts
│   ├── deployment
│   │   ├── deploy
│   │   └── push
│   ├── migration
│   ├── tools
│   └── workers
├── settings
│   ├── common.py
│   ├── dev.py
│   ├── prod.py
│   └── test.py
├── docker-compose-production.yml
├── manage.py
└── README.md
```

Apps - contains all important Django applications. Referenced here are controllers, database structures as well as access points. When in doubt, familiarize yourself with [Django's setup](#).

Docker - contains Docker and environment files with all important variables and configurations. These containers are called using the appropriate docker-compose.yml file with [docker-compose](#). These are the project's docker containers:

- Celery: Queuing service for submissions
- Code-Upload-Worker
- Django: Our backend
- NodeJs: Our frontend
- Worker: Evaluation worker

Frontend - This folder contains the frontend (HTML, CSS, fonts, images etc.) It is Angular-based, but also uses Django's template language. It is important to highlight that the frontend also contains a lot of the business logic, so be mindful when you adapt it. Static and Media content is stored in S3 in AWS.

Scripts - Scripts provided by EvalAI to make your life easier (deploy to AWS, evaluation worker scripts)

Settings - Django Settings for the entire project, also depends on what environment you are using (development, staging or production)

3.2 Roles on the Platform

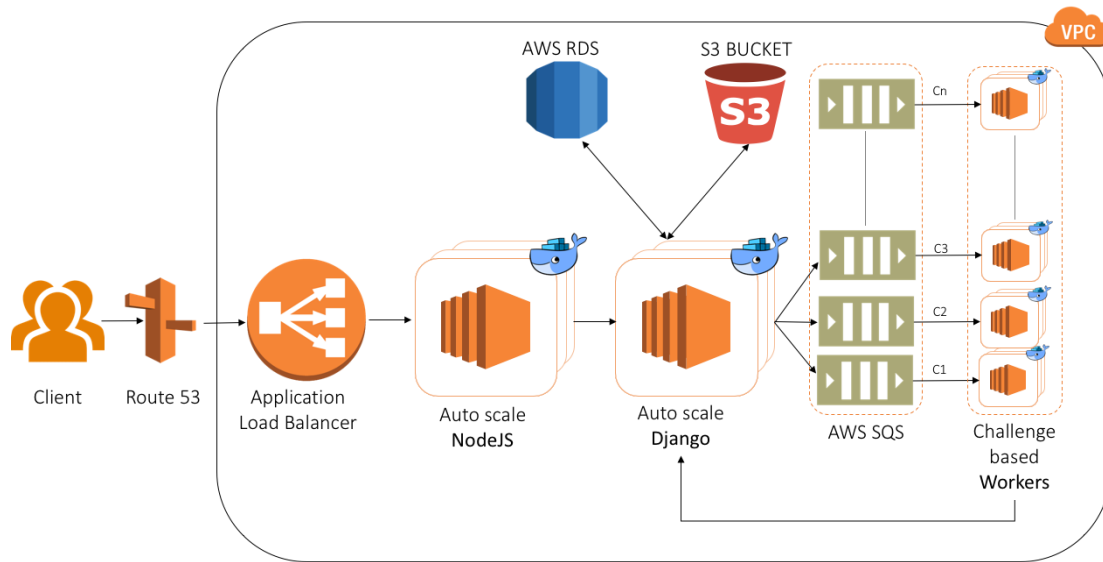
- Admin: someone who manages the evalai instance
- Host: someone who hosts and manages a challenges on an evalai instance
- User: someone who participates in a challenge and uploads preds/models to the platform

3.3 Evaluation structure

ML benchmarking is offered as two different version: text-based and docker-based submissions. Text-based means that the participant submits just their model's predictions on the provided test dataset. These predictions will then be compared to our ground truths and the general performance will be evaluated. Submissions can be made through the UI, as an upload or link to the file.

Docker-based means that the entire model will be submitted as a Docker image. Once uploaded, it will be stored in ECR on AWS and then be spun up using ECS. In an isolated environment it will be tested against our dataset and the results will be returned to the UI. Submissions can only be made through EvalAI's [command line tool](#). The command line tool

needs to be configured to our instance (default is evalai) as the upload target (See 8.4)



4. Our setup

Current level of development:

<https://github.com/Cloud-CV/EvalAI/compare/59e9961d7710abfb8781ebad5e0231d401f78378..2f408b7326d166150bd276ae125a92ef461f1411>

As mentioned previously, we have multiple EC2 instances running on AWS hosting our instances. To work on these, you need an AWS account and been granted admin access by us. Once you have the access, you can work on the instances using Cloud9.

Usually, the project should already be running, but the following commands are helpful during development:

Starting up the project: `sudo docker-compose -f docker-compose-production.yml up`
(Make sure to choose the appropriate .yml file)

Building the project: `sudo docker-compose -f docker-compose-production.yml build`
(Make sure to choose the appropriate .yml file)

Since the project is completely dockerized, it is important to mention that locally changing the code will not change anything. The changes need to be pushed into the corresponding docker container. In short, after you've changed anything, rebuild!

Rebuild single container: `... build <SERVICENAME>`

During development the Django admin panel is also an important tool. You can find it under health.aiaudit.org/api/admin. You will need the admin access password. From the admin panel you have a full overview of the data model, as well as all of the data stored.

5. Workstreams

- Frontend/Angular
- Challenge Onboarding
- Software Development (Debugging)
- Integrating Reporting Package and Model Zoo
- Evaluation Script development

6. Challenges

6.1 Challenge configuration

- Use [this repository](#) as a starting point

```
.
├─ README.md
├─ annotations # Contains the annotations for Dataset splits
│   ├── test_annotations_devsplit.json # Annotations of dev split
│   └─ test_annotations_testsplit.json # Annotations for test split
├─ challenge_data #Test the eval script locally
│   ├── challenge_1 # Evaluation script for the challenge
│   │   ├── __init__.py # Main.py file for evaluation
│   │   └─ main.py # Challenge evaluation script
│   └─ __init__.py # Imports the modules
├─ challenge_config.yaml #Define challenge setup
├─ evaluation_script # Contains the evaluation script
│   ├── __init__.py # Imports the modules
│   └─ main.py # Main `evaluate()` method
├─ logo.jpg # Logo image of the challenge
├─ submission.json # Sample submission file
├─ run.sh # create the configuration zip
├─ templates # HTML templates
│   ├── challenge_phase_1_description.html # Challenge Phase 1
│   │   description template
│   ├── challenge_phase_2_description.html # Challenge Phase 2
│   │   description template
│   ├── description.html # Description template
│   ├── evaluation_details.html
│   ├── submission_guidelines.html # Submission info
│   └─ terms_and_conditions.html # Terms and conditions
└─ worker # test eval script locally
```

```
|   |— __init__.py           # Imports the module
|   |— run.py               # Run the evaluation locally
```

For a **text-based challenge**:

1. Only the folders annotations, evaluation_script and templates are relevant for this type of challenge!
2. Edit HTML files under templates
3. Move ground truths in the annotations folder
4. Edit the evaluation_script to fit your needs (see below)
5. Edit the challenge_config.yml file. Set remote_evaluation and is_docker_based to false. Make sure to reference the correct annotation files. Define dataset splits and phases.
6. When fully configured, run the run.sh file. This will result in a ZIP file called “challenge_configuration”. Upload this file through the EvalAI UI.

For a **code-upload challenge**:

1. Edit HTML files under templates
2. Put the challenge data and annotations into the respective folders
3. Set up the docker images for environment and agent
4. Edit the evaluation_script to fit your needs (see below)
5. Edit the challenge_config.yml file. Set remote_evaluation to false and is_docker_based to true. Make sure to reference the correct annotation files. Define dataset splits and phases.
6. When fully configured, run the run.sh file. This will result in a ZIP file called

Congratulations! you have submitted your challenge configuration for review and EvalAI team has notified about this. EvalAI team will review and will approve the challenge.

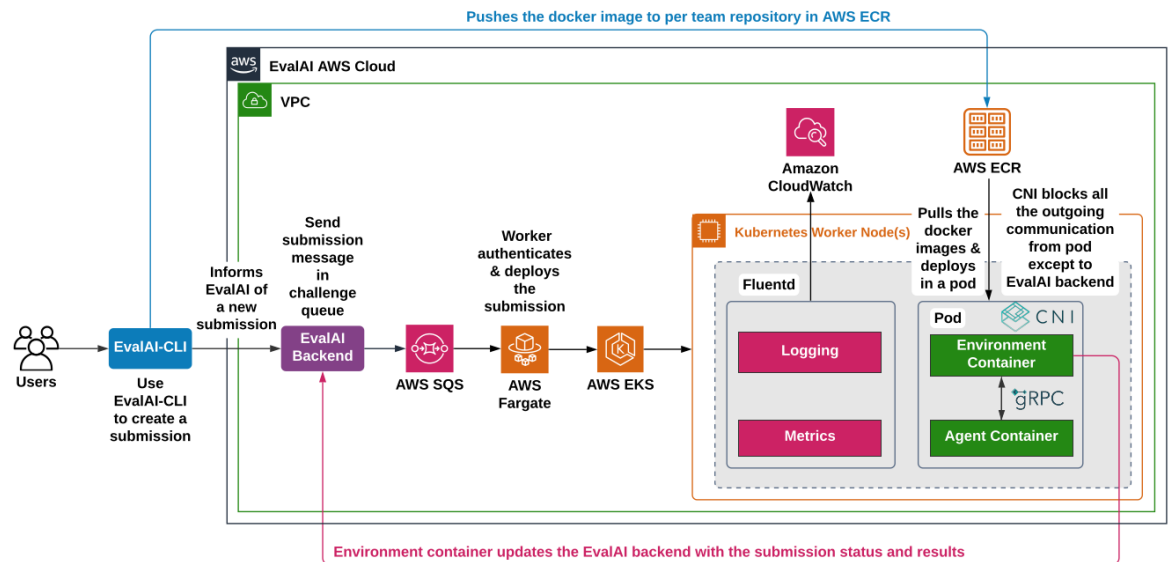
6.2 Evaluation Script

- Generally, decide which metrics and evaluations should be performed based on the modalities of your test case
1. Get the predictions for the test_annotations (ground truths) and the user_submission and compare on the basis of your designed metrics
 2. After calculation make sure to store the results in the corresponding file “submission_result” in the output file, while also clarifying the submission phase.
 3. If using additional libraries, install them beforehand using the provided __init__.py file

6.3 Code-upload evaluation (Docker)

- For full reference, see <https://smartech.gatech.edu/handle/1853/64704>

- Use Github to manage changes on docker-based evaluation. We will build changes and track any errors
1. Set up an environment container including a test environment, as well as an agents API.
 2. Upload docker image (of challenge) onto AWS ECR



3. Participant pushes docker image containing to model which triggers a new SQS message. Fargate deploys a worker who then picks up the submission. EKS deploys the submission on a node while agent image is pulled from ECR. Agent and environment containers communicate via gRPC protocol. After completion, the results are sent to EvalAi's database.

Submit to a challenge as a user

Prerequisites:

- Tagged docker image containing model is ready
 - EvalAI Cli installed
 - Host is set to local instance
 - `evalai host -sh http://localhost:8000`
1. Select Challenge and Participant Team on website
 2. Submit docker image using EvalAi Cli
 - a. `evalai login`
 - b. `evalai set_token <usr_token>`
 - c. `evalai push <image>:<tag> --phase <phase_name>`
 3. Upload only work in EvalAI production environment with AWS credentials set as environment variables in Docker

7. Work done so far

7.1 Deploy production environment

Setting up the production environment was one of the bigger tasks accomplished so far. Firstly, it was essential to fully configure the docker environment files. In those, we defined our AWS resources with access keys, email servers, databases and our evaluation clusters. Alongside that we also needed to configure the settings.py files (common.py and production.py). We also deployed the resources on AWS and needed to configure their access policies (using IAM).

7.2 Write a Retinopathy Evaluation Script

One of the models already developed in the context of this project is concerned with detecting diabetic retinopathy in patients by looking at imaging of the Retina. The model and others can be found [here](#). We have written a preliminary evaluation script that will test how well the created model performs against a provided dataset. As the model-based evaluation is not running yet, we have not tested it yet and will continue to adapt it in the future.

7.3 Running text-based challenge

We have managed to successfully set up the text-based evaluation pipeline and run dummy challenges. This means that we can upload finalized model predictions in the form of JSON or CSV files and compare them against our ground truths. The submissions can be made through the platform's UI where the results will also be displayed after submission.

7.4 Create Challenge questionnaire

7.5 Adapt Frontend and host project

As the goal is for this platform to be a separate project from EvalAi, we chose to remove all mentions of EvalAi from the frontend. In the future we also plan to fully change the frontend and change the layout of the site (insert Marc's mockup). This mostly included simple HTML and CSS changes. We needed to be mindful of changing the frontend though, as part of the business logic is included in it due to Angular. We also set up a domain including SSL on our production instance.

8. Various How To's:

8.1 Manage challenges (as admin)

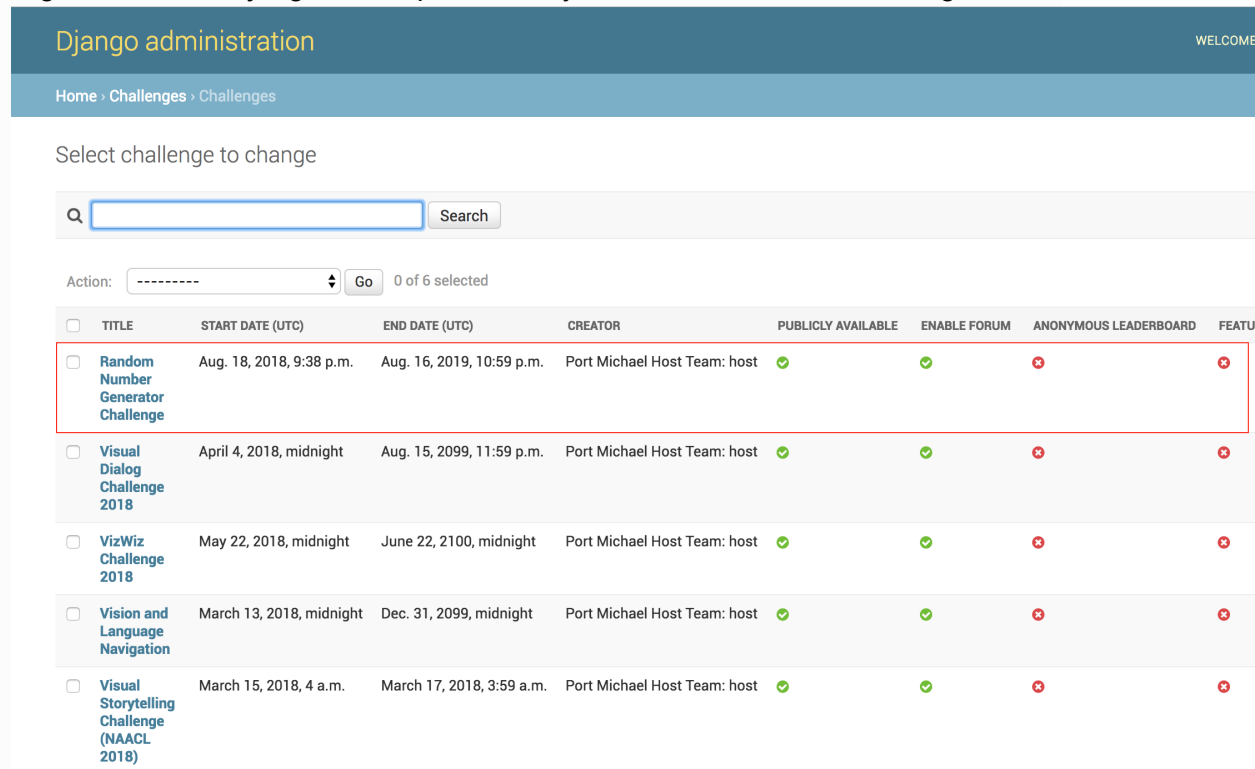
Reference: https://evalai.readthedocs.io/en/latest/approve_challenge.html

Once a challenge config has been uploaded, the challenge has to be approved by the EvalAI Admin (i.e you if you are setting up EvalAI yourself on your server) to make it available to everyone. Please follow the following steps to approve a challenge (if you are):

Let's assume that we want to approve a challenge with name Random Number Generator Challenge.

1. Step 1: Approve challenge using django admin

a. Login to EvalAI's django admin panel, and you will see the list of challenges



The screenshot shows the Django administration interface for EvalAI challenges. The header is "Django administration" with a "WELCOME" message. The breadcrumb trail is "Home > Challenges > Challenges". Below the header, there is a search bar and a table of challenges. The table has columns: TITLE, START DATE (UTC), END DATE (UTC), CREATOR, PUBLICLY AVAILABLE, ENABLE FORUM, ANONYMOUS LEADERBOARD, and FEATU. The first row, "Random Number Generator Challenge", is highlighted with a red border. The other rows are "Visual Dialog Challenge 2018", "VizWiz Challenge 2018", "Vision and Language Navigation", and "Visual Storytelling Challenge (NAACL 2018)".

	TITLE	START DATE (UTC)	END DATE (UTC)	CREATOR	PUBLICLY AVAILABLE	ENABLE FORUM	ANONYMOUS LEADERBOARD	FEATU
<input type="checkbox"/>	Random Number Generator Challenge	Aug. 18, 2018, 9:38 p.m.	Aug. 16, 2019, 10:59 p.m.	Port Michael Host Team: host	✓	✓	✗	✗
<input type="checkbox"/>	Visual Dialog Challenge 2018	April 4, 2018, midnight	Aug. 15, 2019, 11:59 p.m.	Port Michael Host Team: host	✓	✓	✗	✗
<input type="checkbox"/>	VizWiz Challenge 2018	May 22, 2018, midnight	June 22, 2100, midnight	Port Michael Host Team: host	✓	✓	✗	✗
<input type="checkbox"/>	Vision and Language Navigation	March 13, 2018, midnight	Dec. 31, 2019, midnight	Port Michael Host Team: host	✓	✓	✗	✗
<input type="checkbox"/>	Visual Storytelling Challenge (NAACL 2018)	March 15, 2018, 4 a.m.	March 17, 2018, 3:59 a.m.	Port Michael Host Team: host	✓	✓	✗	✗

b. Click on the challenge that you want to approve and scroll to bottom to check the following two fields.

- Approved By Admin
- Publically Available

c. Now, save the challenge. The challenge has been successfully approved by the administrator and is also publicly visible to the users.

2. Step 2: Reload submission worker

a. Since you have just approved the challenge, the submission worker has to be reloaded so that it can fetch the evaluation script and other related files for your

challenge from the database. Now reload the submission worker using the following command:

- b. Run the following command:
 - i. `docker-compose restart worker`
- c. **Submission worker has been successfully reloaded!**
- d. Now, the challenge is ready to accept submissions from participants.

8.2 Extend the Data Model in Django

1. Create a new model in the correct app
 - a. Example would be in the challenge app
2. Define all the relevant fields
3. Create a Serializer to instantiate that model
4. Change in to the Django Admin view (localhost:8000/admin) and see if the new model appears

9. AWS Resources

The following resources are needing for the project and have already been set up:

- EC2
- CloudWatch
- S3 and RDS
- ECS and Fargate
- SQS
- IAM
- EKS

When using any AWS service, please ensure that you are located in eu-central-1, as this is our default location.

10. Links and Q&A

a. Contact

If you have any questions, please contact elora.schoerverth@hhi.fraunhofer.de.
We have our weekly meeting at 4pm CEST [here](#).

b. Project Management

- i. [Slack](#)
- ii. [Azure](#)
- iii. [Github](#)
- iv.