Challenge 01 Notes

# 1. Objective

* Serverless/Kubernetes architecture to support a data sciences workload exposed via API to a front-end application. Design a serverless architecture to support the deployment of a Python-based AI content pipeline with a front-end web application (and authentication)

# 2. Source codes

* Repos:
  + <https://github.com/gilo-agilo/innovation-hub-2022-challenge-01>
  + <https://github.com/gilo-agilo/innovation-hub-2022-challenge-content-tagging>

# 3. Demos

4 demos recorded and published in the Box:

* <https://app.box.com/s/t44z608byvncc6oq1fwdf35jc3oq99oz>

# 4. Obtained results

1. AWS hosted a fully functioning solution (we bring cluster online on-demand for demo).
2. Solution is based on Kubernetes cluster (EKS)
3. Content (assets) are stored on a S3 bucket (to minimize the size of EKS pods and make them stateless)
4. Implemented ML API (Python) to provide access to Data Science model (brains)
5. 1-click deployment of the entire solution by script using AWS CLI
6. Frontend part is built on Angular (used Material UI and Highcharts.js for visualize charts)
7. Backend part is built on Python (ML libraries + flask web parts)
8. Auxiliary lambda functions are built on .NET Core
9. AWS Cognito provides the ability for app to authenticate and authorize users with OAuth and SAML 2.0(Okta federated identity)

**NOTE:** ML pipeline (the training) is outside of the scope for this challenge.

# 5. Feedback

Here we list incoming feedback requests and how the team responded to those requests:

|  |  |  |
| --- | --- | --- |
| **#** | **Request** | **Response** |
| 1 | Use different ML models to validate the applicability of architecture | The team used 2 models:   1. Text-based – Miles Per Gallon 2. Image-based – content tagging |
| 2 | Add authentication | The team used AWS Cognito and implemented:   1. Signup process 2. SSO with Google, Facebook (B2C) 3. SSO with SAML2.0 integrated with the Okta Identity provider(B2B) 4. Added auth to API lambdas |
| 3 | Separate ML model from ElasticSearch to enable re-used in AWS OpenSearch (for SearchCenter challenge #04) | The team is refactored the solution to   1. Implement ML API consumed by the 3rd party (Team 04) 2. Extract configurations from code into configs |
| 4 | Create Image Searching tool | WIP The team has implemented the solution which finds similar images in DB based on some input image  Another task which may be solved is finding similar images by the input text |
|  |  |  |

# 6. Insights

In this section, we capture insights we found valuable as a result of a-ha and oh-shit discoveries:

|  |  |  |
| --- | --- | --- |
| **#** | **Problem** | **Solution(s)** |
| 1 | Each epoch of training the NN model was too slow | During training the NN model we’ve noticed that running each epoch was too slow. The model was trained on AWS Sagemaker and the images dataset is in S3 bucket. The Tensorflow Profile was used to identify what step was the most time consuming. It happened that the preprocessing step was most time consuming. To decrease the time the image transforming step was done once for the whole dataset and saved to the new bucket rather than to make transformation during training. The epoch became 3x faster. |
| 2 |  |  |
| 3 |  |  |
|  |  |  |

# 7. PoC to Enterprise Notes

Here we provide recommendations on possible nuances of this PoC on its path to Enterprise-Ready grade solution.

|  |  |  |
| --- | --- | --- |
| **#** | **Nuance** | **Recommendation(s)** |
| 1 | Stateless mindset for hosting | Try to avoid saving state inside main python code as much as possible. Use cloud as much as possible e.g. for storing files use AWS s3, for database try to use some database in the cloud. For reading data also good idea to put it somewhere out of the code e.g. s3. Idea to make AI code as small as possible and stateless. |
| 2 | Hosting path from simplest to more complex | Following path was done for hosting application (from easier to more complex):   * Use AWS lambda for hosting backend pure code. Very fast approach has limitations for package size. For most AI projects where one library like tensorflow can take about 500mb will not be an option * Use AWS lambda and docker image. The approach is still quite simple and allow to bypass limitation with package size * Use AWS EKS (k8s) for hosting code. This is sophisticated approach that allows to create all in one place: host databases, store data to files etc, so almost has no limitations. The biggest drawback of this approach is complexity, managing EKS cluster requires expertise   In summary it is good to follow recommendations from Stateless mindset for hosting and try to use Lambda functions as much as possible |
| 3 | How to choose the appropriate approach of Image Searching | There are multiple ways of how to construct Image Searching tool from the ML perspective.  The main idea is to present both the input image and images from DB as some embeddings and then to compare these embeddings by some metric. Needed images from DB would have the low distance between their embeddings and the embedding of the input image. In our case for embedding NN model we took a model which classify the image with classes we need and just took not the last layer but the previous one. It can be any NN model (ResNet, VGG, etc) which gives you a high accuracy for the classes you need.  Furthermore, in our case we set that we want to find similar images of cars based on their colors, models, backgrounds, etc. To make the model pay attention especially on these requests we’ve trained multiple embedding models (to classify each group of requests) and then combined them.  In our case it brought higher accuracy of found images then just using single NN model trained for all required classes. But if there would be some other requirements (not so diverse) for the images to search, it may be ok to train just a single model for all classes.  So, the main point here is to choose the best architecture of image classification model and took some of the last layers to make embeddings for images. |
|  | UI data visualization by Hightcharts.js | To visualize complex data on web application Hightcharts.js was chosen. It easy integrated to Angular app and allow discard unnecessary details with svg elements (compared with D3.js). Desirable is to prepare data for visualization on the backend side, since a lot of work with concatenations, data manipulation can be much slower.  The general principle that should be followed is to divide the UI elements into separate components, which will allow in the future to easily customize and expand the development. |