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
7. Backend part is built on Python (ML libraries + flask web parts)
8. Auxiliary lambda functions are built on .NET Core

**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 (B2C) 3. SSO with SAML (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 |
|  |  |  |
|  |  |  |

# 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 |  |  |
| 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 |  |  |
|  |  |  |