KAIBURR

Tasks can be accomplished in any order (but you have to complete either #1 or #2 to be able to complete #3 and #4). Each task will be considered complete if your source code is available on github and you can show a working application.

High Level Summary:

1. Perform as many tasks as possible.
2. Create a code repository in github and upload the code there once done.
3. Add Readme files for each task that clearly describes it.
4. Upload screen shots of input and output to the repo and add them to the Readme files.

Task 1. Java REST API example.

Implement an application in java which provides a REST API with endpoints for searching, creating and deleting “server” objects:

- GET servers. Should return all the servers if no parameters are passed. When server id is passed as a parameter - return a single server or 404 if there’s no such a server.
- PUT a server. The server object is passed as a json-encoded message body. Here’s an example:

```
{
 “name”: ”my centos”,
 “id”: “123”,
 “language”:”java”,
 “framework”:”django”
}
```

- DELETE a server. The parameter is a server ID.
- GET (find) servers by name. The parameter is a string. Must check if a server name contains this string and return one or more servers found. Return 404 if nothing is found.

“Server” objects should be stored in MongoDB database.

Be sure that you can show how your application responds to requests using postman, curl or any other HTTP client.

KAIBURR

Task 2. Swagger codegen.

Create the same REST API as in task #1, but use https://editor.swagger.io/ to create your API definition and generate the server code. Choose any java-based server or server framework, that you like. You can either use the online editor or generate the code manually, e.g. using this document: https://github.com/swagger-api/swagger-codegen/wiki/server-stub-generator-howto. Make sure that you can deploy/run the generated code. Once your stub is ready - implement the same functionality as described in task #1, but now in java.

Finally, be sure that you can show how your application responds to requests using postman, curl or any other HTTP client.

Task 3. Kubernetes.

Use the application that you created in task #1 or task #2. Create dockerfiles and build docker images. Create kubernetes yaml manifests for the application (at least a deployment and a service). It's ok to expose the application with a LoadBalancer or NodePort service or with an ingress. Spin up a single-node local Kubernetes cluster (Docker Desktop, Kind or Minikube) or use a managed cluster like EKS, AKS, GKE etc. Deploy MongoDB to the cluster (it's ok to use a community helm chart for this, any other approach is fine as well). Then deploy the application to the cluster by applying your manifests. The following requirements should be fulfilled:

- you can bring your application up by applying your yaml manifests
- mongodb is running in a separate pod
- the application should take mongo connection details from the environment variables
- the app endpoints should be available from your host machine
- a persistent volume should be used to store the MongoDB data. I.e., when you delete the MongoDB pod the records in the db should not disappear.

Task 4. WEB UI Forms.

Create a basic WEB UI frontend for an application that you created for #1 or #2 using any UI framework of your choice. You should be able to create, show and delete records from your UI.

Task 5. CI-CD Pipeline

Create a CI-CD pipeline for a sample application (built in task 1 and/or 4 above) using any CI-CD tool of your choice like Jenkins, Azure DevOps, Gitlab, Github Actions, AWS

CodePipeline or any other tool of your choice. Include a code build and a docker build step in your pipeline.

Task 6. Data Science example.

Perform a Text Classification on consumer complaint dataset (https://catalog.data.gov/dataset/consumer-complaint-database) into following categories.

0	Credit reporting, repair, or other
1	Debt collection
2	Consumer Loan
3	Mortgage

Steps to be followed -

1. Explanatory Data Analysis and Feature Engineering
2. Text Pre-Processing
3. Selection of Multi Classification model
4. Comparison of model performance
5. Model Evaluation
6. Prediction