Submission requirements (submission will be rejected if any of the below requirements is not met):

- The work should be done by the candidate himself, if we find similar works submitted by several candidates - all such candidates will be rejected. Don't share your work with other candidates, please!
- **Each task should be submitted into a separate repository on GitHub.**
- **Each task submission should be accompanied by one readme.md file located in the repository root with embedded screenshots (see the details below). No screenshots means that the task is not done. Doc, PDF, zip files are not accepted.**
- Screenshot requirements: each screenshot should include the current date/time and your name in any form. For the date/time - it's recommended to include your system date/time widget when making a screenshot. For the name - it can be a username in a shell/command prompt or you can modify your desktop to show your name or you can type in your name in an editor in a separate window. The goal is to show that you are not using screenshots made by someone else.
- AI usage: you can use AI models to complete the tasks, but if we see that the candidate's work is generated by an AI model and the candidate is unable to understand, run or modify the code - such candidates will be rejected. Please, don't submit code that you did not write and do not understand - we'll be checking this in an interview anyway.

General task requirements

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. The repository should have a proper structure - **for example, don't upload all the code as one single zip file or one text document**.
3. Add Readme.md files for each task that clearly describes it (how to compile, run or deploy the application, how to use the API or UI). Doc format is not accepted, please use markdown (.md) format which can be displayed by github properly.
4. Upload screenshots of input and output to the repo and add them to the Readme.md files. **Tasks without screenshots/readme are considered as NOT done.**

Copyright © 2025 Kaiburr LLC. All rights reserved.

5. In addition to screenshots you can record a short video where you present your work. The video is not mandatory but will be a plus.

Task 1. Java backend and REST API example.

Implement an application in Java which provides a REST API with endpoints for searching, creating, deleting and running “task” objects. Task objects represent shell commands that can be run in a kubernetes pod. Each “task” object should contain the following properties:

- id (task ID, String)
- name (task name, String)
- owner (task owner, String)
- command (shell command to be run, String)
- taskExecutions (List<TaskExecuton>)

  TaskExecution:
- startTime (execution start date/time, Date)
- endTime (execution end date/time, Date)
- output (command output, String)

Here’s a sample object encoded as JSON:

```
{
  “id”: “123”,
  “name”: ”Print Hello”,
  “owner”:”John Smith”,
  “command”: “echo Hello World again!”
  “taskExecutions”: [
        {
              “startTime”: “2023-04-21 15:51:42.276Z”,
              “endTime”: “2023-04-21 15:51:43.276Z”,
              “output”: “Hello World!”
        },
        {
              “startTime”: “2023-04-21 15:52:42.276Z”,
              “endTime”: “2023-04-21 15:52:43.276Z”,
              “output”: “Hello World again!”
        }
  ]
}
```

Copyright © 2025 Kaiburr LLC. All rights reserved.

Here's the list of endpoints:

- GET tasks. Should return all the "tasks" if no parameters are passed. When "task" id is passed as a parameter - return a single task or 404 if there's no such task.
- PUT a task. The "task" object is passed as a json-encoded message body. **Application should validate the command provided in the request - the command should not contain unsafe/malicious code**. Here's an example:

```
{
  "id": "123",
  "name": "Print Hello",
  "owner":"John Smith",
  "command": "echo Hello World!"
}
```

- DELETE a task. The parameter is a task ID.
- GET (find) tasks by name. The parameter is a string. Must check if a task name **contains** this string and return one or more tasks found. Return 404 if nothing is found.
- PUT a TaskExecution (by task ID). Execute a shell command. Store a TaskExecution object in the taskExecutions list of a corresponding task.

"Task" 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. Make screenshots with request/response.

Task 2. Kubernetes.

Use the application that you created in task #1. 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, Minikube, etc.) 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.

Copyright © 2025 Kaiburr LLC. All rights reserved.

NOTE: many candidates are trying to submit docker-compose files instead of kubernetes manifests. We only count this task as done if there's a proof that the application is deployed to kubernetes. Make screenshots of your kubectl commands, the list of running pods and a curl command showing that at least 1 application endpoint is available.

Here's the most important part of this task: modify the implementation of the "PUT a TaskExecution" application endpoint. Now instead of running a shell command locally it should programmatically create a new kubernetes pod (using kubernetes API) and run the provided command inside this pod. We recommend using a **busybox** image.

Task 3. WEB UI Forms.

Create a WEB UI frontend for an application that you created for #1 using the React 19 framework, typescript and Ant Design. You should be able to create, show, search, delete records, run commands, view command output from your UI.
Pay attention to usability and accessibility of your UI. It's going to be the major factor in assessing the frontend part of the task.

Task 4. CI-CD Pipeline

Create a CI-CD pipeline for a sample application (built in task 1 and/or 3 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 5. 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

Copyright © 2025 Kaiburr LLC. All rights reserved.

2. Text Pre-Processing
3. Selection of Multi Classification model
4. Comparison of model performance
5. Model Evaluation
6. Prediction

Copyright © 2025 Kaiburr LLC. All rights reserved.