`

DevOps Challenge

This exercise is to assess your technical proficiency with Software Engineering, DevOps, and Infrastructure tasks. There is no need to do all the exercises, but try to get as much done as you can, so we can get a good feel of your skillset. Don't be afraid to step outside your comfort zone and try something new.

If you have any questions, feel free to reach out to us.

Exercise

This exercise is split into several subtasks. We are curious to see where you feel most comfortable and where you struggle. The task should usually take 2-4 hours.

0. Create a public git repository for a source code

Create a public git repository for a source code. Use your favorite provider.

1. Create an API service

Create a simple web application that prints the request header, method, and body. Tech stack and language are your choices.

Example Test Case:

```
curl --header "Content-Type: application/json" --data '{"username":"xyz","password":"xyz"}'
http://${URL}:${PORT}/api
```

Response:

Welcome to our demo API, here are the details of your request:

Headers:

Content-Type: application/json

Method:

Get

Body:

{"username":"xyz", "password":"xyz"}

Bonus points:

- do your best to follow best practices for building cloud-native apps, i.e. The Twelve-Factor Apps, etc.
- Instrument the code with a Prometheus counter/gauge using Prometheus client libraries

2. Dockerize

Create a Dockerfile to build a docker image and automate the setup of your API service with Docker, so it can be run everywhere comfortably with one or two commands.

Bonus points:

- be sure to use best practices for creating Docker files: i.e., proper layer structure, multi-stage builds if needed, security practices for building/running containers.
- cover your Dockerfile with simple integration test using container-structure-test, or other alternatives.

3. CI/CD

The application should be integrated with CI/CD, regardless if it uses simple GitHub actions, or any other related technology to take the code from the repository till the deployment environment

4. Create a helm chart

Create a helm chart to package your service. Be sure the helm chart deploys everything required for operation service in a production-like environment: service, ingress, RBAC entities, etc. Include 3rd party dependencies if needed.

Bonus points:

- implement helm hooks for running migrations

- implement helm test

5. Deploy to Kubernetes

Use your favorite Kubernetes environment to deploy and verify the service works as expected

Bonus points:

- Deploy open-policy-agent into a cluster
- Implement a test policy for open-policy-agent that validates deployments are using non-default service Account and containers are not running from root user.

6. Documentation

- Create a README file with instructions describing how to build, deploy and test the service
- Provide a reasonable amount of documentation. Don't document every line of code/configuration but be sure to provide comments where they're required.
- Provide meaningful comments for git commits
- Leave #TODOs in case you need to take a shortcut to save time or when you're doing something in the way you wouldn't in a real situation.

7. Whatever you can think of

Do you have more ideas to optimize your workflow or automate deployment? Feel free to go wild and dazzle us with your solutions.

Results

Please send us only a public git repository address with your project.

Hints

- Don't have a Kubernetes cluster to test against? Look at - MiniKube, k3s / k3d, GKE (more realistic deployment, may cost something)
- Remember that all scripts and code should be runnable either on Linux or macOS
- You may use whatever source of information you need
- Be sure not to spend more than 4 hours on the project

Good luck!