Endocode Challenge

Hello and thank you for taking the time to do this challenge.

The goal of this challenge is to get a rough overview of your coding, testing, automation and documentation skills.

We have different levels that you can complete. Just see how far you can get and have fun.

Description

- Write an HTTP service in any language you think is appropriate. (i.e. python, go, rust).
- Create your own git repository for this challenge.
- Commit often! Instead of one final big commit, we would like to see small commits that build up to the result of your test, instead of one final commit with all the code.
- When you are ready, just send us the link of your repository with your solution.

Level 1

Create an HTTP service that complies with the following requirements:

1. Command-line flag for the listening port (defaults to 8080) and environment variable override.
2. Use mostly standard libraries.
3. Build entry point is a Makefile.
4. Three HTTP endpoints:
   - `/helloworld` –> returns “Hello Stranger”
   - `/helloworld?name=AlfredENeumann` (any filtered value) –> returns “Hello Alfred E Neumann” (camel-case gets cut by spaces)
   - `/versionz` –> returns a JSON with git hash and name of the project (needs to be compiled in)
5. A structured log is written to standard out with:
   - ISO date
   - HTTP status
   - Request
6. Write a readme file with usage examples.
7. Unit testing of all functionalities (flags, endpoints, etc.).
8. A production-ready Dockerfile, so the service can run safely inside a container.
9. Documentation where it makes sense.

Cool, we have an HTTP service running. Now we want to automate the process including testing of our changes. Let’s set up a CI/CD pipeline.

1. We want to use Jenkins for our CI/CD. Install Jenkins locally.
2. Use Jenkins to create a pipeline to build the HTTP service.
3. Create a stage/step in your pipeline for automated testing.
4. Save your Docker image somewhere you can retrieve it from later.

Level 2

Great! We have an HTTP service and a way of automatically creating it. Now let’s put it somewhere.

1. Install minikube or any other lightweight, locally installable, cluster manager you prefer.
2. Create the required Kubernetes files to deploy our HTTP service into the cluster.
3. Automate the previous step into your Jenkins pipeline.
4. Our HTTP server should now also have an appropriate shutdown sequence. Add a graceful shutdown to your container and pod definitions so it will shutdown according to the following requirements:
   1. Shutdown timeout 30 seconds.
   2. New requests are ignored.
   3. Pod/container is killed after 30 seconds no matter what.

Level 3

Heaven above. That is a lot already. If you still have an appetite, go ahead and make it even better. Let's automate it even more!

1. Create a Helm chart for our HTTP service.
2. Terraform the deployment of our HTTP service into our local cluster. Your Jenkinsfile should now deploy everything into your cluster using Terraform.

Level 4

You're the best. But we can of course do even more and make it all shinier still.

1. Create a general logging system for our cluster.
2. Make our logs graphically available.
3. Now separate the versionz endpoint into a different container in your cluster and make sure that requests are sent there and back to the HTTP service again (a small micro service).

Boss level

- Add query time in `json` on `versionz`
- `musl/alpine` build (or from `scratch` for Golang)
- API is defined using OpenAPI specifications
  - Generate server stubs from it
- Fuzz testing
- Linters
- Add anything you thing can improve our setup and you think is a good addition.

HAVE FUN, BE CREATIVE AND SURPRISE US!