Candidate evaluation

Python Robotics Engineer

Introduction

This document describes tasks devoted to evaluating candidates applying for the Python Robotics Engineer position at Gideon Brothers. The candidate is expected to independently solve the following problems, representing his/her expertise and skills adequate for the respective position. The evaluation consists of two steps, a) evaluation of the submitted solutions and b) one on one evaluation in the company HQ if the submitted solutions satisfy our criteria. Please check the Additional guidelines section for mode submission details.

Goal: Present python robotics engineering skills by proposing the solution for the given problems.

The execution environment assumes a Debian based Linux distro (e.g., Ubuntu) and latest Docker.

1. ROS (Robot Operating System) is a standard robotics engineering framework providing robotics engineers with a set of libraries and tools that help them build robotic applications. In a more complex (multi-component) robotics solutions, it is sometimes necessary to isolate these applications into separate virtual environments such as docker containers. Docker containers are running instances of docker images, which are built by the docker engine using custom Dockerfile. Dockerfile is a text document that contains all the commands for an image assembly.

a) Could you write a custom Dockerfile that could be used to build a docker image with ROS installed? A base image should be an official Ubuntu 18.04 docker image (ubuntu:bionic). Provide the custom Dockerfile.

b) How do you build an image from a Dockerfile? Once the image is built, how do you start and run a container? Provide the command for running a container in an accompanying markdown file.

c) Could you verify that ROS is installed by running a "roscore" command inside a docker container? Provide the output of the executed command in an accompanying markdown file.

2. For an easier docker container management, docker-compose is commonly used. Docker-compose makes use of a docker-compose file (docker-compose.yml, e.g.). This file describes how containers should be named, which command should be executed, which local directories to be mounted inside docker, etc.

a) Could you write a custom docker-compose file that can be used to run your container with a roscore command? Output of the roscore command should be streamed into a log file which should be accessible from the host's file system without entering a docker container. Provide the docker-compose file.

b) Once you have a complete docker-compose file, how do you run, remove and restart your container using docker-compose? Provide the commands for these use cases in an accompanying markdown file.

3. As mentioned in task 1, more complex robotics application have multiple isolated components. However, it is still crucial for these components to communicate in a real time between themselves and even with some components outside of the docker network.

a) Add two more containers to your docker-compose file; one should run a simple ROS publisher (producer) script, while the other should run a simple ROS listener (consumer) script. Both scripts should be written and run in Python 3.8 with the help of Type Hints (typing module) and using a Object-Oriented Paradigm. Same as in task 2, scripts should stream their output to their log files accessible via host's file system. These new containers should be dependent on the "roscore" container and need to be started after it. Along the scripts, provide docker-compose file along with the new Dockerfile(s) (if you modified the one from task 1, or made new ones). Text of the message that the publisher script is publishing on a topic should be taken from a YAML file that is mounted from the host's filesystem.

b) b) Create a markdown documentation for your solution and include one UML class diagram. Document the code with Python docstrings, sphinx style is preferable. Comment the code where necessary.

Tip: *You may use this [ROS tutorial]() as a guide on how to write the scripts, as well as all the code that is provided in them. There is no need to create ROS packages as described there, just use and run them as ordinary Python scripts.*

The first script sends the data over ROS communication protocol, and the other script waits for the data and echoes it continously. Having this working is a validation of your setup.

4. In majority of cases, robotics solutions also communicate with client's systems using other protocol than ROS. Also, logging some of the data in a database is common.

a) Modify the publisher script so the text of the messages that it publishes is set by the external PUT request, i.e. the solution should incorporate a simple web server that listens on 127.0.0.1:8080 and accepts PUT requests that change the message. You may use some existing Python web framework for this (e.g. flask, falcon, fastapi ...). You may test your solution via requests generated by the Postman tool. Initial text of the messages should still be taken from the YAML file – same as in task 3.

b) b) Modify the subscriber script so the messages that it recieves over ROS are also stored in a sqlite or a mongodb database (chose one). Make sure the data is persisted after docker containers are removed (docker-compose down operation). If there is a need for that, create a new separate docker container that will hold a database instance (in which case this new container should also be a part of docker-compose file) - but this is not necessary.

c) c) Create a markdown documentation for your solution and include one UML class diagram and one UML sequence diagram. Document the code with Python docstrings, sphinx style is preferable. Comment the code where necessary.

Additional guidelines

Please include additional instructions and comments in your documentation if necessary. Submit your solution as a private GitLab (https://gitlab.com) git repository that contains a set of Dockerfiles, docker-compose and markdown files with additional comments.

Tasks should be done in order, if possible. Each task should have its own git branch. When finished with a task, merge it to the master branch and tag the merge commit with that task number (e.g. "task-1"). Every subsequent task branch should be a fork of the master branch which, at that moment, contains the previous task implementation. Finally, when done with all the tasks, master branch should contain the implementation of the last task (task 4), while also having all the previous tasks tagged. Don't delete tasks branches once you merge them into master.

Share your solution repository with GitLab user karlo.grlic (karlo.grlic@gideonbros.ai), so your solution may be evaluated by the Gideon Brothers team.

General information

Deadline: The task is expected to be finished within **7 days.** When finished, a link to the repository should be sent to dorotea.kust@gideonbros.ai, and the candidate will be given feedback after an internal evaluation by the Gideon Brothers team.

Note: If a candidate encounters any difficulties in the assignment solving process, the one is expected to communicate them as quickly as possible.