Backend test

Thanks for applying and congratulations for reaching this point of the interview process!

In this stage we will have you write some code. We will evaluate your ability to develop something that's fairly common at Raft. We strive to make this test as close to our daily business as possible and guarantee that every backend engineer at Raft can complete this test.

The informal wording of this test is close to our daily business talk and reading comprehension is part of the test.

About this test

The code in the package

We are providing you with a base project that is very similar to the local setup we use for microservices. This includes all the tools you may need: docker infrastructure, image conversion tools, database migrations, database connectors, queues, autoreload, health checks and, of course, main.py.

Extract the contents, install Docker if you don't already have it installed, then run `docker-compose up` in the directory. You can find more information in the README.md file.

Our goal is to save you the set-up time, but also verify that you'd know your way within our infrastructure.

How long it should take

The test includes a prioritised task list. The test hints at a minimum and a suggested number of tasks to complete. Once you manage to start the base microservice, we ask you to dedicate at least one hour and preferably up to two hours - and above that feel free to spend any time you feel comfortable spending.

To help us calibrate the test for future applicants, please let us know how long you spent on it - this won't be used for grading purposes.

Sending back the code

We won't ask you for a link to Github or Gitlab. **Please, don't put this project on a public repository** - we routinely change the content of the test, but we don't want to run out of ideas too soon.

Simply zip the entire project and send it to the human reference at Raft.

If you want to send a local git repository, use: `git archive --format zip --outputYourName-backend.zip main` (from stackoverflow).

Setting expectations

We don't expect mature code. Even the simplest code gets refined over the years and we understand that you won't reach that level of stability in a couple hours of work. What we expect is a proof of concept that can work in most cases.

We care about **code quality and structure**. Spin up your favourite code linters, make sure it's easy to find where the functions are, that no stray files are included and that the spacing in the code is not random.

We care about **simplicity**. Since this is a proof of concept, it wouldn't grow further than the end of this exercise. There's no need to code this as if you had to extend it for the next 5 years.

We care about **modernity**. Not using `async` on a web server in year MMXXII is not acceptable.Likewise, try to use the latest libraries and modern coding paradigms.

We care about **communication**. If any part of the code that's not easily explained by patternsand naming, or you took some shortcut "hack", a comment would be welcome. If you changed any of the infrastructure, a mention in the README.md would be appreciated.

FAQ

- **Can I use Django? -** No, Django solutions tend to be largely boilerplate, which doesn't give us enough of an understanding of your skillset.
- **Can I use Go or Node.js? -** Only if you don't have any experience with Python. If you have any experience with Python we ask you to complete this test in Python.
- **Can I use libraries? -** Feel free to use any common library. Avoid using libraries that have small userbases. If your library solves the entire challenge, you shouldn't use it.
- **Are you making me write production code for free? -** No, this is a problem we have already solved and have a lot of experience with.

Story

Oh no! Our upload processing pipeline is a monolith and it started running out of memory. Our fellow engineers found that the image processing stage causes huge spikes in RAM usage for about 0.05% of the uploads. Whenever this happens, the operating system kills the Celery worker process. As a side effect, uploads being processed by the same worker are being stopped as well and need to be put back into the queue. Random uploads suddenly take a long time to process. The impact on the users is tangible and they started messaging our Customer Success team.

As a stopgap we have increased the RAM available by 400%, so nothing is on fire for the time being. Unfortunately, when the services auto-scale during peak hours we are more than tripling the cost of the system. We need a quick fix quite soon, even if not perfect.

The team decides we should solve this in two ways:

1. Reduce RAM usage by researching alternative tools for image processing, possibly increasing the processing speed as well.
2. Separate the image processing from the upload pipeline into a microservice, and make it callable via HTTP API. This will ensure we can maximise the usage of our RAM and isolate eventual crashes.

You're the freshest employee at Raft, and **you're asked to develop #2**, which will teachyou a lot about our infrastructure while not needing to worry about affecting the existing code.

Your squad suggests (but doesn't enforce) that you use Imagemagick with Ghostscript as per the old code, while they research how to reduce the RAM usage. They suggest using Celery for queuing the work to avoid draining the FastAPI threads.

You're given a base microservice project that includes most of the infrastructure and libraries you will need. What's next for you is to write a couple API endpoints and Celery tasks. Perhaps you might need a database to avoid frequent disk operations - that's included too.

Specifications

Functional

We are making an image converter that will be available as an API to another internal microservice ("the client").

- The input should be a `pdf`, `png` , `jpeg` and `jpg` via upload

- The stored result of the service must be one or multiple `png` under the resolution of3500x3500 (do not size up the images)
- In case of `pdf`, each page has to become a separate `png`
- The original uploaded files should be stored for retrieval
- The client wants to upload the files, but doesn't want to wait for the processing to complete (return HTTP code 200 after starting the processing)
- The client wants to be given an unique ID for the upload
- The client wants to request if the processing is done by submitting the unique ID
- The client wants to request the paths for the images and the respective image resolution by submitting the unique ID
- The client wants to request the paths for the original files by submitting the unique ID
- The order of pages in the output must be preserved in case of PDFs
- It should be possible to process multiple uploads in parallel
- It should be possible to upload the same filename multiple times but store the results separately

Task priority

We have a limited time, tops a couple of hours to develop the above - or less if you're low on time. Here's a list of what you should preferably implement before you run out of time.

Your squad believes we can reach around point #7 in a reasonable time, but we need to reach #4 at the very least. If you have spare time, you can further improve the solution with the rest of the tasks.

1. Converting a `png` or `jpeg` without a Celery pipeline
2. Returning the path to the resulting file via API
3. Returning the path of the original file via API
4. Running the conversion in a Celery pipeline
5. Converting a `pdf` and update the API to support listing the resulting `png` paths
6. Returning whether the conversion is complete via API (by using the filesystem)
7. Returning image resolution via API (by reading each image)
8. Optimise returning the state of an upload by using a database
9. Optimise returning image paths and resolutions by using a database

Necessary compromises for the test

To simplify the infrastructure, we have set up the project to have a `/scratch` directory that alsosynchronizes with your local disk in the `./scratch` directory in your project. You can use this directory to write the files.

In the real world we would be using containers/buckets in a cloud service or use Minio locally.