Technical Test v2

This is an ANZ test to be taken as an individual, please do not collaborate when completing the assessment.

Test 1

The following test will require you to do the following:

- Convert the current Dockerfile into a multistage build Dockerfile.
- Optimise Dockerfile for caching benefits.

There are many benefits to using multi-stage build techniques when creating Dockerfiles. One such benefit is mitigating security risks, this is accomplished because the attack surface size of the image can be greatly reduced when the image no longer contains unnecessary files, packages, or binaries. It can also enhance caching on layers in previous build steps that no longer need to be clustered in a single RUN statement for optimal layering because the image is discarded and only those artifacts necessary are kept.

The final multi-stage Dockerfile should only have the essential components. You should also consider optimisation for caching, and structure in your final solution.

The below Dockerfile in its current state does not compile - you will also need to debug this issue and consider how this can be resolved.

For all the files that will be required they can be cloned from the following repository - HERE

Original Dockerfile

Note: To find more information around the concept of multi-stage docker builds see the following link: https://docs.docker.com/develop/develop-images/multistage-build/

Dockerfile

```
FROM golang:alpine

ENV GO111MODULE=on

WORKDIR /app

ADD ./ /app

RUN apk update --no-cache

RUN apk add git

RUN go build -o golang-test .

ENTRYPOINT ["/app/golang-test"]

EXPOSE 8000
```

Test 2

For these tests please create a Github or Gitlab repository where we can review your source code and any configuration required for your project to execute. Please make sure the repository is public so it's viewable.

The following test will require you to do the following:

- Create a simple application which has a single "/version" endpoint.
- Containerise your application as a single deployable artefact, encapsulating all dependencies.
- Create a CI pipeline for your application

The application can be written in any programming language. We'd recommend using one the following: NodeJS or GoLang.

Please indicate your preferred programming language.

The application should be a simple, small, operable web-style API or service provider. It should implement the following:

- An endpoint which returns basic information about your application in JSON format which is dynamically generated; The following is expected:
    - Applications Version.
    - Last Commit SHA.
    - Description. (This can be hard-coded)

API Example Response

```
"myapplication": [
  {
    "version": "1.0",
    "lastcommitsha": "abc57858585",
    "description" : "pre-interview technical test"
  }
]
```

The application should have a CI pipeline that is executed when new code is committed and pushed, this pipeline should be comprehensive and cover aspects such as quality, and security; Travis or similar, for example.

Other things to consider as additions:

- Create unit tests and/or a test suite that validates your code.
- Describe or demonstrate any risks associated with your application/deployment.
- Describe your approach to versioning your application/deployment.
- Write a clear and understandable `README` which explains your application and its deployment steps.