kudos - a POC of a CI/CD pipeline (mainly CI for now)

This project is meant to test out the use of

- a container based dev environment
- a fully automated pipeline

The code itself is not so important (and not even finished to be honest).
There are both python and javascript elements fo the project.

The first step is to create a container which holds all of the right tools for coding and testing.
Many of these tools will python modules, as opposed to the python modules which will be used in the actual application code.

An example can be found in the containers/apline-toolbox folder. This container is loaded with: the right version of python as well as some linting software. You can this container locally while still using whatever code editor you want.

This one of the reasons that using the container based environment is useful.
As a team, everyone can agree on a set of tools.
Those tool can then be baked into a container image which everyone uses in common.
In their individual environment, team members can then use those modules which are necessary for actually creating the functionality which they want.
Another big advantage is that this same image can be used in the CI/CD pipeline.
The container image Dockerfile is included here (base-containers).

The most important document here, is, the somewhat hidden, .gitlab-ci.yml
This is file is the representation of the pipeline.
This file contains the instructions to run the various steps for CI.
Every pipline requires a base container to be used as the runner. You can use the same container, desribed above as the runner.
Different tools will work differently, but most work on the same principles.
I'm pretty sure they are all container based, so you just have to be aware of that.
Each tool is going to handle this environment somewhat differently.
So sometimes, some simple `pwd` and `ls -al` commands placed at the right point can show you what is going wrong.

This is where using a standard dev environment makes things easy.
By using this standard image as the basis for the CI pipeline,
one can simply commit code without wondering if the CI env has the right tools necessary

to test and build the code.
For the purpose of the POC, I have include some basic linting tools
for the python code as well as for the build config yml file.

Be careful in the distinction of the `toolkit` or `CI` container, which holds config and testing tools,
as opposed the container actually built to run the code.
The first should be highly standardized, but may be far larger than one would expect.
The latter should be as lean as possible, should only contain what it needs,
but may be quite non standard.
Finally the second should be built by the first (the `CD` is created by the `CI`).

The build folder holds the files used for this build.
I use packer and ansible, which seemed like the most advanced and standard tools for what I was trying to do.
I think this is definitely better than creating manual dockerfiles.

Challenges

For gitlab at least, context seems to be a **BIG** problem.
One of the requirements was to prove that we could store the SHA256 signature of a validated container so that it could be used later.
This proved to be very difficult and extremely clunky.

I used a plan B system using the aws sns service and storing files in s3 buckets.
It's probably better to rely on additional tools like Sonatype manage this kind of context.
This is where we see the clear distinction between CI and CD. The gitlab chain works for CI,
but it completely falls down on natively managing anything outside of itself ... so CD.

The default gitlab runner images don't have much, which is why each project should build up a specific image. This would imply muliple images to be managed, but advanced config of said images is only by a centralized config file, and honestly not fantastic. Having afterwards spend a very short period of time working with TeamCity, there are CI/CD tools vastly more mature than gitlab, but that was what I had to work with.

Next Steps

I didn't really get to much in terms of unit of system testing here,
but they should fit nicely into this toolchain. But the CI pipleine can spin up both the built container as well as any others in order to exectute system testing.