Lifecycle Management for Docker UI

Stephen Coady

September 14, 2016

Images

Primary Goals:

- List
- Push and Pull
- Delete
- Build from Dockerfile

Stretch Goals:

- Visual Representation of an image:
  - This would allow the user to see the history of an image
  - It could also graphically show the changes made at each layer of the image.

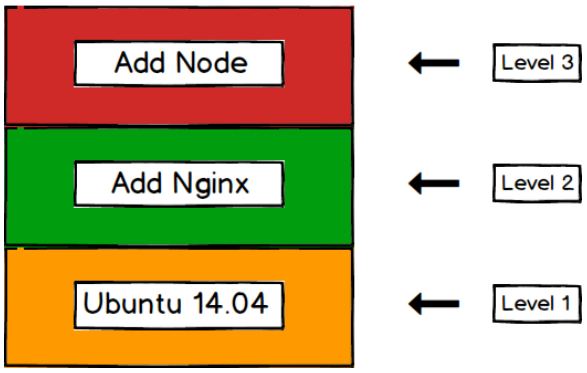

Figure 1: *Graphically Show what happens at each layer of the image*

- Get a tarfile of all images on a server
  - This would allow for quick 'snapshotting' of all images on a server
- Load a tarfile with images onto a server
  - The opposite of above, allow quick startup of containers, however a provisioning tool such as Ansible paired with Docker Compose may make this option impractical if a faster alternative already exists.
- Searching the Docker hub for images to use

Containers

Primary Goals:

- List
- Start/Stop
- Restart
- Remove

Stretch Goals:

- Sending files to a container.
  - I would envisage this being useful for quick provisioning or installation of a service. Either way, it is useful to be able to provide a container with files before it starts running so this gives that option.
- Output/viewing of a running container's log
  - From experience I know it can be useful to view the console output of a container. This can be cumbersome when the container is remote so providing this option could be something to look at. Where it may come in most handy is during development, when seeing what is happening under the hood of a container can be a great debugging tool.

Other Goals

Containerising the application:
This would be a very valuable end product. It makes running the application exponentially easier having it in a container. With the right development process - i.e. keeping this end goal in mind from the start and using the right tools, this will hopefully be a smooth process.

Security:

- App Level Authentication
  - On an app level, something like Passport may suit here. It can be used as a node module to authenticate against a given username and password. A simple version of how this could work is as follows: run the container → log in first time using dummy details → prompted to change login details.
- Remote Server Authentication
  - I have built a secure Docker registry which was used in production, so I am aware of how Docker uses certificate validation using client keys and certificates. It essentially means only connections from clients providing the correct certificate will be allowed.
  - One downside I see to this is that it is quite cumbersome. It is not ideal for the end user to need to set up and sign certificates for each server they wish to manage.

Statistics:

- Network Monitoring
  - Monitoring the network and which containers are linked could be an interesting topic to look at. Using something like vis.js we could visualise the containers running on a server and how they are connected.
- Container Load
  - Here I think it could be interesting to see how the actual server is performing due to the containers it is running. i.e how was the server running for the past 15 minutes, before I added another container that is network intensive?

Connecting to Multiple Servers:
This is one part of the proposal I am not certain about. One solution could be to run the app on each server it is needed on, then talk to these servers using a single point of login (on the user's host machine possibly).

This could consist of the user filling in the details of each server application and then the application authenticating with each server and the UI aggregating the result.

REST API:
For this project I would use either Express or HAPI to build the APIs required.

Express definitely has the larger community, which may prove beneficial, but using Express can sometimes be a tedious task whereas HAPI may do more 'out of the box'.

Further investigation is needed here, and the result will depend upon which of the above two will cater for the project requirements more efficiently.

Testing Strategy:
I would like to complete this project using a Test Driven Development Approach. I feel it suits node applications well, and after the initial outlay of writing tests, allows for rapid development which fits nicely into an agile development environment.

To that extent, I would also look at setting up a continuous integration cycle using something like Jenkins CI. I have previous experience using Jenkins and I feel it would fit well here.

Learning Outcomes

My ideal learning outcomes from this project are as follows:

- Gain a greater understanding of Docker. I feel this project could give me a greater appreciation of the possibilities Docker offers, especially since it is not just about building an application using Docker, but actually interacting with the Docker Engine itself.
- Working as part of an agile process with a dedicated product owner in the form of Red Hat and a scrum master in my college supervisor. This will be an invaluable experience.
- Development best practices. Having access to a professional developer will allow me to see the industry standard when it comes to both code and project management.