Docker GUI

Drusti Thakkar
Department of Computer Engineering
San Jose State University
San Jose, USA
drustihitesh.thakkar@sjsu.edu

Divjyot Singh Khanuja
Department of Computer Engineering
San Jose State University
San Jose, USA
divjyot.khanuja@sjsu.edu

Jasnoor Brar
Department of Computer Engineering
San Jose State University
San Jose, USA
jasnoor.brar@sjsu.edu

Sanjna Dhamejani
Department of Computer Engineering
San Jose State University
San Jose, USA
sanjna.dhamejani@sjsu.edu

***Abstract*—This paper describes a solution implemented for software developers who struggle with remembering multiple docker commands and want a user-friendly application to manage docker containers. Docker GUI provides its user with simple start, stop, restart and exec buttons with complete container logs displayed to maintain docker containers instead of using multiple commands via the terminal. The application is also integrated with docker hub through which a user can create containers from the images they have pushed to their own repository or also directly use images from dockers official repository.**

***Index Terms*—docker, container, images, docker hub**

I. Introduction

This document is a final report for our CMPE 272 Enterprise Software Platforms, Fall 2018 project as graduate students. We came up with this idea based on our discussion with software development students who had a tough time figuring out problems with their dockerised applications and also to curb the hassle faced by them to type multiple commands for basic functions such as start/stop containers.

II. Architecture Flow

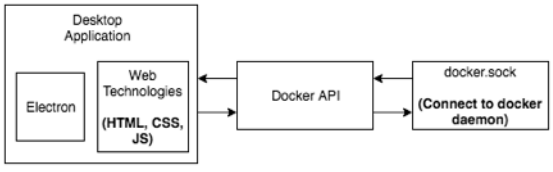

Fig. 1. Basic Application Flow

We have created a desktop application using web technologies such as HTML, CSS and JavaScript. The application is built on the electron framework which allows users to create desktop applications using frontend and backend technologies generally used to create web applications. We have used the docker socket file via Docker API to connect to the docker daemon which translates the docker commands.

III. Methodology

This section discusses the methodology used in developing this project. To start with, we began by analyzing the requirements for the project where we came up with a list of requirements to be implemented as independent features of the application. Post the analysis phase, we came up with a list of technology components which could be used to develop the application. To considerations made for the selection were:

1. Developing a user friendly application doesn't just mean to have integrated functionality in your application without a great user interface. Overall user experience plays a key role in making an application easy to use. To ensure this we chose the JavaScript framework React for our front-end development. One of the key benefits of react are that it provides a Virtual DOM which makes DOM manipulation a faster and smoother process.

2. In addition to the web technologies used, we chose Electron to develop a desktop application since it provides developers with binaries which can be added straight away as a dependency without worrying about any form of integration.

After we finalised the technology components, we started with the implementation phase. Here, we integrated the Docker API for Node provided by dockerode with our frontend application to connect to the docker daemon.

The final stage was the testing phase where we tested our application extensively by creating 20 containers at the same time which helped us analyze the system performance as well any changes reflecting on the user interface. We also tested the application by adding new official docker hub images to our JSON file to ensure it allows the user the create containers seamlessly. On completion of the quality check of the application, we deployed the application on cross platform machines for our final phase i.e. User Acceptance Testing.

IV. Application

A. Application Working

1. User accesses official docker hub repository to create containers.
2. User logs in to his personal docker hub repository to access docker images pushed on hub.
3. User views docker logs displayed on container homepage to analyse issues if any.
4. User starts/stops/restarts/exec into container shell using simple glyphicon buttons.

B. Application Features

1. Docker Hub Integration

Docker Hub Integration allows the user to access his personal docker hub repository and use docker images pushed on cloud from any machine.

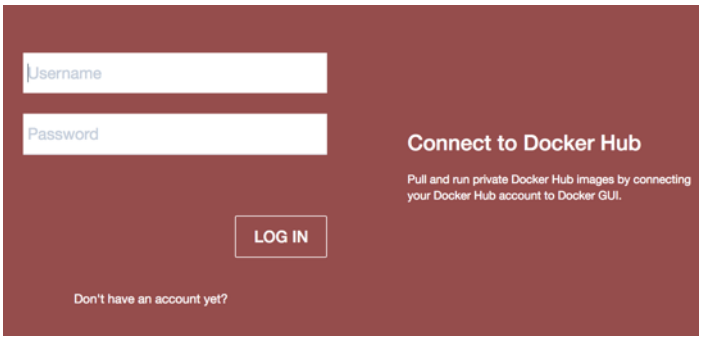

Fig. 2. Docker Hub Integration

2. Simple Buttons to manipulate containers

These buttons provide the user with a user-friendly way of manipulating the containers.

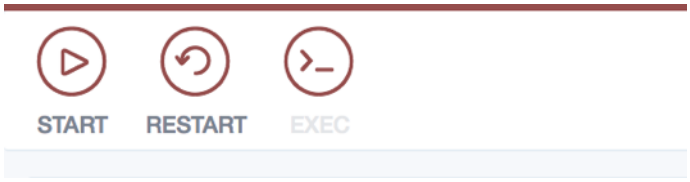

Fig. 3. Buttons to manipulate containers

3. Docker images

Homepage displays a list of docker images from the official docker hub repository and personal repository. User can simply click on the create button on any one the images to spawn a container with the respective application running on it.

Fig. 4. Docker images

4. Docker logs

Docker logs are continuously displayed on screen to provide the user with quick look at the issues/problems with the application if any.

Fig. 5. Docker logs

5. List of open ports for each container

This helps the user to understand the ports on which the application is running and directly use the access URL for a web preview.

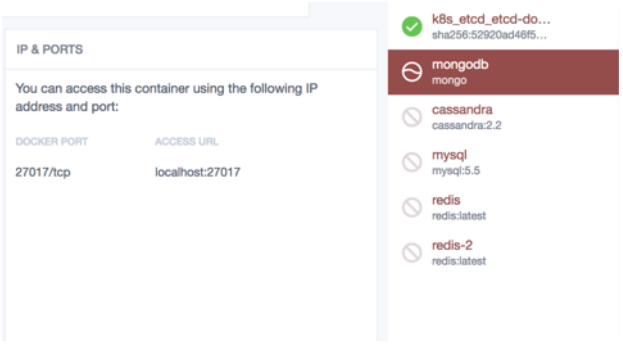

Fig. 6. List of open ports for each container

V. Technology Components

The technologies used to develop this application are HTML, CSS, JavaScript with flavours - React, NodeJs, ShellJs and Electron. Along with these, Docker API is used from the official docker documents to connect to the docker daemon and execute docker commands.

A. ShellJs

ShellJs is an implementation of Unix Shell commands built in integration with NodeJs. ShellJs can be used as a part of any project just by adding it as a project dependency. ShellJs is compatible with every Node version. It can be installed using the command:

```
npm install shelljs –save
```

ShellJS can also be used from outside JavaScript projects by installing it globally.

B. Electron

The biggest benefit of using Electron is that it eliminates the possibility of developing native desktop application for multiple platforms. Electron framework is used to develop cross-platform desktop applications build on web technologies like HTML, CSS and JavaScript. It cas be used as binaries in a project and installed as dependent modules. Electron is installed using the command:

```
npm install electron –save-dev [–save-exact]
```

Electron works by taking a main.js file defined in the package.json file and executes it to create an application window of web pages which in turn interacts with the native graphical user interface of the system OS.

The following code snippet can be used to spin up Electron from your application built using Node.

```
const electron = require('electron')
const proc = require('child_process')

// will print something similar to /Users/maf/.../Electron
console.log(electron)

// spawn Electron
const child = proc.spawn(electron)
```

Fig. 7. Code Snippet

C. Docker API

The docker API is provided by Docker to interact with the docker daemon. We have used the API from an unofficial community supported library available on the official docker website. Below are the few code snippets as examples for the docker API in Node JS.

- Instantiate dockerode
- Container manipulation

```
var Docker = require('dockerode');
var docker = new Docker({socketPath: '/var/run/docker.sock'});
var docker1 = new Docker(); //defaults to above if env variables are not used
var docker2 = new Docker({host: 'http://192.168.1.10', port: 3000});
var docker3 = new Docker({protocol:'http', host: '127.0.0.1', port: 3000});
var docker4 = new Docker({host: '127.0.0.1', port: 3000}); //defaults to http

//protocol http vs https is automatically detected
var docker5 = new Docker({
  host: '192.168.1.10',
  port: process.env.DOCKER_PORT || 2375,
  ca: fs.readFileSync('ca.pem'),
  cert: fs.readFileSync('cert.pem'),
  key: fs.readFileSync('key.pem'),
  version: 'v1.25' // required when Docker >= v1.13, https://docs.docker.com/engine/api/version-history/
});

var docker6 = new Docker({
  protocol: 'https', //you can enforce a protocol
  host: '192.168.1.10',
  port: process.env.DOCKER_PORT || 2375,
  ca: fs.readFileSync('ca.pem'),
  cert: fs.readFileSync('cert.pem'),
  key: fs.readFileSync('key.pem')
});

//using a different promise library (default is the native one)
var docker7 = new Docker({
  Promise: require('bluebird')
  //...
});
//...
```

Fig. 8. Instantiate dockerode

```
// create a container entity. does not query API
var container = docker.getContainer('71501a8ab0f8');

// query API for container info
container.inspect(function (err, data) {
  console.log(data);
});

container.start(function (err, data) {
  console.log(data);
});

container.remove(function (err, data) {
  console.log(data);
});

// promises are supported
docker.createContainer({
  Image: 'ubuntu',
  AttachStdin: false,
  AttachStdout: true,
  AttachStderr: true,
  Tty: true,
  Cmd: ['/bin/bash', '-c', 'tail -f /var/log/dmesg'],
  OpenStdin: false,
  StdinOnce: false
}).then(function(container) {
  return container.start();
}).then(function(container) {
  return container.resize({
    h: process.stdout.rows,
    w: process.stdout.columns
  });
}).then(function(container) {
  return container.stop();
}).then(function(container) {
  return container.remove();
}).then(function(data) {
  console.log('container removed');
}).catch(function(err) {
  console.log(err);
});
```

Fig. 9. Container manipulation

VI. Future Works

As per the suggestions from our Mentor, Prof. Rakesh Ranjan, we have come up with a list of ideas which could enhance this product.

1) Provide the user with an interface to select his requirements and create docker images on just a click.
2) Group docker containers according to the networks they reside in.
3) Integration with Heroku to allow the user to push his images and configure continuous deployment.

VII. Conclusions

This project has helped us dive deep into the intricate details of container technology. Today, when we talk about containerization, the first word that comes to our mind is Docker. Not only did we learn about this booming technology, but also we had the opportunity to try our hands on a new technology - Electron. With all the knowledge acquired during the course of this project, we are moving forward to implement newer and better ideas to further solve developer problems.

VIII. Acknowledgements

We'd like to thank Prof. Rakesh Ranjan for pushing us to create innovative applications which can be used to solve daily problems faced by humans, be it at a business level or technical level or even personal level. His timely motivation has helped us to continue working on the product and come up with more innovative ideas to enhance it.

References

[1] Electron's Binaries https://github.com/electron/electron
[2] Portainer https://github.com/portainer/portainer
[3] Docker Hub https://docs.docker.com/docker-hub/builds/
[4] Docker API - dockerode https://github.com/apocas/dockerode