Portfolio 2

Event management web app built with Laravel, deployed with Docker and Amazon Web Services

Ben Meeder and Noah Eigenfeld

Table of Contents

Overview

In portfolio 2 we created a web app that allows users to add events and other users to then register to go to those events. This rather basic web app was styled with various css styles and made to look professional. Using Laravel as the backend and a mysql database we made it easy for the database to be migrated and seeded with testing information using Laravel's built in database tools. Taking advantage of the built-in database tools allowed us to have duplicate environments on any computer or container. Besides the database tools the best use of new technologies was our deployment of this web app to AWS using a Docker container.

The application itself has several college campuses where a user can "throw an event" they can register that event either on that college's page or on the main page. The request either way is handled through the same route just on the main page the id of the college campus needs to be manually specified. When a user clicks a page to a college there is two lists that appear upcoming events and events that are happening on the current day. Next a user can click a button saying they are going to an event and it will increment that events `attendees` value by one and resort the results by number of people going in descending order.

Deployment of an application was the hardest and most intricate part. Even though Docker simplifies deployment in terms of portability it does not solve the problem of configuring a web server and the challenges that go along with it. Configuring the Laravel database migrations to work on the Docker container were alone challenging in terms of getting mysql up and running and completing the actual migrations successfully. To get mysql on a debian server one needs to install and use a package manager correctly. In this case I used "brew" on my mac, apt-get on the debian Docker container, and yum on the AWS server. Next the Docker container is configured correctly it needs to be converted back into an image, commited to Dockerhub and pushed to Dockerhub. On the AWS server the image is pulled down and one or multiple containers can be started using the `Docker run -p 8000:8000 -t <image name>` command. The Laravel server can be served using the `php artisan serve --host 0.0.0.0 --port 8000` from inside the Docker container. Now from the host the web server can be accessed on http://localhost:8000 and since the host is linked to a public ip address which is also referenced in the DNS then you can access the website from any computer at http://ec2-35-162-135-54.us-west-2.compute.amazonaws.com. One of the main selling points of Docker is that once the setup and configuration is completed for one image it can be deployed on any server or computer. If web traffic to this website was to become too large for one server to handle, the application could be deployed to multiple AWS servers to spread the load out.

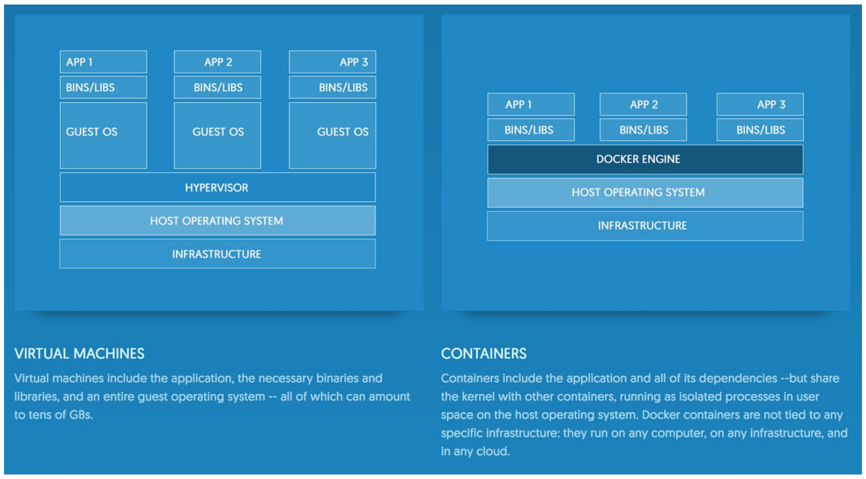
How to install and run - Taken from README.md
Navigate to: ec2-35-162-135-54.us-west-2.compute.amazonaws.com using a web browser

New and Complex

Docker

Currently the processes surrounding web applications production and staging deployment vary widely from company to company. Server config is one of the most challenging problems that face full stack web developers today. We used Docker to deploy our project. Docker is a service that allows code to be easily pushed from a source repository to a public server. Instead of requiring each of the servers running the code to have the same resources installed, Docker allows an image of an application, complete with all of its dependencies, to be created and deployed to each server. This means that any machine that has Docker installed on it will run the application without a problem eliminating server configuration and making server maintenance and deployment much easier.

Docker container wraps an application in a complete system that contains all resources need to run an application including: file system, system libraries, and dependencies. This format guarantees that no matter what machine the application is running on as long as that machine has Docker installed it will run exactly the same. A person can run a Docker container on a Windows, MacOS, Linux and in any architecture such as a physical computer or a local virtual machine, or a cloud based virtual machine. Containers running on a machine share the same operating system and same kernel which is the underlying program for a OS. They start instantly and use less RAM. Images are constructed from layered filesystems and share common files making downloads and disk space easier to manage. Containers also isolate applications from one another and the underlying infrastructure to provide an extra layer of security between the application and the outside world.



VIRTUAL MACHINES

Virtual machines include the application, the necessary binaries and libraries, and an entire guest operating system -- all of which can amount to tens of GBs.

CONTAINERS

Containers include the application and all of its dependencies --but share the kernel with other containers, running as isolated processes in user space on the host operating system. Docker containers are not tied to any specific infrastructure: they run on any computer, on any infrastructure, and in any cloud.

Containers and virtual machines have similar resource isolation and allocation benefits -- but a different architectural approach allows containers to be more portable and efficient.

Amazon Web Services Container

We used Amazon EC2 Container Service to push a Docker image of our application into an instance of an EC2 server. Amazon Web Services provide virtual web servers that customers can use to launch applications and make them visible to the public. AWS has an expansive list of products and services, from cloud computing to big data; however, for the purposes of this project, we just needed EC2 Container Service. EC2 Container Service is compatible with Docker, so there was very little setup involved in deploying our application to the server. To setup Docker on the server we first needed to install Docker which was as easy as `apt-get Docker` and then pull the Docker image from Docker hub. Finally you can just run the server and expose the port our web application is on by running `Docker run -p 8000:80`. Anything running inside the container on port 8000 will be accessible on the server on port 80.

Laravel Database Tools

The second most time consuming task for developers in today's world is unifying a database across all systems and developers. When the schema changes on one machine how do you transform that schema and deploy it on every other machine? Laravel solves this problem by providing database tools such as migrations and seeding to allow developers to quickly setup the database for their application on a new machine. The migration defines the schema of a database. While the seeding creates insert statements that add testing data to the database. These two tools allow a database to be initialized in two commands. Firstly `php artisan migrate` and then `php artisan db:seed` as opposed to several dozen or even hundreds of SQL queries.

Laravel Blade

Blade is a templating engine developed for Laravel to allow inline logic for web applications. Blade uses common programming paradigms such as if and for statements to allow for application logic. It can also easily display variables and protect information by sanitizing it.

Bloom’s Taxonomy

Analysis
Since Laravel is an MVC framework it is important to define those parts of the application. The controller and views are the easiest parts to understand. Models are a little more abstract but basically they turn a relational database into objects that make accessing data a little easier. Controllers are activated manually or automatically when a route is accessed. Controllers implement some logic such as database queries or other operations and then typically return a view. Controllers also can be used to handle form data as can be seen from PagesController.php:

```
    function create(Request $request) {
        $name = $request->input('name');
        $address = $request->input('address');
        $date  = $request->input('date');
        $time  = $request->input('time');
        $descrip = $request->input('description');
        $input = $request->all();

        $datetime = $date . ' ' . $time. ':00';

        $id = DB::select("SELECT id FROM colleges WHERE abbrev=?",[$input['colid']])[0]->id;

        DB::insert("
        INSERT INTO `events` (`event_name`, `address`, `descrip`, `date`, `attendees`,
`location`)
        VALUES(?, ?, ?, ?, 0, ?);", [$name, $address, $descrip, $datetime, $id]);

        $this->updateDates();

        return back();
    }
```

This function is called when a post request is received on the create route as seen here:

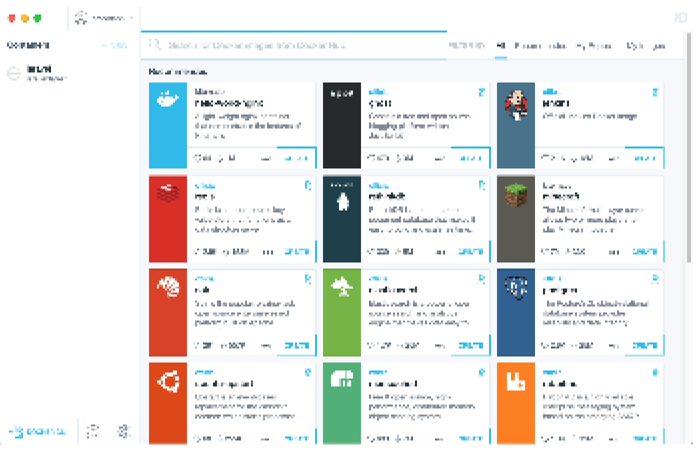
```
Route::post('/create', 'PagesController@create')->name('create');
```

Laravel and other php frameworks are a wonderful innovation because they provide a mechanism for developing fully functioning web apps compared to using php alone. So much of the organizational decisions are already made for the programmer so that they can focus on the creation itself as opposed to worrying about all of the nuts and bolts.

Evaluating

To deploy the web app we used Docker containers running on an Amazon EC2 instance. There are many ways to deploy a web application to the internet that all have varying levels of complexity. Several that I have seen used before are Digitalocean, deployer, and barebones AWS servers. With those types of deployment the version control system usually has a hook that automatically updates the code base when there is a new merge in master. The server must be configured perfectly for the exact use case and one error can bring the entire system down. When updates are available the server must be brought offline and updated and there is not an easy way to test that any configurations work before they are implemented. With Docker containers the container can be configured and tested on a local machine without tampering with the remote server. When the Docker image is finally deployed it can be expected with 100% certainty that it will run exactly the same way on the cloud server as it does on the local machine. Especially with web server deployment, consistency and reliability are the most important factors when choosing a hosting process. Most organizations also have entire dedicated teams to managing the deployment of release of code since this area of development can be the most stressful and time consuming.

Docker containers also have the wonderful ability for developers to share their configurations so the wheel does not need to be reinvented each time. If one takes a look at Dockerhub it can be seen that there are thousands of public images each doing something slightly different. There are Docker containers so that people can run their own video game, big data, apache, nginx, mysql, ruby servers. These configurations allow for so much portability between systems that the containerization of services is one of the biggest leaps forward in web development in the last five years. In our case we started with an Laravel configured debian LAMP stack container and built upon it. We didn't have to install mysql, php, or any other dependencies needed to run the latest version of Laravel. In addition to that but when a new version of Laravel is released the container can be automatically updated to work with that new version of distributed to all developers who are using it to serve their applications.



Dockerhub search viewed on the Kitematic Mac app

Creating
Event management is one of areas of applications we have found in our experiences to be very lacking in the functionality, design, and form departments. Right now most applications either do not have enough features to make for an useful tool or are way too complicated that no one wants to use them. Especially in today's world of organizational apps there is no excuse to not have successful event management applications. If we were to expand this application to include more features, it would be a good idea to expand it to be a group organization tool. For example a student club could post messages, view calendar, throw events, and collect payments all in one place.

For our creation we set out with the goal of making a basic anonymous event management and marketing tool. Users can throw events and anyone can register to go to those events. The events are organized by time and number of people going to be attending. Firstly it was decided to center on college campuses because those are places where there are often many open invite events per week. The first table of our database is the `colleges` table where any college can be added later as the application's user base expands. The second is the events table where each event has an address and a college location which is a foreign key referencing the colleges table. Users can simply enter the website and say they are attending an event by clicking a button. Displaying the number of people going to an event could have a feedback as the more people going to something more people will want to go to it creating a feedback loop.

Laravel also has the ability to bring in external resources and libraries into projects. For this project the only external library we used was FormBuilder. It provides facades to build forms and data inputs that can be embedded right into a html file using blade.

```
{!! Form::open(['route' => 'create']) !!}
        <div class="row uniform">
                <div class="6u 12u$(small)">
                {!! Form::text('name', null, ['placeholder' => 'Event Name', 'id'=>"name"]) !!}
                </div>
        </div>
{!! Form::close() !!}
```

This text above creates a form with one input but it can be expanded to be configured for any number of inputs and any type of input.

Works Cited

https://www.Docker.com/what-Docker
https://kitematic.com/
https://Laravel.com/
https://Laravelcollective.com/docs/5.3/html
http://www.ybrikman.com/writing/2015/11/11/running-Docker-aws-ground-up/#installing-Docker
https://aws.amazon.com/