Cloud Containers and Cloud Research

Daniel R. Cender

Grand Canyon University: CST-323-O500

Dec 1, 2020

**Test Application**

**GitHub URL:** [**https://github.com/DanielCender/CST-323-Test-App**](https://github.com/DanielCender/CST-323-Test-App)

**Deployment URL:** [**Cst323TestApp-env.eba-uvqpnjq2.us-east-1.elasticbeanstalk.com**](http://cst323testapp-env.eba-uvqpnjq2.us-east-1.elasticbeanstalk.com/)

The test application for this course activity set is still in development, though significantly further along than in the last milestone. It will be a multi-page lite-blogging application (think of Twitter more than WordPress), connected to a MySQL database. It’s being developed using NodeJS and the React framework, with Bootstrap for a consistent style system.

Users will “claim” a username for the time they are using the app. After they logout, that name will again be available for others to write posts under (or delete old ones). The main feed page has a basic text editor form and a scrolling view of the latest posts on the site. By clicking on the user’s username near the Logout button, they navigate to a view where they can manipulate all their user’s posts.

**ER Diagram**

As of this milestone, the test application requires only one database table to function.

Graphical user interface, application

Description automatically generated

**UI Designs**

Some basic preliminary designs made in Canva.

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, text

Description automatically generated

A picture containing graphical user interface, text

Description automatically generated

There will be an “About” page, including a feedback form that will create a comment in a different database table.

**Service API Design**

The API servicing this small application will have only a handful of routes for manipulating and querying the Post records from the connected database.

* /api/posts (GET)
* /api/post/:id (GET)
* /api/post/create (POST)
* /api/post/update (POST)

**Screenshots**

Graphical user interface, text, application, email

Description automatically generated

Cloud Computing Research

**Docker Tutorials**

Below are screenshots from completing all the assigned Docker Tutorials.

1. Docker for Beginners – Linux

Graphical user interface, text

Description automatically generated

*Interactive Ubuntu Container*

Text

Description automatically generated

A close up of text on a black background

Description automatically generated

Text

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

A screen shot of a computer

Description automatically generated

Graphical user interface, text, application

Description automatically generated

1. Doing More With Docker Images

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

1. First Alpine Linux Containers

Text

Description automatically generated

Text

Description automatically generated

1. Docker Images Deeper Dive

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**Docker Questions**

1. What is Docker?

Docker is a platform for developing, shipping, and running applications using systems that separate software from infrastructure.

1. What is a Docker File?

A Dockerfile is a specification that defines what Docker image should be used to construct a container. It also gives freedom to admins and developers to determine all necessary commands, open ports, or other settings are necessary to properly run an application in the container.

1. What is a Docker Image?

A Docker Image is essentially a blueprint for how to construct a container to run specific applications or engine runtimes. Images are available in a central repository called Docker Hub, from which developers can host and pull different versions of said images.

1. What is a Docker Container?

A Docker Container is a unit of software that houses the code and all its required dependencies. It ensures that the software can run from one computing environment to another in an isolated box.

1. What is Docker Hub?

Docker Hub is a repository for Docker Images.

1. What is Docker Swarm?

Docker Swarm is a group of physical or virtual machines that are running the Docker application and have been configured to work together in a group. From the perspective of an administrator or developer, they can interact with an entire cluster of machines in the same way they would with just one on their own device.

1. What are five advantages to using Docker Containers?
   1. Mobility: Containers allow programs to run easily in different environments with no unexpected hiccups.
   2. Consistency: Containers are easy to version and allow for a heavily tested, homogenous environment when deploying applications across clusters of machines, physically or virtually.
   3. Automation: Applications can be built with repeatable and isolated deployment configurations, making for easier and faster continuous deployments (CD).
   4. Testing: The consistency of Docker container environments makes them ideal for testing stress loads or regular testing suites for internal app issues.
   5. Modularity: The container system of Docker promotes a prime environment where developers can build and link microservices for larger applications and services.
2. What are five Docker commands you used in the tutorial, and what was the purpose for using each of the comments?
   1. “docker container run <command>”: This operation runs a command in a new container.
   2. “docker image ls”: This command lists all the images installed locally from Docker Hub.
   3. “docker image build”: This command builds a new container using a specified Dockerfile, with other options to add tags and other configurations.
   4. “docker container start”: This command starts a container specified by its identifying number.
   5. “docker image pull <image>”: This command installs the latest version of a specified image from Docker Hub.

**Kubernetes Tutorials**

1. Create a Cluster

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

1. Deploy an App

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

1. Exploring your App

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

1. Expose Your App Publicly

Graphical user interface, text

Description automatically generated

Graphical user interface, application

Description automatically generated

1. Scale Your App

Graphical user interface, text

Description automatically generated

Graphical user interface, application

Description automatically generated

1. Update Your App

Graphical user interface, text

Description automatically generated

Graphical user interface, application

Description automatically generated

**Kubernetes Questions**

1. What is Kubernetes?

Kubernetes is a system that supports deployment of groups of Docker containers. It includes tools that allow users to easily manage these clusters.

1. Why would you have a need to use Kubernetes?

Kubernetes would be a clear choice if I had an application that was expected to receive a hefty amount of traffic. It’d be easier to manage a collection of lower-powered machines in continuity, especially if they would all be running the same software.

1. What are five features that you could leverage from Kubernetes?
2. What are five differences between Kubernetes and Docker Swarm?

**Cloud Computing Research**

1. Technical Milestone Analysis
   1. Problem Statement: TODO
   2. Business Architecture Diagram: TODO
   3. Business/Technical Requirements: TODO
2. A Dockerfile is a file that contains a set of configurable directives that define what sort of system the container should create and how to set its internal environment to suit a very particular use case.

**FROM -**This directive allows a developer to specify a machine image to use when creating the container. This is a required clause for a valid Dockerfile, since it's the logical start of any container creation process. There are many starter images available in public repositories.

**RUN -**This command performs a shell command in the container, much like running commands in your MacOS Terminal or Windows Cmd command line. For Linux-based images, this runs commands in the */bin/sh -c* shell. For Windows images, this runs in the *cmd /S /C*space. Other options can be defined which will execute commands in a *bash* shell or any other available defined one. The basic idea is that the commands defined by RUN will be committed to the image in a layer on top of it, which is then available to the next set of instructions in the Dockerfile. In this way, "commits" behave much the same way as in version control systems like Git. An image can then be created from anywhere in the "commit history" created by the Dockerfile.

**ENV -**This command sets environment variables for the container, which are available to all subsequent portions of the Dockerfile. This can set one or many variables with one call to ENV. This is an easy way to set values which must be specific for different builds of a containerized application, like a test vs. production deployment.

From the Dockerfile documentation page, it looks like there's a ton of options I may never learn about. It's essentially a way to run what are Docker command-line programs via a standardized script format.

1. The idea of High Availability relates to the amount of time out of a specified period when a service can be expected to be fully operational. Cloud providers use a bunch of different methods to achieve this level of system dependability and uptime. Failover is a process where, in the event of a system failure, other systems come into action and take over the responsibilities of the failed system. In the case of a website, a redundant web server could take over all traffic if the primary server went down for a time.

The number of nines is a concept related to the amount of time in a period of a year that a system could be down to meet a level of HA expressed as a percentage including only the number nine. So, one nine means 9% uptime in a year, two nines is 99% uptime, and so on. Mission-critical systems should shoot for the highest number of nines in an SLA as possible, to ensure they don't suffer more than mere minutes of downtime in a given year.

The concept really standardizes a way of viewing system uptime across cloud providers. Four nines (99.99%) only allow for roughly 52 minutes of downtime in a service per year, which requires a hefty amount of redundancy, failover protection, and monitoring to ensure. This level of promised quality raises trust between service providers and consumers while giving potential customers a key metric to use when comparing the quality of different services or systems.

References

Kavis, M. (2014). *Architecting the cloud: Design decisions for cloud computing service models (SaaS, PaaS, and IaaS)*. Hoboken, NJ: Wiley.