Vault with mySQL Docker

Jose Aguilera
Joseph Backowski
Fernando Zavala

Our project is based on HashiCorps' Vault. The basic components of our web service is it runs on kubernetes to be able to have two worker nodes that we need in order to have our two web services to be able to connect to each other. The two files that are ran on these two worker nodes are the mySQL database and the UI vault service. Our services are composed in yml files and some sub files that can be cloned from our groups repo. Basically our services are really simple and straightforward. No need to install any packages to be able to run our services. Just run the docker compose up on the yml files and add the ip address to the vault docker compose file and that's it both of the nodes can connect. After that it's just basic, very straight forward setting up the vault as the admin of the service. All of this will be explained in further detail later on in the paper.

A special thanks to Dr.Ngo's kubernetes program on his github that made it possible for us to launch our vault service on cloudlab. Without his code we would have to create everything from the bottom up, but instead we were able to focus on creating a web service. So. much more of our time was able to be dedicated to it. Also, thanks to HashCorp for providing us a basic UI for our user experience and also mySQL.

During the very first stage of our project we weren't too sure about what we wanted to accomplish in our project. We contemplated whether to launch a storage server like onedrive or google cloud. This storming process lasted about a week until we were able to decide on a concrete choice, a password storage server. We decided this choice because we wanted to do something that would be different from other groups' projects. Witch was one of the unique choices out of the rest of the projects. After this we explored around the internet for possible ideas that we could use in order to start our service on docker. We ended up finding Vault. Vault was the perfect option for what we wanted to do. We researched Vault and found that we can use a mySQL database in order to save the encrypted passwords on the mySQL database. So, then for 2-3 weeks we worked on creating a yml file that could run our services easily. After this we went on attempting to use openstack for the base of the program. During this stage we realized that open stack wasn't doing what we wanted it to do. So, we needed to use something else that would do the job that we wanted it to do. We tried docker swarm but at the end we went with Kubernetes. This provided us with the nodes that we needed in order to launch our servers with their own individual characteristics that Kubernetes provided and also allowing our service to be able to communicate to each other. After this we just buttoned up our service until the due date.

The use of kubernetes was not the first option that we as a group had in mind. Which will be talked about in more detail later on in the report. Our group wanted to build a service that would allow for two nodes that would have their own individual ip address that would allow them to connect to each other. Which openstack didn't give us that option. So, after testing and theorizing our options we decided to use Dr.Ngo's kubernetes files as the base foundation of our two servers that we wanted to have up and running on its base two nodes. We were able to have worker 1 running our mySQL database and worker 2 run our vault service. With the head node allowing us to have a reliable and guaranteed connection/communication between our two servers. The workers in kuberties also having pods that contain containers made it much easier to separate everything that we wanted to run in their own environment. Kuberites was the perfect option for Secure password storage service. That seperated the services so if one node goes down not everything will go with it. Which could happen if someone would decide to attack one of our servers.

Running a service on cloudlab doesn't shelter users from anything that may go wrong. It's a very simple and basic cloud launching provider. Where there isn't much of anything holding your hand on launching something to the web/cloud. But launching it here means that we can launch it anywhere else with no or very little issues.

The mySQL database is the first server that we must run of the two in order to run error free because if we run our vault first we wont be able to connect the database to the vault. The database is the least complicated of the two to run because we just go to the folder named database that we cloned from our repo in our github and run docker-compose up in that folder. Then we just curl ifconfig.me to get the ip address; that we need in the vault in order for it to connect to each other. The yaml file with docker compose makes it easy to run our database in any environment. Not needing to install a bunch of packages in order for us to have this part of our service up and running.

The vault server is the most complicated of the two because not only does it contain a UI it also has to connect to the database also, and has to run more files in order for it to set itself up. The first file that the vault has is the docker-compose file. This file is just like the file in mysql where it launches the server. But it also launches another file in the volumes.config folder. The file is called vault.json. In this file is where we have to set-up certain specifics for our vault service. For example making a database and providing the name and ip address of the database server. For this we made sure it was working and making that database for a vault by checking it with an UI mySQL database, but in the actual project we opted not to use the UI because of the potential of more risks that it can create. But nonetheless we made sure that the vault was encrypting the passwords in our database also. Another great safety precaution. The vault.json also enables the ui and also allows the server to use the nodes or workers ip address.

The complexity of this project is very little and can be run in any linux or similar type of environment. Where we have tested it on powershell, MacOS and on linux systems with no issues at all. It also doesn't need anything to be installed, which makes it so easy to run for anyone. Just download files and run docker compose is pretty much all that is needed and get the ip needed and add it to vault database ip. We hope to make it even more simpler which we will be discussing further in the paper.

Throughout the entire project we faced a lot of challenges, but we were able to overcome them and get our Vault up and running. At first we tried to set up the vault and mySQL through openstack but we had issues getting the vault node and the mySQL node to connect with each other. That's where we realized this project was better suited to do in a kubernetes environment, since we were not even using VMs. Setting up the files needed to run mySQL with vault was a challenge as there were not any guides to specifically connect the two so it took some trial and error to get it all working. At first the mySQL docker-compose.yml did not have the ui image in it so we had trouble accessing the server, soon after we added the adminer ui image to allow us to access it through a browser. So with the ui we were able to take a look at mySQL and see that Vault created it's directory. We did not keep that image in the final version for security reasons but it helped us see what was going on in the backend. With Vault initially I attempted to install it locally on my laptop to see how it worked and how to set it up but I was unable to do it through docker toolbox, the vault would not run, so I gave up on that as that issue was likely related to my docker toolbox installation.

Using the same docker-compose.yml file and config file we were able to set up the Vault on the Kubernetes cloudlab instance. The initial vault would store the files locally but we wanted to store everything on a mySQL server, storing locally worked without an issue but when integrating mySQL we ran into a TCP connection being rejected. mySQL was refusing to connect to the vault when the address in our vault.json file was set to 0.0.0.0:3306. We spent hours trying to figure out what was wrong and why the connection was being refused, even opening up port 3306 in the firewall but that still did not work. The solution ended up being that the address in the vault.json file needed to be the global ip address of the cloudlab server node that mySQL was running on, and finally we were able to get the vault running with it storing the information on our mySQL server. Attempting to automate the process gave us issues as well, we attempted to create a .sh file that would do all the docker compose, and all the setups but we were unable to get that to work. The file would not even install kubernetes or docker to the cloud instance and we were unable to figure out why, but for the most part our process to get this running mainly involves running two docker-compose up commands in the right folders so the automation really isn't all that necessary.

The steps that ended up working for us in the end was to run mySQL on node one, make sure that mySQL is run first otherwise there is no backend for Vault to connect to. Cd into the database directory then run docker-compose up and that will run the mySQL server. Next step is to get the global ip from the node that is running mySQL, copy that ip and on node two, go into the Vault config file "vault.json" edit it and change the mySQL address to that ip with port 3306. Now we can run the Vault docker-compose by using cd to get into the Vault folder and running docker-compose up. Now the Vault is up and running and storing the data into the mySQL server.

For the next iteration of our project, culminating in the third project deliverable, we were looking to implement a few different changes and improvements. Firstly, we were going to focus on having a more thorough integration with the Kubernetes structure. Currently, the images that we use exist and function on Kubernetes nodes, but the method through which they communicate is independent of the Kubernetes internal network. The pods, in essence, are isolated from each other and both publicly broadcast their IP addresses, which is how they work together. This situation presents a number of issues, firstly there is a security concern, as the database the vault system stores secrets on is publically available through its IP. This means that if one knew the IP of the database, they would be able to look at and manipulate the data contained within.

The second improvement that we were aiming to make is to increase the automation within the deployment of the system itself. As the project currently stands, there is a fair bit of work that a person has to do within the terminal in order to completely set up our project. Ideally, there would only need to be a couple of commands that the user needs to execute before the network is fully functional, and that is what we will hope to accomplish before the next deliverable is due for this project. Additionally, we were going to improve the clarity of our GitHub repository, by providing a greater amount of documentation within the files themselves, as well as in the "Readme" files and the descriptions of each subsection.

These two things were what the group had intended to improve on our project between the second and third project deliverables, but in reality we were not able to implement either of these improvements. There are a few reasons why these changes did not occur within our project, the first and most important of which is time. The time between the submission of the second and third project deliverables was less than a week, and as such there was very little time for us to improve our project, especially considering that final exams and projects were also occurring during this week. Another reason why one of these changes was not made was due to us realizing that there was not a need for the improvement. One of our previous goals was to improve the automation of the process, but in reality without us making any other changes to the project, there is very little for the initial setup to do. As such, there was little reason to try and automate what amounts to three lines of input in the terminal.

There are further improvements that we see that could improve the quality of this project, but that are not within our current abilities, or require more time and/or resources than we currently have. The first, and most important, improvement would be to the security of the project. We are going to remove the vulnerability with the exposed database, but a higher security standard would be a welcome change. For example, in order to unseal the vault and to access the data within, several keys are needed. These keys are generated when the vault is created, and the number of keys generated and the number of keys needed to unseal the vault is completely arbitrary. What could potentially be done is to have several different user accounts, and have the number of keys needed to unseal the vault be significantly less than the total number of keys. Then, when a user account is created, they would be given a certain set of keys, and those keys would be specific to the user, so that if a different user tried to unseal the vault with different, but valid keys, they would not be able to access the original user's information. This would serve as another level of security for the vault.

As intimated within the previous section, another improvement that could be made is to increase the number of users on a single pair of vault and database. Currently, there is only a single user account with master control of the vault, and that master account is able to create sub-users without the ability to seal/unseal the vault. To improve the service, there would need to be the ability to have more master accounts, with distinct storage. This could be done via a custom frontend that would pass the requisite keys to the vault, which would then adjust the permissions of the user to match the information provided.

For the final deliverable, it is our opinion that we have technically met the final deliverable, but perhaps not the intent of the final deliverable. The reason that we have met the final deliverable is because we have a product that runs and operates in the cloud, and is quite easy to implement given access to the GitHub repository and the instructions that are contained within the files on that repository. The product functions fully as intended, and provides the user with the service that is promised by the project. These are the reasons why we have met the final deliverable, though there is an argument to be made that we have not met the entirety of the final deliverable. While the product is fully functional and provides the service as described, it is not implemented very well given the capabilities of kubernetes and docker swarm networks. The two docker containers that are launched through the GitHub repository are isolated entities, and both must publicly broadcast their IP addresses in order to access one another. This essentially uses none of the features of Kubernetes and Docker, and it provides a security risk. The database on which the sensitive data is kept is able to be accessed directly through its IP address, and though the data is encrypted by the Vault, that still presents a serious security risk. So, while the product is moderately secure, there are still potential backdoors, and as such one should be wary of putting anything of a sensitive nature in the Vault, which defeats the purpose of the project. The reasons why the project is not quite integrated with Kubernetes have been listed previously, and they are the reason why the final deliverable has been met, but not completely.