# Vault with mySQL Docker

Jose Aguilera
Joseph Backowski
Fernando Zavala

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.

Running a service on cloudlab doesn't shelter users from anything that may go wrong. Its 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 most run of the two in order to run error free because if we ran 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.

Throughout the entire project we faced a lot of challenges, but we were able to overcome them. 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. 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. 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.

For the next iteration of our project, culminating in the third project deliverable, there are a few different changes and improvements that we are looking to implement. Firstly, we are 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 are 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 are 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.

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.