

ECE 473/573 Fall 2024 - Project 2

Service Containerization

Samman Chouhan A20561414

II. Key-Value Store Orchestration

- What is the license of the original example code from the textbook? (Hint: read the 'LICENSE' file)

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>

- Open source software may use other licenses like GNU General Public License (GNU GPL) and GNU Affero General Public License (GNU AGPL). What are the difference between GNU GPL and GNU AGPL for cloud services?

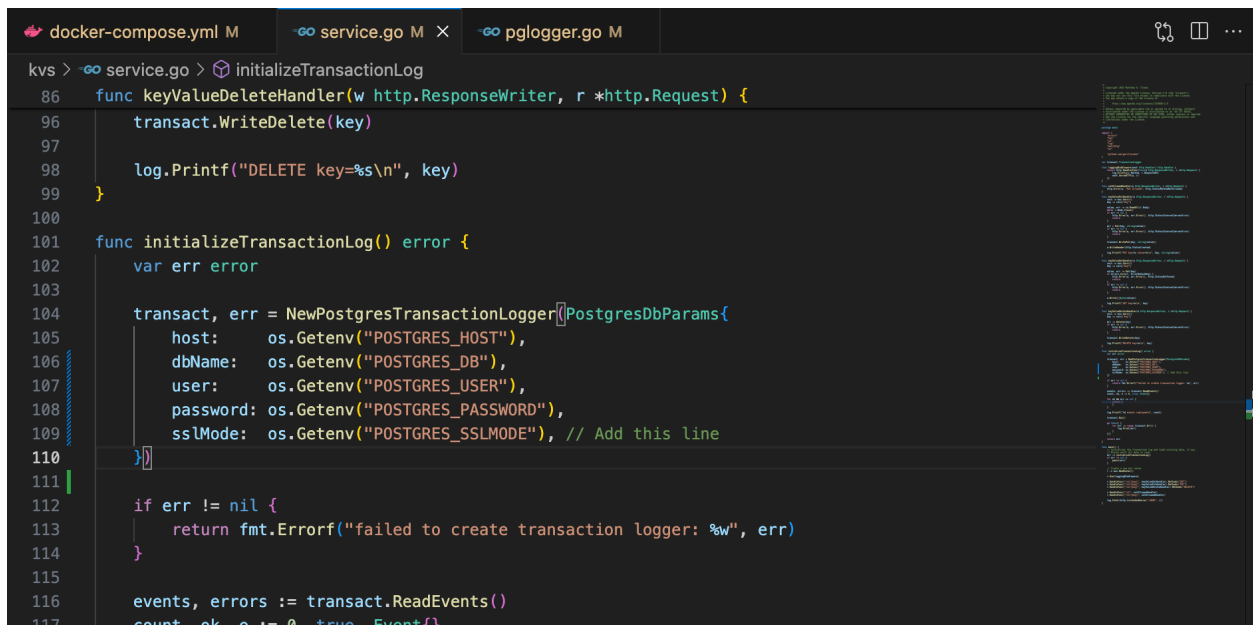
The main difference between GNU GPL and GNU AGPL lies in their application to cloud services. While GPL only requires source code disclosure when the software is distributed to users, AGPL extends this requirement to software used over a network, such as in cloud services or web applications. This means that if you run AGPL-licensed software on a

server and allow users to interact with it remotely, you must make the source code available to those users. This distinction has significant implications for cloud-based software. GPL creates a "SaaS loophole" where companies can use and modify GPL code without sharing their changes if they only offer the software as a service. AGPL closes this loophole by treating network interaction as a form of distribution, ensuring that modifications to the software are shared back to the community even in cloud environments. As a result, AGPL is often chosen for software likely to be used in cloud services, while GPL is more commonly used for software meant for local installation and use.

- The kvs service depends on the postgres service as it needs to load the transaction log from the database. The dependency is defined in `docker-compose.yml` so that postgres starts before kvs. However, the log messages above indicate that kvs still needs to retry connecting to postgres. Why?

The connection retry issue between kvs and postgres services may occur despite the dependency defined in `docker-compose.yml`. This is because the dependency only ensures that postgres starts before kvs, but doesn't guarantee that postgres is fully operational and ready to accept connections when kvs initiates its connection attempts. The postgres service might require additional time to initialize and become fully available, resulting in kvs needing to retry its connection attempts until postgres is ready to handle incoming connections.

III. Manageability via Environment Variables



```
docker-compose.yml M  service.go M X  pglogger.go M
kvs > service.go > initializeTransactionLog
86 func keyValuePairDeleteHandler(w http.ResponseWriter, r *http.Request) {
96     transact.WriteDelete(key)
97
98     log.Printf("DELETE key=%s\n", key)
99 }
100
101 func initializeTransactionLog() error {
102     var err error
103
104     transact, err = NewPostgresTransactionLogger(PostgresDbParams{
105         host:    os.Getenv("POSTGRES_HOST"),
106         dbName:  os.Getenv("POSTGRES_DB"),
107         user:    os.Getenv("POSTGRES_USER"),
108         password: os.Getenv("POSTGRES_PASSWORD"),
109         sslMode: os.Getenv("POSTGRES_SSLMODE"), // Add this line
110     })
111
112     if err != nil {
113         return fmt.Errorf("failed to create transaction logger: %w", err)
114     }
115
116     events, errors := transact.ReadEvents()
117     count ok e := 0 true Event{}
```

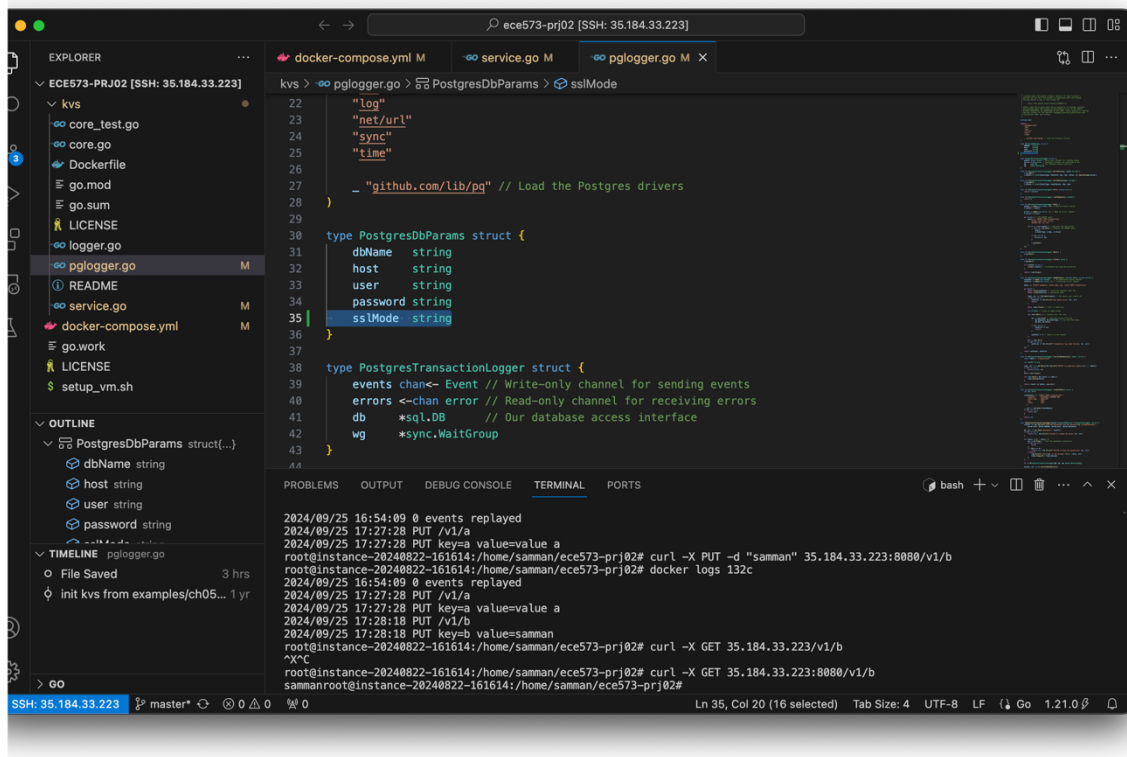
Here the Environment variable are called using the `os.Getenv` function , since it's not a good practice to use hard coded credentials in the source file

We use these variables from the `docker-compose.yml` file which is mentioned below, it's a good standard practice to not use credentials directly

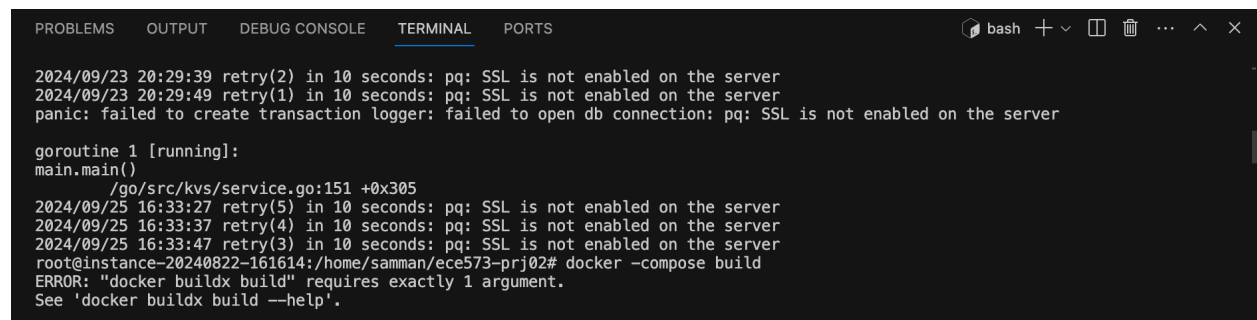
```
docker-compose.yml M X  service.go M  pglogger.go M
docker-compose.yml
1  services:
2    postgres:
3      image: postgres:11
4      environment:
5        - POSTGRES_USER=test
6        - POSTGRES_PASSWORD=hunter2
7        - POSTGRES_DB=kvs
8
9    kvs:
10     build: kvs
11     environment:
12       - POSTGRES_HOST=postgres
13       - POSTGRES_DB=kvs
14       - POSTGRES_USER=test
15       - POSTGRES_PASSWORD=hunter2
16       - POSTGRES_SSLMODE=disable # Add this line
17
18     ports:
19       - "8080:8080"
20
21     depends_on:
22       - postgres
```

Modified Docker-compose file

Here I have also added another variable called POSTGRESS_SSLMODE since I was running this machine on cloud, the issue doesn't appear when working on a local machine For which I have also updated the plogger.go to define the variable which is highlighted



The original error in question was this



Which was resolved using the two updates above

Another thing in order to Update keys on the kvs servers was that we have to enable incoming TCP connection on port 8080 which wasn't the case for me since I wasn't working locally. Also if we try to setup Firewall rules using "ufw" or "iptables" on the ubuntu distribution and try to enable traffic on any port , it will not work

The correct method in my case(for Google Cloud Platform) is to enable it via gcloud shell mentioned below

or to unset it, run:

```
$ gcloud config unset project
g24s_chouhan@cloudshell:~$ gcloud config set project "majestic-altar-433219-h8"
Updated property [core/project].
g24s_chouhan@cloudshell:~ (majestic-altar-433219-h8)$ gcloud compute firewall-rules create example-service --allow=tcp:8080 --description="Allow incoming traffic on TCP port 8080" --direction=INGRESS
Creating firewall...working..Created [https://www.googleapis.com/compute/v1/projects/majestic-altar-433219-h8/global/firewalls/example-service].
Creating firewall...done.
NAME: example-service
NETWORK: default
DIRECTION: INGRESS
PRIORITY: 1000
ALLOW: tcp:8080
DENY:
DISABLED: False
g24s_chouhan@cloudshell:~ (majestic-altar-433219-h8)$
```

Process to Build Create and Start

```
samman@instance-20240822-161614:~/ece573-prj02$ gdocker compose build
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Head "http://%2Fvar%2Frun%2Fdocker.sock/_ping": dial unix /var/run/docker.sock: connect: permission denied
samman@instance-20240822-161614:~/ece573-prj02$ sudo su
root@instance-20240822-161614:/home/samman/ece573-prj02# docker compose build
[+] Building 0.3s (11/11) FINISHED                                docker:default
=> [kvs internal] load build definition from Dockerfile           0.0s
=> => transferring dockerfile: 685B                               0.0s
=> WARN: FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 3) 0.0s
=> WARN: FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 18) 0.0s
=> [kvs internal] load metadata for docker.io/library/golang:1.21 0.0s
=> [kvs internal] load .dockerignore                             0.0s
=> => transferring context: 2B                                     0.0s
=> [kvs internal] load build context                             0.0s
=> => transferring context: 286B                                   0.0s
=> [kvs build 1/4] FROM docker.io/library/golang:1.21           0.0s
=> CACHED [kvs build 2/4] COPY . /go/src/kvs                    0.0s
=> CACHED [kvs build 3/4] WORKDIR /go/src/kvs                   0.0s
=> CACHED [kvs build 4/4] RUN CGO_ENABLED=0 GOOS=linux go build -o kvs 0.0s
=> CACHED [kvs image 1/1] COPY --from=build /go/src/kvs/kvs .   0.0s
=> [kvs] exporting to image                                     0.0s
=> => exporting layers                                           0.0s
=> => writing image sha256:85041f9ac482b944101755136f02f693fa9dff58eede0f58e3302af9e55bd884 0.0s
=> => naming to docker.io/library/ece573-prj02-kvs              0.0s
=> [kvs] resolving provenance for metadata file                 0.0s
root@instance-20240822-161614:/home/samman/ece573-prj02# docker compose create
[+] Creating 2/0
✓ Container ece573-prj02-postgres-1 Running                    0.0s
✓ Container ece573-prj02-kvs-1 Running                          0.0s
root@instance-20240822-161614:/home/samman/ece573-prj02# docker compose start
root@instance-20240822-161614:/home/samman/ece573-prj02#
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
^X^C
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -X PUT -d "value a" 35.184.33.223:8080/v1/a
root@instance-20240822-161614:/home/samman/ece573-prj02# docker logs 132c
2024/09/25 16:54:09 0 events replayed
2024/09/25 17:27:28 PUT /v1/a
2024/09/25 17:27:28 PUT key=a value=value a
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -X PUT -d "samman" 35.184.33.223:8080/v1/b
root@instance-20240822-161614:/home/samman/ece573-prj02# docker logs 132c
2024/09/25 16:54:09 0 events replayed
2024/09/25 17:27:28 PUT /v1/a
2024/09/25 17:27:28 PUT key=a value=value a
2024/09/25 17:28:18 PUT /v1/b
2024/09/25 17:28:18 PUT key=b value=samman
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -X GET 35.184.33.223/v1/b
^X^C
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -X GET 35.184.33.223:8080/v1/b
sammanroot@instance-20240822-161614:/home/samman/ece573-prj02# docker kill ece573-prj02-kvs-1
ece573-prj02-kvs-1
root@instance-20240822-161614:/home/samman/ece573-prj02# docker compose start
[+] Running 1/1
  ✓ Container ece573-prj02-kvs-1 Started 0.7s
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -X GET 35.184.33.223:8080/v1/b
sammanroot@instance-20240822-161614:/home/samman/ece573-prj02# curl -w "\n" -X GET 35.184.33.223:8080/v1/b
samman
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -X PUT -d "Test Pair 1" 35.184.33.223:8080/v1/c
root@instance-20240822-161614:/home/samman/ece573-prj02# docker kill ece573-prj02-postgres-1
ece573-prj02-postgres-1
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -X PUT -d "Test Pair 2" 35.184.33.223:8080/v1/d
root@instance-20240822-161614:/home/samman/ece573-prj02# docker compose start
[+] Running 1/1
  ✓ Container ece573-prj02-postgres-1 Started 0.4s
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -w "\n" -X GET 35.184.33.223:8080/v1/c
Test Pair 1
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -w "\n" -X GET 35.184.33.223:8080/v1/d
Test Pair 2
root@instance-20240822-161614:/home/samman/ece573-prj02#
```

Here in the top we can see that we put a key “a” whose value was “value a” using the PUT method , the kvs service allows three methods on the server , GET , PUT and DELETE

Using the command ***docker logs ‘container id’*** , we can see the logs for the particular service

And then when we use the GET method to retrieve the same key value pair, which I'll mention in the next screenshots, which can be verified from the docker logs

```
root@instance-20240822-161614:/home/samman/ece573-prj02# docker logs 132c
2024/09/25 16:54:09 0 events replayed
2024/09/25 17:27:28 PUT /v1/a
2024/09/25 17:27:28 PUT key=a value=value a
2024/09/25 17:28:18 PUT /v1/b
2024/09/25 17:28:18 PUT key=b value=samman
2024/09/25 17:29:24 GET /v1/b
2024/09/25 17:29:24 GET key=b
2024/09/25 20:38:38 2 events replayed
2024/09/25 20:38:48 GET /v1/b
2024/09/25 20:38:48 GET key=b
2024/09/25 20:48:57 GET /v1/b
2024/09/25 20:48:57 GET key=b
2024/09/25 20:50:01 PUT /v1/c
2024/09/25 20:50:01 PUT key=c value=Test Pair 1
2024/09/25 20:51:33 PUT /v1/d
2024/09/25 20:51:33 PUT key=d value=Test Pair 2
2024/09/25 20:51:33 dial tcp: lookup postgres on 127.0.0.11:53: server misbehaving
2024/09/25 20:51:57 GET /v1/c
2024/09/25 20:51:57 GET key=c
2024/09/25 20:52:01 GET /v1/d
2024/09/25 20:52:01 GET key=d
2024/09/25 20:55:01 3 events replayed
2024/09/25 20:55:05 GET /v1/c
2024/09/25 20:55:05 GET key=c
2024/09/25 20:55:14 GET /v1/d
root@instance-20240822-161614:/home/samman/ece573-prj02#
```

IV. (Bonus) Fault Tolerance Testing

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
2024/09/25 17:27:28 PUT key=a value=value a
2024/09/25 17:28:18 PUT /v1/b
2024/09/25 17:28:18 PUT key=b value=samman
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -X GET 35.184.33.223/v1/b
^X^C
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -X GET 35.184.33.223:8080/v1/b
sammanroot@instance-20240822-161614:/home/samman/ece573-prj02# docker kill ece573-prj02-kvs-1
ece573-prj02-kvs-1
root@instance-20240822-161614:/home/samman/ece573-prj02# docker compose start
[+] Running 1/1
  ✓ Container ece573-prj02-kvs-1 Started 0.7s
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -X GET 35.184.33.223:8080/v1/b
sammanroot@instance-20240822-161614:/home/samman/ece573-prj02# curl -w "\n" -X GET 35.184.33.223:8080/v1/b
samman
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -X PUT -d "Test Pair 1" 35.184.33.223:8080/v1/c
root@instance-20240822-161614:/home/samman/ece573-prj02# docker kill ece573-prj02-postgres-1
ece573-prj02-postgres-1
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -X PUT -d "Test Pair 2" 35.184.33.223:8080/v1/d
root@instance-20240822-161614:/home/samman/ece573-prj02# docker compose start
[+] Running 1/1
  ✓ Container ece573-prj02-postgres-1 Started 0.4s
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -w "\n" -X GET 35.184.33.223:8080/v1/c
Test Pair 1
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -w "\n" -X GET 35.184.33.223:8080/v1/d
Test Pair 2
root@instance-20240822-161614:/home/samman/ece573-prj02# docker kill ece573-prj02-kvs-1
ece573-prj02-kvs-1
root@instance-20240822-161614:/home/samman/ece573-prj02# docker compose start
[+] Running 1/1
  ✓ Container ece573-prj02-kvs-1 Started 0.5s
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -w "\n" -X GET 35.184.33.223:8080/v1/c
Test+Pair+1
root@instance-20240822-161614:/home/samman/ece573-prj02# curl -w "\n" -X GET 35.184.33.223:8080/v1/d
no such key

root@instance-20240822-161614:/home/samman/ece573-prj02#
```

1. Put a key-value pair to the store.
2. Shutdown kvs by **docker kill ece573-prj02-kvs-1**
3. Restart kvs by **docker compose start**
4. Validate that the key-value pair is still there.

We put key value pair , key being “b” and the value being “samman” as instructed we kill the kvs service and restart it again, and we find out that the kv pair is still there

The test procedure for the 'postgres' service is more complicated since the client may continue to connect to the 'kvs' service.

1. Put a key-value pair to the store.
2. Shutdown postgres.
3. Put another key-value pair to the store.
4. Restart postgres.
5. Are the two pairs still there?
6. Shutdown and restart kvs.
7. Are the two pairs still there?

We first put a Key-Value , key – c and value – Test Pair 1

Then we kill postgres service

Then we put another pair , key – d and value – Test Pair 2

And then we restart postgres

After checking , I found both of the pair intact as can be seen in the screenshot above

After killing the kvs service we find that only the first pair is there and the pair seems to be modified , need to do more research on that but it is there , while the second pair is not

Is there any data loss if one service is down? Why?

The potential for data loss in the system involving the "kvs" and "postgres" services is contingent upon the specific architecture and implementation of the application. In this particular scenario, data integrity is compromised when the kvs service undergoes a shutdown process. The way the application is structured and manages its interactions with these services plays a crucial role in determining the vulnerability to data loss during such events.

- If there is, what would you like to do?

To mitigate the risk of data loss in the system, several strategies can be implemented:

1. Ensure continuous operation of the kvs service to minimize interruptions.
2. Implement data replication mechanisms to maintain multiple copies of the data across different nodes.
3. Establish regular data backup routines to create recoverable snapshots of the system's state.
4. Set up comprehensive monitoring and alert systems to promptly detect and respond to potential issues.
5. Utilize local data storage as a buffer or cache to temporarily hold data during service disruptions.

These measures, when properly implemented, can significantly reduce the likelihood and impact of data loss in the system.