

LAB 6 REPORT –EXPLAIN CODE

Table of Contents

[GIẢI THÍCH CODE CHI TIẾT 3](#_Toc161094169)

[FOLDER CLIENT 3](#_Toc161094170)

[FOLDER COMMON 4](#_Toc161094171)

[FOLDER CONFIG 4](#_Toc161094172)

[FOLDER CONSUL 6](#_Toc161094173)

[FOLDER HELLO (có chứa code liên quan đến check health của các instance) 7](#_Toc161094174)

[FOLDER middleware 9](#_Toc161094175)

[FOLDER mongodb/scripts 15](#_Toc161094176)

[Các file init 15](#_Toc161094177)

[File docker-compose 16](#_Toc161094178)

[File hướng dẫn chạy và setup toàn bộ code 21](#_Toc161094179)

[FOLDER router 23](#_Toc161094180)

[FOLDER scripts 24](#_Toc161094181)

[docker\_run\_golangbook.sh 24](#_Toc161094182)

[docker\_teardown\_golangbook.sh 28](#_Toc161094183)

[golang\_build.sh 29](#_Toc161094184)

[golang\_run.sh 30](#_Toc161094185)

[FOLDER structs 31](#_Toc161094186)

[FILE .ENV 32](#_Toc161094187)

[FILE DOCKERFILE 32](#_Toc161094188)

[FILE FABIO.PROPERTIES 33](#_Toc161094189)

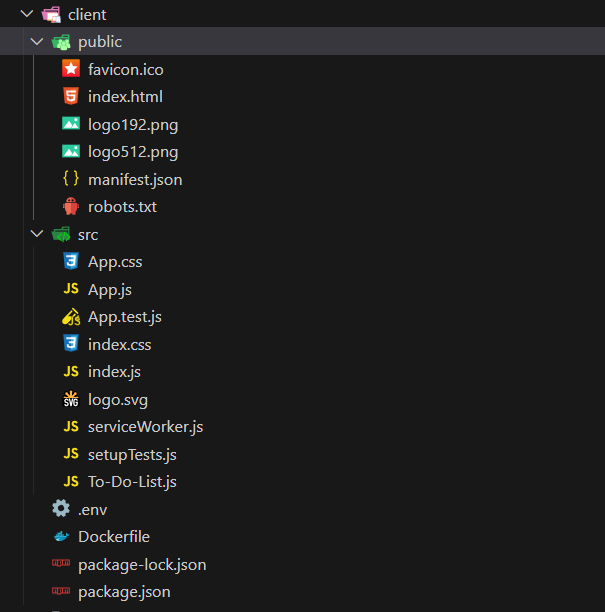
[FILE MAIN.GO 33](#_Toc161094190)

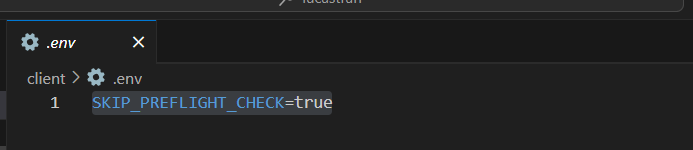
[FILE MAKEFILE 34](#_Toc161094191)

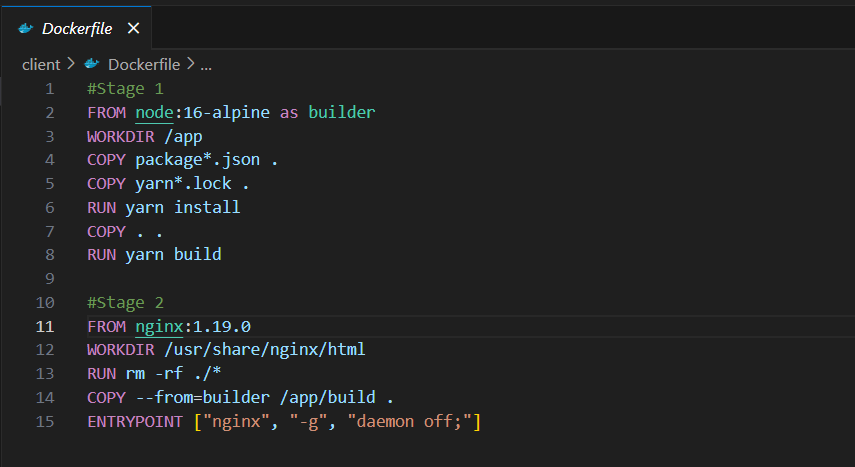
# GIẢI THÍCH CODE CHI TIẾT

## FOLDER CLIENT

Chứa code UI của ứng dụng todolist (sử dụng React) và các file khác như như .env, dockerfile, package.json







## FOLDER COMMON

// Package common implements shared application functions

package common

import (

    "bufio"

    "strings"

)

// Return slice with each line of a multi-line string, splitting on '\n'

func SplitStringLines(s string) []string {

    var lines []string

    sc := bufio.NewScanner(strings.NewReader(s))

    for sc.Scan() {

        lines = append(lines, sc.Text())

    }

    return lines

}

## FOLDER CONFIG

// Package config implements .env file application configuration

package config

import (

    "embed"

    "strconv"

    "strings"

    "golangbook/common"

)

var EnvVarsFile embed.FS

var IPAddress string

var ServerPort int

// AppConfig returns a new decoded Config map from .env file variables or sets from defaults

func AppConfig() (map[string]string, error) {

    envVars, \_ := EnvVarsFile.ReadFile(".env")

    lines := common.SplitStringLines(string(envVars))

    var envs = make(map[string]string)

    for \_, line := range lines {

        keyValuePair := strings.Split(line, "=")

        envs[keyValuePair[0]] = keyValuePair[1]

        if keyValuePair[0] == "IPADDRESS" {

            if keyValuePair[1] == "" {

                IPAddress = "192.168.0.99"

            } else {

                IPAddress = keyValuePair[1]

            }

        }

        if keyValuePair[0] == "SERVERPORT" {

            if keyValuePair[1] == "" {

                ServerPort = 8080

            } else {

                ServerPort, \_ = strconv.Atoi(keyValuePair[1])

            }

        }

    }

    return envs, nil

}

## FOLDER CONSUL

package consul

import (

    "fmt"

    "log"

    "os"

    "strconv"

    "github.com/google/uuid"

    "github.com/hashicorp/consul/api"

)

type ConsulClient struct {

    \*api.Client

}

func ServiceRegistryWithConsul(ipAddress string, port int, myUUID uuid.UUID) {

    config := api.DefaultConfig()

    consul, err := api.NewClient(config)

    if err != nil {

        log.Println(err)

    }

    /\* Mỗi instance dịch vụ nên có một dịch vụ duy nhất \*/

    serviceID := fmt.Sprintf("hello-%v", myUUID)

    /\* Tag nên tuân theo quy tắc của fabio: urlprefix- \*/

    tags := []string{"urlprefix-/"}

    // Dockerport: Điều này được inject trong lệnh `Docker Run`. Nó không tồn tại khi ứng dụng Go chạy bên ngoài container docker

    dockerContainerPort, \_ := strconv.Atoi(os.Getenv("DOCKERPORT"))

    registration := &api.AgentServiceRegistration{

        ID:      serviceID,

        Name:    "hello-todo",

        Port:    dockerContainerPort,

        Address: ipAddress,

        Tags:    tags, /\* Thêm thẻ để đăng ký \*/

        Check: &api.AgentServiceCheck{

            HTTP:     fmt.Sprintf("http://%s:%v/health", ipAddress, dockerContainerPort),

            Interval: "10s",

            Timeout:  "30s",

        },

    }

    registrationErr := consul.Agent().ServiceRegister(registration)

    if registrationErr != nil {

        log.Printf("Failed to register service: %s:%v ", ipAddress, dockerContainerPort)

    } else {

        log.Printf("successfully register service: %s:%v", ipAddress, dockerContainerPort)

    }

}

func NewClient(addr string) (\*ConsulClient, error) {

    conf := &api.Config{

        Address: addr,

    }

    client, err := api.NewClient(conf)

    if err != nil {

        log.Println("error initiating new consul client: ", err)

        return &ConsulClient{}, err

    }

    return &ConsulClient{

        client,

    }, nil

}

## FOLDER HELLO (có chứa code liên quan đến check health của các instance)

// Package hello handles requestss

package hello

import (

    "encoding/json"

    "fmt"

    "math/rand"

    "net/http"

    "time"

    "io/ioutil"

    "github.com/google/uuid"

    "golangbook/structs"

)

var letters = []rune("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")

func init() {

    rand.Seed(time.Now().UnixNano())

}

func getip2() string {

    req, err := http.Get("http://ip-api.com/json/")

    if err != nil {

        return err.Error()

    }

    defer req.Body.Close()

    body, err := ioutil.ReadAll(req.Body)

    if err != nil {

        return err.Error()

    }

    var ip structs.IP

    json.Unmarshal(body, &ip)

    return ip.Query

}

//Hàm này là hàm cũ được dùng để demo app hello-weather đơn giản, chỉ cần

Gọi api để trả về thông tin UUID và data tương ứng

func HelloHandler(w http.ResponseWriter, r \*http.Request, myUUID uuid.UUID) {

    w.Header().Set("Content-Type", "application/json")

    w.Header().Set("Status", "200")

    hTTPResponse := structs.HTTPResponse{

        Status:      200,

        Application: "hello-weather",

        IP: getip2(),

        UUID:        myUUID,

        Data:        randSeq(200000),

    }

    err := json.NewEncoder(w).Encode(hTTPResponse)

    if err != nil {

        fmt.Fprintf(w, "%+v", hTTPResponse)

    }

}

func HealthHandler(w http.ResponseWriter, r \*http.Request) {

    w.WriteHeader(http.StatusOK)

    fmt.Fprintf(w, "Service alive and reachable")

}

func randSeq(n int) string {

    b := make([]rune, n)

    for i := range b {

        b[i] = letters[rand.Intn(len(letters))]

    }

    return string(b)

}

## FOLDER middleware

package middleware

import (

    "context"

    "encoding/json"

    "fmt"

    "golangbook/models"

    "log"

    "net/http"

    "os"

    "github.com/gorilla/mux"

    // "github.com/joho/godotenv"

    "go.mongodb.org/mongo-driver/bson"

    "go.mongodb.org/mongo-driver/bson/primitive"

    "go.mongodb.org/mongo-driver/mongo"

    "go.mongodb.org/mongo-driver/mongo/options"

)

// collection object/instance

var collection \*mongo.Collection

// create connection with mongo db

func init() {

    loadTheEnv()

    createDBInstance()

}

func loadTheEnv() {

    // load .env file

    // err := godotenv.Load(".env")

    // if err != nil {

    //  log.Fatalf("Error loading .env file")

    // }

}

func createDBInstance() {

    // DB connection string

    connectionString := os.Getenv("DB\_URI")

    // Database Name

    dbName := os.Getenv("DB\_NAME")

    // Collection name

    collName := os.Getenv("DB\_COLLECTION\_NAME")

    // Set client options

    clientOptions := options.Client().ApplyURI(connectionString)

    // connect to MongoDB

    client, err := mongo.Connect(context.TODO(), clientOptions)

    if err != nil {

        log.Fatal(err)

    }

    // Check the connection

    err = client.Ping(context.TODO(), nil)

    if err != nil {

        log.Fatal(err)

    }

    fmt.Println("Connected to MongoDB!")

    collection = client.Database(dbName).Collection(collName)

    fmt.Println("Collection instance created!")

}

// GetAllTask get all the task route

func GetAllTask(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Context-Type", "application/x-www-form-urlencoded")

    w.Header().Set("Access-Control-Allow-Origin", "\*")

    payload := getAllTask()

    json.NewEncoder(w).Encode(payload)

}

// CreateTask create task route

func CreateTask(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Context-Type", "application/x-www-form-urlencoded")

    w.Header().Set("Access-Control-Allow-Origin", "\*")

    w.Header().Set("Access-Control-Allow-Methods", "POST")

    w.Header().Set("Access-Control-Allow-Headers", "Content-Type")

    var task models.ToDoList

    \_ = json.NewDecoder(r.Body).Decode(&task)

    // fmt.Println(task, r.Body)

    insertOneTask(task)

    json.NewEncoder(w).Encode(task)

}

// TaskComplete update task route

func TaskComplete(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Content-Type", "application/x-www-form-urlencoded")

    w.Header().Set("Access-Control-Allow-Origin", "\*")

    w.Header().Set("Access-Control-Allow-Methods", "PUT")

    w.Header().Set("Access-Control-Allow-Headers", "Content-Type")

    params := mux.Vars(r)

    taskComplete(params["id"])

    json.NewEncoder(w).Encode(params["id"])

}

// UndoTask undo the complete task route

func UndoTask(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Content-Type", "application/x-www-form-urlencoded")

    w.Header().Set("Access-Control-Allow-Origin", "\*")

    w.Header().Set("Access-Control-Allow-Methods", "PUT")

    w.Header().Set("Access-Control-Allow-Headers", "Content-Type")

    params := mux.Vars(r)

    undoTask(params["id"])

    json.NewEncoder(w).Encode(params["id"])

}

// DeleteTask delete one task route

func DeleteTask(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Context-Type", "application/x-www-form-urlencoded")

    w.Header().Set("Access-Control-Allow-Origin", "\*")

    w.Header().Set("Access-Control-Allow-Methods", "DELETE")

    w.Header().Set("Access-Control-Allow-Headers", "Content-Type")

    params := mux.Vars(r)

    deleteOneTask(params["id"])

    json.NewEncoder(w).Encode(params["id"])

    // json.NewEncoder(w).Encode("Task not found")

}

// DeleteAllTask delete all tasks route

func DeleteAllTask(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Context-Type", "application/x-www-form-urlencoded")

    w.Header().Set("Access-Control-Allow-Origin", "\*")

    count := deleteAllTask()

    json.NewEncoder(w).Encode(count)

    // json.NewEncoder(w).Encode("Task not found")

}

// get all task from the DB and return it

func getAllTask() []primitive.M {

    cur, err := collection.Find(context.Background(), bson.D{{}})

    if err != nil {

        log.Fatal(err)

    }

    var results []primitive.M

    for cur.Next(context.Background()) {

        var result bson.M

        e := cur.Decode(&result)

        if e != nil {

            log.Fatal(e)

        }

        // fmt.Println("cur..>", cur, "result", reflect.TypeOf(result), reflect.TypeOf(result["\_id"]))

        results = append(results, result)

    }

    if err := cur.Err(); err != nil {

        log.Fatal(err)

    }

    cur.Close(context.Background())

    return results

}

// Insert one task in the DB

func insertOneTask(task models.ToDoList) {

    insertResult, err := collection.InsertOne(context.Background(), task)

    if err != nil {

        log.Fatal(err)

    }

    fmt.Println("Inserted a Single Record ", insertResult.InsertedID)

}

// task complete method, update task's status to true

func taskComplete(task string) {

    fmt.Println(task)

    id, \_ := primitive.ObjectIDFromHex(task)

    filter := bson.M{"\_id": id}

    update := bson.M{"$set": bson.M{"status": true}}

    result, err := collection.UpdateOne(context.Background(), filter, update)

    if err != nil {

        log.Fatal(err)

    }

    fmt.Println("modified count: ", result.ModifiedCount)

}

// task undo method, update task's status to false

func undoTask(task string) {

    fmt.Println(task)

    id, \_ := primitive.ObjectIDFromHex(task)

    filter := bson.M{"\_id": id}

    update := bson.M{"$set": bson.M{"status": false}}

    result, err := collection.UpdateOne(context.Background(), filter, update)

    if err != nil {

        log.Fatal(err)

    }

    fmt.Println("modified count: ", result.ModifiedCount)

}

// delete one task from the DB, delete by ID

func deleteOneTask(task string) {

    fmt.Println(task)

    id, \_ := primitive.ObjectIDFromHex(task)

    filter := bson.M{"\_id": id}

    d, err := collection.DeleteOne(context.Background(), filter)

    if err != nil {

        log.Fatal(err)

    }

    fmt.Println("Deleted Document", d.DeletedCount)

}

// delete all the tasks from the DB

func deleteAllTask() int64 {

    d, err := collection.DeleteMany(context.Background(), bson.D{{}}, nil)

    if err != nil {

        log.Fatal(err)

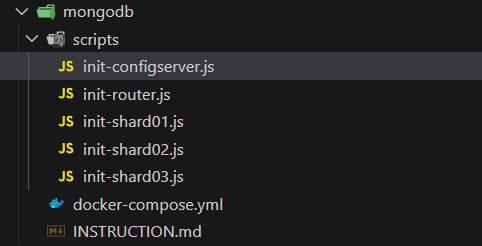
    }

    fmt.Println("Deleted Document", d.DeletedCount)

    return d.DeletedCount

}

## FOLDER mongodb/scripts



### Các file init

init-configserver.js

rs.initiate({\_id: "rs-config-server", configsvr: true, version: 1, members: [ { \_id: 0, host : 'configsvr01:27017' }, { \_id: 1, host : 'configsvr02:27017' }, { \_id: 2, host : 'configsvr03:27017' } ] })



init-shard01

rs.initiate({\_id: "rs-shard-01", version: 1, members: [ { \_id: 0, host : "shard01-a:27017" }, { \_id: 1, host : "shard01-b:27017" }, { \_id: 2, host : "shard01-c:27017" }, ] })

init-shard02

rs.initiate({ \_id: "rs-shard-02", version: 1,members: [         { \_id: 0, host : "shard02-a:27017" },         { \_id: 1, host : "shard02-b:27017" },      { \_id: 2, host : "shard02-c:27017" },      ]   })

init-shard03

rs.initiate({\_id: "rs-shard-03",version: 1,members: [{ \_id: 0, host : "shard03-a:27017" },{ \_id: 1, host : "shard03-b:27017" },{ \_id: 2, host : "shard03-c:27017" },]})

### File docker-compose

version: '3'

services:

## Router

  router01:

    image: mongo:6.0.1

    container\_name: router-01

    command: mongos --port 27017 --configdb rs-config-server/configsvr01:27017,configsvr02:27017,configsvr03:27017 --bind\_ip\_all

    ports:

      - 27117:27017

    restart: always

    volumes:

      - ./scripts:/scripts

      - mongodb\_cluster\_router01\_db:/data/db

      - mongodb\_cluster\_router01\_config:/data/configdb

  router02:

    image: mongo:6.0.1

    container\_name: router-02

    command: mongos --port 27017 --configdb rs-config-server/configsvr01:27017,configsvr02:27017,configsvr03:27017 --bind\_ip\_all

    volumes:

      - ./scripts:/scripts

      - mongodb\_cluster\_router02\_db:/data/db

      - mongodb\_cluster\_router02\_config:/data/configdb

    ports:

      - 27118:27017

    restart: always

    links:

      - router01

## Config Servers

  configsvr01:

    image: mongo:6.0.1

    container\_name: mongo-config-01

    command: mongod --port 27017 --configsvr --replSet rs-config-server

    volumes:

      - ./scripts:/scripts

      - mongodb\_cluster\_configsvr01\_db:/data/db

      - mongodb\_cluster\_configsvr01\_config:/data/configdb

    ports:

      - 27119:27017

    restart: always

    links:

      - shard01-a

      - shard02-a

      - shard03-a

  configsvr02:

    image: mongo:6.0.1

    container\_name: mongo-config-02

    command: mongod --port 27017 --configsvr --replSet rs-config-server

    volumes:

      - ./scripts:/scripts

      - mongodb\_cluster\_configsvr02\_db:/data/db

      - mongodb\_cluster\_configsvr02\_config:/data/configdb

    ports:

      - 27120:27017

    restart: always

    links:

      - configsvr01

  configsvr03:

    image: mongo:6.0.1

    container\_name: mongo-config-03

    command: mongod --port 27017 --configsvr --replSet rs-config-server

    volumes:

      - ./scripts:/scripts

      - mongodb\_cluster\_configsvr03\_db:/data/db

      - mongodb\_cluster\_configsvr03\_config:/data/configdb

    ports:

      - 27121:27017

    restart: always

    links:

      - configsvr02

## Shards

  ## Shards 01

  shard01-a:

    image: mongo:6.0.1

    container\_name: shard-01-node-a

    command: mongod --port 27017 --shardsvr --replSet rs-shard-01

    volumes:

      - ./scripts:/scripts

      - mongodb\_cluster\_shard01\_a\_db:/data/db

      - mongodb\_cluster\_shard01\_a\_config:/data/configdb

    ports:

      - 27122:27017

    restart: always

    links:

      - shard01-b

      - shard01-c

  shard01-b:

    image: mongo:6.0.1

    container\_name: shard-01-node-b

    command: mongod --port 27017 --shardsvr --replSet rs-shard-01

    volumes:

      - ./scripts:/scripts

      - mongodb\_cluster\_shard01\_b\_db:/data/db

      - mongodb\_cluster\_shard01\_b\_config:/data/configdb

    ports:

      - 27123:27017

    restart: always

  shard01-c:

    image: mongo:6.0.1

    container\_name: shard-01-node-c

    command: mongod --port 27017 --shardsvr --replSet rs-shard-01

    volumes:

      - ./scripts:/scripts

      - mongodb\_cluster\_shard01\_c\_db:/data/db

      - mongodb\_cluster\_shard01\_c\_config:/data/configdb

    ports:

      - 27124:27017

    restart: always

  ## Shards 02

  shard02-a:

    image: mongo:6.0.1

    container\_name: shard-02-node-a

    command: mongod --port 27017 --shardsvr --replSet rs-shard-02

    volumes:

      - ./scripts:/scripts

      - mongodb\_cluster\_shard02\_a\_db:/data/db

      - mongodb\_cluster\_shard02\_a\_config:/data/configdb

    ports:

      - 27125:27017

    restart: always

    links:

      - shard02-b

      - shard02-c

  shard02-b:

    image: mongo:6.0.1

    container\_name: shard-02-node-b

    command: mongod --port 27017 --shardsvr --replSet rs-shard-02

    volumes:

      - ./scripts:/scripts

      - mongodb\_cluster\_shard02\_b\_db:/data/db

      - mongodb\_cluster\_shard02\_b\_config:/data/configdb

    ports:

      - 27126:27017

    restart: always

  shard02-c:

    image: mongo:6.0.1

    container\_name: shard-02-node-c

    command: mongod --port 27017 --shardsvr --replSet rs-shard-02

    volumes:

      - ./scripts:/scripts

      - mongodb\_cluster\_shard02\_c\_db:/data/db

      - mongodb\_cluster\_shard02\_c\_config:/data/configdb

    ports:

      - 27127:27017

    restart: always

  ## Shards 03

  shard03-a:

    image: mongo:6.0.1

    container\_name: shard-03-node-a

    command: mongod --port 27017 --shardsvr --replSet rs-shard-03

    volumes:

      - ./scripts:/scripts

      - mongodb\_cluster\_shard03\_a\_db:/data/db

      - mongodb\_cluster\_shard03\_a\_config:/data/configdb

    ports:

      - 27128:27017

    restart: always

    links:

      - shard03-b

      - shard03-c

  shard03-b:

    image: mongo:6.0.1

    container\_name: shard-03-node-b

    command: mongod --port 27017 --shardsvr --replSet rs-shard-03

    volumes:

      - ./scripts:/scripts

      - mongodb\_cluster\_shard03\_b\_db:/data/db

      - mongodb\_cluster\_shard03\_b\_config:/data/configdb

    ports:

      - 27129:27017

    restart: always

  shard03-c:

    image: mongo:6.0.1

    container\_name: shard-03-node-c

    command: mongod --port 27017 --shardsvr --replSet rs-shard-03

    volumes:

      - ./scripts:/scripts

      - mongodb\_cluster\_shard03\_c\_db:/data/db

      - mongodb\_cluster\_shard03\_c\_config:/data/configdb

    ports:

      - 27130:27017

    restart: always

volumes:

  mongodb\_cluster\_router01\_db:

  mongodb\_cluster\_router01\_config:

  mongodb\_cluster\_router02\_db:

  mongodb\_cluster\_router02\_config:

  mongodb\_cluster\_configsvr01\_db:

  mongodb\_cluster\_configsvr01\_config:

  mongodb\_cluster\_configsvr02\_db:

  mongodb\_cluster\_configsvr02\_config:

  mongodb\_cluster\_configsvr03\_db:

  mongodb\_cluster\_configsvr03\_config:

  mongodb\_cluster\_shard01\_a\_db:

  mongodb\_cluster\_shard01\_a\_config:

  mongodb\_cluster\_shard01\_b\_db:

  mongodb\_cluster\_shard01\_b\_config:

  mongodb\_cluster\_shard01\_c\_db:

  mongodb\_cluster\_shard01\_c\_config:

  mongodb\_cluster\_shard02\_a\_db:

  mongodb\_cluster\_shard02\_a\_config:

  mongodb\_cluster\_shard02\_b\_db:

  mongodb\_cluster\_shard02\_b\_config:

  mongodb\_cluster\_shard02\_c\_db:

  mongodb\_cluster\_shard02\_c\_config:

  mongodb\_cluster\_shard03\_a\_db:

  mongodb\_cluster\_shard03\_a\_config:

  mongodb\_cluster\_shard03\_b\_db:

  mongodb\_cluster\_shard03\_b\_config:

  mongodb\_cluster\_shard03\_c\_db:

  mongodb\_cluster\_shard03\_c\_config:

### File hướng dẫn chạy và setup toàn bộ code

Instruction - PSS Style (Primary -Secondary - Secondary)

**=========================================**

---

**## Mongo Components**

\* Config Server (3 member replica set): `configsvr01`,`configsvr02`,`configsvr03`

\* 3 Shards (each a 3 member `PSS` replica set):

    \* `shard01-a`,`shard01-b`, `shard01-c`

    \* `shard02-a`,`shard02-b`, `shard02-c`

    \* `shard03-a`,`shard03-b`, `shard03-c`

\* 2 Routers (mongos): `router01`, `router02`

**## ✨ Steps**

**### 👉 Step 1: Start all of the containers**

```bash

docker-compose up -d

```

**### 👉 Step 2: Initialize the replica sets (config servers and shards)**

```bash

docker-compose exec configsvr01 sh -c "mongosh < /scripts/init-configserver.js"

docker-compose exec shard01-a sh -c "mongosh < /scripts/init-shard01.js"

docker-compose exec shard02-a sh -c "mongosh < /scripts/init-shard02.js"

docker-compose exec shard03-a sh -c "mongosh < /scripts/init-shard03.js"

```

**### 👉 Step 3: Initializing the router**

>Note: Wait a bit for the config server and shards to elect their primaries before initializing the router

```bash

docker-compose exec router01 sh -c "mongosh < /scripts/init-router.js"

```

**### 👉 Step 4: Enable sharding and setup sharding-key**

```bash

docker-compose exec router01 mongosh --port 27017

// Enable sharding for database `MyDatabase`

sh.enableSharding("MyDatabase")

// Setup shardingKey for collection `MyCollection`\*\*

db.adminCommand( { shardCollection: "MyDatabase.MyCollection", key: { oemNumber: "hashed", zipCode: 1, supplierId: 1 } } )

exit

```

**## 📋 Verify**

**### ✅ Verify the status of the sharded cluster**

```bash

docker-compose exec router01 mongosh --port 27017

sh.status()

exit

```

**### ✅ Verify status of replica set for each shard**

> 1 PRIMARY, 2 SECONDARY

```bash

docker exec -it shard-01-node-a bash -c "echo 'rs.status()' | mongosh --port 27017"

docker exec -it shard-02-node-a bash -c "echo 'rs.status()' | mongosh --port 27017"

docker exec -it shard-03-node-a bash -c "echo 'rs.status()' | mongosh --port 27017"

```

**### ✅ Check database status**

```bash

docker-compose exec router01 mongosh --port 27017

use MyDatabase

db.stats()

db.MyCollection.getShardDistribution()

exit

```

**## 🔎 More commands**

```bash

docker exec -it mongo-config-01 bash -c "echo 'rs.status()' | mongosh --port 27017"

docker exec -it shard-01-node-a bash -c "echo 'rs.help()' | mongosh --port 27017"

docker exec -it shard-01-node-a bash -c "echo 'rs.status()' | mongosh --port 27017"

docker exec -it shard-01-node-a bash -c "echo 'rs.printReplicationInfo()' | mongosh --port 27017"

docker exec -it shard-01-node-a bash -c "echo 'rs.printSlaveReplicationInfo()' | mongosh --port 27017"

```

## FOLDER router

package router

import (

    "golangbook/middleware"

    "golangbook/hello"

    "github.com/gorilla/mux"

)

// Router is exported and used in main.go

func Router() \*mux.Router {

    router := mux.NewRouter()

    router.HandleFunc("/api/task", middleware.GetAllTask).Methods("GET", "OPTIONS")

    router.HandleFunc("/api/task", middleware.CreateTask).Methods("POST", "OPTIONS")

    router.HandleFunc("/api/task/{id}", middleware.TaskComplete).Methods("PUT", "OPTIONS")

    router.HandleFunc("/api/undoTask/{id}", middleware.UndoTask).Methods("PUT", "OPTIONS")

    router.HandleFunc("/api/deleteTask/{id}", middleware.DeleteTask).Methods("DELETE", "OPTIONS")

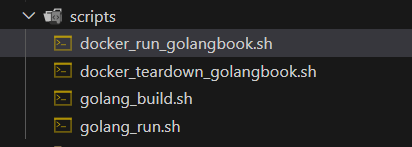
    router.HandleFunc("/api/deleteAllTask", middleware.DeleteAllTask).Methods("DELETE", "OPTIONS")

    router.HandleFunc("/health", hello.HealthHandler).Methods("GET", "OPTIONS")

    return router

}

## FOLDER scripts



### docker\_run\_golangbook.sh

#!/bin/bash

#

# From root of project, run: `bash scripts/docker\_run\_consul\_discovery.sh`

clear

if [ ! -f ../.env ]

then

  export $(cat .env | xargs)

fi

# Color Console Output

RESET='\033[0m'           # Text Reset

REDBOLD='\033[1;31m'      # Red (Bold)

GREENBOLD='\033[1;32m'    # Green (Bold)

YELLOWBOLD='\033[1;33m'   # Yellow (Bold)

CYANBOLD='\033[1;36m'     # Cyan (Bold)

if [ "$IPADDRESS" = "" ]; then

  echo ""

  echo -e "${REDBOLD}Please set your IPADDRESS environment variable in the .env file: ${RESET} E.g.: ${CYANBOLD}IPADDRESS=192.168.0.99${RESET}"

  echo ""

  exit 0

fi

if [ "$DOCKERIMAGE" = "" ]; then

  echo ""

  echo -e "${REDBOLD}Please set your DOCKERIMAGE environment variable in the .env file: ${RESET} E.g.: ${CYANBOLD}DOCKERIMAGE=lucastran/golangbook${RESET}"

  echo ""

  exit 0

fi

if [ "$SERVERPORT" = "" ]; then

  echo ""

  echo -e "${REDBOLD}Please set your SERVERPORT environment variable in the .env file:${RESET} E.g.: ${CYANBOLD}SERVERPORT=8000${RESET}"

  echo ""

  exit 0

fi

if [ "$DOCKERPORT" = "" ]; then

  echo ""

  echo -e "${REDBOLD}Please set your DOCKERPORT environment variable in the .env file:${RESET} E.g.: ${CYANBOLD}DOCKERPORT=8001${RESET}"

  echo ""

  exit 0

fi

if [ "$CONSUL\_HTTP\_PORT" = "" ]; then

  echo ""

  echo -e "${REDBOLD}Please set your CONSUL\_HTTP\_PORT environment variable in the .env file:${RESET} E.g.: ${CYANBOLD}CONSUL\_HTTP\_PORT=8500${RESET}"

  echo ""

  exit 0

fi

if [ "$FABIO\_HTTP\_PORT" = "" ]; then

  echo ""

  echo -e "${REDBOLD}Please set your FABIO\_HTTP\_PORT environment variable in the .env file:${RESET} E.g.: ${CYANBOLD}FABIO\_HTTP\_PORT=9000${RESET}"

  echo ""

  exit 0

fi

if [ "$FABIO\_DASHBOARD\_PORT" = "" ]; then

  echo ""

  echo -e "${REDBOLD}Please set your FABIO\_DASHBOARD\_PORT environment variable in the .env file:${RESET} E.g.: ${CYANBOLD}FABIO\_DASHBOARD\_PORT=9001${RESET}"

  echo ""

  exit 0

fi

if [ "$NUMBER\_OF\_INSTANCES" = "" ]; then

  echo ""

  echo -e "${REDBOLD}Please set your NUMBER\_OF\_INSTANCES environment variable in the .env file: ${RESET} E.g.: ${CYANBOLD}NUMBER\_OF\_INSTANCES=8${RESET}"

  echo ""

  exit 0

fi

echo -e "${CYANBOLD}Building Docker image:${RESET}"

echo -e "  Docker build command: ${GREENBOLD}docker build -t ${DOCKERIMAGE} .${RESET}"

echo ""

docker build -t ${DOCKERIMAGE} .

echo ""

echo -e "  ${GREENBOLD}Complete!${RESET}"

echo ""

echo -e "${CYANBOLD}Starting Consul container:${RESET}"

echo -e "  Docker run command: ${GREENBOLD}docker run -d --rm -p ${CONSUL\_HTTP\_PORT}:${CONSUL\_HTTP\_PORT} -p 8600:8600/udp --name=golangbook-consul consul:1.15 agent -server -ui -node=consul -bootstrap-expect=1 -client=0.0.0.0${RESET}"

docker run -d --rm -p ${CONSUL\_HTTP\_PORT}:${CONSUL\_HTTP\_PORT} -p 8600:8600/udp --name=golangbook-consul consul:1.15 agent -server -ui -node=consul -bootstrap-expect=1 -client=0.0.0.0 2>&1 1>/dev/null

echo -e "  ${GREENBOLD}Complete!${RESET}"

echo ""

echo -e "${CYANBOLD}Starting Fabio container:${RESET}"

echo -e "  Docker run command: ${GREENBOLD}docker run -d --rm -p ${FABIO\_HTTP\_PORT}:${FABIO\_HTTP\_PORT} -p ${FABIO\_DASHBOARD\_PORT}:${FABIO\_DASHBOARD\_PORT} -v ./fabio.properties:/etc/fabio/fabio.properties --name=golangbook-fabiolb fabiolb/fabio${RESET}"

docker run -d --rm -p ${FABIO\_HTTP\_PORT}:${FABIO\_HTTP\_PORT} -p ${FABIO\_DASHBOARD\_PORT}:${FABIO\_DASHBOARD\_PORT} -v ./fabio.properties:/etc/fabio/fabio.properties --name=golangbook-fabiolb fabiolb/fabio 2>&1 1>/dev/null

echo -e "  ${GREENBOLD}Complete!${RESET}"

echo ""

DYNAMIC\_DOCKER\_PORT=${DOCKERPORT}

for (( INSTANCE=1; INSTANCE<=NUMBER\_OF\_INSTANCES; INSTANCE++ ))

do

  echo -e "${CYANBOLD}Starting hello app container instance ${INSTANCE}/${NUMBER\_OF\_INSTANCES}:${RESET}"

  echo ""

  echo -e "  ${CYANBOLD}Starting Docker container:${RESET} ${GREENBOLD}${DOCKERIMAGE}${RESET}"

  echo -e "  Container will forward its external port to the application port: ${GREENBOLD}${DYNAMIC\_DOCKER\_PORT}->${SERVERPORT}${RESET}"

  echo -e "  Docker run command: ${GREENBOLD}docker run -d --env-file .env --rm  -p $DYNAMIC\_DOCKER\_PORT:$SERVERPORT --name golangbook-hello-${INSTANCE} -e DOCKERPORT=${DYNAMIC\_DOCKER\_PORT} -e CONSUL\_HTTP\_ADDR=${IPADDRESS}:${CONSUL\_HTTP\_PORT} -e FABIO\_HTTP\_ADDR=${IPADDRESS}:${FABIO\_HTTP\_PORT} $DOCKERIMAGE${RESET}"

  docker run -d --env-file .env --rm -p $DYNAMIC\_DOCKER\_PORT:$SERVERPORT --name golangbook-hello-${INSTANCE} -e DOCKERPORT=${DYNAMIC\_DOCKER\_PORT} -e CONSUL\_HTTP\_ADDR=${IPADDRESS}:${CONSUL\_HTTP\_PORT} -e FABIO\_HTTP\_ADDR=${IPADDRESS}:${FABIO\_HTTP\_PORT} $DOCKERIMAGE 2>&1 1>/dev/null

  echo -e "  ${GREENBOLD}Complete!${RESET}"

  echo ""

  ((DYNAMIC\_DOCKER\_PORT=DYNAMIC\_DOCKER\_PORT+1))

done

echo ""

docker ps | grep golangbook-

echo ""

echo -e "${GREENBOLD}Complete!${RESET}"

echo ""

echo -e "${CYANBOLD}Dashboards may take a few seconds to become available:${RESET}"

echo -e "  ${CYANBOLD}Console Dashboard is avaiable:${RESET}   http://${IPADDRESS}:${CONSUL\_HTTP\_PORT}/ui/dc1/services ${RESET}"

echo -e "  ${CYANBOLD}Fabio Dashboard is avaiable:${RESET}     http://${IPADDRESS}:${FABIO\_DASHBOARD\_PORT}/routes ${RESET}"

echo -e "  ${CYANBOLD}Fabio Load Balanced Endpoint is:${RESET} http://${IPADDRESS}:${FABIO\_HTTP\_PORT} ${RESET}"

echo ""

echo ""

docker build -t docker-react-image:1.0 ./client

docker run -d -p 4000:80 --name docker-react-container docker-react-image:1.0

### docker\_teardown\_golangbook.sh

#!/bin/bash

#

# From root of project, run: `bash scripts/docker\_teardown\_consul\_discovery.sh`

clear

if [ ! -f ../.env ]

then

  export $(cat .env | xargs)

fi

# Color Console Output

RESET='\033[0m'           # Text Reset

REDBOLD='\033[1;31m'      # Red (Bold)

GREENBOLD='\033[1;32m'    # Green (Bold)

YELLOWBOLD='\033[1;33m'   # Yellow (Bold)

CYANBOLD='\033[1;36m'     # Cyan (Bold)

echo ""

echo -e "${CYANBOLD}Stopping all golangbook containers:${RESET}"

echo -e "  Docker stop command: ${GREENBOLD}docker stop \$(docker ps | grep -e golangbook- -e docker-react- | awk '{print \$1}')${RESET}"

echo ""

echo ""

ERROR=$(docker stop $(docker ps | grep -e golangbook- -e docker-react-| awk '{print $1}') 2>&1 1>/dev/null)

status=$?

if test $status -ne 0

then

  echo -e "${CYANBOLD}  No containers found to stop.${RESET}"

  echo ""

  exit 0

fi

echo ""

# docker ps

echo ""

echo -e "${GREENBOLD}Complete!${RESET}"

echo ""

### golang\_build.sh

#!/bin/bash

#

# From root of project, run: `bash scripts/golang\_build.sh`

clear

# Color Console Output

RESET='\033[0m'           # Text Reset

REDBOLD='\033[1;31m'      # Red (Bold)

GREENBOLD='\033[1;32m'    # Green (Bold)

YELLOWBOLD='\033[1;33m'   # Yellow (Bold)

CYANBOLD='\033[1;36m'     # Cyan (Bold)

echo -e "${CYANBOLD}Formatting \*.go files...${RESET}"

for i in \*.go \*\*/\*.go ; do

  ERROR=$(gofmt -w "$i" 2>&1 1>/dev/null)

  status=$?

  if test $status -ne 0

  then

    echo -e "${REDBOLD}...Error: 'gofmt' command failed!${RESET}"

    echo -e "${REDBOLD}ERROR: ${ERROR}${RESET}"

    echo ""

    exit 1

  fi

  echo "  Formatted: $i"

done;

echo -e "${GREENBOLD}...Complete.${RESET}"

echo ""

echo -e "${CYANBOLD}Building Go app:${RESET} ${GREENBOLD}go build -o bin/golangbook .${RESET}"

ERROR=$(go build -o bin/golangbook . 2>&1 1>/dev/null)

status=$?

if test $status -ne 0

then

  echo -e "${REDBOLD}ERROR:  'go build' command failed!${RESET}"

  echo -e "${REDBOLD}ERROR: ${ERROR}${RESET}"

  echo ""

  exit 1

fi

echo ""

echo -e "${GREENBOLD}Complete: Built native go binary.${RESET}"

echo ""

### golang\_run.sh

#!/bin/bash

#

# From root of project, run: `bash scripts/golang\_run.sh`

clear

# Color Console Output

RESET='\033[0m'           # Text Reset

REDBOLD='\033[1;31m'      # Red (Bold)

GREENBOLD='\033[1;32m'    # Green (Bold)

YELLOWBOLD='\033[1;33m'   # Yellow (Bold)

CYANBOLD='\033[1;36m'     # Cyan (Bold)

echo -e "${CYANBOLD}Formatting \*.go files...${RESET}"

for i in \*.go \*\*/\*.go ; do

  gofmt -w "$i"

  status=$?

  if test $status -ne 0

  then

    echo -e "${REDBOLD}...Error: 'gofmt' command failed!${RESET}"

    echo ""

    exit 1

  fi

  echo "Formatted: $i"

done;

echo -e "${GREENBOLD}...Complete${RESET}"

echo ""

echo -e "${CYANBOLD}Building Go app:${RESET} ${GREENBOLD}go build -o bin/golangbook .${RESET}"

go build -o bin/golangbook .

status=$?

if test $status -ne 0

then

  echo -e "${REDBOLD}...Error: 'go build' command failed!${RESET}"

  echo ""

  exit 1

fi

echo -e "${CYANBOLD}Running Go app:${RESET} ${GREENBOLD}./bin/golangbook${RESET}"

./bin/golangbook

## FOLDER structs

package structs

import "github.com/google/uuid"

type IP struct {

    Query string

}

type HTTPResponse struct {

    Status      int

    Application string

    IP  string

    UUID        uuid.UUID

    Data        string

}

## FILE .ENV

NUMBER\_OF\_INSTANCES=8

SERVERPORT=8000

DOCKERPORT=8001

DOCKERIMAGE=lucastran/golangbook

IPADDRESS=192.168.219.25

CONSUL\_HTTP\_PORT=8500

FABIO\_DASHBOARD\_PORT=9001

FABIO\_HTTP\_PORT=9000

DB\_URI=mongodb://192.168.219.25:27117,192.168.219.25:27118

DB\_NAME=MyDatabase

DB\_COLLECTION\_NAME=todolist

## FILE DOCKERFILE

FROM golang:alpine as golangbook

RUN addgroup -S golangbook \

  && adduser -S -u 10000 -g golangbook golangbook

WORKDIR /go/src/app

COPY . .

RUN CGO\_ENABLED=0 go install -ldflags '-extldflags "-static"' -tags timetzdata

RUN echo ${PWD} && ls -lR

FROM scratch

COPY --from=alpine:latest /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs/

COPY --from=golangbook /go/bin/golangbook /golangbook

COPY --from=golangbook /etc/passwd /etc/passwd

# COPY .env /golangbook

USER golangbook

ENTRYPOINT ["/golangbook"]

## FILE FABIO.PROPERTIES

# REF: https://github.com/fabiolb/fabio/blob/master/fabio.properties

proxy.addr = :9000

registry.consul.register.addr = :9001

ui.addr = :9001

registry.consul.addr = 192.168.219.25:8500

proxy.strategy = rr

## FILE MAIN.GO

package main

import (

    "embed"

    "fmt"

    "log"

    "net/http"

    "os"

    "os/signal"

    "strconv"

    "syscall"

    "github.com/google/uuid"

    "golangbook/config"

    "golangbook/consul"

    "golangbook/router"

)

var myUUID = uuid.New()

//go:embed .env

var envVarsFile embed.FS

func main() {

    config.EnvVarsFile = envVarsFile

    r := router.Router()

    \_, err := config.AppConfig()

    if err != nil {

        log.Fatal("Error: config.AppConfig()")

    }

    signalChannel := make(chan os.Signal, 1)

    signal.Notify(signalChannel, syscall.SIGINT, syscall.SIGTERM)

    go func() {

        <-signalChannel

        fmt.Println("\nShutting down.")

        os.Exit(0)

    }()

    consul.ServiceRegistryWithConsul(config.IPAddress, config.ServerPort, myUUID)

    fmt.Printf("Starting Hello Server: %v:%v", config.IPAddress, config.ServerPort)

    http.ListenAndServe(":"+strconv.Itoa(config.ServerPort), r)

}

## FILE MAKEFILE

SHELL=/bin/bash

.DEFAULT\_GOAL := list

# Color Console Output

RESET=\033[0m

REDBOLD=\033[1;31m

GREENBOLD=\033[1;32m

YELLOWBOLD=\033[1;33m

CYANBOLD=\033[1;36m

list:

    @echo ""

    @echo -e "${GREENBOLD}Targets in this Makefile:${RESET}"

    @echo ""

    @LC\_ALL=C $(MAKE) -pRrq -f $(lastword $(MAKEFILE\_LIST)) : 2>/dev/null | awk -v RS= -F: '/(^|\n)# Files(\n|$$)/,/(^|\n)# Finished Make data base/ {if ($$1 !~ "^[#.]" && $$1 !~ "^[list.]" && $$1 !~ "^[always.]") {print "make "$$1}}' | sort | egrep -v -e '^[^[:alnum:]]' -e '^$@$$'

    @echo ""

    @echo "For details on these commands, see the bash scripts in the 'scripts/' directory."

    @echo ""

golang-run:

    scripts/golang\_run.sh

golang-build:

    scripts/golang\_build.sh

docker-run-golangbook:

    scripts/docker\_run\_golangbook.sh

docker-teardown-golangbook:

    scripts/docker\_teardown\_golangbook.sh