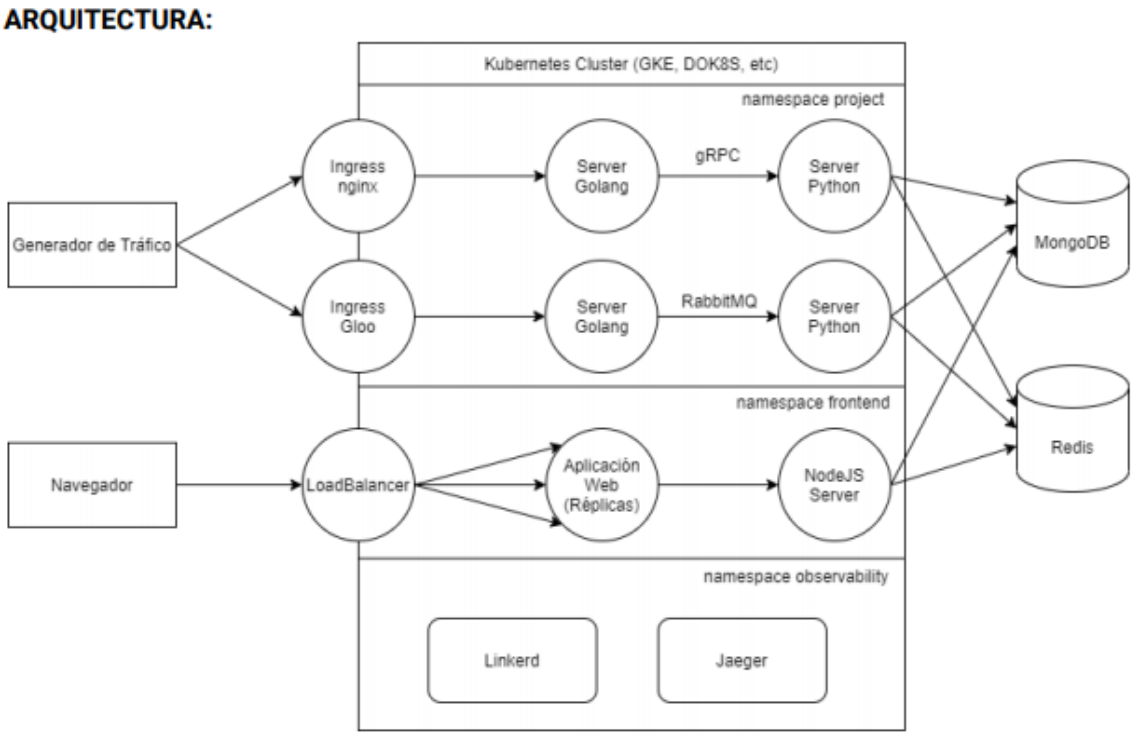
WUNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
SISTEMAS OPERATIVOS 1
SEGUNDO SEMESTRE 2020
ING. ALLAN MORATAYA
TUTOR ACADÉMICO: SEBASTIAN SANCHEZ

PROYECTO

MANUAL TECNICO

MAYNOR DAVID SALGUERO GUILLÉN – 201504192
SERGIO ALEJANDRO SILVA ROLDAN – 201610657
EDGAR ARNOLDO ALDANA ARRIOLA - 201602797
GUATEMALA 14 DE NOVIEMBRE DEL 2020

Arquitectura Utilizada:



Generador de trafico

Consiste en realizar un programa escrito en Go que tendrá el papel de generador de tráfico. Este tráfico será enviado a dos balanceadores de carga.

Variables globales:

```go
type AutoGenerated struct {
    Caso []CasoJ `json:"Caso"`
}

type CasoJ struct {
    Name         string `json:"name"`
    Location     string `json:"location"`
    Age          int    `json:"age"`
    Infectedtype string `json:"infectedtype"`
    State        string `json:"state"`
}

var clienteHttp = &http.Client{}
```

Métodos:

```go
func peticion(caso CasoJ, url string) {
    println("Go rutna en ejecucion")
    usuarioComoJson, err := json.Marshal(caso)
    if err != nil {
        // Maneja el error de acuerdo a tu situación
        log.Fatalf("Error creando petición: %v", err)
    }
    peticion, err := http.NewRequest("POST", url, bytes.NewBuffer(usuarioComoJson))
    if err != nil {
        // Maneja el error de acuerdo a tu situación
        log.Fatalf("Error creando petición: %v", err)
    }
    peticion.Header.Add("Content-Type", "application/json")
    respuesta, err := clienteHttp.Do(peticion)
    if err != nil {
        // Maneja el error de acuerdo a tu situación
        log.Fatalf("Error haciendo petición: %v", err)
    }

    // No olvides cerrar el cuerpo al terminar
    defer respuesta.Body.Close()
    //
}
```

Esta función se encarga de enviar la petición al servidor que se ha indicado por el usuario. Necesita como parámetros el objeto caso que se va a enviar y una url con la dirección del servicio REST.

```go
func main() {
    var cantidadCasos int
    var cantidadGorutinas int

    var URL string
    var ruta string
    for {
        println("1- Ingrese el URL del balanceador de carga a donde se desea enviar informacion.")
        fmt.Scanf("%s", &URL)

        println("2- Ingrese la cantidad de gorutinas a ejecutar.")
        fmt.Scanf("%d", &cantidadGorutinas)
        fmt.Scanf("%d", &cantidadGorutinas)

        println("3- Ingrese la cantidad de solicitudes que tiene el archivo.")
        fmt.Scanf("%d", &cantidadCasos)
        fmt.Scanf("%d", &cantidadCasos)

        println("4- Ingrese la ruta del archivo que se desea cargar.")
        fmt.Scanf("%s", &ruta)
        fmt.Scanf("%s", &ruta)

        jsonFile, err := os.Open(ruta)
        if err != nil {
            fmt.Println(err)
        }
        fmt.Println("Successfully Opened users.json")
        defer jsonFile.Close()
        byteValue, _ := ioutil.ReadAll(jsonFile)
        var data AutoGenerated
        json.Unmarshal(byteValue, &data)

        for i := 0; i < cantidadCasos; i++ {
            go peticion(data.Caso[i], URL)
            time.Sleep(500 * time.Millisecond)
        }
    }
```

Metodo principal donde se despliega los datos requeridos para realizar la generación de trafico, luego se envia por medio de hilos los casos simultaneamente a la dirección indicara.

Formato del Json de entrada:

```json
{
  "Casos": [
    {
      "name": "Pablo Mendoza",
      "location": "Guatemala City",
      "age": 35,
      "infectedtype": "communitary",
      "state": "asymptomatic"
    },
    {
      "name": "Jonás Santos",
      "location": "Guatemala City",
      "age": 23,
      "infectedtype": "communitary",
      "state": "symptomatic"
    }
  ]
}
```

Cliente GO con gRPC

Importaciones externas:

```
"github.com/gorilla/mux"
"google.golang.org/grpc"
pb "google.golang.org/grpc/examples/helloworld/helloworld"
```

Función de error para imprimir en pantalla mensajes de error:

```
func failOnError(err error, msg string) {
    if err != nil {
        log.Fatalf("%s: %s", msg, err)
    }
}
```

Función de prueba de servicio rest para dirección (‘/’)

```
func indexRoute(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Api funcionando")
}
```

Función de petición Post para insertar datos en bases de datos:

```
func insertData(w http.ResponseWriter, r *http.Request) {
    // Obtención objeto json convertido a string
    reqBody, err := ioutil.ReadAll(r.Body)
    if err != nil {
        fmt.Fprintf(w, "Insert valid data")
    }
    bodyString := string(reqBody)

    // Set up a connection to the server.
    conn, err := grpc.Dial("localhost:50051", grpc.WithInsecure(), grpc.With
    Block())
    if err != nil {
        log.Fatalf("did not connect: %v", err)
    }
    defer conn.Close()
    c := pb.NewGreeterClient(conn)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second)
    defer cancel()

    resp, err := c.SayHello(ctx, &pb.HelloRequest{Name: bodyString})
    if err != nil {
        log.Fatalf("could not greet: %v", err)
    }
    log.Printf(resp.GetMessage())
}
```

método principal, donde se levanta el router para el servicio REST:

```
func main() {
    log.Printf("Server inicializado")
    router := mux.NewRouter().StrictSlash(true)
    router.HandleFunc("/", indexRoute)
    router.HandleFunc("/insertarCaso", insertData).Methods("POST")
    log.Fatal(http.ListenAndServe(":9000", router))
}
```

Servidor Python con gRPC

Variables globales:

```
#Datos de conexión base de datos redis
r = redis.Redis(host = '3.21.97.128', port = 80)

#Datos de conexión base de datos mongo
client = MongoClient('18.217.45.142:80')
db = client["infectados"]
col = db["caso"]
```

Método principal para inicializar servidor, que espera el envió de datos utilizando protocolo http2

```
if __name__ == '__main__':
    print('Server inicializado')
    print('Esperando mensajes...')
    logging.basicConfig()
    serve()
```

Metodo que inicializa el servidor en el puerto 50051

```
def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    helloworld_pb2_grpc.add_GreeterServicer_to_server(Greeter(), server)
    server.add_insecure_port('[::]:50051')
    server.start()
    server.wait_for_termination()
```

Metodo de envoi de datos por medio de gRPC

```
    def SayHello(self, request, context):
        cadena = request.name
        jsoncadena = json.loads(cadena)
        print (jsoncadena)
        # Agragar datos a la base de datos mongo
        try:
            col.insert_one(jsoncadena)
            print("Datos insertados de forma correcta en mongo")
```

```
        except:
            print('Error al insertar los datos a mongo')
        # Agragar datos a la base de datos redis
        try:
            r.lpush("caso", dumps(jsoncadena))
            print("Datos insertados de forma correcta en redis")
        except:
            print('Error al insertar los datos a redis')
        print('Esperando mensajes...')
        return helloworld_pb2.HelloReply(message=request.name)
```

Servidor GO con RabbitMQ

Importaciones externas:

```
    "github.com/gorilla/mux"
    "github.com/streadway/amqp"
```

Función de error para imprimir en pantalla mensajes de error:

```
func failOnError(err error, msg string) {
    if err != nil {
        log.Fatalf("%s: %s", msg, err)
    }
}
```

Función de prueba de servicio rest para dirección (‘/’)

```
func indexRoute(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Api funcionando")
}
```

Función de petición Post para insertar datos en bases de datos:

```
func insertData(w http.ResponseWriter, r *http.Request) {
    // Obtención objeto json convertido a string
    reqBody, err := ioutil.ReadAll(r.Body)
    if err != nil {
        fmt.Fprintf(w, "Insert valid data")
    }
    bodyString := string(reqBody)

    // Creación conexión a cola de RabbitMQ
    conn, err := amqp.Dial("amqp://guest:guest@3.139.57.229:5672/")
    failOnError(err, "Fallo al conectar con el servidor de RabbitMQ")
    defer conn.Close()
```

```
    // Apertura del canal
    ch, err := conn.Channel()
    failOnError(err, "Fallo al abrir el canal")
    defer conn.Close()

    // Declaración de la cola
    q, err := ch.QueueDeclare(
        "sopes1",
        false,
        false,
        false,
        false,
        nil,
    )
    failOnError(err, "Fallo al crear cola")

    // Escritura en cola del objeto json en string
    err = ch.Publish(
        "",
        q.Name,
        false,
        false,
        amqp.Publishing{
            ContentType: "text/plain",
            Body:        []byte(bodyString),
        },
    )
    failOnError(err, "Fallo al enviar el mensaje")
    log.Printf("Enviado: %s", bodyString)
}
```

Método principal, donde se levanta el router para el servicio REST:

```
func main() {
    log.Printf("Server inicializado")
    router := mux.NewRouter().StrictSlash(true)
    router.HandleFunc("/", indexRoute)
    router.HandleFunc("/insertarCaso", insertData).Methods("POST")
    log.Fatal(http.ListenAndServe(":7000", router))
}
```

Servidor Python con RabbitMQ

Inicialización del servidor:

```
print('Server inicializado')
channel.basic_consume(queue='sopes1', on_message_callback=callback, auto_ack
=True)
```

```
print('Esperando mensajes...')
channel.start_consuming()
```

Variables globales:

```
#Datos de conexión base de datos redis
r = redis.Redis(host = '3.21.97.128', port = 80)

#Datos de conexión base de datos mongo
client = MongoClient('18.217.45.142:80')
db = client["infectados"]
col = db["caso"]

#Datos de conexión pika lectura RabbitMQ
connection = pika.BlockingConnection(pika.ConnectionParameters(host='3.139.57.229', port='5672'))
channel = connection.channel()
channel.queue_declare(queue='sopes1')
```

método de inserción al encontrar objetos en la cola de rabbit:

```
def callback (ch, method, properties, body):
    cadena = str(body, 'utf-8')
    jsoncadena = json.loads(cadena)
    print (jsoncadena)
    # Agragar datos a la base de datos mongo
    try:
        col.insert_one(jsoncadena)
        print("Datos insertados de forma correcta en mongo")
    except:
        print('Error al insertar los datos a mongo')
    # Agragar datos a la base de datos redis
    try:
        r.lpush("caso", dumps(jsoncadena))
        print("Datos insertados de forma correcta en redis")
    except:
        print('Error al insertar los datos a redis')
    print('Esperando mensajes...')
```

Servidor NODE JS

Variables globales:

```
const express = require("express");
const bodyParser = require("body-parser");
```

```
const cors = require("cors");
const app = express();

const db = require('./app/models/index');
```

```
app.use(cors());

app.use(bodyParser.json({limit: '10mb', extendido: true}));

require("./app/routes/caso")(app);

const dbConnect = () => {
    db.mongoose
        .connect("mongodb://18.217.45.142:80/infectados")
        .catch(err => {
            console.error("** No se pudo conectar a la base de datos **");
            console.error(err);
            process.exit();
        });
    console.log("Conectado a la base de datos")
};

const server = app.listen(process.env.PORT || 5000, () => {
    api.dbConnect(db.url);
});

let api = { server: server, app: app, dbConnect: dbConnect };

module.exports = api;
```

Obtención de todos los productos:

```
exports.getAll = (_req, res) => {
    db.mongoose.connection.db.collection("caso", function(err, collection){
        collection.conut({column_name: "location"}).toArray(function(err, da
ta){
            if (err){
                return res
                    .status(400)
                    .send({ message: "Error obteniendo datos." });
            }
            return res
                .status(200)
                .send({ message: "datos devueltos", data: data });
        })
```

```
    });
};
```

```
exports.getAllMongo = (_req, res) => {
    db.mongoose.connection.db.collection("caso", function(err, collection){
        collection.find({}).toArray(function(err, data){
            if (err){
                return res
                    .status(400)
                    .send({ message: "Error obteniendo datos." });
            }
            return res
                .status(200)
                .send({ message: "datos devueltos", data: data });
        })
    });
};
```

Obtención por rango de edad:

```
exports.getRangoEdad = (_req, res) => {
    db.mongoose.connection.db.collection("caso", function(err, collection){
        collection.find({}).toArray(function(err, data){
            if (err){
                return res
                    .status(400)
                    .send({ message: "Error obteniendo datos." });
            }

            let rangos = [{edades:"0 - 10",cantidad:0},
            {edades:"11 - 20",cantidad:0},{edades:"21 - 30",cantidad:0},{eda
des:"31 - 40",cantidad:0},{edades:"41 - 50",cantidad:0},
            {edades:"51 - 60",cantidad:0},{edades:"61 - 70",cantidad:0},{eda
des:"71 - 80",cantidad:0},{edades:"81 - 90",cantidad:0},
            {edades:"91 - 100",cantidad:0},{edades:"+100",cantidad:0}]

            for(let i = 0; i < data.length ;i++){
                if(data[i].age<11){
                    rangos[0].cantidad += 1
                }
                else if(data[i].age<21){
                    rangos[1].cantidad += 1
                }
                else if(data[i].age<31){
                    rangos[2].cantidad += 1
```

```
                }
                else if(data[i].age<41){
                    rangos[3].cantidad += 1
                }
                else if(data[i].age<51){
                    rangos[4].cantidad += 1
                }
                else if(data[i].age<61){
                    rangos[5].cantidad += 1
                }
                else if(data[i].age<71){
                    rangos[6].cantidad += 1
                }
                else if(data[i].age<81){
                    rangos[7].cantidad += 1
                }
                else if(data[i].age<91){
                    rangos[8].cantidad += 1
                }
                else if(data[i].age<101){
                    rangos[9].cantidad += 1
                }
                else{
                    rangos[10].cantidad += 1
                }
            }

            return res
                .status(200)
                .send({ message: "datos devueltos", data: rangos });
        })
    });
};
```

Declaración de router de servidor rest

```
module.exports = app => {
    const caso = require("../controllers/caso");

    var router = require("express").Router();

    // Devuelve todos los casos
    router.get("/getAllMongo", caso.getAllMongo);

    // Devuelve los 3 departamentos con mas casos de COVID-19
    router.get("/getTop3", caso.getTop3);
```

```
    // Devuelve los departamentos que tienen casos del COVID-19
    router.get("/getDepartamentos", caso.getAllDepartamentos);

    // Devuelve los casos de COVID-19 por rango de edades
    router.get("/getRangoEdad", caso.getRangoEdad);

    // Devuelve el ultimo valor agregado
    router.get("/getRedis", caso.getRedis);

    // La ruta de la api de productos sera url/productos
    app.use("/", router);
};
```