MANUAL TECNICO

GRUPO NO.6

Integrantes:

Jimmy Yorbany Noriega Chavez 200915691
Douglas Omar Arreola Martinez 201603168
Ricardo Antonio Alvarado Ramirez 201603157

Descripción

EL proyecto “Cloud Traffic 1.0”, es un sistema basado en el manejo de la información de las personas que han sido vacunadas en el territorio guatemalteco, en el cual el administrador debera recolectar la información a travez de archivos en formato Json, que es una estructura de archivos, se mostrará un ejemplo, y estarán registrados en la nube atravez de un servicio externo, y se podra visualizar en una pagina web, para que pueda ser interpretadas la información.

Procesos de lo que se compone:

- **Carga de información.**
- **Proceso de inicio de trafico.**
- **Visualización de Informacion.**

Carga de Información:

El administrador del sistema, debera crear un archivo con la información que desea registrar en el sistema, para eso se habilitará una carpaeta en la nube, que para eso se usa el proveedor de Google Cloud Plataform, para gestionar todo el proyecto, estos archivos deben tener la siguiente estructura.

```
[
  {
    "name":"Pablo Mendoza",
    "location":"Ciudad Guatemala",
    "age":35,
    "vaccine_type":"Sputnik V",
    "n_dose": 2
},
...
]
```

Este archivo debera crearse con el siguiente nombre: traffic.json y sera registrado en la carpeta compartida asignada.

Luego de copiar el archivo a cargar, se accedera a la aplicación de trafico del sistema, para poder iniciar la carga de los archivos al sistema.

Proceso de inicio de trafico:

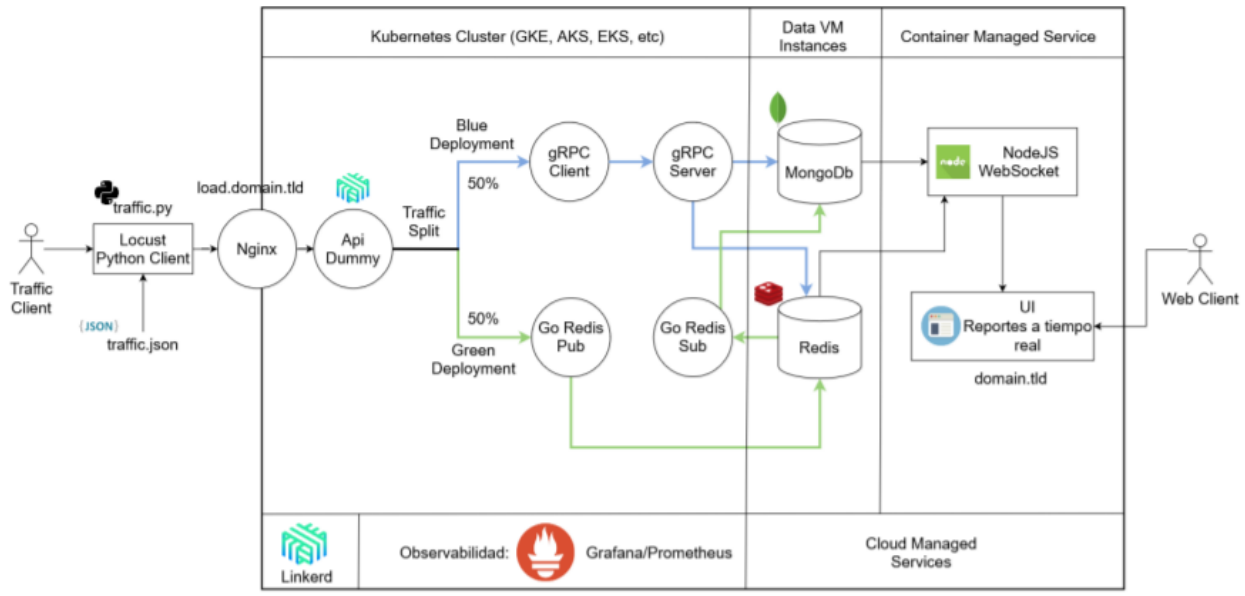
Para este proceso se ha decidio crear un servicio a travez de locust, este se ejecutará a nivel local en la maquina del aministrador, este servicio de trafico esta basado en el lenguaje python, y un archivo shell que permita ejecutar el incio de trafico, a continuación se muestra el archivo con su respectiva configuracion

```
import json
from random import random, randrange
from sys import getsizeof
from locust import HttpUser, task, between
#librerias que usa el sistema
debug = True


def printDebug(msg): # metodo para mostrar mensajes el debug

    if debug:
        print(msg)


def loadData(): #metodo para cargar la informacion
    try:

        with open("traffic.json", 'r') as data_file: #buscamos y recorremos el
archivo

            array = json.loads(data_file.read())
            return array

        print (f'>> Reader: Datos cargados correctamente, {len(array)} datos ->
{getsizeof(array)} bytes.')
    except Exception as e: # si ocurre una excepcion

        print (f'>> Reader: No se cargaron los datos {e}')
        return []

array = loadData()


class Reader(): #clase para leer el archivo


    def pickRandom(self): #para poder acceder de manera aleatoria a los datos

        length = len(array)
```

```
        if (length > 0):

            random_index = randrange(0, length - 1) if length > 1 else 0


            return array.pop(random_index) #luego de tomar el valor del array, lo
libera para ya no ser tomado en cuenta


        else:

            print (">> Reader: No hay más valores para leer en el archivo.")

            return None #finalzia la lectura del archivo

class MessageTraffic(HttpUser):

    wait_time = between(0.1, 0.9) #intervalo entre cada peticion


    def on_start(self):
        print (">> MessageTraffic: Iniciando el envio de tráfico") #mensaje para
definir el inicio del trafico

        self.reader = Reader()



    @task
    def PostMessage(self):

        random_data = self.reader.pickRandom() #tomamos el valor random


        if (random_data is not None):

            data_to_send = json.dumps(random_data) #leemos el archivo

            printDebug (data_to_send)


            self.client.post("/entrada", json=random_data) #endpoint que se va a
consumir


        else:
            print(">> MessageTraffic: Envio de tráfico finalizado, no hay más datos
que enviar.")

            self.stop(True)
```

Nota: El código contiene sus comentarios respecto a los metodos utilizados, para poder ejecutar el archivo de locust usamos la instrucción ./run.sh para poder obtener la ip donde se publicará locust.

Para el control los datos se usara la siguiente arquitectura



En las cuales podemos se usaran la siguiente distribución de rutas, para guardar y mostrar la información:

Primera ruta de acceso:

1. Generador de Tráfico
2. Ingress
3. go-grpc-client
4. go-grpc-server
5. Escribir en la base de datos NoSQL

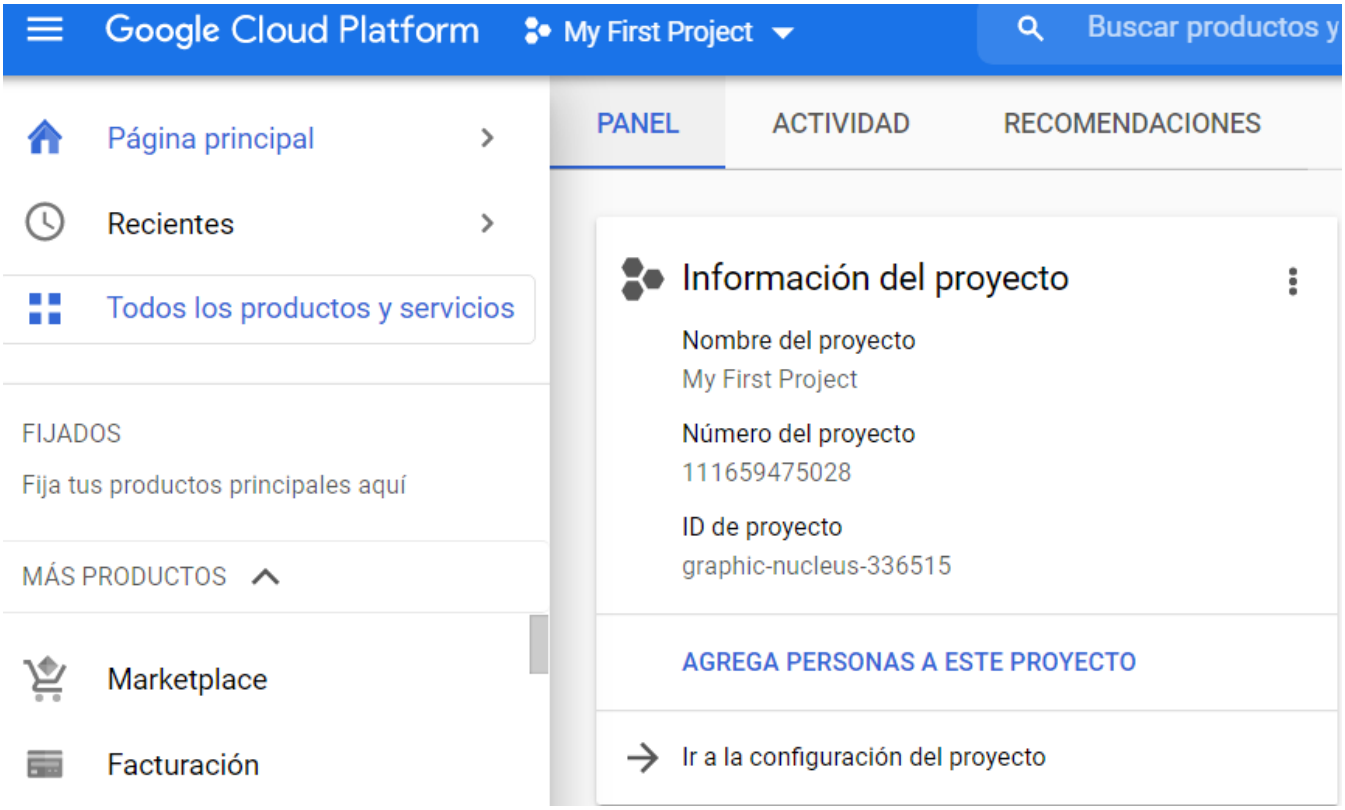
Segunda ruta de acceso:

1. Generador de Tráfico
2. Ingress
3. redis-pub
4. redis-sub
5. Escribir en la base de datos NoSQL

La plataforma en la nube seleccionada para la gestión del sistema es Google Cloud Plataform (GCP), en la cual se usará un cluster de kubernetes para el despliegue completo de la aplicación, excepto locust, que manejarára toda las configuraciones, como primer paso tendremos el

Ingress atravez de Ngnix y Linkerd, lo cuales los procesos de instalacion e injeccion se encuentran en el repositorio oficial, para la recepcion de las peticiones, para esto instalamos y configuramos el ngnix-ingress, con los siguientes archivo yaml.

GCP:



Archivos de configuracion:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dummy
  name: dummy
  namespace: project
spec:
  replicas: 1
  selector:
    matchLabels:
      app: dummy
  template:
    metadata:
      labels:
        app: dummy
    spec:
```

```
      containers:
      - image: douglasmartinez97/go-redis-pub
        imagePullPolicy: Always
        name: dummy
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dummy
  name: dummy
  namespace: project
spec:
  ports:
  - port: 3050
    protocol: TCP
    targetPort: 3050
  selector:
    app: dummy
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: dummy-ingress
  namespace: project
  annotations:
    nginx.ingress.kubernetes.io/configuration-snippet: |
      proxy_set_header l5d-dst-override
$service_name.$namespace.svc.cluster.local:$service_port;
      grpc_set_header l5d-dst-override
$service_name.$namespace.svc.cluster.local:$service_port;
spec:
  ingressClassName: nginx
  rules:
    - host: load.sopes1grupo6.tk
      http:
        paths:
          - backend:
              service:
                name: dummy
                port:
                  number: 3050
            path: /
            pathType: Prefix
```

En este archivo creamos la configuracion para el dummy.yaml para poder admnistrar el acceso y a donde alimentara en el host creado.

```yaml
apiVersion: split.smi-spec.io/v1alpha2
kind: TrafficSplit
metadata:
  name: function-split
  namespace: project
spec:
  service: dummy
  backends:
  - service: clientegrpc
    weight: 50
  - service: redispub
    weight: 50
```

El archivo de configuración splitter.yaml nos permite controlar la administracion del trafico de las peticiones que entren por al servicio de dummy, de esto el 50% utilizara el servicio que contiene el despliegue de cliente de grpc y el pubisher de redis. Estos servicios son solo los endpoint de ingreso, ya que para poder administrar completamente estas dos rutas, se usan 4 despliegues que contienen la siguiente informacion.

- Despliegue de cliente grpc y servidor de grpc

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: clientegrpc
  name: clientegrpc
  namespace: project
spec:
  replicas: 1
  selector:
    matchLabels:
      app: clientegrpc
  template:
    metadata:
      labels:
        app: clientegrpc
    spec:
      containers:
      - image: jimmynoriega/clientegrpc
        imagePullPolicy: Always
        name: clientegrpc
---
```

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: clientegrpc
  name: clientegrpc
  namespace: project
spec:
  ports:
  - port: 3050
    protocol: TCP
    targetPort: 3050
  selector:
    app: clientegrpc
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: servidorgrcp
  name: servidorgrcp
  namespace: project
spec:
  replicas: 1
  selector:
    matchLabels:
      app: servidorgrcp
  template:
    metadata:
      labels:
        app: servidorgrcp
    spec:
      containers:
      - image: jimmynoriega/servidorgrpc
        imagePullPolicy: Always
        name: servidorgrcp
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: servidorgrcp
  name: servidorgrcp
  namespace: project
spec:
  ports:
  - port: 50051
    protocol: TCP
    targetPort: 50051
  selector:
```

```
    app: servidorgrcp
```

- Despliegue de redis publisher y suscriber

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: redispub
  name: redispub
  namespace: project
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redispub
  template:
    metadata:
      labels:
        app: redispub
    spec:
      containers:
      - image: douglasmartinez97/go-redis-pub
        imagePullPolicy: Always
        name: redispub
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: redispub
  name: redispub
  namespace: project
spec:
  ports:
  - port: 3050
    protocol: TCP
    targetPort: 3050
  selector:
    app: redispub
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: redissub
  name: redissub
```

```
  namespace: project
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redissub
  template:
    metadata:
      labels:
        app: redissub
    spec:
      containers:
      - image: douglasmartinez97/go-redis-sub
        imagePullPolicy: Always
        name: redissub
```

Nota: Para los procesos de grpc y redis, fueron creadas basadas en imágenes de docker publicadas en dockerhub con la configuracion distroless para su uso en el entorno de kubernetes.

Para el front end, se uso un sistema de basado en lenguaje react y para su publicacion se utilizo el siguiente archivo aplication.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: aplicacion
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: servidor
  name: servidor
  namespace: aplicacion
spec:
  replicas: 1
  selector:
    matchLabels:
      app: servidor
  template:
    metadata:
      labels:
        app: servidor
    spec:
```

```
      containers:
      - image: rickg96/so1-server
        imagePullPolicy: Always
        name: servidor
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: servidor
  name: servidor
  namespace: aplicacion
spec:
  ports:
  - port: 10000
    protocol: TCP
    targetPort: 10000
  selector:
    app: servidor
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: cliente
  name: cliente-svc
  namespace: aplicacion
spec:
  type: LoadBalancer
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: cliente
```

Para el backend de la aplicacion se utilizo el siguiente archivo server.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: cliente
  name: cliente
  namespace: aplicacion
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      app: cliente
  template:
    metadata:
      labels:
        app: cliente
    spec:
      containers:
      - image: rickg96/so1-client
        imagePullPolicy: Always
        name: cliente
```

Código para la creacion del cliente y servidor de grpc

1. Cliente:

```
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "strconv"
    "time"

    "github.com/gorilla/mux"

    "github.com/gorilla/handlers"


    "google.golang.org/grpc"
    pb "google.golang.org/grpc/examples/helloworld/helloworld"
)

type caso struct {
    Name        string `json:"name"`
    Location    string `json:"location"`
    Age         int    `json:"age"`
    VaccineType string `json:"vaccine_type"`
    Dosis       int     `json:"n_dose"`
}
```

```
const (
    address     = "servidorgrcp:50051"
    defaultName = "world"
)

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

    body, err := ioutil.ReadAll(r.Body)
    if err != nil {

        panic(err)

    }

    var nuevo caso

    err = json.Unmarshal(body, &nuevo)

    if err != nil {

        panic(err)

    }

    //Variable que almacenará el nuevo caso en formato json
    var jsonstr = string(`{"name":"` + nuevo.Name + `","location":"` +
nuevo.Location +  `","age":` + strconv.Itoa(nuevo.Age) + `,"vaccine_type":"` +
nuevo.VaccineType + `","n_dose":` + strconv.Itoa(nuevo.Dosis)+`}`)

    //Metodo gRPC

    /*
        Crea una conexión con el servidor
        grpc.WithInsecure() os permite realizar una conexión sin tener que suar SSL
    */

    conn, err := grpc.Dial(address, grpc.WithInsecure(),
grpc.FailOnNonTempDialError(true), grpc.WithBlock())

    if err != nil {
        log.Printf("No se conectó: %v", err)
    }

    //Realiza la desconexión al final de la ejecución
    defer conn.Close()

    //Crea un cliente con el cual podemos escuchar
    //Se envía como parametro el Dial de gRPC
    c := pb.NewGreeterClient(conn)
```

```
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)

    defer cancel()

    ra, err := c.SayHello(ctx, &pb.HelloRequest{Name: jsonstr})

    if err != nil {

        log.Printf("could not greet: %v", err)

    }
    log.Printf("Greeting: %s", ra.GetMessage())
}

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

    fmt.Fprintf(w, "Conexión exitosa...")
    log.Println("Si inicio el server")

}

//Función principal
func main() {
    router := mux.NewRouter().StrictSlash(true)
    router.HandleFunc("/entrada", CasoNuevo).Methods("POST")
    router.HandleFunc("/inicio", Inicio).Methods("GET")
    log.Println("Si inicio el server")
    log.Fatal(http.ListenAndServe(":3050",
handlers.CORS(handlers.AllowedHeaders([]string{"X-Requested-With", "Content-Type",
"Authorization"}), handlers.AllowedMethods([]string{"GET", "POST", "PUT", "HEAD",
"OPTIONS"}), handlers.AllowedOrigins([]string{"*"}))(router)))
}
```

2. Servidor:

```
package main

import (
    "context"
```

```
    "encoding/json"
    "fmt"
    "log"
    "net"
    //"os"

    "github.com/go-redis/redis/v8"
    //"github.com/joho/godotenv"

    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
    "google.golang.org/grpc"
    pb "google.golang.org/grpc/examples/helloworld/helloworld"
)

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

type casoJSON struct {
    Name        string `json:"name"`
    Location    string `json:"location"`
    Age         int    `json:"age"`
    VaccineType string `json:"vaccine_type"`
    Dosis       int     `json:"n_dose"`
}

//var ctx = context.Background()

const (
    port = ":50051"
)

// server is used to implement helloworld.GreeterServer.
type server struct {
    pb.UnimplementedGreeterServer
}

func keyEdad(age int) string {

    if age >= 0 && age <= 11 {
        return "ninos"
    } else if age >= 12 && age <= 18 {
        return "adolescentes"
    } else if age >= 19 && age <= 26 {
        return "jovenes"
    } else if age >= 27 && age <= 59 {
        return "adultos"
```

```
    } else if age >= 60 {
        return "vejez"
    } else {
        return "vejez"
    }
}
// SayHello implements helloworld.GreeterServer
func (s *server) SayHello(ctx context.Context, in *pb.HelloRequest)
(*pb.HelloReply, error) {
    //log.Printf("Received: %v", in.GetName())



    //************************************************************MONGO
    //Toma el json y lo deserealiza
    data := in.GetName()
    info := casoJSON{}
    fmt.Println(info)
    json.Unmarshal([]byte(data), &info)

    //Mongo
    /*err := godotenv.Load()
    if err != nil {
        log.Println("Error loading .env file")
    }*/
    //fmt.Println(os.Getenv("MONGO_ADDRESS"))
    clienteMongo :=
options.Client().ApplyURI("mongodb://adming6:mongog6so1py2@34.136.166.39:27017")
    cliente, err := mongo.Connect(context.TODO(), clienteMongo)
    if err != nil {
        log.Println(err)
    }

    //insertar los datos en mongo
    collection := cliente.Database("sopes1-data").Collection("registros")
    //collection := cliente.Database("sopes1").Collection("Vacunados")
    insertResult, err := collection.InsertOne(context.TODO(), info)
    if err != nil {
        log.Fatal(err)
    }
    log.Println(insertResult)


    //************************************************************REDIS

    //Conexion a Redis
    /*rdb := redis.NewClient(&redis.Options{
        Addr: "35.238.119.39:6379",
        DB:   0, // default DB
    })*/
```

```
    if info.Age != 0 && info.Name != "" {
        opt, err :=
redis.ParseURL("redis://default:redisg6so1py2@34.136.166.39:6379")
        if err != nil {
            fmt.Println("Error con URL de redis en handler")
            log.Println(err)
        }
        rdb := redis.NewClient(opt)

        // Contador de edades
        _, err = rdb.Incr(ctx, keyEdad(info.Age)).Result()
        if err != nil {
            fmt.Println("Error con Incr")
            log.Println(err)
        }

        // Agregar Nombre a la lista
        _, err = rdb.LPush(ctx, "lNombres", info.Name).Result()
        if err != nil {
            fmt.Println("Error con LPush")
            log.Println(err)
        }
        rdb.LTrim(ctx, "lNombres", 0, 4)

        // Publicar Mensaje
        //rdb.Publish(ctx, "Registros", string(body))
        fmt.Println("Registro Publicado")
    } else {
        fmt.Println("Algunos datos incompletos")
    }


    return &pb.HelloReply{Message: "Hello " + in.GetName()}, nil
}

func main() {

    lis, err := net.Listen("tcp", port)

    if err != nil {
        log.Printf("Falló al escuchar: %v", err)
    }

    s := grpc.NewServer()

    pb.RegisterGreeterServer(s, &server{})

    fmt.Println(">> SERVER: El servidor está escuchando...")
```

```
    if err := s.Serve(lis); err != nil {
        log.Printf("Falló el servidor: %v", err)
    }
}
```

Nota: Para el cliente servidor se uso apoyo en la librería proto-grpc, para controlar la interaccion entre el servidor y el cliente.

Codigo para la ruta de redis:

- Publisher.go

```
package main

import (
    "fmt"
    "log"
    //"os"
    "context"

    "net/http"
    "io/ioutil"
    "encoding/json"


    "github.com/gorilla/mux"
    //"github.com/joho/godotenv"
    "github.com/go-redis/redis/v8"
)

var ctx = context.Background()

type Register struct {
    Name            string  `json:name`
    Location        string  `json:location`
    Age             int     `json:age`
    Vaccine_type    string  `json:vaccine_type`
    N_dose          int     `json:n_dose`
}

func middlewareCors(next http.Handler) http.Handler {
    return http.HandlerFunc (
        func(w http.ResponseWriter, req *http.Request) {
            w.Header().Set("Access-Control-Allow-Origin", "*")
            w.Header().Set("Access-Control-Allow-Credentials", "true")
```

```
            w.Header().Set("Access-Control-Allow-Methods", "POST, GET, OPTIONS, PUT,
DELETE")
            w.Header().Set("Access-Control-Allow-Headers", "Accept, Content-Type,
Content-Length, Accept-Encoding, X-CSRF-Token, Authorization")

            next.ServeHTTP(w, req)
        })
}

func enableCORS(router *mux.Router) {
    router.PathPrefix("/").HandlerFunc(func(w http.ResponseWriter, req *http.Request)
{
            w.Header().Set("Access-Control-Allow-Origin", "*")
    }).Methods(http.MethodOptions)

    router.Use(middlewareCors)
}

func keyEdad(age int) string {

    if age >= 0 && age <= 11 {
        return "ninos"
    } else if age >= 12 && age <= 18 {
        return "adolescentes"
    } else if age >= 19 && age <= 26 {
        return "jovenes"
    } else if age >= 27 && age <= 59 {
        return "adultos"
    } else if age >= 60 {
        return "vejez"
    } else {
        return "vejez"
    }
}

func publisherHandler(w http.ResponseWriter, r *http.Request) {
    body, err := ioutil.ReadAll(r.Body)
    if err != nil {
        fmt.Println("Error con Read Body")
        log.Fatal(err)
    }

    var nuevo Register
    err = json.Unmarshal(body, &nuevo)
    if err != nil {
        fmt.Println("Error con Unmarshal")
        log.Fatal(err)
    }

    if nuevo.Age != 0 && nuevo.Name != "" {
```

```
        //opt, err := redis.ParseURL(os.Getenv("REDIS_ADDRESS"))
        opt, err := redis.ParseURL("redis://default:redisg6so1py2@34.136.166.39:6379")
        //opt, err := redis.ParseURL("redis://172.17.0.3:6379")
        if err != nil {
            fmt.Println("Error con URL de redis en handler")
            log.Fatal(err)
        }
        rdb := redis.NewClient(opt)

        // Contador de edades
        _, err = rdb.Incr(ctx, keyEdad(nuevo.Age)).Result()
        if err != nil {
            fmt.Println("Error con Incr")
            log.Fatal(err)
        }

        // Agregar Nombre a la lista
        _, err = rdb.LPush(ctx, "lNombres", nuevo.Name).Result()
        if err != nil {
            fmt.Println("Error con LPush")
            log.Fatal(err)
        }
        rdb.LTrim(ctx, "lNombres", 0, 4)

        // Publicar Mensaje
        rdb.Publish(ctx, "Registros", string(body))
        fmt.Println("Registro Publicado")
    } else {
        fmt.Println("Algunos datos incompletos")
    }
}

func main() {
    /*
    err := godotenv.Load()
    if err != nil {
        log.Fatal("Error loading .env file")
    }
    */

    router := mux.NewRouter().StrictSlash(true)
    enableCORS(router)

    router.HandleFunc("/entrada", publisherHandler).Methods("POST")

    fmt.Println("Servidor pub en puerto 3050")
    if err := http.ListenAndServe(":3050", router); err != nil {
        log.Fatal(err)
        return
    }
```

```
}
```

- Suscriber.go

```
package main

import (
    "fmt"
    "context"
    "log"
    //"os"
    "encoding/json"

    //"github.com/joho/godotenv"
    "github.com/go-redis/redis/v8"
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
)

type Register struct {
    Name            string  `json:name`
    Location        string  `json:location`
    Age             int     `json:age`
    Vaccine_type    string  `json:vaccine_type`
    N_dose          int     `json:n_dose`
}

var ctx = context.Background()

func main() {
    // Carga de archivo .env
    /*
    err := godotenv.Load()
    if err != nil {
        log.Fatal("Error loading .env file")
    }
    */

    // Conexion Redis
    //opt, err := redis.ParseURL(os.Getenv("REDIS_ADDRESS"))
    opt, err := redis.ParseURL("redis://default:redisg6so1py2@34.136.166.39:6379")
    //opt, err := redis.ParseURL("redis://172.17.0.3:6379")
    if err != nil {
        fmt.Println("Error con URL de redis")
        log.Fatal(err)
    }
```

```
    rdb := redis.NewClient(opt)
    fmt.Println("Conectado a Redis")

    sub := rdb.Subscribe(ctx, "Registros")
    fmt.Println("Suscrito al canal de Redis")

    //Conexion Mongo
    //cOptions := options.Client().ApplyURI(os.Getenv("MONGO_ADDRESS"))
    cOptions :=
options.Client().ApplyURI("mongodb://adming6:mongog6so1py2@34.136.166.39:27017")
    //cOptions := options.Client().ApplyURI("mongodb://172.17.0.2:27017")
    mongoClient, err := mongo.Connect(context.TODO(), cOptions)
    if err != nil {
        fmt.Println("Error creando cliente de Mongo")
        log.Println(err)
    }

    err = mongoClient.Ping(ctx, nil)
    if err != nil {
        fmt.Println("Error conectando al servidor")
        log.Fatal(err)
    }
    fmt.Println("Conectado a MongoDB")
    myMongoDB := mongoClient.Database("sopes1-data")
    collection := myMongoDB.Collection("registros")

    for {
        // Subscriber de Redis
        msg, err := sub.ReceiveMessage(ctx)
        if err != nil {
            fmt.Println("Error reciviendo datos")
            log.Fatal(err)
        } else {
            var nuevo Register
            err = json.Unmarshal([]byte(msg.Payload), &nuevo)
            if err != nil {
                fmt.Println("Error Convirtiendo datos")
                log.Fatal(err)
            }
            fmt.Print("Mensaje recibido del canal '" + msg.Channel + "': ")
            fmt.Println(nuevo.Name + " - " + nuevo.Location)

            //Insertar MongoDB
            _, err := collection.InsertOne(context.TODO(), nuevo)
            if err != nil {
                fmt.Println("Error Insertando datos en MongoDB")
                log.Fatal(err)
            }
            fmt.Println("\tRegistro insertado en MongoDB")
        }
```

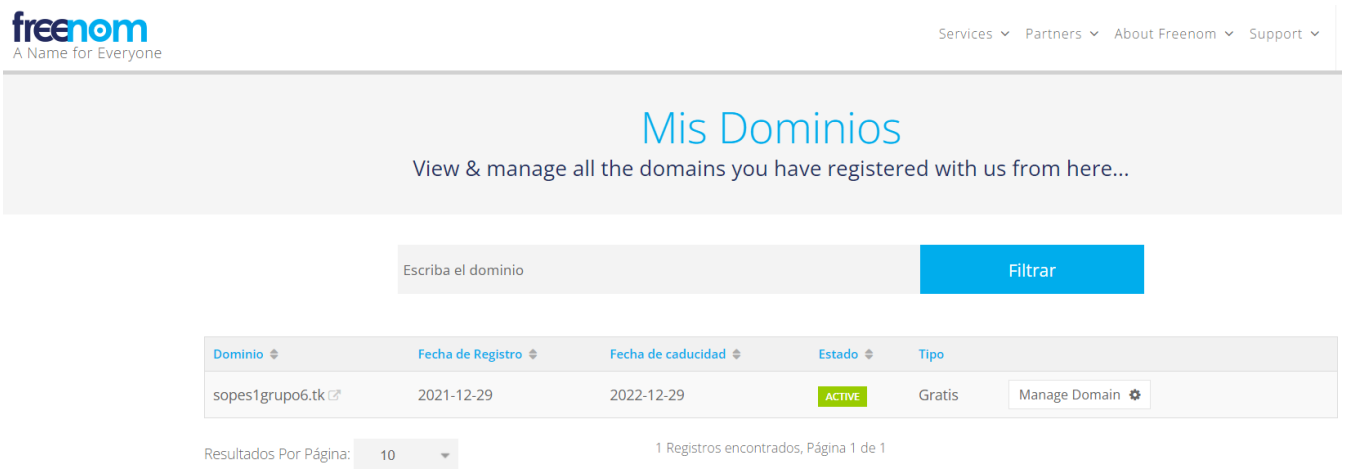
```
    }
}
```

ALMACENAMIENTO PERSISTENTE:

Para manejar la persistencia de la informacion se trabajaron 2 bases de datos para la replicacion de la información entre cada una de ellas, estas son mongo y redis. Esto permite tener la misma información en cada una de las bases de datos.

DOMINIO:

La gestion del dominio se hizo a traves de Frenom, configurando los servidores para el acceso de la misma.



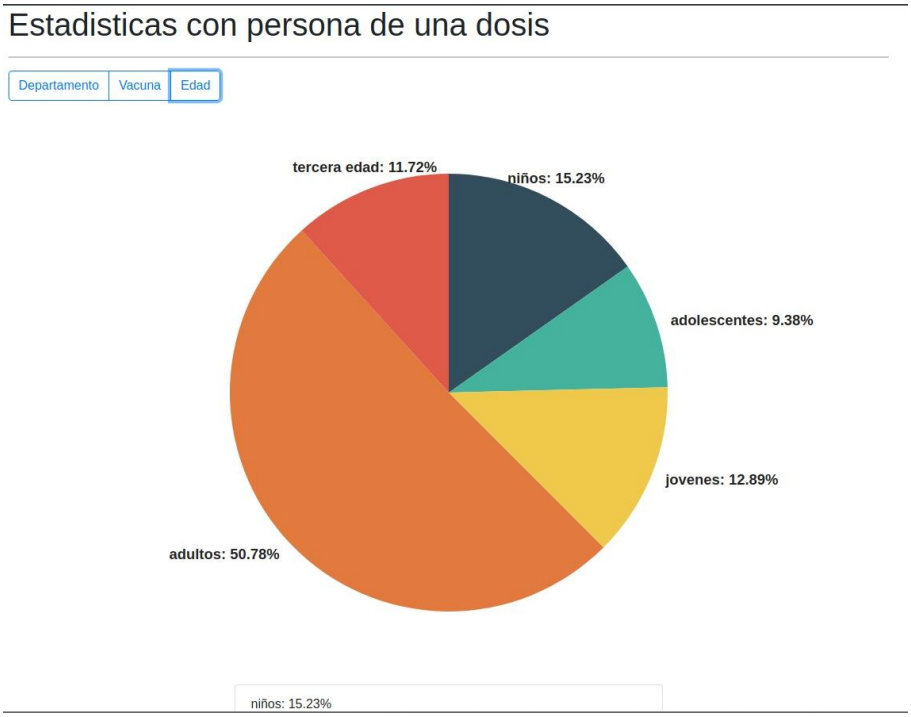
El front-end basado en react se comporta en el endpoint de la siguiente manera:

Primera Dosis

Segunda Dosis

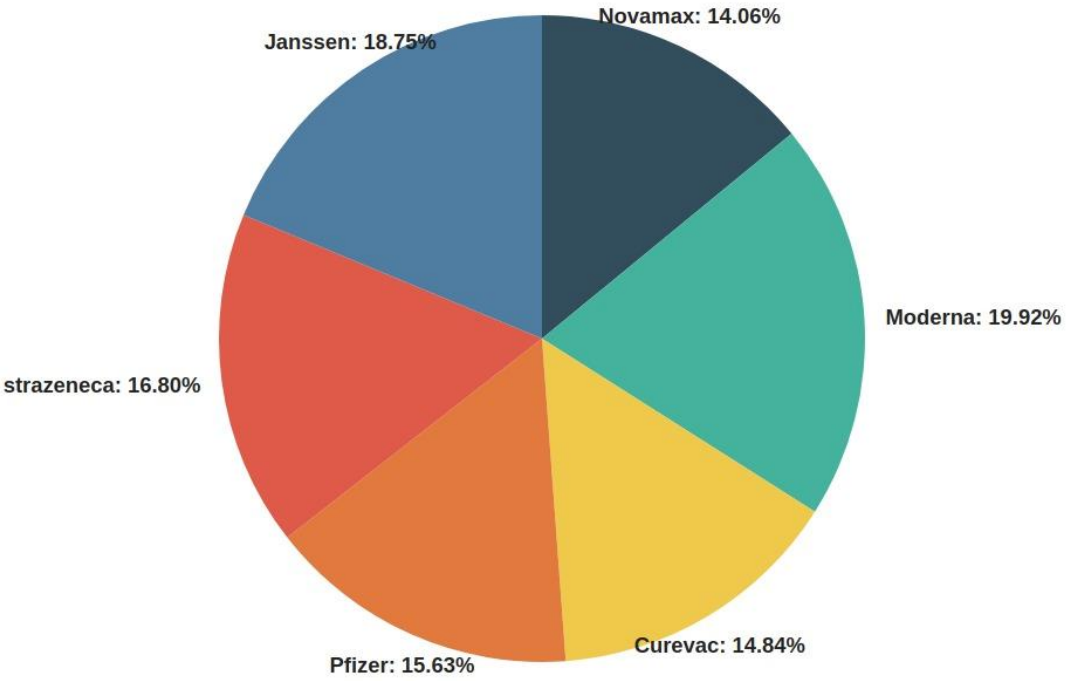
Base de datos mongodb

Base de datos Redis



Estadisticas con persona de una dosis

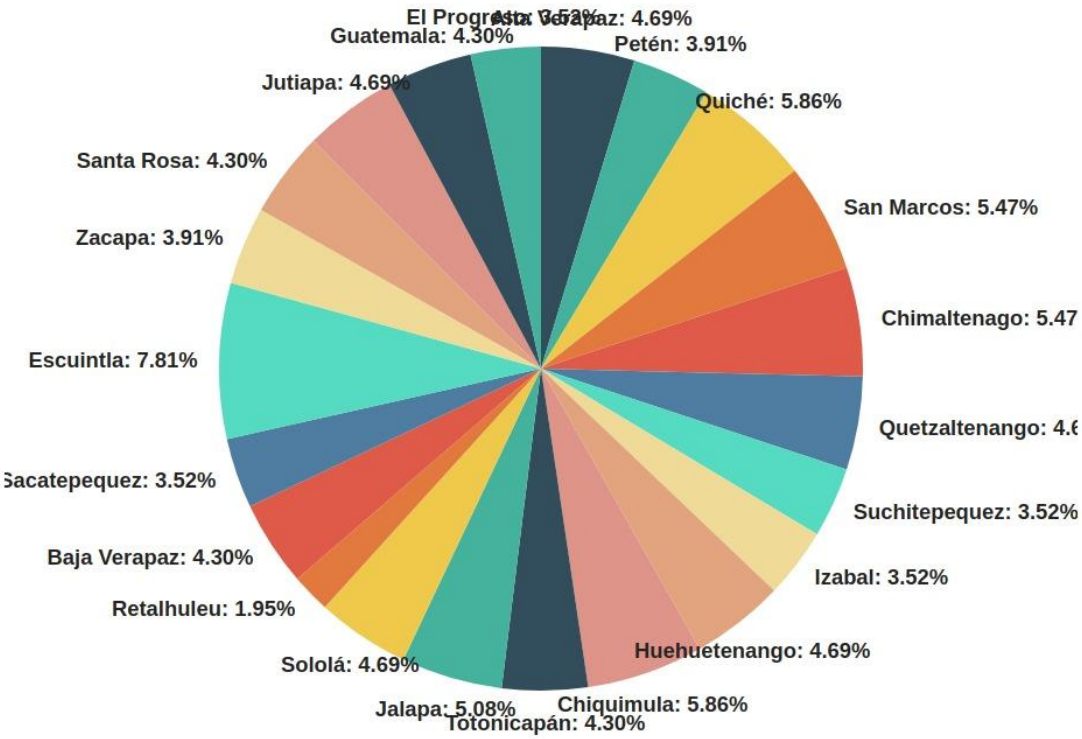
Departamento | Vacuna | Edad



Novamax: 14.06%

Estadisticas con persona de una dosis

Departamento | Vacuna | Edad



Proyecto 2 Una Dosis Dos Dosis Mongo data Redis data

Estadisticas de personas vacunadas

Google Chrome

#	Nombre	Edad	Departamento	Vacuna	No. Dosis
0	raticate	46	Chiquimula	Janssen	2
1	ampharos	57	Petén	Astrazeneca	2
2	blastoise	53	Sacatepequez	Curevac	0
3	seel	27	Escuintla	Curevac	2
4	caterpie	66	Quiché	Curevac	0
5	golduck	14	Jutiapa	Astrazeneca	0
6	machamp	63	Huehuetenango	Novamax	1
7	hoppip	39	Guatemala	Moderna	1
8	magneton	11	Jutiapa	Janssen	0
9	venusaur	12	El Progreso	Curevac	2
10	golduck	12	Chimaltenago	Novamax	0
11	mr-mime	19	Sacatepequez	Curevac	2
12	dratini	60	Guatemala	Janssen	0
13	drowzee	69	Petén	Pfizer	2
14	arbok	46	Jutiapa	Pfizer	2
15	pidgey	25	San Marcos	Janssen	0

Vacunados dosis completa por edades