Сервис «Менеджер паролей GophKeeper»

Функциональность сервиса

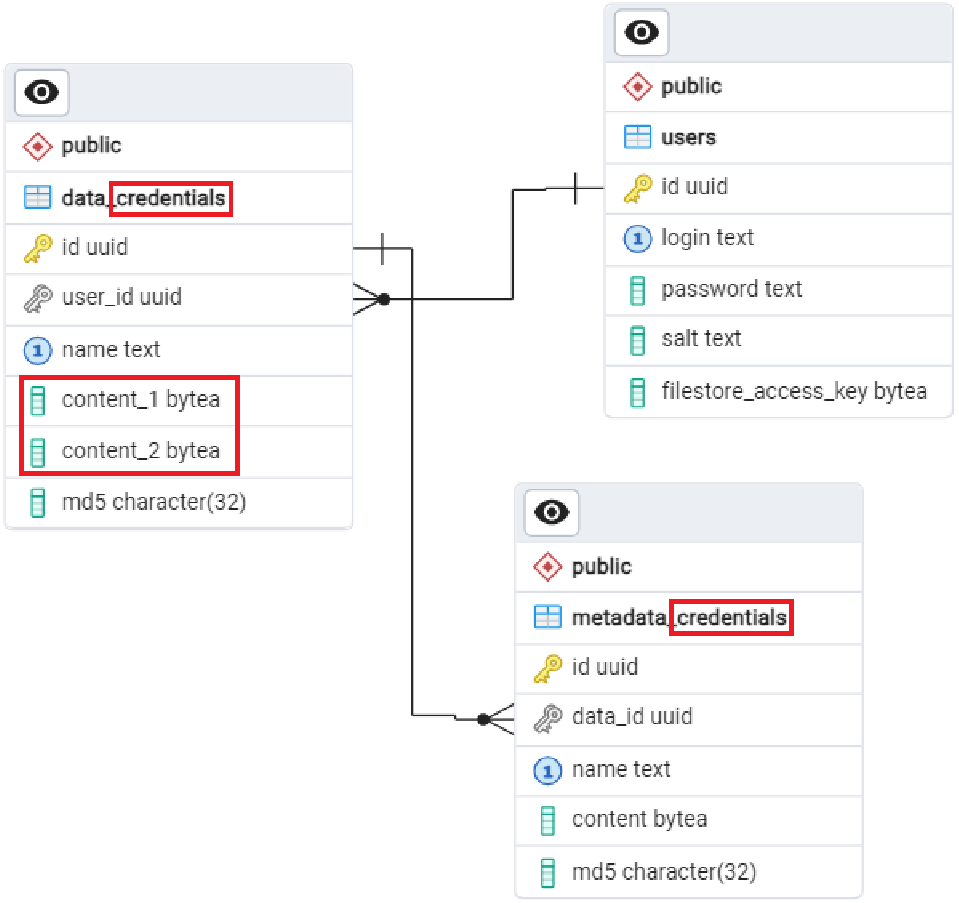
– Регистрация/логин пользователя

– Запись/удаление данных/метаданных

– Чтение данных с фильтрацией по тегу и/или метаданным

– Загрузка/скачивание файлов

База данных



Таблицы:

_credentials

    login, password

_credit_card

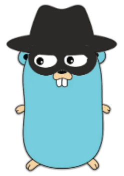
    number, until, holder

_text_note

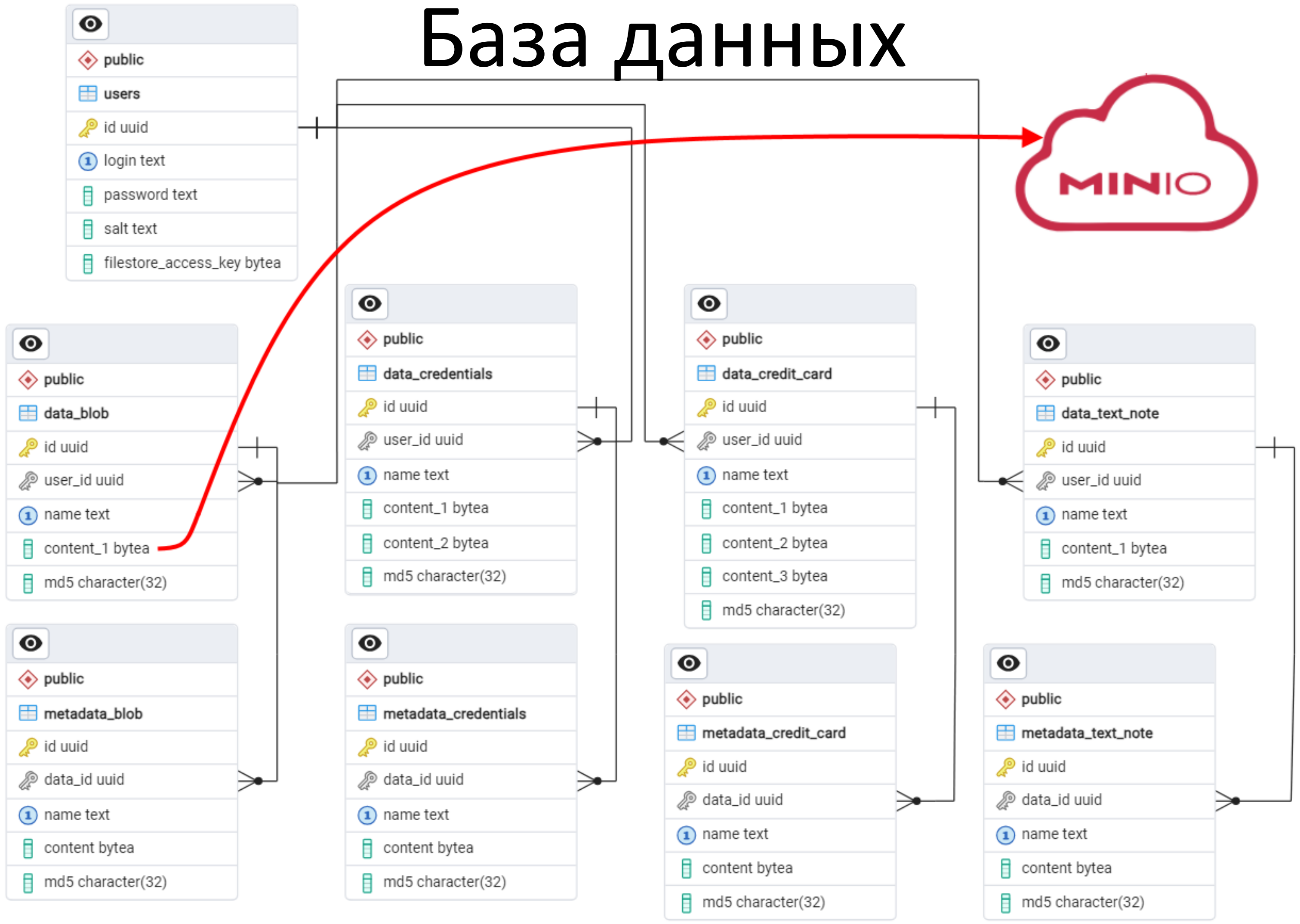
    text

_blob

    file_name



База данных



public
users
- id uuid
- login text
- password text
- salt text
- filestore_access_key bytea

public
data_blob
- id uuid
- user_id uuid
- name text
- content_1 bytea
- md5 character(32)

public
data_credentials
- id uuid
- user_id uuid
- name text
- content_1 bytea
- content_2 bytea
- md5 character(32)

public
data_credit_card
- id uuid
- user_id uuid
- name text
- content_1 bytea
- content_2 bytea
- content_3 bytea
- md5 character(32)

public
data_text_note
- id uuid
- user_id uuid
- name text
- content_1 bytea
- md5 character(32)

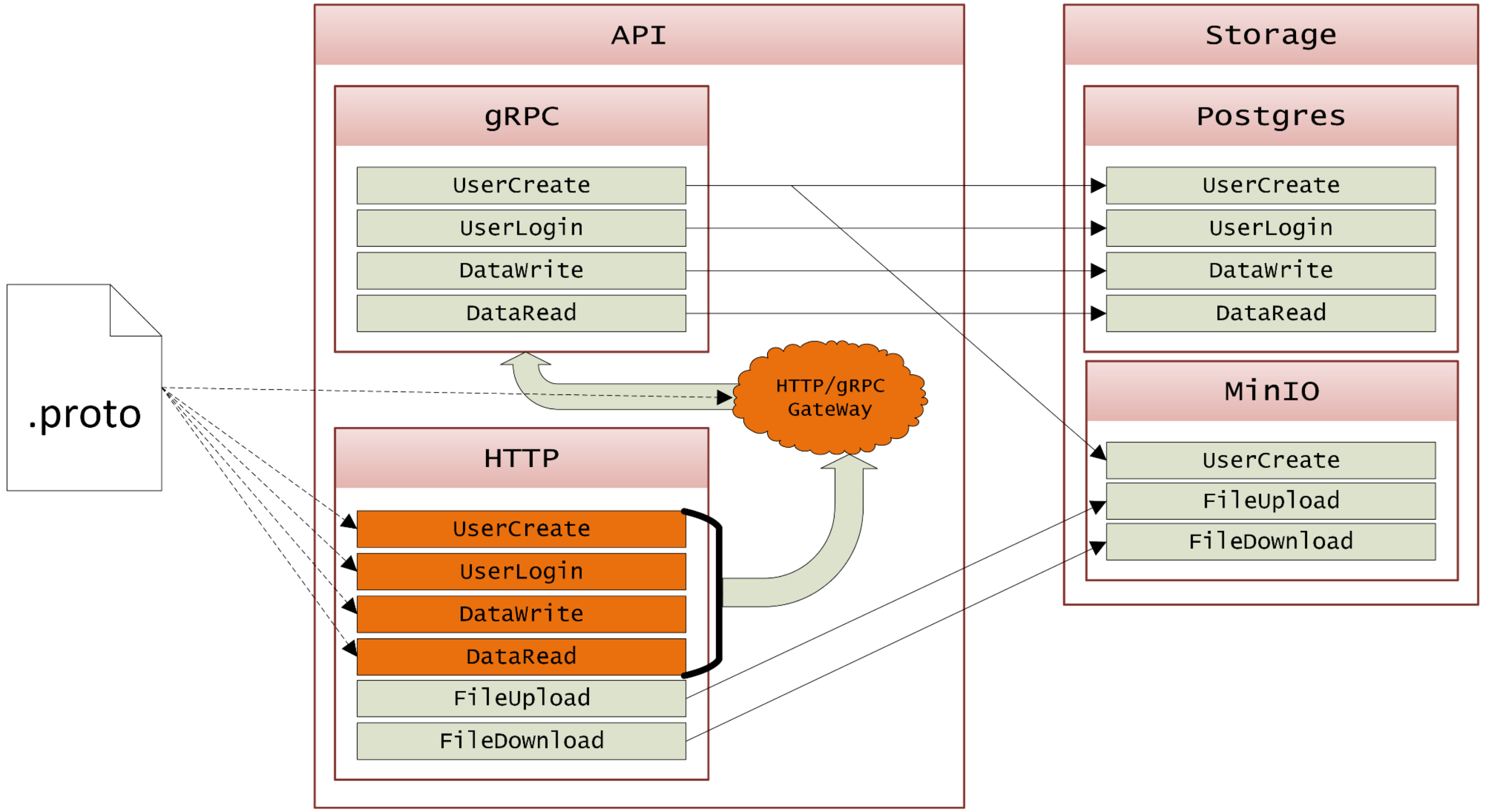
public
metadata_blob
- id uuid
- data_id uuid
- name text
- content bytea
- md5 character(32)

public
metadata_credentials
- id uuid
- data_id uuid
- name text
- content bytea
- md5 character(32)

public
metadata_credit_card
- id uuid
- data_id uuid
- name text
- content bytea
- md5 character(32)

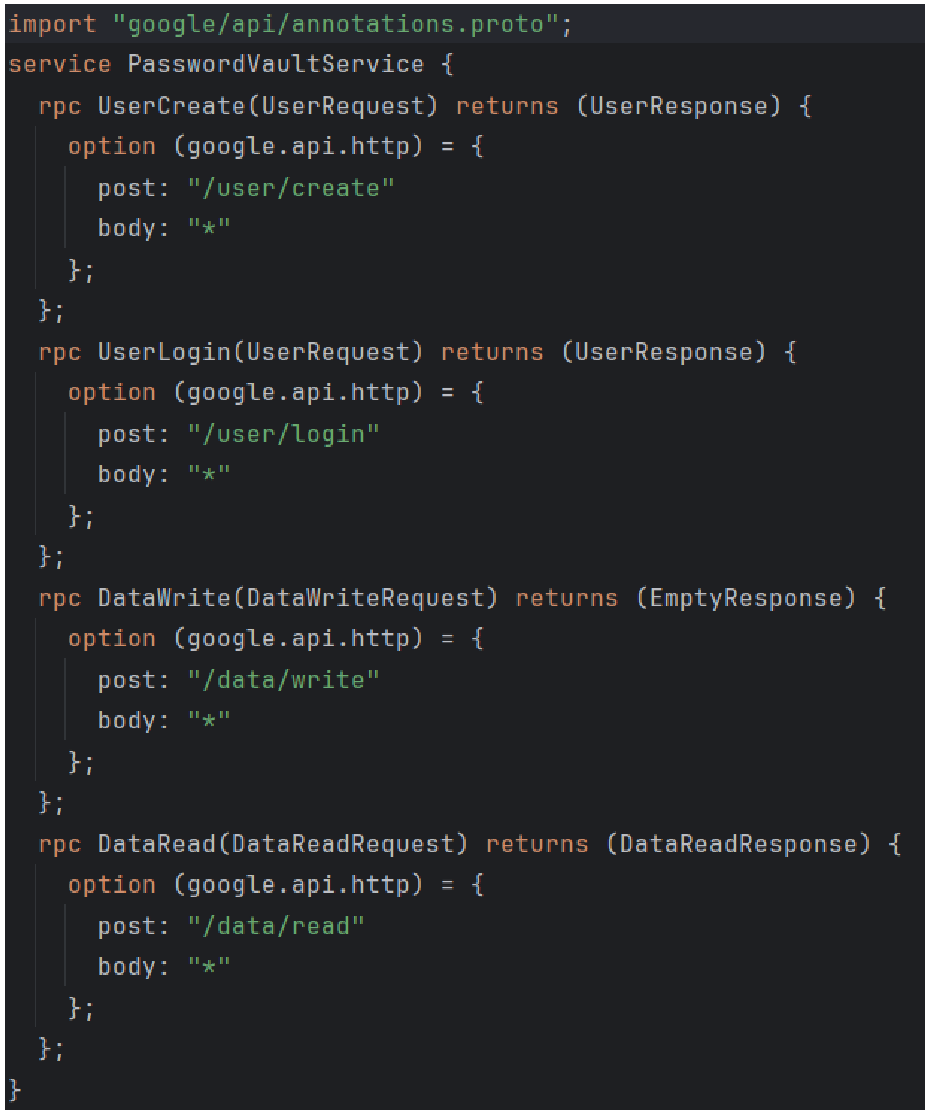
public
metadata_text_note
- id uuid
- data_id uuid
- name text
- content bytea
- md5 character(32)

Структура сервиса



Описание API сервиса

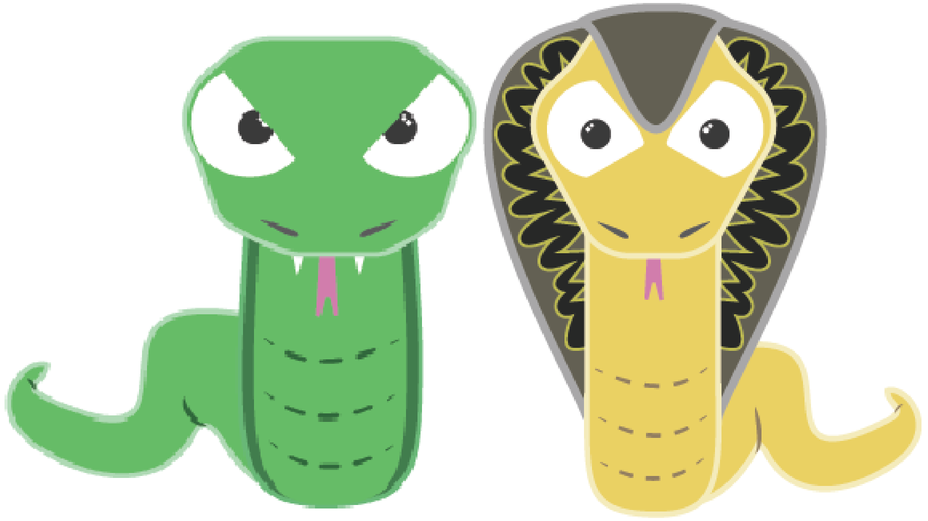
```
protoc -I ./internal/proto/src \
       -I ../googleapis \
    --go_out ./internal/proto/gen \
    --go_opt paths=source_relative  \
    --go-grpc_out ./internal/proto/gen \
    --go-grpc_opt paths=source_relative \
    --grpc-gateway_out ./internal/proto/gen \
    --grpc-gateway_opt paths=source_relative \
    service.proto
```



```
import "google/api/annotations.proto";
service PasswordVaultService {
  rpc UserCreate(UserRequest) returns (UserResponse) {
    option (google.api.http) = {
      post: "/user/create"
      body: "*"
    };
  };
  rpc UserLogin(UserRequest) returns (UserResponse) {
    option (google.api.http) = {
      post: "/user/login"
      body: "*"
    };
  };
  rpc DataWrite(DataWriteRequest) returns (EmptyResponse) {
    option (google.api.http) = {
      post: "/data/write"
      body: "*"
    };
  };
  rpc DataRead(DataReadRequest) returns (DataReadResponse) {
    option (google.api.http) = {
      post: "/data/read"
      body: "*"
    };
  };
}
```

Клиент

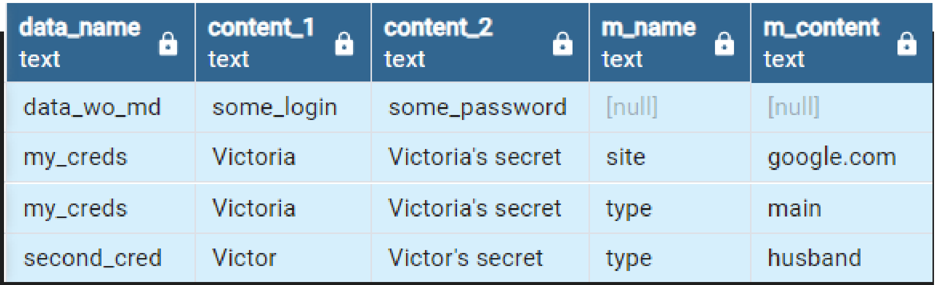
```
./client user ( create | login ) -l “Victoria‟ -p “Victoria’s secret‟
./client data print -n “%‟ -t ( “any‟, “cred‟, “card‟, “note‟, “file‟ )
        -m “site=google.com‟ -m “type=main‟
./client data write ( cred | card | note | metadata ) -n “my_note‟ --dtext=“Text‟
./client data delete ( cred | card | note | file | metadata ) -n “my_file‟
./client file upload -n “my_file‟ --fname=“D:\movie.avi‟
./client file download -n “my_file‟
```



Запрос данных с фильтрацией

```
./client data print -n % -t cred -m “site=google.com“ -m “type=main“
```

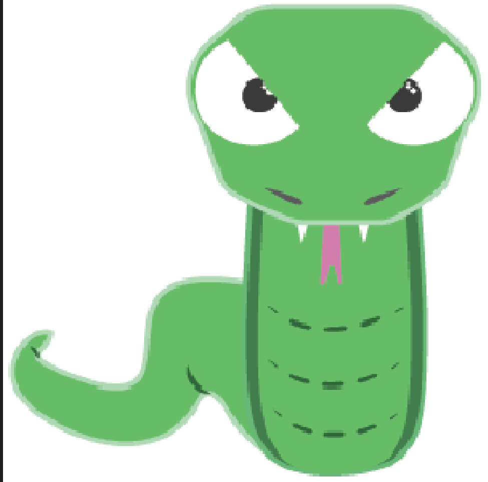
```
SELECT
    data.name AS data_name,
    pgp_sym_decrypt(data.content_1, 'secret') AS content_1,
    pgp_sym_decrypt(data.content_2, 'secret') AS content_2,
    metadata.name AS m_name,
    pgp_sym_decrypt(metadata.content, 'secret') AS m_content
FROM
    (SELECT * FROM data_credentials WHERE user_id = '898130ef-0785-4467-a75f-ee4ef9e40bf3') AS data
    LEFT JOIN metadata_credentials AS metadata ON
        data.id = metadata.data_id
WHERE
    data.name LIKE '%' AND
    EXISTS ( SELECT id FROM metadata_credentials AS metadata WHERE
        data.id = metadata.data_id AND (
        (metadata.name = 'site' AND
            pgp_sym_decrypt(metadata.content, 'secret') LIKE 'google.com') OR
        (metadata.name = 'type' AND
            pgp_sym_decrypt(metadata.content, 'secret') LIKE 'main')
    ))
```



data_name text	content_1 text	content_2 text	m_name text	m_content text
data_wo_md	some_login	some_password	[null]	[null]
my_creds	Victoria	Victoria's secret	site	google.com
my_creds	Victoria	Victoria's secret	type	main
second_cred	Victor	Victor's secret	type	husband

Пакет Viper для парсинга конфигурации

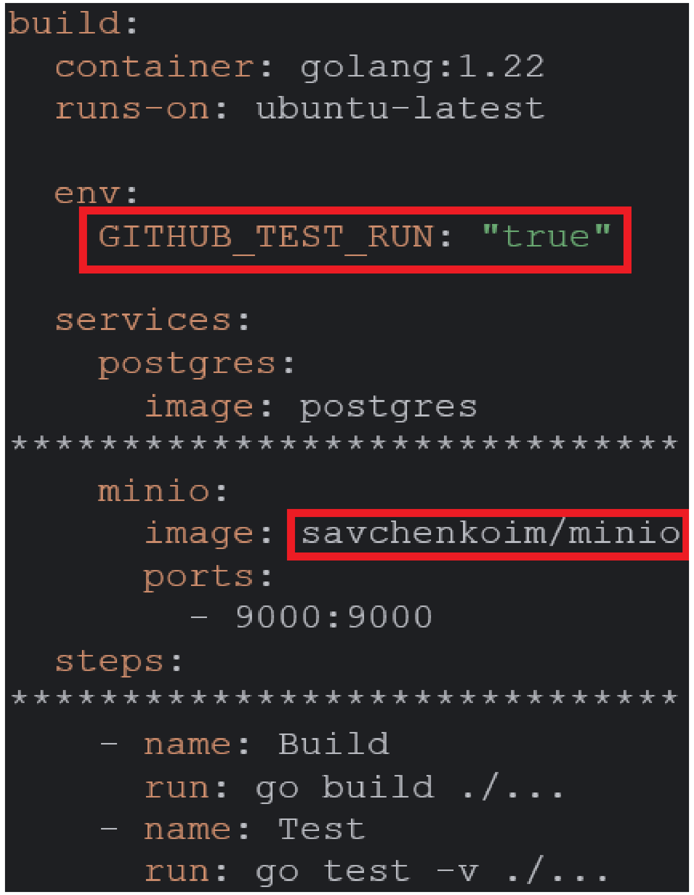
```go
flags := pflag.NewFlagSet("server_cfg", pflag.ExitOnError)
flags.String("par1", "val1", "Parameter 1")
flags.String("par2", "val2", "Parameter 2")
flags.String("cfg", "", "Config file")
err = append(err, flags.Parse(os.Args))

err = append(err, viper.BindPFlags(flags))

err = append(err, viper.BindEnv("par1", "PARAMETER_1"))
err = append(err, viper.BindEnv("par2", "PARAMETER_2"))
err = append(err, viper.BindEnv("cfg", "CONFIG_FILE"))

cfgSet := viper.GetString("cfg") != ""
if cfgSet {
   viper.SetConfigFile(viper.GetString("cfg"))
   err = append(err, viper.ReadInConfig())
}

// Do something
// ******************
// End doing something

if cfgSet {
   err = append(err, viper.WriteConfig())
}
```



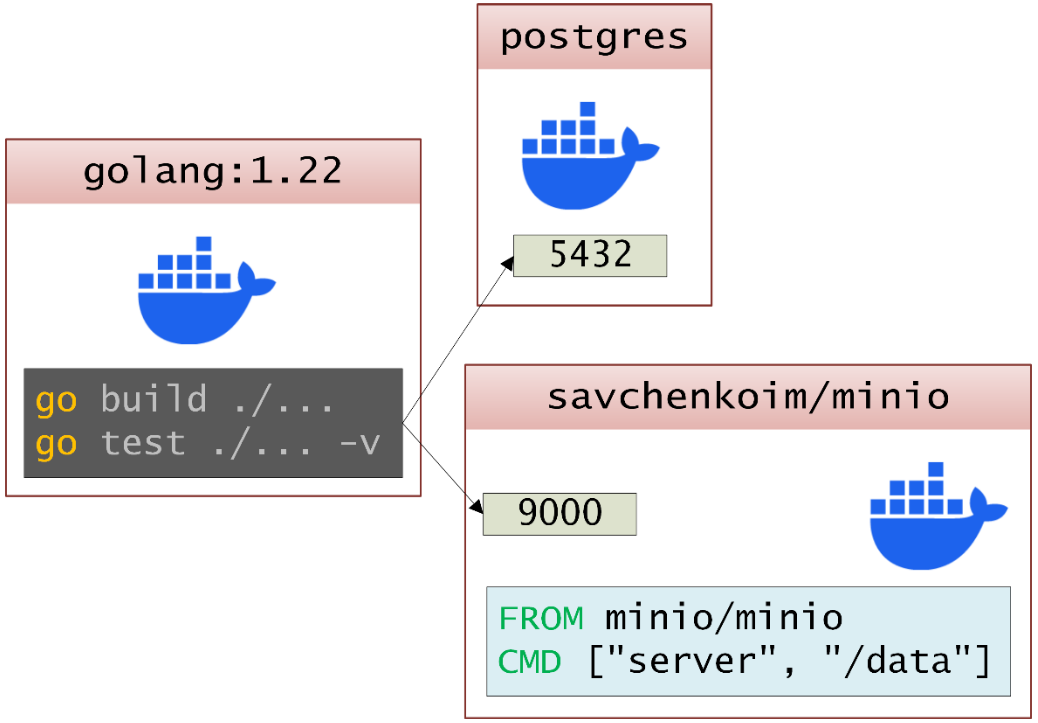
GitHub actions

```go
t.Run("Setting_Up_Test_Environment", func(t *testing.T) {
    if _, ok := os.LookupEnv("GITHUB_TEST_RUN"); ok {
        minioEndpoint = "minio:9000"
        dbConnectionString =
"postgresql://postgres:postgres@postgres/postgres?sslmode=disable"
    } else {
        cMinio, err = testcontainers.NewMinioContainer()
        require.NoError(t, err)
        minioEndpoint, err = cMinio.EndPoint()
        require.NoError(t, err)
        cPostgres, err = testcontainers.NewPostgresContainer()
        require.NoError(t, err)
        dbConnectionString, err = cPostgres.ConnectionString()
        require.NoError(t, err)
    }
})
```

GitHub actions

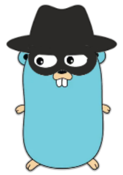


```
build:
  container: golang:1.22
  runs-on: ubuntu-latest

  env:
    GITHUB_TEST_RUN: "true"

  services:
    postgres:
      image: postgres
******************************
    minio:
      image: savchenkoim/minio
      ports:
        - 9000:9000
  steps:
******************************
    - name: Build
      run: go build ./...
    - name: Test
      run: go test -v ./...
```



```
go build ./...
go test ./... -v
```

```
FROM minio/minio
CMD ["server", "/data"]
```



Спасибо!