Consider I have the code

```
package main

import (
	"database/sql"
	"encoding/csv"
	"encoding/json"
	"flag"
	"fmt"
	"io"
	"log"
	"os"
	"os/user"
	"reflect"
	"strconv"
	"strings"
	"sync"
	"sync/atomic"
	"time"

	"github.com/gogo/protobuf/proto"
	"github.com/jmoiron/sqlx"
	_ "github.com/lib/pq"

	// Import all required protobuf message types
	pb_identity_tenant "gopkg.volterra.us/etcdreader/pbgo/
extschema/identityauthority/tenant"
	pb_maurice_application "gopkg.volterra.us/etcdreader/pbgo/
extschema/maurice/application"
	pb_maurice_deployment "gopkg.volterra.us/etcdreader/pbgo/
extschema/maurice/deployment"
	pb_ca "gopkg.volterra.us/etcdreader/pbgo/extschema/pkifactory/
ca"
	pb_cert "gopkg.volterra.us/etcdreader/pbgo/extschema/
pkifactory/cert"
	pb_key "gopkg.volterra.us/etcdreader/pbgo/extschema/
pkifactory/key"
	pb_customer_support "gopkg.volterra.us/etcdreader/pbgo/
```

```
extschema/schema/customer_support"
)

type AppConfig struct {
	CsvFilePath       string
	PostgresConnection string
	BatchSize         int
	WorkerCount       int
	ReportInterval    int
	Username          string
}

type ProtoObject struct {
	Key              string
	UID              string
	Tenant           string
	Namespace        string
	SizeMB           float64
	CreationTime     *time.Time
	ModificationTime *time.Time
	Data             interface{}
	ObjectType       string
}

type Stats struct {
	totalRows        int64
	processedRows    int64
	successCount     int64
	errorCount       int64
	skippedCount     int64
	protobufCount    int64
	nonProtobufCount int64
	objectCounts     map[string]int64 // Count by object type
	objectCountsMutex sync.Mutex      // Protect access to the map
	startTime        time.Time
	lastReportTime   time.Time
}

// Object type constants
const (
	TypeCA              = "ca"
	TypeCert            = "cert"
	TypeKey              = "key"
	TypeCustomerSupport    = "customer_support"
	TypeIdentityTenant     = "identity_tenant"
	TypeMauriceDeployment  = "maurice_deployment"
	TypeMauriceApplication = "maurice_application"
	TypeUnknown            = "unknown"
)

// ObjectTypeRegistry maps key patterns to object types and protobuf
message types
type ObjectTypeInfo struct {
	TypeName    string
	Pattern     string
	ProtoType   proto.Message
	Unmarshaler func([]byte, proto.Message) error
}

var objectTypeRegistry = []ObjectTypeInfo{
	{
		TypeName:    TypeCA,
		Pattern:     "ves.io.pkifactory.ca.Object",
		ProtoType:   &pb_ca.Object{},
		Unmarshaler: proto.Unmarshal,
	},
	{
		TypeName:    TypeCert,
		Pattern:     "ves.io.pkifactory.cert.Object",
		ProtoType:   &pb_cert.Object{},
		Unmarshaler: proto.Unmarshal,
	},
	{
		TypeName:    TypeKey,
		Pattern:     "ves.io.pkifactory.key.Object",
		ProtoType:   &pb_key.Object{},
		Unmarshaler: proto.Unmarshal,
	},
	{
		TypeName:    TypeCustomerSupport,
		Pattern:     "ves.io.schema.customer_support.Object",
		ProtoType:   &pb_customer_support.Object{},
		Unmarshaler: proto.Unmarshal,
	},
	{
		TypeName:    TypeIdentityTenant,
		Pattern:     "ves.io.identityauthority.tenant.Object",
		ProtoType:   &pb_identity_tenant.Object{},
		Unmarshaler: proto.Unmarshal,
	},
	{
		TypeName:    TypeMauriceDeployment,
		Pattern:     "ves.io.maurice.deployment.StatusObject",
		ProtoType:   &pb_maurice_deployment.StatusObject{},
		Unmarshaler: proto.Unmarshal,
	},
	{
		TypeName:    TypeMauriceApplication,
		Pattern:     "ves.io.maurice.application.Object",
		ProtoType:   &pb_maurice_application.Object{},
		Unmarshaler: proto.Unmarshal,
	},
}

func main() {
	appConfig := parseFlags()
	db := initPostgresConnection(appConfig)
	defer db.Close()

	err := createTablesIfNotExist(db)
	if err != nil {
		log.Fatalf("Failed to create PostgreSQL tables: %v", err)
	}

	stats := Stats{
		startTime:      time.Now(),
		lastReportTime: time.Now(),
		objectCounts:   make(map[string]int64),
	}

	processCSVAndInsertData(appConfig, db, &stats)

	duration := time.Since(stats.startTime)
	log.Printf("Processing completed in %s", duration)
	log.Printf("Total rows: %d, Processed: %d, Success: %d, Errors: 
%d, Skipped: %d",
		stats.totalRows, stats.processedRows, stats.successCount,
stats.errorCount, stats.skippedCount)
	log.Printf("Protobuf objects: %d, Non-protobuf values: %d",
		stats.protobufCount, stats.nonProtobufCount)

	// Log counts by object type
	log.Printf("Processed objects by type:")
	for typeName, count := range stats.objectCounts {
		log.Printf("  %s: %d", typeName, count)
	}

	if stats.successCount > 0 {
		log.Printf("Average processing rate: %.2f keys/second",
float64(stats.successCount)/duration.Seconds())
	}
}

func parseFlags() AppConfig {
	csvFilePath := flag.String("csv", "/Users/sh.p/Library/
CloudStorage/OneDrive-F5,Inc/Task3/Data/
gpki_etcd_size_metadata.csv", "Path to CSV file")
	postgresConn := flag.String("postgres", "postgres://
postgres:postgres@localhost:5432/akar?sslmode=disable",
"PostgreSQL connection string")
	batchSize := flag.Int("batch", 100, "Number of items to process in
a batch")
	workerCount := flag.Int("workers", 10, "Number of worker
goroutines")
	reportInterval := flag.Int("report-interval", 10000, "Report
progress after processing this many rows")
	username := flag.String("username", "RealGT1", "Override
system username for tracking")
	flag.Parse()

	configUsername := *username
	if configUsername == "" {
		currentUser, err := user.Current()
		if err == nil {
			configUsername = currentUser.Username
		} else {
			configUsername = "RealGT1"
		}
	}

	return AppConfig{
		CsvFilePath:        *csvFilePath,
		PostgresConnection: *postgresConn,
		BatchSize:          *batchSize,
		WorkerCount:        *workerCount,
		ReportInterval:     *reportInterval,
		Username:           configUsername,
	}
}

func initPostgresConnection(appConfig AppConfig) *sqlx.DB {
	db, err := sqlx.Connect("postgres",
appConfig.PostgresConnection)
	if err != nil {
		log.Fatalf("Failed to connect to PostgreSQL: %v", err)
	}

	// Set connection pool settings
	db.SetMaxOpenConns(appConfig.WorkerCount * 2)
	db.SetMaxIdleConns(appConfig.WorkerCount)
	db.SetConnMaxLifetime(time.Hour)

	log.Println("Successfully connected to PostgreSQL database")
	return db
}

func createTablesIfNotExist(db *sqlx.DB) error {
	// First, check if table exists
	var tableExists bool
	err := db.QueryRow(`SELECT EXISTS (
	 SELECT FROM information_schema.tables
	 WHERE table_name = 'pkifactory_objects'
   )`).Scan(&tableExists)

	if err != nil {
		return fmt.Errorf("error checking if table exists: %v", err)
	}

	if !tableExists {
		// Create simplified main objects table with object_type
```

```
column
		_, err := db.Exec(`
			CREATE TABLE pkifactory_objects (
				object_key TEXT PRIMARY KEY,
				uid TEXT,
				tenant TEXT,
				namespace TEXT,
				size_mb NUMERIC(15,5),
				creation_time TIMESTAMP WITH TIME ZONE,
				modification_time TIMESTAMP WITH TIME ZONE,
				data JSONB,
				object_type TEXT,
				import_time TIMESTAMP WITH TIME ZONE
DEFAULT NOW()
			)
		`)
		if err != nil {
			return fmt.Errorf("failed to create main objects table:
%v", err)
		}
		log.Println("Created pkifactory_objects table")
	} else {
		// Check if object_type column exists
		var columnExists bool
		err := db.QueryRow(`SELECT EXISTS (
			SELECT FROM information_schema.columns
			WHERE table_name = 'pkifactory_objects' AND
column_name = 'object_type'
		)`).Scan(&columnExists)

		if err != nil {
			return fmt.Errorf("error checking if column exists: %v",
err)
		}

		// Add object_type column if it doesn't exist
		if !columnExists {
			_, err := db.Exec(`ALTER TABLE pkifactory_objects ADD
COLUMN object_type TEXT`)
			if err != nil {
				return fmt.Errorf("failed to add object_type column:
%v", err)
```

```
				}
				log.Println("Added object_type column to 
pkifactory_objects table")
			}
		}

		// Now create indexes one by one to better handle errors
		indexes := []struct {
			name   string
			column string
		}{
			{"idx_pkifactory_objects_uid", "uid"},
			{"idx_pkifactory_objects_tenant", "tenant"},
			{"idx_pkifactory_objects_namespace", "namespace"},
			{"idx_pkifactory_objects_creation_time", "creation_time"},
			{"idx_pkifactory_objects_object_type", "object_type"},
		}

		for _, idx := range indexes {
			var indexExists bool
			err := db.QueryRow(`SELECT EXISTS (
				SELECT FROM pg_indexes
				WHERE indexname = $1
			)`, idx.name).Scan(&indexExists)

			if err != nil {
				return fmt.Errorf("error checking if index %s exists: %v", 
idx.name, err)
			}

			if !indexExists {
				_, err := db.Exec(fmt.Sprintf(`CREATE INDEX %s ON 
pkifactory_objects (%s)`, idx.name, idx.column))
				if err != nil {
					return fmt.Errorf("failed to create index %s: %v", 
idx.name, err)
				}
				log.Printf("Created index %s on %s", idx.name, 
idx.column)
			} else {
				log.Printf("Index %s already exists", idx.name)
			}
```

```
	}

	log.Println("Tables and indexes created or already exist")
	return nil
}

func processCSVAndInsertData(appConfig AppConfig, db *sqlx.DB,
stats *Stats) {
	file, err := os.Open(appConfig.CsvFilePath)
	if err != nil {
		log.Fatalf("Error opening CSV file: %v", err)
	}
	defer file.Close()

	// Create a buffered channel to hold items to process
	jobs := make(chan []string, appConfig.BatchSize*2)
	results := make(chan struct {
		success     bool
		isProtobuf  bool
		nonProtobuf bool
		objectType  string
	}, appConfig.BatchSize)

	var wg sync.WaitGroup

	// Start worker goroutines
	for i := 0; i < appConfig.WorkerCount; i++ {
		wg.Add(1)
		go worker(i, jobs, results, &wg, db, appConfig)
	}

	// Start a goroutine to collect results
	go func() {
		for result := range results {
			if result.success {
				atomic.AddInt64(&stats.successCount, 1)
				if result.isProtobuf {
					atomic.AddInt64(&stats.protobufCount, 1)

					// Count by object type - using mutex to protect
map access
					if result.objectType != "" {
```

```
                        stats.objectCountsMutex.Lock()
                        stats.objectCounts[result.objectType]++
                        stats.objectCountsMutex.Unlock()
                    }
                }
                if result.nonProtobuf {
                    atomic.AddInt64(&stats.nonProtobufCount, 1)
                }
            } else {
                atomic.AddInt64(&stats.errorCount, 1)
            }

            processed := atomic.AddInt64(&stats.processedRows,
1)

            // Report progress at intervals
            if processed%int64(appConfig.ReportInterval) == 0 {
                now := time.Now()
                elapsed := now.Sub(stats.lastReportTime)
                rate := float64(appConfig.ReportInterval) /
elapsed.Seconds()

                // Format similar to the original code
                log.Printf("Progress: %d/%d rows (%.2f%%) - %.2f
keys/second",
                    processed, stats.totalRows, float64(processed)/
float64(stats.totalRows)*100.0, rate)

                log.Printf("Success: %d (Proto: %d, Non-Proto:
%d), Errors: %d, Skipped: %d",
                    stats.successCount, stats.protobufCount,
stats.nonProtobufCount,
                    stats.errorCount, stats.skippedCount)

                // Similar to original code, report counts by type
                stats.objectCountsMutex.Lock()
                typeCounts := make(map[string]int64)
                for typeName, count := range stats.objectCounts {
                    typeCounts[typeName] = count
                }
                stats.objectCountsMutex.Unlock()
```

```
				log.Printf("By type: CA: %d, Cert: %d, Key: %d,
CustomerSupport: %d, IdentityTenant: %d, MauriceDeployment: %d,
MauriceApplication: %d",
					typeCounts[TypeCA], typeCounts[TypeCert],
typeCounts[TypeKey],
					typeCounts[TypeCustomerSupport],
typeCounts[TypeIdentityTenant],
					typeCounts[TypeMauriceDeployment],
typeCounts[TypeMauriceApplication])

				stats.lastReportTime = now
			}
		}
	}()

	// Read CSV and count total rows for progress reporting
	log.Println("Counting total rows in CSV file...")
	fileSize, _ := file.Stat()
	log.Printf("CSV file size: %.2f MB", float64(fileSize.Size())/
(1024*1024))

	// Reset file pointer to beginning
	file.Seek(0, 0)
	reader := csv.NewReader(file)
	reader.FieldsPerRecord = -1

	rowNum := int64(0)
	for {
		_, err := reader.Read()
		if err == io.EOF {
			break
		}
		rowNum++
	}
	stats.totalRows = rowNum
	log.Printf("Total rows in CSV file: %d", stats.totalRows)

	// Reset file pointer to beginning again
	file.Seek(0, 0)
	reader = csv.NewReader(file)
	reader.FieldsPerRecord = -1
	rowNum = 0
	log.Println("Starting data processing...")

	for {
		row, err := reader.Read()
		if err == io.EOF {
			break
		}

		if err != nil {
			log.Printf("⚠️ Skipping malformed row #%d: %v",
rowNum, err)
			atomic.AddInt64(&stats.skippedCount, 1)
			continue
		}

		rowNum++
		if len(row) < 3 {
			log.Printf("⚠️ Skipping incomplete row #%d: %v",
rowNum, row)
			atomic.AddInt64(&stats.skippedCount, 1)
			continue
		}

		jobs <- row
	}

	close(jobs)
	wg.Wait()
	close(results)
}

func worker(id int, jobs <-chan []string, results chan<- struct {
	success     bool
	isProtobuf  bool
	nonProtobuf bool
	objectType  string
}, wg *sync.WaitGroup, db *sqlx.DB, appConfig AppConfig) {
	defer wg.Done()

	for row := range jobs {
        key := strings.TrimSpace(row[0])
        value := []byte(row[1])

        // Parse size from the third column and calculate sizeMB with
5 decimal places
        var sizeMB float64
        if len(row) >= 3 {
            if size, err := strconv.ParseInt(strings.TrimSpace(row[2]),
10, 64); err == nil {
                sizeMB = float64(size) / (1024 * 1024) // Convert
bytes to MB with full precision
            }
        }

        // Process the object based on its type
        objectType, protoObj := getProtoObjectType(key)

        if objectType == TypeUnknown {
            // Skip this row as it doesn't match any pattern
            results <- struct {
                success     bool
                isProtobuf  bool
                nonProtobuf bool
                objectType  string
            }{false, false, false, TypeUnknown}
            continue
        }

        // Try to unmarshal the protobuf message
        success, isProtobuf := processProtobufMessage(id, key,
value, sizeMB, objectType, protoObj, db)

        results <- struct {
            success     bool
            isProtobuf  bool
            nonProtobuf bool
            objectType  string
        }{success, isProtobuf, !isProtobuf, objectType}
    }
}

// getProtoObjectType determines the object type based on the key
```

```
pattern
func getProtoObjectType(key string) (string, proto.Message) {
	for _, typeInfo := range objectTypeRegistry {
		if strings.Contains(key, typeInfo.Pattern) {
			// Create a new instance of the proto message
			protoType := reflect.TypeOf(typeInfo.ProtoType)
			if protoType.Kind() == reflect.Ptr {
				protoType = protoType.Elem()
			}

			// Create a new instance of the same type
			protoInstance := reflect.New(protoType).Interface().
(proto.Message)
			return typeInfo.TypeName, protoInstance
		}
	}

	return TypeUnknown, nil
}

// processProtobufMessage handles unmarshaling and storing
protobuf messages
func processProtobufMessage(
	id int,
	key string,
	value []byte,
	sizeMB float64,
	objectType string,
	protoObj proto.Message,
	db *sqlx.DB,
) (bool, bool) {
	// Try to unmarshal the protobuf
	err := proto.Unmarshal(value, protoObj)
	if err != nil {
	//	log.Printf("Worker %d: Error unmarshaling %s protobuf for
key %s: %v", id, objectType, key, err)

		// Store as raw value instead
		rawObj := ProtoObject{
			Key:        key,
			SizeMB:     sizeMB,
			ObjectType: objectType,
			Data:	string(value),
		}

		err = insertIntoPostgreSQL(db, rawObj)
		if err != nil {
		//	log.Printf("Worker %d: Error inserting raw value into
PostgreSQL for key %s: %v", id, key, err)
			return false, false
		}

		return true, false
	}

	// Successfully unmarshaled, now extract common fields
	dbObj := extractCommonFields(key, protoObj, objectType)
	dbObj.SizeMB = sizeMB

	// Convert protobuf to string representation
	protoString := fmt.Sprintf("%+v", protoObj)
	dbObj.Data = protoString

	// Insert into database
	err = insertIntoPostgreSQL(db, dbObj)
	if err != nil {
		log.Printf("Worker %d: Error inserting %s object into
PostgreSQL for key %s: %v", id, objectType, key, err)
		return false, true
	}

	return true, true
}

// extractCommonFields extracts common fields from any protobuf
message using reflection
func extractCommonFields(key string, message proto.Message,
objectType string) ProtoObject {
	obj := ProtoObject{
		Key:	key,
		ObjectType: objectType,
	}

	// Use reflection to access common fields
	v := reflect.ValueOf(message)
	if v.Kind() == reflect.Ptr {
		v = v.Elem()
	}

	// STEP 1: Extract UID, tenant, and timestamps
	// Look for Metadata field for UID
	if metadataField := v.FieldByName("Metadata");
metadataField.IsValid() && !metadataField.IsNil() {
		metadata := metadataField.Elem()

		// Extract UID
		if uidField := metadata.FieldByName("Uid"); uidField.IsValid()
{
			obj.UID = uidField.String()
		}
	}

	// Look for SystemMetadata field for tenant and timestamps
	if sysMetadataField := v.FieldByName("SystemMetadata");
sysMetadataField.IsValid() && !sysMetadataField.IsNil() {
		sysMetadata := sysMetadataField.Elem()

		// Extract Tenant
		if tenantField := sysMetadata.FieldByName("Tenant");
tenantField.IsValid() {
			obj.Tenant = tenantField.String()
		}

		// Extract CreationTimestamp
		if creationField :=
sysMetadata.FieldByName("CreationTimestamp");
creationField.IsValid() && !creationField.IsNil() {
			creationTS := creationField.Elem()
			seconds := creationTS.FieldByName("Seconds").Int()
			nanos := creationTS.FieldByName("Nanos").Int()

			if seconds > 0 || nanos > 0 {
				ct := time.Unix(seconds, nanos)
				obj.CreationTime = &ct
			}
		}
		// Extract ModificationTimestamp
		if modField :=
sysMetadata.FieldByName("ModificationTimestamp");
modField.IsValid() && !modField.IsNil() {
			modTS := modField.Elem()
			seconds := modTS.FieldByName("Seconds").Int()
			nanos := modTS.FieldByName("Nanos").Int()

			if seconds > 0 || nanos > 0 {
				mt := time.Unix(seconds, nanos)
				obj.ModificationTime = &mt
			}
		}
	}

	// STEP 2: Extract namespace (using multiple methods for ALL
object types)

	// METHOD 1: Check Metadata.Namespace (works for cert, key,
customer_support, identity_tenant)
	if metadataField := v.FieldByName("Metadata");
metadataField.IsValid() && !metadataField.IsNil() {
		metadata := metadataField.Elem()
		if namespaceField := metadata.FieldByName("Namespace");
namespaceField.IsValid() && namespaceField.String() != "" {
			obj.Namespace = namespaceField.String()
		}
	}

	// METHOD 2: Check SystemMetadata.Namespace array (works
for ca and others)
	if obj.Namespace == "" &&
v.FieldByName("SystemMetadata").IsValid() && !
v.FieldByName("SystemMetadata").IsNil() {
		sysMetadata := v.FieldByName("SystemMetadata").Elem()
		if namespaceField :=
sysMetadata.FieldByName("Namespace"); namespaceField.IsValid()
&& namespaceField.Kind() == reflect.Slice {
			if namespaceField.Len() > 0 && !
namespaceField.Index(0).IsNil() {
				nsObj := namespaceField.Index(0).Elem()
				// Try Name field (this is common in many objects)
				if nameField := nsObj.FieldByName("Name");
nameField.IsValid() && nameField.String() != "" {
					obj.Namespace = nameField.String()
				}

				// If Name is not available or empty, try Namespace
field
				if obj.Namespace == "" {
					if nsField := nsObj.FieldByName("Namespace");
nsField.IsValid() && nsField.String() != "" {
						obj.Namespace = nsField.String()
					}
				}
			}
		}
	}

	// METHOD 3: Use the default value of "system" if no namespace
found
	if obj.Namespace == "" {
		obj.Namespace = "system" // Default based on the provided
examples
	}

	return obj
}

func insertIntoPostgreSQL(db *sqlx.DB, obj ProtoObject) error {
	// Begin a transaction
	tx, err := db.Beginx()
	if err != nil {
		return fmt.Errorf("failed to begin transaction: %v", err)
	}

	// When data is a string, we need to convert it differently
	var jsonData []byte
	switch v := obj.Data.(type) {
	case string:
		// For string data (raw values or protobuf string
representation),
			// we'll wrap it in a JSON object with a "data" key
			wrapper := map[string]string{"data": v}
			jsonData, err = json.Marshal(wrapper)
			if err != nil {
				tx.Rollback()
				return fmt.Errorf("failed to marshal string data to JSON:
%v", err)
			}
		default:
			// For other types, use regular JSON marshaling
			jsonData, err = json.Marshal(obj.Data)
			if err != nil {
				tx.Rollback()
				return fmt.Errorf("failed to marshal data to JSON: %v",
err)
			}
		}

		// Create SQL NullTime objects for both creation and modification
times
		var creationTime sql.NullTime
		if obj.CreationTime != nil {
			creationTime.Time = *obj.CreationTime
			creationTime.Valid = true
		}

		var modTime sql.NullTime
		if obj.ModificationTime != nil {
			modTime.Time = *obj.ModificationTime
			modTime.Valid = true
		}

		_, err = tx.Exec(`
			INSERT INTO pkifactory_objects (
				object_key, uid, tenant, namespace, size_mb,
creation_time, modification_time, data, object_type
			) VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9)
			ON CONFLICT (object_key) DO UPDATE SET
				uid = EXCLUDED.uid,
				tenant = EXCLUDED.tenant,
				namespace = EXCLUDED.namespace,
				size_mb = EXCLUDED.size_mb,
                creation_time = EXCLUDED.creation_time,
                modification_time = EXCLUDED.modification_time,
                data = EXCLUDED.data,
                object_type = EXCLUDED.object_type,
                import_time = NOW()
    `, obj.Key, obj.UID, obj.Tenant, obj.Namespace, obj.SizeMB,
creationTime, modTime, jsonData, obj.ObjectType)

    if err != nil {
        tx.Rollback()
        return fmt.Errorf("failed to insert object: %v", err)
    }

    // Commit the transaction
    if err = tx.Commit(); err != nil {
        return fmt.Errorf("failed to commit transaction: %v", err)
    }

    return nil
}
```

the above code is made for different namespace

consider I want to access different namespace and perform exact same functions for
virtual host:/akar/db/ves.io.schema.virtual_host.Object.default
route:/akar/db/ves.io.schema.route.Object.default
cluster:/akar/db/ves.io.schema.cluster.Object.default
endpoint:/akar/db/ves.io.schema.endpoint.Object.default

so in my org data flows like this

I want to create 4 tables initially with virtual_host decoded data, same with route, cluster and endpoint object

now that I have data

but how will connect them
for that I have a way.....I want to create a new table where I want complete details of when user requests till the endpoint

to get the data..we have store in separate tables

for example
if I take some key for virtual host

```
{
  "key": "/akar/db/ves.io.schema.virtual_host.Object.default/primary/
fff5a80b-7bfb-4d62-a34a-c3c6016b1478",
  "metadata": {
    "name": "ves-io-http-loadbalancer-sujesh-vh-google-1",
    "namespace": "default",
    "uid": "fff5a80b-7bfb-4d62-a34a-c3c6016b1478",
    "labels": {},
    "annotations": {},
    "description": "",
    "disable": false
  },
  "system_metadata": {
    "uid": "fff5a80b-7bfb-4d62-a34a-c3c6016b1478",
    "creation_timestamp": "2022-10-18T11:21:39.367423904Z",
    "deletion_timestamp": null,
    "modification_timestamp": "2022-10-18T12:32:25.567981172Z",
    "initializers": null,
    "finalizers": [],
    "tenant": "customer2",
    "creator_class": "akar",
    "creator_id": "",
    "trace_info":
"7bca50e8875fcb84:7bca50e8875fcb84:0000000000000000:1",
    "object_index": 0,
    "namespace": [
      {
        "kind": "namespace",
        "uid": "75e7bda8-1b53-414f-b36b-5b39a4b80fcb",
        "tenant": "customer2",
        "namespace": "",
        "name": "default"
      }
    ],
    "creator_cookie": "",
    "owner_view": {
      "kind": "http_loadbalancer",
      "uid": "6b705588-12fe-4a89-b9f4-6d4501c36dc7",
      "namespace": "default",
      "name": "sujesh-vh-google-1"
    },
    "sre_disable": false,
    "vtrp_id": "",
    "vtrp_stale": false,
    "labels": {},
    "direct_ref_hash": ""
  },
  "spec": {
    "gc_spec": {
      "domains": ["mytest2.vhgoogle.com"],
      "routes": [
        {
          "kind": "route",
          "uid": "",
          "tenant": "customer2",
          "namespace": "default",
          "name": "ves-io-http-loadbalancer-sujesh-vh-google-1"
        }
      ],
      "javascript_info": null,
      "advertise_policies": [
        {
          "kind": "advertise_policy",
          "uid": "",
          "tenant": "customer2",
          "namespace": "default",
          "name": "ves-io-http-loadbalancer-sujesh-vh-
google-1-85b45789f"
        }
      ],
      "request_headers_to_add": [],
      "response_headers_to_add": [],
      "response_headers_to_remove": [],
      "tls_certificates_choice": null,
      "type": "HTTP_LOAD_BALANCER",
      "buffer_policy": null,
      "cors_policy": null,
      "proxy": "HTTP_PROXY",
      "jwt": [],
"request_headers_to_remove": [],
"waf_type": {
  "ref_type": null
},
"dynamic_reverse_proxy": null,
"add_location": false,
"compression_params": null,
"custom_errors": {},
"max_request_header_size": 0,
"challenge_type": {
  "no_challenge": {}
},
"user_identification": [],
"rate_limiter": [],
"rate_limiter_allowed_prefixes": [],
"retry_policy": null,
"idle_timeout": 0,
"disable_default_error_pages": false,
"disable_dns_resolve": false,
"temporary_user_blocking": null,
"malicious_user_mitigation": [],
"tls_intercept": null,
"authentication_choice": null,
"server_header_choice": null,
"path_normalize_choice": null,
"strict_sni_host_header_check_choice": null,
"cdn_service": null,
"trust_client_ip_headers_choice": {
  "disable_trust_client_ip_headers": {}
},
"default_lb_choice": null,
"header_transformation_type": null,
"csrf_policy": null,
"cookies_to_modify": [],
"connection_idle_timeout": 0,
"slow_ddos_mitigation": null,
"api_spec": null,
"domain_cert_map": {},
"http_protocol_options": null,
"ddos_auto_mitigation_action": null,
"use_threat_mesh": false,
"masking_config": null,
      "downstream_cos": [],
      "enable_malware_protection": null,
      "coalescing_options": null,
      "dns_volterra_managed": false,
      "dns_domains": [],
      "auto_cert": false,
      "state": "VIRTUAL_HOST_READY",
      "host_name": "ves-io-6b705588-12fe-4a89-
b9f4-6d4501c36dc7.demo1.ac.vh.volterra.us",
      "dns_info": [],
      "auto_cert_state": "AutoCertNotApplicable",
      "auto_cert_info": null,
      "user_domains": ["mytest2.vhgoogle.com"],
      "service_policy_sets": [],
      "loadbalancer_algorithm": "ROUND_ROBIN",
      "volterra_cert": false,
      "bot_defense_choice": null,
      "check_ip_reputation": false,
      "fast_acl": [],
      "l7_acl": [],
      "dns_zones": [],
      "dns_zone_state_choice": null,
      "custom_cert_expiry": null,
      "auto_cert_error_msg": "",
      "http_redirect_options": null,
      "max_direct_response_body_size": 0,
      "ztna_proxy_configurations": null,
      "advertise_on_public": false,
      "sensitive_data_policy": [],
      "dns_proxy_configuration": null
    }
  }
}
```

u can see

```
"routes": [
        {
          "kind": "route",
          "uid": "",
          "tenant": "customer2",
          "namespace": "default",
```

        "name": "ves-io-http-loadbalancer-sujesh-vh-google-1"
      }

so now we have route info

now search in route table
u will get the info
for example

{
  "Key": "/akar/db/ves.io.schema.route.Object.default/primary/
ffeae945-dfd1-4c19-83c7-79dd29be0116",
  "Metadata": {
    "Name": "ves-io-http-loadbalancer-juice-shop-1",
    "Namespace": "nelly-waf-test",
    "Uid": "ffeae945-dfd1-4c19-83c7-79dd29be0116",
    "Labels": {},
    "Annotations": {},
    "Description": "",
    "Disable": false
  },
  "SystemMetadata": {
    "Uid": "ffeae945-dfd1-4c19-83c7-79dd29be0116",
    "CreationTimestamp": "2024-04-01T12:01:31.823564074Z",
    "DeletionTimestamp": null,
    "ModificationTimestamp": "2024-12-30T13:31:38.234100909Z",
    "Initializers": null,
    "Finalizers": [],
    "Tenant": "customer2",
    "CreatorClass": "akar",
    "CreatorId": "",
    "TraceInfo":
"3fc9d49a92fdb99f:3fc9d49a92fdb99f:0000000000000000:1",
    "ObjectIndex": 0,
    "Namespace": [
      {
        "Kind": "namespace",
        "Uid": "68cc99b9-4730-4d4a-8b85-2cc406a9e45a",
        "Tenant": "customer2",
        "Namespace": "",
        "Name": "nelly-waf-test"
      }
    ],

```
   "CreatorCookie": "",
   "OwnerView": {
     "Kind": "http_loadbalancer",
     "Uid": "cdc03d58-bc4f-4b6c-9e5d-78daf26748c0",
     "Namespace": "nelly-waf-test",
     "Name": "juice-shop"
   },
   "SreDisable": false,
   "VtrpId": "",
   "VtrpStale": false,
   "Labels": {},
   "DirectRefHash": ""
 },
 "Spec": {
   "GcSpec": {
     "Routes": [
       {
         "Match": [
           {
             "Path": {
               "PathMatch": {
                 "Regex": "(.*?)"
               }
             },
             "Headers": [],
             "QueryParams": [],
             "HttpMethod": "ANY",
             "IncomingPort": null
           }
         ],
         "RouteAction": {
           "RouteDestination": {
             "Destinations": [
               {
                 "Cluster": [
                   {
                     "Kind": "cluster",
                     "Uid": "",
                     "Tenant": "customer2",
                     "Namespace": "nelly-waf-test",
                     "Name": "ves-io-origin-pool-gil-juiceshop"
                   }
```

```
              ],
              "Weight": 1,
              "EndpointSubsets": {},
              "Priority": 1
            }
          ],
          "RouteDestinationRewrite": null,
          "HostRewriteParams": {
            "AutoHostRewrite": true
          },
          "Timeout": 0,
          "RetryPolicy": {
            "RetryOn": "",
            "NumRetries": 1,
            "PerTryTimeout": 0,
            "RetriableStatusCodes": [],
            "BackOff": null,
            "RetryCondition": ["5xx"]
          },
          "EndpointSubsets": {},
          "MirrorPolicy": null,
          "WebSocketConfig": null,
          "BufferPolicy": null,
          "CorsPolicy": null,
          "HashPolicy": [],
          "Priority": "DEFAULT",
          "SpdyConfig": null,
          "ClusterRetractChoice": {
            "RetractCluster": {}
          },
          "CsrfPolicy": null,
          "QueryParams": null
        }
      }
    }
  ],
  "DisableCustomScript": false,
  "RequestHeadersToAdd": [],
  "ResponseHeadersToAdd": [],
  "RequestHeadersToRemove": [],
  "ResponseHeadersToRemove": [],
  "WafType": null,
      "ServicePolicy": null,
      "DisableLocationAdd": false,
      "SkipLbOverride": false,
      "BotDefenseJavascriptInjectionChoice": null,
      "BotDefenseJavascriptInjectionInlineMode": null
    }
  }
}

"Destinations": [
                {
                  "Cluster": [
                    {
                      "Kind": "cluster",
                      "Uid": "",
                      "Tenant": "customer2",
                      "Namespace": "nelly-waf-test",
                      "Name": "ves-io-origin-pool-gil-juiceshop"
                    }
                  ],
```

now that we got cluster info

now we need endpoints...go to clusters table

```
{
  "Key": "/akar/db/ves.io.schema.cluster.Object.default/primary/
fff3da8e-7493-4e16-ae4b-b49f02acc1c9",
  "Metadata": {
    "Name": "cluster-one-testppkxscalesdysu",
    "Namespace": "sandipd-automation-bng-setup-
scaletest1000012",
    "Uid": "fff3da8e-7493-4e16-ae4b-b49f02acc1c9",
    "Labels": {},
    "Annotations": {},
    "Description": "",
    "Disable": false
  },
  "SystemMetadata": {
    "Uid": "fff3da8e-7493-4e16-ae4b-b49f02acc1c9",
    "CreationTimestamp": "2020-01-07T13:20:07.434934671Z",
    "DeletionTimestamp": null,
    "ModificationTimestamp": "2020-12-22T20:47:09.878536152Z",
    "Initializers": null,
    "Finalizers": [],
    "Tenant": "scale40-mxeorgds",
    "CreatorClass": "",
    "CreatorId": "",
    "TraceInfo": "1ca7701d07f780f0:1ca7701d07f780f0:0:1",
    "ObjectIndex": 0,
    "Namespace": [
     {
      "Kind": "namespace",
      "Uid": "",
      "Tenant": "scale40-mxeorgds",
      "Namespace": "",
      "Name": "sandipd-automation-bng-setup-scaletest1000012"
     }
    ],
    "CreatorCookie": "",
    "OwnerView": null,
    "SreDisable": false,
    "VtrpId": "",
    "VtrpStale": false,
    "Labels": {},
    "DirectRefHash": ""
   },
   "Spec": {
    "GcSpec": {
     "Endpoints": [
      {
       "Kind": "ves.io.schema.endpoint.Object",
       "Uid": "",
       "Tenant": "scale40-mxeorgds",
       "Namespace": "sandipd-automation-bng-setup-
scaletest1000012",
       "Name": "ep-one-testppkxscalesdysu"
      }
     ],
     "HealthChecks": [],
     "LoadbalancerAlgorithm": "ROUND_ROBIN",
     "CircuitBreaker": null,
     "EndpointSubsets": [],
     "DefaultSubset": {},
      "FallbackPolicy": "NO_FALLBACK",
      "TlsParameters": null,
      "ConnectionTimeout": 0,
      "HttpIdleTimeout": 0,
      "OutlierDetection": null,
      "EndpointSelection": "DISTRIBUTED",
      "DnsLookupFamily": "AUTO",
      "DnsDiscoveryType": "STRICT_DNS",
      "HttpProtocolType": null,
      "PanicThresholdType": null,
      "HeaderTransformationType": null,
      "LbSourceIpPersistanceChoice": null,
      "ProxyProtocolType": null
    }
  }
}
```

here
I got endpoints

```
"Endpoints": [
        {
          "Kind": "ves.io.schema.endpoint.Object",
          "Uid": "",
          "Tenant": "scale40-mxeorgds",
          "Namespace": "sandipd-automation-bng-setup-
scaletest1000012",
          "Name": "ep-one-testppkxscalesdysu"
        }
```

perfect

go to end points object table

```
{
  "Key": "/akar/db/ves.io.schema.endpoint.Object.default/primary/
ffd8a01a-49fa-4b8f-996d-1cf061f0bd71",
  "Metadata": {
    "Name": "ves-io-k8s-cluster-anvesh-gcp-s-node-app-endpoint-
pk8s",
    "Namespace": "system",
    "Uid": "ffd8a01a-49fa-4b8f-996d-1cf061f0bd71",
    "Labels": {},
    "Annotations": {},
    "Description": "",
    "Disable": false
  },
  "System Metadata": {
    "Uid": "ffd8a01a-49fa-4b8f-996d-1cf061f0bd71",
    "CreationTimestamp": "2025-02-06T07:41:57.40672155Z",
    "DeletionTimestamp": null,
    "ModificationTimestamp": null,
    "Initializers": null,
    "Finalizers": [],
    "Tenant": "testcorp-hagrmdbk",
    "CreatorClass": "akar",
    "CreatorId": "",
    "TraceInfo":
"72c5f18bf12d014d:72c5f18bf12d014d:0000000000000000:1",
    "ObjectIndex": 47889,
    "Namespace": [
      {
        "Kind": "namespace",
        "Uid": "bba8e605-8885-4b84-9144-36abb53af405",
        "Tenant": "testcorp-hagrmdbk",
        "Namespace": "",
        "Name": "system"
      }
    ],
    "CreatorCookie": "",
    "OwnerView": {
      "Kind": "k8s_cluster",
      "Uid": "ee0558ad-4e07-4630-9109-2932f0c79d8e",
      "Namespace": "system",
      "Name": "anvesh-gcp-s-node"
    },
    "SreDisable": false,
    "VtrpId": "",
    "VtrpStale": false,
    "Labels": {
      "ves.io/child-object": "true"
    },
    "DirectRefHash": ""
  },
  "Spec": {
    "GcSpec": {
      "Where": {
        "RefOrSelector": {
          "Site": {
            "Ref": [
              {
                "Kind": "site",
                "Uid": "",
                "Tenant": "testcorp-hagrmdbk",
                "Namespace": "system",
                "Name": "anvesh-new-gcp"
              }
            ],
            "NetworkType": "VIRTUAL_NETWORK_SITE_LOCAL",
            "InternetVipChoice": {
              "DisableInternetVip": {}
            },
            "Refs": []
          }
        }
      },
      "Port": 6443,
      "EndpointAddress": {
        "K8SClusterApiServer": {}
      },
      "Protocol": "TCP",
      "HealthCheckPort": 0,
      "ProximityChoice": null
    }
  }
}
```

here u have

```
{
                "Kind": "site",
                "Uid": "",
                "Tenant": "testcorp-hagrmdbk",
                "Namespace": "system",
                "Name": "anvesh-new-gcp"
              }
            ],
            "NetworkType": "VIRTUAL_NETWORK_SITE_LOCAL", with
```

port number, protocol

this is the dataflow
above keys are sample data..it may not be linked

hope you got idea..I want new table of complete info from request to endpoint as mentioned