ecommerce-platform / product-service (Step 2)

This is the second service of the project: **product-service** (catalog + inventory) implemented in Go with gRPC, Postgres, and Docker. It includes:

- gRPC service with **ListProducts**, **GetProduct**, **UpdateStock**
- Postgres storage (products table)
- Basic validation & error handling
- Health + reflection
- Proto + Makefile target to generate stubs
- Dockerfile
- Updated `docker-compose.yml` (adds this service alongside `user-service`)

Directory tree (for this step)

```
ecommerce-platform/
└── product-service/
    ├── Dockerfile
    ├── Makefile
    ├── go.mod
    ├── go.sum
    ├── main.go
    ├── proto/
    │   └── product.proto
    └── internal/
        ├── config/
        │   └── config.go
        ├── db/
        │   └── postgres.go
        ├── models/
        │   └── product.go
        └── service/
            └── product_server.go
```

---

proto/product.proto

```
syntax = "proto3";
package product.v1;

option go_package = "github.com/example/ecommerce-platform/product-service/
proto;productpb";

message Product {
  string id = 1;
  string name = 2;
  double price = 3;
  int32 stock = 4;
  string created_at = 5; // RFC3339
}

message ListProductsRequest {}
message ListProductsResponse {
  repeated Product products = 1;
}

message GetProductRequest {
  string id = 1;
}
message GetProductResponse {
  Product product = 1;
}

message UpdateStockRequest {
  string id = 1;
  int32 delta = 2; // positive to add stock, negative to reduce
}
message UpdateStockResponse {
  Product product = 1;
}

service ProductService {
  rpc ListProducts(ListProductsRequest) returns (ListProductsResponse);
  rpc GetProduct(GetProductRequest) returns (GetProductResponse);
  rpc UpdateStock(UpdateStockRequest) returns (UpdateStockResponse);
}
```

---

product-service/Makefile

```
PROTO_DIR=proto

proto:
    protoc
        --go_out=paths=source_relative:.
        --go-grpc_out=paths=source_relative:.
        $(PROTO_DIR)/*.proto

run:
    go run ./...

build:
    go build -o bin/product-service ./...
```

```
clean:
    rm -rf bin
```

---

product-service/go.mod

```
module github.com/example/ecommerce-platform/product-service

go 1.22.0

require (
    github.com/jackc/pgx/v5 v5.6.0
    github.com/jackc/pgx/v5/stdlib v5.6.0
    google.golang.org/grpc v1.65.0
    google.golang.org/grpc/health v1.1.0
    google.golang.org/protobuf v1.34.2
)
```

---

product-service/internal/config/config.go

```
package config

import (
    "log"
    "os"
)

type Config struct {
    Port        string
    DatabaseURL string
}

func FromEnv() Config {
    cfg := Config{
        Port:        getEnv("PORT", "50052"),
        DatabaseURL: getEnv("DATABASE_URL", "postgres://app:app@localhost:
5432/products?sslmode=disable"),
    }
    if cfg.DatabaseURL == "" {
        log.Fatal("DATABASE_URL must be set")
    }
    return cfg
}

func getEnv(key, def string) string {
```

```go
    if v := os.Getenv(key); v != "" {
        return v
    }
    return def
}
```

---

product-service/internal/db/postgres.go

```go
package db

import (
    "context"
    "database/sql"
    "errors"
    "time"

    _ "github.com/jackc/pgx/v5/stdlib"
)

type Postgres struct {
    DB *sql.DB
}

func Connect(dsn string) (*Postgres, error) {
    database, err := sql.Open("pgx", dsn)
    if err != nil {
        return nil, err
    }
    database.SetMaxOpenConns(10)
    database.SetMaxIdleConns(5)
    database.SetConnMaxLifetime(30 * time.Minute)

    ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
    defer cancel()
    if err := database.PingContext(ctx); err != nil {
        return nil, err
    }
    p := &Postgres{DB: database}
    if err := p.migrate(ctx); err != nil {
        return nil, err
    }
    return p, nil
}

func (p *Postgres) migrate(ctx context.Context) error {
    stmt := `
    CREATE TABLE IF NOT EXISTS products (
        id TEXT PRIMARY KEY,
        name TEXT NOT NULL,
        price DOUBLE PRECISION NOT NULL,
        stock INT NOT NULL DEFAULT 0,
        created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
    );
    `
    _, err := p.DB.ExecContext(ctx, stmt)
    return err
}

var ErrNotFound = errors.New("not found")

type ProductRow struct {
    ID        string
    Name      string
    Price     float64
    Stock     int32
    CreatedAt time.Time
}

func (p *Postgres) ListProducts(ctx context.Context) ([]*ProductRow, error) {
    rows, err := p.DB.QueryContext(ctx, `SELECT id, name, price, stock,
created_at FROM products`)
    if err != nil { return nil, err }
    defer rows.Close()

    var products []*ProductRow
    for rows.Next() {
        var pr ProductRow
        if err := rows.Scan(&pr.ID, &pr.Name, &pr.Price, &pr.Stock,
&pr.CreatedAt); err != nil {
            return nil, err
        }
        products = append(products, &pr)
    }
    return products, nil
}

func (p *Postgres) GetProduct(ctx context.Context, id string) (*ProductRow,
error) {
    row := p.DB.QueryRowContext(ctx, `SELECT id, name, price, stock,
created_at FROM products WHERE id=$1`, id)
    var pr ProductRow
    if err := row.Scan(&pr.ID, &pr.Name, &pr.Price, &pr.Stock,
&pr.CreatedAt); err != nil {
        if errors.Is(err, sql.ErrNoRows) { return nil, ErrNotFound }
        return nil, err
    }
    return &pr, nil
}
```

```go
func (p *Postgres) UpdateStock(ctx context.Context, id string, delta int32)
(*ProductRow, error) {
    _, err := p.DB.ExecContext(ctx, `UPDATE products SET stock = stock + $1
WHERE id=$2`, delta, id)
    if err != nil { return nil, err }
    return p.GetProduct(ctx, id)
}
```

product-service/internal/models/product.go

```go
package models

type Product struct {
    ID        string
    Name      string
    Price     float64
    Stock     int32
    CreatedAt string // RFC3339
}
```

product-service/internal/service/product_server.go

```go
package service

import (
    "context"
    "time"

    "github.com/example/ecommerce-platform/product-service/internal/db"
    "github.com/example/ecommerce-platform/product-service/internal/models"
    "github.com/example/ecommerce-platform/product-service/proto"
    "google.golang.org/grpc/codes"
    "google.golang.org/grpc/status"
)

type ProductServer struct {
    proto.UnimplementedProductServiceServer
    db *db.Postgres
}

func NewProductServer(database *db.Postgres) *ProductServer {
    return &ProductServer{db: database}
}

func (s *ProductServer) ListProducts(ctx context.Context, req
*proto.ListProductsRequest) (*proto.ListProductsResponse, error) {
    rows, err := s.db.ListProducts(ctx)
    if err != nil {
        return nil, status.Errorf(codes.Internal, "db: %v", err)
    }
    products := []*proto.Product{}
    for _, r := range rows {
        products = append(products, toProto(r))
    }
    return &proto.ListProductsResponse{Products: products}, nil
}

func (s *ProductServer) GetProduct(ctx context.Context, req
*proto.GetProductRequest) (*proto.GetProductResponse, error) {
    row, err := s.db.GetProduct(ctx, req.Id)
    if err != nil {
        if err == db.ErrNotFound { return nil, status.Error(codes.NotFound,
"product not found") }
        return nil, status.Errorf(codes.Internal, "db: %v", err)
    }
    return &proto.GetProductResponse{Product: toProto(row)}, nil
}

func (s *ProductServer) UpdateStock(ctx context.Context, req
*proto.UpdateStockRequest) (*proto.UpdateStockResponse, error) {
    row, err := s.db.UpdateStock(ctx, req.Id, req.Delta)
    if err != nil {
        if err == db.ErrNotFound { return nil, status.Error(codes.NotFound,
"product not found") }
        return nil, status.Errorf(codes.Internal, "db: %v", err)
    }
    return &proto.UpdateStockResponse{Product: toProto(row)}, nil
}

func toProto(r *db.ProductRow) *proto.Product {
    return &proto.Product{
        Id:        r.ID,
        Name:      r.Name,
        Price:     r.Price,
        Stock:     r.Stock,
        CreatedAt: r.CreatedAt.UTC().Format(time.RFC3339),
    }
}
```

product-service/main.go

```go
package main

```

```go
import (
    "log"
    "net"

    "github.com/example/ecommerce-platform/product-service/internal/config"
    "github.com/example/ecommerce-platform/product-service/internal/db"
    "github.com/example/ecommerce-platform/product-service/internal/service"
    "github.com/example/ecommerce-platform/product-service/proto"
    "google.golang.org/grpc"
    "google.golang.org/grpc/health"
    hpb "google.golang.org/grpc/health/grpc_health_v1"
    "google.golang.org/grpc/reflection"
)

func main() {
    cfg := config.FromEnv()

    pg, err := db.Connect(cfg.DatabaseURL)
    if err != nil { log.Fatalf("db connect: %v", err) }
    defer pg.DB.Close()

    lis, err := net.Listen("tcp", ":"+cfg.Port)
    if err != nil { log.Fatalf("listen: %v", err) }

    gsrv := grpc.NewServer()
    proto.RegisterProductServiceServer(gsrv, service.NewProductServer(pg))

    h := health.NewServer()
    hpb.RegisterHealthServer(gsrv, h)
    reflection.Register(gsrv)

    log.Printf("product-service listening on %s", lis.Addr().String())
    if err := gsrv.Serve(lis); err != nil {
        log.Fatalf("serve: %v", err)
    }
}
```

product-service/Dockerfile

```dockerfile
# --- Build stage ---
FROM golang:1.22 AS builder
WORKDIR /app

COPY go.mod go.sum ./
RUN go mod download

COPY . .
RUN CGO_ENABLED=0 GOOS=linux go build -o /product-service ./main.go
```

```
# --- Runtime stage ---
FROM gcr.io/distroless/base-debian12
COPY --from=builder /product-service /product-service
EXPOSE 50052
ENTRYPOINT ["/product-service"]
```

Updated deployments/docker-compose.yml

```
version: "3.9"
services:
  db_users:
    image: postgres:16
    container_name: users_db
    environment:
      POSTGRES_USER: app
      POSTGRES_PASSWORD: app
      POSTGRES_DB: users
    ports:
      - "5432:5432"
    volumes:
      - users-data:/var/lib/postgresql/data

  db_products:
    image: postgres:16
    container_name: products_db
    environment:
      POSTGRES_USER: app
      POSTGRES_PASSWORD: app
      POSTGRES_DB: products
    ports:
      - "5433:5432"
    volumes:
      - products-data:/var/lib/postgresql/data

  user-service:
    build: ../user-service
    container_name: user_service
    environment:
      DATABASE_URL: postgres://app:app@db_users:5432/users?sslmode=disable
      JWT_SECRET: devsecret
      PORT: "50051"
    depends_on:
      - db_users
    ports:
      - "50051:50051"

  product-service:
    build: ../product-service
    container_name: product_service
    environment:
      DATABASE_URL: postgres://app:app@db_products:5432/products?
sslmode=disable
      PORT: "50052"
    depends_on:
      - db_products
    ports:
      - "50052:50052"

volumes:
  users-data: {}
  products-data: {}
```

How to run (this step)

1. **Generate gRPC code**

```
cd ecommerce-platform/product-service
make proto
go mod tidy
```

2. **Run with Docker Compose**

```
cd ../deployments
docker compose up --build
```

3. **Test with grpcurl**

```
# ListProducts (empty if none inserted)
grpcurl -plaintext -d '{}' localhost:50052 product.v1.ProductService/
ListProducts

# Insert a product manually (psql into db_products) or extend service
with CreateProduct later.

# GetProduct
grpcurl -plaintext -d '{"id":"<PRODUCT_ID>"}' localhost:50052
product.v1.ProductService/GetProduct

# UpdateStock
grpcurl -plaintext -d '{"id":"<PRODUCT_ID>","delta":5}' localhost:50052
product.v1.ProductService/UpdateStock
```

Notes & next steps

- We’ll add a `CreateProduct` method later for admin use (seeding products).
- For now, you can insert products directly into Postgres.
- Next step: **order-service** (which will call `product-service` to check and deduct stock).

This completes Step 2: a working `product-service`.