Step 1: Set Up the REST API Using GoFiber Framework

We'll use GoFiber, a lightweight Go framework for web applications, to create a REST API that interacts with the deployed Hyperledger Fabric chaincode.

1. **Initialize a Go Project:**

#bash

mkdir asset-transfer-api

cd asset-transfer-api

go mod init asset-transfer-api

go get github.com/gofiber/fiber/v2

go get github.com/hyperledger/fabric-sdk-go

2. **Create main.go for the REST API:**

#go

```
package main

import (
    "fmt"
    "log"
    "os"

    "github.com/gofiber/fiber/v2"
    "github.com/hyperledger/fabric-sdk-go/pkg/client/channel"
    "github.com/hyperledger/fabric-sdk-go/pkg/fabsdk"
    "github.com/hyperledger/fabric-sdk-go/pkg/gateway"
    "github.com/hyperledger/fabric-sdk-go/pkg/core/config"
)

// Client to interact with the blockchain
var client *channel.Client
```

```
func main() {
    app := fiber.New()

    // Initialize Hyperledger Fabric SDK
    err := initHyperledgerSDK()
    if err != nil {
        log.Fatalf("Error initializing Fabric SDK: %v", err)
    }

    app.Post("/createAsset", createAsset)
    app.Get("/readAsset/:id", readAsset)
    app.Put("/updateAsset", updateAsset)
    app.Delete("/deleteAsset/:id", deleteAsset)

    log.Fatal(app.Listen(":3000"))
}

func initHyperledgerSDK() error {
    sdk, err := fabsdk.New(config.FromFile("./config.yaml"))
    if err != nil {
        return fmt.Errorf("failed to create SDK: %v", err)
    }

    clientContext := sdk.ChannelContext("mychannel", fabsdk.WithUser("User1"),
fabsdk.WithOrg("Org1"))
    client, err = channel.New(clientContext)
    if err != nil {
        return fmt.Errorf("failed to create channel client: %v", err)
    }

    return nil
```

```
}

// Handlers for creating, reading, updating, and deleting assets
func createAsset(c *fiber.Ctx) error {
    type Asset struct {
        ID             string `json:"ID"`
        Owner          string `json:"Owner"`
        Color          string `json:"Color"`
        Size           int    `json:"Size"`
        AppraisedValue int    `json:"AppraisedValue"`
    }

    var asset Asset
    if err := c.BodyParser(&asset); err != nil {
        return c.Status(400).SendString("Invalid request")
    }

    args := [][]byte{[]byte("CreateAsset"), []byte(asset.ID), []byte(asset.Owner), []byte(asset.Color),
[]byte(fmt.Sprint(asset.Size)), []byte(fmt.Sprint(asset.AppraisedValue))}

    response, err := client.Execute(channel.Request{
        ChaincodeID: "asset-transfer",
        Fcn:         "CreateAsset",
        Args:        args,
    })
    if err != nil {
        return c.Status(500).SendString(fmt.Sprintf("Failed to create asset: %v", err))
    }

    return c.SendString(string(response.Payload))
}
```

```
func readAsset(c *fiber.Ctx) error {
    id := c.Params("id")

    args := [][]byte{[]byte("ReadAsset"), []byte(id)}

    response, err := client.Query(channel.Request{
        ChaincodeID: "asset-transfer",
        Fcn:         "ReadAsset",
        Args:        args,
    })
    if err != nil {
        return c.Status(500).SendString(fmt.Sprintf("Failed to read asset: %v", err))
    }

    return c.SendString(string(response.Payload))
}

func updateAsset(c *fiber.Ctx) error {
    type Asset struct {
        ID             string `json:"ID"`
        Owner          string `json:"Owner"`
        Color          string `json:"Color"`
        Size           int    `json:"Size"`
        AppraisedValue int    `json:"AppraisedValue"`
    }

    var asset Asset
    if err := c.BodyParser(&asset); err != nil {
        return c.Status(400).SendString("Invalid request")
    }
```

```
    args := [][]byte{[]byte("UpdateAsset"), []byte(asset.ID), []byte(asset.Owner), []byte(asset.Color),
[]byte(fmt.Sprint(asset.Size)), []byte(fmt.Sprint(asset.AppraisedValue))}

    response, err := client.Execute(channel.Request{
        ChaincodeID: "asset-transfer",
        Fcn:         "UpdateAsset",
        Args:        args,
    })
    if err != nil {
        return c.Status(500).SendString(fmt.Sprintf("Failed to update asset: %v", err))
    }

    return c.SendString(string(response.Payload))
}

func deleteAsset(c *fiber.Ctx) error {
    id := c.Params("id")

    args := [][]byte{[]byte("DeleteAsset"), []byte(id)}

    response, err := client.Execute(channel.Request{
        ChaincodeID: "asset-transfer",
        Fcn:         "DeleteAsset",
        Args:        args,
    })
    if err != nil {
        return c.Status(500).SendString(fmt.Sprintf("Failed to delete asset: %v", err))
    }

    return c.SendString(string(response.Payload))
```

}

Step 2: Dockerize the REST API

1. **Create a Dockerfile:**

In your project directory, create a file called Dockerfile with the following content:

dockerfile

```
# Use the official Golang image to create a build artifact.
FROM golang:1.16-alpine AS build

# Set the current working directory inside the container
WORKDIR /app

# Copy the Go mod and sum files
COPY go.mod go.sum ./

# Download the Go dependencies
RUN go mod download

# Copy the rest of the application code
COPY . .

# Build the Go application
RUN go build -o main .

# Use a minimal image for deployment
FROM alpine:latest
WORKDIR /root/

# Copy the built binary from the build image
COPY --from=build /app/main .

# Expose port 3000 to the outside world
```

EXPOSE 3000

 # Run the binary program

CMD ["./main"]

2. **Build the Docker Image:**

Run the following command to build the Docker image:

#bash

docker build -t asset-transfer-api .

3. **Run the Docker Container:**

To run the Docker container from the image you just built:

#bash

docker run -p 3000:3000 asset-transfer-api

Step 3: Testing the REST API

Once the container is running, you can interact with the deployed chaincode using tools like Postman or cURL.

- **Create an Asset:**

#bash

```
curl -X POST http://localhost:3000/createAsset \
  -H "Content-Type: application/json" \
  -d '{"ID": "asset1", "Owner": "Alice", "Color": "blue", "Size": 5, "AppraisedValue": 300}'
```

- **Read an Asset:**

#bash

```
curl http://localhost:3000/readAsset/asset1
```

- **Update an Asset:**

#bash

```
curl -X PUT http://localhost:3000/updateAsset \
  -H "Content-Type: application/json" \
  -d '{"ID": "asset1", "Owner": "Bob", "Color": "green", "Size": 10, "AppraisedValue": 400}'
```

- **Delete an Asset:**

#bash

```
curl -X DELETE http://localhost:3000/deleteAsset/asset1
```