สยามชำนาญกิจ
SIAM CHAMNAN KIT

Workshop

Design good APIs

RESTful API
RPC

Start with REST

REST

Representational **S**tate **T**ransfer

Scalability of component interactions
Generality of interfaces
Independent deployment of components
Intermediary components to reduce interaction latency

URI (Uniform Resource Identifier)

Method to access a resource on your server

URI = scheme "://" authority "/" **path** ["?" query] ["#" fragment"]

URI = http://myserver.com/**mypath**?query=1#document

Underscores _ should not be used in URIs

Lowercase letters are preferred as case sensitivity is a differentiator in the path part of a URI

Design URI path for REST

Collection
Document
Controller

Collection

Directory of resources
Parameters to access document
Always use a plural noun for collection name

```
GET /cats       # All cats in collection
GET /cats/1     # A document of cat 1
```

Document

Resource pointing to a single object
It’s have child resources

```
GET /cats/1      # A document of cat 1
GET /cats/1/kittens    # All kittens document of cat 1
GET /cats/1/kittens/1    # kittens 1 of cat 1
```

Controller

Controller resource is like a procedure/method
Use when can’t map to CRUD
Always use a verb

```
POST /cats/1/feed     # Feed cat 1
POST /cats/1/feed?food=fish    # Food cat 1 with a fish
```

HTTP Verb

Name	Description
GET	Retrieve a resources
POST	Create a new resource in a collection or to execute a controller
PUT	Update a resource
DELETE	Remove a resource
PATCH	Perform partial update
HEAD	Retrieve the headers for a resources without body

URI query design

Filter
Pagination
Sorting
Search
Versioning

https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9

Filter

GET /cats?**color=white&sex=male**
GET /cats?**age=gte:5**
GET /cats?**age=lt:5**

Pagination

GET /cats?**page=10**
GET /cats?**limit=10**
GET /cats?**limit=10&offset=10**

Sorting

GET /cats?**sort=age_asc**
GET /cats?**sort=age_desc**

GET /cats?**sort=+age**
GET /cats?**sort=-age**

GET /cats?**sort=age&order=asc**
GET /cats?**sort=age&order=desc**

Search

GET /cats?**search=keyword**
GET /cats?**q=keyword**

Versioning

GET **/v1**/cats
GET **/v2**/cats
GET /cats?**api-version=2**

GET /cats
api-version=2

APIs should more readable and easy to understand

Response code

Code	Description
2xx	Success
3xx	Redirect
4xx	Client error
5xx	Server error

Response data (1)

POST /cats
RESPONSE HTTP 200 OK

```
{
  "status": 400,
  "statusMessage": "Bad Request"
}
```

Response data (2)

POST /cats
RESPONSE HTTP 400 BAD REQUEST

```
{
  "errorMessage": "Name should be"
}
```

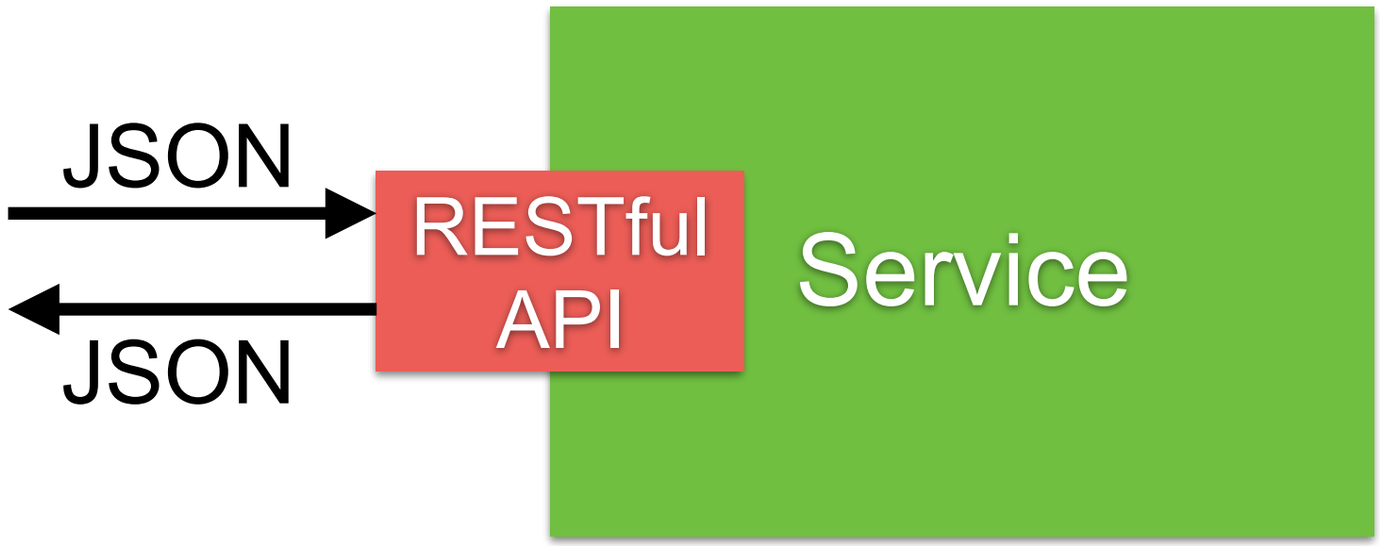
API Documentation

Swagger
API Blueprint
RAML (RESTful API Modeling Language)

Develop RESTful API with Golang

Develop RESTful API with go

net/http package
encoding/json package

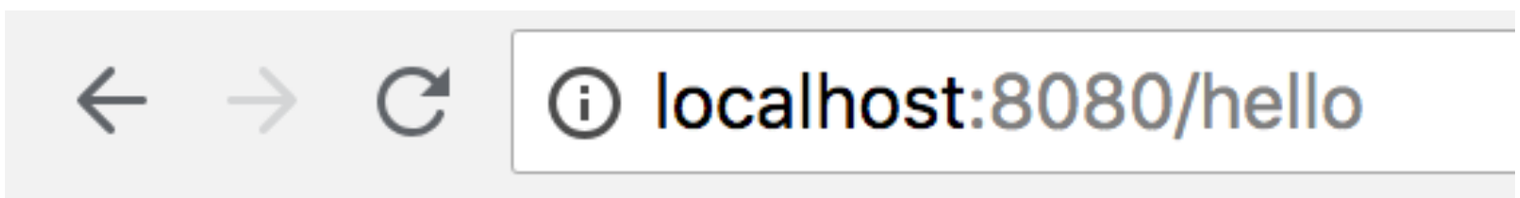


Step 1 Hello API

```
package main

import (
  "net/http"
  "fmt"
  "log"
)

func main() {
  port := 8080
  http.HandleFunc("/hello", helloHandler)
  log.Printf("Server starting on port %v\n", port)
  log.Fatal(http.ListenAndServe(fmt.Sprintf(":%v", port), nil))
}

func helloHandler(w http.ResponseWriter, r *http.Request) {
  log.Printf("Called helloHandler\n")
  fmt.Fprint(w, "Hello World")
}
```

Run program

$go run step_01.go




```
Hello World
```

Build binary

$go build step_01.go

Return response in JSON

Step 2 Response with JSON

We need JSON data ….

```
{
   "header":{
      "code":200,
      "description":"Success"
   },
   "body":{
      "message":"Hello World"
   }
}
```

Step 2 Response with JSON

encoding/json package

```
package main

import (
   "net/http"
   "fmt"
   "log"
   "encoding/json"
)
```

Step 2 Response with JSON

Create data format with Struct

```
type helloWorldResponse struct {
  Header headerResponse  `json:"header"`
  Body bodyResponse `json:"body"`
}

type headerResponse struct {
  Code int `json:"code"`
  Description string `json:"description"`
}

type bodyResponse struct {
  Message string `json:"message"`
}
```

Step 2 Response with JSON

Create response of API

```
func helloHandler(w http.ResponseWriter, r *http.Request) {
  response := helloWorldResponse {
    Header: headerResponse{
      Code: 200,
      Description: "Success",
    },
    Body: bodyResponse { Message: "Hello World" },
  }

  data, err := json.Marshal(response)
  if err != nil {
     panic("Ooops")
  }
  fmt.Fprint(w, string(data))
}
```

Run and see result

$go run step_02.go

Way to working with JSON

json.Marshal()
json.NewEncoder()

Benchmark of two method

$go test -v -run="none" -bench=.
-benchtime="5s" -benchmem

```
BenchmarkHelloHandlerVariable-4                 10000000               559 ns/op
       4 allocs/op
BenchmarkHelloHandlerEncoder-4                  20000000               341 ns/op
       1 allocs/op
BenchmarkHelloHandlerEncoderReference-4         20000000               301 ns/op
       0 allocs/op
PASS
```

step_02/step_02_test.go

Step 3 Change to better solution

Create response of API

```
func helloHandler(w http.ResponseWriter, r *http.Request) {
  response := helloWorldResponse {
    Header: headerResponse{
      Code: 200,
      Description: "Success",
    },
    Body: bodyResponse { Message: "Hello World" },
  }

  encoder := json.NewEncoder(w)
  encoder.Encode(&response)
}
```

Run and see result

$go run step_03.go

Send JSON to API

Step 4 Send JSON to API

try to convert JSON to go struct

```
#Input
{
  "name": "Somkiat"
}
```

```
#Output
{
  "message": "Hello Somkiat"
}
```

Step 4 Send JSON to API

create struct of request and response

```
type helloWorldResponse struct {
  Message string `json:"message"`
}
```

```
type helloWorldRequest struct {
  Name string `json:"name"`
}
```

Step 4 Send JSON to API

API receive and convert JSON to go struct

```
func helloHandler(w http.ResponseWriter, r *http.Request) {
   //Request
   var request helloWorldRequest
   decoder := json.NewDecoder(r.Body)
   err := decoder.Decode(&request)
   if err != nil {
      http.Error(w, "Bad request", http.StatusBadRequest)
      return
   }

   //Response
}
```

Run and see result

$curl localhost:8080/hello -d '{"name":"Somkiat"}'

{"message":"Hello Somkiat"}

Try to benchmark again

DRY (Don’t Repeat Yourself)

chi

Gin

Echo

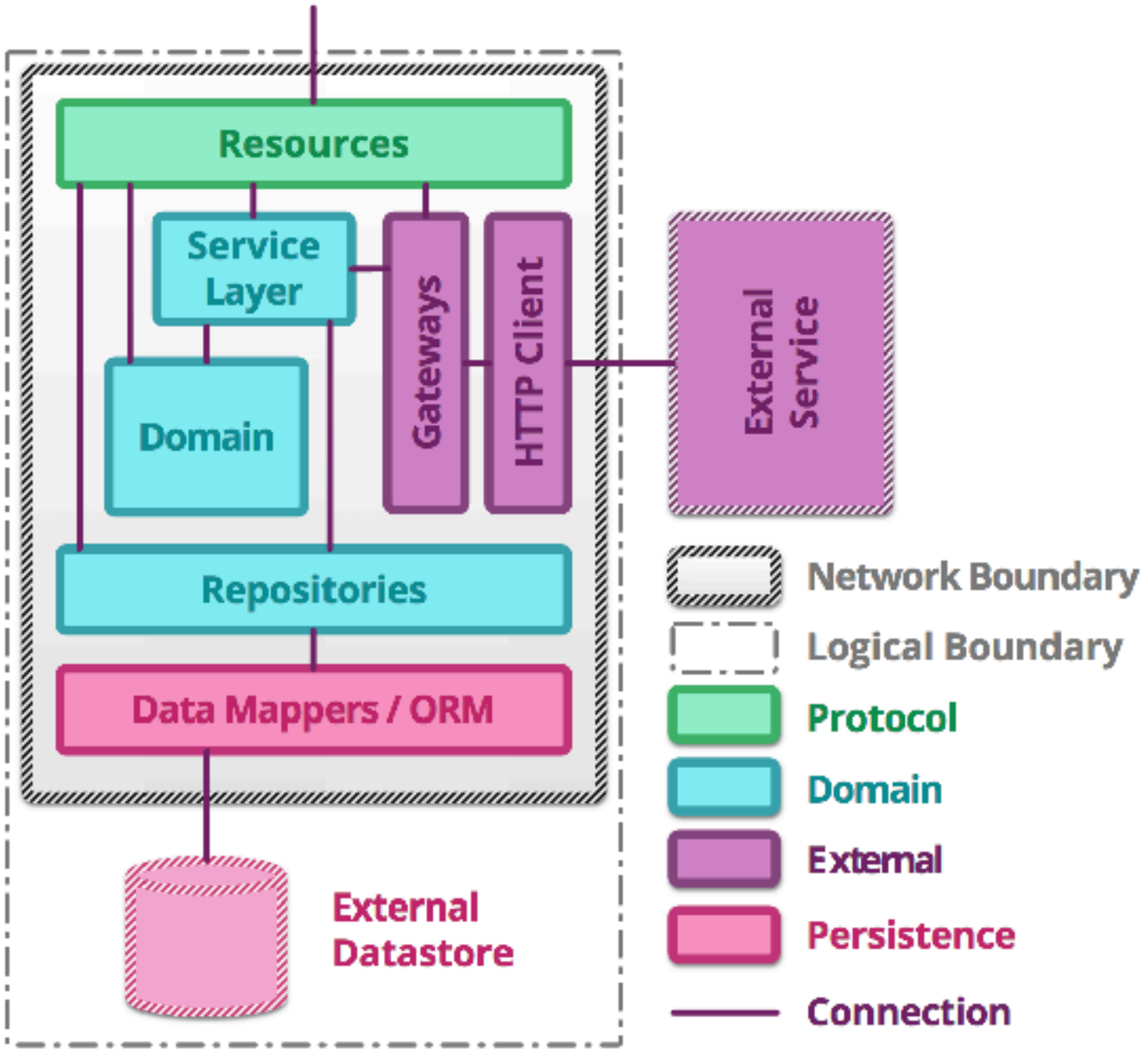
Gokit

Go-micro

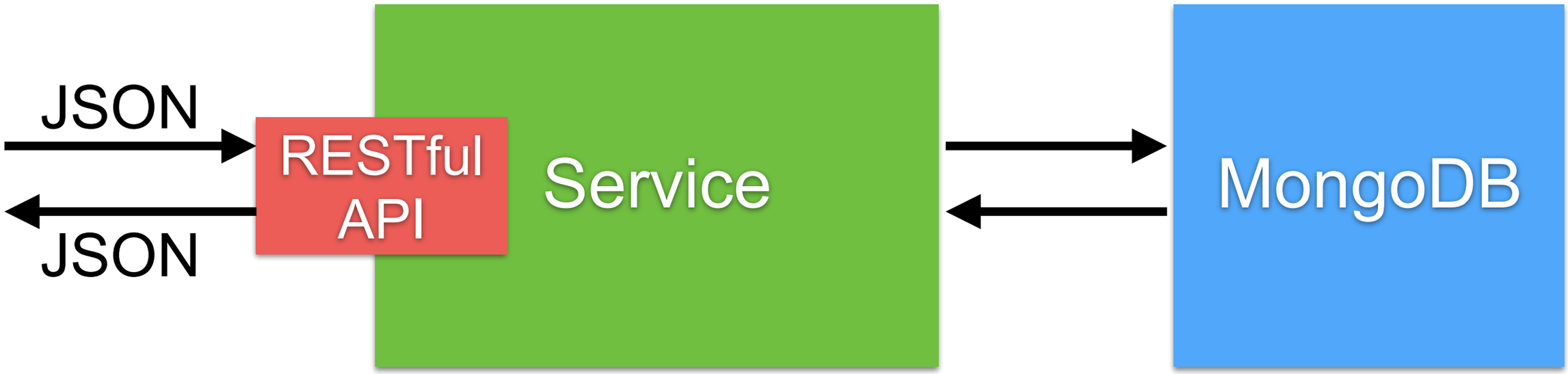
and more ….

Building search service

Project Structure



Search service



Search service structure

api => search_api.go
repository => mongo_repository.go
model => product.go
main => main.go

search_api/step_01

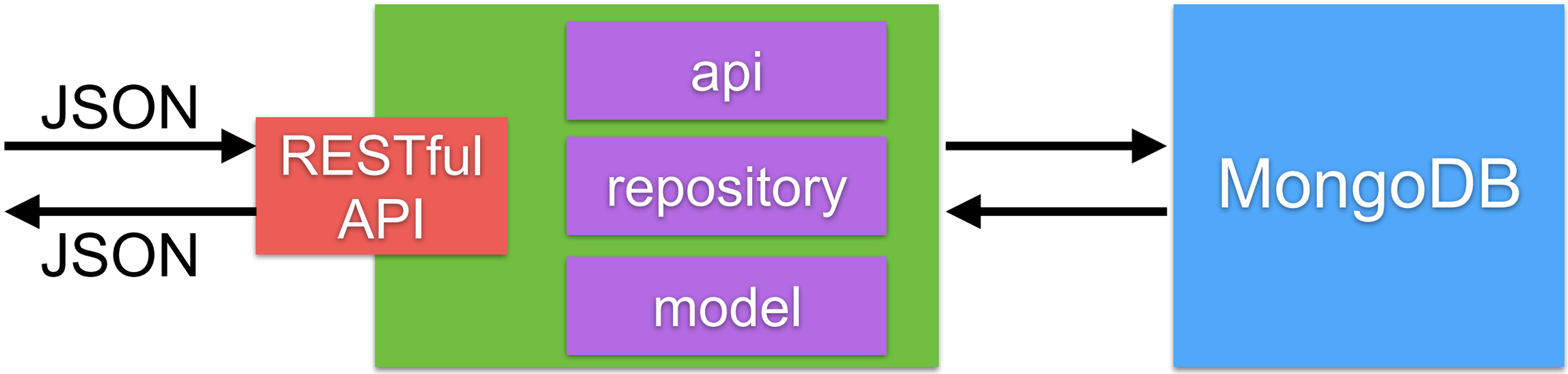
How to run with go ?

$sh run.sh

```
CURDIR=`pwd`
OLDGOPATH=$GOPATH
export GOPATH=$CURDIR
gofmt -w src/
go install main
export GOPATH=$OLDGOPATH
```

Let’s start to develop service

search_api/step_02



1. Create model

model/product.go

```
package model

type Product struct {
    Id    string
    Name  string
    Price float32
}
```

2. Search data from MongoDB

How to connect to MongoDB with go ?

https://labix.org/mgo

2. Search data from MongoDB

How to add library/dependency to project ?

$go get gopkg.in/mgo.v2

$go get gopkg.in/tomb.v2

2. Search data from MongoDB

Go get try to download code into /src !!

2. Search data from MongoDB

Move **/gopkg.in** to **/src/vendor**

2. Search data from MongoDB

repository/mongo_repository.go

```
type Store interface {
  Search(name string) []model.Product
}
```

```
type MongoStore struct {
  session *mgo.Session
}
```

2. Search data from MongoDB

repository/mongo_repository.go

```
func (m *MongoStore) Search(name string) []model.Product {
   s := m.session.Clone()
   defer s.Close()

   var results []model.Product
   c := s.DB("productserver").C("products")
   err := c.Find(model.Product{Name: name}).All(&results)
   if err != nil {
      return nil
   }

   return results
}
```

Repository structure

Build and run search service

$sh run.sh

```
?       api     [no test files]
?       main    [no test files]
?       model   [no test files]
?       repository      [no test files]
```

3. Create search API

api/search_api.go

3. Create search API

Create struct of request and response

```
type searchRequest struct {
  Query string `json:"q"`
}
```

```
type searchResponse struct {
  Products []model.Product `json:"products"`
}
```

3. Create search API

Create handler for search service

```
type Search struct {
   DataStore repository.Store
}
```

3. Create search API

Create handler for search service

```
func (s *Search) ServeHTTP(rw http.ResponseWriter, r *http.Request) {
   decoder := json.NewDecoder(r.Body)
   defer r.Body.Close()

   request := new(searchRequest)
   err := decoder.Decode(request)

   products := s.DataStore.Search(request.Query)

   encoder := json.NewEncoder(rw)
   encoder.Encode(searchResponse{Products: products})
}
```

4. Create main of service

main/main.go

```
func main() {
   serverURI := "localhost"
   if os.Getenv("MONGODB_SERVER") != "" {
      serverURI = os.Getenv("MONGODB_SERVER")
   }

   store, err := repository.NewMongoStore(serverURI)
   if err != nil {
      log.Fatal(err)
   }

   handler := api.Search{DataStore: store}
   err = http.ListenAndServe(":8080", &handler)
   if err != nil {
      log.Fatal(err)
   }
}
```

Build and run search service

$sh run.sh

```
?       api     [no test files]
?       main    [no test files]
?       model   [no test files]
?       repository      [no test files]
```

Run search service

$./bin/main

no reachable servers

We need MongoDB server

Build Ship Run with Docker

Working with Docker-compose

open file docker-compose.yml

```
version: '3'
services:
  mongodb:
    image: mongo:4
    ports:
      - 27017:27017
```

https://hub.docker.com/_/mongo/

Start MongoDB server with

$docker-compose up -d

```
Creating network "step_02_default" with the default driver
Pulling mongodb (mongo:4)...
4: Pulling from library/mongo
b234f539f7a1: Downloading [==================================>                ]   32.5MB/43.12MB
Download complete
5ba5bbeb6b91: Download complete
43ae2841ad7a: Download complete
851B/851B190: Download complete
b270872207e3: Download complete
bd7d91d60f98: Download complete
1020ba9c757f: Download complete
398b5f5b19a9: Download complete
ec34a1504b9b: Download complete
6c52301152b7: Download complete
aca6ce6bd5b2: Downloading [================>                                  ]  27.36MB/86.46MB
wnload complete
8395dda89cc8: Download complete
```

Run search service

$./bin/main

Try to call service

$curl localhost:8080/hello -d '{"q":"Somkiat"}'

Build and run search service

$sh run.sh

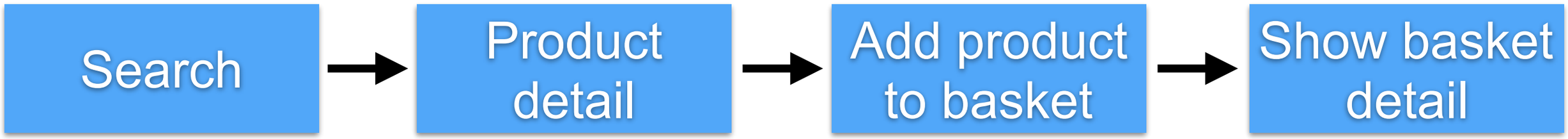
```
?       api     [no test files]
?       main    [no test files]
?       model   [no test files]
?       repository      [no test files]
```

Testing ?

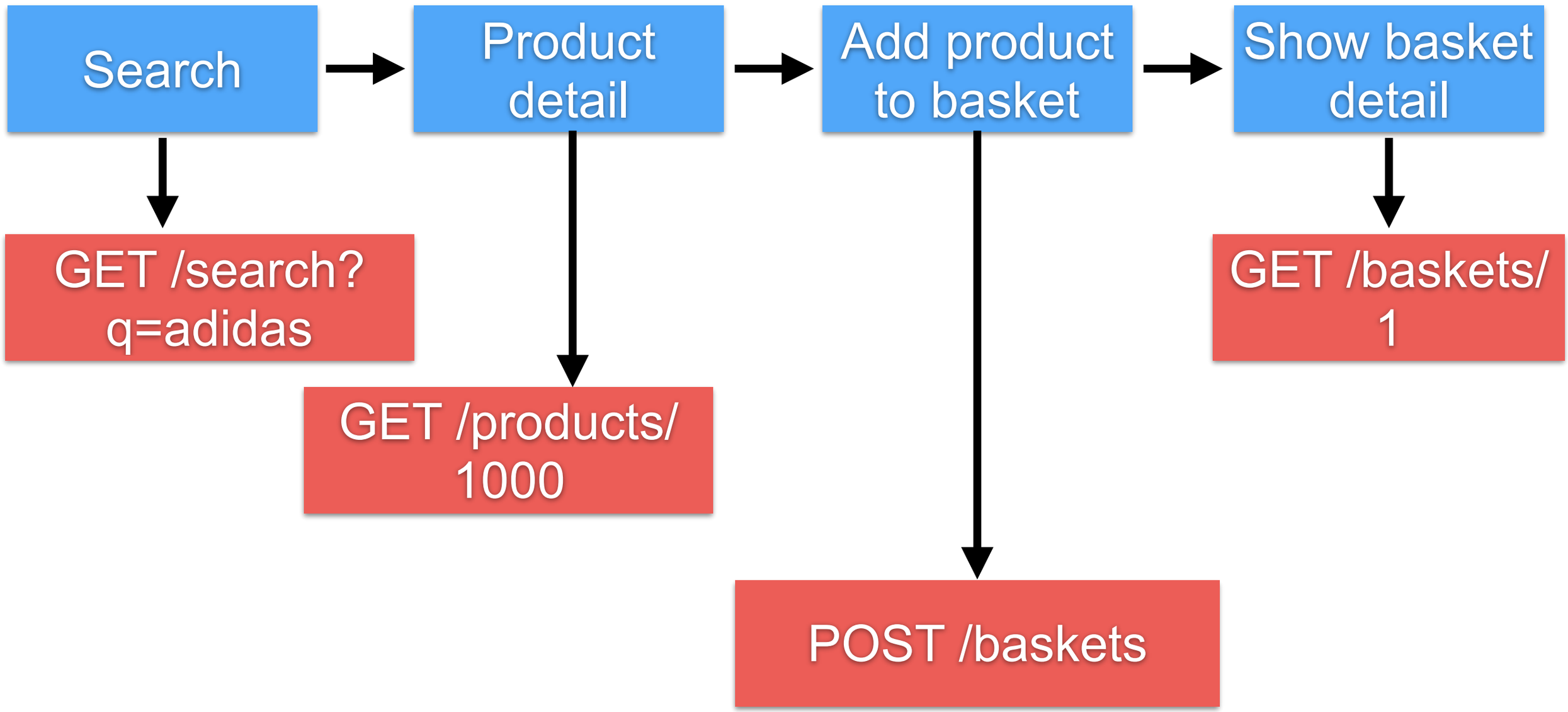
Start with acceptance testing

mock_api

Acceptance Testing



Acceptance Testing



Try to mock REST APIs

Stubby4j
WireMock
JSON Server
more …

Stubby4j

$java -jar stubby4j-6.0.1.jar --data **api.yaml**

```
Loaded: [GET] /search?q=adidas
Loaded: [GET] /product/1000
Loaded: [POST] /baskets
Loaded: [GET] /baskets/1
```

https://github.com/azagniotov/stubby4j

Create acceptance test

atdd/

Install robotframework

$pip install robotframework

http://robotframework.org/

Install library for API testing

$pip install -U requests
$pip install -U robotframework-requests

https://github.com/bulkan/robotframework-requests

Create first test case

Create file **first_case.robot**

```
1     *** Settings ***
2     Library    RequestsLibrary
3     Library    Collections
4
5     *** Variables ***
6
7     *** Testcases ***
8
9     *** Keywords ***
10
11
```

Run test

$pybot first_case.robot

Add new test case

Readable and easy to understand

```
7     *** Testcases ***
8     Try to add one product to empty basket
9       Search product by keyword adidas
10      Get product detail of id=1000
11      Add product id=1000 to empty basket
12      Get basket detail of id=1
```

Run test

$pybot first_case.robot

Create first keyword

Search product by keyword adidas

```
*** Keywords ***
Search product by keyword adidas
  Create Session    search    http://localhost:8882
  ${response}=  Get Request  search  /search?q=adidas
  Should Be Equal   ${response.status_code}  ${200}
  Should Be Equal   ${response.json()['header']['code']}   ${200}
${size_of_product}=  Get length  ${response.json()['body']}
  Should Be Equal   ${size_of_product}  ${2}
  Should Be Equal   ${response.json()['body'][0]['product_id']}  ${1000}
```

Working with HTTP Post

Add product id=1000 to empty basket

```
Add product id=1000 to empty basket
  Create Session    baskets    http://localhost:8882
  &{headers}=  Create Dictionary  Content-Type=application/json
  &{data}=  Create Dictionary
  ...  product_id=${1000}
  ...  product_name=Adidas
  ...  product_price=${1500}
  ...  product_image=http://xxx.jpg
  ...  quantity=${1}
  ${response}=  Post Request  baskets  /baskets
  ...  data=${data}  headers=${headers}
  Should Be Equal As Strings  ${response.status_code}  200
```

Create more keyword

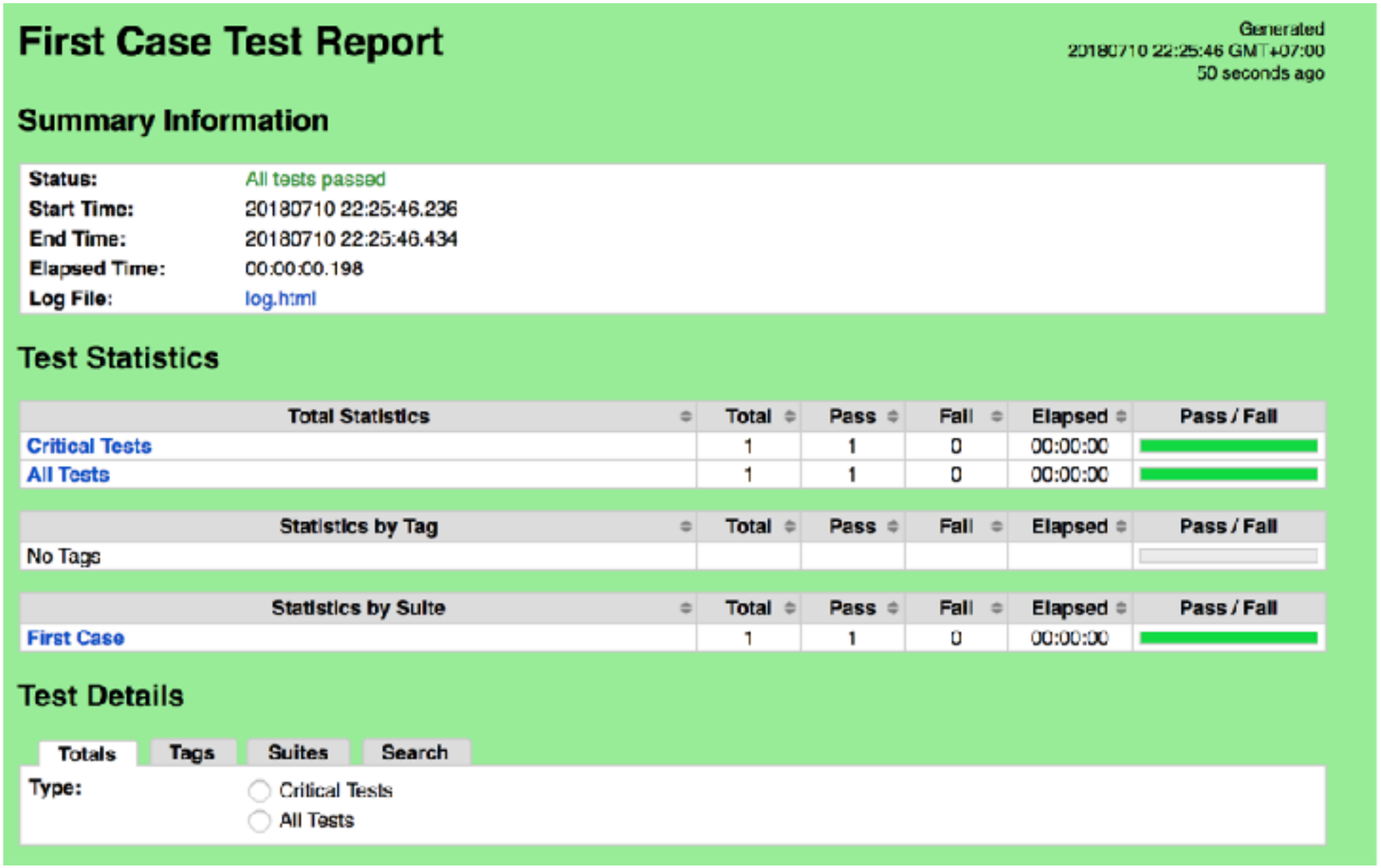
Run test

$pybot first_case.robot

```
                    ..                  _
==============================================================================
First Case
==============================================================================
Try to add one product to empty basket                                | PASS |
------------------------------------------------------------------------------
First Case                                                            | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
```

Testing report

Open file report.html



First Case Test Report

Generated
20180710 22:25:46 GMT+07:00
50 seconds ago

Summary Information

Status:	All tests passed
Start Time:	20180710 22:25:46.236
End Time:	20180710 22:25:46.434
Elapsed Time:	00:00:00.198
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:00	
All Tests	1	1	0	00:00:00	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
First Case	1	1	0	00:00:00	

Test Details

Totals | Tags | Suites | Search

Type: ○ Critical Tests ○ All Tests

Testing report

Open file report.html

Test Execution Log

SUITE **First Case**

Full Name:	First Case
Source:	/Users/somkiat/data/slide/microservice/slide/demo-go/atdd/first_case.robot
Start / End / Elapsed:	20180710 22:25:46.236 / 20180710 22:25:46.434 / 00:00:00.198
Status:	1 critical test, 1 passed, 0 failed 1 test total, 1 passed, 0 failed

TEST **Try to add one product to empty basket**

Full Name:	First Case.Try to add one product to empty basket
Start / End / Elapsed:	20180710 22:25:46.359 / 20180710 22:25:46.434 / 00:00:00.075
Status:	PASS (critical)

KEYWORD **Search product by keyword adidas**

KEYWORD **Get product detail of id=1000**

KEYWORD **Add product id=1000 to empty basket**

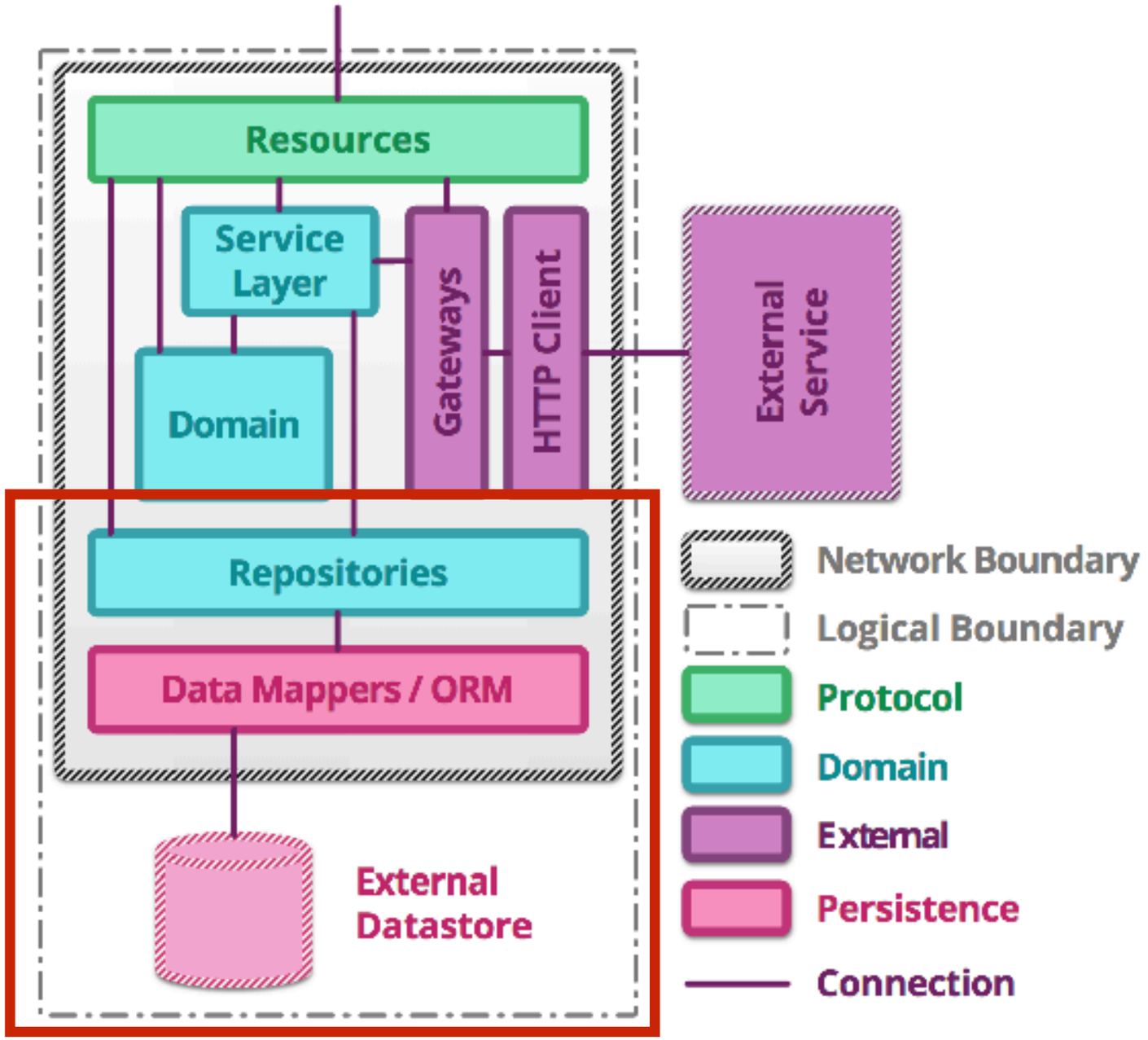
KEYWORD **Get basket detail of id=1**

Start to develop real APIs

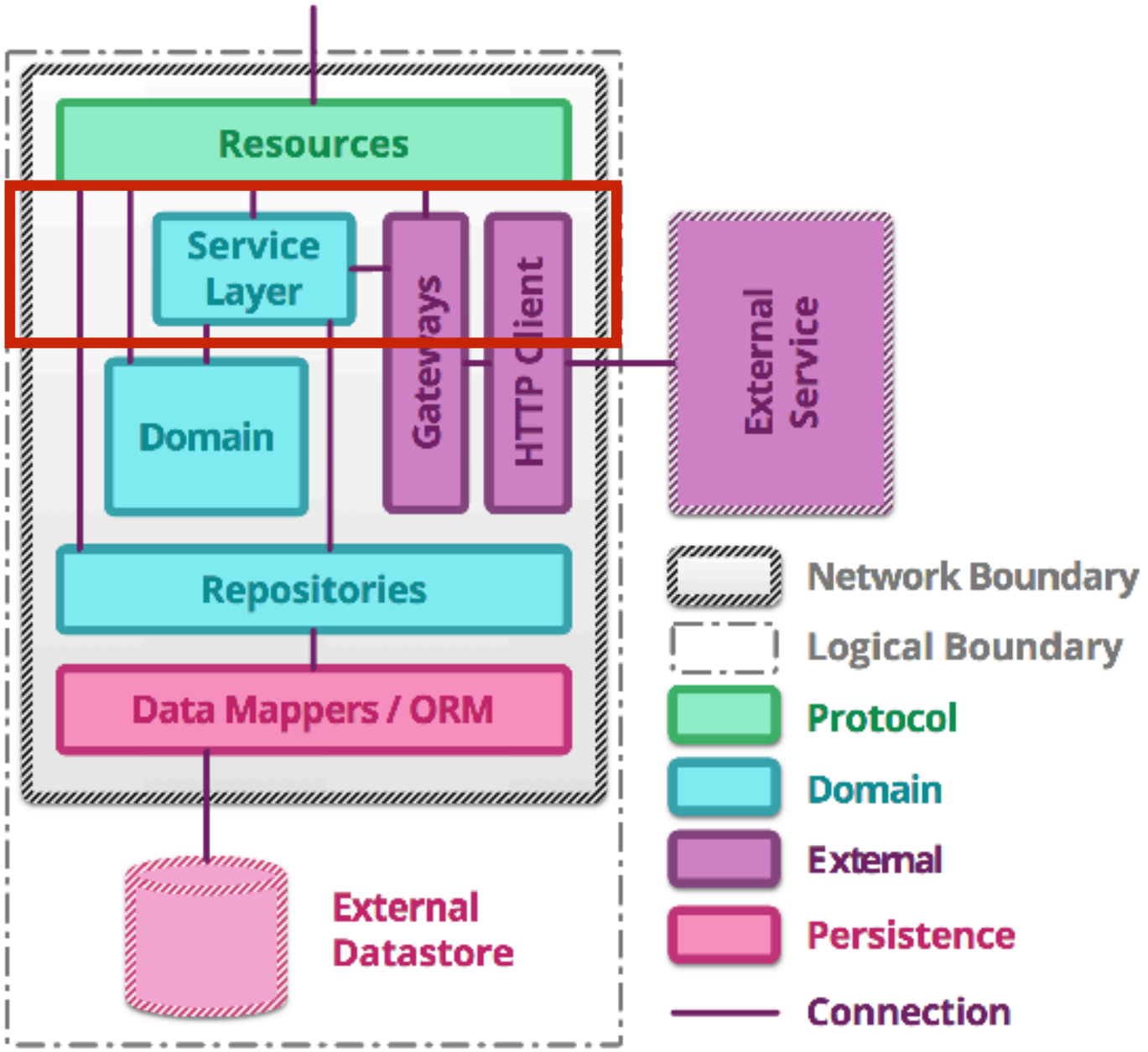
More testing …

Slice testing
in each layer/package

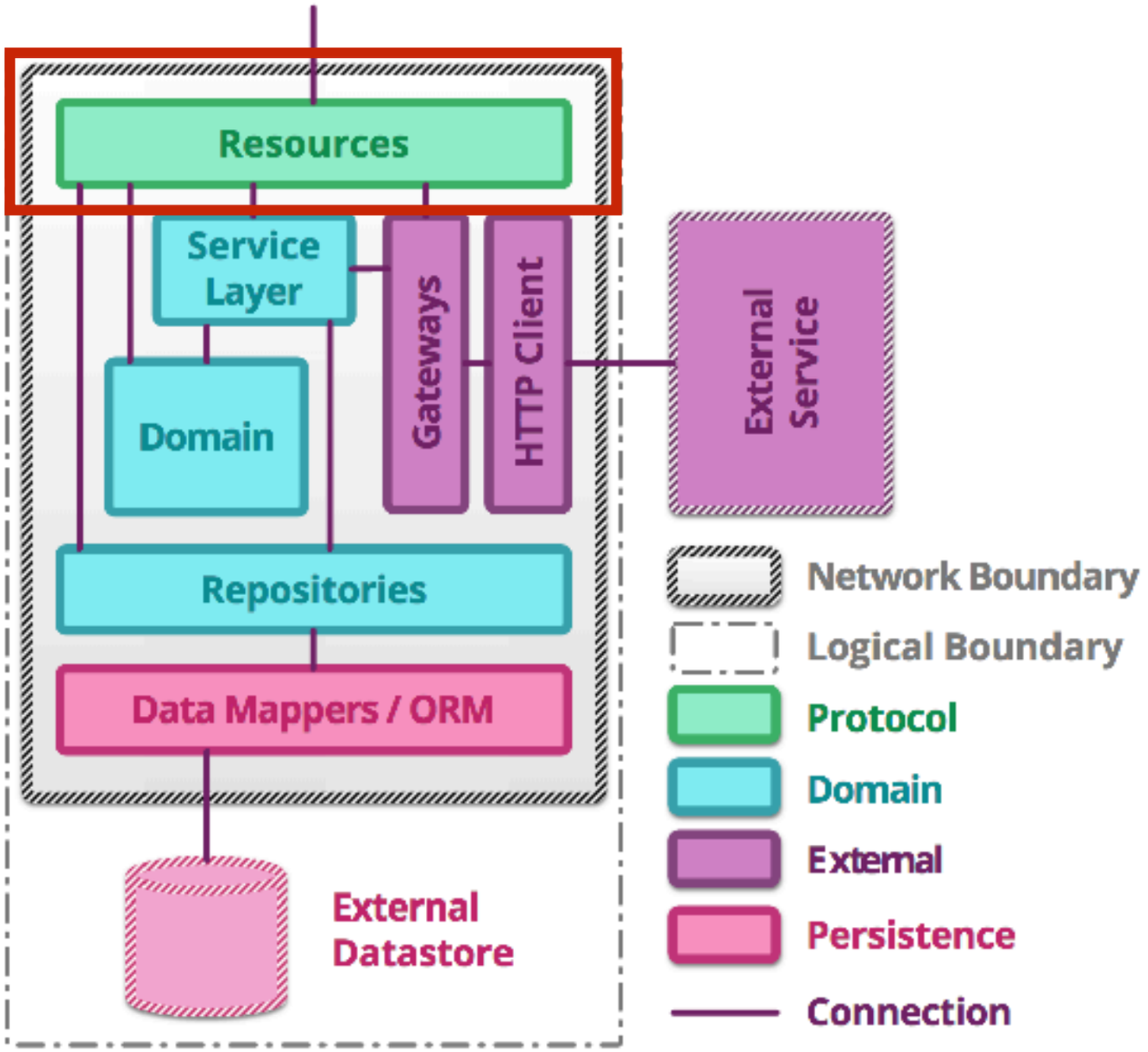
Repository testing



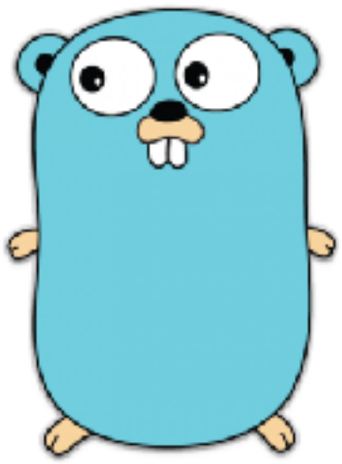
Service testing



API/Resources testing



Deploy with Docker



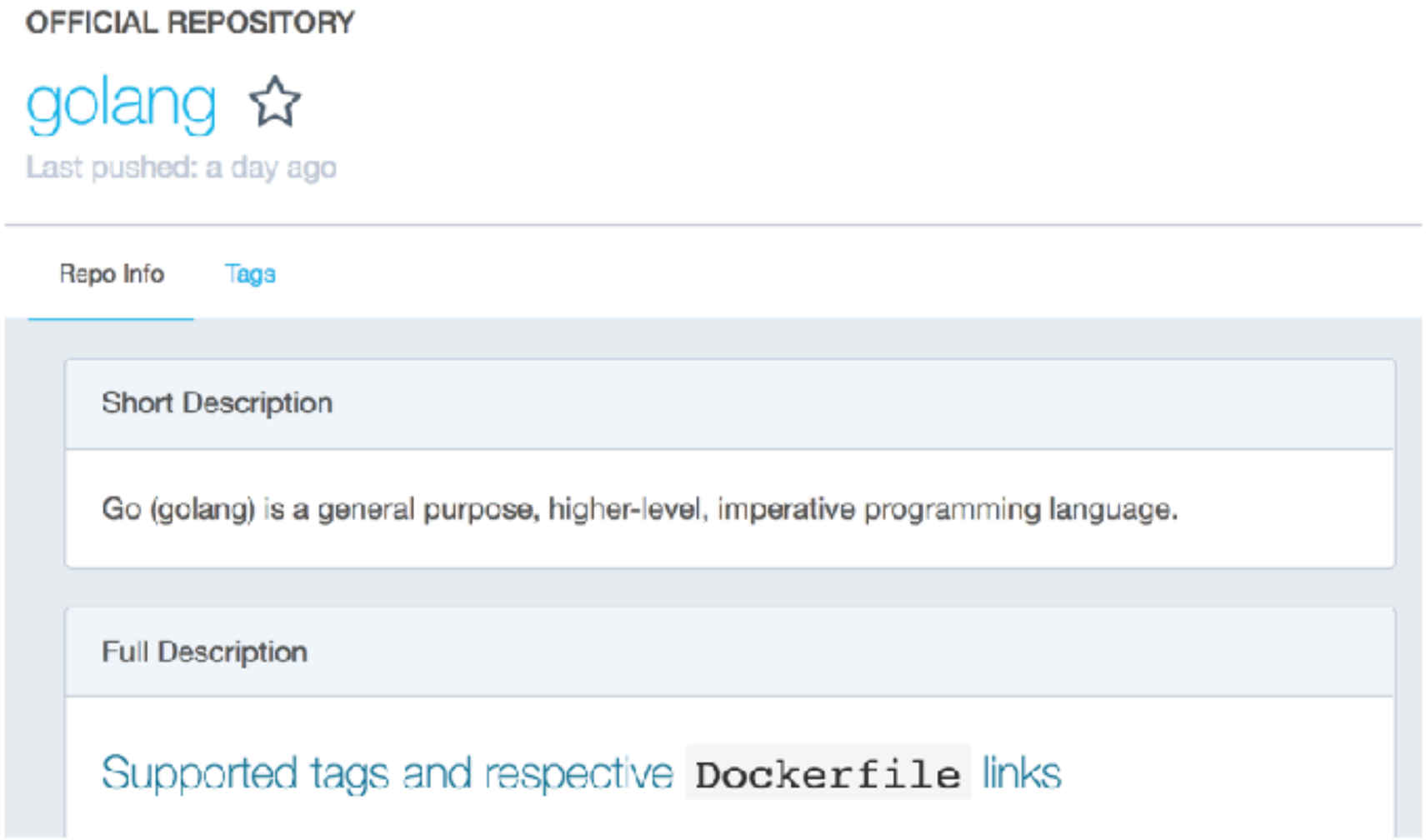
Go

Basic of Docker

Image
Container
Volume
Dockerfile
Docker-compose

Image

Template/blueprint of container



https://hub.docker.com/_/golang/

Download Image

$docker image pull <image name>

List all images

$docker image ls

Container

$docker container run <image name>

Working with Go project

1. Build binary file
2. Run binary in container

Build binary file

```
$docker container run --rm -p 8080:8080
-v $(pwd):/go/ -w /go
golang:1.10.3-alpine3.7 go build -o main main
```

Run binary in container

```
$docker container run --rm
-p 8080:8080
-v $(pwd):/go/ -w /go
golang:1.10.3-alpine3.7 ./main
```

Make it easy with Dockerfile

Dockerfile

```
FROM golang:1.10.3-alpine3.7
WORKDIR /go
COPY . /go
RUN go build -o main main
CMD ["/go/main"]
```

Build image from Dockerfile

$docker image build -t sample:0.1 .

Run binary with container

$docker container run --rm -p 8080:8080 sample:0.1

Using multi-stage build

Dockerfile2

```
FROM golang:1.10.3-alpine3.7 as builder
WORKDIR /go
COPY . /go
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app
main

FROM scratch
WORKDIR /root/
COPY --from=builder /go/app .
CMD ["./app"]
```

Build image from Dockerfile

$docker image build -t sample:0.2
-f Dockerfile2 .

Run binary with container

$docker container run --rm -p 8080:8080 sample:0.2