CSIT6000O Group Project Report

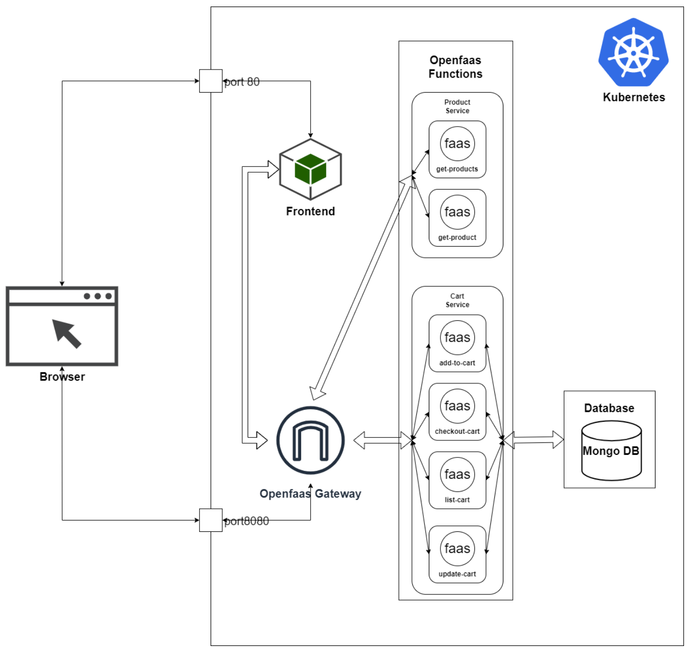
Serverless Shopping Cart

Name	**SID**
Niu Yutong	20176192
BAI Qian	20786466
ZHANG Yiwen	20790766
TONG Jinjian	20788658

1. Introduction

In this project, we modify an existing open source project from aws-sample/serverless-shopping-cart and successfully deploy it on our own server with minikube. We replace all the aws components used in this project with open-source components, including AWS Lambda to Openfaas and DynamoDB to MongoDB, and AWS API gateway to Openfaas gateway. We containerized all the components and made the project easy to deploy on any Kubernetes cluster. We disabled the Login function provided by the AWS Cognito Authorizer and only used shopping cart ID to distinguish different customers for simplicity. The frontend is also updated to integrate with the new infrastructure design.

2. Infrastructure



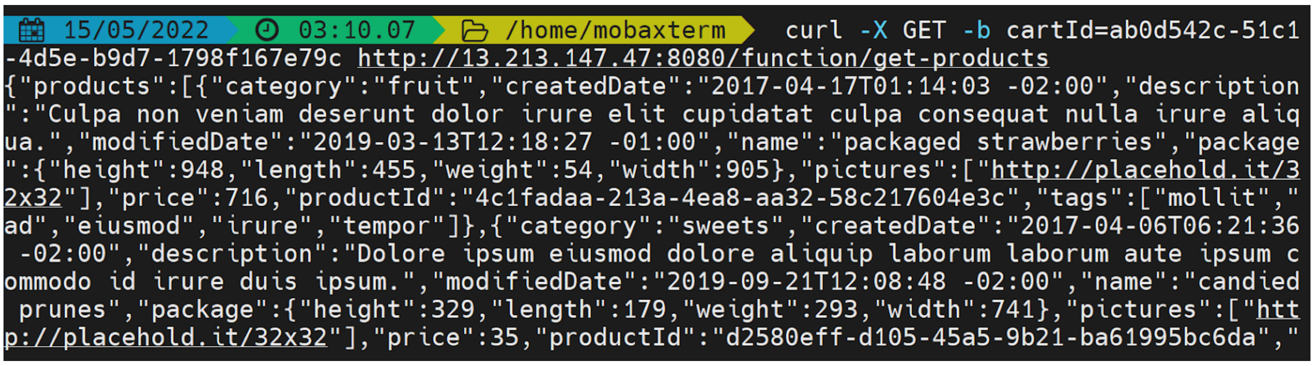
The project is ready to deploy to the Kubernetes cluster. All the components are ready as Docker images. DockerHub is used for the docker image registry. We build an automation script to facilitate the simplest situation where the project is deployed to a single AWS EC2 instance running the minikube single-node Kubernetes cluster. The single Kubernetes node exposes two ports: HTTP port 80 for the Web application frontend and TCP port 8080 for openfaas functions. Port forwarding is used to project the service from Kubernetes services to node port on EC2 instance. All the deployments and services are deployed to a single Kubernetes namespace openfaas-fn for simplicity. Inside the namespace, the service is identified with Kubernetes local DNS names. For example, the database service can be identified as MongoDB-service.openfaas-fn.svc.cluster.local. And the openfaas function such as get-products can be identified as get-products.openfaas-fn.svc.cluster.local. Traffic between openfaas functions and DB is within the local Kubernetes network all the time for better network security. While the communication between the frontend web application and backend openfaas functions is through the host node port for the possibility of separation of frontend and backend on different servers.

3. Backend

The backend services use openfaas in replacement of Lambda functions provided by AWS. Openfaas functions have the advantage that they can be deployed to any Kubernetes cluster since all the functions are actually built into Docker images. To implement the openfaas functions, we choose the template python3-http provided by openfaas. The template implements the openfaas functions based on the Flask framework of python3. Openfaas also provides a gateway to replace the API gateway provided by AWS. The gateway also provides a useful UI for testing the openfaas functions. The functions process the data in the request payloads and update the Mongo DB accordingly. The DB service can be accessed through the local Kubernetes service DNS. The functions can be categorized into two groups: product service and cart service. The product service is a mock product database. The product details are written into a static JSON file. The cart service saves the cart ID from the request cookie and updates DB with the corresponding cart ID. Below can find the details for our openfaas function implementation.

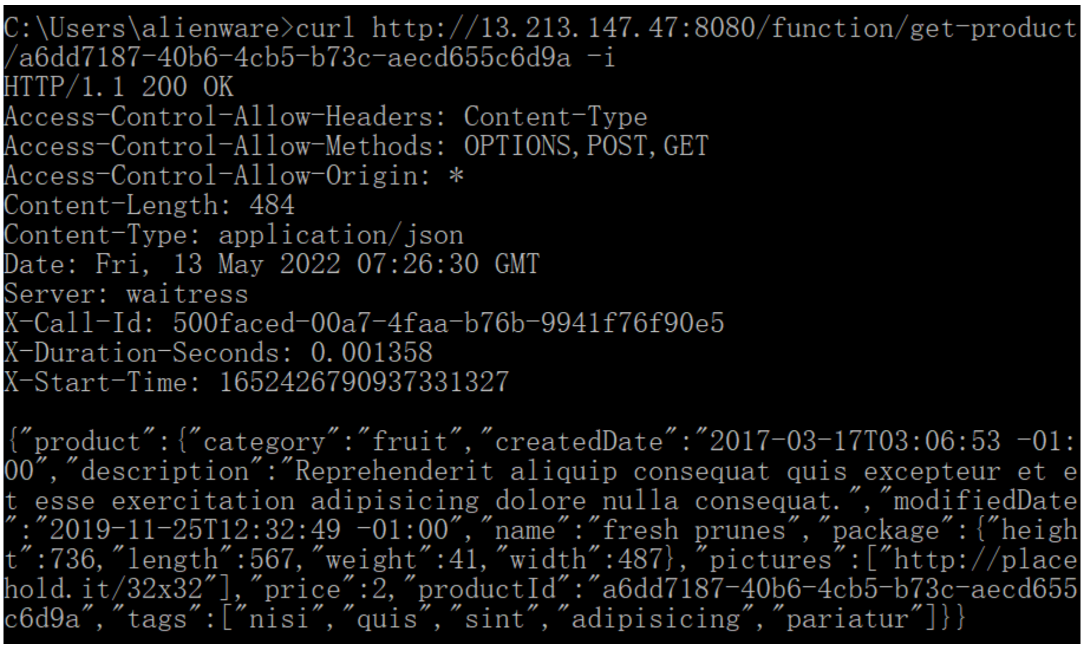
Functions:

- Get-products: Get-products is a function that returns a list of products called by the frontend. The function is called by the GET method and accessed directly by {url}/function/get-products. It returns all the products defined in the product_list JSON file.



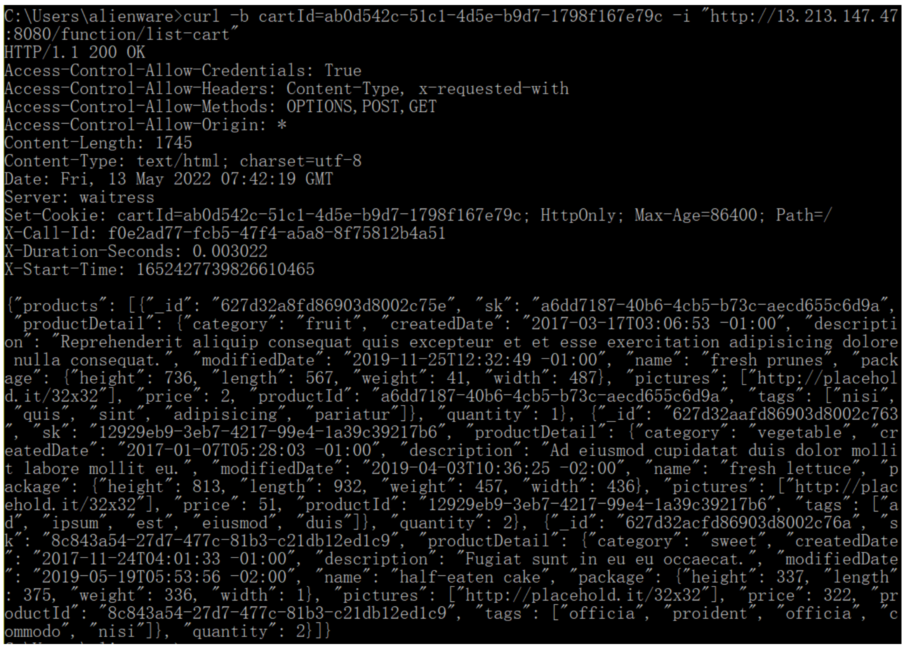
```
 15/05/2022   03:10.07   /home/mobaxterm   curl -X GET -b cartId=ab0d542c-51c1
-4d5e-b9d7-1798f167e79c http://13.213.147.47:8080/function/get-products
{"products":[{"category":"fruit","createdDate":"2017-04-17T01:14:03 -02:00","description
":"Culpa non veniam deserunt dolor irure elit cupidatat culpa consequat nulla irure aliq
ua.","modifiedDate":"2019-03-13T12:18:27 -01:00","name":"packaged strawberries","package
":{"height":948,"length":455,"weight":54,"width":905},"pictures":["http://placehold.it/3
2x32"],"price":716,"productId":"4c1fadaa-213a-4ea8-aa32-58c217604e3c","tags":["mollit","
ad","eiusmod","irure","tempor"]},{"category":"sweets","createdDate":"2017-04-06T06:21:36
 -02:00","description":"Dolore ipsum eiusmod dolore aliquip laborum laborum aute ipsum c
ommodo id irure duis ipsum.","modifiedDate":"2019-09-21T12:08:48 -02:00","name":"candied
 prunes","package":{"height":329,"length":179,"weight":293,"width":741},"pictures":["htt
p://placehold.it/32x32"],"price":35,"productId":"d2580eff-d105-45a5-9b21-ba61995bc6da","
```

- Get-product: Get-product is a function that is not directly called by the frontend. It is more like a helper function so that other functions can retrieve the product information easily. The function is called by the GET method and accessed directly by {url}/function/get-product/{product-id}. It will iterate over the product_list JSON file and return the product info of the product that matches the product-id in the response body. If it fails to locate, it will return a 404 - product not found. Even though it can be accessed by {url}/function/get-product/{product-id}, as we mentioned, it will only be called by other functions in the backend. Therefore, we expose the local service ip of this docker for them, and add a get_product_from_external_service function in the utils. In this function, it will access the 8080 port of PRODUCT_SERVICE_URL which is get-product.openfaas-fn.svc.cluster.localfor and append the product-id after the port. Here is an example:



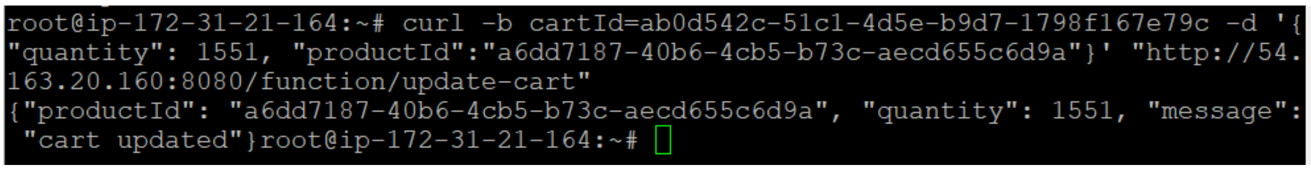
```
C:\Users\alienware>curl http://13.213.147.47:8080/function/get-product
/a6dd7187-40b6-4cb5-b73c-aecd655c6d9a -i
HTTP/1.1 200 OK
Access-Control-Allow-Headers: Content-Type
Access-Control-Allow-Methods: OPTIONS,POST,GET
Access-Control-Allow-Origin: *
Content-Length: 484
Content-Type: application/json
Date: Fri, 13 May 2022 07:26:30 GMT
Server: waitress
X-Call-Id: 500faced-00a7-4faa-b76b-9941f76f90e5
X-Duration-Seconds: 0.001358
X-Start-Time: 1652426790937331327

{"product":{"category":"fruit","createdDate":"2017-03-17T03:06:53 -01:
00","description":"Reprehenderit aliquip consequat quis excepteur et e
t esse exercitation adipisicing dolore nulla consequat.","modifiedDate
":"2019-11-25T12:32:49 -01:00","name":"fresh prunes","package":{"heigh
t":736,"length":567,"weight":41,"width":487},"pictures":["http://place
hold.it/32x32"],"price":2,"productId":"a6dd7187-40b6-4cb5-b73c-aecd655
c6d9a","tags":["nisi","quis","sint","adipisicing","pariatur"]}}
```

- List-cart: The list cart function will first pass the header of request to get_cart_id. In get_cart_id, it will check if there is cartID in the cookie. If so, it will return the value of cartID and return a boolean value with False. Otherwise, it will generate a random id with uuid package and return True for generated. When the function receives a True for the generated, it means that this is a new device and has not added anything to its shopping cart yet. Therefore, it will just make an empty product_list. When generated is False, it means that the device visited our website and might have something in its cart. The function will then send a query to the MongoDB with the cartID, and only retrieve the columns of sk for product_id, quantity, and product detail. That information will be stored in the product_list, and in the end, it will return a statusCode of 200, a header which contains the cartID and max-age of 1 day, and a body with product_list.



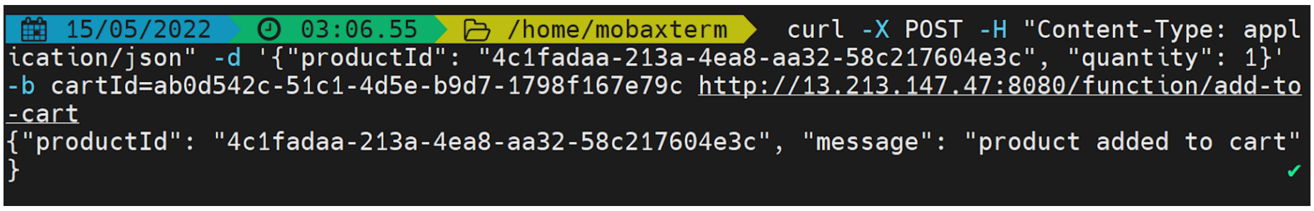
```
C:\Users\alienware>curl -b cartId=ab0d542c-51c1-4d5e-b9d7-1798f167e79c -i "http://13.213.147.47
:8080/function/list-cart"
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: True
Access-Control-Allow-Headers: Content-Type, x-requested-with
Access-Control-Allow-Methods: OPTIONS,POST,GET
Access-Control-Allow-Origin: *
Content-Length: 1745
Content-Type: text/html; charset=utf-8
Date: Fri, 13 May 2022 07:42:19 GMT
Server: waitress
Set-Cookie: cartId=ab0d542c-51c1-4d5e-b9d7-1798f167e79c; HttpOnly; Max-Age=86400; Path=/
X-Call-Id: f0e2ad77-fcb5-47f4-a5a8-8f75812b4a51
X-Duration-Seconds: 0.003022
X-Start-Time: 1652427739826610465

{"products": [{"_id": "627d32a8fd86903d8002c75e", "sk": "a6dd7187-40b6-4cb5-b73c-aecd655c6d9a",
 "productDetail": {"category": "fruit", "createdDate": "2017-03-17T03:06:53 -01:00", "descripti
on": "Reprehenderit aliquip consequat quis excepteur et et esse exercitation adipisicing dolore
 nulla consequat.", "modifiedDate": "2019-11-25T12:32:49 -01:00", "name": "fresh prunes", "pack
age": {"height": 736, "length": 567, "weight": 41, "width": 487}, "pictures": ["http://placehol
d.it/32x32"], "price": 2, "productId": "a6dd7187-40b6-4cb5-b73c-aecd655c6d9a", "tags": ["nisi",
 "quis", "sint", "adipisicing", "pariatur"]}, "quantity": 1}, {"_id": "627d32aafd86903d8002c763
", "sk": "12929eb9-3eb7-4217-99e4-1a39c39217b6", "productDetail": {"category": "vegetable", "cr
eatedDate": "2017-01-07T05:28:03 -01:00", "description": "Ad eiusmod cupidatat duis dolor molli
t labore mollit eu.", "modifiedDate": "2019-04-03T10:36:25 -02:00", "name": "fresh lettuce", "p
ackage": {"height": 813, "length": 932, "weight": 457, "width": 436}, "pictures": ["http://plac
ehold.it/32x32"], "price": 51, "productId": "12929eb9-3eb7-4217-99e4-1a39c39217b6", "tags": ["a
d", "ipsum", "est", "eiusmod", "duis"]}, "quantity": 2}, {"_id": "627d32acfd86903d8002c76a", "s
k": "8c843a54-27d7-477c-81b3-c21db12ed1c9", "productDetail": {"category": "sweet", "createdDate
": "2017-11-24T04:01:33 -01:00", "description": "Fugiat sunt in eu eu occaecat.", "modifiedDate
": "2019-05-19T05:53:56 -02:00", "name": "half-eaten cake", "package": {"height": 337, "length"
: 375, "weight": 336, "width": 1}, "pictures": ["http://placehold.it/32x32"], "price": 322, "pr
oductId": "8c843a54-27d7-477c-81b3-c21db12ed1c9", "tags": ["officia", "proident", "officia", "c
ommodo", "nisi"]}, "quantity": 2}]}
```

- Update-cart: The update cart function is called when the number of an item is directly typed in the input box. cartId will be carried in the header, and productId and quantity will be contained in the body of the request. It will first check if the productId is valid. If not, it will return a 404 status code with the message “product not found”. Then it will check if the quantity is negative, and if it is, it will return a 400 status code with the message “Quantity must not be lower than 0”. After all the conditions are met, it will send a query to update the MongoDB with key pk: cartId and sk: productId. It will set the quantity and a generated timestamp. If the key does not exist, it will insert instead. If everything is correct, it will return a 200 status code with proper head and body. (The following picture only shows the body)



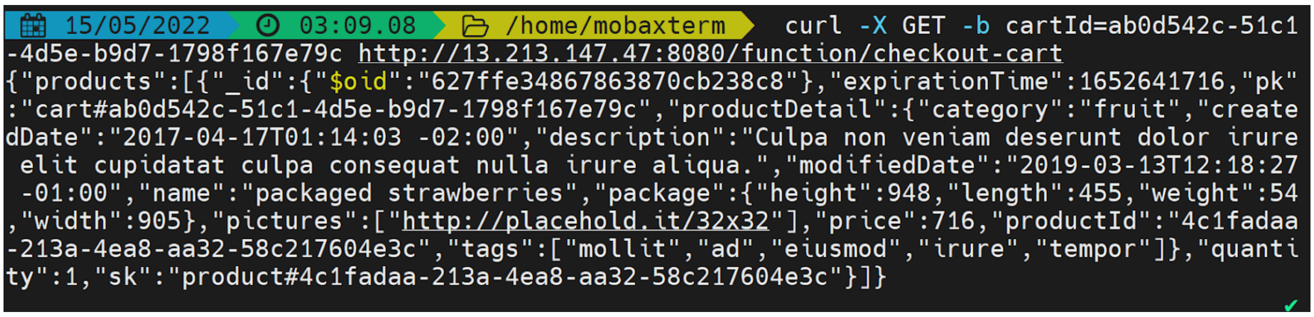
```
root@ip-172-31-21-164:~# curl -b cartId=ab0d542c-51c1-4d5e-b9d7-1798f167e79c -d '{
"quantity": 1551, "productId":"a6dd7187-40b6-4cb5-b73c-aecd655c6d9a"}' "http://54.
163.20.160:8080/function/update-cart"
{"productId": "a6dd7187-40b6-4cb5-b73c-aecd655c6d9a", "quantity": 1551, "message":
 "cart updated"}root@ip-172-31-21-164:~#
```

- add-to-cart: The add-to-cart function is called when the plus or minus button in the frontend is pressed for any product. The cart ID is fetched from cookie header. If cart ID is not set, a new uuid is generated and set on the cookie for the cart ID. When the plus button is pressed, a request to add-to-cart is sent to the backend serverless function add-to-cart, whose payload includes the product ID of the product added to cart and the quantity of the product in the cart will be increased by 1. If the minus button is pressed, the quantity will be decreased by 1. If the product is not found, a 404 status code with the message “product not found” is returned. Else, the product ID with a message “product added to cart” is returned.



```
 15/05/2022  03:06.55  /home/mobaxterm  curl -X POST -H "Content-Type: appl
ication/json" -d '{"productId": "4c1fadaa-213a-4ea8-aa32-58c217604e3c", "quantity": 1}'
-b cartId=ab0d542c-51c1-4d5e-b9d7-1798f167e79c http://13.213.147.47:8080/function/add-to
-cart
{"productId": "4c1fadaa-213a-4ea8-aa32-58c217604e3c", "message": "product added to cart"
}
                                                                                        ✓
```

- checkout-cart: The checkout-cart function is called when the checkout button is pressed in the frontend and the credit card information is input. The cart ID is fetched from cookie header. If no cart ID is defined, a new uuid is assigned as the cart ID. The function is a placeholder since the backend services do not include the actual payment system. So the function basically just deletes all the products in the cart and returns the list of products in the cart for reference. The function mimics the effect of cart checkout that the shopping cart returns to an empty state.



```
 15/05/2022  03:09.08  /home/mobaxterm  curl -X GET -b cartId=ab0d542c-51c1
-4d5e-b9d7-1798f167e79c http://13.213.147.47:8080/function/checkout-cart
{"products":[{"_id":{"$oid":"627ffe34867863870cb238c8"},"expirationTime":1652641716,"pk"
:"cart#ab0d542c-51c1-4d5e-b9d7-1798f167e79c","productDetail":{"category":"fruit","create
dDate":"2017-04-17T01:14:03 -02:00","description":"Culpa non veniam deserunt dolor irure
 elit cupidatat culpa consequat nulla irure aliqua.","modifiedDate":"2019-03-13T12:18:27
 -01:00","name":"packaged strawberries","package":{"height":948,"length":455,"weight":54
,"width":905},"pictures":["http://placehold.it/32x32"],"price":716,"productId":"4c1fadaa
-213a-4ea8-aa32-58c217604e3c","tags":["mollit","ad","eiusmod","irure","tempor"]},"quanti
ty":1,"sk":"product#4c1fadaa-213a-4ea8-aa32-58c217604e3c"}]}
                                                                                        ✓
```

4. Database

4.1 Alternatives Selection

In this section, I'll show how I replaced AWS's native formation DynamoDB with the open-source database. Before replacing this component, we need to know the basic information about this database first. According to the official DynamoDB tutorial:

- Amazon DynamoDB is a fully managed, serverless NoSQL key-value database designed to run high-performance applications of any size.DynamoDB offers built-in security, continuous backup, automatic multi-region replication, in-memory caching, and data export tools.

Therefore, based on the above description, the alternative we are looking for should also be a NoSQL type database and be able to implement most of the features of DynamoDB. Finally, we chose MongoDB.

- MongoDB is written in C++ and is an open-source database system based on distributed file storage. MongoDB is designed to provide a scalable, high-performance data storage solution for WEB applications. MongoDB stores data as a document, with a data structure consisting of key-value (key=>value) pairs.MongoDB documents are similar to JSON objects. Field values can contain other documents, arrays and document arrays.

4.2 Docker Image

After deciding to use MongoDB, the next step is to create the corresponding docker image. At the same time, the database should be initialized when creating the image so that users can use the MongoDB service directly when running the container after instantiating the image.
The dockerfile I wrote to create the MongoDB service is as follows. The logic is very simple, first, pull the official MongoDB image from docker hub, then create the root user for the database, and finally map a database initialization script to the docker volume so it can be called when initializing the database.

```
FROM mongo

ENV MONGO_INITDB_ROOT_USERNAME admin-user
ENV MONGO_INITDB_ROOT_PASSWORD admin-password
ENV MONGO_INITDB_DATABASE admin

ADD ./docker-entrypoint-initdb.d/mongo-init.js /docker-entrypoint-initdb.d/mongo-init.js
```

The initialization script is shown as follows. This script will create a new user and database "cart" during database initialization, then insert a test data into it, and automatically create the item collection when inserting the test data.

```
db.auth('admin-user', 'admin-password')
db.createUser({
    user: 'csit6000o',
    pwd: 'csit6000o',
    roles: ["root"],
});

db = db.getSiblingDB("cart")

db.items.insert({
    pk: "user@test",
    sk: "product@test",
    quantity: 100,
    expirationTime: 10,
    productDetail: {
        name: "xxx",
        price: 29
    }
})
```

4.3 Python API Change

The last thing we need to do is to replace the python API. MongoDB provides a python package so that we can easily use python to operate the database. We can use the following code to connect to MongoDB and then perform add, delete, update and query operations on the database.

```python
from pymongo import MongoClient
# connect to the database
client = MongoClient(host='xx.xx.xx.xx', port=27017,
                     username='csit6000o',
                     password='csit6000o')
# Select the collection you want to operate
mydb = client["cart"]
mycol = mydb["items"]

# Query Data from collection
for x in mycol.find():
    print(x)

# insert data
mydict= { "name": "john", "description": "xxx" }
x = mycol.insert_one(mydict)

# update data
myquery = { "name": "john" }
newvalues = { "$set": { "description": "12345" } }
mycol.update_one(myquery, newvalues)

# delete data
myquery = { "name": "RUNOOB" }
mycol.delete_one(myquery)
```

We just need to replace the previous DynamoDB API with the new one.

5. Frontend Service

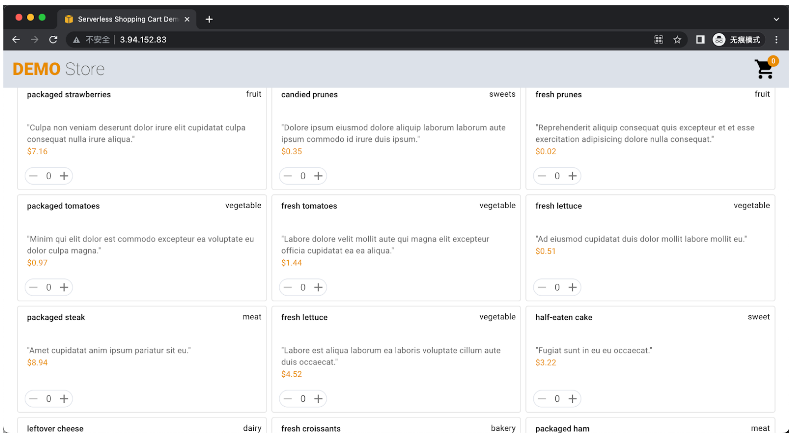
The frontend of the sample project is developed based on Vue Framework. Moreover, the sample project uses the API (AWS API Gateway) and the Auth (AWS Cognito) components in aws-amplify to manage APIs and authentication. We plan to take the original Vue Framework and replace the original API components with our own APIs that work with our backend service. After replacement, we built the frontend into a container and finally deployed it to Kubernetes as a service. The details about how we made our frontend service on Kubernetes are as follows.

- First, we redefined the APIs and deprecated AWS components. Because our backend is built with OpenFaas, whose format of the APIs is different from the AWS API gateway, the API format defined in the frontend needs to be modified. We also deprecated some AWS Cognito-related auth components.
- Secondly, we debugged the frontend framework locally to solve the issues caused by changes in API definitions and AWS dependency. Due to API changes and deprecation, the logic of some button actions needs to be changed, and the request and response of APIs also need to be modified, so that the data can be obtained and displayed correctly.
- Thirdly, we made a Dockerfile to dockerize the framework into a container and

pushed it to the docker hub. We mainly used npm to install the dependency and deploy a server in the container. Lastly, we deployed the docker image as the frontend service on Kubernetes. We made a YAML file to config the deployment of the frontend service.

A demo running on an instance is shown in the following.

- Main Page:



packaged strawberries fruit
"Culpa non veniam deserunt dolor irure elit cupidatat culpa consequat nulla irure aliqua."
$7.16

candied prunes sweets
"Dolore ipsum eiusmod dolore aliquip laborum laborum aute ipsum commodo id irure duis ipsum."
$0.35

fresh prunes fruit
"Reprehenderit aliquip consequat quis excepteur et et esse exercitation adipisicing dolore nulla consequat."
$0.02

packaged tomatoes vegetable
"Minim qui elit dolor est commodo excepteur ea voluptate eu dolor culpa magna."
$0.97

fresh tomatoes vegetable
"Labore dolore velit mollit aute qui magna elit excepteur officia cupidatat ea ea aliqua."
$1.44

fresh lettuce vegetable
"Ad eiusmod cupidatat duis dolor mollit labore mollit eu."
$0.51

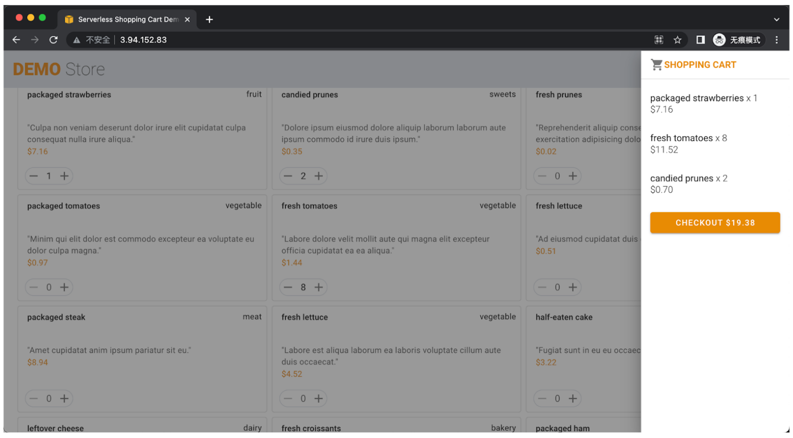
packaged steak meat
"Amet cupidatat anim ipsum pariatur sit eu."
$8.94

fresh lettuce vegetable
"Labore est aliqua laborum ea laboris voluptate cillum aute duis occaecat."
$4.52

half-eaten cake sweet
"Fugiat sunt in eu eu occaecat."
$3.22

leftover cheese dairy
fresh croissants bakery
packaged ham meat

- Shopping Cart Page:



SHOPPING CART

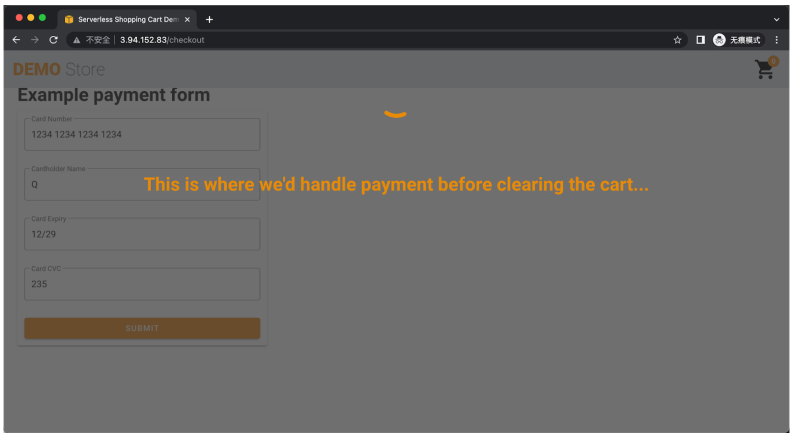
packaged strawberries x 1
$7.16

fresh tomatoes x 8
$11.52

candied prunes x 2
$0.70

CHECKOUT $19.38

- Payment Page:



Example payment form

Card Number: 1234 1234 1234 1234
Cardholder Name: Q
Card Expiry: 12/29
Card CVC: 235

SUBMIT

This is where we'd handle payment before clearing the cart...

Reference

1. OpenFass Docs https://docs.openfaas.com/
2. Sample Project https://github.com/aws-samples/aws-serverless-shopping-cart
3. MongoDB Documentation https://www.mongodb.com/docs/
4. Vue.js Cookbook https://v2.vuejs.org/v2/cookbook/dockerize-vuejs-app.html
5. Serverless with OpenFaas, Kubernetes, and Python https://medium.com/analytics-vidhya/serverless-with-openfaas-kubernetes-and-python-9934e4a80de5
6. Deploy your Serverless Python function locally with OpenFaas in Kubernetes https://yankee.dev/serverless-function-openfaas-kubernetes-locally