Front-End Development for the MPC in the Cloud Sprint-5

QingXing Li, Yicun Hou, Haoyu Xu
Mentors: Joseph Caiani, Máirín Duffy

Project Overview

- An open source framework that utilizes cloud technologies to democratize medical analytics application development and enables healthcare organizations to keep owning their data while benefiting from public cloud processing capabilities.
- **Goal:** Design and test the front-end functions using TypeScript and deploy the ChRIS Store UI to the MOC
- **Stretch Goal:** Deploy the ChRIS Store backend to MOC and implement a tool to track the website traffic

Previous sprints

Sprint1: Set up the ChRIS Store backend in Docker and rewrited ChRIS Store UI with Redux

Sprint2: Write the declaration file of all the APIs by using Typescript, and installed the backend of the Chris UI

Sprint3: Debug the declaration file of all the APIs; Test and Debug Actions & Reducer of all the APIs by using Jest

Sprint4: Debugged and optimized unit testing code and deployed the ChRIS Store UI(MVP)

Sprint 5- What we did

- Track Chris Store traffic and test performance using Apache JMeter
- Tested and tried to deploy the ChRIS Store backend
  - **Main task:** Deploy to MOC using Openshift

Apache JMeter

Apache JMeter is a open source software to designed to load test functional behavior and measure performance.

- Test Static and dynamic resources
- Test Web dynamic applications
- Simulate a heavy load to analyze website performance under different load types

Jmeter test

Test case:

Threads: 2000 Loops: 10 Ramp-up Period: 0

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
chris store test	20000	321	12	2551	610.10	18.15%	789.2/sec	1309.71	107.24	1699.4
TOTAL	20000	321	12	2551	610.10	18.15%	789.2/sec	1309.71	107.24	1699.4

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
chris store test	60000	676	0	36014	1746.73	17.14%	245.8/sec	395.80	24.66	1648.8
TOTAL	60000	676	0	36014	1746.73	17.14%	245.8/sec	395.80	24.66	1648.8

Threads: 100 Loops: 200 Ramp-up Period: 0

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
chris store test	20000	195	11	35969	2441.98	0.00%	471.0/sec	705.05	78.19	1533.0
TOTAL	20000	195	11	35969	2441.98	0.00%	471.0/sec	705.05	78.19	1533.0

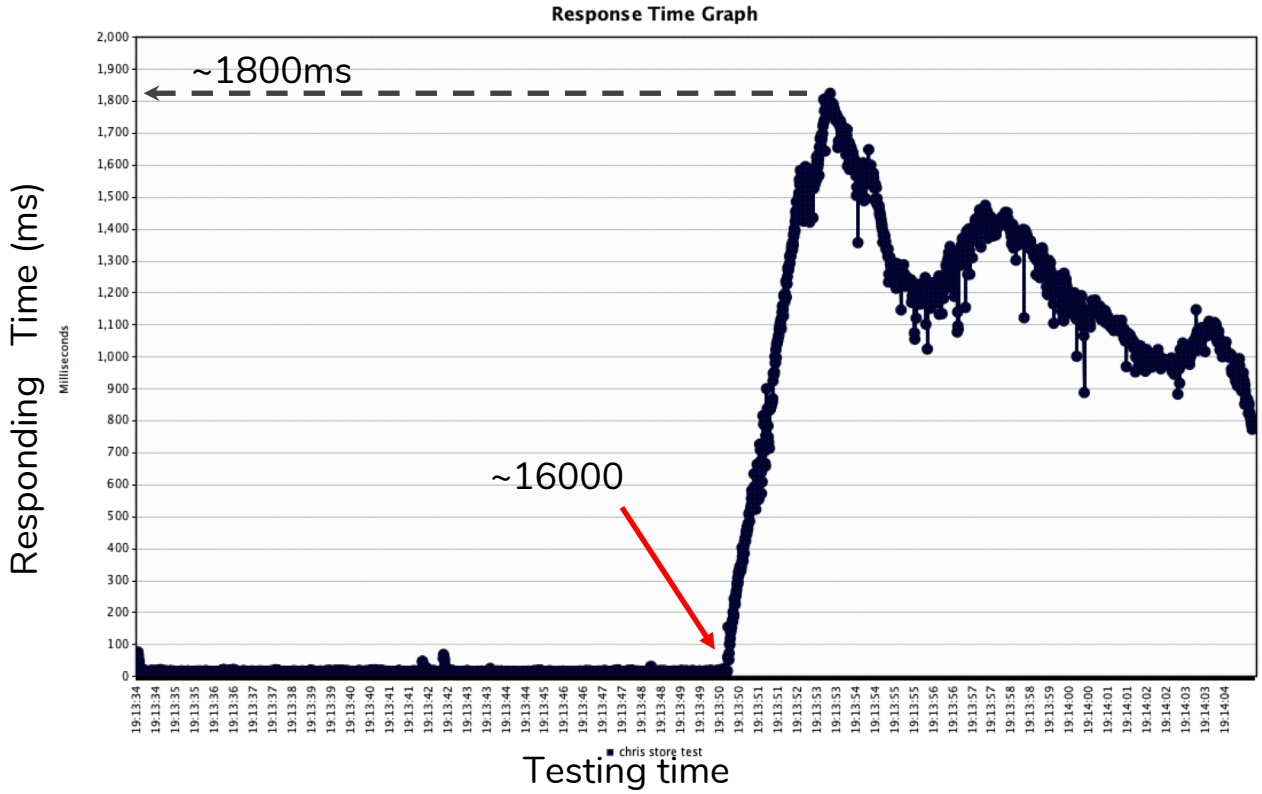
Jmeter Test

Test case:

Threads: 2000

Loops: 10

Error rate: 18.18%



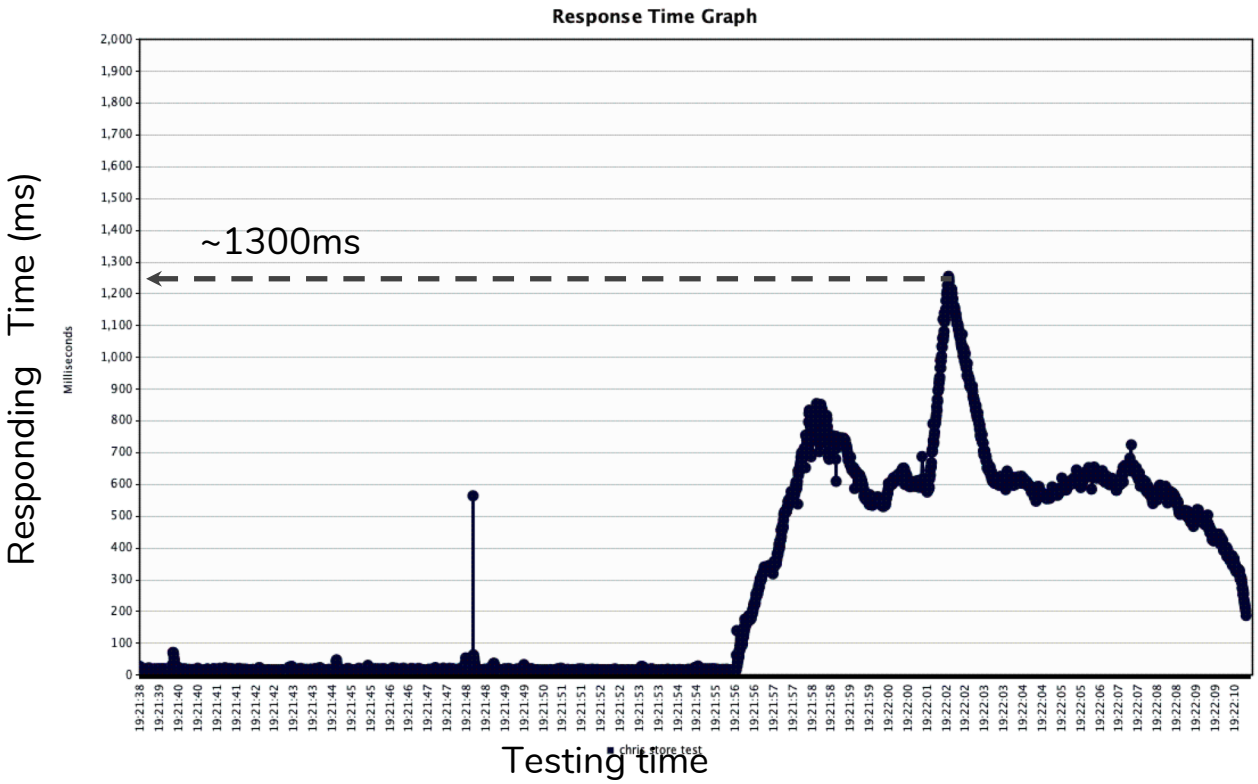
Jmeter Test

Test case:

Threads: 1000

Loops: 20

Error rate: 12.94%



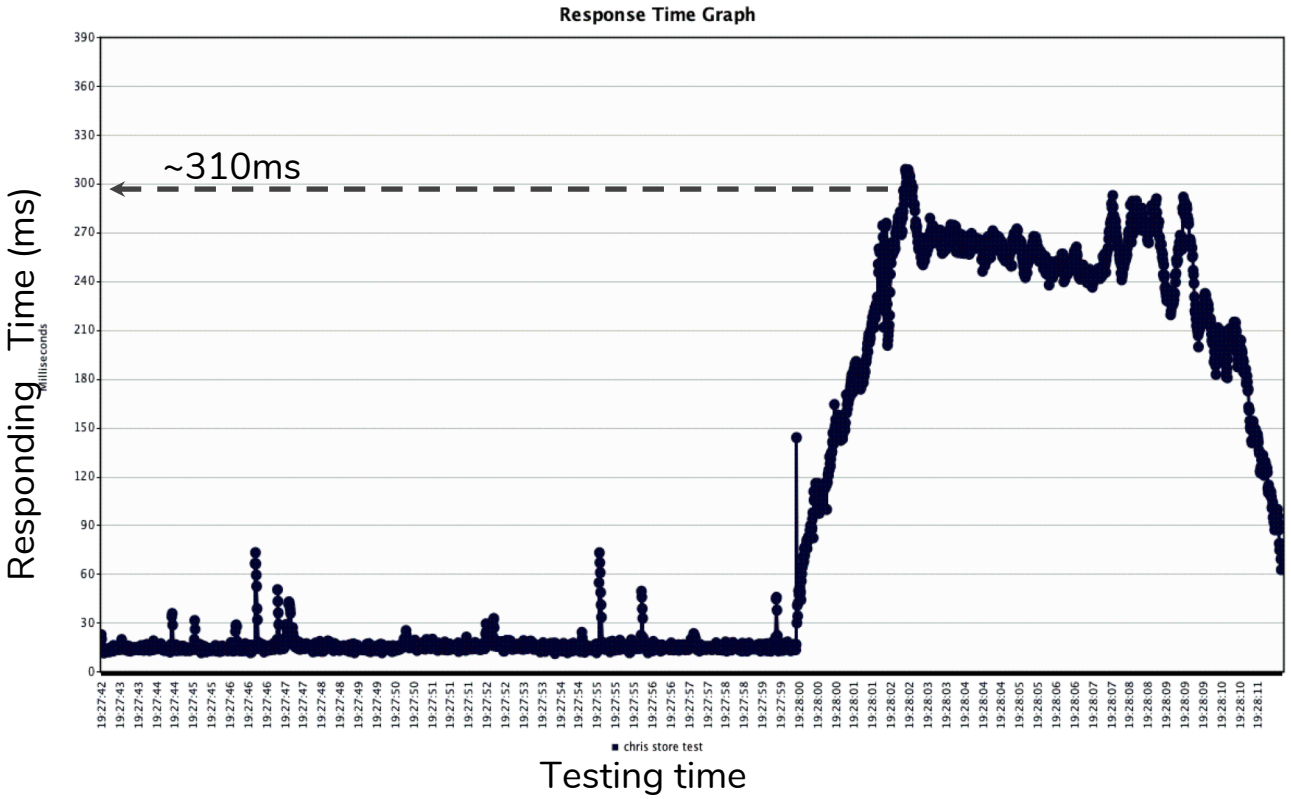
Jmeter Test

Test case:

Threads: 500

Loops: 40

Error rate: 9.94%



Openshift requirement

- Python 3.6
- Requirement.txt : a file contains the information of requirement and dependencies needed

```
Django==2.1.4
django-filter==2.0.0
djangorestframework==3.9.0
django-cors-middleware==1.3.1
mysqlclient==1.3.14
python-swiftclient==3.6.0
django-storage-swift==1.2.19
```

- Manage.py: where to start
- Wsgi.py: config of the protocol between application and server

Deployment - step 1

——Defined the necessary services in the docker-compose.yml file

chris_store_dev

chris_store_dev_db

swift _service

```yaml
services:
  chris_store_dev:
    image:  ${CREPO}/chris_store:dev
    volumes:
      - ./store_backend:/usr/src/store_backend
    ports:
      - "8010:8010"
    depends_on:
      - chris_store_dev_db
      - swift_service
    labels:
      name: "ChRIS_store"
      role: "Development server"

  chris_store_dev_db: ...

  swift_service: ...
```

Deployment - step 2

——Tested the services of the backend in localhost

```
ethanhou 528backend
$ docker-compose ps
         Name                      Command            State            Ports
-------------------------------------------------------------------------------
528backend_chris_store_   docker-entrypoint.sh     Exit 255   3306/tcp, 33060/tcp
dev_db_1                  mysqld
528backend_chrisstore_1   python3 manage.py        Exit 255   0.0.0.0:8010->8010/tcp
                          runserve ...
528backend_swift_servic   /bin/sh -c               Exit 255   0.0.0.0:8080->8080/tcp
e_1                       /usr/local/bin/ ...
ethanhou 528backend
$ docker-compose up -d
```

```
$ docker-compose ps
          Name                      Command            State           Ports
-------------------------------------------------------------------------------
528backend_chris_store_d   docker-entrypoint.sh       Up      3306/tcp, 33060/tcp
ev_db_1                    mysqld
528backend_chrisstore_1    python3 manage.py          Up      0.0.0.0:8010->8010/tcp
                           runserve ...
528backend_swift_service   /bin/sh -c                 Up      0.0.0.0:8080->8080/tcp
_1                         /usr/local/bin/ ...
```

Deployment - step 2

localhost:8010/api/v1/

Apps | Shopping | FIT | Career | BU | Twitter Proj. | EC504 Foswiki | Stack Overflow -... | EC528 Cloud Com... | 百度一下，你就知道

Django REST framework

Log in

Plugin List

Plugin List

OPTIONS GET

A view for the collection of plugins.

GET /api/v1/

localhost:8010/api/v1/

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/vnd.collection+json
Vary: Accept
```


```
{
    "collection": {
        "version": "1.0",
        "href": "http://localhost:8010/api/v1/",
        "items": [],
        "links": [],
        "queries": [
            {
                "href": "http://localhost:8010/api/v1/search/",
                "rel": "search",
                "data": [
                    {
                        "name": "id",
                        "value": ""
                    },
                    {
                        "name": "name",
                        "value": ""
                    },
                    {
```

Deployment - step 3

——Translate the docker-compose.yml file to several yaml files which Openshift can recognized using **Kompose**

Kompose is a conversion tool for Docker Compose to container orchestrators such as Kubernetes (or OpenShift).

- Simplify your development process with docker compose and then deploy your containers to a production cluster
- Easily convert docker-compose.yml to OpenShift deployments in one simple command

```
kompose --provider openshift --file docker-compose.yml convert
```

Deployment - step 3

- The yaml file defines all the services ChRIS Store backend needed:
  - `chris_store_dev`
  - `chris_store_dev_db`
  - `Swift_service`
- All the services were deployed to the Openshift

Deployments

Name	Last Version	Replicas	Created
chris-store-dev-db	#1	1 replica	12 hours ago
swift-service	#1	1 replica	12 hours ago
chris-store-dev	#1	2 replicas	12 hours ago

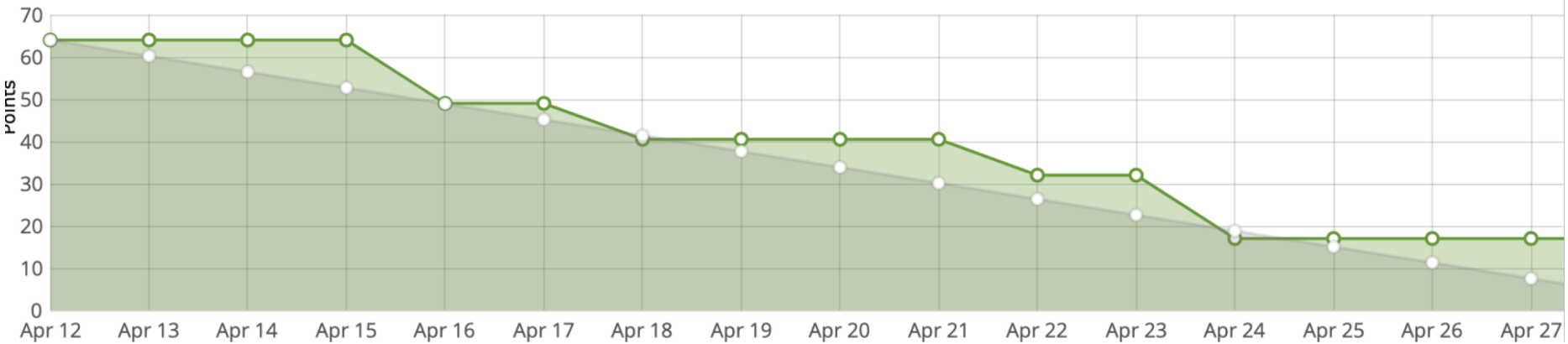
Deployment-step 4

- Push the image to OpenShift and run the application in the OpenShift cluster - challenge
- Use the frontend to point to our backend

Burn Down Chart

FRONT-END-DEVELOPMENT-FOR-MULTI-PARTY-COMPUTATION-I... SPRINT 5 - BACKEND D

47% 64 total points 30 completed points 2 open tasks 5 closed tasks 0 iocaine doses



Next to do

- Finish the step 4 of deploy the Chris Store backend to MOC using OpenShift
- Try to optimize the application stability
- Keep monitor the traffic after deploying the backend

Questions?