

Extending a Microservice Architecture

Nathan R. Hall
Benjamin Kurzyna
Ji Hyuk Shon

Ji Hyuk Shon

- About Me:
 - Pitt Student, BS in Computer Science, Graduation Date: April 2020
- Why I chose this Project:
 - Interested in learning about new technologies such as Docker. Also, very interested in the experience of agile development as well as microservice architecture implementation.

Nathan Hall

- About Me:
 - Pitt Student, BS in Computer Science, Graduation Date: August 2020
- Why I chose this Project:
 - Research and learn how to work with a microservice architecture as a whole. Get practice working with cutting edge containerization technology. Ultimately gain crucial experience in how a professional style development team works.

Benjamin Kurzyna

- About Me:
 - Pitt Student, BS in Computer Science, Graduation Date: April 2020
- Why I chose this Project:
 - Very interested on gaining more experience with code testing and working in groups while using agile based software development. Working in a more developer-level professional setting is invaluable experience. Learning about the microservice architecture and working with containerization technologies was also very appealing.

Outline

- Communication
- Experience
- Project Goals
- Challenges
- Features and Functionality
- Demo
- Questions

Communication

- Constant communication with our point of contacts from NetApp
- Maintained the project through an AGILE development methodology
- Team members keep in contact with a slack workspace and weekly meetings via Zoom teleconferencing
- Using our POCs to talk through bugs, challenges, etc. in our code has been a key to consistent development for this project

Project Experience

- The communication, research, development, and maintenance has been a positive experience thus far
- Project has been successfully maintained through the use of sprints, a kanban board, and git version control
- Spending time to become familiar with code we inherited as well as with microservice architecture and pytest framework

Project Goals

- Research and learn how a microservice architecture works
- Figure out ways to extend the project that was given to us
 - Creation of a test container
 - Creation of a containerized database server
- Optimize the test container and database for future microservices
- Gain real world experience with a hands on project overseen by professionals

Challenges

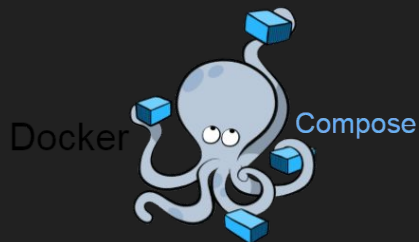
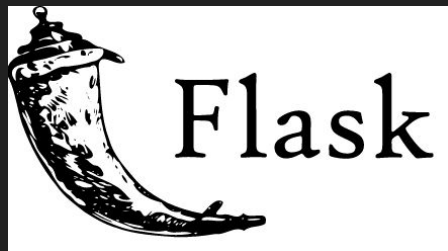
- 1. Understanding/researching how microservice architectures work with container technology
- 2. Setting up environments and running existing program on all systems (0.0.0.0 vs localhost)
- 3. Practicing proper version control techniques that align with agile
 - Maintenance of user stories
 - Issue based branches
 - Pull requests / merging into master

Challenges

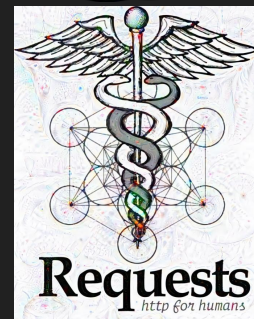
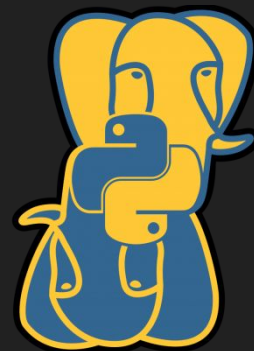
- 4. Since we inherited code, understanding the codebase and how features were implemented
 - Flask Library utilizing RESTful API techniques
 - YAML formatting
 - Pystache
- 5. Implementing the database server with a database per microservice pattern

Technologies & Frameworks

GIVEN



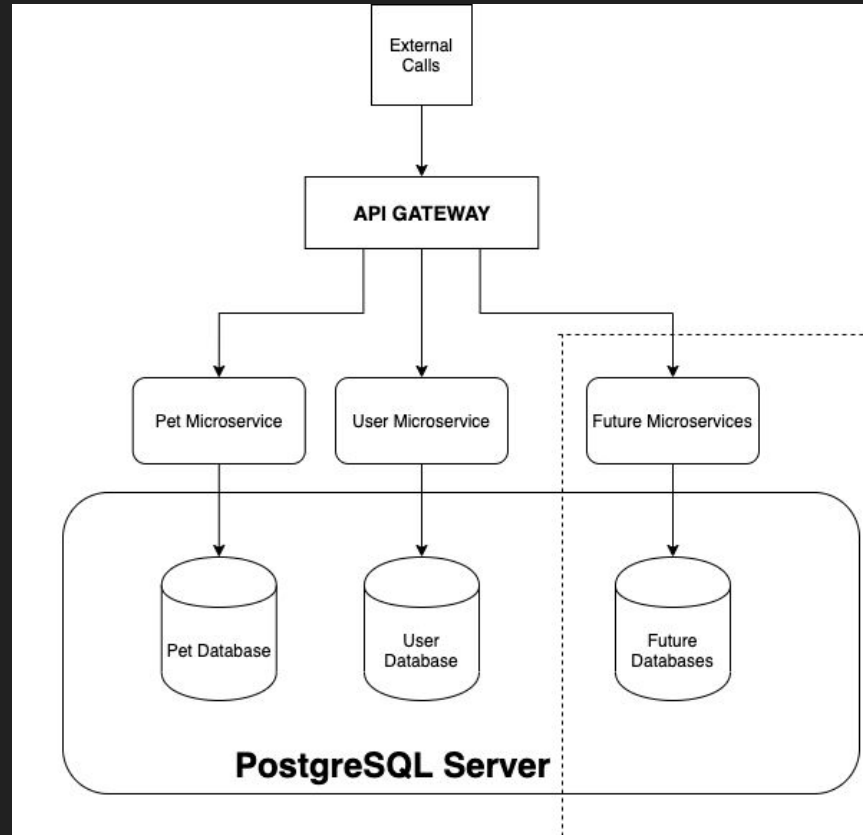
NEW



Choosing a Database Pattern

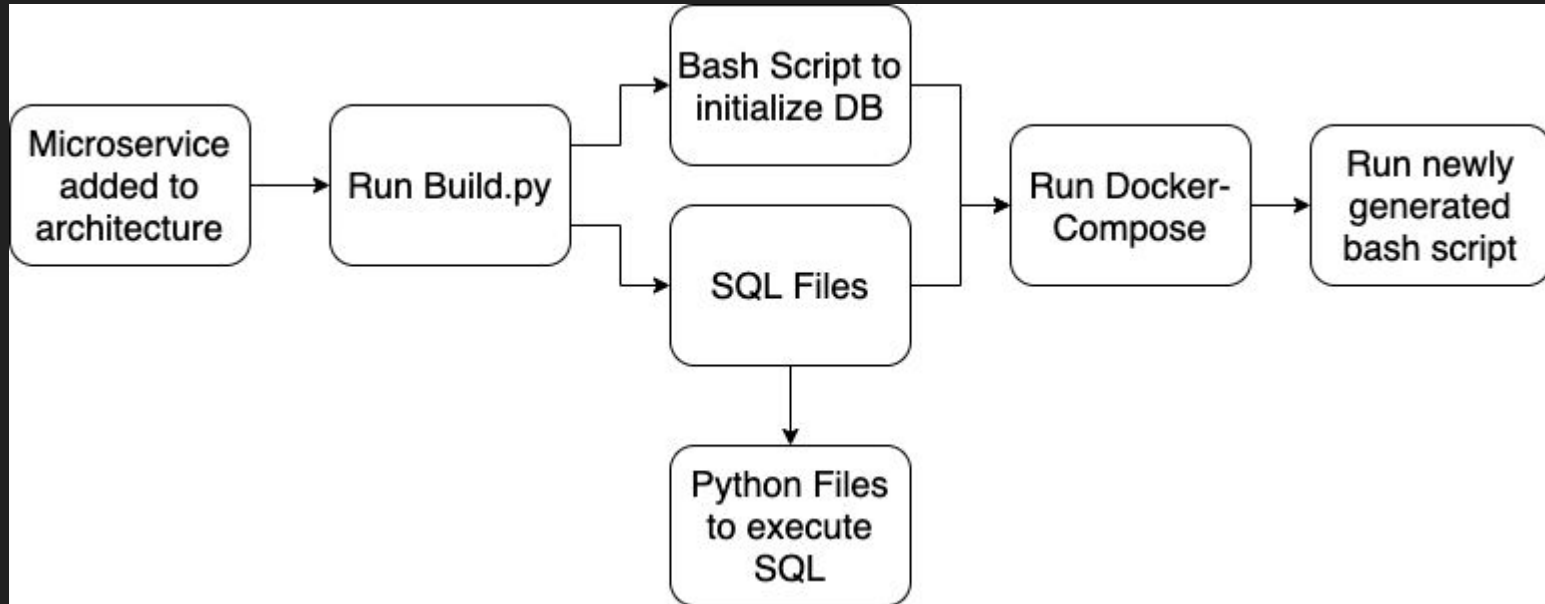
- Shared Database
 - Advantages
 - A single database is simple
 - Disadvantages
 - Increased development time
 - Possible decreases in run time
 - Not Flexible
- Database per Microservice
 - Advantages
 - Loosely coupled
 - Flexible
 - Scalable
 - Disadvantages
 - More advanced queries

Database per Microservice Pattern



----- = Future services

Implementing the Database Pattern



Bash Script to Initialize Database

```
#!/bin/bash
echo "Setting up Databases for microservices.."
echo "..."
```

##services##

```
echo "Setting up {{.}} database"
docker exec -it my_postgres psql -U postgres -c "create database {{.}}_db"
echo "..."
```

##/services##

```
echo "Initializing tables and values into Databases"
echo "..."
```

##services##

```
echo "Running {{.}} DB Script"
python {{.}}-init.py
echo "..."
```

##/services##

```
echo "Done."
```

bash.mustache

```
#!/bin/bash
echo "Setting up Databases for microservices.."
echo "..."
```

Setting up pet database

```
docker exec -it my_postgres psql -U postgres -c "create database pet_db"
echo "..."
```

Setting up user database

```
docker exec -it my_postgres psql -U postgres -c "create database user_db"
echo "..."
```

```
echo "Initializing tables and values into Databases"
echo "..."
```

Running pet DB Script

```
python pet-init.py
echo "..."
```

Running user DB Script

```
python user-init.py
echo "..."
```

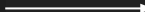
```
echo "Done."
```

Dbsetup shell script

Python File to Execute SQL File

```
conn = psycopg2.connect(  
    host='localhost',  
    port=54320,  
    dbname='{{val}}_db',  
    user='postgres',  
    password='postgres',  
)  
conn.autocommit = True  
  
cur = conn.cursor()  
  
executeScriptsFromFile("SQLFiles/{{val}}-init.sql")  
cur.execute("Select * from {{val}}Table")  
x = cur.fetchall()  
print(x)  
  
cur.close()  
conn.close()
```

pysql.mustache



```
conn = psycopg2.connect(  
    host='localhost',  
    port=54320,  
    dbname='user_db',  
    user='postgres',  
    password='postgres',  
)  
conn.autocommit = True  
  
cur = conn.cursor()  
  
executeScriptsFromFile("SQLFiles/user-init.sql")  
cur.execute("Select * from userTable")  
x = cur.fetchall()  
print(x)  
  
cur.close()  
conn.close()
```

user-init.py

```
conn = psycopg2.connect(  
    host='localhost',  
    port=54320,  
    dbname='pet_db',  
    user='postgres',  
    password='postgres',  
)  
conn.autocommit = True  
  
cur = conn.cursor()  
  
executeScriptsFromFile("SQLFiles/pet-init.sql")  
cur.execute("Select * from petTable")  
x = cur.fetchall()  
print(x)  
  
cur.close()  
conn.close()
```

pet-init.py

SQL Files Created

```
create table if not exists {{service}}Table
(
    {{#attributes}}
    {{key}} {{type}} {{pk}}{{comma}}
    {{/attributes}}
);
```

sql.mustache

```
create table if not exists petTable
(
    id integer PRIMARY KEY,
    name varchar(20) ,
    species varchar(20) ,
    status varchar(20) ,
    subspecies varchar(20)
);
```

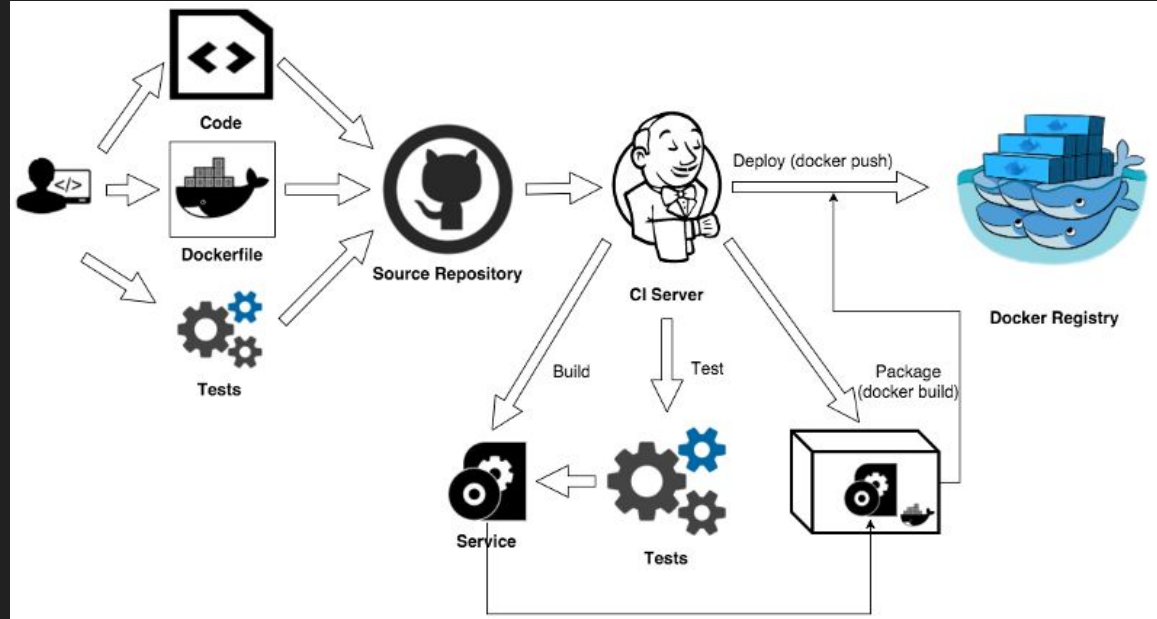
pet-init.sql

```
create table if not exists userTable
(
    id integer PRIMARY KEY,
    name varchar(20) ,
    role varchar(20)
);
```

user-init.sql

Testing Idea

Create a test container which contains a test suite that automatically generates test cases to test all microservices.



Testing (3-step test container approach)

1. Make sure that the containers are all up and running.
2. Make sure that the API calls are successfully reaching the containers
3. Make sure that each microservice does what it's supposed to do (CRUD).

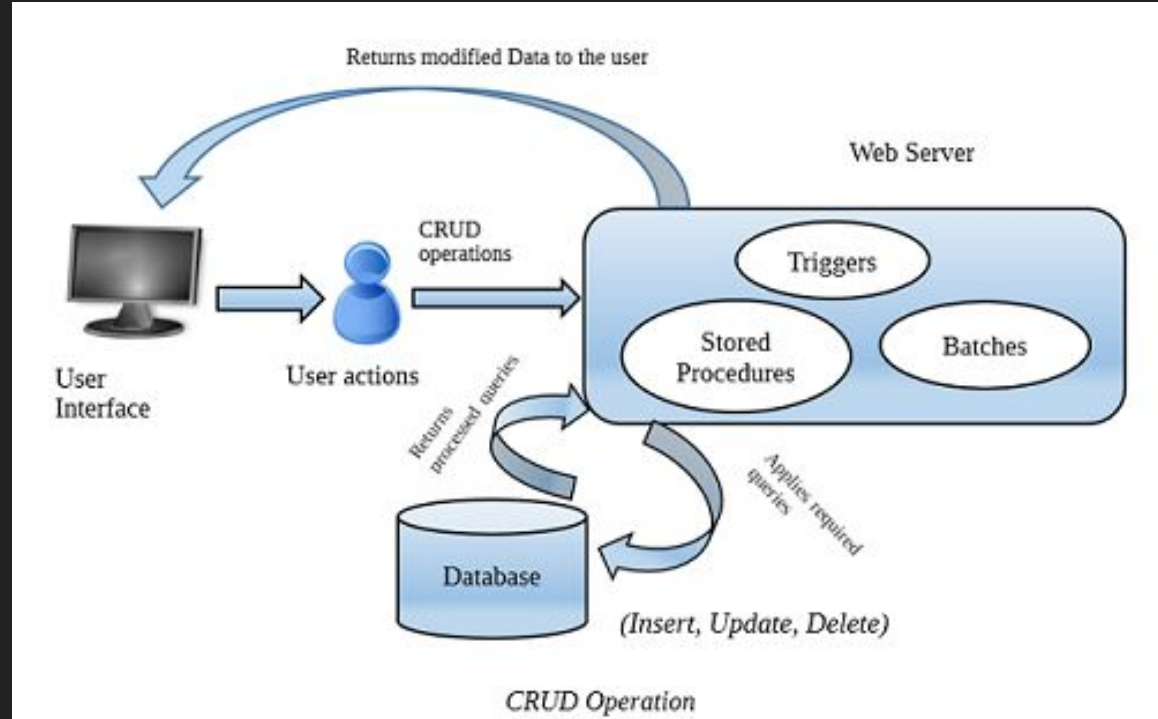
CRUD TESTING

C - Create

R - Read

U - Update

D - Delete



```
Project: CS1980-Capstone-Spring-2020
api_gateway_microservice
  config
  database
  microservices
  api_gateway

test.py
112 url = "http://0.0.0.0:8080/"
113 # Send a GET request
114 response = requests.delete(url)
115 print(response.json())
116 assert response.status_code == 200

test_delete_user()

Terminal: Local Local (2) +
api_gateway_1 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
pet_1 | * Debugger is active!
pet_1 | * Debugger PIN: 585-352-857
user_1 | * Debugger is active!
user_1 | * Debugger PIN: 221-397-171
test_1 | collected 11 items
test_1 |
ui_1 | 172.16.238.7 -- [11/Mar/2020 17:49:39] "GET /ui HTTP/1.1" 200 -
api_gateway_1 | 172.16.238.6 -- [11/Mar/2020 17:49:39] "GET /ui HTTP/1.1" 200 -
test_1 | test.py <Response [200]>
pet_1 | 172.16.238.7 -- [11/Mar/2020 17:49:39] "GET /pet HTTP/1.1" 200 -
api_gateway_1 | 172.16.238.6 -- [11/Mar/2020 17:49:39] "GET /pet HTTP/1.1" 200 -
test_1 | .[]
pet_1 | 172.16.238.7 -- [11/Mar/2020 17:49:39] "POST /pet HTTP/1.1" 201 -
api_gateway_1 | 172.16.238.6 -- [11/Mar/2020 17:49:39] "POST /pet HTTP/1.1" 200 -
test_1 | .[{'id': 0, 'name': 'gizmo', 'status': 'dog', 'species': 'available', 'subspecies': 'lab'}]
pet_1 | 172.16.238.7 -- [11/Mar/2020 17:49:39] "GET /pet/0 HTTP/1.1" 200 -
api_gateway_1 | 172.16.238.6 -- [11/Mar/2020 17:49:39] "GET /pet/0 HTTP/1.1" 200 -
test_1 | .[{'id': 0, 'name': 'gizmo', 'status': 'dog', 'species': 'available', 'subspecies': 'lab'}]
pet_1 | 172.16.238.7 -- [11/Mar/2020 17:49:39] "PATCH /pet HTTP/1.1" 201 -
api_gateway_1 | 172.16.238.6 -- [11/Mar/2020 17:49:39] "PATCH /pet HTTP/1.1" 200 -
test_1 | .[{'id': 0, 'name': 'gizmo', 'status': 'SOLD', 'species': 'dog', 'subspecies': 'lab'}]
pet_1 | 172.16.238.7 -- [11/Mar/2020 17:49:40] "DELETE /pet/0 HTTP/1.1" 200 -
api_gateway_1 | 172.16.238.6 -- [11/Mar/2020 17:49:40] "DELETE /pet/0 HTTP/1.1" 200 -
test_1 | .[{'id': 0, 'name': 'gizmo', 'status': 'SOLD', 'species': 'dog', 'subspecies': 'lab'}]
user_1 | 172.16.238.7 -- [11/Mar/2020 17:49:40] "GET /user HTTP/1.1" 200 -
api_gateway_1 | 172.16.238.6 -- [11/Mar/2020 17:49:40] "GET /user HTTP/1.1" 200 -
test_1 | .[]
user_1 | 172.16.238.7 -- [11/Mar/2020 17:49:40] "POST /user HTTP/1.1" 201 -

1 | TODO | Version Control | Terminal | Python Console | Event Log
Dockerfile detection: You may setup Docker deployment run configuration for the following file(s): api_gateway_microservice/microservices/t... (today 12:11 PM) 116-39 LF UTF-8 4 spaces Git: master Python 3.8 (Venv)
```

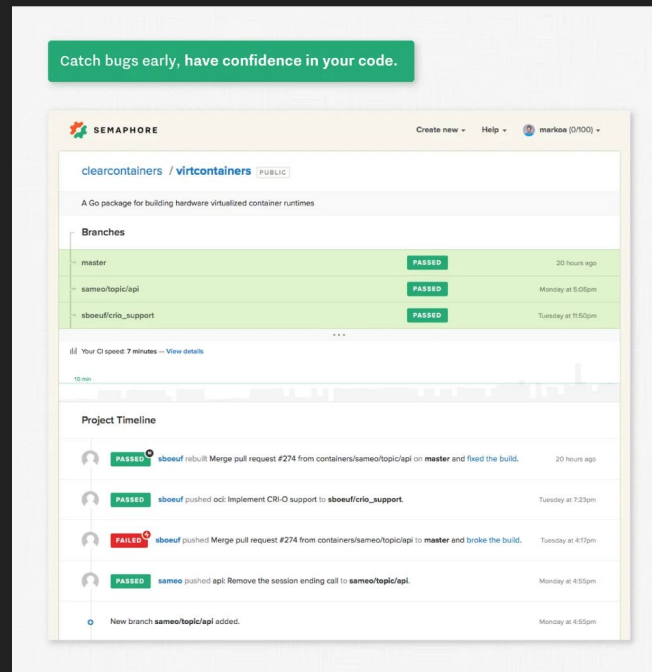
DEMO VIDEO

Future Development Goals

- Add element of Continuous Integration to automate testing
- Create templating scripts to build tests for future microservices
- Clean up swagger UI
- Create a new microservice to test the scalability of the project

Future Development Goals: CI

- Initially interested in using a Github plugin to add continuous integration
 - Automatically run test cases upon pull request
 - Embedded in github UI
 - Simple to use and easy to modify
- Plugins that we looked at:
 - Semaphore
 - Codefresh
 - Check Run Reporter
- Issues with available plugins:
 - Most required external app downloads
 - Cost \$



Future Development Goals: CI Cont.

Our decision:

1. Continue looking for free plugin that satisfies our needs
 - There are a lot of available plugins
2. Make our own solution
 - Create script to email group members
 - Can be done for pull requests or even branch commits

Appreciation Note

- Thank you to the following developers and engineers who play a crucial role in this project:
 - Twesha Mitra
 - Anuradha Kulkarni
 - Tim Banyas
- We look forward to completing the second part of this project with your help

Questions / Comments?