НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Лабораторные работы

По дисциплине «Распределенные системы»

«Реализация микросервиса»

Группа: А-05м-21

Студент: Бекбулатов Р.М.

Преподаватель: Куриленко И.Е.

Москва 2022

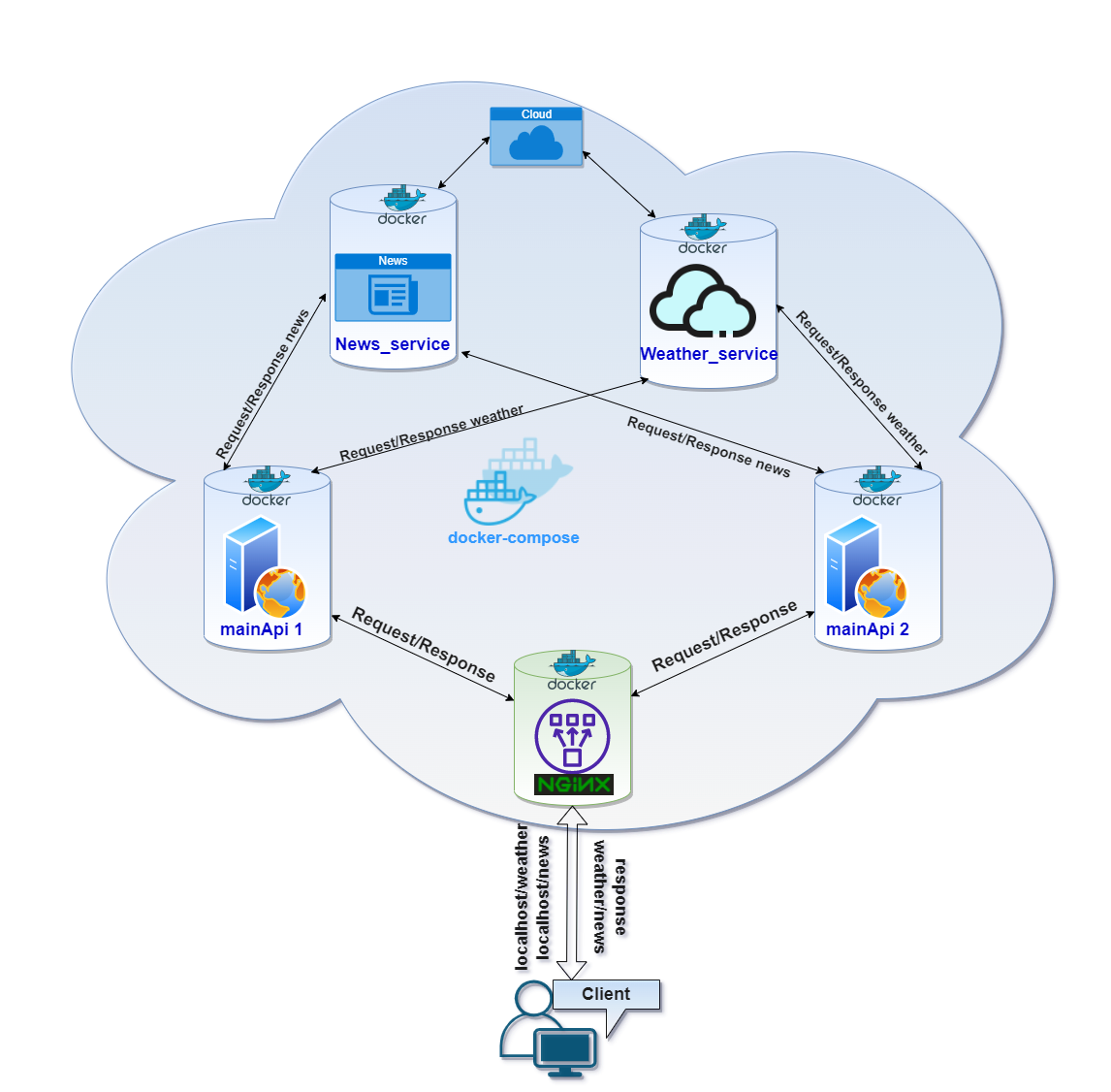
**Лабораторная работа №1 «Проектирование микросервиса»**

**Цели работы**:

Определить какие задачи будет решать микросервис. Разработать техническое описание и техническую документацию по микросервису. Определиться с используемым стеком.

**Задачи микросервиса:**

Создадим два Api-master сервера. Они будет взаимодействовать с двумя микросервисами, которые будут отвечать за информацию о новостях и погоде, которую будет запрашивать клиент. Все будет упаковано в контейнеры docker и развернуто с помощью docker-compose. Также будет использоваться Flask - многопоточность. Для получения данных погоды и новостей используем бесплатные внешние api: api.openweathermap.org и newsapi.org



**Используемый стек:**

**Python** + **Flask**

Содержание файла **requirements.txt**

aniso8601==8.1.0  
blinker==1.4  
certifi==2020.12.5  
chardet==4.0.0  
click==7.1.2  
Flask==1.1.2  
Flask-Jsonpify==1.5.0  
flask-requests==0.0.14  
Flask-RESTful==0.3.8  
Flask-SQLAlchemy==2.4.4  
idna==2.10  
itsdangerous==1.1.0  
Jinja2==2.11.2  
MarkupSafe==1.1.1  
pampy==0.3.0  
pytz==2020.4  
requests==2.25.1  
six==1.15.0  
SQLAlchemy==1.3.20  
urllib3==1.26.3  
Werkzeug==1.0.1

**Лабораторная работа №2 «Разработка микросервиса»**

**Цели работы**: реализовать микросервис.

**Описание основных файлов:**

Файл **master\_assistant.py** – сервера, взаимодействующие с сервисами wether и news. Разбалансированы с помощью nginx.

Файл **weather.py/news.py** – сервисы, запрашивают данные с внешних api

В master\_assistant у нас есть две конечные точки, одна для запроса погоды и одна для запроса новостей. Каждый раз master\_assistant будет решать, из какого микросервиса запрашивать соответствующую информацию.

В каждой службе в нижней части кода мы определяем, на каком порту она будет прослушиваться. Включаем многопоточность сервера Flask.

master\_assistant.py

from flask import Flask, request  
from flask\_restful import Resource, Api, reqparse, jsonify  
import json  
import os  
import requests  
  
app = Flask(\_\_name\_\_)  
api = Api(app)  
  
@app.route('/weather')  
def weather():  
 city = request.args.get('city')  
 if city.isdigit():  
 res = "City name must be string e.g. 'Amsterdam, Berlin'"  
 return res  
 response = requests.get("http://weather:3002/weather?city="+ city)  
 return response.json()  
  
@app.route('/news')  
def news():  
 country\_name = request.args.get('country')  
 if country\_name.isdigit() or len(country\_name) > 2 :  
 resp = "Country name must be string. Choose from below: \n\nThe 2-letter ISO 3166-1 code of the country you want to get headlines for.\nPossible options: ae ar at au be bg br ca ch cn co cu cz de eg fr gb gr hk hu id ie il in it jp kr lt lv ma mx my ng nl no nz ph pl pt ro rs ru sa se sg si sk th tr tw ua us ve za"  
 return resp  
 response = requests.get("http://news:3003/news?country="+ country\_name)  
 return response.json()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(host="0.0.0.0",port=3001,debug=True,threaded=True)

weather.py

from flask import Flask, request  
from flask\_restful import Resource, Api, reqparse   
from json import dumps  
from flask import jsonify  
import json  
import requests  
  
app = Flask(\_\_name\_\_)  
api = Api(app)  
  
@app.route('/weather')  
def weather():  
 city\_name = request.args.get('city')  
 api\_key = "deb96cff96df7c74e93e62661b91c3c2"  
 base\_url = "http://api.openweathermap.org/data/2.5/weather?"  
 complete\_url = base\_url + "appid=" + api\_key + "&q=" + city\_name  
 response = requests.get(complete\_url)  
 return response.json()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(host="0.0.0.0",port='3002',threaded=True,debug=True)

news.py

from flask import Flask, request  
from flask\_restful import Resource, Api, reqparse   
from json import dumps  
from flask import jsonify  
import json  
import requests  
  
app = Flask(\_\_name\_\_)  
api = Api(app)  
  
@app.route('/news')  
def news():  
 country\_name = request.args.get('country')  
 api\_key = "70dcd1c6a0d24ebdb57ff071ff9b8ddc"  
 base\_url = "http://newsapi.org/v2/top-headlines?"  
 complete\_url = base\_url + "country=" + country\_name + "&apiKey=" + api\_key  
 response = requests.get(complete\_url)  
 return response.json()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(host="0.0.0.0",port='3003',threaded=True,debug=True)

**Лабораторная работа №3 «Сборка и доставка»**

**Цели работы**: добавить в проект сборку и доставку.

Каждый описанный сервис был разработан в изолированной среде в виде

docker контейнеров. Ниже приведены содержания соответствующих Dockerfile

Dockerfile

FROM python:3.8-slim-buster  
  
# Install dependencies:  
COPY requirements.txt .  
RUN pip install -r requirements.txt  
  
EXPOSE 3001  
  
COPY master\_assistant.py .  
  
# Run the application:  
CMD python master\_assistant.py

Dockerfile.news

FROM python:3.8-slim-buster  
  
# Install dependencies:  
COPY requirements.txt .  
RUN pip install -r requirements.txt  
  
EXPOSE 3003  
  
COPY news.py .  
  
# Run the application:  
CMD python news.py

Dockerfile.weather

FROM python:3.8-slim-buster  
  
# Install dependencies:  
COPY requirements.txt .  
RUN pip install -r requirements.txt  
  
EXPOSE 3002  
  
COPY weather.py .  
  
# Run the application:  
CMD python weather.py

Dockerfile.nginx

# using Nginx base image  
FROM nginx  
  
# delete nginx default .conf file  
RUN rm /etc/nginx/conf.d/default.conf  
  
# add the .conf file we have created  
COPY nginx.conf /etc/nginx/nginx.conf

Для тестирования функциональности системы был использован инструмент

docker-compose. Ниже приведено содержание docker-compose.yml:

docker-compose.yml.

version: "3.9"  
  
networks:  
 monitoring:  
 driver: bridge  
  
volumes:  
 prometheus\_data: {}  
  
services:  
 nginx:  
 container\_name: nginx  
 build:  
 context: .  
 dockerfile: Dockerfile.nginx  
 ports:  
 - 80:80  
 depends\_on:  
 - master  
 master:  
 build:  
 context: .  
 dockerfile: Dockerfile  
 weather:  
 build:  
 context: .  
 dockerfile: Dockerfile.weather  
 news:  
 build:  
 context: .  
 dockerfile: Dockerfile.news  
  
  
  
 node-exporter:  
 image: prom/node-exporter:latest  
 container\_name: node-exporter  
 restart: unless-stopped  
 volumes:  
 - /proc:/host/proc:ro  
 - /sys:/host/sys:ro  
 - /:/rootfs:ro  
 command:  
 - '--path.procfs=/host/proc'  
 - '--path.rootfs=/rootfs'  
 - '--path.sysfs=/host/sys'  
 - '--collector.filesystem.mount-points-exclude=^/(sys|proc|dev|host|etc)($$|/)'  
 expose:  
 - 9100  
 networks:  
 - monitoring  
  
 prometheus:  
 image: prom/prometheus:latest  
 container\_name: prometheus  
 restart: unless-stopped  
 volumes:  
 - ./prometheus.yml:/etc/prometheus/prometheus.yml  
 - prometheus\_data:/prometheus  
 command:  
 - '--config.file=/etc/prometheus/prometheus.yml'  
 - '--storage.tsdb.path=/prometheus'  
 - '--web.console.libraries=/etc/prometheus/console\_libraries'  
 - '--web.console.templates=/etc/prometheus/consoles'  
 - '--web.enable-lifecycle'  
 expose:  
 - 9090  
 networks:  
 - monitoring

Запущенные микросервисы как докер контейнеры в интерфейсе Docker-Desktop:



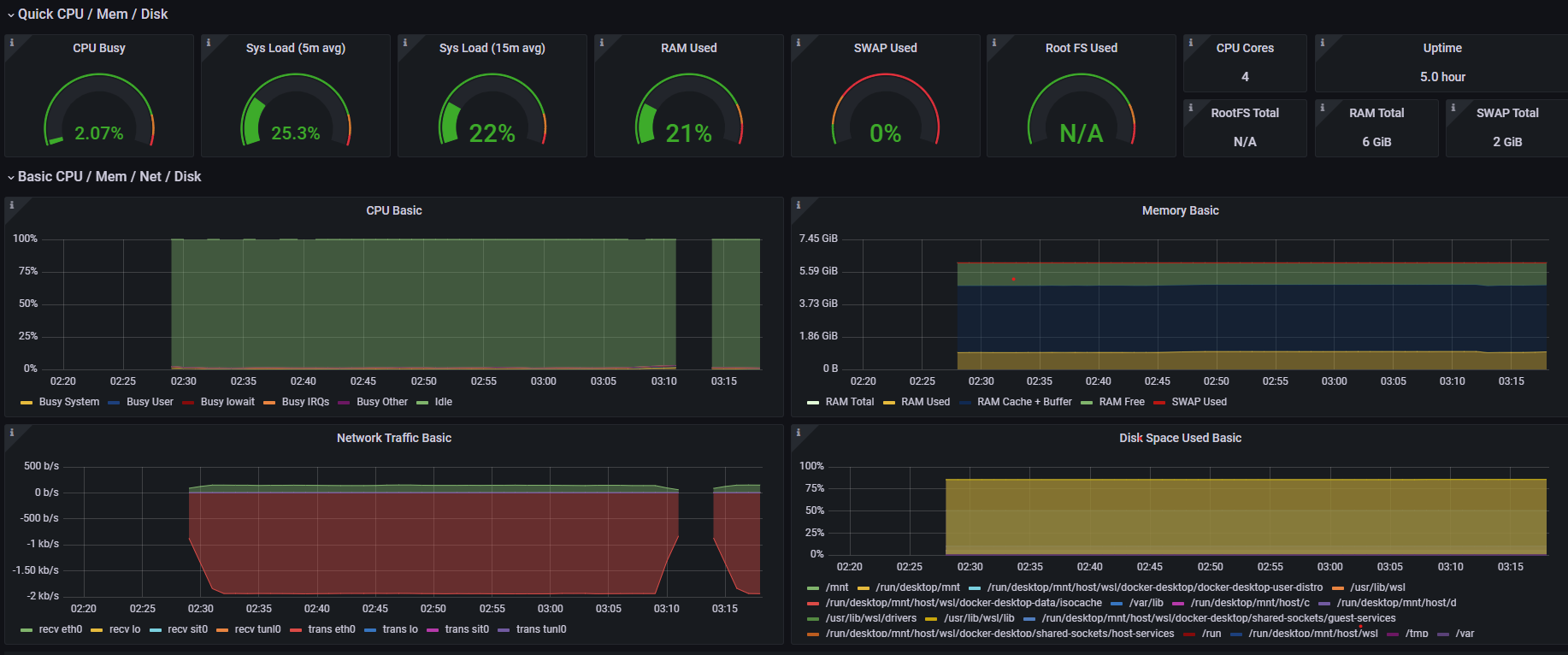
**Лабораторная работа №4 «Метрики»**

Для отслеживания состояния сервера был использован стек Prometheus+Grafana для мониторинга работы сервера. Для этого в файл docker-compose.yml были добавлены сервисы node-exporter, который отвечает

за получение данных для мониторинга из машины, на которой запущен сервер, prometheus для сбора этих данных и grafana для визуализации собранных данных в браузере.

Для визуализации отслеживаемых данных о сервере был использован

распространенный для node-exporter шаблон дашборда Node Exporter Full.



**Лабораторная работа №5 «Балансировка нагрузок»**

NGINX используется в качестве балансировщика нагрузки, который будет распределять входящие запросы по запущенным экземплярам сервиса master\_assistant. Для этого был добавлен сервис nginx в файл docker-compose.yml:

Для масштабирования серверов master- assistant, команда docker-compose up была вызвана с флагом **--master=2**.

Балансировка нагрузки с помощью Nginx по умолчанию использует round-robin алгоритм, если другой метод не определен. Каждый сервер выбирается по очереди в соответствии с порядком, который мы задаем им в файле Nginx.conf. Это уравновешивает количество запросов поровну между нашими двумя master экземплярами.

nginx.conf

events {}  
  
# Define which servers to include in the load balancing scheme.  
  
http {  
 upstream app {  
 server master;  
 server news\_weather\_services\_python\_docker-master-master-1:3001;  
 server news\_weather\_services\_python\_docker-master-master-2:3001;  
 }  
  
  
# This server accepts all traffic to port 80 and passes it to the upstream.  
  
 server {  
 listen 80;  
 server\_name app.com;  
 location / {  
 proxy\_pass http://app;  
 }  
 }  
}

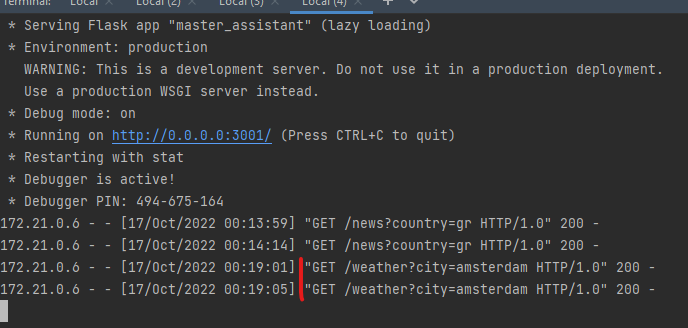
Теперь пришло время проверить, как работает баланс нагрузки. Открою два отдельных терминала и выполню команды:

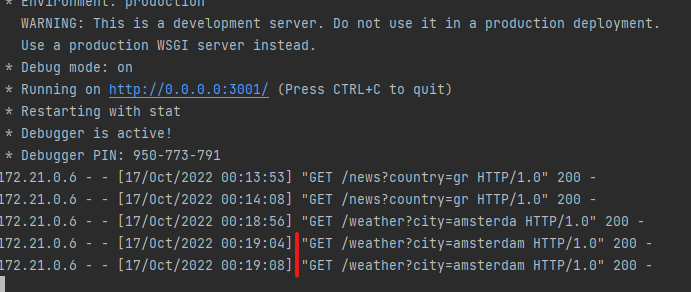
docker logs news\_weather\_services\_python\_docker-master-master-1 –f

docker logs news\_weather\_services\_python\_docker-master-master-2 –f

перейдя по ссылке: <http://localhost/weather?city=amsterdam>

и обновляя страницу можно заметить как запросы перераспределяются между двумя master серверами. Были выполнены 4 запроса в временном промежутке 00:19:00 – 00:19:10





Таким образом, была проверена работа балансировщика нагрузки в режиме

active/active

Возвращаемый JSON при переходе по ссылке: <http://localhost/weather?city=amsterdam>

