به نام خدا

دانشکده مهندسی کامپیوتر

گزارش پروژه اول

درس: رایانش ابری

دانشجو: فرشید نوشی – ۹۸۳۱۰۶۸

معماری سیستم برای انجام یک سرویس اجرای وظیفه توزیع شده طراحی شده است. این سیستم از چندین سرویس شامل سرویس ایجاد شغل، سرویس اجرای کار، سرویس آپلود فایل و سرویس اطلاع رسانی ایمیل تشکیل شده است. این سرویس ها با استفاده از صف های پیام و تماس های RESTful API با یکدیگر ارتباط برقرار می کنند و از سرویس های ابری مختلف از جمله RabbitMQ، Avien DBaaS، MailGun و AWS S3 استفاده می کنند.

برای پیاده سازی از FastAPI برای ایجاد یک فایل پایتون و گذاشتن API ها برای کاربران بابت استفاده استفاده کردیم. این سرور HTTP ۴ API مطلوب را برای کاربران تولید میکند که به صورت دیفالت روی پورت ۸۰۰۰ بالا می‌آید. دو پردازه دیگر مربوط به سرویس‌های اشتغال‌زایی وظایف و اجرای آن‌ها نیز هستند که در ادامه توضیحات مربوط به آن‌ها آورده شده‌اند. در کد نوشته شده کامنت‌های مناسب برای خواندن کد آورده شده‌اند.

```python
from typing import Optional

import requests
from auth0.authentication import GetToken
from auth0.management import Auth0
from authlib.jose import jwt
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel

app = FastAPI()


# Define a model for the user data
class User(BaseModel):
    email: str
    password: str


# Define a function to authenticate the user using Auth0 and create the user if they don't exist
def authenticate_user(email: str, password: str) -> Optional[str]:
    # Set the Auth0 parameters
    auth0_domain = 'dev-yf3s0lg5eryzqb8u.us.auth0.com'
    auth0_audience = 'https://dev-yf3s0lg5eryzqb8u.us.auth0.com/api/v2/'
    auth0_client_id = 'fW0cnOXSB0XfmXGcD4mvdcLHQc0bzF2'
    auth0_client_secret = 'g__0UWe5sUps8UNlAxUPA7USnB2TA4Y4WMkQywDIaWHhrX26vmC6MqcKhYyzmFce'

    # Build the Auth0 API request URL
    auth0_url = f'https://{auth0_domain}/oauth/token'

    # Set the Auth0 API request data
    auth0_data = {
        'grant_type': 'password',
        'username': email,
        'password': password,
        'audience': auth0_audience,
        'client_id': auth0_client_id,
        'client_secret': auth0_client_secret
    }

    # Send the Auth0 API request
    auth0_response = requests.post(auth0_url, data=auth0_data)

    print(auth0_response.json())
```

سرویس اشتغال زایی وظیفه ایجاد مشاغل و ذخیره آنها در پایگاه داده را بر عهده دارد. هنگامی که یک کاربر یک درخواست شغلی ارسال می کند، سرویس درخواست را دریافت می کند و اطلاعات کار را در Avien DBaaS ذخیره می کند. این سرویس همچنین پیامی را به صف RabbitMQ ارسال می کند که حاوی شناسه شغلی است. اطلاعات فایل اجرایی در پایگاه داده ذخیره می شود که شامل محتویات فایل و زبان برنامه نویسی است. این سرویس مستقل از سایر سرویس های موجود در سیستم است و می تواند در صورت لزوم scale شود. در آغاز کار اطلاعات مربوط به کانکشن به سرویس‌های ابری در این فایل آورده شده‌اند.

سرویس اجرای مشاغل وظیفه اجرای کارهایی را که در جدول مشاغل ایجاد شده‌اند بر عهده دارد. این سرویس از پایگاه داده می خواند و کارهای انجام نشده را با وضعیت هیچ بازیابی می کند. سپس این سرویس یک درخواست HTTP را به سرویس CodeX ارسال می کند که وظیفه اجرای کد کاربر را بر عهده دارد. در صورت بروز خطا در اجرای کد کاربر، سرویس از طریق سرویس MailGun برای کاربر ایمیل می فرستد و مشخص می کند که فایل دارای خطا بوده و از اجرای مجدد آن جلوگیری می کند. در پایان، سرویس نتیجه را برای کاربر ایمیل می کند (در صورت وجود خطا، متن خطا و در صورت موفقیت آمیز بودن اجرا، خروجی کد را ایمیل می‌کند). در نهایت، سرویس وضعیت شغل را به اجرا در پایگاه داده تغییر می دهد. سرویس آپلود فایل وظیفه آپلود فایل ها در S3 را بر عهده دارد. هنگامی که کاربر فایلی را ارسال می کند، سرویس فایل را دریافت کرده و آن را در سطل S3 مشخص شده آپلود می کند. سپس این سرویس metadata فایل را در Avien DBaaS ذخیره می کند. این سرویس مستقل از سایر سرویس های موجود در سیستم است و می تواند در صورت لزوم scale شود.

```
    # Store the file contents in the database
    params = {
        'file_id': file_id,
        'language': file_data[2],
        'contents': file_contents
    }
    query_string = urlencode(params)
    cursor.execute("INSERT INTO file_contents VALUES (%s, %s, %s)", (file_id, file_data[2], file_contents))
    conn.commit()

    return "File contents stored successfully"


# Define a function to handle incoming messages from RabbitMQ
Farshid Nooshi
def callback(ch, method, properties, body):
    # Get the file ID from the message body
    file_id = int(body.decode())

    # Execute the file and store its contents in the database
    result = execute_file(file_id)

    # Print the result
    print(result)

    # Acknowledge the message
    ch.basic_ack(delivery_tag=method.delivery_tag)


# Start consuming messages from RabbitMQ
channel.basic_qos(prefetch_count=1)
channel.basic_consume(queue='file_creation', on_message_callback=callback)
channel.start_consuming()
```

```
from urllib.parse import urlencode

import pika
import psycopg2
import requests

# Set up RabbitMQ connection
cloudamqp_url = 'amqps://ghtiaznm:nfMSp4UtGzag-qUSjEL3z77lPIr14rpA@gull.rmq.cloudamqp.com/ghtiaznm'
params = pika.URLParameters(cloudamqp_url)
connection = pika.BlockingConnection(params)
channel = connection.channel()
channel.queue_declare(queue='file_creation')

# Set up database connection
db_url = 'mysql://avnadmin:AVNS_6piYB8BWUXyYQ2Udutn@mysql-2356b3c0-cloud-computing-app.aivencloud.com:24313/defaultdb?ssl-mode=REQUIRED'
conn = psycopg2.connect(db_url)
cursor = conn.cursor()


# Define a function to execute the file and store its contents in the database
Farshid Nooshi
def execute_file(file_id):
    # Get the executable file information from the database
    cursor.execute("SELECT * FROM executable_files WHERE id=%s", (file_id,))
    file_data = cursor.fetchone()

    # Check if the file exists
    if not file_data:
        return "File not found"

    # Execute the file and get its contents
    response = requests.get(file_data[1])
    file_contents = response.text

    # Store the file contents in the database
    params = {
        'file_id': file_id,
        'language': file_data[2],
        'contents': file_contents
    }
    query_string = urlencode(params)
    cursor.execute("INSERT INTO file_contents VALUES (%s, %s, %s)", (file_id, file_data[2], file_contents))
    conn.commit()

    return "File contents stored successfully"
```

برای استفاده راحت‌تر از برنامه‌های دیگر که به صورت پیش‌فرض در فایل Http Server ما نیستند Logger قرار دادیم که بتوانیم در صورت لزوم اطلاعات مربوط به دیباگ نرم افزار را در داخل کنسول مشاهده کنیم. در مورد صحبت با پایگاه داده در کدها نیز از دستورهای SQL برای ریختن و یا ویرایش داده‌ها استفاده کردیم که تغییرات لازم را در پایگاه داده MySQL اعمال کنند. تغییرات نیز زمانی که برای ایجاد شغل‌ها یا خبر اجرایشان بود برروی RabbitMQ همانند نیازمندی مطرح شده قرار گرفته‌اند. برای مدیریت دیتابیس برروی کدها از پکیج psycopg2 استفاده کردیم.

```python
import logging
import os

import boto3
import pika
import psycopg2
import requests
from mailgun import Mailgun

# Configure logger
logging.basicConfig(level=logging.INFO, format='%(asctime)s %(levelname)s %(message)s')
logger = logging.getLogger(__name__)

# RabbitMQ configurations
RABBITMQ_URL = 'amqps://ghtiaznm:nfMSp4UtGzag-qUSjEL3z77lPIr14rpA@gull.rmq.cloudamqp.com/ghtiaznm'

# Avien DB configurations
AVIEN_DB_HOST = 'mysql-2356b3c0-cloud-computing-app.aivencloud.com'
AVIEN_DB_PORT = '24313'
AVIEN_DB_NAME = 'defaultdb'
AVIEN_DB_USER = 'avnadmin'
AVIEN_DB_PASSWORD = 'AVNS_6piYB8BWUXyYQ2Udutn'

# Mailgun configurations
MAILGUN_API_KEY = 'e8e51d665fd3789aad045b5dc5c7a937-81bd92f8-5cb782a1'
MAILGUN_DOMAIN = 'https://api.mailgun.net/v3/sandboxd810fa8fbc0944a39c6fb2760433b07d.mailgun.org'
MAILGUN_SENDER = 'mailgun@sandboxd810fa8fbc0944a39c6fb2760433b07d.mailgun.org'

# S3 configurations
S3_ACCESS_KEY = '11f00242-2593-4a34-bdb8-c45074b28ccc'
S3_SECRET_KEY = '95b771c08e979dae3bf7f8ca946fcac7a639d619'
S3_REGION_NAME = 's3.ir-thr-at1.arvanstorage.ir'

# CodeX service URL
CODEX_SERVICE_URL = 'https://api.codex.jaagrav.in'

# Create a connection to RabbitMQ
rabbitmq_connection = pika.BlockingConnection(pika.URLParameters(RABBITMQ_URL))
rabbitmq_channel = rabbitmq_connection.channel()

# Create a connection to Avien DB
avien_db_connection = psycopg2.connect(
    host=AVIEN_DB_HOST,
    port=AVIEN_DB_PORT,
```

به طور کلی، سیستم به گونه ای طراحی شده است که توزیع شده و مقیاس پذیر باشد. استفاده از صف‌های پیام و تماس‌های API RESTful امکان جداسازی سرویس‌ها و ادغام آسان‌تر با سایر سرویس‌های ابری را فراهم می‌کند. استفاده از سرویس های ابری مانند Avien DBaaS، MailGun و AWS S3 امکان مقیاس پذیری آسان و هزینه تعمیر و نگهداری کم را فراهم می کند.

```python
# Create a Mailgun client
mailgun = Mailgun(api_key=MAILGUN_API_KEY, domain=MAILGUN_DOMAIN)


Farshid Nooshi
def execute_job(job_id):
    """
    Executes the job with the given ID by sending an HTTP request to the CodeX service.
    Sends an email to the user if there is an error in the execution and changes the job status to executed.
    """
    # Get the job information from Avien DB
    avien_db_cursor.execute('SELECT * FROM jobs WHERE id=%s', (job_id,))
    job = avien_db_cursor.fetchone()
    if job is None:
        logger.error(f'Job with ID {job_id} not found')
        return

    # Check if the job is not executed yet
    if job[4] != 'none':
        logger.warning(f'Job with ID {job_id} has already been executed')
        return

    # Download the executable file from S3
    s3_object = s3_client.get_object(Bucket=S3_BUCKET_NAME, Key=job[2])
    executable_content = s3_object['Body'].read().decode('utf-8')

    # Send an HTTP request to the CodeX service
    response = requests.post(CODEX_SERVICE_URL, json={
        'executable': executable_content,
        'language':
            job[3],
        'input':
            job[5]
    })

    # Check if the request was successful
    if response.status_code != 200:
        logger.error(f'Job with ID {job_id} failed to execute')
        mailgun.send_email(
```