СЛЁРМ Обучение для инженеров и технических лидеров

Дата-инженер

Web-серверы и клиенты, создание API

Николай Марков

Цели урока. Что вы узнаете:

1. Как проходит игра престолов между HTTP-клиентами
2. Как выглядит классическая веб-разработка на Python
3. Как пишутся современные асинхронные микросервисы
4. Пара слов о Model & Feature Serving

slurm.io

Николай Марков

О спикере

- Занимаюсь профессиональной разработкой и проектированием уже больше 11 лет
- Начинал программистом на Python, сейчас Principal Architect в компании Aligned Research Group, а также ментор в компании DryLabs
- В разное время работал с сетями, протоколами и различными облаками (AWS, GCP, Azure, OpenStack)
- Писал проекты на Python, а также Golang, C/C++, Scala и Rust
- Сейчас выстраиваю аналитические архитектуры и Data Governance в разных компаниях

slurm.io

Самые популярные клиенты для HTTP

slurm.io

Urllib

```python
import urllib.parse
import urllib.request

# URL - адрес, получающий данные
url = 'http://www.someserver.com/cgi-bin/register.cgi'
# Отправляемые данные
values = {'name' : 'Michael Foord',
          'location' : 'Northampton',
          'language' : 'Python' }

# Кодирование данных
data = urllib.parse.urlencode(values)
# данные должны быть байтами
data = data.encode('utf-8')
req = urllib.request.Request(url, data)
with urllib.request.urlopen(req) as response:
  the_page = response.read()
```

Документация по модулю

slurm.io

Urllib

urllib.request — Extensible library for opening URLs

Source code: Lib/urllib/request.py

The urllib.request module defines functions and classes which help in opening URLs (mostly HTTP) in a complex world — basic and digest authentication, redirections, cookies and more.

See also: The Requests package is recommended for a higher-level HTTP client interface.

Availability: not Emscripten, not WASI.

This module does not work or is not available on WebAssembly platforms `wasm32-emscripten` and `wasm32-wasi`. See WebAssembly platforms for more information.

The urllib.request module defines the following functions:

urllib.request.**urlopen**(*url*, *data=None*, [*timeout*,]*, *cafile=None*, *capath=None*, *cadefault=False*, *context=None*)

Open the URL *url*, which can be either a string or a Request object.

slurm.io

Requests

```python
import requests

# Search GitHub's repositories for requests
response = requests.get(
    'https://api.github.com/search/repositories',
    params={'q': 'requests+language:python'},
)

# Inspect some attributes of the `requests` repository
json_response = response.json()
repository = json_response['items'][0]
print(f'Repository name: {repository["name"]}')
print(f'Repository description: {repository["description"]}')
```

[Обучалка по модулю]

slurm.io

HTTPX

```python
1.  import httpx
2.
3.  # Search GitHub's repositories for httpx
4.  response = httpx.get(
5.      'https://api.github.com/search/repositories',
6.      params={'q': 'httpx+language:python'},
7.  )
8.
9.  # Inspect some attributes of the first repository
10. json_response = response.json()
11. repository = json_response['items'][0]
12. print(f'Repository name: {repository["name"]}')
13. print(f'Repository description: {repository["description"]}')
```

[Обучалка по модулю]

slurm.io

Игра престолов

HTTPX **поддерживает HTTP/2**. В requests его поддержка завязана на пакет urllib3, в котором его пока отказались завозить. Также можно использовать **низкоуровневый** pycurl, он быстрее, но сложнее в использовании

HTTPX **НЕ ходит** по редиректам автоматически

Также существует aiohttp, на него мы посмотрим чуть позже

```
1.  import asyncio
2.  import httpx
3.
4.  async def main():
5.      async with httpx.AsyncClient() as client:
6.          response = await client.get('https://www.example.com/')
7.          print(response)
8.
9.  asyncio.run(main())
```

Основная фишка HTTPX — поддержка **синхронного и асинхронного** интерфейса в одной библиотеке

slurm.io

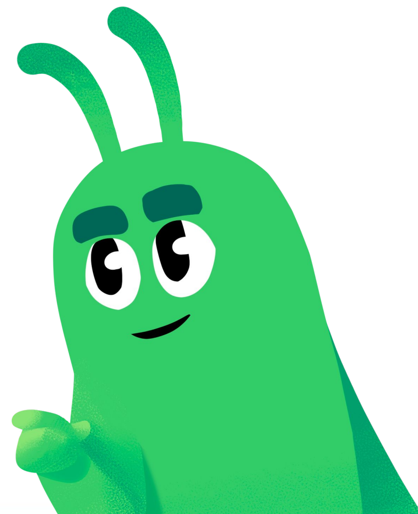
Из чего состоят сервисы

slurm.io

REpresentational State Transfer

Официальной спецификации, описывающей REST, не существует, но вместо нее часто ссылаются на исходную докторскую диссертацию Роя Филдинга (Roy Fielding)

Однако, по итогу реализация у каждого может быть своя



slurm.io

REpresentational State Transfer

- Протокол HTTP поддерживает [большое количество методов](), даже несмотря на то, что в вебе в основном используются только POST и GET

- Эти методы условно ложатся на управление **объектами** и **коллекциями** этих объектов, а URL явно их **идентифицирует**:
  - /api/v1/animals/123/ <- конкретный объект «животное»
  - /api/v1/animals/ <- коллекция объектов «животное»

- Частый вариант применения методов: **GET**-чтение, **POST**-создание, **PUT**-обновление, **DELETE**-удаление

- В идеале сервер **не должен хранить никакое состояние (сессию)**, а вся информация для обработки должна передаваться в запросе

slurm.io

GraphQL

GraphQL часто пытаются продать, как «лучший REST», однако приложения
на обоих запросто могут быть **функционально эквивалентны**.

GraphQL — спецификация не архитектуры**, а языка запросов** к бэкенду.

```
query GetPets {
  pets {
    name
    petType
  }
}
```

```
{
  "data": {
    "pets": [
      {
        "name": "Sandy",
        "petType": "Cat"
      },
      {
        "name": "Hank",
        "petType": "Dog"
      }
    ]
  }
}
```

Не подходит:
GraphQL сложно поддерживать, поэтому не стоит его использовать для **небольшого API.**

Подходит:
Если необходимы графы и сложные взаимосвязи объектов.

slurm.io

OpenAPI (Swagger)

```
1.  openapi: 3.0.0
2.  info:
3.    title: Sample API
4.    description: Sample description of a web service
5.    version: 0.1.9
6.  servers:
7.    - url: http://api.example.com/v1
8.      description: Optional server description, e.g. Main
    (production) server
9.  paths:
10.   /users:
11.     get:
12.       summary: Returns a list of users.
13.       responses:
14.         '200':    # status code
15.           description: A JSON array of user names
16.           content:
17.             application/json:
18.               schema:
19.                 type: array
20.                 items:
21.                   type: string
```

OpenAPI помогает **найти общий язык между фронтендом и бэкендом**, позволяет разрабатывать их параллельно и независимо

Существуют как инструменты по генерации кода на основе спеки, так и встроенная в фреймворки поддержка генерации самой спеки по коду приложения

slurm.io

Swagger UI

Можно делать тестовые запросы и демонстрировать логику работы API напрямую через Swagger UI

slurm.io

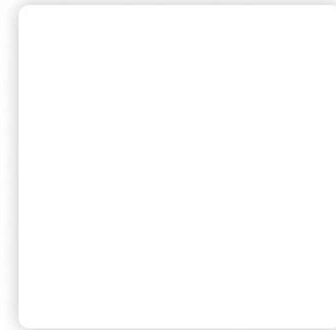
API Gateway

Общий механизм авторизации для доступа сразу ко многим сервисам

Часто API Gateway представляет собой прокси-сервер, позволяющий дополнительно настроить **балансировку**, **маршрутизацию** и даже разные штуки **уровня приложения**

slurm.io

Популярные фреймворки для Python



slurm.io

WSGI

Есть веб-сервера (Apache, Nginx), а есть стандарты взаимодействия между ними и приложениями на конкретных языках программирования

WSGI появился, как потомок CGI специально для Python, и поддерживается многими популярными фреймворками, например, Django и Flask'ом

Для гибкости настройки масштабирования обычно используется промежуточный WSGI-сервер, реализующий дополнительные возможности по масштабированию

Bjoern

uWSGI

Gunicorn

slurm.io

Django

```
mysite
├── manage.py
├── mysite
│   ├── asgi.py
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-310.pyc
│   │   └── settings.cpython-310.pyc
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── news
    ├── admin.py
    ├── apps.py
    ├── __init__.py
    ├── migrations
    │   └── __init__.py
    ├── models.py
    ├── tests.py
    └── views.py
```

Разрабатывался, как фреймворк для контентных веб-сайтов, а не для API

slurm.io

Django

- Имеет админку, удобные средства для работы с БД, генератор форм и шаблонизатор HTML
- Позволяет удобно переиспользовать одни и те же компоненты в разных бэкендах на том же фреймворке
- Легко найти людей на поддержку

- Не поддерживает REST из коробки и требует дополнительный модуль django-rest-framework (GraphQL)
- Встроенный ORM плохо подходит для работы с аналитическими БД
- Раздутая кодовая база и избыточность

slurm.io

Flask

```python
1.  from flask import Flask
2.
3.  app = Flask(__name__)
4.
5.  @app.route("/")
6.  def hello_world():
7.      return "<p>Hello, World!</p>"
```

```
(dj)  meow-nofer@pikachu  ~  flask --app test run
 * Serving Flask app 'test'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [29/Apr/2023 17:11:17] "GET / HTTP/1.1" 200 -
```

```
 meow-nofer@pikachu  ~  curl http://127.0.0.1:5000/
<p>Hello, World!</p>%
```

Flask не имеет встроенных средств для работы с БД, но чаще всего к нему в пару берут SQLAlchemy

slurm.io

Отложенные задания — Celery

- При работе с данными абсолютно нормально иметь долгоиграющие процессы для их обработки на бэкенде
- В идеальном мире они не должны мешать работе самого API
- Таски должны быть персистентными — не исчезать при падении приложения или даже текущего сервера

slurm.io

Отложенные задания — Celery

~$ celery -A tasks worker --loglevel=INFO

```
1.  from celery import Celery
2.
3.  app = Celery(
4.     'tasks',
5.     backend='redis://localhost' ,
6.     broker='pyamqp://guest@localhost//'
7.  )
8.
9.  @app.task
10. def add(x, y):
11.    return x + y
```

```
1.  from tasks import add
2.  result = add.delay(4, 4)
3.  print(result.ready())  # True/False
4.  ...
5.  print(result.get())  # 8
```

slurm.io

На что обратить внимание

- Django - прост, подходит для старта
- Flask - сложнее, но больше гибкости
- Celery - подходит для реализации отложенных заданий

slurm.io

Современные асинхронные микросервисы

slurm.io

Что такое микросервис?

Сервис, который имеет внешний API и решает одну конкретную независимую небольшую задачу, часто со своей личной, независимой базой данных

[Например, телеграм-бот]

[Основной источник от Криса Ричардсона]

slurm.io

Кроулер для опроса URL’ов

- REST API
- Таски? Очередь? Воркеры?
- Celery? Redis?
- …
- DOOMED

slurm.io

Кроулер для опроса URL’ов

~$ pip install aiohttp uvloop cchardet aiodns

```
1.   import asyncio
2.   import uvloop
3.   from aiohttp import web
4.
5.   asyncio.set_event_loop_policy(uvloop.EventLoopPolicy())
6.   app = web.Application()
7.
8.   ...
9.
10.  web.run_app(app, host=host, port=port)
11.
```

Asyncio поддерживает нестандартные имплементации event loop’a

slurm.io

Операции и эндпойнты

```
1.  def run_service(host, port):
2.      app.router.add_post('/submit', submit)
3.      app.router.add_get('/tasks', tasks)
4.      app.router.add_get('/tasks/{task_id}', task_status)
5.      app.router.add_delete('/tasks/{task_id}', task_delete)
6.      web.run_app(app, host=host, port=port)
7.
8.
9.  if __name__ == "__main__":
10.     run_service("0.0.0.0", 7777)
```

slurm.io

Глобальный реестр задач

app['TASKS'] = {}

Конечно, это читерство, но так тоже работает

```
1.   async def tasks(request):
2.       return web.json_response({
3.           k: {
4.               "status": v["status"],
5.               "result": v.get("result", {})
6.           } for k, v in request.app['TASKS'].items()
7.       })
```

slurm.io

Модель данных - Task

```
{
   "status": "running",
   "workers": [],
   "result": {
         "https://example.com": {
            "timestamp": int(time.time()),
            "errors": {
               "http": 404
            }
         },
         "https://another-example.io": {
            "timestamp": int(time.time()),
            "errors": {
               "http": "SSL_ERROR"
            }
         }
   }
}
```

slurm.io

Отслеживание прогресса

```python
1.    async def task_status(request):
2.        task_id = request.match_info.get('task_id')
3.
4.        if not task_id or task_id not in request.app['TASKS']:
5.            return web.Response(status=404)
6.
7.        task = app['TASKS'].get(task_id)
8.        return web.json_response({
9.            "status": task.get("status"),
10.           "result": task.get("result", {})
11.       })
```

```python
1.    async def task_delete(request):
2.        task_id = request.match_info.get('task_id')
3.
4.        if not task_id or task_id not in request.app['TASKS']:
5.            return web.Response(status=404)
6.
7.        del app['TASKS'][task_id]
8.        return web.Response(status=204)
```

slurm.io

Создание и запуск таска

```
1.    async def submit(request):
2.        data = await request.json()
3.        urls_count = len(data["urls"])
4.
5.        task_id = str(uuid.uuid4())
6.        request.app['TASKS'][task_id] = {
7.            "status": "running",
8.            "workers": [],
9.            "result": {}
10.       }
11.       logger.info(
12.           "Task '{0}' created: {1} URLs received" .format(
13.               task_id,
14.               urls_count
15.           )
16.       )
17.
18.       asyncio.ensure_future(
19.           worker(task_id, data["urls"])
20.       )
21.
22.       return web.json_response({"task_id": task_id})
```

Я не могу однозначно рекомендовать сервисы, написанные подобным образом, для продакшена (как минимум лучше использовать **внешнего брокера** и **Swagger**). Однако моя цель — показать выгодные отличия асинхронного подхода перед синхронным

slurm.io

FastAPI

```python
1.  from fastapi import FastAPI, APIRouter
2.  from pydantic import BaseModel
3.
4.  app = FastAPI()
5.  router = APIRouter(prefix="/api/v1")
6.
7.  class Greeting(BaseModel):
8.      message: str
9.
10.
11. @router.get("/greeting", response_model=Greeting)
12. async def root():
13.     return Greeting(message="Hello, World!")
14.
15.
16. app.include_router(router)
```

ASGI

~$ uvicorn test:app

```
meow-nofer@pikachu ~ curl http://127.0.0.1:8000/api/v1/greeting
{"message":"Hello, World!"}%
```

slurm.io

Model & Feature Serving

slurm.io

Model Serving

- MLFlow Serving
- TorchServe
- Tensorflow Serving
- NVIDIA Triton
- BentoML

Каждый фреймворк изобретает свой способ сервить модели

slurm.io

ChatOps

DEV TEAM

CHAT APP

BOT

INFRASTRUCTURE

opsdroid

errbot

slurm.io

Feature Store

- Модели машинного обучения используют наборы признаков в двух процессах:
    - Обучение
    - Инференс
- Это может происходить как при исследовании, так и в продакшене
- Когда и фич, и моделей много — процесс получения выборок по конкретным фичам имеет смысл автоматизировать и регламентировать

tecton

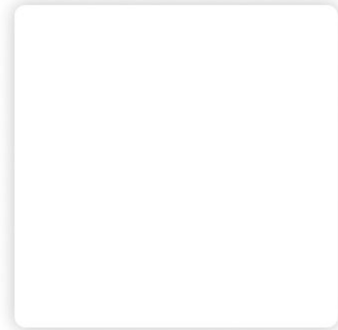
slurm.io

Итоги. О чем поговорили:

1. Какие есть самые популярные HTTP-клиенты для Python и чем они отличаются
2. Какие фреймворки используются в классической веб-разработке
3. Требования к современным веб-сервисам
4. Как можно писать современные асинхронные микросервисы
5. Что такое Model & Feature Serving

slurm.io

Спасибо за внимание!



slurm.io