DevOps – Lecture 06

Microservice-based Application Development

20 Oct 2021

Chinmaya Dehury

Chinmaya.Dehury@ut.ee
Note – DevOps final Exam

• 15 Dec 2021 ; 4-6PM Estonia Time
  • 22 students*
• 07 Jan 2022 ; 2-4PM Estonia Time
  • 8 students*

* As on 19 Oct 2021, 3:40PM
OUTLINE

• An example Application
• Monolith Architecture
• Microservice Architecture
• OpenAPI
• Swagger
What will you learn?

a basic understanding of microservices, containers, and cloud-native applications
SmartBuilding Application – An example

Features
• Add/Remove new smart devices/appliances, e.g. Coffee Machine, Refrigerator
• Energy consumption (daily, monthly consumption)
• Usage/Activities of appliances
• Alerts / Notification

Underline technologies...
• Database
• Frontend
• Prog. Language, many more
SmartBuilding Application - Monolithic approach

Example:

*SmartBuilding.py* - IoT data processing

- `read()` .... : Read data (from multiple .csv files, from local file system)
- `outlier()`, `missing()`, ... : Preprocess the data (e.g. outlier, missing data)
- `mean()`, `max()` ... : Analyze the data
- `pushToMongo()`, `pushToS3()` ... : Push data to other database
- `viz()` ... : Visualize the data
**SmartBuilding Application - Monolithic approach**

Example: *Decomposing monolith application*

**SmartBuilding**.py - IoT data processing

- `read.py` - Read data (from multiple .csv files)
- `outlier.py`, `missing.py` ...- Preprocess the data (e.g. outlier, missing data)
- `mean.py`, `max.py` ...- Analyze the data
- `pushToMongo.py`, `pushToS3.py` ...- Push data to other database
- `viz.py` - Visualize the data
SmartBuilding Application - Microservice Approach

Example: Decomposing monolith application

SmartBuilding App - IoT data processing

- **ConsumeData Service** - Read data (from multiple .csv files) (`read.py`)
- **Outlier detection service, Missing data service** - Preprocess the data (e.g. outlier, missing data) (`outlier.py`, `missing.py`)
- **Data analytics service** - Analyze the data (`mean.py`, `max.py`)
- **Export data service** - Push data to other database (`pushToMongo.py`, `pushToS3.py`)
- **Visualization service** - Visualize the data (`viz.py`)
Monolith vs Microservice – in general
Monolithic Approach

- **Very tightly coupled**
  - Interdependent modules
  - Change in module requires an update in other module
- **Usually has three blocks:**
  - Database
  - Server-side application
  - Client UI
- **Usually this is a centralized approach**

Example:

![Diagram of SmartBuilding App](image)

* Only applicable to very small size application
Monolithic Approach - benefits

• **Faster Development***
  • Very few python files to develop
• **Faster Testing***
• **Faster Deployment***
  • Simply pack the pythons and deploy
• **Faster Scale***

Example:

- SmartBuilding App
  (SmartBuilding.py)
- Data
  (read.py, pushToMongo.py, pushToS3.py)
- Data visualization
  (viz.py)
- Data preprocessing
  (outlier.py, missing.py)
- Data analysis
  (mean.py, max.py)

Database

Visualization
templates

* Only applicable to very simple, small size application
Monolithic Approach - limitations

- **Not suitable for large applications**
- **Slow application development**
  - Difficult to understand the structure
  - The IDEs may not work fine with large codebase
    - E.g. Can you edit 15MB PDF smoothly?
- **Difficult to scale**
  - E.g. what will happen to PushToMongo.py when the data size increases to GBs
- **Reliability**
  - Due to higher interdependency
  - E.g. Minor bug in read.py may crash whole system
- **Flexibility of underlined software stack**
  - Can we develop each module using different technologies?

Example:

- **SmartBuilding App** (SmartBuilding.py)
  - Data (read.py, pushToMongo.py, pushToS3.py)
  - Data visualization (viz.py)
  - Data preprocessing (outlier.py, missing.py)
  - Data analysis (mean.py, max.py)

User

Database

Visualization templates
Microservice Approach

- Collection of *smaller* and *independent* services
- Usually this is a *distributed* approach
- **GOAL**: to accelerate software development by enabling continuous delivery/deployment.
- Services communicate via APIs
- Service:
  - Dedicated function
  - Codebase
  - Processes
  - Lifecycle
  - Database

**Example:**

1. Data storage
2. Data visualization
3. Data pre processing
4. Data analysis

**Visualization templates**

SmartBuilding App

User
Microservice Approach

- Service Lifecycle
  - Created
  - Installed
  - Configured
  - Start
  - Stop
  - Delete/remove
- Suitable for large team, large product/application

Example:

Database

- Services 1: Data storage
- Services 2: Data visualization
- Services 3: Data preprocessing
- Services 4: Data analysis

Visualization templates

User

SmartBuilding App
Microservice Approach – benefits

• **Service:**
  - Independent development
  - Independent test
  - Independent development
  - Independent update

• **Scaling**
  - E.g. create 5 replicas of data preprocessing service when the more frequent is arriving from the sensor
  - Able to scale Horizontally and vertically

• Improved fault isolation

**Example:**
Monolithic -> Microservice Architecture

Example:

### SmartBuilding App (SmartBuilding.py)
- Data
- Data visualization
- Data preprocessing
- Data analysis

### Processes
1. Data storage
2. Data visualization
3. Data preprocessing
4. Data analysis

---

LTAT.06.015 : DevOps : Lec 6 - Microservice-based Application Development
So how the microservices communicate?

• Communication Types:
  
  • **Synchronous**:
    • HTTP is a synchronous protocol.
    • The client sends a request and waits for a response from the service.
  
  • **Asynchronous**:
    • Sender usually doesn't wait for a response.
    • Protocols like AMQP (a protocol supported by many operating systems and cloud environments) use asynchronous messages.

• One-to-one Communication
• One-to-many Communication
So how the microservices communicate?

- **Communication Types:**
  - **One-to-one Communication**
    - A request must be processed by exactly one service.
    - E.g. Finding the min. temp of a particular date from the temp. sensor data
  - **One-to-many Communication**
    - Each request can be processed by zero to multiple receivers.
    - Asynchronous communication
    - Used in Event-driven architecture
    - example is the publish/subscribe mechanism
So how the microservices communicate?

Inter-process communication protocol such as HTTP, AMQP, or a binary protocol like TCP,
So how the microservices communicate?

HTTP/AMQP/TCP or others

HTTP

HTTP/AMQP/TCP or others

HTTP

inter-process communication protocol such as HTTP, AMQP, or a binary protocol like TCP,
HTTP

- **HTTP** is a protocol for fetching resources such as HTML documents.
- Application layer protocol for the World Wide Web
- foundation of any data exchange on the Web
- Follows the Request-Response model
- a client-server protocol,
  - Requests are initiated by the recipient, usually the Web browser.
HTTP Flow

When a client wants to communicate with a server:

• Client opens a TCP connection:

• Client Send an HTTP message:

• Read the response sent by the server, such as:

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
```
HTTP Request Methods

Some basic HTTP request methods are:

• GET
  • requests a representation of the specified resource
  • should only be used to request data (they shouldn't include data).

• PUT
  • creates a new resource or replaces a representation of the target resource with the request payload.

• POST
  • sends data to the server
  • Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles

• DELETE
  • deletes the specified resource.
HTTP Request Methods

Some basic HTTP request methods are:

- **GET**
  - requests a representation of the specified resource
  - should only be used to request data (they shouldn't include data).

- **PUT**
  - creates a new resource or replaces a representation of the target resource with the request payload.

- **POST**
  - sends data to the server
  - Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles

- **DELETE**
  - deletes the specified resource.

**CRUD Functions:**

- **Create** -> POST
- **Read** -> GET
- **Update** -> PUT
- **Delete** -> DELETE
HTTP Status codes

• 200 : OK success
• 201 : Created
• 202 : Accepted
• 400 : Bad Request (server could not understand the request due to invalid syntax)
• 404 : Not found
• 408 : Request Timeout
• 500 : Internal Server Error
• 502 : Bad Gateway
• 503 : Service Unavailable

Complete list of status codes: https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml#http-status-codes-1
How do microservices send data to others?

• Message Format:
  • JSON
  • XML
  • Binary Format: Very selective usage
How do microservices send data to others?

• Message Format:
  • **JSON**
  • XML
  • Binary Format: Very selective usage
JSON file

- **JSON: JavaScript Object Notation**
- Lightweight format for storing and transporting data
  - language-independent data interchange format
- JSON is often used when data is sent from a server to a web page
- "self-describing" and easy to understand
- .json file extension
- Example:

```json
{
  "employees": [
    {
      "firstName": "John", "lastName": "Doe"
    },
    {
      "firstName": "Anna", "lastName": "Smith"
    },
    {
      "firstName": "Peter", "lastName": "Jones"
    }
  ]
}
```
JSON file

- Curly braces hold objects
- Square brackets hold arrays
- Data is separated by commas
- Data is in name/value pairs
- Example:

```json
{
  "employees": [
    {
      "firstName": "John", "lastName": "Doe"
    },
    {
      "firstName": "Anna", "lastName": "Smith"
    },
    {
      "firstName": "Peter", "lastName": "Jones"
    }
  ]
}
```
JSON vs YAML

• YAML is a superset of JSON

• YAML parser can parse JSON files

• YAML: Comments are denoted with a hash/number sign.
  • Not allowed in JSON

• YAML: Hierarchy is denoted by using space characters, Not Tab character
  • JSON: Objects and Arrays are denoted in braces and brackets.

• YAML: String quotes are optional but it supports single and double quotes.
  • JSON: Strings must be in double quotes.

• YAML: Root node can be any of the valid data types.
  • JSON: Root node must either be an array or an object.
Is HTTP is enough?

inter-process communication protocol such as HTTP, AMQP, or a binary protocol like TCP,
REST

- Representational state transfer (REST)
- REST is an API framework built on HTTP
- A set of rules how to use the HTTP (or other) protocol for **communicating** and **exchanging data** with web services
- Provides a simplistic and uniform interface for web services
- RESTful design
  - meets performance needs,
  - Scalable
  - Simple to write and consume
  - Reliable
- URI identifies a resource
  - E.g.
    - https://api.etais.ee/api/projects/
    - https://api.etais.ee/api/openstacktenant-flavors/
    - https://api.etais.ee/api/openstacktenant-images/
    - https://api.etais.ee/api/openstacktenant-instances/
    - https://api.etais.ee/api/openstack-tenants/
REST

• Specifies which operations are allowed on which resources
  • POST, PUT, GET, DELETE, and others

• REST API flow manages both requests and responses
  • HTTP response from the service's endpoint
    • Status code +
    • Headers +
    • data payload (optional)

• **Statelessness** – No session information stored in server
  • Session on client side, Data in a database

REST request example

• Download and Install Postman: [https://www.postman.com/downloads/](https://www.postman.com/downloads/)


• Browse [https://api.etais.ee/api/openstacktenant-instances/](https://api.etais.ee/api/openstacktenant-instances/) and see the output
REST request example

Using RESTed Firefox Extension

1. Request Method
2. Endpoint
3. Header Key “Authorization”
4. Value of “Authorization” key
5. Response Code
6. Response in JSON

You may try some other Endpoints:
- https://api.etais.ee/api/projects/
- https://api.etais.ee/api/openstacktenant-flavors/
- https://api.etais.ee/api/openstacktenant-images/
- https://api.etais.ee/api/openstacktenant-instances/
- https://api.etais.ee/api/openstack-tenants/
REST request example

https://api.etais.ee/api/openstacktenant-instances/?page=24

- Protocol:
  - Can be http:// or https://
- Domain name
  - api.etais.ee is the server’s domain name
- Resource path
  - api/openstacktenant-instances is the resource path of the API call
- Parameter
  - ?page=24 is a parameter, return the 24th page of the data
REST request example – Create a VM in ETAIS

REST API to create a VM in ETAIS – An example

```bash
http -v POST https://waldur.example.com/api/openstacktenant-instances/ \
    name="my instance name" \
    service_project_link="https://waldur.example.com/api/openstacktenant-service-project-link/9/" \
    user_data="version: "1.0"\n" \
    flavor="https://waldur.example.com/api/openstacktenant-flavors/e4c1f185f1964e39840b53581dff728e8/" \
    image="https://waldur.example.com/api/openstacktenant-images/3619fa9b5a2d4cb98ccc3ccbe044f9/" \
    system_volume_size=1024 \n    data_volume_size=1024 \n    internal_ips_set='["subnet":"https://waldur.example.com/api/openstacktenant-subnets/24fd6192a55b4f5e813b33cd15bfbd16/"]' \nAuthorization:"token a2ee5f51ca3c290b960b767e9f5d71f87"
```

Src: https://opennode.atlassian.net/wiki/spaces/WD/pages/23071059/Examples
Direct Client-to-Microservice Communication: Without API Gateway

Very messy communications

- Sensor data preprocess
- Add sensor device
- Sensor list
- Sensor Activity manager
- Data visualization
- Remove sensor device
- Sensor activity service
- Appliances list
Client to microservice with API Gateway

- Sensor data preprocess
- Add sensor device
- Sensor list
- Sensor Activity manager
- Data visualization
- Remove sensor device
- Sensor activity service
- Appliances list
API Gateway

• An API Gateway is a server
• Single entry point into the system
• API Gateway encapsulates the internal system architecture
• provides an API that is tailored to each client

• Responsibilities:
  • Authentication
  • Monitoring
  • load balancing
  • request routing
  • Composition
  • protocol translation
API Gateway

- May invoke multiple microservices and aggregate the data payload
- E.g. Netflix API Gateway gets requests from
  - Televisions
  - set-top boxes
  - Smartphones
  - gaming systems
  - Tablets
Handling large number of services

• Can you remember the name or ID of 50 services?
• How would you figure out if a service is present or not?

Solution:
• Maintain a registry for all the services – Registry Service
• Help to discover a service
Is there any standard for such communication?
Is there any standard for such communication?

- OpenAPI
OpenAPI

• A standard way to describe REST APIs
  • A specification for describing, consuming and visualizing RESTful web services
• Originally known as the Swagger Specification
• OpenAPI Configuration Document
  • Or openAPI definition
  • Contains more than one definition files
  • Uses YAML or JSON
  • Can be validated, tested and ported
OpenAPI

• A standard way to describe REST APIs
  • A specification for describing, consuming and visualizing RESTful web services

• Originally known as the **Swagger** Specification

• OpenAPI Configuration Document
  • *or* openAPI definition
  • Contains more than one definition files
  • Uses YAML or JSON
  • Can be validated, tested and ported
Swagger

- [https://swagger.io/](https://swagger.io/)
- A tool that
  - Simplify API development
  - for users, teams, and enterprises
  - help you *design* and *document* your APIs
- It is open source and professional tool
Swagger/OpenAPI Configuration document

Configuration Document:

• API Endpoint
  • E.g. /user, /upload, /devices, /data, /openstack-instances, /openstacktenant-flavors, /openstacktenant-images, /openstacktenant-instances, etc.

• Operation for each endpoint
  • E.g. GET, POST, PUT, DELETE

• Input Parameter
  • E.g. device_id=sens-897

• Responses and format
Swagger Configuration - Example

```
openapi: 3.0.1
info:
  title: Swagger SmartBuilding App
  description: Smart building application
  termsOfService: http://swagger.io/terms/
  version: 1.0.0
servers:
  - url: https://Devicestore.swagger.io/v2
  - url: http://Devicestore.swagger.io/v2
paths:
  /Device:
    post:
      tags:
        - Device
      summary: Add a new Device to the store
      operationId: addDevice
      requestBody:
        description: Device object that needs to be added to the store
        content:
          application/json:
            schema: $ref: '#/components/schemas/Device'
          required: true
      responses:
        405:
          description: Invalid input
    security:
      - Devicestore_auth:
        - write:Devices
        - read:Devices
        x-codegen-request-body-name: body
```

API definition must include the version of the OpenAPI Specification that this definition is based on.

The `servers` section specifies the API server and base URL.
Swagger Configuration - Example

- **PATH**: The paths section defines individual endpoints (paths) in your API, and the HTTP methods
- Here it will be <url>/Device
- A single path can support multiple operations

**Example:**

```
openapi: 3.0.1
info:
  title: Swagger SmartBuilding App
  description: Smart building application
  termsOfService: http://swagger.io/terms/
  version: 1.0.0
servers:
  - url: https://Devicestore.swagger.io/v2
  - url: http://Devicestore.swagger.io/v2
paths:
  /Device:
    post:
      tags:
        - Device
      summary: Add a new Device to the store
      operationId: addDevice
      requestBody:
        description: Device object that needs to be added to the store
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Device'
            required: true
      responses:
        405:
          description: Invalid input
          content: {}
      security:
        - Devicestore_auth:
          - write:Devices
          - read:Devices
        x-codegen-request-body-name: body
```
Swagger Configuration - Example

```json
openapi: 3.0.1
info:
title: Swagger SmartBuilding App
description: Smart building application
termsOfService: http://swagger.io/terms/
version: 1.0.0
servers:
  - url: https://Devicestore.swagger.io/v2
  - url: http://Devicestore.swagger.io/v2
paths:
  /Device:
    post:
      tags:
      - Device
      summary: Add a new Device to the store
      operationId: addDevice
      requestBody:
        description: Device object that needs to be added to the store
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Device'
            required: true
      responses:
        405:
          description: Invalid input
          content: {}
      security:
      - Devicestore_auth:
        - write:Devices
        - read:Devices
      x-codegen-request-body-name: body
```

**POST**: An operation definition includes parameters, request body (if any), possible response status codes (such as 200 OK or 404 Not Found) and response contents.
Swagger Configuration - Example

```yaml
openapi: 3.0.1
info:
  title: Swagger SmartBuilding App
  description: Smart building application
  termsOfService: http://swagger.io/terms/
  version: 1.0.0
servers:
- url: https://Decicstore.swagger.io/v2
- url: http://Decicstore.swagger.io/v2
paths:
  /Device:
    post:
      tags:
        - Device
      summary: Add a new Device to the store
      operationId: addDevice
      requestBody:
        description: Device object that needs to be added to the store
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Device'
            required: true
      responses:
        405:
          description: Invalid input
          content: {}
      security:
        - Devicestore_auth:
          - write:Devices
            read:Devices
          x-codegen-request-body-name: body
```

**TAG** – used to group operations logically by resources or any other qualifier

These are coming from Tags
Request bodies are typically used with “create” and “update” operations (POST, PUT, PATCH). For example, when creating a resource using POST or PUT, the request body usually contains the representation of the resource to be created.

The Schema Object allows the definition of input and output data types.
Swagger Configuration - Example

```json
1. openapi: 3.0.1
2. info:
3.   title: Swagger SmartBuilding App
4.   description: Smart building application
5.   termsOfService: http://swagger.io/terms/
6.   version: 1.0.0
7. servers:
8.   - url: https://Devicestore.swagger.io/v2
9.   - url: http://Devicestore.swagger.io/v2
10. paths:
11.   /Device:
12.     post:
13.       tags:
14.         - Device
15.       summary: Add a new Device to the store
16.       operationId: addDevice
17.       requestBody:
18.         description: Device object that needs to be added to the store
19.         content:
20.            application/json:
21.               schema:
22.                 $ref: '#/components/schemas/Device'
23.         required: true
24.       responses:
25.           405:
26.             description: Invalid input
27.             content: {}
28.       security:
29.         - Devicestore_auth:
30.           - write:Devices
31.           - read:Devices
32.       x-codegen-request-body-name: body
```

**operationId** is an optional unique string used to identify an operation. Use this value to name the corresponding methods in code.

The Schema Object allows the definition of input and output data types.
Each operation must have at least one response defined, usually a successful response.

- defined by its HTTP status code and the data returned in the response body and/or headers
Swagger Configuration - Example

- The last line is automatically generated by the Swagger CodeGen tool.
- The python file `upload_sensor_data_controller.py` is present inside the `swagger_server.controllers` directory.
- **OperationId** indicates the function name inside the controller/python file (i.e. `upload_sensor_data_controller.py`).
Swagger Editor

Open source tools

- **Swagger Editor**
- **Swagger UI**
- **Swagger CodeGen**
Swagger Editor

- Open source tools
- **Swagger Editor** –
  - Design, describe, and document your API
  - Validate your syntax
  - Instant Visualization
Swagger UI

- Open source tools
- **Swagger Editor**

**Swagger UI** –
- to visualize and interact with the API’s resources without having any of the implementation logic in place.
- Documentation Site generator
Swagger - CodeGen

- Open source tools
- **Swagger Editor**
- **Swagger UI**
- **Swagger CodeGen** –
  - Simplify your build process by generating server stubs and client SDKs for any API
So far...

• How microservices communicate?
• Communication protocols
• RESTful Architecture
• OpenAPI standard
• Swagger tool
So far...

- How microservices communicate?
- Communication protocols
- RESTful Architecture
- OpenAPI standard
- Swagger tool

Do you see any relationship between microservice and containers?
Container and Microservice

• **Microservices** are about the design of software.
• **Containers** are about packaging software for deployment.
Container and Microservice

• You can use a container for hosting a microservice.
• Run microservices within containers.
Container and Microservice

Inner-Loop development workflow for Docker apps

1. Code your app
2. Write Dockerfile/s
3. Create Images defined at Dockerfile/s
4. (Opt.in) Define services by writing `docker-compose.yml`
5. Run Containers / Compose app
6. Test your app or microservices

7. Push or Continue developing

---

LTAT.06.015 : DevOps : Lec 6 - Microservice-based Application Development
Microservice Approach – Limitations

- Development Complexity (whole application)
  - Difficult to deploy
    - Need to manage the deployment of each service (instead of only the bigger application)
- Distributed data management
  - Each service has its own database (private)
  - Maintaining data consistency is a challenge
  - Querying is a challenge
- Debugging hundreds of services
  - Tracing the bugs and errors
- Testing:
  - Need to test each service:
    - Dependencies
    - Communications
    - Input/output parameters
- Communication Interface control

User

SmartBuilding App

Database

Data storage

Data visualization

Data pre-processing

Data analysis

Visualization templates
Microservice Approach – Limitations

- Communication Interface control
- Performance:
  - Communication bottleneck
- Security
  - Communication (APIs)
- Capital Expenditure
- How about the skills
  - Automated testing
  - Automated deployment
  - Automated provisioning
Microservice Approach – Limitations

Types of Headache

Migraine

Hypertension

Stress

Microservices

Src: https://www.dataart.com.ua/media/2813281/y-dolinsky_microservices.pptx
Explore by yourself

• Microservice vs Web service
• Microservice vs Serverless
Lab Sessions

Design of python flask microservices and APIs (tentative)

• Setting up of Swagger UI, Swagger Editor and Grafana services
• Working with Swagger Editor and creating a python-flask microservice
• Design and deploying a python-flask microservices
• Additional python-flask micro services (API versioning
References

- https://microservices.io/
- https://www.nginx.com/blog/refactoring-a-monolith-into-microservices/
- https://solace.com/blog/microservices-advantages-and-disadvantages/
- https://www.bmc.com/blogs/containers-vs-microservices/
- https://swagger.io
- https://developer.cisco.com/learning/lab/what-are-rest-apis/step/1
- https://ois2.ttu.ee/uusois/luus_ois2.ois_public.page?_page=DF9175AB402E2383E097925418BCFB681816956F635EC3A90AEC EB2532C8A79&p_id=0A1A6F159CACB901
- https://www.w3schools.com/whatis/whatis_json.asp
- https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml#http-status-codes-1
- https://eventuate.io/whyeventdriven.html
- https://www.qulix.com/about/monolithic-vs-microservices-architecture
Any Question?

THANK YOU