Services and Microservices

ECE 454 / 751: Distributed Computing

Instructor: Dr. Wojciech Golab

wgolab@uwaterloo.ca

Slides are derived from A. S. Tanenbaum and M. Van Steen,
Distributed Systems: Principles and Paradigms, 2nd Edition, Pearson-Prentice Hall, 2006
as well as
M. Van Steen and A. S. Tanenbaum, Distributed Systems, 3rd Edition, Pearson, 2017.

Learning goals

The objective of this module is to develop a conceptual understanding of the following:

- services and APIs
- Service-Oriented Architectures (SOAs)
- Microservices
- SOAP
- REST

Services

- A **service** is a self-contained and reusable component that performs a specific business task. For example, a service could return the current price of a stock, or append a message to a blog.
- A service provides **encapsulation**: it exposes an interface, and hides its implementation. Internally, a service may be implemented using smaller components arranged in a layered or object-oriented fashion.
- A service is accessed through a communication protocol that may be layered on top of another protocol for transport.
- Services that use Web protocols (e.g., HTTP) for transport are called **Web services**.

Service-oriented architectures

- A **service-oriented architecture (SOA)** is one where functionality is separated into a collection of services that can be developed, deployed, and maintained independently (e.g., by different development teams).
- SOAs often include middleware infrastructure, such as an Enterprise Service Bus (ESB), to facilitate integration of services. In that sense, SOAs resemble object-based architectures.
- Interactions with services usually follow the client-server model, which is a central concept in layered architectures.

SOA example

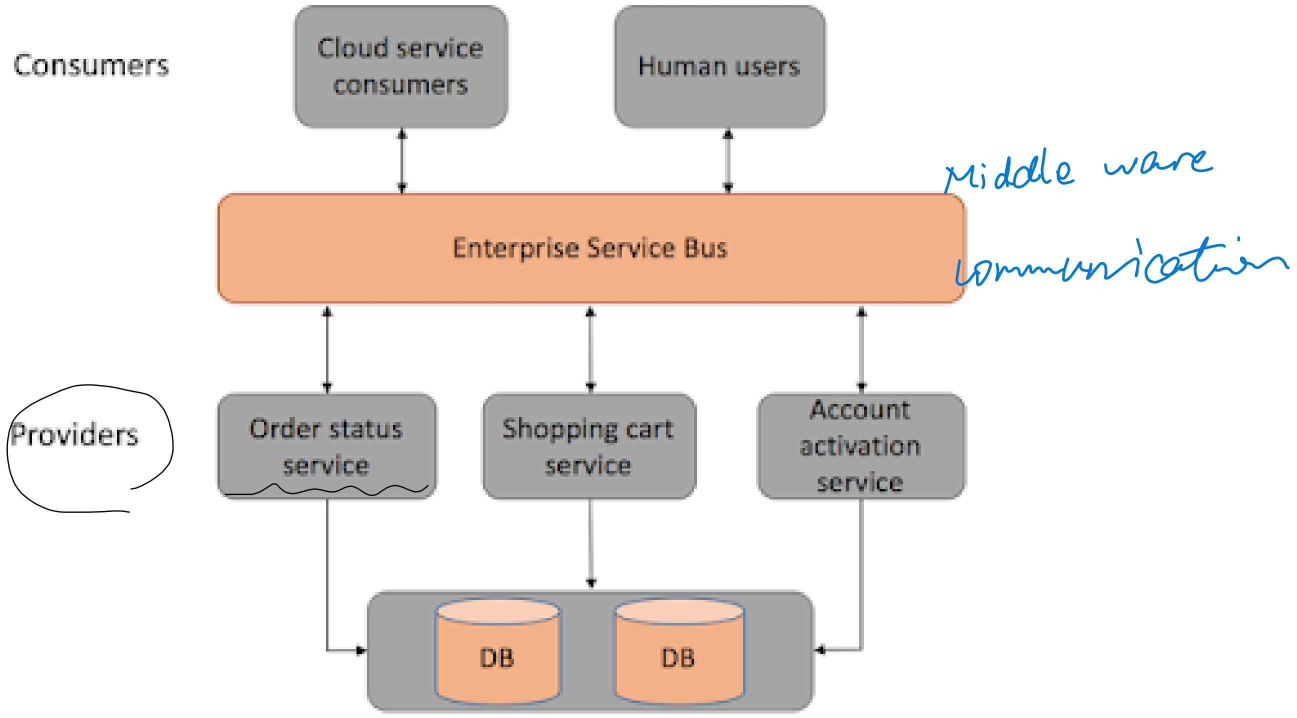

Image credits: Ima Miri
https://dzone.com/articles/microservices-vs-soa-2

Microservices

- Microservice-based architectures take the divide-and-conquer strategy of SOAs to the next level. They are characterized by more fine-grained and loosely coupled components, as well as lighter communication protocols.
- Microservices do not rely on a common communication mechanism (e.g., ESB), and tend not to share data. This increases fault tolerance but complicates coordination among components.
- Microservices are easier to develop, deploy, and maintain independently than larger components in conventional SOAs. This helps scalability.
- A poorly-designed microservices-based architecture can be messy and inefficient. Many business problems can be solved perfectly well using coarse-grained SOAs (or even monolithic applications).

Microservices example

A_0. A_1. A_3 基于此

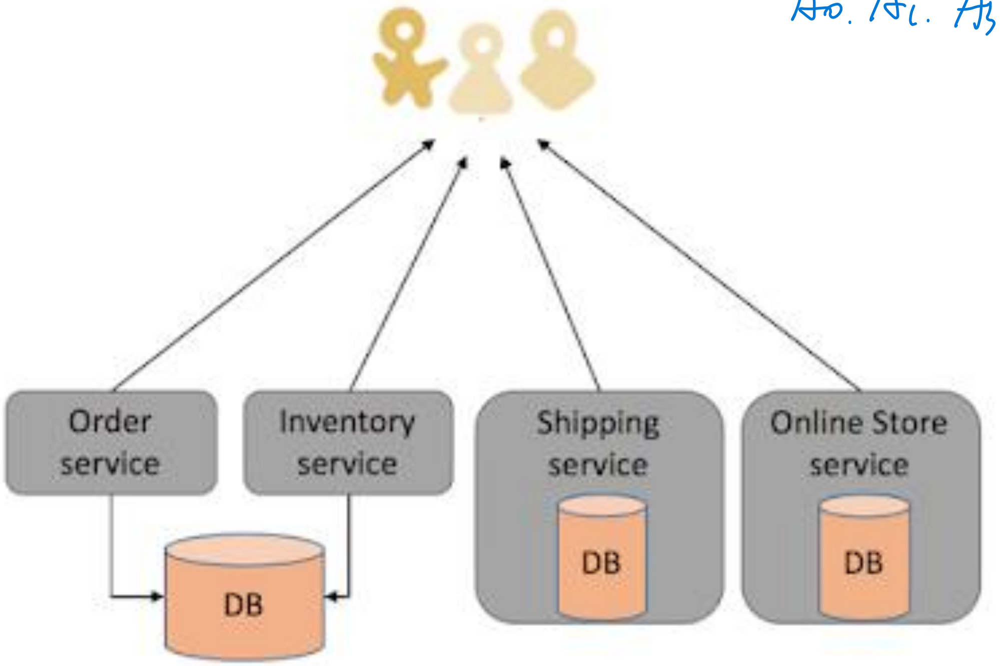

Image credits: Ima Miri
https://dzone.com/articles/microservices-vs-soa-2

SOAP

- The Simple Object Access Protocol (SOAP) was the first widely-used protocol for communication among services in an SOA. It was designed in the late 1990s, after the birth of the World Wide Web (WWW).
- SOAP is usually layered on top of HTTP, and for that reason tends to play well with existing firewalls and proxies. This is important for accessing services remotely over the Internet.
- SOAP uses XML as the message format, which is extensible and human-readable, though somewhat tedious for developers.
- The weaknesses of SOAP include large message size and parsing overhead.

a little bulky

SOAP example from Wikipedia (https://en.wikipedia.org/wiki/SOAP)

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:m="http://www.example.org">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice>
      <m:StockName>T</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

REST

- Simple Web services can be constructed by using the HTTP protocol directly to issue requests, rather than using HTTP to transport a higher-layer protocol like SOAP.
- **Representational State Transfer (REST)** does exactly that by providing access to remote resources using HTTP methods, such as GET, PUT, and POST. This idea was developed at roughly the same time as SOAP.

don't remain session state

- REST prescribes a model of interaction, particularly stateless execution, in addition to defining the format of messages. This makes REST a distinct architectural style, and differentiates it from SOAP, which is merely a protocol.

SOAP = only post, don't get

- In REST, responses to GET requests can be cached using existing HTTP mechanisms. This is not the case for SOAP, which transports requests using non-idempotent HTTP POST messages.

RESTful architectures

- REST-based Web services, and architectures built upon such services, are called **RESTful**. The textbook also refers to such architectures as **resource-based**.
- RESTful architectures have four key characteristics:

  1. Resources are identified through a single naming scheme, namely Uniform Resources Identifiers (URIs).
  2. All services offer the same interface, consisting of at most four operations (PUT, GET, DELETE, POST).
  3. Messages sent to or from a service are fully self-described (e.g., they indicate the resource and content encoding).
  4. After executing an operation at a service, that component forgets everything about the caller.

- The fourth property is called **stateless execution.**

track: cookie & variables

Stateless execution in REST

- HTTP is stateless, meaning that it does not explicitly keep track of users and sessions. Applications must use cookies and hidden form variables to compensate for this, and even then the session state is managed by the application server and not by HTTP itself.
- RESTful Web services are similarly stateless because they use HTTP directly.
- Example: A RESTful service can retrieve a specific image from a gallery, but it cannot retrieve the "next" image because it does not remember the last image accessed by a user in the same session.

JSON

- The parameters of a RESTful request and its response can be encoded in a variety of formats, but many developers prefer JavaScript Object Notation (JSON).
- The JSON data format follows a schema, similarly to XML, but is more compact and easier to parse. This is helpful for debugging complex distributed systems.
- JSON defines a small collection of basic data types: Number, String, Boolean, Array, Object, and null.
- A JSON object is a collection of key-value pairs, where keys must be strings.
- The official Multipurpose Internet Mail Extensions (MIME) type for JSON text is "application/json".

JSON example from Wikipedia

(https://en.wikipedia.org/wiki/JSON)

```
{
  "firstName": "John",
  "lastName": "Smith",
  …
  "age" : 27,
  …
  "children": [],
  "spouse": null
}
```

REST + JSON example

Log in to an ecelinux server and execute the following commands to access a RESTful service:

```
curl -v https://www.bookmarks.dev/api/version
curl -s https://www.bookmarks.dev/api/version
curl -s https://www.bookmarks.dev/api/version | python -m json.tool
```

Try again, but this time enter the URL directly into your Web browser.

REST + JSON example

The first command produces output similar to the following:

```
> GET /api/version HTTP/1.1
> Host: www.bookmarks.dev
> User-Agent: curl/7.68.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: nginx/1.12.0
< Content-Type: application/json; charset=utf-8
< Content-Length: 73
<
{"version":"13.0.0","gitSha1":"edda301514992679d6cee3af4d9bf65aa03c5582"}
```

Summary

- Service-oriented architectures (SOAs) combine properties of the four architectural styles discussed in the previous lecture.
- Microservices are fine-grained, loosely-coupled services.
- Many services use Web protocols, though we will learn later on in the course that binary protocols may be preferable for back-end infrastructure services.
- SOAP is an XML-based protocol for accessing services that is often transported over HTTP.
- REST is a way of using HTTP directly to access services, and specifies a stateless model of interaction. It defines a distinct architectural style.
- RESTful services tend to use JSON, which is more compact and easier to parse than XML.