Applications for cloud computing

SJK005 – Cloud computing

University Master's Degree in Intelligent Systems

University Jaume I

Competences and learning outcomes

1. To know the meaning of Cloud Native Applications.
2. To know what Service Oriented Architecture is.
3. To know what are microservices and its importance for developing Cloud Native Applications.
4. To know RESTful architectural style and how to follow its principles.

Contents

1. Cloud Native Applications definition.
2. Service Oriented Architecture (SOA).
3. Microservices.
4. RESTful services.
5. Summary.

Coud Native definition

According to Cloud Native Computing Foundation:

"Cloud native technologies empower organizations to build and run **scalable applications** in modern, dynamic environments such as **public, private, and hybrid clouds**. **Containers**, service meshes, **microservices**, immutable infrastructure, and **declarative APIs** exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil."

Coud Native definition

Another operational definition:

"Cloud native applications are SaaS applications based on a **modular architecture** where **microservices** are managed in **containers**"

Service Oriented Architecture

Following the definition of The Open Group:

Service-Oriented Architecture (SOA) is an architectural style that supports service orientation.

Service orientation is a way of thinking in terms of services and service-based development and the outcomes of services.

Ref: https://collaboration.opengroup.org/projects/soa-book/pages.php?action=show&ggid=1314

Service Oriented Applications

Characteristics of a service:

1. It logically represents a repeatable business activity with a specified outcome.
2. It is self-contained.
3. It is a black box for its consumers, meaning the consumer does not have to be aware of the service's inner workings.
4. It may be composed of other services.

Ref: https://collaboration.opengroup.org/projects/soa-book/pages.php?action=show&ggid=1314

Microservices

Microservices definition:

"Microservices refer to a style of application architecture where a collection of independent services communicate through lightweight APIs.

A microservices architecture is a cloud-native approach to building software in a way that allows for each core function within an application to exist independently. "

Fuente: https://www.redhat.com/en/topics/microservices/what-are-microservices

Microservices

Characteristics of microservices:

1. Application development simplified.
2. Code changes will be small and of low complexity.
3. Scalability.
4. Microservices are fully tested and validated.

Microservices

Microservices isolates a specific business function.

Assuming we are involved in developing and e-commerce application:

1. Registering new users.
2. Adding new products to the catalogue.
3. Reporting stock products.
4. Clients' purchases.
5. Listing products by category/type/price, etc.
6. Any other ideas?

Microservices

All the above examples can be created, deployed, scaled and maintained in independently of each other.

The services for registering new users has no dependence on the services for reporting stock of products.

The service for reporting stock of products works together with the clients' purchases service (probably sharing some database), but can be created, deployed, scaled and maintained independently.

Microservices

From a technical point of view, microservices encapsulate all dependencies, libraries and environmental requirements.

Later, this will help us to isolate a running microservice in a container.

Microservices

A consistent and well-designed API is crucial when designing microservices.

In this course, we are going to develop microservices following the RESTful architectural style.

This style encourages to make a proper use of the HTTP protocol, the protocol the Web is based on.

RESTful services

RESTful services are web services that make comprehensive use of the HTTP protocol.

RESTful web services are based on four **concepts** and four **properties**.

RESTful web services define a coherent and complete API (Application Programming Interface), the set of operations that a web service makes available and any details (format, HTTP verbs, headers, URIs, etc.) provided by the service.

RESTful services

Concepts:

1. Resources.
2. URIs.
3. Representation.
4. Links.

RESTful services

Properties:

1. Addressable.
2. Stateless.
3. Connection.
4. Uniform interface.

RESTful services (concepts)

Resources:

A resource is a piece of information.

A resource is any element that has sufficient entity to be referenced, that is, to be linked to.

Some examples:

- My company's welcome page.
- The list of all my company's contacts.
- A picture of my vacation.
- A video.
- A monthly mortgage payment calculator.

RESTful services (concepts)

URIs:

URIs are the mechanism that makes my resource visible on the web.

A URI is like a pointer to a resource.

The same resource can have more than one URI pointing to it.

A URI cannot point to more than a single resource.

URIs should be as descriptive as possible:

- http://www.uji.es/agenda/entradas/ is clear.
- http://www.uji.es/1/as2/ is cryptic.

RESTful services (concepts)

Representation:

The same resource can have more than one representation.

The list of all the entries in my address book is a resource. This resource can be represented in XML format, as a spreadsheet, as a table, or even as a pdf. In all cases the resource is the same, what changes is its representation.

RESTful services (concepts)

Links:

One of the features that have made the Web a successful technology is the fact that it is possible to establish links between different resources.

On a web page we can include a link to another web page, or to an image, or to any other resource accessible on the web.

It is important that the representation of my resources contain links to other resources.

After all, this is the meaning of Hypertext.

RESTful services (properties)

Addressable:

As we have seen, resources and the URIs pointing to them are fundamental concepts on the Web (and in REST applications).

A RESTful application must expose all resources of interest through URIs.

This is the meaning of addressable, the resources of my application are accessible through the URIs I have defined for it.

RESTful services (properties)

Stateless:

HTTP is a stateless protocol.

A server has no memory (state) between two successive requests. Each request is seen as complete and independent of any other.

That HTTP is a stateless protocol does not mean that our applications are stateless. In fact, REST stands for Representational State Transfer, the state of the application (its representation) is transferred between the client and the server to give rise to stateful applications.

RESTful services (properties)

Connection:

This property is directly related to the concept of Addressability.

One of the success factors of the web is its navigability, the resources exposed on the web can be navigated thanks to the links that exist between them.

The web is a set of connected, navigable resources.

RESTful services (properties)

Uniform interface:

Whatever the resource, we always have the same set of methods to access any resource.

Each of the **HTTP methods** has well-defined semantics. Our applications must always follow the semantics of the HTTP methods.

For example, GET is used to read a resource. Our application should not use GET to modify the state of a resource.

RESTful services (properties)

Uniform interface:

Method	Operation
POST	CREATE
GET	RETRIEVE
PUT	UPDATE
DELETE	DELETE

We map HTTP methods to CRUD operations.

RESTful services (properties)

Uniform interface: **GET**.

- The GET method is used to retrieve a resource.
- GET is idempotent, secure and cacheable.
- The resource is indicated by a URI.
- Parameters, such as those coming from a form, can be included in the request.
- Language preferences, or the format of the response, can be included in the request headers.
- The body of the response includes the requested resource.
- The response headers include information about the language used, the format, or the length of the response in bytes.

RESTful services (properties)

Uniform interface: **POST**.

- The POST method is used to create or modify a resource.
- POST is neither idempotent, secure, nor cacheable.
- The resource to be served by the request is indicated in the URI of the request.
- The body of the request contains the entity in some format.
- The request headers may include language preferences, or the format of the response.
- The response body includes a representation of the resource.
- The response headers include information about the language used, the format, or the length of the response in bytes.
- Important, if we are creating a new resource on the server, the Location response header will contain the URI of the newly created resource.

RESTful services (properties)

Uniform interface: **PUT**.

- The PUT method is used to create or update resources on the server.
- The URI that receives the request represents the URI of the resource to be created or updated.
- The status code can be either 200 Ok or 204 No Content.
- If a 200 Ok code is returned, the response body must contain a representation of the resource created.
- If a 204 No Content code is returned, the response code will be empty.

RESTful services (properties)

Uniform interface: **DELETE**.

- The DELETE method is used to delete resources from the server.
- The resource to be deleted is identified by the URI that receives the request.
- The body of the request is empty.
- The DELETE method is idempotent.
- The DELETE method is neither secure nor cacheable.

RESTful services

Recipe to create a RESTful API:

1. Determine which are the resources of your application.
2. Determine which are the valid representations for your resources.
3. Define URIs to access them.
4. Define the methods that can act on each of the resources through their URIs.
5. Determine the links between the different resources and include them in their representations.

RESTful services

Enough theory.

How can we implement this?

Let's move to practice.

RESTful services

Let's assume that we are developing a (simple) contact list application.

- We want to:
- List all my contacts.
- List the details of a contact.
- Add new contacts.
- Update existing contacts.
- Remove an existing contact.

RESTful services

Let's follow the recipe for creating RESTful web applications:

1. Determine which are the resources of your application.

The only resource will be any contact in my contact list.

2. Determine which are the valid representations for your resources.

Json is the native web format.

3. Define URIs to access them.

/contacts => URI for accessing all contacts.

/contacts/{id} => URI for accessing one contact.

RESTful services

Let's follow the recipe for creating RESTful web applications:

4. Define the methods that can act on each of the resources through their URIs.

Method	URI	Action
GET	/contacts	Retrieve all contacts.
GET	/contacts/{Id}	Retrieve one contact.
POST	/contacts	Create a new contact.
PUT	/contacts/{Id}	Update an existing contact.
DELETE	/contacts/{Id}	Remove {Id} contact

RESTful services

Let's follow the recipe for creating RESTful web applications:

5. Determine the links between the different resources and include them in their representations.

At this point, we are not going to use links.

Summary

Cloud native applications are modular applications, developed as microservices and encapsulated in containers.

Microservices are applications that provide a stable, well-defined API accessible through a network (Internet) which execute in the cloud.

RESTful services make intensive use on the HTTP protocol. The recipe we have seen is very useful to define a RESTful API.

We are going to see how to develop applications, as microservices which can be encapsulated in a container to be deployed in the cloud.