Microservices. Microservices with WSO2

July Cejas
Chakray Architect

The purpose of this paper is to describe how the approach, Microservices, called "light SOA" can be implemented Through the WSO2 Lean Enterprise Middleware.

Index

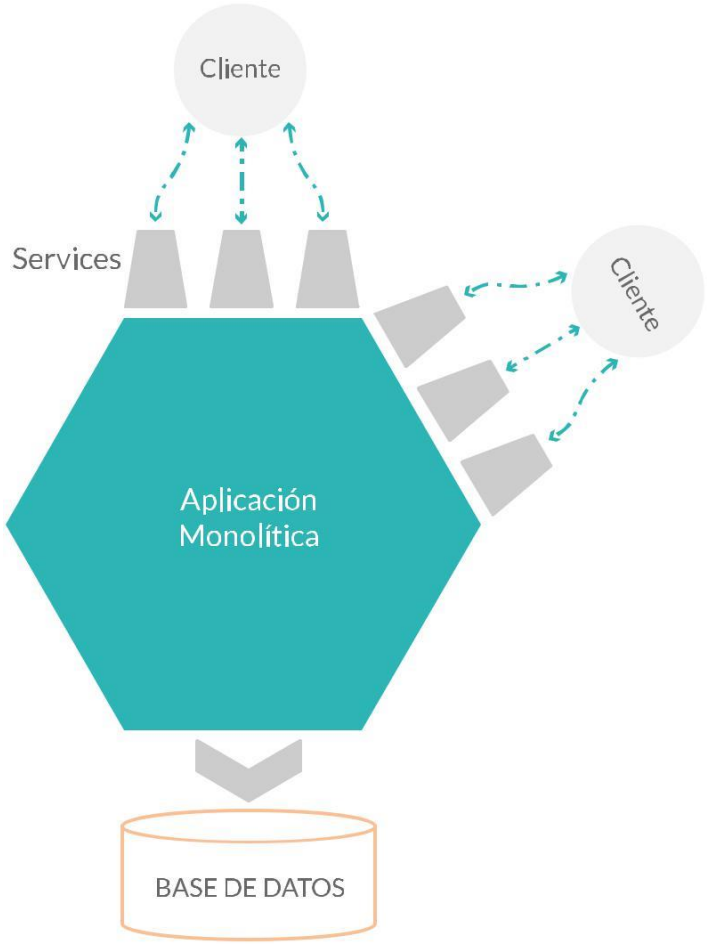
Introduction

Microservices have been breaking into the SOA discipline by its pragmatic, agile approach; its narrow focus on business and not on technology. The purpose of this document is to describe how this so-called "light SOA" approach can be implemented through the WSO2 Lean Enterprise Middleware. In the following sections, we will make a small immersion in the concept of microservice, some reflections and conclusions.

A little story

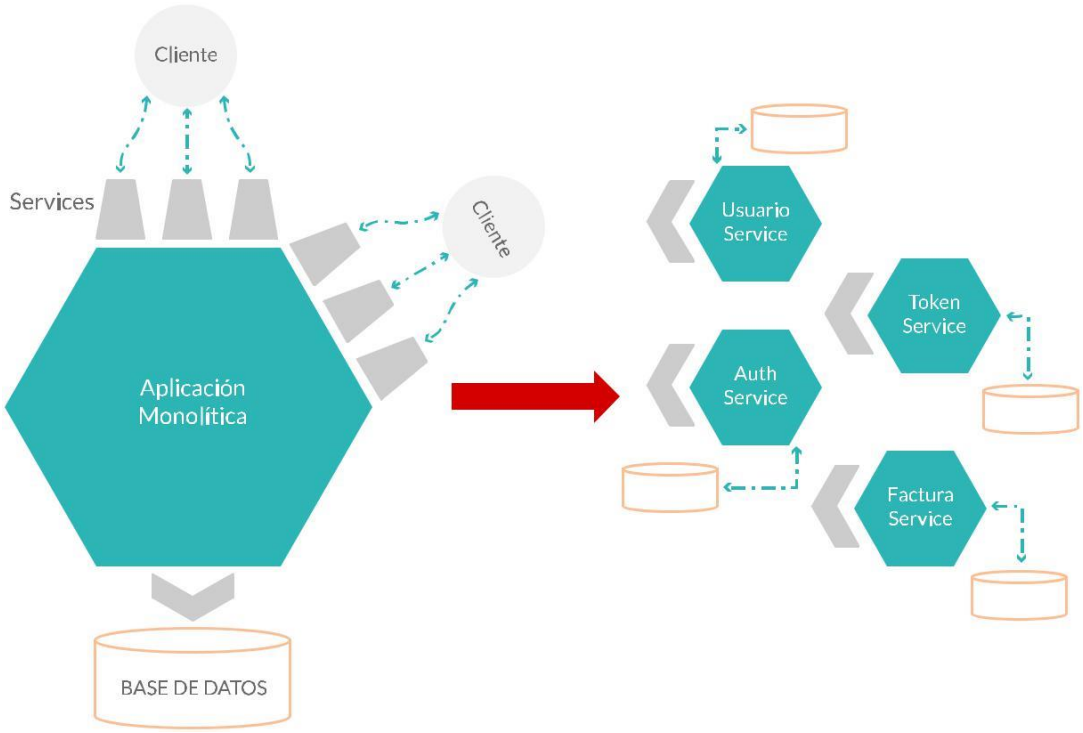
Usually when we start major software projects we start with the identification of vocabulary associated with determining context or domain, are converted into an information model and boarded their relationships later. It is very common to have a really great domain model is then performed using a relational database model. On this model, posed all information needs, relationships, persistence and services. As it grows in this model are generated problems in maintaining relationships, dependencies, performance and capabilities change; especially when growing demands of the system and the existence of new requirements. Generally, if we develop a service for the details of an invoice, We use an underlying model with many relationships to other tables; this model many agencies are established; easily recognizable when a small change in an attribute of a table involves changing many artefacts of software, or download and upload a complete solution; essentially a small change impacts the entire solution.

For years, software development has been approached with this standard approach: establishment of a domain model that represents the vocabulary of a business and its relationships; representing it in all its extension. Microservice comes to a different approach.



What it is a Microservice

In essence, a microservice is a software artifact that is designed to prevent it from being dependent on a large and heavy relationship model; Which must fulfill a specific business function; In addition to having a simple implementation and thought in the integration with others. Its objective is to promote the development of individual components that are capable of evolving and climbing independently. In this approach, contracts are tailored to the needs of the business and not to a domain model and / or a large relational data structure. Their approach is to avoid creating fragile and complex solutions.



In the diagram above it can be seen how a monolithic application can be divided into independent services that fulfill a specific business function. The main features of microservices are listed below:

01 Services have a micro responsibility with a business-minded approach.

02 Live within a container with an independent life cycle (executed in its own process).

03 They are organized vertically on business capabilities and not on technology.

04 They are designed under the principle of low coupling.

05 They are developed with choreography in mind and not orchestration.

06 They are designed to withstand failure (fault tolerant).

07 Its implementation models and its governance are carried out in a decentralized manner.

08 Their implementation models are not shared, for example their domain models and the persistence model are unique and individual.

09 Communicate about lightweight mechanisms such as HTTP or JMS.

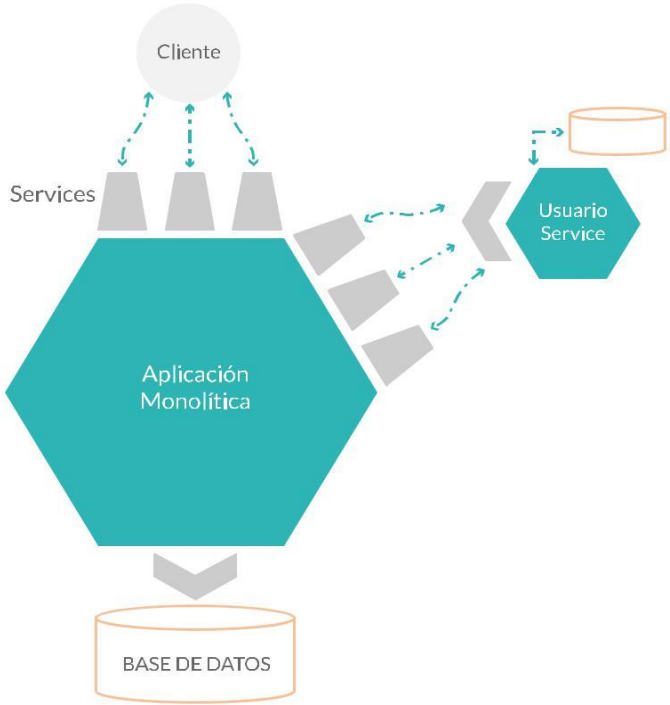
10 They grow individually.

11 They can be self-scaled individually.

About Monolithic Applications

T The development of monolithic architectures will remain an alternative for the development of solutions and will depend on the degree of distribution, size, integration needs and mediation of the solution, to opt for a microservice architecture. In essence, it is important to emphasize that not every solution should be approached under a microservices approach.

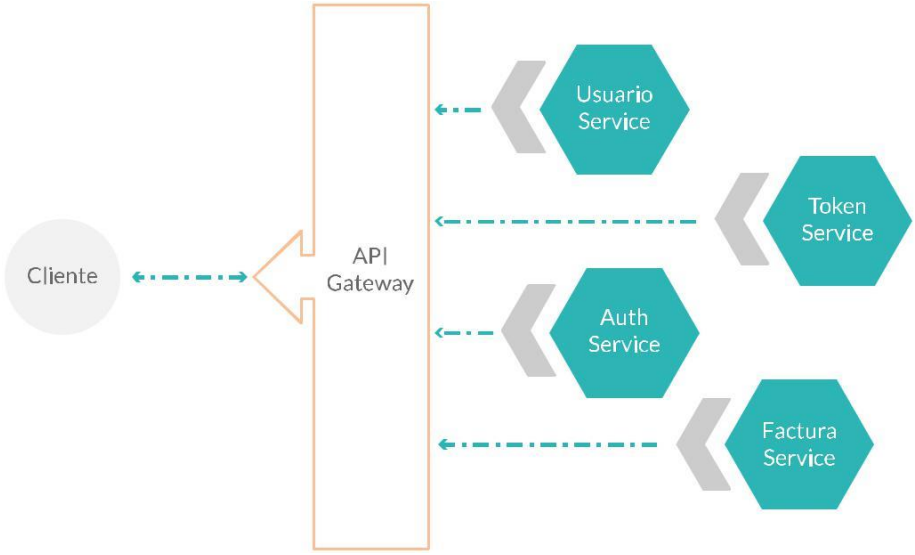
Monolithic applications can be built with a useful modularity, avoiding the complex areas microservices introduce into any architecture. If the limits of the solution can be managed, they are small and there is no need to build a distributed architecture; The solution can be approached under monolithic application strategies. In addition, a monolithic architecture and microservice can be combined taking advantage of the best of both approaches.



A specific recommendation is to start with a monolithic architecture and then break it down or divide it into microservices. With this approach you can explore, discover and exploit the limits and extensions of the solution, establishing a more solid route map for microservice design and architecture.

Microservices and an API Gateway

In Microservice-based architectures it is advisable to use a gateway known as API Gateway. The gateway is responsible for requesting, routing, composing, and translating protocols. All customer requests are directed first to the gateway and then invoked the appropriate microservice. The Gateway often handles requests to multiple microservices to aggregate their results.



The general recommendation is that customers do not directly invoke the microservices, but through a Gateway that allows simplifying and adjusting the requirements to the functional and specific needs of the application. The API Gateway encapsulates the internal system architecture by providing an API that suits each client. The Gateway may have other responsibilities, such as authentication, control, load distribution, caching, manipulation responses, among others.

The API Gateway provides:

01 A single access point.

02 Hides the details associated with the invocation of linked or related microservices.

03 Decreases requests or invocations services through the network.

04 Simplifies client access proxies.

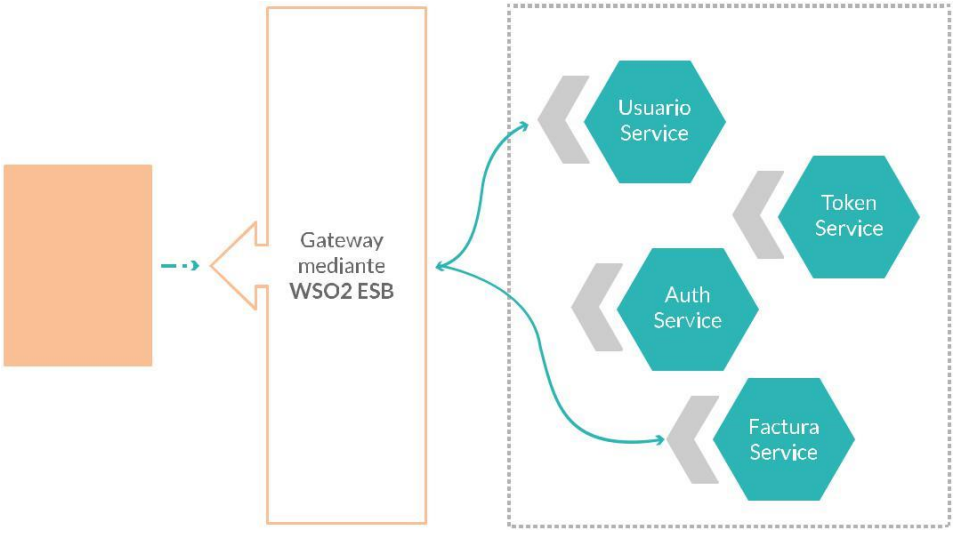
05 Provide a personalized API.

Microservices in WSO2

WSO2 is a mature stack that provides flexible and lightweight deployment models to support the most important and relevant features of microservices. In the next sections I will describe the strategies and patterns that can be incorporated into your WSO2 initiatives to support this approach.

WSO2 ESB pattern as API Gateway Microservice

An Enterprise Service Bus (ESB) is an infrastructure that can perform mediation, routing, enrichment, and policy incorporation services on web services and other artifacts. These features can be used to implement a Gateway for MicroServices architectures. In essence, in this pattern, we use the WSO2 ESB and all its features to deploy a Gateway for our microservices platform.



The WSO2 ESB Gateway can provide a mechanism that allows access to individual microservices from applications by forming a gateway, providing a single point of entry for all clients. This Gateway can be distributed and support high availability (HA) and clustering; It is important to note that this ESB can only be used for Gateway tasks. The following are the functions that this ESB must fulfill as a Gateway:

01 Isolate customers on the logic of integration, composition, among microservices.
02 Isolate the client from the microservices location.
03 Provide an optimal API for each client.

04 Reduce the number of requests (round trip).

05 Simplify the invocation or call to multiple microservices from the client.

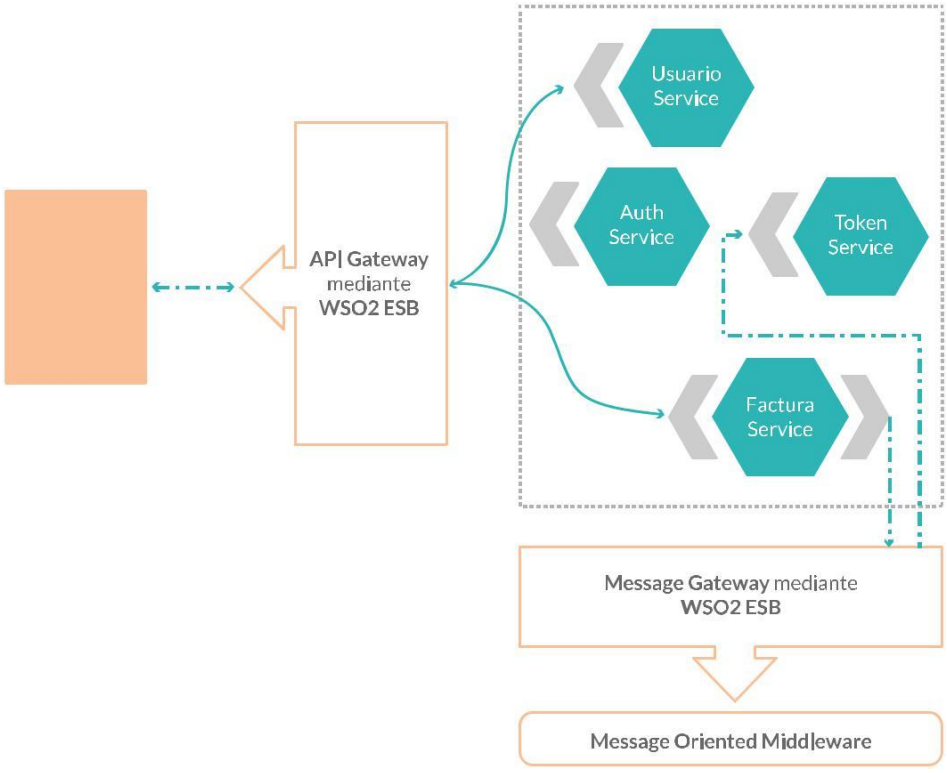
06 Incorporate policies of caching, authentication, authorization, among others.

07 Exposing APIs tailored to customer needs.

WSO2 allows the creation of a Gateway for MicroServices-based architectures that can scale independently, coordinated and managed over HA and clustering; They can also be established in containers such as Docker to manage an independent and clustered life cycle of Linux containers such as Kubernetes.

WSO2 ESB pattern as Message Gateway Microservice

Using the WSO2 ESB as a gateway and limiting its functions to microservices architectures, we can implement two types of gawatey, one for APIs and one for messages. One of the characteristics of microservices is its ability to communicate over light mechanisms such as HTTP or JMS. A recommended design pattern is the design of a messaging gateway that can enrich and include policies for messages that are exchanged between microservices when using JMS as a communication standard. On this approach we can use a WSO2 ESB as a message mediator (JMS messages).

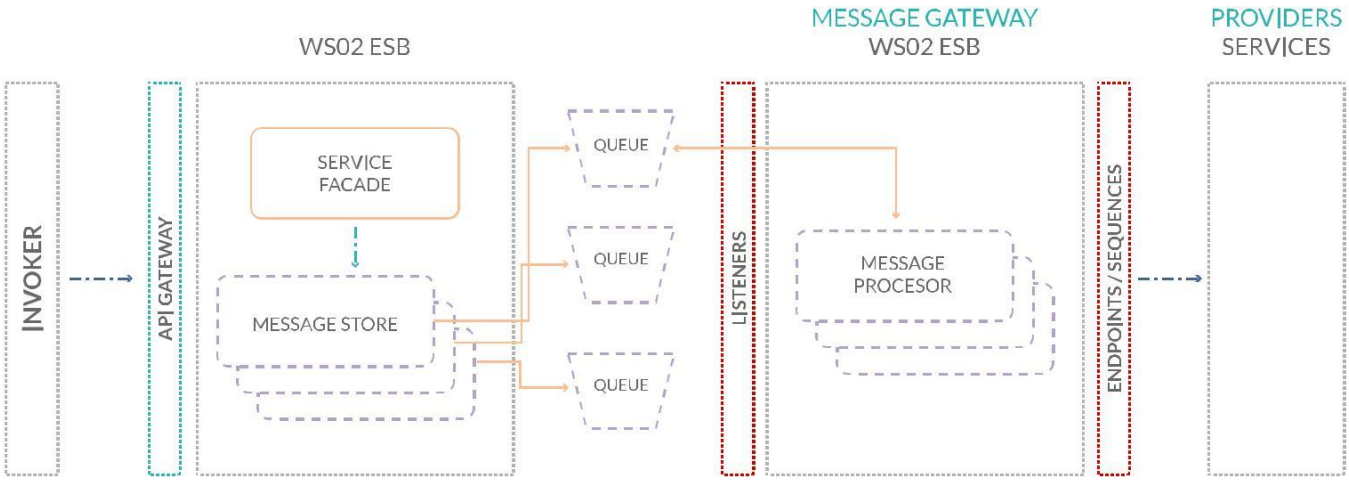


Under this pattern, we can use the WSO2 ESB's ability to manage message storage using the Message Stores (Store Mediator) functionality to store messages temporarily. Currently the Store mediator can be used to store messages in memory, in jms or implementations that can be customized like the use of REDIS, among other technologies. Another component that can be used in this pattern is the message processor, which can consume messages from the message stores and process them. In this processing, messages can be sent to stored and queried endpoints on a record.

WSO2 allows the creation of a Messaging Gateway for MicroServices based architectures that can scale independently, coordinated and managed over HA and clustering; Likewise can be established in containers like Docker. It is important to note that this ESB can only be used for Gateway Message tasks.

WSO2 ESB pattern as API Gateway and Message Gateway

The following graph shows the inclusion of both approaches:



In summary, WSO2 ESB can be used as a component for the conformation of API Gateways and message Gateways; Establishing a single access point for microservices based architectures. Gateways can develop policies for validation, security, cache, delivery guarantee, fault tolerance, logging, audit, among others. Under this approach WSO2 ESB assumes and develops different roles, limiting and standardizing its function in architectural contexts.

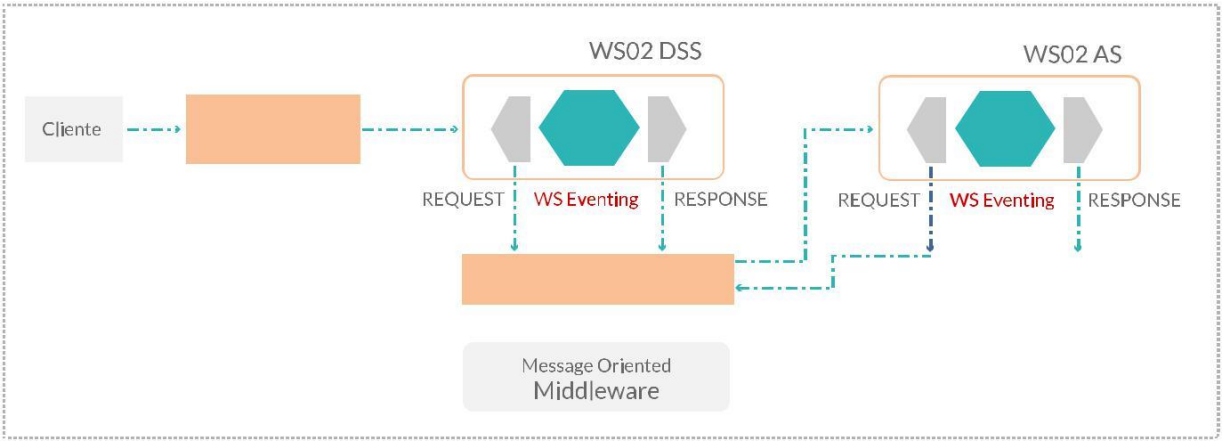
WSO2 pattern Stack and WS-Eventing support for Microservice

WSO2 provides the ability to develop data, decision and domain specific services under various products in your stack. Data services (WSO2 DSS) can respond to the needs of the persistence layer and can communicate with other services in an independent and highly decoupled form based on JMS transport. WSO2 DSS supports the WS-Eventing standard that can be used to trigger events during the occurrence of certain conditions in the data, both in the request and in its response. The event-trigger contains a subscription that any client can use to receive notifications sent from a topic to a messaging queue or even email.

In essence, WSO2 DSS, BRS, AS provide support for the WS-Eventing standard, enabling us to enable communication between services asynchronously. This approach allows services to be decoupled using a lightweight message-based communication style such as JMS. This approach allows, for example, to set endpoints such as the following:

```
jms: / queue Ordertopic transport.jms.DestinationType = & transport.jms?.
ContentTypeProperty = Content-Type & java.naming.provider.url = tcp: // localhost: 61616 & java.naming.
factory.initial = org.apache.activemq.jndi.ActiveMQInitialContextFactory & transport.jms.Connection FactoryType =
Queue & transport.jms.ConnectionFactoryJNDIName = QueueConnectionFactory
```

Likewise, each data service can be exposed on the JMS transport. In the following diagram it can be seen that events can be injected into the request or response that can be triggered by the occurrence of an event, supported by Xpath and WS-Eventing. The payload or payload can be sent to an internal topic and later to a JMS Queue / Topic through an established subscription. In the same way can be used listener for the sending to other queues / topics.

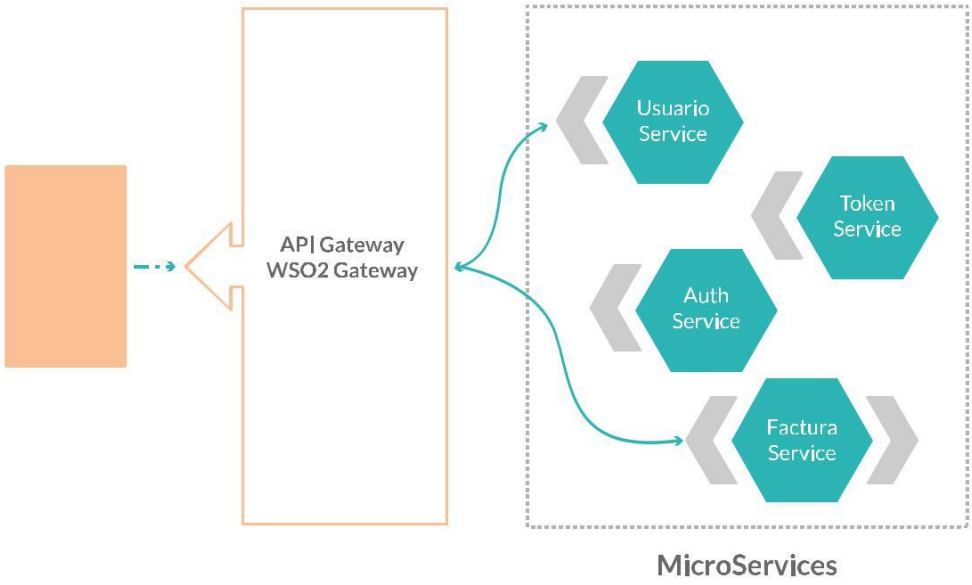


It is important to emphasize that the use of a Facade layer (Gateway) is recommended to avoid the proliferation of point-to-point connections between services.

WSO2 WSO2 pattern Gateway GW to Microservices

WSO2 is currently developing a more focused and targeted Gateway for Facade pattern support and microservices. In this scenario, the WSO2 ESB (limiting its functions) would not be used as a gateway. WSO2 Gateway (WSO2 GW) is a high-performance Gateway, a lightweight gateway for messaging, focused on Gateway / Facade pattern implementation. Its goal is to encapsulate messages between heterogeneous systems with diverse technologies, protocols and standards.

WSO2 MSS will be the gateway of messaging that will serve as Gateway for products such as WSO2 Enterprise Service Bus, WSO2 Security Gateway, among others.



Among the most relevant characteristics:

01 High performance and low latency gateway.

02 10 times faster than HTTP transport currently present in WSO2 ESB.

03 Support thousands of concurrent connections.

04 Routing based on Apache Camel and Spring Framework.

05 Load balancing and failover (Circuit Breaker / Kill Switch)

06 Improved support for error handling.

07 REST / RESTful Camel DSL based APIs.

08 Routing statistics through JMX.

With this approach you can establish RESTfull services directly, for example:

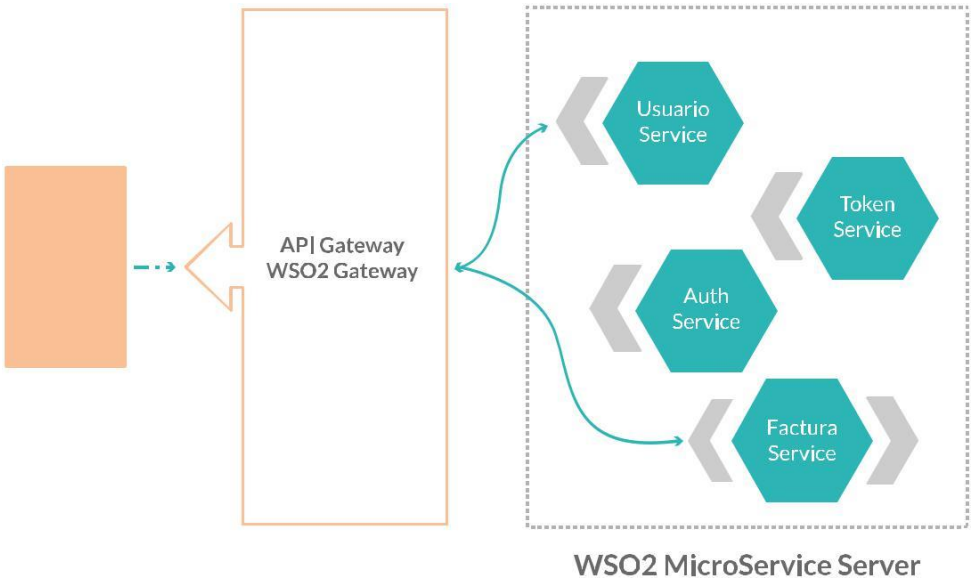
```
<Rest path = "/ iot">
      <Get uri = "/ device"> <to uri = "direct:
       getDispositivos" /> </ get>

      <Get uri = "/ device / {id}"> <uri = to "direct:
     getDispositivosById" /> </ get> </ rest>
```

For example set endpoints such as the following:

```
<Route>
      <From uri = "direct: getDispositivosById" />
      <recipientList>
         <Simple> iot: http: // iot / posts / $ {header.id} </ Simple> </
   recipientList> </ route>
```

WSO2 Microservice pattern with MSS

WSO2 is developing a more isolated from the ESB model for those implementations that require the implementation of microService paradigm through strategies such as Non-blocking IO, processing and concurrent programming based on Netty, Pattern Disruptor, Java 8, Reactive Programming, among others; improving the current limitations of ESB approach and role-based approach. The product is called WSO2 Microservice Server (WSO2 MSS).



WSO2 MSS allows the construction and delivery of applications aimed at microservices on REST services, offering a high-performance architecture (runtime and start) and ensuring low resource consumption. WSO2 MSS has incorporated management metrics and analysis through integration with WSO2 Data Analytics Server (WSO2 DAS), which offers a complete solution for data analysis. Each microService is developed for a single purpose and is deployed independently, which will ensure scalability and reliability.

Among its most important features:

01 Fast execution time.

02 Supported on the WSO2 Carbon 5.0 kernel.

03 Low memory consumption.

04 Simple development model for implementation and monitoring.

05 Using WSO2 Developer Studio for development, including defining their Swagger API through.

06 WSO2 Data Integration Analytics Server for data analysis.

07 Monitoring requests through a unique identifier.

08 Netty 4.0 based transport.

09 JWT based security.

10 Support custom interceptors.

11 Streaming support input and output.

With WSO2 MSS now you can record services. Among the most important notes:

ANNOTATION	PURPOSE
@HTTPMonitoring	Annotation that will allow the microservice to send the detailed information of its behavior at runtime to the WSO2 DAS, for example: startNanoTime, serviceName, serviceClass, responseTime, requestSizeBytes, httpMethod, requestUri, among others.
@Metered	Annotation that allows a metric to measure the rate of events at a given time.
@Timed	Annotation that allows a metric to measure the duration of execution of a particular business function.
@Counted	Annotation that allows a metric to increase or decrease a counter.

To set a microService:

```
@path ( "/ micro") @HTTPMonitoring
MICROSERVICE {public class ...


@GET
@Path ( "/ device / {id}")
@Metered
public int getDispositivo (@PathParam ( "id") int id) {return
 persistence.getDispositivo (id); }
```

To start a microService, you can use:

```
MicroservicesRunner new ()
        .deploy (new MICROSERVICE ())
        . start ();
```

They can be placed interceptors that can manage validation policies, security metrics among others; for example:

```
.addInterceptor (new MetricsInterceptor (MetricReporter.CONSOLE,
MetricReporter.JMX, MetricReporter.DAS))
```

WSO2 MSS is supported by the following technologies: Carbon server, IO Netty, Netty jaxrs Http, GSON, imbusDS Java library; among other Open Source frameworks. Importantly it supports JSON Web Token for the safety of REST services.

General patterns WSO2 ESB Microservices

WSO2 Microservices and Partial Failure

Generally when the Gateway must compose or orchestrate microservices that are distributed it can face a "partial failure" when it invokes a service that responds slowly or is not available. Generally the Gateway can not wait indefinitely for the service to send a response. The Gateway must decide whether to return an error to the client or provide partial information from a cache.

WSO2 allows to incorporate strategies for partial failure and caching through endpoints in WSO2 ESB. When the call to a microservice exceeds a set threshold, a pattern called circuit breaker is implemented, canceling the unnecessary wait when a service is unavailable or does not respond. If the error rate for a service exceeds a specific threshold, WSO2 can trigger a switch and all requests will fail immediately for a period of time.

This pattern can be implemented by properly configuring endpoints in the WSO2 ESB. In the next section, you can see a standard configuration of Endpoints that allows implementing the circuit-breaker pattern.

5

```
<Endpoint name = "CircuitBreakerEP">
  <Address uri = "http: // localhost: 9764 / app / microservices / call">
   <suspendOnFailure>
    <InitialDuration> 60000 </ initialDuration> </
   suspendOnFailure> <markForSuspension>

    <ErrorCodes> 101,507.101508 </ ErrorCodes>
    <retriesBeforeSuspension> 3 </ retriesBeforeSuspension> <RetryDelay>
    400 </ RetryDelay> </ markForSuspension> <timeout>


    <Duration> 300 </ duration>
    <ResponseAction> fault </ responseAction> </
  timeout> </ address> </ endpoint>
```

WSO2 Microservices and Timeouts

WSO2 ESB has the ability to isolate access to the network using timeout patterns. Its main function is to prevent the client (proxy) that requires a service to wait indefinitely for the response of the provider service.

To implement this pattern, WSO2 ESB provides Endpoints; Endpoints provide properties such as the duration of the timeout and the action if the timeout has exceeded a certain threshold. Similarly you can use exception sequences that can invoke cache services or respond to temporary data.

In essence, an endpoint allows for definite disruption when it has reached an established timeout. This pattern prevents the failure from spreading and degrades the services of the solution. In the next section, you can see a standard configuration of Endpoints that allows you to implement the timeout pattern on WSO2 ESB.

```
<Endpoint name = "MICROSERVICE">
  <Address uri = "http: // localhost: 8267 / platform / microService / func"> <timeout>

    <Duration> 320 </ duration>
    <ResponseAction> fault </ responseAction> </
  timeout> </ address> </ endpoint>
```

WSO2 Microservices Bulkheads

WSO2 ESB allows the establishment of strategies to avoid cascading failure through properly configured endpoints; Being able to isolate problems of network or services without affecting the architecture of the solution. The Bulkheads is a pattern that establishes practices to reduce the risk of failure and thus avoid degradation and its consequences through barriers such as the implementation of timeout pattern, retries, circuit-breaker, among others.

WSO2 Microservices and Cache

WSO2 ESB allows the establishment of cache memory or the development of customizable implementations for the use of traditional database or key-value (Redis for example). The gateway or Gateway may include a barrier to ensure that failures in external systems affect functionality.

WSO2 Microservices and Load Distribution

NGINX is a scalable alternative to microservices, providing a high performance web server, easy to deploy, configure and schedule.

WSO2 Microservices and discovery of Endpoints

WSO2 Gateway will need to know the location (IP address and port) of each microService with which it will communicate; Their localization can be done through a registry discovery process (WSO2 Governance Registry), allowing the endpoints to be dynamically established at runtime.

WSO2 Microservices Containers

The operational management of microservices can easily become a very complex task as they grow; To facilitate its administration is recommended the use of containers like Docker. Docker is a virtual environment based on Linux (LinuX Containers) that allows to establish containers for microservices or technologies like mysql, nginx, among others.

WSO2 Microservices Monitoring and Data Analysis

WSO2 ESB and WSO2 MSS enable integration with the WSO2 DAS data analysis product that supports batch, real-time, predictive and interactive analysis. With this strategy the platform monitoring and alerting capability for Microservices is supported individually or centrally.

General recommendations

WSO2 offers complete stack-wide support for building a microservices platform, delivering concrete and workable strategies to the challenges of deployments across distributed environments.

01 Identify the WSO2 Microservice architecture model best suited to your solution.

02 Establish a strategy for responding to latency and fault tolerance for your MicroServices architecture. Use endpoints, and avoid fragility in this area.

03 Establish a Gateway API and a Gateway Messaging for your MicroServices architecture.

04 Develop microservices using the current WSO2 stack or the WSO2 MSS product.

05 Set up a Gateway using either WSO2 ESB or WSO2 GW.

06 Implement an Event Store-based strategy for publishing and subscribing to events.

07 Incorporate a delivery guarantee strategy into your microservice architecture.

08 Framework such as vagrant, saboteur, wiremock, hixtrix, cucumber can be combined to establish a solid stack for testing microservices.

09 Use frameworks like Docker, Kubernetes, and Vagrant to simplify the management of containers and virtual machines.

Essentially

- **Within Within a Microservices architecture the WSO2 ESB can develop functions associated with the conformation of a Gateway API and a Message Gateway.**
- **WSO2 stack products can use the ws-eventing standard to establish unattached and lightweight communication mechanisms based on HTTP or JMS.**
- **Microservices can be implemented on platforms such as WSO2 DSS, BRS, AS within a container with an independent life cycle (executed in its own process), for example Docker.**
- **All microservices on WSO2 can be deployed in a Cluster and HA on container services such as Docker and Kubernetes for administration.**
- **Microservices can be deployed as a war in containers such as Jetty or Tomcat and include the discovery of services using a register (WSO2 GR).**
- **Microservices in WSO2 can be supported by OSGi bundles.**
- **Fault tolerance patterns such as timeout and circuit-breaker can be implemented through endpoints in the gateway (WSO2 ESB or WSO2 GW).**
- **The WSO2 ESB Gateway can inject cache, security, logging, audit, validation, delivery guarantee policies, among others.**
- **The WSO2 platform already includes microservices specializing in identity management services for authentication and authorization.**
- **Each service is Self - Monitoreable, ie it constantly reports its status at runtime.**
- **Each service has a dashboard that allows you to view its behavior at runtime individually.**
- **If the integration of service behavior indicators at runtime is required, they can be configured using events that can enrich a WSO2 DAS platform.**
- **Each service can have aspects of logging, auditing, exceptions independently or these can be injected into a Gateway service (ESB or services on an application server).**
- **OSGI is the basis of WSO2, providing services with uniform interfaces as the primary form of encapsulation.**

WSO2 is an Open Source stack that can be used to implement a MicroService architecture. I hope these implementation models and recommendations will strengthen your initiative and provide a more robust, agile and flexible architecture.

Thank you very much!

About the Author:

July Cejas

Architect Chakray / julio.cejas@chakray.com

Specialist with more than 15 years of experience as Consultant and Architect in the construction of Critical Systems based on free and open source products. He combines his expertise in Computer Security and his passion for free and opensource technology to build disruptive solutions.

CHAKRAY

DOING THE RIGHT THINGS.

WITH THE RIGHT TECHNOLOGY.
TO SUPPORT BUSINESS.

info@chakray.com | www.chakray.com