XORCERY USES JETTY & JERSEY AND MTLS IMPLEMENTATION AND JWT SUPPORT

on **octubre 31, 2023** in **xorcery** with **No hay comentarios.**

In order to better understand Xorcery, we are going to explain the building blocks on which the framework resides. At Xorcery we use Jetty and Jersey.

Jetty

Jetty is a lightweight web server and servlet container prepared to be embedded in our applications. The distribution we use is from Eclipse and has the following characteristics:

- It is an asynchronous HTTP server
- It is a standard Servlet Container
- It is a Web Sockets server
- It has an asynchronous HTTP client
- Supports OSGI, JNDI, JMX, JASPI, AJP.

Jersey

Developing RESTful web services that seamlessly support exposing your data on a variety of rendering media types and abstracting the low-level details of client-server communication is not an easy task without a good set of tools. To simplify the development of RESTful web services and their clients in Java, a standard and portable JAX-RS API has been designed.

The Jersey RESTful Web Services 2.x framework is an open-source, production-quality framework for developing RESTful web services in Java that provides support for JAX-RS APIs and serves as a JAX-RS reference implementation (JSR 311, JSR 339, and JSR 370).

The Jersey RESTful Web Services 3.x framework provides support for Jakarta RESTful Web Services 3.0.

The Jersey framework is more than the JAX-RS reference implementation. Jersey provides its own API that extends the JAX-RS toolkit with additional functions and utilities to further simplify RESTful serving and client development. Jersey also exposes numerous extension SPIs so that developers can extend Jersey to better meet their needs.

Módulos en Java

All code in Xorcery is developed using Java Modules. If you have problems understanding this organization available since Java 9, I recommend you see this link

(you can use the subtitles in Spanish or English).

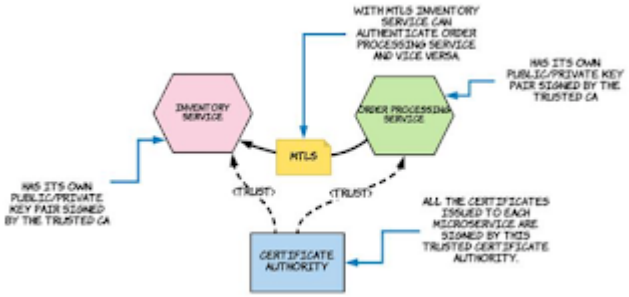
GlassFish HK2 - Dependency Injection

GlassFish HK2 3.0 API is a declarative framework for services using annotations such as Contract and Service. It is based on Jakarta Dependency Injection (DI) standard annotation. We are using it as a dependency injection framework in Jersey and enabling auto-scanning to auto-discover components marked as @Contract and @Service.

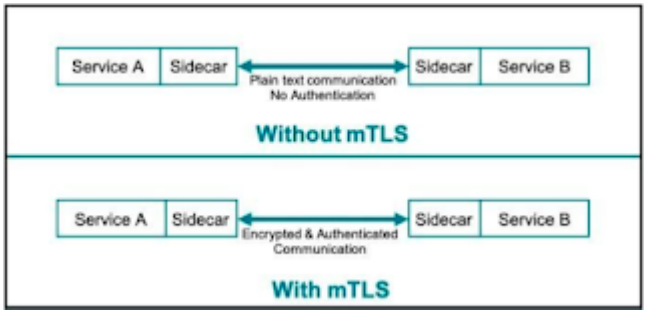
You can find an example of this in this link: https://mkyong.com/webservices/jax-rs/jersey-and-hk2-dependency-injection-auto-scanning/

Use in Xorcery to have mTLS

In the quest to achieve mTLS encrypted server-to-server communication, TLS has been activated for HTTP 1.1 and HTTP/2 and support for client certificates to obtain mTLS support.



You must have read or known about this type of communication when using a Service Mesh. But Xorcery wants to avoid such complexity and give you the same type of secure communication between services.



For this reason, in the source code you will find:

1. **xorcery-certificates**: the code that starts the provisioning, sets up renewal scheduler, and stores cert into KeyStore etc. This can run on both client and server, and defines an SPI
2. **xorcery-certificates-client**: implements SPI and delegates to remote server via HTTPS. Can optionally support HTTP01 ACME challenge
3. **xorcery-certificates-server**: provides JAXRS API that clients (2.) can call, and then delegates to SPI implementations
4. **xorcery-certificates-letsencrypt**: Let's Encrypt implementation of SPI, can be run in "client" or "server", both would work
5. **xorcery-certificates-ca**: our own CA implementation of the SPI

We have sought to design the SPI carefully, to have a cleaner configuration and to make it possible to use Let's Encrypt and our own CA at the same time (they do different things and have advantages and disadvantages each).
They can run in standalone mode (where each server performs provisioning itself) or in client-server mode where provisioning/renewal is centralized.

Servers can now receive certificates from our own CA or Let's Encrypt, or both (which makes SNI possible).

Certificate Signing Request validation now optionally supports self-signing, so there is no need to distribute the provisioning key if you know all CSRs are coming from your own network.

JWT Authentication and Authorization

We also have a **xorcery-jwt** module that:

1. Exposes a login endpoint (username/password). This way, clients will be able to authenticate.
2. It has an SPI to allow the plugin to provide JWT Claims.

Support for Jetty contraints mappings has also been added to **xorcerty-jetty-server** so that authorization requirements can be added. Basically, it will check the "roles" of the JWT claims (as a series of role names) against the role requirements of a mapped route. With the latter we already have a level of authorization.

Use in clients

With everything explained above, you will now be able to configure certificate information, REST API resources, jetty server information, JWT issuer, etc. in your clients.

I take the following configuration from **xorcery-examples**:

```
application.name: "forum"
instance.home: "{{ SYSTEM.jpackage_app-path ? jpackage.app | SYSTEM.user_dir}}"
jpackage.app: "{{ SYSTEM.jpackage_app-path }}/../../lib/app"

# So that we can generate a SSL certificate for the local hostname. Replace with whatever domain name you actually use
instance.domain: local

# Add local convenience names for your own computer into the SSL cert
certificates:
  dnsNames:
    - localhost
    - "{{ instance.host }}"
  ipAddresses:
    - 127.0.0.1
    - "{{ instance.ip }}"

# REST API resources
jersey.server.register:
  - com.exoreaction.xorcery.examples.forum.resources.api.CommentResource
  - com.exoreaction.xorcery.examples.forum.resources.api.ForumResource
  - com.exoreaction.xorcery.examples.forum.resources.api.PostCommentsResource
  - com.exoreaction.xorcery.examples.forum.resources.api.PostResource
  - com.exoreaction.xorcery.examples.forum.resources.api.PostsResource
```

```
dns.client.search:
  - xorcery.test
dns.client.hosts:
    _certificates._sub._https._tcp : "https://127.0.0.1"
dns.client.nameServers:
  - 127.0.0.1:8853

jetty:
  server:
    http:
      port: 8080
    ssl:
      port: 8443
    security:
      jwt:
        issuers:
          server.xorcery.test: "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQ....."

# These features can be extracted into separate services
jwt.server.keys:
  - keyId: "2d3f1d1f-4038-4c01-beb7-97b260462ada"
    alg: "ES256"
    publicKey: "secret:MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQ...."
    privateKey:
"secret:MEECAQAwEwYHKoZIzj0CAQYIKoZIzj0DAQ...."

dns.server.port: 8853

# Log configuration
log4j2.Configuration:
  name: Xorcery Example Forum
  status: warn
  thresholdFilter:
    level: trace
  appenders:
    Console:
      name: STDOUT
      target: SYSTEM_OUT
      PatternLayout:
        Pattern: "%d [%t] %-5level %marker %c{1.}:
%msg%n%throwable"
#    Log4jPublisher:
#      name: Log4jPublisher
#      PatternLayout:
```

```
#       Pattern: "%d [%t] %-5level %marker %c{1.}:
%msg%n%throwable"

  Loggers:
    logger:
      - name: org.apache.logging.log4j
        level: debug
        additivity: false
        AppenderRef:
          ref: STDOUT

      - name: com.exoreaction.xorcery.core.Xorcery
        level: debug

      - name: com.exoreaction.xorcery.service
        level: debug

      - name: com.exoreaction.xorcery.dns
        level: trace

    Root:
      level: info
      AppenderRef:
        - ref: STDOUT
#       - ref: Log4jPublisher
```

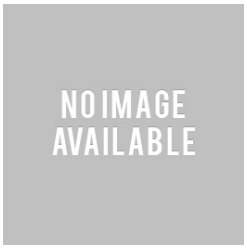
In the following articles we are going to explain what other building blocks Xorcery has and why they are used to finally end up in microservices demos.

Your Pull Requests to this framework are welcome in advance.

Enjoy!

Share:

Related Posts:



Xorcery is born - Build high-performance microservices

Xorcery implements Reactive Streams - Parte 1

Xorcery implementa Reactive Streams - Parte 1

Xorcery usa Jetty & Jersey e implementación de mTLS y soporte a JWT

Xorcery uses Jetty & Jersey and mTLS implementation and JWT support

Entrada más reciente Página Principal Entrada antigua

0 comentarios:

Publicar un comentario

Para dejar un comentario, haz clic en el botón de abajo para acceder con Google.

ACCEDER CON GOOGLE

JoeDayz.pe

Cursos

About

Popular Posts

Xorcery implementa Reactive Streams - Parte 1

¿Qué es Reactive Streams? Para poder entender como nos permite Xorcery trabajar con reactive streams , tenemos que saber que es Reactive ...

Paso de la Oración

Este 10, 11 y 12 de Septiembre ha sido un fin de semana enriquecedor para Miryan y

Categories

#github #java (1)

#guatejug (1)

#historiasdeprogramador (1)

aaii (1)

academia web (2)

acr (1)

para mí. Despues de varios años fuimos a la convivencia...

Blog Archive

grails (5)

groovy (3)

hangouts (1)

highchart-export-server (2)

huacho (1)

hudson (1)

hyperledger-composer (1)

hyperledger-fabric (1)

i-educa (1)

iBATIS (2)

icescrum (1)

informatica (1)

Intigas (1)

ITP_JAVA (1)

jakartaee (4)

jakartaee10 (1)

JasperReports (1)

java (3)

JavaCard (1)

JavaDayUNI (1)

JavaOne (1)

jhipster (2)

jmeter (1)

joedayz (46)

JOERP (4)

jpa (1)

jquery (1)

kafka (3)

kotlin (2)

Kubernetes (3)

lombok (1)

m2eclipse (1)

mac (2)

Matt Raible (1)

Maven (3)

microprofile (6)

microprofile-jwt jakartaee (2)

microprofile-jwt jdbc-realm jakartaee (1)

microprofile-jwt jdbc-realm payara (1)

microservicios (1)

Ministerio del Interior (1)

MJN (5)

móvil (1)

mysql (1)

namespaces (1)

navidad (1)

NET (4)

Nextel (1)

Novell (1)

ocjp (1)

Opentaps (2)

Oracle (1)

oraclecloud (1)

oraclefunctions (1)

oracleopenworld (1)

OSUM (1)

OSX (1)

p6spy (1)

Payara (5)

personal (1)

perujug jconfperu joedayz (2)

php (1)

play (1)

PMP (1)

podcasts (1)

PostgreSQL (8)

programacion (1)

pubsub (1)

PUCP (4)

quadim (2)

quarkus (1)

rackspace (1)

rails (2)

redis (1)

refactoring (2)

Reniec (1)

renovatebot (2)

Rider (1)

ruby (4)

rust (1)

scala (1)

SCD2010 (1)

SCJP (1)

Scrum (3)

Scrum evaluacion (1)

seminarios (1)

Setup (1)

SourceRepo (1)

spring (13)

spring 3.1 (1)

spring android (1)

spring mobile (1)

spring social (1)

spring-boot (9)

spring-boot-admin (2)

spring-cloud (1)

spring-cloud-config (2)

SpringCommunityDay (1)

SpringRoo (2)

springsource (1)

sqlserver (2)

start-up (1)

STS (1)

Subclipse (1)

Subversion (1)

SUN (1)

SUNAT (2)

synergyj (2)

Syscom (1)

Talleres (21)

Telefonica (1)

thedevconf (1)

thymeleaf (1)

Trac (1)

try-with-resources (1)

twitter (1)

Tye (1)

ubuntu (3)

UNI (3)

UNMSM (1)

UPC (1)

videos (1)

vimeo (1)

weblogic (2)

Workspace (1)

WPF (1)

xorcery (6)

xsd (1)

YaRetail (1)

YoMeQuedoEnCasa (1)

zuul (2)