Xorcery Examples - Forum Application

on **noviembre 27, 2023** in **xorcery** with **No hay comentarios.**

This is another example of Xorcery using commands/aggregates, including JAX-RS integration, along with a simple forum application domain example (posts with comments). It now shows t**he full cycle of GET, submit form, parse/process command, turn into events, project into Neo4j, query updates from Neo4j, and render with updated data**.

Setup

We will use the same dependencies from our first Greeter example, so we will only add those that are additional and necessary for this example.

```
<!-- Domain model and projection -->
<dependency>
 <groupId>com.exoreaction.xorcery</groupId>
 <artifactId>xorcery-domainevents-publisher</artifactId>
</dependency>
<dependency>
 <groupId>com.exoreaction.xorcery</groupId>
 <artifactId>xorcery-domainevents-neo4jprojection</artifactId>
</dependency>
```

These dependencies are necessary for the projection to neo4j.

```
 <!-- REST API -->
<dependency>
 <groupId>com.exoreaction.xorcery</groupId>
 <artifactId>xorcery-jsonapi-server-neo4j</artifactId>
</dependency>
<dependency>
 <groupId>com.exoreaction.xorcery</groupId>
 <artifactId>xorcery-domainevents-jsonapi</artifactId>
</dependency>
```

We add these to the REST-API dependencies to support domain events and projection to neo4j.

```
<!-- Integration -->
<dependency>
```

```
    <groupId>com.exoreaction.xorcery</groupId>
    <artifactId>xorcery-dns-registration</artifactId>
  </dependency>
```

In the integration dependencies we only add this for registration in the DNS server.

```
  <!-- Logging -->
  <dependency>
     <groupId>com.exoreaction.xorcery</groupId>
     <artifactId>xorcery-log4j</artifactId>
   </dependency>
```

We added the logging dependency for Xorcery to be able to see the messages that Xorcery publishes in the log.

```
    <!-- These features can be extracted out into their own
  service -->
    <dependency>
     <groupId>com.exoreaction.xorcery</groupId>
     <artifactId>xorcery-jwt-server</artifactId>
    </dependency>
    <dependency>
     <groupId>com.exoreaction.xorcery</groupId>
     <artifactId>xorcery-eventstore</artifactId>
    </dependency>
    <dependency>
     <groupId>com.exoreaction.xorcery</groupId>
     <artifactId>xorcery-opensearch</artifactId>
    </dependency>
    <dependency>
     <groupId>com.exoreaction.xorcery</groupId>
     <artifactId>xorcery-certificates-ca</artifactId>
    </dependency>
    <dependency>
     <groupId>com.exoreaction.xorcery</groupId>
     <artifactId>xorcery-dns-server</artifactId>
    </dependency>
```

Finally, we publish the dependencies that allow us to work with a **JWT server**, an **event store**, an **open search**, and a **DNS server**. What makes Xorcery so special is that it establishes an almost real setup for working with microservices.

In the **src/main/docker** folder, you will find a docker-compose.yml that starts precisely these dependent services.

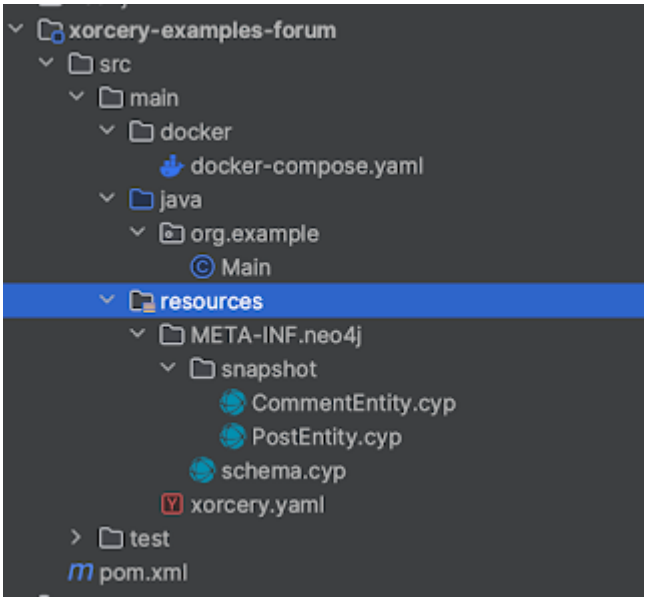
```
version: "3.9"
services:
  exmples-forum-eventstore:
    image: eventstore/eventstore:23.6.0-buster-slim
    environment:
      - EVENTSTORE_CLUSTER_SIZE=1
      - EVENTSTORE_RUN_PROJECTIONS=All
      - EVENTSTORE_START_STANDARD_PROJECTIONS=true
      - EVENTSTORE_EXT_TCP_PORT=1113
      - EVENTSTORE_HTTP_PORT=2113
      - EVENTSTORE_INSECURE=true
      - EVENTSTORE_ENABLE_EXTERNAL_TCP=true
      - EVENTSTORE_ENABLE_ATOM_PUB_OVER_HTTP=true
      - EVENTSTORE_MAX_APPEND_SIZE=8388608
    deploy:
      resources:
        limits:
          cpus: "2"
          memory: 2G
        reservations:
          cpus: "1"
          memory: 1G
    ports:
      - "1113:1113"
      - "2113:2113"
    volumes:
      - type: volume
        source: eventstore-volume-data
        target: /var/lib/eventstore
      - type: volume
        source: eventstore-volume-logs
        target: /var/log/eventstore
  exmples-forum-opensearch:
    image: opensearchproject/opensearch:2.11.0
    environment:
```

```
    - cluster.name=opensearch-cluster
    - node.name=opensearch-node1
    - bootstrap.memory_lock=true # along with the memlock
settings below, disables swapping
    - "OPENSEARCH_JAVA_OPTS=-Xms512m -Xmx512m" #
minimum and maximum Java heap size, recommend setting both
to 50% of system RAM
    - "DISABLE_INSTALL_DEMO_CONFIG=true" # disables execution
of install_demo_configuration.sh bundled with security plugin,
which installs demo certificates and security configurations to
OpenSearch
    - "DISABLE_SECURITY_PLUGIN=true" # disables security plugin
entirely in OpenSearch by setting plugins.security.disabled: true in
opensearch.yml
    - "discovery.type=single-node" # disables bootstrap checks
that are enabled when network.host is set to a non-loopback
address
  ulimits:
    memlock:
      soft: -1
      hard: -1
    nofile:
      soft: 65536 # maximum number of open files for the
OpenSearch user, set to at least 65536 on modern systems
      hard: 65536
  volumes:
    - opensearch-data1:/usr/share/opensearch/data
  ports:
    - 9200:9200
    - 9600:9600 # required for Performance Analyzer
 exmples-forum-opensearch-dashboards:
  image: opensearchproject/opensearch-dashboards:2.11.0
  ports:
    - 5601:5601
  expose:
    - "5601"
  environment:
    - 'OPENSEARCH_HOSTS=["http://exmples-forum-
opensearch:9200"]'
    - "DISABLE_SECURITY_DASHBOARDS_PLUGIN=true" # disables
security dashboards plugin in OpenSearch Dashboards
volumes:
 eventstore-volume-data:
 eventstore-volume-logs:
 opensearch-data1:
networks:
```

```
  bridge:
    driver: bridge
```

Note: You have to have docker desktop installed and previously run this docker-compose and then run it in the Main class of the project.

Now we add the Xorcery configuration and the Neo4j framework of POST and COMMENTS.



In this project, we can see the configuration of logging, rest API resources, certificates, dns client configuration, and jwt server:

```
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.res
ources.api.CommentResource
  -
com.exoreaction.xorcery.examples.forum.res
ources.api.ForumResource
  -
com.exoreaction.xorcery.examples.forum.res
ources.api.PostCommentsResource
  -
com.exoreaction.xorcery.examples.forum.res
ources.api.PostResource
  -
com.exoreaction.xorcery.examples.forum.res
ources.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:
            keys:
            - kid: "2d3f1d1f-4038-4c01-beb7-
```

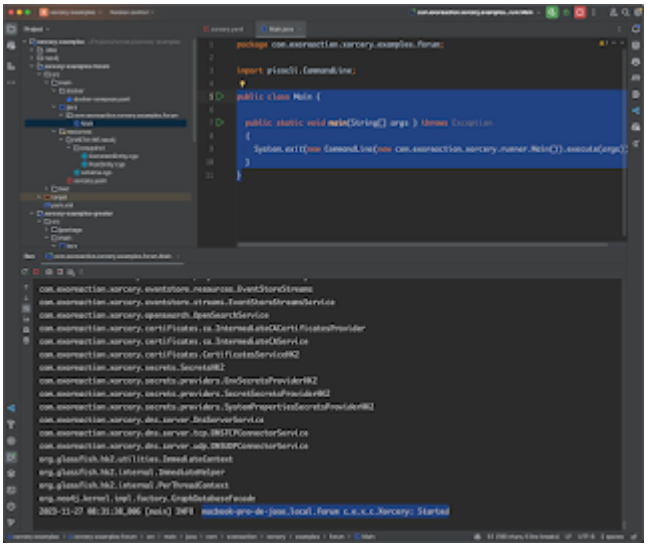
```
97b260462ada"
          alg: "ES256"
          publicKey:
"secret:MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQ
cDQgAEd7L6zz97U1MMaj9MSN325SZ15htR2
6mec0/1A0vt1b8Yfcu0QuiN9E4ijSfMRiof+B5
7P/hkrb+0uRSYLL854Q=="
# These features can be extracted into
separate services
jwt.server.keys:
  - kid: "2d3f1d1f-4038-4c01-beb7-
97b260462ada"
    alg: "ES256"
    publicKey:
"secret:MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQ
cDQgAEd7L6zz97U1MMaj9MSN325SZ15htR2
6mec0/1A0vt1b8Yfcu0QuiN9E4ijSfMRiof+B5
7P/hkrb+0uRSYLL854Q=="
    privateKey:
"secret:MEECAQAwEwYHKoZIzj0CAQYIKoZIzj
0DAQcEJzAlAgEBBCCSHC362NTeBZYTkYGX
K3vfRvoqQum+Uo6DFUDzvX7MuA=="
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
```

We verify that the setup starts without problems by executing the Main class.

```
public class Main {
  public static void main(String[] args ) throws Exception
  {
    System.exit(new CommandLine(new
com.exoreaction.xorcery.runner.Main()).execute(args));
  }
}
```

If everything is fine this will be the result.



We are done with the setup.

Forum Service

We register the ForumService and inject the configuration and service resource objects.

```
@Service(name="forum")
@RunLevel(20)
public class ForumService {
   @Inject
   public ForumService(Configuration configuration,
ServiceResourceObjects serviceResourceObjects) {
      serviceResourceObjects.add(new
ServiceResourceObject.Builder(new
InstanceConfiguration(configuration.getConfiguration("instance")
), "forum")
            .version("1.0.0")
            .attribute("domain", "forum")
            .api("forum", "api/forum")
            .build());
   }
}
```

Model

We create the model for the application by inheriting from Xorcery base classes. For example, the ForumModel inherits from CommonModel.

CommonModel

```
package com.exoreaction.xorcery.domainevents.helpers.model;

public interface CommonModel {
 public static enum Entity {
  id,
  aggregateId,
  externalId,
  createdOn,
  lastUpdatedOn;

  private Entity() {
  }
 }

 public static enum Label {
  Entity,
  Aggregate;

  private Label() {
  }
 }
}
```

ForumModel

```
package com.exoreaction.xorcery.examples.forum.model;
import
com.exoreaction.xorcery.domainevents.helpers.model.CommonM
odel;
public interface ForumModel
    extends CommonModel {
  enum Label {
    Post,
    Comment
```

```
  }
  enum Relationship {
    PostComments
  }
  enum Post {
    title,
    body,
    is_comments_enabled
  }
  enum Comment {
    body
  }
}
```

Now we create the model for Post and Comments, they both inherit from **EntityModel**:

```
package com.exoreaction.xorcery.domainevents.helpers.model;
import
com.exoreaction.xorcery.domainevents.helpers.model.CommonM
odel.Entity;
public interface EntityModel extends Model { // Model inherit
from JsonElement
 default String getId() {
  return (String)this.getString(Entity.id).orElse((Object)null);
 }
 default String getAggregateId() {
  return
(String)this.getString(Entity.aggregateId).orElse((Object)null);
 }
}
```

PostModel

```
public record PostModel(ObjectNode json)
    implements EntityModel {
  public String getTitle() {
    return getString(ForumModel.Post.title).orElse("");
  }
  public String getBody() {
```

```
        return getString(ForumModel.Post.body).orElse("");
    }
}
```

CommentsModel

```
public record CommentModel(ObjectNode json)
        implements EntityModel {
    public String getBody() {
        return getString(ForumModel.Comment.body).orElse("");
    }
}
```

Finally, the records that use the neo4j client to obtain information from the posts or comments.

Posts

```
public record Posts(GraphDatabase db) {
    private static final String POSTS = MessageFormat.format(
            "MATCH ({0}:{0}) WITH {0}, {0} as {1}",
            ForumModel.Label.Post, CommonModel.Label.Entity);
    private final static BiConsumer<GraphQuery, StringBuilder>
clauses = where()
            .parameter(CommonModel.Entity.id, String.class,
"Post.id=$entity_id");
    public GraphQuery posts() {
        return db.query(POSTS).where(clauses);
    }
}
```

Comments

```
public record Comments(GraphDatabase db) {
    private static final String COMMENTS = MessageFormat.format(
            "MATCH ({0}:{0}) WITH {0}, {0} as {1}",
```

```
        ForumModel.Label.Comment, CommonModel.Label.Entity);
    private final static BiConsumer<GraphQuery, StringBuilder>
clauses = where()
        .parameter(CommonModel.Entity.id, String.class,
"Comment.id=$entity_id");
    private static final String POST_COMMENTS =
MessageFormat.format(
        "MATCH ({0}:{0})-[:{1}]->({2}:{2}) WITH {2}, {2} as {3}",
        ForumModel.Label.Post,
ForumModel.Relationship.PostComments,
ForumModel.Label.Comment, CommonModel.Label.Entity);
    private final static BiConsumer<GraphQuery, StringBuilder>
byPostClauses = where()
        .parameter(CommonModel.Entity.id, String.class,
"Post.id=$entity_id");
    public GraphQuery comments() {
      return db.query(COMMENTS).where(clauses);
    }
    public GraphQuery commentsByPost(String postId)
    {
      return
db.query(POST_COMMENTS).where(byPostClauses).parameter(C
ommonModel.Entity.id, postId);
    }
}
```

Entities

This is where we create, and update the posts. Likewise, the creation, updating, and deletion of comments occur. In summary, the **events** are generated according to the generated **commands**.

PostEntity

```
public class PostEntity
      extends Entity<PostEntity.PostSnapshot> {
    @Create
    public record CreatePost(String id, String title, String body)
        implements Command {
    }
    @Update
    public record UpdatePost(String id, String title, String body)
        implements Command {
```

```
    }
    public static class PostSnapshot
        implements EntitySnapshot {
      public String title;
      public String body;
    }
    public void handle(CreatePost command) {
      add(event("createdpost")
            .created("Post", command.id)
            .attribute("title", command.title)
            .attribute("body", command.body)
            .build());
    }
    public void handle(UpdatePost command) {
      add(event("updatedpost")
            .updated("Post", command.id)
            .attribute("title", command.title)
            .attribute("body", command.body)
            .build());
    }
  }
```

CommentEntity

```
  public class CommentEntity
      extends Entity<CommentEntity.CommentSnapshot> {
    @Create
    public record AddComment(String id, String body)
        implements Command {
    }
    @Update
    public record UpdateComment(String id, String body)
        implements Command {
    }
    @Delete
    public record RemoveComment(String id)
        implements Command {
    }
    public static class CommentSnapshot
        implements EntitySnapshot {
      public String body;
    }
    public void handle(AddComment command) {
      add(event("addedcomment")
```

```
            .created("Comment", command.id)
            .attribute("body", command.body)
            .addedRelationship("PostComments", "Post",
metadata.getAggregateId())
            .build());
    }
    public void handle(UpdateComment command) {
        if (snapshot.body.equals(command.body))
            return;
        add(event("updatedcomment")
            .updated("Comment", command.id)
            .attribute("body", command.body)
            .build());
    }
    public void handle(RemoveComment command) {
        add(event("removedcomment").deleted("Comment",
command.id));
    }
}
```

ForumApplication

In this service, we inject the DomainEventPublisher, the GraphDatabae, and the ServiceLocator. We also establish the DomainContext for Post and Comments.

```
@Service(name="forum")
public class ForumApplication {
    private static final Logger logger =
LogManager.getLogger(ForumApplication.class);

    private final DomainEventPublisher domainEventPublisher;
    private final Neo4jEntitySnapshotLoader snapshotLoader;

    private final Supplier<PostEntity> postEntitySupplier;
    private final Supplier<CommentEntity> commentEntitySupplier;

    @Inject
    public ForumApplication(DomainEventPublisher
domainEventPublisher,
                            GraphDatabase database,
                            ServiceLocator serviceLocator
    ) {
        this.domainEventPublisher = domainEventPublisher;
```

```
        this.snapshotLoader = new
Neo4jEntitySnapshotLoader(database);
        postEntitySupplier = () ->
serviceLocator.createAndInitialize(PostEntity.class);
        commentEntitySupplier = () ->
serviceLocator.createAndInitialize(CommentEntity.class);
    }

    public PostsContext posts() {
        return new PostsContext(this, postEntitySupplier);
    }
```

In this class, the publication of command events is handled generically.

```
    public <T extends EntitySnapshot> CompletionStage<Metadata>
handle(Entity<T> entity, DomainEventMetadata metadata,
Command command) {

        try {
            DomainEventMetadata domainMetadata = new
DomainEventMetadata.Builder(metadata.context())
                    .domain("forum")
                    .commandName(command.getClass())
                    .build();

            T snapshot;

            if (Command.isCreate(command.getClass())) {
                // Should fail
                try {
                    snapshotLoader.load(domainMetadata, command.id(),
entity);
                    return CompletableFuture.failedStage(new
BadRequestException("Entity already exists"));
                } catch (Exception e) {
                    // Good!
                    Class<?> snapshotClass = (Class<?>)
((ParameterizedType)
entity.getClass().getGenericSuperclass()).getActualTypeArgument
s()[0];
                    snapshot =
(T)snapshotClass.getConstructor().newInstance();
                }
```

```
            } else {
                snapshot = snapshotLoader.load(domainMetadata,
command.id(), entity);
            }

            List<DomainEvent> events = entity.handle(domainMetadata,
snapshot, command);

            return domainEventPublisher.publish(new
CommandEvents(metadata.context(), events));
        } catch (Throwable e) {
            return CompletableFuture.failedStage(e);
        }
    }
}
```

REST API Resources

The REST API Resources inherit from Xorcery's JsonApiResource and implement some Mixin for extra functionality.

```
# REST API resources
jersey.server.register:
  -
com.exoreaction.xorcery.examples.forum.resources.api.Comment
Resource
  -
com.exoreaction.xorcery.examples.forum.resources.api.ForumRes
ource
  -
com.exoreaction.xorcery.examples.forum.resources.api.PostCom
mentsResource
  -
com.exoreaction.xorcery.examples.forum.resources.api.PostResou
rce
  -
com.exoreaction.xorcery.examples.forum.resources.api.PostsReso
urce
```

Test

After la execution of the Main class:

Let's do clic in **/api/forum**:

forum:macbook-pro-de-jose.local.forum

Links

forum	http://localhost:8080/api/forum

Attributes

version	1.0.0
domain	forum

The form below shows how to use the URI template, instead of having to create the URL manually:

I invite you to try this demo and leave us your comments. Remember that the source code is found at https://github.com/cantara/xorcery-examples.

Enjoy!

Joe

Share:

Related Posts:

Xorcery uses Jetty & Jersey and mTLS implementation and JWT support

Xorcery implements Reactive Streams - Parte 1

Xorcery Ejemplos - Greeter

Xorcery implementa Reactive Streams - Parte 1

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

Página Principal Entrada antigua

0 comentarios:

Publicar un comentario

Escribir comentario

JoeDayz.pe

Cursos

About

Popular Posts

JCONFPERU 2020

On Friday 23rd and Saturday 24th of October, our annual JCONFPERU 2020 conference was held. This year the organizers were: Miryan Ramirez ...

Spring Boot + Thymeleaf + BootStrap

Para este artículo vamos a ver como trabajar un administrador de clientes (abonados,

Categories

#github #java (1)

#guatejug (1)

#historiasdeprogramador (1)

aaii (1)

academia web (2)

acr (1)

inquilinos, miembros, etc) utilizando las siguientes t...

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 (9)

xsd (1)

YaRetail (1)

YoMeQuedoEnCasa (1)

zuul (2)