
ORACLE®

JAX-RS Security

CREATE
THE
FUTURE

Michal Gajdos
michal.gajdos@oracle.com

December, 2014

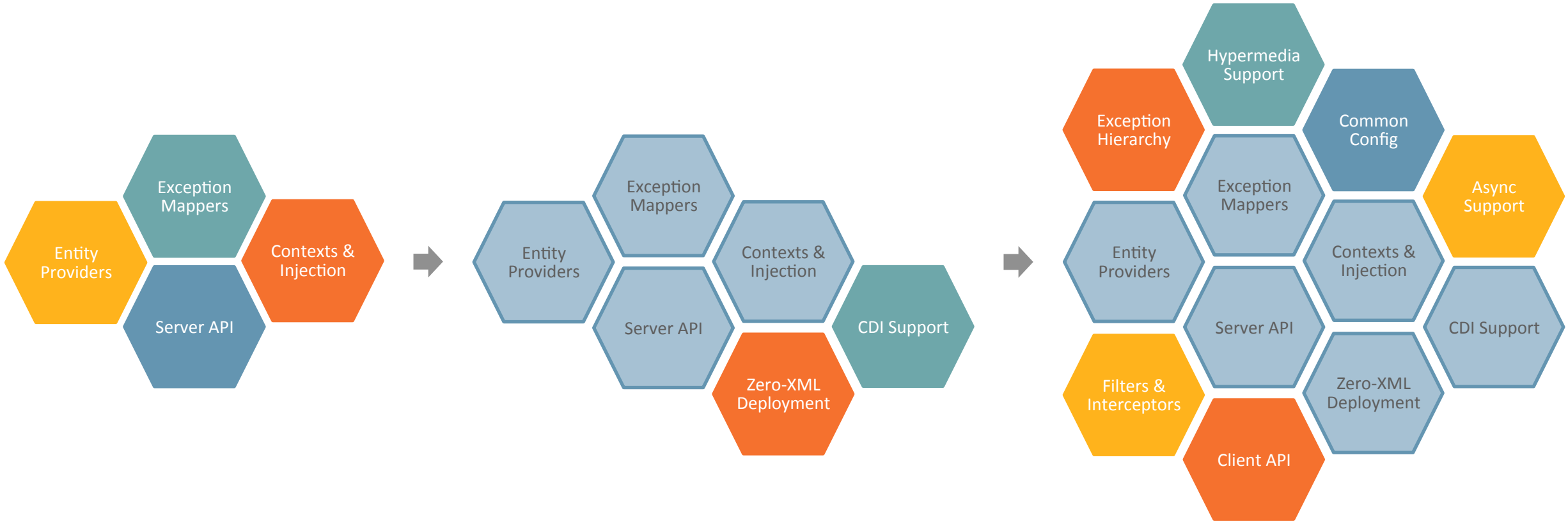
Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Program Agenda

1. Introduction to JAX-RS and Security
2. Declarative Security and Entity Filtering
3. Client Security
4. OAuth

Evolution of JAX-RS API



JAX-RS 1.0 **JAX-RS 1.1** **JAX-RS 2.0**

Introduction to JAX-RS

`http://example-university.com`

`/api/students`

… ***POST*** *(create) new student*

`/api/students/{id}`

… ***GET*** *existing student with given* ***id***

Introduction to JAX-RS

`http://example-university.com`

`/api/students`

*… **POST** new student*

`http://example-university.com/api/students/{id}`

*… **GET** existing student with given **id***

Introduction to JAX-RS

http://example-university.com/api/students

```
POST /api/students HTTP/1.1
Host: example-university.com
Content-Type: application/json
...

{
    "name": "Michal Gajdos"
    "phone": …
    …
}
```

Introduction to JAX-RS

```
http://example-university.com/api/students/mgajdos
```

```
GET /api/students/mgajdos HTTP/1.1
User-Agent: curl/7.30.0
Host: example-university.com
Accept: application/json
```

Introduction to JAX-RS – Resource

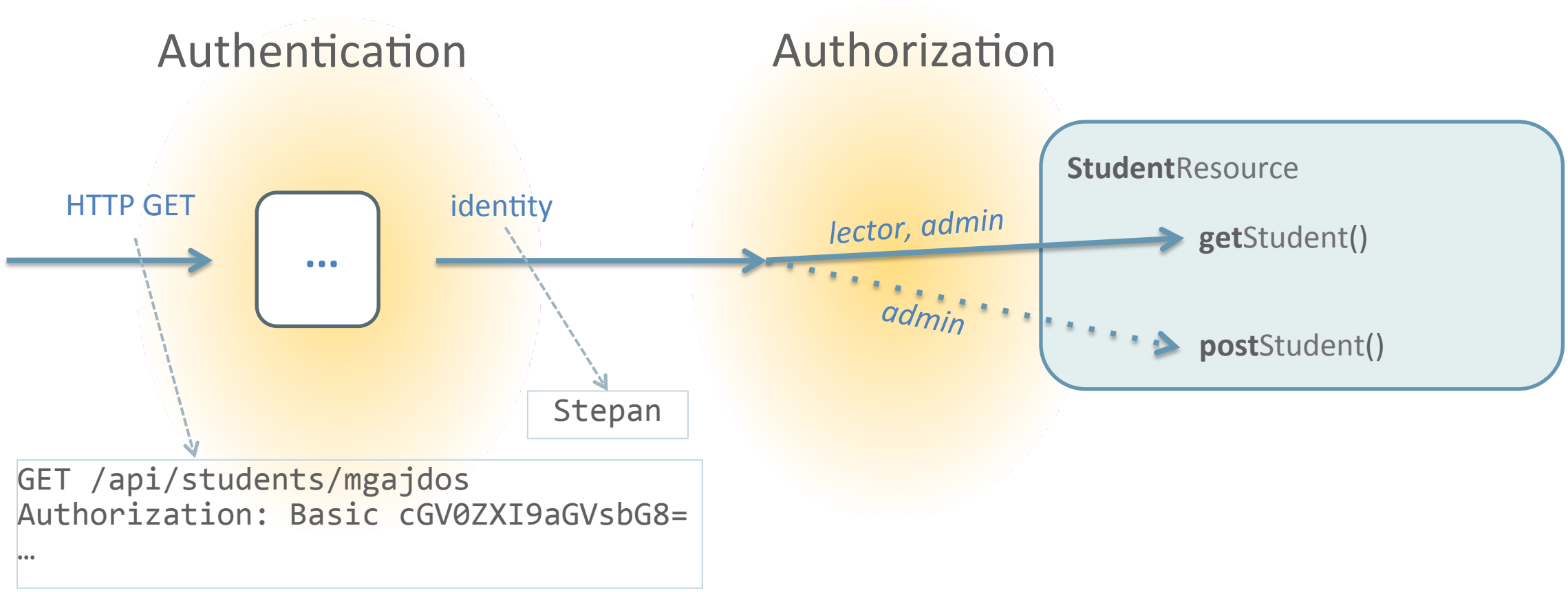
```
@Path("students")
public class StudentResource {
    @GET
    @Path("{id}")
    @Produces("application/json")
    public Student getStudent(@PathParam("id") String id) {
        return StudentService.getStudentById(id);
    }


    @POST
    @Consumes("application/json")
    public Student postStudent(Student student) {
        return StudentService.addStudent(student);
    }
}
```

http://example-university.com/api/**students/{id}**

JAX-RS Security

JAX-RS and Security



JAX-RS and Security

Authentication

- No support in JAX-RS
- Authenticate the REST API caller
- JAX-RS processing is based on Servlet container
  - **Servlet security mechanisms**
- Internal JAX-RS security
  - Based on JAX-RS container request filter (ContainerRequestFilter)
- Result of authentication: JAX-RS **SecurityContext**
- Authentication mechanisms
  - Basic, Digest, OAuth Access Token, JWT (JSON Web Token)

JAX-RS and Security

Authorization

- No support in JAX-RS
- Could use the authorization support based on Servlets
  - web.xml security definition (or @ServletSecurity, @HttpConstraint)
- **Servlet security is “weak” for grain-defined authorization in JAX-RS**
- Implementation specific solutions
  - Based on SecurityContext
  - ContainerRequestFilters
  - @RolesAllowed

JAX-RS and Security – Using web.xml

```
<security-constraint>
    <web-resource-collection>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>admin</role-name>
    </auth-constraint>
</security-constraint>

<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>my-realm</realm-name>
</login-config>
```

JAX-RS and Security – Using web.xml

```
<security-constraint>
    <web-resource-collection>
        <url-pattern>
            /students/*
        </url-pattern>
        <http-method>POST</http-method>
    </web-resource-collection>

    <auth-constraint>
        <role-name>admin</role-name>
    </auth-constraint>

</security-constraint>
```

```
<security-constraint>
    <web-resource-collection>
        <url-pattern>
            /students/*
        </url-pattern>
        <http-method>GET</http-method>
    </web-resource-collection>

    <auth-constraint>
        <role-name>admin</role-name>
        <role-name>lector</role-name>
    </auth-constraint>
</security-constraint>
```

JAX-RS and Security – SecurityContext extension point

```
public interface SecurityContext {

    public Principal getUserPrincipal();

    public boolean isUserInRole(String role);

    public boolean isSecure();

    public String getAuthenticationScheme();

}
```

Introduction to JAX-RS – Resource

```
@Path("students")
public class StudentResource {

    @Context
    private SecurityContext securityContext;

    @POST
    @Consumes("application/json")
    public Student postStudent(Student student) {
        if (!securityContext.isUserInRole("admin")) {
            throw new WebApplicationException(403); // 403 = Forbidden
        }
        return StudentService.addStudent(student);
    }
}
```

Securing Resources in JAX-RS

- New Security JSR planned for Java EE 8
  - Focused on unification, ease of use and portability improvements
  - JAX-RS EG will monitor progress of this JSR
- OAuth 2.0 gaining a lot of adoption from many big players
  - Google, Facebook, Twitter, …
- Vision for REST Services Security Model in Java EE
  - Easy and Straightforward
  - Declarative whenever possible
    - Java EE Security Annotations fit JAX-RS very well
    - Jersey already provides annotation-based authorization

JAX-RS Security – Existing Extension Points

- Server-side
  - `SecurityContext`
    - Get security-related information
  - `ContainerRequestFilter`, `ContainerResponseFilter`
    - Perform security-related tasks at request level
  - `DynamicFeature`
    - Attach security enforcement facilities selectively

- Client-side
  - `ClientRequestFilter`, `ClientResponseFilter`
    - Perform security-related tasks at request level

Existing Jersey Security Features

- Leveraging standard JAX-RS extension points
- Off-the-shelf modules & components ready to use

- Client-side
  - HTTP Basic & Digest Authentication Support
- Server-side
  - Declarative Role-base Authorization (`javax.annotation.security`)
    - `@RolesAllowed`, `@PermitAll`, `@DenyAll`

- OAuth 1.0, 2.0 Support
  - Client and Server Side

Security Annotations

- Define the access to resources based on the user groups
- Security annotations from `javax.annotation.security` package
  - `@PermitAll`, `@DenyAll`, `@RolesAllowed`
  - `SecurityContext`
- `RolesAllowedDynamicFeature`

Registering RolesAllowedDynamicFeature

```
@ApplicationPath("api")
public class MyApplication extends ResourceConfig {

    public MyApplication() {
        packages("com.example.university");

        register(RolesAllowedDynamicFeature.class);
    }
}
```

Security Annotations – Restrictions

```
@Path("students")
@DenyAll
public class StudentResource {
    @GET
    @Path("{id}")
    @Produces("application/json")
    @PermitAll
    public Student getStudent(@PathParam("id") String id) {
        return StudentService.getStudentById(id);
    }

    @POST
    @Consumes("application/json")
    @RolesAllowed("admin")
    public Student postStudent(Student student) {
        return StudentService.addStudent(student);
    }
}
```

No access by default

Public access

Only “admin” role

Entity Filtering

Entity Filtering

Idea and Motivation

- Exposing only part of domain model for input/output
- Reduce the amount of data exchanged over the wire
- Define own filtering rules based on current context
  - Resource method
- Assign security access rules to properties
- Faster prototyping and development
  - One model and one place for defining the rules

Entity Filtering

Jersey 2.3+ / MOXy JSON, Jackson 2.x

- `@EntityFiltering` meta-annotation
  - Create filtering annotations to define context
  - Create filtering annotations with custom meaning to define context
- Security annotations from `javax.annotation.security` package
  - `@PermitAll`, `@DenyAll`, `@RolesAllowed`
  - `SecurityContext`
  - `SecurityEntityFilteringFeature`

Entity Filtering – Model

```
public class Student {

    private String name;
    private String login;
    private String password;

    private DormAddress dormAddress;

    // ...

    // getters and setters
}
```

```
public class DormAddress {

    private String street;
    private String city;

    private String room;

    // ...

    // getters and setters
}
```

Entity Filtering – Annotated Model

```
public class Student {

    public String getName();

    @PermitAll
    public String getLogin();

    @DenyAll
    public String getPassword();

    @RolesAllowed({"lector", "admin"})
    public DormAddress getDormAddress();

    // ...
}
```

```
public class DormAddress {

    public String getStreet();
    public String getCity();

    @RolesAllowed("admin")
    public String getRoom();

    // ...
}
```

Entity Filtering – Un-Restricted Resource

```
@Path("students")
public class StudentResource {
    @GET
    @Path("{id}")
    @Produces("application/json")
    public Student getStudent(@PathParam("id") String id) {
        return StudentService.getStudentById(id);
    }
}
```

```
Student:
    {"name":"Michal Gajdos","login":"gajdm3am"}
Lector:
    {"name":"Michal Gajdos","login":"gajdm3am",
     "dormAddress":{"city":"Praha", "street":"Patkova 3"}}
Admin:
    {"name":"Michal Gajdos","login":"gajdm3am",
     "dormAddress":{"city":"Praha", "street":"Patkova 3","room":"1412"}}
```

Entity Filtering – Restricted Resource

```
@Path("students")
@DenyAll
public class StudentResource {
    @GET
    @Path("{id}")
    @Produces("application/json")
    @PermitAll
    public Student getStudent(@PathParam("id") String id) {
        return StudentService.getStudentById(id);
    }

    @POST
    @Consumes("application/json")
    @RolesAllowed("admin")
    public Student postStudent(Student student) {
        return StudentService.addStudent(student);
    }
}
```

Client Security

JAX-RS 2.0 – Client APIs

```
Client client = ClientBuilder.newClient();

WebTarget universityApi = client.target("http://example-university.com/api");

WebTarget students = universityApi.path("students")
                                  .register(JacksonFeature.class);

WebTarget student = students.path("{id}");

Student mgajdos = student.resolveTemplate("id", "mgajdos")
                         .request("application/json")
                         .get(Student.class);
```

JAX-RS 2.0 – SSLContext and SslConfigurator

```
SslConfigurator sslConfig = SslConfigurator.newInstance()
                .trustStoreFile("./truststore_client")
                .trustStorePassword("pwds65df4")
                .keyStoreFile("./keystore_client")
                .keyPassword("sf564fsds");

Student mgajdos = ClientBuilder.newBuilder()
                    .sslContext(sslConfig.createSSLContext())
                    .register(JacksonFeature.class)
                    .build()
                    .target("https://example-university.com/api/students/mgajdos")
                    .request("application/json”)
                    .get(Student.class);
```

Jersey Client Authentication Filters

- Jersey specific filters
  - ClientRequestFilter and ClientResponseFilter
- HttpAuthenticationFeature
  - HTTP Basic Authentication
  - HTTP Digest Authentication
- OAuth1 and OAuth2
  - OAuth1ClientSupport
  - OAuth2ClientSupport

Jersey Client Authentication Filters

```
HttpAuthenticationFeature basic = HttpAuthenticationFeature
                                    .basic("mgajdos", "secret123")
                                    .nonPreemptive()
                                    .build();

HttpAuthenticationFeature digest = HttpAuthenticationFeature
                                    .digest("mgajdos", "secret123").build();

HttpAuthenticationFeature universal = HttpAuthenticationFeature
                                        .universal("mgajdos", "secret123").build();


WebTarget student = universityApi.path("students/{id}")
                                 .register(JacksonFeature.class)
                                 .register(universal);
```

OAuth

Introduction to OAuth

Resource owner

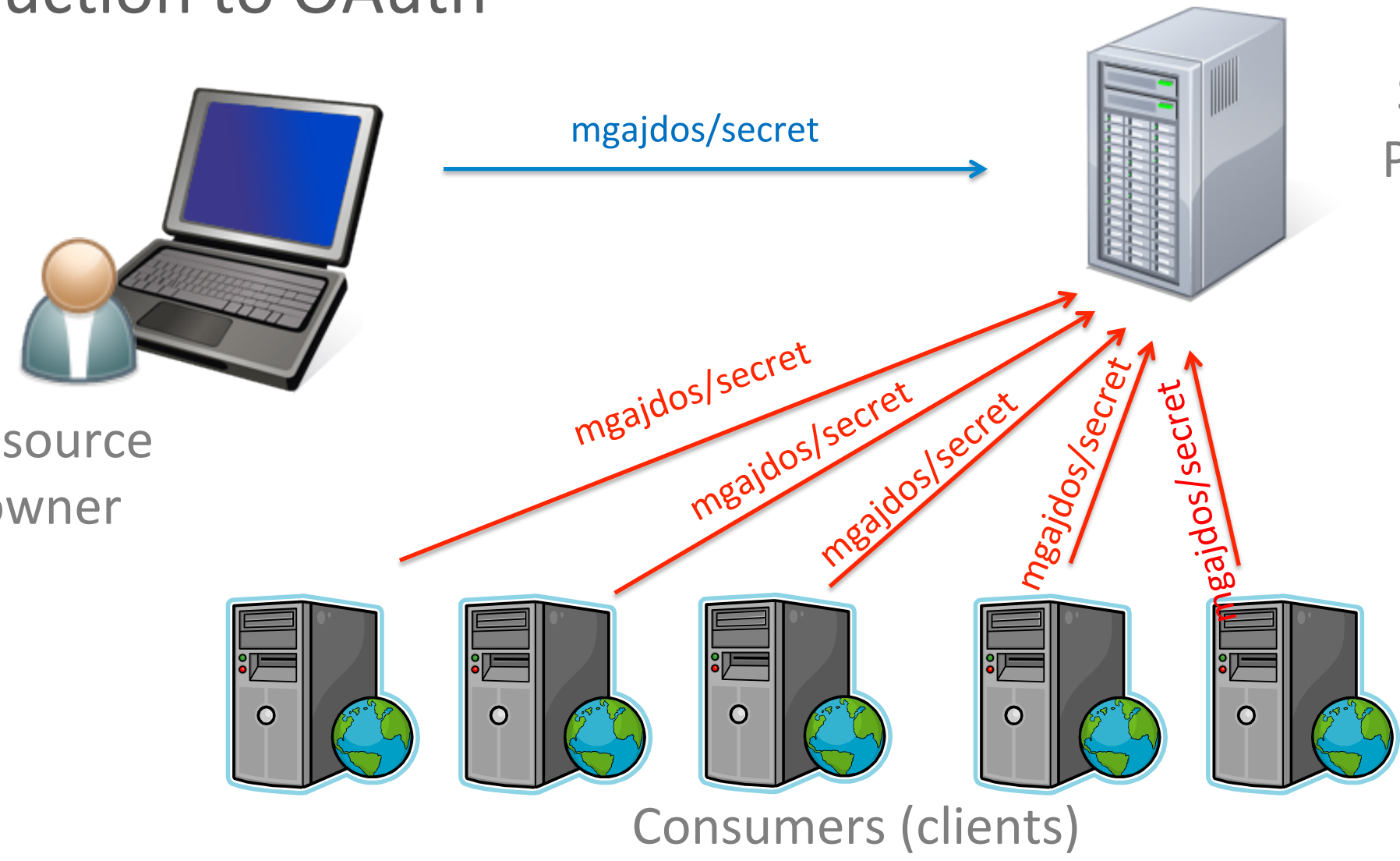
username/password

Service Provider

username/password?

Consumer (client)

Introduction to OAuth



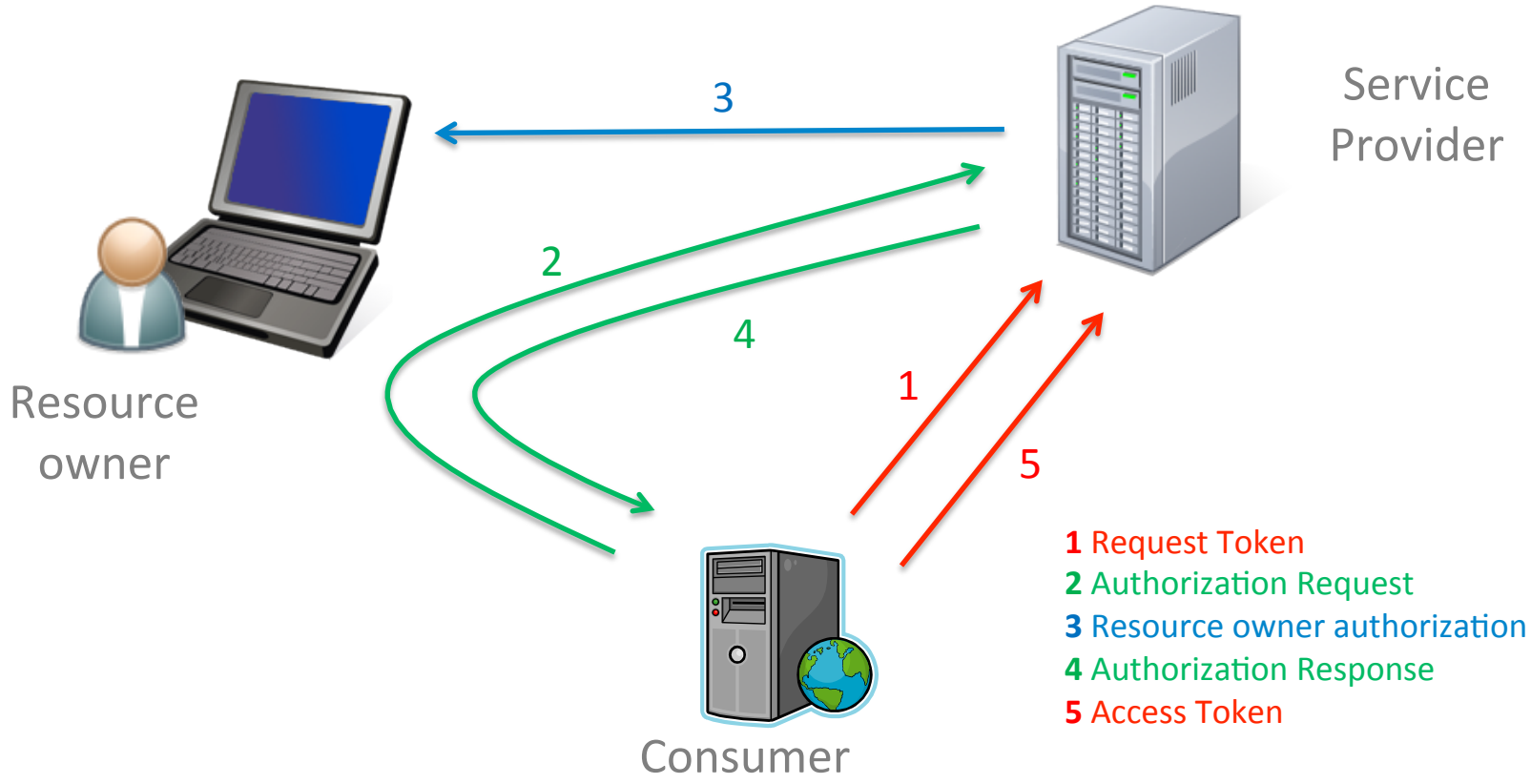
Introduction to OAuth

- Problems of giving consumers my password
  - Revoking access
  - Password change
  - Limit access (different authorization rules)
  - Trust
- Solution is OAuth
  - Access Token
    - Make authenticated requests using Access Token in HTTP header
  - Authorization Flow
    - Process of issuing new Access Token

OAuth1

- IETF OAuth 1.0 (RFC 5849)
  - Previous community version 1.0 and 1.0a
- Signatures added to requests (HMAC-SHA1, RSA-SHA1) based on secret keys
- Authorization process (flow)
  - Process of granting access to the consumer

OAuth1 – Obtaining Access Token



OAuth1

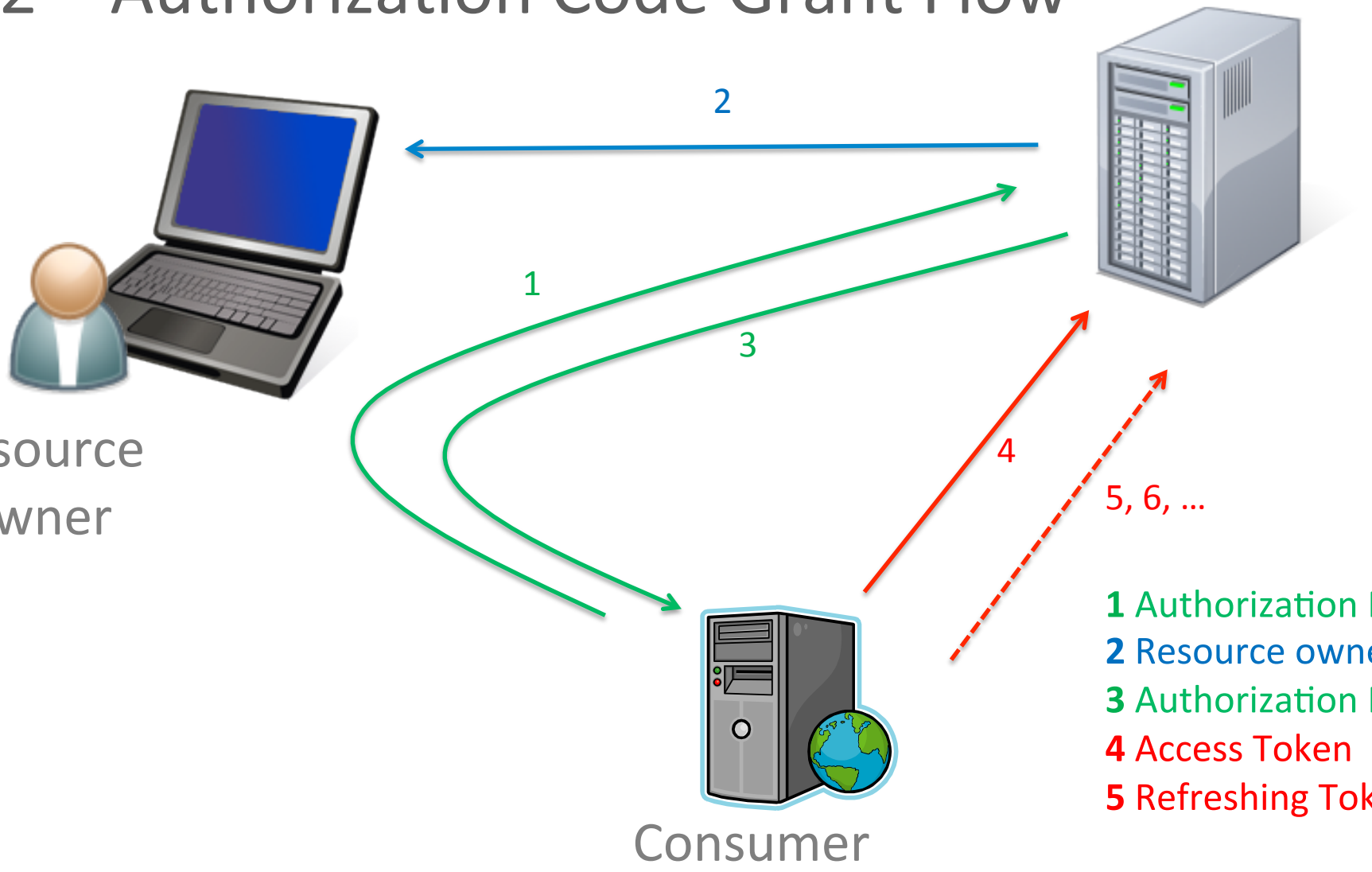
Summary

- Secure
  - Signatures
  - Secret keys (consumer secret, request and access token secret)
  - nonce, timestamp
- Complex for implementation

OAuth2

- OAuth 2.0 (IETF, RFC 6749), released in October 2012
- Not backward compatible, framework (not protocol)
- Does not require signatures (bearer token), SSL
- **Easier** to implement
- Authorization flows
  - Authorization Code Grant (refresh token)
  - Implicit Grant (e.g. Javascript client)
  - Resource Owner Password Credentials Grant (user name + password)
  - Client Credentials Grant (client app authentication)

OAuth2 – Authorization Code Grant Flow



OAuth2 – Authorization Code Grant Flow

Resource owner

Service Provider

Access Token

Consumer

```
GET /api/students/mgajdos HTTP/1.1
User-Agent: curl/7.30.0
Host: example-university.com
Accept: application/json
Authorization: bearer jkr3ljkh3jk
```

JAX-RS/Jersey and OAuth

JAX-RS Client and OAuth

- Non-JAX-RS solution
  - Service Providers (e.g. Google, Facebook) provide libraries in more programming languages to use their services
  - Other libraries
- JAX-RS Client acts as a Consumer/Client party in OAuth scenario
- Functionality
  - Perform Authorization Flow
  - Make authenticated requests (*ClientRequestFilter*s, *ClientResponseFilter*s)

JAX-RS Server and OAuth

- Non JAX-RS solutions based on servlet container
  - Authorization Flow: Servlets or Filters
  - Authenticated requests: JASPIC SAM module (Identity Propagation)
  - Specific solution of application servers (e.g., Weblogic)
- JAX-RS solution
  - Authorization Flow: *JAX-RS resources*
  - Authenticated requests: *ContainerRequestFilters*

Jersey and OAuth

- OAuth 1
  - Client
  - Server
- OAuth 2 (Authorization Code Grant Flow)
  - Client
  - *Server [in progress]*

DEMO

We're Hiring.

Jersey/WebLogic Team is Hiring in Prague!

http://marek.potociar.net/2014/10/15/jersey-is-hiring-in-prague/

Hardware and Software
Engineered to Work Together


ORACLE®