All about

Oracle REST Data Services

07-Jul-2018

aioug

Agenda

- REST Overview
- Introduction to ORDS
- ORDS Architecture
- ORDS APIs for PLSQL Developers
- Securing the REST APIs
- Use Cases with Demo

REST Overview

REST Overview

- REST stands for **Re**presentational **S**tate **T**ransfer
  - It is an architectural ***pattern*** for developing web services as opposed to a ***specification***.
  - REST web services communicate over the HTTP specification.
  - REST uses HTTP vocabulary:
    - Methods (GET, POST, PUT, DELETE, etc.,)
    - HTTP URI syntax (paths, parameters, etc.,)
    - Media types (xml, json, html, plain text, etc.,)
    - HTTP Response codes (200, 404, 503 etc.,)

REST Overview [contd.,]

- Representational
  - Clients possess the information necessary to identify, modify, and/or delete a web resource.
- State
  - All resource state information is stored on the client.
- Transfer
  - Client state is passed from the client to the service through HTTP.

REST Overview [contd.,]

Standard HTTP Methods

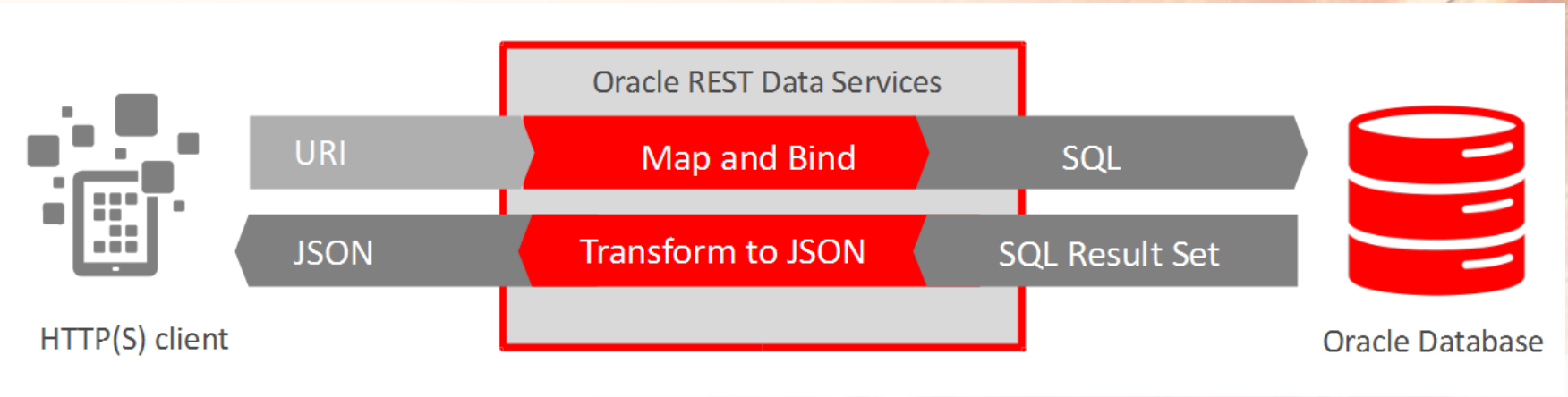
- GET
  - CRUD Operation : Retrieve
  - Usage: Retrieving a resource
- PUT
  - CRUD Operation : Update
  - Usage: Creating or updating a resource at a known URI
- DELETE
  - CRUD Operation : Delete
  - Usage: Deleting a resource
- POST
  - CRUD Operation : Create
  - Usage: Creating a resource within a collection (URI set by server)

ORDS Overview

Oracle REST Data Services

- Consistent data access with modern App Dev frameworks
  - Mid tier application
  - Can map standard http(s) RESTful requests to SQL
  - Can declaratively returns results in JSON format
  - JavaScript friendly and Highly scalable
  - Can connect to Oracle NoSQL and Oracle container databases in Cloud
- Services
  - Formally known as Oracle APEX Listener
  - Access to Relational data over HTTP(s) without installing JDBC/ODBC drivers
  - Oracle JSON collection based schema-less access
  - Comes along with Oracle Database 12.1.0.2 and above
  - New features supports CSV data and Batch load operations
  - Supports Swagger based Open API integration

Architecture



- JSON from Database
  - ORDS creates an URL for the SQL defined as REST api
  - ORDS uses the UCP framework for database connectivity. This helps in mapping and binding the URL with the SQL.
  - ORDS uses Jackson libraries for converting SQL Resultset to JSON and vice-versa.

Implement ORDS

Download

Download latest version of ORDS from OTN

Configure

Configure ORDS parameters, database accounts and url mappings

Deploy

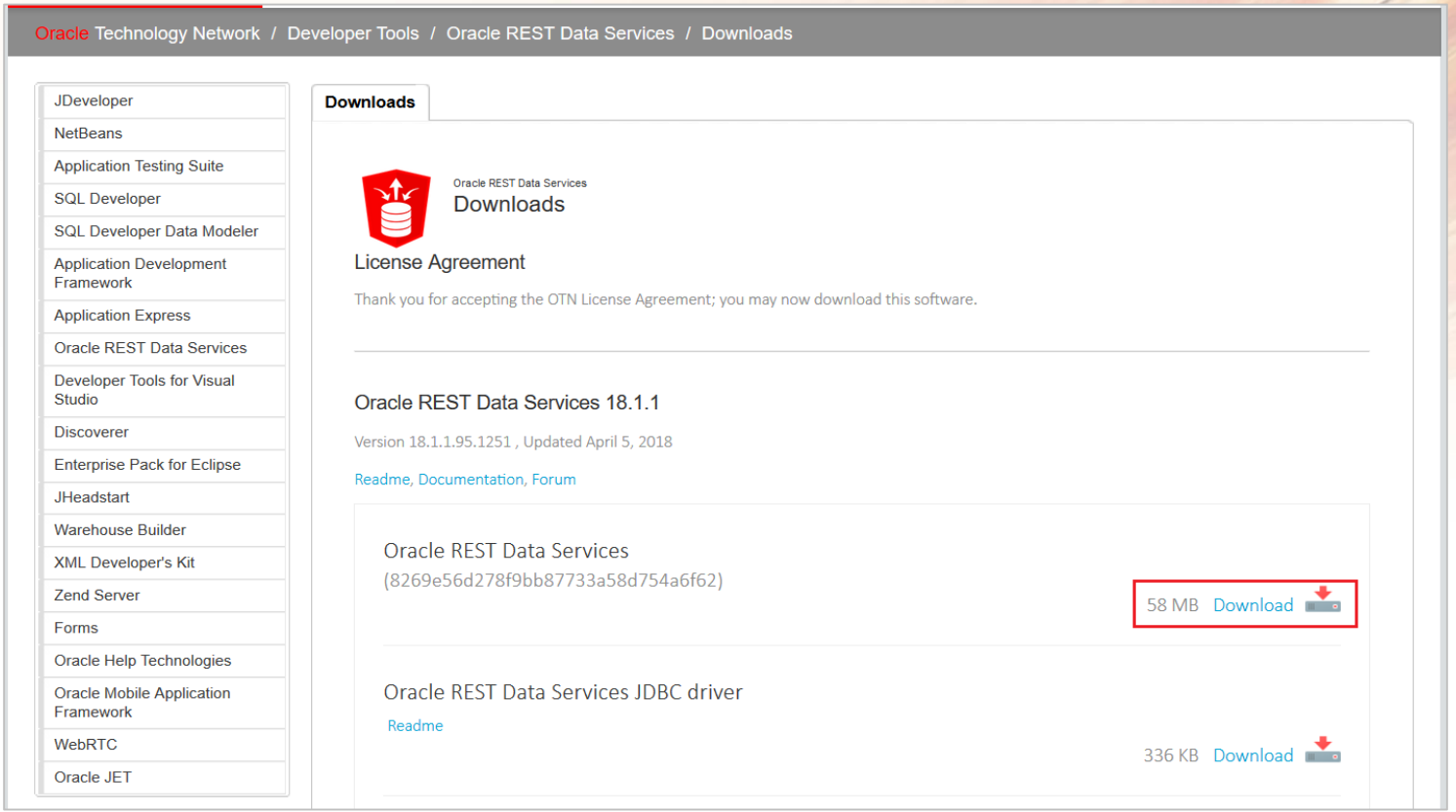
Deploy ords.war to the server or use standalone mode

Publish

Use ORDS apis to expose database objects as REST services

Download ORDS

http://www.oracle.com/technetwork/developer-tools/rest-data-services/downloads/index.html



Installation

Simple	ORDS installation with default parameters. This will reuse existing APEX installation and metadata. `java -jar ords.war install simple`
Advanced	ORDS installation with all necessary parameters. Options available for using APEX installation and metadata. `java -jar ords.war install advanced`
Standalone	Suitable for development use only, and is not supported for use in production deployments. SQL Developer is used to install and manage ORDS Standalone application.

- Unzip downloaded ORDS content into a folder. This path is referred as `/<ORDS_BASE>`
- Create a folder to store ORDS configurations - `/<ORDS_BASE>/conf`
- Update the ORDS parameter file - `/<ORDS_BASE>/params/ords_params.properties`
- Setup conf path as configuration directory, `java -jar ords.war configdir c:\mywork\ords\conf`
- Run installation command, `java -jar ords.war install advanced`

Post - Installation

- ORDS schema created,
  - `ORDS_METADATA` - Stores the metadata about ORDS enabled schemas
  - `ORDS_PUBLIC_USER` - Invoking RESTful services in ORDS enabled schemas
- Database Connection setup,
  - Create Database connection,

    ```
    java -jar ords.war setup --database <db_name>
    ```

  - Setup URL mapping,

    ```
    java -jar ords.war map-url --type base-path /<db_name> <db_name>
    ```

- Verify configuration files created under `<ords_base>/conf` directory

  ```
  defaults.xml, url-mapping.xml
  <db_name>.xml, <db_name>_pu.xml, <db_name>_al.xml, <db_name>_rt.xml
  ```

- Deploy `ords.war` file in Tomcat server

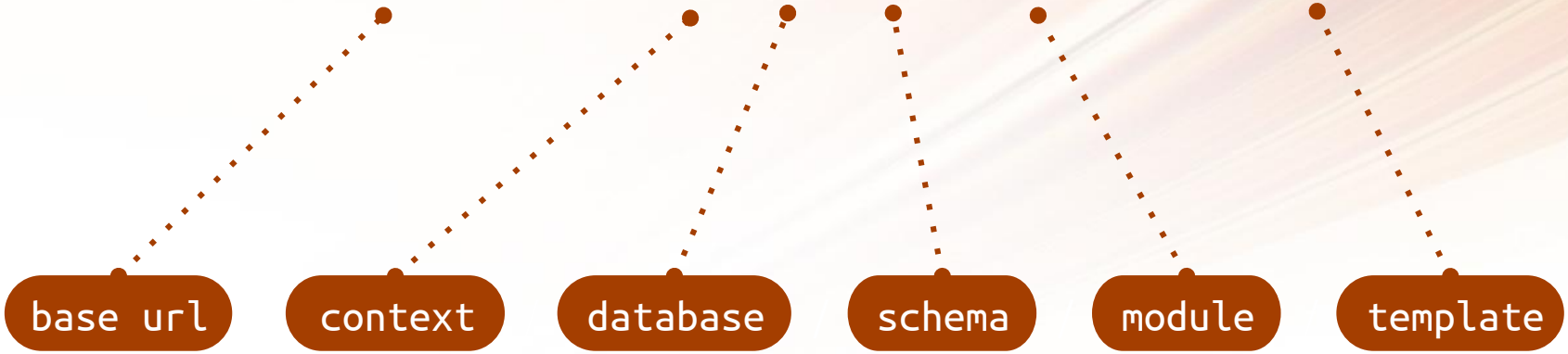
ords_params.properties

```
db.hostname=localhost
db.port=1521
db.servicename=orcl
db.username=ORDS_PUBLIC_USER
migrate.apex.rest=false
plsql.gateway.add=true
rest.services.apex.add=true
rest.services.ords.add=true
schema.tablespace.default=SYSAUX
schema.tablespace.temp=TEMP
standalone.http.port=8888
standalone.mode=true
standalone.use.https=false
user.apex.listener.password=@0588BF3B45D5497836A44AF3A3335B0D2AC30F2284C381888D
user.apex.restpublic.password=@056941E5E7725536B4D021C3DAC3BD9FFAE77983A1F1970F8F
user.public.password=@0539A9876E99F380D7CDA1B619920A81BD7F048D85D58C0751
user.tablespace.default=USERS
user.tablespace.temp=TEMP
```

ORDS - URL Structure

http://localhost:8888/ords/orcl/hr/payroll/dept/:dept_id

ORDS - URL Structure



Enable Schema & Define Module

```
BEGIN
  ORDS.enable_schema
  ( p_enabled                => TRUE
  , p_schema                 => 'HR'
  , p_url_mapping_type       => 'BASE_PATH'
  , p_url_mapping_pattern    => 'hr'
  , p_auto_rest_auth         => FALSE
  );

  ORDS.define_module
  ( p_module_name    => 'payroll'
  , p_base_path      => 'payroll/'
  , p_items_per_page => 10
  );

  COMMIT;
END;
```

ORDS Metadata:

ORDS_SCHEMAS

ORDS_URL_MAPPINGS

ORDS_MODULES

Define SQL as REST service

```
BEGIN
  ORDS.define_template
  ( p_module_name   => 'payroll'
  , p_pattern       => 'dept/'
  );

  ORDS.define_handler
  ( p_module_name   => 'payroll'
  , p_pattern       => 'dept/'
  , p_method        => 'GET'
  , p_source_type   => ORDS.source_type_query
  , p_source        => 'SELECT * FROM departments'
  , p_items_per_page => 5
  );

  COMMIT;
END;
```

ORDS Metadata:

ORDS_TEMPLATES

ORDS_HANDLERS

http://localhost:8888/ords/orcl/hr/payroll/dept/

PLSQL as REST service

```
BEGIN
  ORDS.define_template
  ( p_module_name   => 'payroll'
  , p_pattern       => 'getEmpName/:emp_id'
  );

  ORDS.define_handler
  ( p_module_name   => 'payroll'
  , p_pattern       => 'getEmpName/:emp_id'
  , p_method        => 'GET'
  , p_source_type   => ORDS.source_type_plsql
  , p_source        => 'begin emp_pkg.get_emp_name(:emp_id); end;'
  , p_items_per_page => 5
  );

  COMMIT;
END;
```

Note:

OWA_UTIL, HTP apis are used inside PLSQL procedure to return back to the http request

http://localhost:8888/ords/orcl/hr/payroll/getEmpName/:emp_id

PLSQL with JSON

```
BEGIN
  ORDS.define_template
  ( p_module_name   => 'payroll'
  , p_pattern       => 'createEmp/'
  );

  ORDS.define_handler
  ( p_module_name   => 'payroll'
  , p_pattern       => 'createEmp/'
  , p_method        => 'POST'
  , p_source_type   => ORDS.source_type_plsql
  , p_source        => 'BEGIN emp_pkg.insert_emp ( p_emp_id => :emp_id, ...   ); END;'
  , p_items_per_page => 0
  );

  COMMIT;
END;
```

Payload:

```
{"emp_id":300
,"fname":"Justin", "lname": "Michael Raj“
,"email":"justin@orcl.com"
,"phone":"9234567890","doj":"01-JAN-2010“
,"job":"SA_REP","sal":5000,"comm": 0.25
,"mgr_id":145,"dept_id":80
}
```

http://localhost:8888/ords/orcl/hr/payroll/createEmp/

p_source_type

```
source_type_query            - json/query
source_type_plsql            - plsql/block
source_type_csv_query        - csv/query
source_type_query_one_row    - json/query;type=single
source_type_feed             - json/query;type=feed
source_type_media            - resource/lob
source_type_collection_feed  - json/collection
source_type_collection_item  - json/item
```

AutoREST

Enable AutoREST

```
BEGIN
  ORDS.enable_object
  ( p_enabled      => TRUE
  , p_schema       => 'HR'
  , p_object       => 'JOBS'
  , p_object_type  => 'TABLE'
  , p_object_alias => 'jobs'
  );

  COMMIT;
END;
```

ORDS Metadata:

ORDS_OBJECTS

http://localhost:8888/ords/orcl/hr/metadata-catalog/jobs

AutoREST - SQL Operations

- SELECT

  Method : GET

  http://localhost:8888/ords/orcl/hr/jobs/AC_MGR

  http://localhost:8888/ords/orcl/hr/jobs?q={“job_id”:”AC_MGR”}

  http://../orcl/hr/jobs?q={"min_salary":{"$gte":1500}, "$orderby":{"job_id":"desc"}}

- INSERT

  Method : POST

  Post JSON content as RAW payload

  http://localhost:8888/ords/orcl/hr/jobs

Payload:

```
{"job_id":"IT_CONS“
,"job_title":"IT Consultant“
,"min_salary":40000
,"max_salary":100000}
```

AutoREST - SQL Operations [contd.,]

- UPDATE

  Method : PUT

  Post JSON content as RAW payload

  http://localhost:8888/ords/orcl/hr/jobs/IT_CONS

Payload:

```
{"job_title":"IT Consultant“
,"min_salary":45000
,"max_salary":150000}
```

- DELETE

  Method : DELETE

  http://localhost:8888/ords/orcl/hr/jobs/IT_CONS

AutoREST - Batchload

- Batchload enables loading CSV data into the AutoREST enabled tables.
- Only POST method is supported
- First line in the CSV should contain the Column names
- Date format in the csv data can be specified using the query parameter **dateFormat**
- Sample URL for batchload operation is,

```
http://localhost:8888/ords/orcl/hr/jobs/batchload?dateFormat="DD/MM/YYYY hh24:mi“
```

Payload:

```
job_id,job_title,min_salary,max_salary,created_date
IT_CONS1,Junior IT Consultant,1000,5000,01/01/2018 13:25
IT_CONS2,IT Consultant,3000,8000,02/01/2018 21:54
IT_CONS3,Senior IT Consultant,7000,12000,03/01/2018 09:15
```

Security

API Security & Authentication

Types of Authentication supported by ORDS

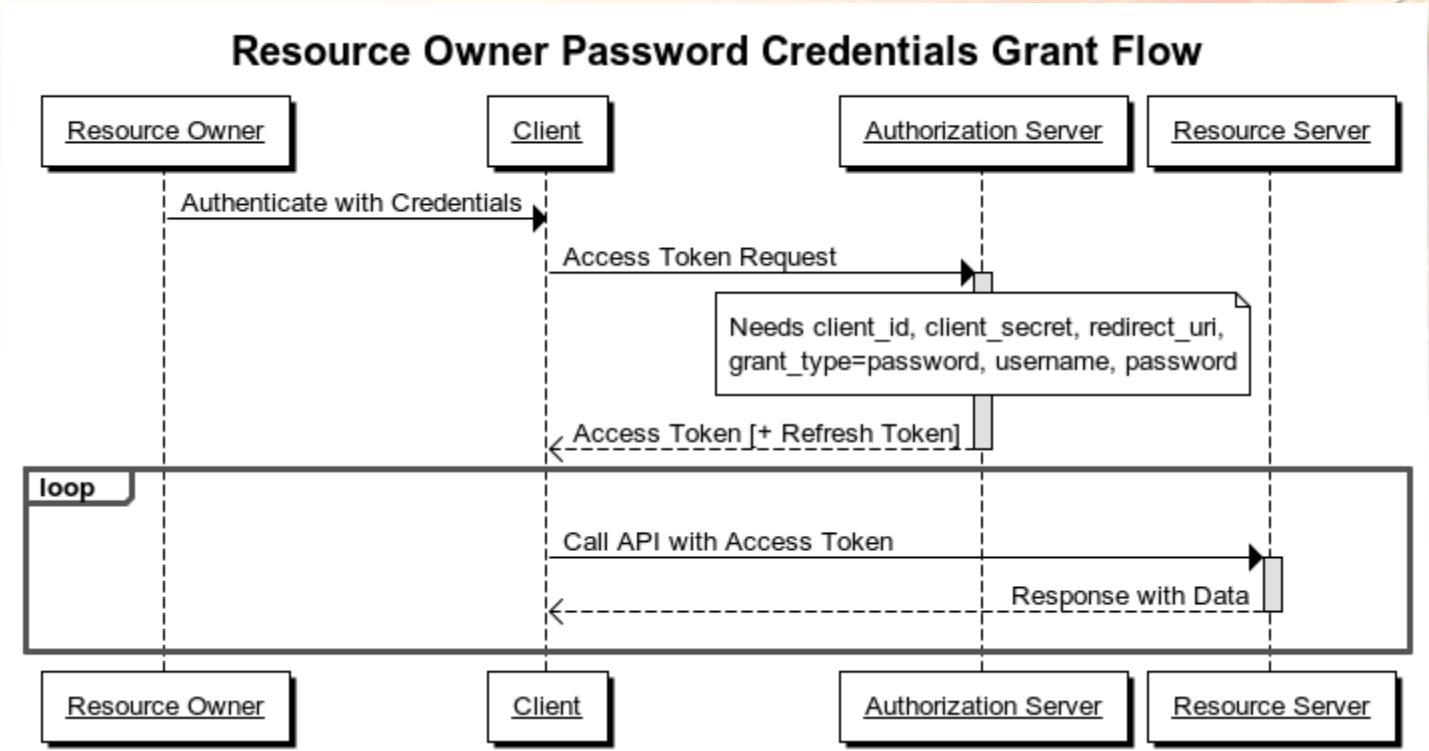
- First Party Authentication or Basic Authentication
  - Create ORDS user and assign roles and privileges to access the API
- OAuth 2.0
  - Resource Owner Credentials
  - Client Credentials
  - Authorization Code
  - Implicit Code

API Security & Authentication [contd.,]

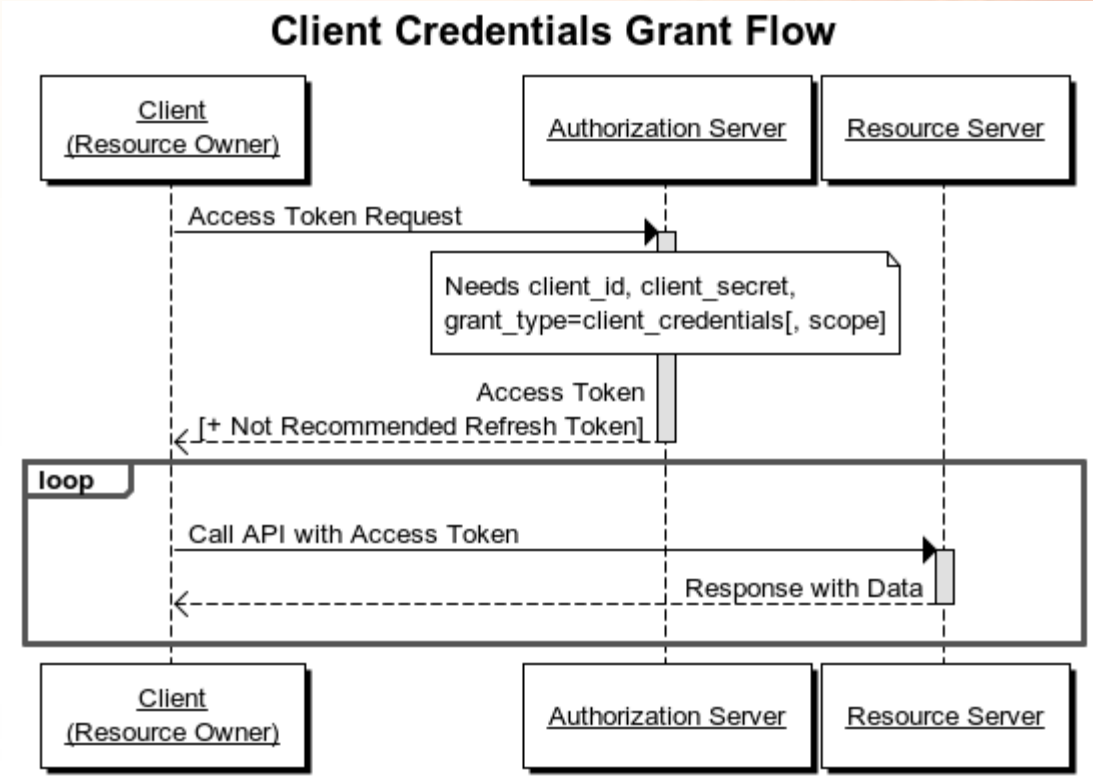
OAuth 2.0

- The OAuth 2.0 protocol (https://tools.ietf.org/html/rfc6749) defines flows to provide conditional and limited access to a RESTful API.
- OAuth 2.0 Authentication Flow types,
  - Authorization Code
    - This is for web applications having its own web server where the client credentials can be stored. The application can use a refresh token to extend the period of access to the api.
  - Client Credentials
    - Gives an application direct access to a RESTful API without requiring a user to approve access to the data managed by the RESTful API.
  - Implicit Code
    - This is suitable for Single Page Applications where client credentials cannot be stored.

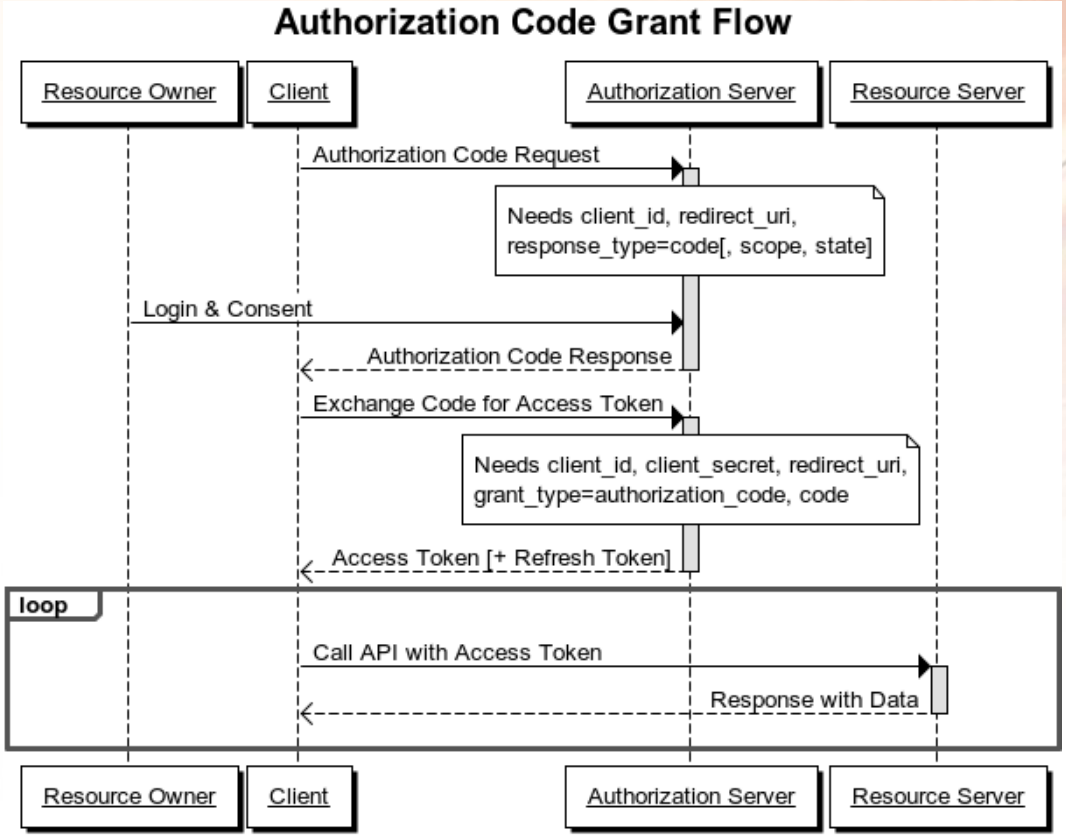
OAuth 2.0 – Authentication Flows



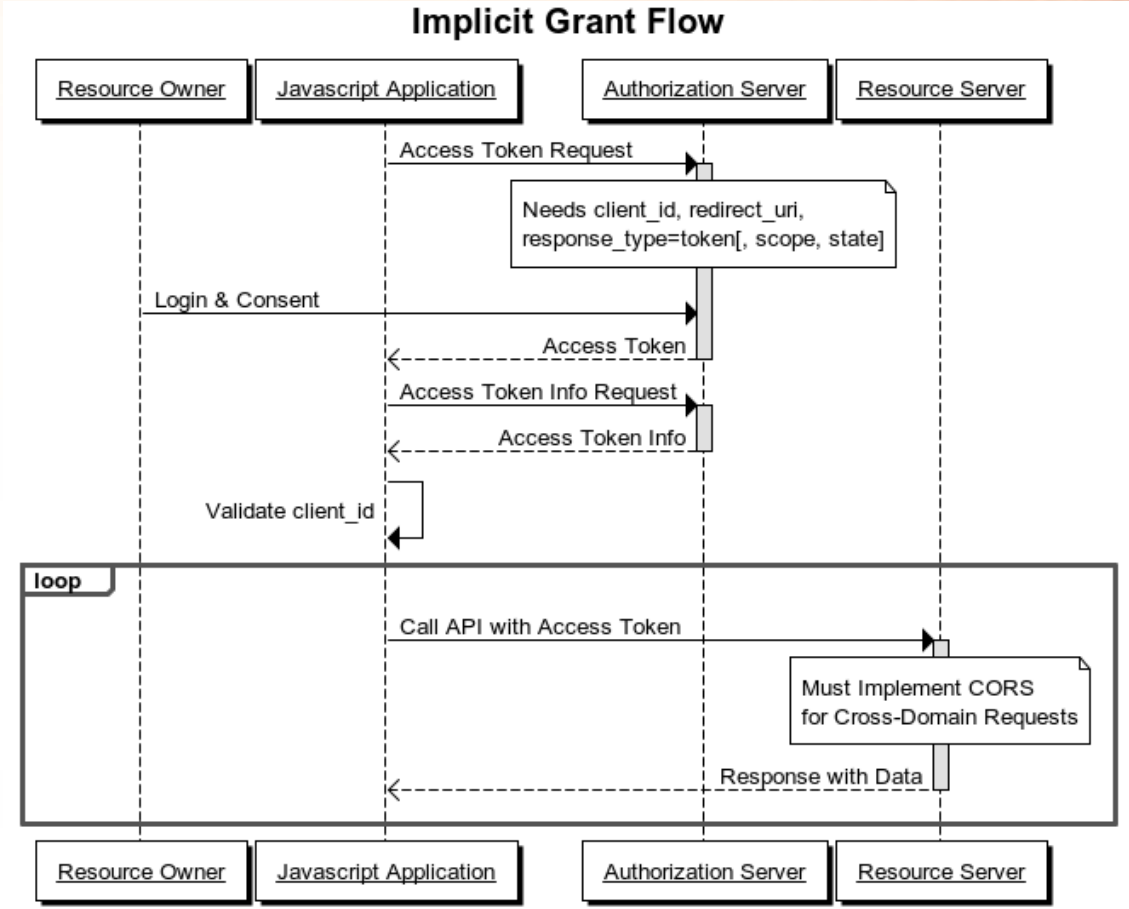
OAuth 2.0 – Authentication Flows



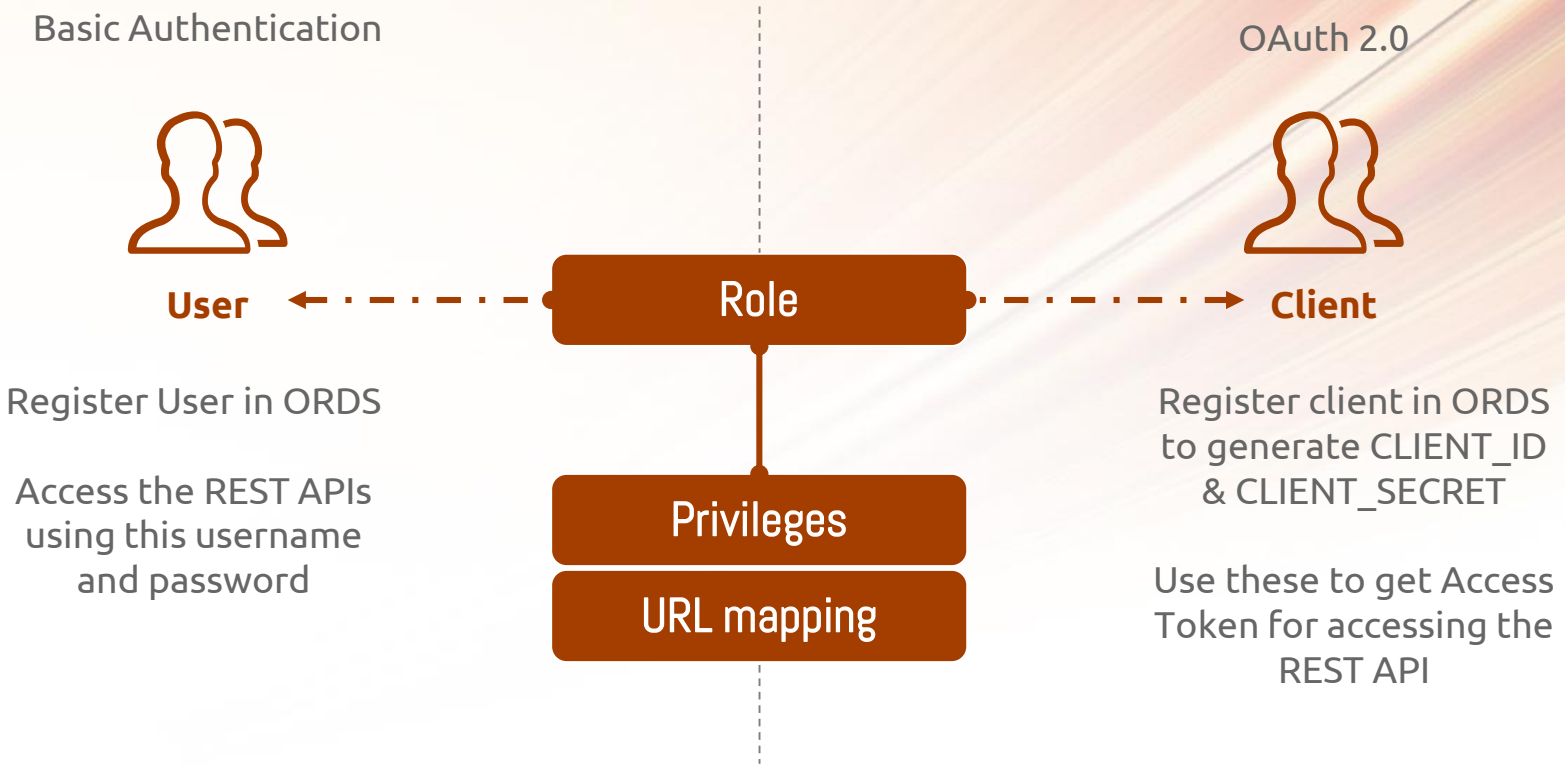
OAuth 2.0 – Authentication Flows



OAuth 2.0 – Authentication Flows



ORDS - Roles and Privileges



ORDS - Roles and Privileges [contd.,]

- Define ORDS Role

```
BEGIN
   ORDS.create_role
        ( p_role_name => 'hrms_role' );
   COMMIT;
END;
```

- Define Privilege

```
DECLARE
   l_arr OWA.vc_arr;
BEGIN
   l_arr(1) := 'hrms_role';
   ORDS.define_privilege
   ( p_privilege_name => 'hrms_prv'
   , p_roles          => l_arr
   , p_label          => 'HRMS Privilege'
   , p_description    => 'Access to HRMS apis' );
   COMMIT;
END;
```

ORDS Metadata:

SEC_ROLES

SEC_PRIVILEGES

SEC_PRIVILEGE_ROLES

USER_ORDS_ROLES

USER_ORDS_PRIVILEGES

USER_ORDS_PRIVILEGE_ROLES

ORDS - Privilege mapped to URL

- Map the Privilege to an URL pattern

```
BEGIN
   ORDS.create_privilege_mapping
   ( p_privilege_name => 'hrms_prv'
   , p_pattern => '/hrms/*'
   );

   COMMIT;
END;
```

ORDS Metadata:

ORDS_PRIVILEGE_MAPPINGS

USER_ORDS_PRIVILEGE_MAPPINGS

ORDS - Basic Authentication

- Create ORDS user with password.

  Execute this command from <ORDS_BASE> path.

  ```
  java -jar ords.war user hrms_usr hrms_role
  ```

- Credentials file is created for Basic Authentication

  ```
  /<ORDS_BASE>/conf/ords/credentials
  ```

Demo…

Questions ?

Thank You

Hariharaputhran & **Justin Michael Raj**
AIOUG Evangelists