Secure RESTful Services

SUMNER technologies

Welcome

About Me

scott@sumnertech.com

@sspendol

ORACLE

Sumner TECHNOLOGIES

accenture

sumnēva Application Success

ORACLE ACE Director

enkitec

SUMNER technologies

SUMNER technologies

About Sumner Technologies

- Originally Established 2005
- Relaunched in 2015
  - Focused exclusively on Oracle APEX solutions
- Provide wide range of APEX related Services
  - Architecture Design & Reviews
  - Security Reviews
  - Health Checks
  - Education
    - On-site, On-line, On-Demand
    - Custom & Mentoring
  - Oracle Database Cloud Consulting
  - Curators of APEX-SERT

SUMNER technologies

ORACLE Gold Partner

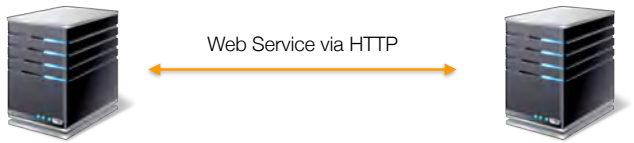
Agenda

- Overview
- Secure Web Services
  - Server
  - Client
- Summary

Overview

Web Services

- Web Services are nothing more than a **procedure that lives on another server**
  - Typically used when two computers exchange data
  - Runs over **HTTP** or **HTTPS**
  - Results typically contain data formatted in either **XML** or **JSON**



Oracle REST Data Services

- **O**racle **R**EST **D**ata **S**ervices
  - Formerly called **Oracle APEX Listener**
  - **Fully Supported** feature of the Oracle Database since 2010
    - Can log SRs against a corresponding Database License Provides HTTP/S Access to Oracle Databases (and other databases)
  - **Maps HTTP(S) RESTful GETS and POSTS to SQL and PL/SQL**
  - Declaratively returns results in JSON or CSV format
- Enables virtually every platform to easily and securely access an Oracle Database

ORDS

ORDS Repositories

- Traditionally, there are actually **two ORDS repositories**
  - One exposed via APEX's SQL Workshop
  - One exposed via ORDS & PL/SQL APIs
- The good news is that in APEX 18.1, the **APEX-based repository will merge into the ORDS repository**
  - Migration tools will be available to consolidate all web services
- We'll focus on the ORDS/API based repository

Postman

- **Postman** is an application that facilitates the development and testing of web services
  - Mac, Windows & Linux
  - Save and share web services w/your team
  - Paid versions offer more features
- Download and try it:
  - https://getpostman.com
- We'll use Postman to illustrate and test web services

Simple Web Service

- Let's create a simple RESTful Service that returns all rows from EMP:
  - **Module**: emp
  - **Template**: emp
  - **Method**: GET
  - **SQL**: `SELECT * FROM emp`

Demo:
Create a Web Service

Web Service URI

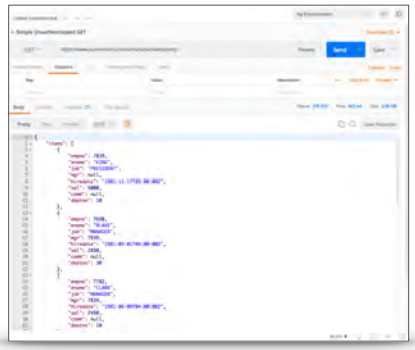
- To test the web service, we need to refer to its URI

  https://servername/ords/schema/module/template

  - **ords**
    - Name of ords.war file; typically ords
  - **schema**
    - Schema Name where web service lives; should be changed so as to not expose the schema name
  - **module**
    - Name of web service module
  - **template**
    - Name of web service template

Testing the Web Service

- In Postman, enter the URI of the web service and click **Send**
  - Result should be a JSON document containing the results of all rows of the EMP table
  - Since this is a GET, you can easily test this in a browser or even command line



Quick Quiz

- With ORDS, there is no easier way to:
  - a) REST-enable your Oracle Database
  - b) Create a massive security risk that could be catastrophic

Answer: c) All of the above

Secure Web Services

Secure Web Services

- Clearly, there has to be a way to secure a web service so that only authorized users can access it
  - And access those which are secured
- **Server-Side**
  - How to create a RESTful web service secured with OAuth
- **Client-Side**
  - How to consume a RESTful web service with secured with OAuth

Server

Dept 10 Web Service

- Let's create another simple RESTful Service that returns Department 10 rows from EMP:
  - **Module**: dept10
  - **Template**: emp
  - **Method**: GET
  - **SQL**: `SELECT * FROM emp WHERE deptno = 10`
- Once created, we can test the new web service in Postman

Demo: Create & Test dept10 Web Service

OAuth2

- ORDS makes use of **OAuth2** to provide secured web services
- OAuth2 is an **authentication framework** that enables applications to use external user credentials
  - Facebook, Google, etc.
  - Delegates all authentication services to the service that hosts the user accounts
  - Better approach, as if you dont' store the credentials, then no one can steal them

OAuth vs. OAuth2

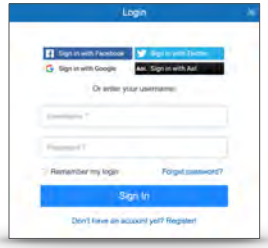
- OAuth2 is a complete re-write of OAuth
  - They are not compatible
  - Most modern systems use OAuth2
- From oauth.com:

> *OAuth 2.0 is a complete rewrite of OAuth 1.0 from the ground up, sharing only overall goals and general user experience. OAuth 2.0 is not backwards compatible with OAuth 1.0 or 1.1, and should be thought of as a completely new protocol.*

OAuth2 - As Seen on the Internet

- You have probably used OAuth2 at some point
- Any site that allows you to login as Facebook, Google, etc. is using OAuth2
  - Credentials stay at the source; only whether or not you have successfully logged in is sent
  - As well as other information

Cambridge Analytica



Adding OAuth2 to a Web Service

- This process is a **bit tricky**
  - Data model & components are a bit strange
  - Configuration is disjointed
    - About 3/4 can be configured via SQL Developer
    - Final 1/4 needs PL/SQL API calls
- **But when done correctly, it works flawlessly**
  - Takes some getting used to

Adding OAuth2 to a Web Service

- Required components:
  - **Module**
  - **Role**
  - **Privilege**
  - **Client**
    - Can only be configured via SQL Plus
- Trick is to **associate them together properly** so that the web service is protected

Roles

- Roles have a single property: **name**
  - Similar to a database role - it alone is meaningless
- Once created, roles can be associated with both **Privileges** and **Clients**
  - This will be the link that protects our web service with a specific set of client credentials, not just any set

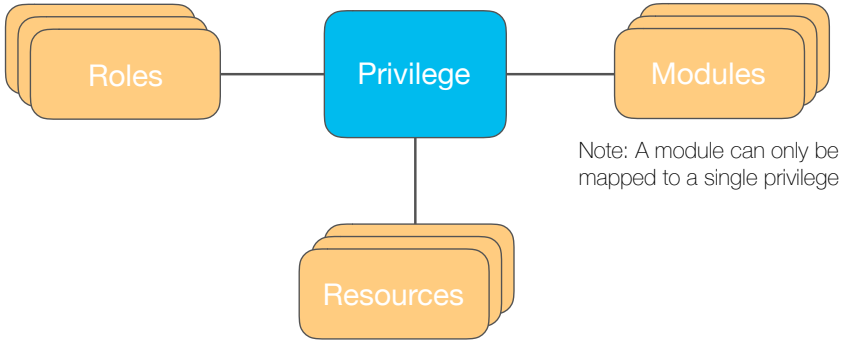
Roles

Demo:
Create a Role

Privileges

- Similar to **Database Privileges**
  - **Grant Select On, Grant Delete On, etc.**
- Used to limit access:
  - By **Module**
  - By **Resource** (URL pattern)
- **A module can only be associated with a single privilege**
  - UI **will not warn you** of this restriction, but rather remove the modules from the other privilege automatically

Privileges



Note: A module can only be mapped to a single privilege

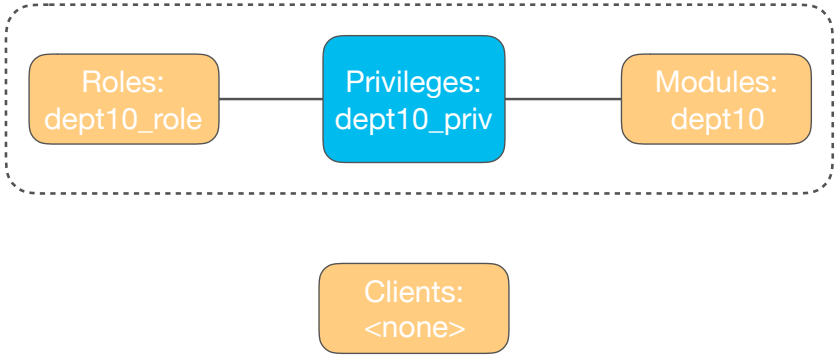
Demo:
Create a Privilege

Testing the Web Service

- Now that the module is associated with a Privilege, try to reload it via Postman
  - Should receive a **401: Unauthorized** error

Unauthorized

- Because the Module is associated with the Privilege, ORDS will not let just anyone view it anymore



Clients

- Credential set that can be associated with roles, privileges or both
- ORDS offers **native support of OAuth2 clients** via a set of APIs
  - Unlike the rest of ORDS, there is **no GUI support for clients** via SQL Developer
  - Must use APIs & Views to manage

Clients

Creating an OAuth2 Client

```
oauth.create_client
  (
  p_name              VARCHAR2 IN,
  p_grant_type        VARCHAR2 IN,
  p_owner             VARCHAR2 IN DEFAULT NULL,
  p_description       VARCHAR2 IN DEFAULT NULL,
  p_allowed_origins   VARCHAR2 IN DEFAULT NULL,
  p_redirect_uri      VARCHAR2 IN DEFAULT NULL,
  p_support_email     VARCHAR2 IN,
  p_support_uri       VARCHAR2 IN DEFAULT NULL,
  p_privilege_names   VARCHAR2 IN DEFAULT NULL
  );
```


Creating a Client for Dept 10

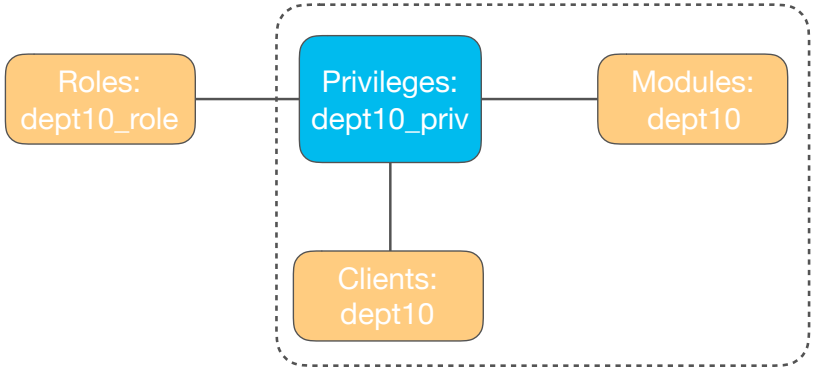
- Let's create a client and map that client to **dept10_priv**

```
BEGIN
oauth.create_client
  (
  p_name              => 'dept10',
  p_grant_type        => 'client_credentials',
  p_description       => 'Department 10',
  p_support_email     => 'dept10@sumnertech.com',
  p_privilege_names   => 'dept10_priv'
  );
COMMIT;
END;
/
```

Privilege mapped to client

Almost there…

- Before we can access the web service, the client and the role need to be associated



Associating a Client & Role

- The only way to associate **Clients** and **Roles** is via API

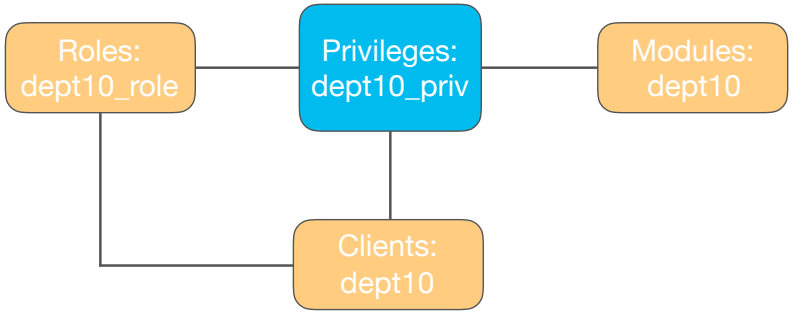
```
oauth.grant_client_role
  (
  p_client_name => 'dept10',
  p_role_name   => 'dept10_role'
  );
```

- Revoking is done the same way

```
oauth.revoke_client_role
  (
  p_client_name => 'dept10',
  p_role_name   => 'dept10_role'
  );
```

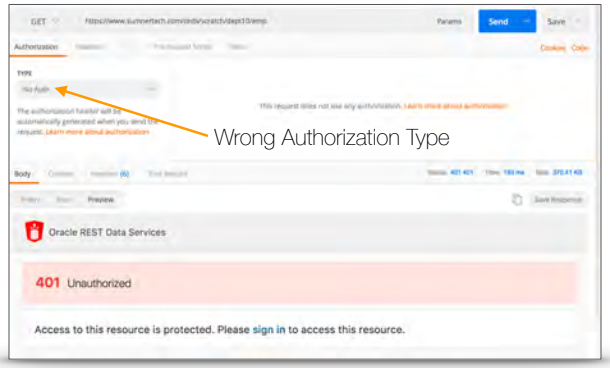

Success

- Clients need to be mapped to a role and a privilege
  - Yes, it is redundant
  - No, it doesn't care; this is how it works



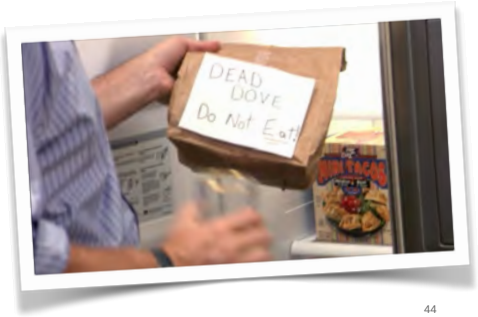
Testing the Web Service

- Now that all required components are associated, reload the web service in Postman



OAuth2 Credentials

- At this point, we're **requesting a protected web service but not providing any credentials**
  - Not sure what you expected…
- In order to access the web service, we must first **provide the OAuth2 credentials** before calling the web service



OAuth2 Credentials

- Before we can access the **dept10** module, we need to provide the client credentials of the **dept10** client
  - **Grant Type**
  - **Access Token URL**
  - **Client ID**
  - **Client Secret**
  - **Client Authentication**

OAuth2 Credentials

- The token will be returned and stored in Postman
  - Click **Use Token** to use this token when making a request

Success

- At this point, the **dept10 module is secured via the dept10 client**
  - Users will need to provide a valid token to access it
  - A valid token can only be obtained by presenting the Client ID & Client Secret to ORDS before making the request

DEPT20 Web Service

- Next, we'll create a **similar web service** based on **department 20**
  - Everything will be the same, but it will have its own role, privilege and client
- Once created, we will need to use its specific **Client ID & Client Secret to get a token** before we can query it
  - Which will be a different client that dept10

Demo:
Create & Test dept20 Web Service

Client

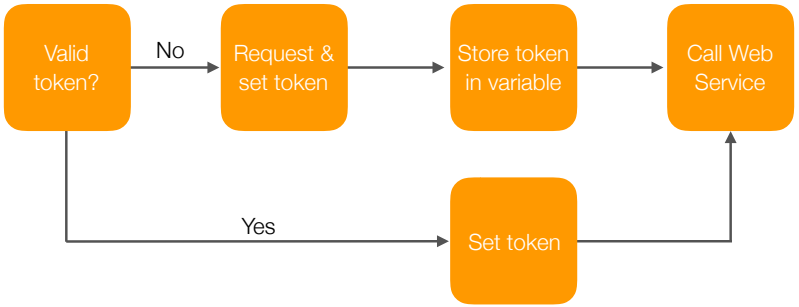
Client

- Now that the web services are secured with their individual OAuth2 clients, we **need a way to access them** outside of Postman
  - In this case, we'll use APEX as the client

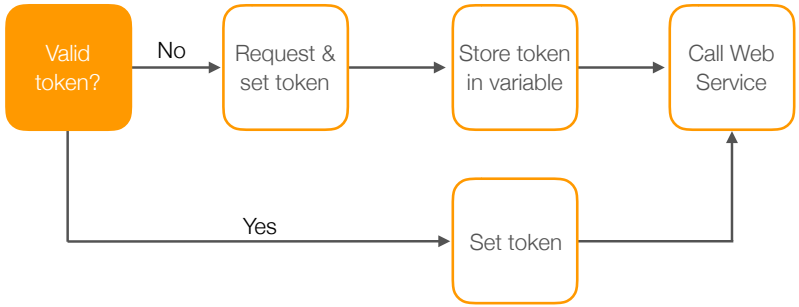
Tokens

- With OAuth2, the **client will use the client credentials to request a token**
  - That token will be returned to the client
  - It will be good for some set amount of time
    - Default for ORDS is 3600 seconds or 1 hour
- **Each request** made from the client to the server **must contain a valid, unexpired token**
  - This will prove that the client is in fact, authorized to access the web service

Token Request Flow



Token Request Flow



Validate Token

- The first step is to determine **whether or not we have a valid token**
  - Token would have been requested and stored in an application item previously
- Need to also check to see that if we do have one, it is **not expired**
  - Token "created on" time would have also been stored in an application item and can be compared to the current time

Validate Token

- This validation code **must be called before each request to the web service**
  - Since APEX sessions are not mapped to database sessions, it also must be called for each page view and/or asynchronous transaction
- Best to put this logic into a procedure and **call before the page renders** and at the **Initialization PL/SQL Code**
  - Shared Components > Security

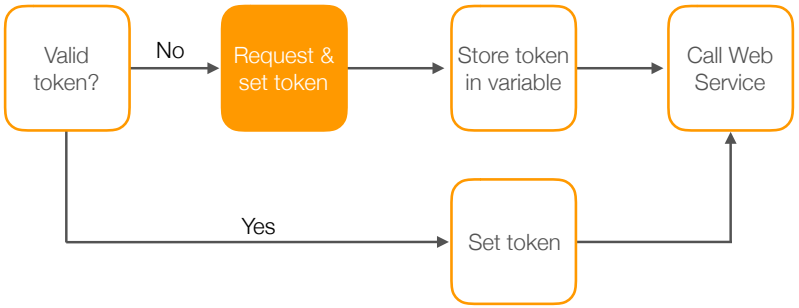
Validate Token

- A lot of APEX applications today are **asynchronous**
  - Partial page refresh, any action on an IR or IG, region refreshes
- These **Ajax-based PL/SQL calls also need a token** if they will touch the web service
  - Thus, we can use the **Initialization PL/SQL Code**
  - This snippet gets calls before any APEX processing - full or partial page

Validate Token

```
IF p_token IS NULL OR SYSDATE >
  TO_DATE(p_token_expires,'DD-MON-YYYY HH24:MI:SS')
THEN

  -- Fetch the credentials

  -- Get a new token

  -- Set the token and expiry time

ELSE

  -- Use the existing token

END IF;
```


Token Request Flow



Requesting a Token

- To request a token, we can use the **APEX_WEB_SERVICE.OAUTH_AUTHENTICATE** API

```
apex_web_service.oauth_authenticate
  (
  p_token_url         IN VARCHAR2,
  p_client_id         IN VARCHAR2,
  p_client_secret     IN VARCHAR2,
  p_flow_type         IN VARCHAR2 DEFAULT oauth_cred,
  p_proxy_override    IN VARCHAR2 DEFAULT NULL,
  p_transfer_timeout IN NUMBER DEFAULT 180,
  p_wallet_path       IN VARCHAR2 DEFAULT NULL,
  p_wallet_pwd        IN VARCHAR2 DEFAULT NULL
  );
```

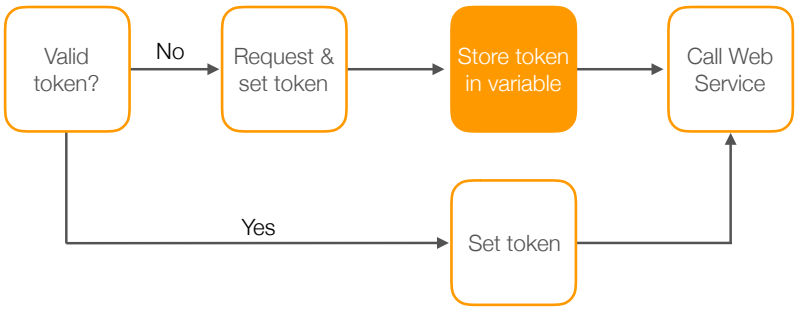
Requesting a Token

- Requesting a token **will also set that token for use in the next web service call**
  - No need to set it again
- Thus, we can skip the call to **APEX_WEB_SERVICE.OAUTH_SET_TOKEN** when requesting a new token

Requesting a Token

```
apex_web_service.oauth_authenticate
  (
  p_token_url     => apex_util.get_session_state('G_URL_TOKEN'),
  p_client_id     => x.clientid,
  p_client_secret => x.clientsecret,
  p_wallet_path   => apex_util.get_session_state('G_WALLET_PATH'),
  p_wallet_pwd    => apex_util.get_session_state('G_WALLET_PASSWORD')
  );
```


Token Request Flow



Storing a Token in a Variable

- It's a good idea to **store this token in a variable** and simply **re-use it for each request**
  - Provided that it is not expired
- Thus, when the token is stored, we can also **capture the current date & time** and inspect that to see if we need to request a new token or not
  - Token duration value will vary by site
  - ORDS default is 3600; can be changed via a parameter
  - Your mileage will vary with other sites

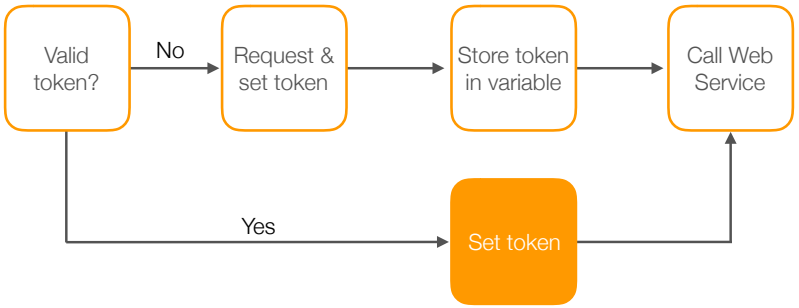
Storing a Token in a Variable

- To do this, call the API **APEX_WEB_SERVICE.OAUTH_GET_LAST_TOKEN** to get the last token which was returned
  - API has no parameters
  - Simply return the results of the call into the variable or item that the token will be stored in
  - Should also **store the time the token was requested**

Storing a Token in a Variable

```
apex_util.set_session_state
  (
  'G_TOKEN_' || p_key,
  apex_web_service.oauth_get_last_token
  );

apex_util.set_session_state
  (
  'G_TOKEN_' || p_key || '_EXPIRES',
  TO_CHAR((SYSDATE+1/24),'DD-MON-YYYY HH24:MI:SS')
  );

apex_util.set_session_state
  (
  'G_TOKEN_' || p_key || '_EXPIRES_DISP',
  apex_util.get_since((SYSDATE+1/24),'DD-MON-YYYY HH24:MI:SS')
  );
```


Token Request Flow



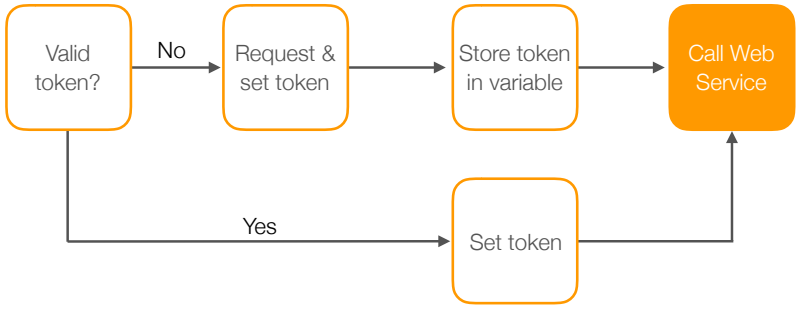
Set the Token

- If the token is valid and not expired, we **do not need to get a new one**
  - But we will need to set that token before calling the web service
- This is done via the **APEX_WEB_SERVICE.OAUTH_SET_TOKEN** API call
  - Takes in a single parameter - the token

Set the Token

```
apex_web_service.oauth_set_token
  (
  p_token => p_token
  );
```


Token Request Flow



Call the Web Service

- The **APEX_WEB_SERVICE** API is a robust set of APIs that make consuming a web service trivial
  - Via APEX or even PL/SQL
- Can call either **SOAP or RESTful** web services
  - Parse responses and encode/decode them
- Manages **cookies & HTTP Headers**
- Provides built-in **support for authentication**
  - **Basic Authentication**
  - **OAuth2**

MAKE_REST_REQUEST

```
APEX_WEB_SERVICE.MAKE_REST_REQUEST(
  p_url              IN VARCHAR2,
  p_http_method      IN VARCHAR2,
  p_username         IN VARCHAR2 DEFAULT NULL,
  p_password         IN VARCHAR2 DEFAULT NULL,
  p_scheme           IN VARCHAR2 DEFAULT 'Basic',
  p_proxy_override   IN VARCHAR2 DEFAULT NULL,
  p_transfer_timeout IN NUMBER DEFAULT 180,
  p_body             IN CLOB DEFAULT EMPTY_CLOB(),
  p_body_blob        IN BLOB DEFAULT EMPTY_BLOB(),
  p_parm_name        IN apex_application_global.VC_ARR2
                        DEFAULT empty_vc_arr,
  p_parm_value       IN apex_application_global.VC_ARR2
                        DEFAULT empty_vc_arr,
  p_wallet_path      IN VARCHAR2 DEFAULT NULL,
  p_wallet_pwd       IN VARCHAR2 DEFAULT NULL);
```

Capturing Data

- Two things you can do with the data
  - **Capture it and store it in a collection/table**
    - Build APEX reports off of that collection/table
  - Display it real-time in a report
    - Via **APEX_JSON** or **JSON_TABLE**
- Which you use depends on a number of things
  - Business Requirements
  - Data Consistency
  - Performance
  - Network

Embed MAKE_REST_REQUEST in SQL

- The **MAKE_REST_REQUEST** returns a JSON document
- Thus, it can be **embedded in a SQL statement** and processed by either **APEX_JSON** (11g+) or **JSON_TABLE** (12c+)
  - **APEX_JSON** will convert the JSON to XML and then use XMLTABLE
  - **JSON_TABLE** works natively with JSON
- SQL statement can easily be the **source of any APEX component**
  - Chart, report, calendar, interactive grid, etc.

SQL using APEX_JSON.TO_XMLTYPE (11g+)

```
SELECT
  x.*
FROM
  xmltable
    (
    '/json/items/row'
    PASSING apex_json.to_xmltype(
      apex_web_service.make_rest_request
      (
      p_url         => :G_URL_EMP,
      p_http_method => 'GET',
      p_wallet_path => :G_WALLET_PATH,
      p_wallet_pwd  => :G_WALLET_PASSWORD
      )
    )
    COLUMNS
      ename    VARCHAR2(4000) PATH 'ename',
      empno    NUMBER         PATH 'empno',
      job      VARCHAR2(255)  PATH 'job',
      mgr      NUMBER         PATH 'mgr',
      hiredate VARCHAR2(255)  PATH 'hiredate',
      sal      NUMBER         PATH 'sal',
      deptno   NUMBER         PATH 'deptno'
    ) x
```


SQL using JSON_TABLE (12c+)

```
SELECT
  jt.*
FROM
  JSON_TABLE
    (
    apex_web_service.make_rest_request
      (
      p_url         => :G_URL_EMP,
      p_http_method => 'GET',
      p_wallet_path => :G_WALLET_PATH,
      p_wallet_pwd  => :G_WALLET_PASSWORD
      ),
    '$'
    COLUMNS
      (
      ename    VARCHAR2(4000) PATH '$.ename',
      empno    NUMBER         PATH '$.empno',
      job      VARCHAR2(255)  PATH '$.job',
      mgr      NUMBER         PATH '$.mgr',
      hiredate VARCHAR2(255)  PATH '$.hiredate',
      sal      NUMBER         PATH '$.sal',
      deptno   NUMBER         PATH '$.deptno'
      )
    ) jt
```

APEX 18.1 Web Sources

- APEX 18.1 make it even easier with **Web Sources**
  - 100% declarative, no-code mechanism to consume web services
  - Web Sources can be used as the source of reports, charts & calendars

Demo:
APEX 18.1 Web Sources

Summary

Summary

- **It's not "if"** web services skills will become essential to developers - **it's "when"**
  - And "when" was probably a couple of years ago
- There is **no easier way** to create a RESTful web service than with **ORDS** & no easier way to consume that web service than with **APEX**
  - APEX 18.1 makes it even easier
- However, doing this **could create a monumental security risk**, if you don't take the time to ensure that all web services are adequately secured
  - When properly secured - which is not difficult - there is no risk


SUMNER
technologies