At a high level, we will be consuming data, preparing it for delivery and placing each message ( drink orders for Sippy, Food orders by Flippy, coffee for CookRight coffee or any other product) into specific locations in a database, accessible via REST API into defined queues , waiting to get called for message delivery by downstream systems. These systems will take message information about actual orders or forcasting of orders and we will have accountability of the events that occurred for each event the queue supports, there will be many queues to manage and report on.

Unlike Asynchronous queues offered by the solution below that is the subject of this document, “GCP Cloud Tasks Services” are designed for asynchronous work but do not provide strong guarantees around the timing of task delivery and is therefore unsuitable for interactive applications where a user is waiting for the result or timeouts occure in the transaction (timings provided later) (<https://cloud.google.com/tasks/docs/dual-overview#use_cases>)

**Support and Recoverability of our proposed solution.**

Once messages are added to these queues, the queue dispatches them and makes sure they are reliably processed by our Miso processes. Complexities associated with that process, such as user-facing latency, server crashes, resource consumption and retry management, are handled by the service.

The tasks themselves are made up of a unique name and configuration information and, optionally, any data from the initial request, called the payload, necessary to process the request.

**The limitations of Cloud Tasks queues with HTTP targets is as follows.**

In the case of generic HTTP Targets, the Cloud Tasks service forwards the task request to the worker, located at any generic HTTP endpoint, based on how the task is configured. This endpoint could be on [Cloud Functions](https://cloud.google.com/functions/docs), [Cloud Run](https://cloud.google.com/run/docs), [GKE](https://cloud.google.com/kubernetes-engine/docs), [Compute Engine](https://cloud.google.com/compute/docs), or even an on-prem web server, based on how the task is configured. These queues dispatch requests at a reliable, configurable rate. They guarantee reliable task execution - upon success, all workers must send an HTTP response code (200-299) to the Cloud Tasks service before the default timeout deadline of 10 minutes, with a maximum of 30 minutes. If a different response is sent, or no response, the task is retried.

Diagram

Description automatically generated

The target must manage scaling workers and cleaning up tasks once they are complete.

We will be implementing this solution in the Cloud, below is the entire installation and configuration list of topics.

Initial login page.

A screenshot of a computer

Description automatically generated with medium confidence

<https://www.oracle.com/cloud/sign-in.html>

cloud account name = oracleapexoci

Graphical user interface, application

Description automatically generated

Identity provider = oracleidentitycloudservice

Any valid email address could have been used.

Graphical user interface, application

Description automatically generated

Username = [oracleapexoci@gmail.com](mailto:oracleapexoci@gmail.com)

MisoYouOracle1!

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Once logged in, click on the 3 bars knows as a BURGER to the top left of the word ORACLE Cloud

A screenshot of a computer

Description automatically generated

Burger

A picture containing text, monitor, screenshot, screen

Description automatically generated

Select autonomous database, below find two databases set up.

Graphical user interface, text, application

Description automatically generated

We will be using -> [DB 202110212330](https://cloud.oracle.com/db/adb/ocid1.autonomousdatabase.oc1.phx.anyhqljsx7hjfgqaog5qywzjmdtaiqjtqpns5jxdxv4gvqvbyajucqdvo7hq)

Graphical user interface, text, application, email

Description automatically generated

A great number of features are supported but take care in not making changes outside the queue creation projess. Graphical user interface, application

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

Under Service console you can select

Graphical user interface, text, application, Word

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

To explained later

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Dashboards give you real time data feedback.

Graphical user interface, application

Description automatically generated

Admin/admin

<https://gd49ef7d5e3cd2d-db202110212330.adb.us-phoenix-1.oraclecloudapps.com/ords/>

Graphical user interface

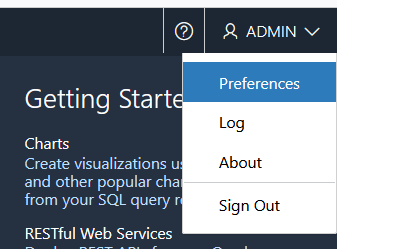
Description automatically generated

A screenshot of a computer

Description automatically generated

Select Database Actions to the right of the ICON APX in GREEN.

Here is the Actions main page. Confirm you are logged in as an admin , if not logout and log back in.



Graphical user interface, application

Description automatically generated

Click or select SQL , (Under the Development text in bold).

The below code will reverse engineer the Access Control List, to inspect what is there and to migrate from other environments.

SET SERVEROUTPUT ON FORMAT WRAPPED LINESIZE 300

CREATE OR REPLACE EDITIONABLE PROCEDURE "ADMIN"."SHOW\_ACLS" AS

--DECLARE

l\_last\_acl dba\_network\_acls.acl%TYPE := '~';

l\_last\_principal dba\_network\_acl\_privileges.principal%TYPE := '~';

l\_last\_privilege dba\_network\_acl\_privileges.privilege%TYPE := '~';

l\_last\_host dba\_network\_acls.host%TYPE := '~';

FUNCTION get\_timestamp (p\_timestamp IN TIMESTAMP WITH TIME ZONE)

RETURN VARCHAR2

AS

l\_return VARCHAR2(32767);

BEGIN

IF p\_timestamp IS NULL THEN

RETURN 'NULL';

END IF;

RETURN 'TO\_TIMESTAMP\_TZ(''' || TO\_CHAR(p\_timestamp, 'DD-MON-YYYY HH24:MI:SS.FF TZH:TZM') || ''',''DD-MON-YYYY HH24:MI:SS.FF TZH:TZM'')';

END;

BEGIN

FOR i IN (SELECT a.acl,

a.host,

a.lower\_port,

a.upper\_port,

b.principal,

b.privilege,

b.is\_grant,

b.start\_date,

b.end\_date

FROM dba\_network\_acls a

JOIN dba\_network\_acl\_privileges b ON a.acl = b.acl

ORDER BY a.acl, a.host, a.lower\_port, a.upper\_port)

LOOP

IF l\_last\_acl <> i.acl THEN

-- First time we've seen this ACL, so create a new one.

l\_last\_host := '~';

DBMS\_OUTPUT.put\_line('-- -------------------------------------------------');

DBMS\_OUTPUT.put\_line('-- ' || i.acl);

DBMS\_OUTPUT.put\_line('-- -------------------------------------------------');

DBMS\_OUTPUT.put\_line('BEGIN');

DBMS\_OUTPUT.put\_line(' DBMS\_NETWORK\_ACL\_ADMIN.drop\_acl (');

DBMS\_OUTPUT.put\_line(' acl => ''' || i.acl || ''');');

DBMS\_OUTPUT.put\_line(' COMMIT;');

DBMS\_OUTPUT.put\_line('END;');

DBMS\_OUTPUT.put\_line('/');

DBMS\_OUTPUT.put\_line(' ');

DBMS\_OUTPUT.put\_line('BEGIN');

DBMS\_OUTPUT.put\_line(' DBMS\_NETWORK\_ACL\_ADMIN.create\_acl (');

DBMS\_OUTPUT.put\_line(' acl => ''' || i.acl || ''',');

DBMS\_OUTPUT.put\_line(' description => ''' || i.acl || ''',');

DBMS\_OUTPUT.put\_line(' principal => ''' || i.principal || ''',');

DBMS\_OUTPUT.put\_line(' is\_grant => ' || i.is\_grant || ',');

DBMS\_OUTPUT.put\_line(' privilege => ''' || i.privilege || ''',');

DBMS\_OUTPUT.put\_line(' start\_date => ' || get\_timestamp(i.start\_date) || ',');

DBMS\_OUTPUT.put\_line(' end\_date => ' || get\_timestamp(i.end\_date) || ');');

DBMS\_OUTPUT.put\_line(' COMMIT;');

DBMS\_OUTPUT.put\_line('END;');

DBMS\_OUTPUT.put\_line('/');

DBMS\_OUTPUT.put\_line(' ');

l\_last\_acl := i.acl;

l\_last\_principal := i.principal;

l\_last\_privilege := i.privilege;

END IF;

IF l\_last\_principal <> i.principal

OR (l\_last\_principal = i.principal AND l\_last\_privilege <> i.privilege) THEN

-- Add another principal to an existing ACL.

DBMS\_OUTPUT.put\_line('BEGIN');

DBMS\_OUTPUT.put\_line(' DBMS\_NETWORK\_ACL\_ADMIN.add\_privilege (');

DBMS\_OUTPUT.put\_line(' acl => ''' || i.acl || ''',');

DBMS\_OUTPUT.put\_line(' principal => ''' || i.principal || ''',');

DBMS\_OUTPUT.put\_line(' is\_grant => ' || i.is\_grant || ',');

DBMS\_OUTPUT.put\_line(' privilege => ''' || i.privilege || ''',');

DBMS\_OUTPUT.put\_line(' start\_date => ' || get\_timestamp(i.start\_date) || ',');

DBMS\_OUTPUT.put\_line(' end\_date => ' || get\_timestamp(i.end\_date) || ');');

DBMS\_OUTPUT.put\_line(' COMMIT;');

DBMS\_OUTPUT.put\_line('END;');

DBMS\_OUTPUT.put\_line('/');

DBMS\_OUTPUT.put\_line(' ');

l\_last\_principal := i.principal;

l\_last\_privilege := i.privilege;

END IF;

IF l\_last\_host <> i.host||':'||i.lower\_port||':'||i.upper\_port THEN

DBMS\_OUTPUT.put\_line('BEGIN');

DBMS\_OUTPUT.put\_line(' DBMS\_NETWORK\_ACL\_ADMIN.assign\_acl (');

DBMS\_OUTPUT.put\_line(' acl => ''' || i.acl || ''',');

DBMS\_OUTPUT.put\_line(' host => ''' || i.host || ''',');

DBMS\_OUTPUT.put\_line(' lower\_port => ' || NVL(TO\_CHAR(i.lower\_port),'NULL') || ',');

DBMS\_OUTPUT.put\_line(' upper\_port => ' || NVL(TO\_CHAR(i.upper\_port),'NULL') || ');');

DBMS\_OUTPUT.put\_line(' COMMIT;');

DBMS\_OUTPUT.put\_line('END;');

DBMS\_OUTPUT.put\_line('/');

DBMS\_OUTPUT.put\_line(' ');

l\_last\_host := i.host||':'||i.lower\_port||':'||i.upper\_port;

END IF;

END LOOP;

END;

/

Paste the above code into the editor, click run and select the dbms\_output window

Graphical user interface, text, application

Description automatically generated

Result from running the above script

-- -------------------------------------------------  
-- /sys/acls/apex21\_MISO\_open\_acl\_file.xml  
-- -------------------------------------------------  
BEGIN  
  DBMS\_NETWORK\_ACL\_ADMIN.drop\_acl (  
    acl          => '/sys/acls/apex21\_MISO\_open\_acl\_file.xml');  
  COMMIT;  
END;  
/  
   
BEGIN  
  DBMS\_NETWORK\_ACL\_ADMIN.create\_acl (  
    acl          => '/sys/acls/apex21\_MISO\_open\_acl\_file.xml',  
    description  => '/sys/acls/apex21\_MISO\_open\_acl\_file.xml',  
    principal    => 'APEX\_210100',  
    is\_grant     => true,  
    privilege    => 'connect',  
    start\_date   => TO\_TIMESTAMP\_TZ('10-FEB-2022 19:17:27.087684000 +00:00','DD-  
MON-YYYY HH24:MI:SS.FF TZH:TZM'),  
    end\_date     => NULL);  
  COMMIT;  
END;  
/  
   
BEGIN  
  DBMS\_NETWORK\_ACL\_ADMIN.assign\_acl (  
    acl         => '/sys/acls/apex21\_MISO\_open\_acl\_file.xml',  
    host        => '\*',  
    lower\_port  => 1,  
    upper\_port  => 9999);  
  COMMIT;  
END;  
/  
   
-- -------------------------------------------------  
-- NETWORK\_ACL\_D2DB0EE691B34C2DE053B310000A0394  
-- -------------------------------------------------  
BEGIN  
  DBMS\_NETWORK\_ACL\_ADMIN.drop\_acl (  
    acl          => 'NETWORK\_ACL\_D2DB0EE691B34C2DE053B310000A0394');  
  COMMIT;  
END;  
/  
   
BEGIN  
  DBMS\_NETWORK\_ACL\_ADMIN.create\_acl (  
    acl          => 'NETWORK\_ACL\_D2DB0EE691B34C2DE053B310000A0394',  
    description  => 'NETWORK\_ACL\_D2DB0EE691B34C2DE053B310000A0394',  
    principal    => 'ADMIN',  
    is\_grant     => true,  
    privilege    => 'smtp',  
    start\_date   => NULL,  
    end\_date     => NULL);  
  COMMIT;  
END;  
/  
   
BEGIN  
  DBMS\_NETWORK\_ACL\_ADMIN.assign\_acl (  
    acl         => 'NETWORK\_ACL\_D2DB0EE691B34C2DE053B310000A0394',  
    host        => '\*',  
    lower\_port  => NULL,  
    upper\_port  => NULL);  
  COMMIT;  
END;  
/  
   
BEGIN  
  DBMS\_NETWORK\_ACL\_ADMIN.add\_privilege (  
    acl       => 'NETWORK\_ACL\_D2DB0EE691B34C2DE053B310000A0394',  
    principal => 'MISO',  
    is\_grant  => true,  
    privilege => 'smtp',  
    start\_date   => NULL,  
    end\_date     => NULL);  
  COMMIT;  
END;  
/  
   
BEGIN  
  DBMS\_NETWORK\_ACL\_ADMIN.add\_privilege (  
    acl       => 'NETWORK\_ACL\_D2DB0EE691B34C2DE053B310000A0394',  
    principal => 'APEX\_210100',  
    is\_grant  => true,  
    privilege => 'smtp',  
    start\_date   => NULL,  
    end\_date     => NULL);  
  COMMIT;  
END;  
/  
   
BEGIN  
  DBMS\_NETWORK\_ACL\_ADMIN.add\_privilege (  
    acl       => 'NETWORK\_ACL\_D2DB0EE691B34C2DE053B310000A0394',  
    principal => 'MISO',  
    is\_grant  => true,  
    privilege => 'smtp',  
    start\_date   => NULL,  
    end\_date     => NULL);  
  COMMIT;  
END;  
/  
   
BEGIN  
  DBMS\_NETWORK\_ACL\_ADMIN.assign\_acl (  
    acl         => 'NETWORK\_ACL\_D2DB0EE691B34C2DE053B310000A0394',  
    host        => 'smtp.email.us-phoenix-1.oci.oraclecloud.com',  
    lower\_port  => 587,  
    upper\_port  => 587);  
  COMMIT;  
END;  
/  
   
BEGIN  
  DBMS\_NETWORK\_ACL\_ADMIN.add\_privilege (  
    acl       => 'NETWORK\_ACL\_D2DB0EE691B34C2DE053B310000A0394',  
    principal => 'APEX\_210100',  
    is\_grant  => true,  
    privilege => 'smtp',  
    start\_date   => NULL,  
    end\_date     => NULL);  
  COMMIT;  
END;  
/  
   
BEGIN  
  DBMS\_NETWORK\_ACL\_ADMIN.add\_privilege (  
    acl       => 'NETWORK\_ACL\_D2DB0EE691B34C2DE053B310000A0394',  
    principal => 'ADMIN',  
    is\_grant  => true,  
    privilege => 'smtp',  
    start\_date   => NULL,  
    end\_date     => NULL);  
  COMMIT;  
END;  
/

PL/SQL procedure successfully completed.  
  
Elapsed: 00:00:00.156

This code must be run as Admin as the OCI cluster in the cloud does not have a sys or system account.

In this world of distributed architectures, it’s critical that services communicate with each other to ensure the application's business logic is implemented properly. It’s well established that messaging is crucial for modern applications, so let’s look at a messaging solution that exists in the Oracle Cloud that you may not be aware of, Advanced Queueing (AQ).

What’s AQ? It’s exactly what it sounds like - a full-featured messaging solution right inside the database. Point-to-point, pub/sub, persistent, and non-persistent messaging - all supported. There are tons of ways to interact - including via PL/SQL, JMS, JDBC, .NET, Python, Node.JS - pretty much any popular language can interface with AQ. Demos tend to be the best way to understand concepts like this, so in this post, we’re going to look at how to enable AQ in your Autonomous DB instance, create a queue, and enqueue and dequeue messages with PL/SQL. To complete the demo, we’ll look at publishing and consuming messages from AQ from a very simple REST API Calls.

To set up AQ, we need to create a few users: an admin and a queue user. In the admin schema, we’ll create a queue table, create and start a queue and grant the queue user permission to access it. Then we’ll test out enqueuing and dequeuing a message with the queue user. All of this work will be done with PL/SQL. I like to use the old-school SQL Developer desktop client for this, but you can use whichever tool you prefer.

Graphical user interface, text, application, email

Description automatically generated

select tablespace\_name, sum(bytes/1024000) from dba\_data\_files

group by tablespace\_name order by  sum(bytes/1024000) desc;

"TABLESPACE\_NAME" "SUM(BYTES/1024000)"

"SAMPLESCHEMA" 209715.20

"DATA" 12685.312

"UNDO\_A0061" 10588.16

"UNDO\_8" 10583.424

"SYSAUX" 9585.152

"SYSTEM" 820.224

"DBFS\_DATA" 102.4

Create the blow users in the DATA tablespace.

1. **Create AQ Users for specific AQ work and Miso to store business logic when needed.**

Connect to your Autonomous instance as admin to create two users. One will be an AQ “admin” and the other will be a “user”.

|  |
| --- |
| CREATE USER aqdemoadmin IDENTIFIED BY "Str0ngPassword!";  GRANT connect TO aqdemoadmin;  GRANT create type TO aqdemoadmin;  GRANT aq\_administrator\_role TO aqdemoadmin;  GRANT UNLIMITED TABLESPACE TO aqdemoadmin;  CREATE USER aqdemouser IDENTIFIED BY "Str0ngPassword!";  GRANT connect TO aqdemouser;  GRANT aq\_user\_role TO aqdemouser;  GRANT UNLIMITED TABLESPACE TO aqdemouser;  GRANT connect TO miso;  GRANT aq\_user\_role TO miso;  GRANT UNLIMITED TABLESPACE TO miso; |

select username from dba\_users where username like 'AQ%';

goto database actions and setup the AQ users for web access.

Graphical user interface, application

Description automatically generated

Click on database users.

Open both AQDEMOADMIN AND AQDEMOUSER and alter the below accordingly.

Graphical user interface, text, application

Description automatically generated

Turn on graph, oml and web access. By clicking on edit.

Graphical user interface, text, application, email

Description automatically generated

For aqdemoadmin

Graphical user interface, text, application

Description automatically generated

### Create Queue Table and Queue

### Log out of the admin account.

Now connect up as the aqdemoadmin user that we just created. The first step here is to create a queue table. If we wanted to, we could create a custom type for our queue payload, but since my intention for this demo is to pass JSON messages, we’ll set the queue\_payload\_type to SYS.AQ$\_JMS\_TEXT\_MESSAGE ([docs](https://docs.oracle.com/cd/B28359_01/appdev.111/b28419/t_jms.htm#i996967)) which will support a simple JSON string.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| create table aqdemoadmin.event\_queue\_tab\_products (product varchar(100));   insert into aqdemoadmin.event\_queue\_tab\_products (product)  values ('sippy');   insert into aqdemoadmin.event\_queue\_tab\_products (product)  values ('flippy');   insert into aqdemoadmin.event\_queue\_tab\_products (product)  values ('cr\_coffee');  create table aqdemoadmin.event\_queue\_tab\_chain (name     varchar2(100));   insert into aqdemoadmin.event\_queue\_tab\_chain (name   )  values ('CKE');   insert into aqdemoadmin.event\_queue\_tab\_chain (name   )  values ('WHITECASTLE');   insert into aqdemoadmin.event\_queue\_tab\_chain (name   )  values ('PANERA');  create or replace view aqdemoadmin.V\_event\_queue\_Tab\_product\_name as  select replace(replace(  'BEGIN      DBMS\_AQADM.create\_queue\_table (         queue\_table            =>  '||chr(39)||'aqdemoadmin.event\_queue\_tab\_<PRODUCT>\_<NAME>'||chr(39)||',         queue\_payload\_type     =>  '||chr(39)||'sys.aq$\_jms\_text\_message'||chr(39)||');  END;  /' ,'<PRODUCT>',product),'<NAME>',name) sqltext ,  replace(replace('aqdemoadmin.event\_queue\_tab\_<PRODUCT>\_<NAME>','<PRODUCT>',product),'<NAME>',name)         queue\_table,  length(replace(replace('aqdemoadmin.event\_queue\_tab\_<PRODUCT>\_<NAME>','<PRODUCT>',product),'<NAME>',name)) queue\_table\_len,  'sys.aq$\_jms\_text\_message' queue\_payload\_type ,  replace(replace('grant select, insert, update, delete on aqdemoadmin.event\_queue\_tab\_<PRODUCT>\_<NAME>','<PRODUCT>',product),'<NAME>',name) || ' to miso,aqdemouser;' grantsql   from aqdemoadmin.event\_queue\_tab\_products ,                  aqdemoadmin.event\_queue\_tab\_chain;     |  | | --- | | BEGIN  DBMS\_AQADM.create\_queue\_table (   queue\_table => 'aqdemoadmin.event\_queue\_tab\_sippy\_CKE',   queue\_payload\_type => 'sys.aq$\_jms\_text\_message'); END; / | | BEGIN  DBMS\_AQADM.create\_queue\_table (   queue\_table => 'aqdemoadmin.event\_queue\_tab\_sippy\_WHITECASTLE',   queue\_payload\_type => 'sys.aq$\_jms\_text\_message'); END; / | | BEGIN  DBMS\_AQADM.create\_queue\_table (   queue\_table => 'aqdemoadmin.event\_queue\_tab\_sippy\_PANERA',   queue\_payload\_type => 'sys.aq$\_jms\_text\_message'); END; / | | BEGIN  DBMS\_AQADM.create\_queue\_table (   queue\_table => 'aqdemoadmin.event\_queue\_tab\_flippy\_CKE',   queue\_payload\_type => 'sys.aq$\_jms\_text\_message'); END; / | | BEGIN  DBMS\_AQADM.create\_queue\_table (   queue\_table => 'aqdemoadmin.event\_queue\_tab\_flippy\_WHITECASTLE',   queue\_payload\_type => 'sys.aq$\_jms\_text\_message'); END; / | | BEGIN  DBMS\_AQADM.create\_queue\_table (   queue\_table => 'aqdemoadmin.event\_queue\_tab\_flippy\_PANERA',   queue\_payload\_type => 'sys.aq$\_jms\_text\_message'); END; / | | BEGIN  DBMS\_AQADM.create\_queue\_table (   queue\_table => 'aqdemoadmin.event\_queue\_tab\_cr\_coffee\_CKE',   queue\_payload\_type => 'sys.aq$\_jms\_text\_message'); END; / | | BEGIN  DBMS\_AQADM.create\_queue\_table (   queue\_table => 'aqdemoadmin.event\_queue\_tab\_cr\_coffee\_WHITECASTLE',   queue\_payload\_type => 'sys.aq$\_jms\_text\_message'); END; / | | BEGIN  DBMS\_AQADM.create\_queue\_table (   queue\_table => 'aqdemoadmin.event\_queue\_tab\_cr\_coffee\_PANERA',   queue\_payload\_type => 'sys.aq$\_jms\_text\_message'); END; / | |

|  |
| --- |
| QUEUE\_TABLE |
| aqdemoadmin.event\_queue\_tab\_sippy\_CKE |
| aqdemoadmin.event\_queue\_tab\_sippy\_WHITECASTLE |
| aqdemoadmin.event\_queue\_tab\_sippy\_PANERA |
| aqdemoadmin.event\_queue\_tab\_flippy\_CKE |
| aqdemoadmin.event\_queue\_tab\_flippy\_WHITECASTLE |
| aqdemoadmin.event\_queue\_tab\_flippy\_PANERA |
| aqdemoadmin.event\_queue\_tab\_cr\_coffee\_CKE |
| aqdemoadmin.event\_queue\_tab\_cr\_coffee\_WHITECASTLE |
| aqdemoadmin.event\_queue\_tab\_cr\_coffee\_PANERA |

**Heads Up!** Native [JSON support for AQ](https://docs.oracle.com/en/database/oracle/oracle-database/21/adque/rel-changes.html#GUID-60EC22A2-48C5-4430-9032-42037FEEB09F) is available in 21c. Since 21c is not yet available on Autonomous DB, we’ll use a simple text data type and pass our own JSON string in our messages.

/\* once being turned on for now \*/

BEGIN

DBMS\_AQADM.create\_queue (

queue\_name => 'aqdemoadmin.Q\_event\_queue\_tab\_sippy\_CKE',

queue\_table => 'aqdemoadmin.event\_queue\_tab\_sippy\_CKE');

DBMS\_AQADM.start\_queue (

queue\_name => 'aqdemoadmin.Q\_event\_queue\_tab\_sippy\_CKE',

enqueue => TRUE);

END;

/

<https://blogs.oracle.com/developers/post/send-and-receive-messages-via-rest-with-advanced-queuing-and-ords>

<https://blogs.oracle.com/developers/post/enterprise-messaging-via-oracle-advanced-queuing-with-autonomous-db-micronaut>

create or replace view aqdemoadmin.V\_event\_queue\_Tab\_START\_QUEUE as

select replace(replace(

'BEGIN

    DBMS\_AQADM.create\_queue (queue\_name => '||chr(39)||'aqdemoadmin.Q\_event\_queue\_tab\_<PRODUCT>\_<NAME>'||chr(39)||',

                                queue\_table=>'||chr(39)||'aqdemoadmin.event\_queue\_tab\_<PRODUCT>\_<NAME>'||chr(39)||');

    DBMS\_AQADM.start\_queue  (queue\_name => '||chr(39)||'aqdemoadmin.Q\_event\_queue\_tab\_<PRODUCT>\_<NAME>'||chr(39)||', enqueue => TRUE);

END;

/' ,'<PRODUCT>',product),'<NAME>',name) sqltext ,

replace(replace('aqdemoadmin.Q\_event\_queue\_tab\_<PRODUCT>\_<NAME>','<PRODUCT>',product),'<NAME>',name)       queue\_name,

length(replace(replace('aqdemoadmin.event\_queue\_tab\_<PRODUCT>\_<NAME>','<PRODUCT>',product),'<NAME>',name)) queue\_name\_len

--replace(replace('grant select, insert, update, delete on aqdemoadmin.event\_queue\_tab\_<PRODUCT>\_<NAME>','<PRODUCT>',product),'<NAME>',name) || ' to miso,aqdemouser;' grantsql

 from aqdemoadmin.event\_queue\_tab\_products ,

                aqdemoadmin.event\_queue\_tab\_chain;

create or replace view aqdemoadmin.V\_event\_queue\_Tab\_GRANT\_QUEUE as

select replace(replace(

'BEGIN

    DBMS\_AQADM.grant\_queue\_privilege (privilege=>'||chr(39)||'ALL'||chr(39)||','||'queue\_name => '||chr(39)||'aqdemoadmin.Q\_event\_queue\_tab\_<PRODUCT>\_<NAME>'||chr(39)||',

    grantee=>'||chr(39)||'aqdemouser'||chr(39)||',grant\_option => FALSE);'||chr(10)||

'END;

/' ,'<PRODUCT>',product),'<NAME>',name) grant\_aqdemouser ,

     replace(replace(

'BEGIN

    DBMS\_AQADM.grant\_queue\_privilege (privilege=>'||chr(39)||'ALL'||chr(39)||','||'queue\_name => '||chr(39)||'aqdemoadmin.Q\_event\_queue\_tab\_<PRODUCT>\_<NAME>'||chr(39)||',

    grantee=>'||chr(39)||'miso'||chr(39)||',grant\_option => FALSE);'||chr(10)||

'END;

/' ,'<PRODUCT>',product),'<NAME>',name) grant\_miso ,

replace(replace('aqdemoadmin.Q\_event\_queue\_tab\_<PRODUCT>\_<NAME>','<PRODUCT>',product),'<NAME>',name)       queue\_name,

length(replace(replace('aqdemoadmin.event\_queue\_tab\_<PRODUCT>\_<NAME>','<PRODUCT>',product),'<NAME>',name)) queue\_name\_len

 from aqdemoadmin.event\_queue\_tab\_products ,

                aqdemoadmin.event\_queue\_tab\_chain;

Lets enqueue a message

DECLARE

    l\_enqueue\_options     dbms\_aq.enqueue\_options\_t;

    l\_message\_properties  dbms\_aq.message\_properties\_t;

    l\_message\_handle      raw(16);

    l\_event\_msg           sys.aq$\_jms\_text\_message;

BEGIN

    l\_event\_msg := sys.aq$\_jms\_text\_message.construct();

    l\_event\_msg.set\_text('{"name": "Todd", "id": 9}');

    dbms\_aq.enqueue(queue\_name => 'aqdemoadmin.Q\_event\_queue\_tab\_sippy\_CKE',

                   enqueue\_options => l\_enqueue\_options,

                   message\_properties => l\_message\_properties,

                   payload => l\_event\_msg,

                   msgid => l\_message\_handle);

    COMMIT;

END;

/

Now we can dequeue (or consume) the message, parse the string as a JSON object, and print one of the keys from that JSON object.

 create table AQDEMOADMIN.EVENT\_QUEUE\_ENQ\_DEQUE\_SCRIPTS

  ( table\_name varchar2(4000),

    queue\_name varchar2(4000),

    enque\_sql  varchar2(4000),

    deque\_sql  varchar2(4000)

    );

|  |
| --- |
| SET SERVEROUTPUT ON  DECLARE      l\_dequeue\_options     dbms\_aq.dequeue\_options\_t;      l\_message\_properties  dbms\_aq.message\_properties\_t;      l\_message\_handle      raw(16);      l\_event\_msg           sys.aq$\_jms\_text\_message;      msg\_text              varchar2(32767);      obj                   json\_object\_t;  BEGIN      dbms\_aq.dequeue(queue\_name => 'aqdemoadmin.Q\_event\_queue\_sippy\_CKE',                     dequeue\_options => l\_dequeue\_options,                     message\_properties => l\_message\_properties,                     payload => l\_event\_msg,                     msgid => l\_message\_handle);      l\_event\_msg.get\_text(msg\_text);      /\* print the raw message \*/      dbms\_output.put\_line(msg\_text);      /\* parse the JSON object \*/      obj := json\_object\_t.parse(msg\_text);      /\* print an element from the JSON object \*/      dbms\_output.put\_line(obj.get\_String('name'));      COMMIT;  END;  / |

<https://blogs.oracle.com/developers/post/send-and-receive-messages-via-rest-with-advanced-queuing-and-ords>

## REST Enable the Schema

It may seem like there are a lot of steps here, but honestly, once you get the schema REST enabled and set up a few necessary bits, adding endpoints becomes a matter of running a single query in your favorite SQL editor. The first step in this process is to REST enable your schema. That can be accomplished with the following query, where AQDEMOUSER is the name of the schema/user that you want to enable and the p\_url\_mapping\_pattern will be the "base" path for all of your URLs.

BEGIN

/\* enable ORDS for schema \*/

ORDS.ENABLE\_SCHEMA(

p\_enabled => TRUE,

p\_schema => 'AQDEMOUSER', /\* enter schema \*/

p\_url\_mapping\_type => 'BASE\_PATH',

p\_url\_mapping\_pattern => 'aqdemo', /\* will be used for base path in URL \*/

p\_auto\_rest\_auth => FALSE

);

COMMIT;

END;

/

By default, all endpoints for this schema will be protected (anonymous calls will return a 401 Unauthorized). Pessimistic security is a good thing, but that means we'll have to create REST and OAuth clients and generate some credentials.

### Create REST Client

Here's how to create the REST client. Notice that I'm specifying /\* as the patterns, which means that this REST client applies to all paths in this schema. You shouldn't have to change anything here, just run it as is.

DECLARE

l\_roles OWA.VC\_ARR;

l\_modules OWA.VC\_ARR;

l\_patterns OWA.VC\_ARR;

BEGIN

l\_roles(1) := 'SQL Developer';

l\_patterns(1) := '/\*';

ORDS.DEFINE\_PRIVILEGE(

p\_privilege\_name => 'rest\_privilege',

p\_roles => l\_roles,

p\_patterns => l\_patterns,

p\_modules => l\_modules,

p\_label => '',

p\_description => '',

p\_comments => NULL

);

COMMIT;

END;

/

### Create OAuth Client

Next, we'll create the OAuth client that is specific to the application that we'll use to call the REST endpoints. You can create as many as you need, just give each a unique name and enter the owner, description, and support email as applicable. Make sure to leave the p\_grant\_type and p\_privilege\_names the same.

BEGIN

OAUTH.create\_client(

p\_name => 'ORDS AQ Demo Client', /\* your OAuth client name \*/

p\_grant\_type => 'client\_credentials',

p\_owner => 'Todd Sharp', /\* OAuth client owner \*/

p\_description => 'A demo client for using AQ with ORDS', /\* description \*/

p\_support\_email => 'toddsharp@email.com', /\* client email contact \*/

p\_privilege\_names => 'rest\_privilege'

);

COMMIT;

END;

/

#### Grant SQL Developer Role to Client

Now we need to grant the SQL Developer role to the newly created OAuth client. Use the same name that you specified as  p\_name in the previous query.

BEGIN

OAUTH.grant\_client\_role(

p\_client\_name => 'ORDS AQ Demo Client', /\*OAuth client name (same as above) \*/

p\_role\_name => 'SQL Developer'

);

COMMIT;

END;

/

#### Obtain the Client Credentials

That's it. Our schema is enabled, and our clients are created. The only thing we need to grab at this point is our OAuth client\_id and client\_secret which is what we'll use when we make our HTTP calls to generate a Bearer token to pass along as our Authorization with calls to the REST endpoints that we're about to create. To grab the credentials:

SELECT id, name, client\_id, client\_secret

FROM user\_ords\_clients;

Which should produce something that looks similar to this:

Table

Description automatically generated

**Heads Up!** Make sure to copy and paste the client\_id and client\_secret fully (don't forget the dots at the end of your values if it has them - they are **not** ellipses!).

Before we can make calls, we need to figure out our base URL. The easy way is to use the OCI CLI (if you have it installed) and then deleting /apex from the end of the 'apex-url'.

oci db autonomous-database get \

--autonomous-database-id ocid1.autonomousdatabase.oc1.phx... \

--query 'data."connection-urls"."apex-url"' \

--raw-output

This gives me:

https://[redacted]-DEMODB.adb.us-phoenix-1.oraclecloudapps.com/ords/apex

So my base URL (after deleting /apex) is:

https://[redacted]-DEMODB.adb.us-phoenix-1.oraclecloudapps.com/ords

Another option is to log in to the OCI Console and navigate to your Autonomous DB instance. From the instance details page, click on 'Service Console'.

Graphical user interface, text, application

Description automatically generated

Within the Service Console, click on 'Development' in the left sidebar and find and copy your RESTful services URL.

Graphical user interface, text, application, email

Description automatically generated

Your REST base URL will be this URL, plus the value you specified as p\_url\_mapping\_pattern when you enabled REST for the schema in step 1 above. Here's how mine would look:

https://[redacted]-DEMODB.adb.us-phoenix-1.oraclecloudapps.com/ords/aqdemo

Now we can generate our OAuth token by making a POST to the /oauth/token endpoint, using the client\_id as the username and the client\_secret as the password using Basic Auth. The body should be x-www-form-urlencoded with a value of grant\_type=client\_credentials. Here's an example of making this call in Postman:

A screenshot of a computer

Description automatically generated with medium confidence

Or, if you prefer cURL (the Authorization header when retrieving an OAuth token is a base64 encoded version of the client\_id:client\_secret value).

curl --location --request POST 'https://[redacted]-demodb.adb.us-phoenix-1.oraclecloudapps.com/ords/aqdemo/oauth/token' \

--header 'Content-Type: application/x-www-form-urlencoded' \

--header 'Accept: application/json' \

--header 'Authorization: Basic al...[redacted]...==' \

--data-urlencode 'grant\_type=client\_credentials'

We'll use the access\_token going forward (but notice that it expires in 3600 seconds or 60 minutes).

## Create AQ Endpoints

So far we've REST enabled our schema, created the OAuth client, and obtained a Bearer token. Now we can create the actual REST endpoints that will be used to send (enqueue) and receive (dequeue) messages on our topic. We'll create these endpoints by using the ORDS.DEFINE\_SERVICE function ([docs](https://docs.oracle.com/en/database/oracle/oracle-rest-data-services/19.1/aelig/ORDS-reference.html#GUID-2ABE5D9C-8C80-4422-922E-DBF3A37F4B21)) which will take care of creating all of the necessary bits behind the scenes for us (resource module, template, and handler).

### Create and Test 'Send Message' Endpoint

The ORDS.DEFINE\_SERVICE function has a few parameters that are used to define the HTTP method, the URL pattern that the endpoint will be available at, etc. To define what action we actually want to be taken when the endpoint is called, we use the p\_source parameter. In this case, we'll pass it a block of PL/SQL code to execute (indicated by setting p\_source\_type to ORDS.source\_type\_plsql). The PL/SQL code that we pass will handle sending a message to our queue using the DBMS\_AQ.ENQUEUE function (see the last post for more information on that).

BEGIN

ORDS.define\_service(

p\_module\_name => 'enqueue',

p\_base\_path => '/queue',

p\_pattern => '/:queue\_name/',

p\_method => 'POST',

p\_source\_type => ORDS.source\_type\_plsql,

p\_mimes\_allowed => 'application/json',

p\_source => 'DECLARE

l\_enqueue\_options dbms\_aq.enqueue\_options\_t;

l\_message\_properties dbms\_aq.message\_properties\_t;

l\_message\_id raw(16);

l\_event\_msg sys.aq$\_jms\_text\_message;

BEGIN

l\_event\_msg := sys.aq$\_jms\_text\_message.construct();

l\_event\_msg.set\_text(:body\_text);

dbms\_aq.enqueue(

queue\_name => :queue\_name,

enqueue\_options => l\_enqueue\_options,

message\_properties => l\_message\_properties,

payload => l\_event\_msg,

msgid => l\_message\_id

);

OWA\_UTIL.mime\_header(''application/json'', FALSE);

OWA\_UTIL.http\_header\_close;

htp.p(''{"sent\_at": "'' || to\_char(current\_timestamp, ''YYYY-MM-DD"T"HH24:MI:SSTZH:TZM'') || ''", "message\_id": "'' || l\_message\_id || ''"}'');

COMMIT;

END;'

);

COMMIT;

END;

/

Once we run this, we'll have an endpoint available at:

https://[redacted]-DEMODB.adb.us-phoenix-1.oraclecloudapps.com/ords/aqdemo/aqdemo/queue

We'll need to add our queue name as a path variable ("/aqdemoadmin.event\_queue") and then we're ready to POST a message to the queue by sending it in raw JSON as the POST body. Don't forget to set the access\_token into the header (Authorization Bearer [access\_token]). Here's how a sample call looks in Postman.

A screenshot of a computer

Description automatically generated

The cURL version looks like this:

curl --location --request POST 'https://[redacted]-demodb.adb.us-phoenix-1.oraclecloudapps.com/ords/aqdemo/queue/aqdemoadmin.event\_queue/' \

--header 'Accept: application/json' \

--header 'Authorization: Bearer aX..[redacted]..g' \

--header 'Content-Type: application/json' \

--data-raw '{

"name": "Jessy Zulauf",

"domain": "amelia.biz",

"randomBoolean": true,

"randomNumber": 567,

"randomDate": "Fri Jan 28 2022 23:29:46 GMT-0500 (Eastern Standard Time)"

}'

**How About a Batch?** You can also enqueue an array of messages using DBMS\_AQ.ENQUEUE\_ARRAY. Check the [docs](https://docs.oracle.com/en/database/oracle/oracle-database/21/adque/aq-operations-using-pl-sql.html#GUID-92FCE63B-7713-443A-84A3-9F543F0A8819) for more info.

### Create and Test 'Get Message' Endpoint

So now that we can successfully send (or enqueue) messages to our queue, it makes sense that we'd want an endpoint to retrieve the latest message from the queue. To do this, we'll use ORDS.DEFINE\_SERVICE again, but this time our p\_source code will utilize the DBMS\_AQ.DEQUEUE function.

BEGIN

ORDS.define\_service(

p\_module\_name => 'dequeue.next',

p\_base\_path => '/dequeue/next',

p\_pattern => '/:queue\_name',

p\_method => 'GET',

p\_source\_type => ORDS.source\_type\_plsql,

p\_source => 'DECLARE

l\_dequeue\_options sys.dbms\_aq.dequeue\_options\_t;

l\_message\_properties sys.dbms\_aq.message\_properties\_t;

l\_message\_handle raw(16);

l\_event\_msg sys.aq$\_jms\_text\_message;

msg\_text varchar2(32767);

ex\_timeout EXCEPTION;

PRAGMA EXCEPTION\_INIT(ex\_timeout, -25228);

BEGIN

l\_dequeue\_options.wait := 1;

dbms\_aq.dequeue(

queue\_name => :queue\_name,

dequeue\_options => l\_dequeue\_options,

message\_properties => l\_message\_properties,

payload => l\_event\_msg,

msgid => l\_message\_handle

);

l\_event\_msg.get\_text(msg\_text);

OWA\_UTIL.mime\_header(''application/json'', FALSE);

OWA\_UTIL.http\_header\_close;

htp.p(msg\_text);

COMMIT;

EXCEPTION

WHEN ex\_timeout THEN

OWA\_UTIL.mime\_header(''application/json'', FALSE);

OWA\_UTIL.http\_header\_close;

htp.p(''{}'');

END;'

);

COMMIT;

END;

/

This will create an endpoint at:

https://[redacted]-demodb.adb.us-phoenix-1.oraclecloudapps.com/ords/aqdemo/dequeue/next/[queue\_name]

That will accept HTTP GET requests. No other parameters are required. Calling this method will look like so in Postman:

A screenshot of a computer

Description automatically generated with medium confidence

The cURL version:

curl --location --request GET

'https://[redacted]-demodb.adb.us-phoenix-1.oraclecloudapps.com/ords/aqdemo/dequeue/next/aqdemoadmin.event\_queue' \

--header 'Accept: application/json' \

--header 'Authorization: Bearer m..[redacted]..g'

### Create and Test 'Get Message Batch' Endpoint

Getting the 'next' message is handy, but sometimes you want to get a batch of messages. To do that, we can create another endpoint, but this time use the DBMS\_AQ.DEQUEUE\_ARRAY function.

We'll first need to create a custom table of AQ$\_JMS\_TEXT\_MESSAGE that will be used to store the result.

CREATE OR REPLACE TYPE t\_msgs AS TABLE OF sys.aq$\_jms\_text\_message NOT NULL;

BEGIN

ORDS.define\_service(

p\_module\_name => 'dequeue.batch',

p\_base\_path => '/dequeue/batch',

p\_pattern => '/:queue\_name',

p\_method => 'GET',

p\_source\_type => ORDS.source\_type\_plsql,

p\_source => 'DECLARE

l\_dequeue\_options dbms\_aq.dequeue\_options\_t;

l\_message\_properties\_array dbms\_aq.message\_properties\_array\_t;

l\_msgid\_array dbms\_aq.msgid\_array\_t;

l\_message\_handle raw(16);

l\_event\_msgs t\_msgs;

l\_array\_size PLS\_INTEGER;

msg\_text varchar2(32767);

ex\_timeout EXCEPTION;

ret\_val PLS\_INTEGER;

PRAGMA EXCEPTION\_INIT(ex\_timeout, -25228);

BEGIN

l\_dequeue\_options.wait := 1;

l\_array\_size := 25;

ret\_val := dbms\_aq.dequeue\_array(

queue\_name => :queue\_name,

dequeue\_options => l\_dequeue\_options,

array\_size => l\_array\_size,

message\_properties\_array => l\_message\_properties\_array,

payload\_array => l\_event\_msgs,

msgid\_array => l\_msgid\_array

);

OWA\_UTIL.mime\_header(''application/json'', FALSE);

OWA\_UTIL.http\_header\_close;

IF l\_event\_msgs.COUNT > 0 THEN

htp.p(''['');

FOR i IN l\_event\_msgs.FIRST .. l\_event\_msgs.LAST LOOP

htp.p(l\_event\_msgs(i).text\_vc);

IF i < l\_event\_msgs.LAST THEN

htp.p('','');

END IF;

END LOOP;

htp.p('']'');

END IF;

COMMIT;

EXCEPTION

WHEN ex\_timeout THEN

OWA\_UTIL.mime\_header(''application/json'', FALSE);

OWA\_UTIL.http\_header\_close;

htp.p(''[]'');

END;');

COMMIT;

END;

/

Clearly, it would be a bad idea to retrieve **all** pending messages in the queue. With this endpoint, I'm limiting it to the 25 most recent messages by setting l\_array\_size to 25. Feel free to adjust to meet your requirements. Let's call the new endpoint with Postman:

Text

Description automatically generated

curl --location --request GET \

'https://[redacted]-demodb.adb.us-phoenix-1.oraclecloudapps.com/ords/aqdemo/dequeue/batch/aqdemoadmin.event\_queue' \

--header 'Accept: application/json' \

--header 'Authorization: Bearer m..[redacted]..g'

## Summary

In this post, we created and exposed secure REST endpoints that allow us to send and receive messages to and from Oracle Advanced Queueing. This method allows us to work with a messaging queue from anywhere we can use HTTP instead of relying on a dedicated SDK.

Graphical user interface

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

Create sales2

A screenshot of a computer

Description automatically generated with medium confidence

Create based on wsdl

Use [Aplascencia@misorobotics.cm](mailto:Aplascencia@misorobotics.cm)

Epfy44yqE\_WEB

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

<http://api-apiint.brinkpos.net/Sales2.svc?singleWsdl>

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence