Multitenant New Security Features
Clarify DevOps/DBA roles

Franck Pachot
dbi services
Switzerland

Keywords:

12cR2, Multitenant, Security, DevOps

Introduction

The CDB/PDB paradigm enforces role segregation between the container DBA and the application DBA. In DevOps environment, you need the finer access privilege definition that comes in 12.1 (common/local users) and in 12.2 (lockdown profiles, PDB isolation).
Can we administrate a Cloud or on-premises database and give PDB administration rights to application teams for their application? And clearly define roles between system DBA and application DBA?

Lockdown profiles

Pluggable databases bring a new separation of database administrator roles. The DBA administers the container database, the CDB, but can then delegate the administration of individual pluggable databases. Let's take an example of a CDB that is a dedicated development environment. The fast and thin provisioning features (snapshot clones) make it possible to give a PDB to each developer. Because it is their database, the CDB administrator can grant them the DBA role for the PDB, so that they can do whatever they wish there, as long as their privileges are limited to this pluggable database.
In 12.1 this strategy is almost impossible to implement. Even if the DBA role is only granted locally to a local PDB user, this privilege enables them to do things that can potentially break the CDB or the server. For example, a local DBA can create files wherever they wish, can execute any program on the host (which will run as the oracle user), and can generate massive trace files. If we want to limit what a local DBA can do, we need better control over these privileges, and this is why 12.2 introduced lockdown profiles.
Here, connected to CDB$ROOT we create a profile for our application DBAs:

```
SQL> create lockdown profile APP_DBA_PROFILE;
Lockdown Profile created.

SQL> select * from DBA_LOCKDOWN_PROFILES;
PROFILE_NAME  RULE_TYPE RULE         CLAUSE CLAUSE_OPT OPTION_VAL STATUS
------------- --------- ------------ ------ ---------- ---------- ------
APPDBA_PROF                                                      DISABLE
```

(2) Disable Database Options

With profiles we can disable access to some features available only with licensed options. For example, if we don't have the partitioning option we must be sure that nobody will create a partitioned table, so let's disable it from our application DBA profile:

```
SQL> alter lockdown profile APPDBA_PROF disable option = ('Partitioning');
Lockdown Profile altered.

SQL> select * from DBA_LOCKDOWN_PROFILES;
PROFILE_NAME  RULE_TYPE RULE         CLAUSE CLAUSE_OPT OPTION_VAL STATUS
------------- --------- ------------ ------ ---------- ---------- ------
APPDBA_PROF   OPTION    PARTITIONING                              DISABLE
```

We can now apply the lockdown profile to the pluggable database PDB1, simply by setting the pdb_lockdown parameter for that container:

```
SQL> alter session set container=PDB1;
Session altered.
SQL> alter system set pdb_lockdown=APPDBA_PROF;
System altered.
```

So now let's try to create a partitioned table in that PDB1:

```
SQL> connect admin/oracle@//localhost/PDB1
Connected.
SQL> create table DEMO(id number) partition by hash(id) partitions 4;
create table DEMO(id number) partition by hash(id) partitions 4
             *
ERROR at line 1:
ORA-00439: feature not enabled: Partitioning
```

And, as can be seen, this is impossible because the feature has been disabled.
The ENABLE and DISABLE clause of ALTER LOCKDOWN PROFILE can also be specified with an ALL option, such as:

```
SQL> alter lockdown profile APPDBA_PROF disable option all;
SQL> alter lockdown profile APPDBA_PROF disable option all except = ('Oracle Data
Guard');
```

Disable Alter System

The ALTER SYSTEM privilege is very powerful, but with the GRANT syntax you can only allow or disallow it. However, with lockdown profiles you have fine-grained control because you can enable or disable specific clauses of the statement. Let's say, for example, that you want to allow your developers to kill sessions in their PDB, but no other ALTER SYSTEM activities. From CDB$ROOT you can add the following rule:

```
SQL> alter lockdown profile APPDBA_PROF disable statement = ('ALTER SYSTEM') clause
all except = ('KILL SESSION');
SQL> select * from DBA_LOCKDOWN_PROFILES;
PROFILE_NAME  RULE_TYPE RULE         CLAUSE       CLAUSE_OPT OPTION_VAL STATUS
------------- --------- ------------ ------------ ---------- ---------- -------
APPDBA_PROF   OPTION    PARTITIONING                                    DISABLE
APPDBA_PROF   STATEMENT ALTER SYSTEM KILL SESSION                       ENABLE
```

With this a user in the PDB that has the lockdown profile assigned will get an 'ORA-01031: insufficient privileges' for any ALTER SYSTEM command, except an ALTER SYSTEM KILL SESSION.
The scope of control can be defined further with the ALTER SYSTEM SET command, because you can even control which parameters are allowed. For example, the following will allow only some parameters to be set at the PDB level:

```
SQL> alter lockdown profile APPDBA_PROF disable statement = ('ALTER SYSTEM') clause
= ('SET');
Lockdown Profile altered.
SQL> alter lockdown profile APPDBA_PROF  enable statement = ('ALTER SYSTEM') clause
= ('SET') option = ('undo_retention', 'temp_undo_enabled', 'resumable_timeout',
'cursor_sharing', 'session_cached_cursors', 'heat_map', 'resource_manager_plan',
'optimizer_dynamic_sampling');
Lockdown Profile altered.
```

We can query the dictionary to see these defined:

```
SQL> select * from DBA_LOCKDOWN_PROFILES where profile_name='APPDBA_PROF';
PROFILE_NAME  RULE_TYPE RULE         CLAUSE  CLAUSE_OPTION              STATUS
------------- --------- ------------ ------- -------------------------- -------
APPDBA_PROF   STATEMENT ALTER SYSTEM SET                                DISABLE
APPDBA_PROF   STATEMENT ALTER SYSTEM SET     CURSOR_SHARING             ENABLE
APPDBA_PROF   STATEMENT ALTER SYSTEM SET     HEAT_MAP                   ENABLE
APPDBA_PROF   STATEMENT ALTER SYSTEM SET     OPTIMIZER_DYNAMIC_SAMPLING ENABLE
APPDBA_PROF   STATEMENT ALTER SYSTEM SET     RESOURCE_MANAGER_PLAN      ENABLE
APPDBA_PROF   STATEMENT ALTER SYSTEM SET     RESUMABLE_TIMEOUT          ENABLE
APPDBA_PROF   STATEMENT ALTER SYSTEM SET     SESSION_CACHED_CURSORS     ENABLE
APPDBA_PROF   STATEMENT ALTER SYSTEM SET     TEMP_UNDO_ENABLED          ENABLE
APPDBA_PROF   STATEMENT ALTER SYSTEM SET     UNDO_RETENTION             ENABLE
```

From a PDB where this lockdown profile is set, you are able to set one of these allowed parameters:

```
SQL> alter system set optimizer_dynamic_sampling=4;
System altered.
```

But you will receive a privilege error when trying to set one that is not in the permitted list:

```
SQL> alter system set optimizer_index_cost_adj=1;
alter system set optimizer_index_cost_adj=1
*
ERROR at line 1:
ORA-01031: insufficient privileges
```

When disabling the change of a parameter, you can also define a value to be set, at the same time, when the PDB_LOCKDOWN parameter is set:

```
alter lockdown profile APPDBA_PROF disable statement=('ALTER SYSTEM')
clause=('SET') option=('cursor_sharing') value=('EXACT');
```

This is an effective means of creating a lockdown profile with several parameters set to values that cannot be changed later.

(2) Disable Features

We will not go into detail on this here, but in the same way that you can disable database options you can also disable features. For example, the following command disables the specified PL/SQL package usage:

```
SQL> alter lockdown profile APPDBA_PROF disable feature =
('UTL_HTTP','UTL_SMTP','UTL_TCP');
```

NOTE: you can disable all networking packages with the “NETWORK_ACCESS” feature name.

PDB Isolation

In 12.2, in addition to the PDB_LOCKDOWN parameter that can be used to set a lockdown profile to limit network access, you can also limit the interaction with the OS file system and processes.

PDB_OS_CREDENTIALS

From a dbms_scheduler job, or through an external procedure, it is possible to run a program on the host server. But, more than likely, you will probably not want to let the pluggable database administrator run anything with the oracle user privileges. In this case you can create a credential, from the root, defining the OS user and password, and also including a domain if you are on Windows:

```
exec dbms_credential.create_credential( credential_name=>'PDB1_OS_USER',
username=>'limitedUser', password=>'secret');
```

You can limit a PDB to this user when running jobs or external procedures:

```
alter session set container=PDB1;
alter system set pdb_os_credential=CDB_PDB_OS_USER scope=spfile;
```

PATH_PREFIX

In a similar vein, a PDB administrator can create a directory and write files anywhere on the system. This was not a problem before multitenant, because the DBA controls both the database and the host, but in multitenant you can delegate some administration tasks to the PDB administrator and then need more control on how they can interact with the host. Since 12.1 it has been possible to define a PATH_PREFIX, as the root of all directories created in a PDB, then defined with a relative path from here. Note that you cannot change the PATH_PREFIX after creation.

CREATE_FILE_DEST

Obviously, another way to write files onto a server is to create tablespaces and add datafiles, and this is also an operation the CDB administrator needs to restrict. Starting with 12.1.0.2, you can now set the CREATE_FILE_DEST to a directory specific for the PDB, so that datafiles are written there. However, a user with create or alter tablespace privileges can still specify a fully qualified file name, and then write everywhere the oracle user can write. OS credentials are not utilized here. NOTE: we have opened an Enhancement Request about this gap, and we hope to have the option to lockdown a PDB administrator to use OMF only, without specifying an absolute file path or a diskgroup, in the near future.

Conclusion

We explained here what is currently documented but it can go far beyond that. Remember that behind 12*c* and multitenant, the goal is the Cloud. If you want to provision a PDB as a Service you need to give all database access to the PDB but none on the system which is shared. Oracle has the goal to provide a managed PaaS with full administration privileges and this is where those security features will grow.