Practice 9

Implementing Connection Load Balancing and TAF

Practice Overview

In this practice you will implement and test different high availability methods for client connections in Oracle RAC database. Specifically, you will perform the following:

- Configure client-side connect-time load balancing
- Configure server-side connect-time load balancing with and without having the Load Balancing Advisory (LBA) enabled
- Configure Transparent Application Failover (TAF) on client side
- Configure Basic TAF on server side
- Configure Preconnect TAF on the client side and on the server side.

Practice Assumptions

- The practice assumes that you have the Oracle RAC database up and running in the virtual machines `srv1` and `srv2`.
- This practice assumes that you have downloaded the files available in the downloadable section of this lecture.
- The scripts available in the downloadable resources in the lecture "*Monitoring and Tuning Oracle RAC Database*" have been downloaded.
  The practice examples assume that the script files have been extracted in the directory `D:\scripts`. If you have extracted the files somewhere else, change the code accordingly when you run it.

A. Configure client-side connect-time load balancing

In this section of the practice, you will configure client-side connect-time load balancing in your hosting machine. You will then perform basic testing on the configuration.

1. Open `tnsnames.ora` file in your hosting PC and add the following naming descriptor in it. Observe that the VIP addresses have been used, not the SCAN IP addresses. Copy the code from the downloadable `tnsnames.ora` file, not from the PDF file.

Also, observe `rac` service has been used, not `soesrv`. The latter has only `rac1` as the preferred instance. Therefore, if you use it in your configuration, all the connections will go to `rac1` instance.

```
SOESRV2=
 (DESCRIPTION =
  (ADDRESS_LIST =
    (LOAD_BALANCE=ON)
    (ADDRESS=(PROTOCOL=TCP)(HOST=192.168.56.81)(PORT=1521))
    (ADDRESS=(PROTOCOL=TCP)(HOST=192.168.56.82)(PORT=1521))
   )
  (CONNECT_DATA=(SERVICE_NAME=rac.localdomain))
  )
```

2. In your hosting PC, open a command prompt window and test connecting to the service using the configuration that you added in the previous step. This window will be referred to in this practice as the client window.

```
sqlplus soe/soe@soesrv2
```

3. Open a Putty terminal window and connect to `srv1` as `oracle`.

4. In the Putty terminal window, use SQL*Plus to connect to `rac` as `sysdba`. This window will be referred to in this practice as the admin window.

```
sqlplus / as sysdba
```

Testing client-side connect-time load balancing when all RAC nodes are available

In the following steps you will examine how the client-side connect-time load balancing configuration works when all the RAC nodes are available.

5. In the **client window**, change the directory to the `scripts` directory and examine the batch file `script7.bat`. It kicks off 20 SQL*Plus client sessions that connect to `rac` using the naming descriptor that you defined earlier. Each client session will wait for 20 seconds then exit from its connection.

6. Run the batch file `script7.bat`

Do not wait for the sessions to finish. **Once you run the batch file, go immediately to the next step.**

```
d:
cd D:\scripts
script7.bat
```

7. In the admin window, run the following query. You will observe that the sessions have been randomly distributed among the instances.

```
SELECT INST_ID, COUNT(*)
FROM   GV$SESSION
WHERE  USERNAME='SOE'
GROUP  BY INST_ID
ORDER  BY INST_ID;
```

Testing client-side connect-time load balancing when a node is unavailable

In the following steps you will examine how the client-side connect-time load balancing configuration works when one of the RAC nodes is unavailable.

8. In the **admin window**, crash rac1 instance.

```
pkill -9 -f ora_pmon_rac1
```

9. In the **client window**, run the batch file again then **go to the next step**.

```
script7.bat
```

10. In the **admin window**, login to rac2 as sysdba and run the same query that you run in the previous section.

    You will observe that all the 20 session have been successfully connected to the available node without any timeout error or noticeable delay.

```
ssh srv2
sqlplus sys/oracle@rac2 as sysdba

SELECT INST_ID, COUNT(*)
FROM   GV$SESSION
WHERE  USERNAME='SOE'
GROUP BY INST_ID
ORDER BY INST_ID;

# exit from srv2
exit
```

11. Verify that rac1 instance has been automatically started by the Clusterware.

```
srvctl status database -d rac
```

B. Configure server-side Load Balancing Advisory for connect-time load balancing

In this section of the practice, you will create a service that has the Load Balancing Advisory (LBA) disabled in one time and enabled in another time.

You will then test the load balancing feature in the service using the following methodology:

- Apply an intense load on one node.
- Make dozens of connections to the service.
- Examine how the connections to the service are distributed among the nodes.
- Apply an intense load on the other node and repeat the same test.

Testing server-side connect-time load balancing when LBA is not enabled

In the following steps, you will create a service that does not have the LBA configured in it. You will then test how the client connections are distributed on the database nodes.

12. In the **admin window**, create and start a service as follows.

The LBA is **not** enabled in this service because neither the `rlbgoal` nor the `clbgoal` has been set. The service has both nodes `rac1` and `rac2` as preferred nodes.

```
srvctl add service -database rac -service lbsrv -preferred rac1,rac2

srvctl start service -database rac -service lbsrv

srvctl config service -d rac -s lbsrv
```

13. Make sure the service is registered in the listener.

```
lsnrctl service | grep lbsrv
```

14. Using SQL*Plus, login as sysdba to the database then query `DBA_SERVICES` to verify that the LBA is not enabled for the service.

When the `GOAL` is `NONE`, it means the LBA is not enabled for the service.

```
sqlplus / as sysdba
col name format a20

SELECT NAME, GOAL, CLB_GOAL
FROM DBA_SERVICES WHERE NAME='lbsrv';
```

15. In the **admin window**, run the following query just to save it in the SQL*Plus buffer:

```
SELECT INST_ID, COUNT(*)
FROM   GV$SESSION
WHERE  USERNAME='SOE'
GROUP BY INST_ID
ORDER BY INST_ID;
```

16. In the hosting PC, add the following naming descriptor configuration to the `tnsnames.ora` file. This configuration connects to the service created in the previous step.

Copy the code from the downloaded `tnsnames.ora` file.

```
lbsrv =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.56.91)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = lbsrv.localdomain)
    )
  )
```

17. In the hosting PC, open a **second** command prompt window. Invoke SQL*Plus and login to the `lbsrv` service as soe user. We will refer to this window as the second client window.

```
sqlplus soe/soe@lbsrv
```

18. Retrieve the node on which the user is connected to and take a note of it.

```
SELECT SYS_CONTEXT('USERENV','INSTANCE_NAME') FROM DUAL;
```

19. In the second client window, run the following PL/SQL block then **go immediately to the next step**. Do not wait for the block to finish.

This block executes purely arithmetic operation. It therefore applies load on the CPU.

```
set timing on

DECLARE
 N NUMBER;
 V DATE := SYSDATE;
BEGIN
 WHILE (SYSDATE-V) < 0.0004 LOOP
  -- this is purely arithmetic operation (load on CPU)
  N := SQRT(DBMS_RANDOM.VALUE(1,1000));
 END LOOP;
END;
/
```

20. In the **first client window**, invoke the following batch file then go straight away to next step.

It kicks off 20 client sessions that connect to the newly created service (`lbsrv`). They last for 20 seconds and then exit.

```
script8.bat
```

21. In the admin window, run the query saved in the buffer. You can use forward slash symbol '/' to re-run the command in the buffer.

You will observe that that sessions have been nearly equally distributed among the nodes.

Conclusion: for a service that does **not** have LBA enabled, the client connections are distributed nearly equally on the RAC instances regardless of the load currently applied on any instance.

Testing server-side connect-time load balancing when LBA is enabled

In the following steps, you will enable the LBA in the `lbsrv` service. You will then test how the connections to the service are distributed on the nodes when one node in the RAC is heavily loaded.

22. In the admin window, modify the `lbrsrv` service so that the LBA is enabled in it as follows:

```
srvctl modify service -db rac -service lbsrv -rlbgoal service_time -clbgoal
short
```

23. Using SQL*Plus, login as sysdba to the database then query `DBA_SERVICES` to verify that the LBA is enabled for the service.

```
sqlplus / as sysdba
col name format a20

SELECT NAME, GOAL, CLB_GOAL
FROM DBA_SERVICES WHERE NAME='lbsrv';
```

24. Run the same query that you ran earlier to check how the sessions are distributed among the nodes. At this stage, there is only one connected session but you run the query just to save it in the SQL*Plus buffer.

```
SELECT INST_ID, COUNT(*)
FROM   GV$SESSION
WHERE  USERNAME='SOE'
GROUP BY INST_ID
ORDER BY INST_ID;
```

25. In the **second client window**, run the same PL/SQL block that you ran earlier in the same window to apply the same load on the CPU. You can use the forward slash symbol to re-run the same code. Go to the next step after executing the block code.

26. In the **first client window**, run the same batch file again. Go immediately to the next step.

```
script8.bat
```

27. In the **admin window**, run the query saved in the buffer.

Observe the almost all the sessions were connected to the node other than the one that is connected to by the second client window. This is because the CPU in the node that is connected to by the second client window is heavily loaded. LBA, therefore, directs the connections to the least loaded node.

28. To bullet proof this conclusion, exit from the **second client window session**. Try connecting again to the same service but to the other node. You cannot specify the node you want to connect to when you use the service name in your connection. You just have to disconnect and connect again once or more till you are connected to the desired node.

Once you are connected to the other node, repeat the test and observe how the LBA will direct the connections to the least loaded node again.

29. Further testing can be done, if you wish to. Exit from the second client window which was applying load on the CPU. Perform the test again. You will observe that the sessions are distributed equally on the nodes this time.

30. Perform the following clean up steps

a) Stop then delete the `lbsrv` service.

```
srvctl stop service -database rac -service lbsrv
srvctl remove service -database rac -service lbsrv
```

b) Remove the `soesrv2` and `lbsrv` connection descriptors from the `tnsnames.ora` file.

c) Exit from SQL*Plus in the second client window and close the window.

At this stage, we end up with having one Putty session connected to `srv1` and one client window which is a command prompt window in your hosting PC.

Conclusion: for a service that has the LBA enabled, the client connections are distributed to the RAC nodes based on the goal that you specified in the service settings when you created the service.

C. Configure TAF Basic Configuration on Client-side

In this section of the practice, you will configure client-side TAF basic configuration. You will then perform basic testing on the configuration.

31. In the **admin window**, query DBA_SERVICES to verify that server-side TAF is disabled in the service rac.localdomain.

When the FAILOVER_METHOD is null, it means the TAF is not enabled for the service.

```
sqlplus / as sysdba

SELECT FAILOVER_METHOD
FROM DBA_SERVICES WHERE NAME='rac.localdomain';
```

32. Open tnsnames.ora file in your hosting PC and add the following connection descriptor in it. This configuration enables TAF in the clients whose using this connection to connect to the service.

Do not copy the code from the PDF file.

```
ctaf =
 (DESCRIPTION =(FAILOVER=ON) (LOAD_BALANCE=ON)
        (ADDRESS=(PROTOCOL=TCP)(HOST=192.168.56.81)(PORT=1521))
        (ADDRESS=(PROTOCOL=TCP)(HOST=192.168.56.82)(PORT=1521))
  (CONNECT_DATA =
        (SERVICE_NAME = rac.localdomain)
        (FAILOVER_MODE = (TYPE=select)
                    (METHOD= basic )
                    (RETRIES=10)
                    (DELAY=10)))
  )
```

33. In the **client window**, invoke SQL*Plus and login to the service using ctaf connection descriptor. Verify that the session is connected to rac1. If it is connected to rac2, keep re-connecting to the service until you are connected to rac1.

```
sqlplus soe/soe@ctaf
SELECT SYS_CONTEXT('USERENV','INSTANCE_NAME') FROM DUAL;
```

34. In the **admin window**, verify that the client session has the failover enabled.

If FAILOVER_TYPE is null, then the failover is not enabled for the session.

```
col username format a10
col service_name format a20

SELECT INST_ID, USERNAME, SERVICE_NAME, FAILOVER_TYPE, FAILOVER_METHOD,
FAILED_OVER
FROM GV$SESSION WHERE USERNAME='SOE';
```

35. In the **client window**, run the following long running SELECT statement. Go to next step after issuing the statement.

```
SELECT A.OBJECT_NAME FROM ALL_OBJECTS A UNION ALL SELECT B.OBJECT_NAME FROM
ALL_OBJECTS B;
```

36. In the **admin window**, crash rac1 instance.

```
pkill -9 -f ora_pmon_rac1
```

37. Observe the reaction of the client session.

You should notice that the client window pauses for a few seconds then proceeds displaying the returned rows. This is the expected behavior.

38. After the SELECT statement finishes displaying the rows, verify that the client session is now connected to rac2.

```
SELECT SYS_CONTEXT('USERENV','INSTANCE_NAME') FROM DUAL;
```

39. In the **admin window**, check out the FIALED_OVER value of the client session. It should be YES.

```
sqlplus sys/oracle@rac as sysdba

col username format a10
col service_name format a20

SELECT INST_ID, USERNAME, SERVICE_NAME, FAILOVER_TYPE,FAILOVER_METHOD,
FAILED_OVER
FROM GV$SESSION WHERE USERNAME='SOE';
```

40. Perform the following cleanup steps:

a) Delete the ctaf configuration from the tnsnames.ora file.

b) rac1 instance should be automatically started by the clusterware. In the admin window, verify that rac1 instance is up and running. If it is not, wait for a few seconds and try again.

```
srvctl status instance -d rac -i rac1
```

D. Configure TAF Basic Configuration on Server-side

In this section of the practice, you will create a new service and enable the basic TAF configuration in it. You will then test this configuration.

41. Create a service named as staf where the basic TAF configuration is enabled in it. Start the service afterwards.

```
srvctl add service -db rac -service staf -preferred rac1 -available rac2 -
failovermethod BASIC -failovertype SELECT -failoverretry 10 -failoverdelay 5

srvctl start service -db rac -service staf

srvctl config service -d rac -s staf
```

42. Verify that the service has the failover enabled.

```
sqlplus / as sysdba
col failover_method format a20
col failover_type format a20

SELECT FAILOVER_METHOD, FAILOVER_TYPE FROM DBA_SERVICES WHERE NAME like 'staf%';
```

43. Verify that the service is registered in the listener.

```
lsnrctl services | grep staf
```

44. In the tnsnames.ora file in your hosting PC, add the following connection descriptor. Observe that TAF is not enabled in this client-side configuration. Do not copy the code from the PDF file.

```
staf=
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.56.91)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = staf.localdomain)
    )
  )
```

45. In the client window, login to the service using staf connection descriptor. Verify that the session is connected to rac1.

```
conn soe/soe@staf

SELECT SYS_CONTEXT('USERENV','INSTANCE_NAME') FROM DUAL;
```

46. In the admin window, verify that the client session has the failover enabled.

```
sqlplus sys/oracle@rac as sysdba

col username format a10
col service_name format a20

SELECT INST_ID, USERNAME, SERVICE_NAME, FAILOVER_TYPE, FAILOVER_METHOD,
FAILED_OVER
FROM GV$SESSION WHERE USERNAME='SOE';
```

47. In the client window, run the following long running SELECT statement. Do not wait for it to finish. Go to the next step straight away after issuing the query.

```
SELECT A.OBJECT_NAME FROM ALL_OBJECTS A UNION ALL SELECT B.OBJECT_NAME FROM
ALL_OBJECTS B;
```

48. In the admin window, crash rac1 instance.

```
pkill -9 -f ora_pmon_rac1
```

49. Observe the reaction of the client session.

You should notice that the client window pauses for a few seconds then proceeds displaying the returned rows. As TAF is enabled in the service, this is the expected behavior.

50. After the SELECT statement finishes displaying the rows, verify that the client session is now connected to rac2.

```
SELECT SYS_CONTEXT('USERENV','INSTANCE_NAME') FROM DUAL;
```

51. In the **admin window**, check out the FIALED_OVER value of the client session.

```
sqlplus sys/oracle@rac as sysdba

col username format a10
col service_name format a20

SELECT INST_ID, USERNAME, SERVICE_NAME, FAILOVER_TYPE, FAILOVER_METHOD,
FAILED_OVER
FROM GV$SESSION WHERE USERNAME='SOE';
```

52. Perform the following cleanup steps:

a) Exit from all SQL*Plus sessions.

b) rac1 instance should be automatically started by the clusterware. In the admin window, verify that rac1 instance is up and running. If it is not, wait for a few seconds and try again.

```
srvctl status instance -d rac -i rac1
```

c) Stop and delete staf service.

```
srvctl stop service -database rac -service staf
srvctl remove service -database rac -service staf
```

d) Delete staf connection descriptor from tnsnames.ora file.

E. Configure TAF PRECONNECT Configuration on Client-side

In this section of the practice, you will set client-side TAF PRECONNECT configuration. You will test the configuration.

53. Open `tnsnames.ora` file in your hosting PC and add the following connection descriptors in it.

Do not copy its code from the PDF file.

```
prectaf1 =
 (DESCRIPTION=
   (ADDRESS=(PROTOCOL=tcp)(HOST=192.168.56.81)(PORT=1521))
   (CONNECT_DATA=
        (SERVICE_NAME=rac.localdomain)
        (FAILOVER_MODE=
               (BACKUP=prectaf2)
               (TYPE=select)
               (METHOD=preconnect)))
  )

prectaf2 =
 (DESCRIPTION=
   (ADDRESS=(PROTOCOL=tcp)(HOST=192.168.56.82)(PORT=1521))
   (CONNECT_DATA=
        (SERVICE_NAME=rac.localdomain)
        (FAILOVER_MODE=
               (BACKUP=prectaf1)
               (TYPE=select)
               (METHOD=preconnect)))
  )
```

54. In the client window, invoke SQL*Plus and login to the service using `prectaf1` connection descriptor. Verify that the session is connected to `rac1`.

```
sqlplus soe/soe@prectaf1

SELECT SYS_CONTEXT('USERENV','INSTANCE_NAME') FROM DUAL;
```

55. In the **admin window**, list the sessions created by soe user.

You will see two sessions. The one that is connected to `rac1` has its `FAILOVER_METHOD` set to `PRECONNECT`. The other one acts as a pre-established connection and will be used only when the first connection goes down. That is why its `FAILOVER_METHOD` is set to `NONE`.

```
sqlplus sys/oracle@rac as sysdba

col username format a10
col service_name format a20

SELECT INST_ID, USERNAME, SERVICE_NAME, FAILOVER_TYPE, FAILOVER_METHOD,
FAILED_OVER
FROM GV$SESSION WHERE USERNAME='SOE'
ORDER BY INST_ID;
```

56. In the **client window**, issue the following long running SELECT statement then go straight away to the next step.

```
SELECT A.OBJECT_NAME FROM ALL_OBJECTS A UNION ALL SELECT B.OBJECT_NAME FROM
ALL_OBJECTS B;
```

57. In the admin window, crash rac1 instance.

```
pkill -9 -f ora_pmon_rac1
```

58. Observe the reaction of the client session.

You should notice that the client window pauses for a few seconds then proceeds displaying the returned rows. This is the expected behavior.

59. After the SELECT statement finishes displaying the rows, verify that the client session is now connected to rac2.

```
SELECT SYS_CONTEXT('USERENV','INSTANCE_NAME') FROM DUAL;
```

60. In the **admin window**, list the sessions created by soe user.

You will see one session connected to rac2 and its FAILED_OVER value is YES.

```
sqlplus sys/oracle@rac as sysdba

col username format a10
col service_name format a20

SELECT INST_ID, USERNAME, SERVICE_NAME, FAILOVER_TYPE,FAILOVER_METHOD,
FAILED_OVER
FROM GV$SESSION WHERE USERNAME='SOE'
ORDER BY INST_ID;
```

61. Perform the following cleanup steps:

a) Exit from the SQL*Plus sessions.

b) Delete the precataf1 and precataf2 configurations from the tnsnames.ora file.

c) rac1 instance should be automatically started by the clusterware.

```
srvctl status instance -d rac -i rac1
```

F. Configure TAF PRECONNECT configuration on server-side

In this section of the practice, you will set up a TAF PRECONNECT configuration on the server side. You will then perform basic testing on it.

Note: although we describe this TAF configuration as "server-side", but actually, as you will see in this section steps, the failover functionality is done by the client side. The difference between the PRECONNECT client-side configuration and the PRECONNECT server-side configuration is that in the latter there will be a shadow service to which the BACKUP connection will get connected to.

62. In the **admin window**, create a service named as pretaf where the tafpolicy setting is set to PRECONNECT. Start the service afterwards.

tafpolicy option by itself does not enable the service failover functionality. When it is set to PRECONNECT, it results in automatically creating a shadow service to the defined service. Failover is defined when you set the FAILOVER_TYPE.

The name of the shadow service will have the formant <*service_name*>_preconnect

```
srvctl add service -db rac -service pretaf -preferred rac1 -available rac2 -
tafpolicy PRECONNECT

srvctl start service -db rac -service pretaf
```

63. Run the following command to obtain the configuration details about the services created as a result of the command above.

You will observe that the configuration settings for the services are identical. Also, observe the failover is disabled in the services.

Note: when I tried enabling the failover in a preconnect service, the pre-connect functionality does not work.

```
srvctl config service -d rac -s pretaf
srvctl config service -d rac -s pretaf_preconnect
```

64. Run the following SQL query.

Again, observe the failover is not enabled for the two services.

```
sqlplus sys/oracle@rac as sysdba

col name format a20
col failover_method format a20
col failover_type format a20

SELECT NAME, FAILOVER_METHOD, FAILOVER_TYPE
FROM DBA_SERVICES WHERE NAME LIKE 'pretaf%';
```

65. In the tnsnames.ora file in your hosting PC, add the following connection descriptor. Do not copy the code from the PDF file.

Two connection names are required. One for the primary service and one for the shadow service.

```
pretaf =
 (DESCRIPTION=(FAILOVER=ON)
   (ADDRESS=(PROTOCOL=tcp)(HOST=192.168.56.81)(PORT=1521))
   (CONNECT_DATA=
        (SERVICE_NAME=pretaf.localdomain)
        (FAILOVER_MODE=
               (BACKUP=pretaf_preconnect)
               (TYPE=select)
               (METHOD=preconnect)))
  )

pretaf_preconnect =
 (DESCRIPTION=(FAILOVER=ON)
   (ADDRESS=(PROTOCOL=tcp)(HOST=192.168.56.82)(PORT=1521))
   (CONNECT_DATA=
        (SERVICE_NAME=pretaf_preconnect.localdomain))
  )
```

66. In the **client window**, login to the service using pretaf connection descriptor. Verify that the session is connected to rac1.

```
sqlplus soe/soe@pretaf

SELECT SYS_CONTEXT('USERENV','INSTANCE_NAME') FROM DUAL;
```

67. In the **admin window**, list the sessions created by soe user.

You will see two sessions. One connected to rac1 (the preferred node of the service) and it has its FAILOVER_METHOD set to PRECONNECT. The other is connected to the shadow service (pretaf_preconnect) on rac2.

```
col username format a10
col service_name format a20

SELECT INST_ID, SERVICE_NAME, FAILOVER_TYPE,FAILOVER_METHOD, FAILED_OVER
FROM GV$SESSION WHERE USERNAME='SOE' ORDER BY INST_ID;
```

68. In the **client window**, run the following long running SELECT statement then get straight away to the next step.

```
SELECT A.OBJECT_NAME FROM ALL_OBJECTS A UNION ALL SELECT B.OBJECT_NAME
FROM ALL_OBJECTS B;
```

69. In the admin window, crash rac1 instance.

```
pkill -9 -f ora_pmon_rac1
```

70. Observe the reaction of the client session.

You should notice that the client window pauses for a few seconds then proceeds displaying the returned rows. This is the expected behavior.

71. In the admin window, check out sessions created by soe user.

You should see now only one session. It is connected to rac2, its FAILOVER_METHOD changed to "PRECONNECT", and its FAILED_OVER status is now YES.

```
sqlplus sys/oracle@rac as sysdba

col username format a10
col service_name format a20
SELECT INST_ID, USERNAME, SERVICE_NAME, FAILOVER_TYPE,FAILOVER_METHOD,
FAILED_OVER
FROM GV$SESSION WHERE USERNAME='SOE';
```

72. Perform the following cleanup steps:

a) Exit from all SQL*Plus sessions.

b) Verify that rac1 is up and running again.

```
srvctl status instance -d rac -i rac1
```

c) Stop and delete pretaf service.

```
srvctl stop service -database rac -service pretaf
srvctl remove service -database rac -service pretaf
```

d) Delete pretaf and the pretaf_preconnect descriptors from the tnsnames.ora file

Summary

In this practice you learnt how to make the following configuration:

- Client-side connect-time load balancing
- Server-side connect-time load balancing with and without having the Load Balancing Advisory (LBA) enabled
- Client side Transparent Application Failover (TAF)
- server side basic TAF
- Client side Preconnect TAF
- Server side Preconnect TAF