Oracle® Fusion Middleware

Release Notes for Oracle Traffic Director

12*c* (12.2.1)

E52353-03

December 2015

This document contains information about the new features and known issues for Oracle Traffic Director 12.2.1. It contains:

What's New in This Release?

The following are the new features in Oracle Traffic Director 12.2.1. For more information, see *Administering Oracle Traffic Director*:

- The administration for Oracle Traffic Director has completely changed in 12.2.1 and is based on the WLS administration infrastructure
- Supports custom health checks, see *External Health Check Executable*
- Supports bandwidth limiting, see *Origin Server Traffic Control*
- Supports Prioritized Backend Connections, where you can prioritize your requests to the back end server. For more information, see *Prioritized Backend Connection Management*
- Supports maintenance mode for origin server pools, see *Origin Server and Origin Server Pool Maintenance*
- Supports connecting to origin servers via a forward proxy, see *Forward Proxy Support in Origin Server Pool*s
- The security library used in 12.2.1 is NZ and the cert/key store is Oracle Wallet. NSS is no longer supported
- Supports Test to production (T2P), where you can move a test environment to a production environment and vice-versa. For more information, see *Oracle Fusion Middleware T2P Utility for Oracle Traffic Director*
- Integrates with *WebLogic Server Multitenant*

Known Issues

This section provides information about the known issues for Oracle Traffic Director 12.2.1 along with possible workarounds:

ORACLE

FIPS Mode Support for OTD 12.2.1 on AIX.PPC64

When FIPS mode is enabled in an OTD Instance on AIX, OTD instance startup fails with the following error:

```
nzos_SetFipsMode is returning NZ error 29231 which is NZERROR_TK_CE_INIT = 29231
/* Crypto engine failed to initialize */
```

The OTD server does not start because of FIPS initialization failure due to RSA third party issues.

The solution to this issue is to take the following actions:

1. Create short symlinks from a short write-protected directory to library folders inside ${MW_HOME} as shown in this example:

   ```
   ln -s /usr/otd_user/12cOTD/mw_home/otd/lib /usr/lib1
   ln -s /usr/otd_user/12cOTD/mw_home/oracle_common/lib /usr/lib2
   ln -s /usr/otd_user/12cOTD/mw_home/lib/ /usr/lib3
   ```

   Where: *MW_HOME* is - `/usr/otd_user/12cOTD/mw_home` and `/usr` is the shortest secure folder

2. In this example in which the OTD instance is named "test", edit the script in

   ${DOMAIN_HOME}/config/fmwconfig/components/OTD/instances/otd_test_Machine-0/bin/startserv

   Change SERVER_LIB_PATH from

   ```
   SERVER_LIB_PATH="${SERVER_LIB_DIR}:${ORACLE_HOME}/lib:${ORACLE_HOME}/oracle_common/lib"
   ```

   to

   ```
   SERVER_LIB_PATH=/usr/lib1:/usr/lib2:/usr/lib3
   ```

Configuration update fails after starting failover

On Linux, if the otd_startFailover command has been succesfully issued at least once, all further configuration updates would fail. This is because the otd_startFailover command would create a keepalived.conf file under the config directory of the corresponding instance. Since the otd_startFailover command has to be executed as a privileged user (using sudo), the keepalived.conf created by it is owned by the privileged user and therefore the node manager is not able to update the instance whenever configuration updates are activated.

You can solve the issue by the following steps:

1. Once the otd_startFailover command has been issued, you have to modify the file permissions of the keepalived.conf for the corresponding instance to the same owner as one of the other configuration files under the config directory of that instance.

2. Alternatively, it can be automated using a WLST script which can invoked instead of the otd_startFailover WLST command. Check the sample script below.

   ```
   # Starts failover for an instance and changes ownership on keepaloved.conf
   # This script can only be run with root privileges
   # Usage:
   # sudo wlst.sh start_failover.py --domain-home=<wls_domain_dir>
   --instance=<instance_name>"
   # Mandatory options - domain-home, instance
   ```

```
import os
import os.path
import getopt

from java.nio.file import Files
from java.io import File

# Constants
DOMAIN_HOME = "domain-home"
INSTANCE_NAME = "instance"

valid_props = [DOMAIN_HOME, INSTANCE_NAME]

def main():
    args = process_args(valid_props, usage)
    try:
        try:
            otd_startFailover(args)
        except Exception, ex :
            handleError(ex)
    finally:
        change_keepalived_conf_ownership(args[DOMAIN_HOME], args[INSTANCE_
NAME])

def change_keepalived_conf_ownership(domain_home, instance_name):
    instance_root = domain_home + "/config/fmwconfig/components/OTD/instances/"
    instance_config_dir = instance_root + instance_name + "/config/"
    keepalived_conf = instance_config_dir + "keepalived.conf"
    if os.path.exists(keepalived_conf):
        server_xml = instance_config_dir + "server.xml"
        owner = Files.getOwner(File(server_xml).toPath(), [])
        Files.setOwner(File(keepalived_conf).toPath(), owner)

def usage():
    print "Usage: sudo wlst.sh start_failover.py --help|-h"
    print "    or sudo wlst.sh start_failover.py --domain-home=<wls_domain_dir>
--instance=<instance_name>"

def process_args(all_props, usage):
    try:
        prop_names = ["help"]
        for prop_name in all_props:
            prop_names.append(prop_name + "=")

        # Parse the command line arguments using getopt.getopt which will
return options (--option)
        # and arguments (any arguments).
        # It also validates whether any irrelavant options/arguments are
specified apart from the ones
        # that are specified in getopt and throws the error appropriately
        # For more information on getopt look at
http://docs.python.org/2/library/getopt.html
        opts, args = getopt.getopt(sys.argv[1:], "h", prop_names)

    except getopt.GetoptError, err:
        # This occurs when invalid options/arguments are specified. So print
the help information and exit
        if (usage != None):
            usage()
```

```
            handle_error(err)
        sys.exit(1)

    props = {}

    # Take the appropriate action based on the options specified and also
override the default props with user specified props
    for key, value in opts:

       if key in ["--help", "-h"]:
            usage()
            sys.exit(1)
        else:
            # Override the instance properties with the user specified
properties.
            # Note that key contains values with "--" prefix, for example
--option.
            # So we need to remove the prefix using key[2:]
            props[key[2:]] = value
    return props

# If the function argument is a java.lang.Exception then print the exception
stack trace and exit
# else just print the error message to stderr and exit
def handle_error(error):
    if isinstance(error, JavaException):
        error.printStackTrace()
    else :
        sys.stderr.write(str(error) + "\n")
    sys.exit(1)

# Execution starts here ...
if __name__ == "main":
    main()
```

Note: The script can be saved anywhere with the name start_failover.py. The script should be invoked as follows: wlst.sh <Path to start_failover.py> --domain-home=<wls_domain_dir> --instance=<instance_name>.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.