
KenTest.JSP

Verify and Monitor Infra Components
of a WebLogic Environment

Ken Gottry
ken@gottry.com

Jun-2003

Objective

- Describe a single JSP that can test and verify all aspects of the infra of a WebLogic environment
- Describe how the JSP can be used as an initial step in troubleshooting to help determine if the problem is infra or app
- Describe how the JSP can be executed by a network device to continually monitor the health of the infra

N.B. This presentation describes a method of verifying the installation and health of any infrastructure environment that uses WebLogic as its application server. This method is independent of the application that may eventually use the infra.

Agenda

- Executive summary
- Overview of monitoring capabilities
- What is a JSP
- KenTest JSP overview
- JDBC connection pools and multi-pools
- Installing KenTest.jsp
- Executing KenTest.jsp
- Passing parameters to KenTest.jsp
- Current version of KenTest.jsp
- Using a more complex SQL query
- Other ways to execute KenTest.jsp
- Appendix A – WebLogic JDBC Multi-pools

Executive Summary

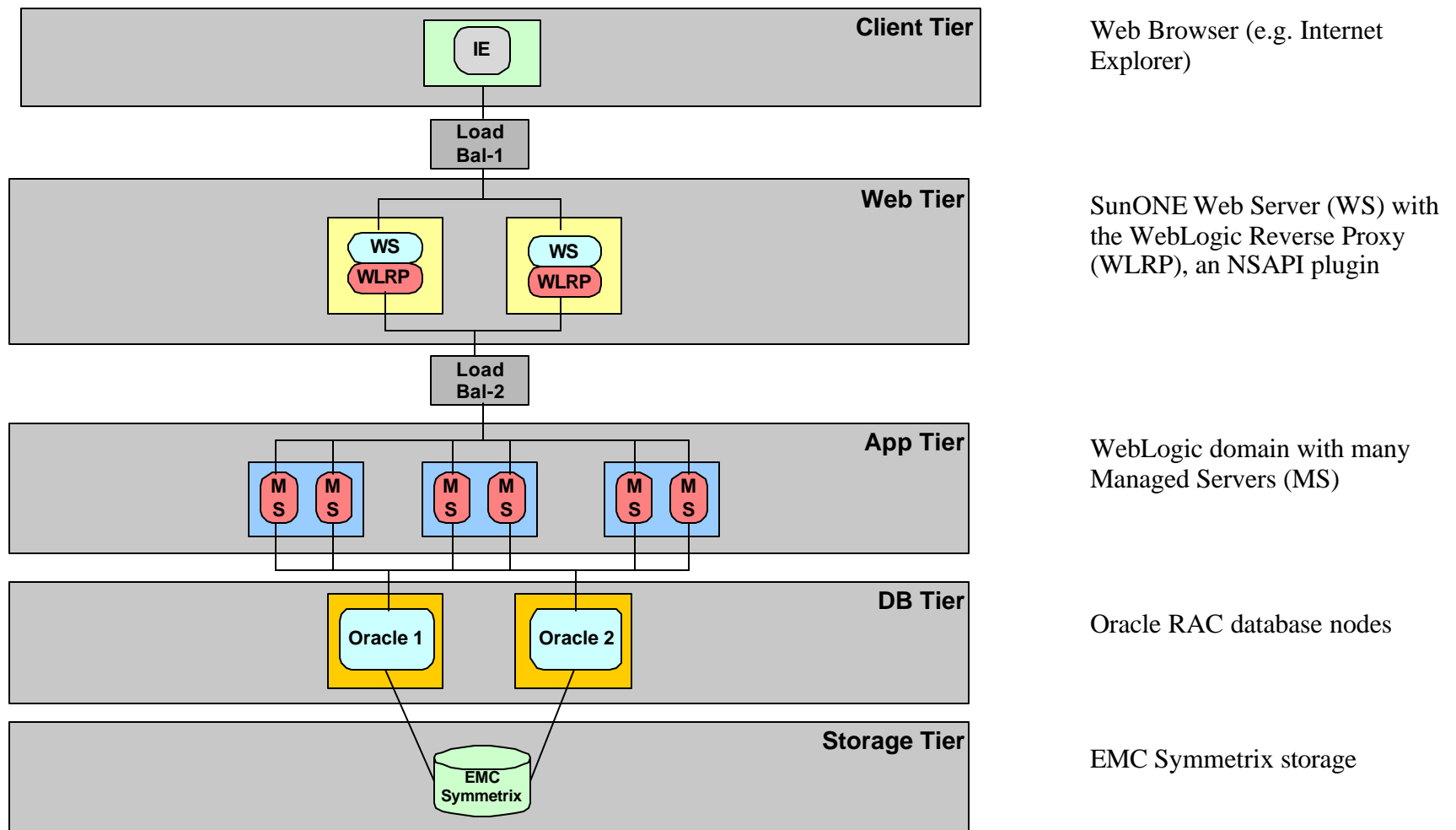
What does KenTest.JSP do?

Executive Summary

- KenTest.JSP is a simple Java program that
 - Is launched from a web browser
 - Selects an entry from a WebLogic JDBC pool
 - Issues a SQL call to the database
 - Returns the results of the SQL query to the browser
- KenTest.JSP can be used to
 - Test the impact of different WebLogic JDBC configuration settings
 - Verify the configuration of the technical infrastructure before the application is ready and installed. The verification includes
 - Config files like SunONE obj.conf and WebLogic config.xml
 - IP and port numbers
 - Firewalls and load balancers
 - Verify the infra following major configuration changes
 - Assist troubleshooting to determine if a problem is related to app or infra
- KenTest.JSP can be used and maintained by your Infra staff with little or no support from your AppDev staff

Overview of Monitoring Capabilities

Typical WebLogic Infra Configuration

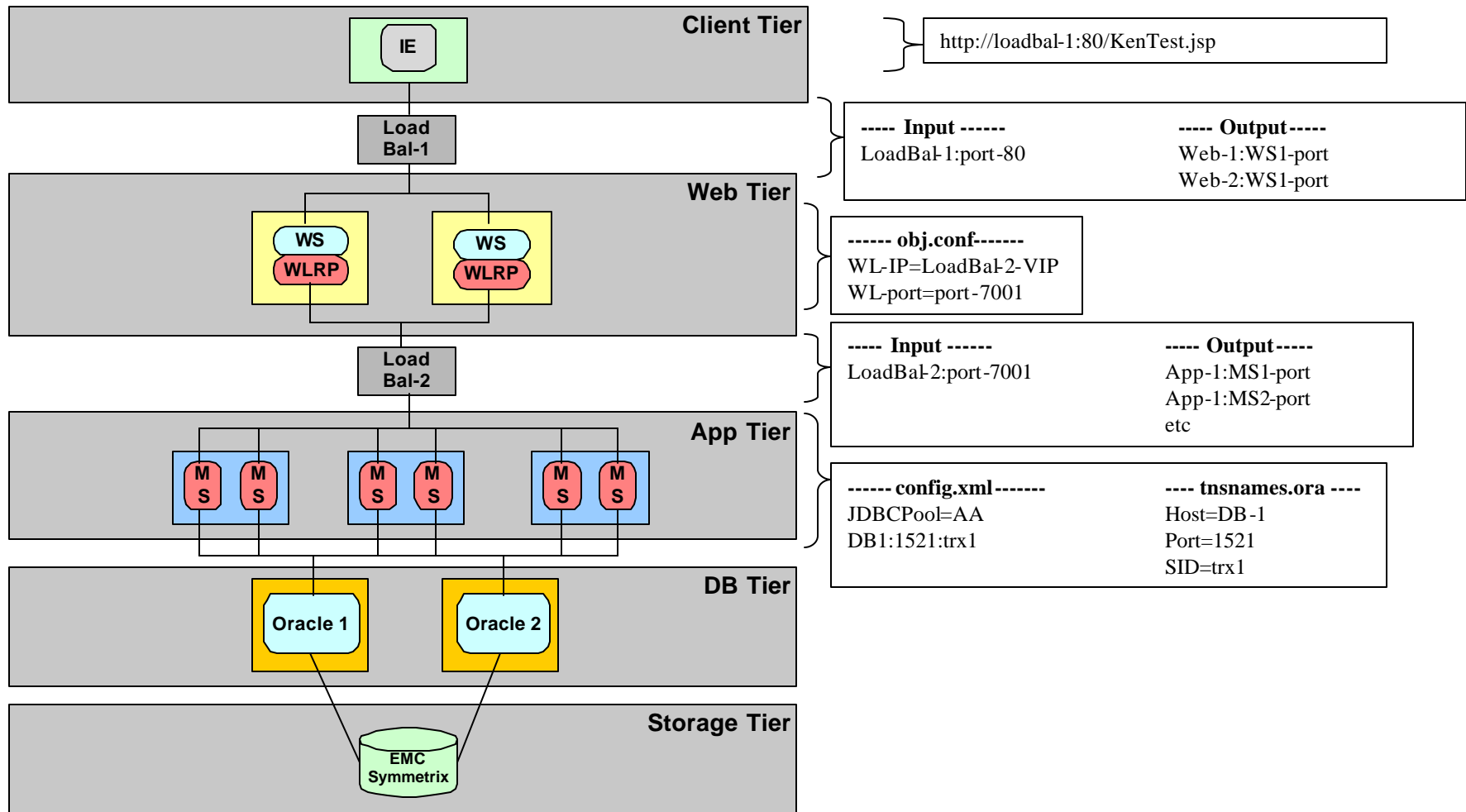


Infra Components Tested

- Network between client and frontend firewall
- Frontend firewall
- Network between firewall and frontend load balancer
- Frontend load balancer
- Network between frontend load balancer and web tier
- SunONE web server process
- WebLogic reverse proxy (WLRP) – NSAPI plug-in
- Network between web tier and backend load balancer
- Backend load balancer
- WebLogic HTTP listener
- WebLogic execute thread pool
- WebLogic JDBC connection pool
- Network between app tier and database tier
- Oracle processes
- Oracle database
- EMC Symmetrix

Tests everything except
the application

Infra Configurations Tested



Typical Uses of KenTest.JSP

- Verify that all infra components are installed and configured correctly
 - IP addresses
 - Ports
 - Firewalls
 - SunONE obj.conf
 - WebLogic config.xml
 - Oracle tnsnames.ora
 - Directories, log files, users, group, permissions
 - EMC volumes
- Perform infra verification
 - Prior to initial app installation
 - First step in troubleshooting any problem
 - Whenever major changes are made to the infra
- Help Desk to determine if a problem is infra or app. Allows trouble calls to be routed to the appropriate support group

What is a JSP

Servlet vs. JSP

Two ways to write Java HTTP programs

1. Servlet

- a) Write Java program that performs application logic and generates HTML code to define screen layout (e.g. KenTest.java)
- b) One-time compilation of program (creates KenTest.class)
- c) Store compiled program (class file) on server in CLASSPATH directory
- d) Browser requests servlet which executes application logic and returns HTML code
`http://www.gottry.com/appdir/KenTest`

2. JSP (Java Server Page)

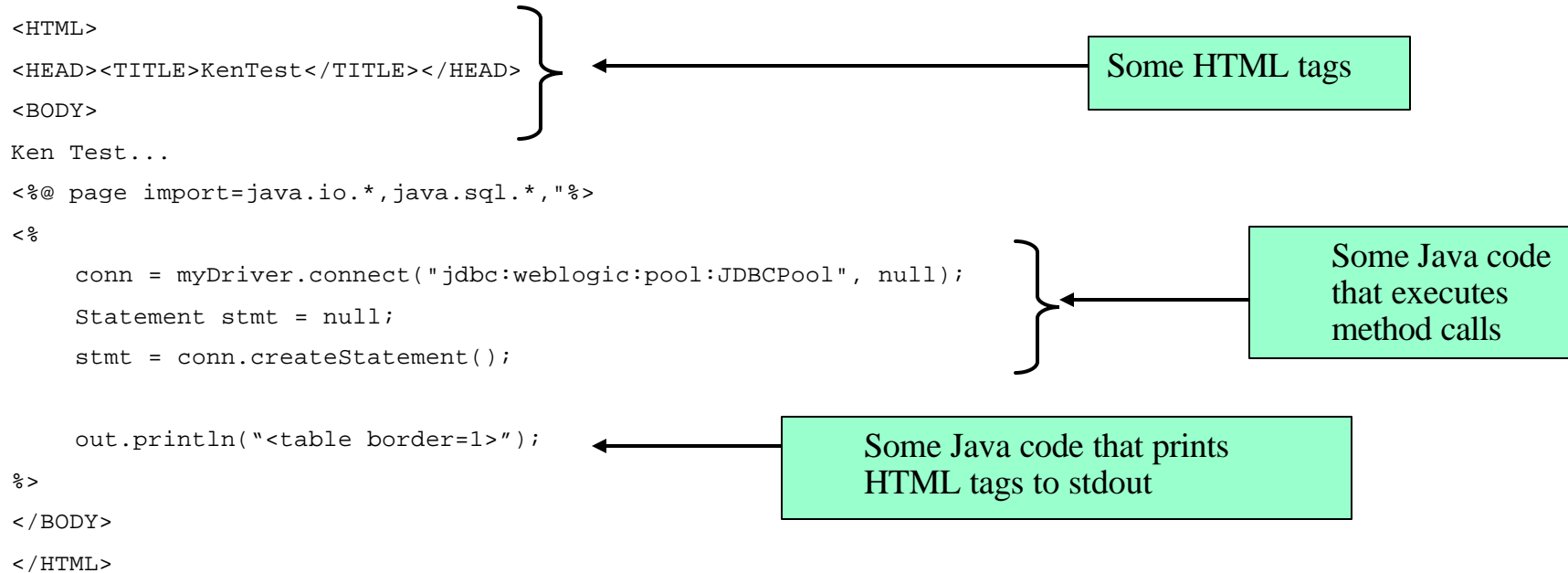
- a) Write JSP text file that contains HTML tags and Java code (e.g. KenTest.jsp)
- b) Store JSP text file on server in Application directory
- c) Browser requests JSP
`http://www.gottry.com/appdir/KenTest.jsp`
- d) JVM dynamically compiles JSP into servlet
- e) JVM executes compiled servlet which returns HTML code

N.B. these definitions are from my infra perspective. I'm sure AppDev folks have more complete, more accurate definitions

JSP is somewhat analogous to Microsoft's ASP (active server page)

KenTest.jsp Overview

Excerpts from a sample JSP



Sample Sections of KenTest.jsp

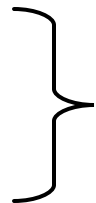
```
<HTML>
<HEAD><TITLE>Ken Test</TITLE></HEAD>
<BODY>Ken Test...
<%@ page import="
java.io.*,java.sql.*,
javax.servlet.*,javax.servlet.http.*,
java.util.Properties,
weblogic.db.jdbc.*"%>

conn = myDriver.connect("jdbc:weblogic:pool:JDBCPool1", null);

rs = stmt.executeQuery("select sysdate from dual");

while (rs.next()) {
    java.sql.Date nowdate = rs.getDate("sysdate");
    out.println(nowdate);
}

conn.close();
</BODY></HTML>
```



Include the Java and WebLogic class files

Select WebLogic JDBC pool to use

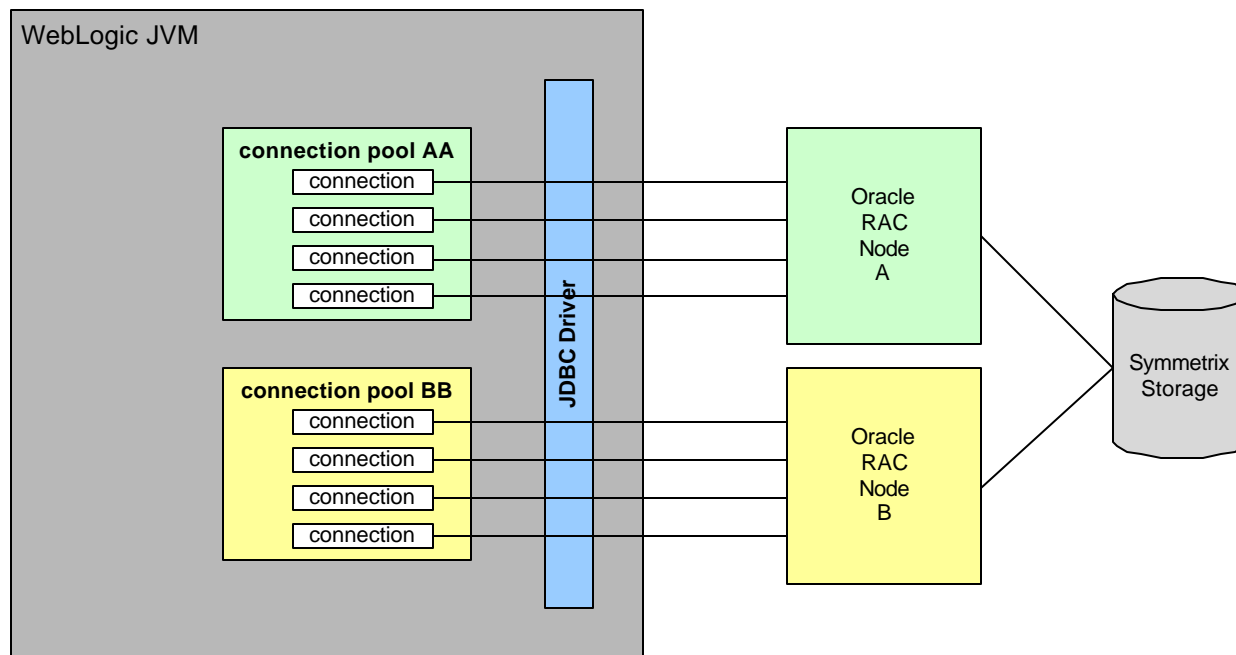
Define SQL stmt and execute it

Retrieve ResultSet and write to stdout

JDBC Connection Pools

JDBC Connection Pool

A JDBC connection pool contains a group of JDBC connections that are created when WebLogic Server starts. JDBC connection pools use a type 2 (thick) or a type 4 (thin) JDBC driver to create physical database connections. The application borrows a connection from the pool, uses it, then returns it to the pool by closing it



Sample JDBC Configuration

```
<JDBCConnectionPool
  CapacityIncrement="1"
  DriverName="oracle.jdbc.driver.OracleDriver"
  InitialCapacity="10"
  JDBCXADebugLevel="0"
  LoginDelaySeconds="0"
  MaxCapacity="10"
  Name="JDBCPool1"
  Properties="tableowner=ken;user=ken;password=ken;dll=ocijdbc9;protocol=thin"
  RefreshMinutes="0"
  ShrinkPeriodMinutes="15"
  ShrinkingEnabled="false"
  SupportsLocalTransaction="false"
  Targets="ManagedServer1"
  TestConnectionsOnRelease="false"
  TestConnectionsOnReserve="true"
  TestTableName="dual"
  URL="jdbc:oracle:thin:@10.1.1.1:1521:sida" />
```

Name of the JDBC pool. You need to specify this in KenTest.jsp

The database you want to query

WebLogic JDBC MultiPools

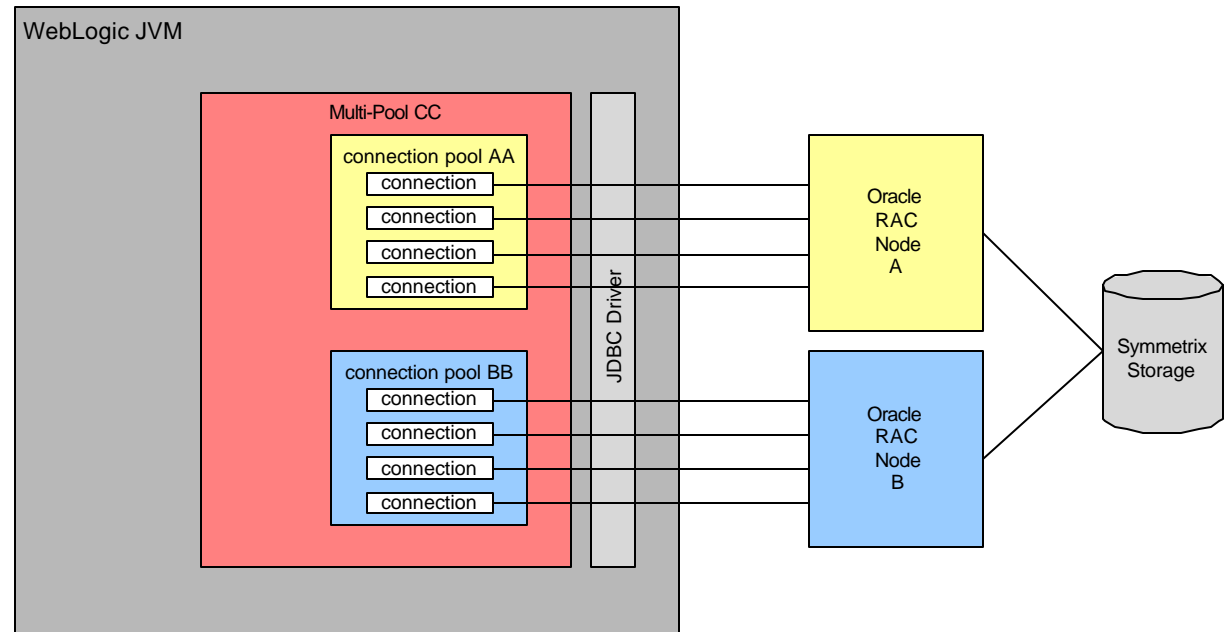
WebLogic's JDBC MultiPool

After defining two or more JDBC connection pools, you define one JDBC multi-pool and assign each JDBC connection pool to be a member of the multi-pool. The multi-pool is given a unique name to differentiate it from the individual JDBC connection pools.

The advantage of the multi-pool is that the application only need know the name of the multi-pool and not the name of each individual JDBC connection pool.

MultiPools can be either

- Load Balancing
- High Availability



Appendix A contains more info about WebLogic JDBC Multi-pools and how KenTest.jsp can be used to test them

Sample JDBC MultiPool Config

```
<JDBCConnectionPool
  CapacityIncrement="1"
  DriverName="oracle.jdbc.driver.OracleDriver"
  InitialCapacity="10"
  MaxCapacity="10"
  Name="JDBCPool1"
  Properties="tableowner=ken;user=ken;password=ken;dll=ocijdbc9;protocol=thin"
  Targets="ManagedServer1"
  URL="jdbc:oracle:thin:@10.1.1.1:1521:sida" />
<JDBCConnectionPool
  CapacityIncrement="1"
  DriverName="oracle.jdbc.driver.OracleDriver"
  InitialCapacity="10"
  MaxCapacity="10"
  Name="JDBCPool2"
  Properties="tableowner=ken;user=ken;password=ken;dll=ocijdbc9;protocol=thin"
  Targets="ManagedServer1"
  URL="jdbc:oracle:thin:@10.1.1.2:1521:sidb" />
<JDBCMultiPool
  Name="MultiPool-HA"
  AlgorithmType="Load-Balancing"
  PoolList="JDBCPool1,JDBCPool2" Targets="managedServer1"/>
```

When KenTest.jsp asks WebLogic for a connection from this MultiPool, it will get a connection from JDBCPool1 and JDBCPool2 in a round-robin load-balancing fashion

Installing KenTest.jsp

Copy JSP to Server Directory

When WebLogic is installed and the Domain Wizard runs, the WebLogic *DefaultWebApp* is installed. It's easiest to install KenTest.jsp in the document root directory for this app.

- Locate the WebLogic home directory (e.g. */usr/local/boa*)
- Locate the projects directory (e.g. *users_projects*)
- Locate the appropriate domain directory (e.g. *kendomainstest*)
- Navigate to the *applications/DefaultWebApp* directory
- Copy the KenTest.jsp into this directory ... for example:

/usr/local/boa/user_projects/kendomainstest/applications/DefaultWebApp

Making Changes to KenTest.JSP

- WebLogic can be started in Development Mode or Production Mode
- A parameter controls how often WebLogic looks for a new version of a JSP
 - 0 look for a new version of the JSP every time it's executed
 - 1 never look for a new version of the JSP
 - >0 look for a new version of the JSP every NN minutes
- Depending upon the settings of these parameters changes to a JSP may not take effect for a long time, perhaps not until WebLogic is restarted.
- Therefore, I usually keep several copies of KenTest on the server, each copy with a different set of execution options

```
-rw-r--r-- 1 weblogic weblogic 1002 May  7 10:39 Kentest.jsp
-rw-r--r-- 1 weblogic weblogic 1010 May  7 10:40 Kentest_no_close.jsp
-rw-r--r-- 1 weblogic weblogic 1513 May  7 15:07 Kentest_sleep.jsp
-rw-r--r-- 1 weblogic weblogic 1004 May 12 14:15 Kentest_thick.jsp
-rw-r--r-- 1 weblogic weblogic 1009 May 20 06:52 Kentest_thick_Multipool.jsp
-rw-r--r-- 1 weblogic weblogic 1289 May 20 07:00 Kentest_thick_Multipool_sleep.jsp
```

Executing KenTest.jsp

Executing the JSP with a Browser

There are four different ways to execute KenTest.jsp

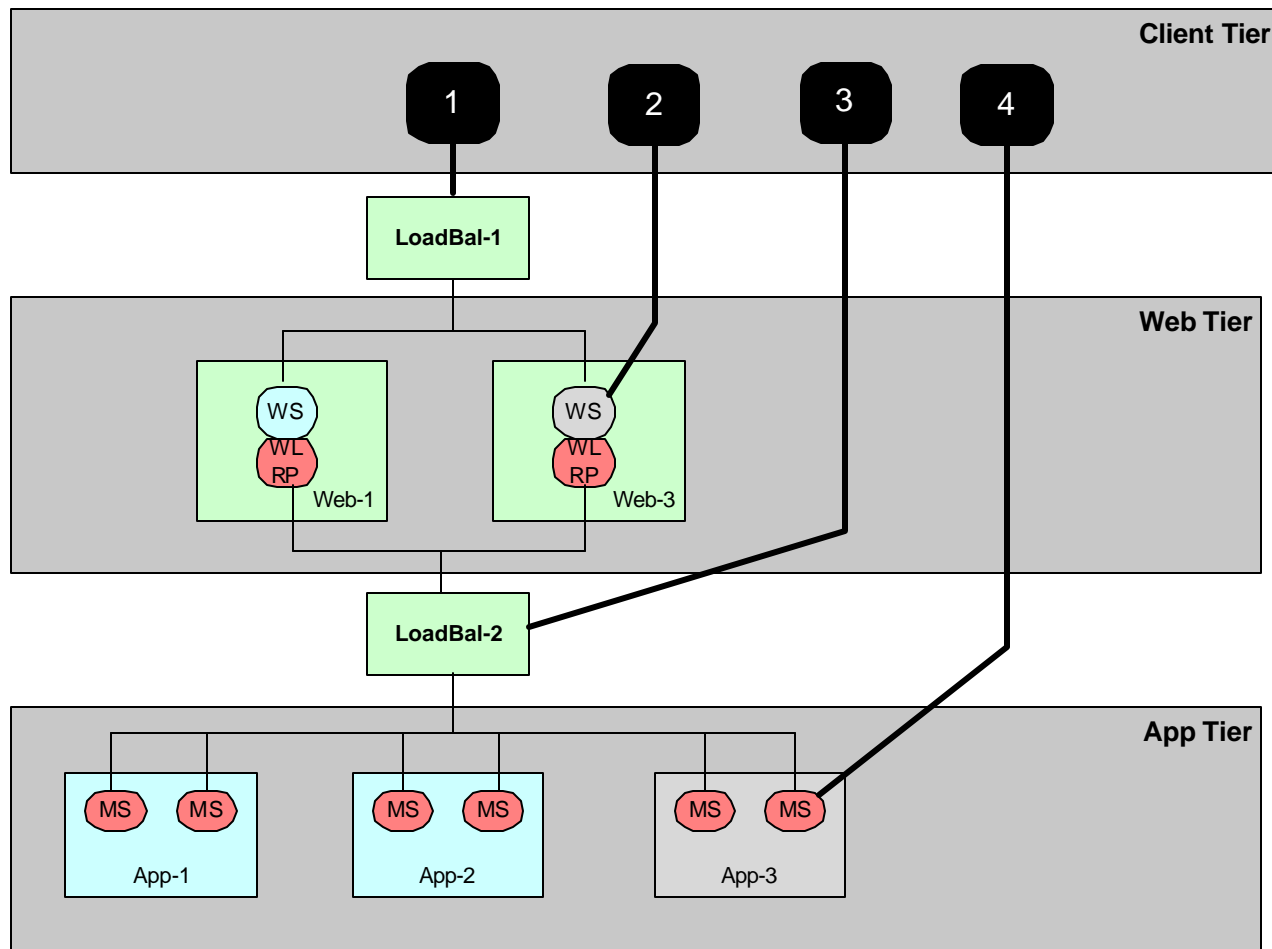
- 1) Point your browser at frontend load balancer
`http://loadbalancer.frontend.gottry.com:80/KenTest.jsp`
- 2) Point your browser at web server
`http://webserver.gottry.com:80/KenTest.jsp`
- 3) Point your browser at backend load balancer (assuming the frontend firewall in front of the web servers allows this traffic)
`http://loadbalancer.backend.gottry.com:7001/KenTest.jsp`
- 4) Point your browser at WebLogic server (assuming backend firewall between web and app tiers allows this traffic)
`http://appserver1.gottry.com:7001/KenTest.jsp`

N.B.- The URL doesn't contain an application directory because the *DefaultWebApp* uses the / directory. If KenTest.jsp is installed as the *InfraUtil* app, then the URL would be

`http://loadbalancer.frontend.gottry.com:80/InfraUtil/KenTest.jsp`

Possible Target Points

This slide illustrates the four different options for executing Kentest.jsp as described on the previous slide



Passing Parameters To KenTest.jsp

Passing Parameters to a JSP

It is possible and usually desirable to pass parameters to a JSP to customize its behavior to meet your processing needs:

- Place a question mark following the JSP
 - (e.g. `http://server.gottry.com/KenTest.jsp?parameters_go_here`)
- Parameters are specified using a *name=value* syntax
 - (e.g. `http://server.gottry.com/KenTest.jsp?salary=100`)
- When specifying multiple parameters, separate them with an ampersand
 - (e.g. `http://server.gottry.com/KenTest.jsp?Firstname=Ken&LastName=Gottry`)
- The name of the parameter on the HTTP request must match exactly (case sensitive) with the name of the parameter within the JSP code
 - `String EmployeeLastName = request.getParameter("LastName");`

Parameters Used by Kentest.JSP

Sleep

- This causes the JSP to sleep for the specified number of milliseconds after issuing the SQL call and before returning the JDBC entry to the pool.
- This is used to simulate processing time within a real application
- Sleep=0 means do not sleep [the default]
 - (e.g. <http://server.gottry.com/KenTest.jsp?Sleep=2000>)

JDBC

- The name of the JDBCConnectionPool (as defined in the WebLogic config.xml) to use.
- Can also specify a JDBCMultiPool
 - (e.g. <http://server.gottry.com/KenTest.jsp?JDBC=LoginPool>)

Close

- Close=Y [the default] indicates the JSP should close the JDBC connection when finished.
- While a well-behaved application should always close a JDBC connection when finished using it, you may want to emulate an ill-behaved application
- Not closing JDBC connections is a good way to watch WebLogic's behavior as the JDBC pool has to grow to support new requests
 - (e.g. <http://server.gottry.com/KenTest.jsp?Close=N>)

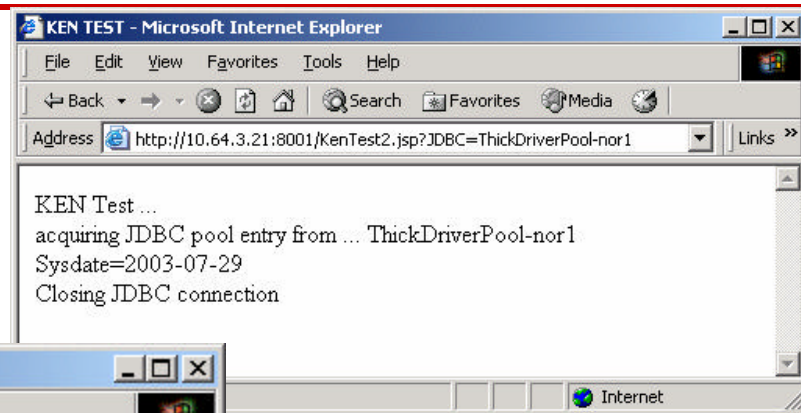
Parameters can be combined

- (e.g. <http://server.gottry.com/KenTest.jsp?Sleep=2000&JDBC=HAMultipool>)

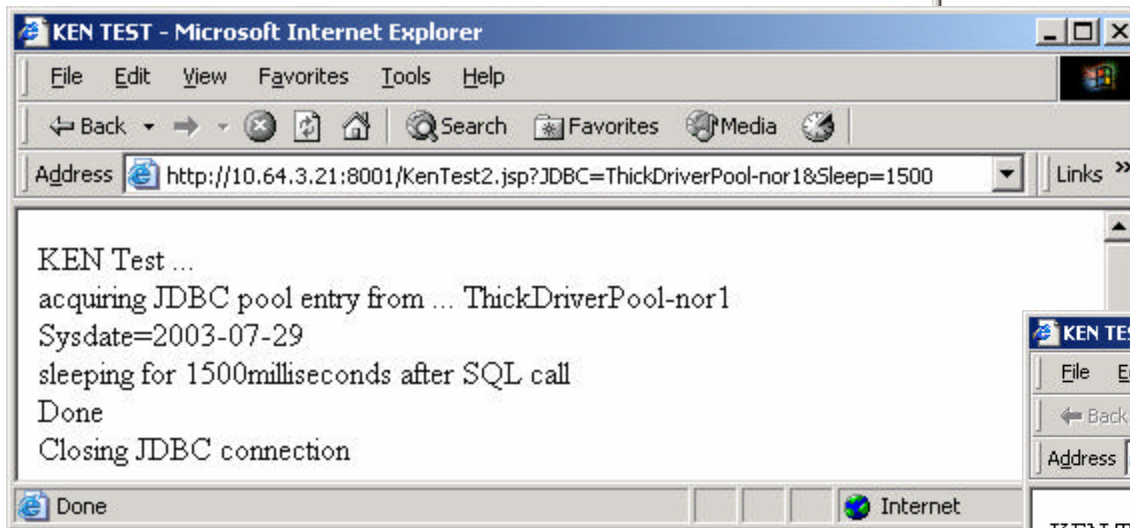
Sample Output from KenTest.jsp

You must always specify the name of the JDBC pool to be used by the JSP or set the default JDBC pool in the JSP source.

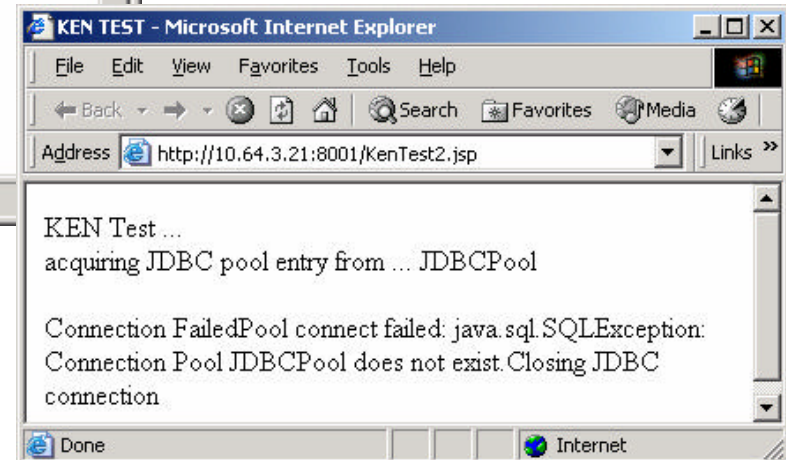
The JSP will display the name of the JDBC pool used and the results of the SQL query (i.e. SysDate)



Sleeping for 1.5 seconds (1500 ms) to simulate application processing that occurs after the SQL call and before the JDBC entry is returned to the pool



You will get a SQL exception if you specify an incorrect JDBC pool name



Current Version Of KenTest.jsp

Current Version of KenTest.jsp

The following slides dissect KenTest.jsp to help you can understand what it does and how it does it.

If you are interested in obtaining a copy of the JSP file, please download it from <http://articles.gottry.com> rather than trying to cut-n-paste from the following slides.

Current Version of KenTest.jsp – Part 1

```
/* Start of Kentest.jsp */
```

```
<HTML>
```

```
<HEAD><TITLE>KEN TEST</TITLE></HEAD>
```

```
<BODY>
```

```
KEN Test ... <BR>
```

← Echo some standard HTML tags to stdout. This begins the output that will be returned to the browser

```
<%@ page import="
```

```
java.io.*,
```

```
java.sql.*,
```

```
javax.servlet.*,
```

```
javax.servlet.http.*,
```

```
java.util.Properties,
```

```
weblogic.db.jdbc.*
```

```
"%>
```

← Import some Java and WebLogic classes. Note – these WebLogic classes are valid for version 7. The names of the JDBC classes have changed in WebLogic 8.1.

```
/* continued on next slide */
```

Current Version of KenTest.jsp – Part 2

```
/* continued from previous slide */
```

```
<%
```

```
/* -----
```

```
get parameters from HTTP request
```

```
http://server.gottry.com:7001/KenTest.jsp?Sleep=0&JDBC=TestPool
```

```
where
```

```
Sleep is number of milliseconds to sleep after issuing the SQL call before releasing the JDBC entry  
back to the pool This simulates "processing time" in a real application
```

```
JDBC is the name of the JDBCConnectionPool as defined in the WebLogic config.xml file. This can also  
be the name of a WebLogic JDBCMultiPool
```

```
Close is a Y/N switch to indicate if the JSP should emulate a well-behaved application and close the  
JDBC entry was done (Close=Y)
```

```
-----*/
```

```
String SleepSeconds = request.getParameter("Sleep");
```

```
String JDBC = request.getParameter("JDBC");
```

```
String CloseIt = request.getParameter("Close");
```

```
if(SleepSeconds == null) { SleepSeconds = "0";}
```

```
if(JDBC == null) { JDBC = "JDBCPool";}
```

```
if(CloseIt == null) { CloseIt = "Y";}
```

```
int SleepInterval = Integer.parseInt(SleepSeconds);
```

```
String JDBCPool = "jdbc:weblogic:pool:" + JDBC;
```

```
%>
```

```
/* continued on next slide */
```

www.gottry.com

Get the parameters from the URL

Default values for parameters

Convert Sleep from string to integer

Convert JDBC pool name to format
required by method call

Current Version of KenTest.jsp – Part 3

```
/* continued from previous slide */
```

```
<%
```

```
Connection conn = null;
try {
    out.println("acquiring JDBC pool entry from ... " + JDBC + "<BR>" );
    out.flush();
    Driver myDriver = (Driver) Class.forName("weblogic.jdbc.pool.Driver").newInstance();
    conn = myDriver.connect(JDBCPOOL, null);

    Statement stmt = null;
    ResultSet rs = null;

    stmt = conn.createStatement();
    rs = stmt.executeQuery("select sysdate from dual");

    while (rs.next()) {
        java.sql.Date nowdate = rs.getDate("sysdate");

        out.println("Sysdate=" + nowdate + "<BR>");
        out.flush();
    }
}
```

Get a DB connection using the JDBC pool

Execute SQL query

Get ResultSet from SQL query

Write ResultSet in HTML
format to stdout

```
/* continued on next slide */
```

Current Version of KenTest.jsp – Part 4

/* continued from previous slide */

```
    if(SleepInterval != 0)

        try {

            out.println("sleeping for " + SleepSeconds + "milliseconds after SQL call<BR>" );
            out.flush();
            Thread.sleep(SleepInterval);
            out.println("Done<BR>" );
            out.flush();

        }

        catch (Exception e){

            out.print("sleep after Failed<BR>" + e.getMessage());

        }

    }
```

If requested, sleep for specified period to simulate application processing time

/* continued on next slide */

Current Version of KenTest.jsp – Part 5

```
/* continued form previous slide */
```

```
    } catch (Exception e){  
        out.print("<BR>Connection Failed" + e.getMessage());    }
```

Catch exception if JDBC
connection failed

```
finally {
```

```
    if (CloseIt == "Y") {
```

```
        out.println("Closing JDBC connection<BR>" );
```

```
        out.flush();
```

```
        if(conn != null)
```

```
            try{
```

```
                conn.close();    }
```

```
            catch (Exception e){
```

```
                out.print("Exception:" + e);    }
```

If requested, close JDBC entry, which does
not close the DB connection just returns
the JDBC entry to the pool

```
        } else {
```

```
            out.println("*** skipping the close for JDBC connection<BR>" );
```

```
            out.flush();    }
```

```
    }
```

```
%>
```

```
</BODY>
```

```
</HTML>
```

Write trailing HTML tags to stdout. This
completes the info that is returned to the browser

```
/* End of Kentest.jsp */
```

Using More Complex SQL

Using a More Complex SQL Query

Below are snippets of an enhanced version of KenTest.jsp. This JSP executes a more complex SQL query that returns multiple rows. The JSP formats each returned row as a row in an HTML Table.

```
String query = "select login_name, login_user_num from login_user";
rs = stmt.executeQuery(query);

<table><tr><td>Login Name</td><td>Login User Number</td></tr>
<%
while (rs.next()) {
    s = rs.getString("login_name");
    n = rs.getInt("login_user_num");
%>
    <TR><td><%=s%> </td> <td><%=n%> </td></tr><%}%>
</table>
```

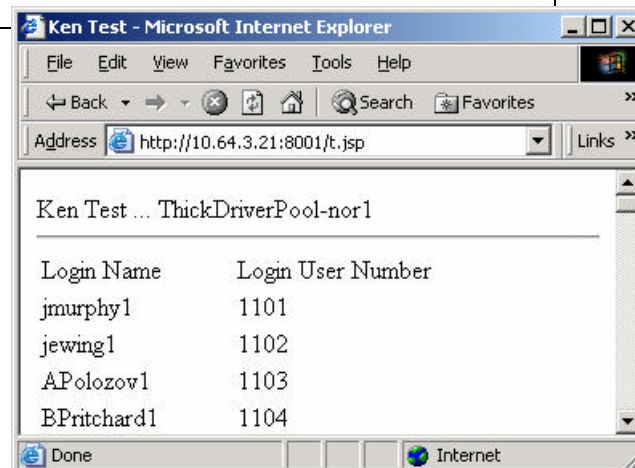
More complex SQL query

HTML tags to define table

Use *while* loop to retrieve rows from DB table

Write data for each table row. Also include HTML tags to define rows and columns

Sample output from this SQL query



Other Ways to Execute KenTest.jsp

Other Ways to Execute the JSP

The example so far have assumed you are executing the JSP by launching a browser (e.g. Internet Explorer or Netscape Communicator). There are other ways to use this JSP

- Use a stress tool like Mercury Interactive's LoadRunner or Segue's Silk (or my favorite The Grinder from SourceForge.Net) to execute the URL repeatedly to conduct a performance test against the infra without any application components
 - Execute the JSP multiple times each time with different parameters, such as different values for *Sleep=*
- Use load balancer (e.g. Cisco content switch) to periodically execute the JSP to monitor the health of all infra components

You can also use Grinder to continually execute KenTest.jsp while you test failover (high availability) behavior. For example, while you are simulating many virtual users, each executing many KenTest.jsp transactions, you can crash one of your DB nodes to see if the WebLogic JDBC HA MultiPool begins to route requests to the surviving DB node. You can monitor the response time of the JSP during the failure detection to assess the impact of a DB failure. See Appendix A for more details

CSS 11000

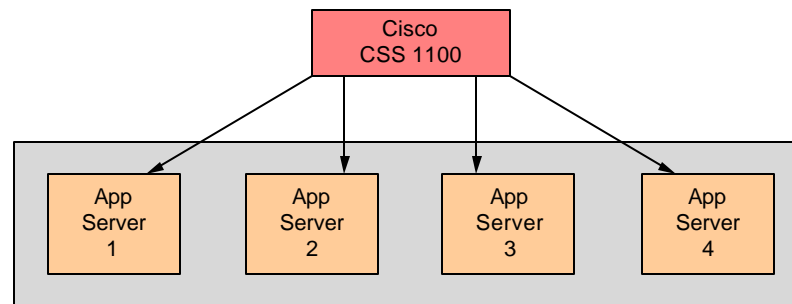
Cisco Content Service Switch

N.B. The examples in this section use the Cisco content switch (CSS 11000) because I have sample config files for that device to illustrate my points. Similar functionality is presumed to be available via network devices from other vendors

CSS App Server Farm

Traditional CSS configuration

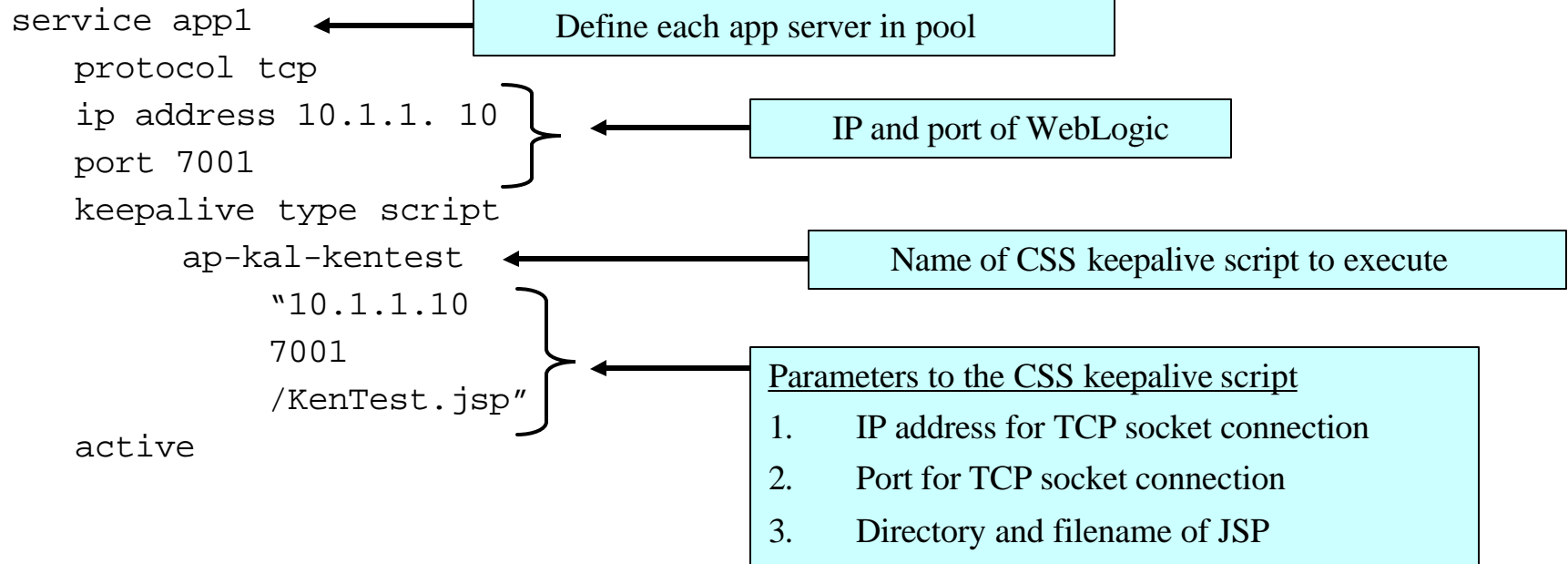
- The CSS is configured to have a “farm” of app servers.
- When a client request is received, the CSS sends the request to the app server that is least busy
- The CSS periodically polls each app server to determine if it is alive.
- When an app server fails to respond to the CSS’s keepalive query, then the app server is temporarily removed from the “farm”.
- The CSS will not forward requests to an app server that has been removed from the “farm”



CSS configuration using KenTest.jsp

- The traditional configuration only determines if the WebLogic server is listening on the specified port.
- By enhancing the CSS configuration to periodically issue a request for KenTest.jsp, the CSS will know if the WebLogic execute thread pool is alive, if the JDBC pool is functioning, and if the network between the app and database server is functioning.
- This enhanced configuration will help the CSS better determine if an app server has become stale and should be removed from the “farm”

CSS Service Definition



CSS Keepalive Script

```
!----- ensure proper arguments
if ${ARGS}[#] "NEQ" "3"
    echo "Bad syntax. Usage: ap-kal-kentest \'10.1.1.1 7001 /KenTest.jsp\'"
    exit script 1
endbranch

!----- snarf command line arguments
set HostName "${ARGS}[1]"
set PortNbr "${ARGS}[2]"
set WebPage "${ARGS}[3]"

!----- create socket connection remote Host
set EXIT_MSG "Connection Failure"
socket connect host ${HostName} port ${PortNbr} tcp 2000

!----- Send the GET request for the KenTest JSP
!----- (e.g. GET /InfraUtils/KenTest.jsp HTTP/1.0\r\n
set EXIT_MSG "Waitfor: Failed"
socket send ${SOCKET} "GET ${WebPage} HTTP/1.0\n"
socket send ${SOCKET} "\n\n"
socket waitfor ${SOCKET} "200 OK" 2000

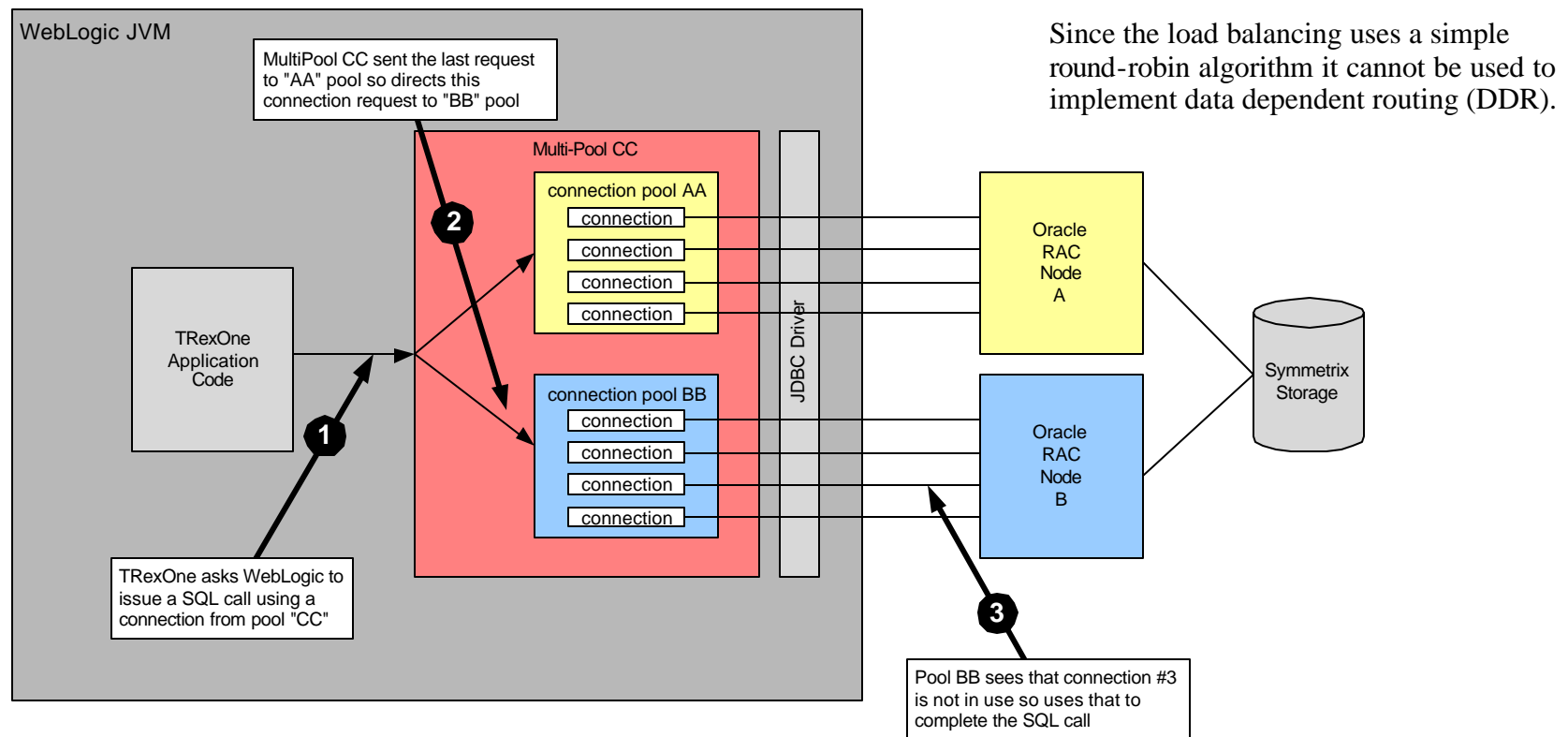
!----- tear down and go home
no set EXIT_MSG
socket disconnect ${SOCKET} graceful
exit script 0
```

Appendix A

WebLogic JDBC MultiPools

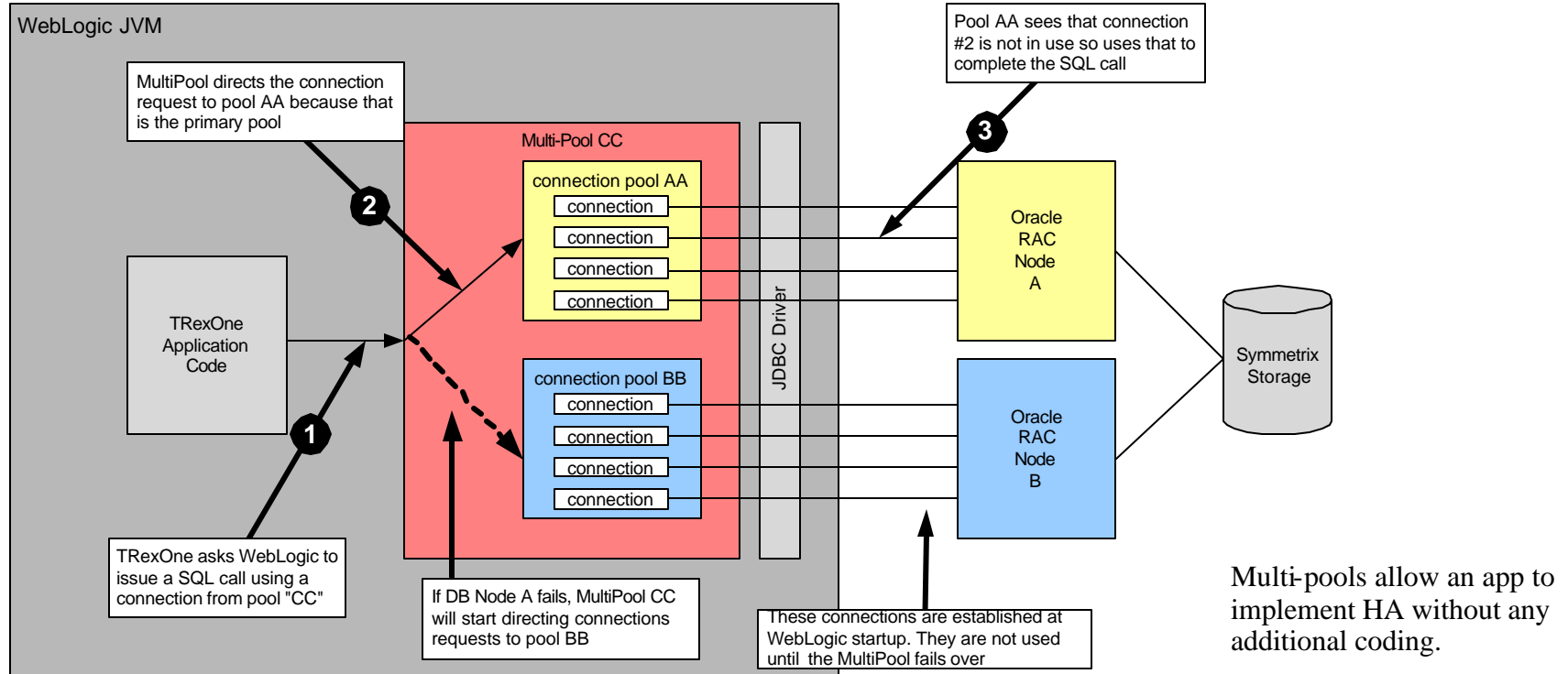
MultiPool – Load Balancing

In the load balancing configuration, the SQL requests are distributed across all of the JDBC connection pools in the multi-pool on a round-robin basis. If one connection pool fails (e.g. because of a failure of the Oracle instance), then that connection pool will be temporarily removed from the multi-pool.



MultiPool – High Availability

The other type of multi-pool, high availability, has one connection pool in the multi-pool designated as primary and the other connection pools are secondary. All SQL requests are routed through the primary connection pool until that pool fails (e.g. because of a failure of the Oracle instance). Then the secondary connection pool becomes the primary pool and all SQL requests are routed through that connection pool.



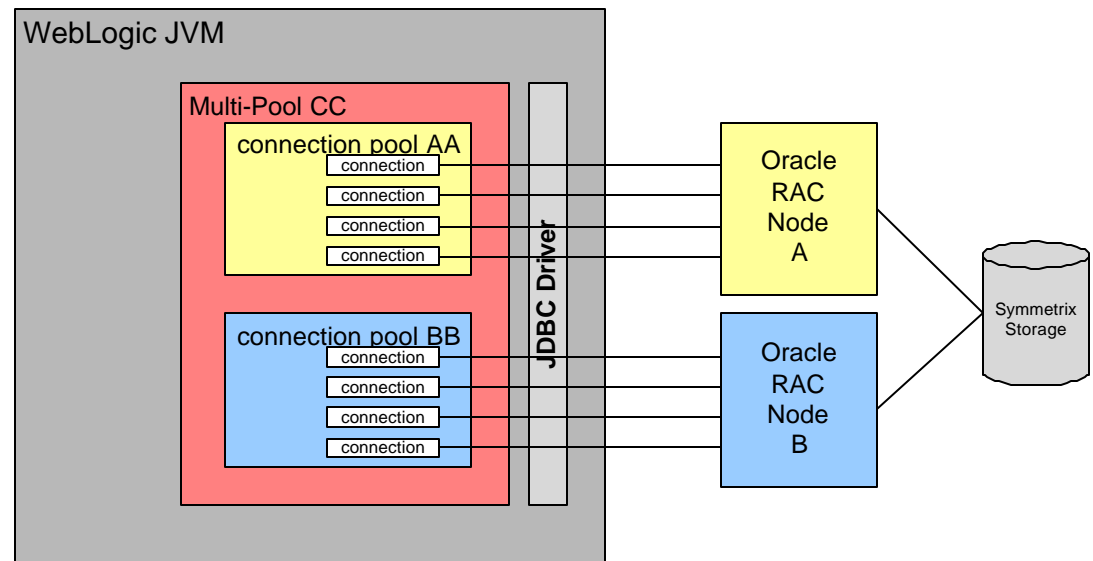
Testing JDBC MultiPools

First, setup a High Availability MultiPool

- a) Define pool AA to point to DB node A and pool BB to point to DB node B.
- b) Create multi-pool CC consisting of (poolAA,poolBB) in HA mode

Next, use KenTest.JSP to test the HA failover of your infra

- a) Configure JSP to use multi-pool instead of specific JDBC pool
- b) Run JSP repeatedly using a load test tool such as LoadRunner or The Grinder
- c) Shutdown Oracle on DB node A
- d) See if (and how long) it takes WebLogic to ensure multi-pool detects failure and starts routing SQL calls through pool BB to DB node B.



KenTest.jsp supports MultiPools. Instead of specifying &JDBC=AA or &JDBC=BB, you would specify &JDBC=CC