Version 2

Implementing load balancing for PAS for OpenEdge based on Tomcat Load Balancing

Valeriy G. Bashkatov
PROGRESS TECHNOLOGIES
2017/NOV

1 Table of Contents

2 Introduction

Load balancing allows you to control incoming traffic by controlling and distributing client requests across multiple network devices to optimize resource utilization and reduce maintenance time.

In the classic OpenEdge AppServer to implement load balancing used Progress Name Server Load Balancer product.

For new application server – Progress Application Server for OpenEdge (PAS for OpenEdge) – load balancing is implemented by using standard HTTP options based on one of the third-party technologies such as Apache HTTP, Apache Tomcat or Amazon Elastic Load Balancing.

In this article, I will talk about how to implement load balancing for PAS for OpenEdge using Apache Tomcat server (Apache Tomcat Load Balancing).

This article does not pretend to describe the completeness of this topic, but only aims to provide a starting point and to demonstrate the initial configuration of Apache Tomcat Load Balancing to spread the load between PAS for OpenEdge instances.

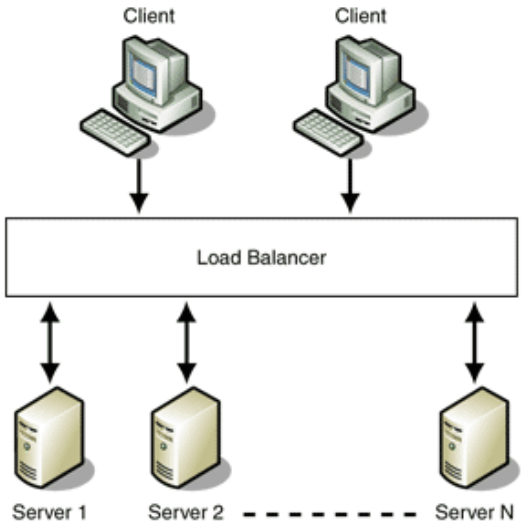
For any additional information on the configuring and work with Apache Tomcat Load Balancing refer to Internet resources, for example, to the official websites https://httpd.apache.org/ *and* http://tomcat.apache.org/*.*

3 Why we need load balancing?

For a typical application that is serviced by a small number of users and that does not have serious business requirements, a load balancing may not be needed.

But the situation is completely different with those applications that are key to business. For example, applications that handle billing operations. If a service of this kind "falls", then some business process also will cease to function.

In this case, we have a single point of failure. To eliminate it we should configure load balancer cluster for our application server.



As result, if one of the application servers fails or if we need to perform maintenance, the load balancer will forward all user requests to the second application server.

1. *Load balancing is a solution to the problem of a single point of failure of application server*.

Then, as we know, application servers may use a lot of memory or CPU resources, especially when many users work with it. The more users, the more server resources are needed to process their requests.

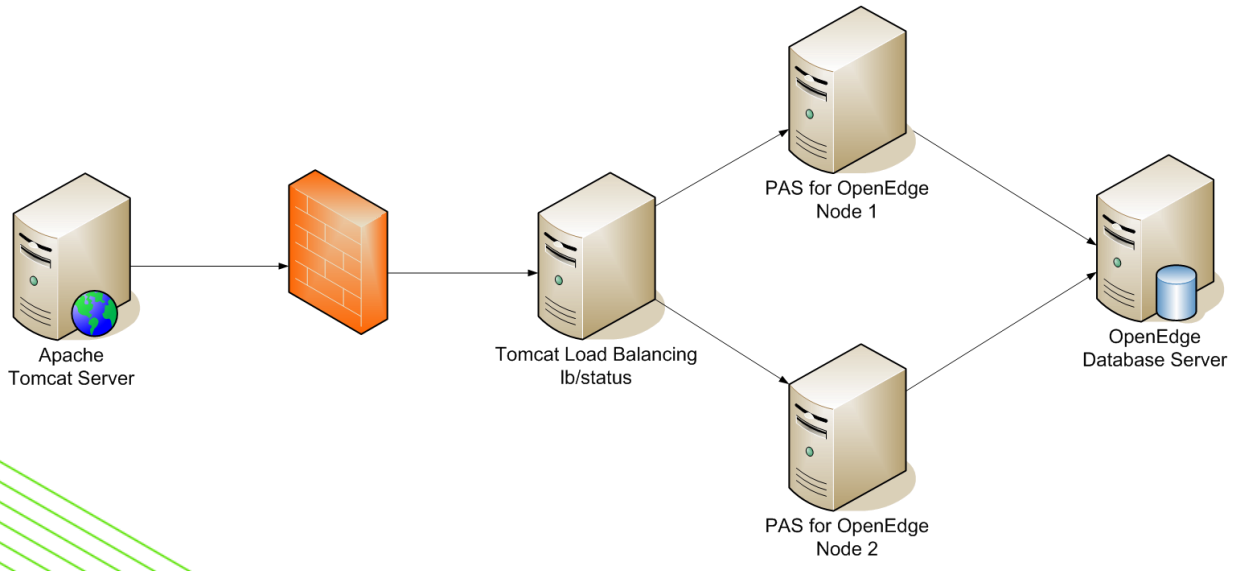
2. *Load balancing it is also scalability and optimization of resource use.*

4 Load Balancing for Apache Tomcat

Apache Tomcat server is built into PAS for OpenEdge. Therefore, to implement load balancing using the Apache Tomcat server, you must assign a virtual load balancer worker (lb), which will be used to redirect client requests from the Apache Web Server into the actual working PAS instances which will process that requests.

In addition, if you want to monitor the status of worker instances, you can add one more a virtual worker in the load balancing scheme and assign it as a metrics gatherer (status). The role of metrics gatherer involves control and inform about the status of the worker instances. But even if the metrics gatherer will not be set, the load balancer will detect when a worker instance to become unresponsive and stop sending requests to them.

The following figure shows the simple configuration of the load balancer, which we implement in this article.



In a real production environment, usually, Apache web server - it's a public server available from the Internet. Thus, Apache HTTP Server must be placed in the demilitarized zone (DMZ) in terms of safety.

As the servers for our example, we will use the five virtual machines with pre-installed from scratch CentOS 7.

The first server is an external Apache Tomcat server. There is nothing except for a web server. But I will not describe how this server is configured, since this is a topic for a separate article and I do not use it in this example.

On servers with instances of PAS for OpenEdge installed the following licenses:

- Progress Production Application Server for OpenEdge (11.7)

On the database server, the following licenses installed:

- Progress OpenEdge Advanced Enterprise RDBMS (11.7)
- Progress 4GL Development System (11.7)

Some of the advantages of Apache Tomcat Load Balancing in comparison with the implementation of load balancing based on the Apache proxy host load balancing:

- Virtual worker, which will perform load balancing (in the figure presented as **lb**), «knows» about the status of each application server instance in its group and will not send requests to an incapacitated instance;
- Tomcat Load Balancing allows you to configure one more a virtual worker as monitor of online statistics for each worker instance in the load balancing group (in the figure is presented as **status**).

The process of configuration the load balancing based on Tomcat consists of the following tasks:

1. Create a worker instances of PAS for OpenEdge on different servers.
2. Install or upgrade an existing local Apache HTTP Server.
3. Configure Apache HTTP Server.
4. Configure MOD_JK module in Apache configuration.
5. Creating and deploying `worker.properties` file.

5 Create and configure the worker instances

As worker instances, among which will be implemented load balancing, we will create two instances of the PAS for OpenEdge, each of which will run in its own virtual machine. For convenience, let's call these instances and, accordingly, their virtual machines as node1 and node2.

In this article, it is assumed that on the virtual machines with worker instances already installed OpenEdge 11.7 with the appropriate licenses (see previous section). For installation OpenEdge refer to the OpenEdge documentation, "OpenEdge Getting Started: Installation and Configuration".

5.1 Configuring of virtual machines

In our example, virtual machines used for the worker instances have the following IP-address: `172.16.95.148` and `172.16.95.149`. By default, both machines hostname is `localhost`. We need to change the hostname `localhost` for these virtual machines on the names `node1` and `node2` respectively.

To change the hostname on CentOS 7, use the following command (as root):

```
hostnamectl set-hostname New_HostName
```

172.16.95.148:

hostnamectl set-hostname node1

172.16.95.149:

hostnamectl set-hostname node2

To make all changes to take effect, we need to restart the service `systemd-hostnamed`:

```
systemctl restart systemd-hostnamed
```

Verify the host name:

```
hostnamectl status
```

You will see something like this:

Static hostname: **node2**
Icon name: computer-vm
Chassis: vm
Machine ID: a5003f6992044150941f05577bde3054
Boot ID: cc43313da082435e82a02c4046096d76
Virtualization: vmware
Operating System: CentOS Linux 7 (Core)
CPE OS Name: cpe:/o:centos:centos:7
Kernel: Linux 3.10.0-514.6.1.el7.x86_64
Architecture: x86-64

From this point, virtual machines for the worker instances will be called `node1` and `node2`.

5.2 Creating and configuring the worker instances

Worker instance of PAS for OpenEdge can be in any directory on the server. For our convenience, we will create them in `WRKDIR` directory, which is typically `/usr/wrk` directory.

5.2.1 Creating a node1 instance

1. Connect to the virtual machine `node1` as root.
2. Execute `proenv` command:

```
[root@node1 ~]# proenv
```

3. Create a worker instance named `node1`:

```
proenv> $DLC/servers/pasoe/bin/tcman.sh create
-p 8820 -P 8821 -s 8822 $WRKDIR/node1
```

Where:

-p 8820 – port number for http;
-P 8821 – port number for https;
-s 8822 – port number for shutdown;
$WRKDIR/node1 – the path and the name of the instance directory.

4. Activate AJP13.

Apache JServ Protocol (AJP) - a binary protocol that enables transfers incoming requests from the web server to the application server. It is generally used in a load-balanced systems. It also supports the monitoring of state of the server.

By default, in PAS for OpenEdge this protocol is disabled, so we must enable it:

a. Let's move to the `bin` directory of the `node1` instance:

```
cd $WRKDIR/node1/bin
```

b. Check the status of AJP13 protocol:

```
tcman.sh feature AJP13

    AJP13=off
```

c. To activate, use the following command:

```
tcman.sh feature AJP13=on
```

d. Check the status again:

```
tcman.sh feature AJP13

    AJP13=on
```

5. Next, set the AJP13 port number. By default, this number is 8009.

In case when in one server is supposed to work multiple instances of PAS for OpenEdge it is recommended for each instance establish a unique AJP13 port number.

a. For `node1` instance set the port number 50001 with the following command:

```
tcman.sh config psc.as.ajp13.port=50001
```

b. Verify that the port is properly established:

```
tcman.sh config psc.as.ajp13.port

    psc.as.ajp13.port=50001
```

c. Open port 50001 in the firewall:

```
firewall-cmd --permanent --add-port=50001/tcp
```

d. To access the web-based applications hosted on PAS for OpenEdge, we need to open one of the HTTP or HTTPS ports depending on the security requirements of the application. For node1 instance we pointed out as the HTTP port 8820 – open this port:

```
firewall-cmd --permanent --add-port=8820/tcp
```

e. Reload firewall to apply the changes:

```
firewall-cmd --reload
```

In the PAS for OpenEdge client access to applications through special protocols: APSV, REST, WEB и SOAP.

By default, in the production version of the PAS for OpenEdge for security reasons all communication protocols disabled (APSV, REST, WEB, SOAP). While in the versions of PAS for OpenEdge designed for the development these protocols are enabled by default.

In this article to demonstrate the work of load balancing, we will use APSV protocol designed to interact with the application server through the ABL(4GL). Accordingly, it is necessary to activate this protocol. To activate the protocol used `adapterEnable` property with a value of 1 in the instance configuration file `conf/openedge.properties`. To activate the protocol, we can edit the configuration file manually or by using a script `oeprop`.

To activate APSV protocol for node1 worker instance in the directory `bin` execute the following command:

```
oeprop.sh node1.ROOT.APSV.adapterEnabled=1
```

To check the status of the port, we can use the same command but without assignment.

Perform the start of the worker instance node1:

```
tcman.sh start
```

5.2.2 Creating a node2 instance

To create a working instance node2, we need to perform the same actions as when creating an instance node1. Because node2 we placed on a separate virtual machine, the ports for that instance we leaving the same as for node1. Differ is only in instance name.

1. Connect to the virtual machine node2 as root.
2. Execute proenv command.
3. Create a worker instance named node2:

```
proenv>$DLC/servers/pasoe/bin/tcman.sh create -p 8820
-P 8821 -s 8822 $WRKDIR/node2
```

4. Activate AJP13 and assign the port; open ports AJP13 and HTTP:

```
cd $WRKDIR/node2/bin

tcman.sh feature AJP13=on

tcman.sh config psc.as.ajp13.port=50001

firewall-cmd --permanent --add-port=50001/tcp

firewall-cmd --permanent --add-port=8820/tcp

firewall-cmd --reload
```

5. Activate APSV protocol:

```
oeprop.sh node2.ROOT.APSV.adapterEnabled=1
```

6. Start worker instance node2:

```
tcman.sh start
```

6 Installing and configuring Apache HTTP Server

Due to the simplicity and flexibility in managing, the Apache Web Server is one of the most popular and powerful web servers in the world. In this part, we will perform its installation and configuration for load balancing.

6.1 Installing Apache

Apache Web Server in our example will be on a separate virtual machine with the IP-address `172.16.95.146`.

In the first for ease of identification, we will change the hostname of the server to `apache`:

```
hostnamectl set-hostname apache

systemctl restart systemd-hostnamed
```

Now perform the installation (as root):

1. Clean up `yum`:

   ```
   yum clean all
   ```

2. Update all our packages:

   ```
   yum -y update
   ```

3. To install Apache, execute a single command is enough:

   ```
   yum -y intall httpd
   ```

4. Allow access to Apache through the firewall. To do this, open the standard ports 80 (HTTP) and 443 (HTTPS):

   ```
   firewall-cmd --permanent --add-port=80/tcp

   firewall-cmd --permanent --add-port=443/tcp
   ```

5. Reload firewall:

   ```
   firewall-cmd --reload
   ```

6. Configure Apache to start at boot:

```
systemctl enable httpd
```

7. Start Web Server:

```
systemctl start httpd
```

8. Check status:

```
systemctl status httpd
```

We should see this string:

```
Active: active (running)
```

9. Stop the web server, because at this stage it is not necessary for us:

```
systemctl stop httpd
```

6.1.1 Installation MOD_JK on the Apache HTTPD

What is **mod_jk**? mod_jk is a module of the Apache web server, which allows applications to interact through HTTPD Server with Apache Tomcat. In other words, mod_jk allows us to connect the Tomcat instance (in this case PAS for OpenEdge) with Apache HTTPD web server.

Mod_jk installation process is simple, but nevertheless it requires compilation.

First, make sure that we have installed all the necessary modules:

```
yum install httpd-devel apr apr-devel apr-util apr-util-devel gcc gcc-c++ make autoconf libtool
```

Then download the mod_jk from the official web-site (at the time of this writing this article, the current version number of the module was 1.2.42):

http://tomcat.apache.org/download-connectors.cgi

To download execute the following:

```
mkdir -p /opt/mod_jk/

cd /opt/mod_jk

wget http://www.eu.apache.org/dist/tomcat/tomcat-connectors/jk/tomcat-connectors-1.2.42-src.tar.gz

tar -xvzf tomcat-connectors-1.2.42-src.tar.gz

cd tomcat-connectors-1.2.42-src/native
```

Compile and install the module in `native/` directory:

```
./configure --with-apxs=/usr/bin/apxs --enable-api-compatibility

make

libtool --finish /usr/lib64/httpd/modules

make install
```

If everything goes with no errors, then in the directory `/etc/httpd/modules` should be appear `mod_jk.so` file:

```
ls -la /etc/httpd/modules/mod_jk.so

-rwxr-xr-x. 1 root root 1553160 Feb  9 18:00 /etc/httpd/modules/mod_jk.so
```

6.1.2 Configuring Apache for load balancing

Configuring Apache HTTP Web server for Tomcat load balancing includes:

- Enabling the AJP13 protocol for communicating with the load balancer.
- Referencing the load balancer.

To enable the AJP13 protocol, un-comment the following modules in the HTTP server's `httpd.conf` file:

```
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
```

To reference the load balancer, add the following definitions to the `httpd.conf` file:

```
#workers.properties load balancing config
LoadModule jk_module apache_install_dir/modules/mod_jk.so
JkWorkersFile apache_install_dir/conf/workers.properties
JkShmFile apache_install_dir/logs/mod_jk.shm
JkLogFile apache_install_dir/logs/mod_jk.log
JkLogLevel info
JkLogStampFormat "[%a %b %d %H:%M:%S %Y]"
JkMount /status status
JkMount /* lb
```

- **LoadModule** - this makes the mod_jk module available for use.
- **JkWorkersFile** - the path to the worker configuration file, which we will create in the next step.
- **JkShmFile** - the path to the shared memory file for the module.
- **JkLogFile** - the path to the module log file.
- **JkLogLevel** - the level of logging for the module. The valid values for this attribute are "debug", "error" or "info".
- **JkMount** - this is used to map a certain URL pattern to a specific worker configured in the workers configuration file. Here, we use it twice - once to enable “/status” as the access URL for a virtual monitoring worker, and once to map all requests we want to be handled by the cluster to the virtual worker that contains the load balancing capability (lb).

Note that the load-balancing instance must be named **`lb`**.

Refer to the Tomcat documentation for the latest information.

7 Creating and deploying the worker.properties file

The `worker.properties` file defines the load-balancing instance, the metrics-gathering instance, and the worker instances that execute client requests. To create and deploy the `worker.properties` file:

1. Create a preliminary `worker.properties` file by running the `tcman workers` command in the bin directory of each instance of PAS for OpenEdge. Since the instances ideally run on separate computers, you must run the command on each of those computers and create your preliminary `worker.properties` file for each instance.
2. Combine the `worker.properties` files that you have generated for all instances into a single `worker.properties` file and place it in the `conf` directory of the Apache HTTP server.
3. Comment out the property, `worker.common.host`, in the combined `worker.properties` file.
4. Add a `workers.`*`instance_name`*`.host=`*`host_name`* property to each instance's configuration.
5. Save and close the file.

Follow these steps for our example:

1. For instance, node1:
    a. Connect to a virtual machine `node1`.
    b. Open `proenv`.
    c. Change directory to `bin`:

```
cd node1/bin
```

    d. Execute command to generate `workers.properties` file:

```
tcman.sh workers
```

`workers.properties` file will be created in the `/usr/wrk/node1/temp` directory.

2. For instance, node2:
    a. Connect to a virtual machine `node2`.
    b. Open `proenv`.
    c. Change directory to `bin`:

```
cd node2/bin
```

    d. Execute command to generate `workers.properties` file:

```
tcman.sh workers
```

`workers.properties` file will be created in the `/usr/wrk/node2/temp` directory.

Merge `workers.properties` files by copy/paste and at the same time remove the duplicate rows. We should get a file with the following contents (to save space commented lines have been deleted from a file):

```
worker.list=node1,node1
worker.common.type=ajp13
worker.common.socket_timeout=10
worker.common.connect_timeout=10000
worker.common.socket_keepalive=true
worker.common.ping_mode=I
worker.common.ping_timeout=10000
worker.common.retry_interval=100
worker.common.recovery_options=7

worker.node1.port=50001
worker.node1.reference=worker.common
worker.node2.port=50001
worker.node2.reference=worker.common
```

Now correct the merged file.

First of all pay attention to `worker.list` line. Our configuration defines two virtual workers and two actual workers, which map to my Tomcat servers. Here I made only one change. The virtual workers “status” and “lb” I defined in the worker.list property, because I’m refer to them in my apache configuration.

```
worker.list=lb,status
```

Then if present, comment `worker.common.host` line.

The next section is a section for our actual worker instances. Here, I define workers for each of my servers, using the port values from the AJP connectors and hosts of that servers.

I've also included optional property for these workers, "lbfactor". The higher the number of this property, the more preference mod_jk will give that worker when load balancing. For example, if I had given the servers lbfactors of 1 and 3, I would find that the round-robin load balancing would prefer one server over the other with a 3 to 1 ratio.

```
worker.node1.port=50001
worker.node1.host=172.16.95.148
worker.node1.reference=worker.common
worker.node1.lbfactor=1

worker.node2.port=50001
worker.node2.host=172.16.95.149
worker.node2.reference=worker.common
worker.node2.lbfactor=1
```

Lastly, I've got a little configuration for my virtual workers. I've set the load balancer worker to have type "lb" and listed the workers which represent Tomcat in the "balance_workers" property. If I had any further servers to add, I would define them as a worker and list them in the same property.

```
worker.lb.type=lb
worker.lb.balance_workers=node1,node2
```

Load balancers use a variety of methods to make sure that requests are sent to the machine that has the most current session data. The easiest of these, and the one we will use for this example, is called "sticky sessions".

Sticky sessions are an important feature if you rely on jSessionIDs and are not using any session-replication layer. If **sticky_session** is **True** a request always gets routed back to the node which assigned this jSessionID.

If that host should get disconnected, the request will be forwarded to another host in our cluster, although not too successfully as the session-id is invalid in its context. You can prevent this by setting **sticky_session_force** to **True**. In this case if the host handling the requested session-id fails, Apache

will generate an internal server error 500. This is especially important for an application server that interacts with an OpenEdge database.

```
worker.lb.sticky_session=True
worker.lb.sticky_session_force=True
```

The latest version of mod_jk enables sticky sessions by default.

If **method** is set to **Request** the balancer will use the number of requests to find the best worker. Accesses will be distributed according to the lbfactor in a sliding time window. This is the default value and should be working well for most applications.

```
worker.lb.method=Request
```

The only configuration that the **status** worker needs in our example is to set the **type** to **status**.

```
worker.status.type=status
```

Here is an example of the final workers.properties file:

```
worker.list=lb,status
worker.common.type=ajp13
worker.common.socket_timeout=10
worker.common.connect_timeout=10000
worker.common.socket_keepalive=true
worker.common.ping_mode=I
worker.common.ping_timeout=10000
worker.common.retry_interval=100
worker.common.recovery_options=7
worker.lb.type=lb
worker.lb.balance_workers=node1,node2
worker.lb.sticky_session=True
worker.lb.sticky_session_force=True
worker.lb.method=Request
worker.status.type=status
worker.node1.port=50001
worker.node1.host= 172.16.95.148
worker.node1.reference=worker.common
worker.node1.lbfactor=1
worker.node2.port=50001
worker.node2.host= 172.16.95.149
worker.node2.reference=worker.common
worker.node2.lbfactor=1
```

Save the changes and put the final `workers.properties` file into directory `/etc/httpd/conf` on the Apache server.

Perform start of Apache HTTP Web Server:

```
systemctl start httpd
```

8 Verifying Load Balancing

To check the work of our configuration we use two test programs written on ABL.

Create a file named `testserver.p` and write the following code:

```
DEFINE OUTPUT PARAMETER cHostName AS CHARACTER NO-UNDO.
INPUT THROUGH 'hostname'.
REPEAT:
   IMPORT UNFORMATTED cHostname.
END.
INPUT CLOSE.
```

`testserver.p` program – it is backend, which will run on node1 and node2 application servers. Compile this file to get the r-code and place it in the openedge/ directory of instances node1 and node2.

The license Progress Production Application Server for OpenEdge allows to run only the compiled code.

Create a file named `testclient.p` and write the following code:

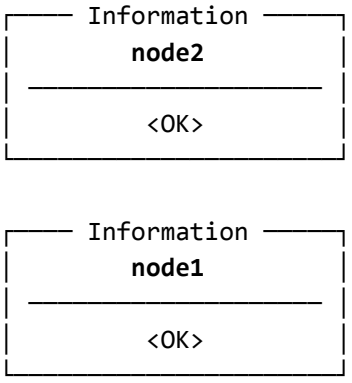
```
DEFINE VARIABLE AppHandle AS HANDLE.
DEFINE VARIABLE cHostName AS CHARACTER.
DEFINE VARIABLE ret       AS LOGICAL.

DEFINE VARIABLE pasHost   AS CHARACTER INIT "172.16.95.146" NO-UNDO.
DEFINE VARIABLE pasPort   AS CHARACTER INIT "80" NO-UNDO.

CREATE SERVER AppHandle.

   ret = AppHandle:CONNECT("-URL http://" + pasHost + ":" + pasPort +
"/apsv").
   IF ret THEN
       RUN testserver.p ON AppHandle(OUTPUT cHostName).

   MESSAGE cHostName VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.

   QUIT.
```

In this program the variable `pasHost` must specify the hostname or the IP-address of the Apache HTTP Web server, in our case it `172.16.95.146`. In the variable `pasPort` must specify the port number by which you access the web server, in our case it the standard port 80.

Run `testclient.p` several times:

```
mpro -p testclient.p
```

As a result, every time you should show a different hostname, depending on what instance of Tomcat load balancing connected:



If your implementation of load balancing does not work, first of all review the Apache HTTP Web Server logs (by default in `/var/log/httpd`), mod_jk logs (`mod_jk.log`, defined by `JkLogFile` parameter in `httpd.conf` file), and logs of each instance of PAS for OpenEdge, which are located in *`instance_name`*`/logs` directory.

8.1 Connect worker instances to the database

And in conclusion a little bit about the database server, which is represented in our configuration in the figure at the beginning of the article.

In order to our worker instances node1 and node2 can connect to the sports2000 database, we need to point out connection parameters in the configuration file *instance_name*/conf/openedge.properties.

This can be done manually by editing agentStartupParam parameter in section [AppServer.SessMgr]. This parameter uses a multisession manager at the start of the agents. The parameter can be changed online while an instance work, but this change affects only new agents that will start after changing the value.

Example of [AppServer.SessMgr] section:

```
[AppServer.SessMgr]
agentStartupParam=-db sports2000 -H 172.16.95.143 -S 20333
```

For change the value of the agentStartupParam parameter can be using oeprop script. To do this in the bin directory of an instance, we need to execute the following command:

```
oeprop.sh AppServer.SessMgr.agentStartupParam=db_connection_param
```

Example of command for our worker instances:

```
  oeprop.sh AppServer.SessMgr.agentStartupParam="-db sports2000
-H 172.16.95.143 -S 20333"
```

To test the connection, change testserver.p code as follows:

```
DEFINE OUTPUT PARAMETER vCustName AS CHARACTER NO-UNDO.

FOR FIRST Customer NO-LOCK.
  vCustName = Customer.Name + ' on PAS '
  + SUBSTRING(SESSION:SERVER-CONNECTION-ID,INDEX(SESSION:SERVER-CONNECTION-
ID,".") + 1).
END.
```

Then, change `testclient.p` as follows:

```
DEFINE VARIABLE AppHandle AS HANDLE.
DEFINE VARIABLE ret       AS LOGICAL.

DEFINE VARIABLE pasHost   AS CHARACTER INIT "172.16.95.146" NO-UNDO.
DEFINE VARIABLE pasPort   AS CHARACTER INIT "80" NO-UNDO.

DEFINE VARIABLE i         AS INTEGER   INIT 0 NO-UNDO.
DEFINE VARIABLE j         AS INTEGER   INIT 10 NO-UNDO.
DEFINE VARIABLE cPASMes   AS CHARACTER.

DEFINE TEMP-TABLE tTab NO-UNDO
    FIELD vMess AS CHARACTER FORMAT "X(30)".

CREATE SERVER AppHandle.

REPEAT i=1 TO j:

    ret = AppHandle:CONNECT("-URL http://" + pasHost + ":" + pasPort +
"/apsv").
    IF ret THEN
    DO:
        RUN testserver.p ON AppHandle(OUTPUT cPASMes).
        CREATE tTab.
        tTab.vMess = cPASMes.
        AppHandle:DISCONNECT().
    END.
    ELSE
        LEAVE.
END.

FOR EACH tTab NO-LOCK:
    DISPLAY tTab.
END.
QUIT.
```

Compile the programs and copy the r-code into the catalog *instance_name*/openedge. Now, as a result of running testclient.p program, on the screen will display a list of messages that are generated in turn depending on which application server the load balancer sent the request to.

```
Lift Tours on PAS node1
Lift Tours on PAS node2
…
…
Lift Tours on PAS node1
Lift Tours on PAS node2
```

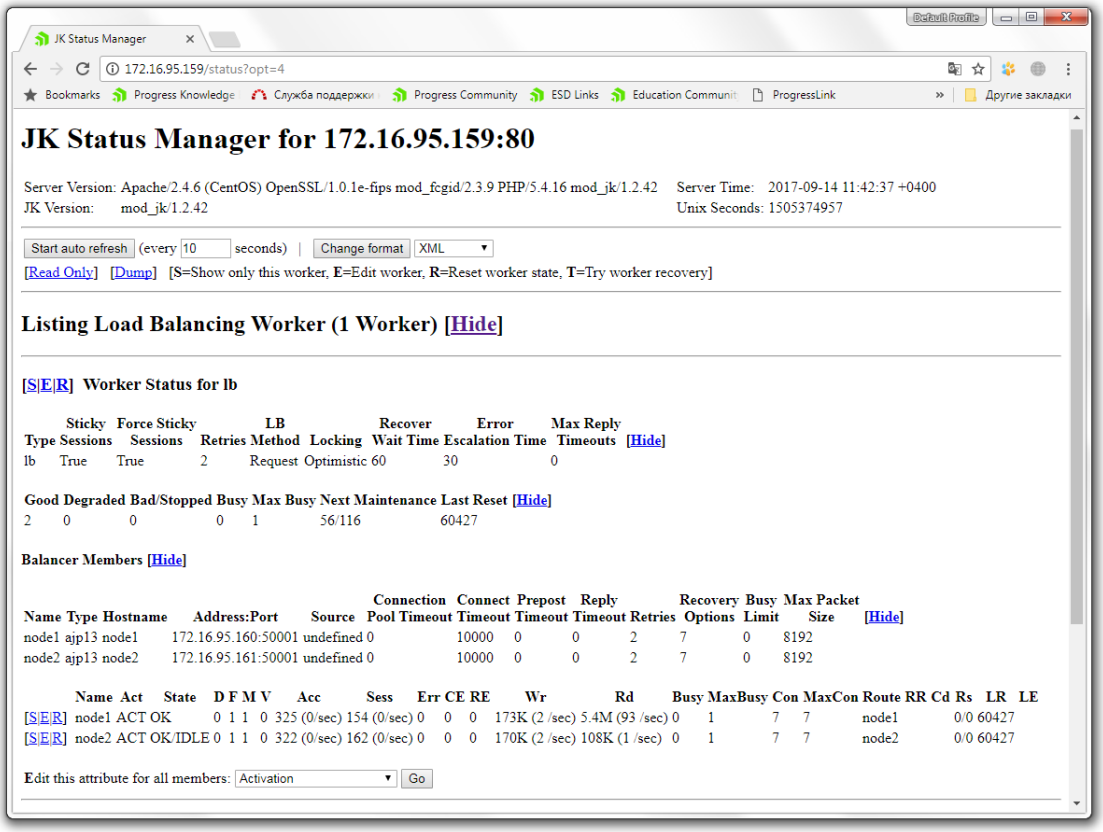
At this point I finished and I hope that this article will be useful as a start point in learning and configuring load balancing for the new application server – Progress Application Server for OpenEdge.

9 Additional materials

Monitoring load balancing

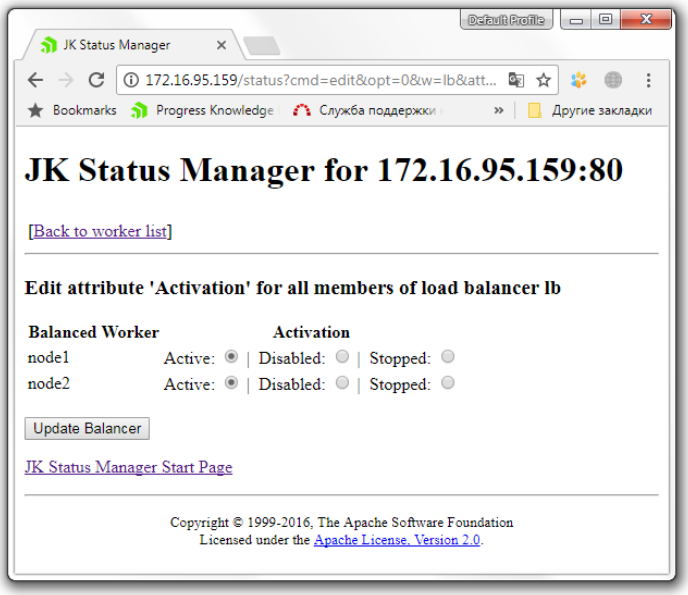
To monitor the load balancer, we have several options. The first of them is JK Status Manager, which available for us through web browser.

http://*<lb-hostname>*/status

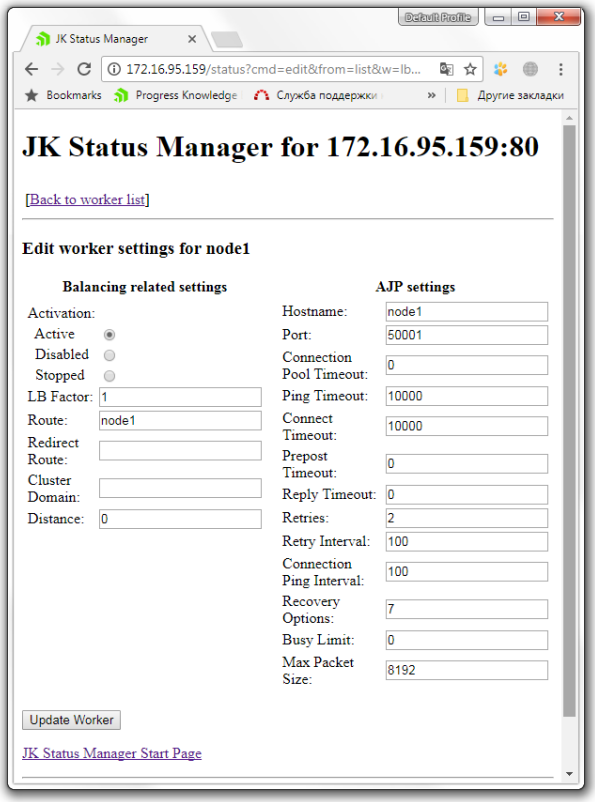


With JkStatusManager the tomcat cluster worker can temporarily disabled for maintenance reasons e.g. software installations, updates or application reconfiguration. To disable a tomcat instance in a cluster set the worker status to 'disabled'. Before doing some maintenance, be sure that there are no active sessions remains on this tomcat worker.

The value 'disabled' means that no new further sessions will be created by the load balancer on this tomcat worker. If all sessions of the worker are finished or timed out the worker is cluster released and can be configured.



In addition, we can manage our worker instances by change some of their properties.



Other options it is log files of Apache, MOD_JK and PAS for OpenEdge instances. Thus, if your balancer does not work, you should look at the log files and JK Status Manager.

- Log files of Apache and Mod_JK
  - /etc/httpd/logs/

    access_log
    error_log
    mod_jk.log

- PAS for OpenEdge logs
  - <PAS-instance-name>/logs/

    <PAS-instance-name>.agent.log

How to add new PAS instance by copy existing

- Stop existing instance, for example node2
  - Disable and stop worker instance in the JK Status Manager
  - Stop PAS instance by “tcman.sh stop”
  - Clean logs by “tcman.sh clean”
- Copy instance by OS command

  cp -R $WRKDIR/node2 $WRKDIR/node3
- Start existing instance (node2) and enable them for balancing in the JK Status Monitor
- Register new instance

  $DLC/servers/pasoe/bin/tcman.sh register node3 $WRKDIR/node3

- Change ports for the new instance (node3)

  tcman config psc.as.http.port=<new http port>
  tcman config psc.as.https.port=<new https port>
  tcman config psc.as.ajp13.port=<new ajp13 port>
  tcman config psc.as.shut.port=<new shutdown port>

- Allows AJP13 port of new instance in the SELinux

  semanage port -a -t http_port_t -p tcp <AJP13 port>

- Add new worker instance into worker.properties

  cd /etc/httpd/conf
  vim worker.properties
  worker.node3.port=<AJP13 port>
  worker.node3.host= <worker host or IP>
  worker.node3.reference=worker.common
  worker.node3.lbfactor=1
  worker.lb.balance_workers=node1,node2,node3

- Restart Apache server

  systemctl restart httpd

SELinux

Because of **SELinux** policy, a service is normally allowed to run on a restricted list of well-known ports. For example, in the case of the **httpd** service, this list is **80**, **443**, **488**, **8008**, **8009**, **8443**. To allow a service to use non-standard ports, you need to follow a specific procedure to change the **SELinux** policy. This is important, if we want to add more than one instance of Tomcat (PASOE) to the same server with separate AJP13 ports to configure load balancing.

First install the **setroubleshoot-server** (to get the **semanage** command) if not installed yet:

yum install -y setroubleshoot-server

Check the status of SELinux:

```
getenforce
```

To get the list of all restricted ports by service, type:

```
semanage port -l
```

To get the list of well-known ports for the **httpd** service, type:

```
semanage port -l | grep -w http_port_t
```

To allow the **httpd** service to run on the **50001 tcp** port (**-a** for add), type:

```
semanage port -a -t http_port_t -p tcp 50001
```

Your Notes: