Lab 4. Integration of Tomcat with the Apache Web Server

The Apache HTTP server is one of the most widely used frontend web servers across the IT industry. This project was open sourced in 1995 and is owned by The Apache Software Foundation.

In this lab, we will discuss the following topics:

- The Apache HTTP installation
- Integration of Apache with Tomcat 8

Why the Apache HTTP server

The Apache HTTP server is one of the most successful and common web servers used in IT industries. The reason being that it is supported by open source communities. In IT industries, the Apache HTTP server is heavily used as a frontend web server for the following reasons:

- Efficiently serves static content
- Increase the speed by 10 percent
- Clustering
- Security
- Multiple website hosting
- Modules
- Decorator

Note: We can create Redirect and Rewrites in application code also. These rules are in the form of servlet classes.

Installation of the Apache Server

We can directly install the Apache package in Debain Linux (Ubuntu), using the `apt-get` command. The following command shows the syntax for the installation:

```
sudo apt-get install -y apache2
```

It is necessary to start the services of HTTP to verify the instance is properly installed. The best way to check the configuration is by running the configtest script. This script comes by default with Apache httpd, only in a non-DOS environment. The script can be found in APACHE_HOME/bin.

```
[root@localhost]# apachectl configtest
Syntax OK
```

- Then restart Apache using the following command:

```
[root@root@localhost]# apachectl start
```

Note: You will get and error because port 80 is already in use. We will change the port in the next step.

Change Apache Port

Let's change the default Apache port. Open `/etc/apache2/ports.conf` in vscode and update `80` with port `81`. Save and close the file.

How to configure the virtual host

Now we need to configure our virtual host to listen to the new port. Open the virtual host file using vscode: `/etc/apache2/sites-enabled/000-default.conf`

At the top of the file, you'll see the beginning of the directive:

`<VirtualHost *:80>`

Change that line to:

`<VirtualHost *:81>`

Restart Apache with the command:

`service apache2 restart`

Once you start Apache, it's very important to verify the instance status. You can verify the system using the `ps` command:

```
ps -ef |grep apache2
```

You should now be able to point a browser to http://localhost:81 (Where localhost is the IP address of the hosting server) to see the Apache welcome site or the welcome page of your virtual host.

```
[root@localhost]# apachectl stop
[root@root@localhost]# apachectl start
```

Then, run the `configtest` command on the server to verify the configuration using the following command:

```
[root@root@localhost]# apachectl configtest
Syntax OK
```

Apache as a Reverse Proxy with mod_proxy

Introduction

A *reverse proxy* is a type of proxy server that takes HTTP(S) requests and transparently distributes them to one or more backend servers. Reverse proxies are useful because many modern web applications process incoming HTTP requests using backend application servers which aren't meant to be accessed by users directly and often only support rudimentary HTTP features.

You can use a reverse proxy to prevent these underlying application servers from being directly accessed. They can also be used to distribute the load from incoming requests to several different application servers, increasing performance at scale and providing fail-safeness. They can fill in the gaps with features the application servers don't offer, such as caching, compression, or SSL encryption too.

In this lab, you'll set up Apache as a basic reverse proxy using the `mod_proxy` extension to redirect incoming connections to one or several backend servers running on the same network. This lab uses a sample application that was deployed using tomcat in previous lab.

Steps:

- Step 1 — Enabling Necessary Apache Modules
- Step 2 — Verify Sample Application Status
- Step 3 — Modifying the Default Configuration to Enable Reverse Proxy

Step 1 — Enabling Necessary Apache Modules

Apache has many modules bundled with it that are available but not enabled in a fresh installation. First, we'll need to enable the ones we'll use in this tutorial.

The modules we need are `mod_proxy` itself and several of its add-on modules, which extend its functionality to support different network protocols. Specifically, we will use:

- `mod_proxy`, the main proxy module Apache module for redirecting connections; it allows Apache to act as a gateway to the underlying application servers.
- `mod_proxy_http`, which adds support for proxying HTTP connections.
- `mod_proxy_balancer` and `mod_lbmethod_byrequests`, which add load balancing features for multiple backend servers.

To enable these four modules, execute the following commands in succession.

```
sudo a2enmod proxy
sudo a2enmod proxy_http
sudo a2enmod proxy_balancer
sudo a2enmod lbmethod_byrequests
```

To put these changes into effect, restart Apache.

```
service apache2 restart
```

Apache is now ready to act as a reverse proxy for HTTP requests. In the next (optional) step, we will create two very basic backend servers. These will help us verify if the configuration works properly, but if you already have your own backend application(s), you can skip to Step 3.

Step 2 — Verify Sample Application Status

Verify sample tomcat application is running that we deployed in last lab using `curl`. Test the server:

```
curl http://127.0.0.1:8080/
```

In the next step, we'll modify Apache's configuration file to enable its use as a reverse proxy.

Step 3 — Modifying the Default Configuration to Enable Reverse Proxy

In this section, we will set up the default Apache virtual host to serve as a reverse proxy for single backend server or an array of load balanced backend servers.

Open the default Apache configuration file using `vscode` or your favorite text editor.

```
/etc/apache2/sites-available/000-default.conf
```

Inside that file, you will find the `<VirtualHost *:80>` block starting on the first line. The example below explains how to configure this block to reverse proxy for a backend server

Reverse Proxying a Backend Server

Replace all the contents within `VirtualHost` block with the following, so your configuration file looks like this:

/etc/apache2/sites-available/000-default.conf

```
<VirtualHost *:81>
    ProxyPreserveHost On

    ProxyPass / http://127.0.0.1:8080/
    ProxyPassReverse / http://127.0.0.1:8080/
</VirtualHost>
```

There are three directives here:

- `ProxyPreserveHost` makes Apache pass the original `Host` header to the backend server. This is useful, as it makes the backend server aware of the address used to access the application.
- `ProxyPass` is the main proxy configuration directive. In this case, it specifies that everything under the root URL (`/`) should be mapped to the backend server at the given address. For example, if Apache gets a request for `/sample`, it will connect to `http://your_backend_server/sample` and return the response to the original client.
- `ProxyPassReverse` should have the same configuration as `ProxyPass`. It tells Apache to modify the response headers from backend server. This makes sure that if the backend server returns a location redirect header, the client's browser will be redirected to the proxy address and not the backend server address, which would not work as intended.

To put these changes into effect, restart Apache.

```
service apache2 restart
```

Now, if you access `http://your_server_ip` in a web browser, you will see your backend server response instead of standard Apache welcome page.

Summary

You now know how to set up Apache as a reverse proxy to one or many underlying application servers. `mod_proxy` can be used effectively to configure reverse proxy to application servers written in a vast array of languages and technologies, such as Python and Django or Ruby and Ruby on Rails. It can be also used to balance traffic between multiple backend servers for sites with lots of traffic or to provide high availability through multiple servers.

While `mod_proxy` with `mod_proxy_http` is the perhaps most commonly used combination of modules, there are several others that support different network protocols. We didn't use them here, but some other popular modules include:

- `mod_proxy_ftp` for FTP.
- `mod_proxy_connect` for SSL tunneling.
- `mod_proxy_ajp` for AJP (Apache JServ Protocol), like Tomcat-based backends.
- `mod_proxy_wstunnel` for web sockets.