Latest WildFly Documentation

Using Wildfly as a Load Balancer

Wildfly 10 adds support for using the Undertow subsystem as a load balancer. Wildfly supports two different approaches, you can either define a static load balancer, and specify the back end hosts in your configuration, or use it as a mod_cluster frontend, and use mod_cluster to dynamically update the hosts.

General Overview

Wildly uses Undertow's proxy capabilities to act as a load balancer. Undertow will connect to the back end servers using its built in client, and proxies requests.

The following protocols are supported:

- http
- ajp
- http2
- h2c (clear text HTTP2)
  Of these protocols h2c should give the best performance, if the back end servers support it.

The Undertow proxy uses async IO, the only threads that are involved in the request is the IO thread that is responsible for the connection. The connection to the back end server is made from the same thread, which removes the need for any thread safety constructs.
If both the front and back end servers support server push, and HTTP2 is in use then the proxy also supports pushing responses to the client. In cases where proxy and backend are capable of server push, but the client does not support it the server will send a X-Disable-Push header to let the backend know that it should not attempt to push for this request.

Load balancer server profiles

WildFly 11 adds load balancer profiles for both standalone and domain modes.

Example: Start standalone load balancer

```
# configure correct path to WildFly installation
WILDFLY_HOME=/path/to/wildfly
```

```
# configure correct IP of the node
MY_IP=192.168.1.1

# run the load balancer profile
$WILDFLY_HOME/bin/standalone.sh -b $MY_IP -bprivate $MY_IP -c standalone-load-balancer.xml
```

It's highly recommended to use private/internal network for communication between load balancer and nodes. To do this set the correct IP address to the `private` interface (`-bprivate` argument).

Example: Start worker node

Run the server with the HA (or Full HA) profile, which has `mod_cluster` component included. If the UDP multicast is working in your environment, the workers should work out of the box without any change. If it's not the case, then configure the IP address of the load-balancer statically.

```
# configure correct path to WildFly installation
WILDFLY_HOME=/path/to/wildfly
# configure correct IP of the node
MY_IP=192.168.1.2

# Configure static load balancer IP address.
# This is necessary when UDP multicast doesn't work in your environment.
LOAD_BALANCER_IP=192.168.1.1
$WILDFLY_HOME/bin/jboss-cli.sh <<EOT
embed-server -c=standalone-ha.xml
/subsystem=modcluster/mod-cluster-config=configuration:write-attribute(name=advertise,value=false)
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=proxy1:add(host=$LOAD_BALANCER_IP,port=8090)
/subsystem=modcluster/mod-cluster-config=configuration:list-add(name=proxies,value=proxy1)
EOT

# start the woker node with HA profile
$WILDFLY_HOME/bin/standalone.sh -c standalone-ha.xml -b $MY_IP -bprivate $MY_IP
```

Again, to make it safe, users should configure private/internal IP address into `MY_IP` variable.

Using Wildly as a static load balancer

To use WildFly as a static load balancer the first step is to create a proxy handler in the Undertow subsystem. For the purposes of this example we are going to assume that our load balancer is going to load balance between two servers, sv1.foo.com and sv2.foo.com, and will be using the AJP protocol.

The first step is to add a reverse proxy handler to the Undertow subsystem:

```
/subsystem=undertow/configuration=handler/reverse-proxy=my-handler:add()
```

Then we need to define outbound socket bindings for remote hosts

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=remote-host1/:add(host=sv1.foo.com, port=8009)
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=remote-host2/:add(host=sv2.foo.com, port=8009)
```

and than we add them as hosts to reverse proxy handler

```
/subsystem=undertow/configuration=handler/reverse-proxy=my-handler/host=host1:add(outbound-socket-binding=remote-host1, scheme=ajp, instance-id=myrc
/subsystem=undertow/configuration=handler/reverse-proxy=my-handler/host=host2:add(outbound-socket-binding=remote-host2, scheme=ajp, instance-id=myrou
```

Now we need to actually add the reverse proxy to a location. I am going to assume we are serving the path `/app`:

```
/subsystem=undertow/server=default-server/host=default-host/location=\/app:add(handler=my-handler)
```

This is all there is to it. If you point your browser to `http://localhost:8080/app` you should be able to see the proxied content.

The full details of all configuration options available can be found in the **subsystem reference (https://wildscribe.github.io/Wildfly/10.0.0.Final/%2Fsubsystem%2Fundertow%2Findex.html)**.