Load Balancer Configuration

Goals

After completing this section, students should be able to do the following:

• Configure Undertow as a load balancer.

JEE Load Balancing Applications

Configuring Infinispan and JGroups to replicate HTTP sessions is only half of the EAP 7 web clustering equation. The other half is configuring a network load balancer to direct HTTP requests to member EAP server instances cluster.

Other clustered EAP 7 services, such as remote JMS and EJB clients, do NOT require Network Load Balancers. They use client software capable of handling load balancing and switching. Its client software also handles the detection of existing cluster members and the identification of other cluster topology changes.

Unfortunately, the HTTP protocol does not provide the required load balancing and switching features. Adding this intelligence to the client means creating a new web browser, which is not practical. The solution is developed through a standard HTTP protocol feature: proxy support.

A web proxy is a web server that receives HTTP requests and forwards them to another web server. It can be used to provide security, caching, and other functions. For most web clients, the proxy hides the actual web servers that serve the application content.

Have a front-end web server (or web proxy) between the users and the application server JEE is a common pattern for several reasons:

• Only the front-end web server should be accessible from the Internet, and the servers of the application must not be exposed to direct attacks.

• Front-end web server can serve static HTML pages, images, CSS, JavaScript and other files, taking some work away from the application server.

• The front-end web server can perform SSL termination and processing, so application servers have more CPU cycles available to handle application logic.

• A front-end web server can hide different types of application servers and present them as a consistent set of URLs, for example, to make a PHP application and a Java EE application appear to be part of the same website.

Using a web proxy to perform load balancing between a number of other back-end web servers allows for efficient deployments, as the web proxy understands the HTTP protocol and can implement, for example, session affinity in an optimized manner.

Session affinity, also known as sticky sessions, is a web proxy mechanism that sends all requests from the same user to the same back end web server. Not having session affinity means that each request from a user can go to a different back-end web server.

Generic network load balancers (working at layers 3 or 4 of the OSI model) can be used for a Java EE application cluster, although they probably won't be as efficient as a web proxy (working at layer 7 of the model). OR IF). Some types of network hardware incorporate a web proxy and can be configured to function as Layer 7 load balancers as well as software-based load balancers.

The reason a Layer 7 load balancer is preferred over a Layer 3 or 4 load balancer for a web application is that Java EE web containers typically implement an HTTP cookie to store a unique session ID. Only load balancers that understand HTTP cookies can balance user sessions with maximum granularity while maintaining session affinity. This is true for most web application runtimes, not just Java EE.

To understand the need for and benefits of session affinity, consider the impact on stateful and stateless applications, and whether they are clustered or not:

- Stateless web applications should work correctly without session affinity, since there is no user data to persist in an HTTP session object. Any back-end server can process any request and return the expected results.
- Having a clustered web container does not make a difference in stateless web applications, since there is no user session data to replicate. They are typically deployed as non-clustered web applications on a non-clustered back-end farm under a load balancer for scalability.
- Stateful web applications can work in a non-cached environment clustered only if the load balancer provides session affinity, since only one back-end web server has data for a specific user session. Having a load balancer on non-clustered web applications offers scalability benefits even if user-transparent failover is not provided: if a back-end server fails, all user sessions on the failed server are lost, but users they can start over on a different server.
- Stateful web applications in a clustered stored environment have scalability advantages and provide transparent switching for all users. They should also work correctly without session affinity, since all back-end servers have access to all user session data, so any back-end server could respond to any request and deliver the expected results.

The discussion above shows that session affinity is an optional feature for many applications, but does not consider performance. All types of web applications generally perform faster and require less hardware resources with session affinity enabled, due to the locality of the data.

Even if an application works as it should without session affinity, a sequence of requests made by the same user is usually the same data set.
This data is cached by many layers of the web server's CPU back-

end to a possibly shared DB instance. Having the same back-end web server perform all the streaming maximizes the use of these caches.

Undertow Load Balancer Architecture and mod_cluster Undertow is a full-featured,

generic web server

that is capable of replacing traditional web server software, such as Apache Httpd and Microsoft IIS, for most use cases. As such, it includes a web proxy component that can perform load balancing.

Undertow is also a web server built specifically to support Java EE developers and administrators, and offers most of the features of Java web containers, such as Apache Tomcat. Among these, two are of particular importance for EAP cluster storage:

- AJP protocol support: The AJP protocol is a binary replacement for the text-based HTTP protocol. It also employs long-lived persistent connections, whereas HTTP connections are single-request or ephemeral (when using the HTTP 1.1 hot-connection feature). AJP was designed to reduce the overhead imposed by the front end web server on users accessing the Java web container. The idea is simple: a web server uses HTTP to connect to the web proxy, and the web proxy uses AJP to connect to the back-end application servers.
- mod_cluster protocol support: The mod_cluster protocol allows a web proxy to dynamically discover back-end web servers and the applications available to each, enabling a true dynamic environment. It also uses an additional HTTP connection to relay load metrics from each back-end application server to the web proxy, so you can make better load-balancing decisions.

Traditional web load balancer software requires static configuration: every detail of the back-end web connection needs to be manually configured on the load balancer, and it keeps trying to connect to failed back-end web servers, resulting in increased network overhead. . Adding more back-end servers involves reconfiguring and possibly restarting the load balancer. The following figure illustrates a traditional web proxy acting as a load balancer:

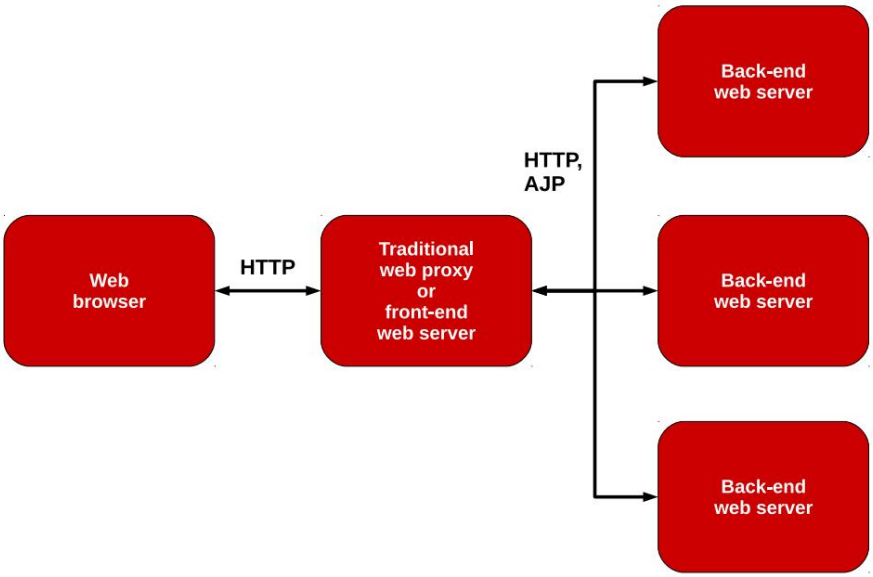

Figure 12.8:
Traditional web proxy as a static load balancer.

For this analysis, a Java EE application server is considered a special case of a back-end web server.

mod_cluster builds the back-end web server list dynamically, reporting new cluster members, newly deployed applications, or failed cluster members without manual configuration.

The mod_cluster protocol requires a client component, which must be implemented by the load balancer, and a server part, which is implemented using the EAP 7 modcluster subsystem. The following figure illustrates an enhanced mod_cluster web proxy that acts as a load balancer. burden:

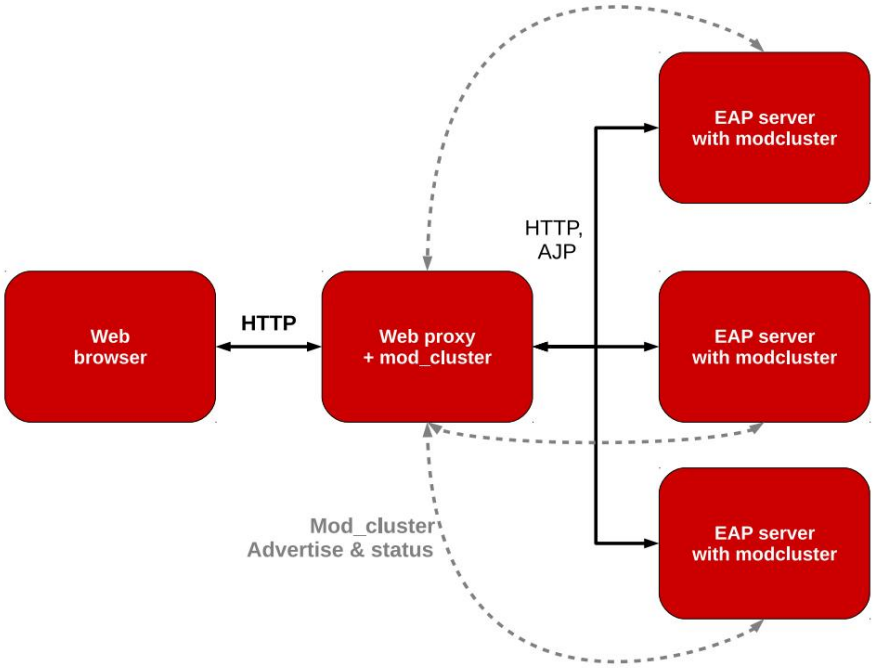

Figure 12.9:
Enhanced web proxy with mod_cluster as dynamic load balancer.

In the figure above, the web proxy + mod_cluster box can be an EAP 7 server instance with Undertow configured to enable mod_cluster or a native web server with a mod_cluster plugin. The EAP server with modcluster box is a back-end web server.

A mod_cluster enhanced web proxy such as Undertow sends publish messages to all back-end web servers listening on the configured multicast address and port. The back-end web servers respond by sending the load balancer their connection parameters and the context paths of deployed applications.

The architecture is fault tolerant: there can be multiple mod_cluster clients on the same network; that is, multiple web proxies acting as load balancers. The back-end web servers receive the publish messages from all the web proxies and the responses to them. The load balancer is NOT a single point of failure.

For networks where multicast traffic is not allowed, publishing is disabled on the mod_cluster client. Each back-end web server is then manually configured with a list of web proxies. All proxies in the list of web servers are sent connection parameters and updates about which applications are deployed and which are uninstalled. Even without multicast, a mod_cluster load balancer does not require static configuration.

Undertow can be configured to act as a static (without mod_cluster) or dynamic (with mod_cluster) load balancer. In either case, it is typically configured as a dedicated EAP server instance, where no application is deployed. This dedicated server instance is NOT a cluster member. A server pool consisting only of multiple dedicated load balancer EAP instances can be used to avoid having a single point of failure.

Configure Undertow as a dynamic load balancer

Configuring Undertow as a dynamic load balancer involves the following high-level steps:

- Add a mod_cluster filter to the Undertow default server.
- Configure the publishing settings in both the undertow and modcluster subsystems.
  - ÿ Either configures multifunction parameters on both subsystems, or disable publishing in the undertow subsystem and configure a proxy list in the modcluster subsystem.

These steps are detailed below using EAP CLI commands as examples.

Add mod_cluster filter to Undertow The default configuration of the undertow subsystem does NOT include a mod_cluster filter, even in the clustered EAP ha and full-ha profiles. This filter must be created and configured to use the correct multicast parameters referencing a socket-binding.

To add the mod_cluster filter and configure it with default EAP values, use the following command:

```
/profile=ha/subsystem=undertow/configuration=filter/mod_cluster=lb:add(\ management-socket-binding=http, advertise-socket-binding=modcluster)
```

The two attributes required by a mod_cluster filter are:

- management-socket-binding: tells Undertow where to receive socket information connection and load balancing metrics of back-end web servers. It must point to the socket-binding in which EAP receives HTTP requests, which is http by default.
- advertise-socket-binding tells Undertow where to send advertising messages publication, that is, the UDP multicast address and port, referencing a socket-binding name.

After creating the filter, it should be enabled on the desired Undertow (virtual) hosts:

```
/profile=ha/subsystem=undertow/server=default-server/host=default-host/filter-ref=lb:add
```

Notice that lb is the name assigned to the mod_cluster filter defined in the previous command.

Configuring Publishing Using Multicast The socket binding groups ha-sockets and full-ha-sockets already define the modcluster socket binding, which uses the multicast address 224.0.1.105 and port 23364. The undertow subsystem uses this socket binding. to know where to send post messages. The undertow subsystem uses the same socket binding to know where to listen for post messages.

It is recommended to change the multicast address to prevent unwanted EAP instances from trying to become load balancers for the clustered EAP server instances.

It is also recommended to configure a publish key shared by the mod_cluster client and the server. To configure the publish key on the application server instances, that is, on the cluster members:

```
/profile=ha/subsystem=modcluster/mod_cluster-config=configuration:\ write-
attribute(name=advertise-security-key,value=secret)
```

To configure the key on the load balancer server instance:

```
/profile=ha/subsystem=undertow/configuration=filter/mod_cluster=lb:\ write-
attribute(name=security-key,value=secret)
```

Note that the above commands affect different EAP server instances: the first, in the modcluster subsystem, affects EAP instances that are members of a cluster. The second, in the undertow subsystem, affects the EAP instance that acts as a load balancer.

Disable Publishing and Configure a Proxy List An
Undertow dynamic load balancer can be configured to NOT use multicast by disabling publishing in the mod_cluster filter. This is done by setting the advertise-frequency attribute to zero and the advertise-socket-binding to null.

```
/profile=ha/subsystem=undertow/configuration=filter/mod_cluster=lb:\ write-
attribute(name=advertise-frequency,value=0)
```

```
/profile=ha/subsystem=undertow/configuration=filter/mod_cluster=lb:\ write-
attribute(name=advertise-socket-binding,value=null)
```

In the example commands above, lb is the name that was assigned to the mod_cluster filter.

Advertise/Publication must also be disabled in the cluster members' modcluster subsystem so that they stop listening for advertisement messages:

```
/profile=ha/subsystem=modcluster/mod_cluster-config=configuration:\ write-
attribute(name=advertise,value=false)
```

Cluster members now require a list of proxies to know which mod_cluster load balancer to send connection parameters and application state to. Each load balancer instance must be configured as outbound socket binding. Assume a single load balancer instance:

```
/socket-binding-group=ha-sockets/remote-destination-outbound-socket-binding=lb:\ add(host=10.1.2.3,
port=8080)
```

The port on the outbound socket binding is the HTTP port of the EAP server instance of the load balancer.

These socket bindings are then used to configure the proxy list in the modcluster subsystem:

```
/profile=ha/subsystem=modcluster/mod_cluster-config=configuration:\ write-
attribute(name=proxies,value=[lb])
```

In the example above, lb is the name assigned to the outgoing socket binding.

Configure Undertow as a static load balancer

Configuring Undertow as a static load balancer involves the following high-level steps:

- **Add outgoing socket bindings that point to each member of the cluster.**
- **Add a reverse-proxy handler to the default Undertow server.**
- **Add each cluster member to the proxy handler.**

These steps are detailed below with sample EAP CLI commands.

Each IP address and AJP port of the cluster members must be configured as an outgoing socket binding. Assuming there are two cluster members:

```
/socket-binding-group=ha-sockets/remote-destination-outbound-socket-binding=\ cluster-
member1/:add(host=10.1.2.3, port=8009)
```

```
/socket-binding-group=ha-sockets/remote-destination-outbound-socket-binding=\ cluster-
member2/:add(host=10.1.2.13, port=8009)
```

The reverse-proxy handler is created in the undertow subsystem:

```
/profile=ha/subsystem=undertow/configuration=handler/reverse-proxy=lb:add
```

Each cluster member is added as a route to the reverse-proxy handler by a host child process:

```
/profile=ha/subsystem=undertow/configuration=handler/reverse-proxy=lb/host=member1:\ add(outbound-socket-
binding=cluster-member1,scheme=ajp,instance-id=hosta:server1,\ path=/app)
```

```
/profile=ha/subsystem=undertow/configuration=handler/reverse-proxy=lb/host=member2:\ add(outbound-socket-
binding=cluster-member2,scheme=ajp,instance-id=hostb:server2,\ path=/app)
```

In the preceding commands, each instance-id attribute in the path must match the jboss.server.node system property for the referenced node. This system property generally takes the form host_name:server_name. The value of the path attribute must match the context path of the clustered applications.

Enable the reverse-proxy handler on the desired Undertow (virtual) hosts:

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

In the example command above, it was necessary to escape the slash (/) in /app using a backslash (\), since the slash is part of the CLI syntax.

Note that this example only configures the load balancer for a single clustered application, whose context path is /app. If the load balancer must support more applications from the same cluster, add each one as a new host child process with a different route attribute.

How Undertow Load Balancers Implement Session Affinity

Java EE web containers typically use the HTTP cookie named JSESSIONID to store a unique session ID as specified by the Servlet API specification. This cookie value is used as an index to retrieve user session data stored in memory.

Undertow uses the session ID cookie to provide an efficient implementation of session affinity by appending the jboss.server.node system property to its value and then matching this additional data to the instance-id attribute of the session id cookie. route.
More detailed explanation:

- **The EAP 7 servlet container attaches the jboss.server.node system property to the HTTP session ID cookie when sending a response to the web browser.**
- **The session cookie is stored in the memory of the web browser, and is sent to the server web with the following HTTP request.**
- **The load balancer intercepting the request gets the value attached to the cookie sent with the request and forwards the request to the back-end web server whose instance-id attribute of the route matches the value.**

use

Note that the EAP server and hostnames that contain symbols can break the load balancer's ability to get the value attached to the session cookie, and this will break session affinity. If this occurs, set the jboss.server.node system property on each node to a safe value, using only letters and numbers.

The dynamic load balancer also uses the instance-id attribute in the same way that a static load balancer does. What mod_cluster actually does is create the load balancing configurations at runtime. The modcluster subsystem on the cluster member's servers provides all the necessary information (including the value of the jboss.server.node system property), and the mod_cluster filter creates a hidden reverse_proxy handler and configures its routes.

Using Apache Httpd and other web servers as a load balancer

Previous versions of EAP did not have an Undertow subsystem, and the embedded web server, provided by the JBoss web subsystem and based on Apache Tomcat, did not offer proxy capabilities.

Red Hat supported a number of native web server plugins, such as load balancers, for many OSes and web servers. Some of them are still supported by Red Hat through the Red Hat JBoss Core Servers product, which is included as part of the EAP 7 subscription.

JBoss Core Services offers the following load balancers, based on the native Apache Httpd modules for Linux, Windows, and Solaris OS:

- mod_jk – This was the first module to implement a static load balancer. based on the AJP protocol and was developed by the Tomcat community. It also works with Microsoft IIS, but this particular configuration is NO longer supported as part of the EAP subscription.
- mod_proxy: This is the native Apache Httpd proxy support, and can function as static load balancer using HTTP or AJP.
- mod_cluster – This is the mod_cluster client for Apache Httpd. It works in conjunction with Apache Httpd's mod_proxy to provide a dynamic load balancer using HTTP or AJP.

Configuration details for load balancers follow the same concepts already presented for Undertow. The details and configuration syntax for each are presented in the EAP 7 Configuration Guide.

References

For more information about Undertow and ModCluster:

- **EAP 7 Configuration Guide; see the "Configure JBoss EAP as a front-end load balancer" section of the "High Availability Configuration" chapter** https://access.redhat.com/documentation/en/red-hat-jboss-enterprise application-platform/7.0/ configuration-guide/configuration-guide
- **Undertow web server project** http://undertow.io
- **Undertow Community Documentation** http://undertow.io/undertow-docs/undertow-docs-1.3.0/index.html

For more information about load balancers based on Native Apache Httpd:

- **Red Hat JBoss Core Services Documentation** https://access.redhat.com/documentation/en/red-hat-jboss-core-services/
- **Apache Module Community Page Httpd mod_cluster** http://mod-cluster.jboss.org/