NGINX

Using NGINX to Load Balance IBM WebSphere

Deployment Guide for NGINX and IBM WebSphere Application Server

Published June 19, 2015
Version 1.1

Table of Contents

This guide explains how to deploy NGINX and NGINX Plus to load balance traffic across a pool of IBM WebSphere Application Servers. It provides complete instructions for customizing both NGINX and IBM WebSphere as required.

About NGINX Plus

NGINX Plus is the commercially supported version of the open source NGINX software. NGINX Plus is a complete application delivery platform, extending the power of NGINX with a host of enterprise-ready capabilities that are instrumental to building web applications at scale:

- Full-featured HTTP and TCP load balancing
- High-performance reverse proxy
- Caching and offload of dynamic and static content
- Adaptive streaming to deliver audio and video to any device
- Application-aware health checks and high availability
- Advanced activity monitoring available via a dashboard or API
- Management and real-time configuration changes with DevOps-friendly tools

Using this Guide

 It was originally created by IBM authors:

- Ed Lu elu@us.ibm.com
- Eric Covener ecovener@us.ibm.com

After reviewing the "Prerequisites and System Requirements" on **page 4**, perform the instructions in the following sections.

Prerequisites and System Requirements

- IBM WebSphere, installed and configured on a system.
- Linux system to host NGINX or NGINX Plus (in on-premise and private-cloud deployments). To avoid potential conflicts with other applications, we recommend you install NGINX Plus on a fresh system. For the list of Linux distributions supported by NGINX Plus, see **NGINX Plus Technical Specifications**.
- NGINX 1.9.1 or later; or NGINX Plus R6 or later.

The instructions assume you have basic Linux system administration skills, including the following. Full instructions are not provided for these tasks.

- Installing Linux software from vendor-supplied packages
- Editing configuration files
- Copying files between a central administrative system and Linux servers
- Running basic commands to start and stop services
- Reading log files

NGINX vs. IBM HTTP Server

NGINX is an open source webserver and reverse proxy that has grown in popularity in recent years due to its scalability. NGINX was first created to solve the C10K problem - serving 10,000 simultaneous connections on a single webserver. NGINX's features and performance have made it a staple of high performance sites -- now powering 1 in 3 of the world's busiest web properties

NGINX Plus is the commercial (subscription-based) version of NGINX open source. It adds a number of features for load-balancing and proxying traffic that can enhance a WebSphere Application Server deployment, including session persistence, health checks, dynamic configuration and live activity data. NGINX Plus is fully supported by NGINX, Inc.

Further information about NGINX Plus can be found here.

On the other hand, IBM HTTP Server (IHS) is the traditional reverse proxy for WebSphere Application Server. IHS and WAS are tightly integrated via the WebSphere WebServer Plug-in and there is a wealth of documentation and expertise in the WebSphere community.

In many ways, NGINX and IHS are similar. They are configured similarly, and behave similarly on the outside. However, on the inside, they are quite different. At low load or ideal network conditions, they perform comparably well. However, NGINX is made to scale well on large numbers of long-lived connections and in the face of large traffic variations - exactly where IHS's one-thread-per-connection struggles.

In addition, NGINX is well-equipped to handle HTTP/2 traffic - it currently supports SPDY, and powers the majority of sites that have adopted this standard. HTTP/2 support is planned for release in late 2015; there are currently no plans to support HTTP/2 within IHS.

On the other hand, there are some WebSphere specific features that are currently only integrated into IHS and Datapower - for example, dynamic clustering of WebSphere Application Servers. Currently, NGINX can only proxy to statically-defined groups of application servers. NGINX Plus does offer an API to configure load balancing groups, or can be configured via DNS.

NGINX may provide significant value in environments where acceleration via SPDY, HTTP/2 or a high performance centralized cache smaller than an appliance is needed.

Basic Load Balancing with NGINX

This section outlines some basic information on setting up NGINX as a load balancer and reverse proxy. The configuration here is essentially equivalent to what one would get with a generated plugin-cfg.xml for the WAS plugin.

NGINX's proxying is based around the concept of an "upstream group", which defines a group of servers. Setting up a simple reverse proxy invloves defining an upstream group, then using it in one or more proxy_pass directives.

Here is an example of an upstream group:

```
http {
    ...

     upstream websphere {
        server 127.0.0.1:9080;
        server 127.0.0.1:9081;
     }
}
```

The upstream group above, named websphere, is defined by two servers. Both are on ip 127.0.0.1, with one server on port 9080 and the other on 9081. Note that the upstream group is placed within an http block.
We use the proxy_pass directive within a location block to point at an upstream:

```
http {
     ...
     server {
          ...

          location /webapp/ {
              proxy_pass http://websphere;
          }
     }
}
```

This tells NGINX to proxy all HTTP requests starting with /webapp/ to one of the servers in the `websphere` upstream. Note that this configuration only applies to HTTP traffic; additional configuration is needed both for SSL and for Proxying Websockets.

For more information on proxying, refer to the official NGINX documentation on the proxy module and the upstream module.

The NGINX guides: Load Balancing part 1 and Load Balancing part 2 provide a step-by-step walkthrough, and the document on-the-fly reconfiguration of NGINX Plus describes how NGINX Plus' load balancing groups can be configured using an API or DNS.

Load Balancing

The default strategy for load balancing among servers in a given upstream group is round-robin. In round-robin load balancing, requests are distributed evenly among all servers in turn. For example, in the above proxying configuration, the first request will go to port 9080, the second to 9081, the third to 9080, and so on.

There are several other load balancing strategies included in NGINX; for more info, see this article on load balancing.

Failover

Failover is also automatically configured when using an upstream group. By default, when a request comes in and it is directed to an unreachable server, NGINX marks that server down for some time and automatically redirects the request to another server. After 10 seconds, NGINX starts sending requests to the downed server. If those requests succeed, the server returns to normal operation; otherwise, it remains down for another 10 seconds.

See the documentation for the server directive to configure parameters related to failover.

Session Persistence

If your architecture has IHS between NGINX and WebSphere, then no session persistence on the NGINX side is required - IHS will handle session persistence. If NGINX is proxying straight to WAS, then session persistence may be beneficial.

Session persistence in the open-source version can be achieved by modifying the load balancing algorithm or using third-party modules. The `ip_hash` load balancing method will provide session persistence for clients that are not changing their IP addresses frequently.

In order to use hash based load balancing methods modify the upstream block:

```
upstream websphere {
    ip_hash;
    server 127.0.0.1:9080;
    server 127.0.0.1:9081;
}
```

Configuring NGINX to Proxy SSL traffic

This section describes how to configure SSL communication between NGINX and WebSphere as well as TLS communication between the client software and NGINX.

Enabling SSL in NGINX

NGINX Plus includes SSL support by default. If using the open source version of NGINX, the SSL module must be enabled manually Building NGINX from Source. During the configure step, pass the argument:

```
--with-http_ssl_module
```

If this argument is not passed to configure, NGINX will not support the directives needed for SSL communication.

Extracting the certificate and private key

It's likely that the server certificate is currently stored in a kdb file for use with WebSphere. Before using the server certificate with NGINX, it must be converted into PEM format, and the private key must be separated out. If you happen to have the certificate and key file in PEM format already, you can skip this subsection.

Note that the commands below using `gskcapicmd` can also be executed in `ikeyman` by selecting the relevant options in the GUI.

1. Find the certificate you wish to export from the kdb. List the certificates by executing:
   `gskcapicmd -cert -list -db key.kdb -stashed`

2. Export the certificate and its associated private key to a pkcs12 file:
   ```
   gskcapicmd -cert -export -db key.kdb -label CERT_LABEL -type cms -
   target /tmp/conv.p12 -target_type p12
   ```

3. Extract the certificate and the private key from the pkcs12 file:
   ```
   openssl pkcs12 -in /tmp/conv.p12 -nocerts -out privkey.pem
   openssl pkcs12 -in /tmp/conv.p12 -clcerts -nokeys -out servercert.pem
   ```

4. Extract any intermediate certificates from the kdb:
   ```
   gskcapicmd -cert -extract -db key.kdb -label INTERMEDIATE_LABEL –file
   /tmp/intermediate.pem -format ascii
   ```

5. Concatenate the intermediate certificates and the server certificate, making sure the server certificate is first:
   ```
   cat servercert.pem /tmp/intermediate1.pem /tmp/intermediate2.pem >
   certchain.pem
   ```

The certificate and secret key are now in a format that is useable by NGINX.

WARNING: `privkey.pem` contains the unencrypted private key of the server, and should be secured appropriately.

Configuring SSL in NGINX

A new server block must be created for the SSL server configuration. The default configuration provides one, commented out by default. Here it is along with a sample location block, for proxying to WebSphere:

```
server {
    listen       443 ssl;
    server_name  myserver;

    ssl_certificate      certchain.pem;
    ssl_certificate_key  privkey.pem;

    ssl_session_cache    shared:SSL:1m;
    ssl_session_timeout  5m;

    ssl_ciphers  HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers  on;

    location /webapp/ {
        proxy_pass https://websphere_ssl
    }
}
```

The arguments to the `ssl_certificate` and `ssl_certificate_key` should be changed to the paths to your certificate bundle and private key in PEM format. The `location` block is the same as that in the previous section on proxying, except that the scheme is changed from `http` to `https`.

For more information on SSL configuration in NGINX, refer to this article and the documentation on the ssl module.

Proxying Websockets

When the WebSphere WebServer Plug-in is used in an Apache-based server, websockets traffic is proxied without any additional configuration. If any other HTTP terminating software is used with websockets, it typically requires explicit configuration.

In NGINX, upstream connections use HTTP/1.0 by default. Websocket connections require HTTP/1.1 along with some other configuration to be proxied correctly. Here is an example configuration:

```
http {
    ...
    map $http_upgrade $connection_upgrade {
        default upgrade;
        ''      close;
    }

    server {
        ...

        location /wstunnel/ {
            proxy_pass http://websphere;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection $connection_upgrade;
        }
    }
}
```

Special rules are needed because the Upgrade header is a hop-by-hop header, i.e. the HTTP specification explicitly notes that is not to be passed on by proxies. In the configuration above, we explicitly pass on the Upgrade header. Additionally, if the request had an `Upgrade` header, then the `Connection` header is set to `upgrade`; otherwise, it is set to `close`.

Some additional issues must be considered if NGINX is proxying to IHS and will need to maintain many thousands of open websockets connections. IHS is not well equipped to handle high loads of long-lived connections. For this reason, it is better to skip over the IHS instance when proxying websockets to the application server. A nested location directive can be used for this, e.g.:

```
location / {
    proxy_pass http://IBMHTTPServer;
    location /wstunnel/ {
        proxy_pass http://websphere;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
    }
}
```

See this article for more information on proxying websockets.

Caching

Scalable HTTP caches have a recognized value on the edge of the network. The Datapower XC10 appliance is an example of a sophisticated, scalable HTTP caching appliance using a WebSphere Extreme Scale (WXS) datagrid for storage.

NGINX provides a scalable disk-based cache that integrates with its reverse proxy capability. The `proxy_cache` directive is the key here. Here is a very simple caching configuration:

```
http {
    ...
    proxy_cache_path /tmp/NGINX_cache/ keys_zone=backcache:10m;

    server {
        listen 80;
        proxy_cache backcache;
        location /webapp/ {
            proxy_pass http://websphere;
        }
    }
}
```

This configuration creates a cache in the directory `/tmp/NGINX_cache/`, and caches all responses that come through the proxy on port 80. Note that the size argument to `keys_zone` (10m in this case) should be set proportional to the number of pages that should be cached. The value is OS dependent, as it relies on file metadata size.

For more complete information on caching in NGINX, refer to the offical documentation and this article.

SPDY Support

At time of writing, NGINX does not support HTTP/2. NGINX does, however, support SPDY, the predecessor of HTTP/2. SPDY is not enabled in the server by default, and support is experimental as the protocol specification is subject to change.

NGINX Plus includes SPDY support by default, but it is a compile-time option for NGINX. To compile SPDY into NGINX, follow the steps in the section below, "Building NGINX from Source". Upon the configure step, instead of running only `./configure`, add the `--with-http_spdy_module` parameter:

```
./configure --with-http_spdy_module
```

After completing the rest of the build steps, you should have a build of NGINX with SPDY enabled. To enable SPDY, simply add the `spdy` keyword to the `listen` statement:

```
server {
    listen       443 ssl spdy;
           ….
}
```

For more information, see the documentation on the SPDY module.

Advanced Load Balancing with NGINX Plus

This section outlines how to configure the additional functionality available with NGINX Plus.

Health checks

Health checks are out-of-band HTTP requests sent to probe a server on a preset interval. They are used to determine whether or not a server is still up without requiring an actual request from a user. To enable health checks for a server, add the `health_check` directive to the location block of a given `proxy_pass` directive:

```
http {
    upstream websphere {
        # Health-monitored upstream groups must be stored in shared memory
        zone websphere 64k;

        server 127.0.0.1:9080 slow_start=30s;
        server 127.0.0.1:9081 slow_start=30s;
    }

    server {
        location /webapp/ {
            proxy_pass http://websphere;
            health_check;
        }
    }
}
```

This configuration will send an out-of-band request for the URI / to each of the servers in the upstream websphere every 5 seconds. Any server that does not respond with a successful response will be marked down. Note that the `health_check` directive is placed within the location block, not the upstream block; this means health checks can be enabled per-application.

Unlike previous upstream groups in this document, the one in the above example has a zone directive. This directive defines a shared memory zone which stores the group's configuration and run-time state, and is required when using the health check feature.

See the documentation for the health_check directive for more information and details on how to customize health checks.

NGINX Plus also provides a slow start feature when failed servers recover and are reintroduced into the load-balancing pool. `slow_start` is configured per server in the upstream group.

Session Persistence

NGINX Plus has a built-in sticky directive to handle session persistence. We can use the `JSESSIONID` cookie, created by WAS, as the session identifier by taking advantage of the learn method. Here is an example configuration:.

```
upstream websphere {
   zone websphere 64k;
   server 127.0.0.1:9080 slow_start=30s;
```

```
        server 127.0.0.1:9081 slow_start=30s;

        sticky learn
               create=$upstream_cookie_JSESSIONID
               lookup=$cookie_JSESSIONID
               zone=client_sessions:1m;
    }
```

Note that for any given request, the `$upstream_cookie_JSESSIONID` variable contains the value of the `JSESSIONID` cookie sent from the backend server in the `Set-Cookie` header. Likewise, the `$cookie_JSESSIONID` variable contains the value of the `JSESSIONID` cookie sent from the client in the Cookie header. These two variables, specified in the create and lookup arguments, specify what values are used to create new sessions and lookup existing sessions, respectively.

The `zone` argument specifies a shared memory zone where sessions are stored. The size passed to the argument - one megabyte, here - determines how many sessions can be stored at a time. The amount of sessions that can be stored in any given space is platform dependent. Note that the name passed to the `zone` argument must be unique for each `sticky` directive.

For more information on sticky sessions, refer to the documentation on the sticky directive.

Enabling the Live Activity Monitoring Dashboard

NGINX Plus includes a live activity monitoring interface that provides key load and performance metrics in real time. Statistics are reported through a RESTful JSON interface, making it very easy to feed the data to a custom or third-party monitoring tool. These instructions deploy the display dashboard that is built into NGINX Plus.

For more information on live activity monitoring, refer to the documentation on live activity monitoring.

Installing the Dashboard

The quickest way to configure the built-in NGINX Plus dashboard is to download the sample configuration file from the NGINX, Inc. website:

```
# curl http://nginx.com/resources/conf/status.txt >
/etc/nginx/conf.d/status.conf
```

Comments in the `status.conf` file explain which directives you must customize for your deployment. In particular, the default settings in the sample configuration file allow anyone on any network to access the dashboard. We strongly recommend that you restrict access to the dashboard with one or more of the following methods:

IP address-based access control lists (ACLs)

In the sample configuration file, uncomment the `allow` and `deny` directives, and substitute the address of your administrative network for 10.0.0.0/8. Only users on the specified network can access the status page.

```
allow 10.0.0/8;
deny all;
```

HTTP basic authentication

In the sample configuration file, uncomment the auth_basic and auth_basic_user_file directives and add user entries to the **/etc/nginx/users** file (for example, by using an htpasswd generator).

```
auth_basic on;
auth_basic_user_file /etc/nginx/users;
```

Client certificates, which are part of a complete configuration of SSL or TLS. For more information, see

NGINX SSL Termination in the NGINX Plus Admin Guide and the documentation for the HTTP SSL module.

Firewall

Configure your firewall to disallow outside access to the port for the dashboard (**8080** in the sample configuration file).

Finalizing Configuration of the Dashboard

Inside the server blocks for each both the ssl and unencrypted servers, add the status_zone directive:

```
server {
    listen 80;
    status_zone websphere;
    ...
}
```

When you reload NGINX Plus with the new configuration:

```
# nginx –s reload
```

The NGINX Plus dashboard is available immediately at **http://*nginx-server-address*:8080**.

Manually Translating plugin-cfg.xml

This section brings together the information from the above sections and uses it to translate a simple `plugin-cfg.xml`. If you skipped straight to this section, that's fine; refer to the above sections when necessary.

First, we must create the `upstream` blocks. Within the `plugin-cfg.xml`, locate a `ServerCluster` block which you wish to translate. Within each `Server` tag within the `ServerCluster`, you should see `Transport` tags, e.g.:

```
<ServerCluster Name="defaultServer_default_node_Cluster" ... >
    <Server ... >
        <Transport Hostname="localhost" Port="9080" Protocol="http"/>
        <Transport Hostname="localhost" Port="9443" Protocol="https">
            ...
        </Transport>
    </Server>
    <Server ... >
        <Transport Hostname="localhost" Port="9081" Protocol="http"/>
        <Transport Hostname="localhost" Port="9444" Protocol="https">
            ...
        </Transport>
    </Server>
</ServerCluster>
```

One upstream is required for each of the `Protocol` values. The hostname and port in each server directive within the upstream should correspond to the Hostname and Port values within the `Transport` tags in the `plugin-cfg.xml`. For the sample snippet above, the corresponding upstream blocks in the `NGINX.conf` could be:

```
http {
    ...
    upstream defaultServer_default_node_Cluster_http {
        zone cluster_http 64k;
        server 127.0.0.1:9080;
        server 127.0.0.1:9081;
    }

    upstream defaultServer_default_node_Cluster_https {
        zone cluster_https 64k;
        server 127.0.0.1:9443;
        server 127.0.0.1:9444;
    }
}
```

Now, we must map URLs to these upstream blocks. Find the `UriGroup` section corresponding to the cluster:

```
<UriGroup Name="default_host_defaultServer_default_node_Cluster_URIs">
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
Name="/webapp/*"/>
</UriGroup>
```

Each `Uri` tag in the `UriGroup` needs at least one corresponding location block with `proxy_pass` directives to point to the proper upstream groups. For the example `UriGroup`, the corresponding HTTP location block could be:

```
server {
    listen 80;
    ...

    location /webapp/ {
        proxy_pass http://defaultServer_default_node_Cluster_http;
    }
}
```

And the HTTPS location block, in the HTTPS-configured server block:

```
server {
    listen 443 ssl;
    ...

    location /webapp/ {
        proxy_pass https://defaultServer_default_node_Cluster_http;
    }
}
```

Note that, by default, the argument to `location` specifies a URL prefix to match against. More complex arguments to the location block might be needed, depending on the value of the Name attribute of the URI tag.

See the NGINX documentation for more details on the location block.

The above should be enough for simple communication between NGINX and Websphere Application Server for a server cluster. The above steps should be repeated for each `ServerCluster` block within the `plugin-cfg.xml`.

Further configuration

In the last section, we skipped some of the configuration within the `ServerCluster` and Server tags themselves, such as `ConnectTimeout` and `ServerIOTimeout`. NGINX has similar configuration options. Here are some of the common arguments to `ServerCluster` and `Server`, and their counterparts in NGINX:

- `LoadBalanceWeight`: weight argument to the server directive.
- `RetryInterval`: fail_timeout argument to the server directive.
- `MaxConnections`: (NGINX Plus only) The `max_conns` argument to the server directive.
- `ConnectTimeout`: `proxy_connect_timeout` directive.
- `ServerIOTimeout`: `proxy_read_timeout` directive.
- `LoadBalance`: See the above section, "Load Balancing".
- `AffinityCookie`: See the above section, "Session Persistence".

Documentation for these directives can be found in the upstream module documentation and the proxy module documentation.

Private WebSphere Headers

Normally, the IHS plugin sets some private headers that are sent only to the application server. These headers may affect application processing. NGINX does not know how to set these headers by default.

Here is a sample config, which sets the private headers:

```
http {
    ...
    map $https $is_ssl {
        default false;
        on      true;
    }
    server {
        ...

        location / {
            proxy_pass http://websphere;

            proxy_set_header "$WSSC" $scheme;
            proxy_set_header "$WSPR" $server_protocol;
            proxy_set_header "$WSRA" $remote_addr;
            proxy_set_header "$WSRH" $host;
            proxy_set_header "$WSRU" $remote_user;
            proxy_set_header "$WSSN" $server_name;
            proxy_set_header "$WSSP" $server_port;
            proxy_set_header "$WSIS" $is_ssl;

            # Note that these vars are only available if
            # NGINX was built with SSL
            proxy_set_header "$WSCC" $ssl_client_cert;
            proxy_set_header "$WSCS" $ssl_cipher;
            proxy_set_header "$WSSI" $ssl_session_id;

            # No equivalent NGINX variable for these headers.
            proxy_hide_header "$WSAT";
            proxy_hide_header "$WSPT";
            proxy_hide_header "$WSFO";
        }
    }
}
```

Most of the headers are set using the `proxy_set_header` directive, in combination with NGINX's embedded variables. The only tricky header is `$WSIS`, whose value is mapped from the value of `https` variable.

Plugin configuration

This section only applies if you are proxying from NGINX to IBM HTTP Server.

Usually, the WAS plugin will not allow the client to set any private headers. However, in this case, the plugin must allow NGINX to set a few headers such as remote IP and host. We can allow NGINX to set these headers through the `TrustedProxyEnable` and `TrustedProxyList` custom properties of the plugin.

The value of `TrustedProxyEnable` should be set to true. and the value of `TrustedProxyList` should be set to the hostname or IP of the machine that is running NGINX. These values can either be manually set in the `plugin-cfg.xml`, or through the WAS administrative console. For instructions on setting them through the admin console, see the knowledgecenter.

Building NGINX from Source

This section only applies if you are using the open-source version of NGINX.

Certain features in the open source version of NGINX can only be enabled when it is compiled from source. The following instructions are for a Unix-like system, with GNU make and buildtools. They were tested on version 1.9.1, but the procedure should be similar for other versions.

1. Download NGINX source code:
    - `wget http://nginx.org/download/nginx-1.9.1.tar.gz`
2. Unpack the archive using an unarchiver, i.e.
    - `tar -xvzf nginx-1.9.1.tar.gz -C $HOME`.
3. Configure NGINX for your platform by cding to the directory you unzipped NGINX to, and running:
    - `./configure --with-http_spdy_module --with-http_ssl_module`
    - Add a `--add-module=THIRD_PARTY_MODULE_ROOT` for each third party module being compiled in, i.e. for session persistence.
    - Arguments to `configure` are generally the way to add features. If you came to this section from a previous section, now would be a good time to go back and check the instructions there.
4. Run `make` to compile NGINX.
5. Run `make install` to install the newly compiled version of NGINX. You may need to run this command as root.
    - **NOTE**: This will install NGINX into the `/usr/local/` directory. Running this as root **will replace any previous installation of NGINX** installed in this directory. Configuration files and logs will be kept, but other files will be overwritten. Binary packages from standard distros install NGINX into the /usr/ directory. If this case please make sure to remove the older version and backup your configuration before running make install.

Configuration Samples

This section provides sample configurations for both basic load balancing and advanced load balancing with NGINX Plus. The configuration file should be saved to `/etc/nginx/conf.d/websphere.conf`. In the main `/etc/nginx/nginx.conf` file ensure that `include /etc/nginx/conf.d/*.conf;` is present.

Basic Load Balancing Sample Configuration

This section provides a sample configuration for use with NGINX Open Source.

```
proxy_cache_path /tmp/NGINX_cache/ keys_zone=backcache:10m;

map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
}

map $https $is_ssl {
    default false;
    on      true;
}

upstream websphere {

    # Use IP Hash for session persistence
    ip_hash;

    # List of WebSphere Application Servers
    server 127.0.0.1:9080;
    server 127.0.0.1:9081;
}

upstream websphere-ssl {

    # Use IP Hash for session persistence
    ip_hash;

    # List of WebSphere Application Servers
    server 127.0.0.1:9443;
    server 127.0.0.1:9444;
}

server {
    listen 80;

    # Return a 302 Redirect to the /webapp/ directory
    # when user requests /
    location = / {
        return 302 /webapp/;
    }

    # A location block is need per URI group
    location /webapp/ {
        proxy_pass http://websphere;
        proxy_cache backcache;

        proxy_set_header "$WSSC" $scheme;
```

```
        proxy_set_header "$WSPR" $server_protocol;
        proxy_set_header "$WSRA" $remote_addr;
        proxy_set_header "$WSRH" $host;
        proxy_set_header "$WSRU" $remote_user;
        proxy_set_header "$WSSN" $server_name;
        proxy_set_header "$WSSP" $server_port;
        proxy_set_header "$WSIS" $is_ssl;

        # Note that these vars are only available if
        # NGINX was built with SSL
        proxy_set_header "$WSCC" $ssl_client_cert;
        proxy_set_header "$WSCS" $ssl_cipher;
        proxy_set_header "$WSSI" $ssl_session_id;

        # No equivalent NGINX variable for these headers.
        proxy_hide_header "$WSAT";
        proxy_hide_header "$WSPT";
        proxy_hide_header "$WSFO";
    }

    # WebSocket configuration
    location /wstunnel/ {
        proxy_pass http://websphere;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
    }

}

server {
    listen 443 ssl spdy;

    ssl_certificate      certchain.pem;
    ssl_certificate_key  privkey.pem;

    ssl_session_cache    shared:SSL:1m;
    ssl_session_timeout  5m;

    ssl_ciphers  HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers  on;

    # Return a 302 Redirect to the /webapp/ directory
    # when user requests /
    location = / {
        return 302 /webapp/;
    }

    # A location block is need per URI group
    location /webapp/ {
        proxy_pass https://websphere;
        proxy_cache backcache;

        proxy_set_header "$WSSC" $scheme;
        proxy_set_header "$WSPR" $server_protocol;
        proxy_set_header "$WSRA" $remote_addr;
        proxy_set_header "$WSRH" $host;
        proxy_set_header "$WSRU" $remote_user;
        proxy_set_header "$WSSN" $server_name;
        proxy_set_header "$WSSP" $server_port;
```

```
        proxy_set_header "$WSIS" $is_ssl;

        # Note that these vars are only available if
        # NGINX was built with SSL
        proxy_set_header "$WSCC" $ssl_client_cert;
        proxy_set_header "$WSCS" $ssl_cipher;
        proxy_set_header "$WSSI" $ssl_session_id;

        # No equivalent NGINX variable for these headers.
        proxy_hide_header "$WSAT";
        proxy_hide_header "$WSPT";
        proxy_hide_header "$WSFO";
    }

    # WebSocket configuration
    location /wstunnel/ {
        proxy_pass https://websphere;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
    }
}
```

Advanced Load Balancing Sample Configuration

This section provides a sample configuration using the advanced features available in NGINX Plus.

```
proxy_cache_path /tmp/NGINX_cache/ keys_zone=backcache:10m;

map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
}

map $https $is_ssl {
    default false;
    on      true;
}

upstream websphere {
    # Health-monitored upstream groups must be stored in shared memory
    zone websphere 64k;

    # List of WebSphere Application Servers
    server 127.0.0.1:9080 slow_start=30s;
    server 127.0.0.1:9081 slow_start=30s;

    # Session Persistence based on JSESSION ID, if necessary
    sticky learn
      create=$upstream_cookie_JSESSIONID
      lookup=$cookie_JSESSIONID
      zone=client_sessions:1m;
}

upstream websphere-ssl {
    # Health-monitored upstream groups must be stored in shared memory
    zone websphere-ssl 64k;

    # List of WebSphere Application Servers
```

```
    server 127.0.0.1:9443 slow_start=30s;
    server 127.0.0.1:9444 slow_start=30s;

    # Session Persistence based on JSESSION ID, if necessary
    sticky learn
      create=$upstream_cookie_JSESSIONID
      lookup=$cookie_JSESSIONID
      zone=client_sessions-ssl:1m;
}

server {
    listen 80;

    # Required for NGINX Plus to provide extended status information.
    status_zone websphere;

    # Return a 302 Redirect to the /webapp/ directory
    # when user requests /
    location = / {
        return 302 /webapp/;
    }

    # A location block is need per URI group
    location /webapp/ {
        proxy_pass http://websphere;
        proxy_cache backcache;

        # Set up active health checks.  If the server responds with a
        # status other than 2xx or 3xx, the health check will fail
        # and the server will be marked down.
        # For more options with health_check, see:
        # http://nginx.org/en/docs/http/ngx_http_upstream_module.html
        health_check;

        proxy_set_header "$WSSC" $scheme;
        proxy_set_header "$WSPR" $server_protocol;
        proxy_set_header "$WSRA" $remote_addr;
        proxy_set_header "$WSRH" $host;
        proxy_set_header "$WSRU" $remote_user;
        proxy_set_header "$WSSN" $server_name;
        proxy_set_header "$WSSP" $server_port;
        proxy_set_header "$WSIS" $is_ssl;

        # Note that these vars are only available if
        # NGINX was built with SSL
        proxy_set_header "$WSCC" $ssl_client_cert;
        proxy_set_header "$WSCS" $ssl_cipher;
        proxy_set_header "$WSSI" $ssl_session_id;

        # No equivalent NGINX variable for these headers.
        proxy_hide_header "$WSAT";
        proxy_hide_header "$WSPT";
        proxy_hide_header "$WSFO";
    }

    # WebSocket configuration
    location /wstunnel/ {
        proxy_pass http://websphere;
        proxy_http_version 1.1;
```

```
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
    }
}

server {
    listen 443 ssl spdy;

    # Required for NGINX Plus to provide extended status information.
    status_zone websphere-ssl;

    ssl_certificate      certchain.pem;
    ssl_certificate_key  privkey.pem;

    ssl_session_cache    shared:SSL:1m;
    ssl_session_timeout  5m;

    ssl_ciphers  HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers  on;

    # Return a 302 Redirect to the /webapp/ directory
    # when user requests /
    location = / {
        return 302 /webapp/;
    }

    # A location block is need per URI group
    location /webapp/ {
        proxy_pass https://websphere-ssl;
        proxy_cache backcache;

        # Set up active health checks.  If the server responds with a
        # status other than 2xx or 3xx, the health check will fail
        # and the server will be marked down.
        # For more options with health_check, see:
        # http://nginx.org/en/docs/http/ngx_http_upstream_module.html
        health_check;

        proxy_set_header "$WSSC" $scheme;
        proxy_set_header "$WSPR" $server_protocol;
        proxy_set_header "$WSRA" $remote_addr;
        proxy_set_header "$WSRH" $host;
        proxy_set_header "$WSRU" $remote_user;
        proxy_set_header "$WSSN" $server_name;
        proxy_set_header "$WSSP" $server_port;
        proxy_set_header "$WSIS" $is_ssl;

        proxy_set_header "$WSCC" $ssl_client_cert;
        proxy_set_header "$WSCS" $ssl_cipher;
        proxy_set_header "$WSSI" $ssl_session_id;

        # No equivalent NGINX variable for these headers.
        proxy_hide_header "$WSAT";
        proxy_hide_header "$WSPT";
        proxy_hide_header "$WSFO";
    }

    # WebSocket configuration
    location /wstunnel/ {
        proxy_pass https://websphere-ssl;
```

```
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection $connection_upgrade;
        }
    }
```

Summary

This concludes the *Using NGINX to Load Balance IBM WebSphere Deployment Guide*. In this guide we went through how to configure to configure NGINX to properly load balalnce IBM WebSphere Application Servers. For further information about NGINX and NGINX Plus, please see the following:

- NGINX Plus Overview
- NGINX Plus Admin Guide
- NGINX Wiki