API

2015-05-18 16:58:20 UTC

Contents

API

The following topics provides information on the API support provided for the NetScaler appliance. Intended for application developers who want to configure and monitor a NetScaler appliance programmatically.

NITRO API	Describes the use of the NITRO APIs for the REST, Java, and .NET platforms.
XML API	Describes the properties and use of the XML API.

NITRO API

The NetScaler NITRO protocol allows you to configure and monitor the NetScaler appliance programmatically.

NITRO exposes its functionality through Representational State Transfer (REST) interfaces. Therefore, NITRO applications can be developed in any programming language. Additionally, for applications that must be developed in Java or .NET, NITRO APIs are exposed through relevant libraries that are packaged as separate Software Development Kits (SDKs).

Note: You must have a basic understanding of the NetScaler appliance before using NITRO.

To use the NITRO protocol, the client application needs only the following:

- Access to a NetScaler appliance, version 9.2 or later.
- To use REST interfaces, you must have a system to generate HTTP or HTTPS requests (payload in JSON format) to the NetScaler appliance. You can use any programming language or tool.
- For Java clients, you must have a system where Java Development Kit (JDK) 1.5 or later is available. The JDK can be downloaded from http://www.oracle.com/technetwork/java/javase/downloads/index.html.
- For .NET clients, you must have a system with .NET framework 3.5 or later installed. The .NET framework can be downloaded from http://www.microsoft.com/downloads/en/default.aspx.

Note: You can also use XML APIs to configure the NetScaler appliance programmatically. For more information, see "XML API".

NITRO API

The NetScaler NITRO protocol allows you to configure and monitor the NetScaler appliance programmatically.

NITRO exposes its functionality through Representational State Transfer (REST) interfaces. Therefore, NITRO applications can be developed in any programming language. Additionally, for applications that must be developed in Java or .NET, NITRO APIs are exposed through relevant libraries that are packaged as separate Software Development Kits (SDKs).

Note: You must have a basic understanding of the NetScaler appliance before using NITRO.

To use the NITRO protocol, the client application needs only the following:

- Access to a NetScaler appliance, version 9.2 or later.
- To use REST interfaces, you must have a system to generate HTTP or HTTPS requests (payload in JSON format) to the NetScaler appliance. You can use any programming language or tool.
- For Java clients, you must have a system where Java Development Kit (JDK) 1.5 or later is available. The JDK can be downloaded from http://www.oracle.com/technetwork/java/javase/downloads/index.html.
- For .NET clients, you must have a system with .NET framework 3.5 or later installed. The .NET framework can be downloaded from http://www.microsoft.com/downloads/en/default.aspx.

Note: You can also use XML APIs to configure the NetScaler appliance programmatically. For more information, see "XML API".

Obtaining the NITRO Package

The NITRO package is available as a tar file on the Downloads page of the NetScaler appliance's configuration utility. You must download and un-tar the file to a folder on your local system. This folder is referred to as <NITRO_SDK_HOME> in this documentation.

The folder contains the NITRO libraries in the lib subfolder. The libraries must be added to the client application classpath to access NITRO functionality. The <NITRO_SDK_HOME> folder also provides samples and documentation that can help you understand the NITRO SDK.

Note:

- The REST package contains only documentation for using the REST interfaces.

How NITRO Works

The NITRO infrastructure consists of a client application and the NITRO Web service running on a NetScaler appliance. The communication between the client application and the NITRO web service is based on REST architecture using HTTP or HTTPS.

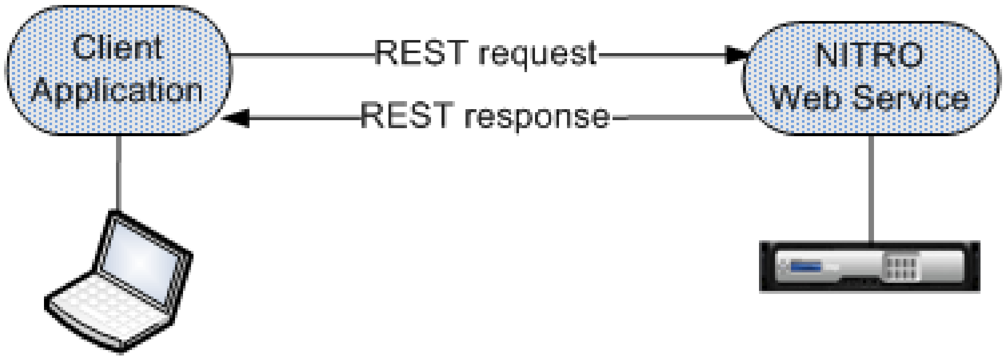


Figure 1. NITRO execution flow

As shown in the above figure, a NITRO request is executed as follows:

1. The client application sends REST request message to the NITRO web service. When using the SDKs, an API call is translated into the appropriate REST request message.

2. The web service processes the REST request message.

3. The NITRO web service returns the corresponding REST response message to the client application. When using the SDKs, the REST response message is translated into the appropriate response for the API call.

To minimize traffic on the NetScaler network, you retrieve the whole state of a resource from the server, make modifications to the state of the resource locally, and then upload it back to the server in one network transaction. For example, to update a load balancing virtual server, you must retrieve the object, update the properties, and then upload the changed object in a single transaction.

Note: Local operations on a resource (changing its properties) do not affect its state on the server until the state of the object is explicitly uploaded.

NITRO APIs are synchronous in nature. This means that the client application waits for a response from the NITRO web service before executing another NITRO API.

Java API

NetScaler NITRO APIs are categorized depending on the scope and purpose of the APIs into system APIs, feature configuration APIs, and feature statistics APIs. Additionally, you can import and export AppExpert applications. You can also troubleshoot NITRO operations.

Tutorials

These tutorials demonstrate the end-to-end usage of NITRO to achieve the following:

- Create Your First NITRO Application
- Create a NetScaler Cluster

Create Your First NITRO Application

After completing this tutorial, you will understand and be able to perform the following tasks:

- Integrate NITRO with the IDE
- Log in to the appliance
- Create a load balancing virtual server (lbvserver)
- Retrieve details of an lbvserver
- Delete an lbvserver
- Save the configurations on the appliance
- Log out of the appliance
- Debug the NITRO application

Before you begin, make sure that you have the latest NITRO SDK and that the client application satisfies the prerequisites for using the NITRO SDK.

Sample Code

For the executable code, see the <NITRO_SDK_HOME>/sample/MyFirstNitroApplication.java sample file.

To create your first NITRO application:

1. Copy the libraries from <NITRO_SDK_HOME>/lib folder to the project classpath.

2. Create a new class and name it **MyFirstNitroApplication**.

3. Create an instance of `com.citrix.netscaler.nitro.service.nitro_service` class. This instance is used to perform all operations on the appliance:

   ```
   nitro_service ns_session = new nitro_service("10.102.29.170","HTTP");
   ```

   This code establishes a connection with an appliance that has IP address 10.102.29.170 and uses the HTTP protocol. Replace 10.102.29.170 with the IP address of the NetScaler appliance that you have access to.
4. Use the `nitro_service` instance to log in to the appliance using your credentials:

   ```
   ns_session.login("admin","verysecret");
   ```

   This code logs into the appliance, with user name as `admin` and password as `verysecret`. Replace the credentials with your login credentials.
5. Enable the load balancing feature:

   ```
   String[] features_to_be_enabled = {"lb"};
   ns_session.enable_features(features_to_be_enabled);
   ```

   This code first sets the features to be enabled in an array and then enables the LB feature.
6. Create an instance of the `com.citrix.netscaler.nitro.resource.config.lb.lbvserver` class. You will use this instance to perform operations on the lbvserver.

   ```
   lbvserver new_lbvserver_obj = new lbvserver();
   ```

7. Use the `lbvserver` instance to create a new lbvserver:

   ```
   new_lbvserver_obj.set_name("MyFirstLbVServer");
   new_lbvserver_obj.set_ipv46("10.102.29.88");
   new_lbvserver_obj.set_servicetype("HTTP");
   new_lbvserver_obj.set_port(88);
   new_lbvserver_obj.set_lbmethod("ROUNDROBIN");
   lbvserver.add(ns_session,new_lbvserver_obj);
   ```

   This code first sets the attributes (name, IP address, service type, port, and load balancing method) of the lbvserver locally and then adds it to the appliance by using the corresponding `add()` method.
8. Retrieve the details of the lbvserver you have created:

   ```
   new_lbvserver_obj = lbvserver.get(ns_session,new_lbvserver_obj.get_name());
   System.out.println("Name : " +new_lbvserver_obj.get_name() +"\n" +"Protocol : " +new_lbvserver_obj.ge
   ```

   This code first retrieves the details of the lbvserver as an object from the NetScaler, extracts the required attributes (name and service type) from the object, and displays the results.
9. Delete the lbvserver you created in the above steps:

```
lbvserver.delete(ns_session, new_lbvserver_obj.get_name());
```

10. Save the configurations:

```
ns_session.save_config();
```

11. Log out of the appliance:

```
ns_session.logout();
```

Debug the NITRO application

All NITRO exceptions are captured by the `com.citrix.netscaler.nitro.exception.nitro_exception` class. For a more detailed description, see Exception Handling.

Create a NetScaler Cluster

After completing this tutorial you will be able to create a two-node NetScaler cluster. To add more appliances to the cluster you must repeat the procedure that adds and joins the node to the cluster.

Sample Code

For the executable code, see the `<NITRO_SDK_HOME>/sample/CreateCluster.java` sample file.

To create a cluster

1. Copy the libraries from <NITRO_SDK_HOME>/lib folder to the project classpath.

2. Create a new class and name it CreateCluster.

3. Log on to one of the appliances that you want to add to the cluster and create a cluster:

   ```
   //Connect to the first appliance that you want to add to the cluster
   nitro_service nonClipSession0 = new nitro_service(nsipAddress0,protocol);
   nonClipSession0.login(uName,password);

   //Create a cluster instance
   clusterinstance newClusterInstance = new clusterinstance();
   newClusterInstance.set_clid(1);
   clusterinstance.add(nonClipSession0,newClusterInstance);

   //Add the appliance to the cluster
   clusternode ClusterNode0 = new clusternode();
   ClusterNode0.set_nodeid(0);
   ClusterNode0.set_ipaddress(nsipAddress0);
   ClusterNode0.set_state("ACTIVE");
   ClusterNode0.set_backplane("0/1/1");
   clusternode.add(nonClipSession0,ClusterNode0);

   //Add the cluster IP address
   nsip newNSIPAddress = new nsip();
   newNSIPAddress.set_ipaddress(clipAddress);
   newNSIPAddress.set_netmask("255.255.255.255");
   newNSIPAddress.set_type("CLIP");
   nsip.add(nonClipSession0,newNSIPAddress);

   //Enable the cluster instance
   clusterinstance.enable(nonClipSession0, newClusterInstance);

   //Save the configurations
   nonClipSession0.save_config();

   //Warm reboot the appliance
   nonClipSession0.reboot(true);
   ```

   The cluster is created and the first node is added to the cluster. This node becomes the initial configuration coordinator of the cluster.

4. Log on to the cluster IP address to add other appliances to the cluster:

   ```
   //Connect to the cluster IP address
   nitro_service clipSession = new nitro_service(clipAddress,protocol);
   clipSession.login(uName,password);

   //Add the node to the cluster
   clusternode ClusterNode1 = new clusternode();
   ClusterNode1.set_nodeid(1);
   ClusterNode1.set_ipaddress(nsipAddress1);
   ```

```
ClusterNode1.set_state("ACTIVE");
ClusterNode1.set_backplane("1/1/1");
clusternode.add(clipSession,ClusterNode1);

//Save the configurations
clipSession.save_config();
```

5. Log on to the appliance that you added in the previous step and join it to the cluster:

   ```
   //Connect to the node that you have just added to the cluster
   nitro_service nonClipSession1 = new nitro_service(nsipAddress1,protocol);
   nonClipSession1.login(uName,password);

   //Join the node to the cluster
   cluster newCluster = new cluster();
   newCluster.set_clip(clipAddress);
   newCluster.set_password(password);
   cluster.join(nonClipSession1,newCluster);

   //Save the configurations
   nonClipSession1.save_config();

   //Warm reboot the appliance
   nonClipSession1.reboot(true);
   ```

   The second node is now a part of the cluster.

6. Verify the details of the cluster by logging on to the cluster IP address

   ```
   //Retrieving the cluster node details
   Long id = new Long(1);
   clusternode node= clusternode.get(clipSession, id);
   System.out.println("Node ID: "+ node.get_nodeid() + " | Admin state: " + node.get_state() + " | Backplane

   //Retrieving the cluster instance details
   Long id1 = new Long(1);
   clusterinstance instance= clusterinstance.get(clipSession, id1);
   System.out.println("Cluster instance ID: "+ instance.get_clid() + " | Operational state: " +instance.get_op
   ```

System APIs

The first step towards using NITRO is to establish a session with the NetScaler appliance and then authenticate the session by using the NetScaler administrator's credentials.

You must create an object of the `com.citrix.netscaler.nitro.service.nitro_service` class by specifying the NetScaler IP (NSIP) address and the protocol to connect to the appliance (HTTP or HTTPS). You then use this object and log on to the appliance by specifying the user name and the password of the NetScaler administrator.

Note: You must have a user account on that appliance. The configuration operations that you perform are limited by the administrative roles assigned to your account.

The following sample code establishes a session with a NetScaler appliance with IP address 10.102.29.60 by using the HTTPS protocol:

```
//Specify the NetScaler appliance IP address and protocol
nitro_service ns_session = new nitro_service("10.102.29.60","https");

//Specify the login credentials
ns_session.login("admin","verysecret");
```

Note: When using HTTPS, you must make sure that the root CA is added to the truststore. By default, NITRO validates the SSL certificate and verifies the hostname. To disable this validation, use the following:

```
ns_session.set_certvalidation(false);
ns_session.set_hostnameverification(false);
```

Note: By default, the connection to the appliance expires after 30 minutes of inactivity. You can modify the timeout period by specifying a new timeout period (in seconds) in the `login` method. For example, to modify the timeout period to 60 minutes:

```
ns_session.login("admin","verysecret",3600);
```

You must use the `nitro_service` object in all further NITRO operations on the appliance. For example to save the configurations on the appliance, you must use the `nitro_service` object as follows:

```
ns_session.save_config();
```

The `nitro_service` class also provides APIs to perform other system-level operations such as enabling and disabling NetScaler features and modes, saving and clearing NetScaler configurations, setting the session timeout, setting the severity of the exceptions to be handled, setting the behavior of bulk operations, and disconnecting from the appliance.

Feature Configuration APIs

NetScaler resources are organized into a set of packages or namespaces. Each package or namespace corresponds to a NetScaler feature. For example, all load-balancing related resources, such as load balancing virtual server, load balancing group, and load balancing monitor are available in `com.citrix.netscaler.nitro.resource.config.lb`.

Similarly, all application firewall related resources, such as application firewall policy and application firewall archive are available in `com.citrix.netscaler.nitro.resource.config.appfw`.

Each NetScaler resource is represented by a class. For example, the class that represents a load balancing virtual server is called `lbvserver` (in `com.citrix.netscaler.nitro.resource.config.lb`). The state of a resource is represented by properties of a class. You can set the value for these properties by using the `set_<propertyname>()` methods provided by the resource class. For example to set the IP address of a load balancing virtual server, the lbvserver class provides the `set_ipv46()` method. Similarly, you can get the value of these properties by using the `get_<propertyname>()` methods of the resource class.

Note: The setter and getter properties are always executed locally on the client. They do not involve any network interaction with the NITRO web service. All properties have basic simple types: integer, long, boolean, and string.

A resource class provides APIs to perform the following operations:

Create | Retrieve | Update | Delete | Enable/Disable | Unset | Bind/Unbind | Global bind | Bulk operations

Create

To create a new resource, instantiate the resource class, configure the resource by setting its properties locally, and then upload the new resource instance to the NetScaler appliance.

The following sample code creates a load balancing virtual server:

```
//Create an instance of the lbvserver class
lbvserver new_lbvserver_obj = new lbvserver();

//Set the properties of the resource locally
new_lbvserver_obj.set_name("MyFirstLbVServer");
new_lbvserver_obj.set_ipv46("10.102.29.88");
new_lbvserver_obj.set_port(88);
new_lbvserver_obj.set_servicetype("HTTP");
new_lbvserver_obj.set_lbmethod("ROUNDROBIN");

//Upload the resource to NetScaler
lbvserver.add(ns_session,new_lbvserver_obj);
```

Retrieve

To retrieve the properties of a resource, you retrieve the resource object from the NetScaler appliance. Once the object is retrieved, you can extract the required properties of the resource locally, without further network traffic.

The following sample code retrieves the details of a load balancing virtual server:

```
//Retrieve the resource object from the NetScaler
new_lbvserver_obj = lbvserver.get(ns_session,"MyFirstLbVServer");

//Extract the properties of the resource from the object locally
System.out.println(new_lbvserver_obj.get_name());
System.out.println(new_lbvserver_obj.get_servicetype());
```

You can also retrieve resources by specifying a filter on the value of their properties by using the `com.citrix.netscaler.nitro.util.filtervalue` class.

For example, you can retrieve all the load balancing virtual servers that have their port set to 80 and servicetype to HTTP:

```
filtervalue[] filter = new filtervalue[2];
filter[0] = new filtervalue("port","80");
filter[1] = new filtervalue("servicetype","HTTP");
lbvserver[] result = lbvserver.get_filtered(ns_session,filter);
```

You can also retrieve all NetScaler resources of a certain type, such as all services in the NetScaler appliance, by calling the static `get()` method on the service class, without providing a second parameter, as follows:

```
service[] resources = service.get(ns_session);
```

Update

To update the properties of a resource, instantiate the resource class, specify the name of the resource to be updated, configure the resource by updating its properties locally, and then upload the updated resource instance to the NetScaler appliance.

The following sample code updates the service type and load balancing method of a load balancing virtual server:

```
//Create an instance of the lbvserver class
lbvserver update_lb = new lbvserver();

//Specify the name of the lbvserver to be updated
update_lb.set_name("MyFirstLbVServer");

//Specify the updated service type and lb method
update_lb.set_servicetype("https");
update_lb.set_lbmethod("LEASTRESPONSETIME");

//Upload the resource to NetScaler
lbvserver.update(ns_session,update_lb);
```

Note: Some properties in some NetScaler resources are not allowed to be modified after creation. The port number or the service type (protocol) of a load balancing virtual server or a service, are examples of such properties. Even though the update method appears to succeed, these properties retain their original values on the appliance.

Delete

To delete an existing resource, invoke the `delete()` method on the resource class, by passing the name of the resource.

The following sample code deletes a load balancing virtual server with name "MyFirstLbVServer":

```
lbvserver remove_lb = new lbvserver();
remove_lb.set_name("MyFirstLbVServer");
lbvserver.delete(ns_session, remove_lb);
```

Enable/Disable

To enable a resource, invoke the `enable()` method.

The following sample code enables a load balancing virtual server named "lb_vip":

```
lbvserver obj = new lbvserver();
obj.set_name = "lb_vip";
lbvserver.enable(ns_session, obj);
```

Note: To disable a resource, invoke the `disable()` method.

```
lbvserver.disable(ns_session,obj);
```

Unset

To unset the value that is set to a parameter, invoke the `unset()` method on the resource class, by passing the name of the resource and the parameters to be unset. If the parameter has a default value, the value is reset to that value.

The following sample code unsets the load balancing method and the comments of a load balancing virtual server named "lb_123":

```
lbvserver lb1 = new lbvserver();
lb1.set_name("lb_123");
String args[] = {"comment", "lbmethod"};
lbvserver.unset(ns_session, lb1, args);
```

Bind/Unbind

NetScaler resources form relationships with each other through the process of binding. This is how services are associated with a load balancing virtual server (by binding them to it), or how various policies are bound to a load balancing virtual server. Each binding relationship is represented in NITRO by its own class.

To bind one NetScaler resource to another, you must instantiate the appropriate binding class (for example, to bind a service to a load balancing virtual server, you must instantiate the `lbvserver_service_binding` class) and add it to the NetScaler configuration (by using the static `add()` method on this class).

Binding classes have a property representing the name of each resource in the binding relationship. They can also have other properties related to that relationship (for example, the weight of the binding between a load balancing virtual server and a service).

The following sample code binds a service to a load balancing virtual server, by specifying a certain weight for the binding:

```
lbvserver_service_binding bindObj = new lbvserver_service_binding();
bindObj.set_name("MyFirstLbVServer");
bindObj.set_servicename("svc_prod");
bindObj.set_weight(20);
lbvserver_service_binding.add(ns_session,bindObj);
```

Note: To unbind a resource from another, invoke the `delete()` method from the resource binding class, by passing the name of the two resources.

The following code sample unbinds a service from a server:

```
lbvserver_service_binding bindObj = new lbvserver_service_binding();
bindObj.set_name("MyFirstLbVServer");
bindObj.set_servicename("svc_prod");
lbvserver_service_binding.delete(ns_session,bindObj);
```

Global bind

Some NetScaler resources can be bound globally to affect the whole system. For example, a compression policy can be bound to an load balancing virtual server, in which case the policy affects only the traffic on that load balancing virtual server. However, if bound globally, it can affect any traffic on the appliance, regardless of which virtual servers handle the traffic.

Some NITRO classes can be used to bind resources globally. These classes have names that follow the following pattern: `<featurename>global_<resourcetype>_binding`.

For example, the class `aaaglobal_preauthenticationpolicy_binding` is used to bind preauthentication policies globally.

The following sample code creates a preauthentication action and a preauthentication policy that uses that action, and then binds the policy globally at priority 200:

```
aaapreauthenticationaction preauth_act1;
aaapreauthenticationpolicy preauth_pol1;
aaaglobal_aaapreauthenticationpolicy_binding glob_binding;
preauth_act1 = new aaapreauthenticationaction();
preauth_act1.set_name("preauth_act1");
preauth_act1.set_preauthenticationaction("ALLOW");
aaapreauthenticationaction.add(ns_session,preauth_act1);
```

```
preauth_pol1 = new aaapreauthenticationpolicy();
preauth_pol1.set_name("preauth_pol1");
preauth_pol1.set_rule("CLIENT.APPLICATION.PROCESS(antivirus.exe) EXISTS");
preauth_pol1.set_reqaction("preauth_act1");
aaapreauthenticationpolicy.add(ns_session,preauth_pol1);

glob_binding = new aaaglobal_aaapreauthenticationpolicy_binding();
glob_binding.set_policy("preauth_pol1");
glob_binding.set_priority(200);
aaaglobal_aaapreauthenticationpolicy_binding.add(ns_session,glob_binding);
```

Bulk operations

You can create, retrieve, update, and delete multiple resources simultaneously and thus minimize network traffic. For example, you can add multiple load balancing virtual servers in the same operation. To perform a bulk operation, you instantiate an array of the resource class, configure the properties of all the instances locally, and then upload all the instances to the NetScaler with one command.

To account for the failure of some operations within the bulk operation, NITRO allows you to configure one of the following behaviors:

- **Exit.** When the first error is encountered, the execution stops. The commands that were executed before the error are committed.
- **Rollback.** When the first error is encountered, the execution stops. The commands that were executed before the error are rolled back. Rollback is only supported for add and bind commands.
- **Continue.** All the commands in the list are executed even if some commands fail.

Note: You must configure the required behavior while establishing a connection with the appliance.

```
nitro_service ns_session = new nitro_service("10.102.29.60","http");
ns_session.set_onerror(OnerrorEnum.CONTINUE);
ns_session.login("admin","verysecret");
```

The following sample code creates two load balancing virtual servers:

```
//Create an array of lbvserver instances
lbvserver[] lbs = new lbvserver[2];

//Specify properties of the first lbvserver
lbs[0] = new lbvserver();
lbs[0].set_name("lbvserv1");
lbs[0].set_servicetype("http");
lbs[0].set_ipv46("10.70.136.5");
lbs[0].set_port(80);

//Specify properties of the second lbvserver
lbs[1] = new lbvserver();
lbs[1].set_name("lbvserv2");
```

```
lbs[1].set_servicetype("https");
lbs[1].set_ipv46("10.70.136.5");
lbs[1].set_port(443);

//Upload the properties of the two lbvservers to the NetScaler
lbvserver.add(ns_session,lbs);
```

Cluster APIs

For managing clusters, you can add or remove a cluster instance or an individual node and perform a few other instance or node operations such as viewing instance or node properties. You can also configure the cluster IP address. Other cluster-management tasks include joining a NetScaler appliance to the cluster and configuring a linkset.

Cluster Instance Operations

The `com.citrix.netscaler.nitro.resource.config.cluster.clusterinstance` class provides APIs to manage a cluster instance.

The following sample code creates a cluster instance with ID 1:

```
clusterinstance new_cl_inst_obj = new clusterinstance();
//Set the properties of the cluster instance locally
new_cl_inst_obj.set_clid(1);
new_cl_inst_obj.set_preemption("ENABLED");

//Upload the cluster instance
clusterinstance.add(ns_session,new_cl_inst_obj);
```

Cluster Node Operations

The `com.citrix.netscaler.nitro.resource.config.cluster.clusternode` class provides APIs to manage cluster nodes.

The following sample code adds a cluster node with NSIP address 10.102.29.60:

```
clusternode new_cl_node_obj = new clusternode();
//Set the properties of the cluster node locally
new_cl_node_obj.set_nodeid(0);
new_cl_node_obj.set_ipaddress("10.102.29.60");
new_cl_node_obj.set_state("ACTIVE");
new_cl_node_obj.set_backplane("0/1/1");

//Upload the cluster node
clusternode.add(ns_session,new_cl_node_obj);
```

Add a Cluster IP Address

The `com.citrix.netscaler.nitro.resource.config.ns.nsip` class provides the `add()` API to configure an IP address. To configure the IP address as a cluster IP address, you must specify the type as CLIP.

The following sample code configures a cluster IP address on NetScaler appliance with IP address 10.102.29.60:

```
nsip new_nsip_obj = new nsip();
//Set the properties locally
new_nsip_obj.set_ipaddress("10.102.29.61");
new_nsip_obj.set_netmask("255.255.255.255");
new_nsip_obj.set_type("CLIP");

//Upload the cluster node
nsip.add(ns_session,new_nsip_obj);
```

Add a Spotted IP Address

The `com.citrix.netscaler.nitro.resource.config.ns.nsip` class provides the `add()` API to configure an IP address. To configure the IP address as spotted, you must specify the ID of the node that must own the IP address. This configuration must be done on the cluster IP address.

The following sample code configures a spotted SNIP address on a node with ID 1:

```
nsip new_nsip_obj = new nsip();
//Set the properties locally
new_nsip_obj.set_ipaddress("10.102.29.77");
new_nsip_obj.set_netmask("255.255.255.0");
new_nsip_obj.set_type("SNIP");
new_nsip_obj.set_ownernode(1);

//Upload the cluster node
nsip.add(ns_session,new_nsip_obj);
```

Join NetScaler Appliance to Cluster

The `com.citrix.netscaler.nitro.resource.config.cluster.cluster` class provides the `join()` API to join a NetScaler appliance to the cluster. You must specify the cluster IP address and the nsroot password of the configuration coordinator.

The following sample joins a NetScaler appliance to a cluster:

```
cluster new_cl_obj = new cluster();
//Set the properties of the cluster  locally
new_cl_obj.set_clip("10.102.29.61");
new_cl_obj.set_password("verysecret");

//Upload the cluster
cluster.add(ns_session,new_cl_obj);
```

Linkset Operations

The `com.citrix.netscaler.nitro.resource.config.network.linkset` class provides the APIs to manage linksets.

To configure a linkset, do the following:

1. Add a linkset by invoking the `add()` method of the `linkset` class.

2. Bind the interfaces to the linkset using the `add()` method of the `linkset_interface_binding` class.

The following sample code creates a linkset LS/1 and bind interfaces 1/1/2 and 2/1/2 to it:

```
//Create the linkset
linkset new_linkset_obj = new linkset();
new_linkset_obj.set_id("LS/1");
linkset.add(ns_session,new_linkset_obj);

//Bind the interfaces to the linkset
linkset_interface_binding new_linkif_obj = new linkset_interface_binding();
new_linkif_obj.set_id("LS/1");
new_linkif_obj.set_ifnum("1/1/2 2/1/2");
linkset_interface_binding.add(ns_session,new_linkif_obj);
```

Feature Statistics APIs

The NetScaler appliance collects statistics about the usage of its features and the corresponding resources. You can retrieve these statistics by using NITRO API. The statistics APIs are available in different packages from the configuration APIs.

The APIs to retrieve statistics of NetScaler features are available in packages that have the following pattern: `com.citrix.netscaler.nitro.resource.stat.<feature>`.

For example, APIs to retrieve statistics of the load balancing virtual server are available in the `com.citrix.netscaler.nitro.resource.stat.lb` package.

The following sample code retrieves the statistics of a load balancing virtual server and displays some of the statistics returned:

```
lbvserver_stats stats = lbvserver_stats.get(ns_session,"MyFirstLbVServer");
System.out.println(stats.get_curclntconnections());
System.out.println(stats.get_deferredreqrate());
```

Note: Not all NetScaler features and resources have statistic objects associated with them.

AppExpert Application APIs

To export an AppExpert application, you must instantiate the `com.citrix.netscaler.nitro.resource.config.app.application` class, configure the properties of the AppExpert locally, and then export the AppExpert application.

The following sample code exports an AppExpert application named "MyApp1":

```
application myapp = new application();
myapp.set_appname("MyApp1");
myapp.set_apptemplatefilename("myapp_template");
application.export(ns_session,myapp);
```

You can also import an AppExpert application. You must instantiate the `com.citrix.netscaler.nitro.resource.config.app.application` class, configure the properties of the AppExpert locally, and then import the AppExpert application.

The following sample code imports an AppExpert application named "MyApp1":

```
application myapp = new application();
myapp.set_appname("MyApp1");
myapp.set_apptemplatefilename("myapp_template");
application.Import(ns_session,myapp);
```

Exception Handling

The status of a NITRO request is captured in the `com.citrix.netscaler.nitro.exception.nitro_exception` class. This class provides the following details of the exception:

- **Session ID.** The session in which the exception occurred.
- **Severity.** The severity of the exception: error or warning. By default, only errors are captured. To capture warnings, you must set the warning flag to true, while connecting to the appliance.
- **Error code.** The status of the NITRO request. An error code of 0 indicates that the NITRO request is successful. A non-zero error code indicates an error in processing the NITRO request.
- **Error message.** Provides a brief description of the exception.

For a list of error codes, see the errorlisting.html file available in the <NITRO_SDK_HOME>/doc/api_reference folder.

.NET API

NetScaler NITRO APIs are categorized depending on the scope and purpose of the APIs into system APIs, feature configuration APIs, and feature statistics APIs. Additionally, you can import and export AppExpert applications. You can also troubleshoot NITRO operations.

Tutorials

These tutorials demonstrate the end-to-end usage of NITRO to achieve the following:

- Create Your First NITRO Application
- Create a NetScaler Cluster

Create Your First NITRO Application

After completing this tutorial, you will understand and be able to perform the following tasks:

- Integrate NITRO with the IDE
- Log in to the appliance
- Create a load balancing virtual server (lbvserver)
- Retrieve details of an lbvserver
- Delete an lbvserver
- Save the configurations on the appliance
- Log out of the appliance
- Debug the NITRO application

Before you begin, make sure that you have the latest NITRO SDK and that the client application satisfies the prerequisites for using the NITRO SDK.

Sample Code

For the executable code, see the <NITRO_SDK_HOME>/sample/MyFirstNitroApplication.cs sample file.

To create your first NITRO application:

1. Copy the libraries from <NITRO_SDK_HOME>/lib folder to the project classpath.

2. Create a new class and name it **MyFirstNitroApplication**.

3. Create an instance of `com.citrix.netscaler.nitro.service.nitro_service` class. This instance is used to perform all operations on the appliance:

   ```
   nitro_service ns_session = new nitro_service("10.102.29.170", "http");
   ```

   This code establishes a connection with an appliance that has IP address 10.102.29.170 and uses the HTTP protocol. Replace 10.102.29.170 with the IP address of the NetScaler appliance that you have access to.
4. Use the `nitro_service` instance to log in to the appliance using your credentials:

   ```
   ns_session.login("admin","verysecret");
   ```

   This code logs into the appliance, with user name as `admin` and password as `verysecret`. Replace the credentials with your login credentials.
5. Enable the load balancing feature:

   ```
   String[] features_to_be_enabled = {"lb"};
   ns_session.enable_features(features_to_be_enabled);
   ```

   This code enables load balancing on the appliance.
6. Create an instance of the `com.citrix.netscaler.nitro.resource.config.lb.lbvserver` class. You will use this instance to perform operations on the lbvserver.

   ```
   lbvserver new_lbvserver_obj = new lbvserver();
   ```

7. Use the `lbvserver` instance to create a new lbvserver:

   ```
   new_lbvserver_obj.name = "MyFirstLbVServer";
   new_lbvserver_obj.ipv46 = "10.102.29.88";
   new_lbvserver_obj.servicetype = "HTTP";
   new_lbvserver_obj.port = 80;
   new_lbvserver_obj.lbmethod = "ROUNDROBIN";
   lbvserver.add(ns_session,new_lbvserver_obj);
   ```

   This code first sets the attributes (name, IP address, service type, port, and load balancing method) of the lbvserver locally and then adds it to the appliance by using the corresponding `add()` method.
8. Retrieve the details of the lbvserver you have created:

   ```
   lbvserver new_lbvserver_obj1 = lbvserver.get(ns_session,new_lbvserver_obj.name);
   System.Console.Out.WriteLine("Name : " +new_lbvserver_obj1.name +"\n" +"Protocol : " +new_lbvserver_
   ```

   This code first retrieves the details of the lbvserver as an object from the NetScaler, extracts the required attributes (name and service type) from the object, and displays the results.
9. Delete the lbvserver you created in the above steps:

   ```
   lbvserver.delete(ns_session, new_lbvserver_obj.name);
   ```

10. Save the configurations:

    ```
    ns_session.save_config();
    ```

11. Log out of the appliance:

    ```
    ns_session.logout();
    ```

Debug the NITRO application

All NITRO exceptions are captured by the `com.citrix.netscaler.nitro.exception.nitro_exception` class. For a more detailed description, see Exception Handling.

Create a NetScaler Cluster

After completing this tutorial you will be able to create a two-node NetScaler cluster. To add more appliances to the cluster you must repeat the procedure that adds and joins the node to the cluster.

Sample Code

For the executable code, see the <NITRO_SDK_HOME>/sample/CreateCluster.cs sample file.

To create a cluster

1. Copy the libraries from <NITRO_SDK_HOME>/lib folder to the project classpath.

2. Create a new class and name it CreateCluster.

3. Log on to one of the appliances that you want to add to the cluster and create a cluster:

   ```
   //Connect to the first appliance that you want to add to the cluster
   nitro_service nonClipSession0 = new nitro_service(nsipAddress0,protocol);
   nonClipSession0.login(uName,password);

   //Create a cluster instance
   clusterinstance newClusterInstance = new clusterinstance();
   newClusterInstance.clid = 1;
   clusterinstance.add(nonClipSession0,newClusterInstance);

   //Add the appliance to the cluster
   clusternode ClusterNode0 = new clusternode();
   ClusterNode0.nodeid = 0;
   ClusterNode0.ipaddress = nsipAddress0;
   ClusterNode0.state = "ACTIVE";
   ClusterNode0.backplane = "0/1/1";
   clusternode.add(nonClipSession0,ClusterNode0);

   //Add the cluster IP address
   nsip newNSIPAddress = new nsip();
   newNSIPAddress.ipaddress = clipAddress;
   newNSIPAddress.netmask = "255.255.255.255";
   newNSIPAddress.type = "CLIP";
   nsip.add(nonClipSession0,newNSIPAddress);

   //Enable the cluster instance
   clusterinstance.enable(nonClipSession0, newClusterInstance);

   //Save the configurations
   nonClipSession0.save_config();

   //Warm reboot the appliance
   nonClipSession0.reboot(true);
   ```

   The cluster is created and the first node is added to the cluster. This node becomes the initial configuration coordinator of the cluster.

4. Log on to the cluster IP address to add other appliances to the cluster:

   ```
   //Connect to the cluster IP address
   nitro_service clipSession = new nitro_service(clipAddress,protocol);
   clipSession.login(uName,password);

   //Add the node to the cluster
   clusternode ClusterNode1 = new clusternode();
   ClusterNode1.nodeid = 1;
   ClusterNode1.ipaddress = nsipAddress1;
   ```

```
ClusterNode1.state = "ACTIVE";
ClusterNode1.backplane = "1/1/1";
clusternode.add(clipSession,ClusterNode1);

//Save the configurations
clipSession.save_config();
```

5. Log on to the appliance that you added in the previous step and join it to the cluster:

```
//Connect to the node that you have just added to the cluster
nitro_service nonClipSession1 = new nitro_service(nsipAddress1,protocol);
nonClipSession1.login(uName,password);

//Join the node to the cluster
cluster newCluster = new cluster();
newCluster.clip = clipAddress;
newCluster.password = password;
cluster.join(nonClipSession1,newCluster);

//Save the configurations
nonClipSession1.save_config();

//Warm reboot the appliance
nonClipSession1.reboot(true);
```

The second node is now a part of the cluster.

6. Verify the details of the cluster by logging on to the cluster IP address

```
//Retrieving the cluster node details
uint id = 1;
clusternode node= clusternode.get(clipSession, id);
System.Console.Out.WriteLine("Node ID: " + node.nodeid + " | Admin state: " + node.state + " | Backplan

//Retrieving the cluster instance details
uint id1 = 1;
clusterinstance instance= clusterinstance.get(clipSession, id1);
System.Console.Out.WriteLine("Cluster instance ID: "+ instance.clid + " | Operational state: " +instance.c
```

System APIs

The first step towards using NITRO is to establish a session with the NetScaler appliance and then authenticate the session by using the NetScaler administrator's credentials.

You must create an object of the `com.citrix.netscaler.nitro.service.nitro_service` class by specifying the NetScaler IP (NSIP) address and the protocol to connect to the appliance (HTTP or HTTPS). You then use this object and log on to the appliance by specifying the user name and the password of the NetScaler administrator.

Note: You must have a user account on that appliance. The configuration operations that you perform are limited by the administrative roles assigned to your account.

The following sample code establishes a session with a NetScaler appliance with IP address 10.102.29.60 by using the HTTPS protocol:

```
//Specify the NetScaler appliance IP address and protocol
nitro_service ns_session = new nitro_service("10.102.29.60","https");

//Specify the login credentials
ns_session.login("admin","verysecret");
```

Note: By default, the connection to the appliance expires after 30 minutes of inactivity. You can modify the timeout period by specifying a new timeout period (in seconds) in the `login` method. For example, to modify the timeout period to 60 minutes:

```
ns_session.login("admin","verysecret",3600);
```

You must use the `nitro_service` object in all further NITRO operations on the appliance. For example to save the configurations on the appliance, you must use the `nitro_service` object as follows:

```
ns_session.save_config();
```

The `nitro_service` class also provides APIs to perform other system-level operations such as enabling and disabling NetScaler features and modes, saving and clearing NetScaler configurations, setting the session timeout, setting the severity of the exceptions to be handled, setting the behavior of bulk operations, and disconnecting from the appliance.

Feature Configuration APIs

NetScaler resources are organized into a set of packages or namespaces. Each package or namespace corresponds to a NetScaler feature. For example, all load-balancing related resources, such as load balancing virtual server, load balancing group, and load balancing monitor are available in `com.citrix.netscaler.nitro.resource.config.lb`.

Similarly, all application firewall related resources, such as application firewall policy and application firewall archive are available in `com.citrix.netscaler.nitro.resource.config.appfw`.

Each NetScaler resource is represented by a class. For example, the class that represents a load balancing virtual server is called `lbvserver` (in `com.citrix.netscaler.nitro.resource.config.lb`). The state of a resource is represented by properties of a class. You can get and set the properties of the class.

Note: The setter and getter properties are always executed locally on the client. They do not involve any network interaction with the NITRO web service. All properties have basic simple types: integer, long, boolean, and string.

A resource class provides APIs to perform the following operations:

Create | Retrieve | Update | Delete | Enable/Disable | Unset | Bind/Unbind | Global bind | Bulk operations

Create

To create a new resource, instantiate the resource class, configure the resource by setting its properties locally, and then upload the new resource instance to the NetScaler appliance.

The following sample code creates a load balancing virtual server:

```
//Create an instance of the lbvserver class
lbvserver new_lbvserver_obj = new lbvserver();

//Set the properties of the resource locally
new_lbvserver_obj.name = "MyFirstLbVServer";
new_lbvserver_obj.ipv46 = "10.102.29.88";
new_lbvserver_obj.port = 88;
new_lbvserver_obj.servicetype = "HTTP";
new_lbvserver_obj.lbmethod = "ROUNDROBIN";

//Upload the resource to NetScaler
lbvserver.add(ns_session,new_lbvserver_obj);
```

Retrieve

To retrieve the properties of a resource, retrieve the resource object from the NetScaler appliance. Once the object is retrieved, you can extract the required properties of the resource locally, without incurring further network traffic.

The following sample code retrieves the details of a load balancing virtual server:

```
//Retrieve the resource object from the NetScaler
new_lbvserver_obj = lbvserver.get(ns_session,"MyFirstLbVServer");

//Extract the properties of the resource from the object locally
Console.WriteLine(new_lbvserver_obj.name);
Console.WriteLine(new_lbvserver_obj.servicetype);
```

You can also retrieve resources by specifying a filter on the value of their properties by using the `com.citrix.netscaler.nitro.util.filtervalue` class.

For example, you can retrieve all the load balancing virtual servers that have their port set to 80 and servicetype to HTTP:

```
filtervalue[] filter = new filtervalue[2];
filter[0] = new filtervalue("port","80");
filter[1] = new filtervalue("servicetype","HTTP");
lbvserver[] result = lbvserver.get_filtered(ns_session,filter);
```

You can also retrieve all NetScaler resources of a certain type, such as all services in the NetScaler appliance, by calling the static `get()` method on the service class, without providing a second parameter, as follows:

```
service[] resources = service.get(ns_session);
```

Update

To update the properties of a resource, instantiate the resource class, specify the name of the resource to be updated, configure the resource by updating its properties locally, and then upload the updated resource instance to the NetScaler appliance.

The following sample code updates the service type and load balancing method of a load balancing virtual server:

```
//Create an instance of the lbvserver class
lbvserver update_lb = new lbvserver();

//Specify the name of the lbvserver to be updated
update_lb.name = "MyFirstLbVServer";

//Specify the updated service type and lb method
update_lb.servicetype = "https";
update_lb.lbmethod = "LEASTRESPONSETIME";

//Upload the resource to NetScaler
lbvserver.update(ns_session, update_lb);
```

Note: Some properties in some NetScaler resources are not allowed to be modified after creation. The port number or the service type (protocol) of a load balancing virtual server or a service, are examples of such properties. Even though the update method appears to succeed, these properties retain their original values on the appliance.

Delete

To delete an existing resource, invoke the static method delete() on the resource class, by passing the name of the resource.

The following sample code deletes a load balancing virtual server with name "MyFirstLbVServer":

```
lbvserver remove_lb = new lbvserver();
remove_lb.name("MyFirstLbVServer");
lbvserver.delete(ns_session, remove_lb);
```

Enable/Disable

To enable a resource, invoke the `enable()` method.

The following sample code enables a load balancing virtual server named "lb_vip":

```
lbvserver obj = new lbvserver();
obj.name = "lb_vip";
lbvserver.enable(ns_session, obj);
```

Note: To disable a resource, invoke the `disable()` method.

```
lbvserver.disable(ns_session, obj);
```

Unset

To unset the value that is set to a parameter, invoke the `unset()` method on the resource class, by passing the name of the resource and the parameters to be unset. If the parameter has a default value, the value is reset to that value.

The following sample code unsets the load balancing method and the comments of a load balancing virtual server named "lb_123":

```
lbvserver obj = new lbvserver();
obj.name = "lb_123";
String[] args = { "lbmethod","comment" };
lbvserver.unset(ns_session, lb1, args);
```

Bind/Unbind

NetScaler resources form relationships with each other through the process of binding. This is how services are associated with a load balancing virtual server (by binding them to it), or how various policies are bound to a load balancing virtual server. Each binding relationship is represented in NITRO by its own class.

To bind one NetScaler resource to another, you must instantiate the appropriate binding class (for example, to bind a service to a load balancing virtual server, you must instantiate the `lbvserver_service_binding` class) and add it to the NetScaler configuration (by using the static `add()` method on this class).

Binding classes have a property representing the name of each resource in the binding relationship. They can also have other properties related to that relationship (for example, the weight of the binding between a load balancing virtual server and a service).

The following sample code binds a service to a load balancing virtual server, by specifying a certain weight for the binding:

```
lbvserver_service_binding bindObj = new lbvserver_service_binding();
bindObj.name = "MyFirstLbVServer";
bindObj.servicename = "svc_prod";
bindObj.weight = 20;
lbvserver_service_binding.add(ns_session,bindObj);
```

Note: To unbind a resource from another, invoke the `delete()` method from the resource binding class, by passing the name of the two resources.

The following code sample unbinds a service from a server:

```
lbvserver_service_binding bindObj = new lbvserver_service_binding();
bindObj.name("MyFirstLbVServer");
bindObj.servicename("svc_prod");
lbvserver_service_binding.delete(ns_session,bindObj);
```

Global bind

Some NetScaler resources can be bound globally to affect the whole system. For example, a compression policy can be bound to an load balancing virtual server, in which case the policy affects only the traffic on that load balancing virtual server. However, if bound globally, it can affect any traffic on the appliance, regardless of which virtual servers handle the traffic.

Some NITRO classes can be used to bind resources globally. These classes have names that follow the following pattern: `<featurename>global_<resourcetype>_binding`.

For example, the class `aaaglobal_preauthenticationpolicy_binding` is used to bind preauthentication policies globally.

The following sample code creates a preauthentication action and a preauthentication policy that uses that action, and then binds the policy globally at priority 200:

```
aaapreauthenticationaction preauth_act1;
aaapreauthenticationpolicy preauth_pol1;
aaaglobal_aaapreauthenticationpolicy_binding glob_binding;
preauth_act1 = new aaapreauthenticationaction();
preauth_act1.name = "preauth_act1";
preauth_act1.preauthenticationaction = "ALLOW";
aaapreauthenticationaction.add(ns_session, preauth_act1);

preauth_pol1 = new aaapreauthenticationpolicy();
preauth_pol1.name = "preauth_pol1";
```

```
preauth_pol1.rule = "CLIENT.APPLICATION.PROCESS(antivirus.exe) EXISTS";
preauth_pol1.reqaction = "preauth_act1";
aaapreauthenticationpolicy.add(ns_session, preauth_pol1);

glob_binding = new aaaglobal_aaapreauthenticationpolicy_binding();
glob_binding.policy = "preauth_pol1";
glob_binding.priority = 200;
aaaglobal_aaapreauthenticationpolicy_binding.add(ns_session,glob_binding);
```

Bulk operations

You can create, retrieve, update, and delete multiple resources simultaneously and thus minimize network traffic. For example, you can add multiple load balancing virtual servers in the same operation. To perform a bulk operation, you instantiate an array of the resource class, configure the properties of all the instances locally, and then upload all the instances to the NetScaler with one command.

To account for the failure of some operations within the bulk operation, NITRO allows you to configure one of the following behaviors:

- **Exit.** When the first error is encountered, the execution stops. The commands that were executed before the error are committed.
- **Rollback.** When the first error is encountered, the execution stops. The commands that were executed before the error are rolled back. Rollback is only supported for add and bind commands.
- **Continue.** All the commands in the list are executed even if some commands fail.

Note: You must configure the required behavior while establishing a connection with the appliance.

```
nitro_service ns_session = new nitro_service("10.102.29.60","http");
ns_session.onerror = OnerrorEnum.CONTINUE;
ns_session.login("admin","verysecret");
```

The following sample code creates two load balancing virtual servers:

```
//Create an array of lbvserver instances
lbvserver[] lbs = new lbvserver[2];

//Specify details of first lbvserver
lbs[0] = new lbvserver();
lbs[0].name = "lbvserv1";
lbs[0].servicetype = "http";
lbs[0].ipv46 = "10.70.136.5";
lbs[0].port = 80;

//Specify details of second lbvserver
lbs[1] = new lbvserver();
lbs[1].name = "lbvserv2";
lbs[1].servicetype = "https";
lbs[1].ipv46 = "10.70.136.5";
lbs[1].port = 443;
```

```
//upload the details of the lbvservers to the NITRO server
lbvserver.add(ns_session,lbs);
```

Cluster APIs

For managing clusters, you can add or remove a cluster instance or an individual node and perform a few other instance or node operations such as viewing instance or node properties. You can also configure the cluster IP address. Other cluster-management tasks include joining a NetScaler appliance to the cluster and configuring a linkset.

Cluster Instance Operations

The `com.citrix.netscaler.nitro.resource.config.cluster.clusterinstance` class provides APIs to manage a cluster instance.

The following sample code creates a cluster instance with ID 1:

```
clusterinstance new_cl_inst_obj = new clusterinstance();
//Set the properties of the cluster instance locally
new_cl_inst_obj.clid = 1;
new_cl_inst_obj.preemption = "ENABLED";

//Upload the cluster instance
clusterinstance.add(ns_session,new_cl_inst_obj);
```

Cluster Node Operations

The `com.citrix.netscaler.nitro.resource.config.cluster.clusternode` class provides APIs to manage cluster nodes.

The following sample code adds a cluster node with NSIP address 10.102.29.60:

```
clusternode new_cl_node_obj = new clusternode();
//Set the properties of the cluster node locally
new_cl_node_obj.nodeid = 0;
new_cl_node_obj.ipaddress = "10.102.29.60";
new_cl_node_obj.state = "ACTIVE";
new_cl_node_obj.backplane = "0/1/1";

//Upload the cluster node
clusternode.add(ns_session,new_cl_node_obj);
```

Add a Cluster IP Address

The `com.citrix.netscaler.nitro.resource.config.ns.nsip` class provides the `add()` API to configure an IP address. To configure the IP address as a cluster IP address, you must specify the type as CLIP.

The following sample code configures a cluster IP address on NetScaler appliance with IP address 10.102.29.60:

```
nsip new_nsip_obj = new nsip();
//Set the properties locally
new_nsip_obj.ipaddress = "10.102.29.61";
new_nsip_obj.netmask = "255.255.255.255";
new_nsip_obj.type = "CLIP";

//Upload the cluster node
nsip.add(ns_session,new_nsip_obj);
```

Add a Spotted IP Address

The `com.citrix.netscaler.nitro.resource.config.ns.nsip` class provides the `add()` API to configure an IP address. To configure the IP address as spotted, you must specify the ID of the node that must own the IP address. This configuration must be done on the cluster IP address.

The following sample code configures a spotted SNIP address on a node with ID 1:

```
nsip new_nsip_obj = new nsip();
//Set the properties locally
new_nsip_obj.ipaddress = "10.102.29.77";
new_nsip_obj.netmask = "255.255.255.0";
new_nsip_obj.type = "SNIP";
new_nsip_obj.ownernode = 1;

//Upload the cluster node
nsip.add(ns_session,new_nsip_obj);
```

Join NetScaler Appliance to Cluster

The `com.citrix.netscaler.nitro.resource.config.cluster.cluster` class provides the `join()` API to join a NetScaler appliance to the cluster. You must specify the cluster IP address and the nsroot password of the configuration coordinator.

The following sample code joins a NetScaler appliance to a cluster:

```
cluster new_cl_obj = new cluster();
//Set the properties of the cluster locally
new_cl_obj.clip = "10.102.29.61";
new_cl_obj.password = "verysecret";

//Upload the cluster node
cluster.add(ns_session,new_cl_node_obj);
```

Linkset Operations

The `com.citrix.netscaler.nitro.resource.config.network.linkset` class provides the APIs to manage linksets.

To configure a linkset, do the following:

1. Add a linkset by invoking the `add()` method of the `linkset` class.

2. Bind the interfaces to the linkset using the `add()` method of the `linkset_interface_binding` class.

The following sample code creates a linkset LS/1 and bind interfaces 1/1/2 and 2/1/2 to it:

```
//Create the linkset
linkset new_linkset_obj = new linkset();
new_linkset_obj.id = "LS/1";
linkset.add(ns_session,new_linkset_obj);

//Bind the interfaces to the linkset
linkset_interface_binding new_linkif_obj = new linkset_interface_binding();
new_linkif_obj.id = "LS/1";
new_linkif_obj.ifnum = "1/1/2 2/1/2";
linkset_interface_binding.add(ns_session,new_linkif_obj);
```

Feature Statistics APIs

The NetScaler appliance collects statistics about the usage of its features and the corresponding resources. You can retrieve these statistics by using NITRO API. The statistics APIs are available in different namespaces from the configuration APIs.

The APIs to retrieve statistics of NetScaler features are available in namespaces that have the following pattern: `com.citrix.netscaler.nitro.resource.stat.<feature>`.

For example, APIs to retrieve statistics of the load balancing virtual server are available in the `com.citrix.netscaler.nitro.resource.stat.lb` namespace.

The following sample code retrieves the statistics of a load balancing virtual server and displays some of the statistics returned:

```
lbvserver_stats stats = lbvserver_stats.get(ns_session,"MyFirstLbVServer");
Console.WriteLine(stats.curclntconnections);
Console.WriteLine(stats.deferredreqrate);
```

Note: Not all NetScaler features and resources have statistic objects associated with them.

AppExpert Application APIs

To export an AppExpert application, you must instantiate the `com.citrix.netscaler.nitro.resource.config.app.application` class, configure the properties of the AppExpert locally, and then export the AppExpert application.

The following sample code exports an AppExpert application named "MyApp1":

```
application myapp = new application();
myapp.appname = "MyApp1";
myapp.apptemplatefilename = "myapp_template";
application.export(ns_session,myapp);
```

You can also import an AppExpert application. You must instantiate the `com.citrix.netscaler.nitro.resource.config.app.application` class, configure the properties of the AppExpert locally, and then import the AppExpert application.

The following sample code imports an AppExpert application named "MyApp1":

```
application myapp = new application();
myapp.appname = "MyApp1";
myapp.apptemplatefilename = "myapp_template";
application.Import(ns_session,myapp);
```

Exception Handling

The status of a NITRO request is captured in the `com.citrix.netscaler.nitro.exception.nitro_exception` class. This class provides the following details of the exception:

- **Session ID.** The session in which the exception occurred.
- **Severity.** The severity of the exception: error or warning. By default, only errors are captured. To capture warnings, you must set the warning flag to true, while connecting to the appliance.
- **Error code.** The status of the NITRO request. An error code of 0 indicates that the NITRO request is successful. A non-zero error code indicates an error in processing the NITRO request.
- **Error message.** Provides a brief description of the exception.

For a list of error codes, see the errorlisting.html file available in the <NITRO_SDK_HOME>/doc/api_reference folder.

REST Web Services

REST (REpresentational State Transfer) is an architectural style based on simple HTTP requests and responses between the client and the server. REST is used to query or change the state of objects on the server side. In REST, the server side is modeled as a set of entities where each entity is identified by a unique URL. For example, the load balancing virtual server entity is identified by the URL http://<NSIP>/nitro/v1/config/<lbvserver>/<lbvserver_name>.

Each resource also has a state on which the following operations can be performed:

- **Create.** Clients can create new server-side resources on a "container" resource. You can think of container resources as folders, and child resources as files or subfolders. The calling client provides the state for the resource to be created. The state can be specified in the request by using XML or JSON format. The client can also specify the unique URL that will identify the new object. Alternatively, the server can choose and return a unique URL identifying the created object. The HTTP method used for Create requests is POST.

- **Read.** Clients can retrieve the state of a resource by specifying its URL with the HTTP GET method. The response message contains the resource state, expressed in JSON format.

- **Update.** You can update the state of an existing resource by specifying the URL that identifies that object and its new state in JSON or XML, using the PUT HTTP method.

- **Delete.** You can destroy a resource that exists on the server-side by using the DELETE HTTP method and the URL identifying the resource to be removed.

In addition to these four CRUD operations (Create, Read, Update, and Delete), resources can support other operations or actions. These operations use the HTTP POST method, with the URL specifying the operation to be performed and the request body specifying the parameters for that operation.

NetScaler NITRO APIs are categorized depending on the scope and purpose of the APIs into system APIs, feature configuration APIs, and feature statistics APIs.

Performing System Level Operations

The first step towards using NITRO is to establish a session with the NetScaler appliance and then authenticate the session by using the NetScaler administrator's credentials. You must specify the username and password in the `login` object. The session ID that is created must be specified in the request header of all further operations in the session.

Note: You must have a user account on the appliance to log on to it. The configuration operations that you can perform are limited by the administrative roles assigned to your account.

To connect to a NetScaler appliance with NSIP address 10.102.29.60 by using the HTTP protocol:

- **URL.** https://10.102.29.60/nitro/v1/config/login/
- **Method.** POST
- **Request.**
  - **Header.**

    Content-Type:application/vnd.com.citrix.netscaler.login+json

    Note: Content types such as 'application/x-www-form-urlencoded' that were supported in earlier versions of NITRO can also be used. You must make sure that the payload is the same as used in earlier versions. The payloads provided in this documentation are only applicable if the content type is of the form 'application/vnd.com.citrix.netscaler.login+json'.
  - **Payload.**

```
{
    "login":
    {
        "username":"admin",
        "password":"verysecret"
    }
}
```

- **Response.**
  - **Header.**

```
HTTP/1.0 201 Created
Set-Cookie:
NITRO_AUTH_TOKEN=##87305E9C51B06C848F0942; path=/nitro/v1
```

Note: By default, the connection to the appliance expires after 30 minutes of inactivity. You can modify the timeout period by specifying a new timeout period (in seconds) in the `login` object. For example, to modify the timeout period to 60 minutes, the request payload is:

```
{
   "login":
   {
      "username":"admin",
      "password":"verysecret",
      "timeout":3600
   }
}
```

You can also connect to the appliance to perform a single operation, by specifying the username and password in the request header of the operation. For example, to connect to an appliance while adding a load balancing virtual server:

- **URL.** https://10.102.29.60/nitro/v1/config/lbvserver/
- **Method.** POST
- **Request.**
  - **Header.**

    ```
    X-NITRO-USER:admin
    X-NITRO-PASS:verysecret
    Content-Type:application/vnd.com.citrix.netscaler.lbvserver
    +json
    ```

  - **Payload.**

    ```
    {
      "lbvserver":
      {
        ...
        ...
        ...
    }
    ```
- **Response.**
  - **Header.**

    ```
    HTTP/1.0 201 Created
    ```

You can also perform other system-level operations such as enabling NetScaler features and modes, saving and clearing NetScaler configurations, setting the session timeout, setting the severity of the exceptions to be handled, setting the behavior of bulk operations, and disconnecting from the appliance.

For more information on the REST messages, see the Configuration node of the <NITRO_SDK_HOME>/index.html file.

Example 1: Enable the load balancing feature

- **URL.** http://10.102.29.60/nitro/v1/config/nsfeature?action=enable
- **HTTP Method.** POST
- **Request.**

- **Header**

  ```
  Cookie:NITRO_AUTH_TOKEN=tokenvalue
  Content-Type:application/vnd.com.citrix.netscaler.nsfeature+json
  ```

- **Payload**

  ```
  {
      "nsfeature":
      {
          "feature":
          [
              "LB",
          ]
      }
  }
  ```

Example 2: Save NetScaler configurations

- **URL.** http://10.102.29.60/nitro/v1/config/nsconfig?action=save
- **HTTP Method.** POST
- **Request.**
  - **Header**

    ```
    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.nsconfig+json
    ```

  - **Payload**

    ```
    {
        "nsconfig":{}
    }
    ```

Example 3: Disconnecting from the appliance

- **URL.** https://10.102.29.60/nitro/v1/config/logout/
- **HTTP Method.** POST
- **Request.**
  - **Header**

    ```
    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.logout+json
    ```

  - **Payload**

    ```
    {
        "logout":{}
    }
    ```

Note: Make sure that you have saved the configurations before performing this operation.

Configuring NetScaler Features

A NetScaler appliance has multiple features, and each feature has multiple resources. Each NetScaler resource, depending on the operation to be performed on it, has a unique URL associated with it. URLs for configuration operations have the format `http://<NSIP>/nitro/v1/config/<resource_type>/<resource_name>`. For example, to access the lbvserver named MyFirstLbVServer on a NetScaler with IP 10.102.29.60, the URL is `http://10.102.29.60/nitro/v1/config/lbvserver/MyFirstLbVServer`.

Using NITRO you can perform the following operations:

Create | Retrieve | Update | Delete | Enable/Disable | Unset | Bind/Unbind | Bulk operations

For more information on the REST messages, see the Configuration node of the <NITRO_SDK_HOME>/index.html file.

Create

To create a new resource (for example, an lbvserver) on the appliance, specify the resource name and other related arguments in the specific resource object. For a lbvserver resource, the object would be an `lbvserver` object.

To create an lbvserver named "MyFirstLbVServer":

- **URL.** http://10.102.29.60/nitro/v1/config/lbvserver/
- **HTTP Method.** POST
- **Request.**
  - **Header**

    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.lbvserver+json
  - **Payload**

    ```
    {
        "lbvserver":
        {
            "name":"MyFirstLbVServer",
            "servicetype":"http"
        }
    }
    ```

Retrieve

NetScaler resource properties can be retrieved as follows:

- To retrieve details of all resources of a specific type, specify the resource type in the URL.

  URL format: `http://<NSIP>/nitro/v1/config/<resource_type>`

- To retrieve details of a specific resource on the NetScaler appliance, specify the resource name in the URL.

  URL format:
  `http://<NSIP>/nitro/v1/config/<resource_type>/<resource_name>`

- To retrieve specific details of a resource, specify the resource details that you want to view in the URL.

  URL format: `http://<NSIP>/nitro/v1/config/<resource_type>/<resource_name>?attrs=<attrib1>,<attrib2>`

- To retrieve details of resources on the basis of some filter, specify the filter conditions in the URL.

  URL format: `http://<NSIP>/nitro/v1/config/<resource_type>?filter=<attrib1>:<value>,<attrib2>:<value>`

- If the request is likely to result in a large number of resources, you can divide the results into pages and retrieve them page by page.

  For example, assume that you have a NetScaler that has 53 lbvservers and you want to retrieve all the lbvservers. So, instead of retrieving all 53 in one response, you can configure the results to be divided into pages of 10 lbvservers each (6 pages total), and retrieve them from the NetScaler page by page.

  URL format: `http://<NSIP>/nitro/v1/config/<resource_type>?pageno=<value>&pagesize=<value>`

  You specify the page count with the `pagesize` parameter and the page number that you want to retrieve with the `pageno` parameter.

- To get the number of resources that are likely to be returned by a request, you can use the `count` query string parameter to ask for a count of the resources to be returned, rather than the resources themselves.

  URL format: `http://<NSIP>/nitro/v1/config/<resource_type>?count=yes`

To retrieve the details of an lbvserver named "MyFirstLbVServer":

- **URL.** http://10.102.29.60/nitro/v1/config/lbvserver/MyFirstLbVServer/
- **HTTP Method.** GET
- **Request.**
  - **Header**

    Cookie:NITRO_AUTH_TOKEN=tokenvalue
- **Response.**

- **Header**

  ```
  HTTP/1.0 200 OK
  Content-Type:application/vnd.com.citrix.netscaler.lbvserver+json
  ```

- **Payload**

  ```
  {
     "lbvserver":
     [
        {
           "name":"MyFirstLbVServer",
           "servicetype":"http",
           "insertvserveripport":"OFF",
           "ip":"0.0.0.0",
           "port":80,
           ...
        }
     ]
  }
  ```

Update

To update the details of an existing resource on the NetScaler appliance, specify the resource name, and the arguments to be updated, in the specific resource object.

To change the load balancing method to ROUNDROBIN and update the comment property for a load balancing virtual server named "MyFirstLbVServer":

- **URL.** http://10.102.29.60/nitro/v1/config/lbvserver/MyFirstLbVServer/
- **HTTP Method.** PUT
- **Request.**
  - **Header**

    ```
    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.lbvserver+json
    ```

  - **Payload**

    ```
    {
       "lbvserver":
       {
          "name":"MyFirstLbVServer",
          "lbmethod":"ROUNDROBIN",
          "comment":"Updated comments"
       }
    }
    ```

Delete

To delete a NetScaler resource, specify the resource name in the URL.

To delete a load balancing virtual server named "MyFirstLbVServer":

- **URL.** http://10.102.29.60/nitro/v1/config/lbvserver/MyFirstLbVServer
- **HTTP Method.** DELETE

Enable/Disable

To enable a resource on the NetScaler appliance, specify the resource name in the specific resource object.

To enable a load balancing virtual server named "MyFirstLbVServer":

- **URL.** http://10.102.29.60/nitro/v1/config/lbvserver?action=enable
- **HTTP Method.** POST
- **Request.**
  - **Header**

    ```
    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.lbvserver+json
    ```
  - **Payload**

    ```
    {
        "lbvserver":
        {
            "name":"MyFirstLbVServer"
        }
    }
    ```

Note: To disable a resource, in the URL specify the action as "disable".

Unset

To unset the value that is set to a parameter, specify the action as "unset" and in the payload, specify the parameters to be unset.

To unset the load balancing method and the comments specified for a load balancing virtual server named "MyFirstLbVServer":

- **URL.** http://10.102.29.60/nitro/v1/config/lbvserver?action=unset
- **HTTP Method.** POST
- **Request.**
  - **Header**

    ```
    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.lbvserver+json
    ```
  - **Payload**

    ```
    {
        "lbvserver":
    ```

```
    {
        "name":"MyFirstLbVServer",
        "lbmethod":true,
        "comment":true,
    }
}
```

Bind/Unbind

To bind a resource to another, specify the name of the two resources and specify the weight for the binding.

To bind a service named "svc_prod" to a load balancing virtual server named "MyFirstLbVServer", by specifying a certain weight for the binding:

- **URL.** http://10.102.29.60/nitro/v1/config/lbvserver_service_binding/
- **HTTP Method.** PUT
- **Request.**
  - **Header**

    ```
    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.lbvserver_service_binding+json
    ```

  - **Payload**

    ```
    {
        "lbvserver_service_binding":
        {
            "name":"MyFirstLbVServer",
            "servicename":"svc_prod",
            "weight":111,
        }
    }
    ```

Note: To unbind, specify the arguments in the URL as follows:

- **URL.** http://10.102.29.60/nitro/v1/config/lbvserver_service_binding/MyFirstLbVServer ?args=servicename:svc_prod
- **HTTP Method.** DELETE

Bulk operations

You can create, retrieve, update, and delete multiple resources simultaneously and thus minimize network traffic. For example, you can add multiple load balancing virtual servers in the same operation. To perform a bulk operation, specify the required parameters in the same request payload.

To account for the failure of some operations within the bulk operation, NITRO allows you to configure one of the following behaviors:

- **Exit.** When the first error is encountered, the execution stops. The commands that were executed before the error are committed.

- **Rollback.** When the first error is encountered, the execution stops. The commands that were executed before the error are rolled back. Rollback is only supported for add and bind commands.
- **Continue.** All the commands in the list are executed even if some commands fail.

You must specify the behavior of the bulk operation in the request header using the X-NITRO-ONERROR parameter.

To add two load balancing virtual servers in one operation and continue if one command fails:

- **URL.** http://10.102.29.60/nitro/v1/config/lbvserver/
- **HTTP Method.** POST
- **Request.**
  - **Header**

    ```
    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.lbvserver_list+json
    X-NITRO-ONERROR:continue
    ```

  - **Payload**

    ```
    {
        "lbvserver":
        [
            {
                "name":"new_lbvserver1",
                "servicetype":"http"
            },

            {
                "name":"new_lbvserver2",
                "servicetype":"http"
            }
        ]
    }
    ```
- **Response**
  - **Header**

    ```
    HTTP/1.0 207 Multi Status
    ```

  - **Payload**

    ```
    {
        "errorcode":273,
        "message":"Resource already exists",
        "severity":"ERROR",
        "response":
        [
            {
                "errorcode": 0,
    ```

```
                "message": "Done",
                "severity": "NONE"
            },
            {
                "errorcode": 273,
                "message":"Resource already exists",
                "severity": "ERROR"
            }
        ]
    }
```

Binding NetScaler Resources

NetScaler resources form relationships with each other through the process of binding. This is how services are associated with an lbvserver (by binding them to it), or how various policies are bound to an lbvserver. Each binding relationship is represented by its own object. A binding resource has properties representing the name of each NetScaler resource in the binding relationship. It can also have other properties related to that relationship (for example, the weight of the binding between an lbvserver resource and a service resource).

Note: Unlike for NetScaler entities, you use a PUT HTTP method, instead of POST, for adding new binding resources.

For more information on the REST messages, see the Configuration node of the <NITRO_SDK_HOME>/index.html file.

To bind a service to a load balancing virtual server named "MyFirstLbVServer" and specify a weight for the binding:

- **URL.** http://10.102.29.60/nitro/v1/config/lbvserver_service_binding/MyFirstLbVServer ?action=bind
- **HTTP Method.** PUT
- **Request.**
  - **Header**

    ```
    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.lbvserver_service_binding+json
    ```

  - **Payload**

    ```
    {
        "lbvserver_service_binding":
        {
            "servicename":"svc_prod",
            "weight":20,
            "name":"MyFirstLbVServer"
        }
    }
    ```

To retrieve list of all the services bound to a virtual server "lbv1":

- **URL.** http://10.102.29.60/nitro/v1/config/lbvserver_service_binding/lbv1?attrs=servicename
- **HTTP Method.** GET

For more information on retrieving information, see the "Retrieving properties of a resource" section in Configuring NetScaler Features.

Globally Bind Resources

Some NetScaler resources can be bound globally to affect the whole system. For example, if a compression policy is bound to an lbvserver, the policy affects only the traffic on that lbvserver. However, if bound globally, it can affect any traffic on the appliance, regardless of which virtual servers handle the traffic.

The names of NITRO resources that can be used to bind resources globally have the pattern <featurename>global_<resourcetype>_binding. For example, the object aaaglobal_preauthenticationpolicy_binding is used to bind preauthentication policies globally.

To bind the policy named preautpol1 globally at priority 200:

- **URL.** http://10.102.29.60/nitro/v1/config/aaaglobal_aaapreauthenticationpolicy_binding?action=bind
- **HTTP Method.** PUT
- **Request.**
  - **Header**

    ```
    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.aaaglobal_aaapreauthenticationpolicy_binding+j
    ```
  - **Payload**

    ```
    {
        "aaaglobal_aaapreauthenticationpolicy_binding":
        {
            "policy":"preautpol1",
            "priority":200
        }
    }
    ```

Configuring a NetScaler Cluster

You can use NITRO to add or create and manage a NetScaler cluster.

Cluster Instance Operations

All operations on a cluster instance must be performed on the `clusterinstance` object.

To create a cluster instance with ID 1:

- **URL.** http://10.102.29.60/nitro/v1/config/clusterinstance/
- **HTTP Method.** POST
- **Request.**
  - **Header**

    ```
    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.clusterinstance+json
    ```

  - **Payload**

    ```
    {
        "clusterinstance":
        {
            "clid":1,
            "preemption":"ENABLED"
        }
    }
    ```

Cluster Node Operations

All operations on a cluster node must be performed on the `clusternode` object.

To add a cluster node with NSIP address 10.102.29.60:

- **URL.** http://10.102.29.60/nitro/v1/config/clusternode/
- **HTTP Method.** POST
- **Request.**
  - **Header**

    ```
    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.clusternode+json
    ```

  - **Payload**

```
{
    "clusternode":
    {
        "nodeid":1,
        "ipaddress":"10.102.29.60",
        "state":"ACTIVE",
        "backplane":"1/1/2"
    }
}
```

Add a Cluster IP Address

To define a cluster IP address, specify the required parameters in the `nsip` object.

To configure a cluster IP address on NetScaler appliance with IP address 10.102.29.60:

- **URL.** http://10.102.29.60/nitro/v1/config/nsip/
- **HTTP Method.** POST
- **Request.**
  - **Header**

    ```
    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.nsip+json
    ```

  - **Payload**

    ```
    {
        "nsip":
        {
            "ipaddress":"10.102.29.61",
            "netmask":"255.255.255.255",
            "type":"CLIP"
        }
    }
    ```

Add a Spotted IP Address

To configure an IP address as spotted, specify the required parameters in the `nsip` object. This configuration must be done on the cluster IP address.

To configure a spotted SNIP address on a node with ID 1:

- **URL.** http://10.102.29.60/nitro/v1/config/nsip/
- **HTTP Method.** POST
- **Request.**
  - **Header**

    ```
    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.nsip+json
    ```

  - **Payload**

```
{
    "nsip":
    {
        "ipaddress":"10.102.29.77",
        "netmask":"255.255.255.0",
        "type":"SNIP",
        "ownernode":1
    }
}
```

Join NetScaler Appliance to Cluster

To join an appliance to a cluster, specify the required parameters in the `cluster` object.

To join a NetScaler appliance to a cluster:

- **URL.** http://10.102.29.60/nitro/v1/config/cluster/
- **HTTP Method.** POST
- **Request.**
  - **Header**

    ```
    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.cluster+json
    ```

  - **Payload**

    ```
    {
        "cluster":
        {
            "clip":"10.102.29.61",
            "password":"verysecret"
        }
    }
    ```

Linkset Operations

To configure a linkset, do the following:

1. Create a linkset by specifying the required parameters in the `linkset` object.

   To add a linkset LS/1:

   - **URL.** http://10.102.29.60/nitro/v1/config/linkset/
   - **HTTP Method.** POST
   - **Request.**
     - **Header**

       ```
       Cookie:NITRO_AUTH_TOKEN=tokenvalue
       Content-Type:application/vnd.com.citrix.netscaler.linkset+json
       ```

     - **Payload**

```
{
    "linkset":
    {
        "id":"LS/1"
    }
}
```

2. Bind the required interfaces to the linkset by specifying the interfaces in the `linkset_interface_binding` object.

   To bind interfaces 1/1/2 and 2/1/2 to linkset LS/1:

   - **URL.** http://10.102.29.60/nitro/v1/config/linkset_interface_binding/LS%2F1?action=bind
   - **HTTP Method.** PUT
   - **Request.**
     - **Header**

       Cookie:NITRO_AUTH_TOKEN=tokenvalue
       Content-Type:application/vnd.com.citrix.netscaler.linkset_interface_binding+json
     - **Payload**

       ```
       {
           "linkset_interface_binding":
           {
               "id":"LS/1",
               "ifnum":"1/1/2 2/1/2"
           }
       }
       ```

Retrieving Feature Statistics

The NetScaler appliance collects statistics about the usage of its features and the corresponding resources. NITRO can retrieve these statistics.

- URL to get statistics of a feature must have the format `http://<NSIP>/nitro/v1/stat/<feature_name>`.
- URL to get the statistics of a resource must have the format: `http://<NSIP>/nitro/v1/stat/<resource_type>/<resource_name>`.

For more information on the REST messages, see the Statistics node of the <NITRO_SDK_HOME>/index.html file.

To get the statistics of a lbvserver named "MyFirstLbVServer":

- **URL.** http://10.102.29.60/nitro/v1/stat/lbvserver/MyFirstLbVServer
- **HTTP Method.** GET
- **Request.**
  - **Header.**

    Content-Type:application/vnd.com.citrix.netscaler.lbvserver+json
- **Response.**
  - **Header**

    HTTP/1.0 200 OK
  - **Payload**

```
{
    "lbvserver":
    [
        {
            "name":"MyFirstLbVServer",
            "establishedconn":0,
            "vslbhealth":0,
            "primaryipaddress":"0.0.0.0",
            ...
        }
    ]
}
```

Note: Not all NetScaler features and resources have statistic objects associated with them.

To export an AppExpert application, specify the parameters needed for the export operation in the `apptemplateinfo` object. Optionally, you can specify basic information about the AppExpert application template, such as the author of the configuration, a summary of the template functionality, and the template version number, in the `template_info` object. This information is stored as part of the template file that is created.

To export an AppExpert application named "MyApp1":

- **URL.** http://10.102.29.60/nitro/v1/config/apptemplateinfo?action=export
- **HTTP Method.** POST
- **Request.**
  - **Header**

    Cookie:NITRO_AUTH_TOKEN=tokenvalue
    Content-Type:application/vnd.com.citrix.netscaler.apptemplateinfo+json
  - **Payload**

```
{
   "apptemplateinfo":
   {
      "appname":"MyApp1",
      "apptemplatefilename":"BizAp.xml",
      "template_info":
      {
         "templateversion_major":"2",
         "templateversion_minor":"1",
         "author":"XYZ",
         "introduction":"Intro",
         "summary":"Summary"
      },
   }
}
```

To import an AppExpert application, specify the parameters needed for the import operation in the `apptemplateinfo` object.

To import an AppExpert application named "MyApp1":

- **URL.** http://10.102.29.60/nitro/v1/config/apptemplateinfo?action=import
- **HTTP Method.** POST
- **Request.**
  - **Header**

```
Cookie:NITRO_AUTH_TOKEN=tokenvalue
Content-Type:application/vnd.com.citrix.netscaler.apptemplateinfo+json
X-NITRO-ONERROR:rollback
```

- **Payload**

```
{
  "apptemplateinfo":
  {
     "apptemplatefilename":"BizAp.xml",
     "deploymentfilename":"BizAp_deployment.xml",
     "appname":"MyApp1"
  }
}
```

To import an AppExpert application by specifying different deployment settings:

- **URL.** http://10.102.29.60/nitro/v1/config/apptemplateinfo?action=import
- **HTTP Method.** POST
- **Request.**
  - **Header**

```
Cookie:NITRO_AUTH_TOKEN=tokenvalue
Content-Type:application/vnd.com.citrix.netscaler.apptemplateinfo+json
X-NITRO-ONERROR:rollback
```

  - **Payload**

```
{
  "apptemplateinfo":
  {
    "apptemplatefilename":"BizAp.xml",
    "appname":"Myapp2"
     "deploymentinfo":
     {
        "appendpoint":
        [
           {
              "ipv46":"11.2.3.8",
              "port":80,
              "servicetype":"HTTP"
           }
        ],
        "service":
        [
           {
              "ip":"12.3.3.15",
              "port":80,
              "servicetype":"SSL"
           },
           {
              "ip":"14.5.5.16",
              "port":443,
                    "servicetype":"SSL"
                }
            ],
        }
    }
}
```

Handling Exceptions

The response header provides the status of an operation by using HTTP status codes and the response payload provides the requested resource object (for GET method) and error details (for unsuccessful operation). NITRO does not provide a response payload for successful POST, PUT and DELETE methods. For successful GET method, the response payload consists only the requested resource object.

The following table provides the HTTP status codes:

Status	HTTP Status Code	Description
Success	200 OK	Request successfully executed.
	201 CREATED	Entity created.
Failure	400 Bad Request	Incorrect request provided.
	401 unauthorized	Not provided login credentials.
	403 forbidden	User is unauthorized
	404 Not Found	User is trying to access a resource not present in the NetScaler.
	405 Method Not Allowed	User is trying to access request methods not supported by NITRO.
	406 Not Acceptable	None of the values supplied by the user in the Accept header can be satisfied by the server.
	409 Conflict	The resource already exists on the NetScaler.
	503 Service Unavailable	The service is not available.
	599	NetScaler specific error code.
Warning	209 X-NITRO-WARNING	Warnings are captured by specifying the login URL as http://<nsip>/nitro/v1/config/login/?warning=yes.
Combination of success and failure (for bulk operation with X-NITRO-ONERROR set as continue)	207 Multi Status	Some commands are executed successfully and some have failed.

Note: The content-type in the response header of an unsuccessful operation, consists of error MIME type instead of resource MIME type.

For a more detailed description of the error codes, see the API reference available in the <NITRO_SDK_HOME>/doc folder.

NITRO Changes Across NetScaler Releases

NetScaler has introduced some changes in the NITRO API since the NetScaler 9.3 release. This could raise some compatibility issues for the following users:

- Users migrating from NetScaler 9.3 to 10.1
- Users migrating from NetScaler 9.3 to 10.5

Note: There are no changes introduced since the NetScaler 10.1 release. Therefore, you should not face any compatibility issues when migrating from NetScaler 10.1 to 10.5.

These NITRO changes from 9.3 to 10.1 or 10.5 are categorized as follows:

- Resources Removed
- APIs Removed
- API Return Type Changed
- Attribute Type Changed
- Attributes Removed
- SDK Specific Changes

Note: Unless otherwise specified, these changes are applicable to both REST and SDKs.

Resources Removed

Resource	Replace with...	Comments
lbmonitor_lbmetrictable_binding	lbmonitor_metric_binding	

APIs Removed

Resource	API	Comments
vserver	GET	Perform the GET operation on specific virtual server types such as lb/cr/cs.
filterpolicy	POST with "action=unset"	This API is removed as unsetting the attributes('action') of a policy makes it invalid.
auditsyslogpolicy	POST with "action=unset"	This API is removed as unsetting the attributes('action') of a policy makes it invalid.
auditnslogpolicy	POST with "action=unset"	This API is removed as unsetting the attributes('action') of a policy makes it invalid.
authorizationpolicy	POST with "action=unset"	This API is removed as unsetting the attributes('action') of a policy makes it invalid.

API Return Type Changed

Resource	API	Comments
snmpengineid	GET	Return type changed to an array.
nshostname	GET	Return type changed to an array.

Attribute Type Changed

Resource	Attribute	Comments
appfwpolicy_lbvserver_binding	activepolicy	Data type changed from Boolean to Integer.
appfwpolicy_appfwglobal_binding	activepolicy	Data type changed from Boolean to Integer.
vlan	portbitmap	Data type changed from uint to ulong.
vlan	tagbitmap	Data type changed from uint to ulong.

Attributes Removed

Resource	Attribute	Replace with...	Comments
policypatset_pattern_binding	indextype	- NA -	This attribute is moved to 'policypatset' resource as this attribute is applicable at patset level.
system_stats	powersupply1failure	powersupply1status	Change is applicable from NetScaler 9.3 (65.8).
system_stats	powersupply2failure	powersupply2status	Change is applicable from NetScaler 9.3 (65.8).
server_servicegroup_binding	servicetype	svctype	
server_service_binding	servicetype	svctype	
crvserver	hits	- NA -	Hits are calculated per policy binding hence moved this parameter to binding resources.
crvserver	dstvsvr	destinationvserver	
crvserver	destvserver	domain	
crvserver	dnsvserver	dnsvservername	
appflowpolicylabel	type	policylabeltype	
sslcipher	ciphgrpals	ciphergroupname	This change is applicable for sslcipher_*_binding resources also.
csvserver_cspolicy_binding	targetvserver	targetlbvserver	
csvserver_cspolicy_binding	targetvserver	targetlbvserver	
rewriteaction	allow_unsafe_pi1, allow_unsafe_pi	bypassSafetyCheck	

SDK Specific Changes

Class	Method	Replace with...	Comments
Routerbgp	- NA -	- NA -	This class is removed as all router configurations are deprecated in 9.2.
dnsptrrec	get(dnsptrrec obj, nitro_service session)	get(nitro_service session, String reversedomain)	
dnsaddrec	get(dnsaddrec obj, nitro_service session)	get(nitro_service session, String hostname)	
dnsnsrec	get(dnsnsrec obj, nitro_service session)	get(nitro_service session, String domain)	
snmpengineid	unset(nitro_service session, String[] args)	unset(nitro_service session, snmpengineid resource, String[] args)	
arp	arp.get(nitro_service session, String ipaddress)	arp.get(nitro_service session, arp resource)	
nsip	get(nitro_service session, String ipaddress)	get(nitro_service client, nsip resource)	
nsip6	get(nitro_service session, String ipv6address)	get(nitro_service session, nsip6 resource)	
dnsmxrec	dnsmxrec.get(dnsmxrec obj, nitro_service session)	dnsmxrec[] get(nitro_service service, dnsmxrec_args args)	

Unsupported NetScaler Operations

Some NetScaler operations that are available through the command line interface and through the configuration utility, are not available through NITRO APIs. The following list provides the NetScaler operations not supported by NITRO:

- install API
- diff API on nsconfig resource
- UI-internal APIs (update, unset, and get)
- show ns info
- Application firewall APIs:
  - importwsdl
  - importcustom
  - importxmlschema
  - importxmlerrorpage
  - importhtmlerrorpage
  - rmwsdl
  - rmcustom
  - rmxmlschema
  - rmxmlerrorpage
  - rmhtmlerrorpage
- CLI-specific APIs:
  - ping
  - ping6
  - traceroute
  - traceroute6
  - nstrace
  - scp
  - configaudit

- show defaults
- show permission
- batch
- source

XML API

Developers and administrators can use the NetScaler Application Programming Interface (API), nsconfig, to implement customized client applications. The nsconfig API, which mirrors the NetScaler command line interface (CLI), is based on the Web Services Description (WSDL) specification. It includes a filterwsdl command to reduce compilation time and file size. You can secure your API applications at the NetScaler IP address or at the IP address of the subnet on which the NetScaler is deployed.

The following topics describe the properties and use of the API.

Introduction	General information about the API, requirements, and software-version information.
The NS Config Interface	How to use the API.
Examples of API Usage	Basic examples of how to use the API.
The Web Service Description Language (WSDL)	How to use the WSDL-based interface schema to support your client applications, and how to use WSDL Filter to reduce file size and compilation time.
Securing API Access	How to secure API access.

Introduction to the API

The API enables programmatic communications between client applications and the NetScaler appliance, providing the following benefits:

- Developers can control the NetScaler from a custom application. The API enables the client application to configure and monitor the NetScaler.
- Developers can create client applications easily and quickly, using a language and platform with which they are comfortable.
- The API provides a secure, end-to-end, standards-based framework that integrates into the existing infrastructure.

Based on the Simple Object Access Protocol (SOAP) over HTTP, the API consists of the NSConfig interface. NSConfig includes methods for setting and querying the configuration. These methods allow the client application using the NSConfig interface to perform almost all operations that an administrator would normally perform with the CLI or GUI.

In addition, the NetScaler provides an interface description, based on the Web Services Definition Language (WSDL), that facilitates the development of client applications.

Hardware and Software Requirements

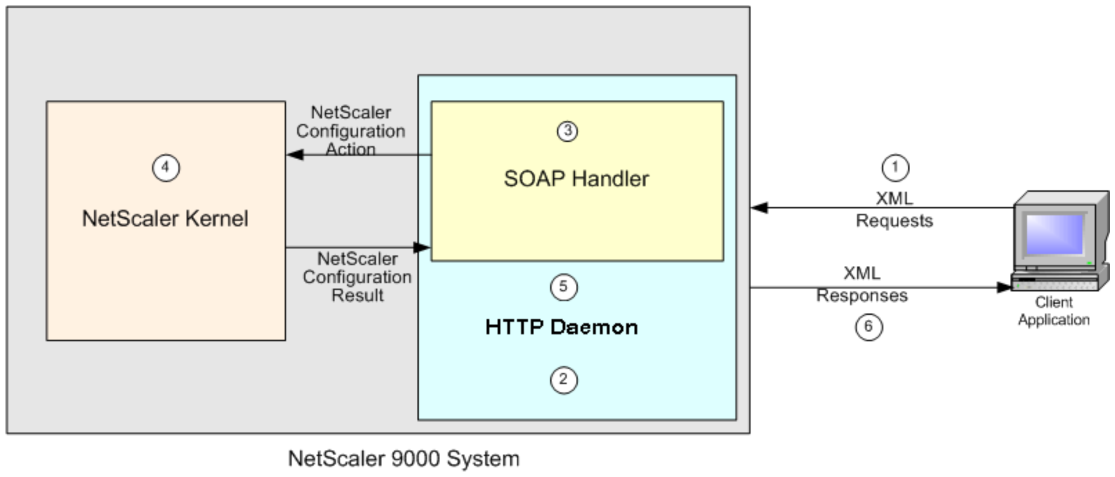
To work with the API, your system needs to meet the following hardware and software setup and requirements:

- A client workstation.
- Access to a NetScaler, version 8.0 or higher.
- A SOAP client tool kit (supporting SOAP version 1.1 and above), and the development environment for the tool kit. For example, if you use a Visual Basic tool kit, you must have Visual Basic installed on your system.

API Architecture

The API architecture is designed to allow NSConfig client requests to be routed, through the HTTP daemon running on the target NetScaler, to a SOAP handler that translates the SOAP request into a call to the (internal) kernel configuration API.

Figure 1. The API Architecture



The order in which the NetScaler processes requests through the API is as follows:

- The client formats a request containing XML conforming to the SOAP protocol and sends it to the NetScaler.
- The HTTPD server instance on the NetScaler routes this request to a SOAP handler.
- The SOAP handler interprets the SOAP headers and maps the enclosed request to an internal configuration function.
- The kernel acts on the request and returns one or more responses.
- The SOAP handler translates the response(s) to a SOAP response message.
- The XML response is sent back to the client in an HTTP response.

The NSConfig Interface

The NSConfig interface closely mirrors the structure of the NetScaler command line interface (CLI). Administrators and programmers who are familiar with the CLI can easily create and implement custom applications to query or set the configuration on their NetScaler.

The NSConfig interface includes methods for most of the CLI commands. In most cases the method and the command name are the same. See the PortType section of the WSDL for a complete list of methods and their names.

For example, you use the add lb vserver CLI command to create a load balancing virtual server, as follows:

```
add lb vserver <vServerName> <serviceType> [<IPAddress> <port>]
```

where:

<vServerName> = A name for the virtual server.

<serviceType> = (HTTP | FTP | TCP | UDP).

<IPAddress> = The IP address used by the virtual server.

<port> = The port that the virtual server listens on.

Following is the corresponding API call, in the C language:

```
int ns__addlbvserver(void *handle,
        string vServerName,
        string serviceType,
        string IPAddress,
        unsignedShort port,
    ns__addlbvserverResponse *out);
```

Note: The exact syntax of the API call depends on the language used to write the client program. The above `ns__addlbvserver` function prototype is similar to the one that would be generated by the gSOAP package at http://www.cs.fsu.edu/~engelen/soap.html.

The result returned for all NSConfig requests consists of:

Rc

An integer return code. The value is zero if the request succeeded. A non-zero value indicates that the request failed.

Message

A string message. Contains meaningful information only if the request fails (rc is non-zero) (for example, “Required argument missing”).

List

A type-specific list of result entities. This element is present only for requests that retrieve information from the NetScaler. For example, the API method names starting with get, which correspond to the CLI show commands, return a list.

Command names in the NetScaler CLI typically consist of three terms, separated by spaces, identifying the operation, the feature that is being operated on, and the specific item that is being operated on. For example, to create a new application firewall profile, you type add appfw profile, followed by the command arguments. The corresponding API methods omit the spaces. For example, the API method for add appfw profile is `addappfwprofile`. The same principle applies to CLI command names that have only two terms. For example, add monitor becomes `addmonitor`. The other exceptions to this pattern are as follows:

1. The CLI show command is changed to get in the API, as shown below.

   show lb vserver => `getlbvserver`

   show service => `getservice`

2. The following commands are omitted from the API:

   - Commands that apply to the CLI itself (for example, clear CLI prompt).
   - The batch, ping, grep, more, shell, and scp commands.
   - The show router bgp and show router map commands.
   - All stat commands.

3. Message "part" names in the API are the same as the corresponding CLI argument names. As in the CLI, case does not matter, and these names can be abbreviated. For more information, *Citrix NetScaler Command Reference Guide* at http://support.citrix.com/article/CTX132384

4. The result of a GET method (which corresponds to a show command in the CLI) is always an array of a type defined in the WSDL. The elements of these complex types generally correspond to arguments to the corresponding add/set command/method.

5. Authorization must be performed once, by sending a login request. The response contains a Set-Cookie HTTP header, and the cookie must be sent with each subsequent request. This is addressed in the Perl examples using by HTTP::Cookies. HTTP::Cookies are used for API client authentication purposes (to log into the NetScaler). In Perl, SOAP::Lite cannot perform this authentication process; HTTP::Cookies are used instead.

6. In some programming languages, such as Perl, it is possible to invoke the programming language API without using the WSDL.

Examples of API Usage

The following examples show how to develop an API call from a standard CLI command, how to generate the SOAP request, and how the NetScaler responds to that request:

Example: Setting the Configuration

Example: Querying the Configuration

Example: Setting the Configuration

This example shows a CLI command, the corresponding API method, the resulting XML request, and the XML response that is sent back to the client.

Note: The actual API method and the XML SOAP message contents may differ from the example shown below. The XML shown will be encased in a SOAP envelope, which will in turn be carried in an HTTP message. For more information, see the W3C web site at http://www.w3.org/TR/SOAP.

The following CLI command creates a Load Balancing virtual server:

```
> add lb vserver vipLB1 HTTP 10.100.101.1 80
```

Following is the corresponding API method:

```
> ns__addlbvserver (handle, “vipLB1”, “HTTP”, “10.100.101.1”, 80, &out);
```

The XML generated for this request is as follows.

```
<ns:addlbvserver>
<vServerName xsi:type="xsd:string" >vipLB1</vServerName>
<serviceType xsi:type="ns:vservicetypeEnum>HTTP</ serviceType>
<IPAddress xsi:type="xsd:string">10.100.101.1</IPAddress>
<port xsi:type="xsd:unsignedInt" >80</port>
< /ns:addlbvserver >
```

The XML response to the above request is as follows.

```
<ns:addlbvserverResponse>
<rc xsi:type="xsd:unsignedInt">0</rc>
<message xsi:type="xsd:string">Done</message>
</ns:addlbvserverResponse>
```

Example: Querying the Configuration

This example shows an API request that queries the configuration and receives a list of entities.

Note: The actual API method and the XML SOAP message contents may differ from the example shown below.

The following CLI command shows the configured Load Balancing virtual servers:

```
> show lb vservers
```

Sample output of the show lb vservers command is as follows.

```
> show lb vservers
2 configured virtual servers:
1)  vipLB1 (10.100.101.1:80) - HTTP Type: ADDRESS State:
    DOWN
    Method: LEASTCONNECTION Mode: IP
    Persistence: NONE
2)  vipLB2 (10.100.101.2:80) - HTTP Type: ADDRESS State:
    DOWN
    Method: LEASTCONNECTION Mode: IP
    Persistence: NONE
Done
```

Following is the corresponding API method to show the list of Load Balancing virtual servers.

```
ns__getlbvserver(handle, NULL, &out)
```

The XML generated for this request is as follows.

```
<ns:getlbvserver></ns:getlbvserver>
```

The XML response to the above request is as follows.

```
<ns:getlbvserverResponse>
    <rc xsi:type="xsd:unsignedInt">0</rc>
    <message xsi:type="xsd:string">Done</message>
    <List xsi:type="SOAP-ENC:Array"
        SOAP-ENC:arrayType="ns:lbvserver[2]">
        <item xsi:type="ns:lbvserver">
        <vServerName xsi:type="xsd:string>vipLB1
            </vServerName>
            <serviceType xsi:type="xsd:string>HTTP</ serviceType>
            <IPAddress xsi:type="xsd:string >10.100.101.1
            </IPAddress>
        <port xsi:type="xsd:unsignedInt">80</port>
    </item>
    <item xsi:type="ns:lbvserver">
```

```
            <vServerName xsi:type="xsd:string>vipLB2
            </vServerName>
            <serviceType xsi:type="xsd:string>HTTP</ serviceType>
            <IPAddress xsi:type="xsd:string >10.100.101.2
            </IPAddress>
               <port xsi:type="xsd:unsignedInt">80</port>
            </item>
         </List>
      </ns:getlbvserverResponse>
```

The Web Service Definition Language (WSDL)

The NetScaler WSDL describes services for the entire range of NetScaler services. The NetScaler provides two WSDL files:

NSConfig.wsdl

Configuration APIs are defined in this file. The NSConfig.wsdl file is found on the NetScaler at http://<NSIP>/api/NSConfig.wsdl, where <NSIP> is the IP address of your NetScaler. This file is much larger than the NSStat.wsdl file. With the help of a third-party tool (such as gSOAP), developers can use this file to generate client stubs. A custom application can then call the stubs to send requests to the NetScaler. The application can be in any standard programming language that is supported by the third-party tool. Common programming languages for this purpose include Perl, Java, C, and C#. You can use the filterwsdl command to select only the service definitions that are relevant to the API calls made in your script.

NSStat.wsdl

Statistical APIs are defined in this file. The NSStat.wsdl file is found on the NetScaler at http://<NSIP>/api/NSStat.wsdl, where <NSIP> is the IP address of your NetScaler.

Creating Client Applications with the NSConfig.wsdl File

A client application can be created by importing the NSConfig.wsdl file with the gSOAP WSDL Importer to create a header file with C or C++ declarations of the SOAP methods. The gSOAP compiler is then used to translate this header file into stubs for the client application.

1. Get the NSConfig.h header file from the WSDL file.

    a. Run the wsdl2h program that comes with gSOAP on the WSDL file. The wsdl2h program is in the following location.

    ```
    > ./wsdl2h NSConfig.wsdl
    ```

    The output of wsdl2h is as follows:

    ```
    **  The gSOAP WSDL parser for C and C++ 1.0.2
    **  Copyright (C) 2001-2004 Robert van Engelen, Genivia, Inc.
    **  All Rights Reserved. This product is provided "as is", without any warranty.
    Saving NSConfig.h
    Reading file 'NSConfig.wsdl'
    Cannot open file 'typemap.dat'
    Problem reading type map file typemap.dat.
    Using internal type definitions for C instead.
    ```

    b. Run the soapcpp2 program to compile the header file and complete the process, as shown below. > soapcpp2 NSConfig.h
2. Generate the XML files and stubs as follows:

    ```
    > ./soapcpp2 -c -i NSConfig.h
    ```

    Following is sample output for this command:

    ```
    **  The gSOAP Stub and Skeleton Compiler for C and C++ 2.4.1
    **  Copyright (C) 2001-2004 Robert van Engelen, Genivia, Inc.
    **  All Rights Reserved. This product is provided "as is", without any warranty.
    Saving soapStub.h
    Saving soapH.h
    Saving soapC.c
    Saving soapClient.c
    Saving soapServer.c
    Saving soapClientLib.c
    Saving soapServerLib.c
    Using ns1 service name: NSConfigBinding
    Using ns1 service location: http://NetScaler.com/api Using ns1 schema namespace: urn:NSConfig
    Saving soapNSConfigBindingProxy.h client proxy
    Saving soapNSConfigBindingObject.h server object
    ```

```
Saving NSConfigBinding.addserver.req.xml sample SOAP/XML request
Saving NSConfigBinding.addserver.res.xml sample SOAP/XML response
Saving NSConfigBinding.disableserver.req.xml sample SOAP/ XML request
Saving NSConfigBinding.disableserver.res.xml sample SOAP/ XML response
Saving NSConfigBinding.enableserver.req.xml sample SOAP/ XML request
Saving NSConfigBinding.enableserver.res.xml sample SOAP/ XML response
[ ... Similar lines clipped ... ]
Saving NSConfigBinding.nsmap namespace mapping table
Compilation successful
```

This creates the stub files soapC.c, soapClient.c and stdsoap2.c.

3. Link the stub files you created with your source code to create a stand-alone binary that invokes the API.

Filter WSDL

The NetScaler WSDL describes services for the entire range of NetScaler services. When you use the NetScaler API in your scripts, by linking to the WSDL and attempting to compile the application, the entire WSDL is included, unnecessarily increasing compilation time and the size of the program.

Filter WSDL is a tool for selecting only those service definitions from the NetScaler WSDL that are relevant to the API calls made in the script. You can use the filter WSDL tool to filter NSConfig.wsdl and NSStat.wsdl files.

The NetScaler provides two WSDL files, one for the configuration APIs (NSConfig.wsdl) and the other for statistical APIs (NSStat.wsdl). The WSDL file for the configuration API is much larger. Therefore, it is important to use filter WSDL when compiling programs written with the configuration API.

Filter WSDL is a program that works on the Windows, FreeBSD and Linux platforms, and it can be run from the CLI.

The syntax for running filter WSDL is as follows:

```
filterwsdl <fromwsdl> <pattern>
```

where:

fromwsdl = The wsdl file that you want to filter

pattern = API method names or patterns that should be filtered

For example, if you want to filter all the service definitions for the API method addlbvserver from the NetScaler WSDL file, NSConfig.wsdl, you can use the command:

```
> filterwsdl NSConfig.wsdl "addlbvserver"
```

The output of this command is sent to the screen by default, but it can be redirected to a file on the NetScaler by using the UNIX redirect operator (>). The output of the previous command can be saved into a file called NSConfig-Custom.wsdl by using the command as follows:

```
> filterwsdl NSConfig.wsdl "addlbvserver" > NSConfig-Custom.wsdl
```

In this case, the original WSDL file is 1.58 MB, but the filtered WSDL file is 6 KB.

The pattern used in the filterwsdl command can include the + and - operators and the wildcard operator (*) to create more generic filters.

For example, if you want to filter the service definitions for all the available load balancing methods, you can use the following command:

```
> filterwsdl NSConfig.wsdl "*lb"*
```

This command will filter all the Load Balancing methods but will also include GSLB methods, because the pattern lb will be matched by all GSLB methods also. To include only LB methods and exclude all GSLB methods, use the command as follows:

```
> filterwsdl NSConfig.wsdl +"*lb" -"glsb"*
```

Securing API Access

Secure access to CLI objects can be based on the NetScaler IP address or on the subnet IP address on which the NetScaler is deployed. To provide secured API access based on the NetScaler IP address, you must configure the NetScaler to use transparent SSL mode with clear text port.

To configure secure API access based on the NetScaler IP

1. Create a loopback SSL service and configure it use transparent SSL mode with clear text port:

   ```
   add service secure_xmlaccess 127.0.0.1 SSL 443 -clearTextPort 80
   ```

2. Add certificate and key:

   ```
   add certkey cert1 –cert /nsconfig/ssl/ssl/cert1024.pem –key
   /nsconfig/ssl/ssl/rsakey.pem
   ```

   Note: You can use an existing certificate and key or use the NetScaler Certificate Authority Tool to create a key and test certificate for secure access.

3. Bind the certificate and key to the service:

   ```
   bind certkey secure_xmlaccess cert1 -Service
   ```

4. Add a custom TCP monitor to monitor the SSL service you have added:

   ```
   add monitor ssl_mon TCP -destport 80
   ```

5. Bind the custom TCP monitor to the SSL service:

   ```
   bind monitor ssl_mon secure_xmlaccess
   ```

To configure secure API access based on the subnet IP

1. Create an SSL VIP in the appropriate subnet:

   ```
   add vserver <vServerName> SSL <Subnet-IP> 443
   ```

2. Create a loopback HTTP service:

   ```
   add service <serviceName> 127.0.0.1 HTTP 80
   ```

3. Bind the service to the SSL VIP:

   ```
   bind lb vserver <vServerName> <serviceName>
   ```

4. Add the certificate and the key:

   ```
   add certkey cert1 –cert /nsconfig/ssl/ssl/cert1024.pem -key
   /nsconfig/ssl/ssl/rsakey.pem
   ```

   Note: You can use an existing certificate and key or use the NetScaler Certificate Authority Tool to create a key and test certificate.

5. Bind the Certificate and the Key to the SSL VIP:

   ```
   bind certkey <vServerName> cert1
   ```