Citrix NetScaler 1000V NITRO Getting Started Guide

Citrix NetScaler 10.1
October 3, 2013

Cisco Systems, Inc.
www.cisco.com

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco website at www.cisco.com/go/offices.

Contents

Chapter 1

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 Java and .NET libraries that are packaged as separate Software Development Kits (SDKs).

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

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.

Chapter 2

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 (JARs for Java and DLLs for .NET) 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.

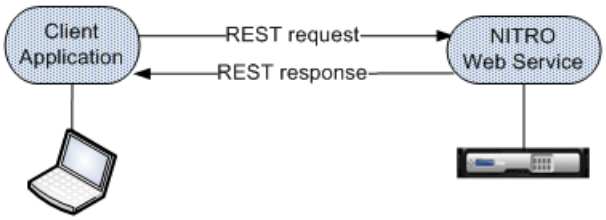
Note: The REST package contains only documentation for using the REST interfaces.

Chapter 3

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 3-1. 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 Java or .NET 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 Java or .NET 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.

Chapter 4

Java API

Topics:

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.

Note: All NITRO operations are logged in the `/var/nitro/nitro.log` file on the appliance.

Tutorials

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

- Create Your First NITRO Application

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.get_servicetype());
```

   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.

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 in NITRO. 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 other feature-level operations such as creating a resource, retrieving resources and resource properties, updating a resource, binding resources, and performing bulk operations on resources.

Creating a Resource

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);
```

Retrieving Properties of a Resource

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);
```

Updating a Resource

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.

Performing 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);
```

Binding Resources

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);
```

Globally Binding Resources

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,glo
b_binding);
```

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.

Chapter 5

.NET API

Topics:

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.

Note: All NITRO operations are logged in the `/var/nitro/nitro.log` file on the appliance.

Tutorials

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

- Create Your First NITRO Application

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_obj1.servicetype);
```

   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.

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 in NITRO. 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 other feature-level operations such as creating a resource, retrieving resources and resource properties, updating a resource, binding resources, and performing bulk operations on resources.

Creating a Resource

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);
```

Retrieving Properties of a Resource

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);
```

Updating a Resource

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.

Performing 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);
```

Binding Resources

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);
```

Globally Binding Resources

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,glo
b_binding);
```

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.

Chapter 6

REST Web Services

Topics:

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.

Note: All NITRO operations are logged in the `/var/nitro/nitro.log` file on the appliance.

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 operations such as creating a resource, retrieving resources and resource properties, updating a resource, binding resources, and performing bulk operations on resources.

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

Creating a resource

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"
     }
}
```

Retrieving properties of a resource

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,
            ...
        }
    ]
}
```

Updating a resource

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"
    }
}
```

Enabling a resource

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"
    }
}
```

Performing 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_aaapreauthenticationpoli
cy_binding+json
```

  - **Payload**

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

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.

Managing AppExpert Applications

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.

Chapter 7

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`