Citrix NetScaler MAS - NITRO Python - Getting Started Guide

Citrix® NetScaler® MAS 11.1

Last Updated: March 2016

Document code: June 28 2016

Contents

NITRO API

The Citrix NetScaler MAS NITRO protocol allows you to configure and monitor the NetScaler MAS 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 or Python, the NITRO protocol is exposed as relevant libraries that are packaged as separate Software Development Kits (SDKs).

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

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

- Access to a NetScaler MAS, release11.1.
- To use REST interfaces, you must have a system to generate HTTP or HTTPS requests (payload in JSON format) to the NetScaler MAS. You can use any programming language or tool.
- For Java clients, you must have a system where Java Development Kit (JDK) 1.5 or above version 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 where .NET framework 3.5 or above version is available. The .NET framework can be downloaded from http://www.microsoft.com/downloads/en/default.aspx.
- For Python clients, you must have a system where Python 2.7 or above version and the Requests library (available in <NITRO_SDK_HOME>/lib) is installed.

Obtaining the NITRO Package

The NITRO package is available as a tar file on the **Downloads** page of the NetScaler MAS 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.

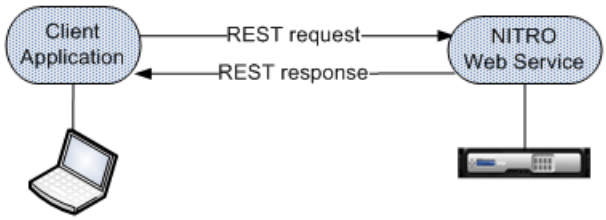
Note:

- The REST package contains only documentation for using the REST interfaces.
- For the Python SDK, the library must be installed on the client path. For installation instructions, read the `<NITRO_SDK_HOME>/README.txt` file.

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 1-1. 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 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.

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.

API Categorization

NetScaler MAS NITRO APIs are categorized depending on the scope and purpose of the APIs into system APIs and configuration APIs. You can also troubleshoot NITRO operations.

System APIs

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

You must create an object of the `nitro_service` class by specifying the IP address of the appliance 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 administrator.

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

The following sample code connects to a NetScaler MAS with IP address 10.102.31.16 by using HTTPS protocol:

```
# Specify the IP address of the appliance and service type
nitroservice = nitro_service ("10.102.31.16", "https")

# Specify the login credentials
nitroservice.login("nsroot", "verysecret")
```

Note: You must use the `nitro_service` object in all further NITRO operations on the appliance.

To disconnect from the appliance, invoke the `logout()` method as follows:

```
nitroservice.logout()
```

Configuration APIs

The NITRO protocol can be used to configure resources of the NetScaler MAS.

The APIs to configure a resource are grouped into packages or namespaces that have the format `massrc.com.citrix.mas.nitro.resource.config.<resource_type>`. Each of these packages or namespaces contain a class named <resource_type> that provides the APIs to configure the resource.

For example, the NetScaler resource has the `massrc.com.citrix.mas.nitro.resource.config.ns` package or namespace.

A resource class provides APIs to perform other operations such as creating a resource, retrieving resource details and statistics, updating a resource, deleting resources, and performing bulk operations on resources.

Creating a Resource

To create a new resource (for example, a NetScaler instance) on the NetScaler MAS, do the following:

1. Set the value for the required properties of the resource by using the corresponding property name. The result is a resource object that contains the details required for the resource.

   Note: These values are set locally on the client. The values are not reflected on the appliance till the object is uploaded.

2. Upload the resource object to the appliance, using the static `add()` method.

The following sample code creates a NetScaler instance named "ns_instance" on the NetScaler MAS:

```
newns = ns()

# Set the properties of the NetScaler locally
newns.name = "ns_instance"
newns.ip_address = "10.70.136.5"
newns.netmask = "255.255.255.0"
newns.gateway = "10.70.136.1"
newns.image_name = "nsvpx-9.3.45_nc.xva"
newns.profile_name = "ns_nsroot_profile"
newns.vm_memory_total = 2048.0
newns.throughput = 1000.0
newns.pps = 1000000.0
newns.license = "Standard"
newns.username = "admin"
newns.password = "admin"

# Here number_of_interfaces we have considered is 2
interface_array=[]
interface_array = [network_interface() for _ in range(2)]

# Adding 10/1
interface_array[0].port_name = "10/1"

# Adding 10/2
interface_array[1].port_name = "10/2"
newns.network_interfaces = interface_array

# Upload the NetScaler instance
result = ns.add(nitroservice, newns)
```

Retrieving Resource Details

To retrieve the properties of a resource on the NetScaler MAS, do the following:

1. Retrieve the configurations from the appliance by using the `get()` method. The result is a resource object.
2. Extract the required property from the object by using the corresponding property name.

The following sample code retrieves the details of all NetScaler resources:

```
# Retrieve the resource object from the NetScaler MAS
returned_ns = ns.get(nitroservice)

# Extract the properties of the resource from the object
for i in range(0,len(returned_ns)):
    print returned_ns[i].ip_address
    print returned_ns[i].netmask
```

Retrieving Resource Statistics

A NetScaler MAS collects statistics on the usage of its features. You can retrieve these statistics using NITRO.

The following sample code retrieves statistics of a NetScaler instance with ID 123456a:

```
obj = ns()
obj.id = "123456a"
stats = ns.get(nitroservice, obj)
print "CPU Usage:" + stats.ns_cpu_usage
print "Memory Usage:" + stats.ns_memory_usage
print "Request rate/sec:" +stats.http_req
```

Updating a Resource

To update the properties of an existing resource on the appliance, do the following:

1. Set the id property to the ID of the resource to be updated.
2. Set the value for the required properties of the resource by using the corresponding property name. The result is a resource object.

   Note: These values are set locally on the client. The values are not reflected on the appliance till the object is uploaded.

3. Upload the resource object to the appliance, using the `update()` method.

The following sample code updates the name of the NetScaler instance with ID 123456a to 'ns_instance_new':

```
update_obj = ns()

# Set the ID of the NetScaler to be updated
update_obj.id = "123456a"

# Get existing NetScaler details
update_obj = ns.get(nitroservice, update_obj)

# Update the name of the NetScaler to "ns_instance_new" locally
update_obj.name = "ns_instance_new"

# Upload the updated NetScaler details
result = ns.update(nitroservice, update_obj)
```

Deleting a Resource

To delete an existing resource, invoke the static method `delete()` on the resource class, by passing the ID of the resource to be removed, as an argument.

The following sample code deletes a NetScaler instance with ID 1:

```
obj = ns()
obj.id = "123456a"
ns.delete(nitroservice, obj)
```

Bulk Operations

You can query or change multiple resources simultaneously and thus minimize network traffic. For example, you can add multiple NetScaler appliances in the same operation.

Each resource class has methods that take an array of resources for adding, updating, and removing resources. To perform a bulk operation, specify the details of each operation locally and then send the details at one time to the server.

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.
- **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, by setting the `onerror` param in the `nitro_service()` method.

The following sample code adds two NetScalers in one operation:

```
newns=[]
newns = [ns() for _ in range(2)]

# Specify details of first NetScaler
newns[0].name = "ns_instance1"
newns[0].ip_address = "10.70.136.5"
newns[0].netmask = "255.255.255.0"
newns[0].gateway = "10.70.136.1"

# Specify details of second NetScaler
newns[1].name = "ns_instance2"
newns[1].ip_address = "10.70.136.8"
newns[1].netmask = "255.255.255.0"
newns[1].gateway = "10.70.136.1"

# Upload the details of the NetScalers to the NITRO server
result = ns.add(nitroservice, newns)
```

Exception Handling

The `errorcode` field indicates the status of the operation.

- An errorcode of 0 indicates that the operation is successful.
- A non-zero errorcode indicates an error in processing the NITRO request.

The error message field provides a brief explanation and the nature of the failure.

All exceptions in the execution of NITRO APIs are caught by the `massrc.com.citrix.mas.nitro.exception.nitro_exception` class. To get information about the exception, you can use the `getErrorCode()` method.

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