Scripting system management operations on the IBM MQ

Appliance using the REST API

Jamie Squibb |Apr 28 2017 Updated

In this post I explain how the REST API can be used to perform system management operations. I also demonstrate how these operations can be performed using Perl, which is a more traditional scripting language.

**Understanding the REST API**

The REST management API on the IBM MQ Appliance is based on that provided by the IBM DataPower Gateway Appliance. The basic structure is as follows:

* The REST management API, once enabled, is available on the appliance via the URI https://hostname:port/mgmt/, where the default port number is 5554.
* To determine what status information can be queried point your browser (or perform a REST HTTP GET) at /mgmt/status/. This query returns a list of the available status providers and the URI that can be used to invoke them. Each provider can be queried on the IBM MQ Appliance by appending ‘default/’ for the domain qualifier, then the provider name. For example, to query the users that are currently logged in you can use /mgmt/status/default/ActiveUsers.
* System management tasks are divided into configuration changes, operations and file management. Different URI trees are available for each of these operations.
* Configuration changes are performed by defining, modifying or deleting objects. Some system objects must always exist but others can be instantiated as necessary. For example, an administrative user is created by defining an object of type User. The REST URI for configuration objects is /mgmt/config/default/<object-class>/<object-name> and the list of object classes can be determined by querying /mgmt/config/. To query the attributes of an object use an HTTP GET operation, to remove an object use a DELETE operation, to define a new object use a POST operation, or to define/modify an object use a PUT operation. Note that when using POST the URI does not include the object name. Object attributes are not returned in GET responses if their value is not set so to obtain a list of all the available attributes you can query metadata information via /mgmt/metadata/default/<object-class>. To find out more about an object attribute you can append /<attribute-name> to the metadata URI. You can also query /mgmt/types/default/<attribute-type> to find out more information about attribute types, such as the permitted values in an enumeration.
* Operations are requested using a POST to /mgmt/actionqueue/default. The body of a request specifies the action to perform and the value of any parameters it requires. To obtain the list of operations that are available you can query /mgmt/actionqueue/default/operations. You can similarly query /mgmt/metadata/default/operations/<operation-name> to determine the parameters an operation supports.
* To upload or download files you use /mgmt/filestore/default/<file-path>. The file content is either returned or sent in the request body as a base64 encoded string. See Anna’s corresponding article for more information on this.

**Scripting REST operations using Perl**

As per many scripting languages, Perl has a rich library of optional modules. There are actually a few alternative modules available that support REST operations but in this blog post I illustrate how to use the LWP module, which provides HTTP support, plus the JSON module, which provides support for generating and parsing JSON. These modules can be obtained from CPAN if they are not already installed. You also need to ensure that the LWP::Protocol::https module is installed because the REST API requires an SSL/TLS connection. If you are using Red Hat Enterprise Linux (RHEL) 7.2 then these modules are installed by the following packages:

* perl-libwww-perl-6.05-2.el7.noarch
* perl-LWP-Protocol-https.noarch
* perl-JSON.noarch
* perl-JSON-PP.noarch

The Data::Dumper module, which is usually installed with Perl, is also particularly useful to query data structures when writing or debugging scripts.

To perform a REST operation using these modules a user agent must be defined that essentially represents the equivalent of a web browser. The user agent needs to be configured to use SSL/TLS and provide valid credentials to the appliance through basic authentication. The following code snippet illustrates how to do this.

#!/usr/bin/perl -w

use strict;

use Data::Dumper;

use LWP 5.64;

use JSON;

# The host name or IP address of the appliance

my $host = "myhost.mydomain";

# The port of the REST management interface

my $port = 5554;

# SSLv3 is not permitted so we must use TLS instead

# Also, disable certificate validation to simplify this example

my %ssl\_options = ( SSL\_version => "TLSv1",

                    verify\_hostname => 0 );

# Create a HTTPS user agent object

my $agent = LWP::UserAgent -> new (ssl\_opts => \%ssl\_options);

# Specify basic authentication credentials

$agent -> credentials

(

  $host . ":" . $port,

  "",

  "admin" => "password"

);

**Querying information about a user**

To query information about an existing user a GET operation can be performed against the corresponding User object. Below is an example of how to obtain information about the user called bob.

my $uri = "https://" . $host . ":" . $port . "/mgmt/config/default/User/bob";

my $response = $agent -> get($uri);

if ( $response -> is\_success() )

{

  # Woohoo it worked!

  ...

}

else

{

  # Doh, it failed

  print "Unable to query information about bob – HTTP response (" . $response -> code() . ")\n";

  ...

}

A successful JSON response will contain something like the following:

{

  "\_links" : {

    "self" : {

      "href" : "/mgmt/config/default/User/bob"

    },

    "doc" : {

      "href" : "/mgmt/docs/config/User"

    }

  },

  "User" : {

    "name" : "bob",

    "mAdminState" : "enabled",

    "UserSummary" : "Bob (Operations Team)",

    "AccessLevel" : "group-defined",

    "GroupName" : {

      "value": "PowerUsers",

      "href" : "/mgmt/config/default/UserGroup/PowerUsers"

    }

  }

}

In this response we can see some links that identify the URI we queried and a User object that represents Bob. The object identifies that he is a member of a group that has been previously created called PowerUsers, which defines the authority he has been granted. Note the GroupName attribute is a reference to the group and the href attribute of the reference specifies the URI that can be used to query information about it.

To parse this response so it can be processed programmatically the JSON Perl module can be used. This module parses JSON attributes to scalar values, JSON objects to Perl hashes and JSON lists to Perl arrays. The following code snippet illustrates how it can be used to parse the response containing information about Bob:

my $json = decode\_json($response -> content);

my $bob = $json -> {"User"};

print "User name: " . $bob -> {"name"} . "\n";

print "User group: " . $bob -> {"GroupName"} -> {"value"} . "\n";

I mentioned the Data::Dumper module above. This is useful if you are not sure how the response has been parsed. The following will print out a prettified structure of the JSON response:

print Dumper($json);

**Defining a new user**

To create a new user either a PUT or a POST operation is used. Using POST will fail if the user already exists; using PUT will create the user if they don’t exist or update the user’s attributes if they do. When updating an existing user you only need to specify the user name and the attributes you wish to modify. Let’s say Bob has a new colleague called Alice who requires the same level of access. To create her user account we need to send the following JSON request:

{

  "User" : {

    "name" : "alice",

    "mAdminState" : "enabled",

    "UserSummary" : "Alice (Operations Team)",

    "AccessLevel" : "group-defined",

    "GroupName" : {

      "value": "PowerUsers",

    },

    "Password" : "xRf4!dkE"

  }

}

Note that the request is similar to the response that is returned for Bob, except the \_links and href attributes have been omitted. Also the initial password for the user must be specified and by default Alice must change it the first time she logs in.

To script the creation of Alice’s user account we can define a string containing the JSON. Alternatively, we can generate the JSON from an equivalent Perl data structure using the JSON module. The following code snippet illustrates one way we can use the latter approach:

# Construct a hash for the group reference

my %power\_users = ( "value" => "PowerUsers" );

# Construct a hash for the user object

my %alice =

(

  "name"        => "alice",

  "mAdminState" => "enabled",

  "UserSummary" => "Alice (Operations Team)",

  "AccessLevel" => "group-defined",

  "GroupName"   => \%power\_users,

  "Password"    => "xRf4!dkE"

);

# Construct the JSON request with a qualifier for the object class

my %new\_user = ( "User" => \%alice );

# Encode the Perl data structure as JSON

my $json = encode\_json(\%new\_user);

# Issue a POST request to create the user

my $uri = "https://" . $host . ":" . $port . "/mgmt/config/default/User";

my $response = $agent -> post($uri, Content => $json);

# Check the request was successful

...

**Resetting a user’s password**

The final example I’ll illustrate in this blog post is how to perform an action using a request for an operation to be performed. The example I’ll use is how to administratively reset the password for another user, which they’ll need to change the next time they log in. The code snippet below illustrates how to do this with a POST request using Perl (see the /mgmt/metadata/default/operations/UserResetPassword URI for information about the operation’s parameters). This time I show how to specify the JSON explicitly using a Perl HERE document (multi-line raw text).

# Define the JSON request

my $json = <<"--JSON--";

{

  "UserResetPassword" : {

    "User" : "alice",

    "Password" : "S4K-3DDv"

  }

}

--JSON--

# Reset Alice’s password using a POST request

my $uri = "https://" . $host . ":" . $port . "/mgmt/actionqueue/default";

my $response = $agent -> post($uri, Content => $json);

# Check the request was successful

...

**Summary**

The IBM MQ Appliance provides a comprehensive REST management API for performing system configuration changes and querying status information. Hopefully this blog post has provided an introduction to how configuration changes can be performed using REST operations and how they can be scripted. If you don’t want to use Perl then there are plenty of other scripting languages available that support REST calls and it should be fairly simple to convert the above code snippets to your language of choice.

**Related links**

[An Example for how to configure Role Based Management on MQ Appliance to allow access to LDAP users [Maya Anilson]](https://community.ibm.com/community/user/imwuc/viewdocument/an-example-for-how-to-configure-rol?CommunityKey=183ec850-4947-49c8-9a2e-8e7c7fc46c64&tab=librarydocuments)