REST API access and authentication in Active IQ Unified Manager

Active IQ Unified Manager 9.13

NetApp
February 12, 2024

Table of Contents

REST API access and authentication in Active IQ Unified Manager

The Active IQ Unified Manager REST API is accessible by using any REST client or programming platform that can issue HTTP requests with a basic HTTP authentication mechanism.

A sample request and response:

- **Request**

```
GET
https://<IP
address/hostname>:<port_number>/api/v2/datacenter/cluster/clusters
```

- **Response**

```
{
  "records": [
    {
      "key": "4c6bf721-2e3f-11e9-a3e2-
00a0985badbb:type=cluster,uuid=4c6bf721-2e3f-11e9-a3e2-00a0985badbb",
      "name": "fas8040-206-21",
      "uuid": "4c6bf721-2e3f-11e9-a3e2-00a0985badbb",
      "contact": null,
      "location": null,
      "version": {
        "full": "NetApp Release Dayblazer__9.5.0: Thu Jan 17 10:28:33
UTC 2019",
        "generation": 9,
        "major": 5,
        "minor": 0
      },
      "isSanOptimized": false,
      "management_ip": "10.226.207.25",
      "nodes": [
        {
          "key": "4c6bf721-2e3f-11e9-a3e2-
00a0985badbb:type=cluster_node,uuid=12cf06cc-2e3a-11e9-b9b4-
00a0985badbb",
          "uuid": "12cf06cc-2e3a-11e9-b9b4-00a0985badbb",
          "name": "fas8040-206-21-01",
          "_links": {
            "self": {
```

```
            "href": "/api/datacenter/cluster/nodes/4c6bf721-2e3f-11e9-
a3e2-00a0985badbb:type=cluster_node,uuid=12cf06cc-2e3a-11e9-b9b4-
00a0985badbb"
          }
        },
        "location": null,
        "version": {
          "full": "NetApp Release Dayblazer__9.5.0: Thu Jan 17
10:28:33 UTC 2019",
          "generation": 9,
          "major": 5,
          "minor": 0
        },
        "model": "FAS8040",
        "uptime": 13924095,
        "serial_number": "701424000157"
      },
      {
        "key": "4c6bf721-2e3f-11e9-a3e2-
00a0985badbb:type=cluster_node,uuid=1ed606ed-2e3a-11e9-a270-
00a0985bb9b7",
        "uuid": "1ed606ed-2e3a-11e9-a270-00a0985bb9b7",
        "name": "fas8040-206-21-02",
        "_links": {
          "self": {
            "href": "/api/datacenter/cluster/nodes/4c6bf721-2e3f-11e9-
a3e2-00a0985badbb:type=cluster_node,uuid=1ed606ed-2e3a-11e9-a270-
00a0985bb9b7"
          }
        },
        "location": null,
        "version": {
          "full": "NetApp Release Dayblazer__9.5.0: Thu Jan 17
10:28:33 UTC 2019",
          "generation": 9,
          "major": 5,
          "minor": 0
        },
        "model": "FAS8040",
        "uptime": 14012386,
        "serial_number": "701424000564"
      }
    ],
    "_links": {
      "self": {
        "href": "/api/datacenter/cluster/clusters/4c6bf721-2e3f-11e9-
```

```
a3e2-00a0985badbb:type=cluster,uuid=4c6bf721-2e3f-11e9-a3e2-
00a0985badbb"
        }
      }
    },
```

- *IP address/hostname* is the IP address or the fully qualified domain name (FQDN) of the API server.
- Port 443

  443 is the default HTTPS port. You can customize the HTTPS port, if required.

To issue HTTP requests from a web browser, you have to use REST API browser plugins. You can also access the REST API by using scripting platforms such as cURL and Perl.

Authentication

Unified Manager supports the basic HTTP authentication scheme for APIs. For secure information flow (request and response), the REST APIs are accessible only over HTTPS. The API server provides a self-signed SSL certificate to all clients for server verification. This certificate can be replaced by a custom certificate (or CA certificate).

You must configure user access to the API server for invoking the REST APIs. The users can be local users (user profiles stored in the local database) or LDAP users (if you have configured the API server to authenticate over LDAP). You can manage user access by logging in to the Unified Manager Administration Console user interface.

HTTP status codes used in Active IQ Unified Manager

While running the APIs or troubleshooting issues, you should be aware of the various HTTP status codes and error codes that are used by Active IQ Unified Manager APIs.

The following table lists the error codes related to authentication:

HTTP status code	Status code title	Description
200	OK	Returned on successful execution of synchronous API calls.
201	Created	Creation of new resources by synchronous calls, such as configuration of Active Directory.
202	Accepted	Returned on successful execution of asynchronous calls for provisioning functions, such as creating LUNs and files shares.

HTTP status code	Status code title	Description
400	Invalid request	Indicates input validation failure. User has to correct the inputs, for example, valid keys in a request body.
401	Unauthorized request	You are not authorized to view the resource/Unauthorized.
403	Forbidden request	Accessing the resource you were trying to reach is forbidden.
404	Resource not found	The resource you were trying to reach is not found.
405	Method Not Allowed	Method not allowed.
429	Too Many Requests	Returned when the user sends too many requests within a specific time.
500	Internal server error	Internal server error. Failed to get the response from server. This internal server error may or may not be permanent. For example, if you run a `GET` or `GET ALL` operation and receive this error, it is recommended that you repeat this operation for a minimum of five retries. If it is a permanent error, then the status code returned continues to be 500. If the operation succeeds, the status code returned is 200.

Recommendations for using the APIs for Active IQ Unified Manager

When using the APIs in Active IQ Unified Manager, you should follow certain recommended practices.

- All response content type must be in the following format for a valid execution:

```
application/json
```

- The API version number is not related to the product version number. You should use the latest version of the API available for your Unified Manager instance. For more information about Unified Manager API

versions, see the “REST API versioning in Active IQ Unified Manager” section.

- While updating array values using a Unified Manager API, you must update the entire string of values. You cannot append values to an array. You can only replace an existing array.
- You can use filter operators, such as pipe (|) and wild card (*) for all query parameters, except for double values, for example, IOPS and performance in the metrics APIs.
- Avoid querying objects by using a combination of the filter operators wild card (*) and pipe (|). It might retrieve an incorrect number of objects.
- When using values for filter, ensure that the value does not contain any `?` character. This is to mitigate risks of SQL injection.
- Note that the `GET` (all) request for any API returns a maximum of 1000 records. Even if you run the query by setting the `max_records` parameter to a value higher than 1000, only 1000 records are returned.
- For performing administrative functions, it is recommended that you use the Unified Manager UI.

Logs for troubleshooting

System logs enable you to analyze the causes of failure and troubleshooting issues that may arise while running the APIs.

Retrieve the logs from the following location for troubleshooting issues related to the API calls.

Log location	Use
`/var/log/ocie/access_log.log`	Contains all API call details, such as the user name of the user invoking the API, start time, execution time, status, and URL. You can use this log file to check the frequently-used APIs, or troubleshoot any GUI workflow. You can also use it to scale analysis, based on the execution time.
`/var/log/ocum/ocumserver.log`	Contains all API execution logs. You can use this log file to troubleshoot and debug the API calls.
`/var/log/ocie/server.log`	Contains all Wildfly server deployments and start/stop service related logs. You can use this log file to find the root cause of any issues occurring during the start, stop, or deployment of the Wildfly server.
`/var/log/ocie/au.log`	Contains acquisition unit related logs. You can use this log file when you have created, modified, or deleted any objects in ONTAP but they do not get reflected for the Active IQ Unified Manager REST APIs.

Job objects asynchronous processes

Active IQ Unified Manager provides the `jobs` API that retrieves information about the Jobs performed while running other APIs. you must know how asynchronous processing works using the Job object.

Some of the API calls, particularly those that are used for adding or modifying resources, can take longer to complete than other calls. Unified Manager processes these long-running requests asynchronously.

Asynchronous requests described using Job object

After making an API call that runs asynchronously, the HTTP response code 202 indicates the request has been successfully validated and accepted, but not yet completed. The request is processed as a background task which continues to run after the initial HTTP response to the client. The response includes the Job object anchoring the request, including its unique identifier.

Querying the Job object associated with an API request

The Job object returned in the HTTP response contains several properties. You can query the state property to determine if the request completed successfully. A Job object can be in one of the following states:

- `NORMAL`
- `WARNING`
- `PARTIAL_FAILURES`
- `ERROR`

There are two techniques you can use when polling a Job object to detect a terminal state for the task, either success or failure:

- Standard polling request: The current Job state is returned immediately.
- Long polling request: When the job state moves to `NORMAL, ERROR,` or `PARTIAL_FAILURES.`

Steps in an asynchronous request

You can use the following high-level procedure to complete an asynchronous API call:

1. Issue the asynchronous API call.
2. Receive an HTTP response 202 indicating successful acceptance of the request.
3. Extract the identifier for the Job object from the response body.
4. Within a loop, wait for the Job object to reach the terminal state `NORMAL, ERROR,` or `PARTIAL_FAILURES.`
5. Verify the terminal state of the Job and retrieve the Job result.

Hello API server

The *Hello API server* is a sample program that demonstrates how to invoke a REST API in Active IQ Unified Manager using a simple REST client. The sample program provides

you basic details about the API server in the JSON format (the server supports only `application/json` format).

The URI used is: `https://<hostname>/api/datacenter/svm/svms`. This sample code takes the following input parameters:

- The API server IP address or FQDN
- Optional: Port number (default: 443)
- User name
- Password
- Response format (`application/json`)

To invoke REST APIs, you can also use other scripts such as Jersey and RESTEasy to write a Java REST client for Active IQ Unified Manager. You should be aware of the following considerations about the sample code:

- Uses an HTTPS connection to Active IQ Unified Manager to invoke the specified REST URI
- Ignores the certificate provided by Active IQ Unified Manager
- Skips the host name verification during the handshake
- Uses `javax.net.ssl.HttpsURLConnection` for a URI connection
- Uses a third-party library (`org.apache.commons.codec.binary.Base64`) for constructing the Base64 encoded string used in the HTTP basic authentication

To compile and execute the sample code, you must use Java compiler 1.8 or later.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URL;
import java.security.SecureRandom;
import java.security.cert.X509Certificate;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import org.apache.commons.codec.binary.Base64;

public class HelloApiServer {

    private static String server;
    private static String user;
    private static String password;
    private static String response_format = "json";
    private static String server_url;
    private static String port = null;
```

```
    /*
     * * The main method which takes user inputs and performs the *
necessary steps
     * to invoke the REST URI and show the response
     */ public static void main(String[] args) {
        if (args.length < 2 || args.length > 3) {
            printUsage();
            System.exit(1);
        }
        setUserArguments(args);
        String serverBaseUrl = "https://" + server;
        if (null != port) {
            serverBaseUrl = serverBaseUrl + ":" + port;
        }
        server_url = serverBaseUrl + "/api/datacenter/svm/svms";
        try {
            HttpsURLConnection connection =
getAllTrustingHttpsUrlConnection();
            if (connection == null) {
                System.err.println("FATAL: Failed to create HTTPS
connection to URL: " + server_url);
                System.exit(1);
            }
            System.out.println("Invoking API: " + server_url);
            connection.setRequestMethod("GET");
            connection.setRequestProperty("Accept", "application/" +
response_format);
            String authString = getAuthorizationString();
            connection.setRequestProperty("Authorization", "Basic " +
authString);
            if (connection.getResponseCode() != 200) {
                System.err.println("API Invocation Failed : HTTP error
code : " + connection.getResponseCode() + " : "
                        + connection.getResponseMessage());
                System.exit(1);
            }
            BufferedReader br = new BufferedReader(new
InputStreamReader((connection.getInputStream())));
            String response;
            System.out.println("Response:");
            while ((response = br.readLine()) != null) {
                System.out.println(response);
            }
            connection.disconnect();
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }

    /* Print the usage of this sample code */ private static void
printUsage() {
        System.out.println("\nUsage:\n\tHelloApiServer <hostname> <user>
<password>\n");
        System.out.println("\nExamples:\n\tHelloApiServer localhost admin
mypassword");
        System.out.println("\tHelloApiServer 10.22.12.34:8320 admin
password");
        System.out.println("\tHelloApiServer 10.22.12.34 admin password
");
        System.out.println("\tHelloApiServer 10.22.12.34:8212 admin
password \n");
        System.out.println("\nNote:\n\t(1) When port number is not
provided, 443 is chosen by default.");
    }

    /* * Set the server, port, username and password * based on user
inputs. */ private static void setUserArguments(
            String[] args) {
        server = args[0];
        user = args[1];
        password = args[2];
        if (server.contains(":")) {
            String[] parts = server.split(":");
            server = parts[0];
            port = parts[1];
        }
    }

    /*
     * * Create a trust manager which accepts all certificates and * use
this trust
     * manager to initialize the SSL Context. * Create a
HttpsURLConnection for this
     * SSL Context and skip * server hostname verification during SSL
handshake. * *
     * Note: Trusting all certificates or skipping hostname verification *
is not
     * required for API Services to work. These are done here to * keep
this sample
     * REST Client code as simple as possible.
     */ private static HttpsURLConnection
```

```
getAllTrustingHttpsUrlConnection()    {        HttpsURLConnection conn =
null;        try {            /* Creating a trust manager that does not
validate certificate chains */            TrustManager[]
trustAllCertificatesManager = new                    TrustManager[]{new
X509TrustManager(){
     public X509Certificate[] getAcceptedIssuers(){return  null;}
     public void checkClientTrusted(X509Certificate[]
certs, String authType){}
     public void checkServerTrusted(X509Certificate[]
certs, String authType){}            }};            /* Initialize the
SSLContext with the all-trusting trust manager */
     SSLContext sslContext = SSLContext.getInstance("TLS");
sslContext.init(null, trustAllCertificatesManager, new
SecureRandom());
HttpsURLConnection.setDefaultSSLSocketFactory(sslContext.getSocketFactory(
));            URL url = new URL(server_url);            conn =
(HttpsURLConnection) url.openConnection();            /* Do not perform an
actual hostname verification during SSL Handshake.            Let all
hostname pass through as verified.*/
conn.setHostnameVerifier(new HostnameVerifier() {               public
boolean verify(String host, SSLSession                        session) {
return true;                }            });        } catch (Exception e)
{            e.printStackTrace();        }        return conn;    }

    /*
     * * This forms the Base64 encoded string using the username and
password *
     * provided by the user. This is required for HTTP Basic
Authentication.
     */ private static String getAuthorizationString() {
        String userPassword = user + ":" + password;
        byte[] authEncodedBytes =
Base64.encodeBase64(userPassword.getBytes());
        String authString = new String(authEncodedBytes);
        return authString;
    }

}
```

Copyright information

Trademark information