VERITAS

NetBackup Flex Appliance Management with REST API

Automated appliance administration

Introduction

Flex appliances are designed to minimize and simplify data protection management tasks. They feature a rich and user-friendly web-based user interface. Additionally, some support and management tasks can be performed using the Flex operating system shell. Flex appliances also support programmatic management methods via the Flex application programming interface (API). With many customers deploying appliances on a large scale, the need for flexible and comprehensive means of management automation has become even more apparent. With the introduction of the Flex 5 operating environment, the appliance application programming interface (API) has been revamped to deliver simplified, elastic and secure management options featuring a representation state transfer (REST) architecture.

The objective of this paper is to introduce the reader to the new appliance management options with some practical examples. The code snippets shown in this document are intended to illustrate the means of obtaining different access tokens using the cURL command line tool. Additionally, a sample request to create NetBackup primary and media server instances is also included – see Appendix.

REST API

The application programming interface provides means of communication and data exchange between different programs or software components. While there are multiple API architectural styles, REST has become widely adopted. REST deploys standard HTTP methods such as GET, POST, DELETE, PUT, PATCH which correspond to common Create, Read, Update, Delete (CRUD) database operations. See Table 1.

Operation	HTTP Method	Example
Create	POST	Add user
Read	GET	Retrieve users
Update	PUT	Edit password
Update	PATCH	Edit roles
Delete	DELETE	Remove users

Table 1.

The stateless nature of a REST-driven API means that client requests must contain all the data required for the request processing, including credentials for authentication and authorization. In practice this may translate to sending a username and password with every request. To eliminate the perpetual passing of user credentials with every REST call, token-based authentication has been developed. When a user successfully authenticates, the application issues an access token. This token is to be included in any future API requests, eliminating the need for continually passing user credentials. In addition to eliminating the need for authentication credentials, tokens also carry a payload which may include user identity and associated roles further simplifying not only authentication routine but authorization as well. The process of obtaining the access token begins with the client sending login credentials (1) – see Figure 1. The server (Flex appliance) authenticates the user and then generates a token, copies it, stores it and sends a copy to the requesting client (2). A client passes a token with every request (3). For a given request, if the incoming token is valid, the request is processed (4).

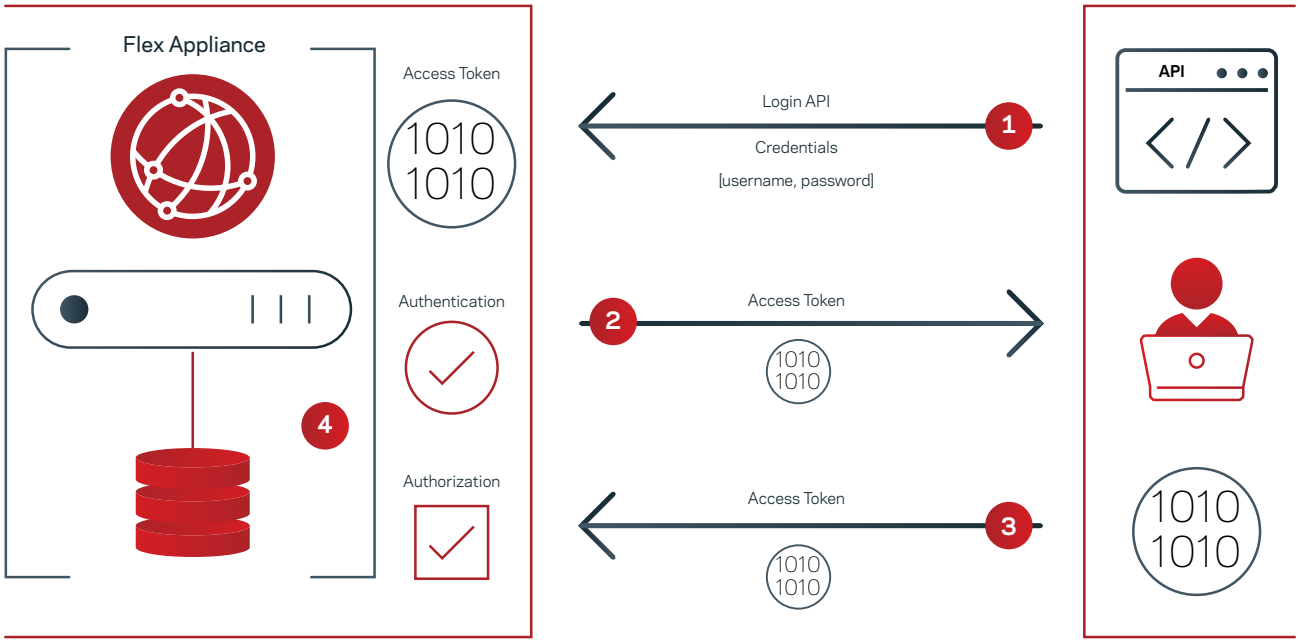

Figure 1.

Access tokens have a limited lifetime, referred to as time-to-live (TTL). A token becomes invalid once its TTL expires.

Flex Appliance Tokens

Flex operating environment supports three different types of API tokens:

- JSON Web Token – JWT
- Service Token
- Personal Token

Parallel usage of each token is not mutually exclusive, which means that using JSON Web Token does not preclude generation and usage of personal and service tokens. Each type of token is meant to provide convenient and secure means of appliance management and can be applied in different deployment scenarios depending on customer requirements.

Table 2 shows the main differences between the three types of tokens supported by Flex appliances.

Function	JSON Web Token	Service Account Token	Personal Token
Login API Required	Yes	Yes	No
Alter Token's TTL	No	No	Yes
Revoke Token	No	No	Yes
Inherits Privileges	Yes	Yes	Yes
Multifactor Authentication Required (if enforced)	Yes	Yes	No
Token Tracking	No	No	Yes

Table 2.

JSON Web Token

JSON web token (JWT) provides essential means of authentication and authorization on Flex appliance. The default JWT is valid for 15 minutes and inherits the role or roles associated with the account used to obtain the token. In the examples below, the received token will have admin role privileges since the admin account credentials are utilized to obtain it.

This token is obtained with the /v1/login **POST** call. The example curl command is shown below:

```
curl --location 'https://<console-hostname>/api/v1/login' \
--request POST \
--insecure \
--header 'Content-Type: application/json' \
--data-raw '{
    "username": "user",
    "password": "password"
}'
```

Where `<console-hostname>` is appliance console (Web UI) IP address or hostname and username and password values are the account used to login and its credentials.

If multi-factor authentication (MFA) is enabled, the login request must also include the value of time-based one-time password (`totp`).

```
curl --location 'https://<console-hostname>/api/v1/login' \
--request POST \
--insecure \
--header 'Content-Type: application/json' \
--data-raw '{
    {
    "username": "user",
    "password": "password",
    "totp": "016610"
}'
```

An example of data returned by the login API is shown below.


```
{
    "username": "user",
    "token":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MjQxMDIzNDcsImZpbmdlcnByaW50cyI6bnVs(…)[deleted]",
    "refreshToken":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MjQxMjMwNDcsImdpZCI6IjAwMDAwMDAwLTAw
(…) [deleted] ",
    "userId": "6fcbec2e-dc53-4b1c-9b06-d3dea2ff18d6"
}
```


A "refreshToken" is returned along with the "token". When the token expires, the `/v1/token/renew` POST call should be used to receive another valid access token. This method does not require sending the user credentials. Instead, both the access and refresh tokens must be included in the request's header. See the `curl` command example below:

```
curl --location 'https://<console-hostname>/api/v1/token/renew' \
--request POST \
--insecure \
--header "x-auth-token:  $TOKEN" \
--header "refresh-token: $REFRESH_TOKEN"
```

In this example, `$TOKEN` and `$REFRESH_TOKEN` are environment variables corresponding to the values of "`token`" and "`refreshToken`" respectively. The Flex Appliance will return a new, valid JWT access token.

The refresh token has a six-hour time-to-live (TTL). The regular user account is limited to a combined total of ten Web UI and API concurrent sessions. The number of active sessions can be obtained using the Web UI (User management panel) or through the API with the GET `/v1/sessions/user` call. See the example output below showing three sessions for two different users.


```
  {
        "id": "9b3c9994-0d32-4885-beee-ff3b9a56ff52",
        "uid": "71e38ef8-40d4-4e1b-b75a-247875b104dc",
        "not-before": 1725379205,
        "not-after": 1725400805,
        "source": "<ip_address>"
    },
    {
        "id": "f8ee9d38-955d-427b-a36a-241c147c9be9",
        "uid": "6fcbec2e-dc53-4b1c-9b06-d3dea2ff18d6",
        "not-before": 1725378394,
        "not-after": 1725399994,
        "source": "<ip_address>"
    },
   {
        "id": "3857524a-d73b-4f54-93aa-1a2b7445ed0e",
        "uid": "6fcbec2e-dc53-4b1c-9b06-d3dea2ff18d6",
        "not-before": 1725379029,
        "not-after": 1725400629,
        "source": "<ip_address>"
    }
```

Service Token

With Flex 5, the "service" account has been introduced. This type of account is designed specifically for API-based appliance management and automation. The console (Web UI) login for service accounts is not permitted. The intent of the service account is to provide customers with easy means to create API access tokens where a token's TTL can be defined by the requestor but cannot equal or exceed the maximum value set by the security administrator. The token(s) associated with a service account are not tracked by Flex appliance and cannot be revoked. The ability to control the TTL eliminates the need for logout, as the short-lived tokens will expire and the requested TTL can be just long enough to perform the management tasks. The service account can have any number of tokens and active sessions. The maximum TTL for tokens to-be-issued by a service account can be adjusted.

In addition, service tokens can be used in combination with external, centrally-controlled authentication mechanisms such as Active Directory or LDAP to enable efficient password security management and enforcement. See Figure 2.

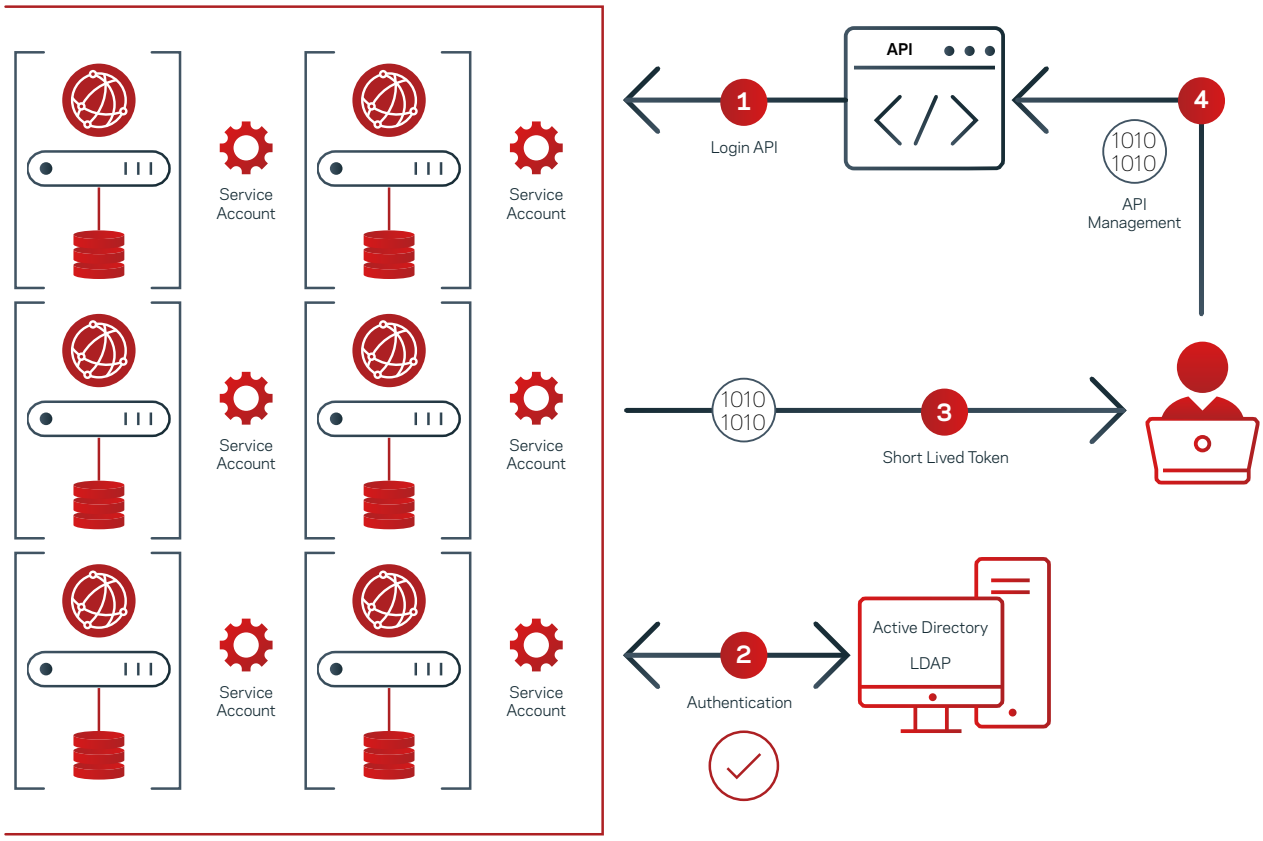

Figure 2.

The process of obtaining the service account token begins with the user requesting sending credentials and the token's TTL (1). The user is authenticated by an external AD or LDAP server (2). If the authentication is successful and the requested TTL is less than maximum TTL defined for the service account, the appliance issues access token (3). In the last step, the service token is used to connect and execute automation scripts (4).

Any local user account can be converted to the service account. During the conversion process the service token's maximum time-to-live (TTL) must be defined. Reverting a service account to user account is not possible.

The service token inherits privileges (roles) assigned to the associated service account.

Conversion of the user account to service account can be done via Web UI or using the `/v2/users/{uid}/token-ttl-limit` **PUT** call where `{uid}` is the user ID of the service account. If the `{uid}` refers to a "regular" user account, then this account will be converted to the service account, otherwise the token's TTL for the already existing service account will be modified. For example:

```
curl --location 'https://<console-hostname>/api/v2/users/{uid}/token-ttl-limit' \
--request PUT \
--insecure \
--header 'X-Auth-Token: $TOKEN' \
--header 'Content-Type: application/json' \
--data '300'
```

where `300` is the time-to-live value in seconds for the service token; the value of `{uid}` is obtained with `/v1/users` GET call.

The service token is obtained with `/v1/login` POST method. The "`time-to-live`" parameter in the request body must be specified and be less than maximum TTL for the service token. Using the maximum permitted TTL value from previous example, a requested token's TTL must be less than 300 seconds.

In the example below, the requested service token's TTL is 240 seconds, where `serviceAccount` is the service account issuing the token

```
curl --location 'https://<console-hostname>/api/v1/login' \
--request POST \
--insecure \
--header 'Content-Type: application/json' \
--data-raw '{
    "username": "serviceAccount",
    "password": "password",
    "time-to-live": 240
}'
```

The issued service token inherits roles from the associated account. The refresh token is not generated with the service token.

Personal Token

The personal token, also introduced in Flex version 5 of the Veritas operating environment, lets appliance users create API access tokens associated with their account. This token inherits the roles (privileges) of the owner's account. There can be only one personal token per account. Like to the service account token, TTL for a personal token may be set and modified. Possession of a personal token is all that is needed for appliance management, unlike a service token, which requires authorization (login) before it can be issued. This feature makes personal token an excellent choice for environments where multifactor authentication (MFA) is enforced, eliminating the need for MFA code orchestration for the API-based authentication. Each personal token is managed and tracked by the appliance. It can also be revoked, reissued and its expiration can be modified.

Figure 3 depicts an example environment where a data protection administrator can manage multiple appliances with personal tokens without the need to have the login credentials for managed appliances.

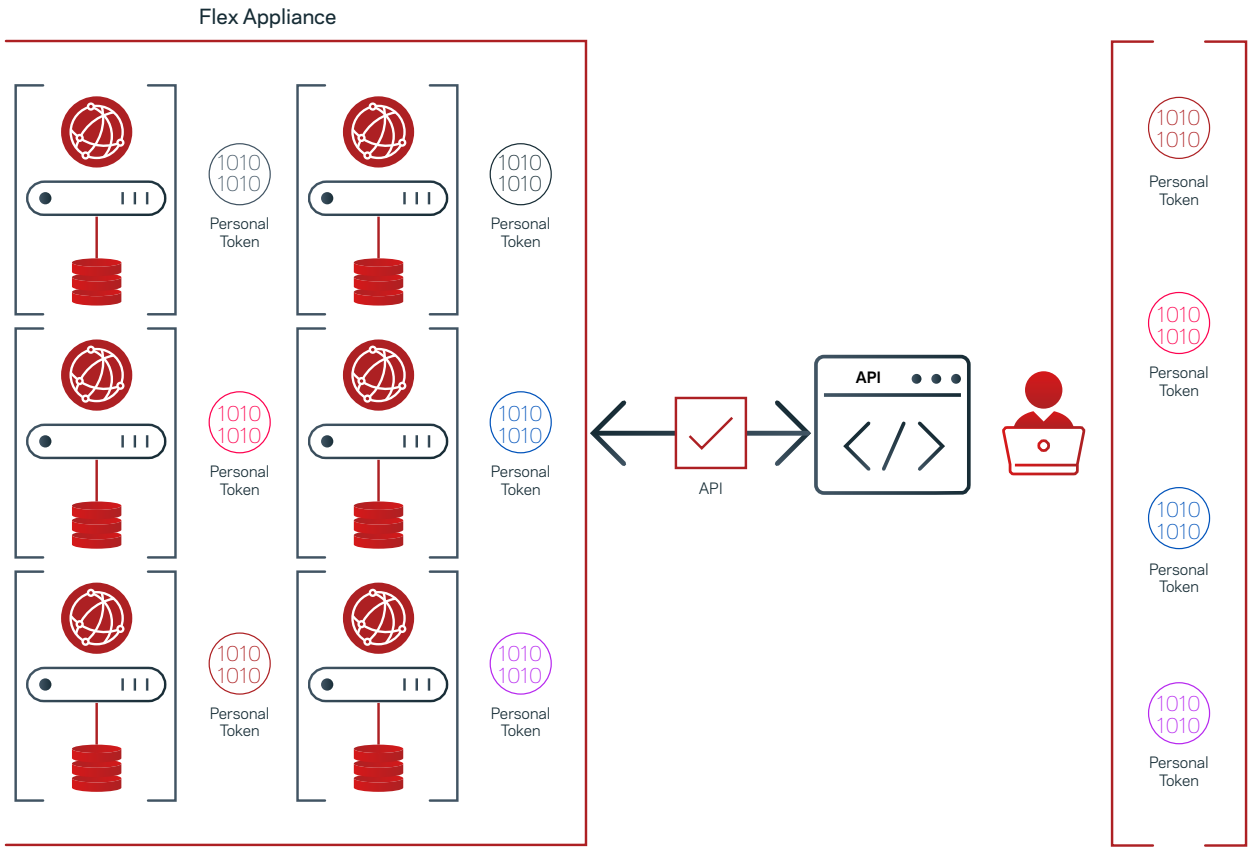

Figure 3.

To generate personal token use Web UI or a **POST** login call with `revocable` and `time-to-live parameters`:

```
curl --location 'https://<console-hostname>/api/v1/login' \
--insecure \
--header 'Content-Type: application/json' \
--data-raw '{
    "username": "user",
    "password": "password",
    "revocable": true,
    "time-to-live": 3600
}'
```

After the token is issued it must be saved because it cannot be retrieved later. Any following API calls using personal token will not require login.

To revoke the personal token, simply execute the **DELETE** call:

```
curl --location 'https://<console-hostname>/api/v2/users/{uid}/tokens' \
--insecure \
--request DELETE \
--header 'X-Auth-Token: eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjAsImZpbmdlcnByaW50cyI6bnVs(…)
[deleted]'
```

where X-Auth-Token value in the header is the personal token to be deleted.

Summary

Flex appliances provide modern, secure and reliable means of protecting data. They are the pillar of the data protection environment. With the refreshed REST application programming interface, secure appliance automation and management are simpler to develop and implement.

Enhancements introduced in Flex 5 simplify the development and implementation of appliance automation and management at scale.

The new and revamped API gives customers greater flexibility in selecting the secure and convenient management methods best fitting their data protection environment.

Appendix

Flex Appliance REST API documentation:

https://sort.veritas.com/public/documents/FAPP/5.0/veritas5360/productguides/html/flex-restapis-5.0/#/Appliance%7CHealth%20monitoring/getHealthHardwareStatus

Please note that with every release of Veritas Flex operating environment corresponding version of Flex API is also changed with possible syntax improvements and new calls introduced.

Always use the documentation appropriate to the Flex version on your appliances.

Examples:

Create primary server instance

```
{
    "application.name":"NetBackupMaster",
    "application.version":"10.3.0.1",
"network": [
        {
            "id":"hostname",
            "value":"primary1"
            },
        {
            "id":"interface",
            "value":"bond1"
            },
        {
            "id":"ipaddress",
            "value":"10.x.y.z"
            },
        {
            "id":"tenant",
            "value":"9110483505358597527"
            },
        {
```

```
            "id":"domainname",
            "value":"domain.com"},
        {
            "id":"nameservers",
            "value":"10.x.y.z"
        },
        {
            "id":"searchdomains",
            "value":"domain.com"
        },
        {
            "id":"etchosts",
            "value":""
            }
    ],
"volume": [
        {
            "id": "logs",
            "value": "30GB"
        },
        {
            "id": "catalog",
            "value": "30GB"
        }
    ]
}
```

Create media server instance

```
{
    "application.name":"NetBackupMedia",
    "application.version":"10.3.0.1",
"network":[
        {
            "id":"hostname",
            "value":"media1"
            },
        {
            "id":"interface",
            "value":"bond1"
            },
        {
```

```
            "id":"ipaddress",
            "value":"10.x.y.z"
            },
        {
            "id":"tenant",
            "value":"9110483505358597527"
            },
        {
            "id":"domainname",
            "value":"domain.com"},
        {
            "id":"nameservers",
            "value":"10.x.y.z"
        },
        {
            "id":"searchdomains",
            "value":"domain.com"
        },
        {
            "id":"etchosts",
            "value":""
            }
    ],
"volume": [
        {
            "id": "msdpdata",
            "value": "250GB"
        },
        {
            "id": "advdisk",
            "value": "200GB"
        },
        {
            "id": "staging",
            "value": "100GB"
        }
    ],
"envvars": [
        {
            "id":"ENV_NB_MASTER",
            "value": "primary1.domain.com"
        },
```

```
        {
            "id": "ENV_NB_CATYPE",
            "value": "0"
        },
        {
            "id": "ENV_NB_CAFPRN",
            "value": "CA:D7:43:9A:93:7A:DF:50:31:45:CE:9F:B5:90:6D:E5:4F:C4:4A:7B"
        },
        {
            "id": "ENV_NB_SECRETTOKEN",
            "value": ""
        }
    ]
}
```

Where `ENV_NB_MASTER` is the primary server hostname and `ENV_NB_CAFPRN` is the value of the NetBackup primary server Certificate Authority fingerprint and the following sample disk space allocations:

- Primary Server Logs (`logs`): 30 GB
- Primary Server Catalog (`catalog`): 30 GB
- Media Server MSDP (`msdp`): 250 GB
- Media Server AdvanceDisk (`advdisk`): 200 GB
- Media Server Staging (`staging`): 100 GB

VERITAS

2625 Augustine Drive
Santa Clara, CA 95054
+1 (866) 837 4827
veritas.com

For global contact information visit:
veritas.com/company/contact

V1214 10/24