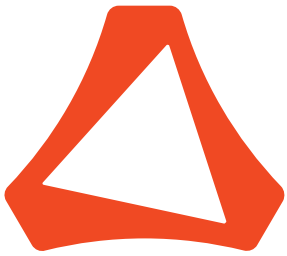

ALTAIR

ONLY FORWARD

Altair Accelerator 2024.1.0

Tutorial: Using Accelerator's REST API to Submit and List Jobs

Contents

Use Altair Accelerator's REST API to Submit and List Jobs

1

This chapter covers the following:

Overview

Altair Accelerator supports a Representational State Transfer (REST) API that enables a developer to submit, monitor, and control batch jobs as well as query the batch scheduler queues, jobs, and hosts in useful ways. A REST API is based on a URL name space targeted by HTTP operations like POST and GET to submit jobs or make queries. REST APIs are popular ways for developers to write portable applications, both UI and command line, that can interface to the Accelerator queue over secure HTTPS connections. This tutorial will introduce key concepts and will also provide working example Python programs that interface to Accelerator via REST.

Developers use a variety of languages in web clients when interfacing to a REST API, including Python, PHP, Java, Node.js and others. This tutorial will focus on the use of Python in example code.

For more specific information, see REST API in the *VOV Subsystem Reference Guide*.

Also in this Tutorial

Key REST Concepts

The components of a REST operation are the following:

- Resource Path
- HTTP Verb
- Body
- Header

The URLs in the application's REST URL name space provide the targets for the read (GET), write (POST), or update (PATCH) HTTP request types. The parameters of the REST operations always include a URL and an access token that authorizes access to the services. The following sections describe more about these concepts.

Resource Path

The resource path describes the object to be acted upon. It is in the form of a URL with the following structure:

```
BASE_URL + "/api/" + VERSION + "/" + REST
```

`BASE_URL` is the initial part of the URL for the Accelerator queue to access. For example, if the Accelerator NC_QUEUE environment variable was set to `My_Test_Queue`, then use the `nc cmd vovbrowser` command to obtain a BASE_URL.

The VERSION is the REST API version, currently v3.

REST is the remaining part of the URL that applies to the specific types of objects you will be referencing.

The resource path might look something like this when you are referencing the version information about the Accelerator queue scheduling server known as vovserver.

```
https://server.domain.myco.com:6330/api/v3/project/1/version
```

HTTP Verb

The verb describes the action to take regarding the resource.

POST	Creates a resource
GET	Retrieves one or more resources
PUT	Updates or controls a resource
PATCH	Updates a resource
DELETE	Deletes a resource

Body

The body of a REST request defines parameters and options that apply to the action being taken. A POST or PUT request has a body. A GET or DELETE request has no body. A Python dictionary data structure is the usual format for the body, and it consists of a set of keywords with associated values.

Header

The header contains metadata about the message being sent to the server which includes, importantly, an *access token*. REST HTTP requests are initiated across the network from a node other than the one on which the Accelerator server is running. The access token provides the assurance that the sender has permission to request the REST action on the server.

An authentication request type is available via a POST that supplies a username and password to the server, to which the server responds with the access token. When a REST request comes in later with this access token, the server can quickly and reliably extract the user identity and permissions as a prelude to processing the request. Accelerator REST utilizes a style of access token known as JSON Web Tokens (JWTs).

Get Ready for REST

REST API usage prerequisites are described in this section. These prerequisites will enable REST v3 applications to run with Accelerator. Some of the prerequisites apply to the Accelerator queue configuration and others apply to the host where the REST application will run, also called the "submit host".

Prerequisites for the Accelerator queue:

- The web port must be configured. In version 2021.2.0 through 2022.1.1, the web provider must be set to "internal". See the `-webport` and `-webprovider` options in the -h help screen documentation for VOV project start commands like `ncmgr start` and `lmmgr start`.
- **Optional:** SSL/TLS should be enabled for security reasons. Since passwords are passed with HTTP authorization requests to vovserver, the security of the connection is important. Enable SSL/TLS for the NC queue as follows:
  1. Add this line to `SWD/policy.tcl`:

     ```
     set config(ssl.enable) 1
     ```

  2. If REST requests will be sent from Python scripts running on CentOS 7 or earlier, TLS 1.2 will be used by the REST application you are writing. Configure vovserver to accept TLS 1.2 protocol by adding the following line to `SWD/policy.tcl`:

     ```
     set config(http.minSSLVersion) "TLSv1.2"
     ```

  3. **Optional:** Append a line to `SWD/setup.tcl` that sets VOV_HOST_HTTP_NAME to the fully qualified host name (FQHN) where vovserver is running. The FQHN is the output of `hostname -f`.

     ```
     setenv VOV_HOST_HTTP_NAME FQHN
     ```

  4. **Optional:** For a full HTTPS security, a CA-signed domain-wide SSL certificate is installed in `$VOVDIR/local/ssl` or a host-specific CA-signed SSL certificate is installed in `SWD/config/ssl`. This will allow your REST network traffic to be fully secured. See Advanced REST Usage for more about this.
  5. Reread and activate the changed server configuration parameters via `vovproject reread` or similar commands.

Prerequisites for the submit host:

- The submit host must have network access to the server running the NC queue.
- Python version 3 or higher is required to use the `vov_rest_v3.py` REST access library module.
- The Python "requests" package must be installed.
- Copy `vov_rest_v3.py` from `$VOVDIR/scripts/python/vov_rest_v3.py` to the directory on the submit host where your Python application resides.

Example Application: REST 101

Programs that use the Accelerator REST API can do so in two ways. The preferred approach is to utilize the Python library module "vov_rest_v3", which is provided with Accelerator software packages in the file `vov_rest_v3.py`. This Python module simplifies coding by hiding details of the HTTP operations that interface directly to the low level REST API. The vov_rest_v3 module requires Python version 3 or higher.

For more advanced users, a later section of this guide shows how to interface directly to the REST API. Direct use can give you more control over the following:

- Management of JWT access tokens
- Policy of your application regarding insecure HTTPS configurations
- Control over the use of connection "keep-alive"

Direct REST API interface will be covered later. This approach works with Python version 1 and higher.

To get a quick start using REST, examine a simple "REST 101" example Python program that queries REST to return the name of an NC queue, shown below. The lines of the program are numbered and annotated with explanations.

```
 1   #!/usr/bin/python3
   2   #
   3   # nc_rest_101.py
   4   #
   5   import os, sys, getpass, json
   6   import vov_rest_v3
   7
   8   url = os.environ['NC_URL']
   9
  10   vrest = vov_rest_v3.VOVRestV3()
  11   vrest.authorize(url, getpass.getpass('Password:') )
  12   r = vrest.submitRequest("GET", url + "/api/v3/project/1/project")
  13
  14   print ( json.dumps(json.loads(r), indent=2) )
```

Here is a line-by-line guide to the above program.

Line 1	This Python program is compatible with Python 3.
Line 6	Import the vov_rest_v3 Python module that comes with the Accelerator product. Copy it locally from $VOVDIR before running the Python program.
Line 8	Pull the URL for the NC queue out of an environment variable. The URL is what `nc cmd vovbrowser` shows.
Line 10	Allocate `vrest`, an instantiation of the VOVRestV3 Python class.
Line 11	The `authorize()` method function authenticates using the password and allocates a JWT token stored in the object. This example obtains your user password by interactive prompt.

> **Note:** This example uses username/password authentication. To use key-based authentication, see .

Line 12	The HTTP get request is sent, returning a string containing the HTTP response data in JSON format.
Line 14	The HTTP response string is parsed into JSON and pretty-printed.

Here is a terminal session that shows how to run the program.

```
% echo $NC_QUEUE
My_NC_QUEUE
% export NC_URL=`nc cmd vovbrowser`
% echo $NC_URL
http://myhost:6330
% cp $VOVDIR/scripts/python/vov_rest_v3.py .
% python3 nc_rest_101.py
Password:
{
  "startrow": 1,
  "endrow": 1,
  "query": "SELECT project FROM 1",
  "errormsg": "",
  "columns": [
    {
      "col": 0,
      "field": "project"
    }
  ],
  "rows": [
    [
      "My_NC_QUEUE"
    ]
  ]
}
```

Example Applications: Job Submit and List

To demonstrate the utility of the REST API, here are two simple Python programs that imitate the function of the `nc run` and `nc list` commands that are familiar CLI tools from the Accelerator product. The source code for these tools, named `nc_run.py` and `nc_list.py`, is found on the following pages.

These REST programs only require that the NC_URL environment variable be set to the Accelerator queue URL, as returned by `nc cmd vovbrowser`. Here is a terminal image that demonstrates the simple REST tools.

```
% cp $VOVDIR/scripts/python/vov_rest_v3.py .
% export NC_URL="`nc cmd vovbrowser`"
% ./nc_run.py sleep 33
Password:
New job is 45080
% ./nc_run.py sleep 66
Password:
New job is 45083
% ./nc_list.py
Password:
```

```
       ID   STATUSNC    PRIORITYP       HOST        COMMAND
000045080    Running       normal    myhost   vw sleep 33 > JOBLOG.185633
000045083    Running       normal    myhost   vw sleep 66 > JOBLOG.903723
```

nc_run.py

```
#!/usr/bin/python3
#
# nc_run.py
#
# Usage
#
#     export NC_URL=<URL FOR NC QUEUE>
#     ./nc_run.py <command> [args]
#

import os, sys, random, json, getpass
import vov_rest_v3

def getMyPassword():
    return getpass.getpass('Password:')

# Main body
nc_url = os.environ["NC_URL"]
scheme = nc_url.split(":")[0]
hostport = nc_url.split("/")[2]
url = "{0}://{1}".format(scheme, hostport)
command = " ".join(sys.argv[1::])

vrest = vov_rest_v3.VOVRestV3()
vrest.authorize(url, getMyPassword())

# Job attributes - required
VOV_JOB_DESC = {
    "command" : command,
    "logfile" : "JOBLOG." + str(random.randint(100000,999999)),
    "rundir" : os.getcwd(),
    "env" : "BASE",
}

# Job attributes - optional / User specified
VOV_JOB_DESC.update( {
    "priority,sched" : 4,
} )
r = vrest.submitRequest("POST", url + "/api/v3/jobs", jsonData=VOV_JOB_DESC)
print ("New job is %s" % json.loads(r)["jobid"])
```

nc_list.py

```
#!/usr/bin/python3
#
# nc_list.py
#
# Usage
#
#       export NC_URL=<URL FOR NC QUEUE>
#       ./nc_list.py
#

import os, sys, json, getpass
```

```
import vov_rest_v3

def getMyPassword():
    return getpass.getpass('Password:')

def listJob(vr, url):
    query = ( url + '/api/v3/query'
              + '?select=id,statusnc,PRIORITYPP,host,command'
              + '&from=System:User:' + os.environ['USER'] )
    response = vr.submitRequest("GET", query)
    return response

def prettyPrint( text ):
    dd = json.loads(text)
    for ii in range(0, len(dd['columns']) ) :
        sys.stdout.write("%9.9s " % dd['columns'][ii]['field'])
    sys.stdout.write("\n")
    if ('rows' not in dd):
        return
    for rr in range (0, len(dd['rows']) ) :
        row = dd['rows'][rr]
        for ii in range(0, len(dd['columns']) ) :
            if (ii < len(dd['columns'])-1):
                sys.stdout.write("%9.9s " % str(row[ii]))
            else:
                sys.stdout.write("%10.30s" % str(row[ii]))
        sys.stdout.write("\n")

#
# Main body
#
nc_url = os.environ['NC_URL']
scheme = nc_url.split(":")[0]
hostport = nc_url.split("/")[2]
url = "{0}://{1}".format(scheme, hostport)

vrest = vov_rest_v3.VOVRestV3()
vrest.authorize(url, getMyPassword())
json_text = listJob(vrest, url)
prettyPrint(json_text)
```

Launch Jobs with Non-default Options

The `nc_run.py` example program shown earlier can submit an Accelerator job with no customizations. In reality, users often need to specify non-default properties for the jobs they submit. Examples are resources, slot counts, auto-kill times, and numerous other options that can be seen on the `nc run -h` CLI command help information and in the documentation with the *Altair Accelerator Administrator Guide*.

To view available job launch options, find the VOV_JOB_DESC table of options in Define Jobclasses of the *Altair Accelerator Administrator Guide*. Here is an excerpt of that table:

Field in Array	Description
autokill	Set the autokill flag (option -kill)
check,directory	Set it to 0 to disable checking of canonicalization of current directory (option -D)
env	Environment of the job (option -e). Set this to "" or to DEFAULT to force the use of an environment snapshot.
force	Force the job to be rescheduled (option -F)
group	Group the job belongs to (options -g and -G)
inputs	List of input files (dependencies) (option -i)
priority, default	Default priority (NOT USED)

Although the above document has good descriptions of the fields/keys that can be provided to job launch requests, some of those fields are usable in Accelerator jobclass definitions but not in REST job requests. To see the precise list of fields/keys that are supported in REST requests, see the Swagger docs described in the REST Request Detailed Documentation section later in this tutorial.

You can modify the `nc_run.py` script previously shown (or the `nc_run2.py` script in an upcoming section) with additional options specified as key-value pairs in the VOV_JOB_DESC Python dictionary. For example, here is the modification to `nc_run.py` that specifies an auto-kill time of 60 seconds. The Python dictionary "update" method function provides a convenient way to do that.

```
VOV_JOB_DESC.update( {
   "autokill"    : 60,
 } )

 vrest.submitRequest("POST", resturl, jsonData=VOV_JOB_DESC)
```

The vov_rest_v3.py Python Library Module

The Accelerator software package provides a Python library module called `vov_rest_v3.py` to make REST API usage from Python more convenient. The module implements a VOVRestV3 Python class with member functions described in the following text box. These functions hide the details associated with authenticating, session handling, and error handling. More example programs follow that utilize this convenient Python library layer.

VOVRestV3 Python Class Description

The VOVRestV3 Python class provides an interface to the Accelerator v3 REST API.

Location

$VOVDIR/../common/scripts/python
import vov_rest_v3

Member Functions

authorize (url, username='', password='')

The vovserver scheduling server authenticates a user for a VOVRestV3 and obtains a validating access token behind the scenes, storing it in the VOVRestV3 object.

Positional Arguments	url – (string) the URL as returned by `nc cmd vovbrowser`
	password -- (string) the current user password
Keyword Arguments	username – (string) user name, usually the same as the Linux user name known to vovserver. If this argument is not provided it defaults to the current user identified by $USER
Return Value	None. Raises exceptions upon failure.

submitRequest (method, url, queryParams={}, jsonData={})

Submits a REST/ HTTP request to the server.

Positional Arguments	method – (string) one of: "GET", "POST", "PUT", "DELETE", "PATCH".
	url – (string) the URL as returned by `nc cmd vovbrowser -url RESTPATH`.
	Examples of RESTPATH are: `/api/v3/jobs` or `/api/v3/projects/1`.
Keyword Arguments	queryParams – (dictionary) keywords and values in string fromat for GET requests
	jsonData – (dictionary) keywords and values in string fromat for POST requests
Return Value	(string) The HTTP request response text.

getJWT()

Retrieve the JSON Web Token (JWT) for the object.

Return Value: (string) the JWT string

setJWT(token)

Replace the JSON Web Token (JWT) for the object.

Positional Arguments	token – (string) the replacement JWT to be remembered in the object and used for subsequent REST requests
Return Value	None.

REST Request Detailed Documentation

The Accelerator REST v3 API interface is described in detail under the Swagger documentation that comes with the Accelerator software. To browse the Swagger REST documentation, browse to the URL displayed as the output of this command:

```
nc cmd vovbrowser -url /html/vovrest.html
```

The information at this page will help you construct a valid REST request.

For example, browse on the Swagger documentation page to the “jobs” object. You will see that a POST request is used, and the REST URL segment is “/api/v3/jobs”.

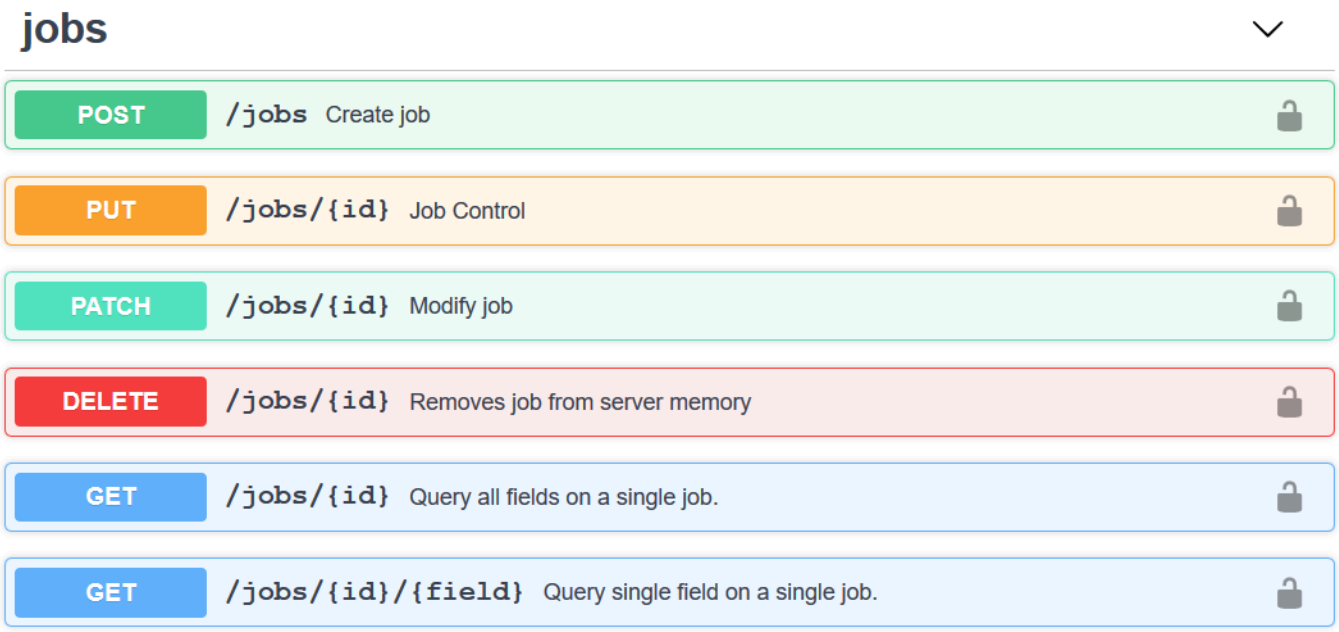

Figure 1: REST Object-Verb Reference

Click on the POST, PUT, PATCH, DELETE, or GET buttons for more details about the required parameters and supported dictionary keys and values for each request type. This gives you what you need to know to construct a complete and working job submit request via Python or curl.

Issuing REST Requests from the Swagger Web UI

The Swagger Rest API page provides a learning interface that helps you formulate and execute well-formed REST requests to the associated NC queue. The web UI page shows a construction of the REST request and response text in curl command and JSON language formats.

Here are the steps to formulate and execute a REST request:

1. Access the VOV REST API page in a browser. The URL is shown by the command:

   ```
   nc cmd vovbrowser -url /html/vovrest.html
   ```

2. If SSL/TLS is enabled for the NC queue, then select the https URL in the "Server" drop-down menu on the left side of the page near the top.
3. Scroll down to the Object-Verb section for the desired REST request.
4. Click on **Try it Out** on the right.
   This activates a web form on the page to accept your specified parameters for a REST request before executing it.
5. Edit the parameters for your REST request in the web form.
   Example A

   If you are selecting a GET operation on a job object, the job id and sometimes a job object field name are required. Enter these items in the web form.

   Example B

   If you are selecting a POST operation on a job object to submit a job to Accelerator, the parameters are more extensive. Edit the job creation parameters box. A template window in the web form illustrates the format, which is consistent with a Python dictionary syntax with keyword/value pairs. Replace the template with your desired job creation specifications.

   Most of the keyword/value pairs in the template for job creation are optional, but a few are required. A simple and minimal working example job creation parameter list follows. To specify other properties of a job, choose some additional keywords from the template text that appears pre-populated in this sub-window.

   ```
   {
          "command" : "sleep 60",
          "logfile" : "JOBLOG.01",
          "rundir"  : "/tmp",
          "env"     : "BASE"
   }
   ```

6. Click the blue **Execute** button.
   This sends the REST request.
7. Scroll down to see the Server Response. A code of 200 - 299 indicates success.
8. Troubleshoot if the request failed. Common errors and remedies include:
   1. If error text is "Bad Request", then check that the right URL is selected in the Servers drop-down menu at the upper left.
   2. If an authentication error is seen, logout of the web UI and then log in again so the Swagger page gets a fresh JWT access token.
   3. If an error with "Error: Bad Request" is seen, check the specified text in the Job Control Parameters sub-window. A common mistake is to add an invalid comma after the last keyword/value pair.

4. A server error is returned if you specify a job logfile that is the same as another job in the system. Change the logfile name and retry the request.

Advanced REST Usage

This section will cover some more advanced REST API topics: management of JWT access tokens, HTTPS secure and insecure connections, and connection keep-alive. The example application, `nc_info.py`, imitates the familiar Accelerator CLI command `nc info`, which returns information about a job. This application will interface to the REST API directly, instead of using the vov_rest_v3 module interface layer.

This REST application requires the NC_URL environment variable to be set to the Accelerator queue URL, as returned by `nc cmd vovbrowser`. Also, this Python program needs JWT-handling module `getToken.py` from the next section in this guide. Create that python file before running `nc_info.py`.

Here is a shell session that shows how to run `nc_info.py`. In this example, two jobs are started, and an invocation of `nc_info.py` will query and display information about the two running NC jobs.

```
% nc run -v 2 sleep 123
Job <000001138> is submitted
% nc run -v 2 sleep 456
Job <000001143> is submitted
% export NC_URL=`nc cmd vovbrowser`
% echo I will need `ls getToken.py`
I will need getToken.py
% ./nc_info.py 000001138 000001143
Password:
Job             000001138
User,Group      user99,/time/users.user99
Command         vw sleep 123 > vnc_logs/20211012/132628.122875
Status          Running
   Host         myhost
   Duration     31s

Job             000001143
User,Group      user99,/time/users.user99
Command         vw sleep 456 > vnc_logs/20211012/132633.122899
Status          Running
   Host         myhost
   Duration     26s
```

nc_info.py

```
#!/usr/bin/python
#
# nc_info.py
#
# Usage
#
#      export NC_URL=<URL FOR NC QUEUE>
#      ./nc_info.py JOB_ID [JOB_ID ...]
#

import os, sys, json, requests
from getToken import getJWT

def getMyPassword():
    import getpass
    return getpass.getpass('Password:')

def infoPrint( text ):
    dd =  json.loads(text)
    id    = find(dd, 'ID')
    user  = find(dd, 'USER')
    group = find(dd, 'GROUP')

    print ("%-16s%s" % ("Job", id) )
    print ("%-16s%s,%s" % ("User,Group", user, group))
    print ("%-16s%s" % ("Command", find(dd, 'COMMAND')))
    print ("%-16s%s" % ("Status", find(dd, 'STATUSNC')))
    print ("%-16s%s" % ("   Host", find(dd, 'HOST')))
    print ("%-16s%s" % ("   Duration", find(dd, 'DURATIONPP')))

def find(jobdump, key):
    for c in range (0, len(jobdump['columns'])):
        if (jobdump['columns'][c]['field'] == key):
            break
    return jobdump['rows'][0][c]

#
# Main body
#
nc_url = os.environ['NC_URL']
scheme = nc_url.split(":")[0]
hostport = nc_url.split("/")[2]
url = "{0}://{1}".format(scheme, hostport)

ss = requests.Session()   # use keep-alive
jwt = getJWT(ss, url, os.environ["USER"], getMyPassword())
for arg in range (1,len(sys.argv)):
    jobid = sys.argv[arg]
    query = url + '/api/v3/jobs/' + jobid
    r = ss.get(query, headers={"Authorization": jwt},
   verify=True)
    infoPrint(r.text)
    print ("")
```

JWT Access Tokens

To authorize REST access via the API, REST requests must pass an access token in the request header. In the `nc_info.py` example, the `jwt` variable holds the access token. Access tokens expire about 4 hours after issue. An application that will run for several hours should adopt some strategy to allow for access token expiry. One strategy would be to re-authenticate, using login name and password, prior to any burst of REST activity that will be known to be complete in a few hours or less. Another strategy is to check the request return status and re-authenticate at that time, using the new access token thereafter.

The recommended way to authenticate user name and password to issue an access token is the VOVRestV3 Python class `authorize()` method function. If you would like direct control over the access token allocation, as in the `nc_info.py` example, see the `getJWT()` function in the `getToken.py` example in the next section.

HTTPS Security Considerations

REST requests and responses are sent over the HTTP communication protocol, and suitable HTTP connections can be configured with three possible security levels, ranging from insecure to very secure:

1. The basic (insecure) way to start the Accelerator queue server and web server is to use the default HTTP connection on the server's webport. The REST API interface always is allowed on HTTP webport connections.
2. An intermediate level of security is possible if the HTTPS protocol is requested by configuring server parameter `ssl.enable` to 1 in `policy.tcl`. In this case, a self-signed SSL certificate will be generated when the Accelerator server starts. If the REST application uses the VOVRestV3 python module method functions, this type of connection will be supported without warnings being issued. If direct access to the REST API is implemented using the Python request method functions `get()` and `post()`, then communication will be allowed on these connections only if the optional `verify=False` keyword argument is passed to those functions.
3. A very secure HTTPS connection will be established if an SSL certificate that was signed by a trusted Certificate Authority (CA) is added to the Accelerator server configuration. This is the recommended way to configure Accelerator products. If the REST application author would like to require level 3 security, then the direct access to REST via request method functions `get()` and `post()` must be used, and the keyword argument `verify=True` must be specified.

Connection Keep-Alive

The use of HTTP keep-alive, or persistent connection, is an important technique that optimizes applications during times of very frequent REST requests. HTTP keep-alive and the resultant reuse of HTTP connections will occur when using the VOVRestV3 `submitRequest()` method function or when using a "session" allocated by the `request.Session()` method function.

The `nc_info.py` Python application illustrates the use of keep-alive by the latter method in its main loop across job IDs. Each call to `ss.get()` will reuse the same HTTP connection. If the `ss.get()` call were to be replaced by `request.get()`, the keep-alive feature would not persiste an HTTP connection between subsequent requests. In that case, each request would build a new HTTP connection before issuing the request.

JWT Token Allocation

Accelerator REST API v3 uses JSON Web Tokens (JWTs) to implement the access tokens used in REST requests. A JWT access token is allocated by an HTTP POST request by the client that passes a matching Linux username and password along with an implementation-defined REST URL. The server responds by verifying the username and password match and returning the JWT access token to the client.

The `authorize()` method function in the VOVRest Python module should be used for most convenient authorization and automatic JWT handling. This method is used in the later examples in this tutorial. The `getToken.py` module that follows shows the low level interface to JWT allocation. This module must be provided with the first few working Python code examples shown in this tutorial.

getToken.py

```
# JWT Token utilities

import os, requests

#
# Function getJWT()
#
# Arguments
#
#       url       - A URL for a VOV project
#       user      - user name for authentication
#       password  - password for authentication
#
def getJWT(url, username, password):
    scheme = url.split(":")[0]
    hostport = url.split("/")[2]
    baseUrl = "{0}://{1}".format(scheme, hostport)
    tokenUrl = baseUrl + "/api/v3/token"
    myauth={ 'username' : username , 'password' : password }
    r = requests.post(tokenUrl, data=myauth)
    if ( r.status_code > 300 ):
        print ("JWT Error code %d" % r.status_code)
        print ("          returned status: ", r.json() )
        print ("          error message : %s" % r.json()['error'])
        exit(1)
    token = r.json()
    jwtToken = token['token_type'] +":" + token['access_token']
    return jwtToken

#
# Function getMyPassword()
#
# This example function simply prompts the user to type the account password.
#
def getMyPassword():
    import getpass
    return getpass.getpass('Password:')
```

The Python code in the **getToken.py** module also contains a placeholder password prompt function **getMyPassword()**. The handling of user passwords and JWT tokens in practice will be up to the REST application developers in accordance with their own best practices for handling and storing security-sensitive information. If the REST application runs for many hours, new

JWT tokens will need to be allocated and authenticated after the previous ones expire. The application needs to provide a way to provide the password each time a JWT token is authenticated. Additional methods for renewing or allocating JWT tokens are being considered for future Accelerator software releases.

Legal Notices

Intellectual Property Rights Notice

Copyrights, trademarks, trade secrets, patents and third party software licenses.

This Intellectual Property Rights Notice is exemplary, and therefore not exhaustive, of intellectual property rights held by Altair Engineering Inc. or its affiliates. Software, other products, and materials of Altair Engineering Inc. or its affiliates are protected under laws of the United States and laws of other jurisdictions. In addition to intellectual property rights indicated herein, such software, other products, and materials of Altair Engineering Inc. or its affiliates may be further protected by patents, additional copyrights, additional trademarks, trade secrets, and additional other intellectual property rights. For avoidance of doubt, copyright notice does not imply publication. Copyrights in the below are held by Altair Engineering Inc. or its affiliates. Additionally, all non-Altair marks are the property of their respective owners.

This Intellectual Property Rights Notice does not give you any right to any product, such as software, or underlying intellectual property rights of Altair Engineering Inc. or its affiliates. Usage, for example, of software of Altair Engineering Inc. or its affiliates is governed by and dependent on a valid license agreement.

Altair Simulation Products

Altair® AcuSolve® ©1997-2023

Altair Activate®©1989-2023

Altair® Battery Designer™ ©2019-2023

Altair Compose®©2007-2023

Altair® ConnectMe™ ©2014-2023

Altair® EDEM™ © 2005-2023

Altair® ElectroFlo™ ©1992-2023

Altair Embed® ©1989-2023

Altair Embed® SE ©1989-2023

Altair Embed®/Digital Power Designer ©2012-2023

Altair Embed® Viewer ©1996-2023

Altair® ESAComp® ©1992-2023

Altair® Feko® ©1999-2023

Altair® Flow Simulator™ ©2016-2023

Altair® Flux® ©1983-2023

Altair® FluxMotor® ©2017-2023

Altair® HyperCrash® ©2001-2023

Altair® HyperGraph® ©1995-2023

Altair® HyperLife® ©1990-2023

Altair® HyperMesh® ©1990-2023

Altair® HyperSpice™ ©2017-2023

Altair® HyperStudy® ©1999-2023

Altair® HyperView® ©1999-2023

Altair® HyperViewPlayer® © 2022-2023

Altair® HyperWorks® ©1990-2023

Altair® HyperXtrude® ©1999-2023

Altair® Inspire™ ©2009-2023

Altair® Inspire™ Cast ©2011-2023

Altair® Inspire™ Extrude Metal ©1996-2023

Altair® Inspire™ Extrude Polymer ©1996-2023

Altair® Inspire™ Form ©1998-2023

Altair® Inspire™ Mold ©2009-2023

Altair® Inspire™ PolyFoam ©2009-2023

Altair® Inspire™ Print3D ©2021-2023

Altair® Inspire™ Render©1993-2023

Altair® Inspire™ Studio ©1993-2023

Altair® Material Data Center™ ©2019-2023

Altair® MotionSolve® ©2002-2023

Altair® MotionView® ©1993-2023

Altair® Multiscale Designer® ©2011-2023

Altair® nanoFluidX® ©2013-2023

Altair® OptiStruct® ©1996-2023

Altair® PollEx™ ©2003-2023

Altair® PSIM™ © 2022-2023

Altair® Pulse™ ©2020-2023

Altair® Radioss® ©1986-2023

Altair® romAI™ © 2022-2023

Altair® S-FRAME® © 1995-2023

Altair® S-STEEL™ © 1995-2023

Altair® S-PAD™ © 1995-2023

Altair® S-CONCRETE™ © 1995-2023

Altair® S-LINE™ © 1995-2023

Altair Packaged Solution Offerings (PSOs)

Altair HPC & Cloud Products

Altair Data Analytics Products

Third Party Software Licenses

AcuConsole contains material licensed from Intelligent Light (www.ilight.com) and used by permission.

For a complete list of Altair Accelerator Third Party Software Licenses, please click herehere.

Technical Support

Altair provides comprehensive software support via web FAQs, tutorials, training classes, telephone and e-mail.

Altair One Customer Portal

Altair One (https://altairone.com/) is Altair's customer portal giving you access to product downloads, Knowledge Base and customer support. We strongly recommend that all users create an Altair One account and use it as their primary means of requesting technical support.

Once your customer portal account is set up, you can directly get to your support page via this link: www.altair.com/customer-support/.

Altair Training Classes

Altair training courses provide a hands-on introduction to our products, focusing on overall functionality. Courses are conducted at our main and regional offices or at your facility. If you are interested in training at your facility, please contact your account manager for more details. If you do not know who your account manager is, e-mail your local support office and your account manager will contact you

Telephone and E-mail

If you are unable to contact Altair support via the customer portal, you may reach out to the technical support desk via phone or e-mail. You can use the following table as a reference to locate the support office for your region.

When contacting Altair support, please specify the product and version number you are using along with a detailed description of the problem. It is beneficial for the support engineer to know what type of workstation, operating system, RAM, and graphics board you have, so please include that in your communication.

Location	Telephone	E-mail
Australia	+61 3 9866 5557 +61 4 1486 0829	anz-pbssupport@altair.com
China	+86 21 6117 1666	pbs@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr

Location	Telephone	E-mail
Malaysia	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0) 46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
South Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
United Kingdom	+44 (0)1926 468 600	pbssupport@europe.altair.com

See www.altair.com for complete information on Altair, our team and our products.

Index

U

V