splunk>

Developing with the Splunk REST API – Lab Exercises

Overview

Welcome to the Splunk Education lab environment, which is a Linux server, located in the cloud. This server will be used for program development, but it also contains a Splunk server. The Splunk server has been installed in **`/opt/splunk`**.

Your instructor will give you the IP address to access the Linux server, as well as the necessary Linux and Splunk user credentials.

In the lab exercises, you will:

- Call the Splunk REST endpoints from the command line, using **`curl`** or **`Postman`**. This helps you to learn about the various REST endpoints, including the output that they produce.
- Embed calls to REST endpoints into existing Python programs that are provided.
- Read through commented Python code to see how to parse and use output from REST endpoints.

You will access the Splunk Web interface via HTTP using a browser on your local computer.

You will complete the lab exercises directly on the Linux server. Log into Linux via SSH to edit and run `curl` commands and Python programs. You can edit the Python programs using either `vi` or `nano`. (If you are not familiar with vi, you will likely find nano to be much easier to use.)

As an alternative to vi and nano on the Linux server, many text editors have SFTP built into them, such as Notepad++ (Windows), BBEdit (Mac) or VSCode (all platforms), they're all free. You can edit the Python programs on your remote computer and use one of these tools to send files to the Linux server via SFTP. (You can use SFTP tools to move the files manually.)

A complete solution set for all Python scripts and `curl` commands can be found in **`/home/student/rest-course/solutions`**. There's an answer guide for all curl commands on the credential tool given to you by your instructor which also supports exporting all commands as a txt file, or if you prefer there's a file with all commands you can get on your assigned class server, in the directory **`/home/student/rest-course/solutions/curl.txt`**.

To receive a certificate of completion for this course, you must complete at least 75% of the classwork (optional parts of the labs not included on the 75%). At the end of the class, you will be able to download all your working files for later reference.

Lab Connection Info

You may want to write lab connection information (provided by instructor) in the spaces below.

Item	My Credentials	Notes
Host IP:		This is the IP of the Linux server you will use. This value is supplied by your instructor.

splunkd Management Port:	8089	The default management port. It is used for REST connections.
Splunk Web URL:	https://Server-EIP or DNS server, provided by your instructor.	*Server IP* is referenced earlier in this table and supplied by your instructor. No port number needed; the default HTTP port is used.
Splunk Web username:	student	Note that this user is assigned the admin role.
Splunk Web password:	restC0der	
Linux username:	student	You will use the same user/password to access Linux as you do to access Splunk.
Linux password:	restC0der	
Linux server login:	ssh student@Server-EIP	*Server IP* is referenced earlier in this table and supplied by your instructor.

Lab Exercise 1 – Introduction to the Splunk REST API

Description

In this lab exercise, you will explore the lab environment. Then, at the Linux server command line, you will use `curl` or `Postman` to call REST endpoints and then observe the Splunk server output.

NOTE: An answer key is provided for the `curl` commands in the credential page provided by your instructor (which supports local text exports as well) and in the solution file located in your lab machine: **`/home/student/rest-course/solutions/curl.txt`**

Steps

Task 1: Set your Splunk server name.

NOTE: Every student has his or her own Splunk server. In order to get credit for the lab exercises, your instructor must be able to track your work. Please set your Splunk server name as described in the following steps.

1. Get the host IP number for your Splunk server from the instructor and record it in the Lab Connection Info table, which precedes this lab exercise. This is the IP address of the Linux server you will use in the lab exercises.

   As noted in the Lab Connection Info table:

   - The Splunk Web URL consists only of the host IP number. In the URL, it is not necessary to specify a port number. The default HTTP port – 80, not the Splunk default of 8000 – is used.
   - The **splunkd** Management Port, used for REST connections, uses the default port (8089), however for this class only, SSL was disabled on port 8089, so you must call it http and not https in your curl calls.
   - To access the Splunk Web, your user name is **`student`**, and the password is **`restC0der`**. This account uses the **admin** role.
   - All your REST API calls will all be performed under the **`restclient`** account, and the password is also **`restC0der`**, this way we keep separate what comes from the interface from what comes from your REST calls using **curl** or **Postman**.
2. Use your browser to log into Splunk Web as the student user. (Splunk Web is accessed with this URL: **`http://`*`Server-EIP`***, where *`Server-EIP`* is the IP address given to you by your instructor on the credentials tool.) The Splunk Home (launcher) app appears.
3. Towards the top of the browser window, from the Splunk bar, click **Settings > Server settings**. The Server Settings page appears.
4. On the Server Settings page, click **General settings**.
5. Set the **Splunk server name** field to your **first name initial** plus your **full last name**. For example, if your name is **Mitch L. Fleischman**, set the Splunk server name to **mfleischman**. Also, at the bottom at the same page under “Default host name” put the same value.

NOTE: Failure to set the Splunk server name appropriately could cause you NOT to get credit for the lab exercises!

6. At the bottom of the page, click **Save**. After successfully changing the Splunk server name, you should see this:

   For your change to become effective, you must restart the Splunk server. In the next steps, you will restart the Splunk server.
7. Towards the top of the browser window, from the Splunk bar, click **Settings > Server controls**. The Server Controls page appears.
8. Click **Restart Splunk**. A dialog appears, asking if you are sure you want to restart Splunk.
9. In the dialog, click **OK**. The Splunk server restarts. When the Splunk server is back up, a Restart successful dialog appears.
10. To dismiss the dialog, click **OK**. The Splunk Web login page appears.

Task 2: Create a Splunk session.

11. Start a SSH terminal session and connect to the Linux server:

    ```
    ssh student@Server-EIP
    ```

12. Using what you learned in the lecture module, write and execute a `curl` command (or build a call by calling the proper parameters on `Postman`) that logs into the local Splunk server using port 8089 as the **restclient** user. (Don't forget to use the `-k` argument in `curl` to disable ssl verification. Your lab environment has only the default certificates, and you need the `-k` argument to successfully use the REST endpoints.)

    For example:

    ```
    curl -k http…
    ```

    If successful, you should see something like what is shown below:

```
<response>
  <sessionKey>ea1TakrfnsDmRvmXd7k80vwpWTGADC6aL_i^p9BClsiMouR96s_smKHcHRNKN6wCajvBuJtkhZT91Xp
OI9a6Rj2HUqiFMloZy5c9kfkDfndE8n8YHbVzWNrulF</sessionKey>
</response>
```

    Note that the `sessionKey` value is returned in XML. That is because you did not specify an output format – and the REST API generally defaults to XML.

Task 3: Delete a Splunk session.

13. Delete the Splunk session that you created in a previous step. (Note: the **restclient** user in Splunk is assigned the admin role.)

    When done, you should see a lot of output – in either XML or JSON, depending on how you set the output mode. (If you did not set the output mode, then the output appears in XML.)

    For XML, you should see output that begins something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This is to override browser formatting; see server.conf[httpServer] to disable. . . . . .
 . . . . . .
. . . . . .
 . . . . . .
. . . . . .
 . . . . . .
. . . . . .
 . . . . . .
. . . . . .
 . . . . . .
. . . . . .
 . .-->
<?xml-stylesheet type="text/xml" href="/static/atom.xsl"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:s="http://dev.splunk.com/ns/rest" xmlns:opens
earch="http://a9.com/-/spec/opensearch/1.1/">
  <title>httpauth-tokens</title>
  <id>https://localhost:8089/services/authentication/httpauth-tokens</id>
  <updated>2018-01-30T23:54:24+00:00</updated>
  <generator build="004410bdc029" version="7.1.0"/>
  <author>
    <name>Splunk</name>
  </author>
  <link href="/services/authentication/httpauth-tokens/_acl" rel="_acl"/>
```

14. Optional: repeat the `curl` command to delete the session. You should receive an error. Here is an example of the XML output you would receive:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <messages>
    <msg type="ERROR">Not a valid session: ea1TakrfnsDmRvmXd7k80vwpWTGADC6aL_i^p9BClsiMouR96s
_smKHcHRNKN6wCajvBuJtkhZT91XpOI9a6Rj2HUqiFMloZy5c9kfkDfndE8n8YHbVzWNrulF</msg>
  </messages>
</response>
```

Task 4: Call a REST endpoint using a cookie.

15. Create a Splunk session, similar to what you did in a previous lab step. However, this time, add an argument to produce a cookie.

NOTE: Cookies are sent within an output header. In the `curl` statement, add the `-i` argument in `curl` so that the output headers – including the cookie – are shown.

Also note that `splunkd_8089` must be part of the cookie value.

If successful, in the output, you should see `Set-Cookie` followed by a cookie name and value. Note that the cookie and its value are preceded by "`Set-Cookie`".

```
HTTP/1.1 200 OK
Date: Fri, 02 Feb 2018 21:21:29 GMT
Expires: Thu, 26 Oct 1978 00:00:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, max-age=0
Content-Type: text/xml; charset=UTF-8
X-Content-Type-Options: nosniff
Content-Length: 159
Connection: Keep-Alive
Set-Cookie: splunkd_8089=GXpehUt403pn8q6yc3XFvn5W0Pt0uTYbwpnAG^HzXuBn2Md9v8iAKCgihBfPlZ
_DInlJUB1eYMZaEBPUCV7HmE^HpP4RCg4RatkYMs9Z_v3y; Path=/; Secure; HttpOnly; Max-Age=3600;
 Expires=Fri, 02 Feb 2018 22:21:29 GMT
X-Frame-Options: SAMEORIGIN
Server: Splunkd

<response>
  <sessionKey>GXpehUt403pn8q6yc3XFvn5W0Pt0uTYbwpnAG^HzXuBn2Md9v8iAKCgihBfPlZ_DInlJUB1eY
MZaEBPUCV7HmE^HpP4RCg4RatkYMs9Z_v3y</sessionKey>
</response>
```

16. Below, you will find an endpoint that lists the indexes on your Splunk instance. Modify this command to use the cookie that you generated in the previous step.

    ```
    http://localhost:8089/services/data/indexes
    ```

    If successful, you should see a very long string of XML output. If unsuccessful, you should receive just a few lines of output, including an error message.

Task 5: Call a REST endpoint without creating a session.

17. Edit the `curl` command that you used to list the indexes on your Splunk instance to remove the cookie authentication; instead, authenticate with Splunk without specifying an existing session.

    The output should look similar to what was generated in the previous task.

splunk>

Lab Exercise 2 – Namespaces and Object Management

Description

In this lab exercise, you will observe how namespaces affect the results returned from REST endpoints. As in the first lab exercise, you will access the REST endpoints using `curl`.

NOTE: Program results screenshots shown here are for illustration only – to give you an approximate idea of what successful program execution looks like. The actual data varies depending on when the program is run.

NOTE: An answer key is provided for the `curl` commands in the credential page provided by your instructor (which supports local text exports as well) and in the solution file located in your lab machine: **`/home/student/rest-course/solutions/curl.txt`**

Steps

Task 1: List all saved searches in all namespaces, except users' private searches.

1. At the command line on the Linux server, execute a single `curl` command (or `Postman` call) that logs into Splunk as the `restclient` user (all passwords in this class: `restC0der`) and lists the names of all saved searches in all namespaces, except users' private searches.

   If successful, you should see a long stream of XML output. If not successful, you should see a short stream of output, containing an error message.

2. Edit the command or call you wrote in the previous step and append this `grep` statement – and then execute the command again.

   ```
   | grep \<title\>
   ```

   If successful, you should see output that looks something like this – multiple lines of `<title>` elements. The first `<title>` element contains the string "savedsearch"; each additional `<title>` element contains the name of a saved search.

```
<title>savedsearch</title>
  <title>Bucket Copy Trigger</title>
  <title>DMC Alert - Abnormal State of Indexer Processor</title>
  <title>DMC Alert - Critical System Physical Memory Usage</title>
  <title>DMC Alert - Expired and Soon To Expire Licenses</title>
  <title>DMC Alert - Missing forwarders</title>
  <title>DMC Alert - Near Critical Disk Usage</title>
  <title>DMC Alert - Saturated Event-Processing Queues</title>
  <title>DMC Alert - Search Peer Not Responding</title>
  <title>DMC Alert - Total License Usage Near Daily Quota</title>
  <title>DMC Asset - Build Full</title>
  <title>DMC Asset - Build Standalone Asset Table</title>
  <title>DMC Asset - Build Standalone Computed Groups Only</title>
  <title>DMC Forwarder - Build Asset Table</title>
  <title>DMC License Usage Data Cube</title>
  <title>Errors in the last 24 hours</title>
  <title>Errors in the last hour</title>
  <title>instrumentation.anonymized.eventsByTime</title>
  <title>instrumentation.anonymous.firstEvent</title>
  <title>instrumentation.deployment.app</title>
  <title>instrumentation.deployment.clustering.indexer</title>
  <title>instrumentation.deployment.forwarders</title>
  <title>instrumentation.deployment.index</title>
  <title>instrumentation.deployment.node</title>
  <title>instrumentation.lastSent</title>
  <title>instrumentation.license.firstEvent</title>
  <title>instrumentation.licenseUsage</title>
  <title>instrumentation.licensing.stack</title>
  <title>instrumentation.performance.indexing</title>
  <title>instrumentation.performance.search</title>
  <title>instrumentation.reporting</title>
```

Task 2: List all saved searches for user terry in the search namespace.

3. Modify the command you executed in the previous task to output the names of all saved searches of user **terry** in the **search** namespace. As before, use the `grep` command to show only lines containing the `<title>` tag. Make sure a list of searches is returned without any errors.

Task 3: List the ACL information for pat-search, in the search app, owned by user pat.

4. At the command line, execute a single `curl` command that logs into Splunk as the **restclient** user and lists the **ACL** information for object pat-search. When finished, you should see a stream of XML output, containing the ACL information.

   Look through the stream of XML output for the `<content>` element; that is where you will see the ACL information, as shown below:

```
    <content type="text/xml">
      <s:dict>
        <s:key name="eai:acl">
          <s:dict>
            <s:key name="app">search</s:key>
            <s:key name="can_change_perms">1</s:key>
            <s:key name="can_list">1</s:key>
            <s:key name="can_share_app">1</s:key>
            <s:key name="can_share_global">1</s:key>
            <s:key name="can_share_user">1</s:key>
            <s:key name="can_write">1</s:key>
            <s:key name="modifiable">1</s:key>
            <s:key name="owner">pat</s:key>
            <s:key name="perms"/>
            <s:key name="removable">1</s:key>
            <s:key name="sharing">user</s:key>
          </s:dict>
        </s:key>
      </s:dict>
    </content>
  </entry>
</feed>
```

Task 4: Change the permissions on pat-search so that anyone with the power user role can edit the search.

5. Edit the command you executed in the previous step so that anyone with power user role can edit the search – that is, so they can change the search, if desired.

NOTE: The power user role is specified in the REST API as `power`.

When done, a stream of XML output is sent containing the ACL information. You should see that the permissions were successfully changed, as shown below.

```
<s:key name="perms">
  <s:dict>
    <s:key name="write">
      <s:list>
        <s:item>power</s:item>
      </s:list>
    </s:key>
  </s:dict>
</s:key>
```

Task 5. Change the sharing on pat-search to global.

6. Edit the command you executed in the previous step to change the sharing on pat-search to global. When done, towards the end of the XML output, you should see that the sharing was successfully changed, as shown below.

```
            <s:key name="removable">1</s:key>
            <s:key name="sharing">global</s:key>
          </s:dict>
        </s:key>
      </s:dict>
    </content>
  </entry>
</feed>
```

Lab Exercise 3 – Parsing Output

Description

In this lab exercise, you will edit a Python program that authenticates with Splunk.
You will also read and execute a Python program that connects to Splunk, extracts a sessionKey from the response, and gets the names and sharing level for saved searches on the Splunk server.

If time permits, you will add information that also prints the permissions assigned to each saved search.

NOTE: The **`auth.py`** Python program that you will be editing in this lab exercise can be found in **`/home/student/rest-course/python`**. (In other exercises, you will edit other Python scripts on the same directory. Do not change other scripts for now.)

NOTE: Completed versions of the programs you will be editing (or using) in the lab exercises can be found in **`/home/student/rest-course/solutions`**.

Steps

Task 1: Authenticate with the Splunk server and print out the sessionKey.

In this task, you will edit `auth.py`. You will execute it at the command line:
`python auth.py`

When you execute the completed program, it connects to Splunk and prints out the value for the `sessionKey` that was returned – for example:

```
[student@ip-10-0-134-156 solutions]$ python auth.py
sessionKey is BgIHAhgmWhvOK_IkPwYQO81e0TDj0gcPFrbdHFwLG1r1IkCH4KueL9iMig^vMY6rtw
9AASNlXeCbRErha_GhNjcr7S2CU6Sa6Jy9F2jMnN
```

1. At the command line on the Linux server, change directory to `python`:
   `cd ~/rest-course/python`
2. Open `auth.py` (e.g., using nano or vi) and look for the comment **Lab3, Task 1, A**.
3. Follow the instructions in the comment to replace ++ with the correct endpoint information.
4. Look for the comment **Lab3, Task 1, B**.
5. Follow the instructions in the comment to replace ++ with the correct argument and value to cause Splunk to send json output.
6. Look for the comment **Lab3, Task 1, C**.
7. Follow the instructions in the comment to replace ++ with code that parses the Splunk output as a python dictionary. (This dictionary is called `parsed_json`.)
8. Look for the comment **Lab3, Task 1, D**.
9. Follow the instructions in the comment to replace ++ with the code needed to extract the sessionKey value from `parsed_json`.
10. Save your work.
11. Execute `auth.py`:
    `python auth.py`
    If it executes correctly it should output something like this:

```
[student@ip-10-0-134-156 solutions]$ python auth.py
sessionKey is BgIHAhgmWhvOK_IkPwYQO81e0TDj0gcPFrbdHFwLG1r1IkCH4KueL9iMig^vMY6rtw
9AASNlXeCbRErha_GhNjcr7S2CU6Sa6Jy9F2jMnN
```

Task 2: Read through and execute a Python program that prints out the name of each saved search and its sharing level.

In this task, you will read the through the Python code (in `~/rest-course/solutions`) in `savedsearch.py`, noting the REST API calls that are made and how the output from REST is parsed and used. You will execute it at the command line, passing in a user and an app:
`python savedsearch.py user app`

When you execute the completed program, it connects to Splunk and prints out all of the saved searches in the namespace specified by the user and app combination.

For example, if you execute:

`python savedsearch.py terry search`

Then this is returned:

```
[student@ip-10-0-105-202 solutions]$ python savedsearch.py terry search
Errors in the last 24 hours,app
Errors in the last hour,app
License Usage Data Cube,app
Orphaned scheduled searches,app
terry-search,user
www-search,user
```

12. Log into Splunk Web as **student** and navigate to **Settings > Searches, Reports, and Alerts**.
13. Set the **Type** to **Reports** and the **App** to the **Search & Reporting (search)**.

Name	Actions	Next Scheduled Time	Display View	Owner	App	Alerts	Sharing	Status
Errors in the last 24 hours	Edit Run	none	none	nobody	search	0	App	Enabled
Errors in the last hour	Edit Run	none	none	nobody	search	0	App	Enabled
License Usage Data Cube	Edit Run	none	none	nobody	search	0	App	Enabled
Messages by minute last 3 hours	Edit Run	none	report_builder_display	nobody	search	0	App	Enabled
Orphaned scheduled searches	Edit Run	none	none	nobody	search	0	App	Enabled
Splunk errors last 24 hours	Edit Run	none	none	nobody	search	0	App	Enabled
pat-search	Edit Run	none	none	pat	search	0	Global	Enabled
terry-search	Edit Run	none	none	terry	search	0	Private	Enabled
www-search	Edit Run	none	none	terry	search	0	Private	Enabled

Notice that:

- There are several reports owned by "nobody" and shared to all users in the **search** app. These are default reports that ship with Splunk.
- There are also 2 private reports: one is owned by user **pat** and 2 are owned by user **terry**.

14. Go back to the SSH session on the Linux server; change directory to `~/rest-course/solutions` by executing this command:

    `cd ~/rest-course/solutions`

15. Open **`savedsearch.py`** in your chosen editor.

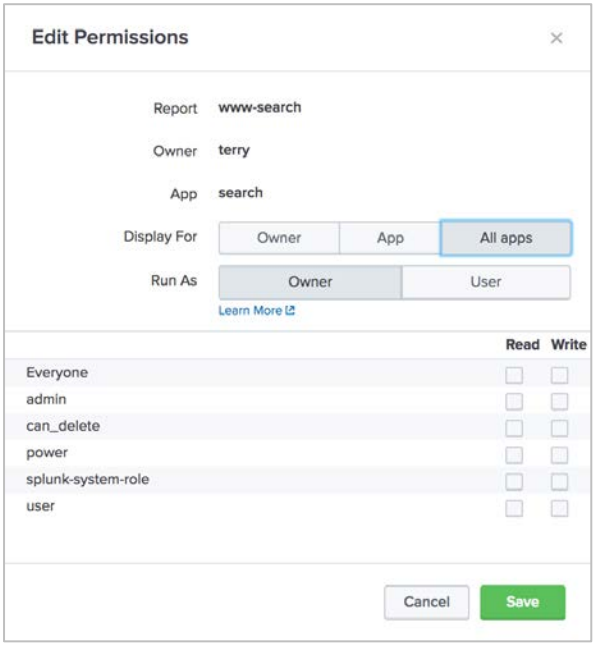
16. As with the previous task, all instructional comments begin with Lab Exercise 3, Task 2. Locate the instructional comments and note how the Python code uses REST API calls to authenticate with the Splunk server and parse and then print out information about saved searches.
17. Execute the program, listing all saved searches owned by any user in the search app. (This should be the same list that you saw earlier in Splunk Web.)

    ```
    python savedsearch.py - search
    ```

    ```
    Errors in the last 24 hours,app
    Errors in the last hour,app
    License Usage Data Cube,app
    Messages by minute last 3 hours,app
    Orphaned scheduled searches,app
    pat-search,user
    Splunk errors last 24 hours,app
    terry-search,user
    www-search,user
    ```

18. Use the `savedsearch.py` program to test each of the following use cases:
    - Owner = - (wildcard) and app = -
    - Owner = pat and app = -
    - Owner = terry and app = -
    - Owner = nobody and app = -
    - Owner = pat and app = search
    - Owner = nobody and app = search

    Next, you will change the permissions on www-search.
19. In Splunk Web, you should still see the Searches, Reports, and Alerts page. If you do not see this page, navigate to **Settings > Searches, Reports, and Alerts** and configure the page to show **Reports** from this App: **Search & Reporting (search)**.
20. In the row for **www-search**, under Actions, click the **Edit > Edit Permissions** link.
21. In the Edit Permissions dialog, set **Display for** to **All Apps** (which is global sharing).



Notice that after you click All apps, the dialog expands to allow you to set other permissions.

22. Next to **Everyone**, click the checkboxes for **Read** and for **Write**, and then click **Save**.
23. Run the `savedsearches.py` program again, for the user case where Owner = **nobody** and app = **search**.

    You should see **www-search** in the list now, since it is shared globally.

NOTE: The **nobody** user is associated with savedsearches whose owner is a particular app. The nobody user is also associated with savedsearches that were created by users who are no longer valid users on the Splunk server.

Task 3 (optional): Print out the permissions data with each saved search.

In this task, you will edit `savedsearchc.py`. You will execute it at the command line, passing in a user and an app:
`python savedsearchc.py <user> <app>`

If you want, instead of editing the Python code directly, look at the solution (in `~/rest-course/solutions`) and read the instructional comments – as you did in the previous task.

When you execute the completed program, it connects to Splunk and prints out all of the saved searches in the namespace specified by the user and app combination.

For example, if you execute:

```
python savedsearchc.py pat search
```

Then this is returned:

```
Errors in the last 24 hours,app
   Read: ["*"]
   Write: ["admin"]
Errors in the last hour,app
   Read: ["*"]
   Write: ["admin"]
License Usage Data Cube,app
   Read: ["*"]
   Write: ["admin", "power"]
Orphaned scheduled searches,app
   Read: ["*"]
   Write: ["admin", "power"]
pat-search,global
   Read:  Owner only
   Write: ["power"]
www-search,global
   Read: ["*"]
   Write: ["*"]
```

NOTE: The code must consider the structure of the permissions data – and the fact that with private objects, some permissions are *not* explicitly set. (See below.)

Permissions data can look like this:

```
   "perms": {
     "read": [
        "*"
     ],
     "write": [
       "admin",
       "power"
```

```
        ]
    },
```

However, sometimes, perms may be null – meaning that the object is private and only the owner has access to it. Other times, perms may have a list of read permissions defined but no write permissions defined – meaning that only the owner can write to the object.
`savedsearchc.py` must define what to do when perms or its contents is null. If not, python generates an error when it tries to evaluate the null input.

24. Return to the python directory by executing the command:

    `cd ~/rest-course/python`

25. then check with the `pwd` command:

    ```
    [student@ip-10-0-0-201 python]$ pwd
    /home/student/rest-course/python
    ```

26. Open `savedsearchc.py` in your chosen editor. (ex: `nano savedsearchc.py`)
27. All instructional comments begin with Lab Exercise 3, Task 3. Locate the instructional comments and note how the Python code uses REST API calls to authenticate with the Splunk server and parse and then print out permissions information about saved searches.
28. Execute the program so that it outputs a list of searches:

    `python savedsearchc.py`

29. Use the `savedsearchc.py` program to test each of the following use cases:
    - Owner = - (wildcard) and app = -
    - Owner = pat and app = -
    - Owner = terry and app = -
    - Owner = nobody and app = -
    - Owner = pat and app = search
    - Owner = nobody and app = search

Lab Exercise 4 – Oneshot Searching

Description

In this lab exercise, you will:

- Construct `curl` commands (or `Postman` calls) that use REST endpoints to execute oneshot searches.
- OPTIONAL: Read and execute a Python program that executes a oneshot search and parses the results.

Steps

Task 1: Use `curl` to execute a oneshot search using the default earliest and latest time and return results in json format.

1. On the Linux server, at the command line, construct a single `curl` command that logs into Splunk as the **restclient** user, runs the search shown below as a oneshot search, and prints the output to the command line in json format:

   ```
   index=* sourcetype=access_combined price=* product_name=* | head 2
   ```

2. Execute the `curl` command. If successful, you should see output that looks similar to this:


```
{"preview":false,"init_offset":0,"messages":[],"fields":[{"name":"_bkt"},{"name":"_cd"}
,{"name":"_indextime"},{"name":"_kv"},{"name":"_raw"},{"name":"_serial"},{"name":"_si"}
,{"name":"_sourcetype"},{"name":"_time"},{"name":"host"},{"name":"index"},{"name":"line
count"},{"name":"price"},{"name":"productId"},{"name":"product_name"},{"name":"sale_pri
ce"},{"name":"source"},{"name":"sourcetype"},{"name":"splunk_server"}],"results":[{"_bk
t":"web~27~A512281F-8AE8-4DB2-8247-
5CB0805E3B51","_cd":"27:91814","_indextime":"1519419752","_kv":"1","_raw":"64.120.15.15
6 - - [23/Feb/2020:21:02:32] \"GET /oldlink?itemId=EST-
14&JSESSIONID=SD6SL10FF10ADFF4959 HTTP 1.1\" 200 925
\"http://www.buttercupgames.com/cart.do?action=view&itemId=EST-14&productId=SC-MG-G10\"
\"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4) AppleWebKit/536.5 (KHTML, like Gecko)
Chrome/19.0.1084.46 Safari/536.5\" 164","_serial":"0","_si":["ip-10-0-81-
92","web"],"_sourcetype":"access_combined","_time":"2020-02-
23T21:02:32.000+00:00","host":"www2","index":"web","linecount":"1","price":"19.99","pro
ductId":"SC-MG-G10","product_name":"SIM
Cubicle","sale_price":"16.99","source":"/opt/log/www2/access.log","sourcetype":"access_
combined","splunk_server":"ip-10-0-81-92"},{"_bkt":"web~27~A512281F-8AE8-4DB2-8247-
5CB0805E3B51","_cd":"27:91765","_indextime":"1519419719","_kv":"1","_raw":"64.120.15.15
6 - - [23/Feb/2020:21:01:59] \"POST /cart.do?action=purchase&itemId=EST-
7&JSESSIONID=SD6SL10FF10ADFF4959 HTTP 1.1\" 200 911
\"http://www.buttercupgames.com/cart.do?action=addtocart&itemId=EST-
7&categoryId=SIMULATION&productId=SC-MG-G10\" \"Mozilla/5.0 (Macintosh; Intel Mac OS X
10_7_4) AppleWebKit/536.5 (KHTML, like Gecko) Chrome/19.0.1084.46 Safari/536.5\"
587","_serial":"1","_si":["ip-10-0-81-
92","web"],"_sourcetype":"access_combined","_time":"2020-02-
23T21:01:59.000+00:00","host":"www2","index":"web","linecount":"1","price":"19.99","pro
ductId":"SC-MG-G10","product_name":"SIM
Cubicle","sale_price":"16.99","source":"/opt/log/www2/access.log","sourcetype":"access_
combined","splunk_server":"ip-10-0-81-92"}], "highlighted":{}}
```

Task 2: Use `curl` commands or `Postman` calls to search and produce results in csv and in raw format. Use a request parameter so that the search returns only results from yesterday.

The previous search produced the results relative to the present time. In other words, you used the default for the latest time – which is now.
In this search, you will use a request parameter to ensure that only results from yesterday are returned.

3. Edit the previous `command or call`, adding request parameters to cause:
    - Results from yesterday to be sent
    - Limit the search results to show only the fields `_time`, `price`, and `product_name`
    - The results format to be **csv**
4. Execute the `curl` statement. If successful, you should see a block of comma-delimited output, similar to this:

```
"_time",price,"product_name","_bkt","_cd","_indextime","_kv","_raw","_serial","_si","_s
ourcetype"
"2020-02-22T23:58:20.000+00:00","39.99","Dream Crusher","web~27~A512281F-8AE8-4DB2-
8247-5CB0805E3B51","27:55287",1519343900,1,"107.3.146.207 - - [22/Feb/2020:23:58:20]
""POST /product.screen?productId=DC-SG-G02&JSESSIONID=SD7SL2FF2ADFF4950 HTTP 1.1"" 200
2027 ""http://www.buttercupgames.com/cart.do?action=view&itemId=EST-17&productId=DC-SG-
G02"" ""Mozilla/5.0 (iPad; CPU OS 5_1_1 like Mac OS X) AppleWebKit/534.46 (KHTML, like
Gecko) Version/5.1 Mobile/9B206 Safari/7534.48.3"" 912",0,"ip-10-0-81-92
web","access_combined"
"2020-02-22T23:58:14.000+00:00","24.99","Mediocre Kingdoms","web~27~A512281F-8AE8-4DB2-
8247-5CB0805E3B51","27:55274",1519343894,1,"107.3.146.207 - - [22/Feb/2020:23:58:14]
""GET /cart.do?action=changequantity&itemId=EST-18&productId=DB-SG-
G01&JSESSIONID=SD7SL2FF2ADFF4950 HTTP 1.1"" 200 2442
""http://www.buttercupgames.com/category.screen?categoryId=STRATEGY"" ""Mozilla/5.0
(iPad; CPU OS 5_1_1 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko) Version/5.1
Mobile/9B206 Safari/7534.48.3"" 603",1,"ip-10-0-81-92
web","access_combined"
```

5. Edit the previous `curl` command or `Postman` call to return the results in raw format, remove the `fields` command and then execute it.
6. If successful, you should see output that looks similar to this:

```
107.3.146.207 - - [22/Feb/2020:23:58:20] "POST /product.screen?productId=DC-SG-
G02&JSESSIONID=SD7SL2FF2ADFF4950 HTTP 1.1" 200 2027
"http://www.buttercupgames.com/cart.do?action=view&itemId=EST-17&productId=DC-SG-G02"
"Mozilla/5.0 (iPad; CPU OS 5_1_1 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko)
Version/5.1 Mobile/9B206 Safari/7534.48.3" 912
107.3.146.207 - - [22/Feb/2020:23:58:14] "GET
/cart.do?action=changequantity&itemId=EST-18&productId=DB-SG-
G01&JSESSIONID=SD7SL2FF2ADFF4950 HTTP 1.1" 200 2442
"http://www.buttercupgames.com/category.screen?categoryId=STRATEGY" "Mozilla/5.0 (iPad;
CPU OS 5_1_1 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko) Version/5.1
Mobile/9B206 Safari/7534.48.3" 603
```

Task 3 - OPTIONAL: Read through and execute a Python program that executes a oneshot search, returns the results in json format, and for each search result, prints the *field : value* returned.

7. Change to the **solutions** directory by executing this command:
    `cd ~/rest-course/solutions`
8. Open **`oneshot.py`** in your chosen editor.
9. All instructional comments begin with Lab Exercise 4, Task 3. Locate the instructional comments and note how the Python code uses REST API calls to execute a oneshot search, get the results, and

parse through the results so that they can be printed at the command line. There's not to replace here, just understand how the code works.

10. Test the program by executing at the command line:

    ```
    python oneshot.py
    ```

    If your program works, you should see output that looks something like this:

    ```
    Result:
          price :  3.99
          product_name :  Fire Resistance Suit of Provolone
    Result:
          price :  5.99
          product_name :  Holy Blade of Gouda
    ```


Lab Exercise 5 – Normal and Export Searching

Description

In this lab exercise, you will:

- Construct `curl` commands or `Postman` calls that use REST endpoints to execute normal search, get its status, and get the search results.
- Construct `curl` commands or `Postman` calls that use REST endpoints to execute an export search and get the search results.
- OPTIONAL: Edit a python program that executes a normal search, get its status, and get the search results.
- OPTIONAL: Read and execute a python program that executes an export search and gets the search results.

Steps

Task 1: Use `curl` commands or `Postman` calls to create a normal search, get its status, and get its results, in json format.

1. On the Linux server, at the command line, construct a single `curl` command or `Postman` call that logs into Splunk as the **restclient** user, creates a new search job, executes the search shown below as a normal search, and outputs the sid of the search job. For the `output_mode`, use `json`.

```
search index=web sourcetype=access_combined price=* product_name=* latest=@d | fields - _* | fields product_name, price | head 1 | stats sum(price)
```

2. Execute the `curl` command. If successful, the sid of the search job is output:

```
{"sid": returned_sid_value_here }
```

   For example:

```
{"sid":" "1520006220.338"}
```

3. Construct a `curl` command or `Postman` call that gets status information for the search job. For the `output_mode`, use `json`. Don't forget to use the `sid` from the previous step in your search. If the `sid` expires, re-execute the command from step 1 and get the new `sid`.
4. Edit the `curl` command you constructed, to append the following to pipe the output to a file, as directed below (not necessary for `Postman` since the output is pre-formatted on screen)

NOTE: `jq` is a utility that "pretty prints" JSON. This makes the JSON sent by Splunk easy to read in a text editor. (Note that some browsers can "pretty print" JSON automatically.)

```
| jq . > lab5-task1-status.json
```

5. Execute the `curl` command or `Postman` call.
6. Optional: Open `lab5-task1-status.json` in the text editor of your choice – such as nano or vi. Use the editor's Find feature to find the `isDone` property. (For nano, enter ctrl-W and then `isDone`. For vi, type `/isDone`.) As your search was very fast, `isDone` should be set to `true`.
7. Construct a `curl` command that obtains the results for the search you executed. For the output_mode, use json.
8. Execute the `curl` command or `Postman` call. The output should look something like this:

```
{"preview":false,"init_offset":0,"messages":[{"type":"INFO","text":"Your timerange was substituted based on your search string"}],"fields":[{"name":"sum(price)"}],"results":[{"sum(price)":"39.99"}],"highlighted":{}}
```

Task 2: Use curl to create an export search and get its results in json_rows format.

9. At the command line, construct a single `curl` command that logs into Splunk as the `restclient` user, creates a new export search and outputs the results.
   For the `output_mode`, use `json_rows`.
   ```
   search index=web sourcetype=access_combined price=* product_name=* latest=@d| fields _time, price, product_name | head 2 | table product_name
   ```
10. Execute the `curl` command. The output should look something like this:

```
{"preview":false,"init_offset":0,"messages":[{"type":"DEBUG","text":"Configura
tion initialization for /opt/splunk/etc took 11ms when dispatching a search
(search ID: 1520033453.401)"},{"type":"DEBUG","text":"base lispy: [ AND
index::web sourcetype::access_combined ]"},{"type":"DEBUG","text":"search
context: user=\"restclient\", app=\"search\", bs-
pathname=\"/opt/splunk/etc\""},{"type":"INFO","text":"Your timerange was
substituted based on your search
string"}],"fields":["product_name"],"rows":[["Dream Crusher"],["Puppies vs.
Zombies"]]}
```

Task 3: OPTIONAL - Edit a python program to execute a normal search and print its results.

11. In the SSH session on the Linux server, open **`search.py`** in your chosen editor.
    (If the file does not open, make sure the command line points to the `python` directory by executing the command below.)
    ```
    cd ~/rest-course/python
    ```
12. Look for the instructional comments beginning with Lab Exercise 5, Task 3. Use these comments as a guide to complete the missing sections of code. (As before, all missing sections of code are represented with ++.)
13. Test the program by executing at the command line:
    ```
    python search.py
    ```
    If your program works, you should see output that looks something like this:

```
Result
     Time:         2020-03-05T13:52:18.000+00:00
     Category:     STRATEGY
     Product Name: Final Sequel
Result
     Time:         2020-03-05T13:47:01.000+00:00
     Category:     ARCADE
     Product Name: Manganiello Bros.
Result
     Time:         2020-03-05T13:43:00.000+00:00
     Category:     SIMULATION
     Product Name: SIM Cubicle
Result
     Time:         2020-03-05T13:37:57.000+00:00
     Category:     SHOOTER
     Product Name: World of Cheese
Result
     Time:         2020-03-05T13:31:09.000+00:00
     Category:     ACCESSORIES
     Product Name: Holy Blade of Gouda
```


Task 4 - OPTIONAL: Read through and then execute a python program that executes an export search and prints its results.

14. Change directory to `~/rest-course/solutions`:
    ```
    cd ~/rest-course/solutions
    ```

15. In the SSH session on the Linux server; open **`export.py`** in your chosen editor.
16. Look for the instructional comments that begin with Lab Exercise 5, Task 4. Note how the Python code uses REST API calls to authenticate with the Splunk server, execute an export search, get the search results, and parse through the results so that they can be printed at the command line.
17. Test the program by executing at the command line:
    ```
    python export.py
    ```

    The results should look something like this:

    ```
    Result
        Product Name: Dream Crusher
        Category:     STRATEGY
    Result
        Product Name: Orvil the Wolverine
        Category:     ARCADE
    Result
        Product Name: SIM Cubicle
        Category:     SIMULATION
    Result
        Product Name: World of Cheese Tee
        Category:     TEE
    Result
        Product Name: Fire Resistance Suit of Provolone
        Category:     ACCESSORIES
    ```


splunk>

Lab Exercise 6 – Advanced Searching

Description

In this lab exercise, you will use `curl` or `Postman` to practice using the Splunk search endpoints to

- Create and use real-time searches
- Manage search jobs
- Obtain a subset of search results (instead of the whole results set)
- OPTIONAL: You will also read through and then execute a Python program that executes a search and pages through the results.

Steps

Task 1: Use `curl` commands or `Postman` calls to create a real-time search over a sliding 8 hour window, manage the search job, and retrieve results in json format.

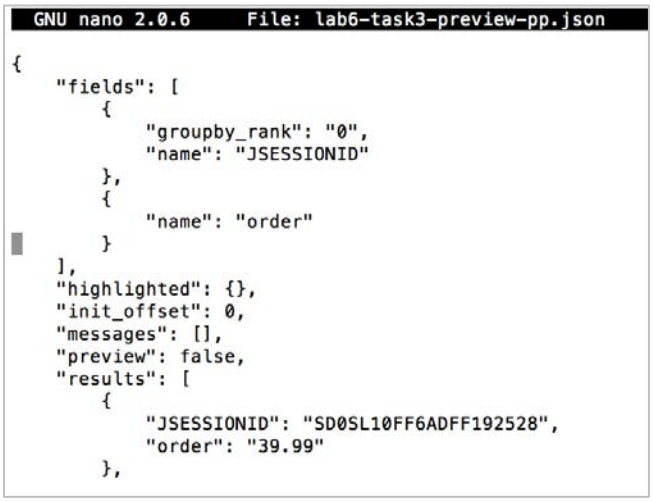
The next two tasks test your ability to use the REST search endpoints to execute, manage, and get results for a real-time search. This search finds the total price for orders from the Buttercup Games web site. (Orders are defined as successful purchases that are associated by the same JSESSIONID.)

1. Construct a `curl` command that creates a real-time search that finds the total price for orders from the Buttercup Games website that begins 8 hours ago and continues indefinitely. (Note that orders are defined as successful purchases that are associated by the same JSESSIONID.)
   ```
   search index=web action=purchase status=200 | fields - _si, _raw, _cd, _bkt, _sourcetype, _indextime, _kv, _serial | fields price, JSESSIONID | stats sum(price) as order by JSESSIONID
   ```
2. Execute the command. If successful, a `sid` value is returned.
3. Store the `sid` value that is returned in a text file (`lab6-realtime-sid.txt`). You will need this later in the lab exercise when you retrieve results for this search.
4. Construct a `curl` command that extends the ttl (time to live) for the job to one hour (hint: 1 hour is 3600 seconds).
5. Execute the `curl` command. If it was successful, this message is returned:
   ```
   {"messages":[{"type":"INFO","text":"Search job's ttl was changed to 3600"}]}
   ```

Task 2: Use `curl` commands or `Postman` calls to get the search status and then get the preview results for the real-time search that you executed earlier in json format.

6. Construct a `curl` command that gets the status of the real-time search that you created in Task 1. (For the `sid`, use the value you stored earlier in `lab6-realtime-sid.txt`. If that search was already finalized, you can execute the command again and get a new `sid`.)
7. Append the following to the command that you constructed in the previous step:
   ```
   | jq . > lab6-realtime-status.json
   ```
8. Execute the command.
9. Open `lab6-realtime-status.json` in the text editor of your choice.
10. Use the editor's Find feature to find the property that contains the number of preview results. This number should be more than zero. Assuming the search has not FAILED (or did not get created properly), then this value should be something greater than zero. Exit the editor.
11. Construct a `curl` statement that gets the preview results for the search you executed in the previous steps.
12. Append the following to the command you constructed in the previous step:
    ```
    | jq . > lab6-task2-preview.json
    ```

13. Execute the command.
14. Open `lab6-task2-preview.json` in the text editor of your choice.
15. Use the editor's Find feature to search for the results that have been found at this time. Verify that there is at least one result. (Hint: look for the `results` array.) Exit the editor.



Task 3: Use `curl` commands or `Postman` calls to pause and then finalize the search that you created in a previous task.

This task is purely a training exercise, to give you an additional opportunity to interact with a search job using the REST API. You would not generally pause and then immediately finalize a search.

16. Construct a `curl` command that pauses the search from the previous task.
17. Execute the command. If it was successful, this message is returned:
    `{"messages":[{"type":"INFO","text":"Search job paused."}]}`
18. Construct a `curl` command that finalizes the search from the previous task.
19. Execute the command. If successful, this message is returned:
    `{"messages":[{"type":"INFO","text":"Search job finalized."}]}`
20. Optional: use the `curl` command you constructed earlier to obtain the latest search status; verify that the `dispatchState` is `DONE`.

Task 4: Use `curl` commands or `Postman` calls to get the results for the finalized search – and then cancel the search.

21. Construct a `curl` command that gets the results for the search you executed in the previous step.
22. Append the following to the command that you constructed in the previous step:
    `| jq . > lab6-task4-results.json`
23. Execute the command.
24. Open `lab6-task4-results.json` in the text editor of your choice.
25. Use the editor's Find feature to find the search results. Exit the editor.
26. Construct a `curl` command that deletes the search from the previous task.
27. Execute the command. If successful, this message is returned:
    `{"messages":[{"type":"INFO","text":"Search job cancelled."}]}`

Task 5 - Use `curl` commands or `Postman` calls to create and execute a saved search.

28. Construct a `curl` command that creates a saved search in the search app. The search, provided below, should execute over the last 24 hours.

```
index=web action=purchase status=200 product_name=* clientip=*
earliest_time=-24h@h
| fields - _si, _raw, _cd, _bkt, _sourcetype, _indextime, _kv, _serial, _time
| fields product_name, clientip
| table product_name, clientip
```

29. Append the following to the command that you constructed in the previous step:

```
| jq . > lab6-task5-create.json
```

30. Execute the command. Information about the saved search is placed in `lab6-task5-create.json`.
31. Open `lab6-task5-create.json` in the text editor of your choice.
32. Verify that the search was created without error. (If you see a lot of JSON output, then the saved search was created successfully; you will see all the properties for the new saved search.)

Task 6 - OPTIONAL: Read through and then execute a python program that executes a search and pages through the results.

In this task, you will read through `offsetsearch.py` and then execute it at the command line. The program executes a search and writes the results in groups (i.e., "pages") of five.

33. In the SSH session on the Linux server, open **`offsetsearch.py`** in your chosen editor. (If the file does not open, make sure the command line points to the `python` directory by executing the command below.)

```
cd ~/rest-course/solutions
```

34. All instructional comments begin with Lab Exercise 6, Task 6. Locate the instructional comments and note how the Python code uses REST API calls to authenticate with the Splunk server, execute a search, check the search status, and then page through the results.
35. Test the program by executing each command at the command line.

```
python offsetsearch.py
```

The program writes out how many search results were found and then writes out the first "page" of search results:

```
The search returned 33 results
Result
     Time:         2020-04-25T16:30:12.000+00:00
     Category:     SHOOTER
     Product Name: World of Cheese
Result
     Time:         2020-04-25T16:29:35.000+00:00
     Category:     ARCADE
     Product Name: Orvil the Wolverine
Result
     Time:         2020-04-25T16:28:26.000+00:00
     Category:     ARCADE
     Product Name: Benign Space Debris
Result
     Time:         2020-04-25T16:24:40.000+00:00
     Category:     TEE
     Product Name: Manganiello Bros. Tee
Result
     Time:         2020-04-25T16:13:18.000+00:00
     Category:     STRATEGY
     Product Name: Dream Crusher
Press Enter to continue..
```

Lab Exercise 7 – Working with KV Stores

Description

In this lab exercise, you will:

- Create a new collection in the `search` app, called `employees`
- Populate the `employees` collection, using the contents of `employees.json`
- Produce `.json` output of all `employees` records, sorted by `last_name`
- Update the value of the `rating` field of a record in the `employees` collection
- Query the `employees` collection data for the record with `last_name` of `Geerhart` and `city` is `Berlin`
- (If time permits) Apply a constraint to the `employees` collection so that the `rating` field must be numeric, and test it by trying to add a non-numeric value and getting an error.

Steps

Task 1: Create a new collection in the `search` app, called `employees`.

1. To create the `employees` collection in the `search` app, what namespace must you use? _____
2. Using the information that you noted in the previous step, construct a `curl` command that creates the `employees` collection in the `search` app.
3. Append the following to the command that you constructed in the previous step:
   `| jq . > lab7-task1-collections.json`
4. Execute the command.
5. Open `lab7-task1-collections.json` in your chosen editor.
6. Use the editor's Find feature to find the word employees. You should see something like what is shown below – providing confirmation that the `employees` collection was created.


```
{
  "links": {
    "create": "/servicesNS/nobody/search/storage/collections/config/_new",
    "_reload": "/servicesNS/nobody/search/storage/collections/config/_reload",
    "_acl": "/servicesNS/nobody/search/storage/collections/config/_acl"
  },
  "origin": "http://34.212.92.149:8089/servicesNS/nobody/search/storage/collections/config",
  "updated": "2019-01-08T23:46:18+00:00",
  "generator": {
    "build": "be11b2c46e23",
    "version": "7.2.1"
  },
  "entry": [
    {
      "name": "employees",
```


7. Exit the editor.

Task 2: Populate the `employees` collection using the contents of `employees.json`.

`employees.json` is located in the `/home/student` directory on your Splunk server.

8. To construct a request to an endpoint that adds records to a collection, what must be specified in the `Content-Header`? _____
9. Use the information that you noted in the previous step to construct a `curl` command that uses `/home/student/rest-course/employees.json` to create records in the `employees` collection.

NOTE: Due to how the `Content-Header` is set, request parameters are *not* passed as in previous exercises. There can only be one -d and it must be used to pass the record data for the collection.

- In the URL, set the `output_mode` to `json`.
- Pass the filename using the `-d @/home/student/rest-course/employees.json`

For details, please refer to the lecture module – or to the solutions in `curl.txt`.

10. Execute the command. The output should look like this:
    ```
    [ "abertorelli", "eartois", "hgeerhart", "jchan", "lshepherd", "mlabonq",
    "kvonstrohm", "mkrebs" ]
    ```
11. If the output does not look like what is shown above, review the command you constructed to find the problem. Fix the problem and try executing it again.

Task 3: Produce .json output of all employees records sorted by last_name.

12. Which request parameter is used to specify sorted output from a collection? _____

NOTE: Sorting works only if all request parameters are passed within the URL. For details, please refer to the lecture module – or to the solutions in `curl.txt`.

13. Using the information that you noted in the previous step, construct a `curl` command that outputs the `employees` records sorted by `last_name`.
14. Append the following to the command that you constructed in the previous step:
    ```
    | jq . > lab7-task3-sorted.json
    ```
15. Execute the command.
16. Open `lab7-task3-sorted.json` in your chosen editor.
17. Scroll through the file content. You should see that the records are shown alphabetically, in ascending order, by `last_name`. If the records do not appear in this order, check the request that you executed. The request parameters must be passed within the URL. If needed, correct the request and execute it again.

Task 4: Update the value of the `rating` field of a record in the `employees` collection.

In this task, for the record whose _key is `hgeerhart`, update the `rating` field to `4`.

18. At a minimum, when constructing a request to update a field in a record, what other fields also must be updated? _____
19. Construct a `curl` command that updates the record with _key of `hgeerhart` to set the `rating` field to `4`.
20. Execute the command. The output should look like this:
    ```
    {"_key": "hgeerhart"}
    ```
    If instead an error occurred, fix the error and execute the command again.

Task 5: Query the `employees` collection data for the record where `last_name` is `Geerhart` and city is Berlin.

21. Construct a `curl` command that queries the `employees` collection for the record where `last_name` is `Geerhart` and `city` is `Berlin`.

    Hint: you need to pass the search JSON properly URL-encoded. There are many URL encode/decode tools available over the internet to make your job easier here. You can google for one

of these tools or you can try this one: https://meyerweb.com/eric/tools/dencoder/

22. Execute the command. You should see this result – the data for Helga Geerhart:

```
[ { "first_name" : "Helga", "last_name" : "Geerhart", "contact" : "+49 089
12345678", "city" : "Berlin", "office" : "cube 47", "rating" : 4, "notes" : [
"1st Note", "2nd note" ], "_user" : "nobody", "_key" : "hgeerhart" } ]
```

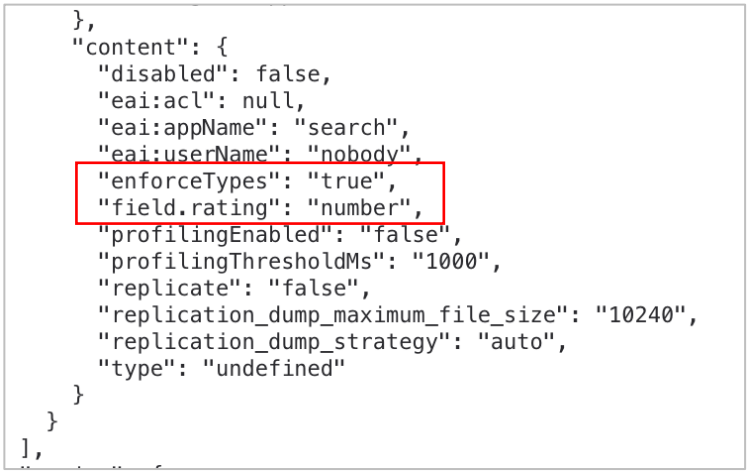
If you do not see this data, correct the request and re-execute it.

Task 6: (optional – if you have time) For new records in the `employees` collection, require that the `rating` field must be numeric.

23. Construct a `curl` command that requires that the `rating` field be a number.
24. Append the following to the command that you constructed in the previous step:

```
| jq . > lab7-task6-constraint.json
```

25. Execute the command.
26. Open `lab7-task6-constraint.json` in your chosen editor. Scroll down to the `content` object (or use your editor's Find feature) to verify that `enforce_types` is set to `true` and that `field.rating` is set to `number`.



If this is not what you see, fix and then re-execute the request.

27. Test the constraint that you created by executing this `curl` command. It attempts to change the `rating` field to `awesome`. Since `awesome` is not a numeric field, an error occurs. The command is found below. If you want to copy and paste the command, you can also find it in `curl.txt`.

```
curl -X POST -k -s -u restclient:restC0der
http://localhost:8089/servicesNS/nobody/search/storage/collections/data/employ
ees?output_mode=json -H 'Content-Type: application/json' -d '{ "first_name" :
"Rene", "last_name" : "Artois", "contact" : "+33 1 44 55 68", "city" :
"Nouvions", "office" : "cube 45", "rating" : "awesome", "notes" : [ ], "_user"
: "nobody", "_key" : "rartois" }'
```

28. Execute the request. You should see this error:

```
{"messages":[{"type":"ERROR","text":"JSON in the request is invalid. (Failed
to convert key='rating' with value='awesome' to type '1')"}]}
```

Lab Exercise 8 – Using the HTTP Event Collector (HEC)

Description

In this lab exercise, you will:

- Construct `curl` commands that use REST endpoints to create an HEC token and ingest data.
- Read and execute a Python program that enables HEC ingestion at the command line.

Steps

Task 1: Use a `curl` command to create an HEC token.

1. In the ssh session on the Linux server, change directory to the rest-course – i.e., at the command line, execute:
   ```
   cd ~/rest-course/python
   ```
   In the `/rest-course/python` directory, there are some data files that you will ingest using HEC by executing `curl` commands.
2. Construct a `curl` command to create an HEC token:
   - Name: lab8
   - Sourcetype: _json
   - Index: games
3. Append the following to the command that you constructed in the previous step:
   ```
   | jq . > lab8-task1.json
   ```
4. Execute the command.
5. Open `lab8-task1.json` in the text editor of your choice. Scroll down to view the contents of the content object. You should see keys (i.e., properties) that were set for the token that you created – including the `name`, the (default) `index`, and the (default) `sourcetype`, and `token`.
   Note that the `indexes` key is set to `[]`. That means that any index can be used. It is the same as using Splunk Web to create a token and leaving **Select allowed indexes** empty.
6. Copy the value of the `token` key on to the computer's clipboard. You need this value for the next task, to paste when creating a `curl` command to send event data.
7. Exit the editor.

Task 2: Use curl commands to use HEC tokens to ingest JSON and RAW event data into the web index.

8. Change directory to `~/rest-course/solutions` using this command:
   ```
   cd ~rest-course/solutions
   ```
9. Open `lab8-task2a-data.json` in the text editor of your choice. You will be ingesting this data using the new token you created. After viewing the file contents, close the editor.
10. Construct a `curl` command to ingest `lab8-task2a-data.json` to the `web` index.

> **NOTE:** To send a file with a `POST` request using `curl`, you preface the filename with `@`. Therefore, the curl command for ingesting `lab8-task2a.json` should include what is shown below. (There is no request parameter name – just the filename.)
>
> ```
> -d @lab8-task2a-data.json
> ```

11. Execute the command. This message should be returned:
    ```
    {"text":"Success","code":0}
    ```
12. Edit the command to ingest `lab8-task2b-data.json`.

13. Execute the edited command. This message should be returned:

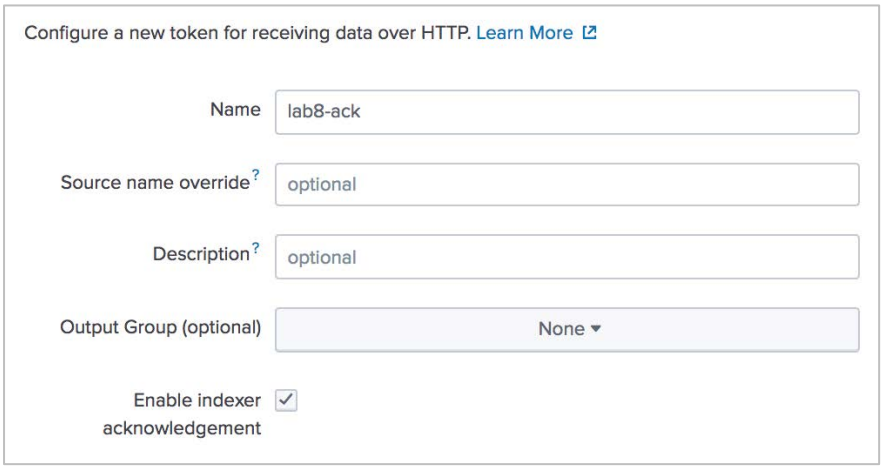
```
{"text":"No data","code":5}
```

14. Open `lab8-task2b-data.json` in the text editor of your choice. View the data. Why didn't this data get ingested? (For the answer, refer to the Solutions on the last page of the lab document.)
15. Re-write the previous `curl` command so that `lab8-task2b-data.json` is ingested as RAW.
16. Execute the command. This message should be returned:

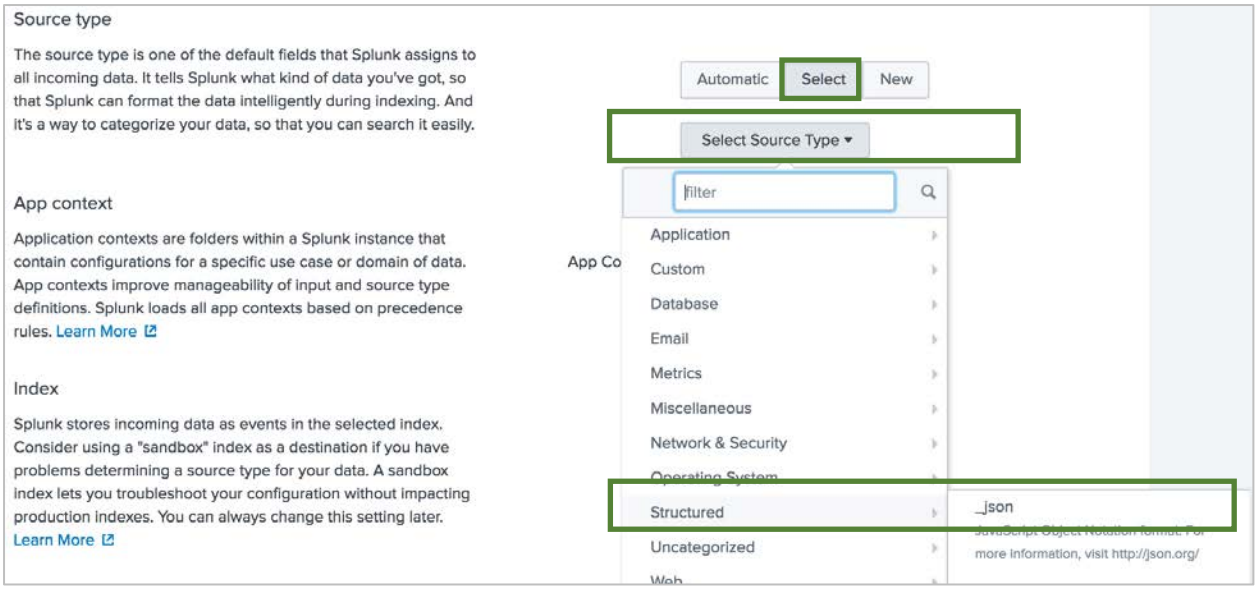
```
{"text":"Success","code":0}
```

Task 3: Use Splunk Web to create an HEC token (lab8-ack) that uses indexer acknowledgement.

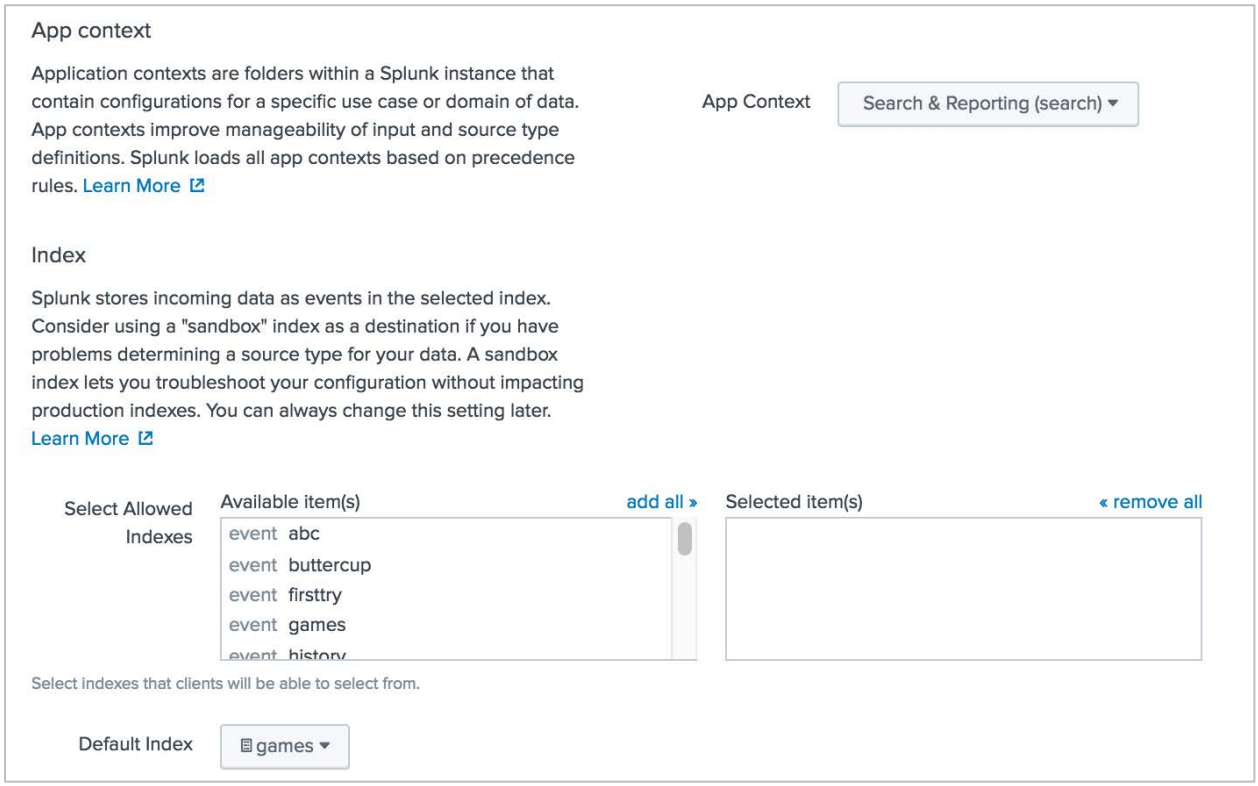
17. Open a browser window and navigate to Splunk Web, using the URL provided by your instructor.
18. Use your browser to log into Splunk Web using the URL **`http://`*`Server-EIP`***, as `student`/`restC0der`.
19. After you are logged in, towards the upper right of the Splunk Web main page, click **Settings > Data inputs**. The Data inputs page appears.
20. On the Data inputs page, to the right of **HTTP Event Collector**, click **+ Add New**. The Add Data page appears on the Select Source step.
21. For the **Name**, enter **lab8-ack**.
22. Click the **Enable indexer acknowledgment** checkbox.



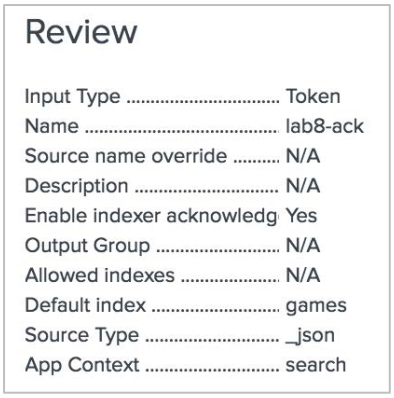
23. Click **Next>**. The page for the Input Settings step appears.
24. For the **Source type**, click **Select** – and from **Select Source Type**, select **Structured > _json**.



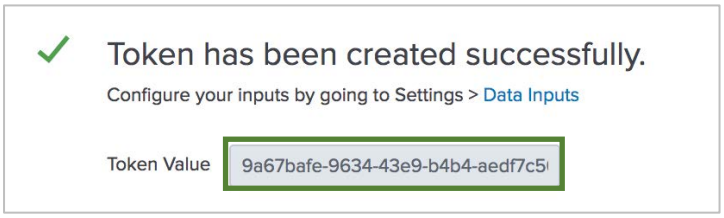
25. For the **App Context**, click **Search & Reporting (search)**, which selects the search app.
26. For the **Default Index**, select **games**.



27. Click **Review>**. The page for the Review step appears and should match the settings below. If they aren't matching the instructions, click **<Back** and make any necessary corrections.



28. Click **Submit**>. A success page appears, showing the Token Value.
29. Copy the **Token Value** on to your computer's clipboard as you will need it in the next task. Make sure you select the entire contents of the Token Value field! (You might want to paste this value into a text editor, for easier use later.)



Task 4: Use `curl` commands to use HEC tokens with indexer acknowledgment enabled.

30. Create a `curl` command that uses an HEC token with indexer acknowledgement to ingest `lab8-task4-data.json` into the games index.

> **NOTE:** For the channel, use 934793C0-FC91-979E-265A-7EAACEFBC724. Recall from the lecture that a channel is a GUID value which serves to identify a client. (When selecting a GUID to represent a client, any random GUID could be used.)

31. Execute the command. A “Success” message, including an `ackId` should be returned.

```
{"text":"Success","code":0,"ackId":0}
```

    Note the `ackId` value. You will need this for the next `curl` command.
32. Create a `curl` command that verifies that the file you ingested was received by the indexer.
33. Execute the command. You should see a message similar to the one shown below. (Note that the number matches the ackId that was returned by the previous request.)

```
{"acks":{"0":true}}
```

Task 5: Read through and execute a Python program that ingests data using HEC – as JSON or RAW, with or without indexer acknowledgement.

In this task, you will read through and then execute `hec.py`. However, unlike the other programs in the `solutions` directory, you will need to make one small change: to set the value of the token_ack variable.

NOTE: The solution file (`hec.py`, in `/home/student/rest-course/solutions`) does NOT have a valid value for `token_ack`. Currently, it is set to ++. If you use the solution file for `token_ack`, replace ++ with the actual value. (You created the HEC token lab8-ack in a previous task.)

You will execute it at the command line, passing in the format of the data to ingest (JSON or RAW), the name of the file containing the data to ingest, and whether or not to use indexer acknowledgement:

python hec.py [JSON | RAW] | filename | [ack]

When you execute `hec.py`, it connects to Splunk, ingests data, and outputs the ingestion status. If indexer acknowledgement is used, then the indexer acknowledgment success or failure is output.

34. In the SSH session on the Linux server, open **`hec.py`** in your chosen editor.
    (If the file does not open, make sure the command line points to the `python` directory by executing the command below.)
    ```
    cd ~/rest-course/solutions
    ```
35. Look for this text: ++. You will see the current value of `token_ack`. Replace ++ with the value of the lab8-ack token that you created earlier.
36. Look for the instructional comments beginning with Lab Exercise 8, Task 5. Note how the Python code uses REST API calls to authenticate with the Splunk server and ingest data in json or raw format using a token. Also note how the REST API handles data ingestion using a token and acknowledgement – including how it obtains and parses the output from the Splunk server.
37. Use the program to ingest json data by executing at the command line:
    ```
    python hec.py json my_json_data.json
    ```
    You should see that this message is returned:
    ```
    Your data was sent to the indexer!
    ```
38. Use the program to ingest and acknowledge json data by executing at the command line:
    ```
    python hec.py json my_json_data.json ack
    ```
    Then this is returned:
    ```
    Your data was sent to the indexer!
    Your content was acknowledged by the indexer!
    ```
39. Use the program to ingest raw data by executing at the command line:
    ```
    python hec.py raw my_raw_data.txt
    ```
    You should see that this is returned:
    ```
    Your data was sent to the indexer!
    ```
40. Use the program to ingest and acknowledge raw data by executing at the command line:
    ```
    python hec.py raw my_raw_data.txt ack
    ```
    Then this is returned:
    ```
    Your data was sent to the indexer!
    Your content was acknowledged by the indexer!
    ```
41. Observe the error that gets sent when you specify json as the data format but specify a raw text file as data:
    ```
    python hec.py json my_raw_data.txt
    ```
    You should see this error:
    ```
    Invalid data format
    ```

Task 6: Search for the events that you ingested into the games index.

42. In Splunk Web, navigate to the Search page for the Search & Reporting app. (You can do this by navigating to the app panel and then clicking Search & Reporting.)
43. In the search box, execute this search:

```
index=games
```

All of the events that you ingested via HEC should now appear as search results.

Solutions

Lab Exercise 8

Task 2, Step 13. The file content is correctly formatted JSON.

```
[
  {
   "misc": "Lab 8 Task 2",
   "contractor": "Shania Jenkins",
   "role": "Game Creator"
   },
  {  "contractor": "Mehmet Ozgur",
     "role": "Management Consultant",
     "projects" : ["Hark - Who Goes There", "Tatertots"]
  }
]
```

However, it does not conform to the standards required by Splunk. For example, the custom fields are not enclosed within an `event` object. For this to work, the file content should be:

```
[
  { "event": {
      "misc": "Lab 8 Task 2",
      "contractor": "Shania Jenkins",
      "role": "Game Creator"
      }
   },
  { "event": {
       "contractor": "Mehmet Ozgur",
       "role": "Management Consultant",
       "projects" : ["Hark - Who Goes There", "Tatertots"]
       }
  }
]
```