Developing with the Splunk REST API

Document Usage Guidelines

- Should be used only for enrolled students
- Not meant to be a self-paced document, an instructor is needed
- Do not distribute

Course Objectives

- Introduce the Splunk Enterprise REST API
- Identify how to perform different types of calls to perform tasks
- Understand and use the Splunk authentication features
- Manipulate objects, perform tasks and perform different types of searches, all using the API
- Define how to parse the outputs provided by the REST API using Python or other language
- Manipulate KV Stores using the REST API
- Ingest data using the HTTP Event Collector

Course Prerequisites

- Required:
  - Splunk Fundamentals 1
  - Splunk Fundamentals 2
- Recommended:
  - Splunk Enterprise Data Administration or Splunk Cloud Administration
  - Software development or scripting experience

Course Outline

Module 1: Introducing the Splunk REST API

Module 2: Namespaces and Object Management

Module 3: Parsing Output

Module 4: Oneshot Searches

Module 5: Normal and Export Searches

Module 6: Advanced Searching and Job Management

Module 7: Working with KV Stores

Module 8: Using the HTTP Event Collector

Appendix: Useful Admin REST APIs

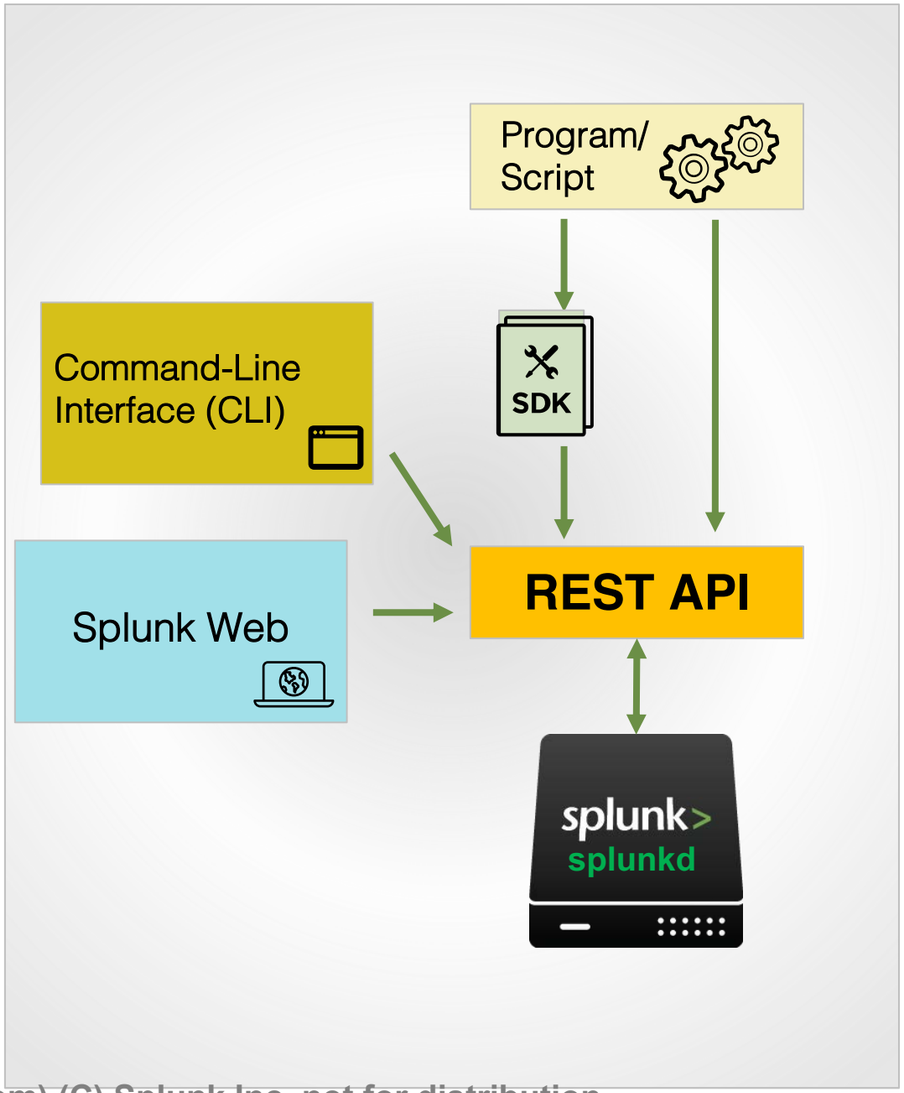
Module 1:
Introducing the Splunk REST API

Objectives

- Introduce the Splunk development environment and its REST endpoints
- Determine which Splunk server you should be connected to in order to accomplish a task
- Authenticate on a Splunk server with and without a session

What is the Splunk REST API?

- The Splunk **REST API**:
  - An **A**pplication **P**rogramming **I**nterface based on the Representational State Transfer (REST) architecture
  - Provides methods for accessing all Splunk product features
  - Accesses the Splunk server using HTTP or HTTPS
- All communication with the splunkd service is done through the REST API

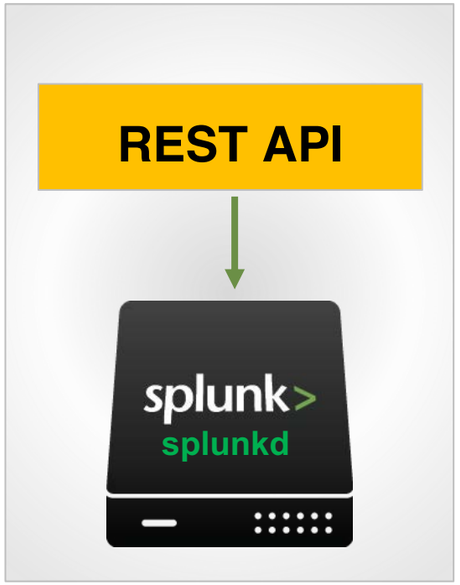


Splunk REST Endpoints

- The **splunkd** process is the server for REST calls
- REST URLs are based on the following format:

  scheme`://`*host*`:`*port*`/`**services(NS)**`/`*endpoint*

- *scheme*: HTTPS (default) or HTTP
- *host*: IP or DNS of server
- *port*: defaults to 8089
- *endpoint*: name of resource, such as `auth/login`, `search/jobs`, etc.



Note

The lab environment for this course uses HTTP to call REST endpoints.

Which Splunk Server to Connect To?

All Splunk servers (including Universal Forwarders) run a **splunkd** process and support REST endpoint access – for example:

- Connect to **search heads** to execute searches or work with knowledge objects (also valid for Splunk Cloud)
- Connect to **indexers** to manage indexes or upload events
- Connect to **forwarders** to create new inputs
- Connect to **deployment servers** to manage deployment servers and **deployment clients**
- Connect to the **cluster master** to access and configure cluster members
- Connect to the **license server** to perform license management

REST API on Splunk Cloud

- Splunk Cloud also exposes a REST API, and you can use it to perform most of the activities learned in this class.
- However, on Splunk Cloud some endpoints are disabled, as part of the system is managed exclusively by the Splunk Cloud administrators
  - For example: endpoints that change configuration files or system configurations are not enabled for updates and remain read-only
  - Some features are enabled by request, like the HTTP Event Collector
- Contact Splunk Cloud support for more information

Custom REST APIs

- Apps can implement their own endpoints, effectively extending the API. These endpoints execute Python code and can interact with Splunk, adding a lot of power and flexibility to your applications
  - The Splunk Python SDK contains examples on how to create your own APIs
  - Get the Python SDK: dev.splunk.com
- Some apps like **Enterprise Security** (ES) and **IT Service Intelligence** (ITSI) expose REST endpoints that can be used to integrate with external applications
  - For example, an external application reads notable event information from ES or KPIs from ITSI
  - These endpoints are also accessible through the service port

REST User Accounts

- Any Splunk account can connect to Splunk using REST endpoints
  - Access is constrained to the current permissions of the account
- The user account must be in a role that allows the actions you want to perform
  - Search, real-time search, administer objects, etc.
- User accounts in the admin role are unrestricted
- A good practice is to create accounts exclusively for REST API access and only give a minimum necessary set of permissions

Note

Consider creating a special REST user account in the admin role for REST endpoint client applications to use.

Splunk REST Logging

- All activities on a Splunk server are logged in the **`_internal`** and/or **`_audit`** indexes
- If a REST client application authenticates as a specific user, it is easy to search or monitor these indexes for application events

```
index=_internal user=restclient …
```

- The **`client_ip`** field also helps determine the log entries that belong to a specific client

Requests

- The following CRUD operations are supported: **GET**, **POST**, **DELETE**
- The REST endpoints are used with HTTP requests
  - Use **GET** requests to get data about resources
  - Use **POST** requests to create or modify resources
  - Use **DELETE** requests to delete a resource
- The documentation for each endpoint includes:
  - A list of the supported CRUD operations
  - The request parameters available for each operation

Note

The Splunk REST API documentation can be found here:
http://docs.splunk.com/Documentation/Splunk/latest/RESTREF/RESTprolog

Note

CRUD refers to **C**reate, **R**etrieve, **U**pdate, and **D**elete operations.

Requests (cont.)

- Parameters passed in a request must be URL-encoded
  - This allows Splunk to process request parameters that contain reserved characters. These characters need to be properly URL-encoded before dispatching the call to splunkd
  - Reserved characters include:

```
! * ( ) ; : @ & = + $ , /  ? # [ ]
```

```
http://localhost:port/services/saved/searches/myreport
/dispatch?args.comment=%22this%20text%20has%20been%20urlecoded%22
```

Note

You can use urlencoder.org to manually URL-encode text

 |

Responses and Atom Syndication

- Most – but not all – REST API "success" responses are based on the Atom Syndication format
  - This response formatted in XML
- However, JSON and other formatting can be specified instead of Atom
  - Available encoding formats vary per endpoint
  - Atom and JSON available to almost all endpoints
  - See also: docs.splunk.com/Documentation/Splunk/latest/RESTUM/RESTusing#Atom_Feed_response

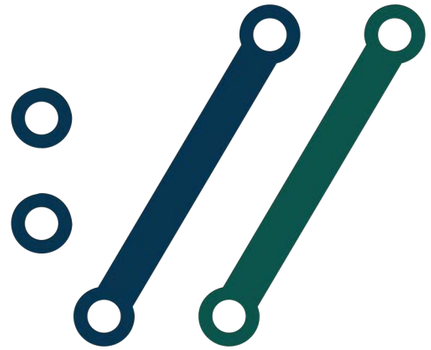
Using `curl` to test REST

- You can use the UNIX `curl` command to call any REST endpoint:

```
$curl -s http://host:port/services/endpoint
```

- To pass arguments for a command, use `-d`:

```
$curl -s http://host:port/services/endpoint
-d arg1=value -d arg2=value…
```



- Some other `curl` arguments:
  - `-i` returns output headers
  - `-k` skips SSL certificate validation
  - `-G` sends `GET` request, with its data URL encoded
  - `-H` allows you to specify a header
  - `-s` strips extra output formatting
  - `-u` lets you specify user credentials for password authentication
  - `-v` returns verbose output
  - `-X` http method: GET, POST, DELETE

Using `curl` to test REST (cont.)

- To send a request, use the **-X** argument followed by the request type, for example:

```
$curl -X POST -s http://host:port/services/endpoint
-d arg1=value -d arg2=value…
```

- By default, the request output is written to **STDOUT** – in the training environment, this is your command window
- If you want to pass POST-style parameters when executing a GET call, use the option -G, and curl converts the -d arguments into url-encoded arguments

> **Note**
>
> If a request type is not explicitly specified using the **-X** argument, **curl**'s default behavior is to send a **GET** request. However, if there are **-d** arguments, the default behavior is to send a **POST** request.

Using `Postman` to Test REST Calls

- The REST API can be called by any application, programming/script language, appliance or resource capable of making HTTP calls
- In this class you can also use **`Postman`**, a visual tool designed for REST API testing that works on all major platforms: Windows, MacOS or Linux
- It can work from your browser, or you can download a desktop version (desktop agent)
- It has free and paid versions

Note

To download Postman go to:
https://www.postman.com/downloads/

Authenticating with the REST API

- Use the **`auth/login`** endpoint to authenticate with splunkd
  - The **`username`** and **`password`** are passed as arguments
  - Upon successful authentication:
    - A session is created on splunkd
    - A **`sessionKey`** token is returned, which can be used in subsequent REST API calls

```
$curl –k -s http://localhost:8089/services/auth/login -d
username=admin –d password=splunk3r

<response>
  <sessionKey>5efb11a80a0ff34e89dab51e7a2e29f1</sessionKey>
</response>
```


Example: Create a Session

Once the **sessionKey** is obtained, it can be re-used later by adding it to the request header

```
$curl –k -s http://localhost:8089/services/auth/login -d username=admin
–d password=splunk3r

<response>
  <sessionKey>5efb11a80a0ff34e89dab51e7a2e29f1</sessionKey>
</response>
```

```
$curl -H "Authorization: Splunk 5efb11a80a0ff34e89dab51e7a2e29f1"
http://localhost:8089/services/…
```

Note

All subsequent examples assume that the **sessionKey** is in the request header.

 |

auth/login and Cookies

- **auth/login** has an optional argument: **cookie**
- If **allowCookieAuth** is enabled on the Splunk server, then you can have **auth/login** return a **Set-Cookie** header
  - Set **cookie=1** in the **POST** request

```
$curl –i –k -s http://localhost:8089/services/auth/login -d username=admin
–d password=splunk3r –d cookie=1
HTTP/1.1 200 OK
…
Set-Cookie: splunkd_8089=
Dcn6QLpFC^cAt0N4Zb8EoOBiEozOKPXhM_Fzs85MqNBzh965IbhSq1TKpduCs9^91aDPcpERtKn9h3L^IYV4rl
IZG77IPfj9NXXbBfUJymuiePz7FTNHR^IFvjI3; …

<response>
  <sessionKey>5efb11a80a0ff34e89dab51e7a2e29f1</sessionKey>
</response>
```

auth/login and Cookies (cont.)

- The information from the **Set-Cookie** header can be used in subsequent Splunk requests
  - Put the cookie value in double quotes (using the cookie returned previously)

```
$curl –k -s --cookie "splunkd_8089=
Dcn6QLpFC^cAt0N4Zb8EoOBiEozOKPXhM_Fzs85MqNBzh965IbhSq1TKpduCs9^91aDPcpE
RtKn9h3L^IYV4rlIZG77IPfj9NXXbBfUJymuiePz7FTNHR^IFvjI3"
http://host:port/endpoint
```

- Even if a cookie is available, it is still possible to use the Authorization header

```
$curl –H "Authorization: Splunk sessionKey-value" -s
http://host:port/services/…
```


Deleting a Session

- To delete a session, either:
  - Let the session time out (60 minutes, by default), *or*
  - Use the **`/authentication/httpauth-tokens/`*`sessionKey`*** with a **`DELETE`** request

```
$ curl -X DELETE -k -s -u admin:splunk3r
http://localhost:8089/services/authentication/httpauth-
tokens/f1319d13d04b2b482cabd6569cd2c7c9
```

- By default, only users in the admin role can delete their own sessions. That is why the **`-u`** argument is specified here, with admin credentials

Example: Delete a Session (Success)

- Another solution: have a Splunk administrator add the **edit_httpauth** capability to the user's role
- If successful, Splunk returns the session information.

```
<?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" xml
ns:opensearch="http://a9.com/-/spec/opensearch/1.1/">
  <title>httpauth-tokens</title>
  <id>https://localhost:8089/services/authentication/httpauth-tokens</id>
  <updated>2018-01-31T21:42:24+00:00</updated>
  <generator build="004410bdc029" version="7.1.0"/>
  <author>
    <name>Splunk</name>
  </author>
  <link href="/services/authentication/httpauth-tokens/_acl" rel="_acl"/>
  <opensearch:totalResults>7</opensearch:totalResults>
  <opensearch:itemsPerPage>30</opensearch:itemsPerPage>
  <opensearch:startIndex>0</opensearch:startIndex>
  <s:messages/>
  <entry>
    <title>025e678973b075c5bdba81fd9b5abbd3</title>
    <id>https://localhost:8089/services/authentication/httpauth-tokens/025e678973b075
c5bdba81fd9b5abbd3</id>
```

Note

Recall that most REST API responses are based on XML Atom Syndication, but later you'll learn that other formats can also be used.

Example: Delete a Session (Failure)

If the deletion failed, an error message is returned

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <messages>
    <msg type="ERROR">error information here</msg>
  </messages>
</response>
```

Note

Failure responses are sometimes very simple – like this – and not in a full Atom formatting.

Sessionless Authentication

- It is possible to call a single REST endpoint without creating a session in Splunk
- Requires that user credentials be passed with the REST endpoint
  – For example:

```
$curl -k -u admin:splunk3r -s http://localhost:8089/services/authentication/users

<entry>
    <title>admin</title>
…
 <entry>
    <title>mfleischman</title>
…
```

Note

Even though use of this method is OK during development, Splunk discourages its use in production. Token-based authentication is preferred in production environments.

User-Scoped Authentication Tokens

- Splunk supports authentication using **JWT** (JSON Web Tokens)
  - JWT is a specification that allows users to format and express tokens as JSON objects
  - JWT tokens are scoped per user
  - Tokens are valid only during their defined date-time range
- Tokens are stored on the server and can be:
  - Enabled, disabled, or revoked
  - Used with LDAP
  - Used across the search-head cluster

- Configure and view tokens using the REST API endpoints or using Splunk Web

Working with User-Scoped Tokens

- Generally, admins create tokens for users
- By default:
  - The admin role has all token related capabilities
  - Users have only the `list_tokens_own` capability – making them able to list their tokens that were created for them
- Token attributes include:
  - `status`: token status (enabled or disabled)
  - `lastUsed`: the time at which the token was last used
  - `lastUsedIP`: the last IP address from which token was last used

Authenticating with a User-Scoped Token

- Send the token in the Authorization header, using the Bearer schema
  - A sample token is shown below

```
curl -k -s -H "Authorization: Bearer
eyJraWQiOiJzcGx1bmsuc2VjcmV0IiwiYWxnIjoiSFM1MTIiLCJ2ZXIiOiJ2MSIsInR0eXAiOiJzdGF0aWMifQ
.eyJpc3MiOiJhZG1pbiBmcm9tIGlwLTEwLTAtNDItOTkiLCJzdWIiOiJhZG1pbiIsImF1ZCI6InRyYWluIiwia
WRwIjoic3BsdW5rIiwianRpIjoiYWQ1NTAwMGEyNDRiNmI2MTkzZGQ0YzZjZDM4YjEyNDE0OGIxZTJhYTYxMGM
yMzA1ODI3MGY3YmIyODJlMzUzMyIsImlhdCI6MTU1ODQ2MDA1NCwiZXhwIjoxNTYxMDUyMDU0LCJuYnIiOjE1N
Tg0NjAwNTR9" http://localhost:8089/services/search/jobs -d exec_mode=oneshot -d
search="search index=web sourcetype=access_combined" -d output_mode=json
```


REST API Documentation

The official REST API documentation:
https://docs.splunk.com/Documentation/Splunk/latest/RESTREF/RESTprolog

Other useful resources:

- REST API Tutorials:
  https://docs.splunk.com/Documentation/Splunk/latest/RESTTUT/RESTTutorialIntro
- REST API User Manual:
  https://docs.splunk.com/Documentation/Splunk/latest/RESTUM/RESTusing
- Custom REST endpoints (create your own endpoints on custom apps):
  https://dev.splunk.com/enterprise/docs/devtools/customrestendpoints

REST API Reference Manual

- REST endpoints are organized into categories
- Click a category to reveal a list of endpoints and their documentation
- For example, for information about **`auth/login`**, click the **Access control** category

Resource groups

Resources are grouped into the following categories.

Category	Description
Access control	Authorize and authenticate users.
Applications	Install applications and application templates.
Clusters	Configure and manage indexer clusters and search head clusters.
Configuration	Manage configuration files and settings.
Deployment	Manage deployment servers and clients.
Inputs	Manage data input.
Introspection	Access system properties.
Knowledge	Define indexed and searched data configurations.
KV store	Manage app key-value store.
Licensing	Manage licensing configurations.
Outputs	Manage forwarder data configuration.
Search	Manage searches and search-generated alerts and view objects.
System	Manage server configuration.
Workload management	Manage system resources for search workloads.

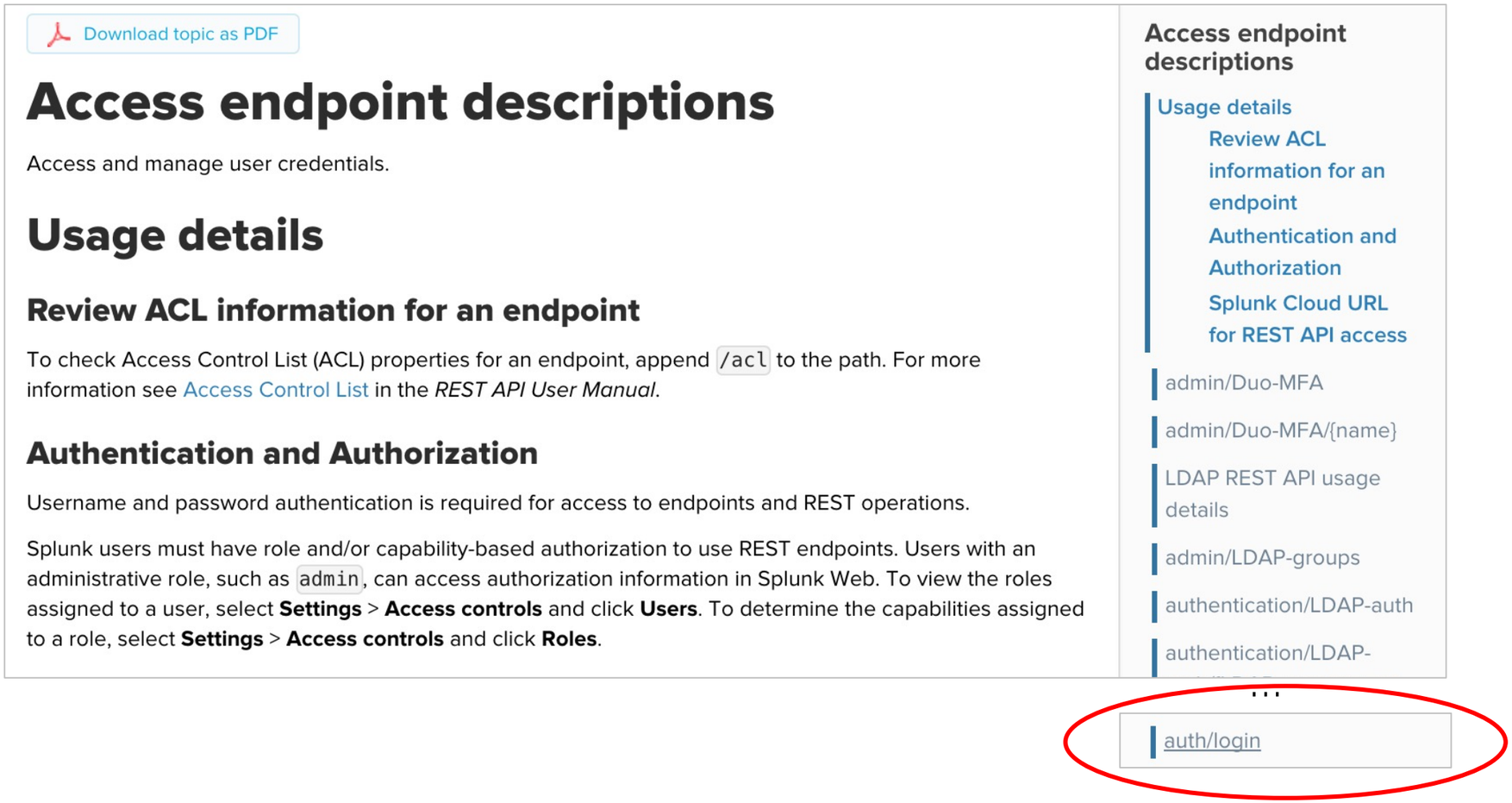
See the Endpoints reference list for an alphabetical list of endpoints.

Note

If you do not know the category of a particular endpoint, click **Endpoints reference list**. This shows an alphabetical list of all endpoints.

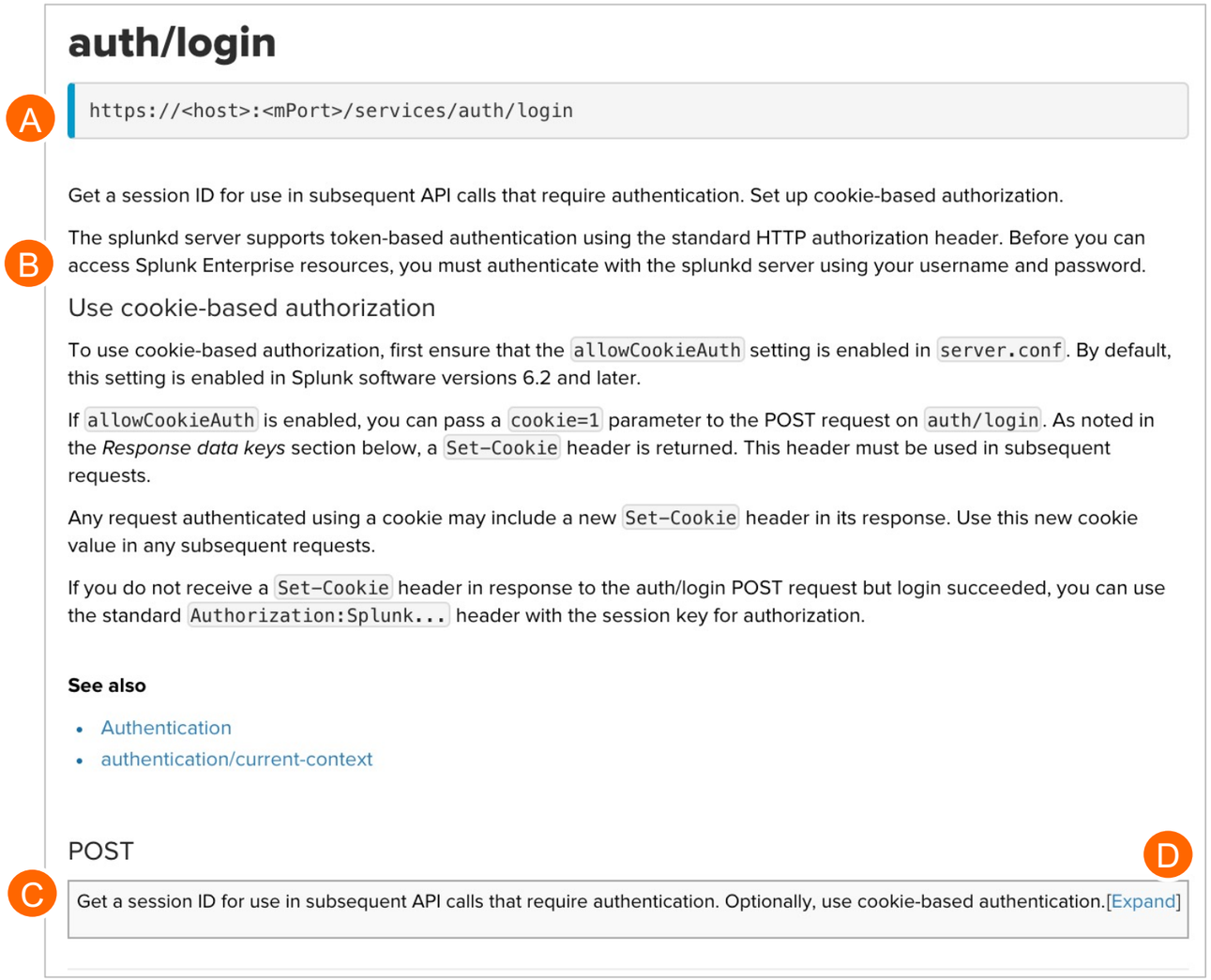
Example: `auth/login`

On the right, scroll to the endpoint – in this case, `auth/login`:



Example: `auth/login` (cont.)

- The endpoint documentation includes:
  - A The URL syntax
  - B A description
  - C Information about the request parameters
- To get information about request parameters, click the **Expand** link D



Note

`auth/login` does not have any `GET` request parameters, so only `POST` is shown.

Example: `auth/login` (cont.)

- After clicking **Expand**, the `POST` request parameters are shown
- Note some parameters may be marked as **required**
- In addition, most endpoints support the `output_mode` request parameter to change the output format (for example to JSON instead of XML)
  - Formats available vary per endpoint

POST

Get a session ID for use in subsequent API calls that require authentication. Optionally, use cookie-based authentication or multifactor authentication. [Collapse]

Request parameters

Name	Type	Description
cookie	Boolean, only used value is 1.	To use cookie-based REST auth, pass in `cookie=1`. Cookies will only be returned if the cookie parameter is passed in with the value of 1.
password	String	**Required**. Current *username* password.
passcode	String	**Required for users with RSA multifactor authentication**. The passcode associated with RSA multifactor authentication. This is a combination of the user's RSA token and PIN.
username	String	**Required**. Authenticated session owner name.

Note

In the lab exercise, you will use the `output_mode` with `auth/login` to produce a `JSON` response.

Example: `auth/login` (cont.)

Below the request parameters, information is shown about the response generated by the endpoint

Response data keys

Note: Only a `<response>` element is returned instead of a full `<atom>` feed.

Name	Description
sessionKey	Session ID.

A `Set-Cookie` HTTP header is returned if cookie-based authentication is requested.

Failure to authenticate returns the following response.

```
<response>
     <messages>
         <msg type="WARN">Login failed</msg>
     </messages>
</response>
```

Example request and response

XML Request

```
curl -k -u admin:changeme  https://localhost:8089/services/auth/login -d userna
me=admin -d password=changeme
```

XML Response

```
<response>
    <sessionKey>192fd3e46a31246da7ea7f109e7f95fd</sessionKey>
</response>
```

Note

The response from `auth/login` is very simple. Later in the course, you are shown longer request responses.

Module 1 Lab Exercise

Time: 20 minutes

Tasks:

- Set the name of your Splunk server
- Explore your Splunk environment
- Create a Splunk session
- Delete a Splunk session
- Call a Splunk endpoint using a cookie
- Call a Splunk endpoint without creating a session

Module 2:
Namespaces and Object Management

Objectives

- Understand general CRUD with the REST API
- Describe how a namespace affects access to objects
- Use the `servicesNS` node and a namespace to access objects
- Describe how the sharing level and access control lists affect access to objects
- Modify the sharing level and the permissions of an object
- Define the `rest` command

Generic CRUD Operations on REST API

- In Splunk, most of the APIs available follow similar standards for object operations. These operations are called CRUD (Create, Read, Update and Delete)
- Splunk uses three different HTTP methods to perform these operations:
  - **GET** to obtain information about an object or read data
  - **POST** to create or update objects and launch searches*
  - **DELETE** to remove objects
- In `curl`, the http method is specified with the option -X. For example: `curl -X POST…`

Note *

When you launch a search using POST you are effectively creating a job on the Splunk server; therefore, it's a create operation.

Generic CRUD operations – Read

- To retrieve lists of objects and their properties or results of searches, make a GET call to the endpoint
  - Refer to the documentation to ensure the endpoint supports GET

```
curl -X GET –s -k -u username:password
http(s)://host:servicePort/services/endpoint
```

- To retrieve information about a specific object, append the object name after the endpoint

```
curl -X GET –s -k -u username:password
http(s)://host:servicePort/servicesNS/endpoint/objectName
```


Generic CRUD operations – Create/Update

- To create an object, make a **POST** call to the endpoint. Give the object a name and set additional properties
  - Use the **-d** option in **`curl`** to pass as many parameters as you need

```
curl -X POST –s -k -u username:password
http(s)://host:servicePort/services/endpoint
-d name=newObjectName -d property1=value1 –d property2=value2...
```

- To edit/update an object, make a **POST** call to the endpoint/object name and set new properties

```
curl -X POST –s -k -u username:password
http(s)://host:servicePort/services/endpoint/objectName
-d property1=newValue1 –d property2=newvalue2...
```


Generic CRUD operations – Delete

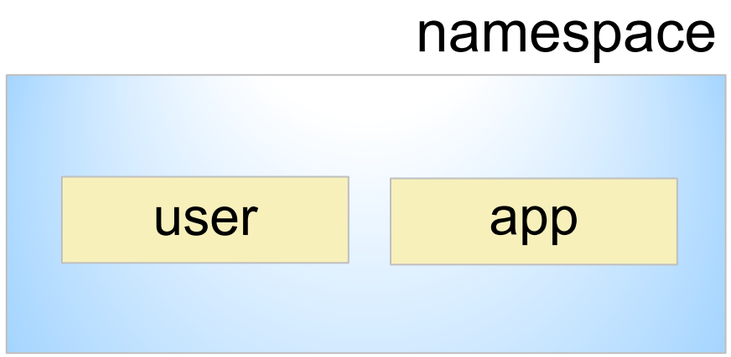
- To delete an object, make a **DELETE** call to the specified object

```
curl -X DELETE -s -k -u username:password
http(s)://host:servicePort/services/endpoint/objectName
```

- Important considerations:
  - Objects can only be deleted one at a time
  - DELETE calls to endpoints without specifying an object name throw a fatal error: The method is not allowed
  - Deleting an object is final, there's no undo. BE CAREFUL!

Namespaces

- A session operates within a **namespace**, which is a combination of a user ID (the owner) and an app name
  - Example: **pat/search**
- If a specific owner and app value are not specified, a default namespace is used, based on:
  - The user ID
  - The user's default app
- The namespace filters the objects that can be returned to a user by the REST API



Why Namespaces Matter

- The user's **role** defines access – read, write, or none – to knowledge objects, indexes, and other Splunk entities
- The **app** can contain knowledge objects that affect searches executed in a namespace
  - Includes saved searches and field extractions
  - The **app** attribute can also be a wildcard (-), providing access to all accessible apps at one time
- Users in the **Admin** role:
  - Have total access to all objects in Splunk
  - Can specify a namespace **user** attribute to other usernames or to wildcards

servicesNS

- To specify a namespace with an endpoint, use the **servicesNS** node: **servicesNS/*user*/*app***
- To get information about the saved search **abc**, in **terry/search**

```
curl -k -X GET -s -u restclient:restC0der
http://localhost:8089/servicesNS/terry/search/saved/searches/abc
```

- To list saved searches that are not private for user terry, regardless of the app, replace the app with **-**, for example

```
curl -k -X GET -s -u restclient:restC0der
http://localhost:8089/servicesNS/terry/-/saved/searches
```

Note

The **saved/searches** endpoint lists information about saved searches (i.e., reports.)

servicesNS (cont.)

- To see all saved searches for the search app, regardless of the user, replace the user with -, for example

```
curl -k -X GET -s -u restclient:restC0der
http://localhost:8089/servicesNS/-/search/saved/searches
```

- Because this example does not specify a namespace, it returns all the objects that this user terry is able to see

```
curl -k -X GET -s -u terry:restC0der
http://localhost:8089/servicesNS/-/-/saved/searches
```


servicesNS (cont.)

- To see all saved searches in all apps (except user private searches)
  - Replace the user *and* the app with -
  - Execute as a user in the admin role

```
curl -k -X GET -s -u restclient:restC0der
http://localhost:8089/servicesNS/-/-/saved/searches
```

Note

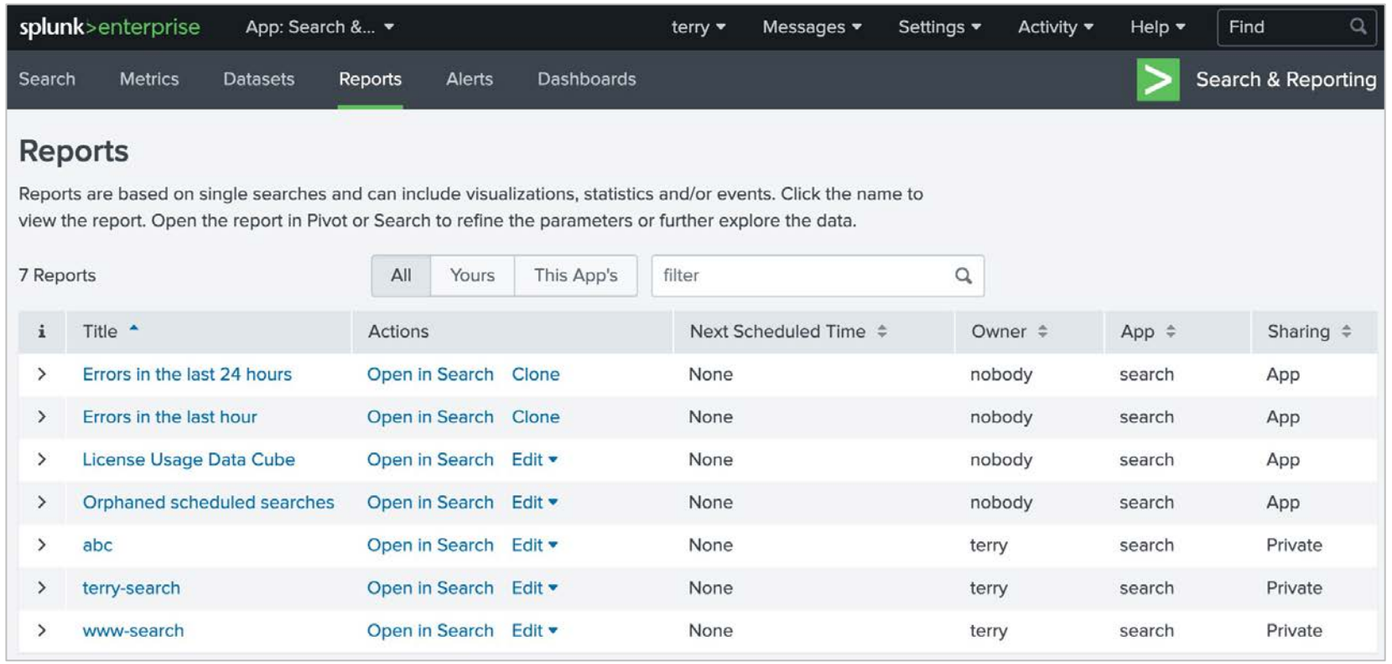
In your student environment, restclient is a user who has been assigned to the admin role.

Sharing Level and Access Control Lists

- Each object in an app and has a **sharing level** and an **Access Control List** (ACL)
- Sharing can be set to **user**, **app**, or **global**
  - **user**: only the owner has access (also known as private)
  - **app**: all users who can access the app can access, based on the ACL
  - **global**: all users in Splunk can access, based on the ACL
- Other users can have read/write, read only, or no access, depending on role entries in the ACL
- Entity owners always have read/write permission for the entity

Identifying ACL

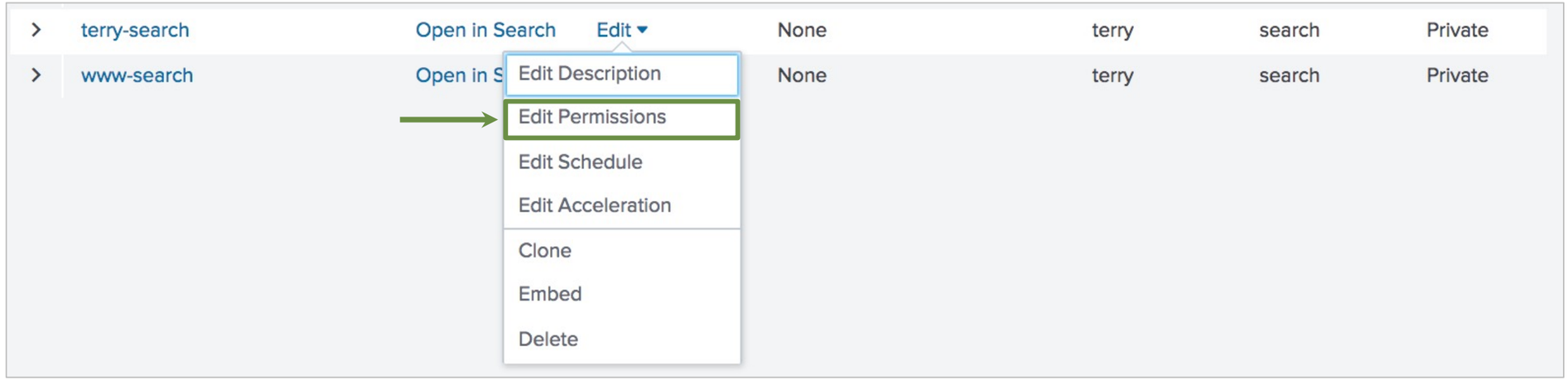
- For example, here is list of all reports that user **terry** can see



- Terry can see **public** reports (other owners, shared at the app level) and also his own **private** reports, visible only to him

Identifying ACL (cont.)

- Access control lists are sometimes referred to as permissions



- Access control lists are a mapping of role and access
- Access is specified as **read** or **write**
  - If no access is assigned, only the object owner and users in the role have access to the object

Getting ACL Information

- Many REST endpoints support adding `/acl` to the endpoint
  - Directs Splunk to return the ACL data for the object

```
curl -k -X GET -s -u restclient:restC0der
http://localhost:8089/servicesNS/terry/search/saved/searches/terry-
search/acl
```


Getting ACL Information (cont.)

When no values are defined for **perms**, the default is that only the object owner has access. Changing the permissions causes sub-elements and values to be added here.

Note

For information about ACL attributes, refer to: http://docs.splunk.com/Documentation/Splunk/latest/RESTREF/RESTprolog

```
<feed>…
 <entry>…
   <content>
     <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">terry</s:key>
       <s:key name="can_share_global">1</s:key>
       <s:key name="perms"/>
       <s:key name="removable">1</s:key>
       <s:key name="sharing">user</s:key>
     <s:/dict>
    </content>
</entry>…
```


Changing ACL Properties

- Many REST endpoints support actions to change an object's ACL properties
- To update an ACL property, except for the **`app`** property:
  - A Specify the namespace in the endpoint path
  - B Include the **`sharing`** and **`owner`** parameters with the **`POST`** request
- For example, to change write permission to the admin and user roles on terry-search:

```
curl -k -X POST -s -u restclient:restC0der
http://localhost:8089/servicesNS/terry/search/saved/searches/terry-
search/acl -d owner=terry -d sharing=app
-d perms.write='admin, user'
```


Changing ACL Properties (cont.)

Note that **perms** got expanded and now contains the newly defined sharing properties.

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

```
<feed>…
 <entry>…
   <content>
     <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">terry</s:key>
       <s:key name="can_share_global">1</s:key>

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


Changing ACL Properties (cont.)

- To change permissions for Everyone, specify *****
- For example, to change read permission for everyone on www-search, in the terry/search namespace:

```
curl -k -X POST -s -u restclient:restC0der
http://localhost:8089/servicesNS/terry/search/saved/searches/www-
search/acl -d owner=terry -d sharing=app -d perms.read='*'
```

```
<s:key name="perms">
  <s:dict>
    <s:key name="read">
      <s:list>
        <s:item>*</s:item>
      </s:list>
    </s:key>
  </s:dict>
</s:key>
```


Changing the Sharing Level

- To change the sharing level
  - Specify the namespace in the endpoint path
  - With the **`POST`** request:
    - A Include the **`owner`** parameter
    - B Set the **`sharing`** to the desired level: **`user`**, **`app`**, or **`global`**
- For example, to change sharing to global on terry-search:

```
curl -k -X POST -s -u restclient:restC0der
http://localhost:8089/servicesNS/terry/search/saved/searches/terry-
search/acl -d owner=terry -d sharing=global
```

A B

Changing the Sharing Level (cont.)

New sharing level is displayed on the API call response.

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


Moving an Object to a Different App

- To change app context for an object:
  - A Specify the namespace in the endpoint path
  - B Add **`/move`** to the desired endpoint
  - C Include the **`app`** and **`user`** parameters with the **`POST`** request

```
curl -k -X POST -s -u restclient:restC0der A
http://localhost:8089/servicesNS/terry/search/saved/searches/terry-
search/move -d owner=terry -d app=sample_app
                B                  C
```

Note

If a non-existent app is specified, an error is produced.

Moving an Object to a Different App (cont.)

The properties on the results of the call should reflect the new app. On the configuration files the object has been physically removed from the settings of the original app and placed on the configuration files of the destination app.

In this case, the configuration file where reports are saved is savedsearches.conf.

```
<feed>…
 <entry>…
   <content>
     <s:dict>
       <s:key name="app">sample_app</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">terry</s:key>
       <s:key name="can_share_global">1</s:key>
       <s:key name="perms">…<s:key>
       <s:key name="removable">1</s:key>
       <s:key name="sharing">user</s:key>
     <s:/dict>
    </content>
</entry>…
```


The `rest` Command

- To get data from a REST endpoint and use it within a Splunk search processing language (SPL) string, use the **`rest`** command
- The **`rest`** command is useful to get data for use in reports and dashboards that is not otherwise available in SPL – for example:
  - Splunk metadata
  - Information about apps, permissions, roles, KOs or any object
  - Any information that can be obtained from a REST endpoint
- Only supports **GET** calls. POST and DELETE aren't supported
- Can be used to easily test REST API calls from Splunk Web

`rest` Command Syntax

Basic syntax:

```
| rest endpointURI [optionalArguments]
```

Argument	Description
`count`	Limits the number of results returned. Default=0 (unlimited)
`requestParameters`	Request parameters can be passed as `parameter=value` pairs
`splunk_server`	• The search peer from which to return results • Only one value can be specified – but a wildcard (*) can be used • Default: all search peers return data
`splunk_server_group`	Specify one or more groups of search peers ("server groups") to use
`timeout`	Specify the timeout in seconds when waiting for the REST endpoint to respond (Default=60)

Example: Get Splunk User Information

- This example gets all Splunk users, except for admin and restclient
- Note that REST endpoints do not necessarily produce fields that are the same as those used in SPL
  - For example, the username is represented as **`title`**

```
| rest /services/authentication/users
| where NOT title="admin" OR
      title="restclient"
| rename title as "Username"
| fields Username
```

Username
pat
student
terry

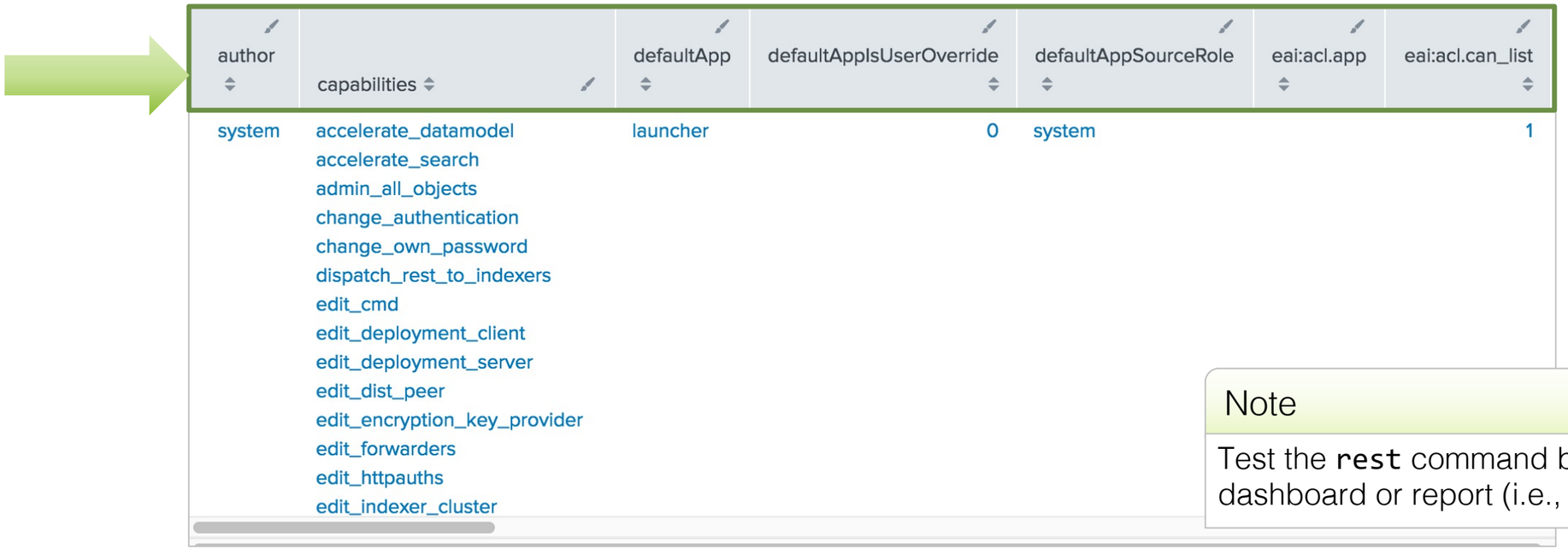
Note

The endpoint must begin with `/services` or `/servicesNS`.

Example: Get Splunk User Information (cont.)

- If you are unsure of a field's name
  - Refer to docs.splunk.com *or*
  - Use the **`rest`** command to call the endpoint and look at the column headings

```
| rest /services/authentication/users
```



author	capabilities	defaultApp	defaultAppIsUserOverride	defaultAppSourceRole	eai:acl.app	eai:acl.can_list
system	accelerate_datamodel	launcher	0	system		1
	accelerate_search					
	admin_all_objects					
	change_authentication					
	change_own_password					
	dispatch_rest_to_indexers					
	edit_cmd					
	edit_deployment_client					
	edit_deployment_server					
	edit_dist_peer					
	edit_encryption_key_provider					
	edit_forwarders					
	edit_httpauths					
	edit_indexer_cluster					

Note

Test the **`rest`** command before using it in a dashboard or report (i.e., saved search.)

Example: Get Names of Saved Searches

- Parameters are available for many methods:
  - Pagination and filtering parameters work for every method
  - Specific parameters for each API
- In this example, get all saved searches that include "Error"
- To do this, use the **`search`** request parameter

```
| rest /servicesNS/-/-/saved/searches
    search=Error
| fields title
```

title
Errors in the last 24 hours
Errors in the last hour
instrumentation.reporting.errors
instrumentation.reportingErrorCount
Splunk errors last 24 hours

Pagination and filtering parameters documentation:
https://docs.splunk.com/Documentation/Splunk/latest/RESTREF/RESTprolog#Pagination_and_filtering_parameters

Module 2 Lab Exercise

Time: 25 minutes

Tasks (using `curl`)

- List all saved searches for all users in all namespaces
- List all saved searches for user pat in any namespace
- List all saved searches for user terry in the search namespace
- List the ACL information for pat-search, in the search app, owned by user pat
- Change the permissions on pat-search so that anyone with power user role can edit the search
- Create the sharing on pat-search to global

Module 3:
Parsing Output

Objectives

- Understand the general structure of ATOM-based output
- Generate and use JSON output
- Write code that uses the API and parse responses

Atom Syndication Format

- Most – but not all – REST API "success" responses in XML are based on Atom
- Response output generally includes:
  - A Top-level elements with metadata
  - B One or more `<entry>` elements

```
<feed>
  <title>…</title>
  <id>…</id>
  <updated>…</updated>
  <generator />
  <author>…</author>
  <link>…</link>
  <opensearch:totalResults>…</opensearch:totalResults>
  <opensearch:itemsPerPage>…</opensearch:itemsPerPage>
  <opensearch:startIndex>…</opensearch:startIndex>
  <s:messages>…</messages>
  <entry>…</entry>
  <entry>…</entry>
  …
</feed>
```

A

B

- Generally, what you want to parse is found nested within `<entry>`

 |

Review: REST Output

- Earlier, you saw that this command:

```
curl -k -X GET -s -u restclient:restC0der
http://localhost:8089/servicesNS/terry/search/saved/searches
```

- Produced this output, with each saved search name returned as a `<title>` within an `<entry>`

```
<feed>…
  <entry><title>Errors in the last 24 hours</title>…</entry>
  <entry><title>Errors in the last hour</title>…</entry>
  <entry><title>License Usage Data Cube</title>…</entry>
  …
</feed>
```

- Next, take a closer look at the contents of `<entry>`

<entry> Element

Each <entry> in the output begins like this:

```
<entry>
  <title>terry-search</title>
  <id>http://localhost:8089/servicesNS/terry/search/saved/searches/terry-search</id>
  <updated>2020-10-01T15:39:42+00:00</updated>
  <link href="/servicesNS/terry/search/saved/searches/terry-search" rel="alternate"/>
  <author>
    <name>terry</name>
  </author>
  <link href="/servicesNS/terry/search/saved/searches/terry-search" rel="list"/>
  <link href="/servicesNS/terry/search/saved/searches/terry-search/_reload" rel="_reload"/>
   …
  <link href="/servicesNS/terry/search/saved/searches/terry-search/history" rel="history"/>
  <content>
     …
  </content>
</entry>
```


<entry> Element (cont.)

Element	Description
title	Human readable name for the returned entry
id	Management URL for accessing the endpoint. It is also a unique path to this object within this instance of Splunk
updated	Date that this entity was last changed
link	URI for the endpoint to this entry, relative to the management port of a server instance

```
<entry>
  <title>terry-search</title>
  <id>http://localhost:8089/servicesNS/terry/search/saved/searches/terry-search</id>
  <updated>2020-10-02T15:39:42+00:00</updated>
  <link href="/servicesNS/terry/search/saved/searches/terry-search" rel="alternate"/>…
  …
```

<entry> Element (cont.)

Element	Description
author	Generally the owner of the entity, as defined in the ACL
link	Different from the link shown earlier in the entry. Consists of one or more URIs for the endpoint to this entry. Each URI refers to an action available on this entity, shown in the rel attribute
content	Container for content returned by the operation for an entry – usually, as an array of key/value pairs

```
  <author>
    <name>terry</name>
  </author>
  <link href="/servicesNS/terry/search/saved/searches/terry-search" rel="list"/>
  <link href="/servicesNS/terry/search/saved/searches/terry-search/_reload" rel="_reload"/>
  <link href="/servicesNS/terry/search/saved/searches/terry-search" rel="edit"/>
  …
  <content>…</content>
</entry>
```

<content> Element

- Key value pairs:
  - A key is represented as the name attribute with a **<s:key>** element
  - The value is the content of the **<s:key>** element
  - Key/value pairs vary per REST endpoint
- A set of key/value pairs is represented as a **<s:dict>** element

```
<content type="text/xml">
  <s:dict>
    <s:key name="action.email">0</s:key>
    <s:key name="action.email.sendresults" />
    <s:key name="action.email.to"></s:key>
    ...
    <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:dict>
     </s:key>
  …
  </s:dict>
</content>
```

Note

For information about the key/value pairs returned by a particular endpoint, refer to that endpoint's documentation on **docs.splunk.com**.

Not All Endpoint Output is the Same

- The output shown in the previous slides is used for some but not all REST endpoints. The output for each REST endpoint varies
- Before processing the output, make sure you know how the output is formatted
- The REST endpoint documentation:
  - Provides a description of each field returned in the output
  - Shows how the output for the endpoint is supposed to look for XML

Endpoint Documentation

- For example, for `saved/searches`, when called with `GET`, the **Returned values** list:
  - The name of each value that is returned
  - A description of each value

Returned values

Name	Description
action.email	Indicates the state of the email action.
action.email.sendresults	Indicates whether search results are attached to the email.
action.email.to	List of addresses for email recipients.
action.populate_lookup	Indicates the state of the populate lookup action.
action.rss	Indicates the state of the RSS action.
action.script	Indicates the state of the script action.
action.summary_index	Indicates the state of the summary index action.
alert.digest_mode	Indicates if alert actions are applied to the entire result set or to each individual result.
alert.expires	Sets the period of time to show the alert in the dashboard. Defaults to 24h. Uses [number][time-unit] to specify a time. For example: 60 = 60 seconds, 1m = 1 minute, 1h = 60 minutes = 1 hour.
alert.severity	The alert severity level. Valid values are: 1 DEBUG 2 INFO 3 WARN 4 ERROR 5 SEVERE 6 FATAL

Note

The Returned values for `saved/searches` are truncated here to fit this slide. For the full list, refer to docs.splunk.com.

Endpoint Documentation (cont.)

- After **Returned values**, a sample **XML Response** is shown
- To parse this from a program, use the programming languages:
  - Pattern matching utilities and/or
  - HTML DOM utilities

XML Response

```
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/"
      xmlns:s="http://dev.splunk.com/ns/rest">
  <title>savedsearch</title>
  <id>https://localhost:8089/services/saved/searches</id>
  <updated>2011-07-13T11:56:35-07:00</updated>
  <generator version="102824"/>
  <author>
    <name>Splunk</name>
  </author>
  <link href="/services/saved/searches/_new" rel="create"/>
  <link href="/services/saved/searches/_reload" rel="_reload"/>
  <!-- opensearch nodes elided for brevity. -->
  <s:messages/>
  <entry>
    <title>Errors in the last 24 hours</title>
    <id>https://localhost:8089/servicesNS/nobody/search/saved/searches/Errors%20in%20
the%20last%2024%20hours</id>
    <updated>2011-07-13T11:56:35-07:00</updated>
    <link href="/servicesNS/nobody/search/saved/searches/Errors%20in%20the%20last%202
```


JSON Output

- The REST endpoint documentation does not provide examples for how JSON output is formatted
- For JSON, print the endpoint output to a file or to the screen and examine the output to see
  - The data that is returned
  - How the data is formatted
- Use your programming language's JSON utilities to extract and parse the data

JSON Output (cont.)

- To force the endpoint to format output as something other than XML, use the **`output_mode`** request parameter
- For example, to view the output for **`/saved/searches`** in JSON

```
curl -k -X GET -s -u restclient:restC0der
http://localhost:8089/servicesNS/terry/search/saved/searches
 -d output_mode=json
```

Note

For information about the output formats available for a particular endpoint, refer to that endpoint's documentation on docs.splunk.com.

JSON Formatted Output


```
{ "links":
   { "create": "\/servicesNS\/terry\/search\/saved\/searches\/_new",
     "_reload": "\/servicesNS\/terry\/search\/saved\/searches\/_reload",
     "_acl": "\/servicesNS\/terry\/search\/saved\/searches\/_acl"
},
"origin": "http:\/\/localhost:8089\/servicesNS\/terry\/search\/saved\/searches",
"updated": "2020-11-05T23:03:24+00:00",
"generator": {
  "build": "5c84e0a2f02d",
  "version": "8.1.0" },
"entry": [
  {…},
  {…},
  {…}
  …
  ]
}
```


JSON formats the entries within a list (collection) – using square brackets []

JSON formats each **entry** as a separate JSON object. Each entry is formatted in its own set of curly braces{}

 |

`entry` Object Overview

- Note that the human-readable name for an entry object is the **`"name"`**
  - In XML formatted output, it was found in the **`<title>`** element
- Note also that **`acl`** and **`content`** are embedded as separate objects within the **`entry`**
  - In XML formatted output, `<s:key name="eai:acl">` was nested within `<content>`

```
{
   "name": "Errors in the last 24 hours",
   "id":
     "http:\/\/localhost:8089\/servicesNS\/nobody\/search\/saved\/searches\/Errors…hours",
   "updated": "1970-01-01T00:00:00+00:00",
   "links": {…},
   "author": "nobody",
   "acl": {…}
   "content": {…}
}
```

Note

The nobody user is associated with savedsearches owned by an app. The nobody user is also associated with savedsearches created by users who are no longer valid on the Splunk server.

acl Object

The information in the acl object is similar to what is found in XML

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

```
"acl": {
  "app": "search",
  "can_change_perms": true,
  "can_list": true,
  "can_share_app": true,
  "can_share_global": true,
  "can_share_user": false,
  "can_write": true,
  "modifiable": true,
  "owner": "nobody",
  "perms": { … },
  "removable": false,
  "sharing": "app"
},
```


`content` Object

- The **`content`** object is a container for content returned by the operation for each entry
- The data contained within this object varies per endpoint, reflecting properties of the object

```
"content": {
  "action.email": false,
  "action.email.sendresults": null,
  "action.email.to": "",
  "action.populate_lookup": false,
  "action.rss": false, "action.script": false,
  "action.summary_index": false,
  "actions": "",
  …
}
```


Parsing a JSON Response

- Programming languages have various techniques to iterate through JSON
- The lectures and lab exercises in this course use **Python**
- In Python, the **`json`** package contains methods for parsing JSON text
- In a *nix environment, the **`jq`** package provides the ability to parse and pretty print JSON files – for example

```
curl -k -X GET -s -u restclient:restC0der
http://localhost:8089/servicesNS/terry/search/saved/searches -d
output_mode=json | jq . > my-prettyPrintedOutput.json
```


Python Code Example

- The next example shows a Python program that uses REST API calls to:
  - Log into Splunk and get a `sessionKey`
  - Pass the `sessionKey` to the `/saved/searches` endpoint, to get information about all the saved searches in **terry/search**
  - Parse the results to print the names of all the saved searches
- In this example, the calls to the REST API specify `output_mode=json`

Python Code Example (cont.)

```
# authentication parameters
  parameters = urllib.parse.urlencode({'username': username,
                                       'password': password,
                                       'output_mode': 'json'})

# prepare the request
  req = urllib.request.Request(
         url = "https://localhost:8089/services/auth/login",
         data = parameters.encode('ascii'),
         method = 'POST')

# execute request
  f = urllib.request.urlopen(req)
  serverResponse = f.read().decode('utf-8')
  f.close()
```

```
{"sessionKey":"UHKeO6YwaQnMT8uTSaMM74^PmAUwVo9PyIiqXPFLDOpRI
ZujbgQDBHpwJKbApWKjpRkl9BjHuVZ6qAiLR6GYXl5h5G84_kYxAc5gGQJc7
N"}
```

- You can use **`urllib.request`** to authenticate with the Splunk Server REST API port
- Note that the request parameter **`output_mode`** is set to **`json`**
- The response – contained in **`serverResponse`** – is a string, formatted as **`json`**

Python Code Example (cont.)

- On a Splunk test system without a valid SSL key on the service port, you may want to disable SSL checks during development.
  - This is an optional step and only recommended on development systems.
  - On production systems the best practice is to generate an SSL certificate and get it installed to encrypt and certify connections on port 8089

```
# Creates a context disabling SSL verification
  ctx = ssl.create_default_context()
  ctx.check_hostname = False
  ctx.verify_mode = ssl.CERT_NONE

# execute request, adding the context
  f = urllib.request.urlopen(req, context=ctx)
  serverResponse = f.read().decode('utf-8')
  f.close()
```

Note

For the labs on this class we have disabled SSL on your server's port 8089, for convenience, so this code is not needed. But if you want to connect to your own Splunk instances it might be necessary to disable SSL checks during development.

Python Code Example (cont.)

- Use `json.loads()` to convert:
  - JSON arrays to Python lists
  - JSON objects to Python maps
- Thus, `serverResponse` is turned into a Python dictionary `parsed_json`
  - This makes it easy to extract the `sessionKey`

```
{"sessionKey":"UHKeO6YwaQnMT8uTSaMM74^PmAUwVo9PyIiqXPFLDOpRIZujb
gQDBHpwJKbApWKjpRkl9BjHuVZ6qAiLR6GYXl5h5G84_kYxAc5gGQJc7N"}
```

```
parsed_json = json.loads(serverResponse)
sessionKey = parsed_json['sessionKey']
```

```
UHKeO6YwaQnMT8uTSaMM74^PmAUwVo9PyIiqXPFLDOpRIZujbgQDBHpwJKbApWKj
pRkl9BjHuVZ6qAiLR6GYXl5h5G84_kYxAc5gGQJc7N"
```


Python Code Example (cont.)

- The **`sessionKey`** can then be passed in new calls to REST endpoints to access the Splunk server
- For example, to call the **`/saved/searches`** endpoint

```
…
HEADERS = {'Authorization': F'Splunk {sessionKey}'}

DATA = urllib.parse.urlencode({'output_mode': 'json'})
req = urllib.request.Request(
    url = "https://localhost:8089/servicesNS/terry/search/saved/searches",
    data = DATA.encode('ascii'), method = 'GET', headers = HEADERS)

serverResponse = urllib.request.urlopen(req).read().decode('utf-8')
```

- **`serverResponse`** now contains information about saved searches in the namespace terry/search

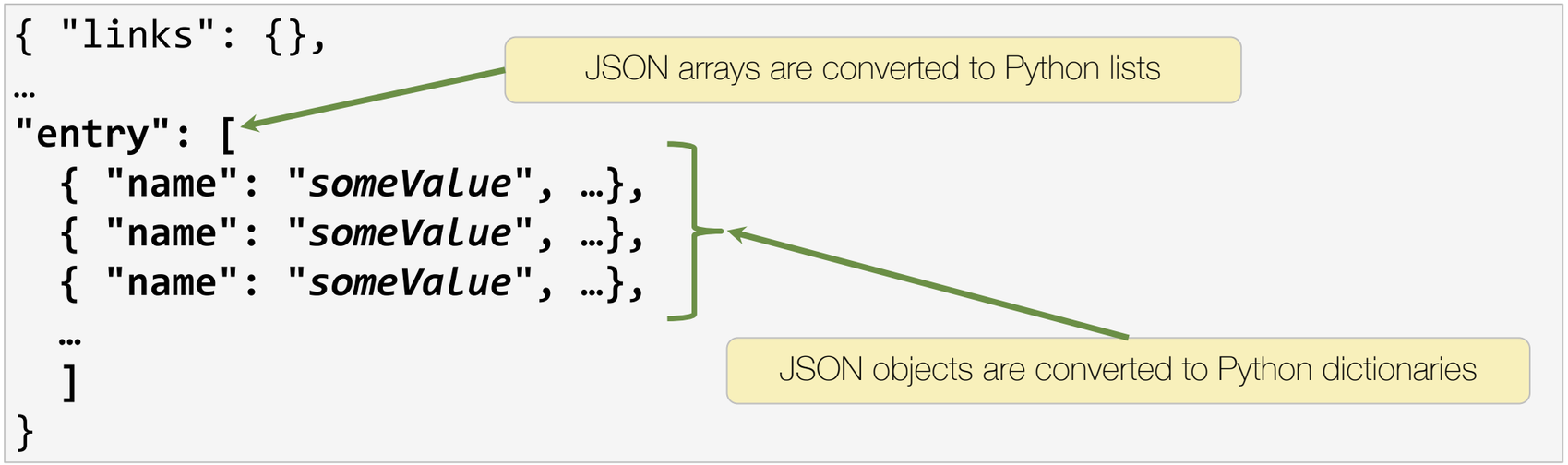
Python Code Example (cont.)

- Before writing code to format the JSON output, it is necessary to know how the output looks
  - For example, the nesting of data, arrays, and key/value pairs
- If you are not familiar with the output format, print it to screen or to a file. Tools like Postman also format the output, allowing it to be easily examined
- One way to "pretty print" it to a file

```
file = open('someFile.json','w')
myTest = json.dumps(serverResponse, indent=3)
file.write(myTest)
```


Python Code Example (cont.)

- This JSON was shown earlier to be in this format



- To parse this JSON output, it is necessary to:
  - Iterate through the Python **`entry`** object list
  - For each each dictionary in the list, get the value of **`name`**

Python Code Example (cont.)

```
{ "links": {},
…
"entry": [
  { "name": "Errors in the last 24 hours", …},
  { "name": "Errors in the last hour", …},
  { "name": "License Usage Data Cube", …},
  …
  ]
}
```

```
…
list = parsed_json['entry']
for entry in list:
  print(entry['name'])
```

```
Errors in the last 24 hours
Errors in the last hour
License Usage Data Cube
Orphaned scheduled searches
terry-search
www-search
```


Module 3 Lab Exercise

Time: 20 minutes

Tasks:

- Modify the Python code templates adding Splunk REST API calls to:
  - Authenticate with Splunk and retrieve a **`sessionKey`**
  - Produce a list of names of saved searches and their sharing level, from JSON output
  - (If time permits) Print out the permissions data for each saved search

Note

In this lab exercise – and optionally on the rest of the lab exercises in this course – you will edit provided Python programs to implement the features described here.

Module 4:
Oneshot Searches

Objectives

- Review search language syntax and search best practices
- Execute a oneshot search
- Parse search results

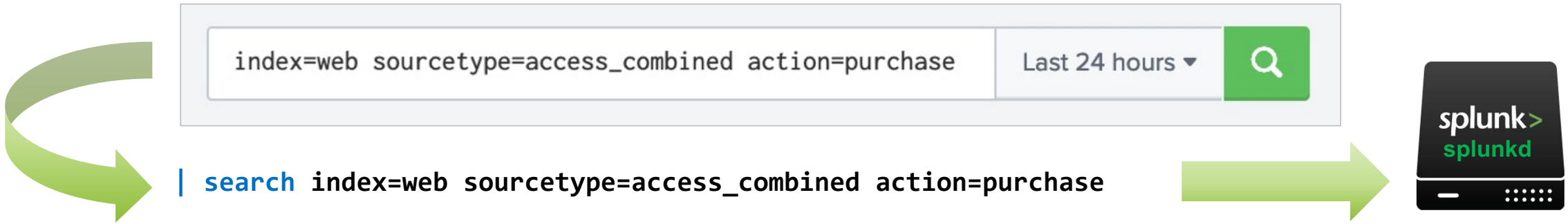
Splunk Search Language (SPL)

- Review: SPL searches always begin with a **generating command,** followed by a list of terms and boolean operators

```
|command searchCondition
```

  - Operators include **`AND`**, **`OR`**, **`NOT`**, **`()`**

- Splunk Web: if the SPL does not begin with a generating command, the **`search`** command is assumed – and is added automatically to the underlying REST request



Splunk Search Language (SPL) (cont.)

- When sending SPL to a REST endpoint, it **must** begin with a generating command

```
| search index=web sourcetype=access_combined action=purchase
```

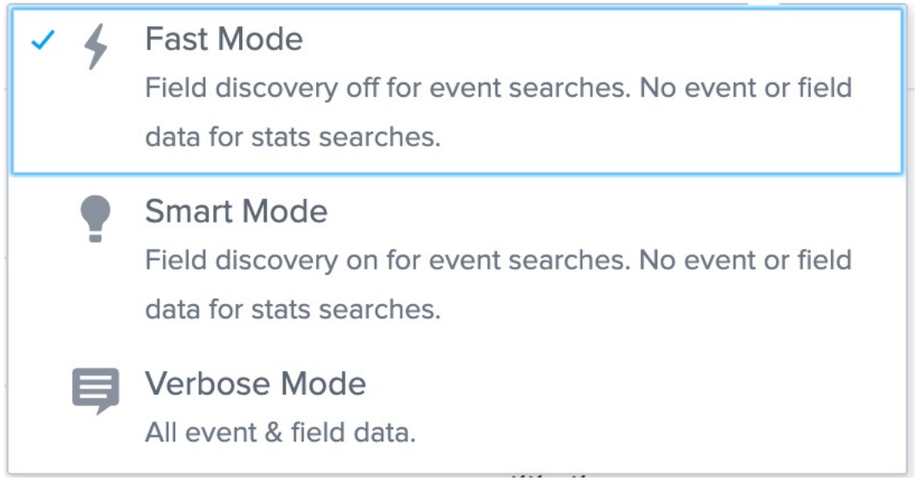
- Other considerations:
  - A pipe | at the beginning of the query is optional
  - All terms are "whole word" by default, but wildcard * is supported
  - A query can send its results to other SPL commands using a pipe |, the same way as searches on the web interface
  - Other generating commands: *inputlookup*, *loadjob*, *makeresults*, *tstats*, *datamodel*, *from*, *dbxquery* (DB Connect app)

Metadata Fields

- There are various metadata fields assigned to each event—some examples:
  - **source**: the source of the event data
  - **sourcetype**: the kind of data (ie: firewall data, mail server, etc.)
  - **host**: the host name the data originated from
  - **_raw**: the raw data collected from the source
  - **_time**: the time/datestamp from the event data
- Fields starting with underscore (_) are normally hidden when using Splunk Web but are present on the REST results like any other field

Specifying Fields

- By default, searches executed by a REST endpoint do not do full field extraction
  - Corresponds to **Fast Mode** in Splunk
- To force full field extraction, add `| fields *`
  - Corresponds to **Verbose Mode** in Splunk
- Or, use `| fields` to specify exactly which fields to output



Query Time Ranges

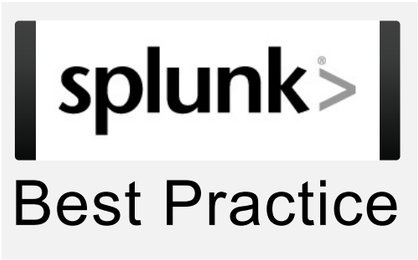
- Time range can be added as a set of parameters or directly in the query string as a pair of fields:

```
search index=… earliest=-24h latest=now |…
```

- **`earliest`** defaults to epoch (Jan. 1, 1970), **`latest`** defaults to now
- **`earliest`** and **`latest`** can accept relative or absolute time indicators
- Time range can be relative (see above), absolute, or real-time
  - Times are in the server's time zone
- For more information on timestamp formats refer to the Search Time Ranges documentation page.

Search Best Practices

- Return the smallest set of data possible
  - Set specific time range
  - Remove un-needed fields
  - Combine searches and retrieve as much data as you can on a single call
- Define saved searches and invoke them
- Run scheduled summarizing searches on the server for later use
- Define calculated fields, tags, event types, etc. as knowledge objects instead of putting definitions in the search text



Reducing Result Size

- Use the smallest time range possible
- Use the **`fields`** function to reduce the number of fields in the result:

```
search ... | fields x, y, z
```

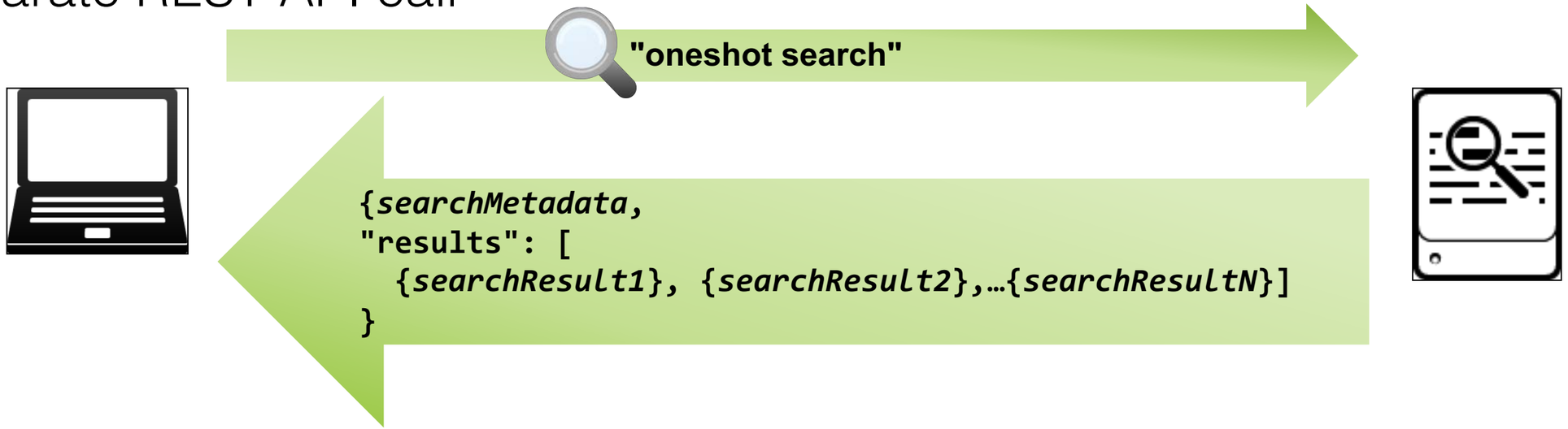
  - The **`fields`** function does NOT hide internal fields (**`_raw`**, **`_time`**, **`_sourcetype`**, etc.) unless explicitly specified.
  - Use selective functions (**`head`**, **`tail`**, **`top`**, **`rare`**, etc.) to truncate large results
- Use transforming commands (**`stats`**, **`chart`**, **`timechart`**, etc.) to summarize data

Search and Namespaces

- Namespaces affect the execution of a search and can change the search results
- Events are stored in **indexes**, which have their own role-based access control (RBAC).
- A search can include `index=`*`indexname`* to specify an index
- Other Splunk configurations, like field extractions, aliases, tags, event types, etc., are conditional based on namespace
  - Therefore, the search behaves according to the permissions and objects visible by the user/app namespace

Overview: Oneshot Search

- A "**oneshot**" search sends a search to the search head, **synchronously**
- The search head sends one response, which includes all the search metadata and search results
  - When the response is returned, the search is complete
  - There is no need to check the search status for completion using a separate REST API call



Overview: Oneshot Search (cont.)

- Send a `POST` request to the `search/jobs` endpoint
- Required parameters:
  - `search`: set equal to a string, formatted per the rules of Splunk SPL
  - `exec_mode`: set to `oneshot`
- Optional parameters:
  - `output_mode`: set to an output format: `xml` (default), `json`, `csv` or `raw`
  - `earliest_time`: beginning of time range, equivalent to `earliest`
  - `latest_time`: end of time range, equivalent to `latest`

Executing a Oneshot Search

This search returns the last three purchases from the Buttercup Games online store

```
$curl -k -s -u restclient:restC0der
http://localhost:8089/services/search/jobs -d search="search index=web
earliest=-4h@h latest=now sourcetype=access_combined action=purchase |
fields price, product_name | head 3" -d exec_mode=oneshot -d
output_mode=json > oneshotOutput.json
```

In Unix/Linux the shell I/O redirection operator “>” redirects the output to another place. In the command above, instead of showing the output on the terminal it was redirected to be saved in the file oneshotOutput.json

For more info: https://www.tecmint.com/linux-io-input-output-redirection-operators/

Oneshot Output: `fields`

oneshotOutput.json

```
{  "preview": false,
   "init_offset": 0,
   "messages": [],
   "fields": [
    {      "name": "price"    },
    {      "name": "product_name"    },
    {      "name": "_bkt"    },
    {      "name": "_cd"    },
    {      "name": "_indextime"    },
    {      "name": "_kv"    },
    {      "name": "_raw"    },
    {      "name": "_serial"    },
    {      "name": "_si"    },
    {      "name": "_sourcetype"    }
   ],
   "results": [{…},{…},{…}],
   "highlighted": {}}
```

- The endpoint returns **`fields`**
- **`fields`** contains an array of objects
  - Each object contains a **`name`** key
  - The value of the **`name`** key is the name of field returned by Splunk
- The **`results`** object contains an array of search results

Note

Internal fields can be removed either with the **`fields`** command (e.g., **`fields -_*`**) or using a script or program.

Oneshot Output: `results`

```
{  …,
   "results": [
      { "price": "5.99",
        "product_name", "Holy Blade of Gouda",
        "_bkt": "web~27~A512281F-8AE8-4DB2-8247-5CB0805E3B51",
        "_cd": "27:4508",
        "_indextime": "1519237684",
        "_kv": "1",
        "_raw": "194.215.205.19 - - [21/Oct/2020:18:28:04] …",
        "_serial": "0",
        "_si": [
            "ip-10-0-81-92",
            "web"      ],
        "_sourcetype": "access_combined",
        "_time": "2020-10-21T18:28:04.000+00:00",
},
      {…},  ],
"…}
```

- In the `results` array, each search result is returned as a JSON object
- Each key name in `results` is:
  - A field returned by Splunk
  - Followed by the value that the search returned for that field

Oneshot Search: Time Intervals

- Time range can be embedded in the search string *or* as part of request arguments:
  - **`earliest_time`**: set to search start time
  - **`latest_time`**: set to search end time

```
$curl -k -s -u username:password http://host:port/services/search/jobs
-d search="search mySearchHere" -d earliest_time=-24h@h -d latest_time=now
-d exec_mode=oneshot -d output_mode=json
```

- If the time range is embedded in the search, the **`earliest_time`** and **`latest_time`** arguments are ignored

Oneshot Output: Additional Output Modes

- When search results are returned, the **/search/jobs** endpoint allows two additional output modes: **csv** and **raw**
- Output mode **raw** only returns results when your search returns raw data
- Output mode csv works for any search returning fields

```
$curl -k -s -u restclient:restC0der
http://localhost:8089/services/search/jobs -d search="search index=web
sourcetype=access_combined | stats count, sum(price) as price by
host" -d exec_mode=oneshot -d output_mode=csv
```

```
host,count,price
www1,337,"4747.77"
www2,239,"2960.44"
www3,312,"4087.99"
```

Note

The output modes **json**, **csv** and **raw** also work with the normal and export searches discussed in the next module

Processing Results

- First, send the `POST` request to the `search/jobs` endpoint, passing these arguments
  - A `search`: The search string
  - B `output_mode`: `json` (or the desired format)
  - C `exec_mode`: `oneshot`

```
…
SEARCHSTR = 'search index=web sourcetype=access_combined action=purchase product_name=* price=* | fields
product_name, price | head 2'
HEADERS = {'Authorization': F'Splunk {sessionKey}'}
DATA = urllib.parse.urlencode({'search': SEARCHSTR, 'output_mode': 'json', 'exec_mode': 'oneshot'})
                                                   A                    B                    C

req = urllib.request.Request(url = "https://localhost:8089/services/search/jobs",
      data = DATA.encode('ascii'), method = 'POST', headers = HEADERS)

serverResponse = urllib.request.urlopen(req).read().decode('utf-8')
```

Processing Results (cont.)

- D Get the `list` of results
- E Iterate through the `list`
- F Retrieve each `field` and `value`
- G If the field does not match a string beginning with _
  - Print the `field` and the associated `value`

```
…
parsed_json = json.loads(serverResponse)
entry_list = parsed_json['results']

 for result in entry_list:
    print("Result: ")
    for field, value in result.items():
       if not field.startswith('_'):
          print("%5s: %s" % (field, value))
```

```
Result:
      price :  4.99
      product_name :  Puppies vs. Zombies
Result:
      price :  39.99
      product_name :  Dream Crusher
```

> **Note**
>
> By removing fields whose name begin with an underscore we are removing internal fields, such as _time, _raw, etc…

Controlling Search Execution Time

- A **oneshot** search runs for as long as it takes to complete and the caller is blocked, waiting for the results. Therefore, it is recommended for quick searches, returning small and predictable data sets
- This is mostly driven by the time span of data the search retrieves from the indexers
  - Time range is the biggest factor
  - Total amount of data in the time range also matters

Module 4 Lab Exercise

Time: 20 minutes

Tasks (using `curl` or `Postman`):

- Execute a oneshot search and produce results in JSON format
- Execute a oneshot search and produce results in CSV and in RAW formats
  - Use a request parameter to return only results from yesterday
- **OPTIONAL**: Edit a Python program to execute a oneshot search
  - Return the results in JSON format
  - For each search result, print the *field* : *value*

Module 5:
Normal and Export Searches

Objectives

- Identify types of searches
- Create normal and export searches
- Get:
  - Search results
  - Search job status and other search job properties

Blocking Searches

- Blocking searches execute **synchronously**
  - Like the **oneshot** search in the previous module
  - Designed for searches that won't take long to execute and return a small, controlled number of results.
- They are called "blocking" because the caller waits for the result set to be fully processed and the response is delivered as part of the original call
- When results are complete:
  - A search job is created
  - Results are sent back to caller as response from original call

Normal Searches

- Normal searches execute **asynchronously**
- Normal searches cause a search job to be created immediately
  - You can poll the search job to check execution status
  - When search has completed, you can retrieve results
    - If desired, retrieve intermediate results prior to search completion (preview)
- Good for searches that take longer to execute or are unpredictable

Executing a Normal Search

- As with a oneshot search, a **normal** search is called by sending a `POST` request to the `/search/jobs` endpoint
- However, the `exec_mode` argument is set to `normal` (default)
- In normal mode, the endpoint:
  - Creates a Splunk search job
  - Begins the search
  - Returns the job identifier, as the value of `sid`

```
$curl -X POST -k -s -u username:password
http://host:port/services/search/jobs -d search="search mySearchHere" -d
output_mode=json -d exec_mode=normal

{"sid":"1519670895.34"}
```


Getting the Search Status

- To obtain the status of a search job, send a **`GET`** request to the **`search/jobs/`*`sid`*** endpoint
  - Append the **`sid`** returned by Splunk to the **`/search/jobs`** endpoint to get information for a specific job, as shown below

```
$curl –X GET -k -s -u username:password
http://host:port/services/search/jobs/1519670895.34 -d output_mode=json
> searchOutput.json
```


Getting the Search Status (cont.)

- In the results, the search status is returned as the `dispatchState` key
  - This is found in the `entry` array, within the `content` object
- Values include `QUEUED`, `PARSING`, `RUNNING`, `PAUSED`, `FINALIZING`, `FAILED`, `DONE`
- If the `dispatchState` is `DONE`, then the `isDone` key is set to `true`

```
{ "links": {}
  …
"entry":[
 {…},
  …
 "content" : {
  …
  "dispatchState": "DONE",
  …
  "isDone": true
…
}
```


Some Data Available from a Search

Key Name	Description
`diskUsage`	Size in bytes this job is using for persistent storage
`doneProgress`	A number between 0 and 1.0 that indicates the approximate progress of the search
`eventCount`	The number of events returned by a generating command (i.e., the base search)
`eventSearch`	The portion of a search that finds event data (i.e., the base search)
`isDone`	If the search is complete, equals true
`isFailed`	If the search failed, equals true (such as for a syntax error)
`reportSearch`	Empty (None) for event search; contains report commands for a transforming search
`resultCount`	Number of events in the total result. If the search does not include transforming commands, it is equal to the event count. Otherwise, it is equal to the number of results produced by the transforming command.
`runDuration`	Seconds the search took to execute
`ttl`	Time in seconds until this job is deleted
`sid`	Job ID

Note

For other values available from a search, refer to the "Returned values" section of the `/search/jobs` endpoint documentation.

Getting Search Results

- When **`isDone = true`**, the search is complete
  - Use the **`search/jobs/`*`sid`*`/results`** endpoint to get the search results

```
$curl –X GET -k -s -u username:password
http://host:port/services/search/jobs/1519670895.34/results
-d output_mode=json  > searchOutput.json
```

  - In this example, suppose that the search was:

```
search index=* sourcetype=access_combined price=*
product_name=* latest=@d
| fields _time, price, product_name
| head 10
| table _time, product_name, price
```


Getting Search Results (cont.)

```
{
  "preview": false,
  "init_offset": 0,
   "messages": [],
   "fields": [
    {
      "name": "_time"
    },
    {
      "name": "product_name"
    },
    {
      "name": "price"
    }
  ],
  "results":[…]
}
```

Fields returned by the search in the **fields** array

Search results can be found in the **results** array

```
search index=*
sourcetype=access_combined price=*
product_name=* latest=@d
| fields _time, price, product_name
| head 10
| table _time, product_name, price
```


Getting Search Results (cont.)

```
{…
  "results": [
    {
      "_time": "2020-10-26T23:55:20.000+00:00",
      "product_name": "Manganiello Bros.",
      "price": "39.99"
    },
    {
      "_time": "2020-10-26T23:55:16.000+00:00",
      "product_name": "Manganiello Bros.",
      "price": "39.99"
    },
    …
 ],
…
}
```

Each search result is an object in the **`fields`** array

Getting Search Results (cont.)

- Suppose that the search was:

```
search index=*
sourcetype=access_combined price=*
product_name=* latest=@d
| fields _time, price, product_name
| head 10
| stats sum(price) as total_price
```

- The format of the results change to display the fields returned by the transforming command

```
{
  "preview": false,
  "init_offset": 0,
   "messages": [],
   "fields": [
    {
      "name": "total_price"
    },

  "results":[
     {
      "total_price": "324.90"
     }
   ],
  "highlighted":{}
}
```


Parsing: Check for Search Completion

- If the value for **`isDone`** on the **`content`** array is not **`true`**, then wait a second and check again.

```
def get_search_status(sessionKey, sid):
  HEADERS = {'Authorization': F'Splunk {SessionKey}'}
  DATA = urllib.parse.urlencode({'output_mode': 'json'})

  req = urllib.request.Request(url = "https://localhost:8089/services/search/jobs/{sid},
        data = DATA.encode('ascii'), method = 'GET', headers = HEADERS)
  waitForResult = False # search not yet ready

  while not waitForResult:
    with urllib.request.urlopen(req) as f:
      response = f.read().decode('utf-8')
      parsed_json = json.loads(response)
    waitForResult = parsed_json['entry'][0]['content']['isDone'] # when true, then search is ready.
    time.sleep(1)

  return parsed_json
```


Processing Results

A. Use the `/search/jobs/sid/results` to get the search results

A.
```
URL = "https://localhost:8089/services/search/jobs/{sid}/results"
HEADERS = {'Authorization': F'Splunk {sessionKey}'}
DATA = urllib.parse.urlencode({'output_mode': 'json'})

req = urllib.request.Request(url=URL, data=DATA.encode('ascii'),
      method = 'GET',headers = HEADERS)
results = None

with urllib.request.urlopen(req) as f:
   response = f.read().decode('utf-8')
B  results = json.loads(response)['results']

C for entry in results:
   print("Result")
   print("     Time:         " + entry['_time'])
   print("     Category:     " + entry['categoryId'])
   print("     Product Name: " + entry['product_name'])
```

B. The results are returned in the `results` array

C. Iterate through the `results` array to obtain each result

Export Search

- An export search returns a **stream handle** to the results
  - Results on export searches are returned as they are processed. Unlike **oneshot**, there's no need to wait for whole processing to finish as results begin to flow as soon as the first line gets ready on the search head
- Fast execution
  - One trip to server, similar to oneshot
- Best way to extract data as-is for backup or transfer to another system

Executing an Export Search

To execute an export search, use the **search/jobs/export** endpoint

```
searchStr = 'search index=* sourcetype=access_combined action=purchase status<400 \
product_name=* | fields - _* | fields product_name, categoryId | head 5'

CMDURL = '/services/search/jobs/export'
HEADERS = {'Authorization': F'Splunk {sessionKey}'}
DATA = urllib.parse.urlencode({'search': searchStr})

req = urllib.request.Request(url = baseurl + CMDURL, data = DATA.encode('ascii'),
      method = 'POST', headers = HEADERS)

with urllib.request.urlopen(req) as f:
   response = f.read().decode('utf-8')
   json_object = json.loads(response)
```


Results of an Export Search

```
{"preview":false,"offset":0,"result":
   {"product_name":"Fire Resistance Suit of Provolone",  "categoryId":"ACCESSORIES"}}
{"preview":false,"offset":1,"result":
   {"product_name":"Fire Resistance Suit of Provolone","categoryId":"ACCESSORIES"}}
{"preview":false,"offset":2,"result":{"product_name":"Orvil the Wolverine","categoryId":"ARCADE"}}
{"preview":false,"offset":3,"result":{"product_name":"Final Sequel","categoryId":"STRATEGY"}}
{"preview":false,"offset":4,"lastrow":true,"result":{"product_name":"Holy Blade of
    Gouda","categoryId":"ACCESSORIES"}}
```

- To support the streaming nature of the endpoint, Splunk presents the data in a format that is easy to parse in a continuous manner, result by result
  - Each search result row is as an individual, complete JSON object
  - HOWEVER, the response as a whole is NOT a valid JSON object.

Results of an Export Search (cont.)

- JSON parsing utilities:
  - If you pass individual rows as the results, a parsing tool can parse the output content line by line.
  - If you try to parse the entire result set at once, the parsing tool won't be able to parse the content and will cause errors. The full result set is not a valid JSON document, although the individual lines are.
  - If you develop a parser to interpret results, be sure to parse each line of the result individually.

Another Approach

In addition to **`output_mode=json`**, export searches have additional modes. Use **`json_rows`** for a row-by-row export, on a single valid json document

```
searchStr = 'search index=* sourcetype=access_combined action=purchase status<400 \
product_name=* | fields - _* | fields product_name, categoryId | head 5'

CMDURL = '/services/search/jobs/export'
HEADERS = {'Authorization': F'Splunk {sessionKey}'}
DATA = urllib.parse.urlencode({'search': searchStr}, 'output_mode': 'json_rows'})

req = urllib.request.Request(url = baseurl + CMDURL, data = DATA.encode('ascii'),
      method = 'POST', headers = HEADERS)

with urllib.request.urlopen(req) as f:
   response = f.read().decode('utf-8')
   json_object = json.loads(response)
```


Another Approach (cont.)

JSON parsers are able to process the entire result correctly as one single JSON document when using **`output_mode=json_rows`**

```
{
  …
  "fields": [
     "product_name",
     "categoryId"
   ],
   "rows": [
      [ "Orvil the Wolverine", "ARCADE" ],
      [ "Puppies vs. Zombies", "STRATEGY" ],
      [ "Orvil the Wolverine", "ARCADE" ],
      [ "Dream Crusher", "STRATEGY" ],
      [ "Manganiello Bros.", "ARCADE" ]
    ]
}
```


Yet Another Approach

- Here is the same data, but this time using `output_mode = json_cols`:
- This result set can also be parsed as a whole, single JSON object.
- Both `json_rows` and `json_cols` modes also work with **normal** and **oneshot** searches

> **Note**
>
> Besides json and json_cols, another valid format is **xml**.

```
{
  …
  "fields": [
     "product_name",
     "categoryId"
   ],
   "columns": [
      [ "Orvil the Wolverine" ],
      [ "Puppies vs. Zombies" ],
      [ "Orvil the Wolverine" ],
      [ "Dream Crusher" ],
      [ "Manganiello Bros." ]
    ,
    [
      [ "ARCADE" ],
      [ "STRATEGY" ],
      [ "ARCADE" ],
      [ "STRATEGY" ],
      [ "ARCADE" ]
    ]
}
```


Module 5 Lab Exercise

Time: 20 minutes

Tasks (using `curl` or `Postman`):

- Create a normal search, get its status, and get its results in JSON format
- Create an export search and get its results in `json_rows` format and experiment with other formats
- OPTIONAL: Edit a Python program to execute a normal search and print its results
- OPTIONAL: Edit a Python program to execute an export search and print its results

Module 6:
Advanced Searching and Job Management

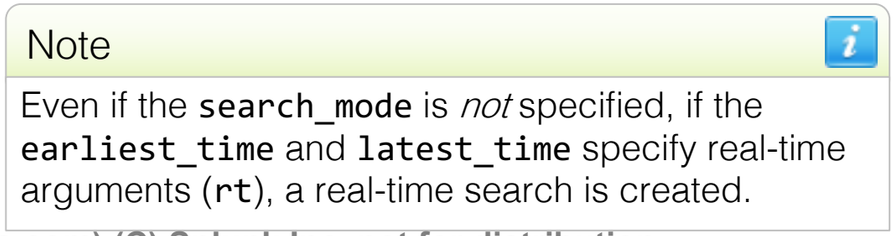
Objectives

- Execute a real-time search
- Work with large results sets
- Work with saved searches
- Manage search jobs

Real-time Searches

- Begin monitoring for matches to the search query you submit from now into the future, like a **`tail`** command
- Execute using the same REST endpoints as other searches – for example: **`search/jobs`**, **`search/jobs/export`**
  - Set the **`search_mode`** to **`realtime`**
  - Set the **`earliest_time`** and **`latest_time`** to **`rt`** or variations

```
$curl –X POST -k -s -u username:password
http://host:port/services/search/jobs -d search="search mySearchHere"
-d search_mode=realtime –d earliest_time=rt –d latest_time=rt …
```



Real-time Searches (cont.)

- `auto_cancel`: the number of seconds of *inactivity* a job can have before being automatically terminated
  - "Inactivity": the time during which job status or events are not retrieved
  - Thus, a search could still be **finding** search results and yet be considered inactive
  - The default is 0, which is unlimited
- This is a very useful option for real-time searches

```
$curl -X POST -k -s -u username:password http://host:port/services/search/jobs
-d search="search mySearchHere" -d search_mode=realtime -d earliest_time=rt
-d latest_time=rt -d auto_cancel=1200 -d output_mode=json

{"sid":"rt_1519670895.34"}
```

The `sid` value for real-time searches are prefixed with `rt_`

Using a Sliding Window

- For `earliest_time` and/or `latest_time`, specify a window of time, using SPL time modifiers for realtime
  - Preface the SPL time modifier with `rt-`
- For example, to run a search in real-time over the previous ten minutes, snapped to the minute

```
$curl –X POST -k -s -u username:password
http://host:port/services/search/jobs -d search="search mySearchHere"
-d search_mode=realtime –d earliest_time=rt-10m@m –d latest_time=rt -d
output_mode=json –d auto_cancel=1200
```

Search Status and Real-Time Searches

```
{ …    "
  "entry": [
   { …
    "content": {
     …
     "numPreviews: : 20
     "resultCount" : 0,
     …
    }
 ]
}
```

- To get the search status, use the **`search/jobs/`*`sid`*** endpoint
- Until a real-time search is stopped, the **`resultCount`** is 0
- Real-time searches enable previews of the results, as a snapshot.
  - Previews are results that can be viewed while the search is still executing
  - **`numPreviews`** indicates the number of results than can be viewed at this time

Getting Search Preview Results

- To get search previews, use the `search/jobs/sid/results_preview` endpoint
- The arguments are the same as for `/search/jobs/sid/results`

```
$curl –X GET -k -s -u username:password
http://host:port/services/search/jobs/rt_1519670895.34/results_preview
-d output_mode=json  > searchOutput.json
```

The regular endpoint "/results" can only retrieve results when a search is finalized. However real-time searches are constantly running, and this endpoint can only retrieve data when the search is interrupted.

The "/results_preview" endpoint can retrieve partial result sets for any search while the job is executing. It is particularly useful on real-time searches as it can retrieve the current result set for the search and also can be used on normal searches.

Getting Search Preview Results (cont.)

The preview results are formatted the same way as "regular" results – as objects in a **`results`** array

 | 31 August 2021

Real-Time Searching is Expensive

- Real-time searches consume more system resources than other searches
  - Use them carefully!
  - Set `auto_cancel` to prevent "idle" real-time searches from consuming valuable system resources

- The Splunk Administrator can:
  - Limit the the number of real-time searches that can execute
  - Disable real-time searches for specific users and roles
  - Disable real-time searches for specific Splunk indexes

Processing Many Results

- While a search is executing, its progress can be checked:
  - The number of preview results is shown as the **`numPreviews`**
  - The number of events is shown as the **`eventCount`**
- When a search has **finalized** and **completed**, you can check the number of results returned as the **`resultCount`**
- By default, all search results are retrieved at once

```
{ …    "
  "entry": [
   { …
    "content": {
     …
     "resultCount" : 250,
     …
    }
 ]
}
```


Processing Many Results (cont.)

- To retrieve a subset of results, pass these request parameters when sending requests to **…/`results`** or **…/`results_preview`**
  - **`offset`**:
    - Retrieve results starting from this point in the **`results`** array
    - The **`results`** array begins at 0 (which is the default value)
  - **`count`**: the number of results to retrieve, starting from the **`offset`**
    - The default is 100

Note

`offset` and **`count`** must be sent URL encoded.

Examples

- Retrieve first 50 results – that is, results 0 through 49

```
$curl -G -k -s -u username:password
http://host:port/services/search/jobs/rt_1519670895.34/results_preview
-d output_mode=json -d offset=0 -d count=50  > searchOutput.json
```

- Retrieve another 50 results –that is, results 50 through 99

```
$curl -G -k -s -u username:password
http://host:port/services/search/jobs/rt_1519670895.34/results_preview
-d output_mode=json -d offset=50 -d count=50 > searchOutput.json
```

Note

`-G` causes the request arguments to be sent URL encoded like a GET call.

Managing Search Jobs

- A search job persists for 10 minutes after the search completes execution (known as TTL – Time to Live)
  - Administrators can change period of time in `limits.conf`
  - Saved searches can have any retention, defined in the search
- Users can save a job
  - The job remains on the server for 7 days
- While a search job persists, there is access to its results
  - The results can be obtained without re-executing the search
- While a search job exists, it can be managed by sending a `POST` request to the `search/jobs/`*`sid`*`/control` endpoint

Managing Search Jobs (cont.)

- Use the `action` parameter to control the search job
  - `pause`: Suspends the execution of the current search
  - `unpause`: Resumes the execution of the current search, if paused
  - `finalize`: Stops the search and provides intermediate results to the `/results` endpoint
  - `cancel`: Stops the current search and deletes the result cache
  - `save`: Saves the search job, storing search artifacts on disk for 7 days
  - `setttl`: Set the time to live ("ttl") value for the job, in seconds
  - `touch`: Extend the job expiration time to now + the job's defined ttl

Note

The default time to live (ttl) value for jobs is set by a Splunk administrator in `limits.conf`.

Examples

- Pause a search job

```
$curl –X POST -k -s -u username:password
http://host:port/services/search/jobs/1519670895.34/control -d
output_mode=json -d action=pause

{"messages":[{"type": "INFO", "text": "Search job paused."}]}
```

- Set the ttl (time to live) for the job to be five hours

```
$curl –X POST -k -s -u username:password
http://host:port/services/search/jobs/1519670895.34/control -d
output_mode=json -d action=setttl -d ttl=18000

{"messages":[ {"type": "INFO", "text":" job's ttl was changed to 18000."}]}
```


Examples (cont.)

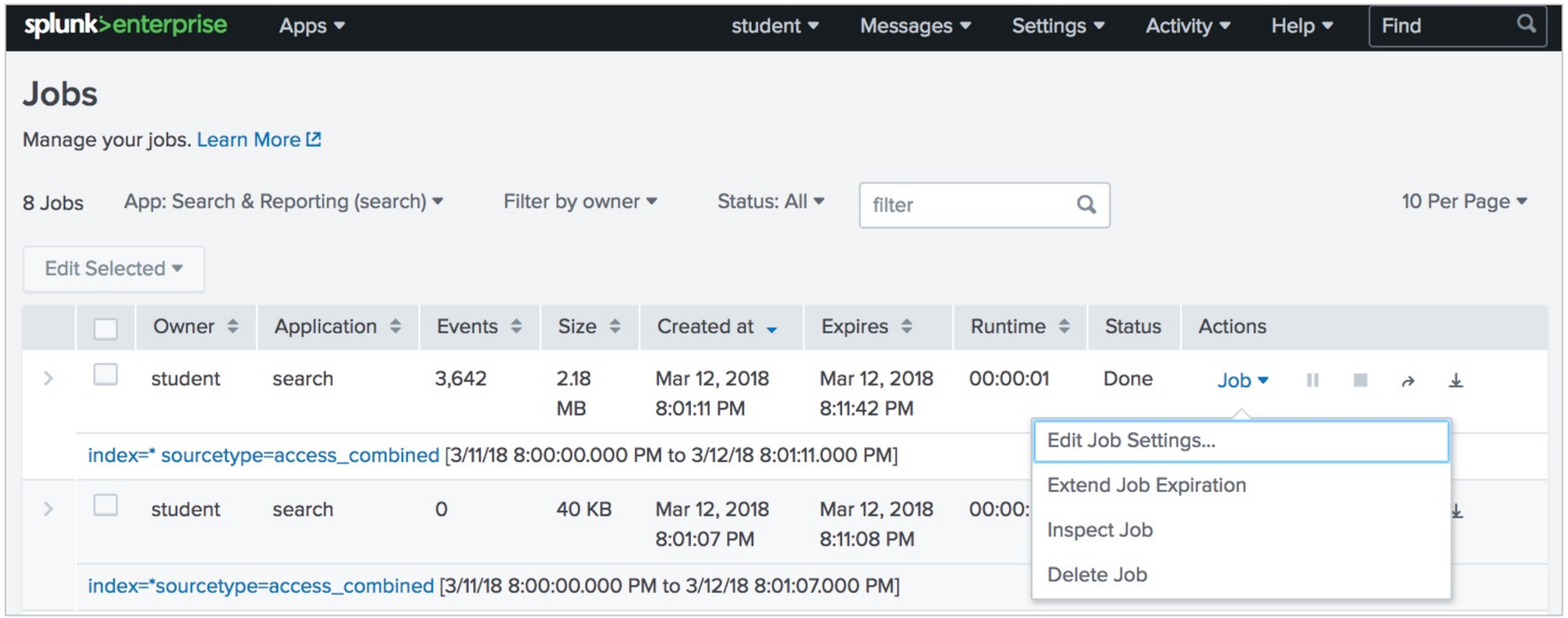
- Cancel (i.e., delete) a search job

```
$curl –X POST -k -s -u username:password
http://host:port/services/search/jobs/1519670895.34/control -d
output_mode=json -d action=cancel

{ "messages": [  { "type": "INFO","text": "Search job cancelled."}    ]}
```

- Remember: to get the search status, you can use the `search/jobs/sid/results` or `/results_preview` endpoint
  - The search status can be found in `dispatchState`
  - Values: `QUEUED`, `PARSING`, `RUNNING`, `PAUSED`, `FINALIZING`, `FAILED`, `DONE`

Controlling Jobs using Splunk Web

Recall that jobs can also be controlled using Splunk Web (**Activity > Jobs**)



Get a List of Search Jobs

To get a list of search jobs, send a `GET` request to the `search/jobs` endpoint

```
$curl –X GET -k -s -u username:password
http://host:port/services/search/jobs -d output_mode=json
> searchOutput.json
```


Get a List of Search Jobs (cont.)

- Each search job is represented as an object in the **`results`** array
- Within the search job object, most of the information about the search is found in its nested **`content`** object

```
{…
  "entry": [
   {
     "name": "search index=main …."
     "id": "http://52.24.159.106:8089/services/search/jobs/1520650860.384"
      …
     "content": {
       "canSummarize": false,
       …
       "diskUsage": 61632,
       "dispatchState": "DONE",
       …
       "isDone": true,
       …
      },
    {…}
    {…}
   …
   }
}
```

Note

For more information about returned values, refer to: http://docs.splunk.com/Documentation/Splunk/latest/RESTREF/RESTsearch#search.2Fjobs

Creating a Saved Search

- A search can be saved on the Splunk server – to be executed later or on a schedule. Also called a **report**
- To create a saved search, use the `/saved/searches` endpoint
- See the docs for the arguments you can set on a saved search:
  - Reports have MANY properties, including execution mode, scheduling, formatting, visualization, actions and ACL properties
  - Generally, you want to set these request parameters:
    - `name`: a value to identify the saved search
    - `description`: a brief description of what the search does
    - `search`: the search string to execute (see next slide…)

Creating a Saved Search (cont.)

- For a saved search, if the generating command is `search` there's no need to include it in the query text. This is the normal way to write a search, as you would do in the web interface:

```
index=main sourcetype=access_combined action=purchase
```

- For a comprehensive list of all properties, you can set on reports/saved searches, refer to the documentation.
- If you need to specify the generating command, then the pipe | is required

```
| search index=main sourcetype=access_combined action=purchase
| inputlookup products.csv | stats sum(price) by product_name
```

Example

```
$curl –X POST -k -s -u student:restC0der
http://host:port/services/saved/searches -d name= mySavedSearch
-d search=mysearch -d output_mode=json -d description="Something
Meaningful"
```

or to specify a namespace, such as `student/search`:

```
$curl –X POST -k -s -u restclient:restC0der
http://host:port/servicesNS/student/search/saved/searches -d
name=mySavedSearch -d search=mysearch -d output_mode=json -d
description="Something Meaningful"
```

By specifying a namespace you are effectively saving the search in the namespace app and defining the owner as the user in the namespace.

This is also valid for ANY other object created with the REST API as well.

Example (cont.)

- Upon successful execution, the endpoint returns an object with the full body of the saved search/report
- The body of a successful request includes an **`entry`** array with all the properties and values of the new saved search

```
{…
  "entry": [
    {
      "name": "mysearch",
      "id":
"http://host:port/servicesNS/student/
search/saved/searches/ElReporto",
      "updated": "timestamp",
      "links": {…},
      "author": {…},
      "acl": {…},
      "fields": {…},
      "content": {…}
     }
…
}
```


Executing a Saved Search

- To execute the search, use the `saved/searches/name/dispatch` endpoint, for example:

```
$curl -k -X POST -s -u restclient:restC0der
http://localhost:port/services/saved/searches/mysearch/dispatch

{"sid":"restclient__restclient__search__mysearch_at_1520629852_56"}
```

A B

- Note that the `sid` value for a saved search is different than a normal job, and includes:
  - A The username of the user executing the search
  - B The namespace information (in this example, restclient/search)

Getting the Results

- Get the results of a saved search using the `search/jobs/sid/results` endpoint
- For example:

```
curl -X GET -k -s -u restclient:restC0der
http://localhost:8089/services/search/jobs/sid/results?output_mode=json
```

Since this is a GET request, parameters must be passed as part of the URL, after the question mark and they also should be url-encoded

Getting the Results (cont.)

Results from a saved search are presented in the same way as other kinds of searches – in a **`results`** list

```
{…
   "results":[
     {"product_name":"Puppies vs. Zombies","clientip":"198.35.3.23"},
     {"product_name":"Orvil the Wolverine","clientip":"198.35.3.23"},
     …
  ],
   "highlighted":{}
}
```

```
…
cmdurl = '/services/search/jobs/' + sid +
 '/results'  serverResponse =
  myhttp.request(baseurl + cmdurl, 'GET',
  headers={'Authorization': 'Splunk %s' %
  sessionKey},body={'output_mode':'json'})[1]
parsed_json = json.loads(serverResponse)
list = parsed_json['results']
  for entry in list:
    print("Result")
    print("   Product Name: " + entry['product_name'])
    print("   Client IP:    " + entry['clientip'])
```


Saved Searches and Variables

- Saved searches can include variables and execute based on the values passed in them
- Preface each variable with **`args.`**
  - For example, this search provides data for purchases made on the Buttercup Games website, for a specified **`categoryId`**

```
index=web sourcetype=access_combined action=purchase categoryId=$args.cat$
```

- When dispatching the saved search, pass the value of the variable as a request parameter

```
$curl –X POST -k -s -u restclient:restC0der
http://host:port/services/saved/searches/mysearch/dispatch -d args.cat="ACCESSORIES"

{"sid":"restclient__student__search__mysearch_at_1520629852_56"}
```


Module 6 Lab Exercise

Time: 30 minutes

Tasks (using `curl` or `Postman`):

- Create a real-time search over a sliding 8-hour window and perform several job-control activities.
- Create and execute a saved search
- **OPTIONAL:** Execute a Python program to execute a search and page through results

Module 7:
Working with KV Stores

Module Objectives

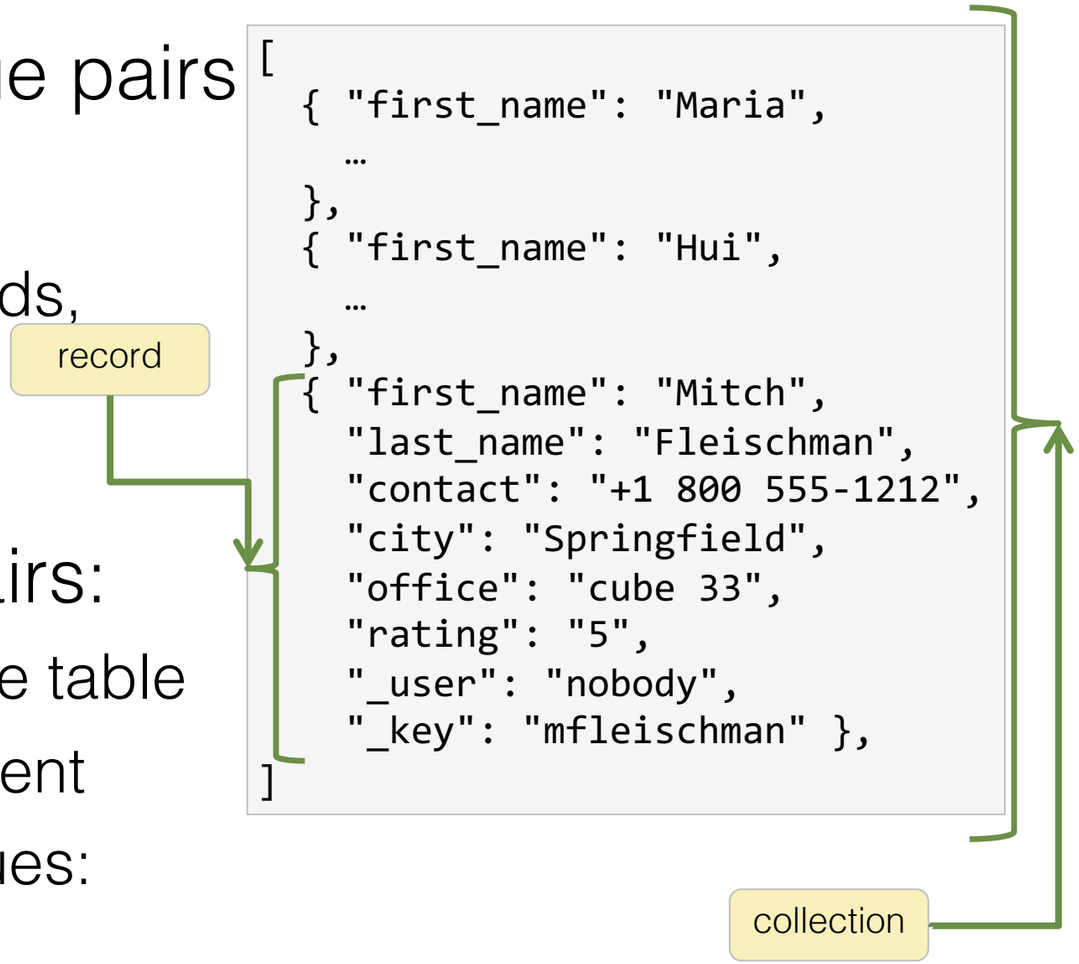
- Define the function of a KV Store
- Define collections and records
- Perform CRUD operations on collections and records

What is the KV Store?

- The App Key Value Store (KV Store) provides a way to save and retrieve data within Splunk apps
- A KV Store can be used to:
  - Store user metadata
  - Manage a UI session by storing the user or application state, as the user interacts with the app
  - Cache results from search queries by Splunk or an external data store
  - Store any data you want, for any purpose

How Does the KV Store Work?

- Stores data in collections, as key-value pairs
- Collections:
  - Are containers for data in the form of records, similar to rows in a database table
  - Exist within the context of an app
- Records contain a set of key/value pairs:
  - Keys (fields) are like columns in a database table
  - Contain the data values as a JSON document
  - Optionally, enforce data types for field values: **`boolean`**, **`string`**, **`number`**, **`object`**, **`array`**, and **`null`**



How Does the KV Store Work? (cont.)

- `_key`:
  - A reserved field containing unique ID for each record
  - If not specified, it is automatically generated
  - To modify a record, its key ID must be provided
- `_user`:
  - A reserved field with the user ID for each record
  - Cannot be overridden
- Accelerations:
  - Make searches with accelerated fields return faster (database indexed field)
  - Store a portion of the collection's data set in an easy-to-traverse form

```
[
  { "first_name": "Maria",
    …
  },
  { "first_name": "Hui",
    …
  },
  { "first_name": "Mitch",
    "last_name": "Fleischman",
    "contact": "+1 800 555-1212",
    "city": "Springfield",
    "office": "cube 33",
    "rating": "5",
    "_user": "nobody",
    "_key": "mfleischman" },
]
```


KV Store vs. CSV Lookups

	Pros	Cons	Comments
KV Store	• Enables per-record insert/updates • Allows optional data type enforcement on write operations • Allows you to define field accelerations to improve search performance • Provides REST API access to the data collection	–	• Designed for large collections • Is the easiest way to develop an application that uses key/value data • Good solution when data requires user interaction with the REST interface and when you have a frequently-changing data set
CSV	• Performs well for files that are small or rarely modified • CSV files are easier to modify manually • Supports case-sensitive field lookups	• Does not provide multi-user access locking • Requires a full rewrite of a file for edit operations	• Good solution when the data set is small or changes infrequently, and when distributed search is required

Creating a Collection

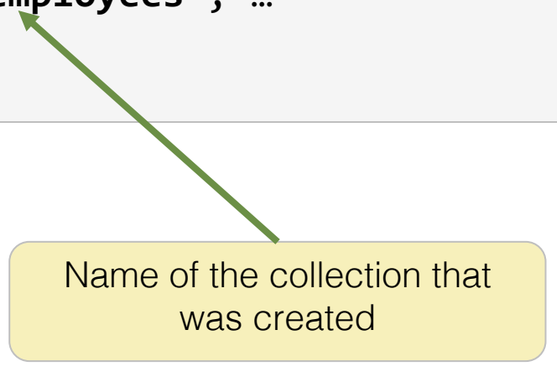
- To create a collection in a KV store, send a **`POST`** request to this endpoint: **`/storage/collections/config`**
- A request *must* include:
  - A namespace
    - In the namespace, the user "nobody" must be specified
  - A name for the KV store
- For example, to create a collection called **`employees`** in the **`search`** app:

```
$curl -X POST -k -s -u restclient:restC0der
http://localhost:8089/servicesNS/nobody/search/storage/collections/config -d
'output_mode=json&name=employees'
```

Creating a Collection (cont.)

Successful creation of a collection produces a result that begins like this:

```
{
 "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://34.212.244.5:8089/servicesNS/nobody/search/storage/collections/config",
   "updated": "2019-01-02T18:16:29+00:00", "generator": { … },
   "entry": [
     { "name": "employees", …
     ],
   … }
```



Adding Records to a Collection

- To add records to a collection, send a **`POST`** request to this endpoint: **`/storage/collections/data/`*`collectionName`***
- In the request header, set the **`Content-Type`** to **`application/json`**
  - All additions and updates to the KV store must be in JSON
- For example, to add this to the **`employees`** KV store:

```
{ "first_name": "Mitch",
   "last_name": "Fleischman",
   "contact": "800 555-1212",
   "city": "Springfield",
   "office": "cube 33",
   "rating": 2,
   "notes":["1st Note","2nd note","3rd note", "4th note", "Enough with the notes already"]
}
```

Adding Records to a Collection (cont.)

- POST call to `/storage/collections/data/<collection name>`

```
$curl -X POST -k -s -u restclient:restC0der
http://localhost:8089/servicesNS/nobody/search/storage/collections/data/employees?outp
ut_mode=json -H 'Content-Type: application/json' -d \
'{ "_key": "mfleischman",
   "first_name": "Mitch",
   "last_name": "Fleischman",
   "contact": "800 555-1212",
   "city": "Springfield",
   "office": "cube 33",
   "rating": 2,
   "notes":["1st Note","2nd note","3rd note", "4th note", "Enough with the notes"] }'
```

- Successful execution produces this result:

```
{"_key":"mfleischman"}
```


Adding Records to a Collection (cont.)

- In the collection, this data is added:

```
[
  { "first_name": "Mitch",
    "last_name": "Fleischman",
    "contact": "800 555-1212",
    "city": "Springfield",
    "office": "cube 33",
    "rating": "2",
    "_user": "nobody",
    "_key": "mfleischman" }
]
```

- Note the addition of the **`_user`** field
  - It is taken from the namespace – and is therefore always set to nobody

The _key is Optional, but…

- If a request to add an entry to a collection does *not* include a **_key**:
  - A **_key** field is created
  - A value is automatically assigned to it
- For example, here the **_key** is created and **5c2d06a84a1f6d0b19117a83** is assigned as its value:

```
$curl -X POST -k -s -u restclient:restC0der
http://localhost:8089/servicesNS/nobody/search/storage/collections/data/employees?output_mo
de=json -H 'Content-Type: application/json' -d \
'{ "first_name": "Helga", "last_name": "Geerhart",
   "contact": "+49 (089) 5555 1213", "city": "München", "office": "cube 47", "rating": 5,
   "notes":["1st Note","2nd note"] }'

{"_key":"5c2d06a84a1f6d0b19117a83"}
```


More Than One Record is One Too Many

- Using the **`/storage/collections/data/`*`collectionName`*** endpoint, only one record can be added or updated at a time
- For example, when you try to add two records, note the error that occurs:

```
$curl -X POST -k -s -u restclient:restC0der
http://localhost:8089/servicesNS/nobody/search/storage/collections/data/employees?output_
mode=json -H 'Content-Type: application/json' -d '[ \
  { "first_name": "Mimi", "last_name": "Labonq", "contact": "+33 1 44 55 66", "city":
    "Nouvions", "office": "cube 21", "rating": 4, "notes":["1st Note","2nd note"] },
  { "first_name": "Edith", "last_name": "Artois", "contact": "+33 1 44 55 67", "city":
    "Nouvions", "office": "cube 52", "rating": 1, "notes":["1st Note"] } ]'

{"messages":[{"type":"ERROR","text":"JSON in the request is invalid. (\nJSON parse error
at offset 2 of file \"{\n  …}]}
```


Adding Multiple Records with `batch_save`

To add multiple records, send a **`POST`** request to the `/storage/collections/data/`*`collectionName`*`/batch_save` endpoint

```
$curl -X POST -k -s -u restclient:restC0der
http://localhost:8089/servicesNS/nobody/search/storage/collections/data/employees/
batch_save?output_mode=json -H 'Content-Type: application/json' -d \
'[ { "first_name": "Mimi", "last_name": "Labonq", "contact": "+33 1 44 55 66",
"city": "Nouvions", "office": "cube 21", "rating": 4, "notes":["1st Note","2nd
note"] },{ "first_name": "Edith", "last_name": "Artois", "contact": "+33 1 44 55
67", "city": "Nouvions", "office": "cube 52", "rating": 1, "notes":["1st Note"] }
]'

[ "5c2d07d54a1f6d0b19117a86", "5c2d07d54a1f6d0b19117a87" ]
```


Querying a Collection

- To query a collection, send a `GET` request to the `/storage/collections/data/`*`collectionName`* endpoint
- Use the `query` parameter to specify the information to return from the collection

  `query={`*`querystring`*`}`

- The `query` parameter:
  - Should begin with `{` and end with `}`
  - If no boolean operators are specified, equality is used
  - All keys and values must be quoted (`"`) and separated by a colon (`:`)
  - All characters must be URL-safe (url-encoded)
    - For example, instead of `"`, use its ASCII value, preceded by a percent sign: `%22`

The `query` Parameter

- To find the `employee` record whith `first_name` key equal to **Mitch**, set the `query` parameter to:

```
{%22first_name%22:%22Mitch%22}
```

- Since this search uses equality, no boolean operator is needed in the `query` parameter

Querying a Collection: Example 1

- Find the **employee** record with **last_name** equal to **Fleischman**:

```
$curl -X GET -g -k -s -u restclient:restC0der
'http://localhost:8089/servicesNS/nobody/search/storage/collections/data/employees?query
={%22last_name%22:%22Fleischman%22}&output_mode=json'

[ { "first_name" : "Mitch", "last_name" : "Fleischman", "contact" : "800 555-1212",
"city": "Springfield", "office" : "cube 33", "rating" : "2", "_user" : "nobody", "_key"
: "mfleischman" } ]
```

- Note that in the **curl** command, the **-g** argument is used
  - This argument disables the URL globbing parser
  - This makes it possible to embed **{}** in the URL

Using Boolean Operators

These boolean operators can be specified with the **`query`** parameter:

- **`$and`** Logical AND
- **`$not`** Logical NOT
- **`$or`** Logical OR
- **`$gt`** Greater Than
- **`$gte`** Greater Than or Equal
- **`$lt`** Less Than
- **`$lte`** Less than or Equal
- **`$ne`** Not equal to

Querying a Collection: Example 2

Find **employee** records with **rating** less than 3

```
$curl -X GET -g -k -s -u restclient:restC0der
'http://localhost:8089/servicesNS/nobody/search/storage/collections/data/employees?query
={%22rating%22:{%22$lt%22:3}}&output_mode=json'

[ { "first_name" : "Edith", "last_name" : "Artois", "contact" : "+33 1 44 55 67", "city"
: "Nouvions", "office" : "cube 21", "rating" : 1, "notes" : [ "1st Note" ], "_user" :
"nobody", "_key" : "eartois" }, { "first_name" : "Mitch", "last_name" : "Fleischman",
"contact" : "800 555-1212", "city" : "Springfield", "office" : "cube 33", "rating" : 2,
"notes" : [ "1st Note", "2nd note", "3rd note", "4th note", "Enough with the notes
already" ], "_user" : "nobody", "_key" : "mfleischman" } ]
```

The non-URL encoded query is:

```
{"rating":{"$lt":3}}
```


Querying a Collection: Example 3

- When including more than one boolean operator inside **query**, enclose them in [ ]

  { %22*booleanOperator1*%22:[{%22*value*%22:{%22*booleanOperator2*%22}},
  {%22*value*%22:{%22*booleanOperator3*%22}},…]}

- For example, to find **employee** records with **rating** greater than or equal to 2 and less than 5:

```
$curl -X GET -g -k -s -u restclient:restC0der
'http://localhost:8089/servicesNS/nobody/search/storage/collections/data/employees?
query={%22$and%22:[{%22rating%22:{%22$gte%22:2}},{%22rating%22:{%22$lt%22:5}}]}&output_
mode=json' > myResults.json
```

The non-URL encoded query is:

```
{"$and":[{"rating":{"$gte":2}},{"rating":{"$lt":5}}]}
```


Updating or Deleting a Record

- To update or delete a record, it is necessary to know the value of the record's `_key`
  - The value of the `_key` field is used to construct an endpoint name
- To update a record, send a `POST` request to the `/storage/collections/data/`*`collectionName/key`* endpoint
  - Pass the fields/values to update as a request argument
- To delete a record, send a `DELETE` request to the `/storage/collections/data/`*`collectionName/key`* endpoint

Update a Record: Example

In the **employees** collection, update the **city** and **office** fields for the record with **_key** of **mfleischman**:

```
$curl -X POST -k -s -u restclient:restC0der
http://localhost:8089/servicesNS/nobody/search/storage/collections/data/employees/
mfleischman?output_mode=json -H 'Content-Type: application/json' -d'{ "city": "New
York", "office": "cube 145" } '

{"_key":"mfleischman"}
```

Upon the success of the record update, the value of the **_key** field is returned

Warning

Only the fields included will be present in the updated event, so If you're updating a record be sure to include ALL FIELDS, not only the ones being changed.

Delete a Record: Example

In the **employees** collection, to delete the record with **key** ***mfleischman***:

```
$curl -X DELETE -k -s -u restclient:restC0der
http://localhost:8089/servicesNS/nobody/search/storage/collections/data/employees/
mfleischman?output_mode=json -H 'Content-Type: application/json'
```

Upon the success of the record deletion, nothing is returned

Delete All Records in a Collection

- **Important**: Omitting the **`key`** from the endpoint causes all records in the collection to be deleted
- Needless to say, it is very dangerous.
  YOU HAVE BEEN WARNED
- For example, to delete all of the records in the **`employees`** collection:

```
$curl -X DELETE -k -s -u restclient:restC0der
http://localhost:8089/servicesNS/nobody/search/storage/collections/data/employees?
output_mode=json -H 'Content-Type: application/json'
```

After all the records are deleted, nothing is returned

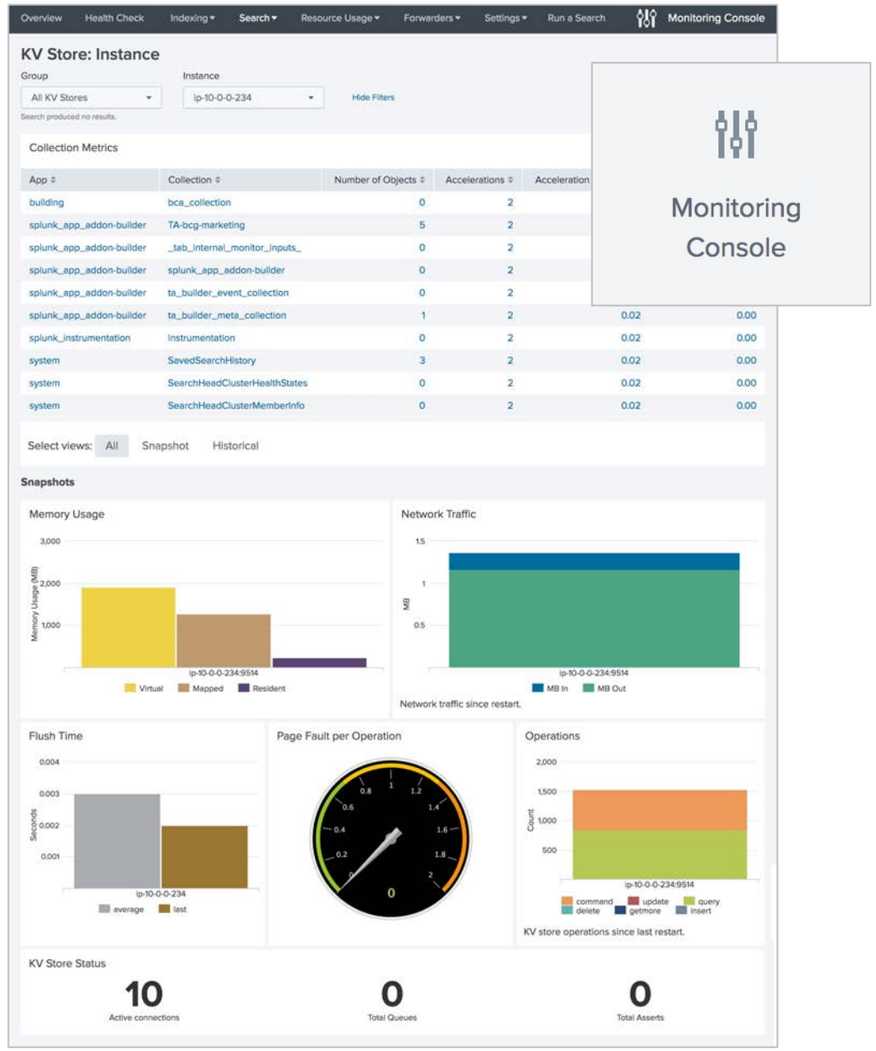
Delete a Collection

- To delete a collection, send a **DELETE** request to this endpoint: **/storage/collections/config/*collectionName***
- For example, in the **search** app, to delete the **employees** collection:

```
$curl -X DELETE -k -s -u restclient:restC0der
http://localhost:8089/servicesNS/nobody/search/storage/collections/config/empl
oyees -d 'output_mode=json'

{"links":{"create":"/servicesNS/nobody/search/storage/collections/config/_new"
,"_reload":"/servicesNS/nobody/search/storage/collections/config/_reload","_ac
l":"/servicesNS/nobody/search/storage/collections/config/_acl"},…"messages":[]
}
```

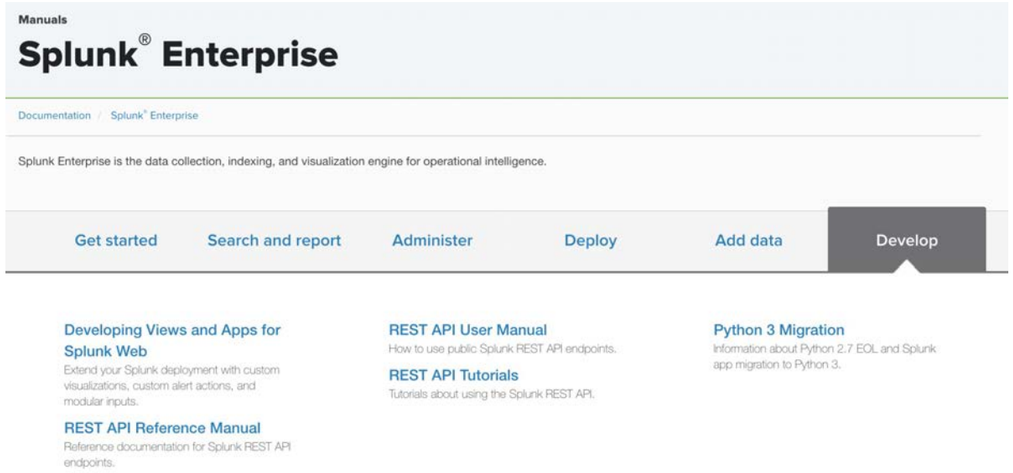
Monitoring KV Store Activity

- Monitoring Console
  - Available to administrators at:
    Settings > Monitoring Console > KV Store
- KV Store: Instance
- KV Store: Deployment
- Diagnose the health and performance of the KV Store
  - System optimization
  - Troubleshooting



Useful References

- Use configuration files to create a KV Store collection
  dev.splunk.com/view/SP-CAAAEZJ
- Configure KV Store lookups
  docs.splunk.com/Documentation/Splunk/latest/Knowledge/ConfigureKVstorelookups
- Use lookups with KV Store data
  dev.splunk.com/view/SP-CAAAEZH
- KV Store troubleshooting
  docs.splunk.com/Documentation/Splunk/latest/Admin/TroubleshootKVstore



Module 7 Lab Exercise

Time: 25 minutes

Tasks (using `curl` or `Postman`):

- Create a new collection in the search app called `employee`
- Populate the `employee` collection, using the contents of `employees.json`
- Produce `.json` output of all employee records, sorted by `last_name`
- Update the value of the rating field of a record in the `employee` collection
- Query the `employee` collection data for the record with `last_name` of Geerhart and whose `city` is Berlin

- OPTIONAL: Apply a constraint to the `employee` collection so that the `rating` field must be numeric

Module 8:
Using the HTTP Event Collector (HEC)

Objectives

- Describe the HTTP Event Collector (HEC)
- Create and use HEC tokens
- Input data using HEC endpoints
- Get indexer event acknowledgements
- Creating other inputs

Introduction to the HTTP Event Collector

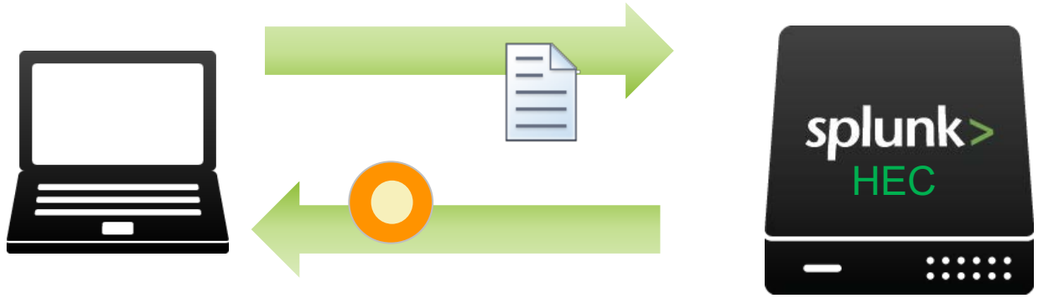
- HTTP Event Collector – or HEC – is a way to send data to Splunk
- Sources send data through HTTP or HTTPS directly to an HEC-enabled Splunk server
  - A local forwarder is not needed
  - Authentication is done using specialized tokens
  - Clients can be located outside the corporate network

Note

HEC can also be configured for high availability environments. Refer to docs.splunk.com for more information - or to the *Splunk Enterprise Data Administration* or *Splunk Cloud Administration* courses.

HEC and Tokens

- Each token is a GUID
- Tokens are sent with data to HEC
  - The token does not work for anything else - such as downloading data
- If HEC receives a valid token, it accepts the connection and ingests the event data
  - Clients can deliver a payload of application events as raw text or in JavaScript Object Notation (JSON) format



HEC and Indexing

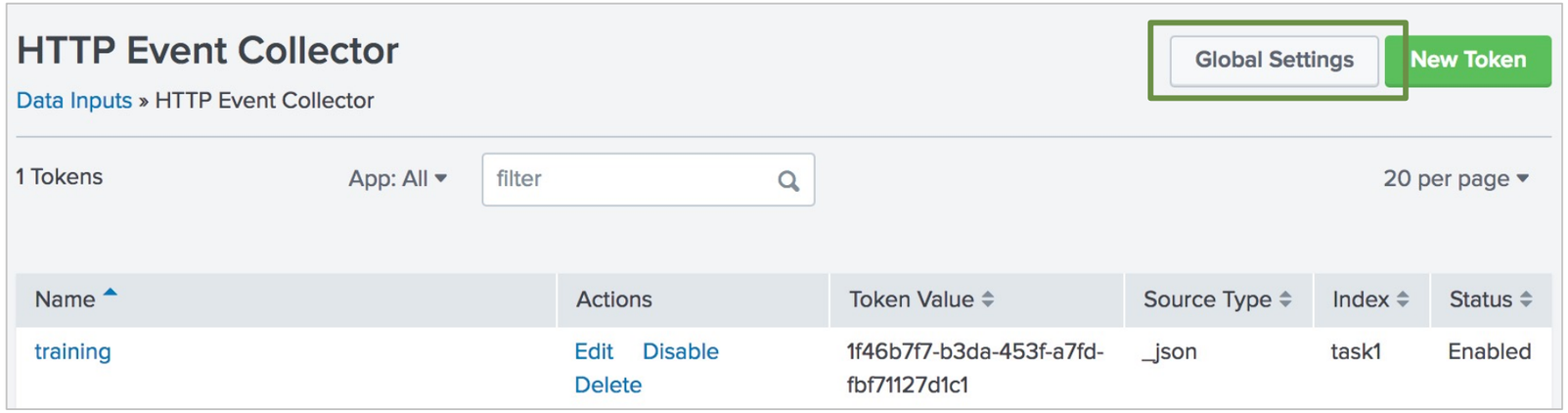
- Splunk indexes events received via HEC based on the token configuration and event data
  - The source, source type, and index that was specified in the token and/or data are assigned to the event
  - Event data can override token configuration
- HEC uses its own port, *separate* from the regular Splunk REST API endpoints
  - By default, the HEC endpoints use port 8088
  - This is configurable
- Can be configured for distributed environments

Note

Splunk HEC REST endpoints are separate from the other Splunk endpoints.

HEC Must Be Enabled

- By default, HEC is disabled
- In Splunk Enterprise, administrators can enable HEC
  - Navigate to **Settings > Data inputs > HTTP Inputs**
  - Click **Global Settings** > **Enabled**



- In Splunk Cloud, a support ticket is no longer required to enable HEC.

Creating Tokens

- Creating and editing tokens requires the **edit_token_http** capability, and you can use the web interface, cli or REST
  - Granting this capability does NOT provide any other admin related capabilities
- With **edit_token_http** capability, tokens can be created using Splunk Web or using a REST endpoint
- Create multiple tokens for different purposes
  - Create a token that works with getting data from a particular source type to a particular set of indexes
  - Repeat as needed for your various source types and indexes

Using REST to Create Tokens

- In addition to Splunk Web, tokens can be created or edited using Splunk REST endpoints
- To create a token, send a **`POST`** request to **`data/inputs/http`**
  - **`name`** is the only required request parameter
- (Optional) To set a default value for the **`sourcetype`** or **`index`** for use with this token, send it as a request parameter - for example:
  - Suppose when the token is created, the **`index`** is set to **`abc`**
  - When HEC requests are set using this token and no **`index`** is specified, the event data in the HEC request is stored in the **`abc`** index

```
curl -X POST -k -s -u username:training
"http://localhost:8089/services/data/inputs/http" -d name=myToken
-d index=abc -d output_mode=json
```


Using REST to Create Tokens (cont.)

- To edit an existing token, send a **POST** request to `services/data/inputs/http/tokenName`

```
curl -X POST -k -s -u username:training
"http://localhost:8089/services/data/inputs/http/myToken"
-d output_mode=json …
```

- Namespaces are also supported

Note

To get, set, delete, or enable/disable tokens, use Splunk REST endpoints, not the Splunk HEC endpoints. Therefore, use the Splunk port – not the HEC port!

Using REST to Enable/Disable Tokens

- To enable/disable tokens, send a **POST** request to:
  - `services/data/inputs/http/tokenName/enable`
  - `services/data/inputs/http/tokenName/disable`
- For example, to disable the projectX token, associated with the search app

```
curl -k -s -u username:password -X
"http://localhost:8089/servicesNS/admin/search/
data/inputs/http/projectX/disable" -d output_mode=json
```

Note

To construct the namespace, you *cannot* use `-`. Instead, use any user with access to the object. admin (and users with the admin role) have access to all objects.

Using REST to Get Token Information

- To get information about all tokens, send a **`GET`** request to **`data/inputs/http`**

```
curl -X GET -k -s -u username:password -X
"http://localhost:8089/services/data/inputs/http"
-d output_mode=json
```

- To get information about an existing token, send a **`GET`** request to **`servicesNS/`*`namespace`*`/data/inputs/http/`*`tokenName`**
  - For example, to find information about the projectX token, associated with the search app (i.e., the **`-/search`** namespace)

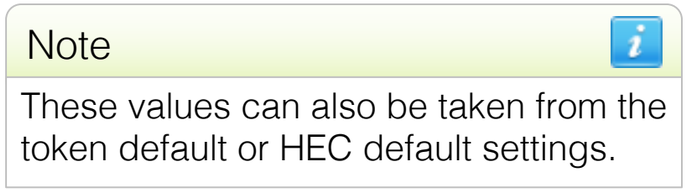
```
curl -X GET -k -s -u username:password -X
"http://localhost:8089/servicesNS/admin/search/data/inputs/http/projectX"
-d output_mode=json
```


HEC and Event Data

- Two types of event data can be sent to HEC endpoints:
  - **JSON** (through endpoint `/services/collector`)
  - **RAW** (through endpoint `/services/collector/raw`)
- Events can be sent individually or in groups ("batches")
- If configured, HEC can send acknowledgements to indicate the indexing status for events sent to it

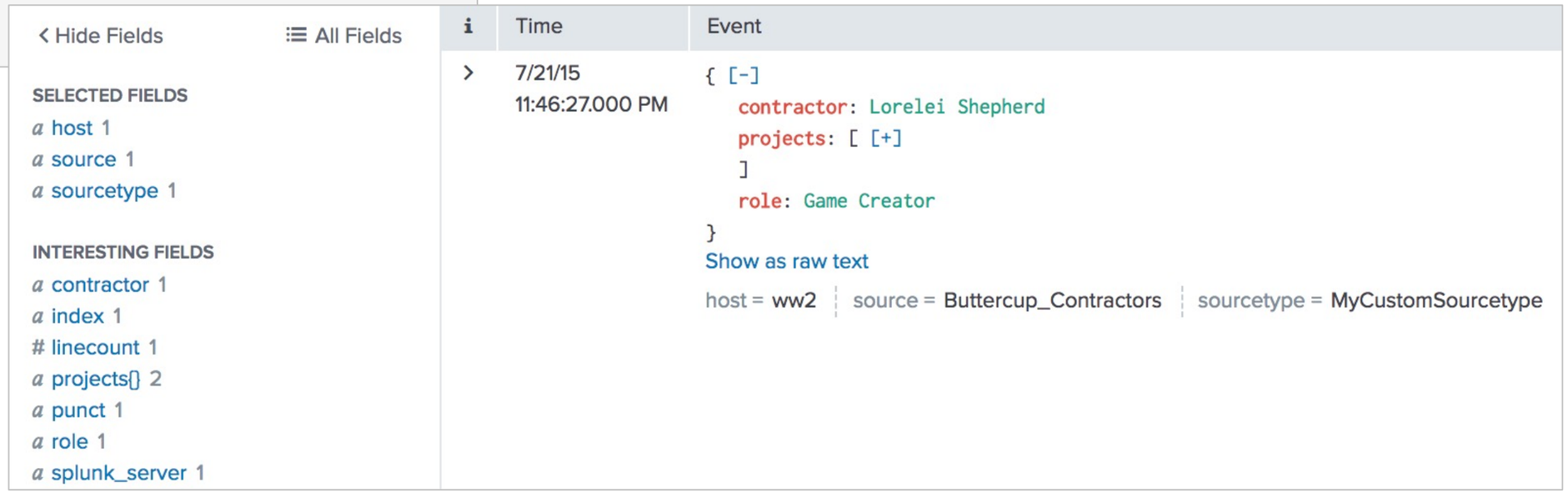
HEC and JSON Event Data

- JSON data can include the following Splunk fields as objects:
  - `time`: in epoch format
  - `host`: a computer/server/device name
  - `source`: where the object originated
  - `sourcetype`: a Splunk sourcetype
- JSON data can also include any custom fields inside an `event` object. This extraction happens automatically
  - No additional configuration is necessary!



HEC and JSON Event Data: Example 1

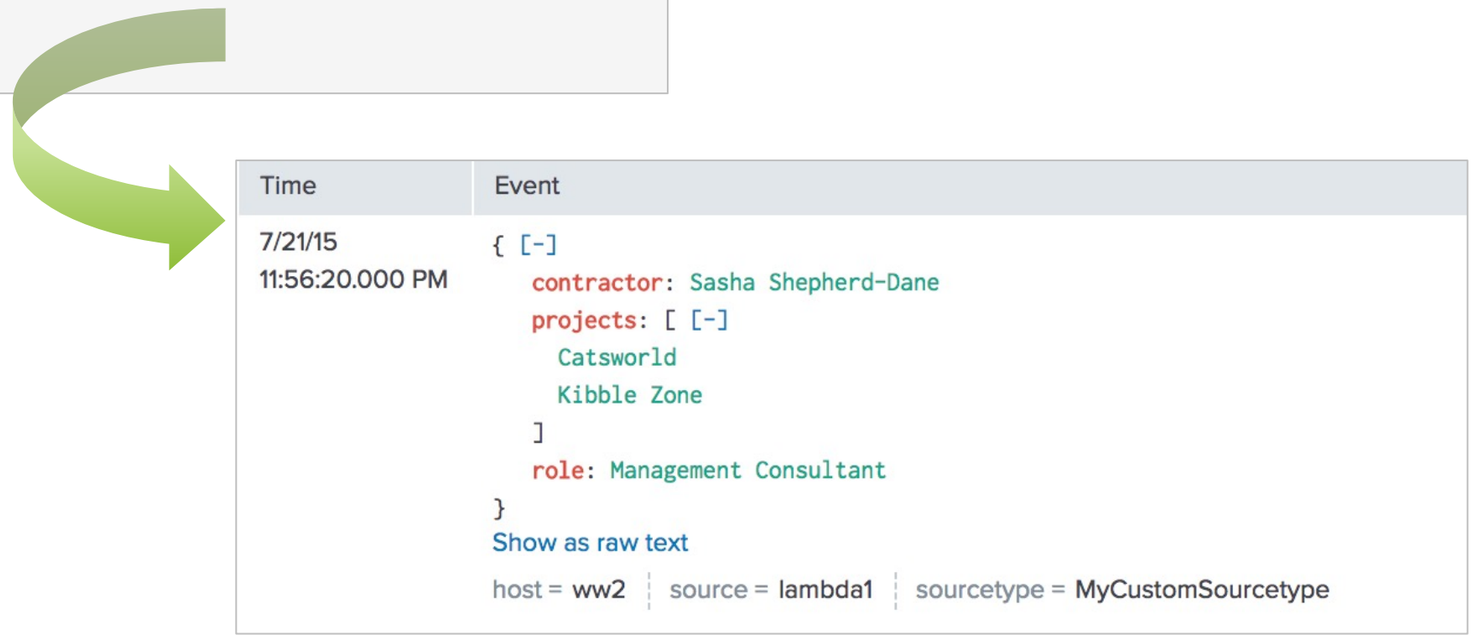
```
{
  "time": 1437522387,
  "host": "www2",
  "source": "Buttercup_Contractors",
  "sourcetype": "MyCustomSourcetype",
  "event": {
    "contractor": "Lorelei Shepherd",
    "role": "Game Creator",
    "projects" : ["Bones!", "Squirrel Tag"]
  }
}
```



HEC and JSON Event Data: Example 2

```
{
  "time": 1437522980,
  "host": "www2",
  "sourcetype": "MyCustomSourcetype",
  "event": {
    "contractor": "Sasha Shepherd-Dane",
    "role": "Management Consultant",
    "projects" : ["Catworld", "Kibble Zone"]
  }
}
```

- In the data, no source is specified
- HEC applied the default source from the token configuration



Ingesting JSON Data (Single Event)

- To ingest JSON data, send a `POST` request to `services/collector`
- The token is sent in the Authorization header
- The JSON event is sent as an argument by itself
- If successful, code 0 is returned
  - Any other code = failure

```
curl -X POST -k -s
"http://localhost:8088/services/collector"
-H "Authorization: Splunk tokenValue" -d '{
  "time": 1437522980,
  "host": "www2",
  "sourcetype": "MyCustomSourcetype",
  "event": {
    "contractor": "Lorelei Shepherd",
    "role": "Game Creator",
    "projects" : ["Bones!", "Squirrel Tag"]
  }
}'

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

Note

For a complete list of error codes, refer to http://docs.splunk.com/Documentation/Splunk/latest/Data/TroubleshootHTTPEventCollector

Note

On this slide and for the following slides in this lecture, Splunk HEC endpoints are used. *Note the port number!*

Ingesting JSON Data (Multiple Events)

- Send events one after the other (no commas)
- Optionally, pass common parameter values with the request
  - Pass as query parameters – i.e. in the URI string
  - If also passed in the event, the event values take precedence
- If time is not specified, the indexing time is used

```
curl -X POST -k -s
"http://localhost:8088/services/collector?host=ww2&sourcetype=MyCustomSourcetype"
-H "Authorization: Splunk tokenValue" \
-d '{"host":"www3", "event": { "contractor": "Bella Gatto", "role": "Game Creator"}}
{"event":{ "contractor": "Bob Loblaw", "role": "Management Consultant", "projects" :
["Gift Hunt"]}}'

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


Ingesting RAW Data

- Use this approach to ingest string data
- Send a **`POST`** request to **`services/collector/raw`**
- This endpoint requires a **`channel`**, as a query parameter
  - Must be a GUID. The GUID is an identifier of the sender of the request, and ACKs are returned by channel
- Event breaks are determined by the sourcetype configuration

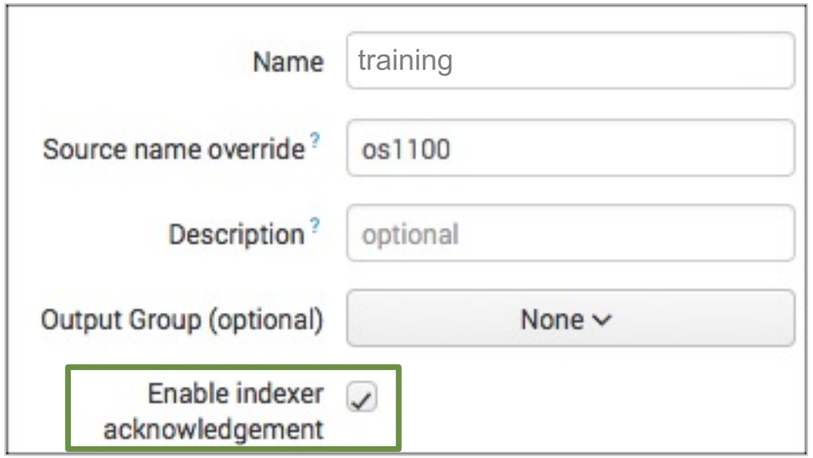
```
curl -X POST -k -s
"http://localhost:8088/services/collector/raw?host=ww3&sourcetype=simple&channel
=channelValue" -H "Authorization: Splunk tokenValue" -d "Event 1, sushi
Event 2, bagels, more bagels, even more bagels
Event 3, sanguinaccio, souvlaki, bigos"

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

Note

For more information about configuring sourcetypes, refer to the documentation.

Overview: Indexer Acknowledgement

- A network outage or Splunk system failure can cause events to be lost before they are indexed
- To make sure events are indexed, configure indexer acknowledgement
  - This is done on a per token basis
  - Can be enabled/disabled using Splunk Web or by sending a REST request

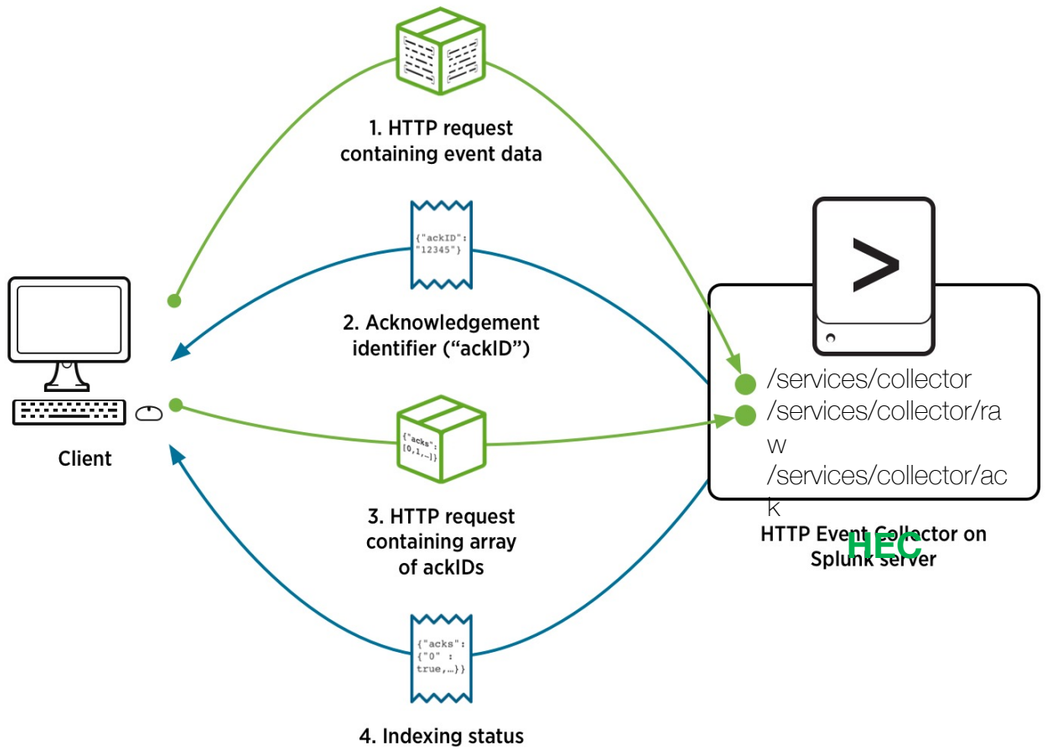



```
curl -X POST -k -s -u username:training
"http://localhost:8089/services/data/inputs/http/training"
-d useACK=1 -d output_mode=json
```

- Acknowledgement requires a channel to be sent

Overview: Indexer Acknowledgement (cont.)

- Each client request must provide a channel
  - A channel is a unique identifier created by the client
  - When an event is indexed, HEC sends the channel an **ACK ID**
- Client uses the `services/collector/ack` endpoint using one or more ACK IDs
- After an ACK ID has been received, it is released from memory



Note

Client uses `/services/collector/ack` to pull acknowledgments. If a request is not made to the `services/collector/ack` endpoint, the acknowledgments are not sent.

Indexer Acknowledgement Example

1. Client `POST` provides a channel

```
curl -X POST -k -s "http://localhost:8088/services/collector?
channel=client_provided_channel" -H "Authorization: Splunk tokenValue"
 -d '{ "event":"event 1"}{"event":"event 2"}'
```

2. Splunk HEC responds to the channel with an ACK ID for the request

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

> **Note**
>
> One `ackId` value is returned per request, *not* per event.

Indexer Acknowledgement Example (cont.)

3. Client requests ACK status check for ACK ID x (0 in this example)

```
curl -X POST -k -s "http://localhost:8088/services/collector/ack?
channel=client_provided_channel "
 -H "Authorization: Splunk tokenValue"
 -d '{ "acks":[0]}'
```

4. Splunk HEC responds with the indexer acknowledgment, either true (indexed) or false (unknown)

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


Performance

- Batching multiple events into single request can be quicker
  - Because a request's metadata applies to all events in the request, less data is sent overall
- If you want to use Acknowledgements you have to write the lookback logic to check packages; it is worth it

Standard Splunk Input Methods

- All Splunk inputs are supported by endpoints:
  - Monitor (directory/file monitoring)
  - Script (output from a program/script)
  - TCP/UDP
  - Modular inputs, such as DB Connect and others
- Use these endpoints when more granular control over how events are loaded into Splunk is needed
  - For example, use these endpoints to configure new data sources on multiple forwarders instead of using **Splunk Forwarder Management/Deployment Server**
  - Requires more setup and configuration than HEC

Note

Typically, to input event data, one would not contact forwarders directly; the Deployment Server can route event data to forwarders. Refer to the *Splunk Enterprise Data Administration* or the *Splunk Cloud Administration* courses for details.

Example: Monitor Input

- Create a new monitor on the file **news.log** in **/var/log**

```
$curl -X POST -k -s -u restclient:restC0der
http://localhost:8089/services/data/inputs/monitor
-d name=/var/log/news.log -d output_mode=json > monitorInput.json
```

- The input endpoints are documented here: https://docs.splunk.com/Documentation/Splunk/latest/RESTREF/RESTinput
- Note that calls like this can be made to the universal forwarder as well

> **Note**
>
> For details about monitor inputs, refer to docs.splunk.com – or to the *Splunk Enterprise Data Administration* or the *Splunk Cloud Administration* courses.

Module 8 Lab Exercise

Time: 20 minutes

Tasks (using `curl` or `Postman`):

- Create a new HEC token
- Use HEC tokens to ingest JSON and RAW event data
- Use Splunk Web to create an HEC token that uses indexer acknowledgement
- Use HEC tokens to
  - Ingest JSON event data and receive an ACK ID
  - Get an indexer acknowledgement that the previous event data was indexed
- **OPTIONAL:** Edit a python program to send JSON or RAW event data to HEC, with or without indexer acknowledgement

Appendix:
Useful Admin REST APIs

Objectives

- Get system information
- Managing Splunk Config Files
- Managing Indexes

Splunk System Information APIs

Splunk exposes several APIs that can be used to obtain information about the system and perform diagnostics. The most common APIs to get information from Splunk enterprise/cloud systems can be found under the `/server` endpoint:

- `/server/info` and `/server/sysinfo`
- `/server/introspection`
- `/server/health`
- `/server/roles`
- `/server/settings`
- … and others!

Instance and System Information APIs

- `/server/info`

Returns information about the Splunk instance such as **version**, **build**, **license info** and **status**, **overall system health**, **server roles** and others.

It is the first API to check in order to get the status of a Splunk instance.

```
| rest /services/server/info | transpose
```

- `/server/sysinfo`

Returns information about the **hardware** and **OS** hosting Splunk, such as memory, kernel version, disk space, available cores, architecture and relevant settings for **THP** and **ulimit**.

Use it to find out if your server settings matches requirements.

```
| rest /services/server/sysinfo | transpose
```


Introspection APIs

- `/server/introspection` is a set of APIs that provide real-time snapshot information on the performance of certain subsystems in Splunk.
  - It is used by the Monitoring Console in many reports to display the most current information on relevant areas of Splunk.
- It has several levels: **indexer**, **kvstore**, **pipelines**, **processors**, **queues** and **search**, and each one can also contain sublevels.
- The sublevels are listed as part of each API's response. Use it to explore the APIs!

Note

For details on pipelines, processors and queues, the course **Troubleshooting Splunk Enterprise** has a very comprehensive explanation on how these processes work and how to debug ingestion problems using processor/queue information.

Introspection APIs (cont.)

- Some examples of useful calls to introspection APIs:

```
| rest /services/server/introspection/indexer | table average_KBps status
```

  - Returns the average indexing rate of the server and indexing status

average_KBps	status
3.5253266767987625	normal

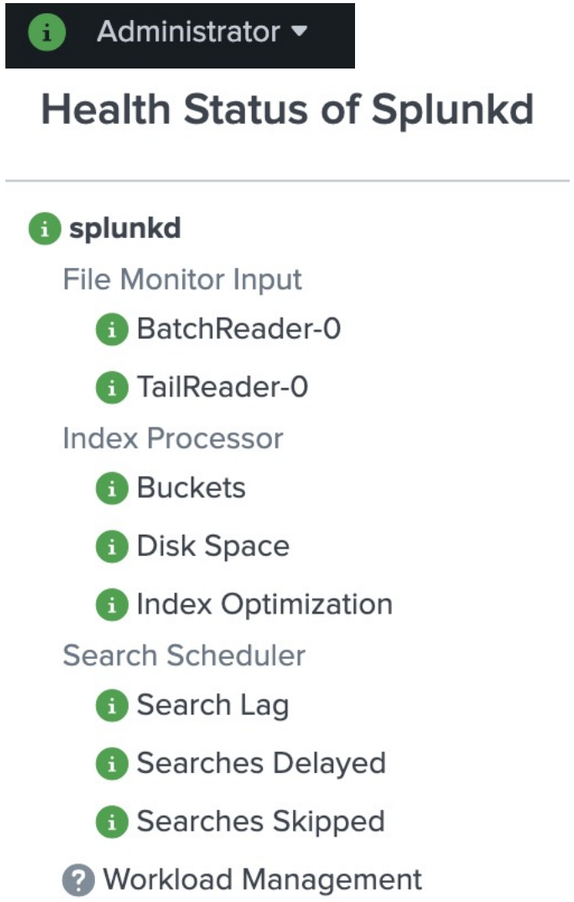
```
| rest /services/server/introspection/search/scheduler
| table avg_conc_util avg_lag searches_dispatched searches_skipped
```

  - Information on scheduled searches: average concurrency, average lag, how many scheduled searches dispatched in the last hour, count of skipped searches…

avg_conc_util	avg_lag	searches_dispatched	searches_skipped
0	31	4	0

Health APIs and Proactive Health Monitoring

- The Health APIs power the **Proactive Health Monitoring** feature in Splunk, which constantly monitors the health status of the main components in Splunk.
  - You can access this feature by clicking the icon next to your name on the Splunk menu. This icon will display a color representing the status of the system: green, yellow or red.
  - APIs in this class will return the color representing the status of each feature. The APIs may display more or fewer options depending on the roles of the machine, for example: if the machine is a deployment server, the deployment APIs will be enabled and the web interface will display the corresponding items and colors.

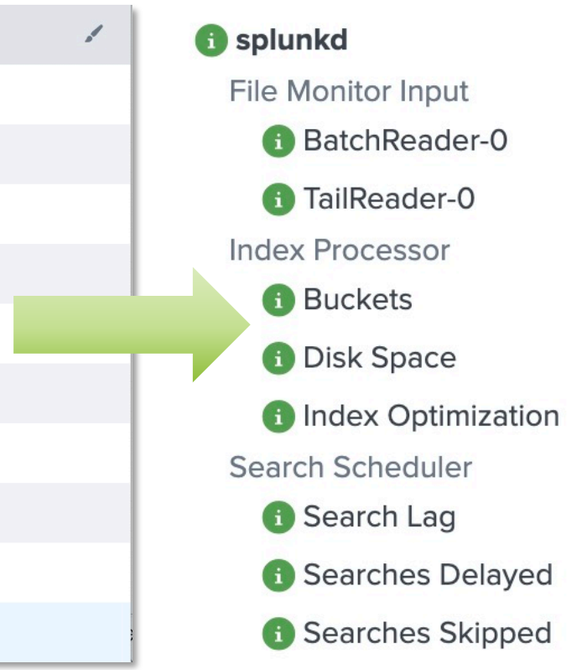


Health APIs

- To get the health information for all systems, according to the enabled server roles:

```
| rest /services/server/health/splunkd/details | transpose | search column=features.*
```

column	row 1
features.File Monitor Input.features.BatchReader-0.health	green
features.File Monitor Input.features.TailReader-0.health	green
features.File Monitor Input.health	green
features.Index Processor.features.Buckets.health	green
features.Index Processor.features.Disk Space.health	green
features.Index Processor.features.Index Optimization.health	green
features.Index Processor.health	green
features.Search Scheduler.features.Search Lag.health	green
features.Search Scheduler.features.Searches Delayed.health	green
features.Search Scheduler.features.Searches Skipped.health	green



Server Roles API

- Returns the server roles for each Splunk server. If executed on an instance with search peers, it displays the roles for all peers.

```
| rest /services/server/roles | table splunk_server role_list
```

splunk_server	role_list
ip-10-0-0-200	indexer license_master

- The ideal machine to execute this API is the the one where you enabled the **Monitoring Console** and set the server roles appropriately.

Server Settings APIs

- Returns the main server settings for this server. It maps the settings on "**Settings > Server Settings > General Settings**" and other settings mapped to the `server.conf` configuration file.
  - It maps many useful settings such as: **Splunk Server Name**, **Default Hostname**, **Run Splunk Web**, **Enable SSL**, **HTTP Port**, **Management Port** and some not on the web interface such as **Python Version**.
  - This API is read-only.

```
| rest /services/server/settings | fields - "eai:acl.*" | transpose
```

column	row 1
SPLUNK_DB	/opt/splunk/var/lib/splunk
SPLUNK_HOME	/opt/splunk
appServerPorts	8065
author	system
dfsDisabled	1
dfsMasterPort	9000
enableSplunkWebSSL	1
host	ip-10-0-0-200
host_resolved	ip-10-0-0-200
httpport	8000
id	https://127.0.0.1:8089/services/server/
kvStoreDisabled	0
kvStorePort	8191
mgmtHostPort	8089
minFreeSpace	5000
pass4SymmKey	7HPDetCqn97qaSXJduIHSsMSaud/Lv9W
published	
python.version	python3
serverName	ip-10-0-0-200
sessionTimeout	1h
splunk_server	ip-10-0-0-200
startwebserver	1

Managing Splunk Configuration Files

The `/properties` endpoint can be used to read or write to any configuration files in Splunk.

- Displays the consolidated, in-memory version of any configuration file in the system, if the caller has the permissions to do so.
  - Regular users can read from most configuration files.
  - Administrators have full power on any configuration file.
- Namespaces can be used to access a version of a configuration file that belongs to an app.
- Configuration files can also be created by calling this endpoint.
- For Splunk Cloud, access to this API is limited, as configuration files are maintained by the cloud admin team.

Splunk Configuration Files (cont.)

- To get a list of all configuration files in the system:

```
/services/properties
```

- To get all stanzas in a configuration file:

```
/services/properties/{file}
```

- To get all key properties of a stanza:

```
/services/properties/{file}/{stanza}
```

- To access the value of a key* property:

```
/services/properties/{file}/{stanza}/{key}
```

*Calling the key will return the property in plain text, not XML or JSON.

Generic config file layout:

```
[stanza]
key = value
key2 = value2

[anotherStanza]
key3 = “string on value 3”
```

Note

When using the REST API, call the configuration file name without the extension “.conf”.

Example: for props.conf call /services/properties/props/…

Splunk Configuration Files Example

- Creating a stanza at the **app** level then setting the properties:

```
$curl -s -k -u usr:pwd https://localhost:8089/servicesNS/-/search/properties/props -X
POST -d __stanza=proxylogs
```

  - No visible result is returned but you'll get HTTP status **201** (created).

```
$curl -s -k -u usr:pwd https://localhost:8089/servicesNS/-/search/properties/props
/proxylogs -X GET -X POST -d NO_BINARY_CHECK=true -d CHARSET=UTF-8
```

  - The second command returns a response:
  - Note this response is XML, but it is much simpler than responses given by other APIs.

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <messages>
    <msg type="INFO">Successfully modified 2 key(s)</msg>
  </messages>
</response>
```


Splunk Configuration Files Example (cont.)

- The resulting **`props.conf`** config file:

```
[proxylogs]
CHARSET = UTF-8
NO_BINARY_CHECK = true
```

  - This config file is located at `/opt/splunk/etc/apps/`**`search/local`**`/props.conf`

Splunk Configuration Files – Namespaces

- Note the namespace using "**-**" (dash) as the user. This tells Splunk to record the configuration file **props.conf** at the app level for the app search, local directory (`SPLUNK_HOME/etc/apps/<appname>/local`)
  - If a user is specified, the config file is recorded at the **user** directory `SPLUNK_HOME/etc/users/<username>/<appname>/local` instead.
  - If you don't use namespaces, the configuration files will be changed/edited at the system level (`SPLUNK_HOME/etc/system/local`).
- Make sure to use namespaces and avoid modifying any configuration files at the system level unless you know what you're doing.

File Explorer API

- If you need information about files on a specific directory, Splunk provides an API that works as a file browser:

`admin/file-explorer/<URL-encoded file path>`

- This API is very useful when you need to:
  - Verify if a file exists or not, or if Splunk has access to it.
  - Check basic properties like the file size and last modified time.
- The directory passed to the API must be URL-Encoded.
- This API does **not** support wildcards of any type on the path. It must be an exact path.

Warning

This is an internal, undocumented API. You won't find information about it in the official REST API documentation.

File Explorer API (cont.)

- For example, to obtain information about all files located at `/opt/log/www1`, first URL-Encode the path, which becomes `%2Fopt%2Flog%2Fwww1`, then use it on the API call:

```
| rest /services/admin/file-explorer/opt%2Flog%2Fwww1
| eval lastModifiedTime = strftime(lastModifiedTime,"%d/%m/%Y %H:%M:%S")
| table name fileSize lastModifiedTime title
```

name	fileSize	lastModifiedTime	title
access.log	35632	31/08/2021 19:43:35	opt/log/www1/access.log
secure.log	11368	31/08/2021 19:39:07	opt/log/www1/secure.log

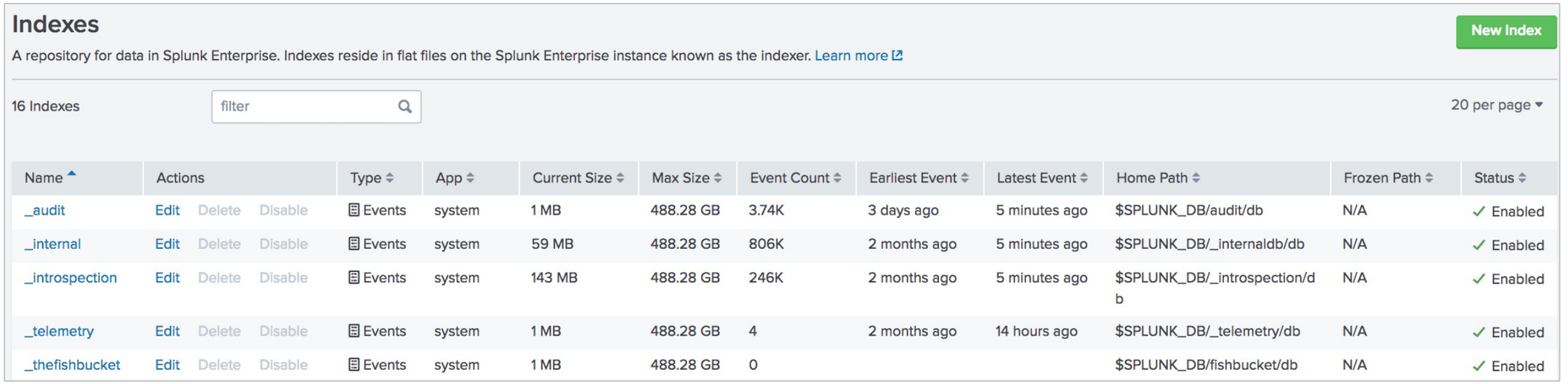
- This API only supports the GET method.

Note

You can use urlencoder.org to manually encode the filesystem paths to this API.

Managing Indexes in Splunk Web

- In general, administrators manage indexes, however sometimes developers need to manage them.
  - For example, testing and prototyping on non-production systems
- Indexes can be managed using Splunk Web
- If needed, indexes can also be managed using REST endpoints



Indexes

A repository for data in Splunk Enterprise. Indexes reside in flat files on the Splunk Enterprise instance known as the indexer. Learn more

New Index

16 Indexes | filter | 20 per page

Name	Actions	Type	App	Current Size	Max Size	Event Count	Earliest Event	Latest Event	Home Path	Frozen Path	Status
_audit	Edit Delete Disable	Events	system	1 MB	488.28 GB	3.74K	3 days ago	5 minutes ago	$SPLUNK_DB/audit/db	N/A	Enabled
_internal	Edit Delete Disable	Events	system	59 MB	488.28 GB	806K	2 months ago	5 minutes ago	$SPLUNK_DB/_internaldb/db	N/A	Enabled
_introspection	Edit Delete Disable	Events	system	143 MB	488.28 GB	246K	2 months ago	5 minutes ago	$SPLUNK_DB/_introspection/db	N/A	Enabled
_telemetry	Edit Delete Disable	Events	system	1 MB	488.28 GB	4	2 months ago	14 hours ago	$SPLUNK_DB/_telemetry/db	N/A	Enabled
_thefishbucket	Edit Delete Disable	Events	system	1 MB	488.28 GB	0			$SPLUNK_DB/fishbucket/db	N/A	Enabled

List Indexes

- To list all indexes, send a **`GET`** request to the **`data/indexes`** endpoint
- Optionally, specify an index data type:

  `all` | **`event`** | **`metric`** (default: **`all`**)

- To specify an **`output_mode`** with this endpoint, it is necessary to use url encoding – for example, using **`-G`** in **`curl`**

```
curl -G -k -s -u restclient:restC0der
http://localhost:8089/services/data/indexes
-d output_mode=json
```

Note

In the REST API reference, information about the index endpoints can be found under Introspection Endpoints.

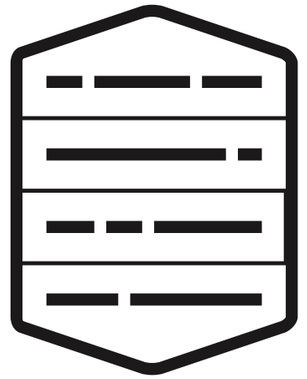
Getting Index Information

```
{ …
  "entry": [
    { "name": "_audit",
      …
      "acl": {
         "app": "system",
        "can_list": true,
        "can_write": true,
        "modifiable": false,
        "owner": "nobody",
        "perms": {
          "read": [ "*" ],
          "write":
            [ "admin", "splunk-system-role" ]
       },
       "removable": false,
       "sharing": "system"
    }
    "content" : {
        …
      },
    }…
}
```

- Data for each index is found within its own object in the **`entry`** array
- The index name is indicated by the **`name`** key
- The permission and sharing information can be found in the **`acl`** object
- Most of the index attributes are shown as keys inside the **`content`** object

Overview: Creating an Index

- To create an index, send a `POST` request to the `data/indexes` endpoint
- Arguments passed to the endpoint
  - Configure the index attributes
  - Must be sent url-encoded
- Only the `name` is required
- All other arguments have defaults or are optional



Index

Some Index Arguments to Consider

- **`maxTotalDataSizeMB`**: maximum index size, in MB
  - Default: 500000
- **`maxDataSize`**: maximum bucket size
  - Default: **`auto`**, which is 750 MB
  - For 10 GB buckets, set to **`auto_high_volume`**
- **`coldToFrozenDir`**:
  - Optional setting used to archive older data to a specified filesystem location
  - If not set, when data ages out of the index, it is deleted

Note

For a list of arguments that can be set, refer to the data/indexes documentation page

Creating an Index

Create a new index named **MyIndex**, with 10 GB buckets and a maximum size of 10 TB

```
curl -X POST -k -s -u restclient:restC0der
http://locahost:8089/services/data/indexes -d
name=MyIndex -d maxDataSize=auto_high_volume -d
maxTotalDataSizeMB=10000000 -d output_mode=json >
createIndex.json
```


Creating an Index (cont.)

```
{ …
  "entry": [
    { "name": "myindex",
      …
      "acl": {…}
      "content" : {
        …
        "coldToFrozenDir": "",
        "maxDataSize": "auto_high_volume",
        "maxTotalDataSizeMB": 10000000,
        …
   },
   …
}
```

The **name** is always stored in lowercase, regardless of the case of the value passed in the **name** request parameter

In the **entry** array, the **content** object contains the properties and their values that are configured during index creation

Note

Note that **coldToFrozenDir** is set to **""**. Recall that the default behavior for the index is to delete data that ages out of it. Putting a valid filesystem path here would archive old data. Refer to docs.splunk.com for more information - or to the *Splunk Enterprise Data Administration* or *Splunk Cloud Administration* courses.

Getting Information about a Specific Index

- To get information about a specific index, send a **`GET`** request to the **`/services/data/indexes/`*`indexName`*
  - If the index is not in your default namespace: **`/servicesNS/`*`user`*`/`*`app`*`/data/indexes/`*`indexName`*
- By default, the **`summarize`** request parameter is set to **`false`**
  - Returns a full listing of the index metadata
- For a quicker response, set **`summarize`** to **`true`**
  - Returns a slightly smaller set of the index metadata

```
curl –X GET -k -s -u restclient:restC0der
http://localhost:8089/services/data/indexes/myIndex
-d output_mode=json -d summarize=true
```

Getting Information about a Specific Index (cont.)

```
{ …
  "entry": [
    { "name": "_audit",
      …
      "acl": {
        "app": "system",
        "can_list": true,
        "can_write": true,
        "modifiable": false,
        "owner": "nobody",
        "perms": {
          "read": [ "*" ],
          "write":
            [ "admin", "splunk-system-role" ]
       },
       "removable": false,
       "sharing": "system"
    }
    "content" : {
        …
      },
    }…
}
```

- This data is in the same format as what you saw earlier, after the `POST` request to `/data/indexes`
  - This is true regardless of the `summarize` setting
- Since the metadata is only for a single index, the `entry` list contains only one object
- This object includes:
  - A `name` object
  - An `acl` object
  - A `content` object (which contains most of the metadata)

Deleting an Index

To delete an index, send a **DELETE** request to `/data/indexes/`*`indexName`*

```
curl -X DELETE -k -s -u restclient:restC0der
http://locahost:8089/services/data/indexes/myIndex
-d output_mode=json > deleteIndex.json
```


Deleting an Index (cont.)

```
{ …
  "entry": [
    { "name": "_audit",
      …
      "acl": {
         "app": "system",
        "can_list": true,
        "can_write": true,
        "modifiable": false,
        "owner": "nobody",
        "perms": {
          "read": [ "*" ],
          "write":
            [ "admin", "splunk-system-role" ]
       },
       "removable": false,
       "sharing": "system"
    }
    "content" : {
        …
      },
    }…
}
```

- Deleting an index returns the same information as listing all indexes
  - Each index is shown as an object in the **`entry`** array
  - The deleted index does NOT have an object in the **`entry`** array
    - No information about it is shown

Deleting Data from an Index

- Once events have been indexed, they *cannot* be changed
- Instead of changing the indexed events, you can:
  - Mask the events, so they do not appear in search results
    - Use the **`delete`** SPL command
    - For example, delete all events on the *ponies* index containing the word "bad":

```
index=ponies bad | delete
```

  - Delete ALL events in the index
    - Use the **`clean`** CLI command
    - For example, remove all events in ponies:

```
splunk clean eventdata -index ponies
```

  - Let the events age out of the index

Note

For more information about managing indexes – including using the **`delete`** SPL command and **`clean`** CLI command, refer to docs.splunk.com or to the *Splunk Enterprise Data Administration* or the *Splunk Cloud Administration* courses.

Thank You