**1.Splunk API Access Using Python**

Splunk exposes a REST API that you can call with Python (requests or splunk-sdk-python).  
Example using requests:

import requests

from requests.auth import HTTPBasicAuth

# Splunk details

splunk\_host = "https://your-splunk-host:8089"

username = "admin"

password = "changeme"

# Example: Search query

search\_query = 'search index=\_internal | head 10'

payload = {

'search': search\_query,

'output\_mode': 'json'

}

# Call Splunk REST API

response = requests.post(

f"{splunk\_host}/services/search/jobs",

auth=HTTPBasicAuth(username, password),

data=payload,

verify=False # disable SSL verify if self-signed cert

)

print(response.status\_code, response.text)

**The 8089 port is the Splunk Management Port for REST API.  
👉 Use /services endpoints (e.g., /services/search/jobs) to interact with Splunk.**

**Alternatively, use the Splunk Python SDK:**

**### pip install splunk-sdk**

import splunklib.client as client

service = client.connect(

host="your-splunk-host",

port=8089,

username="admin",

password="changeme"

)

for app in service.apps:

print(app.name)

**2. Working with Index via API**

Create an index:

service.indexes.create("my\_new\_index")

List indexes:

for index in service.indexes:

print(index.name)

**3. Working with Alerts via API**

Splunk alerts are stored as **saved searches**.

Create a saved search (alert):

kwargs = {

"name": "error\_alert",

"search": 'search index=\_internal error',

"alert\_type": "always",

"alert.severity": "3",

"alert.suppress": "0",

"alert.track": "1",

"is\_scheduled": "1",

"cron\_schedule": "\*/5 \* \* \* \*", # every 5 minutes

"action.email": "1",

"action.email.to": "you@example.com"

}

service.saved\_searches.create(\*\*kwargs)

Trigger a saved search:

search = service.saved\_searches["error\_alert"]

job = search.dispatch()

**4. Authentication Methods**

Splunk supports two main ways:

1. **Basic Authentication** (username/password)
   * Used like HTTPBasicAuth(user, pass)
   * Simple but less secure.
2. **Token Authentication (Preferred)**

First, generate a **Bearer Token**:

curl -ku admin:changeme https://splunk-host:8089/services/authorization/tokens \ -d name=mytoken -d "audience=users"

Use token in Python:

headers = {"Authorization": "Bearer <TOKEN>"}

requests.get(url, headers=headers, verify=False)

**5. Permissions Needed**

For API access, the user (or token role) needs **capabilities** assigned in Splunk roles:

* **Search Data** → search
* **Create Alerts** → schedule\_search, edit\_sourcetypes
* **Create/Manage Indexes** → edit\_index, delete\_by\_keyword
* **Use REST API** → rest\_apps\_management, rest\_properties\_get, etc.
* **Basic API Access** → edit\_tcp, edit\_udp (if ingesting data)

👉 Usually, the **power** or **admin** role has these by default. For production, create a custom role with only necessary capabilities.

So in summary:

* Use **port 8089 REST API** with requests or splunk-sdk.
* **Saved searches = alerts**, create via API.
* **Indexes** can be listed/created via API.
* Start with **Basic Auth** (username/password), but in production use **Bearer Token**.
* Ensure role has *search, schedule\_search, edit\_index, rest\_ permissions*\*.

**Ready-to-run Python script** that covers:

1. **Connect to Splunk**
2. **Run a query**
3. **Create an index**
4. **Create an alert (saved search)**

### Install SDK

pip install splunk-sdk

**What this script does**

1. Connects to Splunk (via SDK).
2. Runs a sample search (index=\_internal | head 5).
3. Creates a new index (my\_new\_index) if it doesn’t exist.
4. Creates a scheduled alert (error\_alert) that runs every 5 minutes and emails results.

**Python Script**

import splunklib.client as client

import splunklib.results as results

# ----------------------------

# 1. Connect to Splunk

# ----------------------------

splunk\_host = "your-splunk-host"

splunk\_port = 8089 # Management port for API

username = "admin"

password = "changeme"

service = client.connect(

host=splunk\_host,

port=splunk\_port,

username=username,

password=password

)

print("✅ Connected to Splunk")

# ----------------------------

# 2. Run a Search Query

# ----------------------------

query = "search index=\_internal | head 5"

job = service.jobs.create(query)

# Wait until job is done

while not job.is\_done():

pass

print("✅ Search completed. Results:")

for result in results.ResultsReader(job.results(output\_mode="json")):

if isinstance(result, dict):

print(result)

# ----------------------------

# 3. Create a New Index

# ----------------------------

index\_name = "my\_new\_index"

if index\_name not in [idx.name for idx in service.indexes]:

service.indexes.create(index\_name)

print(f"✅ Index '{index\_name}' created")

else:

print(f"ℹ️ Index '{index\_name}' already exists")

# ----------------------------

# 4. Create an Alert (Saved Search)

# ----------------------------

alert\_name = "error\_alert"

# Delete existing alert with same name (if any)

if alert\_name in service.saved\_searches:

service.saved\_searches[alert\_name].delete()

# Create a new saved search (alert)

kwargs = {

"search": "search index=\_internal error",

"is\_scheduled": "1",

"cron\_schedule": "\*/5 \* \* \* \*", # every 5 minutes

"alert\_type": "always",

"alert.severity": "3",

"alert.track": "1",

"action.email": "1",

"action.email.to": "you@example.com"

}

service.saved\_searches.create(alert\_name, \*\*kwargs)

print(f"✅ Alert '{alert\_name}' created and scheduled")

To access Splunk's REST API using Python—especially for tasks like querying indexes, managing alerts, or retrieving search results—you’ll need to set up **authentication** and ensure your user account has the appropriate **roles and permissions**. Here's a breakdown tailored to your DevOps and cloud engineering background:

Authentication Basics

Splunk supports several authentication methods, but for most Python-based API access, **Basic Authentication** or **Token Authentication** is used.

1. **Basic Authentication**

* You send your Splunk **username and password** encoded in Base64 via the Authorization header.
* Example header:

Authorization: Basic <base64-encoded-username:password>

2. **Token Authentication (Recommended for security)**

* Generate a **Personal Access Token** in Splunk (requires admin privileges).
* Use it like this:

Authorization: Bearer <your-token>

Required Roles and Permissions

To interact with alerts, indexes, and searches, your Splunk user must have specific capabilities. These are tied to **roles** assigned to your account.

Minimum Role Capabilities

| **Capability** | **Purpose** |
| --- | --- |
| search | Run searches on indexes |
| list\_searches | View saved searches and alerts |
| edit\_searches | Create or modify saved searches |
| schedule\_search | Schedule alerts or reports |
| list\_alerts | View triggered alerts |
| edit\_alerts | Create or modify alerts |
| admin\_all\_objects | (Optional) Access all objects across apps |

You can check or modify these in **Settings → Roles → [Your Role] → Capabilities**.

Python Example (Basic Search)

import requests

splunk\_url = "https://your-splunk-instance:8089"

search\_query = "search index=main error | head 10"

auth\_token = "your-token"

headers = {

"Authorization": f"Bearer {auth\_token}"

}

data = {

"search": search\_query,

"output\_mode": "json"

}

response = requests.post(f"{splunk\_url}/services/search/jobs", headers=headers, data=data)

print(response.json())

**Python script to monitor Splunk alerts or automate index queries**

**Prerequisites**

**Before we dive into code, make sure you have:**

* Splunk Enterprise or Splunk Cloud with REST API enabled
* A user account with search, list\_alerts, and schedule\_search capabilities
* A Personal Access Token or username/password (token preferred)
* Python 3.x with requests installed (pip install requests)

**Python Script: Splunk Search + Alert Monitor**

import requests

import time

# === CONFIGURATION ===

SPLUNK\_HOST = "https://your-splunk-instance:8089"

TOKEN = "your-splunk-token" # Use Bearer token for security

SEARCH\_QUERY = "search index=main error OR failure | head 10"

HEADERS = {

"Authorization": f"Bearer {TOKEN}"

}

# === STEP 1: Create a Search Job ===

def create\_search\_job(query):

payload = {

"search": query,

"output\_mode": "json"

}

response = requests.post(f"{SPLUNK\_HOST}/services/search/jobs", headers=HEADERS, data=payload)

response.raise\_for\_status()

sid = response.json()["sid"]

return sid

# === STEP 2: Poll for Completion ===

def wait\_for\_job(sid):

while True:

response = requests.get(f"{SPLUNK\_HOST}/services/search/jobs/{sid}", headers=HEADERS, params={"output\_mode": "json"})

response.raise\_for\_status()

job\_status = response.json()["entry"][0]["content"]["dispatchState"]

if job\_status == "DONE":

break

time.sleep(2)

# === STEP 3: Fetch Results ===

def get\_results(sid):

response = requests.get(f"{SPLUNK\_HOST}/services/search/jobs/{sid}/results", headers=HEADERS, params={"output\_mode": "json"})

response.raise\_for\_status()

return response.json()["results"]

# === MAIN EXECUTION ===

if \_\_name\_\_ == "\_\_main\_\_":

sid = create\_search\_job(SEARCH\_QUERY)

wait\_for\_job(sid)

results = get\_results(sid)

for result in results:

print(result)

**Optional: Monitor Alerts**

To list triggered alerts:

def list\_alerts():

response = requests.get(f"{SPLUNK\_HOST}/services/saved/searches", headers=HEADERS, params={"output\_mode": "json"})

response.raise\_for\_status()

alerts = response.json()["entry"]

for alert in alerts:

if alert["content"].get("is\_scheduled") and alert["content"].get("alert\_type"):

print(f"Alert: {alert['name']} | Type: {alert['content']['alert\_type']}")