Transform Splunk Enterprise Management with Integrated CI/CD Pipelines

DEV1408

.conf25

splunk>

Forward-looking statements

This presentation may contain forward-looking statements regarding future events, plans or the expected financial performance of our company, including our expectations regarding our products, technology, strategy, customers, markets, acquisitions and investments. These statements reflect management's current expectations, estimates and assumptions based on the information currently available to us. These forward-looking statements are not guarantees of future performance and involve significant risks, uncertainties and other factors that may cause our actual results, performance or achievements to be materially different from results, performance or achievements expressed or implied by the forward-looking statements contained in this presentation.

For additional information about factors that could cause actual results to differ materially from those described in the forward-looking statements made in this presentation, please refer to our periodic reports and other filings with the SEC, including the risk factors identified in our most recent quarterly reports on Form 10-Q and annual reports on Form 10-K, copies of which may be obtained by visiting the Splunk Investor Relations website at www.investors.splunk.com or the SEC's website at www.sec.gov. The forward-looking statements made in this presentation are made as of the time and date of this presentation. If reviewed after the initial presentation, even if made available by us, on our website or otherwise, it may not contain current or accurate information. We disclaim any obligation to update or revise any forward-looking statement based on new information, future events or otherwise, except as required by applicable law.

In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. We undertake no obligation either to develop the features or functionalities described, in beta or in preview (used interchangeably), or to include any such feature or functionality in a future release.

splunk> .conf25

From Code to Console: Streamlining Splunk Knowledge Object Deployments with CI/CD

William ‘Hunter’ Jarvis

Developer / Splunk Enterprise TO | Health Insurance Industry

.conf25

splunk>

Table of Contents

Introduction

Who are we?
How do we use Splunk?

- Health Insurance company based in the SE
- Partners with Splunk for over 6 years
- Utilize Splunk for IT Operations
- Solely on-premise
- 5 tier search-head-cluster
- 14 tier indexer-cluster
- Intake approx. 1.8B events / day
- Currently service 1,000 - 1,100 users
- Approx. 1,100 alerts, 950 reports, 750 dashboards
- Run 175,000+ searches per day

What problem did we solve?

How we initially did things

- Allowed users to publish via the web GUI
- Little to no oversight of searches/scheduling
- No backup of Knowledge Objects

What problems did this create?

Difficult to track privileges and who had them.

Allowed for the introduction of ‘bad’ searches.

Search schedules spikes at popular times.

Risk of KO/config loss. No revision history.

The initial “solution”

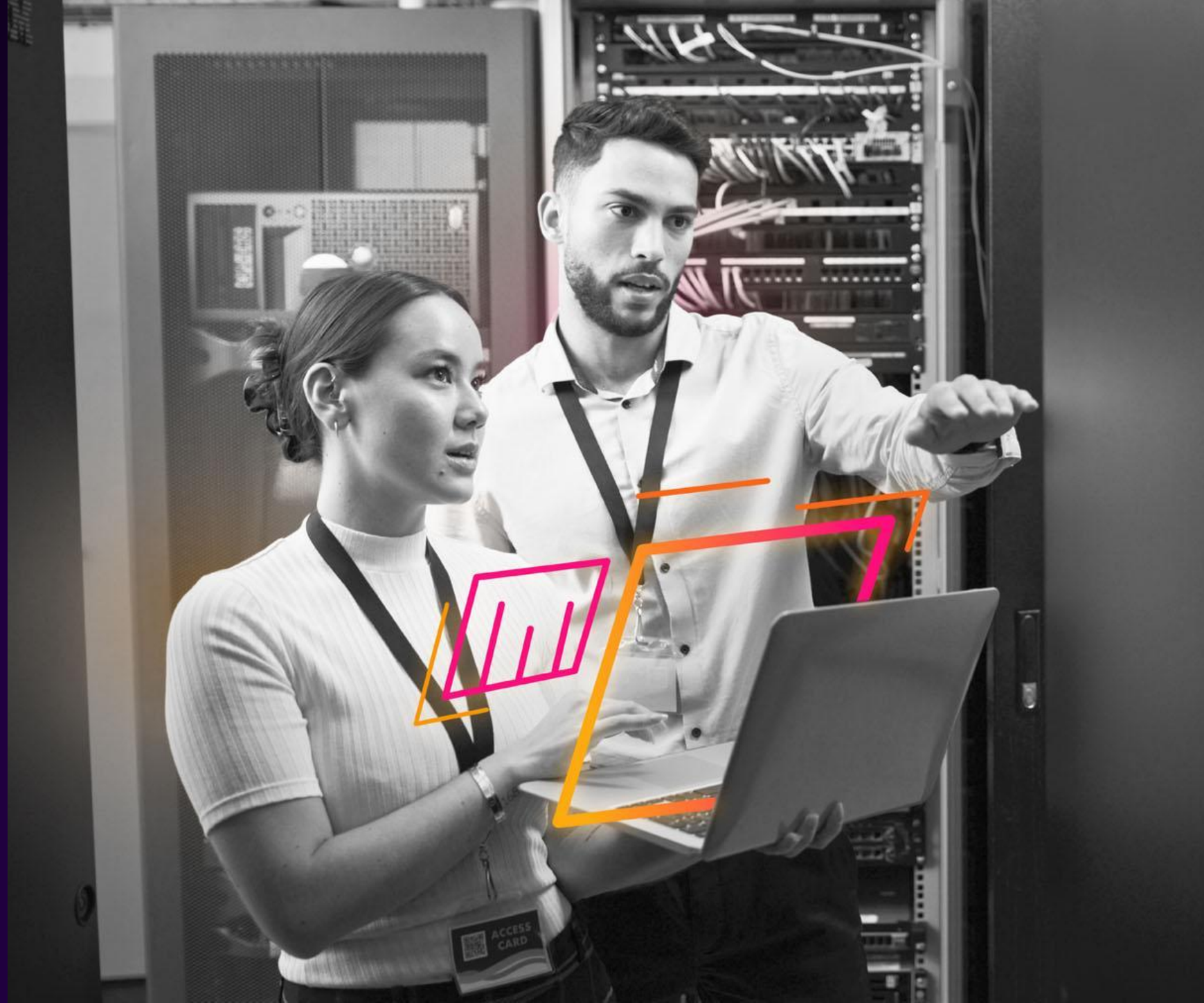
- Moved to a service request model
- Admins worked service request for each customer need

- This soon lead to an unmanageable backlog of work
- Splunk admins were inundated with requests and customers were not happy with their wait times

The Problem

How do we allow our customers to self-serve their Splunk knowledge objects and configurations while maintaining oversight from the Splunk admins?

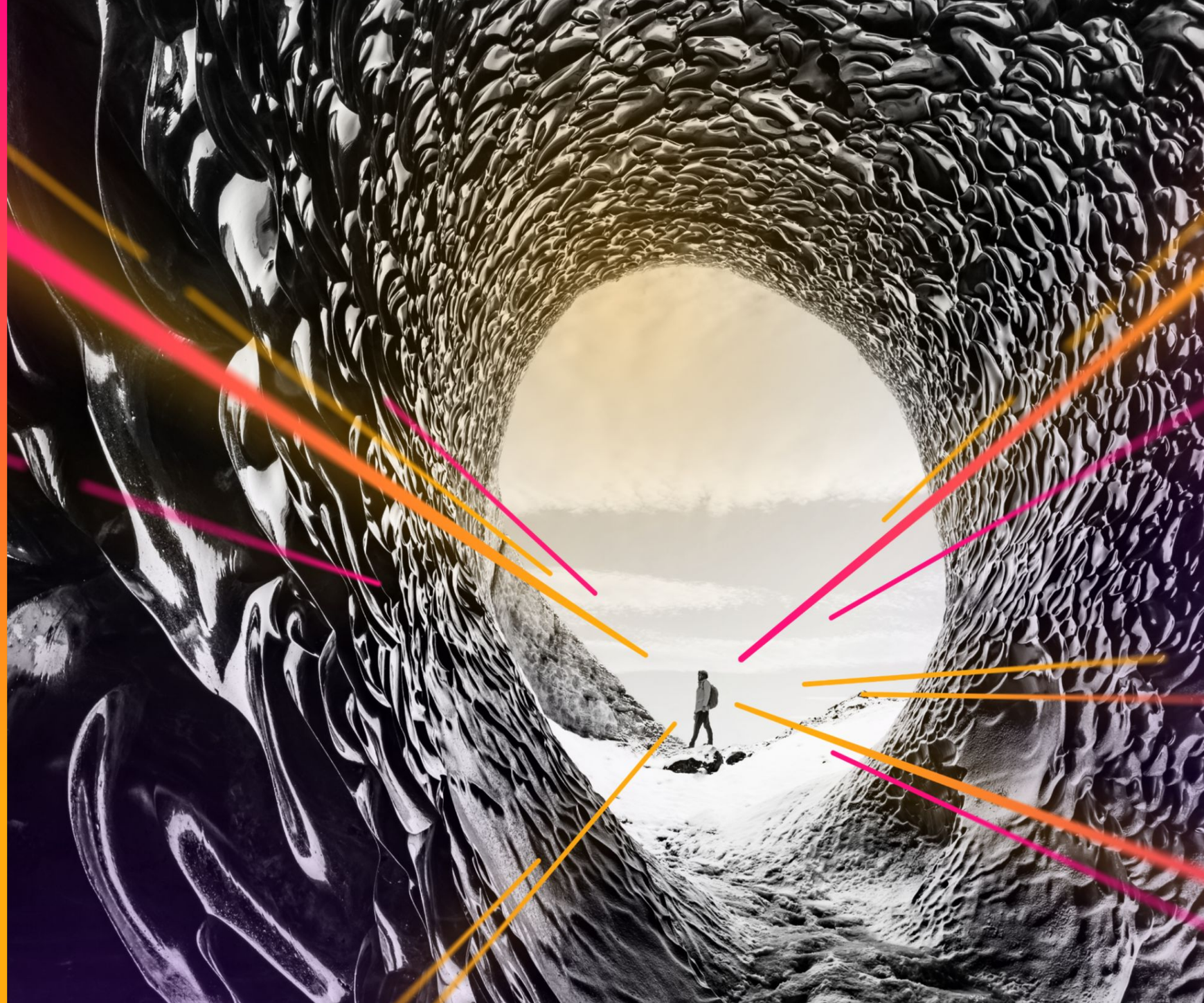
Bonus points for backups and revision history



How did we solve it?

CI/CD Pipeline

After careful research and consideration, we determined that a CI/CD pipeline was the best solution.



What tools did we use?

- GitHub Enterprise
  - GitHub repositories store the code
  - Acts as the backup to the knowledge objects stored on disk
  - Commit history allows for version control and rollbacks
- Python
  - Custom script developed in-house
  - Obtains the git history and any altered or new code
  - Accesses the Splunk search heads via the REST API to post or delete knowledge objects
- Jenkins
  - CI/CD pipeline tool used kick-off the pipeline and run the script
  - Handles cloning the repo and connectivity to the Splunk infrastructure

Automation walkthrough

- Two GitHub repositories were established.
  - One houses code and configurations for alerts, reports and lookups.
  - Another for classic dashboard & dashboard studio code
- Each repository has:
  - A python script to deploy the knowledge objects
  - A config.json file which stores Splunk server IPs and API endpoints to make the POST and DELETE calls etc.
  - A permissions.json file storing the permissions and ownership configurations for the knowledge objects.
  - A Jenkinsfile. This is the Jenkins configuration file written in Groovy.

config.json (alerts/reports/lookups)

```
{
    "username": "",
    "password": "",
    "use_https": true,
    "url": "<SPLUNK.SERVER.IP",
    "url_qa": "<QA.SPLUNK.SERVER.IP",
    "port": "8089",
    "uri_path_search": "servicesNS/nobody/search/saved/searches",
    "uri_path_lookups": "servicesNS/nobody/search/data/lookup-table-files",
    "uri_path_lookups_upload": "services/data/lookup_edit/lookup_contents",
    "search_path": "search",
    "lookup_path": "lookups",
    "archive_path": "archive",
    "metadata_ext": ".metadata"

}

```

```
sh "python3 $WORKSPACE/splunk_api.py --username ${USER} --password ${PASS}"
```

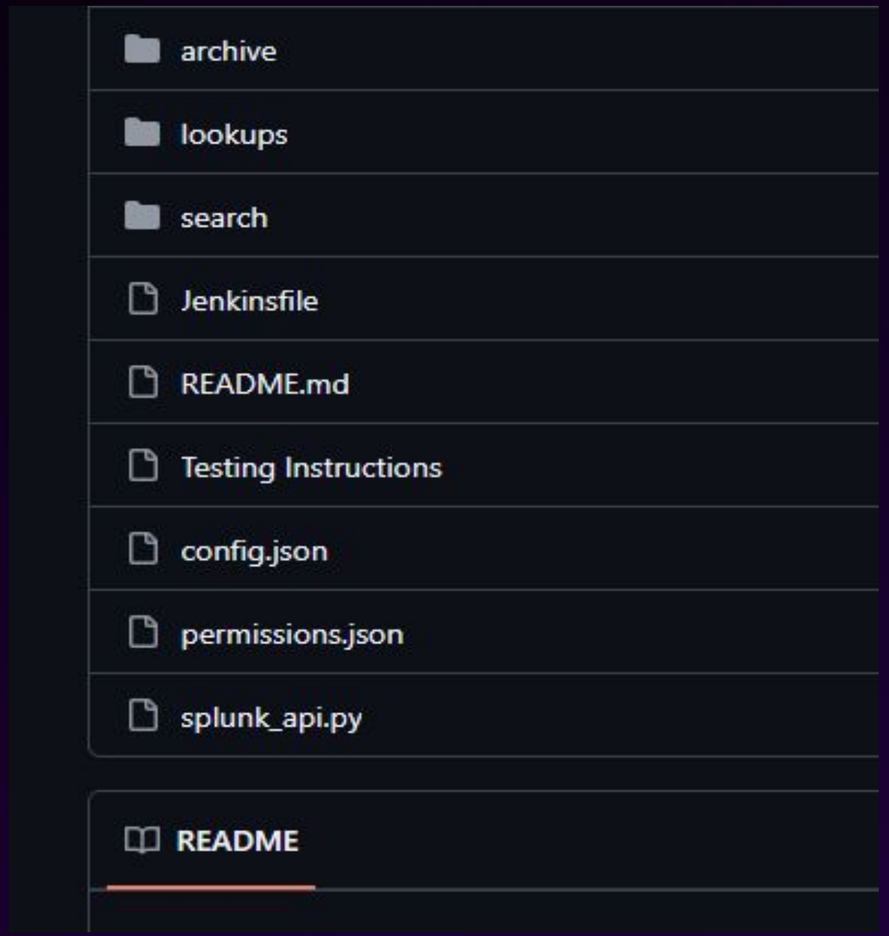
config.json (dashboards)

```
{
    "username": "",
    "password": "",
    "use_https": true,
    "url": "SPLUNK.SERVER.IP",
    "port": "8089",
    "uri_path": "servicesNS/nobody/search/data/ui/views",
    "search_path": "dashboards",
    "metadata_ext": ".metadata"

}

```

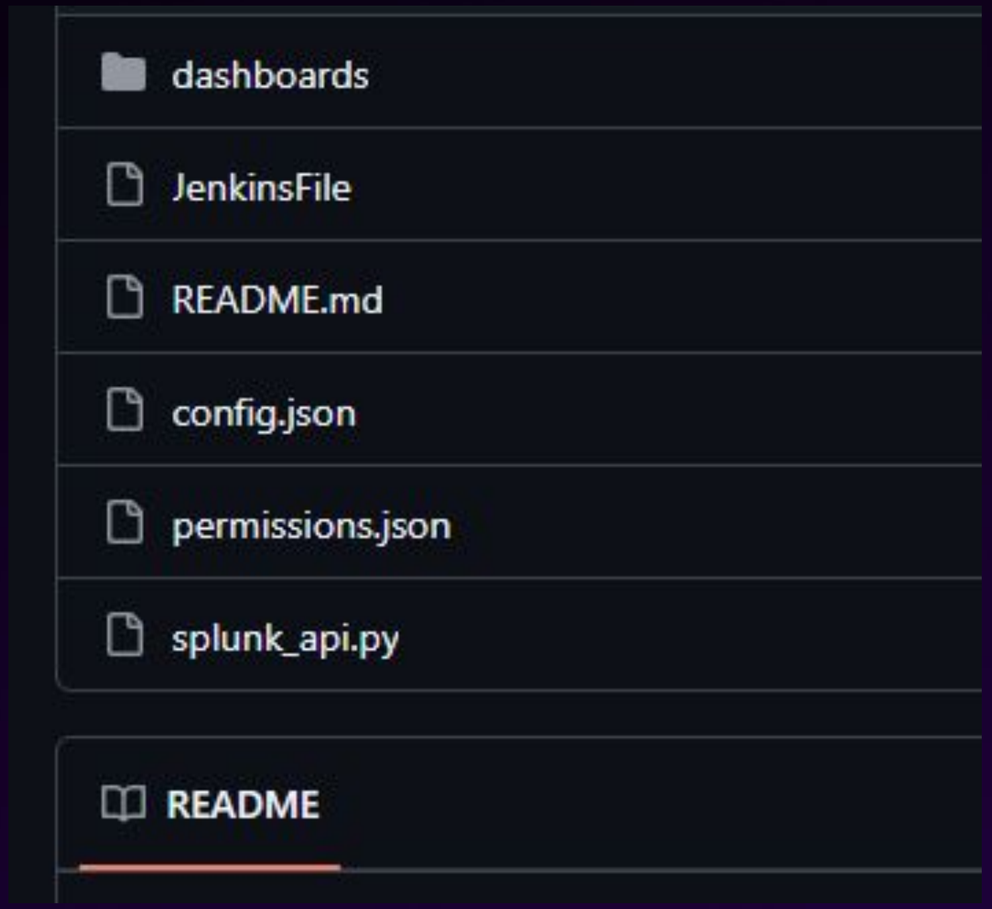
permissions.json

```
{
    "user": "nobody",
    "app": "search",
    "sharing": "global",
    "owner": "nobody",
    "perms.read": "*",
    "perms.write": "admin"
}
```

Repository Structure (alerts/reports/lookups)



Repository Structure (dashboards)



Automation walkthrough cont.

- The python script for alerts, reports and lookups:
  - Reads configuration and credentials from a JSON file
  - Detects changed Splunk knowledge object files using Git
  - Processes search, alert, and lookup files from designated directories as defined in the config.json file
  - Loads associated metadata for each object by matching the filename with the corresponding filename ending in .metadata
  - Prepares and formats data for API submission
  - Uses the Splunk REST API to create, update, or delete objects on the search head cluster
  - Updates permissions to make objects public for all users

Automation walkthrough cont.

```python
# Main function
def main():
    # Creating an ArgumentParser object
    parser = argparse.ArgumentParser(description="<Your parser description here>",)

    # Adding arguments to the parser
    parser.add_argument("-v", "--verbose",
                        required=False,
                        help="Increase output verbosity",
                        default=0,
                        action="count")
    parser.add_argument("--username",
                        required=False,
                        help="username accessing Splunk",
                        default="api_user",
                        metavar="<string>")
    parser.add_argument("--password",
                        required=False,
                        help="password accessing Splunk",
                        default="",
                        metavar="<string>")
    parser.add_argument("--processall",
                        required=False,
                        help="Process all files",
                        default=False,
                        action="store_true")
    parser.add_argument("--test",
                        required=False,
                        help="Use QA Splunk URL",
                        default=False,
                        action="store_true")
```

```python
# Get the last files changed in the merge
changed_files = get_changed_files()
```

```python
# Function to get the list of files changed in the last merge
def get_changed_files():
    files_changed = []
    # Run the git command to get the list of files changed in the last merge
    process = subprocess.Popen(['git', 'log', '-m', '--name-only', '-1'],
```

Automation walkthrough cont.

```
for dirpath, dirnames, filenames in os.walk(dir_path + "/" + config["archive_path"]):
    for filename in filenames:
        if (any(filename in files for files in changed_files)):
            print (os.path.join(dirpath, filename))
            if (re.search('\.metadata$', dirpath + "/" + filename)):
                print ("Skipping file: " + dirpath + "/" + filename)
                continue
            else:
                print ("Using file: " + dirpath + "/" + filename)
                post_data = {}
                make_api_call(config, args, filename, post_data, "archive")
```

Automation walkthrough cont.

```
for dirpath, dirnames, filenames in os.walk(dir_path + "/" + config["search_path"]):
    # For each file in the directory
    for filename in filenames:
        # If the file is in the list of changed files or if the processall argument is True
        if (any(filename in files for files in changed_files) or args.processall):
            print (os.path.join(dirpath, filename))
            # If the file is a metadata file, skip it
            if (re.search('\.metadata$', dirpath + "/" + filename)):
                print ("Skipping file: " + dirpath + "/" + filename)
                continue
            else:
                # Otherwise, open the file and read its contents
                print ("Using file: " + dirpath + "/" + filename)
                with open(dirpath + "/" + filename) as f:
                    post_data = {}
                    post_data["name"] = filename
                    post_data["search"] = f.read().replace('\n', ' ').replace('\r', ' ')
                    post_data["dispatch.earliest_time"] = "-15m"
                    post_data["dispatch.latest_time"] = "now"
                    # If there is a metadata file for this file, open it and load its contents
                    if (os.path.isfile(dirpath + "/" + filename + config["metadata_ext"])):
                        with open(dirpath + "/" + filename + config["metadata_ext"]) as meta_f:
                            metadata = json.load(meta_f)
                            post_data.update(metadata)
                    # Make an API call with the post data
                    make_api_call(config, args, filename, post_data,"search")
```

Automation walkthrough cont.

```
def get_lookup_data(dirpath,filename):
    # Read data from CSV file
    lookup_content = []
    try:
        with open(dirpath + "/" + filename, encoding='utf-8', errors='ignore') as file:
            reader = csv.reader(file)
            for row in reader:
                print(row)
                lookup_content.append(row)

    except Exception as e:
        print("Error reading {} : {}".format(filename,e))


    post_data = {}
    post_data={ "namespace": "search",
            "lookup_file": filename,
            "contents": json.dumps(lookup_content),
            "owner": "nobody" }
    return post_data
```

Automation walkthrough cont.

```
if call_type=="lookup":
    # Lookups are updated using the Lookup Editor App endpoints
    url_string = protocol + "://" + server + ":" + config["port"] + "/" + config["uri_path_lookups_upload"]
    # Lookup permissions uri path
    url_lookup_permissions = protocol + "://" + server + ":" + config["port"] + "/" + config["uri_path_lookups"] + "/"
else:
    # search uri path
    url_string = protocol + "://" + server + ":" + config["port"] + "/" + config["uri_path_search"] + "/"
```

```
try:
    if call_type == "search":
        # Check to see if the saved search already exists
        response = requests.get(url_string + urllib.request.quote(object_name) + "?output_mode=json", auth = requests.auth.HTTPBasicAuth(args.username,args.password), verify=False)
        results = response.text

        # If the search exists, update it
        if str(response.status_code)=="200":
            print ("Search Exists, Updating existing one")
            del post_data["name"]
            response =  requests.post(url_string + urllib.request.quote(object_name) + "?output_mode=json", auth = requests.auth.HTTPBasicAuth(args.username,args.password), verify=False, data=post_data)
            results = response.text

        # If the server response indicates that the search does not exist (HTTP status code 404)
        elif(str(response.status_code)=="404"):
            print ("Search does not exist, creating new one")
            try:
```

Automation walkthrough cont.

```
else:
    print ("Lookup is being added/updated")
    try:
        # Attempt to create/update a lookup by sending a POST request
        response = requests.post(url_string + "?output_mode=json", auth = requests.auth.HTTPBasicAuth(args.username,args.password), verify=False, data=post_data)
        results = response.text
        print (results)
    except urllib.error.HTTPError as e:
        # If an HTTP error occurs, print the error and exit the program
        print ("Unknown Error: " + str(e))
        exit(1)
```

Automation walkthrough cont.

```
# Update permissions
print ("Updating Permissions")
# Open the permissions.json file and load its contents into post_data
with open(dir_path + "/" + "permissions.json") as perm_file:
    post_data = json.load(perm_file)
    if call_type == "lookup":
        # The lookup acl endpoint doesn't not need the app and user parameters so these need to be removed.
        del post_data["app"], post_data["user"]
        response = requests.post(url_lookup_permissions + urllib.request.quote(object_name) + "/acl", auth = requests.auth.HTTPBasicAuth(args.username,args.password), verify=False, data=post_data)
        results = response.text
        print ("Updated Lookup Permissions")
    else:
        # Send a POST request to update the permissions of the search
        response = requests.post(url_string + urllib.request.quote(object_name) + "/acl", auth = requests.auth.HTTPBasicAuth(args.username,args.password), verify=False, data=post_data)
        results = response.text
```

Automation walkthrough cont.

- The python script for dashboards slightly differs by:
  - Processing dashboard XML from the dashboard directory (and subdirectories)
  - Prepares and formats dashboard data for API submission by stripping unnecessary whitespaces and handling special characters
  - The user must account for the XML wrapper for dashboard studio json code
    - Settings > All Configurations
    - Search for dashboard name
    - Click on dashboard name and copy all code including the xml wrapper

Automation walkthrough cont.

```
    "port": "8089",
    "uri_path": "servicesNS/nobody/search/data/ui/views",
    "search_path": "dashboards",
```

```
    with open(dirpath + "/" + filename, encoding="utf-8") as f:
        post_data = {}
        post_data["name"] = filename
        post_data["eai:data"] = f.read().replace('\n', ' ').replace('\r', ' ').replace('\t', ' ')
        logger.info("Post Data: " + str(post_data))
        make_api_call(config, args, filename, post_data)
```

Workflow

- Customers create searches & dashboards in the web UI
- Fork the repo and submit a PR with their updated/new code

- Splunk admins review the PR
- Request changes & assist as necessary
- Merge the PR when appropriate, kicking off the pipeline

Workflow cont.

```
index=windows sourcetype=*perf* cpu OR processor earliest=-5m latest=now
| where cpu_usage_percent>95 OR '% Processor Time'>95
| eval
    cpu_value=coalesce(cpu_usage_percent, '% Processor Time'),
    alert_time=strftime(_time, "%m/%d/%Y %H:%M:%S"),
    severity=if(cpu_value>=98, "Critical", "High"),
    ticket_summary="High CPU Alert: " + host + " at " + round(cpu_value,1) + "%"
| stats
    latest(alert_time) as "Alert Time",
    max(cpu_value) as "Peak CPU %",
    latest(severity) as "Severity",
    latest(ticket_summary) as "Ticket Summary"
    by host
| table host "Alert Time" "Peak CPU %" "Severity" "Ticket Summary"
```

Code | Blame | 18 lines (18 loc) · 574 Bytes

```
{
    "action.script": 1,
    "action.script.filename": "script_name.py",
    "actions": "script,bigpanda_alert",
    "action.keyindicator.invert": "0",
    "action.makestreams.param.verbose": "0",
    "alert.suppress": "1",
    "alert.suppress.period": "1h",
    "alert.track": "1",
    "alert_comparator": "greater than",
    "alert_threshold": "0",
    "alert_type": "number of events",
    "cron_schedule": "5 8 * * 0,2,3,4,5",
    "is_scheduled": "1",
    "disabled": "0",
    "request.ui_dispatch_app": "search",
    "request.ui_dispatch_view": "search"
}
```

```
| rest /servicesNS/-/-/saved/searches
| table title app search description cron_schedule is_scheduled dispatch.earliest_time dispatch.latest_time eai:acl.owner eai:acl.app eai:acl.sharing eai
    :digest disabled actions
```

Benefits of CI/CD automation

- Freedom for customers to self-serve & learn Splunk at their pace
- Greatly reduced development work for Splunk Admins
- Backups of Splunk knowledge objects
- Revision history and rollbacks
- Long-term audit trail

Q/A

Thank you