OPENLATTICE

NCRIC & Astrometrics Data Pipeline and Integration Documentation

This document explains the data integrations and ETL scripts developed by OpenLattice, Inc. for the Northern California Regional Intelligence Center (NCRIC), as part of the Astrometrics application.

OpenLattice data integrations extracted data from several ALPR vehicle data streams, standardized them via our entity data model, and loaded them onto secure cloud-based OpenLattice servers. Here they were separated out into different entity sets (in other words, datasets) according to their source law enforcement agency, and mapped on the website. Agency-specific datasets allowed for permissions to be set at the agency level for Astrometrics users. Integrations achieve this via extract-transform-load (ETL) scripts.

The OpenLattice integration stack will be copied onto Maiveric servers, and the data pipeline replicated. Slight modifications were made to allow integration scripts to be standalone from some Openlattice developed python libraries, but otherwise, this pipeline is identical to what was running in OpenLattice. All scripts have been uploaded to a GitHub repository called ncric-transfer, here. ***NOTE*** *further modifications will be needed once the Maiveric environment is set up, to to use details specific to it such as endpoints, ports, and database names.*

What is in NCRIC data?
The ALPR data fields, at minimum, that we integrate from each data source are:

- License Plate (i.we, ABC1234)
- Timestamp (and time zone)
- Latitude and Longitude
- ALPR identifier/camera (i.e., Patrol1, Main St, NW corner of First and B)
- Agency identifier (i.e., San Francisco PD, Oakland Parking)
- Image (encoded in base64)
- Read identifier (need to distinguish this in the read, and also for the images)

In some data streams, we also receive a camera location. Historically we integrated labels (i.e., blue, sticker, truck) and confidence metrics for a read, but these are not currently part of the pipeline.

Data Sources In Operation

- Boss4
- Santa Clara (SCSO)
- Flock

Boss4 & Santa Clara

Both BOSS4 and Santa Clara ALPR data streams are identical in their delivery methods to OpenLattice, and nearly identical in data structure. The BOSS4 data is located in S3 bucket **sftp.openlattice.com** under **ncric** subfolder. The SCSO data is located in S3 bucket **sftp.openlattice.com** under **ncricscso** subfolder.

NOTE *these bucket names and subfolder names are critical argument inputs in the integration python cron jobs files, and must be changed in the code to whatever bucket names and subfolders are set up on the Maiveric side.*

Data is in a nested JSON file with ~500 data points per JSON file. Each JSON file is named in a similar structure of ncric/READS_OpenLattice_20201026T223918_20201026T224259_609759672.json. We used this filename structure to filter out integrations by date.

*S3 authentication and access***:**

- S3 was only accessed via running the integration python scripts on Rundeck, on the AWS Box. To connect to S3 via python, we use the AWS SDK for python, boto3. Rundeck has proper credentials for S3 already stored, so there's no need for authentication through the boto3 package.
- To inspect any data in the S3 sftp folder, we set up a job on Rundeck to search for and retrieve files

Flock

Flock data is pulled from the Flock Safety API daily, as outlined in this GitHub repository, flapper. Data is pulled early in the morning, and then data integrated over the course of the day given the volume of Flock data.

It is then written to a PostgreSQL database in a table called `flock_reads`, where it is ingested in the cleaning and integration pipeline (details below).Writing to PostgreSQL is part of the code in the FlockSafety flapper repo.

OpenLattice Data Pipeline Design

EDM: OpenLattice uses an entity data model (EDM), stored in a graph database. Here *entity types* represent top-level concepts (e.g., facilities, hospitals, regional offices) and are composed of *property types* that define their structure and characteristics. *Associations* define and specify directional relationships between two entity types, which in relational databases are implied. Therefore, the need for foreign keys and repetitive rows in various tables is eliminated. Expanding the EDM as new concepts are needed is also very easy. We can also easily accommodate any requests for changes in how data is integrated here in the future. OpenLattice takes external data and conceptualizes it in our EDM, mapping fields to various entity types, properties, and associations.

Entity Sets: Entity sets are data sets. Each is an instantiation of a certain entity type or association (i.e., created in their blueprint mold). Each data integration therefore has a data model representing how its data is conceptualized, shown below for the CDSS MyChildCare website.

Flights: In order to extract and load data into our data model, OpenLattice has developed a concept called 'flights' that describes how columns from the original data representation correspond to the OpenLattice data model. These flights are yaml files that specify exactly which original data fields map onto which properties and entity sets once integrated.

Cleaning and Integrating with Pyntegrations: "Pyntegrations" is an OpenLattice python library that stores integration definitions, which each consist of a flight, a cleaning script, and a source sql statement to specify which data to bring in from the staging database. A modified, more standalone version has been created for the NCRIC data called **"pyntegrationsncric"**.

In pyntegrations there are base classes for integration configurations ("Integration"), standardized cleaning ("clean_and_upload"), and integrating ("integrate_table"). There are also reusable functions for accessing one's credentials to the entity sets (one must have at least "read" permissions to integrate successfully). The primary integration and cleaning script is called **integration_definitions.py**, where child classes inherit from the base classes above to load, clean, and integrate data. All needed data cleaning is conducted in Python before the integration step.

NOTE: *Whenever one changes any pyntegrationsncric files (integration_definitions, flight, config file, etc) one must reinstall pyntegrationsncric (run* `python setup.py install` *and remember to check that your virtual environment is active) for those changes to be reflected in the integration. See Appendix 3 for step-by-step details.*

NCRIC Data Pipeline

Data is taken from various places, as described in the section above on data sources. It is loaded into python in-memory where all cleaning and transformations that are needed, occur. Some of the cleaning and transformations required are:

- Standardizing agency names across ALPR data sources, so that counts by agency can be cleanly aggregated
- Converting Boss4 latitude and longitude to decimal degrees
- Filtering out known file errors, such as:
    - Add closing braces and quotations to unterminated strings in BOSS4 json files
    - Filter out bad dates
- Adding in a string tag for the data source (i.e., Flock, SCSO)

Associations and entity sets are integrated in separate flights for efficiency, because most data, once integrated, does not change - e.g. camera names and locations, and agency names. Only new ALPR reads (i.e. " vehicle records") are added, and so we save processing time by avoiding overwriting unchanged data.

There are three separate flights ***for each agency***:

1. Main vehicle records flight
    - Integrates reads, and creates associations from each read to the proper camera and agency
2. images flight
3. Hotlist flight

Several intermediate tables are produced, and written back to the Atlas staging database. Ultimately these are all of the tables that are found on Atlas, and their origins:

Table 1. Tables used in cleaning and integration, written to Postgres.

Name	**Description**
standardized_agency_names	Table that shows all of the agency names that have been found in the data, and their standardization. Every data stream uses this table to: ● filter out data into particular agencies and integrate them into their proper datasets ● Write the standardized agency name with that read's data instead of what was received (for accurate showing and counts in Astrometrics) This table is used in the BOSS4 and SCSO integration jobs.
standardized_agency_names_flock	Same as "standardized_agency_names", except specific to Flock Safety's data stream. This table is used in the Flock integration jobs.
hotlist_daily	The daily list of hotlist plates received from NCRIC, and saved onto postgres.
flock_reads	The raw data from the Flock Safety API puller, then written to Postgres for cleaning and integration.

The NCRIC Data Model

A very simple data model houses all necessary properties into a handful of entity sets. The majority of information is stored in a "vehicle records" dataset. Agencies, cameras, and camera locations rarely change and are stored in separate entity sets that rarely need to be written to. The integrations create associations between vehicle records and these other periphery entity sets. The data model it is basically:

Vehicle records -> recorded by -> Agencies

Vehicle records -> recorded by -> Image Sources (i.e. cameras)

Vehicle records -> results in -> Notifications (hotlist alert)

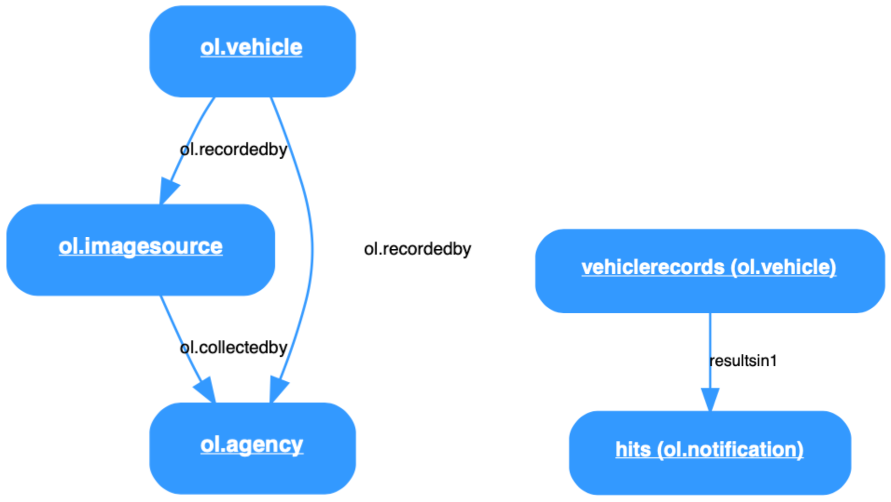


Figure 1. The data model for the main NCRIC flights on the left, and that for the hits flight (run separately), on the right.

A separate flight integrates the daily hotlist of vehicles marked as stolen. Hotlist reads are also stored in a completely separate entity set.

Table 2. Entity types and their IDs from the Openlattice EDM used for ALPR data. There is one entity set *per agency*. See Appendix 1 for the complete list of agency names.

Entity Set Name & ID	Entity Type & ID	Data it contains
NCRICVehicleRecords<agency> (e.g., NCRICVehicleRecordsAthertonPD)	ol.vehicle, `3c6dad54-c4b4-4dfb-bd8b-d8dd56e342ec`	Vehicle records
NCRICImageSources	ol.imagesource, `6b513215-2566-491c-9d08-02a282f4123e`	Cameras
NCRICAgencies (dataset of all received agency names & spellings), NCRICStandardizedAgencies (dataset of standardized agency names)	ol.agency, `e33ad963-60fd-489d-8cdb-9faca522e18a`	Agencies
astrometrics_47b646d7a01a4232b25b15c880ea4046_apphotlistreads	ol.vehicle, `3c6dad54-c4b4-4dfb-bd8b-d8dd56e342ec`	Hotlist reads

4cac0633-b0be-4680-9cd5-26f0bbc2dfc c		
NCRICRecordedBy	ol.recordedby, 9b44a35d-d414-4f7e-929a-92175017b809	"recorded by" association
NCRICCollectedBy	ol.collectedby, 65416b16-626c-495a-9904-a22a4d113276	"collected by" association

Location of scripts:
The module pyntegrationsncric contains all of the cleaning and integration code. This code is found here. Files within are organized as follows.
utils folder:
Scripts with python base classes and helper functions that are used in the main integration and cleaning scripts:

- `utils/integration_base_classes.py`: contains the "Integration" base class which represents an integration configuration.
    - The base Integration object also automatically deserializes the flight to a Flight object
- `utils/flight.py:` a direct copy of parts of the openlattice python library of functions relating to deserializing a flight, put here to enable this repository to be standalone

`utils/openlattice_functions.py`: reusable functions for obtaining authentication credentials
Flight files:

- `ncric_flights` folder = the main flights for each agency's data (1 per agency)
- `ncric_image_flights` folder = the flights for each agency's images (1 per agency)

alpr_cleaning folder:

- integration_definitions_s3.py = main code for grabbing raw data from S3 buckets and the integration definitions for each flight (Boss43 and SCSO). These are inherited in their respective folders below
- integration_definitions_atlas.py = deprecated file, which was used when ALPR data streams were benign written directly to Postgres instead of to S3 buckets
- Pdf files = entity data models
- ncric_hits_flight.yaml = the flight for the hotlist

boss4, flock, scso folders:
Each has an **integration_definitions.py** file which contains the code that is called upon in integration and cleaning jobs (section above) and defines what they do.

Cleaning and Integration Scripts
All scripts are automated to run on a recurring basis, using Rundeck.

There are three jobs in Rundeck that clean and integrate data, and one additional job that runs a python script to delete any data that is no longer relevant. I.e., it could be that there are facilities that are not in the facility universe anymore but once were. Each job has a script which has been written into Rundeck, and scheduled.

Notes:

It is known that ALPR reads could take days or even longer to be recorded on various ALPRR servers, from the time they are recorded (the read datetime) given various factors- offline cameras, network outages in certain areas, etc. For this reason, simply integrating real time reads as they come in will miss some small percentage of the data. Again, this is a known issue and has been confirmed by NCRIC. We deal with this issue by running a second integration a set number of days after the fact to ensure that a higher percentage of data is captured - these are the "asynchronous" jobs in Table 3 below. While not 100% perfect this strikes a balance between effort and returns, as acknowledged by NCRIC. Future pipeline iteration to catch more of the long tail of reads at the extremes of time ranges to reach NCRIC servers, is possible.

Boss4 & SCSO

- The Boss4 and SCSO jobs call upon the "standardized_agency_names" postgres table. This table must be present for the jobs to succeed.
- In Boss4 data, the data stream from Fremont PD sufficiently large that the rest of the pipeline ran more smoothly when it was separated. For this reason Fremont PD is separated into its own automated jobs.
- In Boss4 data, the data stream from the Santa Clara Sheriff's Office is the only agency name with an apostrophe and required a few extra lines of code, which we separated out into its own job to minimize potential failures to the rest of the data.

Flock:

- The Flock jobs call upon the "standardized_agency_names_flock" postgres table. This table must be present for the jobs to succeed.
- For unknown reasons, the Flock integration job scheduled to run at 2 am does not start (could be interference with the Flock data puller being extremely active around that time). A separate job has been created that runs later in the day, and catches all of the data that the 2 am PST job is scheduled to cover (4 - 10 am UTC)

Table 3. Jobs and names (on Rundeck) running pyntegrationsncric python files, used in cleaning and integration

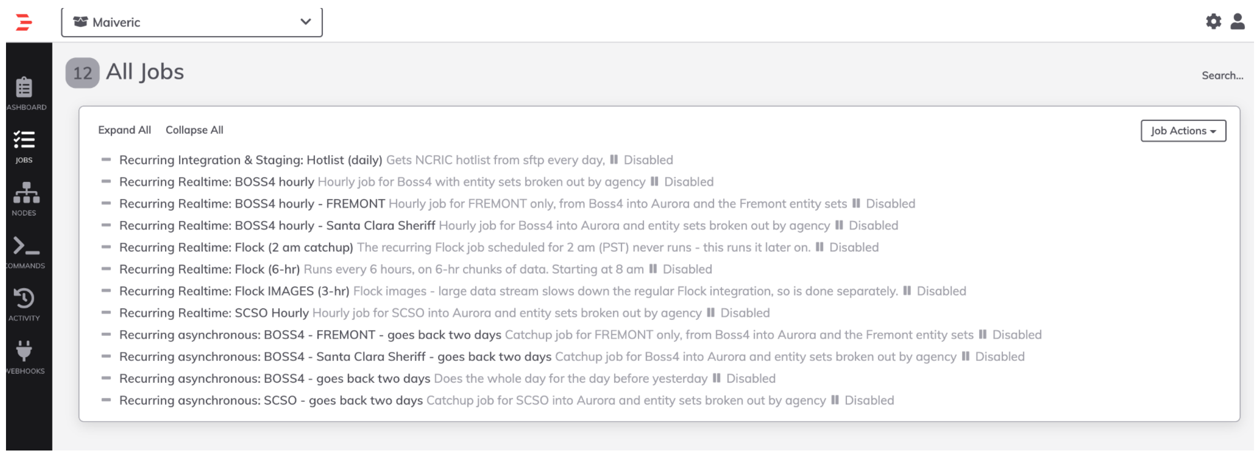
	Job Name	Integration target	Source data	Schedule
1	Recurring Realtime: BOSS4 hourly	Integrates all data *except for* Fremont and Santa Clara Sheriff's agencies.	Boss4 data, from S3 bucket.	Every :10 past the hour (cron 0 10 0 ? * * *)
2	Recurring Realtime: BOSS4 hourly - FREMONT	Integrates all of the data tagged with Fremont PD, from Boss4.	Boss4 data, from S3 bucket.	Every :05 past the hour (cron 0 05 0 ? * * *)
3	Recurring Realtime: BOSS4 hourly - Santa Clara Sheriff	Integrates all of the data tagged with Santa Clara Sheriff's Office, from Boss4.	Boss4 data, from S3 bucket	Every :15 past the hour (cron 0 15 0 ? * * *)
4	Recurring Realtime: Flock (6-hr)	Integrates all data from Flock Safety except images	Flock Safety data (all but images), from the Flock api	Every 6 hours, starting at 8 am.

5	Recurring Realtime: Flock IMAGES (3-hr)	Integrates all images from Flock Safety	Flock Safety data (images only), from the Flock api	Every 3 hours, starting at 0:20 (cron 0 20 0/3 ? * * *)
6	Recurring Realtime: Flock (2 am catchup)	Same as job #4, scheduled for 2 am (PST) which never runs - this runs it later on. - Unclear why it never runs.	Flock Safety data from the Flock api	12 pm Pacific Time, and selects the same time window that the 2 am job should catch - 4-10 am UTC)
7	Recurring Realtime: SCSO Hourly	Integrates all data from the Santa Clara (SCSO) ALPR data stream.	SCSO data, from S3 bucket	Every :10 past the hour (cron 0 10 0 ? * * *)
8	Recurring asynchronous: BOSS4 - FREMONT - goes back two days	Integrates all of the data tagged with Fremont PD, from Boss4 from 2 days ago	Boss4 data, from S3 bucket	4 am Pacific Time
9	Recurring asynchronous: BOSS4 - Santa Clara Sheriff - goes back two days	Integrates all of the data tagged with Santa Clara Sheriff, from Boss4 from 2 days ago	Boss4 data, from S3 bucket	8 am Pacific Time
10	Recurring asynchronous: BOSS4 - goes back two days	Integrates all data *except for* Fremont and Santa Clara Sheriff's agencies, from 2 days ago	Boss4 data, from S3 bucket	1 am Pacific Time
11	Recurring asynchronous: SCSO - goes back two days	Integrates all data from the Santa Clara (SCSO) ALPR data stream from 2 days ago	Boss4 data, from S3 bucket	12:23 am Pacific Time

All integrations are run by a headless account as the "user" who is accessing and writing into the entity sets, this integration user and their password must be first set up. See Appendix x for details. The username is typically an email; the OpenLattice NCRIC integrations user was ncric@openlattice.com.

To check on integration jobs, log into your Rundeck web UI, and click on job of interest (Figure 4). NOTE that before doing this, again, one must:

- Set up Rundeck
- Set up an integration "user" and a Rundeck user (different things)
- Import the job definitions that are left for you in the `rundeck_job_definitions` folder of the `ncric-transfer` repository.



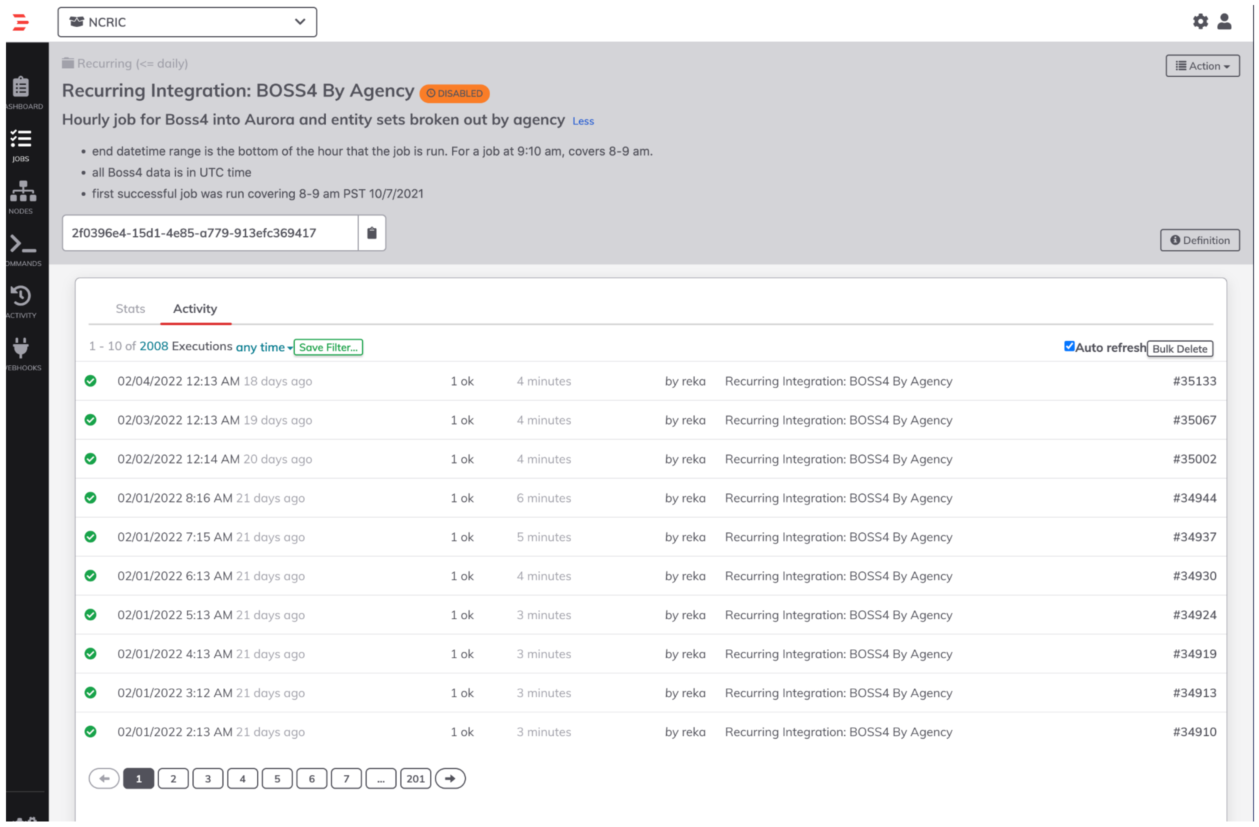
12 All Jobs

Expand All Collapse All

- Recurring Integration & Staging: Hotlist (daily) Gets NCRIC hotlist from sftp every day, Disabled
- Recurring Realtime: BOSS4 hourly Hourly job for Boss4 with entity sets broken out by agency Disabled
- Recurring Realtime: BOSS4 hourly - FREMONT Hourly job for FREMONT only, from Boss4 into Aurora and the Fremont entity sets Disabled
- Recurring Realtime: BOSS4 hourly - Santa Clara Sheriff Hourly job for Boss4 into Aurora and entity sets broken out by agency Disabled
- Recurring Realtime: Flock (2 am catchup) The recurring Flock job scheduled for 2 am (PST) never runs - this runs it later on. Disabled
- Recurring Realtime: Flock (6-hr) Runs every 6 hours, on 6-hr chunks of data. Starting at 8 am Disabled
- Recurring Realtime: Flock IMAGES (3-hr) Flock images - large data stream slows down the regular Flock integration, so is done separately. Disabled
- Recurring Realtime: SCSO Hourly Hourly job for SCSO into Aurora and entity sets broken out by agency Disabled
- Recurring asynchronous: BOSS4 - FREMONT - goes back two days Catchup job for FREMONT only, from Boss4 into Aurora and the Fremont entity sets Disabled
- Recurring asynchronous: BOSS4 - Santa Clara Sheriff - goes back two days Catchup job for Boss4 into Aurora and entity sets broken out by agency Disabled
- Recurring asynchronous: BOSS4 - goes back two days Does the whole day for the day before yesterday Disabled
- Recurring asynchronous: SCSO - goes back two days Catchup job for SCSO into Aurora and entity sets broken out by agency Disabled

Figure 2. Rundeck job list. Shown are a copy of all of the needed jobs that were running on OpenLattice's infrastructure, copied into a Rundeck Project called "Maiveric".

Then click on "Activity" for a list of jobs and their outcomes. Clicking further onto each one will show you the job log.



Recurring (<= daily)

Recurring Integration: BOSS4 By Agency DISABLED

Hourly job for Boss4 into Aurora and entity sets broken out by agency Less

- end datetime range is the bottom of the hour that the job is run. For a job at 9:10 am, covers 8-9 am.
- all Boss4 data is in UTC time
- first successful job was run covering 8-9 am PST 10/7/2021

2f0396e4-15d1-4e85-a779-913efc369417

1 - 10 of 2008 Executions any time

✓	02/04/2022 12:13 AM 18 days ago	1 ok	4 minutes	by reka	Recurring Integration: BOSS4 By Agency	#35133
✓	02/03/2022 12:13 AM 19 days ago	1 ok	4 minutes	by reka	Recurring Integration: BOSS4 By Agency	#35067
✓	02/02/2022 12:14 AM 20 days ago	1 ok	4 minutes	by reka	Recurring Integration: BOSS4 By Agency	#35002
✓	02/01/2022 8:16 AM 21 days ago	1 ok	6 minutes	by reka	Recurring Integration: BOSS4 By Agency	#34944
✓	02/01/2022 7:15 AM 21 days ago	1 ok	5 minutes	by reka	Recurring Integration: BOSS4 By Agency	#34937
✓	02/01/2022 6:13 AM 21 days ago	1 ok	4 minutes	by reka	Recurring Integration: BOSS4 By Agency	#34930
✓	02/01/2022 5:13 AM 21 days ago	1 ok	3 minutes	by reka	Recurring Integration: BOSS4 By Agency	#34924
✓	02/01/2022 4:13 AM 21 days ago	1 ok	3 minutes	by reka	Recurring Integration: BOSS4 By Agency	#34919
✓	02/01/2022 3:12 AM 21 days ago	1 ok	3 minutes	by reka	Recurring Integration: BOSS4 By Agency	#34913
✓	02/01/2022 2:13 AM 21 days ago	1 ok	3 minutes	by reka	Recurring Integration: BOSS4 By Agency	#34910

1 2 3 4 5 6 7 ... 201

Figure 3. Example job activity in Rundeck.

To change the job schedule, click on the "Actions" menu in upper right, choose "edit job", and then go to the "schedule" tab. It is recommended to become familiar with the Rundeck options, which include the workflow (shows the exact script being run), the schedule, optional email notifications, and more.

Code logic:
For each of the flights, Python code is being executed from within a bash script. This code is written directly into each Rundeck job. One can see it under the “Workflow” tab when you are editing a job.

<u>*Flock code:*</u>
We chunk through the reads given large daily volumes. A brief overview of the code (below) is:

1. Starts with a shebang that calls up the proper python virtual environment on your AWS box. Here the python venvironment is called “openlattice”.
2. Import needed libraries
3. Next, each agency found in that ALPR data stream is listed out (from a list of unique agencies in the data thus far) and cleaned/integrated in a loop that iterates over the agency list. The incoming ALPR data streams are filtered out according to their agencies, and each agency’s data is integrated into a separate dataset with the agency name (minus spaces) in the middle of the flight, for instance “`ncric_AthertonPD_flight.yaml`”.
4. Nex, specify the start and end datetimes in which the jobs are run. These differ slightly based on the frequency with which jobs are run. If you want to change to job frequency, change it in the “schedule” tab of the Rundeck job, and also here within the python script.
5. Grab the data for the previous chunk of time (every 6 hrs for Flock) from the `flock_reads` table, standardize the agency name by a join to the `standardized_agency_names_flock` table.
6. Instantiate the integration definition (inherited from an OpenLattice class).   ***NOTE*** *you must insert your base_url here - "https://api.openlattice.com" will no longer exist.*
7. Calculate the number of loops we need to do based on the specified *chunk_size*. The first query in the statement below calculates the number of rows in the just-created cleaned data table, then divides that by the *main_chunk_size* (or *image_chunk_size* in the image integration) to get the total number of times to loop through the data.
8. Loop through the data. The *query* variable changes each iteration to add a different OFFSET so that all the rows in the table are eventually fed into the shuttle. ***NOTE*** *one must also specify the proper path to your copy of shuttle - it will be different than what is shown here.*
9. Drop cleaned tables.

```python
#!/opt/anaconda/envs/openlattice/bin/python

from pyntegrationsncric.pyntegrations.ca_ncric.flock.integration_definitions import FLOCKIntegration
import pyntegrationsncric.pyntegrations.ca_ncric.utils.openlattice_functions as of
import openlattice
import pandas as pd
import sqlalchemy as sq
import math
from datetime import datetime, time, timedelta

agencies = [
    "Atherton PD",
    "CHP",
    "Danville PD",
    "GGB Vista Point",
    "Livermore PD",
    "Napa PD",
    "NCRIC",
    "Piedmont PD",
    "San Mateo PD",
    "Vacaville PD",
    "Vallejo PD"
    ]

# End datetime is the beginning of the hr that the job starts in.
dt_end = datetime.now().replace(microsecond=0, second=0, minute=0) - timedelta(hours = 24)
dt_start = dt_end  - timedelta(hours = 7)

#### Main integration -
for agency in agencies:
    agency_no_space = agency.replace(" ", "")
    flight =
f"/opt/openlattice/pyntegrationsncric/pyntegrations/ca_ncric/ncric_flights/ncric_{agency_no_space}_fligh
t.yaml"

    mainsql=f'''
        select f."readid",f."timestamp", f."type", f."plate", f."confidence",
                f."latitude", f."longitude", f."cameraid", f."cameraname", f."platestate", f."speed",
                f."direction", f."model", f."hotlistid", f."hotlistname", f."cameralocationlat",
                f."cameralocationlon", f."cameranetworkid", f."cameranetworkname",
s."standardized_agency_name" from flock_reads f left join standardized_agency_names_flock s
                on cast(f.cameranetworkid AS TEXT) = s."ol.id"
        where "timestamp" >= '{dt_start}' and
            "timestamp" <= '{dt_end}' and
            s.standardized_agency_name = '{agency}';'''

    print(f"{agency}: Integrating main flight for query {mainsql}.")

    # NOTE you must insert your base_url here - "https://api.openlattice.com" will no longer exist
    integration=FLOCKIntegration(sql=mainsql, flight_path=flight,
                                clean_table_name_suffix='recur',
                                base_url="https://api.openlattice.com")
    clean_table=integration.clean_and_upload()

    ######## Run cleaned table in batches ######
    dbuser=os.environ.get("RD_OPTION_DB_USER") # Store credential within Rundeck "options" above, and
name it "db_user". Retrieve here
    dbpw=os.environ.get("RD_OPTION_DB_PASSWORD") # Store credential within Rundeck "options" above, and
name it "db_password". Retrieve here
    db="org_47b646d7a01a4232b25b15c880ea4046" # Insert your database name here
    host="atlas.openlattice.com" # Insert your hostname here
    eng=sq.create_engine(f'''postgresql://{dbuser}:{dbpw}@{host}:30001/{db}''')

    res=eng.execute(f'''select count(*) from {clean_table};''')
    records = pd.DataFrame(res).loc[0, 0]
    print('''%s records to process''' % (str(records)))
    limit=@option.main_chunk_size@
    its = math.ceil(records/limit)

    for i in range(its):
        query=f'''select * from {clean_table} limit %s offset %s;''' % (str(limit), str(limit*i))
        print(query)
        print(f'''Processing records {str(limit*i)} through {str(limit*(i+1))}''')

        integration.integrate_table(
            sql=query,
            shuttle_path="opt/openlattice/shuttle/shuttle-0.0.4-SNAPSHOT/bin/shuttle",
            drop_table_on_success=False,
            memory_size=4,
            shuttle_args=' --upload-size @option.main_fetch_upload@ --fetchsize
@option.main_fetch_upload@ --read-rate-limit 0 --shuttle-config openlattice-alpha-config us-gov-west-1
--data-store alpr'
        )

    of.drop_table(eng, clean_table)
```

Boss4/SCSO code:
For Boss4 and SCSO data, raw data tables are extracted from S3 buckets and stored in different tables in postgres, one for each integration flight. They are then cleaned and integrated in the same way as Flock data.

This added step is shown in the job code as:

```
# NOTE: CHANGE s3_bucket and s3_prefix to fit the S3 bucket name your files are in
main_integration =
pyntegrations.ca_ncric.scso.integration_definitions.SCSOIntegration(date_start=dt_start_str,

date_end=dt_end_str,
                                                                                         limit=1,

s3_bucket="sftp.openlattice.com",

s3_prefix="ncricscso")
main_table_name, images_table_name = main_integration.get_raw_data_from_s3()
```

Appendix 1: NCRIC agencies

Each of the following 53 agencies was found in the ALPR data that OpenLattice integrated, and reads from these agencies were integrated into separate datasets to allow for assigning separate permissions to them.

Rio Vista PD
GGB Vista Point
HPD
Sunnyvale DPS
Manual Entries
Santa Clara PD
NCRIC
Humboldt County
Dixon PD
Daly City PD
Santa Clara County Sheriff's Office
Alameda County Sheriff
Fremont PD
Livermore PD
Ceres PD

Central Marin Police Authority
San Mateo VTTF
Suisun PD
Atherton PD
Bay Area Rapid Transit

Modesto PD
Yosemite National Park
Brisbane PD
HIDTA
Menlo Park PD
Shasta County Sheriff
Marin CC
Campbell PD
San Bruno PD
Los Altos PD

Sunnyvale PD
Napa PD
Hillsborough PD
Solano County Sheriffs Office
SFO Airport Police
V5Sys
Palo Alto PD
Morgan Hill PD
Benicia PD
NPD
San Mateo PD
South San Francisco PD
San Francisco PD
Danville PD
CHP
Rocklin PD

Piedmont PD	Vallejo PD	Newark PD
San Leandro PD	Vacaville PD	Chico PD

Appendix 2: Setting up the ETL environment

The following steps must be completed on your AWS instance. They are documented here; these instructions can be followed if setup needs to be repeated in the future.

First we set up the python environment by creating a virtual environment and loading needed libraries. This is a one-time occurrence.

- Create a new python virtual environment with `python3 -m venv my_env`. The last argument is the name of your virtual environment, in the above example called "`my_env`".
- Install libpq-dev (needed for the python psycopg2 module) with `sudo apt-get install libpq-dev`
- Before installing new modules, one can check which modules are already installed by logging onto the box, activating your virtual environment with `source my_env/bin/activate` and running `pip list`.
- For any not installed, add their names after `pip install` as written below. These libraries need to be installed:

```
pip install SQLAlchemy pandas DateTime numpy pyyaml uuid pytz
auth0-python wheel geopy psycopg2 pandarallel
```

Then install the Openlattice python API library. We recommend cloning the open-source Openlattice api libraries, found on GitHub, onto the AWS box. Use these apis to generate a jwt token for the integration user, which then allows one to integrate into the entity sets.

If the GitHub repository were installed at `~/openlattice/api-clients:`

- Log in and make sure you are the `openlattice` user with `sudo su - openlattice`
- Navigate to `~/openlattice/api-clients/python`
- Run `pip3 install .`

FInally, install the Openlattice python library for integrations, `pyntegrationsncric` (remember to check that the virtual environment is active). The files here are a direct copy of this github repository.

- Log in and make sure you are the `openlattice` user with `sudo su - openlattice`
- Navigate to wherever you cloned the repository, for instance `../ncric-transfer/pyntegrationsncric`
- Run `pip3 install .`

Appendix 3: Creation of the NCRIC organization and entity sets

To make the organization that organizes the user and entity sets for the MyChildCare website, a POST call must be made to https://api.openlattice.com/datastore/organizations with the following JSON body. ***NOTE*** *the domain https://api.openlattice.com will no longer exist, and one must direct this call to whatever domain one sets up and clones the current Openlattice APIs to.*

***NOTE** please modify the description, principal and title as you wish. The text shown here is for example only.*

```
{
    "apps": [],
    "connections": [],
    "description": "NCRIC Organization",
    "emailDomains": [],
    "grants": {},
    "members": [],
    "principal": {
        "id": " ncricorganization",
        "type": "ORGANIZATION"
    },
    "roles": [],
    "title": "NCRIC Organization"
}
```

To make the entity sets for the NCRIC ALPR integrations, a POST call must be made to the Openlattice datastore entity-sets API - e.g., https://api.openlattice.com/datastore/entity-sets with the following JSON body. ***NOTE** the domain https://api.openlattice.com will no longer exist, and one must direct this call to whatever domain one sets up and clones the current Openlattice APIs to.*

NOTE: One must create an entity set with the specific name of all of the entity sets being referenced in all of the NCRIC flights, or each integration flight will fail. See the “Location of scripts” section on page 6 for details of where to find all of the flights. The first few entity sets from the Alameda County Sheriffs flight are shown here for reference.

- When creating the entity sets, one must reference the proper entity type ID for that entity set, the details of which are given above in Table 3.
- The organizationID must come from the organization that is created in the API call immediately preceding this one, in Appendix 3.
- Name, title, description, entityTypeId, and organizationId are required.

```
[
    {
        "name": "NCRICVehicleRecordsAlamedaCountySheriff",
        "title": "NCRICVehicleRecordsAlamedaCountySheriff",
        "description": "NCRICVehicleRecordsAlamedaCountySheriff",
        "entityTypeId": "3c6dad54-c4b4-4dfb-bd8b-d8dd56e342ec",
        "organizationId": "***",
        "contacts": [],
        "flags": []
    }, {
        "name": "NCRICImageSources",
        "title": "NCRICImageSources",
        "description": "NCRICImageSources",
        "entityTypeId": "6b513215-2566-491c-9d08-02a282f4123e",
        "organizationId": "***",
        "contacts": [],
        "flags": []
    }, {
        "name": "NCRICAgencies",
        "title": "NCRICAgencies",
        "description": "NCRICAgencies",
        "entityTypeId": "e33ad963-60fd-489d-8cdb-9faca522e18a",
        "organizationId": "***",
        "contacts": [],
        "flags": []
    }
]
```