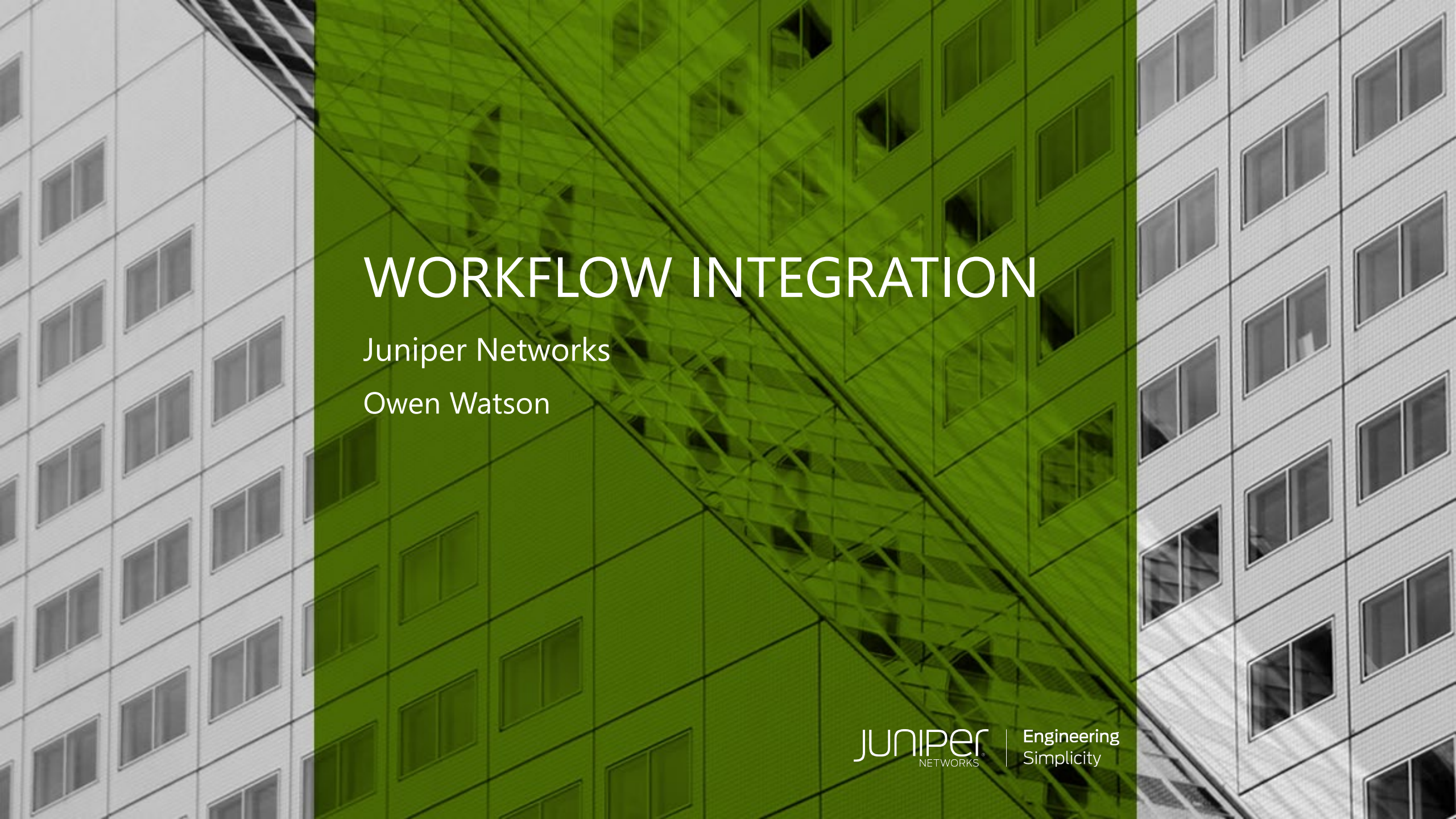

WORKFLOW INTEGRATION

Juniper Networks

Owen Watson

Juniper Networks | Engineering Simplicity

REST API

Workflow Integration with Python

~~BACKGROUND~~ ORCHESTRATION IS PROBLEMATIC!

We have a complex, interlinked set of ‘work flows’ using KNIME, Oozie, Alteryx, etc

These are spread across a variety of systems and services

They depend on each other… but they aren’t linked together

Failures and timing have a negative downstream effect

WHY ME?

My area of interest is the (almost) ‘last mile’

Take the output from other workflows and bring it all together

Long running processes, so successful and current upstream data is essential

Relying on best-guess timing, which is not ideal

INTEGRATE ALL THE THINGS!

KNIME SERVER

Provides a REST API to allow for job control.

Which is great, but:

- There is virtually no documentation for the API…

Uses Mason: https://github.com/JornWildt/Mason

- There are no pre-built libraries to use it…

BRINGING IT ALL TOGETHER

Mason looks good, but is yet another dependency.

As it's JSON based, ignore it and parse it as JSON.

Single Python library that lets you:

Start (A)synchronously / Stop / Monitor jobs and then react accordingly

PYTHON

```
    cleaned_request_kwargs["auth"] = (self.__knime_user, self.__knime_key)

    # --- The 'method' parameter is not used by the requests library, it's only used internally to determine the type of
    # ---  request to make, and so it's value is stored and then removed from the kwargs prior to calling the request library
    if "method" not in cleaned_request_kwargs:
        raise Exception(self.__error_no_req_method)

    request_method = cleaned_request_kwargs["method"]
    del cleaned_request_kwargs["method"]


    try:
        if request_method == self.__request_method_get:
            knime_server_url_contents = requests.get(**cleaned_request_kwargs)
        elif request_method == self.__request_method_post:
            knime_server_url_contents = requests.post(**cleaned_request_kwargs)
        elif request_method == self.__request_method_delete:
            knime_server_url_contents = requests.delete(**cleaned_request_kwargs)

        knime_server_url_contents_status_code = knime_server_url_contents.status_code
    except:
        raise ConnectionError(self.__error_connection_error)

    if knime_server_url_contents_status_code != self.__status_code_ok and knime_server_url_contents_status_code != self.__status_code_ok_no_content and knime_server_url_contents_status_code != self.__status_code_ok_created:
        if knime_server_url_contents_status_code == self.__status_code_access_denied:
            raise AccessDenied(self.__error_access_denied)
        elif knime_server_url_contents_status_code == self.__status_code_not_found:
            raise NotFound(self.__error_no_end_point)
        else:
            raise ConnectionError(self.__error_connection_error)

    return knime_server_url_contents

# --- Starts the workflow specified based on the name and path.
# --- The workflow is started asynchronously, i.e. this function doesn't block until the workflow job is complete.
# --- The specific job ID associated with the workflow is returned if available, otherwise 'None'.
# --- The expectation is that the calling script with periodically check for job status to determine success / failure.
# ---
def __executeWorkflowJob(self, workflow_name, workflow_path=""):
    new_job_url = self.__knime_repository_path  + workflow_path + "/" + workflow_name
```

WORKFLOW INTEGRATION

Include as part of Python script running in Oozie

Start a pre-requisite KNIME job and then wait for completion and fail / continue

Start a tangential workflow asynchronously, letting it run in the background, or pseudo synchronously.

```
from pyknime import *

# Create a new connection to the knime server, noting this doesn't pre-authenticate
ks = Knime(server="http://knimeserver.internal",
           port="8080",
           username="owatson",
           password="password")

# Start task synchronously, i.e. block until the job finishes or the timeout cancels it.
ks_job_details = ks.startWorkflowJob("my_workflow",
                                     "workflow_folder",
                                     async=False,
                                     timeout=900)
# get the ID of the job...
ks_job_id = ks_job_details.keys()[0]

# ...and check whether it succeeded
ks_job_state = ks_job_details[ks_job_id]
```

WAIT… THAT'S ALL?

Pretty much, yeah.

Simple, but critical integration point.

Not big, flashy, or even noticeable but provides a glue to better control our workflows

AVAILABILITY

Currently it's only available internally.

Re-commenting to match Python standards and will be published on GitHub.

Or, email me, and I'll (try and) send you a copy: owatson@juniper.net