Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or the expected performance of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC.

The forward-looking statements made in this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward-looking statements we may make. 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. Splunk undertakes no obligation either to develop the features or functionality described or to include any such feature or functionality in a future release.

splunk> listen to your data

Mike Fettis
@ridingintraffic
https://ridingintraffic.github.io/
https://medium.com/@michael.j.fettis/
https://github.com/ridingintraffic/splunk-cli-did

Splunk as a CLI

and some other weird sh1t

Splunk in the real world

make your choice

The Blue Pill

- Single pain of glass
- Phantom automation
- Gps map visualizations
- Dashboards

Splunk in the real world

make your choice

The Red Pill

- terminals
- scripts and automation
- json
- api-s
- Docker

“Take the blue pill and stay in the fuzzy shiny world of front ends and UIs ”

or

“Take the red pill and follow me down into the real world of DevOps and SRE.”

-- some guy

Splunk good

complicated data
unstructured data
not knowing what you are looking for
the friendly limits the scriptablity

Data and usefulness

you don’t need the viz anymore.

We are here for the data.

Easy in - hard out

Reliant on the application and front end

Don’t leave splunk

Stay here, please.

How?

Make it all modular.

Make it immutable

Make it ephemeral

Treat Splunk as just another tool in the box instead of THE TOOLBOX.

¯_(ツ)_/¯

Why?

When scripting I want the splunk brain

I want the splunk data

I want to leverage splunk into an operator model

Easily plug splunk into playbooks and run books

SRE and OPS

Instead of writing a doc explaining “click here” go to this url…

Checklists

- Run this script
- Check this output
- Feed arguments to this script
- Repeatability!!

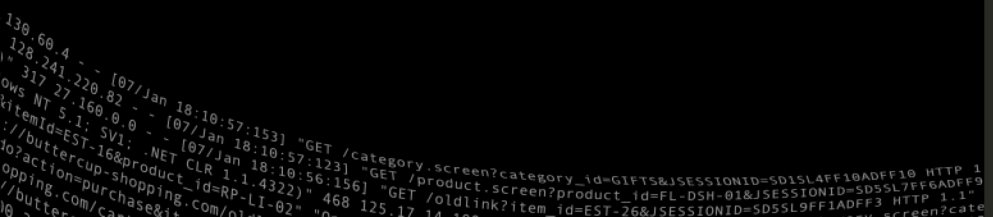
- much easier to follow a checklist at 3am

Step 1: python SDK

accepts search query as argument

- Login to search head
- Run search
- Return table
- Return json

```
from splunklib.binding import HTTPError
import splunklib.client as client

try:
    from utils import *
except ImportError: ...

FLAGS_TOOL = [ "verbose" ]

FLAGS_CREATE = [ ...
]

FLAGS_RESULTS = [
    "offset", "count", "search", "field_list", "f", "output_mode"
]

ENV_VARS = [
    "SPLUNK_USERNAME",
    "SPLUNK_PASSWORD",
    "SPLUNK_HOST"
    ]
ENV_DICT = {}
ENV_DICT.update({"HOST": os.uname()[1]})

for var in ENV_VARS: ...

def cmdline(argv, flags, **kwargs): ...

def main(argv):
    usage = 'usage: %prog [options] "search"'

    flags = []
    flags.extend(FLAGS_TOOL)
```



o your data

Step 2: wrap it in bash

This is where the subtitle goes

Modularize the data source

Splunk as a CLI

takes arguments and returns json

easily churn out scripts for searches

search-example

```
#!/usr/bin/env bash
# search-example: example making use of the search python script
#
# Usage: search  "<splunk spl>" --output_mode=json | jq
#
#
#
# Arguments:
# Environment Variables:
# USERNAME_AD
# PASSWORD_AD
# SPLUNK_HOST

export SPLUNK_USERNAME=$USERNAME
export SPLUNK_PASSWORD=$PASSWORD
export SPLUNK_HOST="splunk.MAH-DOMAIN..com"

if [ -z "$USERNAME" ] || [ -z "$PASSWORDD" ]
then
  echo "one of \$USERNAME \$PASSWORD is empty"
else
  if [ -f /.dockerenv ]; then
      # in docker
      searchlocation="/cmd/search/search"
    else
      #normal world
      searchlocation="./search/search"
    fi
    $searchlocation "| rest /services/server/info | table *" --output_mode=json | jq
fi
```

Step 3:

Just like the rest:

AWS cli has similar bash scripts to get AWS info

Ec2 describe, ASG info s3/rds whatever

Map and form the data via JQ

Data goes in data comes out.

All of it is just scripts and arguments and json blobs.

Read from splunk / Send to splunk

Step 4:
Docker

Docker

Build the dependencies and libraries into a docker container.

Build once hand it to your entire team

Everyone has the same toolset

Everyone has the same syntax

Run books / playbooks operator interface.

Step 5:
MOAR Docker

Wrap the docker container into a bash script

Now the scripts inside of docker get invoked as CLI commands from the host

Docker behaves like a CLI

All the scripts in docker can be written in any language you want

Ruby Bash Python Go

Splunk query scripts look just like postgres query scripts

They all take arguments in, spit json out.

Step 6:
Plug and Chug

If the splunk scripts are accepting arguments then you can take the Postgres scripts or the was scripts that output raw data and feed that directly into splunk scripts.

Splunk query finds an AWS instance that is misbehaving:

- Splunk outputs instance ID
- AWS cli accept instance ID and terminates the instance.

Step 7:
Splunk Docker

Run splunk out of a docker instance.

Fork the container and add a little something extra

Allow Splunk-docker to run docker-in-docker

—but why? —

Step 8:

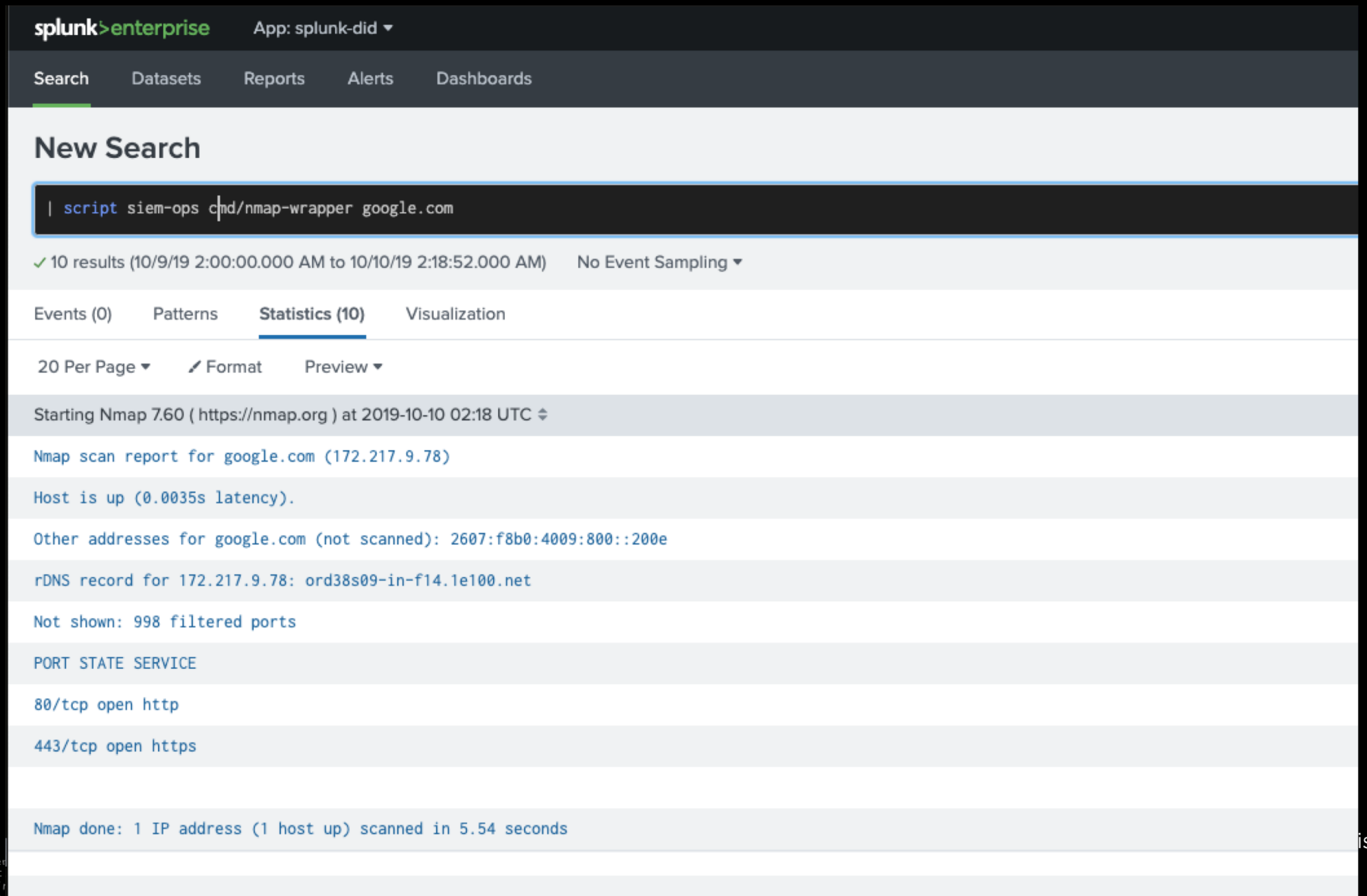
Hijack the script command

From the search box we can invoke a python script to runs the docker container that runs the script

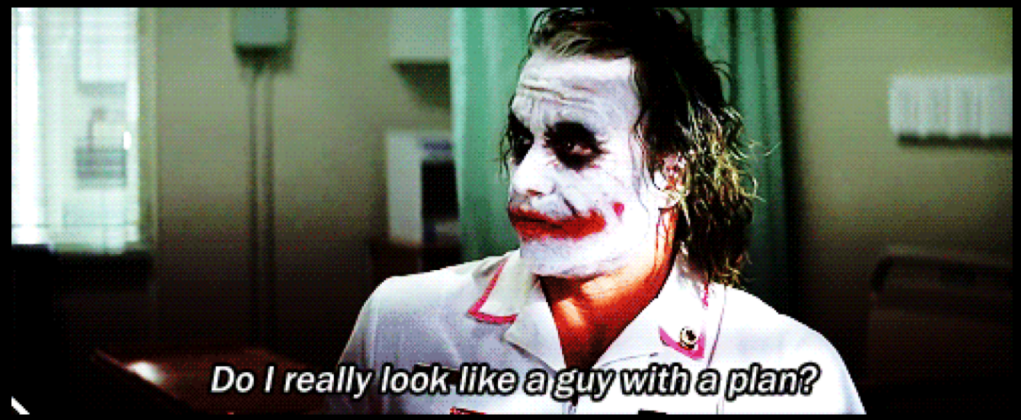
Terminal:

```
✔ ~/git/github/ridingintraffic/splunk-did/Docker [master|✚ 2]
21:18 $ docker run splunker:latest cmd/nmap-wrapper help
cmd/nmap-wrapper:
  usage:
    nmap <host>
  example:
    nmap google.com
✔ ~/git/github/ridingintraffic/splunk-did/Docker [master|✚ 2]
21:18 $ docker run splunker:latest cmd/nmap-wrapper google.com

Starting Nmap 7.60 ( https://nmap.org ) at 2019-10-10 02:18 UTC
Nmap scan report for google.com (172.217.9.78)
Host is up (0.0036s latency).
Other addresses for google.com (not scanned): 2607:f8b0:4009:800::200e
rDNS record for 172.217.9.78: ord38s09-in-f14.1e100.net
Not shown: 998 filtered ports
PORT    STATE SERVICE
80/tcp  open  http
443/tcp open  https

Nmap done: 1 IP address (1 host up) scanned in 11.80 seconds
```

splunk:



Prove it



Thanks

Mike Fettis
@ridingintraffic
https://ridingintraffic.github.io/
https://medium.com/@michael.j.fettis/