The bridge to possible

Do's and Don'ts in Network Test Automation

Oliver Boehmer, CX Principal Architect
@oboehmer

cisco Live!

BRKOPS-2312

Cisco Webex App

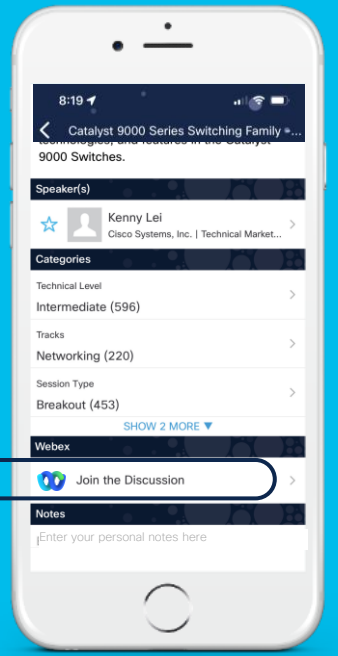
Questions?

Use Cisco Webex App to chat with the speaker after the session

∧ during and

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install the Webex App or go directly to the Webex space
4. Enter messages/questions in the Webex space



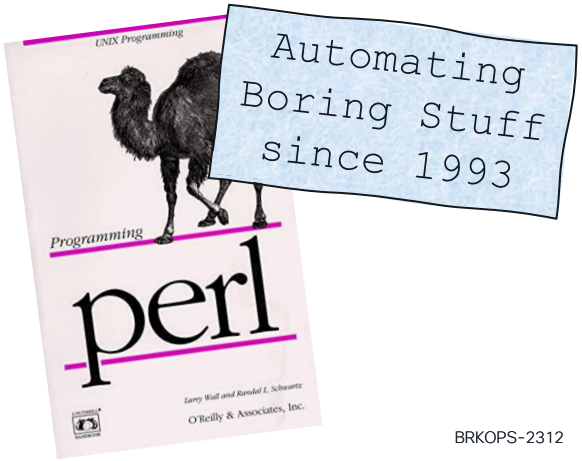
Webex spaces will be moderated until February 24, 2023.

Your Host for the Next 60 Minutes

- Joined Cisco in 2000
- Principal Architect in Cisco Services / CX
- Enterprise & SP Networking Background
- Automation Evangelist
- CX Test Automation Architect
- Active RobotFramework and Genie contributor

CISCO CERTIFIED CCIE 20 YEARS

Cisco DEVNET 500



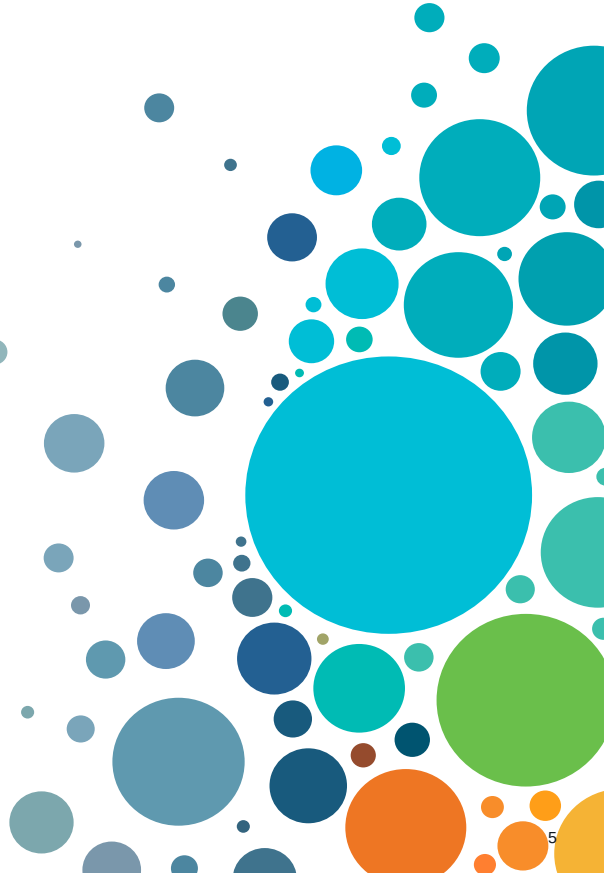
Test Automation is a Journey

We'll Cover a Few Steps Today

Examples @ https://github.com/oboehmer/BRKOPS-2312

Do Not Reinvent The Wheel:

Picking a Suitable Test Automation Framework



Main Functions of a Testing Framework

Running tests, filtering tests, re-running failed tests, etc.

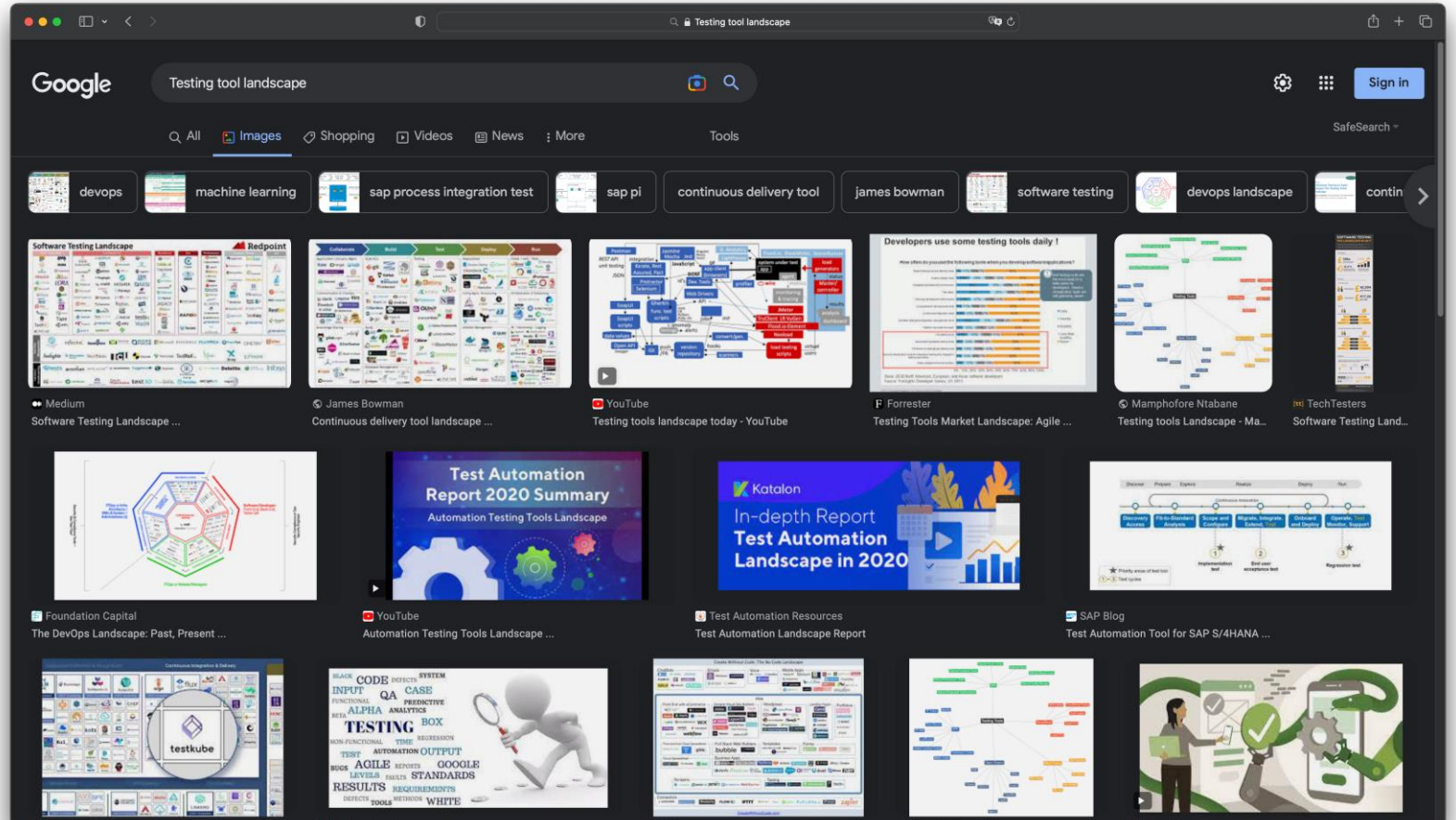
log.html Logging and reporting

import Integrate 3rd party libraries

$var Parameterization

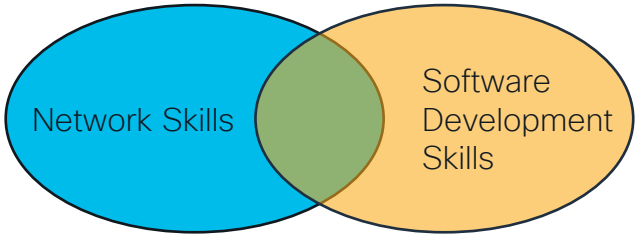
%include Re-use test cases / Macros

Plethora of Different Testing Tools and Frameworks

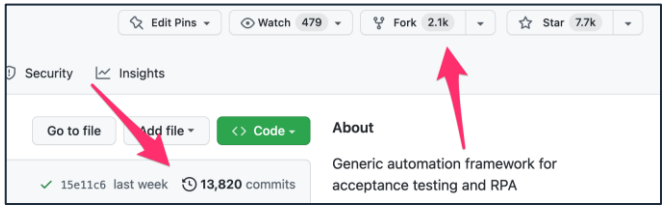


When Selecting a Network Testing Framework …

- Tailor to your audience (aka test case writers) and their skills



- Pick one with healthy ecosystem and community



- Pick one that handles CLI well

```
Router1>
Router1>show foo
```

If You Asked Me, Here Are My Favourites

RobotFramework

ROBOT
FRAME
WORK

- Low-code, Keyword-based framework
- Used by us Cisco Services (but also by Nokia, Juniper and many, many others)
- Large Ecosystem
- https://robotframework.org

If You Asked Me, Here Are My Favourites

RobotFramework

ROBOT FRAME WORK/

- Low-code, Keyword-based framework
- Used by us Cisco Services (but also by Nokia, Juniper and many others)
- Large Ecosystem
- https://robotframework.org

pyATS

- Python based framework
- simple test cases also in YAML
- Used by Cisco Engineering to validate IOS-XE, IOS-XR, NXOS
- https://developer.cisco.com/pyats/

Both freely available

RobotFramework Example

```
*** Settings ***
Library         pyats.robot.pyATSRobot
Library         unicon.robot.UniconRobot
Library         genie.libs.robot.GenieRobot
Suite Setup     setup test environment

*** Test Cases ***
Verify OSPF Neighbor
    ${result}=  parse "show ip ospf neighbor GigabitEthernet2 detail" on device "
    check neighbor id       ${result}    10.0.0.2
    check neighbor state    ${result}    full

*** Keywords ***
setup test environment
    use testbed "testbed.yaml"
    connect to device "r1"

check neighbor id
    [Arguments]    ${result}    ${id}
    ${nbr_id}=     dq query    data=${result}    filters=contains('neighbors').get_values('neighbor_router_id')
    Should be Equal    ${nbr_id}[0]    ${id}

check neighbor state
    [Arguments]    ${result}    ${state}
    ${nbr_state}=  dq query    data=${result}    filters=contains('neighbors').get_values('state')
    Should be Equal    ${nbr_state}[0]    ${state}
```

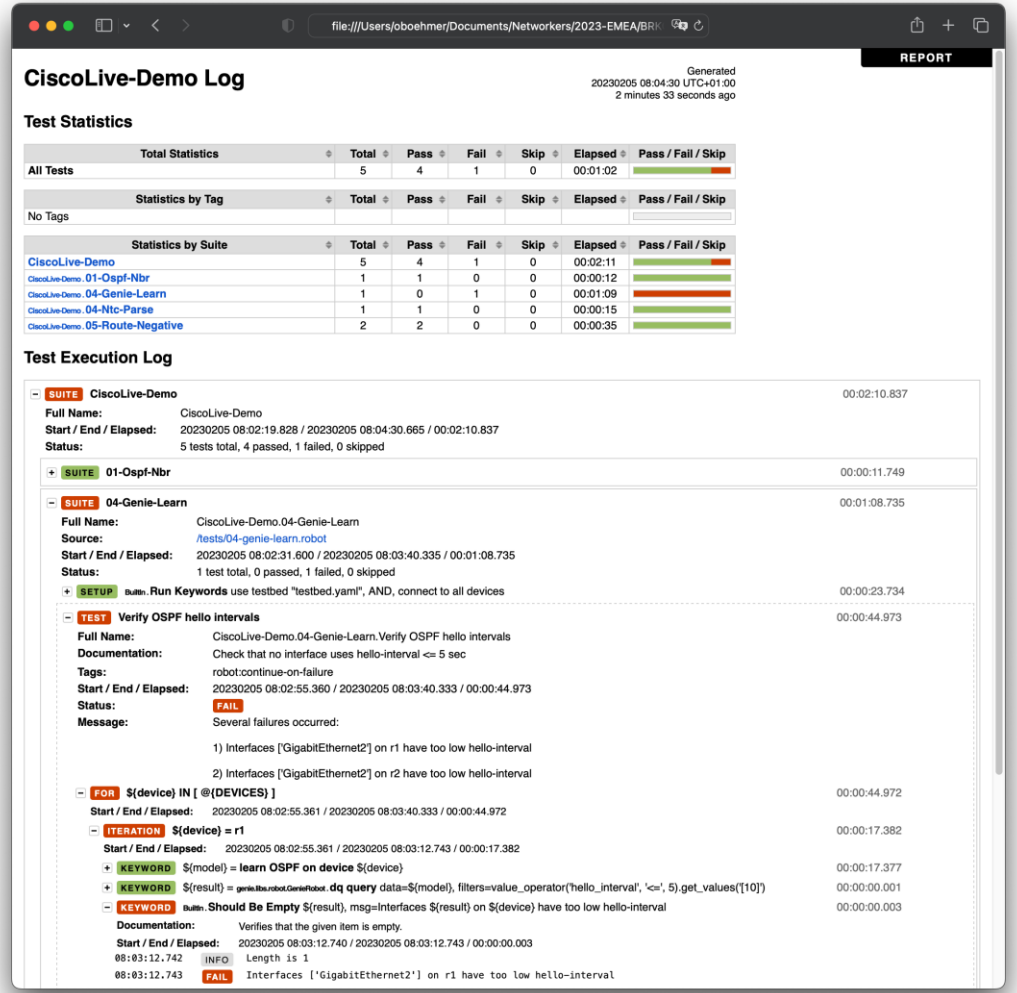
Import Keyword Libraries
(essentially python class whose methods are exposed as keywords)

A Test Case passes if all its keywords pass

Robot keywords for repeated steps or to hide complexity from test case definition

RobotFramework Logging

- Powerful logging capability
- All execution logs also available as XML



https://oboehmer.github.io/BRKOPS-2312/

pyATS Example – aetest (python)

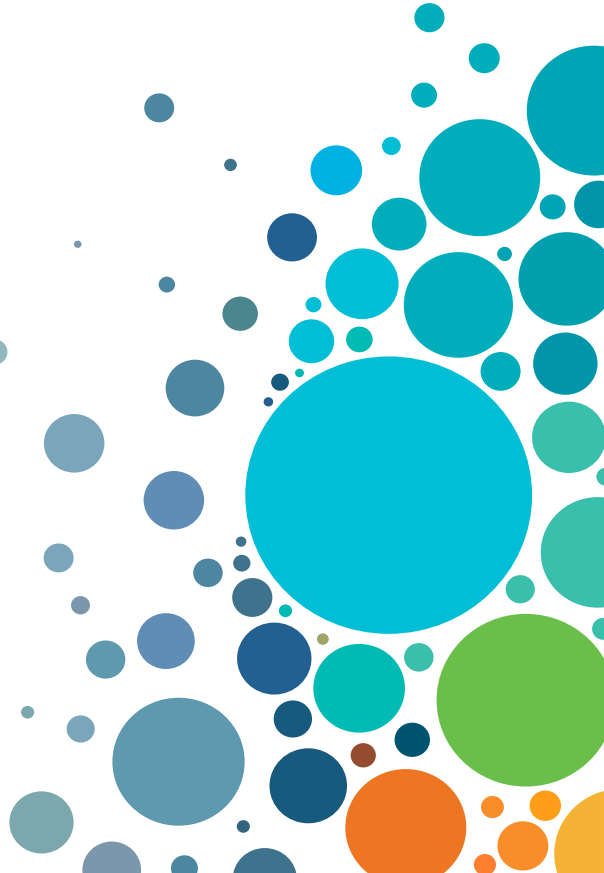
```python
from pyats import aetest


class CommonSetup(aetest.CommonSetup):

    @aetest.subsection
    def connect(self, testbed):
        testbed.devices.r1.connect()
        self.parent.parameters['uut'] = testbed.devices.r1


class VerifyOspfNeighbors(aetest.Testcase):

    @aetest.test
    def verify_ospf_neighbors(self, uut):
        output = uut.parse('show ip ospf neighbor GigabitEthernet2 detail')
        neighbor_rid = output.q.contains('neighbors').get_values('neighbor_router_id')[0]
        if neighbor_rid != '10.0.0.2':
            self.failed(f'Neighbor router id {neighbor_rid} is unexpected')
        neighbor_state = output.q.contains('neighbors').get_values('state')[0]
        if neighbor_state != 'full':
            self.failed(f'Neighbor state {neighbor_state} is unexpected')
        self.passed(f'Neighbor router {neighbor_rid} and state {neighbor_state} as expected')


class CommonCleanup(aetest.CommonCleanup):

    @aetest.subsection
    def subsection_cleanup_one(self, testbed):
        testbed.disconnect()
```

pyATS Example – Blitz (yaml)

```yaml
# pyats Genie Blitz example
#
# run this via
#
#   pyats run genie --trigger-datafile 03-ospf-nbr-blitz.yaml --trigger-uids "Or('TestOspfNeighbors')" \
#                   --testbed-file testbed.yaml

TestOspfNeighbors:
  source:
    pkg: genie.libs.sdk
    class: triggers.blitz.blitz.Blitz

  test_sections:
    - test_ospf_neighbors:
      - parse:
          device: r1
          command: show ip ospf neighbor GigabitEthernet2 detail
          include:
          - contains('neighbors').contains('10.0.0.2').contains_key_value('neighbor_router_id', '10.0.0.2')
          - contains('neighbors').contains('10.0.0.2').contains_key_value('state', 'full')
```

DUT? What's a DUT?

Dealing with your (physical) Testbed

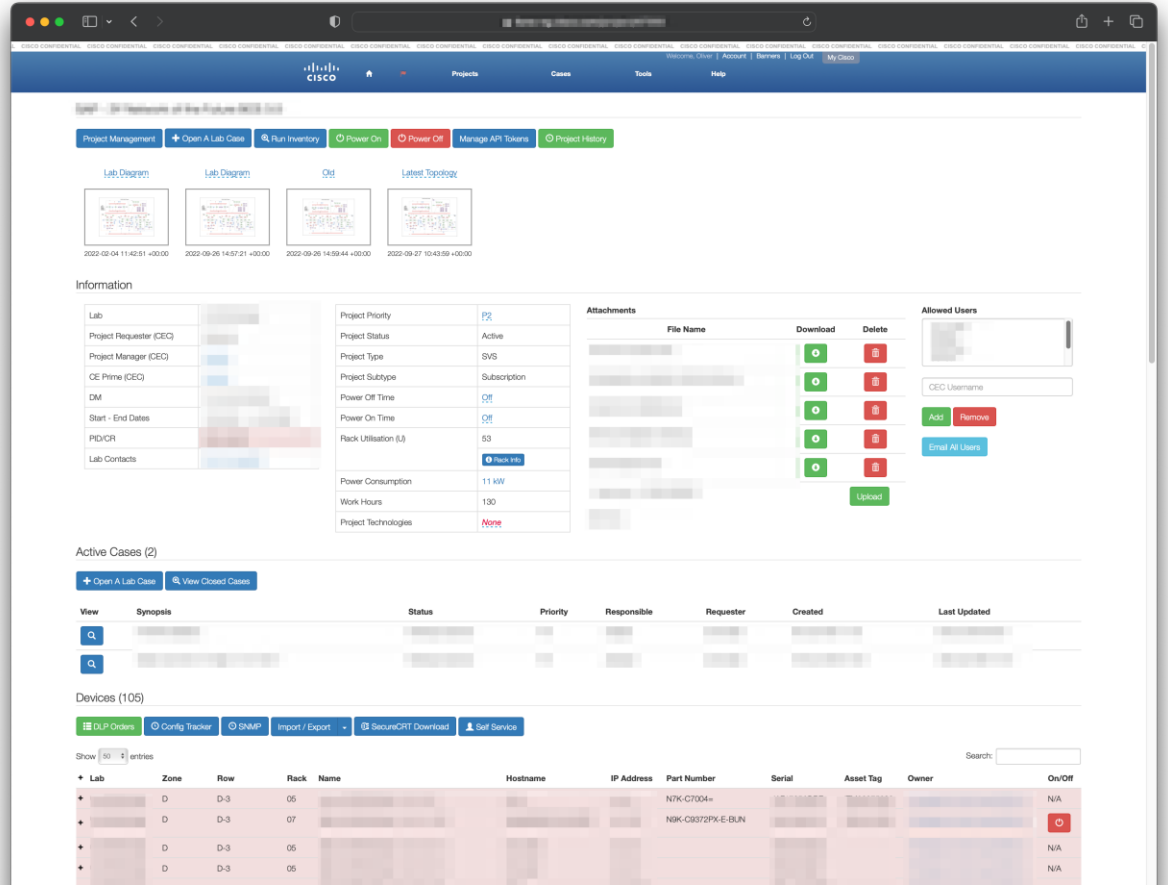


cisco Live!

*) Device Under Test

Lab Device Inventory Management

- Not an easy task, but super-helpful for
  - Central repository
  - Lab Documentation
  - Device Reservation
  - Remote power-on/off
  - Scheduled power-off out-of-hours/weekends



Captain Obvious: Don't Hardcode Topology Info Within Your Test Cases

2

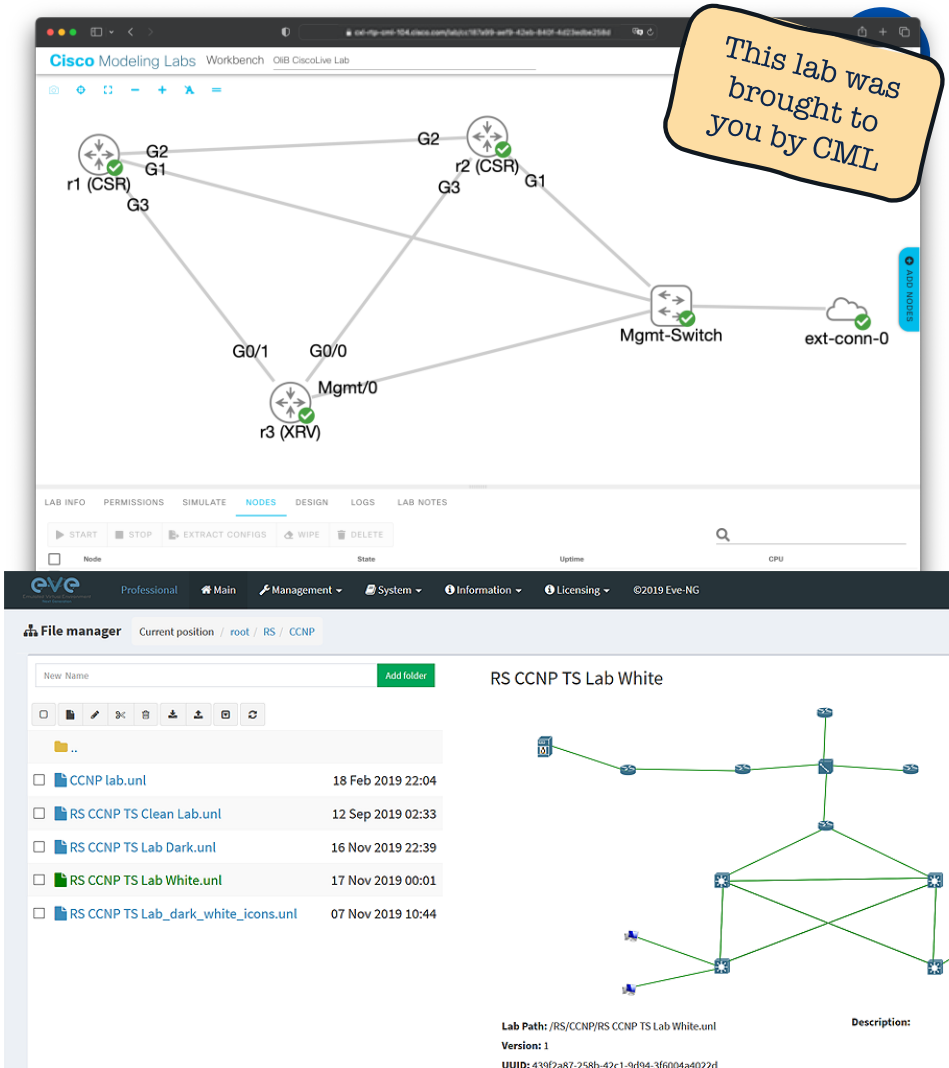
- All decent test frameworks support some form of external parameter definition
- Use it!
- Allows you to easily port your tests to a different environment
- Testbed.yaml (as shown here) required for pyATS

```yaml
testbed:
  credentials:
    default:
      username: cldemo
      password: "%ENV{DEVICE_PASSWORD}"

# Please note that the device names (i.e. r1, r2, etc. in below example)
# need to match the actual hostname configured on the device!!!
devices:
  r1:
    os: iosxe
    connections:
      cli:
        protocol: ssh
        ip: 10.12.63.11
  r2:
    os: iosxe
    connections:
      cli:
        protocol: ssh
        ip: 10.12.63.12
  r3:
    os: iosxr
    credentials:
      default:
        username: cisco
        password: cisco
    connections:
      cli:
        protocol: ssh
        ip: 10.12.63.13
```

Full pyATS Testbed Schema:
https://pubhub.devnetcloud.com/media/pyats/docs/topology/schema.html

Physical or Virtual Devices?

- Virtualisation rulez!
- Throw in a couple of “real” devices as device-under-test (DUT)
- Scale/stress testing typically demands physical kit



Prepare Your Testbed

- Don’t assume “your” routers are in the state you left them
- Set them to a well-defined state
- Verify the state before starting tests

pyATS
print(network.profile)

“Kleenex”

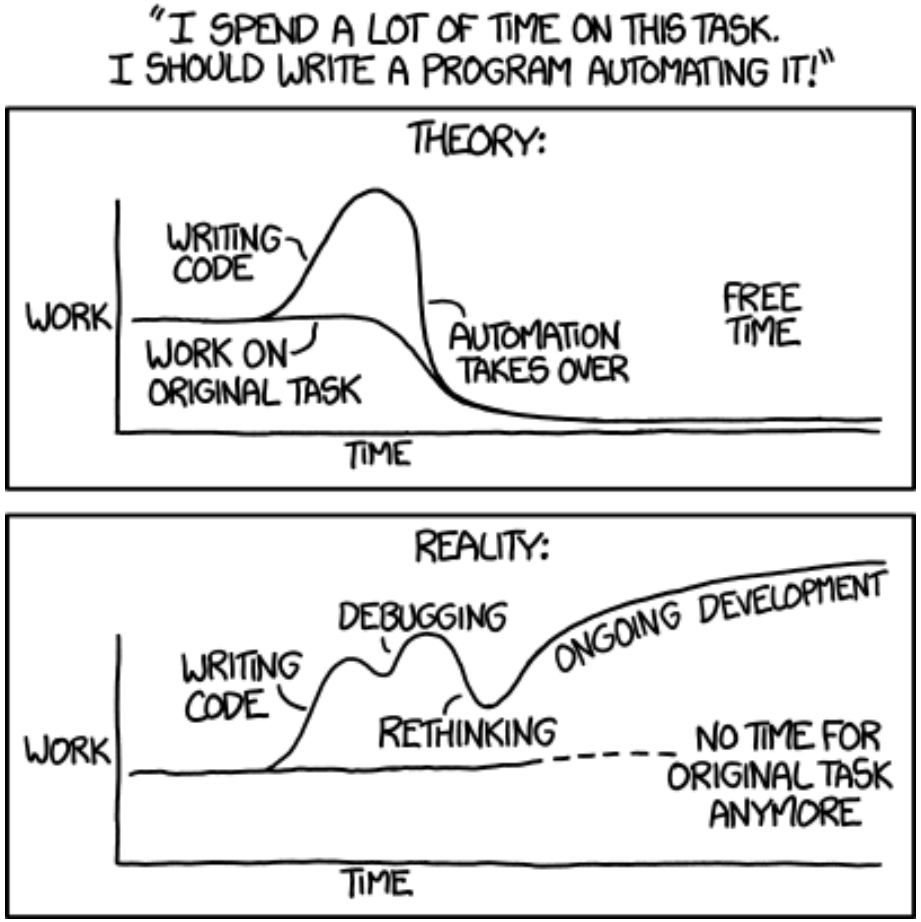
```yaml
cleaners:
  PyatsDeviceClean:
    module: genie.libs.clean
    groups: [WAN]

groups:
  WAN:
    devices: [router1, router2]
    order:
      - connect
      - copy_to_device
      - change_boot_variable
      - write_erase
      - reload
      - apply_configuration
      - verify_running_image
      - ping_server

    ####################

    connect:
      via: a

    ping_server:
      server: 10.2.0.254
      vrf: Mgmt-intf

    write_erase:

    copy_to_device:
      origin:
        hostname: ftp
        files:
          - isr4300-universalk9.16.06.05.SPA.bin
      destination:
        directory: 'bootflash:'
      protocol: ftp
      overwrite: True
      verify_num_images: False
      check_file_stability: True
      min_free_space_percent: 50
      vrf: Mgmt-intf

    change_boot_variable:
      images:
        - bootflash:/isr4300-universalk9.16.06.05.SPA.bi

    verify_running_image:
      images:
        - bootflash:isr4300-universalk9.16.06.05.SPA.bin

    reload:
      reconnect_via: a
      check_modules:
        check: False

    # apply basic configuration to startup, will be define
    # devices further down
    apply_configuration:

devices:
  router1:
    apply_configuration:
      configuration_from_file: configs/r-1.txt
  router2:
    apply_configuration:
      configuration_from_file: configs/r-2.txt
```

pyATS Clean Documentation:
https://pubhub.devnetcloud.com/media/genie-docs/docs/clean/index.html
Examples: https://github.com/CiscoTestAutomation/examples/blob/master/clean

Finally!

Writing *better* network device/ service tests

Been There, Done That???



https://xkcd.com/1319

- Start with things you do very often!
- Be pragmatic in your approach, network testing is no software-development contest
- Always prefer APIs over Web User Interfaces or CLI
- I feel a low-code approach with a suitable test framework helps to avoid this trap

Plan and Organize Your Test Cases

- Get an overview of your complete test plan to identify repetitive tasks
- Define and maintain your own library of re-usable tests / keywords
- Keep test cases short and concise, when possible
- Build self-contained tests: Avoid dependencies between test cases
- Maintain your test cases in Git, treat them like code (with pipeline, linter, dry-run, etc.)

```
# Sample directory structure for our projects
├── data
│   ├── common.yaml
│   └── topology
│       └── testbed.yaml
├── jobfiles
│   ├── asr9010
│   │   ├── 1.1_Router_Management
│   │   │   ├── 1.1.1.1_Initial_Hardware_ROMMON.robot
│   │   │   ├── 1.1.1.2_Software_Verification.robot
│   │   │   ├── …
│   │   ├── 1.2_Functional
│   │   │   ├── 1.2.1.3_Netflow_MPLS.robot
│   │   │   ├── 1.2.1.4_Netflow_IPv4.robot
│   │   │   ├── …
│   │   ├── 1.4_Network_Services
│   │   │   ├── …
│   │   ├── 1.5_Resiliency_and_HA
│   │   │   ├── …
│   │   ├── 1.6_Service_Availability
│   │   │   ├── …
│   │   └── 1.7_Stability
│   │   │   ├── …
└── resources
    ├── CX_Baseline_Robot
    │   ├── cx_baseline_robot.resource
    │   ├── keywords
    │   │   ├── baseline_bfd.resource
    ├── CX_SP_Robot
    │   ├── cx_sp_robot.resource
    │   ├── keywords
    │   │   ├── iosxr
    │   │   │   ├── compare_resources
    │   │   │   │   ├── compare_Parsed.py
    │   │   │   │   └── iosxr_compare_parsed.yaml
    │   │   │   ├── iosxr_bfd.resource
    └── project.resource
```

Parsing CLI Output

3

- Leverage pyATS Genie parsers when parsing command output
- Alternatively, Network-to-Code's (NTC) Texfsm templates*) are quite powerful too
  - Example in git repo (04-ntc-parse.robot)

04-genie-learn.robot

```
*** Settings ***
Library         pyats.robot.pyATSRobot
Library         unicon.robot.UniconRobot
Library         genie.libs.robot.GenieRobot
Library         Collections
Suite Setup     Run Keywords    use testbed "testbed.yaml"    AND    connect to all devices

*** Variables ***
&{OSPF_MODELS}                      # will contain parsed OSPF models from all devices
@{DEVICES}        r1    r2    r3      # r1,r2 are iosxe, r3 is iosxr device

*** Test Cases ***
Verify OSPF hello intervals
    [Documentation]    Check that no interface uses hello-interval <= 5 sec
    [Tags]    robot:continue-on-failure

    FOR    ${device}    IN    @{DEVICES}
        ${model}=     learn OSPF on device    ${device}
        ${result}=    dq query    data=${model}
        ...                       filters=value_operator('hello_interval', '<=', 5).get_values('[10]')

        Should Be Empty    ${result}    msg=Interfaces ${result} on ${device} have too low hello-interval
    END

*** Keywords ***
learn OSPF on device
    [Arguments]    ${device}
    ${result}=    learn "ospf" on device "${device}"
    # convert result to qdict.. a bit ugly
    ${qdict}=     Evaluate    genie.conf.base.utils.QDict($result.to_dict())
    Log Dictionary    ${qdict}
    RETURN    ${qdict}
```

*) https://github.com/networktocode/ntc-templates

Testing Negative Scenarios

- Testing negative scenarios is very common
- Often you can easily turn a positive test into a negative one

3

05-route-negative.robot

```
*** Settings ***
Library         pyats.robot.pyATSRobot
Library         unicon.robot.UniconRobot
Library         genie.libs.robot.GenieRobot

Suite Setup     Run Keywords    use testbed "testbed.yaml"    AND    connect to all devices

*** Variables ***
${edge_device}        r3
${core_device}        r1
${route_positive}     10.100.99.0/24
${route_negative}     192.168.99.0/24

*** Test Cases ***
Positive route distribution test
    ${cmd}=    Set Variable    router static address-family ipv4 unicast ${route_positive} Null0
    configure "${cmd}" on device "${edge_device}"
    Sleep    1
    validate route on device    ${core_device}    ${route_positive}
    [Teardown]    configure "no ${cmd}" on device "${edge_device}"



*** Keywords ***
validate route on device
    [Arguments]    ${device}    ${prefix}
    ${output}=      parse "show ip route ospf" on device "${device}"
    ${routes}=      dq query    data=${output}    filters=contains('${prefix}')
    Should Not Be Empty    ${routes}    msg=Route not found
```

Testing Negative Scenarios

- Testing negative scenarios is very common
- Often you can easily turn a positive test into a negative one

05-route-negative.robot

```
*** Settings ***
Library         pyats.robot.pyATSRobot
Library         unicon.robot.UniconRobot
Library         genie.libs.robot.GenieRobot

Suite Setup     Run Keywords    use testbed "testbed.yaml"    AND    connect to all devices

*** Variables ***
${edge_device}        r3
${core_device}        r1
${route_positive}     10.100.99.0/24
${route_negative}     192.168.99.0/24

*** Test Cases ***
Positive route distribution test
    ${cmd}=    Set Variable    router static address-family ipv4 unicast ${route_positive} Null0
    configure "${cmd}" on device "${edge_device}"
    Sleep    1
    validate route on device    ${core_device}    ${route_positive}
    [Teardown]    configure "no ${cmd}" on device "${edge_device}"

Negative route distribution test
    ${cmd}=    Set Variable    router static address-family ipv4 unicast ${route_negative} Null0
    configure "${cmd}" on device "${edge_device}"
    Sleep    1
    Run Keyword and Expect Error    Route not found
    ...    validate route on device    ${core_device}    ${route_negative}
    [Teardown]    configure "no ${cmd}" on device "${edge_device}"

*** Keywords ***
validate route on device
    [Arguments]    ${device}    ${prefix}
    ${output}=      parse "show ip route ospf" on device "${device}"
    ${routes}=      dq query    data=${output}    filters=contains('${prefix}')
    Should Not Be Empty    ${routes}    msg=Route not found
```

Sweet Spot for Automation: Convergence / Failure Tests

3

- Pain to do without automation help
- Some actions might require someone walking down to pull cables/linecards/etc.
- “Proper” Traffic Generators help, but simple ping on adjacent Linux VM also do good

```
*** Test Cases ***
Reload of A9K-24X10GE-1G-SE Linecard
    [Tags]    DUT=${DUT_1}    Test_type=Resilience    Disruptive=Yes
    clear snmp and syslog logs
    Step 1 - Start Ixia traffic and confirm no traffic drops
    Step 2 - Check status before down event
    Step 3 - Down event
    Step 4 - Check status after down event and before up event
    Step 5 - Get traffic stats from Ixia after down event
    Step 6 - Reset traffic statistics and confirm no traffic drops
    Step 7 - Up event
    Step 8 - Check status after up event
    Step 9 - Get traffic stats from Ixia after up event
    Step 10 - Reset traffic statistics and confirm no traffic drops
    Compare parsed outputs from before down event (step 2) and after
    Compare parsed outputs from before down event (step 2) and after
```

Full test in 06-Convergence-Reload_of_LC.robot

Wrapping Up

Key Take-Aways

- An **appropriate** testing framework for your organization makes ALL the difference

- Leverage virtualization
- Clean your test bed to ensure it is in a defined state

- Focus on repetitive task first
- Create re-usable test cases / test primitives
- Re-use existing parsers

Further Reading / Playing / Testing

- All examples from this session: https://github.com/oboehmer/BRKOPS-2312
- RobotFramework
  - Docs: https://docs.robotframework.org/
  - How to Write Good Test Cases: https://github.com/robotframework/HowToWriteGoodTestCases/blob/master/HowToWriteGoodTestCases.rst
  - Some sample robot tests from previous CiscoLive events: https://oboehmer.github.io/CLEUR20-CXTA/ https://developer.cisco.com/learning/labs/devwks-2155
- pyATS
  - Docs: https://developer.cisco.com/docs/pyats/
  - Many examples: https://github.com/CiscoTestAutomation/examples
  - DevNet Learning Labs:
    - Intro to pyATS: https://developer.cisco.com/learning/labs/intro-to-pyats
    - Introduction to NetDevOps using pyATS: for Non-programmers (DEVWKS-1749): https://developer.cisco.com/learning/labs/devwks-1749m/
    - Writing Reusable Libraries (Parsers/APIs) using pyATS (DEVWKS-3280): https://developer.cisco.com/learning/labs/devwks-3280m/

Learn More on Test Automation at CiscoLive 2023

	Tuesday	Wednesday	Thursday
08:30	Automated Testing for your Network LTRATO-2001, 4 hours		Test Automation for everyone using CXTA LTROPS-1964, 4 hours
14:00	You are here ☺ BRKOPS-2312 Significance of CX Test Automation Framework for Validating SP Technology DEVNET-2959, 45 mins		Robot Framework – Automated Cisco NSO Testing TSCSPG-2011, 30 mins
16:00		Building your Certification Test Suites using pyATS DEVNET-2744, 45 mins	

Continue Your Education

Visit the Cisco Showcase for related demos.

Book your one-on-one Meet the Engineer meeting.

Attend any of the related sessions at the DevNet, Capture the Flag, and Walk-in Labs zones.

Visit the On-Demand Library for more sessions at ciscolive.com/on-demand.

Complete your Session Survey

- Please complete your session survey after each session. Your feedback is important.
- Complete a minimum of 4 session surveys and the Overall Conference survey (open from Thursday) to receive your Cisco Live t-shirt.

- All surveys can be taken in the Cisco Events Mobile App or by logging in to the Session Catalog and clicking the "Attendee Dashboard" at https://www.ciscolive.com/emea/learn/sessions/session-catalog.html


cisco
The bridge to possible
Thank you
cisco Live!


CISCO Live!
ALL IN