tnc21 Making Waves 21-25 June 2021

GÉANT

Experiences in designing and operating an automated CI/CD pipeline for the AmLight SDN orchestrator

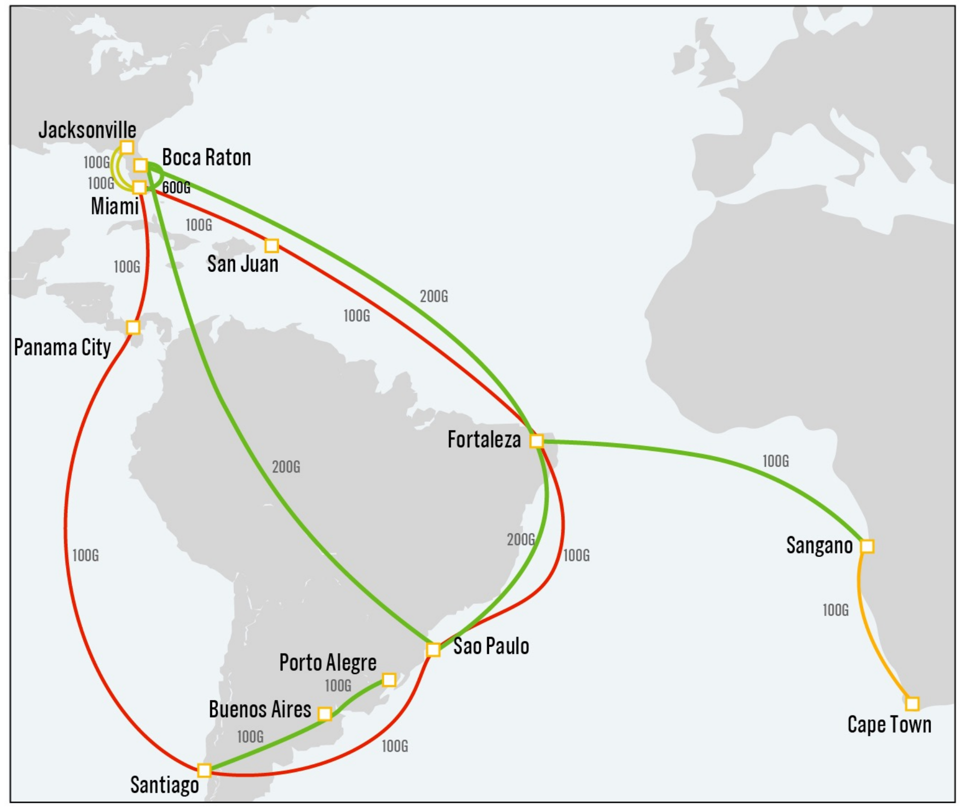
Italo Valcy da Silva Brito <italo@amlight.net>

Miami, FL, US

June 21st, 2021

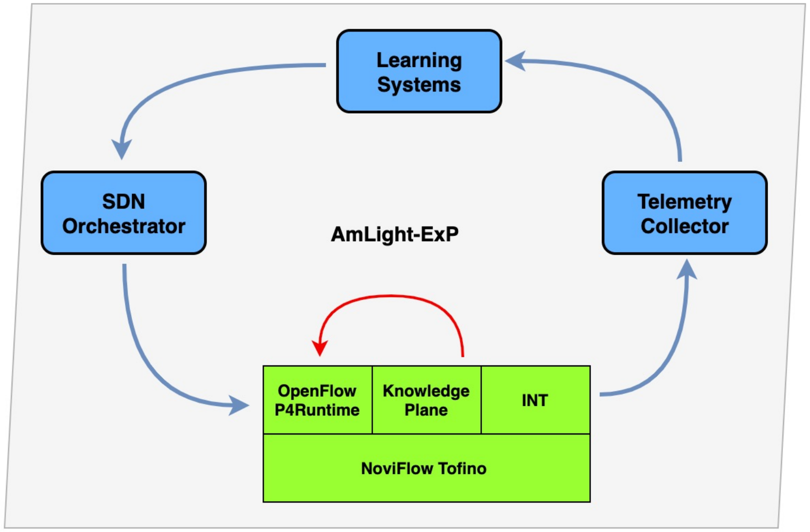
Context: AmLight-ExP 2021 Network Topology

- AmLight-ExP network, +600Gbps
- Open Exchange Points: Miami, Fortaleza, Sao Paulo, Santiago, Cape Town
- Flexible SDN infrastructure since 2014
  - Programmable control-plane
  - Slicing
  - L2VPN services
  - In-band Network Telemetry
  - Pursuing programmable data-plane



SDN Orchestrator

- Major Goal: Improving Resiliency, Increasing Flexibility and Self-Management
  - SDN Orchestrator plays an important role
- Providing a Closed-Loop Orchestration
- Developing and Deploying a new SDN orchestrator to operate AmLight network:
  - User requirements not easily supported by “commodity” NOS and SDN Controllers
  - Pathfinding application handling unusual metrics (reliability, ownership, max delay)
  - Granular network telemetry per flows, per protocols, etc
  - Increased capacity to innovate
  - **Critical infrastructure**



Kytos-ng

- The AmLight SDN Orchestrator is being built over the Kytos-ng SDN Platform.
  - https://kytos.io
  - https://kytos-ng.github.io

- Features are implemented through Kytos Napps (Network applications) - https://napps.kytos.io
  - Based on the concept of micro-services

- Kytos-ng SDN Platform is possible thanks to the support provided by Rednesp (Research and Education Network of Sao Paulo), AMPATH/AmLight team, and by the Kytos open source community.

Problem statement

- How do we continuously update/deploy the SDN Orchestrator, providing an agile service delivery model for the users and operators in a integrated, secure, reliable and seamless manner?

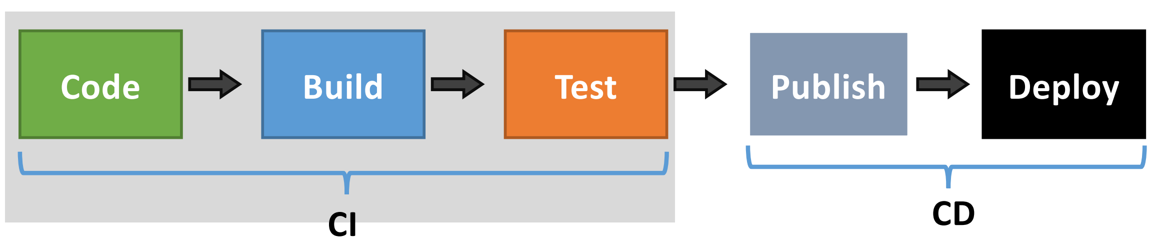
Problem statement

- How do we continuously update/deploy the SDN Orchestrator, providing an agile service delivery model for the users and operators in a integrated, secure, reliable and seamless manner?

    - Can we benefit from a *continuously delivery pipeline*?

What is a continuously delivery pipeline?

- Continuous Integration / Continuous Delivery (Deployment)
- Agile product delivery (software, infrastructure and network) leveraging automation as much as possible
- Aligned with DevOps culture and best practices
- Build, Test and Deploy
- Pipeline as code

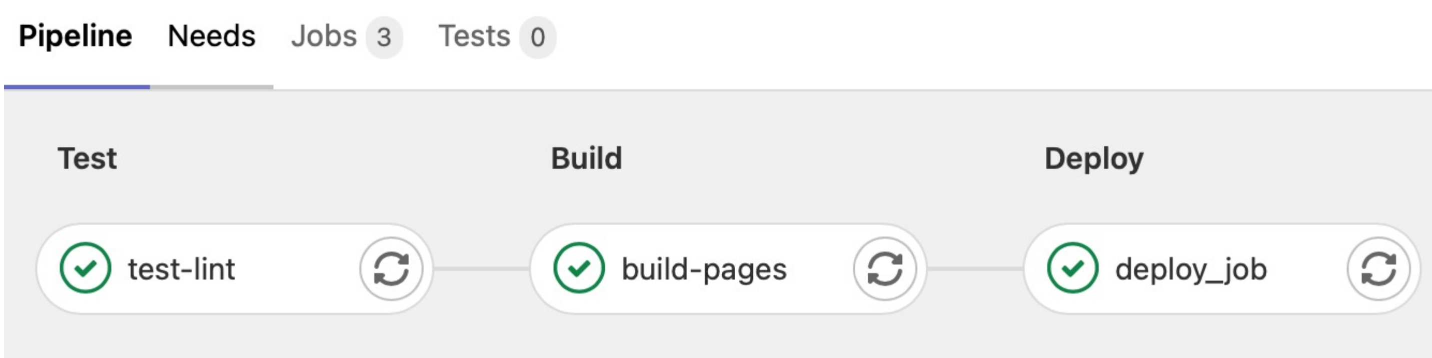


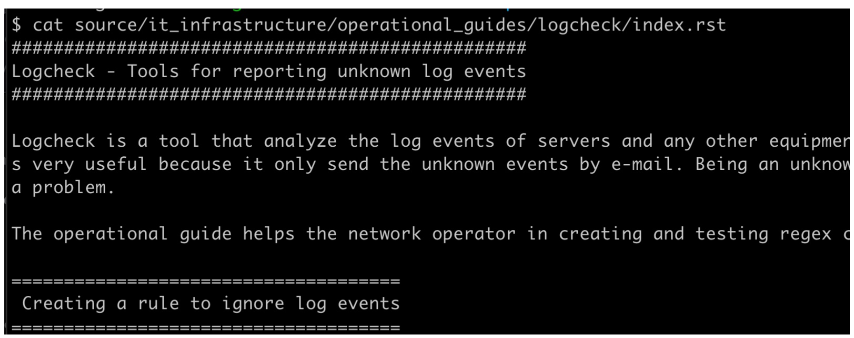
Example of an infrastructure/network pipeline

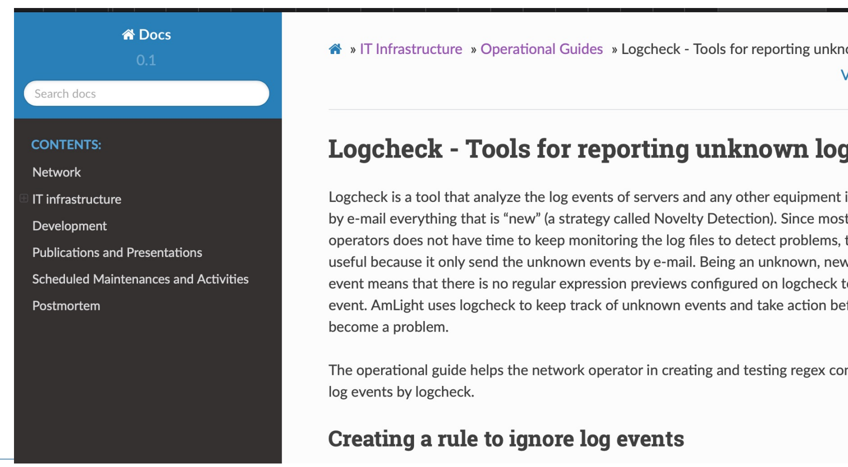
Configure BGP routing policies via Ansible

- Convert the BGP policies from a “descriptive language” to a target-specific configuration language
- Unit tests (correctness validation of the changed policy)
- Provisioning of the test infrastructure (e.g., virtual lab)
- Integration tests (network prefixes affected, overlap with other policies, traffic impact)
- Acceptance tests (how effective is the policy)
- Deploy in production (apply the config in the router)
  - Not necessarily automated

Another example



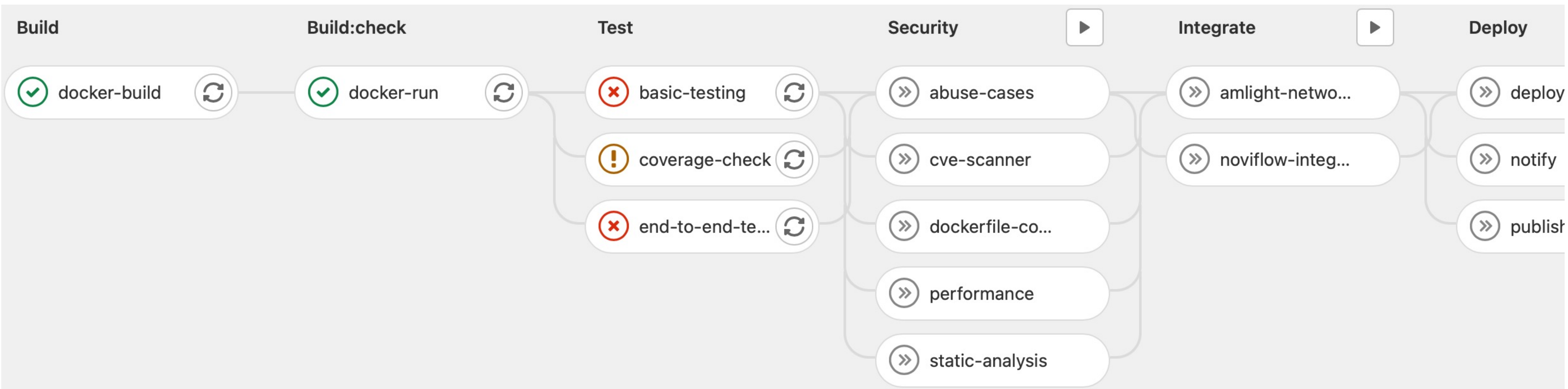




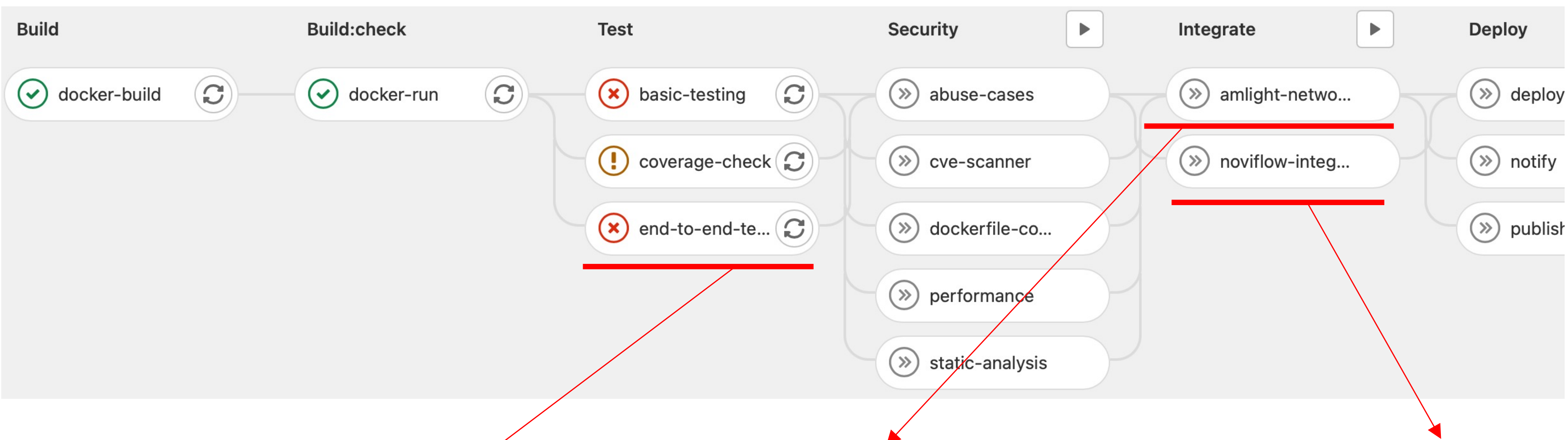
Goals for the Kytos-ng CI/CD pipeline

- Agility
  - Be able to quickly test new Kytos-ng napps (network applications) which add key services or features for our customers (e.g., INT, L3VPN, Security DPI, Threat Containment, etc)
  - Be able to evaluate the impact of changes in existing features (changes in services' APIs, changes affecting the data plane, changes affecting the control plane)
  - Be able to quickly identify bugs/mistakes and help fixing them
- Security
  - As a critical software (the network orchestrator), the more validations the better
  - Resiliency and availability must be guaranteed
  - Minimize the introduction of new security bugs
  - Contribute to the vulnerability management process

Pipeline design



Pipeline design



End-to-end tests: validate if the main services are working properly (Network emulator)

Virtual scenario/emulator using the same NOS in production, same topology, same user services

Hardware test: integration test with the same hardware (pre-production)

End-to-end tests

- Goal: validate the features taking into consideration the full work-flow from the user perspective:
  - User authentication
  - Submit a user request
  - Service provisioning
  - Service operation
  - Persistency
  - Consistency
  - Integration with the southbound API (OpenFlow)

Example

User request

```python
def test_015_create_evc_inter_switch(self):
    time.sleep(10)
    payload = {
        "name": "my evc1",
        "enabled": True,
        "dynamic_backup_path": True,
        "uni_a": {
            "interface_id": "00:00:00:00:00:00:00:01:1",
            "tag": {
                "tag_type": 1,
                "value": 15
            }
        },
        "uni_z": {
            "interface_id": "00:00:00:00:00:00:00:02:1",
            "tag": {
                "tag_type": 1,
                "value": 15
            }
        }
    }
    api_url = KYTOS_API + '/mef_eline/v2/evc/'
    response = requests.post(api_url, data=json.dumps(payload),
    assert response.status_code == 201
    data = response.json()
    assert 'circuit_id' in data
    time.sleep(20)
```

Control plane validation

```python
    # Each switch must have 3 flows: 01 for LLDP + 02 for the EVC (ingress + egress)
    s1, s2 = self.net.net.get('s1', 's2')
    flows_s1 = s1.dpctl('dump-flows')
    print(flows_s1)
    flows_s2 = s2.dpctl('dump-flows')
    print(flows_s2)
    assert len(flows_s1.split('\r\n ')) == 3
    assert len(flows_s2.split('\r\n ')) == 3

    # make sure it should be dl_vlan instead of vlan_vid
    assert 'dl_vlan=15' in flows_s1
    assert 'dl_vlan=15' in flows_s2
```

Data plane validation

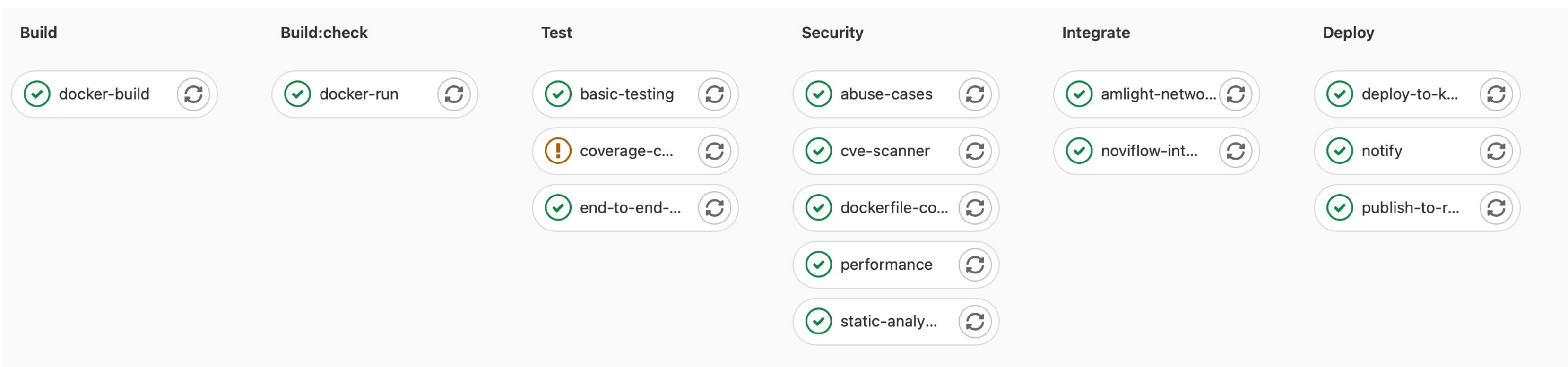
```python
    # Make the final and most important test: connectivity
    # 1. create the vlans and setup the ip addresses
    # 2. try to ping each other
    h11, h2 = self.net.net.get('h11', 'h2')
    h11.cmd('ip link add link %s name vlan15 type vlan id 15' % (h11.intfNames()[0]))
    h11.cmd('ip link set up vlan15')
    h11.cmd('ip addr add 15.0.0.11/24 dev vlan15')
    h2.cmd('ip link add link %s name vlan15 type vlan id 15' % (h2.intfNames()[0]))
    h2.cmd('ip link set up vlan15')
    h2.cmd('ip addr add 15.0.0.2/24 dev vlan15')
    result = h11.cmd('ping -c1 15.0.0.2')
    assert ', 0% packet loss,' in result
```

End-to-end tests

- Code repository: https://github.com/amlight/kytos-end-to-end-tests
  - 128 tests being executed (project start in Jun/2020)
  - Basic tests, concurrent requests, heavy usage conditions, fail scenarios, tricky requests
  - Testing all the main Kytos-ng components and additional napps
  - Using different network topologies
  - Runs in a daily basis (or on demand) and over the master branches (or for specific pull requests)
  - Three people involved: Network Engineer, Software Engineer and Quality Assurance Tester
- Current results:
  - 109 passed, 0 failed, 16 xfailed, 3 xpassed
  - Duration: 96 min

Other results (side effects)

- Use case 1: Information leakage due to sensitive information being published in the Git repository together with the code
- Use case 2: Change in the L2VPN service would deliver the packets to the user with wrong headers (i.e., VLAN ether-type as a service provider VLAN instead of the original one)
- Use case 3: A firmware release with a bug that eventually delivers corrupted telemetry reports for certain traffic profiles



Ongoing actions

- Adding new tests and scenarios (identifying use cases, abuse cases, tricky requests, etc)
- Validate how the SDN Orchestrator deals with failures in the infrastructure or southbound API
  - Example 1: Storage backend failing to persist information
  - Example 2: OpenFlow port which does not support output action
  - Example 3: switch reconnecting
- Improve the testing process performance, to enable a per pull-request pipeline execution

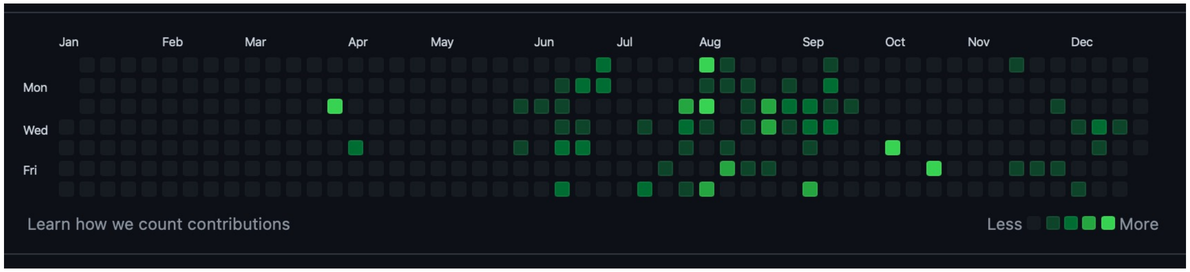
Outcomes

- Main outcome: more reliable, secure and robust SDN Orchestrator, with enhanced testing coverage and quality
- Automation is usually much more difficult and time consuming then manual processes, advantages will be seen in a long-term run
- Always evolving project, applying the best practices from DevOps culture and Software Engineering

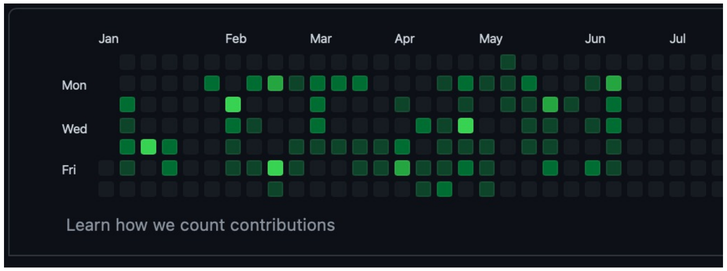
Lessons Learned

- It's been a learning journey for a Network Engineer directly participate in the software development lifecycle: creating tests, interacting with developers, creating blueprints, reviewing PRs, submitting PRs
- The interaction with developers always give many insights and they are very productive
- I've learned many concepts, tools and frameworks through out the process: Ansible, pytest, pylint, infrastructure as code, gitlab-ci, docker, gNMI/ gNOI, etc.
  - I'm thankful for many books, blogs, authors for all I learned (special thanks to https://gomex.me)

Github contributions activity



2020



2021

tnc21 Making Waves
21-25 June 2021

GÉANT

Thank you

Any Questions?

Italo Valcy <italo@amlight.net>

As part of the GÉANT 2020 Framework Partnership Agreement (FPA), the project receives funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 856726 (GN4-3).