Introduction to ANTA

Objectives and pre-requisites

Pre-requisite: Object Oriented Programming

- Python OOP Tutorial 1: Classes and Instances
- Python OOP Tutorial 2: Class Variables
- Python OOP Tutorial 4: Inheritance

Objectives

- Know how to **use ANTA** to perform NRFU testing
- Get familiar with **ANTA test development**
- Understand and **leverage object-oriented programming** with Python

Not covered

- Asynchronous programming
- Developing within the ANTA framework

ANTA in a Nutshell

ANTA stands for Arista Network Testing Automation

- Open-Source project developed under Apache License
- Driven by AS, SE and Solution Engineer community

Use cases

- Automate NRFU (Network Ready For Use) test on a preproduction network
- Automate tests on a live network (periodically or on demand)
- Execute tests as part of a CI pipeline

Python only

- No mandatory cumbersome framework
- Easier troubleshooting
- Fast and efficient using asyncio

Easy setup

- `pip install anta`
- Docker image: `ghcr.io/arista-netdevops-community/anta`

ANTA CLI


```
$ anta --help
Usage: anta [OPTIONS] COMMAND [ARGS]...

  Arista Network Test Automation (ANTA) CLI

Commands:
  debug  Debug commands for building ANTA
  exec   Execute commands to inventory devices
  get    Get data from/to ANTA
  nrfu   Run NRFU against inventory devices
```

ANTA framework needs 2 important inputs from the user to run: a **device inventory** and a **test catalog**

> Useful tip: use environement variables and a file to define the numerous ANTA options. Eventually use https://direnv.net/.

ANTA Device Inventory

The device inventory lists all the devices on which ANTA can perform network tests. ANTA **requires eAPI to be enabled** on the devices.

```yaml
anta_inventory:
  hosts:
  - host: 10.100.164.103
    name: spine1
    tags: ['spine']
  - host: ld004.lon.aristanetworks.com
    name: spine2
    tags: ['spine']
  - host: cal008.lon.aristanetworks.com
    name: leaf1
    port: 80
    tags: ['leaf']
```

What is possible to do when defining a device?

- Specify IP or a resolvable `host`, specify another eAPI `port` than 443
- Give a `name` alias
- Assign `tags` to devices

ANTA Test Catalog

The ANTA test catalog defines the tests to be loaded, their inputs and tags.

```
anta.tests.connectivity:
  - VerifyReachability:
      hosts:
        - source: Management0
          destination: 1.1.1.1
          vrf: MGMT
        - source: Management0
          destination: 8.8.8.8
          vrf: MGMT
anta.tests.system:
  - VerifyUptime:
      minimum: 10
  - VerifyReloadCause:
  - VerifyCoredump:
  - VerifyAgentLogs:
anta.tests.mlag:
  - VerifyMlagStatus:
      filters:
        tags: ['leaf']
```

Q: What are the tests available in ANTA?

A: Refer to the API Documentation or this test catalog example.

Q: What is the purpose of tags?

A: Run a subset of the test catalog on tagged devices. Refer to the documentation for more details.

Developing tests with ANTA

> To go into more details, refer to this documentation

ANTA provides an abstract class AntaTest. This class does the heavy lifting and provide the logic to define, collect and test data. A test in ANTA is an implementation of **AntaTest** where mandatory **class variables** and at least a **method** need to be defined:

```python
from anta.models import AntaTest, AntaCommand


class VerifyTemperature(AntaTest):
    """
    This test verifies if the device temperature is within acceptable limits.

    Expected Results:
      * success: The test will pass if the device temperature is currently OK: 'temperatureOk'.
      * failure: The test will fail if the device temperature is NOT OK.
    """

    name = "VerifyTemperature"
    description = "Verifies if the device temperature is within the acceptable range."
    categories = ["hardware"]
    commands = [AntaCommand(command="show system environment temperature", ofmt="json")]

    @AntaTest.anta_test
    def test(self) -> None:
        command_output = self.instance_commands[0].json_output
        temperature_status = command_output["systemStatus"] if "systemStatus" in command_output.keys() else ""
        if temperature_status == "temperatureOk":
            self.result.is_success()
        else:
            self.result.is_failure(f"Device temperature exceeds acceptable limits. Current system status: '{temperature_status}'")
```

Let's review step by step how this test has been written.

Developing tests with ANTA

```python
class VerifyTemperature(AntaTest):
    name = "VerifyTemperature"
    description = "Verifies if the device temperature is within the acceptable range."
    categories = ["hardware"]
    commands = [AntaCommand(command="show system environment temperature", ofmt="json")]
```

Mandatory class attributes

- `name` (`str`): Name of the test. Used during reporting.
- `description` (`str`): A human readable description of your test.
- `categories` (`list[str]`): A list of categories in which the test belongs.
- `commands` (`list[Union[AntaTemplate, AntaCommand]]`): A list of command to collect from devices. This list **must** be a list of AntaCommand or AntaTemplate instances.

Define an `AntaCommand` object

```python
AntaCommand(
    command="<EOS command to run>",
    ofmt="<eAPI output - json or text - default is json>",
    version="<eAPI version - valid values are 1 or “latest” - default is “latest”>",
    revision="<eAPI revision of the command. Valid values are 1 to 99. Revision has precedence over version>"
)
```

Rendering AntaTemplate instances will be discussed later.

Developing tests with ANTA

```python
class VerifyTemperature(AntaTest):
    @AntaTest.anta_test
    def test(self) -> None:
        command_output = self.instance_commands[0].json_output
        temperature_status = command_output["systemStatus"] if "systemStatus" in command_output.keys() else ""
        if temperature_status == "temperatureOk":
            self.result.is_success()
        else:
            self.result.is_failure(f"Device temperature exceeds acceptable limits. Current system status: '{temperature_status}'")
```

Coding the test

test(self) -> None is an abstract method that must be implemented. It contains the test logic that can access the collected command outputs using the `instance_commands` instance attribute and **must** set the `result` instance attribute accordingly. It must be implemented using the `AntaTest.anta_test` decorator that provides logging and will collect commands before executing the `test()` method.

> Useful tip: when coding this method, use `<EOS command> | json` in EOS CLI to see the JSON output of the command to being parsed

That's it! We've just reviwed a simple test in ANTA.

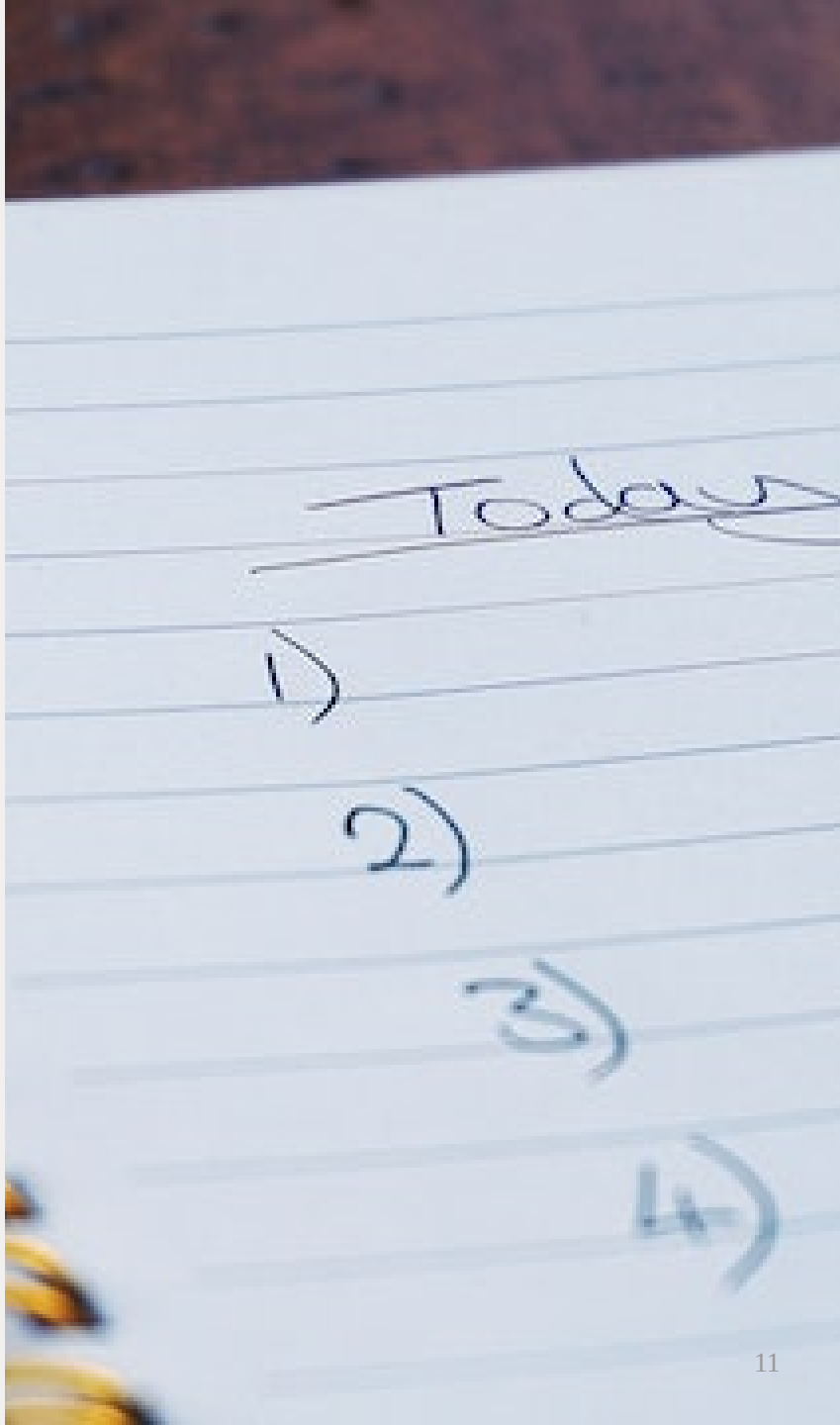
Fun with ANTA

`ANTA Hackathon Script`

Introduction

- This hackathon is split into 3 sections. Each section takes around 20 minutes to complete. But it can takes more time if you are running it on your own.
- Work in team, only one submission per team
- Repository is available at titom73/anta-malaga
- An Arista Test Drive is required. **
- Ask questions at any time!
- Google Chat: #anta-field on Google Chat

> ** There is an alternative cLab setup integrated with this repository and a container with all dependencies (see the last slide for details). Use at your own risk! ATD is the only supported option.



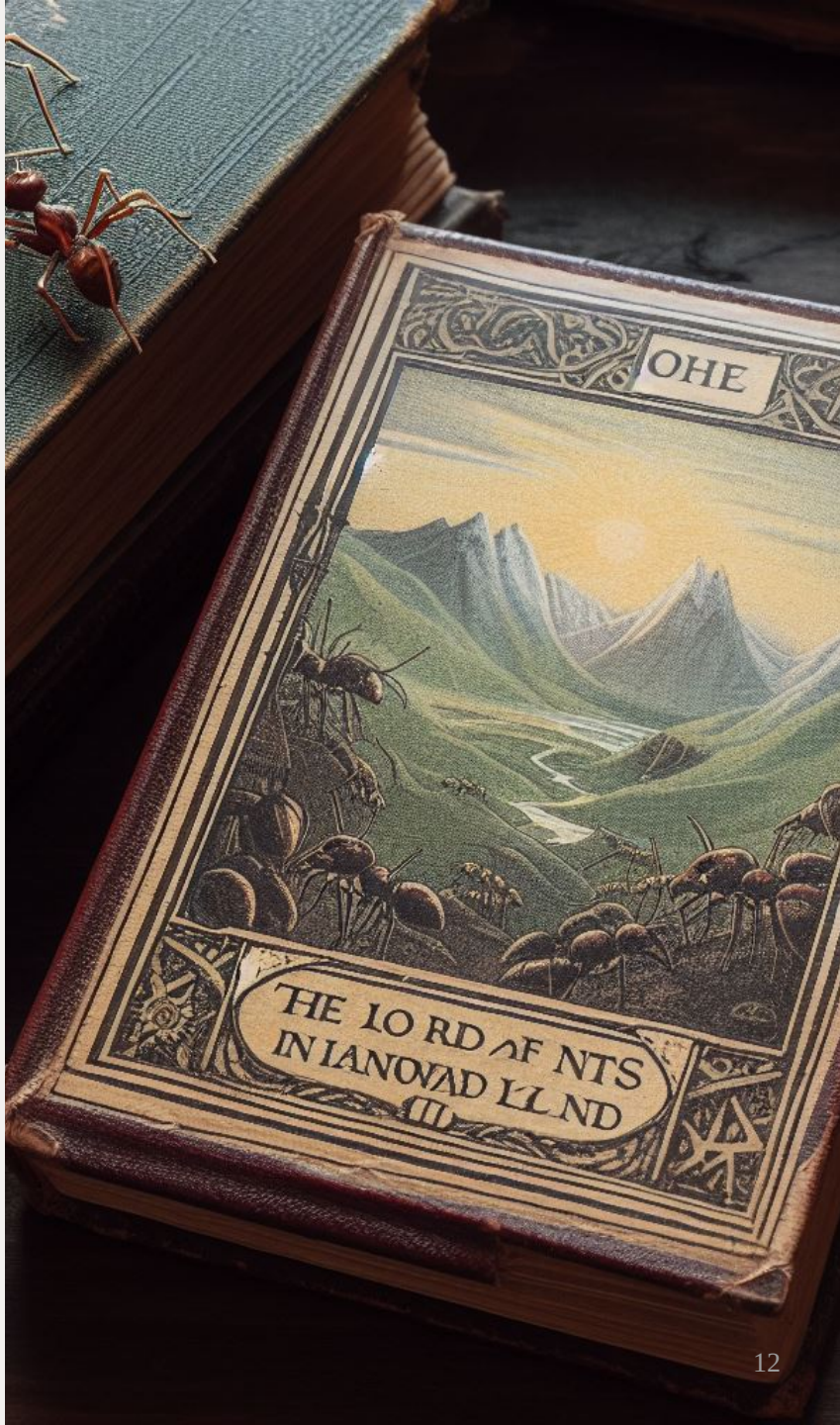
Context

Your customer told you their servers can’t reach ressources and they configure servers and fabric with LACP

You don’t have access to configuration of devices and you need to identify his issue.

Test your fabric to check:

- If all BGP sessions are UP and established.
- If all interfaces are UP and do not have error or drop counters.
- EVPN family is configured.



Stage #1

1. Create an ANTA inventory from scratch
2. Create an ANTA test catalog from the test plan
3. Is there any issue on the fabric?

That’s all folks!

Use your brain and RTFM:

- Use Inventory & Catalog
- ANTA CLI overview
- Get Inventory Information
- ANTA Tests catalog

Stage #2

Now, you have seen that PortChannel4 is down

Develop a test to check LACP status:

- Check `PortChannel` is in `collecting` mode
- Check `PortChannel` is in `distributing` mode

Tips

- Use `pip install -e .`
- Develop your tests under `./anta_custom/<your_file>.py`
- Your test will be available with `anta_customer.<your_file>.<your_test>`
- Test development example

Stage #3

Traffic is still not passing correctly. Check that all connected routes are correctly installed in routing table. You should have at least 3 connected routes installed in VRF `Tenant_A_OP_Zone`

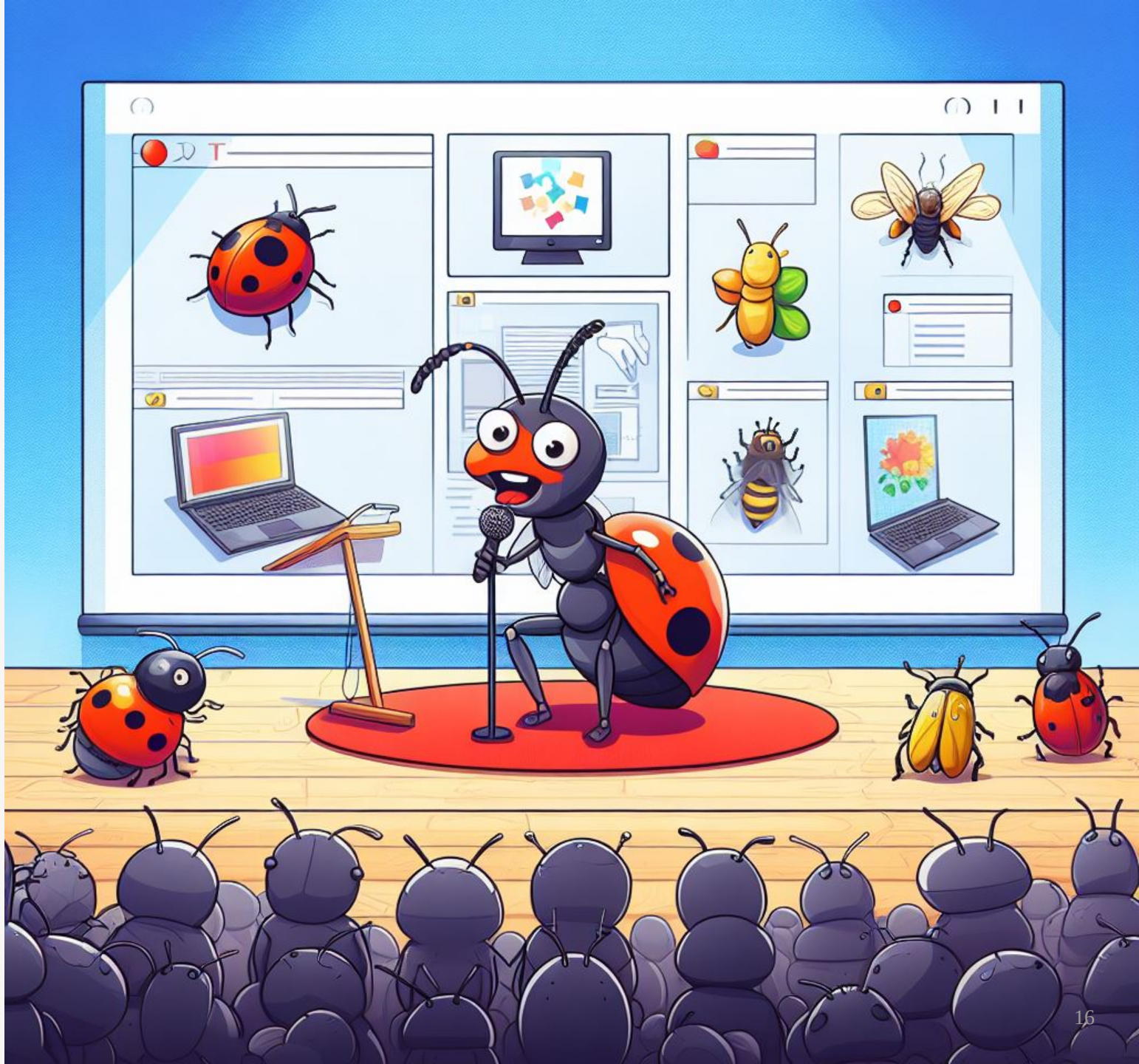
Develop a test based on `AntaTemplate` to validate number of connected routes in VRFs:

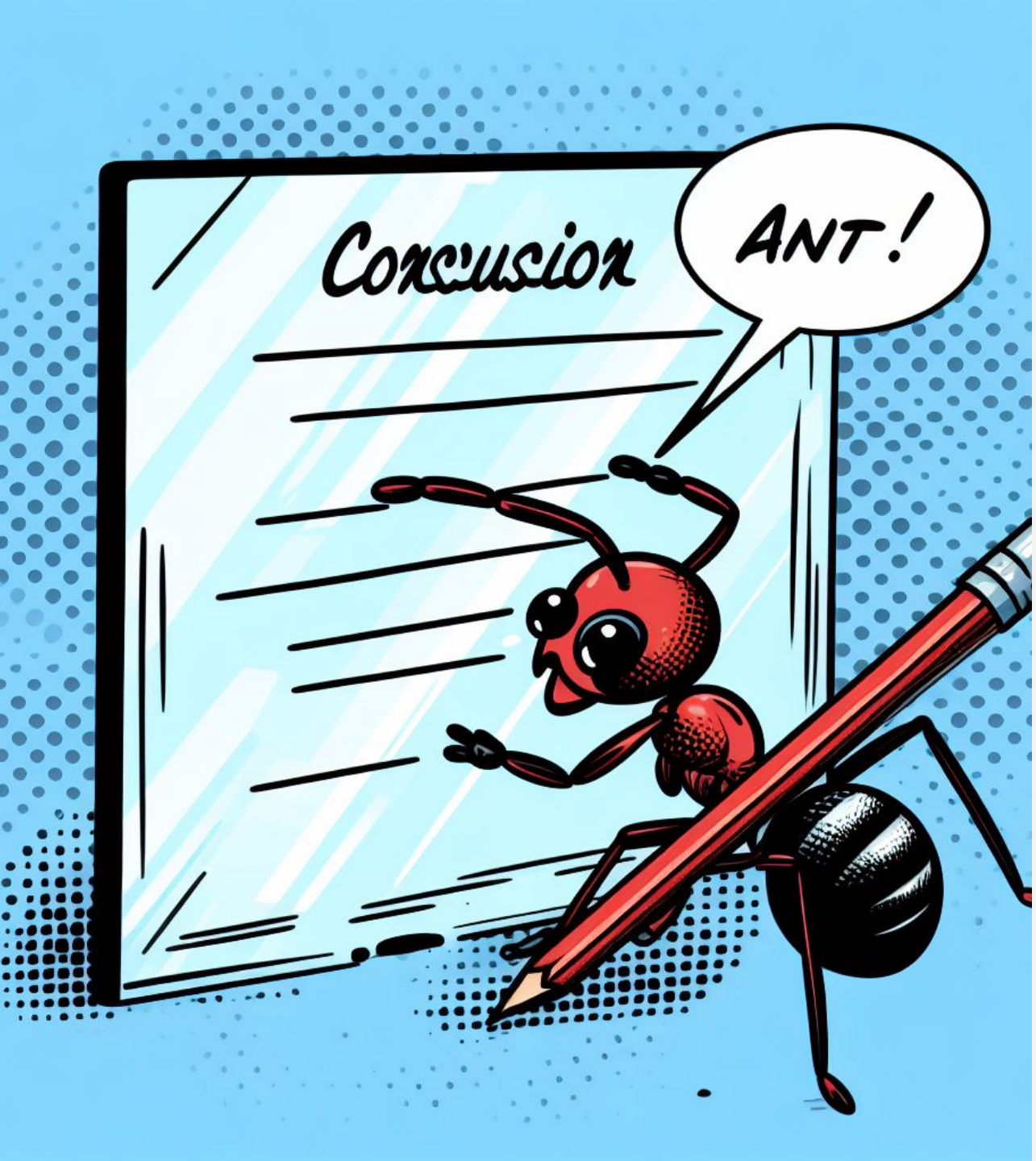
- Test **must** use `anta.models.AntaTemplate`
- VRF name is an template input for your test.
- Run ANTA to check routes status and fix configuration in EOS.

Stage #4

You can now build additional tests to make your customer happier with his Arista Fabric.

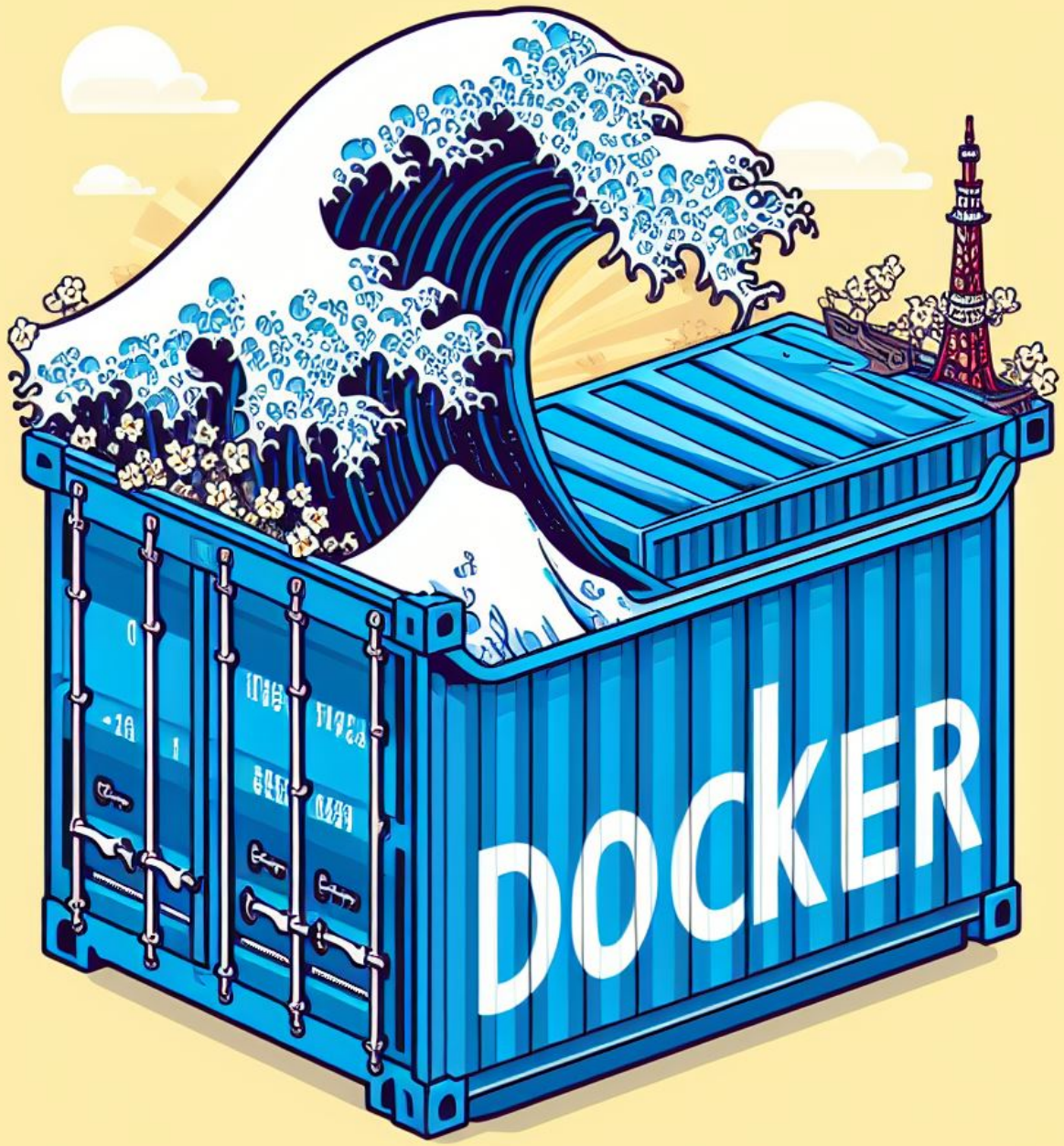
Stage is yours !





Conclusion

- How do you think you can use ANTA in the field ?
- What key feature is missing for customer’s usage ?



ANTA Environment in a Dev Container

- Hackathon repo includes a dev container with AVD, ANTA and Containerlab pre-installed.
- Set ARTOKEN environment variable first to download cEOS image inside the container. Or add it as a bind mount.
- Simply clone the repository in VSCode to start a devcontainer. You'll need Docker Desktop and VSCode Dev Containers extension installed.
- Investigate the environment yourself. It's a hackathon. 😄
- This will not work on ARM-base MacBooks, sorry! There is no image to pull. But! You can run the container as Github Codespace.
- Feel free to ask questions, but not for support! Dev Container has a lot of benefits, but ATD is the only supported option.

> BREAKING NEWS: AVD Dev Containers preview is coming soon (you can already pull some). Let's build more demos, hackathons, examples, POCs, workshops, etc. for us and our customers!


HANK
YOU!
THANK
YOU'!