DEVNET Create

A Journey to Cuddle with an API

Tim Miller

Technical Solutions Architect, Cisco

@broadcaststorm

Tim Miller

Technical Solutions Architect, Cisco

Tim has been automating infrastructure since before it was cool with HPC clusters over 20 years ago. He leverages that expertise on the GVE DC team in Cisco, evangelizing various cloud native technologies like Terraform, Ansible, K8s and the APIs, Python, and methods to deploy them.

DEVNET Create

Agenda

- Backstory
- The Many Faces of User Experience
- Wrap Up

DEVNET Create

Intended Audience

- Script hackers
- Dabblers in coding
- Systems/network engineers serious about coding
- Those ready to take their first step into a more formal approach to building a software project

The Backstory

DEVNET Create

Hierarchy of Automation

REST API	SDK	Libraries/ Modules	Automation Frameworks
Postman Collections Python Requests	Language Bindings Models and Methods	Typical user tasks Hides Internal App Complexity	Low barrier to entry Minimal/No Code

Hierarchy of Automation

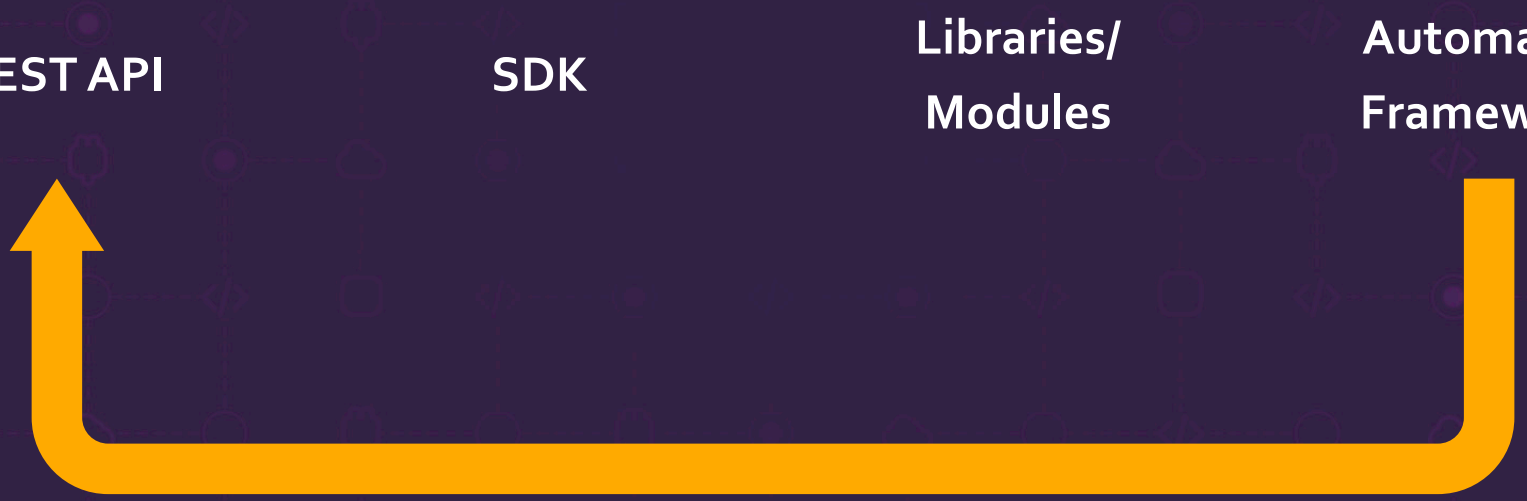
REST API

SDK

Libraries/
Modules

Automation
Frameworks

Hierarchy of Automation



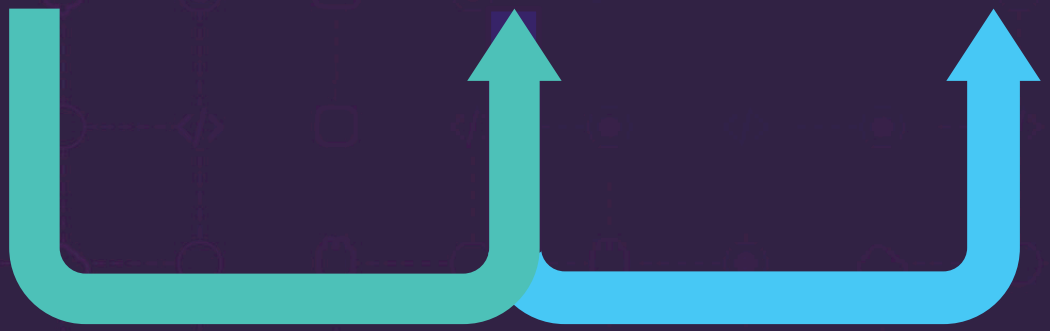
Hierarchy of Automation

REST API

SDK

Libraries/
Modules

Automation
Frameworks



DEVNET
Create

Hierarchy of Automation

REST API

Automation Frameworks

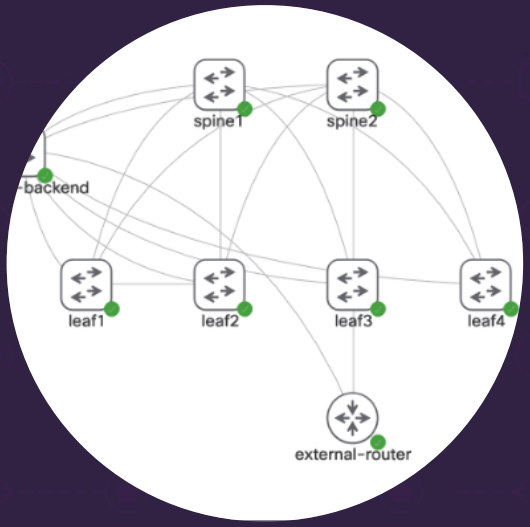
What do I want to build?

DEVNET Create

Requirements vs User Experience

- Focus on UX
  - Helped make key architectural decisions early
  - Permitted flexibility on deferring decisions
- Language matters
  - Goals and User Experience provide guidance
  - Requirements (for me) led to analysis paralysis

High Level Goals



Command Line

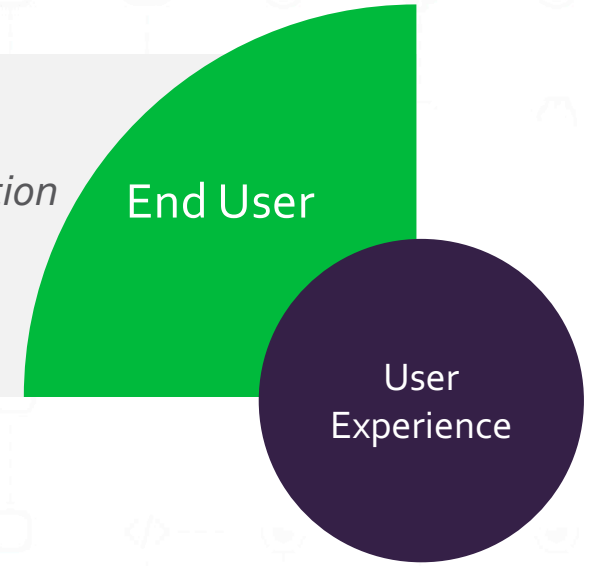
Demo Setup

Leverage Elsewhere

DEVNET Create

The Many Faces of UX

utilctl **resource** *action*
utilctl **resource sub** *action*



utilctl.py switch add --help

```
Usage: utilctl.py switch add [OPTIONS] FABRIC ROLE NAME

  Add a switch to the specifed fabric with the specified role, identifying the
  switch by its switch name.  Optionally, you can provide the serial number or
  mgmt0 IP in case of duplicate switch names.

Arguments:
  FABRIC  Fabric in which to add switch  [required]
  ROLE    Switch role in fabric  [required]
  NAME    Switch name  [required]

Options:
  --serial TEXT  Switch Serial Number
  --ip TEXT      Switch mgmt0 IP Number
  --help         Show this message and exit.
```

End User Goals

What we will do

- Focus on a CLI analogs to common GUI actions

What we won't do

- Generic inputs - not a CLI replacement for OpenAPI GUI or Postman

Technology Outcome

Python modules for CLI

- Click, Typer

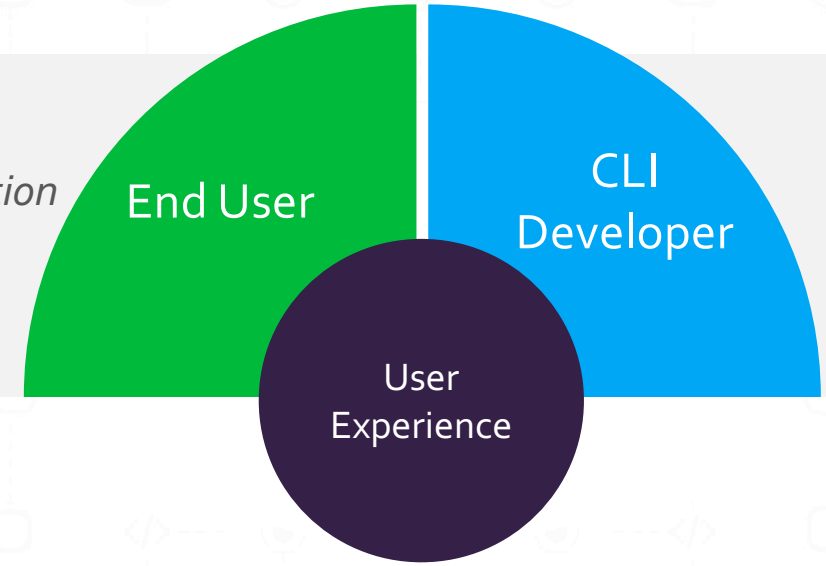
Directory Layout

- cli/**resource**/*action*.py

Leverage existing default values if not provided

The Many Faces of UX

utilctl **resource** *action*
utilctl **resource** **sub** *action*



Avoid service logic
UI module flexibility

cli/switch/actions.py

```python
@resource.command(no_args_is_help=True)
def list(
    fabric: str = typer.Argument(..., help="Fabric in which to add switch"),
    name: str = typer.Argument(None, help="Switch name"),
):
    """
    List all switches in the given fabric, if no name is given.
    List switch details if a switch name is provided.
    """

    typer.echo(f"Fabric: {fabric}, Switch Name: {name}")

    # Call the underlying library and retrieve the results
    results = switch.list_switch(fabric, name)

    # Output the results in some meaningful way.  One option would display
    # all switches in fabric, second option would display switch details
    pass

```

DEVNET Create

CLI Developer Goals

- Logical Tasks
  - Login to Service
  - Error Handling
  - Wrappers around resource operations for input validation
- Common Data Structures
  - Connection, Authentication, Switch Identification

Technology Outcomes

Directory Layout

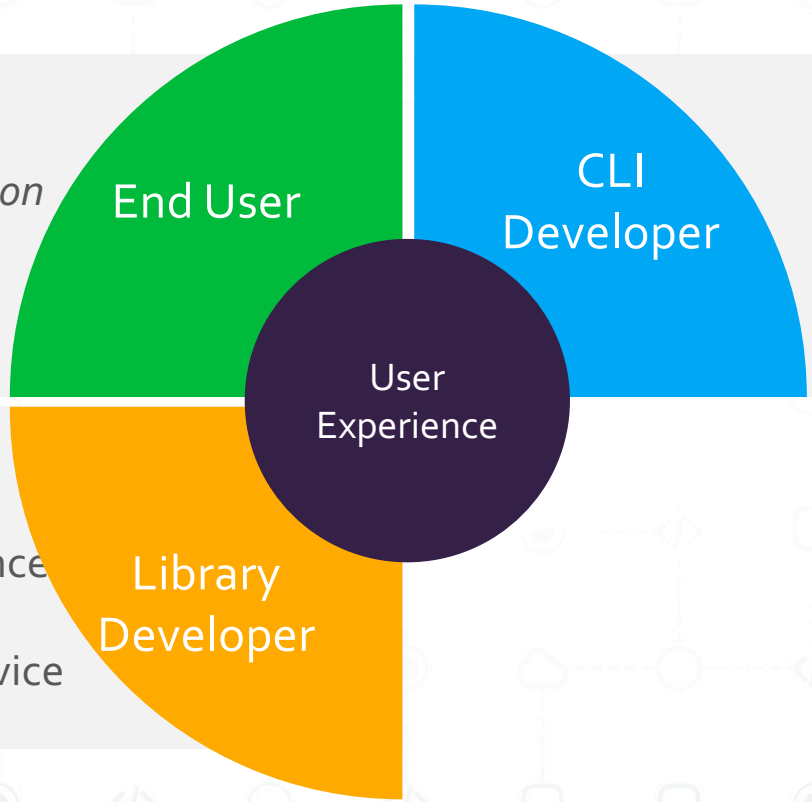
- sdk/ **resource**/*.py

Identified clear need to build an application library

- Helped focus decisions around standards, object behavior
- Goal to return all data as Python native data types

The Many Faces of UX

utilctl **resource** *action*
utilctl **resource** **sub** *action*



Avoid service logic
UI module flexibility

API Version Independence
Python Bindings to API Objects/Classes for Service

sdk/switch/add_switch.py

```python
def add_switch(fabric: str, role: str, name: str, serial: str = "", ip: str = ""):
    """
    Add a switch to the specified fabric, with the specified switch role
    and name.  Optionally identify the switch with serial number or mgmt0
    IP address.
        fabric, role, name are required.
        serial, ip are optional.
    """

    # Create connection to the service
    conn = session.create_session()

    # Check with the API to see if fabric is valid

    # Check with the API to see if role is valid

    # Make API call to add the switch with the specified name, serial, and IP

    # Create results data structure, return to UI for output processing
    pass

```

Library Goals

- Insulate API changes, new features
- Well defined classes for product specific resources
  - Interfaces, Switches, Fabrics
- Template/Policy
  - Abstraction of global functionality
  - Specialization for each specific template
  - Policy instantiation functionality

Technology Outcomes

Abstraction layer for API calls

api/rest/all/core.py, api/rest/v11_4/*.py, …

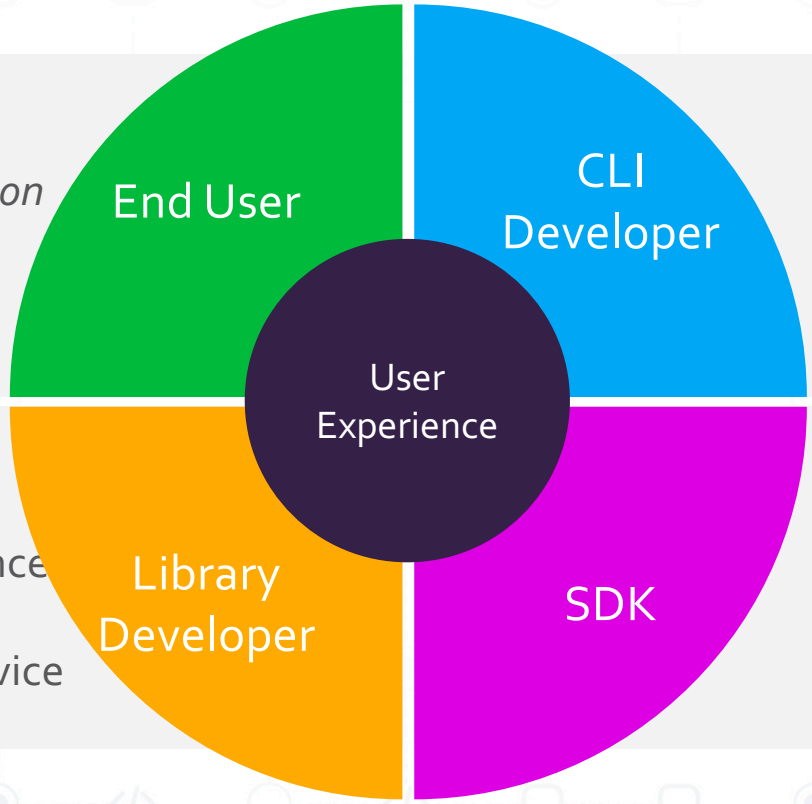
Model hierarchy with version dependence

api/models/all/core.py, api/models/v11_4/*.py, …

Embed version dependent selection in connection

The Many Faces of UX

utilctl **resource** *action*
utilctl **resource** **sub** *action*



Avoid service logic
UI module flexibility

API Version Independence
Python Bindings to API Objects/Classes for Service

Do I have to build all this by hand?
Isn't there a better way?

DEVNET Create

Autogenerated Python API interface

Relying on Python templating of generator tool developers

- Security and validity is still your responsibility

Does not mitigate the need to understand how to consume API

Ease of use strongly dependent on OpenAPI spec

- Missing return types may result in a lot of manual processing

api/models/all/core.py

```
6   class v12_0(core):
7       """
8       API Methods for Service version 11.5.  The unique, version specific methods
9       should be stored in here.  Anything supported and unchanged across the
10      supported versions should be placed in api.models.core
11      """
12      def __init__(self, session = None):
13          """
14          Reference to the service session is stored locally in order
15          to make the direct API calls
16          """
17
18          if not session:
19              raise Exception('Sample')
20
21          self._api = rest(session)
22
23      def get_record(self):
24          """
25          Example method of get_record in v12.0
26          """
27
28          print('v12_0: get_record')
29          pass
30
```

DEVNET Create

SDK Goals

API changes will happen, but historically few/minor

- Default to methods landing in common object (“all”)
- Break code out to version specific when necessary

No further abstraction needed. Version dependence handles any URL/resource endpoint changes.

Technology Outcomes

Object oriented approach

- Inheritance and polymorphism

Leverage api/rest directory for possible autogen option

Wrapping it all up

DEVNET Create

Summary

Reframing how you view the project can lead to more natural technology choices without all the details

You don’t have to figure the project all out on day 1

You do have to enjoy the project

References

Sample Project Repo

https://github.com/CiscoSE/DevNetCreate21-TS47-Journey

OpenSource Project Resources

https://opensource.guide/starting-a-project/

CLI Parsers

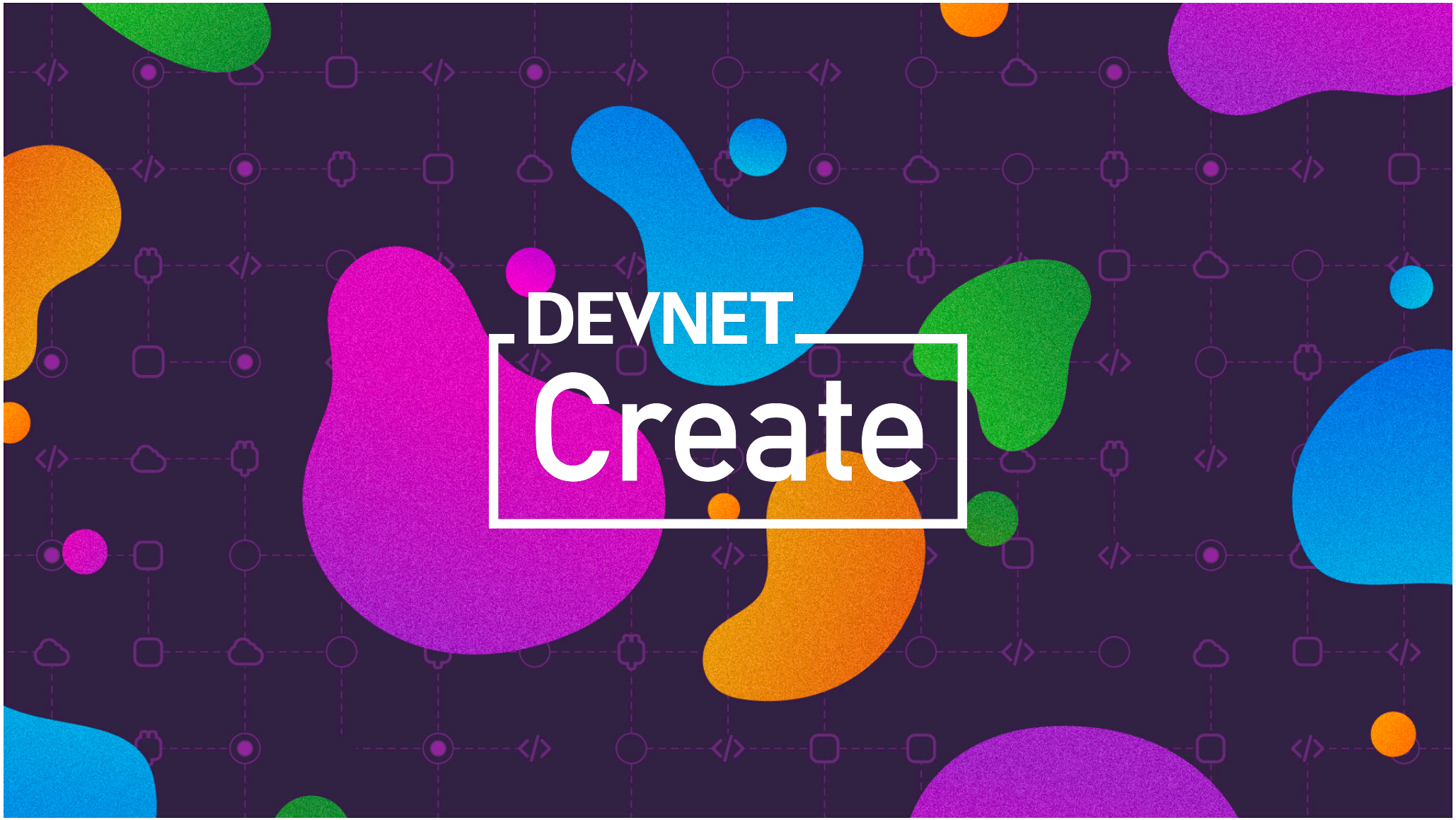
Click: https://click.palletsprojects.com/

Typer: https://typer.tiangolo.com

OpenAPI Generator

https://openapi-generator.tech

https://github.com/OpenAPITools/openapi-generator


DEVNET
Create