

The background features a vibrant, abstract design with a color gradient from dark blue on the left to bright yellow and white on the right. The design consists of overlapping, wavy horizontal bands and a radial pattern of lines emanating from a bright white point on the right side, creating a sense of motion and energy.

CISCO *Live!*

Let's go



The bridge to possible

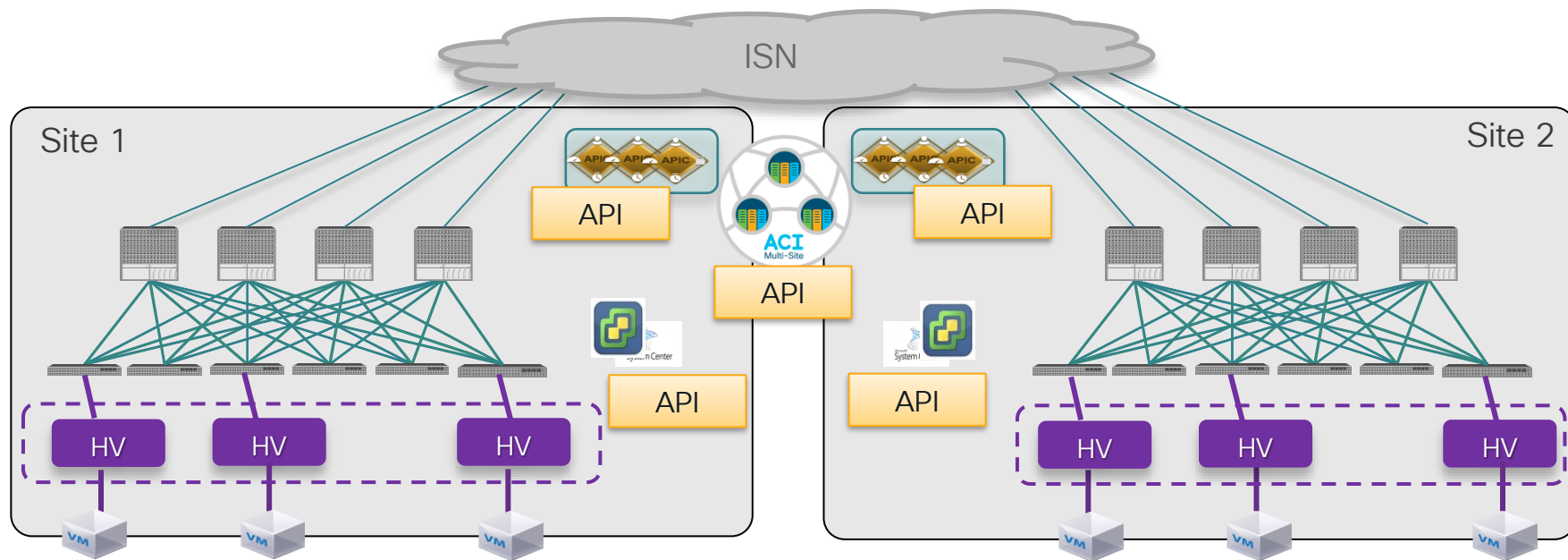
API Middle-tier for Dummies, or How to Make your UI Developers Happy by Stitching all your APIs Together.

Ramses Smeyers, Distinguished Engineer – CX TAC

Agenda

- Why
- Start small
- Introducing Flask
- Our use-case
- End-2-end demo

Average DC landscape



Why

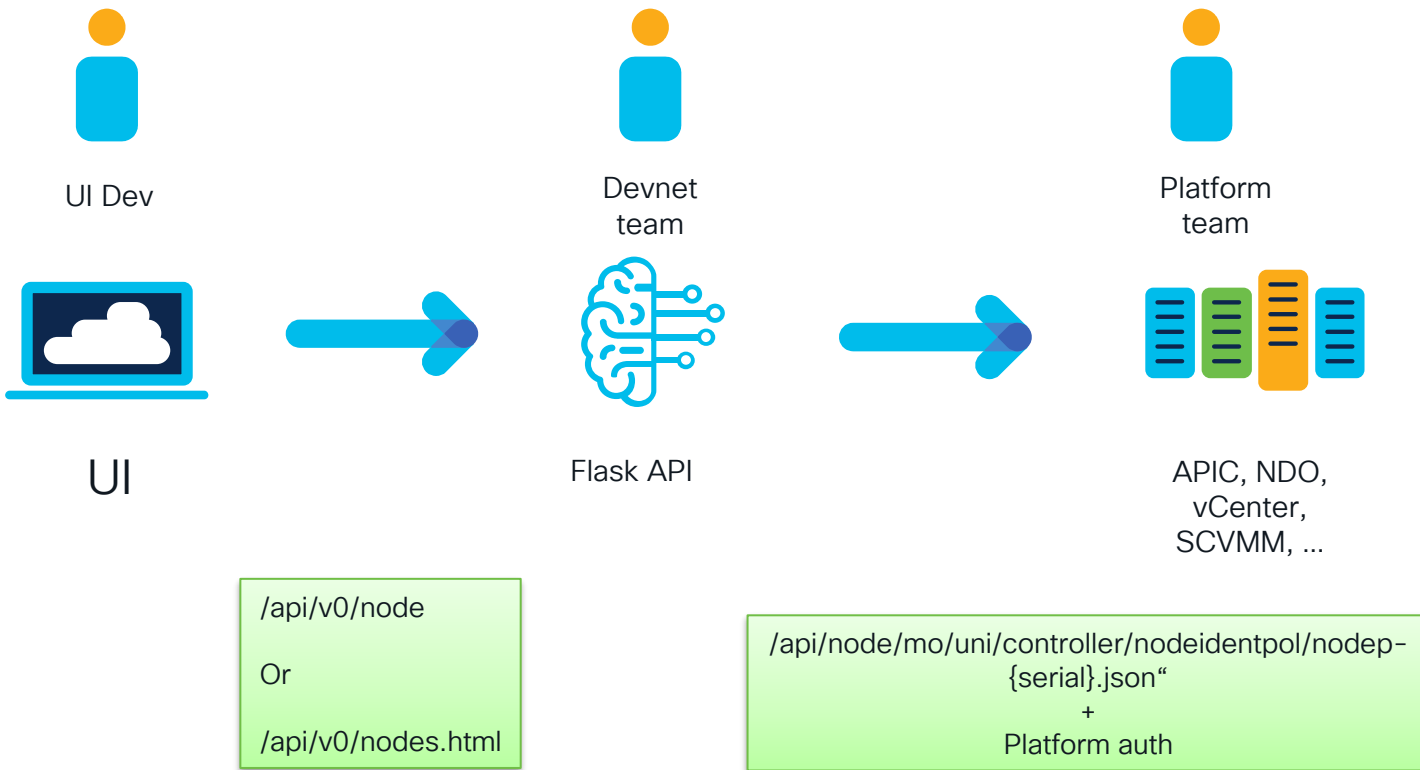
Automation options

- Cisco Intersight Cloud Orchestrator
- NSO
- Ansible Automation Platform
- Terraform Cloud
-
- Python

Our session will focus on this



A possible option – manage an ACI switch



Start small

What do you want to automate ?

- Ask the “Platform team” the re-occurring change requests
 - What is repetitive
 - What is information intensive
- Check Product audit logs for re-occurring changes

Document

- Expose amazing documentation
 - Pref. in <https://spec.openapis.org/oas/v3.1.0.html> format
- Auto-generate your API spec



Introducing flask

Flask



- Python based API
 - Self-documenting
 - Auto generates OpenAPI spec
 - Build in validators
 - Response marshalling
 - Request Parsing
 - Simple yet elegant
-
- Lot's of other Flask based API's out there, but we'll use Restx

Let's get it started

```
from flask import Flask
from flask_restx import Resource, Api

app = Flask(__name__)
api = Api(app)

@api.route('/hello')
class HelloWorld(Resource):
    def get(self):
        return {'hello': 'world'}

api.add_resource(HelloWorld, "/hello" endpoint="hello world")

if __name__ == '__main__':
    app.run(debug=True)
```

```
$ python3 api.py
```

```
* Serving Flask app 'api'
```

```
* Debug mode: on
```

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

```
* Running on http://127.0.0.1:5000
```

Press CTRL+C to quit

```
* Restarting with stat
```

```
* Debugger is active!
```

```
* Debugger PIN: 416-240-977
```

```
$ curl http://127.0.0.1:5000/hello
{
  "hello": "world"
}
```

Documentation

```
from flask import Flask
from flask_restx import Resource, Api
```

```
app = Flask(__name__)
api = Api(app, doc="/doc")
```

```
class HelloWorld(Resource):
    def get(self):
        return {'hello': 'world'}
```

```
api.add_resource(HelloWorld, "/hello", endpoint="hello world")
```

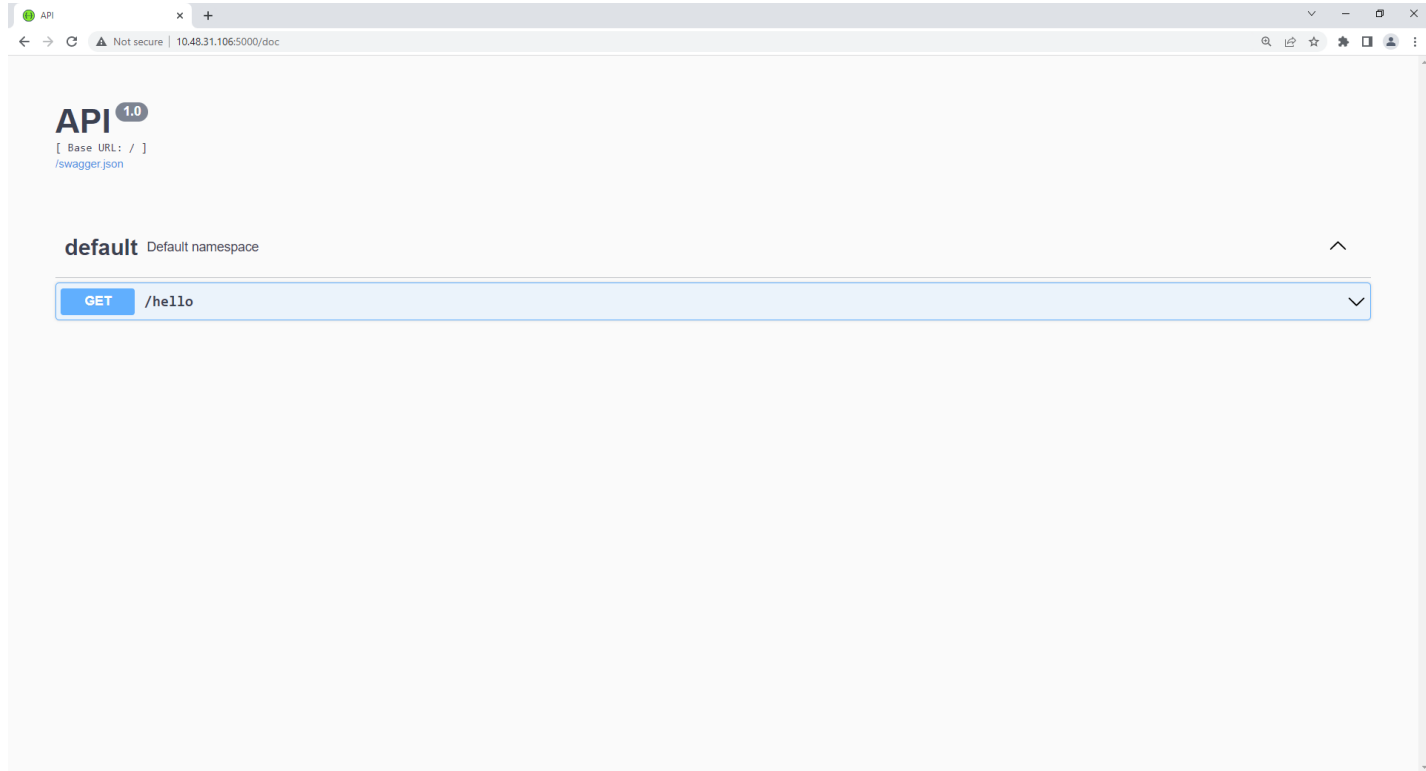
```
if __name__ == '__main__':
    app.run(debug=True, host="0.0.0.0")
```

```
$ python3 api.py
* Serving Flask app 'api'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a
production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://10.48.31.106:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 416-240-977
10.55.17.89 -- [23/Jan/2023 16:07:56] "GET /hello HTTP/1.1" 200 -
```

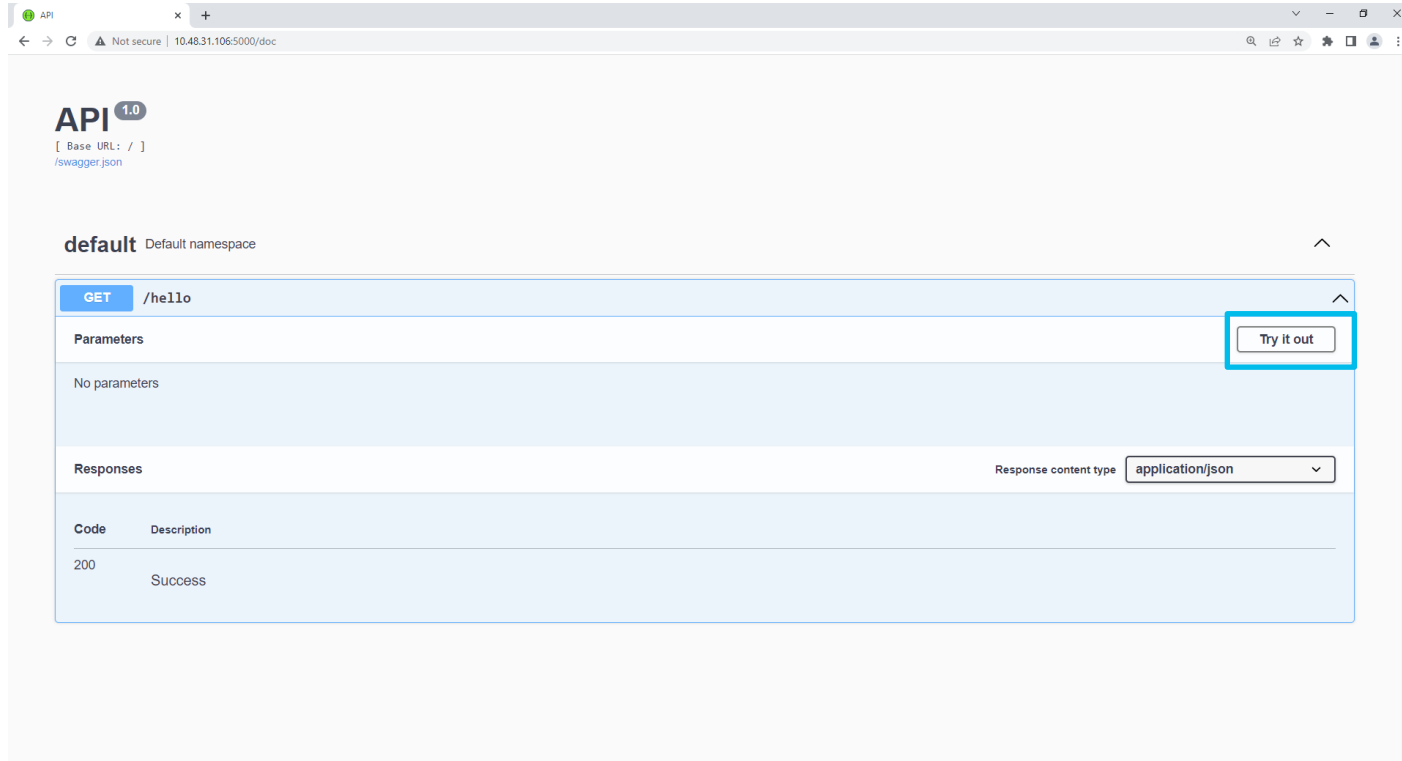
Auto generate docs

Listen on all IP
addresses → only
use for DEV

Documentation – swagger UI



Documentation – swagger UI



Swagger UI – Try it out

The screenshot shows the Swagger UI interface for a GET endpoint at /hello. The 'Parameters' section is empty. The 'Execute' button is highlighted in blue. The 'Responses' section shows a 200 status code with a JSON response body: {"hello": "world"}. The 'Response headers' section shows the following headers: connection: close, content-length: 25, content-type: application/json, date: Mon, 23 Jan 2023 15:07:56 GMT, and server: Werkzeug/2.2.2 Python/3.9.14.

GET /hello

Parameters Cancel

No parameters

Execute Clear

Responses Response content type: application/json

Curl

```
curl -X 'GET' \
  'http://10.48.31.106:5000/hello' \
  -H 'accept: application/json'
```

Request URL

<http://10.48.31.106:5000/hello>

Server response

Code	Details
200	Response body <pre>{ "hello": "world" }</pre> Download Response headers <pre>connection: close content-length: 25 content-type: application/json date: Mon, 23 Jan 2023 15:07:56 GMT server: Werkzeug/2.2.2 Python/3.9.14</pre>

Responses

Code	Description
200	Success

Request Parsing

- reqparse is modelled as argparse
 - handles input validation
- auto generates documentation when used in Flask

API Expect – control input

```
from flask import Flask
from flask_restx import Resource, Api, reqparse
```

```
app = Flask(__name__)
api = Api(app, doc="/doc")
```

```
translate_parser = reqparse.RequestParser(bundle_errors=True)
translate_parser.add_argument("word", required=True, type=str)
```

```
class Translate(Resource):
```

```
    @api.expect(translate_parser, validate=True)
    def get(self):
        args = translate_parser.parse_args()
        word = args["word"]
```

```
        dictionary = {
            "car": "auto",
            "house": "huis"
        }
```

```
        return {'translated': dictionary[word]}
```

```
api.add_resource(Translate, "/translate", endpoint="Translate")
```

```
if __name__ == '__main__':
    app.run(debug=True, host="0.0.0.0")
```

Configure request parser

Enforce parser

API Expect – control input – swagger

API ^{1.0}
[Base URL: /]
[/swagger.json](#)

default Default namespace ^

GET /translate ^

Parameters Try it out

Name	Description
word * required string (query)	word

Responses Response content type: application/json

Code	Description
200	Success

Response marshalling

- Renders response data
 - Enforces strict model
- Auto generates documentation

API Marshal – control output

```
from flask import Flask
from flask_restx import Resource, Api, reqparse, fields, marshal

app = Flask(__name__)
api = Api(app, doc="/doc")

translate_parser = reqparse.RequestParser(bundle_errors=True)
translate_parser.add_argument("word", required=True, type=str)

translate_response_model = api.model(
    "translate_response",
    {
        "success": fields.Boolean(required=True, choices=(False, True), default=True),
        "translated": fields.String(required=True),
    },
)

class Translate(Resource):
    @api.marshal_with(translate_response_model, code=200, description="Translate OK")
    @api.expect(translate_parser, validate=True)
    def get(self):
        args = translate_parser.parse_args()
        word = args["word"]

        dictionary = {
            "car": "auto",
            "house": "huis"
        }

        return {'translated': dictionary[word]}

api.add_resource(Translate, "/translate", endpoint="Translate")

if __name__ == '__main__':
    app.run(debug=True, host="0.0.0.0")
```

Create response model

Enfore model

API Marshal – control output – swagger

default Default namespace

GET /translate

Try it out

Name	Description
word * required string (query)	word
X-Fields string(\$mask) (header)	An optional fields mask X-Fields

Responses Response content type: application/json

Code	Description
200	Translate OK Example Value Model <pre>{ "success": true, "translated": "string"}</pre>

Models

translate_response >

API Marshal – control output – swagger

Models

```
translate_response {  
  success* boolean  
  translated* string  
}
```

```
class Translate(Resource):  
    @api.marshal_with(translate_response_model, code=200, description="Translate OK")  
    @api.expect(translate_parser, validate=True)  
    def get(self):  
        args = translate_parser.parse_args()  
        word = args["word"]  
  
        dictionary = {  
            "car": "auto",  
            "house": "huis"  
        }  
  
        return {"translated": dictionary[word]}
```

Curl

```
curl -X 'GET' \  
  'http://10.48.31.106:5000/translate?word=car' \  
  -H 'accept: application/json'
```

Request URL

```
http://10.48.31.106:5000/translate?word=car
```

Server response

Code	Details
------	---------

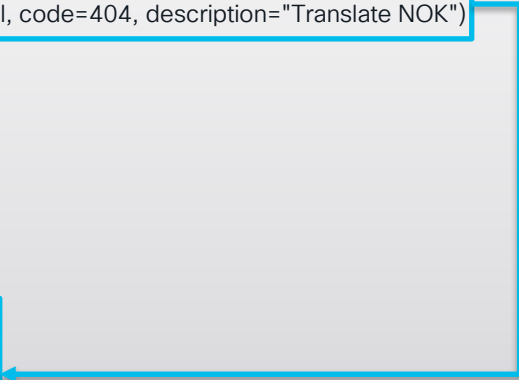
200	
-----	--

Response body

```
{  
  "success": true,  
  "translated": "auto"  
}
```

API Marshal – there's more than 200

```
class Translate(Resource):  
    @api.marshal_with(translate_response_model, code=200, description="Translate OK")  
    @api.marshal_with(translate_response_model, code=404, description="Translate NOK")  
    @api.expect(translate_parser, validate=True)  
    def get(self):  
        args = translate_parser.parse_args()  
        word = args["word"]  
  
        dictionary = {  
            "car": "auto",  
            "house": "huis"  
        }  
  
        if word in dictionary:  
            return {'translated': dictionary[word]}  
        else:  
            return {'success': False }, 404
```



API Marshal – there's more than 200 – swagger

Curl

```
curl -X 'GET' \
  'http://10.48.31.106:5000/translate?word=card' \
  -H 'accept: application/json'
```

Request URL

http://10.48.31.106:5000/translate?word=card

Server response

Code	Details
404	Error: NOT FOUND Response body <pre>{ "success": false, "translated": null }</pre> Response headers <pre>connection: close content-length: 49 content-type: application/json date: Tue, 31 Jan 2023 10:43:44 GMT server: Werkzeug/2.2.2 Python/3.9.14</pre>

Responses

Code	Description
200	Translate OK Example Value Model <pre>{ "success": true, "translated": "string" }</pre>
404	Translate NOK Example Value Model <pre>{ "success": true, "translated": "string" }</pre>

What about POST ? – quick code-refactor

```
from flask import Flask
from flask_restx import Resource, Api, reqparse, fields, marshal

app = Flask(__name__)
api = Api(app, doc="/doc")

translate_parser = reqparse.RequestParser(bundle_errors=True)
translate_parser.add_argument("word", required=True, type=str)

translate_response_model = api.model(
    "translate_response",
    {
        "success": fields.Boolean(required=True, choices=(False, True), default=True),
        "translated": fields.String(required=True),
    },
)

dictionary = [
    {"english": "car", "dutch": "auto"},
    {"english": "house", "dutch": "huis"},
]
```

```
class Translate(Resource):
    @api.marshal_with(translate_response_model, code=200, description="Translate OK")
    @api.marshal_with(translate_response_model, code=404, description="Translate NOK")
    @api.expect(translate_parser, validate=True)
    def get(self):
        args = translate_parser.parse_args()
        word = args["word"]

        search = next((item for item in dictionary if item["english"] == word), False)
        if search:
            return {"translated": search["dutch"]}
        else:
            return {"success": False}, 404

api.add_resource(Translate, "/translate", endpoint="Translate")

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0")
```

What about POST ? – an example

```
word_add_parser = reqparse.RequestParser(bundle_errors=True)
word_add_parser.add_argument("english", required=True, type=str)
word_add_parser.add_argument("dutch", required=True, type=str)
```

```
word_model = api.model(
    "word",
    {
        "english": fields.String(),
        "dutch": fields.String(),
    },
)
```

```
words_response_model = api.model(
    "words", {"words": fields.List(fields.Nested(word_model))}
)
```

```
class Words(Resource):
    @api.expect(word_add_parser, validate=True)
    def post(self):
        args = word_add_parser.parse_args()
        english = args["english"]
        dutch = args["dutch"]

        dictionary.append({"english": english, "dutch": dutch})

        return
```

```
@api.marshal_with(words_response_model, code=200, description="get words response")
def get(self):
    return {"words": dictionary}

api.add_resource(Words, "/words", endpoint="Words")
```

Models	
translate_response >	
words ▾ { words	▾ [word ▾ { english dutch }] string string
word ▾ { english dutch	string string

Responses	
Code	Description
200	get words response
Example Value Model	
{ "words": [{ "english": "string", "dutch": "string" }] }	

Our UI developer gets full insights into the data returned by our API endpoint without having to try the API

Marshaling is nice

```
dictionary = [  
    {"english": "car", "dutch": "auto"},  
    {"english": "house", "dutch": "huis"},  
]
```

```
return {"words": dictionary}
```

```
word_model = api.model(  
    "word",  
    {  
        "english": fields.String(),  
        "dutch": fields.String(),  
    },  
)
```

```
word_model = api.model(  
    "word",  
    {  
        "english": fields.String(),  
    },  
)
```

```
word_model = api.model(  
    "word",  
    {  
        "english": fields.String(),  
        "dutch": fields.String(),  
        "success": fields.Boolean(required=True, choices=(False, True), default=True),  
    },  
)
```

```
{  
    "words": [  
        {  
            "english": "car",  
            "dutch": "auto"  
        },  
        {  
            "english": "house",  
            "dutch": "huis"  
        }  
    ]  
}
```

```
{  
    "words": [  
        {  
            "english": "car"  
        },  
        {  
            "english": "house"  
        }  
    ]  
}
```

```
{  
    "words": [  
        {  
            "english": "car",  
            "dutch": "auto",  
            "success": true  
        },  
        {  
            "english": "house",  
            "dutch": "huis",  
            "success": true  
        }  
    ]  
}
```

Render_template

- Flask can auto-generate a web page
- Based on templates
- Use of Jinja2 template engine
- You can build an entire web-app with it

Return HTML page with our words

```
class Words_UI(Resource):
    def get(self):
        headers = {'Content-Type': 'text/html'}
        return make_response(render_template('words.html', dictionary=dictionary), headers)

api.add_resource(Words_UI, "/words.html", endpoint="Words UI endpoint")
```



English	Dutch
car	auto
house	huis

```
<html>
  <head>
    <title>All known words</title>
  </head>
  <body>
    <table border=1>
      <tr><td>English</td><td>Dutch</td></tr>
      {% for entry in dictionary %}
        <tr>
          <td>{{ entry['english'] }}</td>
          <td>{{ entry['dutch'] }}</td>
        </tr>
      {% endfor %}
    </table>
  </body>
</html>
```

Our use-case

Manage an ACI fabric – node management

- Provide a UI to
 - Show all nodes
 - Remove a node
 - Add a node
- Option 1) Use Flask as API + Javascript UI
- Option 2) Use Flask and render_template

Use Flask as API + Javascript UI

Our API – response model / parser

```
model_node = api.model(
    "node",
    {
        "model": fields.String(),
        "serial": fields.String(),
        "dn": fields.String(),
        "role": fields.String(),
        "name": fields.String(),
        "id": fields.String(),
        "fabricSt": fields.String(),
    },
)

model_nodes = api.model("nodes", {"nodes":
fields.List(fields.Nested(model_node))})
```

Response model nodes

```
gen_response_model = api.model(
    "gen_response",
    {
        "success": fields.Boolean(required=True, choices=(False, True)),
        "message": fields.String(required=True),
    },
)
```

Response model – generic

```
# Node parser
node_parser = reqparse.RequestParser(bundle_errors=True)
node_parser.add_argument("id", required=True, type=int)
```

Delete node parser

```
node_add_parser = node_parser.copy()
node_add_parser.add_argument("serial", required=True, type=str)
node_add_parser.add_argument("name", required=True, type=str)
node_add_parser.add_argument("role", required=True, choices=("leaf", "spine"))
```

Add node parser

Our API – retrieve nodes

```
class node(Resource):  
    """API Class for node."""  
  
    @api.marshal_with(model_nodes, code=200, description="get node response")  
    def get(self):  
        """Retrieve all nodes in an ACI fabric."""  
        logging.debug("Hit node->get")  
        ACI = ACIModule(aci_hostname, aci_username, aci_password)  
        nodes = ACI.get_nodes()  
  
        cleaned_nodes = []  
        for node in nodes["imdata"]:  
            cleaned_nodes.append(node["fabricNode"]["attributes"])  
        # logging.debug(cleaned_nodes)  
  
        return {"nodes": cleaned_nodes}
```

Our API – add node

```
@api.expect(node_add_parser, validate=True)
@api.marshal_with(gen_response_model, code=200, description="Add node response")
def post(self):
    """Add a node to an ACI fabric."""
    logging.debug("Hit node->post")

    args = node_add_parser.parse_args()
    serial = args["serial"]
    nodeId = str(args["id"])
    name = args["name"]
    role = args["role"]

    ACI = ACIModule(aci_hostname, aci_username, aci_password)
    ACI.add_node(serial,nodeId,name,role)

    return {"success": True, "message": "Node succesfully added"}
```

```
graph TD
    A["@api.expect(node_add_parser, validate=True)"] --> C["args = node_add_parser.parse_args()"]
    B["@api.marshal_with(gen_response_model, code=200, description='Add node response')"] --> D["return {'success': True, 'message': 'Node succesfully added'}"]
    C --> E["ACI.add_node(serial,nodeId,name,role)"]
    F["nodeId = str(args['id'])"] --> E
```

Our API – delete node

```
@api.expect(node_parser, validate=True)
@api.marshal_with(gen_response_model, code=200, description="Delete node response")
def delete(self):
    """Remove a node from an ACI fabric."""
    args = node_parser.parse_args()
    nodeid = str(args["id"])

    ACI = ACIModule(aci_hostname, aci_username, aci_password)
    ACI.delete_node(nodeid)

    return {"success": True, "message": "Node successfully removed"}
```

```
graph TD
    A["@api.expect(node_parser, validate=True)"] --> C["args = node_parser.parse_args()<br/>nodeid = str(args['id'])"]
    B["@api.marshal_with(gen_response_model, code=200, description='Delete node response')"] --> D["return {'success': True, 'message': 'Node successfully removed'}"]
    C --> D
```

Our API – versioning

```
api.add_resource(node, "/api/v0/node", endpoint="node")
```

Host through Apache – main vhost

```
<virtualhost 10.48.31.106:80>
Header always set Access-Control-Allow-Origin "*"
Header always set Access-Control-Allow-Methods "POST, GET, OPTIONS, DELETE, PUT"
Header always set Access-Control-Max-Age "1000"
Header always set Access-Control-Allow-Headers "x-requested-with, Content-Type, origin, authorization, accept, client-security-token"
Header always set Access-Control-Allow-Credentials "true"

DocumentRoot /var/www/html
ServerName rs-coding.cisco.com

CustomLog /var/log/httpd/rs-coding.cisco.com-access.log combined
ErrorLog /var/log/httpd/rs-coding.cisco.com-error.log

<location /api/>
    ProxyPassReverse http://127.0.0.1:4242/api/
    ProxyPass http://127.0.0.1:4242/api/
</Location>
<location /doc>
    ProxyPassReverse http://127.0.0.1:4242/doc
    ProxyPass http://127.0.0.1:4242/doc
</Location>

<Location "/swaggerui">
    ProxyPassReverse http://127.0.0.1:4242/swaggerui
    ProxyPass http://127.0.0.1:4242/swaggerui
</Location>
<Location "/swagger.json">
    ProxyPassReverse http://127.0.0.1:4242/swagger.json
    ProxyPass http://127.0.0.1:4242/swagger.json
</Location>

</VirtualHost>
```

You can also host
through nginx or
any other proxy

Don't just run, use WSGI

- Flask is a micro-web framework for Python
 - Flask is not a webserver
- Use WSGI to allow Apache to interact with Python



Host through Apache – WSGI vhost

```
Listen 4242
<VirtualHost 127.0.0.1:4242>
```

```
WSGIDaemonProcess devnet_flask_api user=rsmeyers group=rsmeyers threads=15
WSGIScriptAlias / /home/rsmeyers/devnet-flask-api/devnet_flask_api.wsgi
WSGIScriptReloading On
```

```
<Directory /home/rsmeyers/devnet-flask-api>
    WSGIProcessGroup devnet_flask_api
    WSGIApplicationGroup %{GLOBAL}
    Require all granted
</Directory>
</VirtualHost>
```

```
[rsmeyers@rs-coding devnet-flask-api]$ pwd
/home/rsmeyers/devnet-flask-api
[rsmeyers@rs-coding devnet-flask-api]$ cat devnet_flask_api.wsgi
#!/usr/bin/python3
```

```
import sys
sys.path.insert(0, '/home/rsmeyers/devnet-flask-api')

sys.stdout = sys.stderr

from devnet_flask_api import app as application
```

rs-coding.cisco.com/doc

DELETE /api/v0/node Remove a node from an ACI fabric

GET /api/v0/node Retrieve all nodes in an ACI fabric

Parameters

Name **Description**

X-Fields
string(\$mask) An optional fields mask
(header)

X-Fields

Execute **Clear**

Responses Response content type: application/json

Inspector Console Debugger Network Style Editor Performance Memory Storage Accessibility Application

Filter URLs

Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	rs-coding.cisco.com	node	swagger-ui-bundle.js:2 (fetch)	json	2.37 kB	1.86...

Headers Cookies Request Response Timings Stack Trace

Filter headers

GET http://rs-coding.cisco.com/api/v0/node

Status: 200 OK

Version: HTTP/1.1

Transferred: 2.37 kB (1.86 kB size)

Referrer Policy: strict-origin-when-cross-origin

Request Priority: Highest

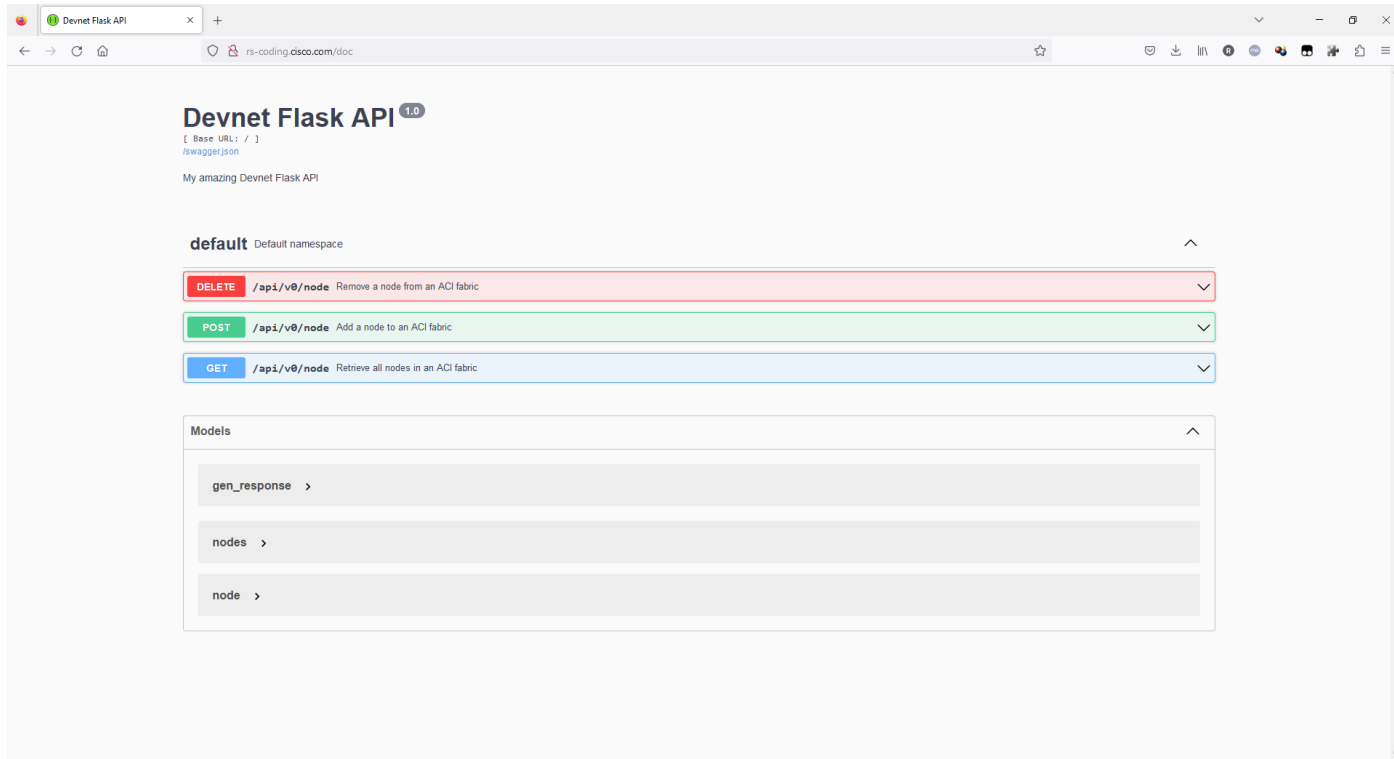
Response Headers (508 B) Raw

- Access-Control-Allow-Credentials: true
- Access-Control-Allow-Headers: x-requested-with, Content-Type, origin, authorization, accept, client-security-token
- Access-Control-Allow-Methods: POST, GET, OPTIONS, DELETE, PUT

1 request 1.86 kB / 2.37 kB transferred Finish: 301 ms

Proxied by apache

Our swagger page

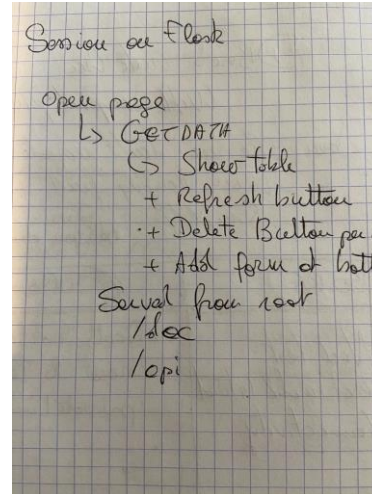
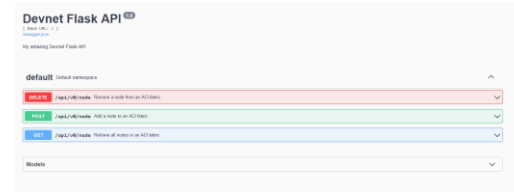


Meet our UI developer

- Expert in Angular/React/JS/...
- No prior DC knowledge
- His only input
 - <http://rs-coding.cisco.com/doc>
 - A quick Webex discussion



Didier Colens



Javascript to the rescue

API Code

```
export default class MyAmazingDevnetFlaskApi {
  constructor(baseUrl = 'http://rs-coding.cisco.com/api/v0/node') {
    this.baseUrl = baseUrl;
  }

  async getNodes() {
    const response = await fetch(this.baseUrl);
    return response.json();
  }

  async deleteNode(id) {
    const searchParams = new URLSearchParams({ id });

    const response = await fetch(`${this.baseUrl}?${searchParams.toString()}`, {
      method: 'DELETE',
    });

    return response.json();
  }

  async addNode(node) {
    const searchParams = new URLSearchParams(node);

    const response = await fetch(`${this.baseUrl}?${searchParams.toString()}`, {
      method: 'POST',
    });

    return response.json();
  }
}
```

Delete row code

```
document.getElementById('table').addEventListener('click', async (event) => {
  if (event.target.classList.contains('deleteRow')) {
    event.preventDefault();
    console.log(`delete for ${event.target.getAttribute('data-id')} clicked`);

    const button = event.target;

    try {
      button.disabled = true;
      await api.deleteNode(button.getAttribute('data-id'));
      table.removeRow(button.getAttribute('data-rowIndex'));
    } catch (error) {
      console.error(
        `failed to delete ${button.getAttribute('data-id')}`,
        error,
      );
    } finally {
      button.disabled = false;
    }
  }
});
```

The result

My amazing Devnet Flask API

Search...

model	serial	dn	role	name	id		fabricSt
N9K-C93180YC-FX	FDO24310LXQ	topology/pod-1/node-103	leaf	bdsol-aci12-leaf3	103	Delete	active
N9K-C93180YC-FX	FDO262210BJ	topology/pod-1/node-105	leaf	bdsol-aci12-leaf5	105	Delete	active
N9K-C93180YC-FX	FDO24311CJN	topology/pod-1/node-104	leaf	bdsol-aci12-leaf4	104	Delete	active
N9K-C9332C	FDO241800MS	topology/pod-1/node-201	spine	bdsol-aci12-spine1	201	Delete	active
APIC-SERVER-M3	WZP24390JH5	topology/pod-1/node-2	controller	bdsol-aci12-apic2	2	Delete	unknown
N9K-C93180YC-FX	FDO24311CJS	topology/pod-1/node-101	leaf	bdsol-aci12-leaf1	101	Delete	active
N9K-C93180YC-FX	FDO243119M0	topology/pod-1/node-102	leaf	bdsol-aci12-leaf2	102	Delete	active
N9K-C93180YC-FX	FDO26230LOY	topology/pod-1/node-106	leaf	bdsol-aci12-leaf6	106	Delete	active
N9K-C9332C	FDO24180LQW	topology/pod-1/node-202	spine	bdsol-aci12-spine2	202	Delete	active
APIC-SERVER-M3	WZP241211TR	topology/pod-1/node-1	controller	bdsol-aci12-apic1	1	Delete	unknown
APIC-SERVER-L3	WZP232807BD	topology/pod-1/node-3	controller	bdsol-aci12-apic3	3	Delete	unknown

Add a new node

id

serial

name

role

What if I do not
know Javascript

Doing it all in Python / Flask

- Flask is a micro-web framework for Python
- Look at it as the easy way for a Python coder to build an entire webapp
- Use `render_template` to auto generate web pages

Use Flask and render_template

Our new endpoint – serves an html page

```
class Start(Resource):
    def get(self):
        """Retrieve all nodes in an ACI fabric."""
        ACI = ACIModule(aci_hostname, aci_username, aci_password)
        nodes = ACI.get_nodes()

        headers = {'Content-Type': 'text/html'}
        return make_response(render_template('start.html', nodes=nodes), headers)

@api.expect(node_add_parser, validate=True)
def post(self):
    """Add a node to an ACI fabric."""
    logging.debug("Hit node->post")

    args = node_add_parser.parse_args()
    serial = args["serial"]
    nodeId = str(args["id"])
    name = args["name"]
    role = args["role"]

    ACI = ACIModule(aci_hostname, aci_username, aci_password)
    ACI.add_node(serial, nodeId, name, role)

    sleep(1)

    nodes = ACI.get_nodes()
    headers = {'Content-Type': 'text/html'}
    return make_response(render_template('start.html', nodes=nodes), headers)
```

```
@api.expect(node_parser, validate=True)
def delete(self):
    """Remove a node from an ACI fabric."""

    args = node_parser.parse_args()
    nodeId = str(args["id"])

    ACI = ACIModule(aci_hostname, aci_username, aci_password)
    ACI.delete_node(nodeId)

    nodes = ACI.handle_req("get", "node/class/fabricNode.json")
    headers = {'Content-Type': 'text/html'}
    return make_response(render_template('start.html', nodes=nodes), headers)

api.add_resource(Start, "/api/v0/nodes.html", endpoint="Nodes HTML page")
```

- Same code as before
- No Marshall
- Return HTML page

Out HTML template

```
<html>
  <head>
    <title>All known nodes</title>
    <script type="module" src="http://rs-coding.cisco.com/index-render.js"></script>
  </head>
  <body>
    <h1>All nodes</h1>
    <table border=1>
      <tr>
        <td>ID</td>
        <td>Name</td>
        <td>Model</td>
        <td>Serial</td>
        <td>DN</td>
        <td>Role</td>
        <td>fabricSt</td>
        <td>Delete</td>
      </tr>
      {% for entry in nodes['imdata'] %}
        <tr>
          <td>{{ entry['fabricNode']['attributes']['id'] }}</td>
          <td>{{ entry['fabricNode']['attributes']['name'] }}</td>
          <td>{{ entry['fabricNode']['attributes']['model'] }}</td>
          <td>{{ entry['fabricNode']['attributes']['serial'] }}</td>
          <td>{{ entry['fabricNode']['attributes']['dn'] }}</td>
          <td>{{ entry['fabricNode']['attributes']['role'] }}</td>
          <td>{{ entry['fabricNode']['attributes']['fabricSt'] }}</td>
          <td>
            <button class="deleteRow" type="button" fabricNode="{{
entry['fabricNode']['attributes']['id'] }}" >Delete</button>
          </td>
        </tr>
      {% endfor %}
    </table>
```

```
<h2>Add node</h2>
<form action="nodes.html" method=post>
  <ul>
    <li>
      <label>Number:</label>
      <input type="number" name="id" />
    </li>
    <li>
      <label>Serial:</label>
      <input type="string" name="serial" />
    </li>
    <li>
      <label>Name:</label>
      <input type="string" name="name" />
    </li>
    <li>
      <label>Role:</label>
      <select name="role">
        <option value="leaf">leaf</option>
        <option value="spine">spine</option>
      </select>
    </li>
  </ul>
  <input type="submit" value=add >
</form>
</body>
</html>
```

POST is easy
DELETE requires a trick

HTML only knows GET and POST

HTML

```
<button class="deleteRow" type='button' fabricNode="{ { entry['fabricNode'] ['attributes'] ['id'] } }" >Delete</button>
```

index-render.js

```
document.addEventListener('click', async (event) => {  
  if (event.target.classList.contains('deleteRow')) {  
    event.preventDefault();  
    console.log(`delete for ${event.target.getAttribute('fabricNode')} clicked`);  
  
    const button = event.target;  
  
    try {  
      var baseUrl = 'http://rs-coding.cisco.com/api/v0/nodes.html';  
      const response = await fetch(`${baseUrl}?id=${event.target.getAttribute('fabricNode')}`, {  
        method: 'DELETE',  
      });  
      window.location.href = baseUrl;  
    } catch (error) {  
      console.error(  
        `failed to delete ${button.getAttribute('data-id')}`,  
        error,  
      );  
    } finally {  
      button.disabled = false;  
    }  
  }  
});
```

Perform delete

Refresh page

Use a framework, Angular, React, ...

The result

All known nodes

rs-coding.cisco.com/api/v0/nodes.html

120%

All nodes

ID	Name	Model	Serial	DN	Role	fabricSt	Delete
103	bdsol-aci12-leaf3	N9K-C93180YC-FX	FDO24310LXQ	topology/pod-1/node-103	leaf	active	Delete
105	bdsol-aci12-leaf5	N9K-C93180YC-FX	FDO262210BJ	topology/pod-1/node-105	leaf	active	Delete
201	bdsol-aci12-spine1	N9K-C9332C	FDO241800MS	topology/pod-1/node-201	spine	active	Delete
2	bdsol-aci12-apic2	APIC-SERVER-M3	WZP24390JH5	topology/pod-1/node-2	controller	unknown	Delete
101	bdsol-aci12-leaf1	N9K-C93180YC-FX	FDO24311CJS	topology/pod-1/node-101	leaf	active	Delete
102	bdsol-aci12-leaf2	N9K-C93180YC-FX	FDO243119M0	topology/pod-1/node-102	leaf	active	Delete
106	bdsol-aci12-leaf6	N9K-C93180YC-FX	FDO26230L0Y	topology/pod-1/node-106	leaf	active	Delete
104	bdsol-aci12-leaf4	N9K-C93180YC-FX	FDO24311CJN	topology/pod-1/node-104	leaf	active	Delete
202	bdsol-aci12-spine2	N9K-C9332C	FDO24180LQW	topology/pod-1/node-202	spine	active	Delete
3	bdsol-aci12-apic3	APIC-SERVER-L3	WZP232807BD	topology/pod-1/node-3	controller	unknown	Delete
1	bdsol-aci12-apic1	APIC-SERVER-M3	WZP241211TR	topology/pod-1/node-1	controller	unknown	Delete

Add node

- Number:
- Serial:
- Name:
- Role:

add

Our swagger page

Devnet Flask API ^{1.1}

[Base URL: /]
</swagger.json>

My amazing Devnet Flask API

default Default namespace ^

GET /api/v0/node Retrieve all nodes in an ACI fabric

POST /api/v0/node Add a node to an ACI fabric

DELETE /api/v0/node Remove a node from an ACI fabric

GET /api/v0/nodes.html Retrieve all nodes in an ACI fabric

POST /api/v0/nodes.html Add a node to an ACI fabric

DELETE /api/v0/nodes.html Remove a node from an ACI fabric

Models

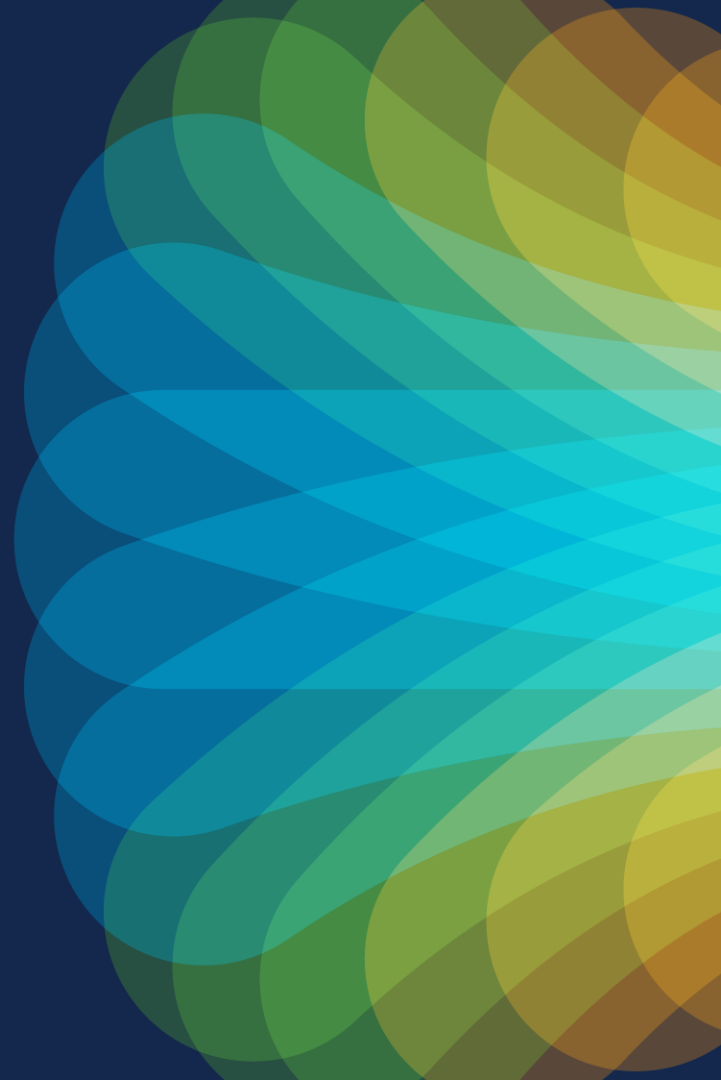
Demo



The bridge to possible

Thank you

CISCO *Live!*



The background features a vibrant, multi-colored abstract design. On the left, there are horizontal, wavy bands of color in shades of red, orange, yellow, and green. On the right, a bright white light source emits a series of sharp, radiating lines in various colors, including blue, green, and yellow, creating a sunburst effect.

cisco *Live!*

Let's go