
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

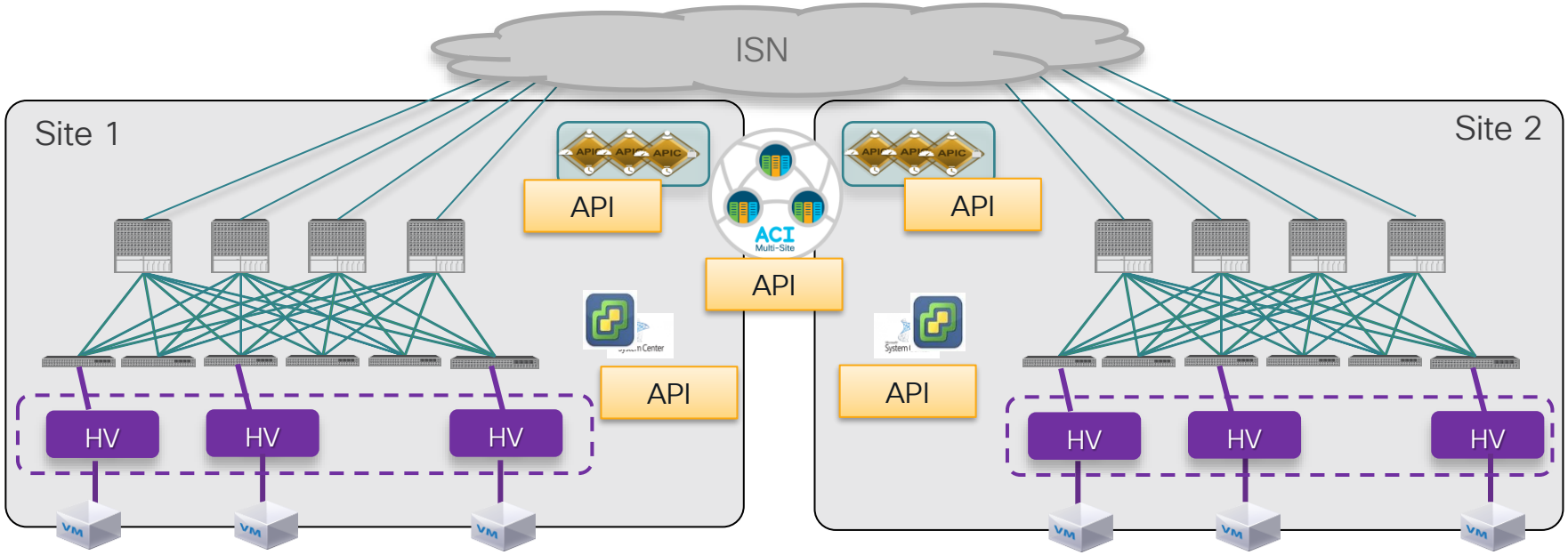
cisco Live!

DEVNET-2158

Agenda

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

Average DC landscape



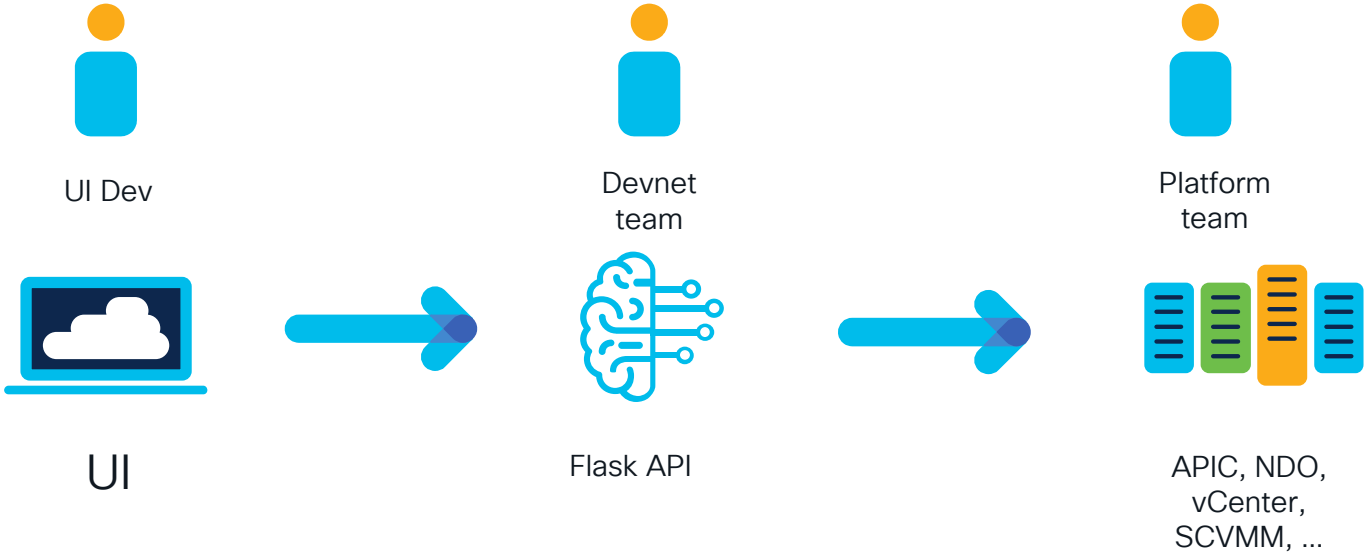
Why

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

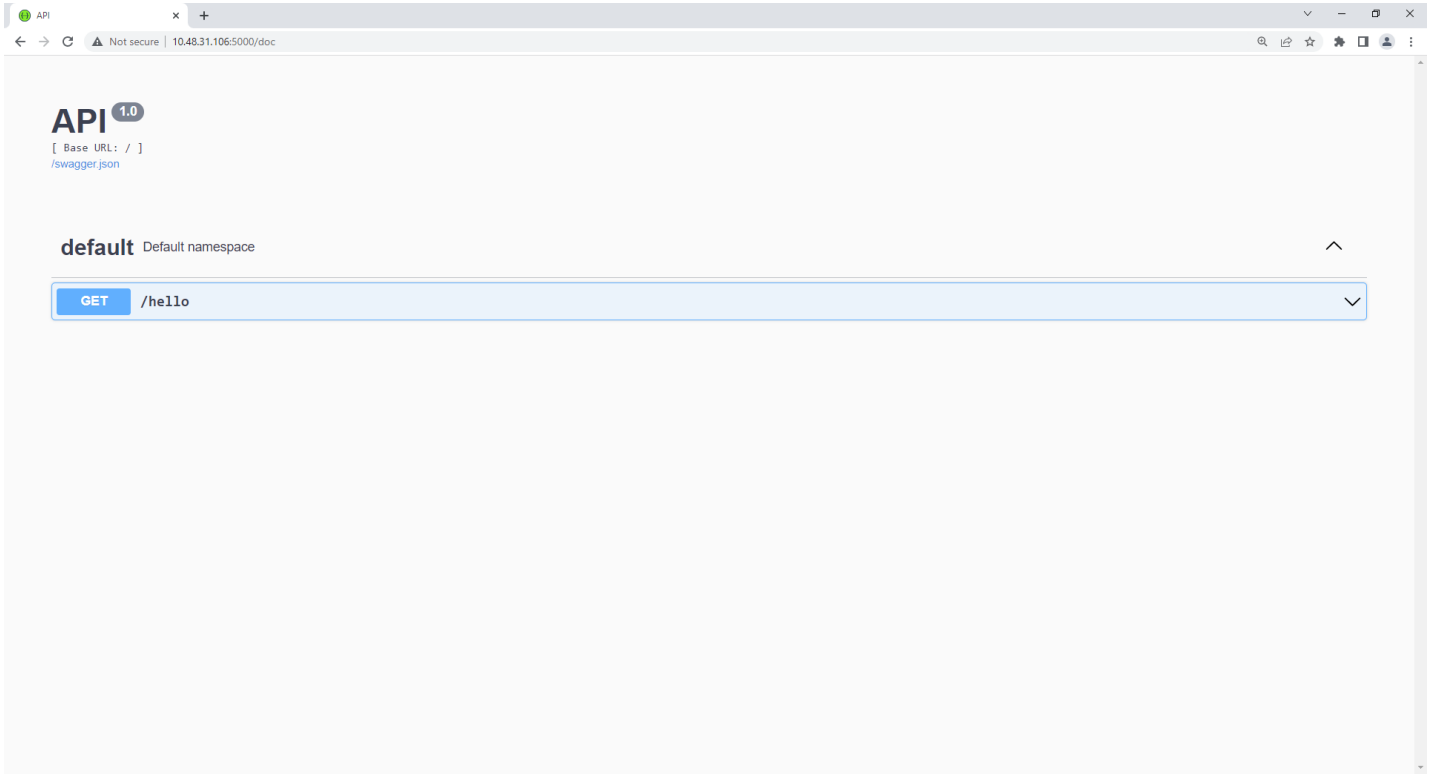
```
$ python3 api.py
 * Serving Flask app 'api'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a
production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.48.31.106:5000
Press 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 -
```

```
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")
```

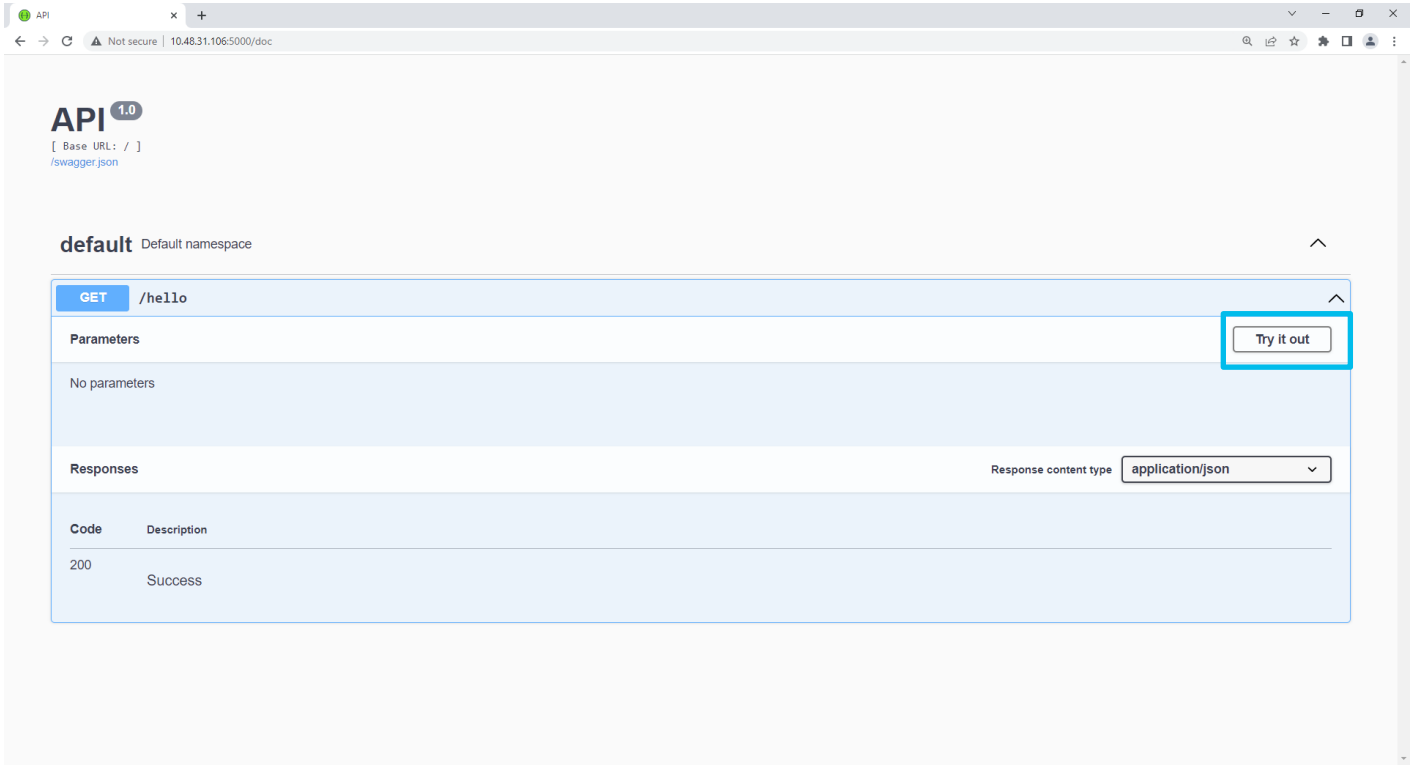
Auto generate docs

Listen on all IP addresses → only use for DEV

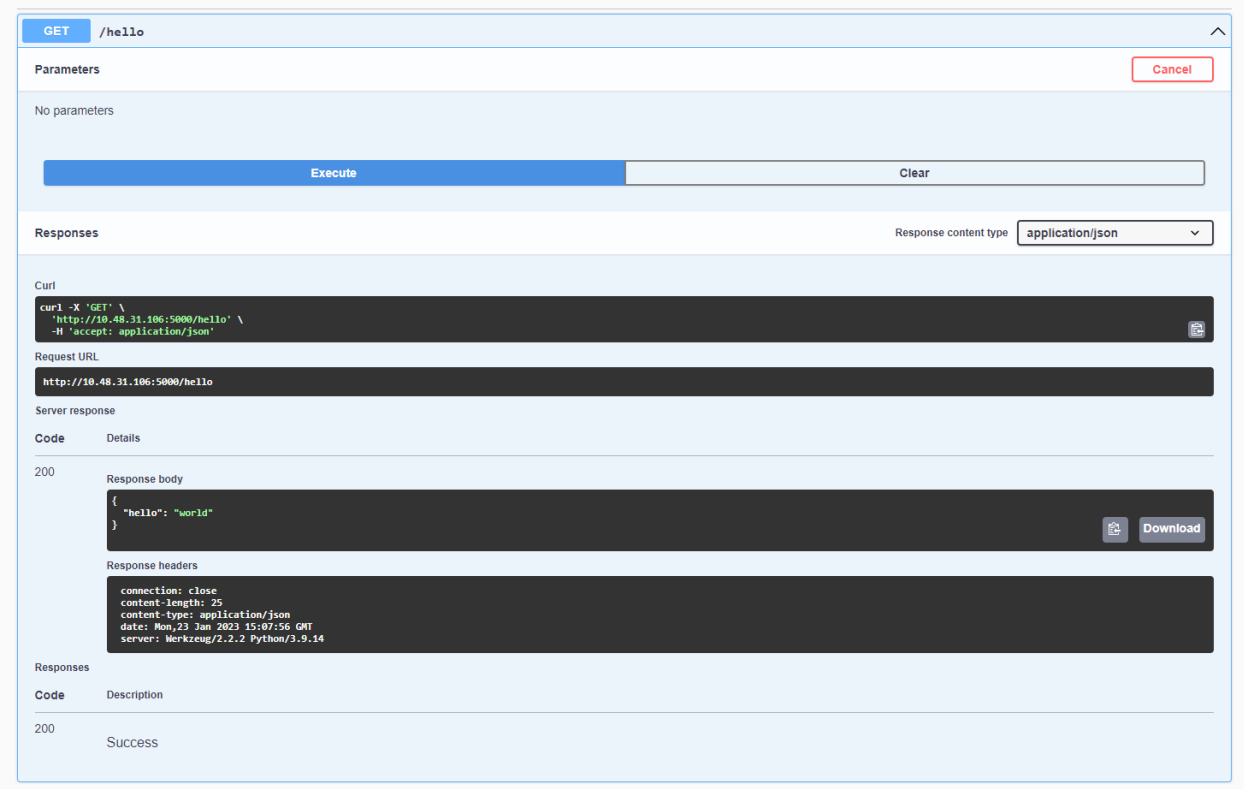
Documentation – swagger UI



Documentation – swagger UI



Swagger UI – Try it out



Request Parsing

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

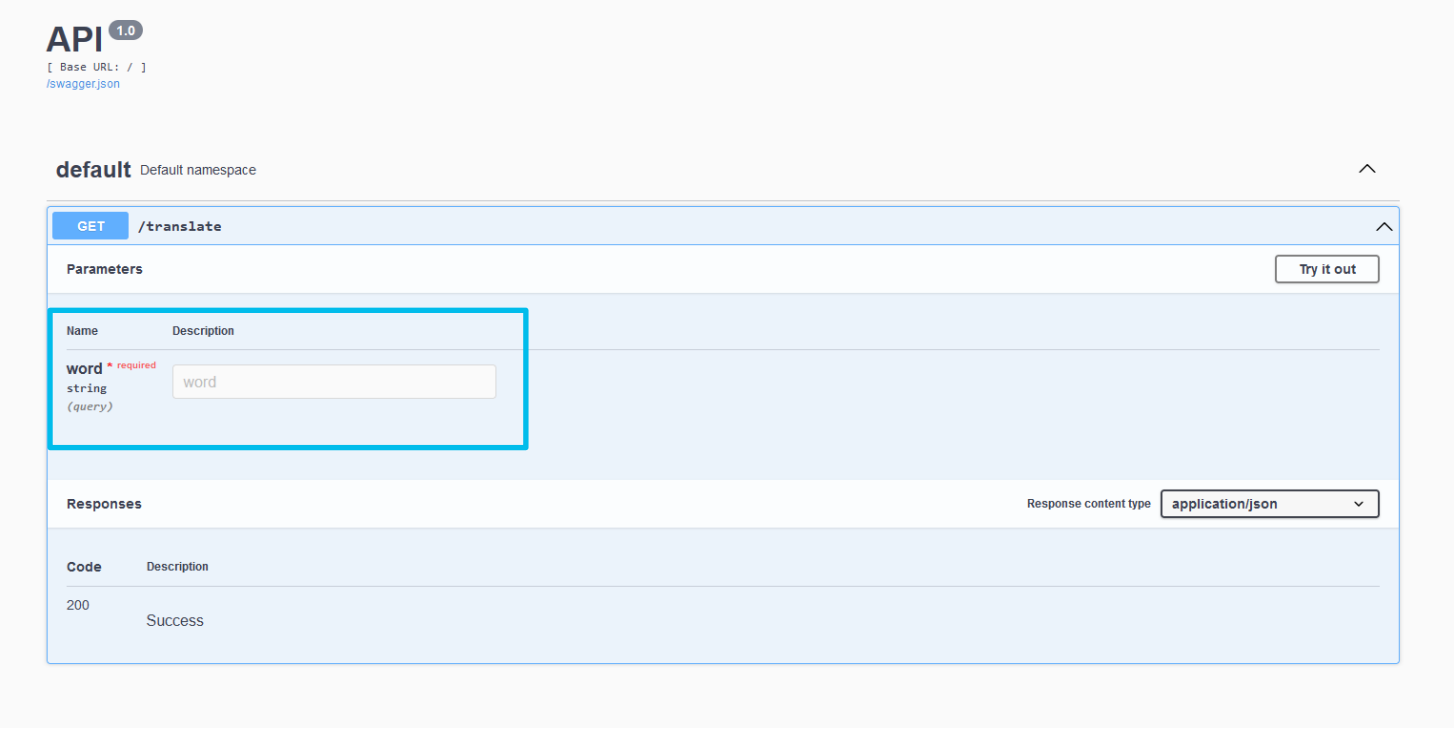
API Expect – control input

```
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



Response marshalling

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

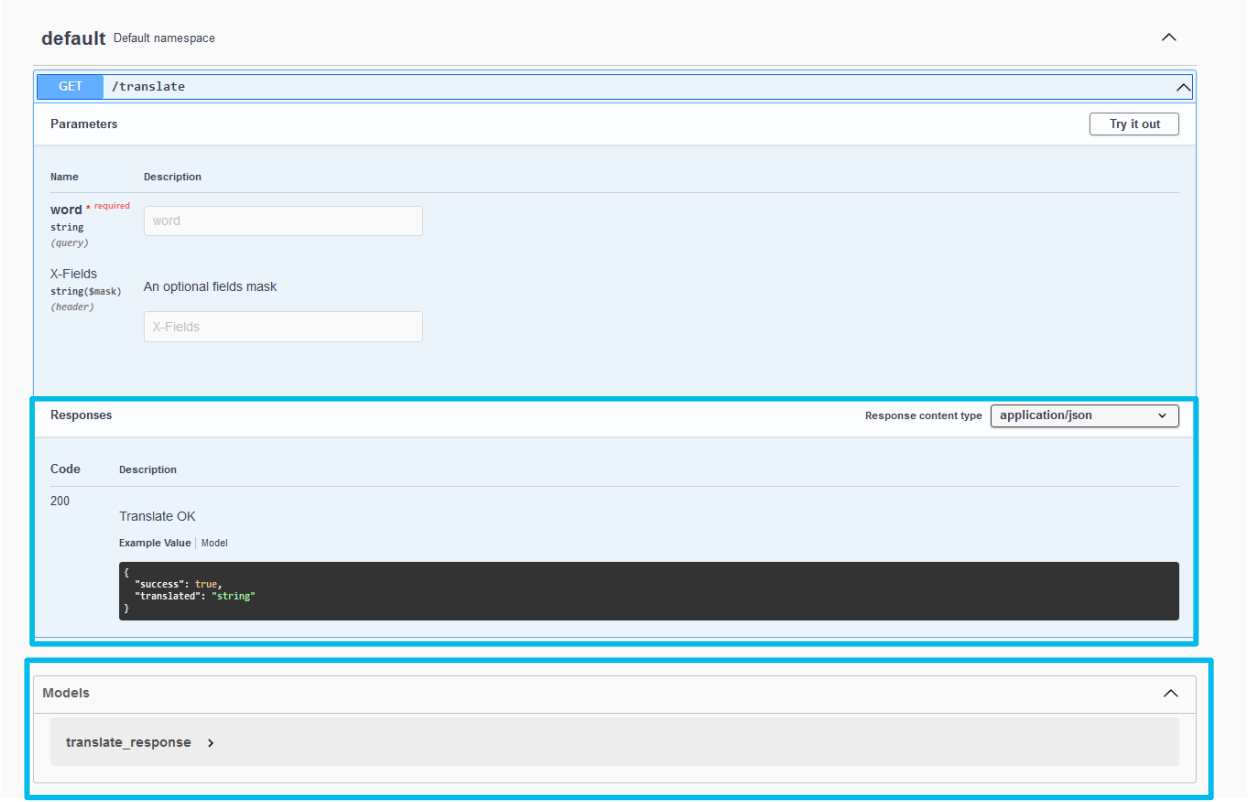
API Marshal – control output

```
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")
```

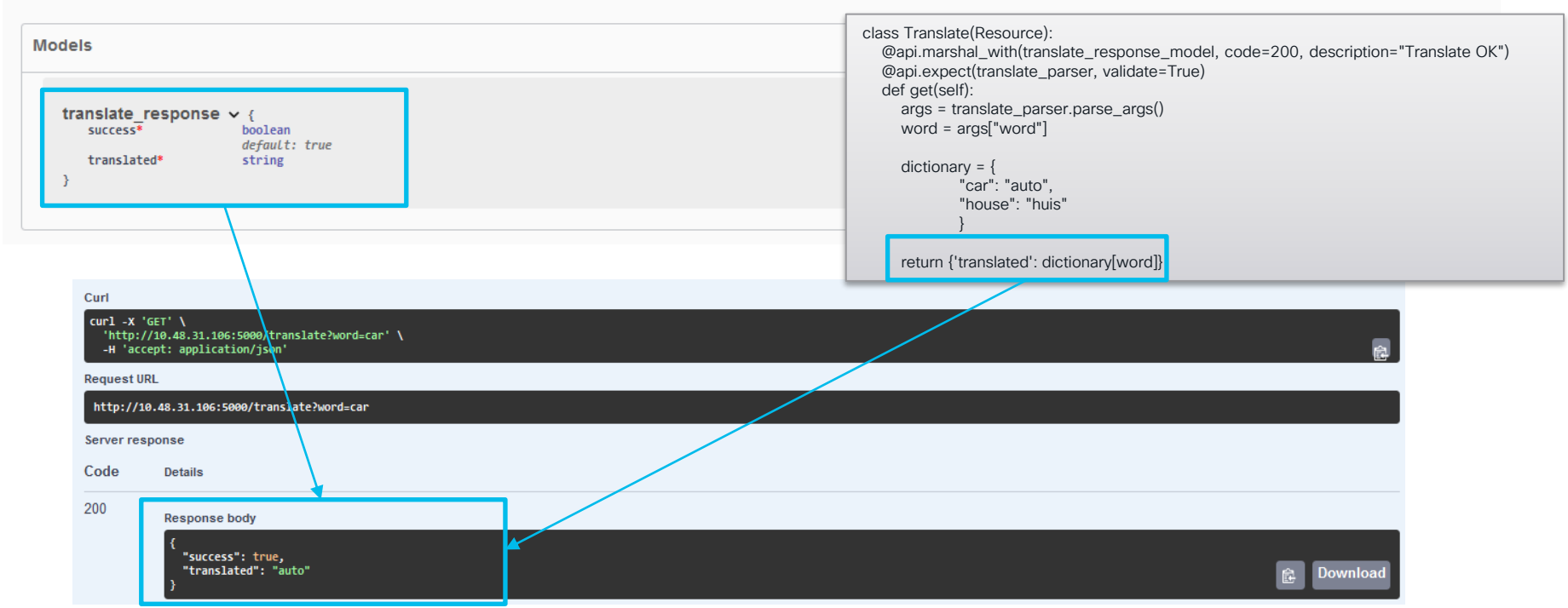
Create response model

Enfore model

API Marshal – control output – swagger



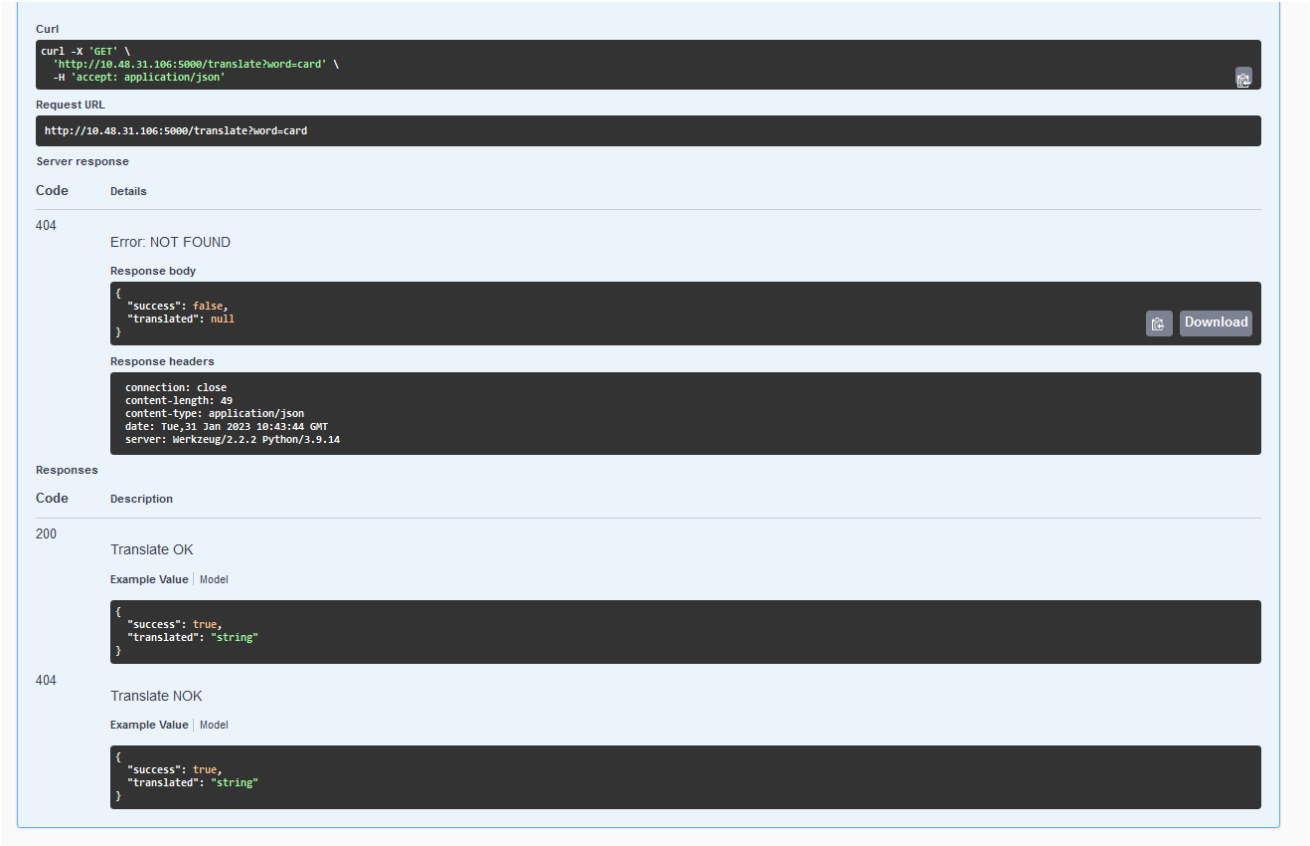
API Marshal – control output – swagger



API Marshal – there's more then 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 then 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

```
{
  "success": false,
  "translated": null
}
```

Download

Response headers

```
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
```

Responses

Code	Description
200	Translate OK

Example Value | Model

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

Code	Description
404	Translate NOK

Example Value | Model

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

What about POST ? – quick code-refactor

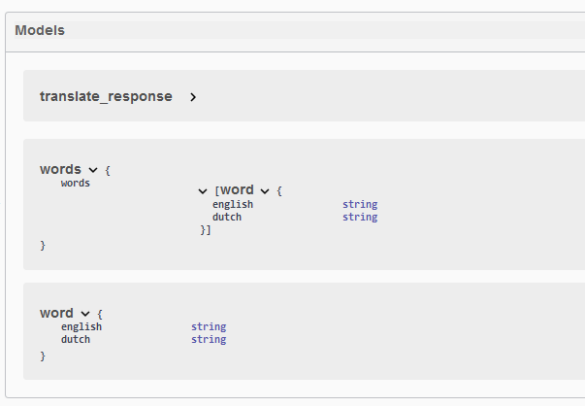
```
from flask import Flask
from flask_restx import Resource, Api, reqparse, fields, marshal

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"},
]
```

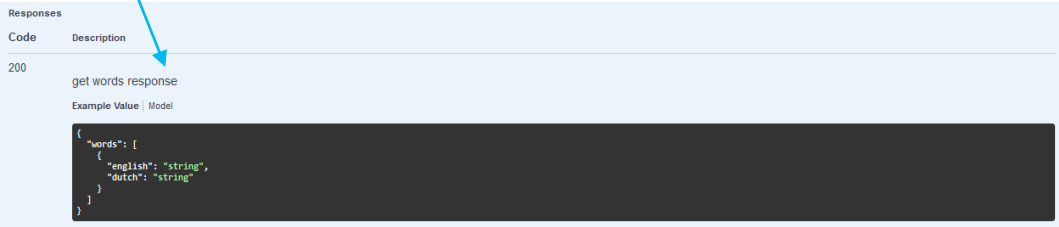
```
class Translate(Resource):
    @api.marshal_with(translate_response_model, code=200, description="Translate OK")
    @api.marshal_with(translate_response_model, code=404, description="Translate NOK")
    @api.expect(translate_parser, validate=True)
    def get(self):
        args = translate_parser.parse_args()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





Our UI developper 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}

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

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

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

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

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

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

```
<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))})
```

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

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

```
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"))
```

Response model nodes

Response model – generic

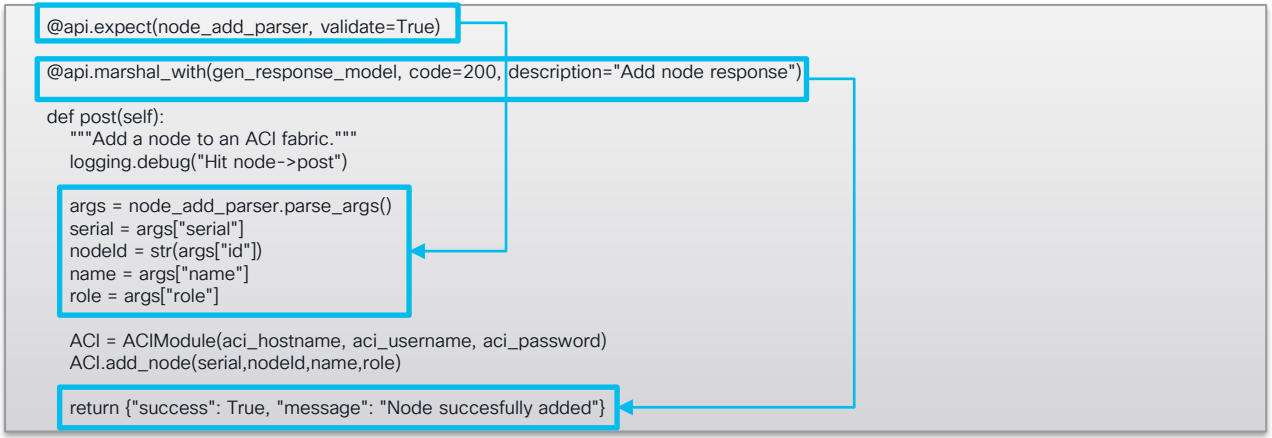
Delete node parser

Add node parser

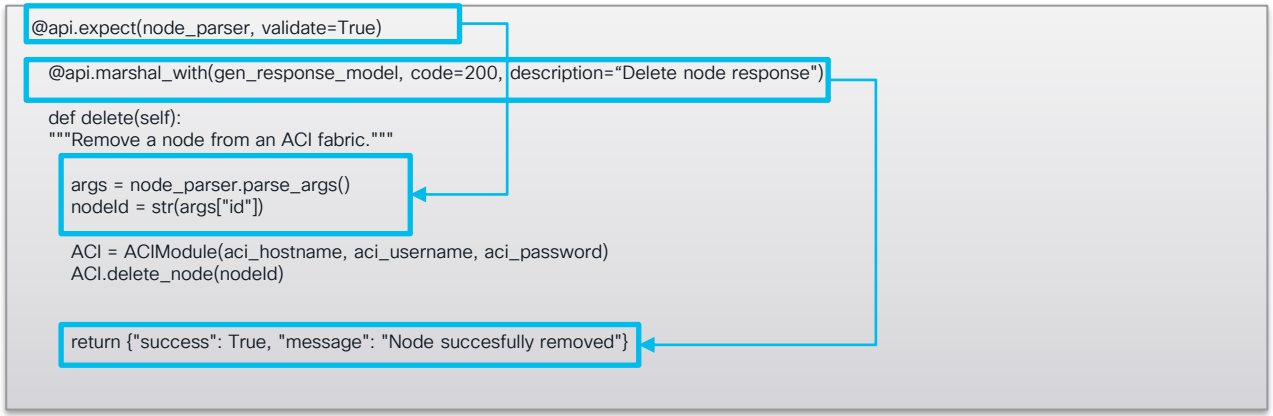
Our API – retrieve nodes

```
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



Our API – delete node



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

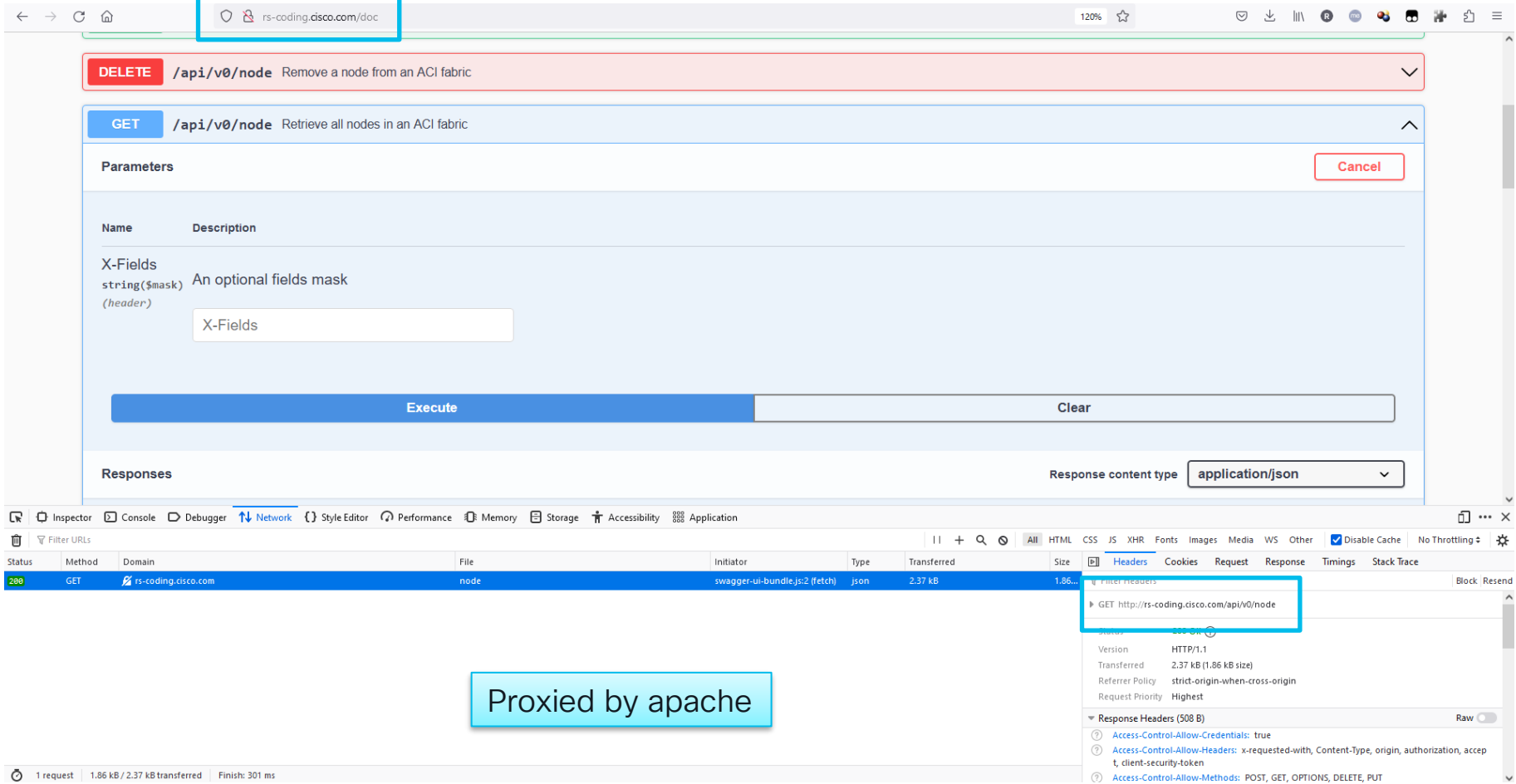
- Flask is a micro-web framework for Python
  - Flask is not a webserver

- Use WSGI to allow Apache to interact with Python

Host through Apache – WSGI vhost

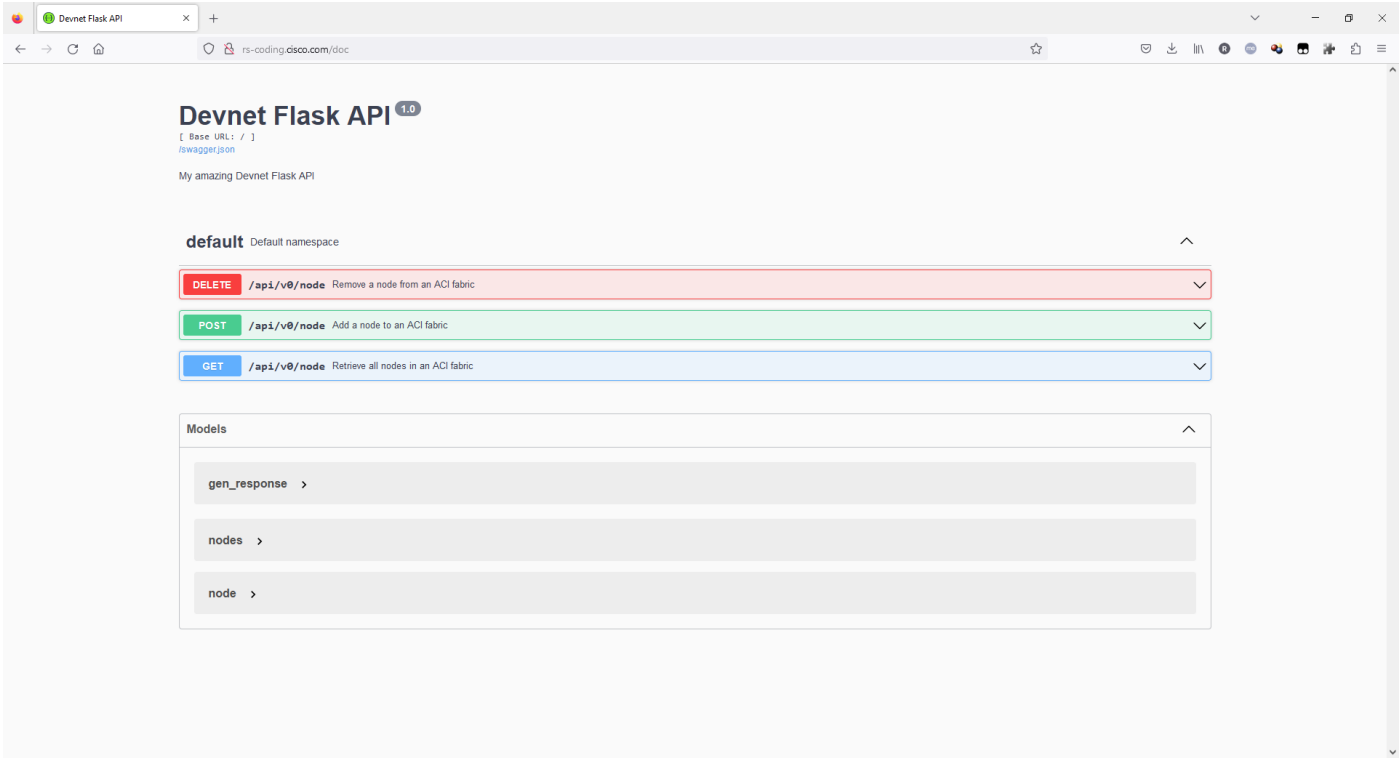
```
Listen 4242
<VirtualHost 127.0.0.1:4242>

    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
```



Proxied by apache

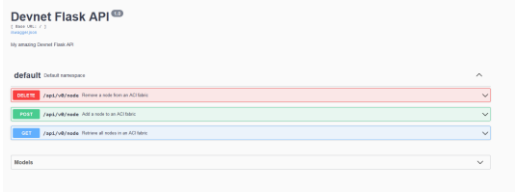
Our swagger page

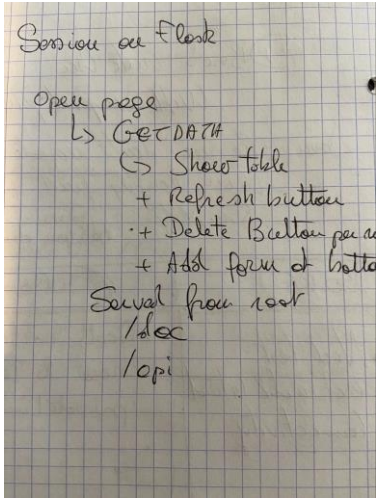


Meet our UI developer

- 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

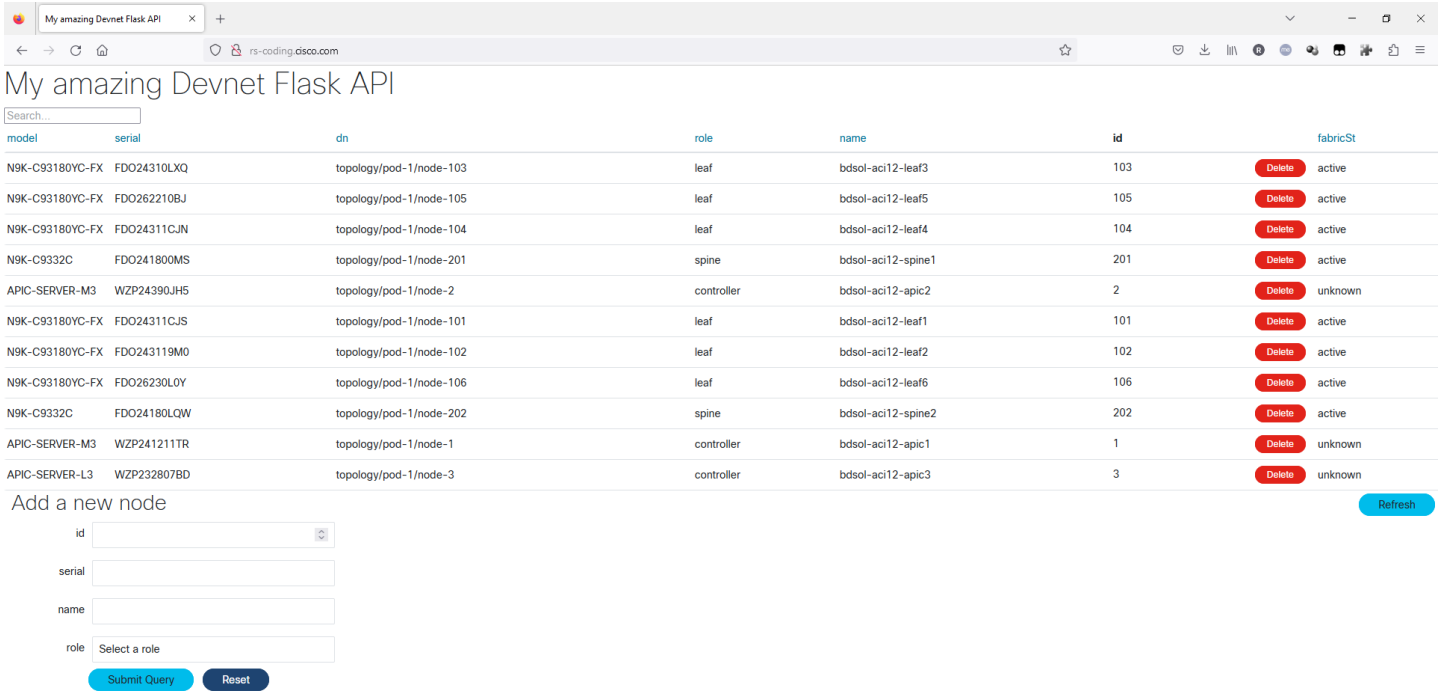
API Code

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

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

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

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

    return response.json();
  }

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

    const response = await fetch(`${this.baseUri}?${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



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

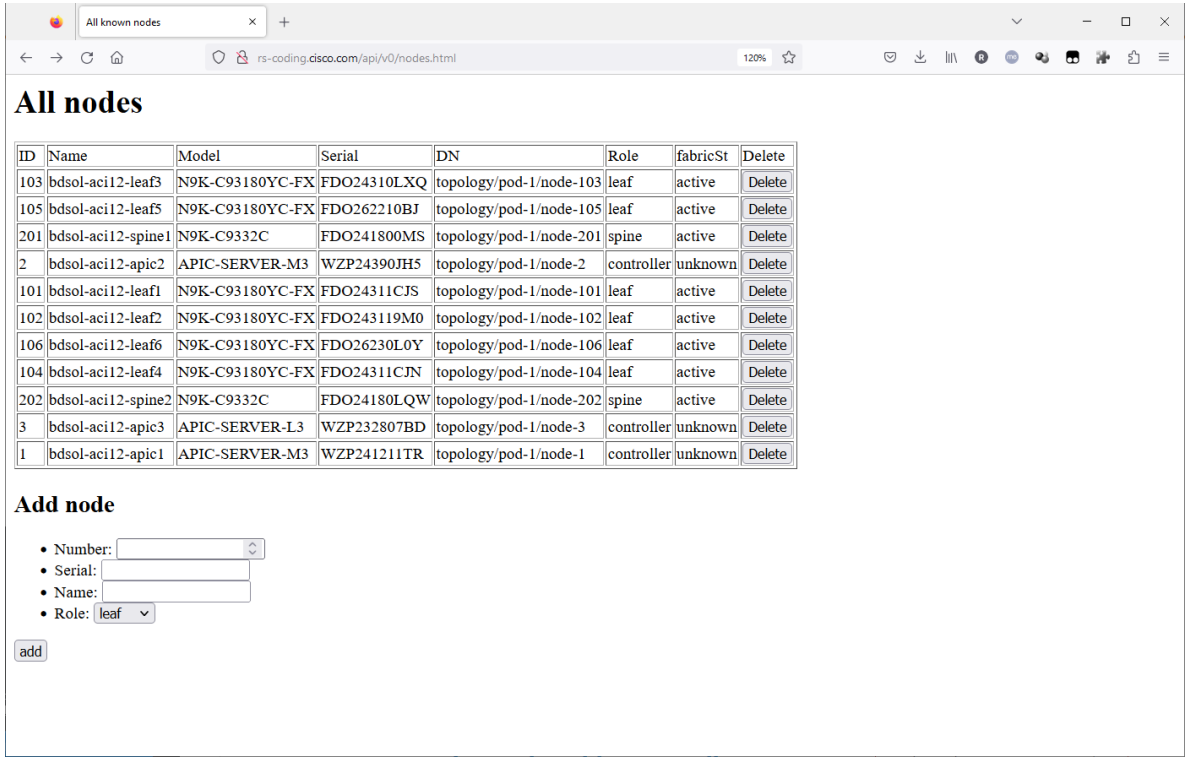
```
document.addEventListener('click', async (event) => {
    if (event.target.classList.contains('deleteRow')) {
        event.preventDefault();
        console.log(`delete for ${event.target.getAttribute('fabricNode')} clicked`);

        const button = event.target;

        try {
            var baseUri = 'http://rs-coding.cisco.com/api/v0/nodes.html';
            const response = await fetch(`${baseUri}?id=${event.target.getAttribute('fabricNode')}`, {
                method: 'DELETE',
            });
            window.location.href = baseUri;

        } catch (error) {
            console.error(
                `failed to delete ${button.getAttribute('data-id')}`,
                error,
            );
        } finally {
            button.disabled = false;
        }


    }
});
```

Perform delete

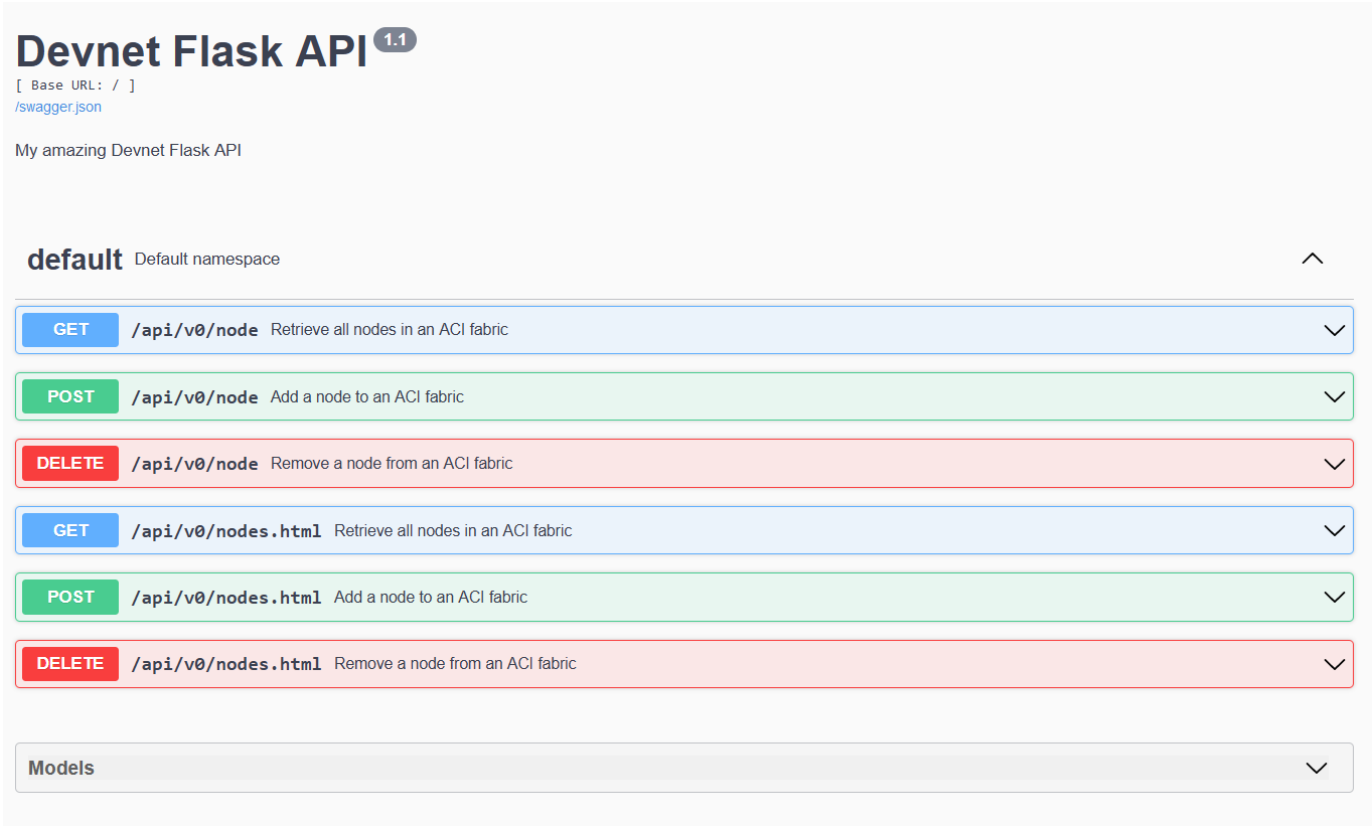
Refresh page

Use a framework, Angular, React, ...

The result



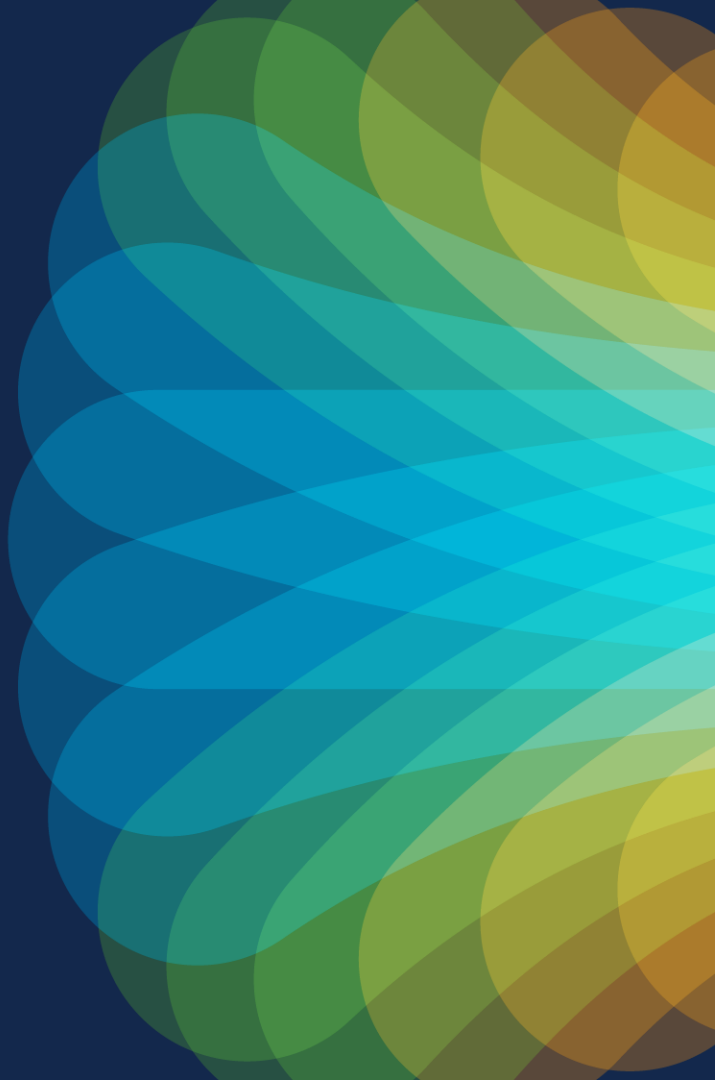
Our swagger page



Demo

The bridge to possible

Thank you



cisco Live!


cisco Live!
Let's go