APIs are getting larger, how do Overlays help?

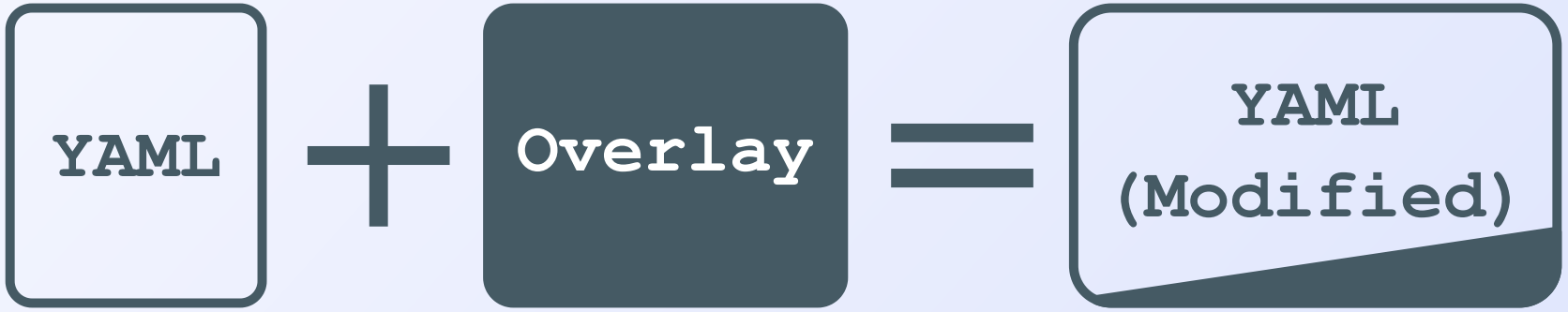
They help separate concerns.

Before we begin

And in the interest of being polite...

Overlays - tl;dr

They're YAML files that take a YAML document as input and produce a modifed document as output.



Warm up

I'm Josh Ponelat

(Pah-neh-lat)

- Lead on Swagger (open source) at SmartBear.
- Product manager on SwaggerHub.
- But mostly a tool-maker.

@jponelat
https://ponelat.com

This talk

- The *other* people involved in APIs
  - Docs, DevOps and PMs
- **Overlays:** How they work
- Shower thoughts
  - APIs as Pets or Toys?
  - A tool or a standard?
  - Anti-patterns and pitfalls

Level setting.

- Technically Overlays work with YAML, not APIs.
- OpenAPI/AsyncAPI/JSON Schema are all YAML(ish) based.
- I'll use the word API to mean API Definition or Document, for this talk.
- If in doubt I'm usually referring to OpenAPI, not AsyncAPI/JSON Schema. Force of habit.

This page intentionally left blank

> API design is no longer the concern of one person.
> Different areas are handled by different folks.

Archimedes, 250 BCE

It's not just about Barry

Barry is a back-end engineer.

API design is now more than the shape of the API.

The *other* people

Let's call her, Ellen

Documentation writer

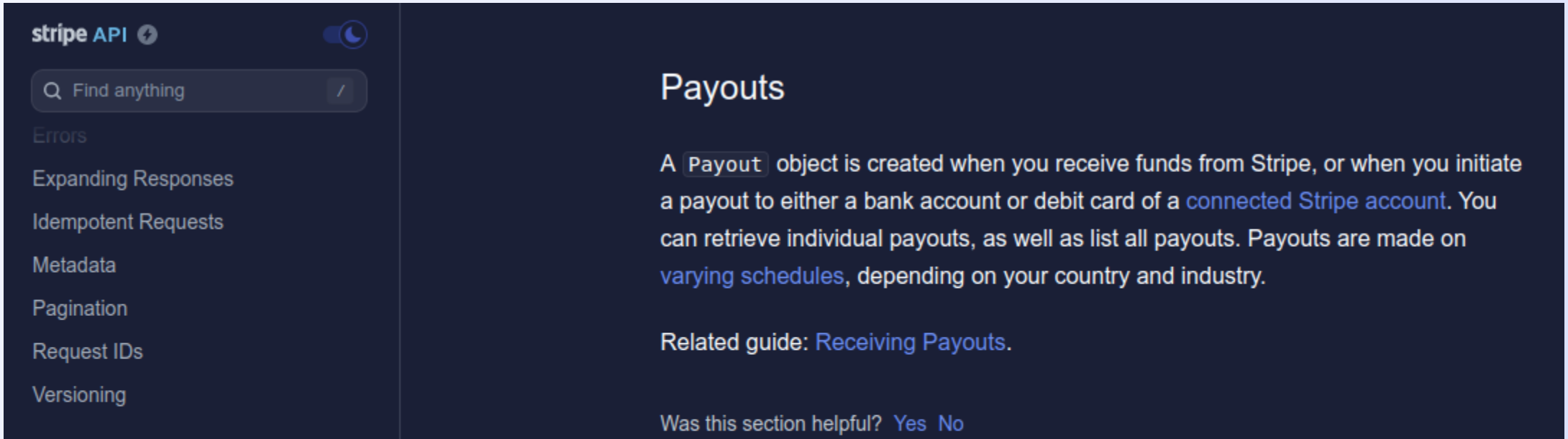
Responsible for
Documentation and translations.

Cares about
Written word, developer experience and accuracy.

Pains (related)
Access to source files, copying changes to curated files.

Documentation fields

- Markdown `description`, `summaries`, `examples`
- Variations of the above for i18n 
- Anything that makes it more "Stripe" like.



Let's call him, Nathan

DevOps engineer

Responsible for

Deployments, gateways and infrastructure

Cares about

API Security, URLs and server names

Pains (related)

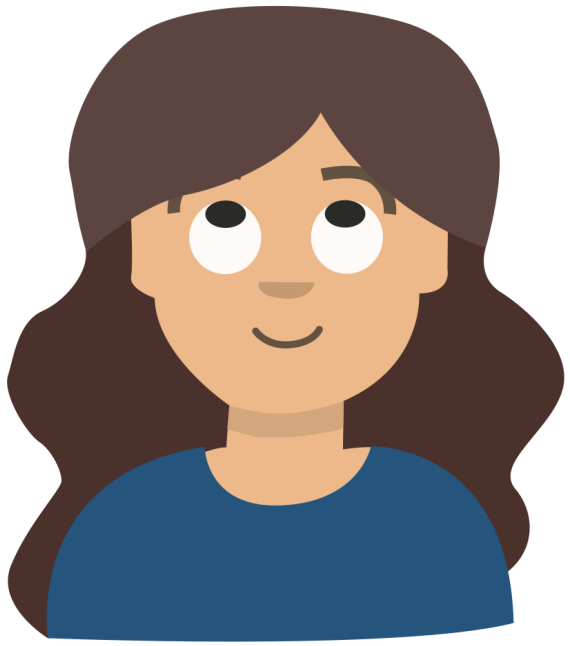
Custom scripting to inject annotations

DevOps engineer - Scripting

Annotating APIs to include infrastructure details, possibly via bespoke scripting.

```
x-amazon-apigateway-cors: ...
x-ms-parameter-grouping: ...
x-google-audiences: ...
x-kusk: ...
```

```
security:
- ...Gateway specific
servers:
- ...Different envs
```



Let's call them, Jim

Product Manager

Responsible for
Customers, "The Market"

Cares about
Visibility curation

Pains (related)
Juggling commitments

Public and Private endpoints

```yaml
openapi: 3.1.0
paths:
  /foo:
    x-internal: true
  /bar: {}
```

Alternative

```yaml
openapi: 3.1.0
paths:
  /foo:
    x-audiences: [public]
  /bar:
    x-audiences: [partner-bob]
```

Conclusion?

Lots of concerns

Drum roll... 

Overlays. 

The answer to everything

Overlays - An Example

```yaml
overlays: 1.0.0
info:
  title: Add an emoji
  version: 1.0.0
extends: https://petstore3.swagger.io/api/v3/openapi.json
actions:
- target: '$.paths."/pet".post'
  update:
    summary: Add a new pet to the store 🐕!
```

How do Overlays work?

- Target some parts of the document and mutate them.
- Layer in these changes together to form an Overlay document

The parts of an Overlay document

1. Some boilerplate
2. Extend some (URL of an) API.
3. List of actions
   a). Each action: Target then mutate things.

Boilerplate

```
overlays: 1.0.0
```

Next → Info

Boilerplate - Info

```
overlays: 1.0.0
info:
  title: Add an emoji
  version: 1.0.0
```

Next → Extends

Extend some API

```yaml
overlays: 1.0.0
info:
  title: Add an emoji
  version: 1.0.0
extends: https://petstore3.swagger.io/api/v3/openapi.json
```

Next → Actions

List of Actions

```yaml
overlays: 1.0.0
info:
  title: Add an emoji
  version: 1.0.0
extends: https://petstore3.swagger.io/api/v3/openapi.json
actions:
- # Your action...
```

Next → An Action

An action

A **Target** and a **Mutation**

```
# Target
target: ...

# Mutations
update: ...
remove: ...
```

JP: Let's look at targetting things

Targeting with JSONPath

- Gaining traction as the defacto standard for querying JSON/YAML.
- Mostly because it's being standardized and because it aims to do one thing, target nodes.

JSONPath Examples

Examples: 🎯

- `$.paths."/pet".post'` — One specific thing
- `$.paths.*.*.` — Wildcards
- `$.tags[?(@.name == "pet")]` — Filters/Expressions
- `$..description` — All decendants

Mutating things

- `update` merges in a value
- `remove` it uh... removes it.

```
update:
  summary: Add a new pet to the store 🐕!
  description: Something descriptive
```

```
remove: true
```

Putting them together into an action

```yaml
target: '$.paths."/pet".post'
update:
  summary: Add a new pet to the store 🐐!
```

All together now!

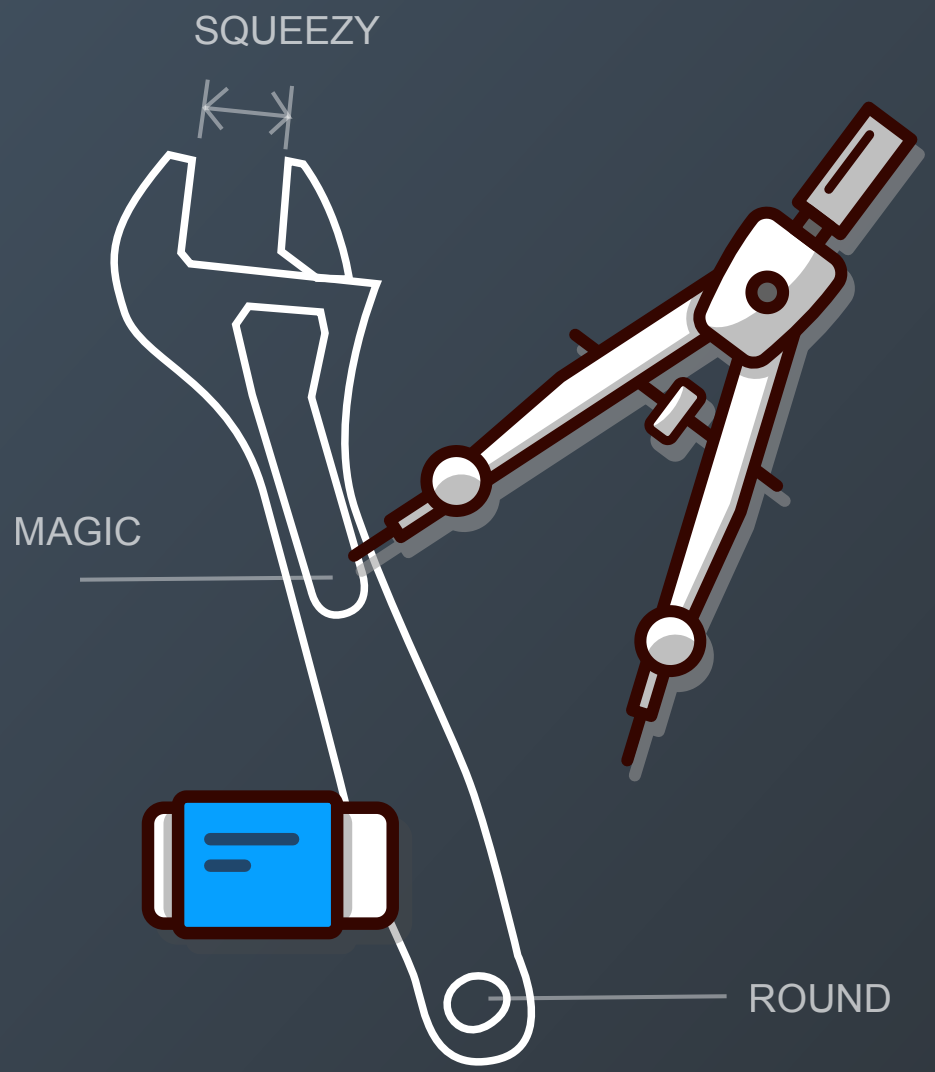
```yaml
overlays: 1.0.0
info:
  title: Add an emoji
  version: 1.0.0
extends: https://petstore3.swagger.io/api/v3/openapi.json
actions:
- target: '$.paths."/pet".post'
  update:
    summary: Add a new pet to the store 🐕!
```

We can now Overlay

- Patch files broadly or specifically
- Extract concerns
- Handle (some) changes to the underlying APIs

Shower thoughts

...and design considerations



A tool or a standard?

Why make yet-another-standard (the 15th one!)?

- It starts with a tool
- A standard is meant to have many tools
- Overlays are meant to be in many places

Pets vs Toys

Pet: Lovingly represent a single service, every detail?

Toy: Cater to consumers, showing only what is needed?

API shape proliferation

One API (service) could have many API (definitions), depending on consumers.

> **Assumption:** The more APIs, the more that composition and Overlays will be needed.

Anti-patterns and pitfalls

Dodgy practices that can cause you to trip.

Invalid definitions

We can move a lot of stuff out of APIs and into Overlays.
Possibly leaving our APIs invalid.

An invalid OpenAPI definition

```yaml
# Requires the overlay to be valid, missing 'info'
openapi: 3.0.3
paths: {}
```

```yaml
overlays: 1.0.0
actions:
- target: '$'
  update:
    info:
      title: Why am I here?
      version: 1.0.0
```

Bad: Limits the amount of tooling you can use.

Incomplete APIs

Using Overlays to describe necessary parts of the API.
This could leave you with an incomplete definition, that now *requires* an Overlay.

Instead of enriching, it becomes structural.

Consider: **Traits for OpenAPI**

Semantic whack-a-mole

JSONPath is awesome,
but it doesn't consider the semantics of the underlying specifications.

It is possible to miss targets using JSONPath.

Whack-a-parameter in OpenAPI

```yaml
paths:
 /foo:
   parameters:
   - name: bob

   get:
     parameters:
         - name: frank
   post:
     parameters: []
```

1. `$.paths.*.*.parameters` — usual
2. `$.paths.*.parameters` — not usually considered

Do you need Overlays?

Start with, "No, I don't need Overlays."
— It is another moving part

Then ponder the following.

- Do you need variations of an API?
- Is the source inaccessable? Code annotations, traffic inference.
- Are there independent features of the API?

Alternatives

- Redocly-CLI https://github.com/Redocly/redocly-cli
- JSONette https://jsonnet.org/
- Geneva https://github.com/smizell/geneva

example1.jsonnet

```
// Edit me!
{
  person1: {
    name: "Alice",
    welcome: "Hello " + self.name + "!",
  },
  person2: self.person1 { name: "Bob" },
}
```

output.json

```
{
  "person1": {
    "name": "Alice",
    "welcome": "Hello Alice!"
  },
  "person2": {
    "name": "Bob",
    "welcome": "Hello Bob!"
  }
}
```

The folks behind this

- The OpenAPI SIG, bi-weekly meet.
- We're made up of tooling vendors
- We need your help
- Prototype: https://github.com/ponelat/overlays-cli (npm: `overlays-cli`)
- Specification: https://github.com/OAI/Overlay-Specification
- Discussion: https://github.com/OAI/Overlay-Specification/discussions

Closing remarks

Thanks to the folks who helped hone this talk.

- Hezzie @Hezzieponelat
- Fabrizio Ferri-Benedetti @remoquete
- Adam Altman @adamaltman
- Frank Kilcommins @fkilcommins
- Borrowed some img from illustrations.co