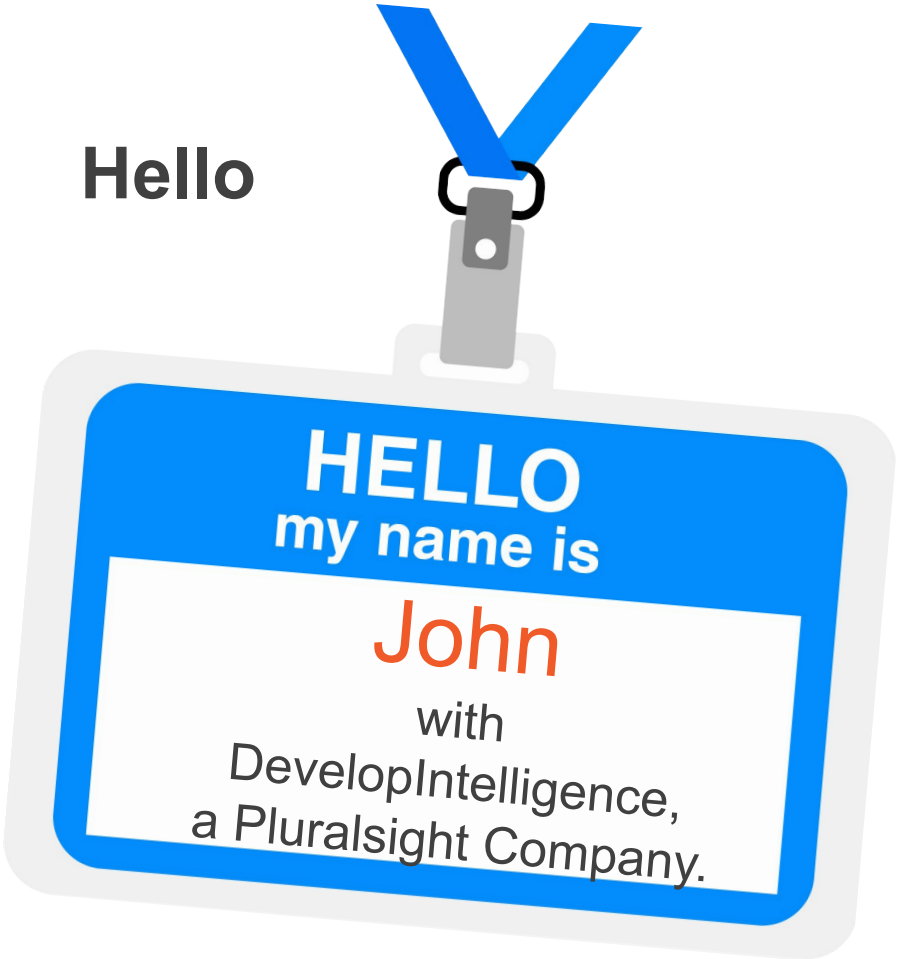
Welcome

API Fundamentals 101

DevelopIntelligence

A PLURALSIGHT COMPANY



- I have 28+ years of Industry experience
- Last 10+ years I have been involved with Designing Restful APIs,
- **Github profile**:- https://www.github.com/jwkidd3

Objectives

- Understand Design First vs Code First Approach to write/implement APIs
- Know the DIfferent types of APIs
- Understand the different API Protocols
- Learn the advantages of API Documentation
- Learn Restful API Basics
- Understand the Open API Syntax at a high level
- Design an API using Stoplight
- Understand API Virtualization & how it enables parallel development

Agenda

- API contracts
  - Drawback of not having standards
  - Summary of common standards
- Documenting APIs
  - Drawback of not having standards
  - Summary of common standard and tools
- RESTful APIs: the basics
  - Drawback of not having standards
  - Common standards
- Motivation
- APIs: Purpose and Nomenclature
  - Benefits and drawbacks of APIs
  - Key terms in API development
- API Types
  - Open
  - Partner
  - Internal
- API Protocols
  - REST
  - SOAP
  - RPC

Agenda

- Forge API Error Handling Mechanism in AutoDesk
  - Error Responses & Response Status Code
- API Tools in AutoDesk Landscape
  - Stoplight - API Design
  - Apigee - API Management
  - Mural & Confluence Wiki for Design
  - Overview of How various tools hang together in the landscape
- Overview of Improving API experience through Dogfooding
- Forge API Standardardization & Governance in AutoDesk
  - Core Principles
  - API Standards
  - API Versioning
  - HATEOAS Rest Principle
  - Localization
  - Filtering
  - Caching
  - Pagination
  - Resource Naming
  - Naming conventions

Agenda

- API Lifecycle
  - API versioning
  - Review of most common approaches to lifecycle – pros and cons
  - Drawback of not having standards
  - Summary of common standards
- Overview of DevOps as used in API development
- Security for APIs
  - Drawback of not having standards
  - Common standards
- API Ecosystem
  - API First
  - DevOps

Motivation - Jeff Bezos API Mandate

Have you heard about the Jeff Bezos API Mandate ??

Motivation

1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology is used. HTTP, Corba, Pubsub, custom protocols — doesn't matter.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired.

   https://nordicapis.com/the-bezos-api-mandate-amazons-manifesto-for-externalization/

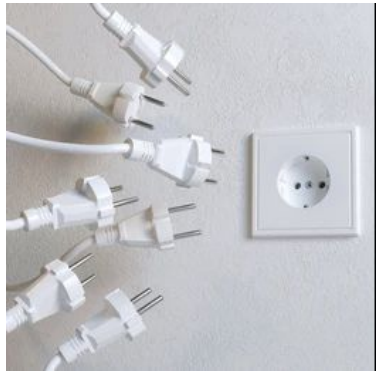
   — JEFF BEZOS API Mandate 2002

Problem Statement

There are **Different Systems** that power your business & there needs to have a consistent way to connect these systems together for seamless operation

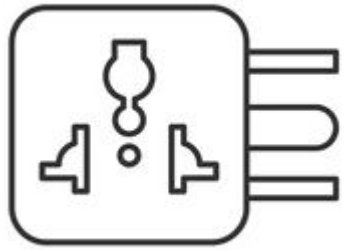
Solution - APIs

Problem



Different Systems that power your business & there needs to be a consistent way of to connect these systems together for seamless operation

Solution



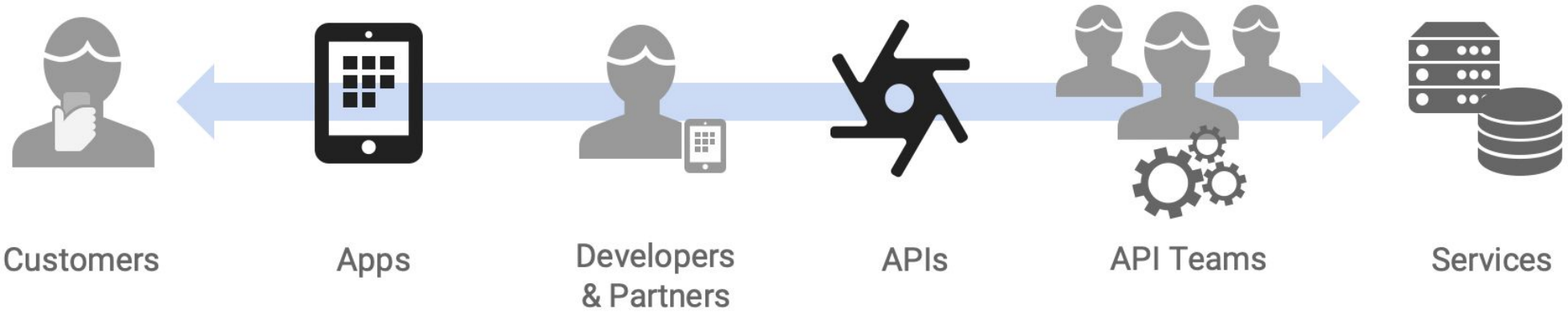
Universal Power Socket

Similar to the universal power socket, APIs provide a single interface that allows developers to connect to the data and services that power your business.

What are APIs

APIs are the de facto standard by which companies exchange data and build consistent cross-channel customer experiences.

A Digital Product - to build connected digital experiences



API & Its Purpose

API is an acronym that stands for “application programming interface,” and it allows apps to send information between each other. While there are numerous protocols and technologies involved, the underlying purpose of APIs is always the same: to let one piece of software communicate with another. - **Okta**

An application programming interface (API) is a connection between computers or between computer programs. It is a type of software interface, offering a service to other pieces of software.A document or standard that describes how to build or use such a connection or interface is called an API specification. A computer system that meets this standard is said to implement or expose an API. The term API may refer either to the specification or to the implementation. **- Wikipedia**

On the Web, APIs make it possible for big services like Google Maps or Facebook to let other apps “piggyback” on their offerings. Think about the way Yelp, for instance, displays nearby restaurants on a Google Map in its app, or the way some video games now let players chat, post high scores and invite friends to play via Facebook, right there in the middle of a game.

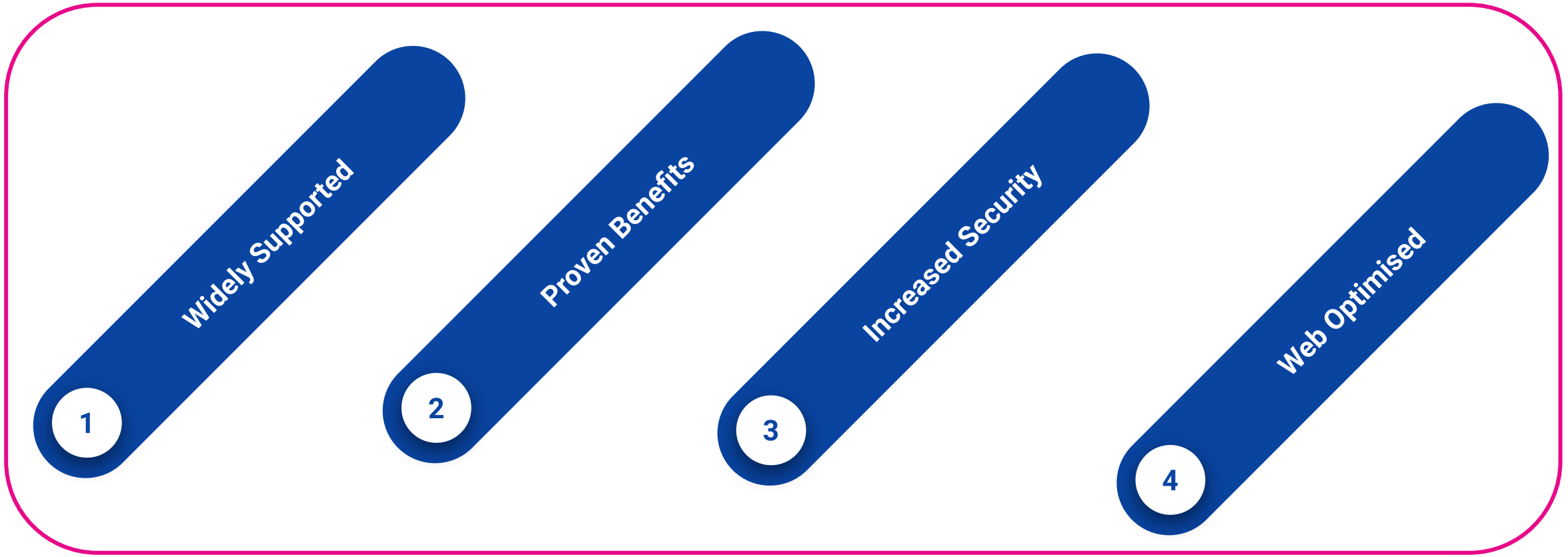
Real World Examples of APIs

Can you give me examples of APIs that you have heard or are aware of ??

Real World Examples of APIs

- https://www.skyscanner.com.au/
- https://www.airbnb.com.au/partner
- https://forge.autodesk.com/en/docs/

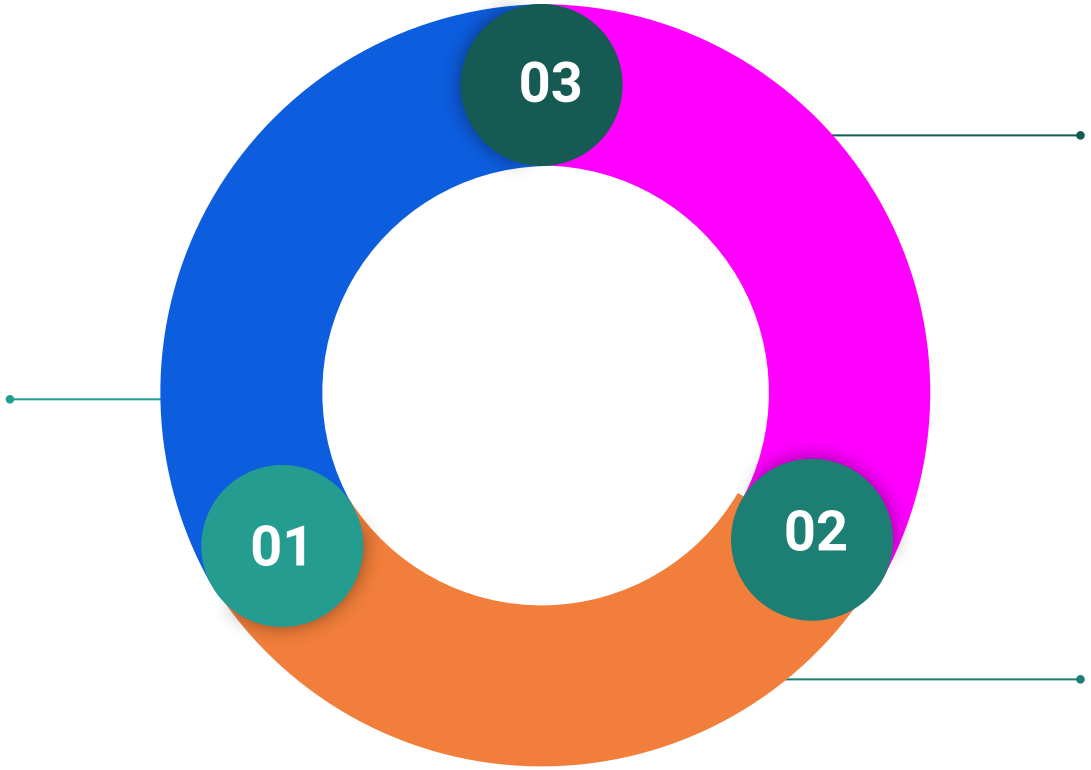
Why APIs?


1
Widely Supported
2
Proven Benefits
3
Increased Security
4
Web Optimised

API Categorizations

Services

APIs are reusable services on which an organisation can develop software.



Interactions

Connect devices - mobile phones, machine to machine calls, communication with internal & external systems

Products

APIs can be bundled into products which can be exposed for external consumption & also monetized

API Benefits

Data

The ability to measure and share information across an organisation and external ecosystems will help generate value.

+

Interactions

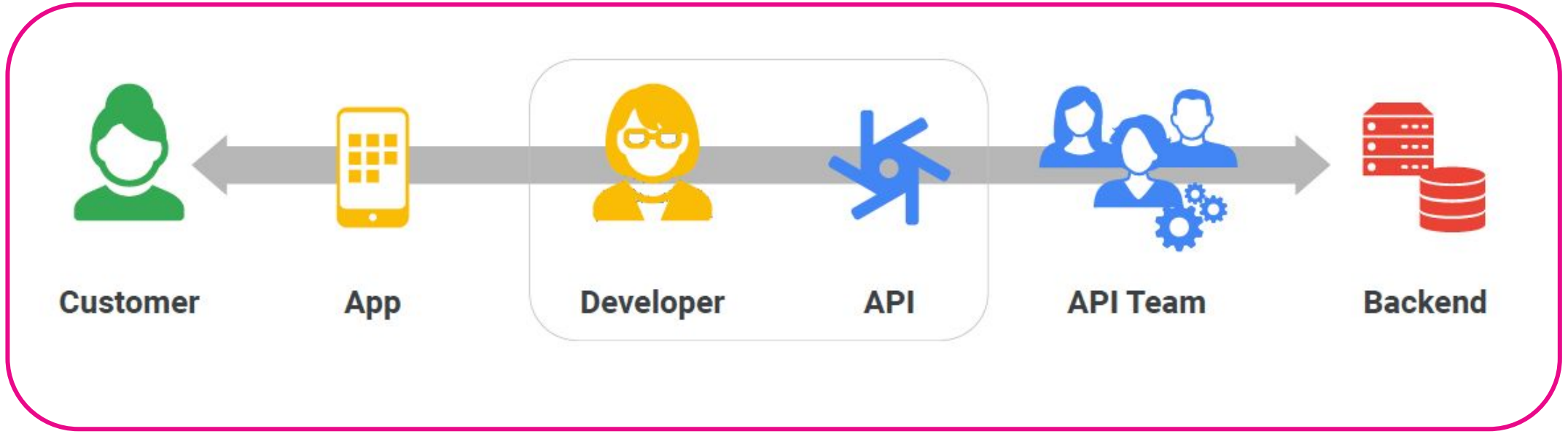
Securely expose and provide access to services that **enable outcomes**, and **empower** your employees, partners, and customers.

→

Insights

Data and interactions equal insights. Design and leverage analytics that helps Product Owner make **informed decisions and define their strategy.**

API Digital Value Chain



Source/Reference*:- Apigee

Benefits of APIs

Helps expose internal assets externally in a secure & consistent manner

Helps to connect app - provides interoperability

API helps create new experiences. It lets applications innovate

APIs help in monetisation

Businesses have now started to drive API as Products mindset

Drawbacks of APIs

APIs act as single point of entry to internal assets such as data on CRM or backends DBs, hence it becomes paramount to secure APIs. APIs become vulnerable to attack

Known Actors start to execute performance testing without informing. Hence it becomes important to have API Monitoring in place

APIs do not store state. State is stored externally in some kind of a cache or DB

API Nomenclature

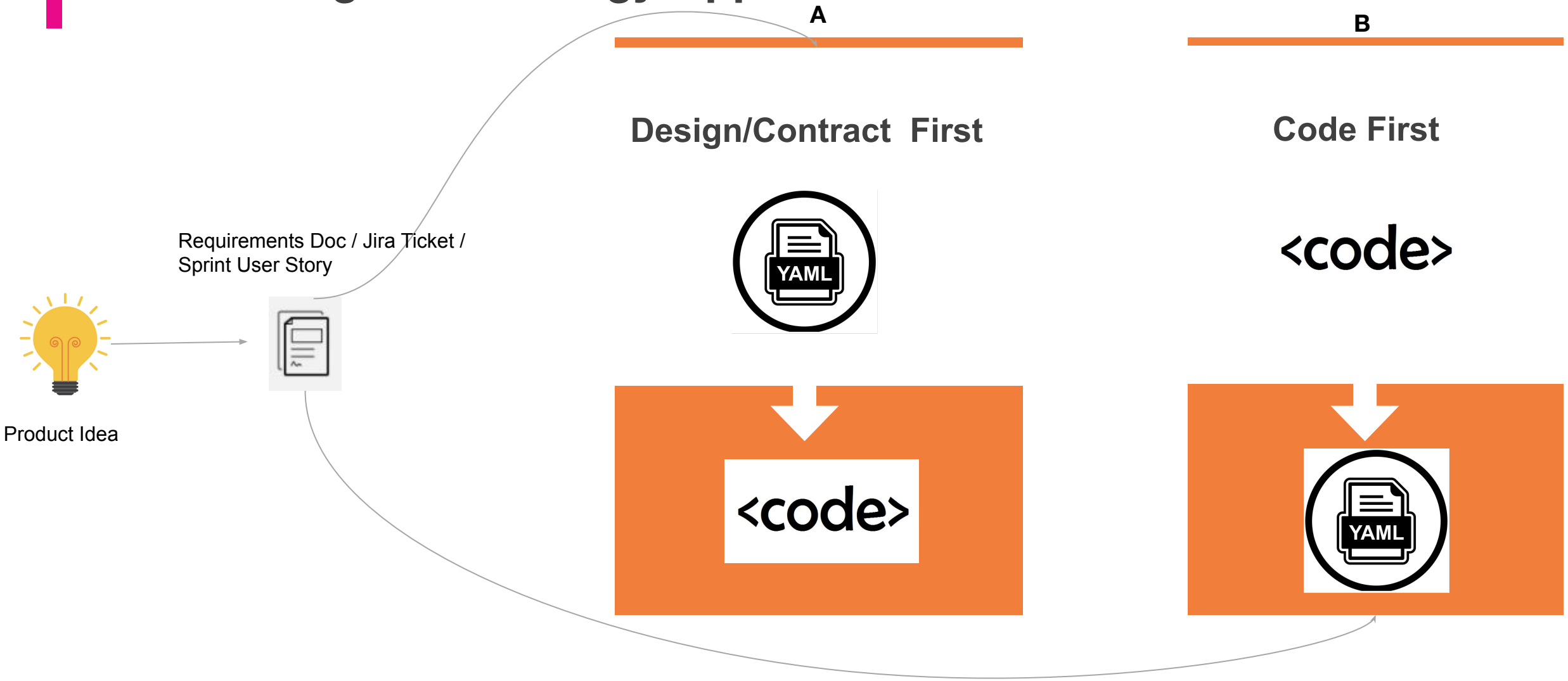
https://api.autodesk.com/v1/employees?employeeId=548257

API Building Methodology/Approaches

Code-First

Design/Contract-First

API Building Methodology/Approaches



API Building Methodology/Approaches

Code-First

A code-first approach typically involves coding an API from business requirements and then generating a machine-readable API definition from that code.

It can also mean implementing the API in code and then -

1. Use comments/annotations to generate documentation/specification
2. Manually writing specification from Scratch

Image Source*:- Stoplight.io

Note:- A code-first approach doesn't mean that you won't have an API design. Instead, you have a design process that is distributed among various code documents.

API Building Methodology/Approaches

Design/Contract-First

API design-first means you describe every API design in an iterative way that both humans and computers can understand—before you write any code. This ensures every team speaks the same language, and every tool they use leverages the same API design.

This approach begins with every stakeholder participating in the API design to write an API contract that satisfies all parties. It also ensures humans and machines understand the finished product. With this approach, teams spend a lot of time in the design phase to ensure that when it comes time to start coding, they're writing code that won't need to be scrapped and rewritten.

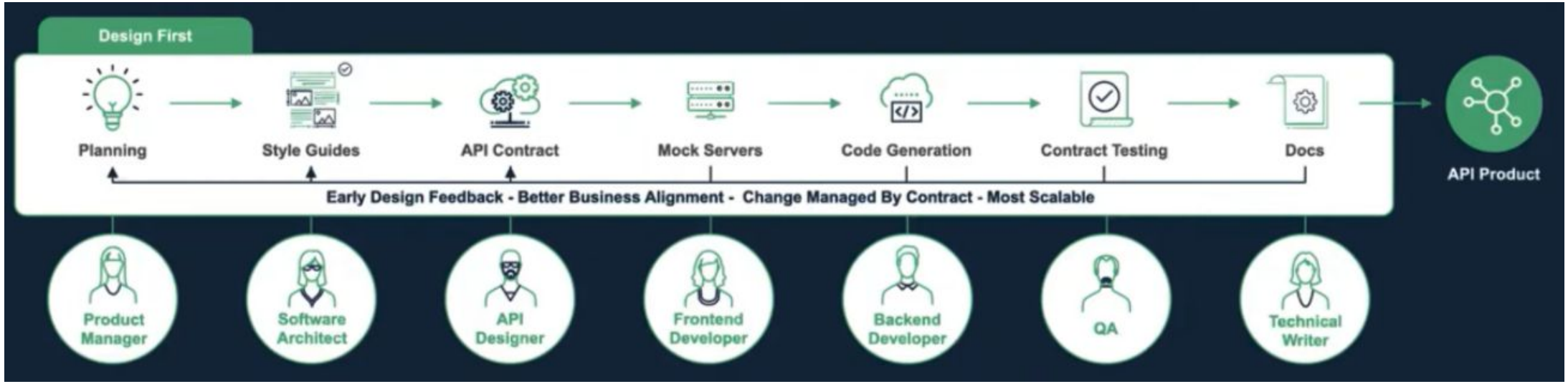


Image Source*:- Stoplight.io

API Design Methodology(Contd.)

Advantages of Design/Contract-First Approach

- **Shake out issues early** - Uncover business & technical gaps

- **Ability to Work in Parallel** - You can use API description documents to create and mock APIs, which allow you to try out API designs and test APIs before deploying them. You don't have to wait on other teams to complete different phases of the API.

- **Ensuring a Positive DX** - You can use API description documents and third-party tools to automatically generate things that improve the developer experience, like interactive documentation, client libraries, and SDKs.

- **Reduce Development Costs** - Fixing issues once the API is coded costs far more than fixing them during the design phase. You can also reduce development costs by reusing components across multiple API projects

API Design Methodology(Contd.)

Benefits of Code First Approach

- **Deliver Simple APIs Fast** - If you need to deploy an API fast, going right to coding after receiving the business requirements will speed up API deployment. If the API has only a few endpoints or is only for internal use, it may not make sense to spend a lot of time creating a design contract. You can also find tools that speed up code-first processes like testing and deployment.

- **Familiarity for Devs** - Code-first follows the traditional method of software development that most developers are familiar with. This approach typically has no learning curve because developers don’t have to learn a new language or use unfamiliar design tools to create API contracts.

API Design Methodology(Contd.)

Disadvantages of Code First Approach

- **Bottlenecks** - A code-first approach typically means developers follow a to API development, leading to bottlenecks. Teams working on the API will finish each phase in sequential order, so they cannot work in parallel.
- **Useless or Bloated APIs** - When you code first and then get feedback after several versions of the coded API, you often end up with bloated APIs that have resources users don't need. You also run the risk of building APIs that don't help your customers at all.
- **No Documentation** - When teams finally finish coding an API, creating documentation that the team must maintain can feel like a giant chore. So, documentation often becomes an afterthought instead of a priority and never gets done.
- **Improper Utilisation of Resources** - It costs far more to make changes to an already-coded API than to an API design. You tend to have to comb through and write a lot of code to fix the API once you get feedback from users. And if you coded first and didn't create documentation, you could end up creating because no one on the team remembers how the code works.

https://blog.stoplight.io/api-first-api-design-first-or-code-first-which-should-you-choose

API First Adoption Principle

1. Who is going to consume your APIs (Keeping the Consumer in Mind) & What they want to achieve. What is the business problem the API is trying to solve

1. Design the API Specification First and worry about the implementation & technology stack later

API Types

- **Public APIs** -This type of APIs is open for public consumption with minimal or no authentication baked into it. Also known as Public APIs

  https://mixedanalytics.com/blog/list-actually-free-open-no-auth-needed-apis/

  https://github.com/public-apis/public-apis

- **Partner API** - This type of APIs are offered to consumers after the API provider & the API consumer enter into a formal agreement that defines an NDA & some sort of License usage. Usually OAuth 2.0 grant type is baked into the APIs to protect the APIs & identify the partners uniquely.

  https://developer.uber.com/

- **Internal API** - Also called as Private APIs. These are usually meant for consumption by internal systems or products or across various Line of Business but in the same company.

Key Terms in API Development

- HTTP

- REST

- APIs

- OpenAPI

- Swagger

API Protocols

- REST
- SOAP
- RPC
- GraphQL

API Contract / Open API Specification(OAS)

Open API Spec (OAS) is both

- Machine Readable
- Human Readable

What is API Documentation

API Documentation is a technical process of defining the contract of the APIs.

The contract contains the request, response , error description & authentication mechanism of the API along with SDKs, examples & tutorials of how to integrate/consume the APIs

It tells the developer/consumer that

- What this API can do
- Purpose of the API
- Various methods in this API - Mostly CRUD operations

Documenting APIs

For Whom & Why will you document APIs ??

Why is Documenting APIs Important

- Organic & Inorganic Developer Adoption
- Saves Time & Effort : Technical Writer/Architects and API Developer
- The Developer understands the usage of API just by looking at the documentation
- Visibility/Clarity across the entire API Development Lifecycle and various teams such as backend/frontend/QA/Product teams
- API Evangelism

Examples of Good API Documentation

https://nordicapis.com/5-examples-of-excellent-api-documentation/

https://stoplight.io/api-documentation-guide/basics/

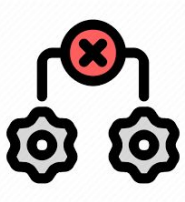
Popular Tools For Documenting API

SwaggerHub - SmartBear

Stoplight

Apigee

Disadvantages of not Documenting API



- Teams develop in Isolation
- The nature of API Development is not collaborative
- API Development Lifecycle is elongated
- API Development mechanism is not consistent & standards are not adhered
- Lack of Governance in API Development

API Description Format

What are some of the API Description Formats you are aware of ??

API Description Format

YAML - Yet Another Markup Language

JSON - JavaScript Object Notation

RAML - Restful API Modeling Language

WADL - Web Application Description Language (Used for Rest APIs)

WSDL - Web Service Definition Language (For SOAP Services - XML based services over HTTP)

Restful API Basics

What is Rest ??

What are Restful APIs ??

Restful API Basics

- Rest is not a standard rather it is an architectural style/pattern
- Payload is generally JSON(JavaScript Object Notation) , some legacy implementations also offer XML (for legacy/backward compatibility)

What are Restful APIs ??

- APIs that adhere to the Restful architecture style are called Restful

Restful API Basics

Restful APIs leverage common web HTTP concepts like Urls, Http verbs, and Http status codes

Method	Idempotent
GET	YES
POST	NO
PUT	YES
DELETE	YES
PATCH	NO
HEAD	YES
OPTIONS	YES

Method	Meaning
GET	Read data
POST	Insert data
PUT or PATCH	Update data, or insert if a new id
DELETE	Delete data

Restful API Basics

Restful APIs leverage common web HTTP concepts like Urls, Http verbs, and Http status codes

STATUS CODE	EXPLANATION
200 - OK	The request succeeded.
204 - No Content	The document contains no data.
301 - Moved Permanently	The resource has permanently moved to a different URI.
401 - Not Authorized	The request needs user authentication.
403 - Forbidden	The server has refused to fulfill the request.
404 - Not Found	The requested resource does not exist on the server.
408 - Request Timeout	The client failed to send a request in the time allowed by the server.
500 - Server Error	Due to a malfunctioning script, server configuration error or similar.

Status codes indicate the result of the HTTP request.

1XX – informational

2XX – success

3XX – redirection

4XX – client error

5XX – server error

Restful API Basics - Using Http Headers

- Content-Type

  At server side, an incoming request may have an entity attached to it. To determine it's type, server uses the HTTP request header **Content-Type.** Some common examples of content types are "text/plain", "application/xml", "text/html", "application/json", "image/gif", and "image/jpeg".

```
Content-Type: application/json
```

- Accept

  Similarly, to determine what type of representation is desired on the client-side, an HTTP header **ACCEPT** is used

```
Accept: application/json
```

Restful API Basics

- **Driven By Nouns & Resource Oriented**

  GET api.autodesk.com/v1/employees - Retrieves a list of employees
  GET api.autodesk.com/v1/employees/{employeeId} - Retrieves a particular employee by employeeId

- **Driven By Http Verbs**

  GET - search or retrieve
  POST - create
  PUT/PATCH - update
  DELETE - remove

Restful API Basics

- **Example 1**

  GET api.autodesk.com/v1/employees - Retrieves a list of employees
  api.autodesk.com/v1/employees/{employeeId} - Retrieves a particular employee by employeeId

- **Driven By Http Verbs**

  GET - search or retrieve
  POST - create
  PUT/PATCH - update
  DELETE - remove

Restful API Basics

Example 2

- Resource
- GET /v1/partners/trips

Restful API Basics

Example 2 - Uber Trips

- Resource
- GET /v1/partners/trips

Authorization

OAuth 2.0 bearer token with the `partner.trips` scope.

Query Parameters

Name	Type	Description
`offset` (*optional*)	`integer`	Offset the list of returned results by this amount. Default is zero.
`limit` (*optional*)	`integer`	Number of items to retrieve per page. Default is 10, maximum is 50.
`from_time` (*optional*)	`integer`	Unix timestamp of the start time to query. Queries starting from the first trip if omitted.
`to_time` (*optional*)	`integer`	Unix timestamp of the end time to query. Queries up to the last trip if omitted.

https://developer.uber.com/docs/drivers/references/api/v1/partners-trips-get

Restful API Basics

- **Example 2 - Uber Trips**
  - Resource
  - GET /v1/partners/trips

Response

Status-Code: 200 OK


```
{
  "count": 1200,
  "limit": 1,
  "trips": [
    {
      "fare": 6.2,
      "dropoff": {
        "timestamp": 1502844378
      },
      "vehicle_id": "0082b54a-6a5e-4f6b-b999-b0649f286381",
      "distance": 0.37,
      "start_city": {
        "latitude": 38.3498,
        "display_name": "Charleston, WV",
        "longitude": -81.6326
      },
      "status_changes": [
        {
          "status": "accepted",
          "timestamp": 1502843899
        },
        {
          "status": "driver_arrived",
          "timestamp": 1502843900
        },
        {
          "status": "trip_began",
          "timestamp": 1502843903
        },
        {
          "status": "completed",
          "timestamp": 1502844378
        }
      ],
      "surge_multiplier": 1,
      "pickup": {
        "timestamp": 1502843903
      },
      "driver_id": "8LvWuRAq2511gmr8EMkovekFNa2848lyMaQevIto-aXmnK9oKNRtfTxYLgPq9OSt8EzAu5pDB7XiaQIrcp-zXgOA5EyK4h00U6D1o7aZpXIQah--U77Eh7LEBiksj2rahB==",
      "status": "completed",
      "duration": 475,
      "trip_id": "b5613b6a-fe74-4704-a637-50f8d51a8bb1",
      "currency_code": "USD"
    }
  ],
  "offset": 0
}
```


https://developer.uber.com/docs/drivers/references/api/v1/partners-trips-get

Restful API Basics

- **Example 3**
  - Resource
  - POST /v1/customers

Restful API Basics

- **Example 3**
  - Resource
  - POST /v1/customers
  - Request:-

POST /v1/customers Select library

```
$ curl https://api.stripe.com/v1/customers \
  -u sk_test_4eC39HqLyjWDarjtT1zdp7dc: \
  -d description="My First Test Customer (created for API docs)"
```

Restful API Basics

Example 3 - Create customer in Stripe

- Resource
- POST /v1/customers
- Request:-

POST /v1/customers — Select library

```
$ curl https://api.stripe.com/v1/customers \
  -u sk_test_4eC39HqLyjWDarjtT1zdp7dc: \
  -d description="My First Test Customer (created for API docs)"
```

https://stripe.com/docs/api/customers/create

Restful API Basics

Example 3

- Resource
- POST /v1/customers

https://stripe.com/docs/api/customers/create

RESPONSE


```
{
  "id": "cus_4QE0v7gcdox28x",
  "object": "customer",
  "address": null,
  "balance": 0,
  "created": 1405636867,
  "currency": "usd",
  "default_source": "card_14HNYk2eZvKYlo2C0zXVbXp3",
  "delinquent": true,
  "description": "My First Test Customer (created for API docs)",
  "discount": null,
  "email": null,
  "invoice_prefix": "3A30AC2",
  "invoice_settings": {
    "custom_fields": null,
    "default_payment_method": null,
    "footer": null
  },
  "livemode": false,
  "metadata": {
    "order_id": "6735"
  },
  "name": null,
  "next_invoice_sequence": 75228,
  "phone": null,
  "preferred_locales": [],
  "shipping": null,
  "tax_exempt": "none"
}
```

Restful API Basics - Dont’s

Example 4 - What you should not be doing

Resource

- POST /v1/createCustomers
- GET /v1/getCustomers
- Violate Http

Restful API Basics - Quiz

https://play.kahoot.it/v2/?quizId=fdfdee7f-6a36-458f-a915-606a64825de2

Welcome

API Fundamentals - Day 2

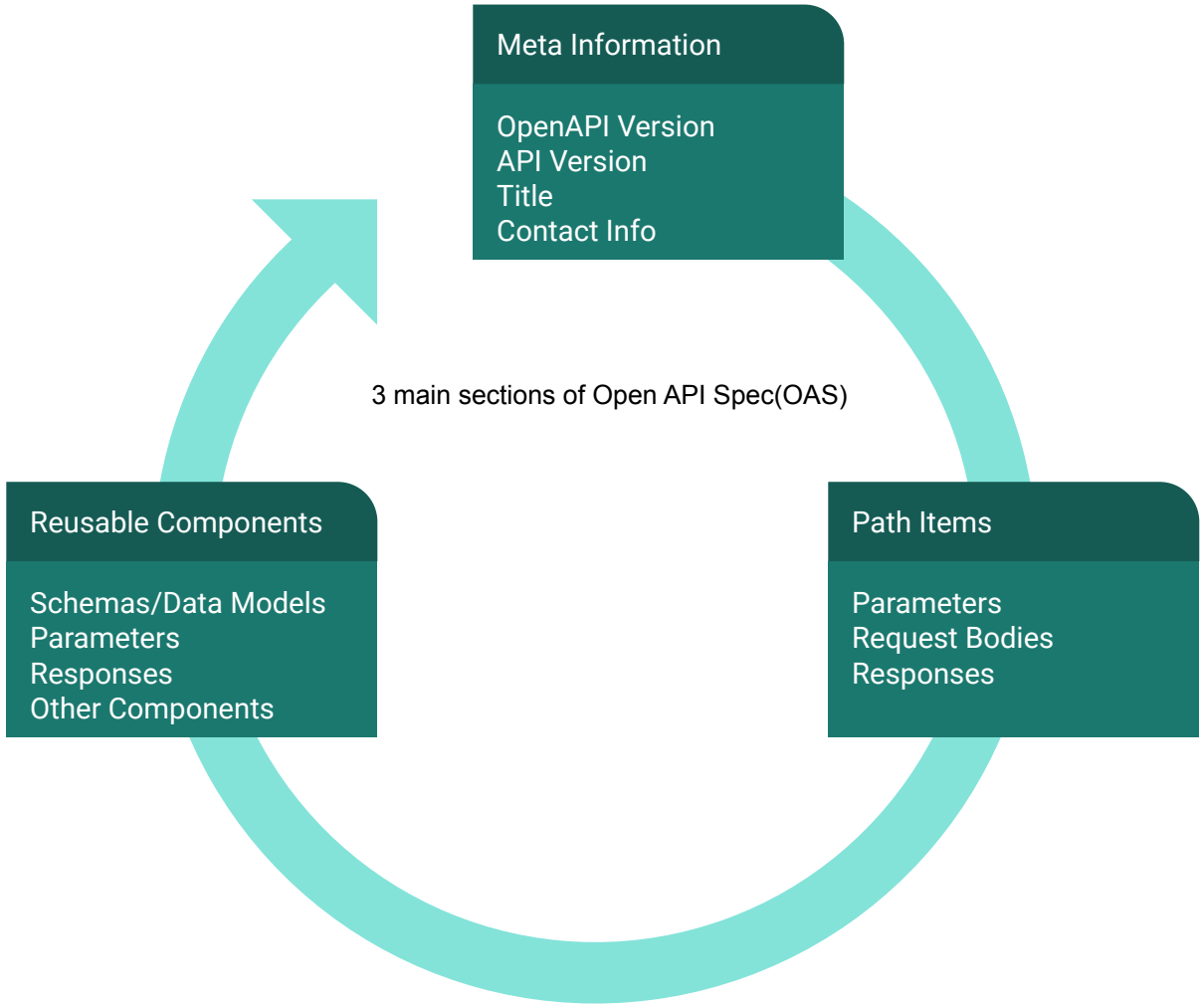
DevelopIntelligence

A PLURALSIGHT COMPANY

Restful API Basics - Richardson Maturity Model

https://restfulapi.net/richardson-maturity-model/

Restful APIs - Getting Started With OAS 3.0




```yaml
#API Metadata
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple Hello World API
  description: A simple API to illustrate OpenAPI concepts
  contact:
    name: Harry Bond
    email: harry@example.com

servers:
  - url: https://example.io/v1

security:
  - BasicAuth: []

#Path Items
paths: {}


#Re-Usable Components
components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic

```

OAS Syntax/Attributes/Objects

```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: Employees API
  description: An API that deal with Employees creation & retrieving to illustrate OpenAPI
    concepts

servers:
  - url: https://api-test.autodesk.com/v1

security:
  - BasicAuth: []

paths:
  /employees:
    get:
      description: Returns a list of employees
      parameters:
        #  ----- Added line  ----------------------------------------
        - $ref: '#/components/parameters/PageLimit'
        - $ref: '#/components/parameters/PageOffset'
        #  ---- /Added line  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of employees
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ------------------------------
                  $ref: '#/components/schemas/Employee'
                  #  ---- /Added line  ------------------------------
        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400BadRequest'
          #  ---- /Added line  ----------------------------------------

    post:
      description: Lets a user create a new employee
```

Create a new API

OAS Syntax/Attributes/Objects - Step 1

```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: Employees API
  description: An API that deal with Employees creation & retrieving to illustrate OpenAPI
    concepts

servers:
  - url: https://api-test.autodesk.com/v1

security:
  - BasicAuth: []

paths:
  /employees:
    get:
      description: Returns a list of employees
      parameters:
        #  ----- Added line  ----------------------------------------
        - $ref: '#/components/parameters/PageLimit'
        - $ref: '#/components/parameters/PageOffset'
        #  ---- /Added line  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of employees
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ------------------------------
                  $ref: '#/components/schemas/Employee'
                  #  ---- /Added line  ------------------------------
        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400BadRequest'
          #  ---- /Added line  ----------------------------------------

    post:
      description: Lets a user create a new employee
```

Open API Version

openapi: 3.0.0

OAS Syntax/Attributes/Objects - Step 2

```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: Employees API
  description: An API that deal with Employees creation & retrieving to illustrate OpenAPI
    concepts

servers:
  - url: https://api-test.autodesk.com/v1

security:
  - BasicAuth: []

paths:
  /employees:
    get:
      description: Returns a list of employees
      parameters:
        #  ----- Added line  ----------------------------------------
        - $ref: '#/components/parameters/PageLimit'
        - $ref: '#/components/parameters/PageOffset'
        #  ---- /Added line  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of employees
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ------------------------------
                  $ref: '#/components/schemas/Employee'
                  #  ---- /Added line  ------------------------------
        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400BadRequest'
          #  ---- /Added line  ----------------------------------------

    post:
      description: Lets a user create a new employee
```

Metadata - Description of the API

info:
version: 1.0.0
title: Employees API
description: An API that deal with Employees creation & retrieving to illustrate OpenAPI concepts
contact:

OAS Syntax/Attributes/Objects - Step 3

```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: Employees API
  description: An API that deal with Employees creation & retrieving to illustrate OpenAPI
    concepts

servers:
  - url: https://api-test.autodesk.com/v1

security:
  - BasicAuth: []

paths:
  /employees:
    get:
      description: Returns a list of employees
      parameters:
        #  ----- Added line  ----------------------------------------
        - $ref: '#/components/parameters/PageLimit'
        - $ref: '#/components/parameters/PageOffset'
        #  ---- /Added line  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of employees
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ----------------------------
                  $ref: '#/components/schemas/Employee'
                  #  ---- /Added line  ----------------------------
        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400BadRequest'
          #  ---- /Added line  ----------------------------------------

    post:
      description: Lets a user create a new employee
```

Server/Host Information - Location of the API

servers:
- url: 'https://api-test.autodesk.com/v1'
  description: Test Server

OAS Syntax/Attributes/Objects - Step 4

```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: Employees API
  description: An API that deal with Employees creation & retrieving to illustrate OpenAPI
    concepts

servers:
  - url: https://api-test.autodesk.com/v1

security:
  - BasicAuth: []

paths:
  /employees:
    get:
      description: Returns a list of employees
      parameters:
        #  ----- Added line  ----------------------------------------
        - $ref: '#/components/parameters/PageLimit'
        - $ref: '#/components/parameters/PageOffset'
        #  ---- /Added line  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of employees
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ------------------------------
                  $ref: '#/components/schemas/Employee'
                  #  ---- /Added line  ------------------------------
        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400BadRequest'
          #  ---- /Added line  ----------------------------------------

    post:
      description: Lets a user create a new employee
```

Define Security Mechanism

security:
- BasicAuth: []

OAS Syntax/Attributes/Objects - Step 5

```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: Employees API
  description: An API that deal with Employees creation & retrieving to illustrate OpenAPI
    concepts

servers:
  - url: https://api-test.autodesk.com/v1

security:
  - BasicAuth: []

paths:
  /employees:
    get:
      description: Returns a list of employees
      parameters:
        #  ----- Added line  ----------------------------------------
        - $ref: '#/components/parameters/PageLimit'
        - $ref: '#/components/parameters/PageOffset'
        #  ---- /Added line  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of employees
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ---------------------------------
                  $ref: '#/components/schemas/Employee'
                  #  ---- /Added line  ---------------------------------
        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400BadRequest'
          #  ---- /Added line  ----------------------------------------

    post:
      description: Lets a user create a new employee
```

Define Path Information

/employees:
  get:
    description: Returns a list of employees
    parameters:
      # ----- Added line ----------
      - $ref: '#/components/parameters/PageLimit'
      - $ref: '#/components/parameters/PageOffset'
      # ---- /Added line ----------

OAS Syntax/Attributes/Objects - Optional Step

```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: Employees API
  description: An API that deal with Employees creation & retrieving to illustrate OpenAPI
    concepts

servers:
  - url: https://api-test.autodesk.com/v1

security:
  - BasicAuth: []

paths:
  /employees:
    get:
      description: Returns a list of employees
      parameters:
        #  ----- Added line  ----------------------------------------
        - $ref: '#/components/parameters/PageLimit'
        - $ref: '#/components/parameters/PageOffset'
        #  ---- /Added line  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of employees
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ------------------------------
                  $ref: '#/components/schemas/Employee'
                  #  ---- /Added line  ------------------------------
        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400BadRequest'
          #  ---- /Added line  ----------------------------------------

    post:
      description: Lets a user create a new employee
```

Add comments

```
#  ----- Added line  ----------------------------------------
    - $ref: '#/components/parameters/PageLimit'
```

OAS Syntax/Attributes/Objects - Step 6

```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: Employees API
  description: An API that deal with Employees creation & retrieving to illustrate OpenAPI
    concepts

servers:
  - url: https://api-test.autodesk.com/v1

security:
  - BasicAuth: []

paths:
  /employees:
    get:
      description: Returns a list of employees
      parameters:
        #  ----- Added line  ----------------------------------------
        - $ref: '#/components/parameters/PageLimit'
        - $ref: '#/components/parameters/PageOffset'
        #  ---- /Added line  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of employees
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ------------------------------
                  $ref: '#/components/schemas/Employee'
                  #  ---- /Added line  ------------------------------
        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400BadRequest'
          #  ---- /Added line  ----------------------------------------

    post:
      description: Lets a user create a new employee
```

Define Parameters

parameters:
----- Added line ----------
- $ref: '#/components/parameters/PageLimit'
- $ref: '#/components/parameters/PageOffset'
---- /Added line ----------

OAS Syntax/Attributes/Objects - Step 7

```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: Employees API
  description: An API that deal with Employees creation & retrieving to illustrate OpenAPI
    concepts

servers:
  - url: https://api-test.autodesk.com/v1

security:
  - BasicAuth: []

paths:
  /employees:
    get:
      description: Returns a list of employees
      parameters:
        #  ----- Added line  ----------------------------------------
        - $ref: '#/components/parameters/PageLimit'
        - $ref: '#/components/parameters/PageOffset'
        #  ---- /Added line  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of employees
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ------------------------------
                  $ref: '#/components/schemas/Employee'
                  #  ---- /Added line  ------------------------------
        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400BadRequest'
          #  ---- /Added line  ----------------------------------------

    post:
      description: Lets a user create a new employee
```

Define Responses

```yaml
responses:
        '200':
          description: Successfully returned a list of employees
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ------------------------------
                  $ref: '#/components/schemas/Employee'
                  #  ---- /Added line  ------------------------------
```

OAS Syntax/Attributes/Objects - Step 8

```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: Employees API
  description: An API that deal with Employees creation & retrieving to illustrate OpenAPI
    concepts

servers:
  - url: https://api-test.autodesk.com/v1

security:
  - BasicAuth: []

paths:
  /employees:
    get:
      description: Returns a list of employees
      parameters:
        #  ----- Added line  ----------------------------------------
        - $ref: '#/components/parameters/PageLimit'
        - $ref: '#/components/parameters/PageOffset'
        #  ---- /Added line  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of employees
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ---------------------------------
                  $ref: '#/components/schemas/Employee'
                  #  ---- /Added line  ---------------------------------
        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400BadRequest'
          #  ---- /Added line  ----------------------------------------

    post:
      description: Lets a user create a new employee
```

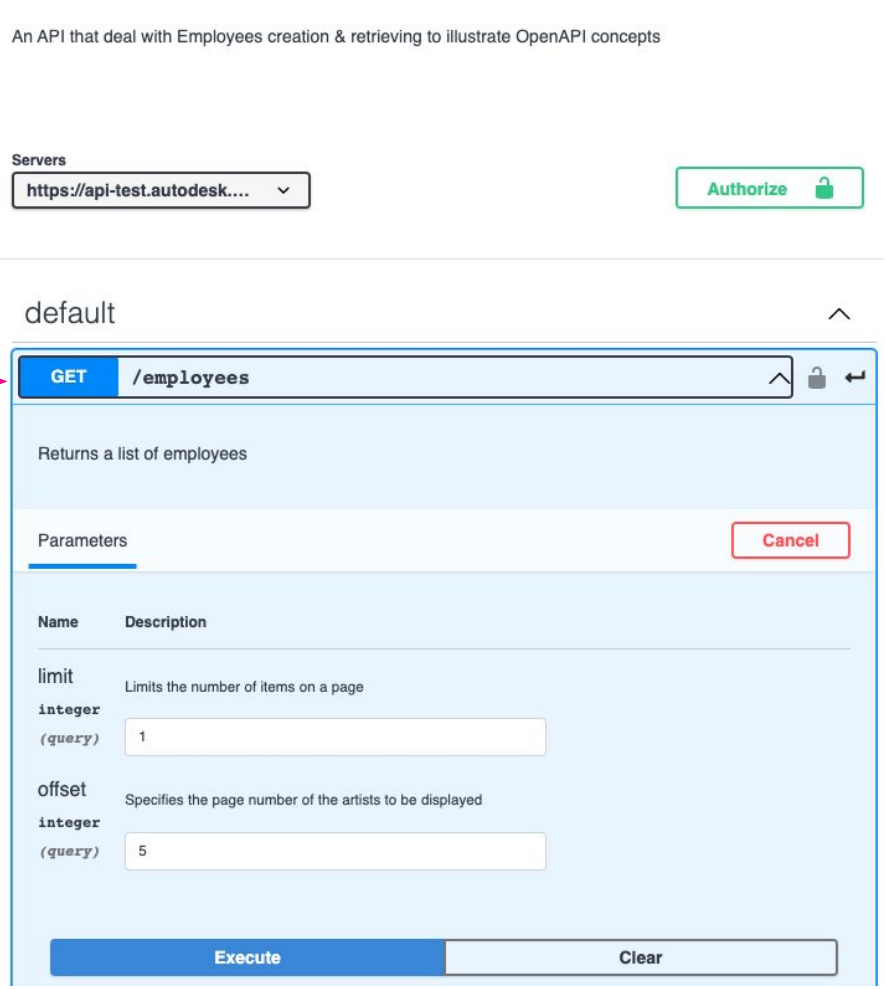
Define Error Responses

```
'400':
        # ----- Added line  ----------------------------------------
        $ref: '#/components/responses/400BadRequest'
        # ---- /Added line  ----------------------------------------
```

OAS Syntax/Attributes/Objects

```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: Employees API
  description: An API that deal with Employees creation & retrieving to illustrate OpenAPI
    concepts

servers:
  - url: https://api-test.autodesk.com/v1

security:
  - BasicAuth: []

paths:
  /employees:
    get:
      description: Returns a list of employees
      parameters:
        #  ----- Added line  ----------------------------------------
        - $ref: '#/components/parameters/PageLimit'
        - $ref: '#/components/parameters/PageOffset'
        #  ---- /Added line  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of employees
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ----------------------------------
                  $ref: '#/components/schemas/Employee'
                  #  ---- /Added line  ----------------------------------
        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400BadRequest'
          #  ---- /Added line  ----------------------------------------

    post:
      description: Lets a user create a new employee
```

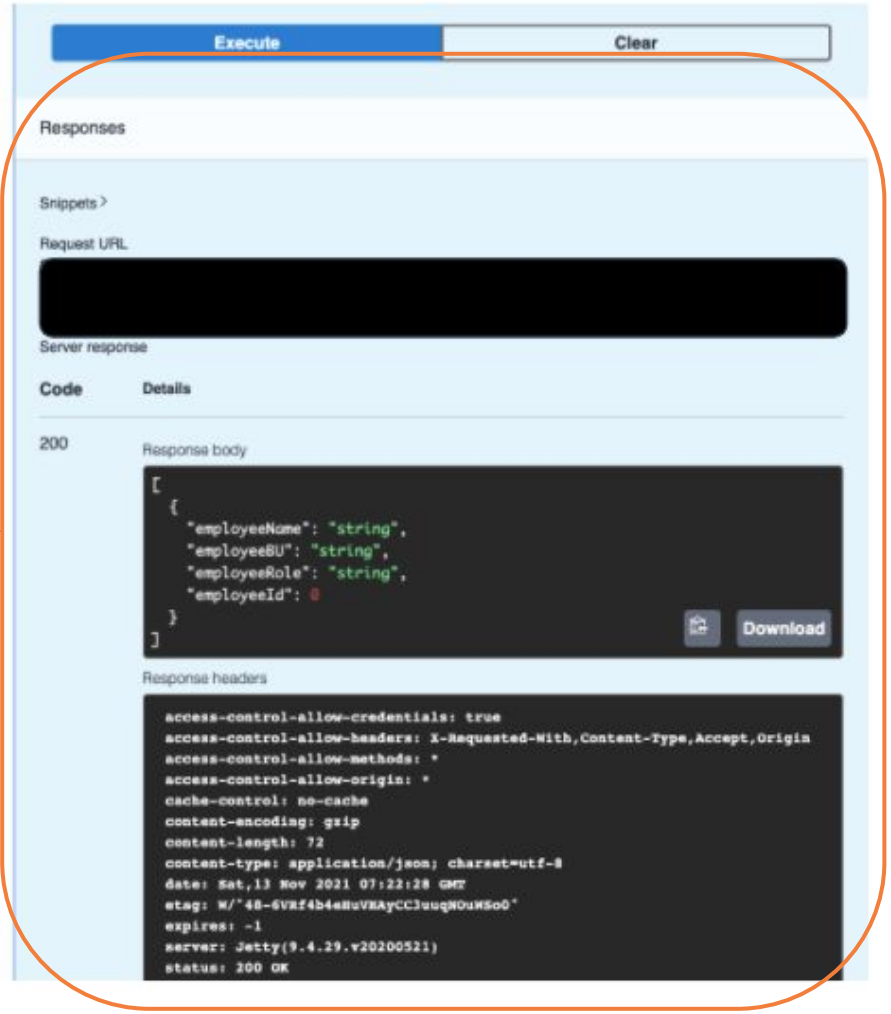
Last Saved: 10:45:13 am - Nov 14, 2021 ✔ VALID



OAS Syntax/Attributes/Objects

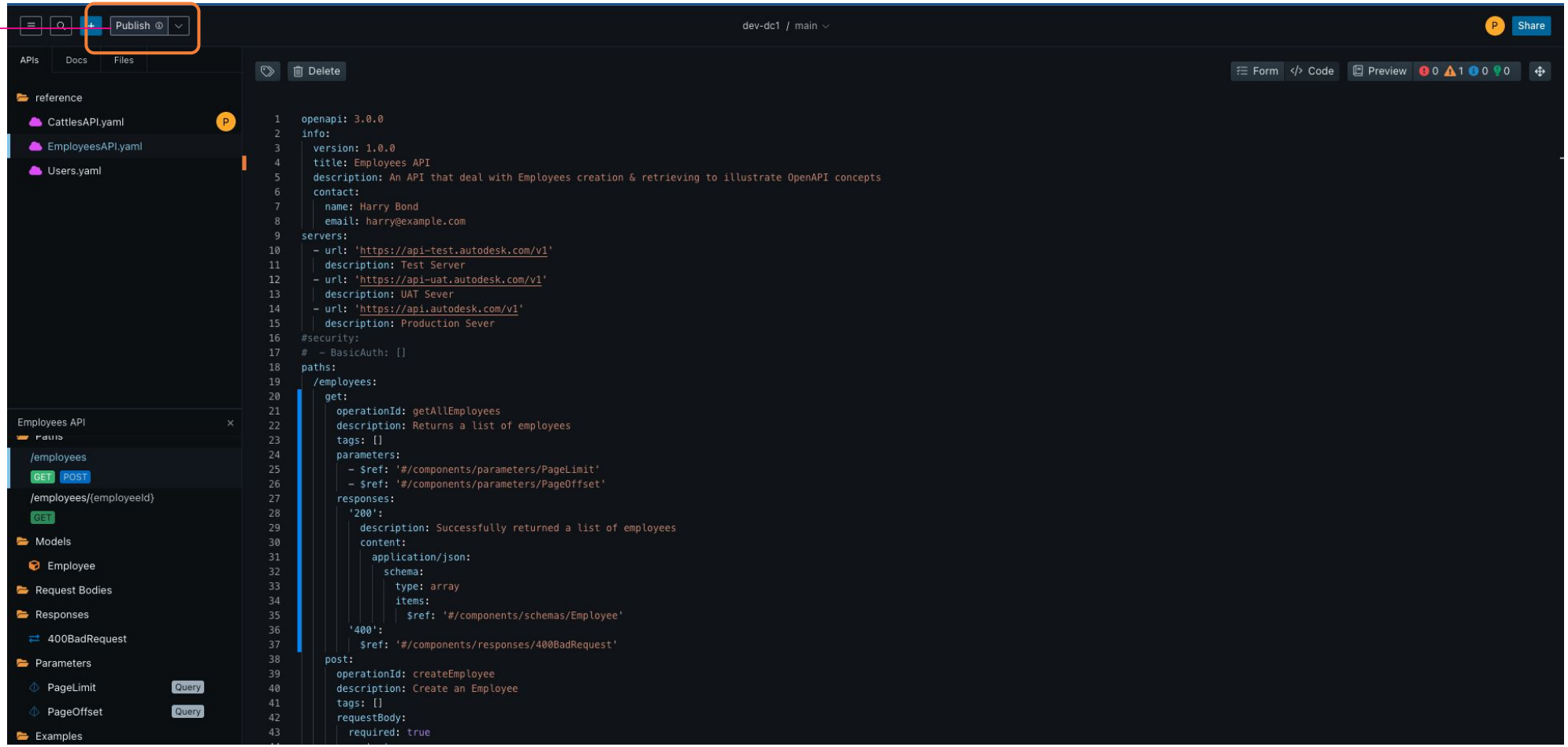
```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: Employees API
  description: An API that deal with Employees creation & retrieving to illustrate OpenAPI
    concepts

servers:
  - url: https://api-test.autodesk.com/v1

security:
  - BasicAuth: []

paths:
  /employees:
    get:
      description: Returns a list of employees
      parameters:
        #  ----- Added line  ----------------------------------------
        - $ref: '#/components/parameters/PageLimit'
        - $ref: '#/components/parameters/PageOffset'
        #  ---- /Added line  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of employees
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ----------------------------------
                  $ref: '#/components/schemas/Employee'
                  #  ---- /Added line  ----------------------------------
        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400BadRequest'
          #  ---- /Added line  ----------------------------------------

    post:
```

Last Saved: 10:45:13 am - Nov 14, 2021 ✔ VALID



OAS Syntax/Attributes/Objects

Publish



https://meta.stoplight.io/docs/studio/ZG9jOjg3-publishing-in-studio

Build an OAS Using Stoplight

- GUI Based
- Editor based

Design-First With Stoplight - Demo

Two ways of Building OAS spec using Stoplight

- GUI Based
- Editor based

Stoplight Studio Demo

- Design an API with Studio
- Components Re-Use
- API Linting with Spectral
- Mock the API
- Try it out yourself

https://meta.stoplight.io/docs/studio/ZG9jOjgx-documentation-quickstart-guide

Employees - Open API Spec - Fix - Hands-On

Stoplight Studio Demo

Fix an incorrect OpenAPI Spec

Stoplight Public Workspace

https://dc1.stoplight.io/docs/dc-demo/YXBpOjl3OTE0NzQz-fix-employee-v1-api

API Standards

API Standardisation & Governance in Autodesk

- Core Principles
- API Standards
- API Versioning
- HATEOAS Rest Principle
- Localization
- Filtering
- Caching
- Pagination
- Resource Naming
- Naming conventions

API Standards - Core Principles

OpenAPI Specification

- Every HTTP/1.1 REST API **MUST** be described using an API description format.
- The API description format used **MUST** be the OpenAPI Specification (formerly known as Swagger).

  Current APIs __MUST__ use [OpenAPI 2](https://swagger.io/specification/v2/) or [OpenAPI 3](https://swagger.io/specification/). OpenAPI 3 is recommended.
- Every API description **MUST** be published in the API design platform.

API Standards - Core Principles

Autodesk OpenAPI Tools

- + [Documentation Pipeline](https://wiki.autodesk.com/display/FCPA/Forge+Documentation+Pipeline+Onboarding) - Documentation generation for the Forge Developer Portal.

- + [Resiliency SDK](https://git.autodesk.com/cloudplatform-apim/forge-rsdk-codegen) - Generates client SDKs from a Swagger file in multiple languages with resiliency patterns built in.

API Standards - Core Principles

API Design Platform

- [Stoplight](https://platform.stoplight.autodesk.com) is our primary platform supporting the API first workflow. Stoplight **SHOULD** be used during API design.
- Every API description **MUST** be stored in a git repository. Stoplight or another API design platform **MAY** be used to edit the file, but it should be stored permanently in a repository.
- The Swagger files **MUST** be the single source of truth to learn about existing APIs within the organization. The Swagger file **MUST** contain documentation for each API which will be used to generate API reference documentation.

API Standards - Core Principles

NOTE: Stoplight supports API-first approach in multiple ways: They validate API description for correctness and automatically generate API documentation to drive the discussion between stakeholders. (No more emails with API description flying between stakeholders)
Autodesk-managed service is available at platform.stoplight.autodesk.com and it is accessible from all of the company's networks, including VPN. Because it is deployed behind the corporate firewall, it can access git.autodesk.com, which allows for data synchronization between the two systems. More detailed information about the service is available on the service [main page](https://platform.stoplight.autodesk.com).

API Standards - Core Principles

Contract

- Approved API Design, represented by its API Description, **MUST** represent the contract between API stakeholder, implementors, and consumers.
- Any change to an API **MUST** be accompanied by a related update to the contract (API Description).

Contract Validation

- Every API description (contract) using HTTP(S) protocol **MUST** be tested against its API implementation.
- In addition to local runs, the tests **MUST** be an integral part the API implementation's CI/CD pipeline.
- The CI/CD pipeline **MUST** be configured to run the test after every change of the API implementation.

API Standards - Core Principles

Design Maturity

API Design Maturity

> How to design an API

- When designing an API an [API First](API_First.md) process **SHOULD** be followed. This practice surfaces design problems earlier in the lifecycle when they are easier to correct.
- Every API design **MUST** be resource-centric ([Web API Design Maturity Model Level 2](http://amundsen.com/talks/2016-11-apistrat-wadm/2016-11-apistrat-wadm.pdf). That is an API design **MUST** revolve around Web-styled resources, relations between the resources and the actions the resources may afford.
- An API **MAY** choose to support Web Maturity Model Level 3 ([HATEOAS](HATEOAS.md)) if appropriate for the service.

API Standards - Core Principles

API Design Implementation Maturity

> How to implement the API design

- Every API design implementation using the HTTP 1.1 protocol **MUST** use the appropriate HTTP Request Method ([Richardson Maturity Model Level 2](https://martinfowler.com/articles/richardsonMaturityModel.html#level2)) to implement an action afforded by a resource.

Robustness

- Every API implementation and API consumer **MUST** follow Postel's law:
- > Be conservative in what you send, be liberal in what you accept.
- >– John Postel
- That is, send the necessary minimum and be tolerant as possible while consuming another service ([tolerant reader](https://martinfowler.com/bliki/TolerantReader.html)).

API Standards - Core Principles

Rules for Extending

- Any modification to an existing API **MUST** avoid breaking changes and **MUST** maintain backward compatibility.
- In particular, any change to an API **MUST** follow the following Rules for Extending:
- You **MUST NOT** take anything away (related: Minimal Surface Principle, Robustness Principle)
- You **MUST NOT** change processing rules
- You **MUST NOT** make optional things required
- Anything you add **MUST** be optional (related Robustness Principle)
- See [API Versioning](Version_Versioning.md) for more details.

NOTE: These rules cover also renaming and change to identifiers (URIs). Names and identifiers should be stable over the time including their semantics.

API Standards - Core Principles

Loose Coupling

- In addition to the robustness principle, API consumers (clients) **MUST** operate independently of API implementation internals.
- Similarly, the API consumers **MUST NOT** assume or rely on any knowledge of the API service internal implementation.

API Standards - Versioning

 # API Versioning

> The fundamental principle is that you can’t break existing clients, because you don’t know what they implement, and you don’t control them.

> In doing so, you need to turn a backwards-incompatible change into a compatible one.

>> – [Mark Nottingham](https://www.mnot.net/blog/2011/10/25/web_api_versioning_smackdown)

Any modification to an existing API **MUST** maintain backward compatibility. Any change that does not maintain backward compatibility must use a new major version for that API, while maintaining support for the old version for a given period of time.

API Standards - Versioning

 # API Versioning

In particular, any change to an existing API **MUST** follow the following **Rules for Extending**:

- You **MUST NOT** take anything away
- You **MUST NOT** change processing rules
- You **MUST NOT** make optional things required
- Anything you add **MUST** be optional

All the rules above apply for:

- **Resource identifier** (resource name / URI) including any **query parameters** and their semantics
- **Resource metadata** (_e.g._ HTTP headers)
- **Actions** the resource affords (_e.g._ available HTTP Methods)
- **Relation** to other resources (_e.g._ Links)
- **Representation format** (_e.g._ HTTP request and response bodies)

API Standards - Versioning

 # API Versioning

URI-based versioning

The major version of a resource must include a version identifier in the path.

- Resource URL before a breaking change:

  https://developer.api.autodesk.com/data/v1/banners

- Resource URL after a breaking change:

  https://developer.api.autodesk.com/data/v2/banners

The resource version path should only include the major version number, like `/v1` or `/v3`. If your system uses semantic versioning, then the minor and patch version numbers __MUST NOT__ be included in the version path.

The old version __MUST__ remain available for a specified period of time. The exact period depends on business and technical decisions.

API Standards - Versioning

All new resources __MUST__ use a `/v1` (or the most recent [Global Version](API_Versioning.md#global-vs-per-resource-versioning)) version identifier in its path, following the service name, and preceding all other parts of the path.

API Standards - Versioning

 # API Versioning

API Versions for Changelogs

When an API or service is updated, typically the changes are are published publicly to let people know what changed. Changelogs should identify versions using dates

A service __MAY__ choose to use semantic version numbers internally, but the changes should be documented to clients publicly using dates, not the semantic version number.

Recommended Reading

+ [API Versioning Has No "Right Way"](https://blog.apisyouwonthate.com/api-versioning-has-no-right-way-f3c75457c0b7)

API Standards - Filtering

 # Filtering

A search (filter) operation on a collection resource __SHOULD__ be defined as safe, idempotent and cacheable, therefore using the GET HTTP request method.

Every search parameter __SHOULD__ be provided in the form of a query parameter. In the case of search parameters being mutually exclusive or requiring the presence of another parameter, the explanation **MUST** be part of operation's description.

Filters are specified in the query string using the name of a field, and a value or range of values to test against. The syntax in the query string is:

?filter[<fieldName>]=<valueToFilterOn>

Filters **MAY** supply a range of values rather than a specific value, using the `..` notation as shown in the examples below. This can be used for numeric ranges and date ranges.

API Standards - Filtering

 # Filtering

Examples

```
    GET /books?filter[title]=Great Expectations

Will return all books with the title `Great Expectations`.

    GET /books?filter[price]=10..20

Will return all books with a price in the range of 10 to 20, inclusive.

    GET /books?filter[price]=..50

Returns all the books with a price up to 50 dollars.

    GET /books?filter[price]=10..&filter[title]=The Bible

Will return all books with a price over 10, and the title `The Bible`
```

API Standards - Pagination

 # Pagination

Access to lists of data items __MAY__ support pagination for best client-side batch processing and iteration experience. This holds true for all lists that are (potentially) larger than just a few hundred entries.

There are two pagination techniques:

- Offset/Limit-based pagination: numeric offset identifies the first entry in the page

- Cursor-based — aka key-based — pagination: a unique key element identifies the first entry in the page.

Offset/Limit-based Pagination (Preferred Option)

An offset-based pagination __MUST__ accept the following query parameters:

- offset: Offset from the start of the collection to the first entry in the page. Zero based.

- limit: Determines the maximum number of objects that __MAY__ be returned. A query __MAY__ return fewer than the value of limit due to filtering or other reasons.

API Standards - Pagination

 # Pagination

The response of the offset based pagination MUST contain a pagination object with the following fields:

- offset: The offset value supplied in the request.

- limit: The limit value used by the server in the response. Note that this will be different from request in case of server overrides.

- totalResults: Optional. The total number of results that match the query irrespective of limit.

- nextUrl: [Link](Links.md) that will return the next page of data. If not included, this is the last page of data. It is allowed that a page may be empty but contain a next paging link. Consumers should stop paging only when the next link no longer appears.

- previousUrl: Optional. [Link](Links.md) that will return the previous page of data. If the previous link is supported, then the absence indicates the first page of results.

API Standards - Pagination

 # Pagination

Example

A response of offset-based paginated collection

```
{
   "pagination": {
    "limit": 20,
    "offset": 0,
    "totalResults": 21,
    "nextUrl": "https://developer.api.autodesk.com/data/v1/folder/contents?limit=20&offset=20"
   }, "results": [
    {...},
        ]
 }
```

API Standards - Resource Names

 # Resource Names

In resource-oriented APIs, *resources* are named entities, and *resource names* are their identifiers. Each resource **MUST** have its own unique resource name.

The resource name is made up of the ID of the resource itself, the IDs of any parent resources, and its API service name. We'll look at resource IDs and how a resource name is constructed below.

A *collection* is a special kind of resource that contains a list of sub-resources of identical type. For example, a directory is a collection of file resources. The resource ID for a collection is called collection ID.

The resource name is organized hierarchically using collection IDs and resource IDs, separated by forward slashes. If a resource contains a sub-resource, the sub-resource's name is formed by specifying the parent resource name followed by the sub-resource's ID - again, separated by forward slashes.

API Standards - Resource Names

Example 1: A storage service has a collection of buckets, where each bucket has a collection of objects

```
| API Service Name| Collection ID | Resource ID| Collection ID | Resource ID |
|-------------|----------|--------|----------|---------|
|storage.googleapis.com | `/buckets` | `/{bucket-id}` | `/objects` | `/{object-id}` |
```

API Standards - Resource Names

 # Resource Names

Example 2: An email service has a collection of users

Each user has a settings sub-resource, and the settings sub-resource has a number of other sub-resources, including customFrom:

API Service Name	Collection ID	Resource ID	Collection ID	Resource ID
mail.googleapis.com	`/users`	`/name@example.com`	`/settings`	`/customFrom`

An API producer can choose any acceptable value for resource and collection IDs as long as they are unique within the resource hierarchy.

API Standards - Resource Names

 # Resource Names

Prefix placement

The prefix is usually placed in the resource path in front of the specific resource name but it can be part of the API service domain name. Services defining their own domain can put part of the prefix in the domain name:

- https://developer.api.autodesk.com/sales/pricing/v1/items
- https://sales.api.autodesk.com/pricing/v1/items
- https://enterprise.api.autodesk.com/order/v3/orders

API Standards - Resource Names

 # Resource Names

Core APIs

Core APIs, consist of core Forge services, such as authentication and Forge Data Management, should follow this pattern:

/core/<category>/<service>/<version>/<resource path>

Examples:

```
/core/data/oss/v2/buckets

/core/data/asset-graph/v1/collections
```

API Standards - Resource Names

 # Resource Names

Customer Data APIs

Customer Data APIs consist of APIs for particular products, like BIM360. These APIs should use the following template:

<industry>/<service>/<version>/<resource path>

Examples:

```
/construction/issues/v2/projects/{projectId}/issues/{issueId}/attachments

/production/components/v3/entities
```

API Standards - Resource Names

 # Resource Names

Enterprise/Business APIs

Enterprise/Business APIs consists of B2B or B2C APIs for various domains like subscriptions, orders, channel partners, etc. These should use the following template:

<domain>/<version>/<resource path>

Examples:

```
order/v3/order-details/{id}
sales/v2/offerings/{id}
account/v2/accounts/{csn}
```

API Standards - Resource Names

 # Resource Names

Generic Pattern

APIs not covered by the sections above should still use an API prefix. This is the suggested form, but others could be considered, as appropriate.

<category>/<service>/<version>/<resource path>

Example:

```
/sales/pricing/v1/items
```

https://nordicapis.com/10-best-practices-for-naming-api-endpoints/

API Standards - Error Responses

 # Error Responses

An error response is intended for use with the HTTP status codes 4xx and 5xx. Error response **MUST NOT** be used with 2xx status code responses.

At the minimum, any error response **MUST** have the 'title' and 'detail' fields.

Example

```
{
   "title": "Authentication required",
   "detail": "Missing authentication credentials for the Greeting resource."
 }
```

API Standards - Error Responses

Error Responses

Optional Fields

It **SHOULD** have the 'type' field with the identifier of the error.

```
{
    "type": "https://developer.api.autodesk.com/problems/scv/unauthorized",
    "title": "Authentication required",
    "detail": "Missing authentication credentials for the Greeting resource."
  }
```

> **NOTE:** The type field is an identifier, and as such it **MAY** be used to denote additional error codes.

Additional Fields

If needed, the error response **MAY** include additional fields.

API Tools in Autodesk Landscape

- Stoplight - API Design & Documentation, Collaboration, Service Virtualisation/Mocking, API Doc Linting, API Governance, Code Generation &
- AWS - API Management
- Mural & Confluence Wiki for Design
- Overview of How various tools hang together in the landscape
- Forge API Linter - https://git.autodesk.com/devx/forge-api-linter

API Tools in Autodesk Landscape

Stoplight - API Design

- Provides a platform with visual modeling tools to create an OpenAPI document for your API — without requiring you to know the OpenAPI spec details or code the spec line by line. See http://stoplight.io/ for more information.

- Stoplight lets you switch between visual views and code views of the spec depending on your authoring preference. You're not limited to either the code or the visual designer -- you can switch to the view you prefer.

API Tools in Autodesk Landscape

AWS - API Management

AWS provides services for you to manage the whole lifecycle of your API. Specifically, AWS lets you “manage API complexity and risk in a multi- and hybrid-cloud world by ensuring security, visibility, and performance across the entire API landscape.” Supports the OpenAPI spec.

API Tools in Autodesk Landscape

Mural & Confluence Wiki for Design

- Ideation & Brainstorming
- Facilitate Workshops
- Strategy & Planning
- Client Engagements

AWS API Gateway Demo

API Gateway

- API
- Developers
- Products
- Apps

API Tools in Landscape

Overview of How various tools hang together in the landscape

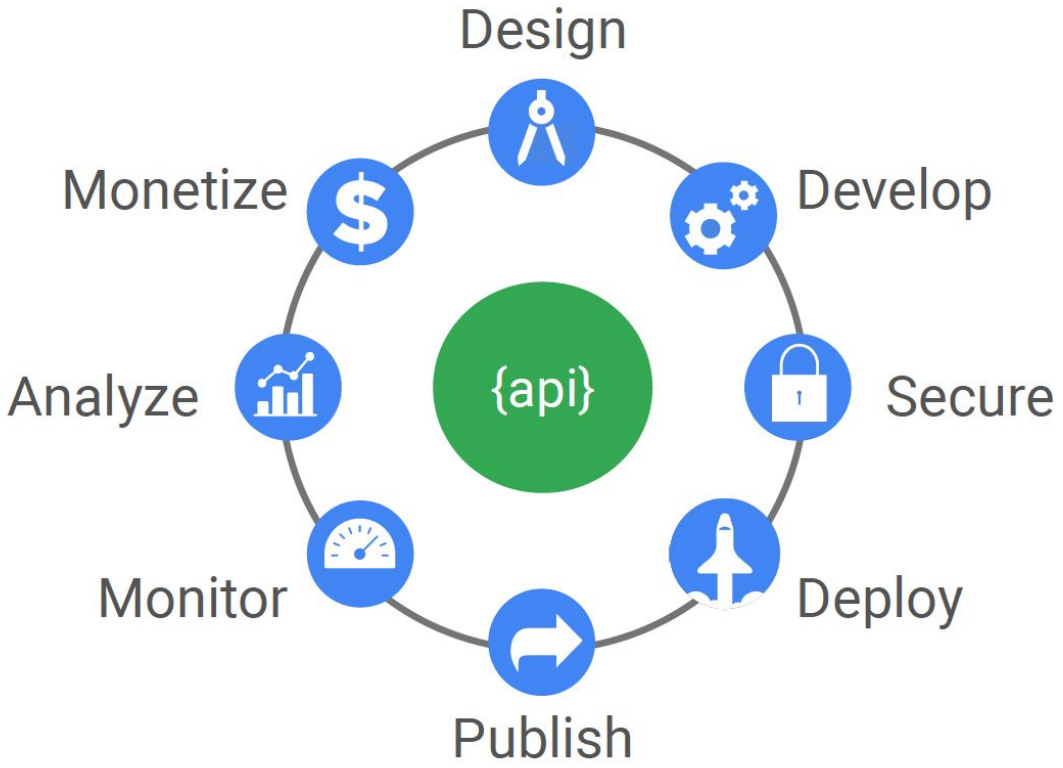
Improve API Experience With Autodesk Inhouse Mechanism - Eat your own DogFood

Overview of improving API experience through Dog-Fooding

API Lifecycle Management

- What is API Lifecycle Management ??

API Lifecycle Management



- Do you follow this style of development ??

DevOps Overview

What is Devops ??

DevOps Overview

What is Devops ??

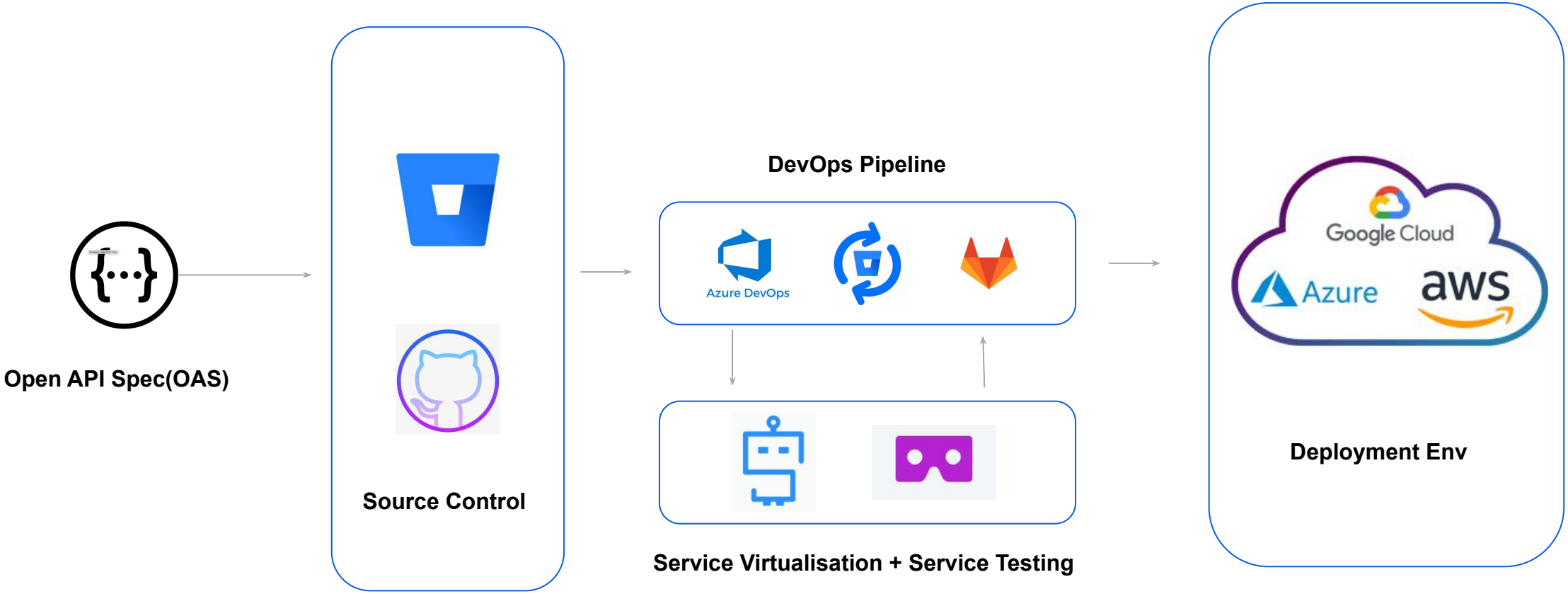
- To me DevOps is a marriage between Software development & automation. Embedding Automation into Software Development Lifecycle for removing manual errors, achieving consistency and having visibility through out the Software lifecycle.
- It is the process of collaboration , integration & automation of software and IT operations
- DevOps pipelines ensure that the delivery process is efficient, transparent, and secure by creating a consistent development environment and automating the process. The focus on DevOps enables developers to improve the control of production infrastructure and understand the production environment.

Refer:- https://nordicapis.com/the-role-of-apis-in-devops/

Relationship of API and DevOps

- Devops is all about automation & removing manual efforts
- Different tools requires API for integrations. Eg:- A test report that gets generated is sent to the slack channel of the gating team for production deployment approval. How does that happen ??
- Cloud native environments set-up using IaC (Infrastructure as Code). Spinning up new environments for performance/load testing of APIs
- Every CI/CD tool requires APIs to integrate with the tools that are called in the pipelines
- Promote code from lower environment to upper environment
- User onboarding & off-boarding process

Automated Testing DevOps(CI/CD) Pipeline



Advantages of Automated Testing in the DevOps(CI/CD) Pipeline

- Issues are discovered in the early stages (Just need to involve QA at an earlier stage - Right from Sprint Planning)
- Decrease in number of defects in production
- Time/Efforts saved in Manual Testing
- Wider adoption of API-First strategy in the Organization
- APIs/Code/Software can be promoted to production with Confidence

Disadvantages of Automated Testing in the DevOps(CI/CD) Pipeline

- Learning curve
- Adoption & Self-sufficiency take time
- Combination of People, process & technology. And it is not easy to drive all three mind-set together

API Security - Common Authentication Mechanism

- OAuth 2.0
- OIDC
- JWT
- Opaque Tokens
- Basic Authentication using Username & Password

API Ecosystem

https://www.raconteur.net/infographics/api-ecosystem/

Additional Resources

- https://apievangelist.com/
- https://nordicapis.com/5-examples-of-excellent-api-documentation/
- https://apithedocs.org/
- https://www.youtube.com/watch?v=5j_dzGCxtoM&t=75s
- https://www.youtube.com/watch?v=KcQJ3Q7h_Fo

Thank you!

If you have additional questions, please reach out to me at:
jkidd@kiddcorp.com