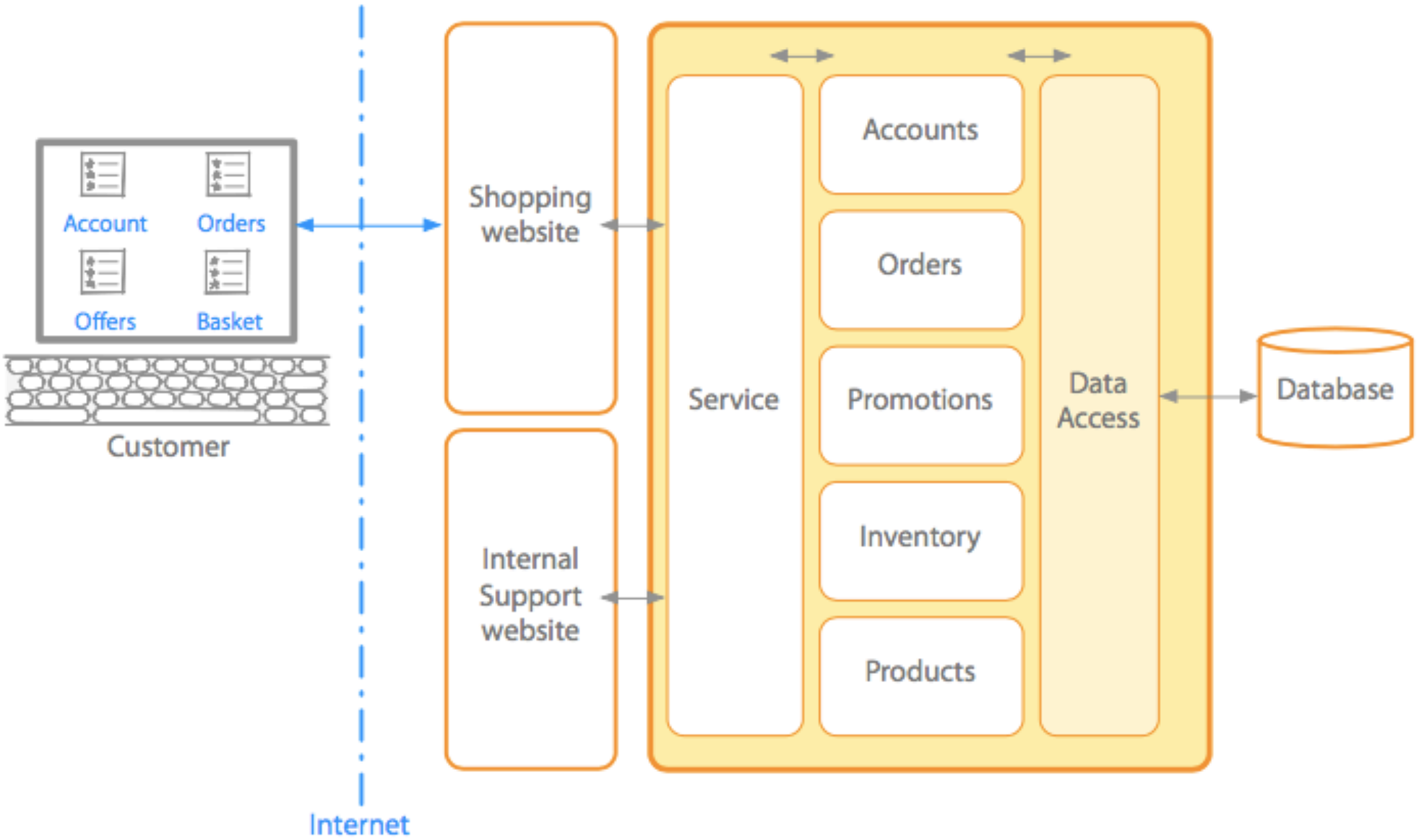
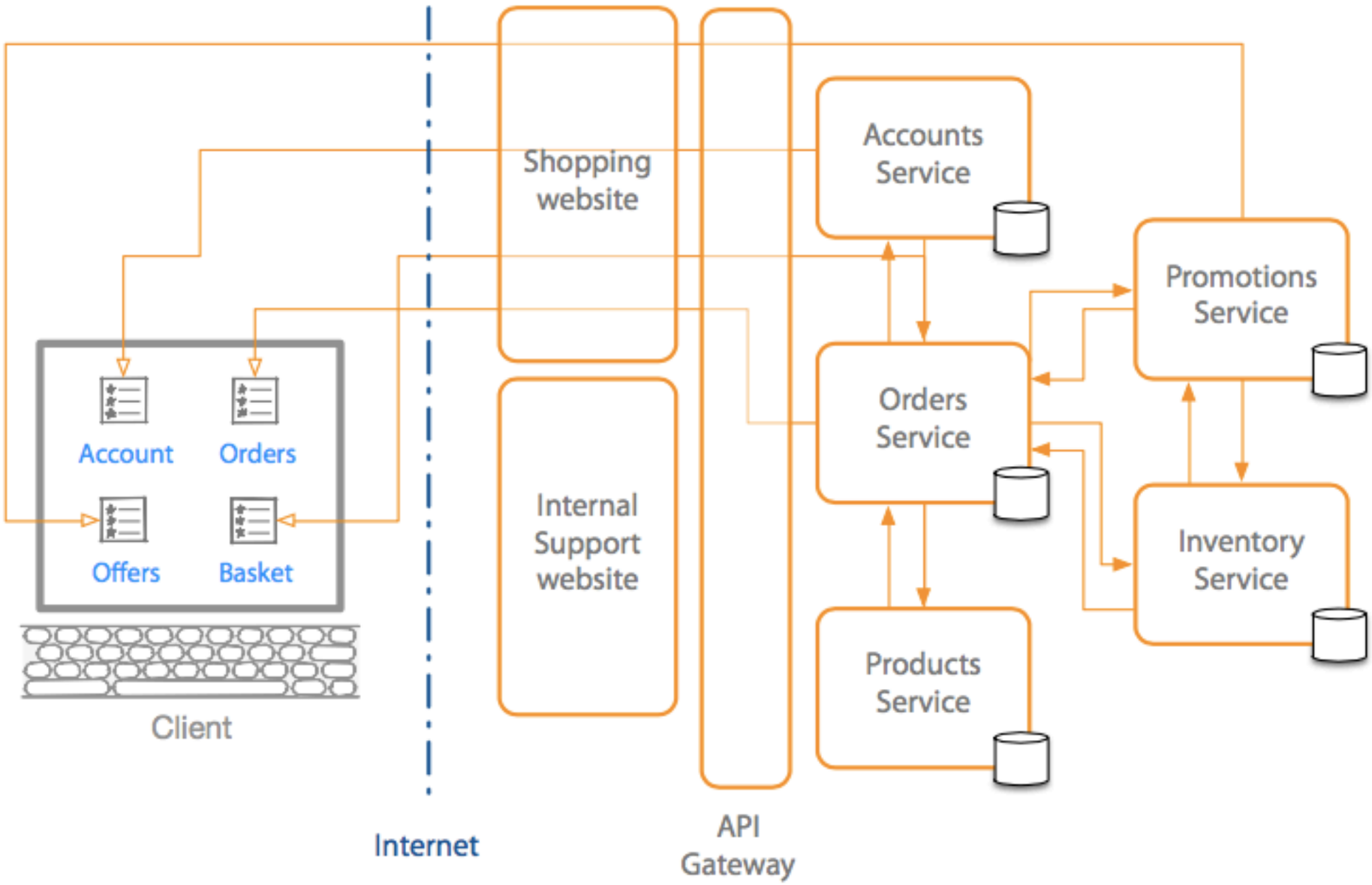
Microservices

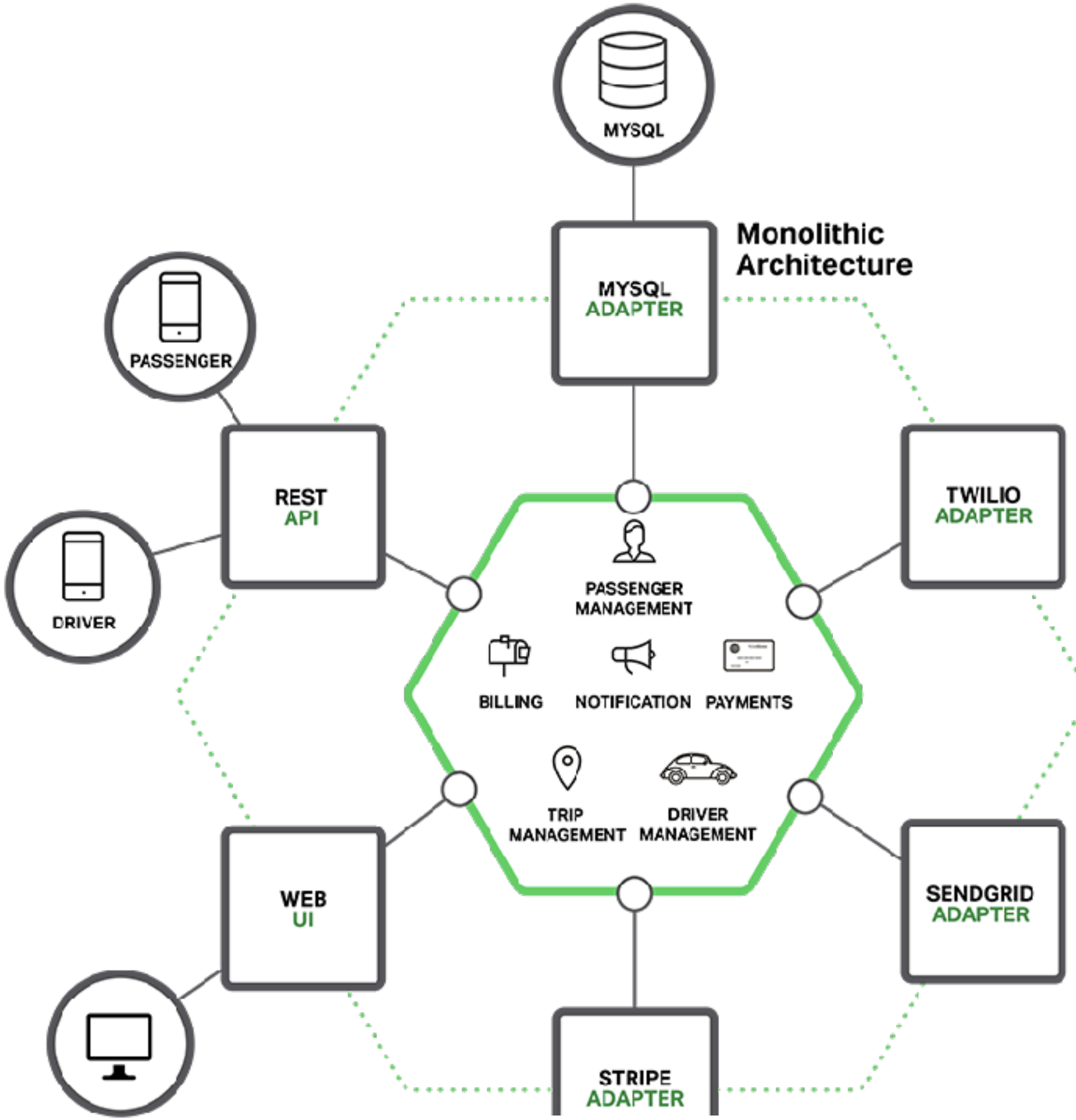
With Restful API

Monolithic

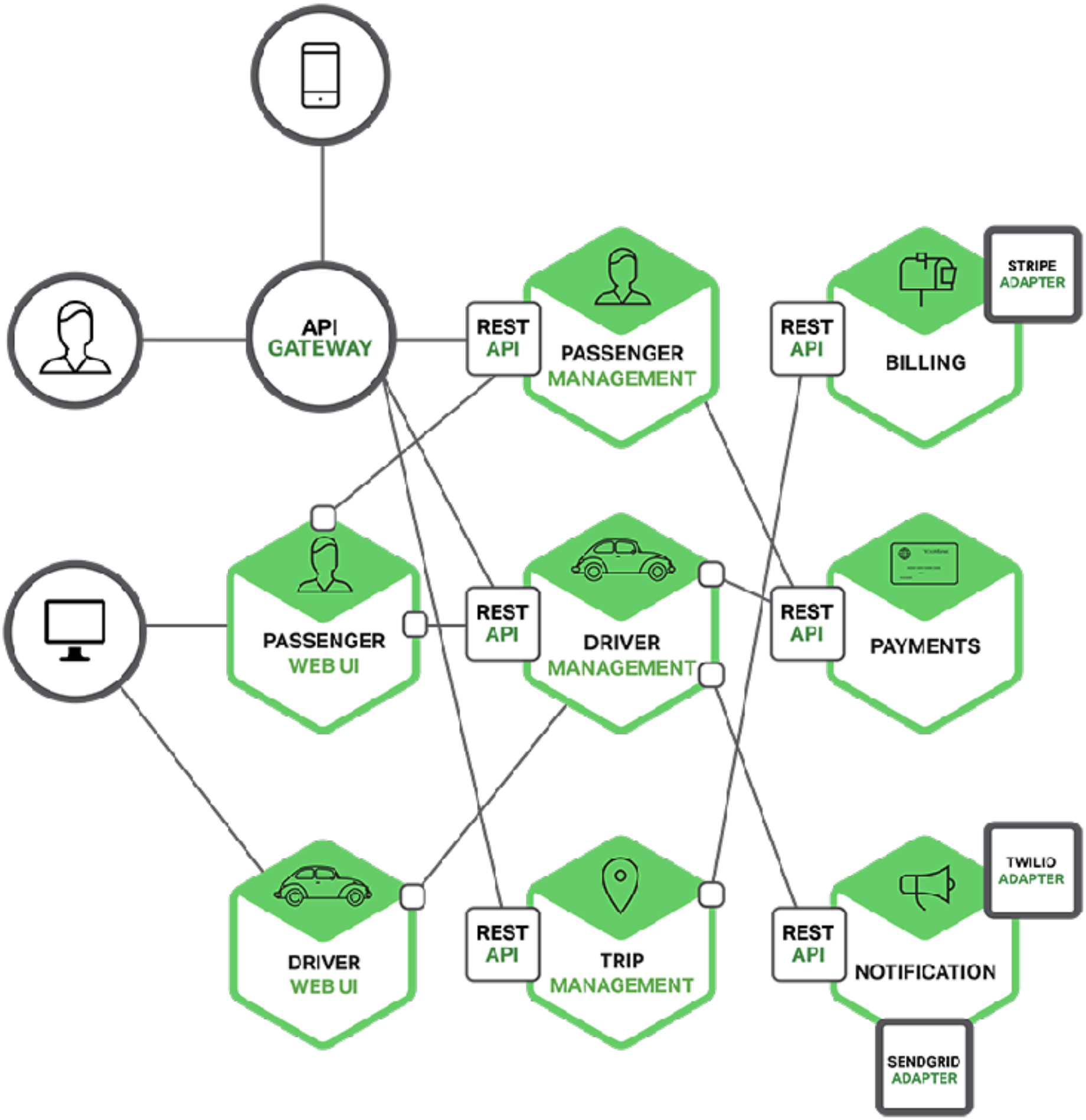

Account
Orders
Offers
Basket
Customer
Internet
Shopping website
Internal Support website
Accounts
Orders
Promotions
Inventory
Products
Service
Data Access
Database

Microservices

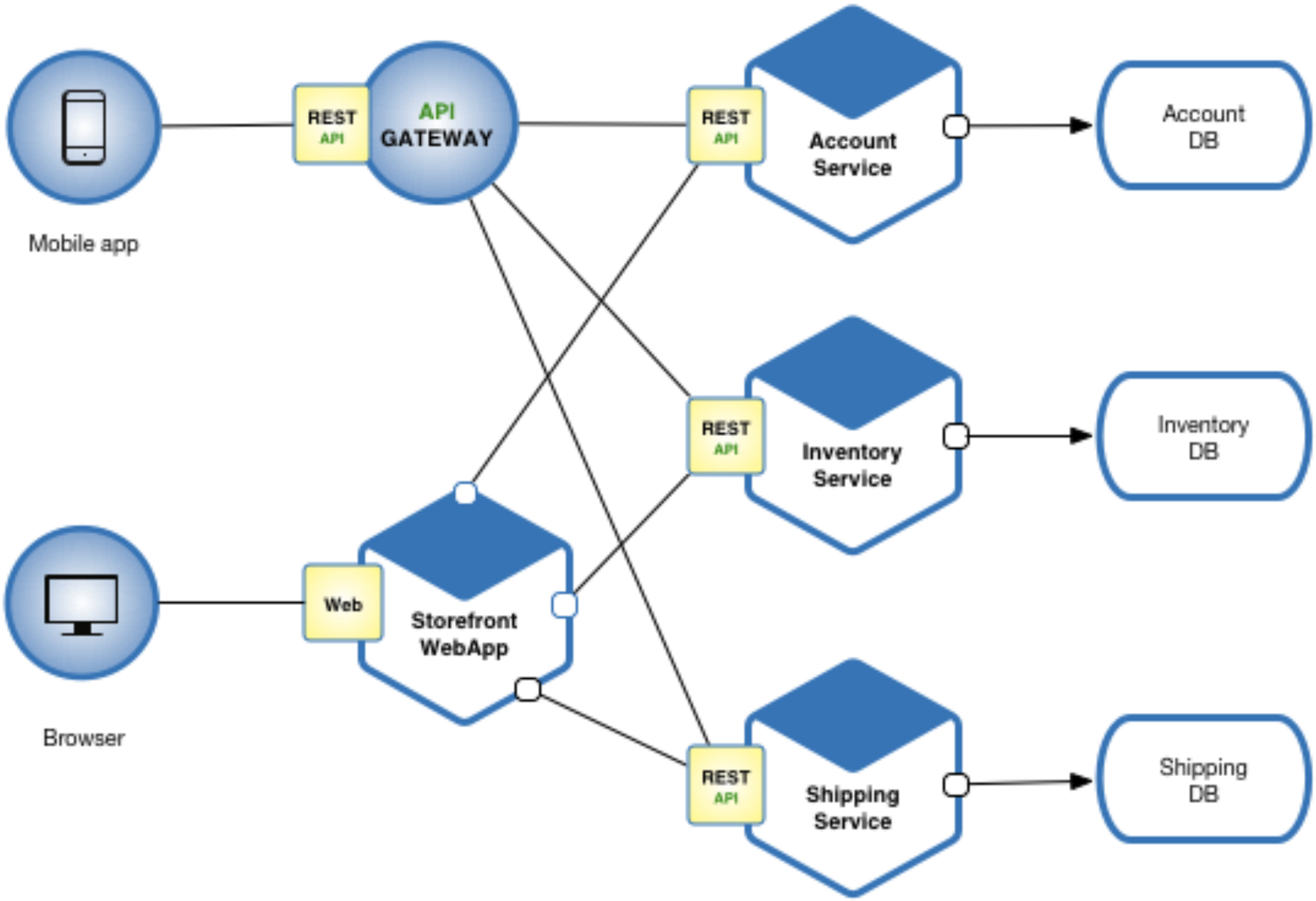




https://www.nginx.com/blog/introduction-to-microservices

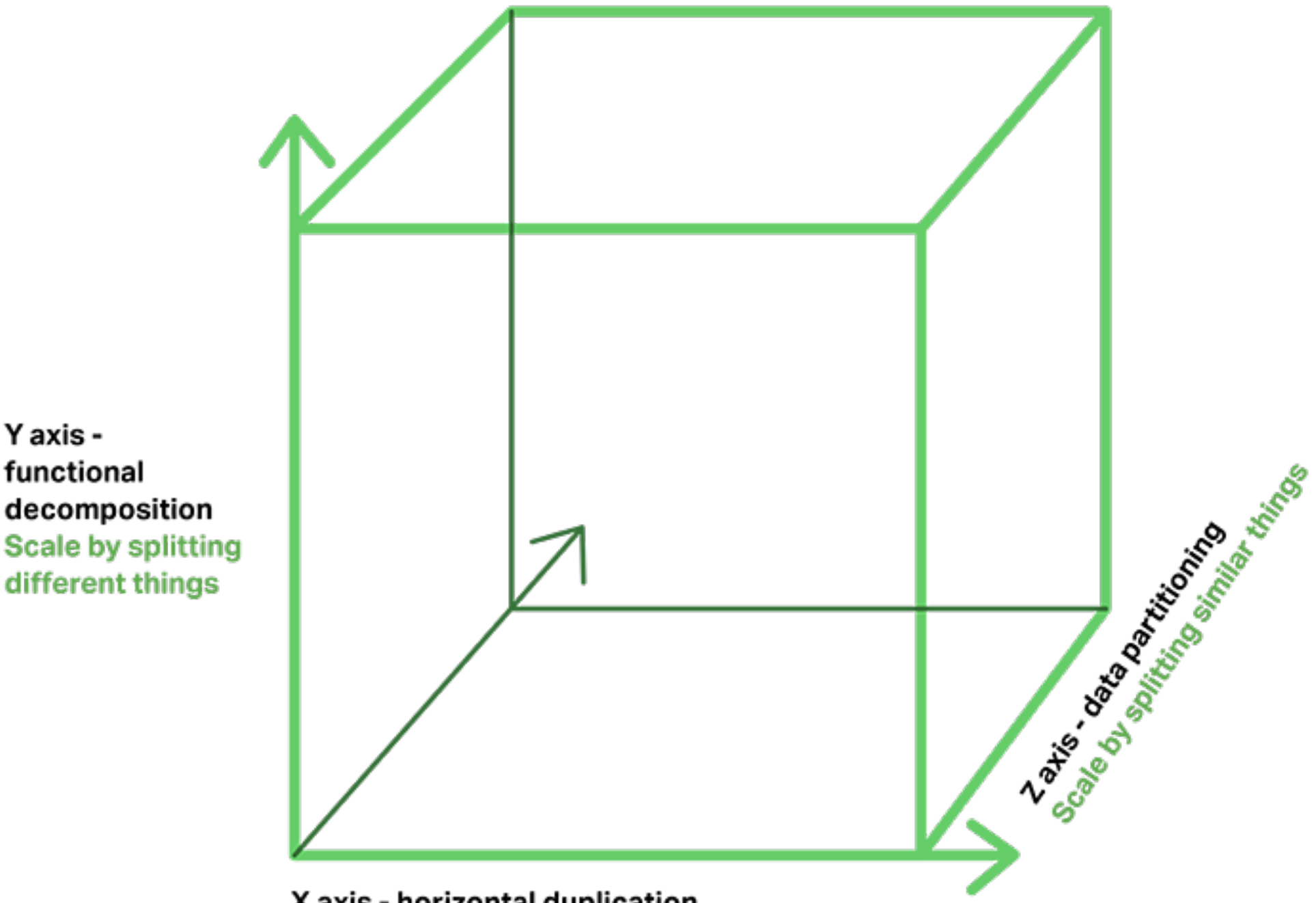

API
GATEWAY
REST
API
PASSENGER
MANAGEMENT
REST
API
BILLING
STRIPE
ADAPTER
PASSENGER
WEB UI
REST
API
DRIVER
MANAGEMENT
REST
API
PAYMENTS
DRIVER
WEB UI
REST
API
TRIP
MANAGEMENT
REST
API
NOTIFICATION
TWILIO
ADAPTER
SENDGRID
ADAPTER

E-commerce Application



https://microservices.io/patterns/microservices.html

The Art of Scalability



Design Principle

High Cohesion
Single thing done well
Single focus

Autonomous
Independently changeable
Independently deployable

Business Domain Centric
Represent business function or represent a business domain

Resillience
Embrace failure
Default or Degrade functionality

Observable
See System Health
Centralize logging and monitoring

Automation
Tools for testing and feedback
Tools for deployment

Approach

High Cohesion
Keeps splitting service until it only has one reason to change

Autonomous
Loosely Coupled system
Versioning strategy
Microservice Ownership by team

Business Domain Centric
Course grain business domain
Subgroups into functions and areas

Resillience
Design for known failures
Design for fail fast (recover fast)

Observable
Real-time centralized monitoring
Centralized logging

Automation
Continuous Integration
Cntinuous Deployment

Restful Services

RESTful Services provide uniform, standardized access with clear request requirements and response structure


API Endpoint
API Endpoint
API Endpoint

Uniform Interface	Stateless	Cacheable
Resources defined via URLs Client sends representation (e.g. JSON) HTTP Protocol	No client information stored on server All required information contained in request	Clients can cache data if marked as cacheable

Client and server are clearly separated	Client can't tell if connected to end server or intermediate	Executable code may be transferred from client to server
Client-Server	Layered System	Code-on-Demand (optional)

Usage

Product

Components A

Components B

17

17

API Example

API	Description	Request body	Response body	HTTP Response Code
GET /api/customer	Get all customers	None	Array of customers	200/OK
GET /api/customer/{id}	Get an customer by ID	None	Customer	200/OK
POST /api/customer	Add a new customer	Customer	Customer	201/Created
PUT /api/customer/{id}	Update an existing customer	Customer	None	200/OK
DELETE /api/customer/{id}	Delete a customer	None	None	204/No Content

Meeting Scheduler API

Users should be able to create, update, and delete meetings. Furthermore, other users should be able to register and unregister for any created meetings. Lastly, it should be possible to retrieve a list of all meetings or data about an individual meeting

Decomposition of the API

Create Meeting

Register for Meeting

Get list of all meetings

Update Meeting

Unregister for Meeting

Get data about individual meetings

Delete Meeting

Create Users

Main Http Method

GET
Retrieve a resource

POST
Add a resource

PUT
Replace a resource

PATCH
Update parts of a resource

DELETE
Remove a resource

Meeting Schedule API

POST

Create Meeting

POST

Register for Meeting

GET

Get List of all Meetings

PATCH Replace Fields

Update Meeting

DELETE

Unregister from Meeting

GET

Get Data about individual Meeting

DELETE

Delete Meeting

POST

Create User

Why URL Styling?

Expressive
Clear idea of accessed resource or action

Updatable
Easy usage of multiple API versions

Intuitive Navigation
Allows inference of other endpoints

Comparison of URL Styles

GET, POST, ...			
/api/meeting/get/5			
/api/meeting/5			
/api/v1/meeting/5			

Final Structure

POST
Create Meeting
↑ Title, Description, Time, User ID
↓ Message, Summary, URL, Users
/api/v1/meeting

POST
Register
↑ User ID, Meeting ID
↓ Message, User, Meeting, URL
/api/v1/meeting/registration

GET
Get List of all Meetings
↑ (null)
↓ List of Meetings, URL
/api/v1/meeting

PATCH
Update Meeting
↑ Meeting Data, User ID, Meeting ID
↓ Message, Summary, URL, Users
/api/v1/meeting

DELETE
Unregister
↑ User ID, Meeting ID
↓ Message, User, Meeting, URL
/api/v1/meeting/registration

GET
Get Meeting
↑ Meeting ID
↓ Meeting Info, URL
/api/v1/meeting

DELETE
Delete Meeting
↑ Meeting ID, User ID
↓ Message,
/api/v1/meeting

POST
Create User
↑ Name, E-Mail, Password
↓ Message, User, Meeting, URL
/api/v1/meeting/user

POST
User Signin
↑ E-Mail, Password
↓ ???
/api/v1/user/signin

Authentication required
No authentication required

Resource Controllers

- Laravel resource routing assigns the typical "CRUD" routes to a controller with a single line of code.

```
php artisan make:controller PhotoController --resource
```

- This command will generate a controller at app/Http/Controllers/PhotoController.php. The controller will contain a method for each of the available resource operations

Resource Controller

- Register a resourceful route to the controller

```
Route::resource('photos', 'PhotoController');
```

- This single route declaration creates multiple routes to handle a variety of actions on the resource

Actions Handled

Verb	URI	Action	Route Name
GET	`/photos`	index	photos.index
GET	`/photos/create`	create	photos.create
POST	`/photos`	store	photos.store
GET	`/photos/{photo}`	show	photos.show
GET	`/photos/{photo}/edit`	edit	photos.edit
PUT/PATCH	`/photos/{photo}`	update	photos.update
DELETE	`/photos/{photo}`	destroy	photos.destroy

Partial Resource Routes

```
Route::resource('photo', 'PhotoController', ['only' => [
    'index', 'show'
]]);
```

```
Route::resource('photo', 'PhotoController', ['except' => [
    'create', 'store', 'update', 'destroy'
]]);
```

RESTful Controller

HTTP Method & URL	Controller Action
GET /post	index [show all posts]
GET /post/create	create [get creation form]
POST /post	store [store post on server]
GET /post/{post}	show [get single post]
GET /post/{post}/edit	edit [get edit form]
PUT/PATCH /post/{post}	update [save update on server]
DELETE /post/{post}	destroy [delete post on server]

Route Group

```php
// app/Http/routes.php

Route::group(['prefix' => 'api/v1'], function() {
    Route::post('/post', [
        // Route Configuration
    ]);
});
```

RESTful Microservices

/api/customer/calculateInvoice/2017

/api/routes/optimizeRoutes/

/api/Offers/expireVouchers/

/api/customer/emailVoucher/

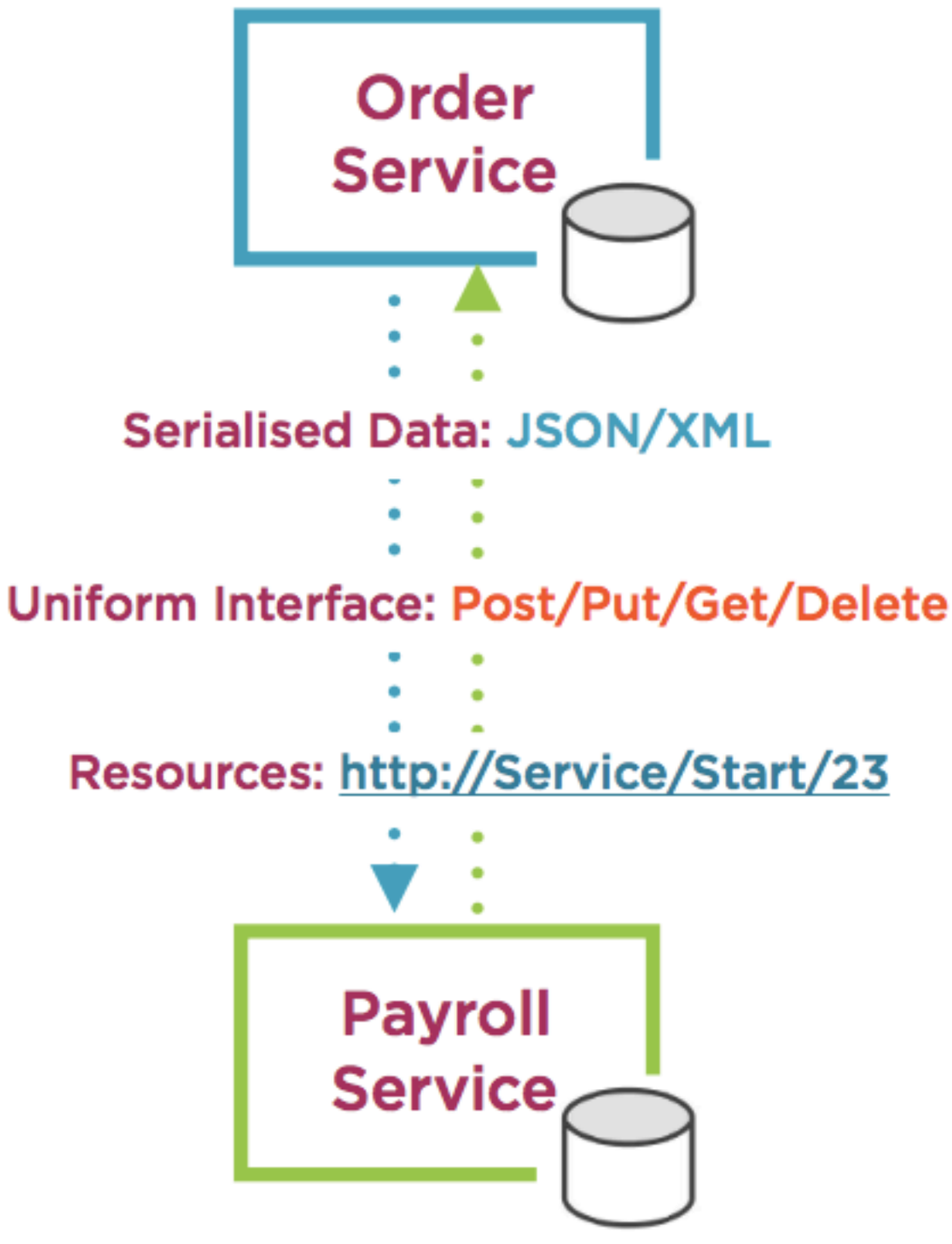
/api/reports/consolidate/

/api/Offers/expireVouchers/453

Additions for Microservices

- Not all about resource CRUD
- Action\Task based endpoints
- Verbs instead of nouns
- Query string and or request body
- Response body for output
- Callback address for output
- HTTP status codes

Task Based Microservices



- POST to start task
  https://myPayrollApi.net/start/{id}
- GET to retrieve task status
  https://myPayrollApi.net/task/{task-id}
- DELETE to cancel task
  https://myPayrollApi.net/task/{task-id}

Documenting Microservices Contract

Tools

Apiary

Swagger

RAML

Apiary

ORACLE + apiary

How Apiary Works Product Plans Company Help Sign in Sign up

Powerful API Design Stack. Built for Developers.

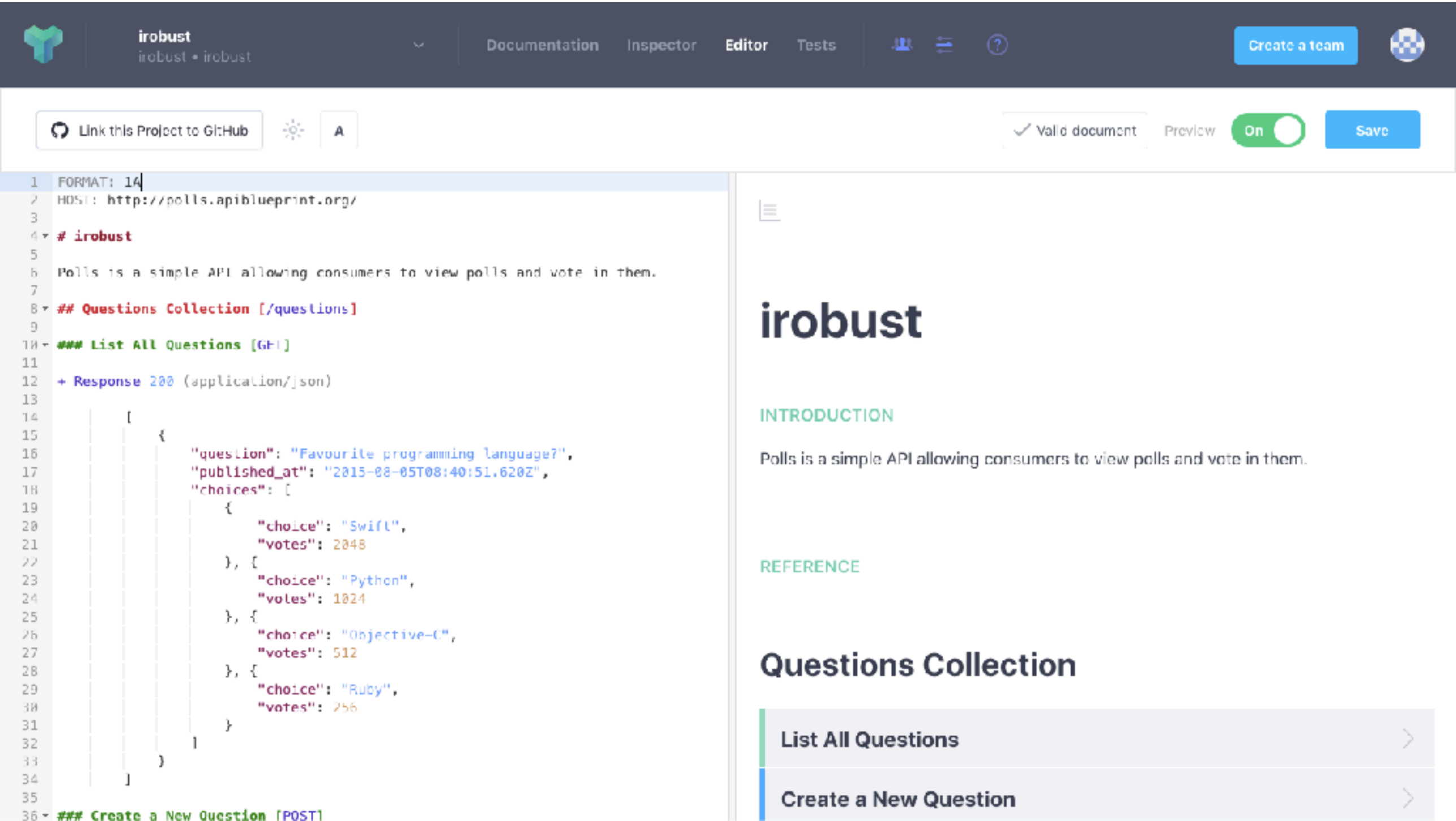
Work together to quickly design, prototype, document and test APIs

Sign up free with GitHub

No GitHub? **Sign up free with email!**

Set your team on the right course – **Visibility. Consistency. Process.**

Apiary


irobust
irobust • irobust
Documentation
Inspector
Editor
Tests
Create a team
Link this Project to GitHub
A
Valid document
Preview
On
Save
FORMAT: 1A
HOST: http://polls.apiblueprint.org/
irobust
Polls is a simple API allowing consumers to view polls and vote in them.
Questions Collection [/questions]
List All Questions [GET]
+ Response 200 (application/json)
[
{
"question": "Favourite programming language?",
"published_at": "2015-08-05T08:40:51.620Z",
"choices": [
{
"choice": "Swift",
"votes": 2048
}, {
"choice": "Python",
"votes": 1024
}, {
"choice": "Objective-C",
"votes": 512
}, {
"choice": "Ruby",
"votes": 256
}
]
}
]
Create a New Question [POST]
irobust
INTRODUCTION
Polls is a simple API allowing consumers to view polls and vote in them.
REFERENCE
Questions Collection
List All Questions
Create a New Question

Apiary

```
FORMAT: 1A
HOST: http://polls.apiblueprint.org/

# irobust

Polls is a simple API allowing consumers to view polls and vote in them.

## Questions Collection [/questions]

### List All Questions [GET]

### Create a New Question [POST]
```

Apiary

```
### List All Questions [GET]

+ Response 200 (application/json)

        [
            {
                "question": "Favourite programming language?",
                "published_at": "2015-08-05T08:40:51.620Z",
                "choices": [
                    {
                        "choice": "Swift",
                        "votes": 2048
                    }, {
                        "choice": "Python",
                        "votes": 1024
                    }, {
                        "choice": "Objective-C",
                        "votes": 512
                    }, {
                        "choice": "Ruby",
                        "votes": 256
                    }
                ]
            }
        ]
```

+ Request (application/json)

```
{
    "question": "Favourite programming language?",
    "choices": [
        "Swift",
        "Python",
        "Objective-C",
        "Ruby"
    ]
}
```

+ Response 201 (application/json)

    + Headers

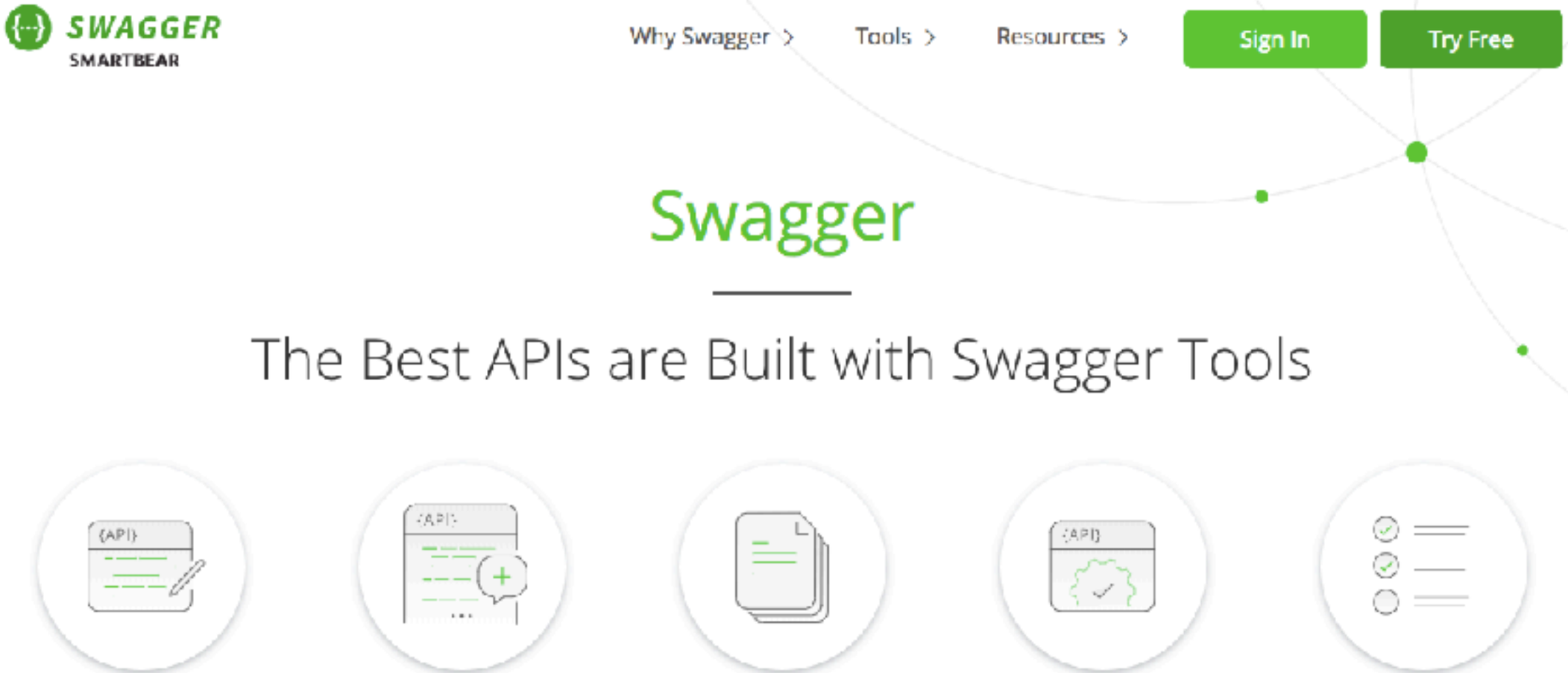
```
Location: /questions/2
```

    + Body

```
{
    "question": "Favourite programming language?",
    "published_at": "2015-08-05T08:40:51.620Z",
    "choices": [
        {
            "choice": "Swift",
            "votes": 0
        }, {
            "choice": "Python",
            "votes": 0
        }, {
            "choice": "Objective-C",
            "votes": 0
        }, {
            "choice": "Ruby",
```

Request and Response

Swagger

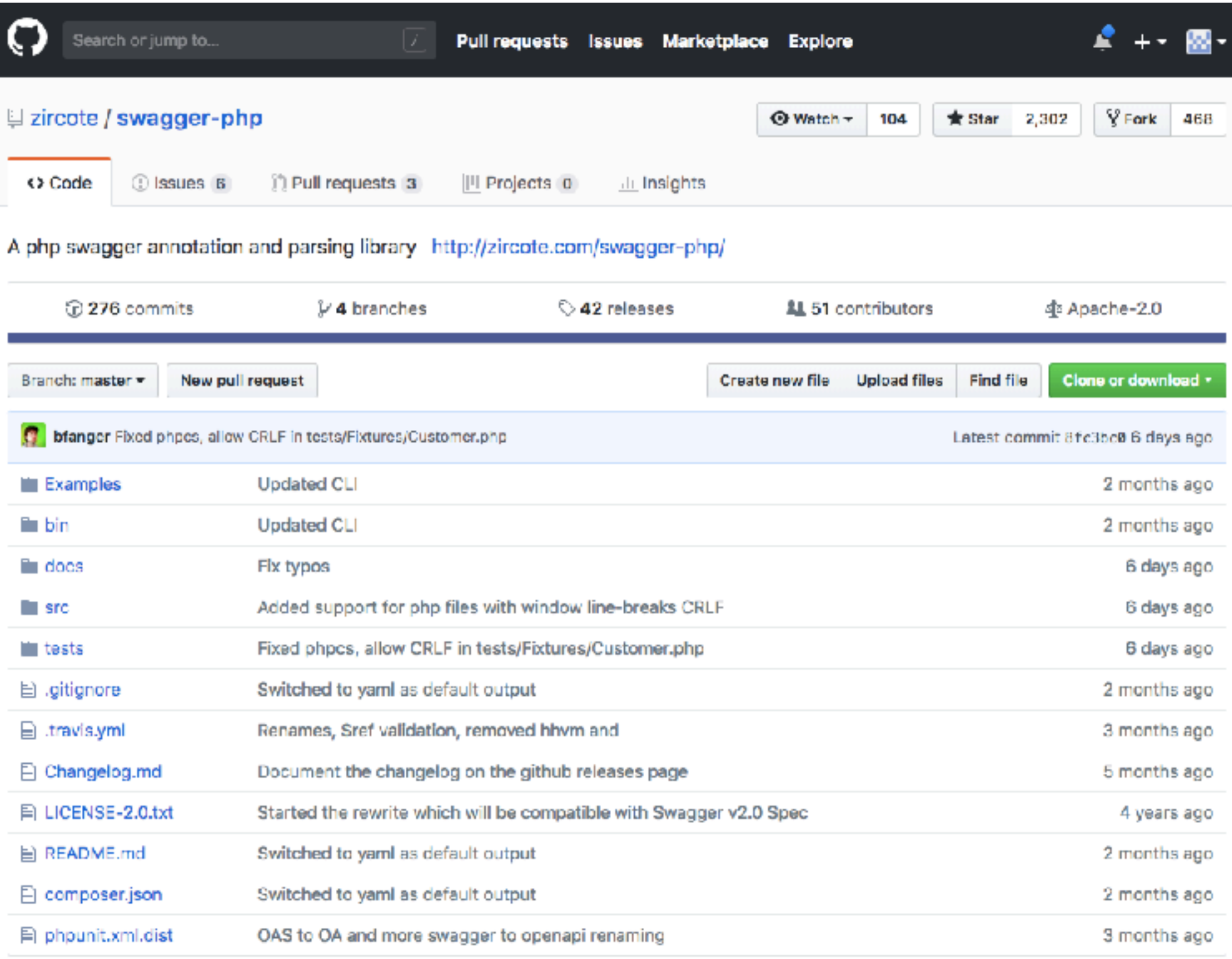

SWAGGER
SMARTBEAR
Why Swagger
Tools
Resources
Sign In
Try Free
Swagger
The Best APIs are Built with Swagger Tools
Design
Design and model APIs according to specification-based standards
Build
Build stable, reusable code for your API in almost any language
Document
Improve developer experience with interactive API documentation
Test
Perform simple functional tests on your APIs without overhead
Standardize
Set and enforce API style guidelines across your API architecture

Swagger Component

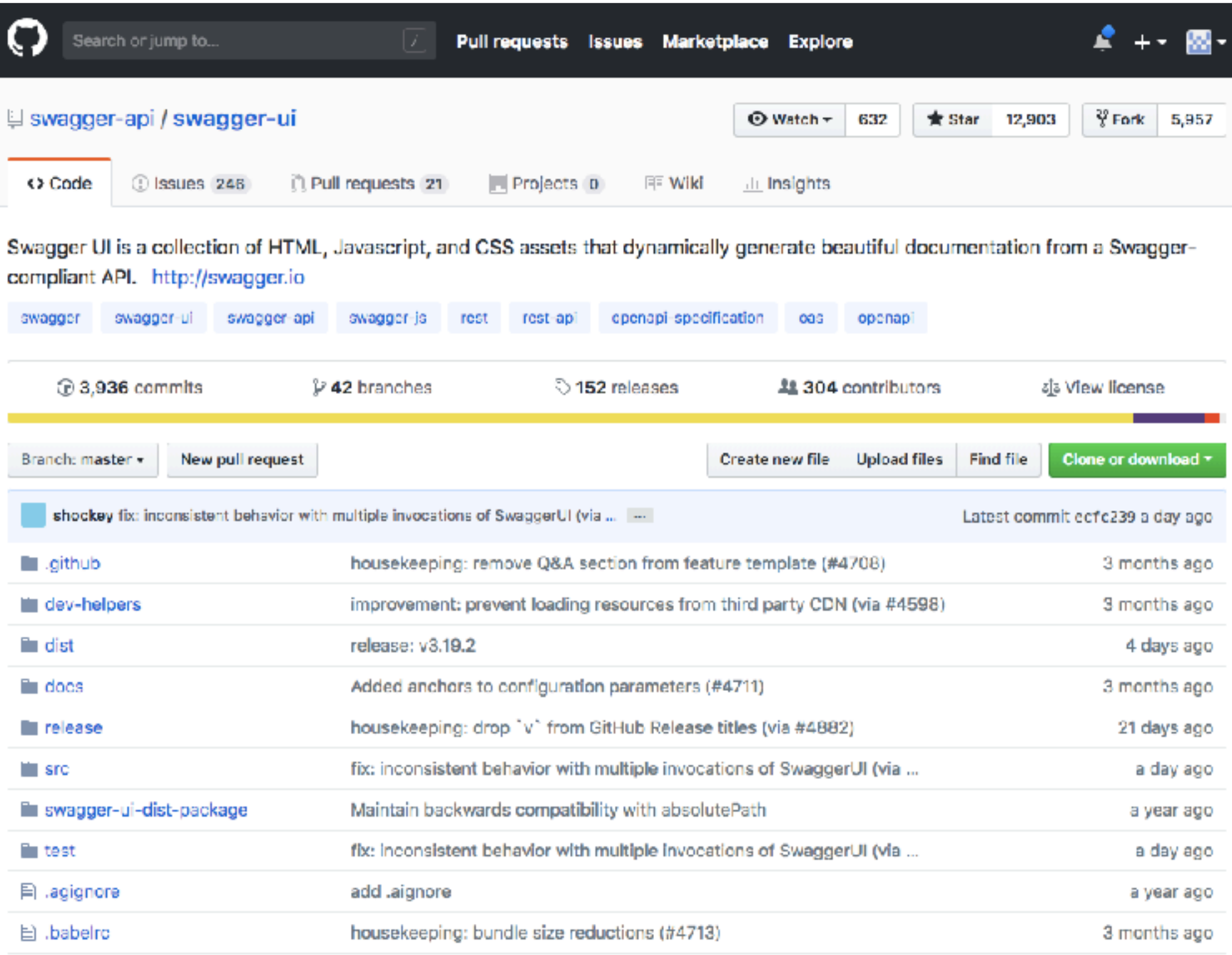
Swagger PHP

Swagger UI

Swagger PHP


Search or jump to...
Pull requests
Issues
Marketplace
Explore
zircote / swagger-php
Watch 104
Star 2,302
Fork 468
Code
Issues 6
Pull requests 3
Projects 0
Insights
A php swagger annotation and parsing library http://zircote.com/swagger-php/
276 commits
4 branches
42 releases
51 contributors
Apache-2.0
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
bfanger Fixed phpcs, allow CRLF in tests/Fixtures/Customer.php
Latest commit 8fc3bc0 6 days ago
Examples Updated CLI 2 months ago
bin Updated CLI 2 months ago
docs Fix typos 6 days ago
src Added support for php files with window line-breaks CRLF 6 days ago
tests Fixed phpcs, allow CRLF in tests/Fixtures/Customer.php 6 days ago
.gitignore Switched to yaml as default output 2 months ago
.travis.yml Renames, Sref validation, removed hhvm and 3 months ago
Changelog.md Document the changelog on the github releases page 5 months ago
LICENSE-2.0.txt Started the rewrite which will be compatible with Swagger v2.0 Spec 4 years ago
README.md Switched to yaml as default output 2 months ago
composer.json Switched to yaml as default output 2 months ago
phpunit.xml.dist OAS to OA and more swagger to openapi renaming 3 months ago

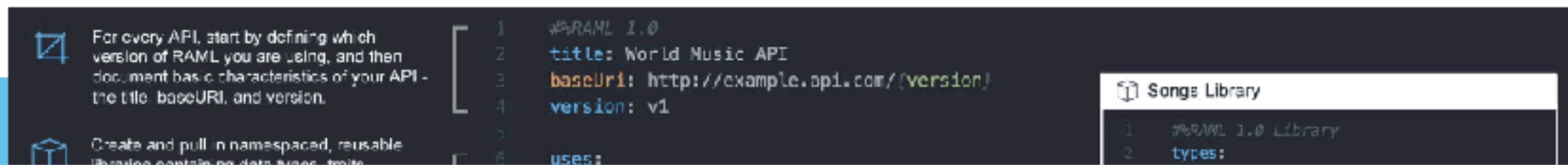
Swagger UI


Search or jump to...
Pull requests
Issues
Marketplace
Explore
swagger-api / swagger-ui
Watch 632
Star 12,903
Fork 5,957
Code
Issues 246
Pull requests 21
Projects 0
Wiki
Insights
Swagger UI is a collection of HTML, Javascript, and CSS assets that dynamically generate beautiful documentation from a Swagger-compliant API. http://swagger.io
swagger
swagger-ui
swagger api
swagger js
rest
rest api
openapi specification
oas
openapi
3,936 commits
42 branches
152 releases
304 contributors
View license
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
shockey fix: inconsistent behavior with multiple invocations of SwaggerUI (via ...
Latest commit ccfc239 a day ago
.github housekeeping: remove Q&A section from feature template (#4708) 3 months ago
dev-helpers improvement: prevent loading resources from third party CDN (via #4598) 3 months ago
dist release: v3.19.2 4 days ago
docs Added anchors to configuration parameters (#4711) 3 months ago
release housekeeping: drop `v` from GitHub Release titles (via #4882) 21 days ago
src fix: inconsistent behavior with multiple invocations of SwaggerUI (via ... a day ago
swagger-ui-dist-package Maintain backwards compatibility with absolutePath a year ago
test fix: inconsistent behavior with multiple invocations of SwaggerUI (via ... a day ago
.agignore add .aignore a year ago
.babelrc housekeeping: bundle size reductions (#4713) 3 months ago

RAML


RAML
GitHub
Forum
Slack
Twitter
For Developers
For Enterprises
Projects
Blog
About
The simplest way to design APIs
RESTful API Modeling Language (RAML) makes it easy to manage the whole API lifecycle from design to sharing. It's concise - you only write what you need to define - and reusable. It is machine readable API design that is actually human friendly.
Write once. Use many. Creative laziness encouraged.
DESIGN
BUILD
TEST
DOCUMENT
SHARE
RAML 1.0 Specification
RAML 0.8 Specification


healthwise
SONIC
Spotify
PBS
CISCO
vmware


For every API, start by defining which version of RAML you are using, and then document basic characteristics of your API - the title, baseURI, and version.
#%RAML 1.0
title: World Music API
baseUri: http://example.api.com/{version}
version: v1
uses:
Songs Library
#%RAML 1.0 Library
types:

RAML

For every API, start by defining which version of RAML you are using, and then document basic characteristics of your API - the title, baseURI, and version.

Create and pull in namespaced, reusable libraries containing data types, traits, resource types, schemas, examples, & more.

Annotations let you add vendor specifc functionality without compromising your spec

Traits and resourceTypes let you take advantage of code reuse and design patterns

Easily define resources and methods, then add as much detail as you want. Apply traits and other patterns, or add parameters and other details specific to each call.

Describe expected responses for multiple media types and specify data types or call in pre-defined schemas and examples. Schemas and examples can be defined via a data type, in-line, or externalized with !include.

Write human-readable, markdown-formatted descriptions throughout your RAML spec, or include entire markdown documentation sections at the root.

```yaml
#%RAML 1.0
title: World Music API
baseUri: http://example.api.com/{version}
version: v1

uses:
  Songs: libraries/songs.raml

annotationTypes:
  monitoringInterval:
    parameters:
      value: integer

traits:
  secured: !include secured/accessToken.raml

/songs:
  is: [ secured ]
  get:
    (monitoringInterval): 30
    queryParameters:
      genre:
        description: filter the songs by genre
  post:
  /{songId}:
    get:
      responses:
        200:
          body:
            application/json:
              type: Songs.Song
            application/xml:
              schema: !include schemas/songs.xml
              example: !include examples/songs.xml
```

Songs Library

```yaml
#%RAML 1.0 Library
types:
  Song:
    properties:
      title: string
      length: number
  Album:
    properties:
      title: string
      songs: Song[]
  Musician:
    properties:
      name: string
      discography: (Song | Album)[]
```

songs.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="song">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="title" type="xs:string">
          </xs:element>
          <xs:element name="artist" type="xs:string">
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
</xs:schema>
```

RAML

Interface Definition Languages

- Way to document microservice contracts
  Before or after implementation

- Platform and language independent

- Computer readable format

- Supporting tools provide extra functionality

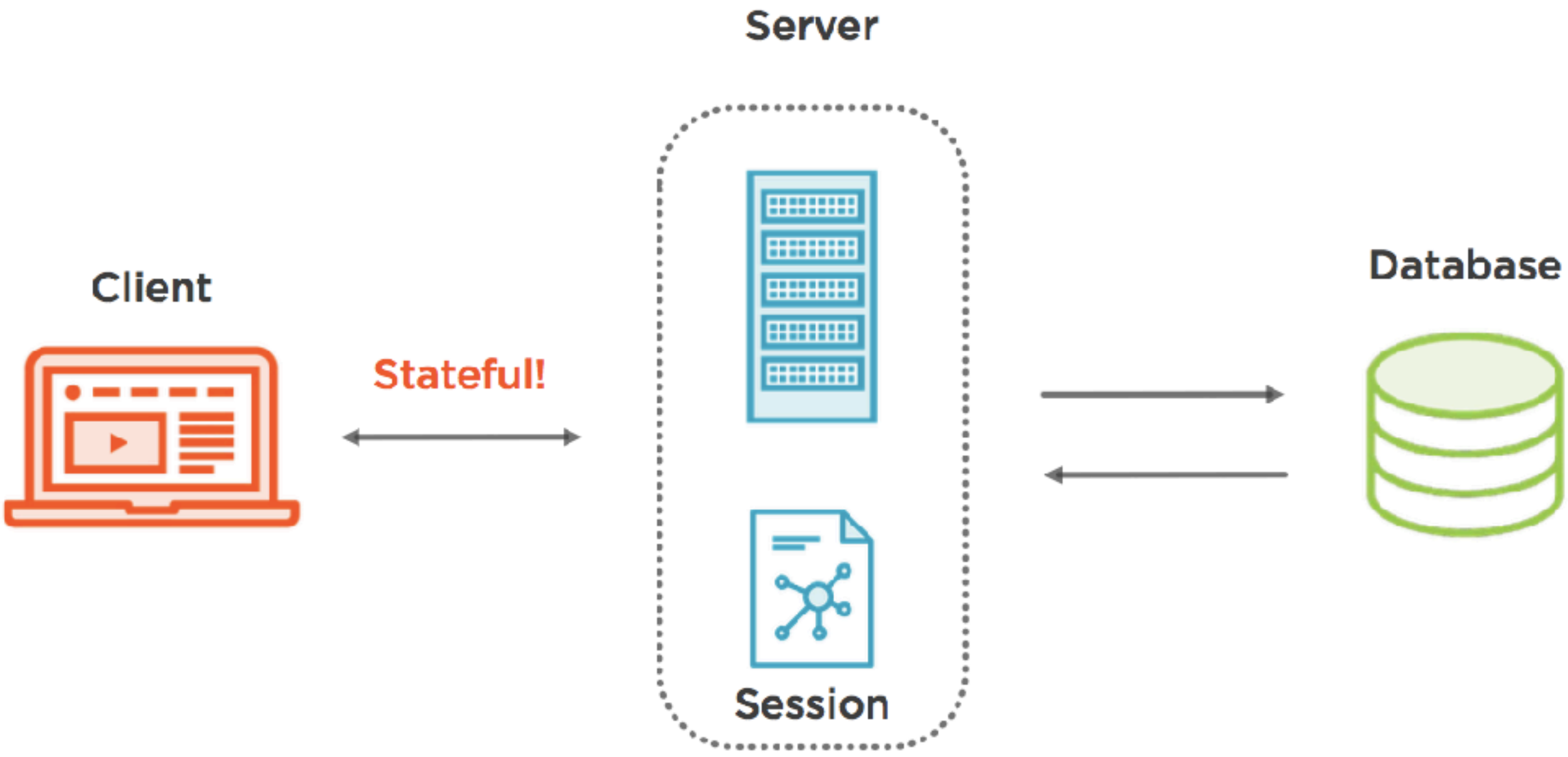
- Generate mock APIs

Interface Definition Languages

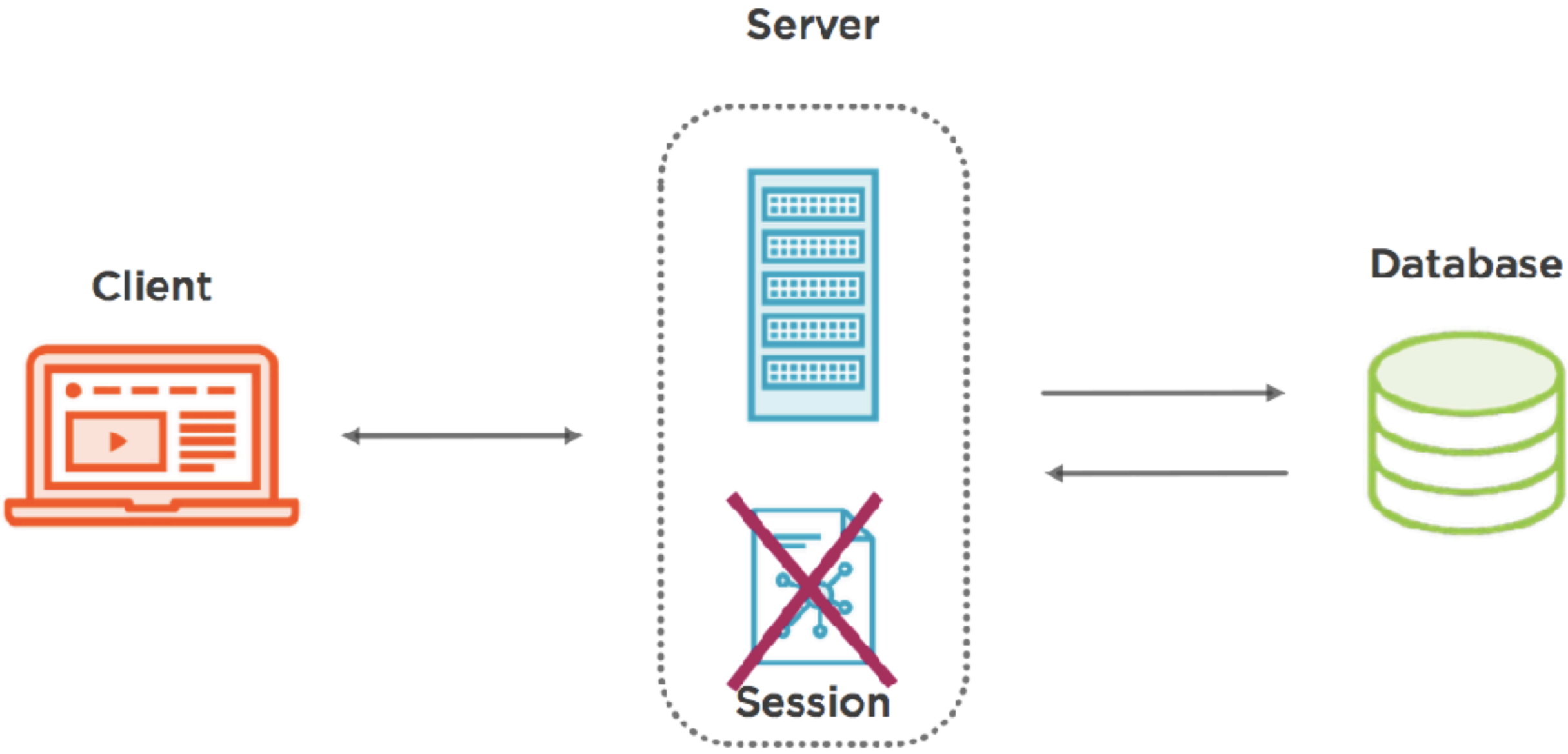
- Code generators for client and or service
- Validate client requests and server responses
- Generate documentation
- Tools that produce websites to call your API
- Generate test suites for your API endpoints

Authen & Authorize Framework

Traditional Authentication



Service are Stateless



Things Used to be a Little Different

Past

- XML
- SOAP
- SAML or WS-*

Present

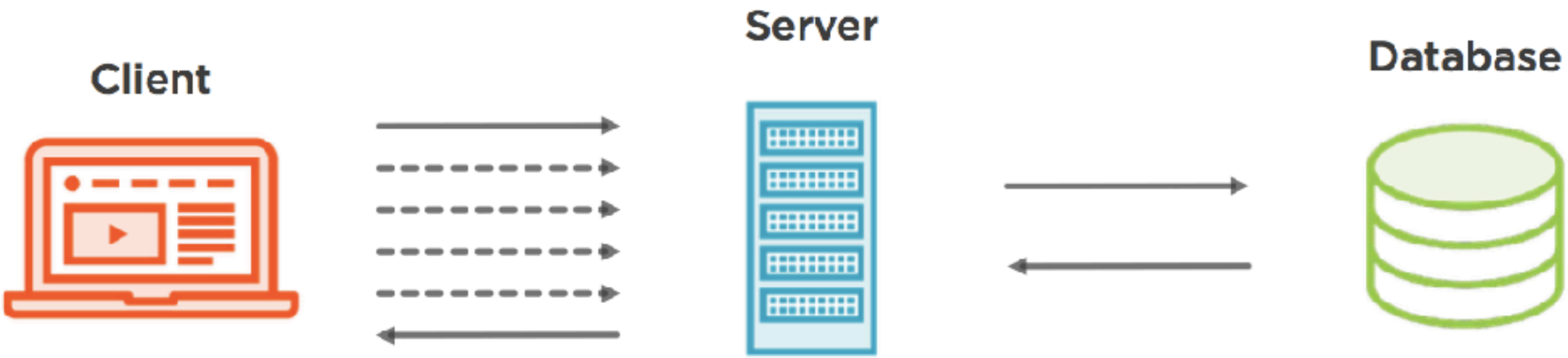
- JSON
- HTTP APIs
- OAuth & OpenID Connect

Vocabulary

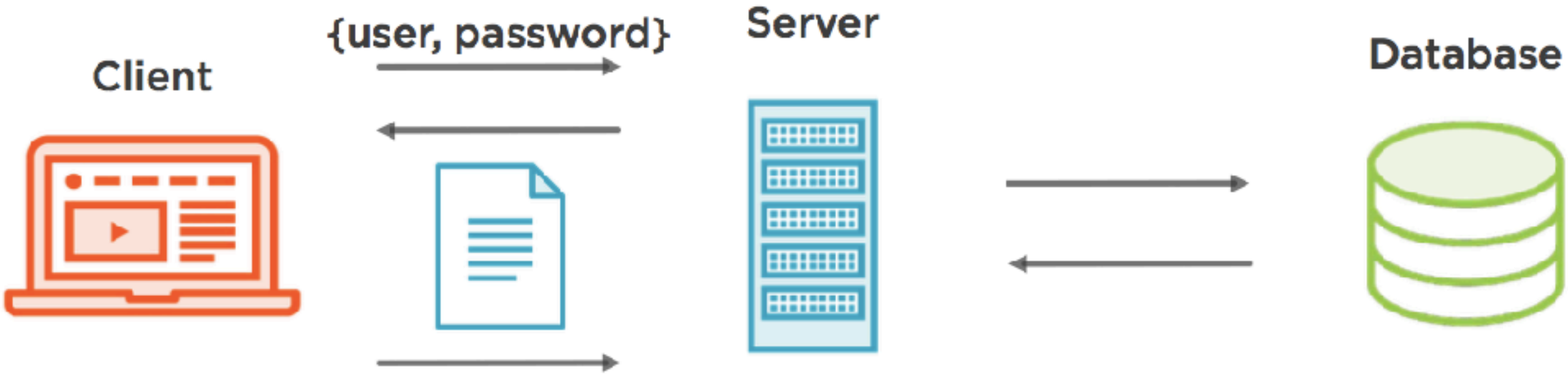
Authentication

Authorization

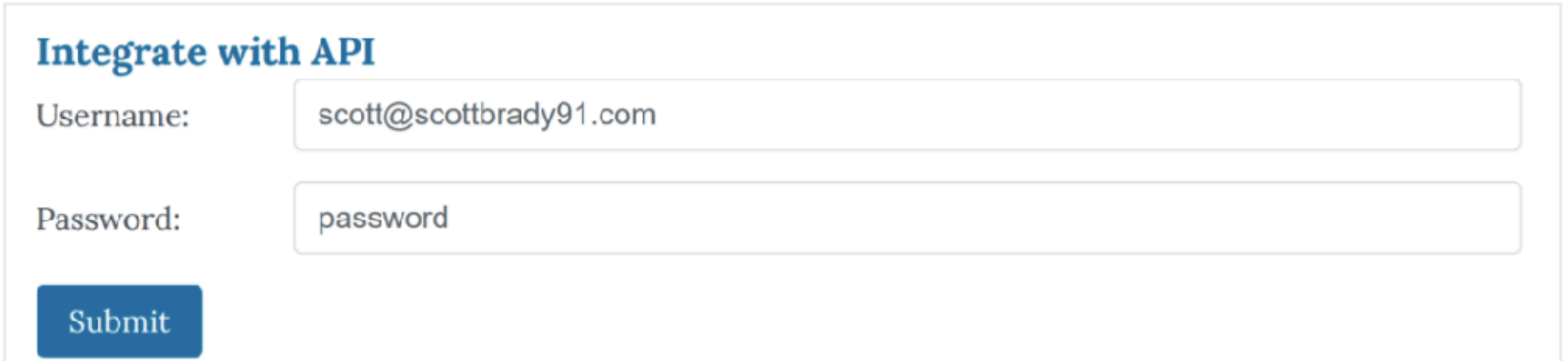
Authentication with Each Request



Single Sign-On



Credential Sharing



Problems

Impersonation

Revocation

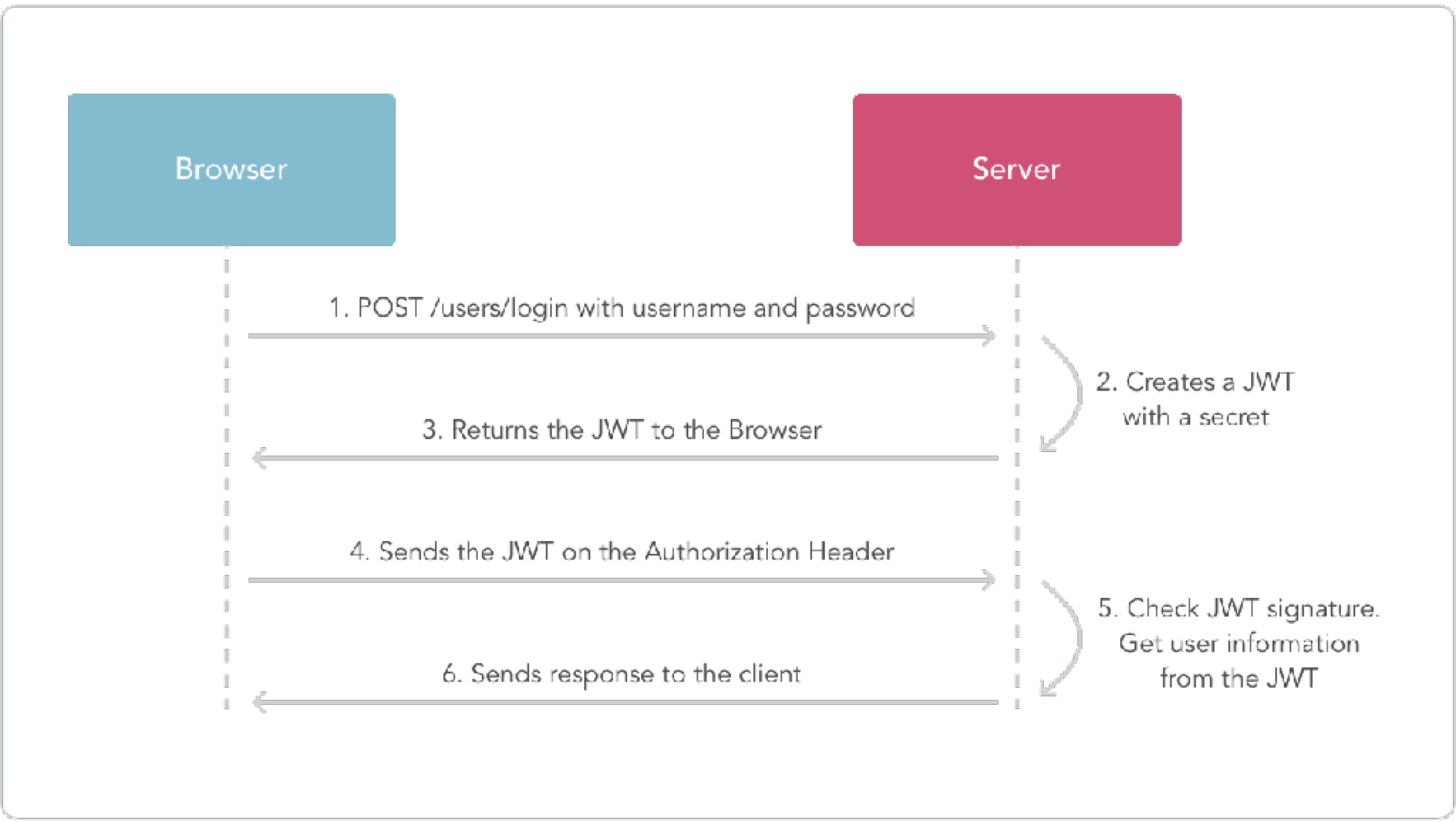
Exposed user credentials

Something you know

Federation

Incompatibilities

JWT


Browser
Server
1. POST /users/login with username and password
2. Creates a JWT with a secret
3. Returns the JWT to the Browser
4. Sends the JWT on the Authorization Header
5. Check JWT signature. Get user information from the JWT
6. Sends response to the client

Problems

Impersonation

Revocation

Exposed user credentials

Something you know

Federation

Incompatibilities

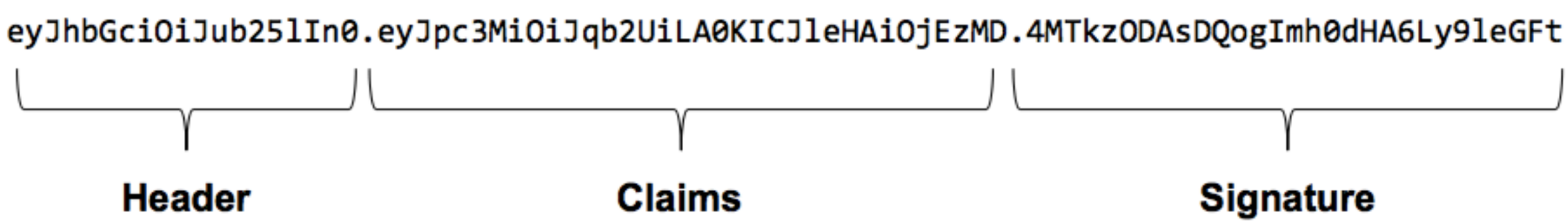
JWT

Header

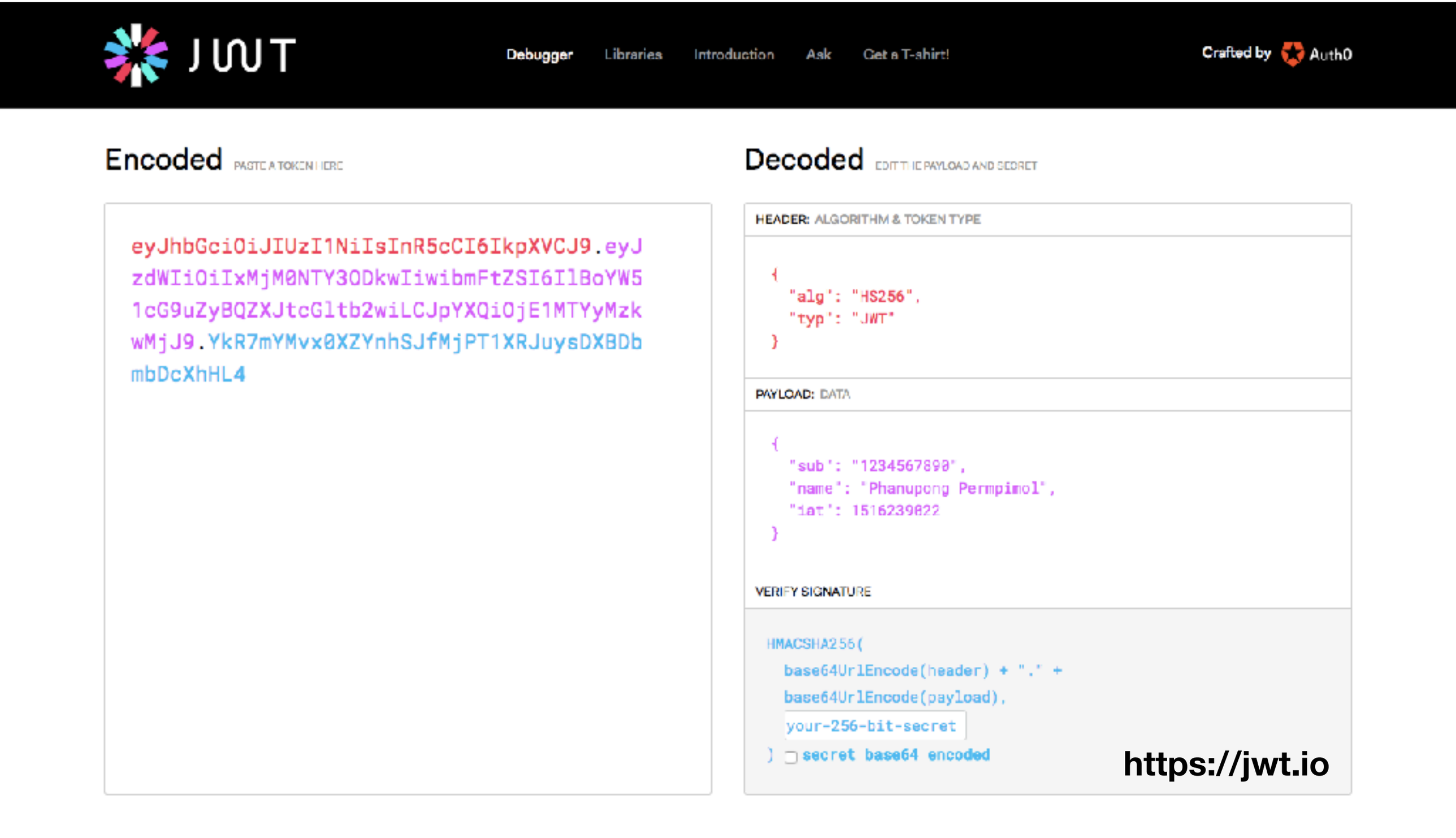
```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

Claims

```
{
  "iss": "http://myIssuer",
  "exp": "1340819380",
  "aud": "http://myResource",
  "sub": "alice",

  "client": "xyz",
  "scope": ["read", "search"]
}
```



JWT


JWT
Debugger
Libraries
Introduction
Ask
Get a T-shirt!
Crafted by Auth0
Encoded
PASTE A TOKEN HERE
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IlBoYW51cG9uZyBQZXJtcGltb2wiLCJpYXQiOjE1MTYyMzkwMjJ9.YkR7mYMvx0XZYnhSJfMjPT1XRJuysDXBDbmbDcXhHL4
Decoded
EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
{
"alg": "HS256",
"typ": "JWT"
}
PAYLOAD: DATA
{
"sub": "1234567890",
"name": "Phanupong Permpimol",
"iat": 1516239022
}
VERIFY SIGNATURE
HMACSHA256(
base64UrlEncode(header) + "." +
base64UrlEncode(payload),
your-256-bit-secret
) secret base64 encoded
https://jwt.io

Limitations

No Standard

Expiration

JWT vs OAuth2

JWT	OAuth2
Authentication Protocol	Authentication Framework
Allows encoded claims to be transferred between two parties	Uses roles, client types & profiles, auth grants and endpoints to manage authentication
Token is issued upon identification	May issue JWT as auth mechanism
Client stores token and sends it with subsequent requests	Error-free implementation is a complex task

Anyways: Use SSL!

PHP OAuth2 Server

OAuth 2.0 Server PHP

Changes

Fork me on GitHub

An OAuth2 Server Library for PHP

Implement an OAuth 2.0 Server cleanly into your PHP application. Download the Code from GitHub to get started.

Requirements

PHP 5.3.9+ is required for this library. However, there is a stable release and development branch for **PHP 5.2.x+** 5.3.8 as well.

Installation

This library follows the zend PSR-0 standards. A number of autoloaders exist which can autoload this library for that reason, but if you are not using one, you can register the `OAuth2\Autoloader`:

```php
require_once('/path/to/oauth2-server-php/src/OAuth2/Autoloader.php');
OAuth2\Autoloader::register();
```

Using Composer? Execute the following command:

```
composer.phar require bshaffer/oauth2-server-php "^1.10"
```

This will add the requirement to the composer.json and install the library.

> It is highly recommended you check out the `v1.10.0` tag to ensure your application doesn't break from backwards-compatibility issues. However, if you'd like to stay on the bleeding edge of development, you can set this to `dev-master` instead.

Get Started With This Library

Looking through the cookbook examples is the best way to get started. For those who just skim the docs for code samples, here is an example of a bare-bones OAuth2 Server implementation:

```php
$storage = new OAuth2\Storage\Pdo(array('dsn' => $dsn, 'username' => $username, 'password' =
$server = new OAuth2\Server($storage);
$server->addGrantType(new OAuth2\GrantType\AuthorizationCode($storage)); // or any grant ty
```

- Overview
- Get Started
- Main Concepts
- Grant Types
- Controllers
- Storage
- Request and Response
- Scope
- User ID
- OpenID Connect
- JWT Access Tokens
- Grant Types
- Controllers
- Storage
- Cookbook

This project is open source. Please help us by forking the project and adding to it

Laravel

Expand All

Prologue

Getting Started

Architecture Concepts

The Basics

Frontend

Security

API Authentication (Passport)

Introduction
Installation
    # Frontend Quickstart
    # Deploying Passport
Configuration
    # Token Lifetimes
Issuing Access Tokens
    # Managing Clients
    # Requesting Tokens
    # Refreshing Tokens
Password Grant Tokens
    # Creating A Password Grant Client
    # Requesting Tokens
    # Requesting All Scopes
Implicit Grant Tokens
Client Credentials Grant Tokens
Personal Access Tokens
    # Creating A Personal Access Client
    # Managing Personal Access Tokens
Protecting Routes
    # Via Middleware

ASP.net

IdentityServer

Home Services Products Training Open Source Customers Contact

identity
SERVER

The Identity and Access Control solution that works for you

We help companies using .NET to build identity and access control solutions for modern applications, including single sign-on, identity management, authorization, and API security.

Based on successful open source projects like IdentityServer, we provide the flexibility to design solutions to meet your requirements.

Spring


spring by Pivotal
PROJECTS
GUIDES
BLOG
Projects
Spring Boot
Spring Framework
Spring Data
Spring Cloud
Spring Cloud Data Flow
Spring Security
Spring Security Kerberos
Spring Security OAuth
Spring Security SAML
Spring Session
Spring Integration
Spring HATEOAS
Spring REST Docs
Spring Batch
Spring IO Platform
Spring Security OAuth
2.3.3
Overview
Learn
Spring Security OAuth provides support for using Spring Security with OAuth (1a) and OAuth2 using standard Spring and Spring Security programming models and configuration idioms.
Features
Support for OAuth providers and OAuth consumers
Oauth 1(a) (including two-legged OAuth, a.k.a. "Signed Fetch")
OAuth 2.0
Applying security to an application is not for the faint of heart, and OAuth is no exception. Before you get started, you're going to want to make sure you understand OAuth and the problem it's designed to address. There is good documentation at the OAuth site. You will also want to make sure you understand how Spring and Spring Security work.
You're going to want to be quite familiar with both OAuth (and/or OAuth2) and Spring Security, to maximize the effectiveness of this developers guide. OAuth for Spring Security is tightly tied to both technologies, so the more familiar you are with them, the more likely you'll be to recognize the terminology and patterns that are used.

Confusion

OAuth != Authentication

Access tokens do not represent the user

The client is not the token's intended audience

Client cannot reliably verify the token

OAuth 2.0

Authorization framework

Built for HTTP APIs

Scoped access

Delegation protocol

Player

Protected Resource (HTTP API)

Client (requesting application)

Resource Owner (the user)

Authorization Server

Authorization Server: User Authentication

WIRED BRAIN
COFFEE

Login

Username

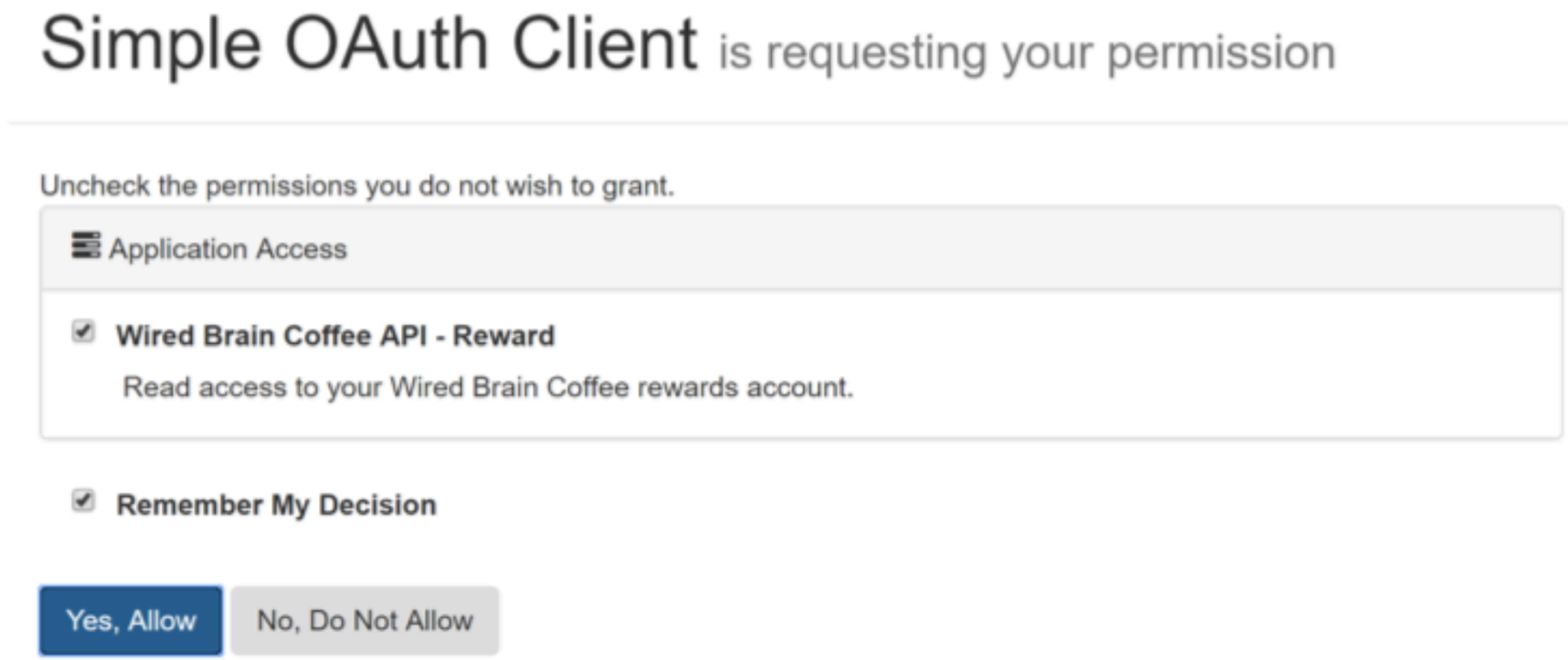
Username

Password

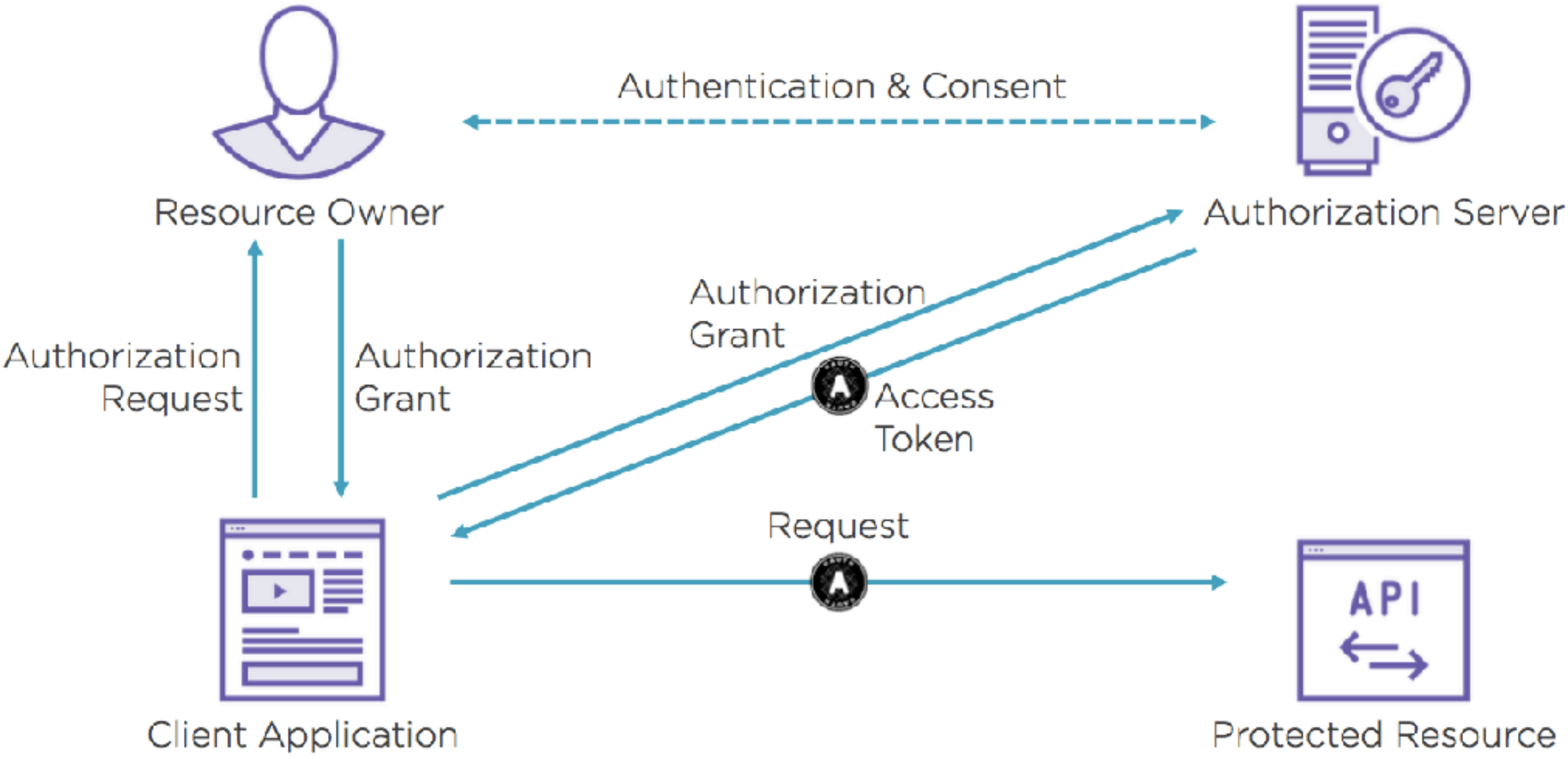
Password

Login

Authorization Server: User Consent



How OAuth works?



Access Token Usage

```
Authorization: Bearer f4ht3qqw80by4d28aa7384oenx2f8pqo
```

OAuth Scope

A permission to do something within a protected resource on behalf of the resource owner

Scopes

customer_api

customer_api.read

customer_api.write

Refresh Token

Swap for new tokens

Allows for long-lived access

Highly confidential

User should be informed

Refresh Token Request

```
POST /token HTTP/1.1
  Host: server.example.com
  Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
 Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token
&refresh_token=tGzv3JOkF0XG5Qx2TlKWIA
&scope=api1
```

Refresh Token Response


```
{
    "access_token": "5BaFFasdfrje2aCsicoiJefn",
    "token_type": "Bearer",
    "expires_in": 3600,
    "refresh_token": "9oih321s0i2xnoutnsglr",
    "scope": "api2.read offline_access"
}
```

Grant Type

- Authorization code
- Implicit
- Client Credential
- Resource Owner Password Credentials (ROPC)

Authorization Code

- Designed for “confidential clients”
- Best for websites with a server back end
- Explicit user & client authentication

Authorization Request

```
https://authserver.example.com/authorize
    ?response_type=code
    &client_id=s6BhdRkqt3
    &redirect_uri=https://client.example.com/callback
    &state=xyz
    &scope=api1 api2.read
```

Authorization Response

```
https://client.example.com/callback
?code=SplxlOBeZQQYbYS6WxSbIA
&state=xyz
```

Token Request

```
POST /token HTTP/1.1
  Host: server.example.com
  Content-Type: application/x-www-form-urlencoded
   Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW


grant_type=authorization_code
&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https://client.example.com/cb
&client_id=s6BhdRkqt3&client_secret=gX1fBat3bV
```

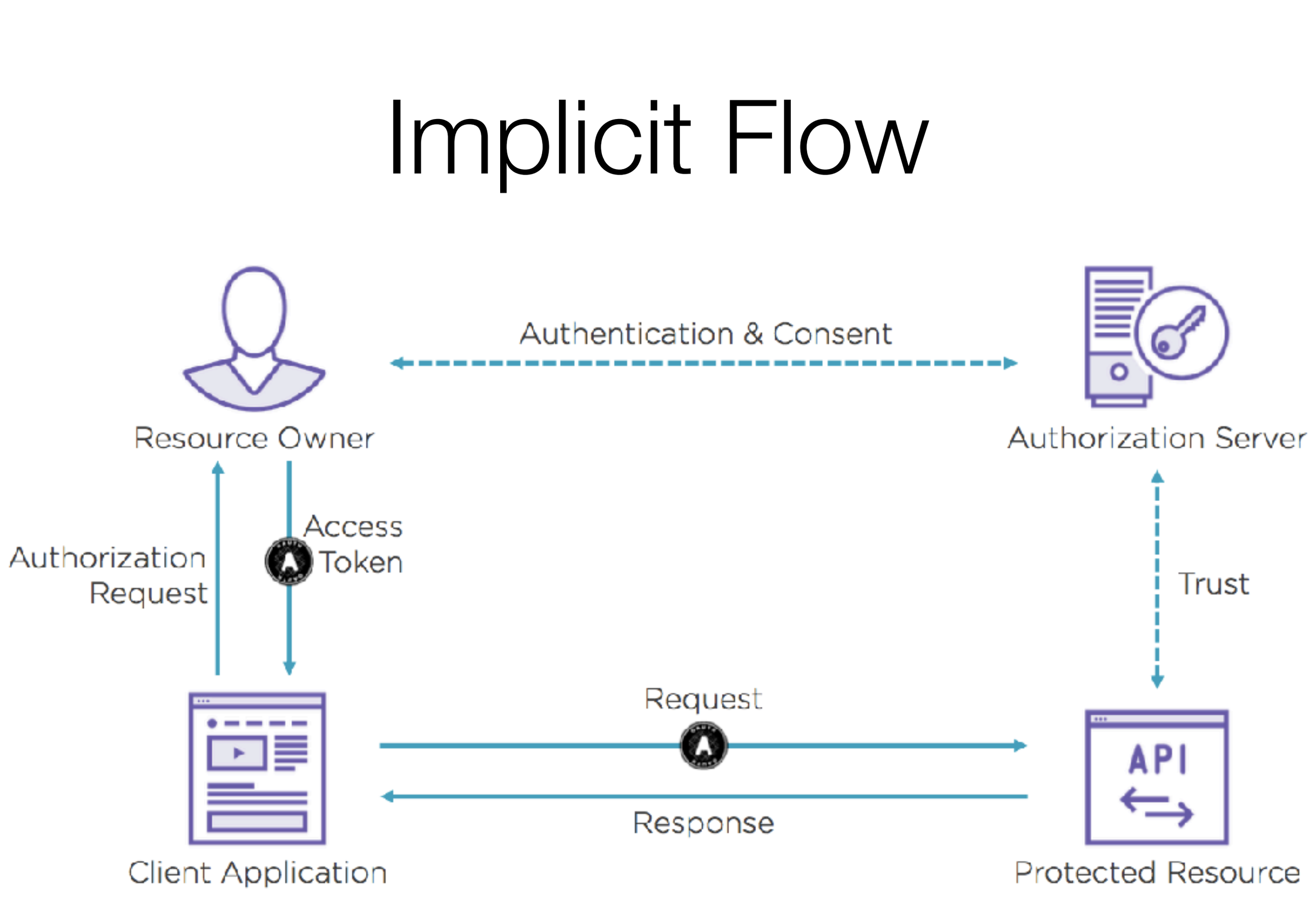
Token Response


```
{
    "access_token": "2YotnFZFEjr1zCsicMWpAA",
    "token_type": "Bearer",
    "expires_in": 3600,
    "scope": "api2.read"
}
```

Implicit

- Designed for “public clients”
- Best for clients accessing resources directly from the browser
- No explicit client authentication

Implicit Flow



Authorization Request

```
https://authserver.example.com/authorize
    ?response_type=token
    &client_id=s6BhdRkqt3
    &redirect_uri=https://client.example.com/callback
    &state=xyz
    &scope=api1 api2.read
```

Authorization Response

```
https://client.example.com/callback

    #access_token=2YotnFZFEjr1zCsicMWpAA
    &token_type=example
    &expires_in=3600
    &state=xyz
```

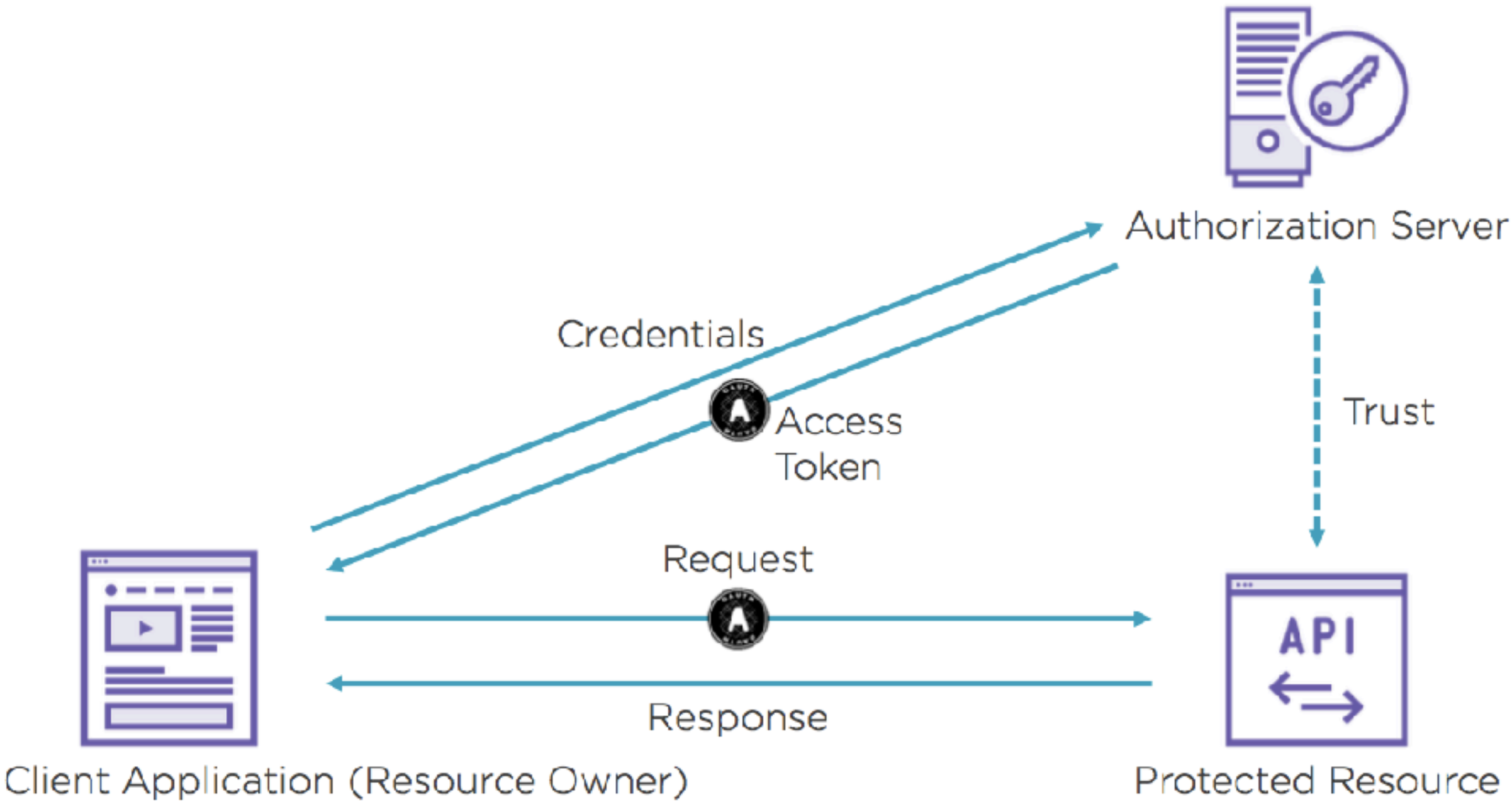
Security Concerns

- Access tokens exposed to resource owner
- Access tokens accessible to third party JavaScript
- No validation that access tokens are intended for client

Client Credential

- Designed for client applications who are the resource owner
- Best for machine-to-machine communication
- Requires client authentication

Client Credential Flow



Token Request

```
POST /token HTTP/1.1
   Host: server.example.com
   Content-Type: application/x-www-form-urlencoded
   Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=client_credentials
&scope=api1 api2.read
```

Token Response


```
{
    "access_token": "2YotnFZFEjr1zCsicMWpAA",
    "token_type": "Bearer",
    "expires_in": 3600,
    "scope": "api2.read"
}
```

API Key vs Basic AuthN

- Not sending the credentials with every request
- Access tokens
- Timed access without manual input

ROPC

- Designed as a stop-gap for legacy applications
- Negates most of the benefits of OAuth
- Should no longer be used

Token Request

```
POST /token HTTP/1.1
  Host: server.example.com
  Content-Type: application/x-www-form-urlencoded
  Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=password
&username=johndoe
&password=A3ddj3w
&scope=api1 api2.read
```

Token Response

```
{
    "access_token": "2YotnFZFEjr1zCsicMWpAA",
    "token_type": "Bearer",
    "expires_in": 3600,
    “scope": "api2.read"
}
```

Do not use this grant type

Best Practice for Public Clients

Why Can’t We Just Use the Implicit Flow?

- Reliance upon redirect URI
- No web server
- We aren’t the only ones listening...
- Refresh tokens
- Codes or tokens - both at risk

The implicit flow should not be used for native applications

PCKE


Authorization Request
2. Include code_challenge
Authorization Code
Token Request
4. Include code_verifier
Access Token
Native Client
1. Generate code_verifier
Authorization Server
3. Store code_challenge
5. Compare t(code_verifier) and code_challenge

BEST CURRENT PRACTICE

Internet Engineering Task Force (IETF)
Request for Comments: 8252
BCP: 212
Updates: 6749
Category: Best Current Practice
ISSN: 2070-1721

W. Denniss
Google
J. Bradley
Ping Identity

October 2017

RFC 8252
(OAuth for Native Apps)

OAuth 2.0 for Native Apps

Abstract

OAuth 2.0 authorization requests from native apps should only be made through external user-agents, primarily the user's browser. This specification details the security and usability reasons why this is the case and how native apps and authorization servers can implement this best practice.

Status of This Memo

This memo documents an Internet Best Current Practice.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on BCPs is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8252.

https://tools.ietf.org/html/rfc8252

Error Types

- invalid_request
- invalid_client (401)
- invalid_grant unauthorized_client
- unsupported_grant_type
- invalid_scope

Response Modes

Query String

Hash Fragment

Form Post

Form Post Mode

```
<html>
   <head><title>Submit This Form</title></head>
   <body onload="javascript:document.forms[0].submit()">
    <form method="post" action="https://client.example.org/callback">
      <input type="hidden" name="state" value="xyz"/>
      <input type="hidden" name="code" value="SplxlOBeZQQYbYS6WxSbIA"/>
    </form>
  </body>
</html>
```

AppAuth

AppAuth

Native App SDK for OAuth 2.0 and OpenID Connect implementing modern best practices

AppAuth for Android

AppAuth for iOS and macOS

AppAuth for JS

AppAuth is a client SDK for native apps to authenticate and authorize end-users using OAuth 2.0 and OpenID Connect. Available for iOS, macOS, Android and Native JS environments, it implements modern security and usability best practices for native app authentication and authorization.

It strives to directly map the requests and responses of those specifications, while following the idiomatic style of the implementation language. In addition to mapping the raw protocol flows, convenience methods are available to assist with common tasks like performing an action with fresh tokens.

It follows the best practices set out in RFC 8252 - OAuth 2.0 for Native Apps including using in-app browser tabs (like SFAuthenticationSession and Android Custom Tabs) where available. Embedded user-agents (known as web-views) are explicitly *not* supported due to the usability and security reasons documented in Section 8.12 of RFC 8252.

It also supports the PKCE extension to OAuth which was created to secure authorization codes in public clients when custom URI scheme redirects are used. The library is friendly to other extensions (standard or otherwise) with the ability to handle additional params in all protocol requests and responses.

https://appauth.io

Laravel

AuthN and AuthZ

Authentication

- The authentication configuration file is located at config/auth.php

- Laravel ships with several pre-built authentication controllers, which are located in the App\Http\Controllers\Auth namespace. The RegisterController handles new user registration, the LoginController handles authentication, the ForgotPasswordController handles e-mailing links for resetting passwords, and the ResetPasswordController contains the logic to reset passwords. Each of these controllers uses a trait to include their necessary methods. For many applications, you will not need to modify these controllers at all

Routing

- Laravel provides a quick way to scaffold all of the routes and views you need for authentication using one simple command.

```
php artisan make:auth
```

Views

- The php artisan make:auth command will create all of the views you need for authentication and place them in the resources/views/auth directory.
- The make:auth command will also create a resources/views/layouts directory containing a base layout for your application.

Path Customization

- When a user is successfully authenticated, they will be redirected to the /home URI. You can customize the post-authentication redirect location by defining a redirectTo property on the LoginController, RegisterController, and ResetPasswordController.

```
protected $redirectTo = '/';
```

Authenticated User

- You may access the authenticated user via the Auth facade.

```php
use Illuminate\Support\Facades\Auth;


$user = Auth::user();
```

Authenticated User

- Determining If The Current User Is Authenticated

```php
use Illuminate\Support\Facades\Auth;

if (Auth::check()) {
    // The user is logged in...
}
```

Authenticated User

- Manually Authenticating Users

```php
public function authenticate()
{
    if (Auth::attempt(['email' => $email, 'password' => $password])) {
        // Authentication passed...
        return redirect()->intended('dashboard');
    }
}
```

Authenticated User

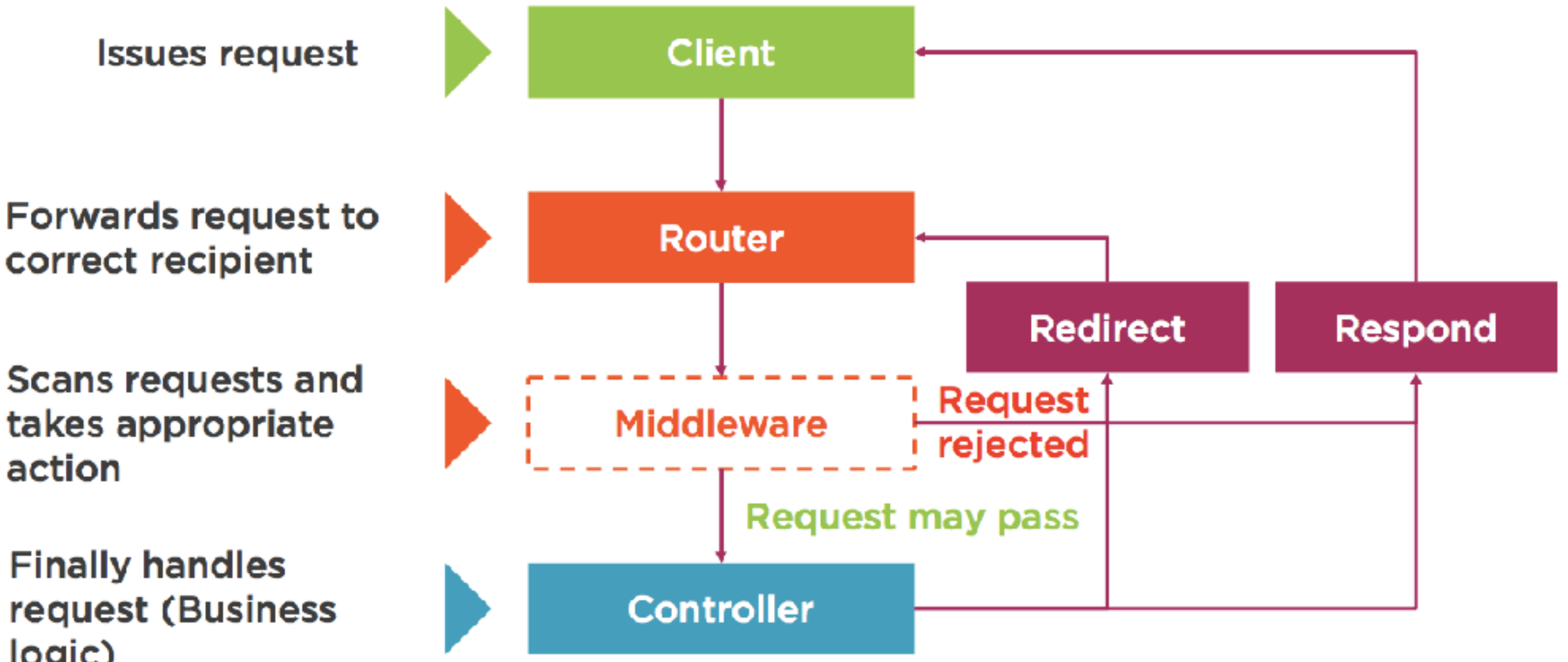
- Specifying Additional Conditions

```php
if (Auth::attempt(['email' => $email, 'password' => $password, 'active' => 1])) {
    // The user is active, not suspended, and exists.
}
```

- Logging Out

```php
Auth::logout();
```

Request & Response Flow



Controller Middleware

```php
// app/Http/Controllers/PostController.php

public function __construct()
{
    $this->middleware('auth');
    $this->middleware('log', [
        'only' => ['store', ‘update']
    ]);
}
```

Route Middleware

```php
Route::get('/', function () {
    //
})->middleware('web');

Route::group(['middleware' => ['web']], function () {
    //
});
```

Laravel Passport

Installation

1. `composer require laravel/passport`
2. `php artisan migrate`
3. `php artisan passport:install`

Add Trait to Model

```php
<?php

namespace App;

use Laravel\Passport\HasApiTokens;
use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use HasApiTokens, Notifiable;
}
```

Register Passport Routes

AuthServiceProvider

```
public function boot()
{
    $this->registerPolicies();

    Passport::routes();
}
```

Change Driver

config/auth.php

```
'guards' => [
    'web' => [
        'driver' => 'session',
        'provider' => 'users',
    ],

    'api' => [
        'driver' => 'passport',
        'provider' => 'users',
    ],
],
```

Configuration

```php
public function boot()
{
    $this->registerPolicies();

    Passport::routes();

    Passport::tokensExpireIn(now()->addDays(15));

    Passport::refreshTokensExpireIn(now()->addDays(30));
}
```

Grant Tokens

New Middleware

```
use Laravel\Passport\Http\Middleware\CheckClientCredentials;
```

```
protected $routeMiddleware = [
    'client' => CheckClientCredentials::class,
];
```

Grant Token

Retrieve token from oauth/token

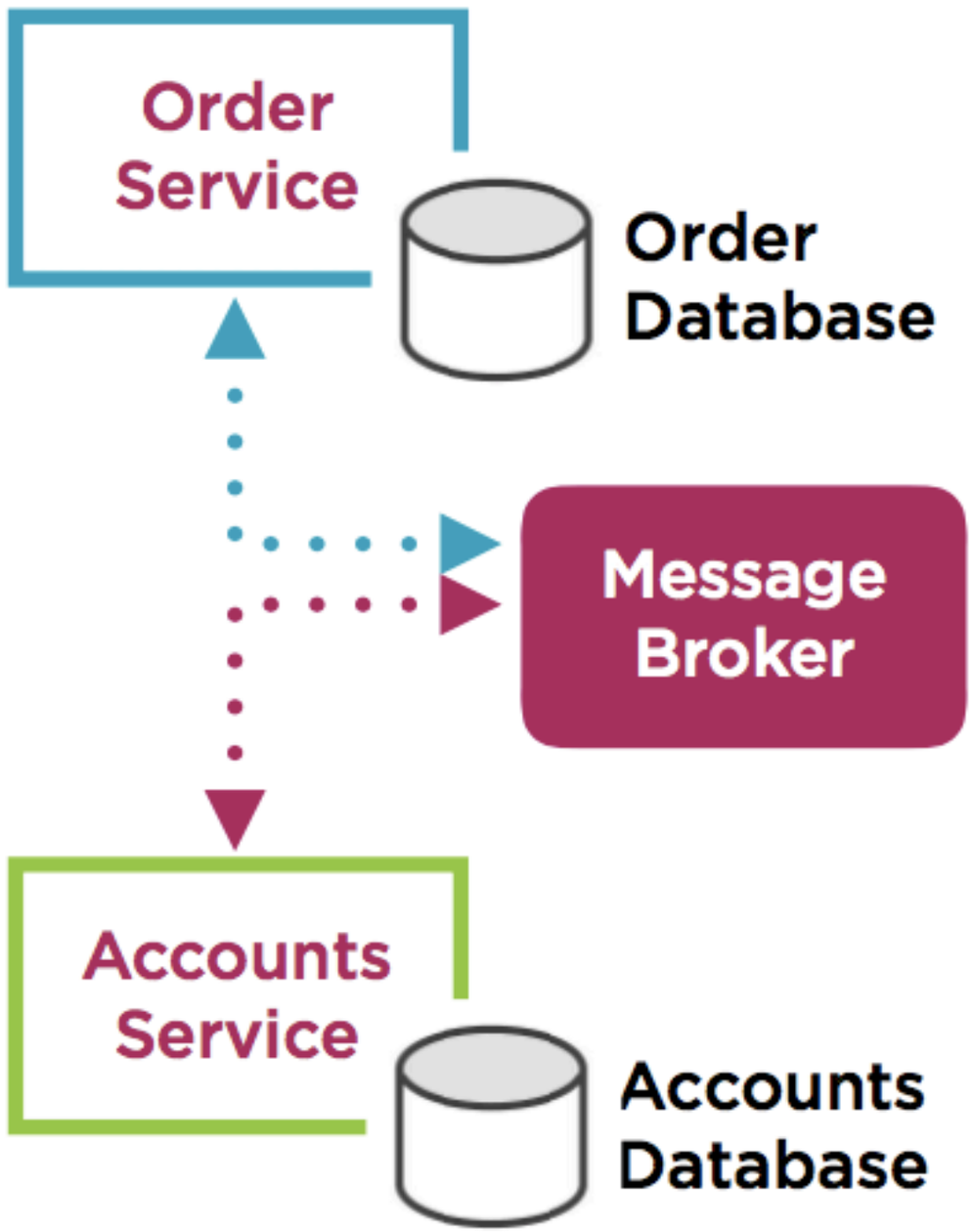
```php
$guzzle = new GuzzleHttp\Client;

$response = $guzzle->post('http://your-app.com/oauth/token', [
    'form_params' => [
        'grant_type' => 'client_credentials',
        'client_id' => 'client-id',
        'client_secret' => 'client-secret',
        'scope' => 'your-scope',
    ],
]);

return json_decode((string) $response->getBody(), true)['access_token'];
```

Messaging

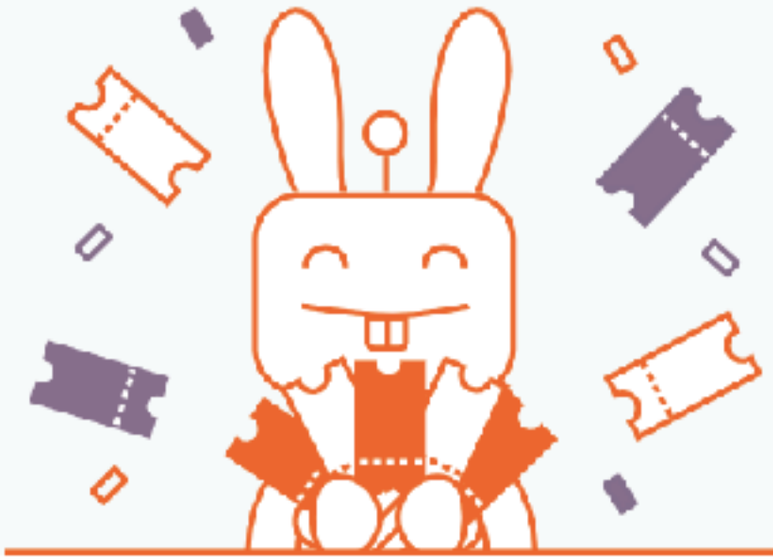
Event Based

- Transaction/action as an event
- Messages using message brokers
- Decouples client and service
- Queuing Pattern

Async API Call



RabbitMQ



RabbitMQ Summit 2018
Nov 12 | London, UK

Tickets on Sale through November 10

RabbitMQ is the most widely deployed open source message broker.

With more than 35,000 production deployments of RabbitMQ world-wide at small startups and large enterprises, RabbitMQ is the most popular open source message broker.

Updates

1. RabbitMQ 3.7.8 — 20 Sep 2018
2. RabbitMQ 3.7.7 — 05 Jul 2018
3. RabbitMQ 3.7.6 — 13 Jun 2018

More updates →

Tweets

RabbitMQ

https://www.rabbitmq.com/

Asynchronous Messaging

Supports multiple messaging protocols, message queuing, delivery acknowledgement, flexible routing to queues, multiple exchange type.

Developer Experience

Deploy with BOSH, Chef, Docker and Puppet. Develop cross-language messaging with favorite programming languages such as: Java, .NET, PHP, Python, JavaScript, Ruby, Go, and many others.

Distributed Deployment

Deploy as clusters for high availability and throughput; federate across multiple availability zones and regions.

Enterprise & Cloud Ready

Pluggable authentication, authorization, supports TLS and LDAP. Lightweight and easy to deploy in public and private clouds.

Tools & Plugins

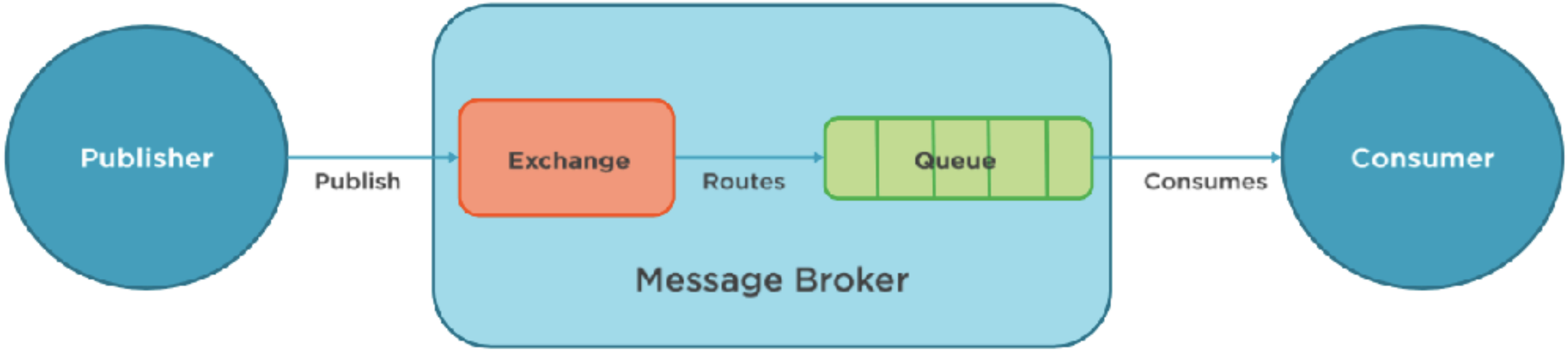
Diverse array of tools and plugins supporting continuous integration, operational metrics, and integration to other enterprise systems. Flexible plug-in approach for extending RabbitMQ functionality.

Management & Monitoring

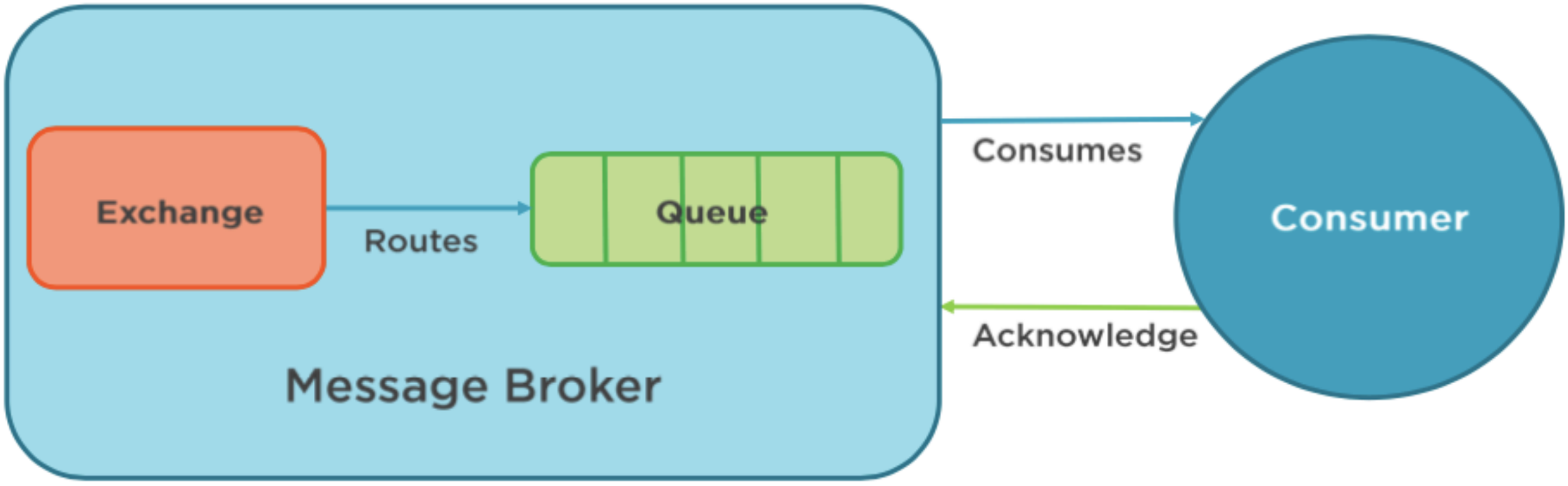
HTTP-API, command line tool, and UI for managing and monitoring RabbitMQ.

AMQP Messaging Standard

- Advance Messaging Queuing Protocol (AMQP)
- RabbitMQ Supports version 0-9-1



AMQP Messaging Standard



Exchanges

Direct Exchanges

Fanout Exchanges

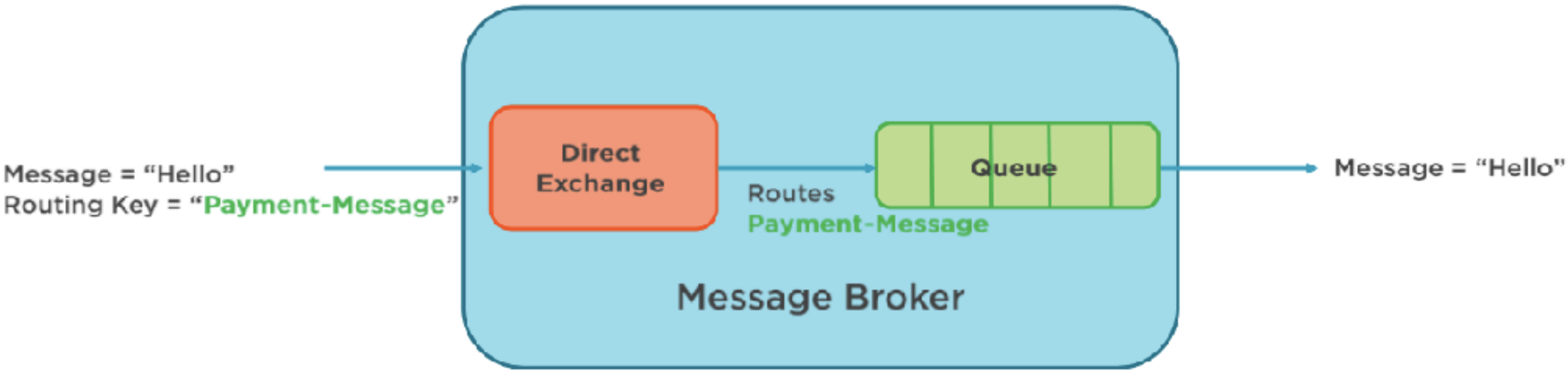
Topic Exchanges

Header Exchanges

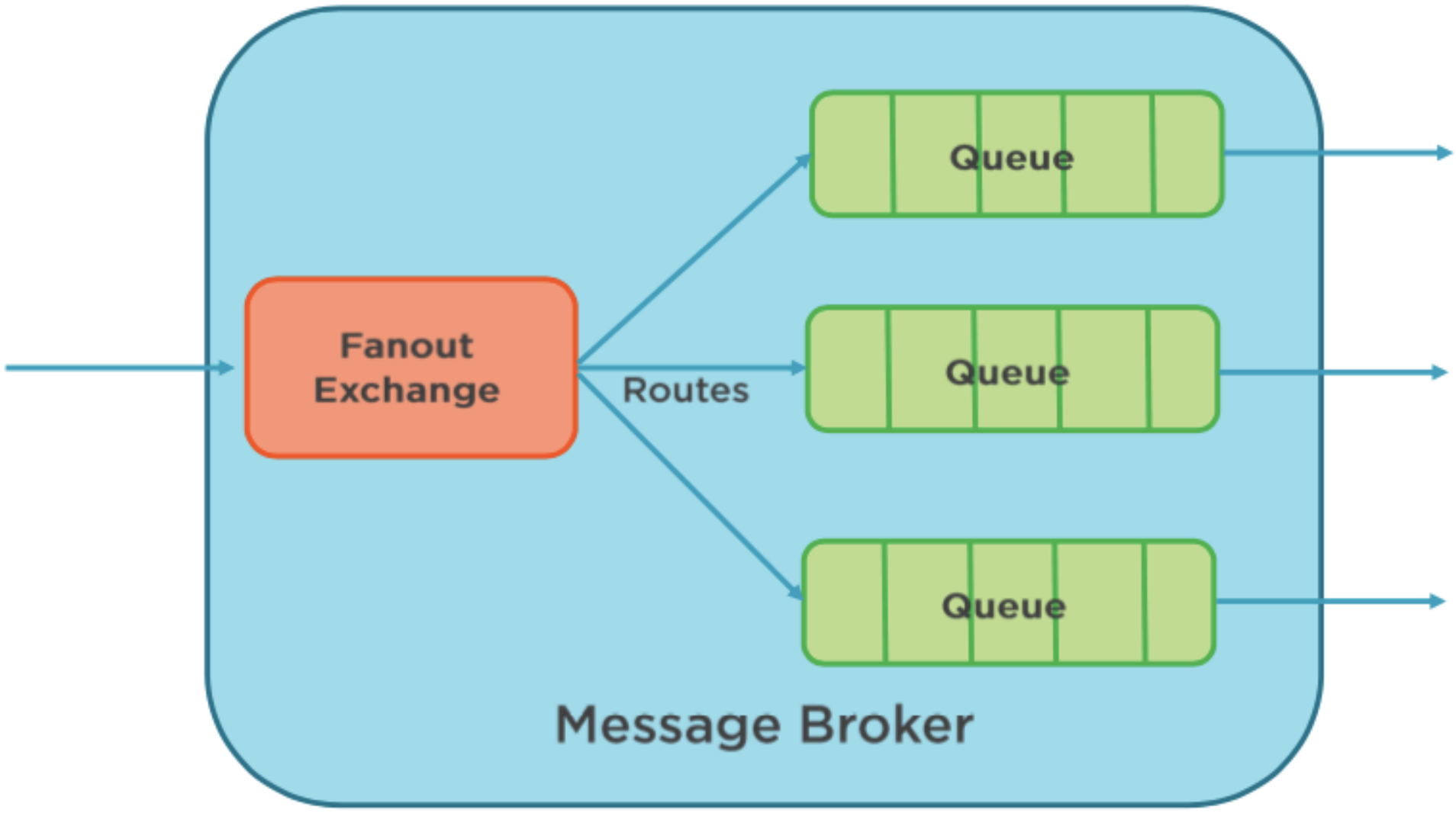
Exchange

Name	*The name of the exchange*
Durability	*Persisting the messages to disk*
Auto-Delete	*Delete message when not needed*
Arguments	*These are message broker-dependent*

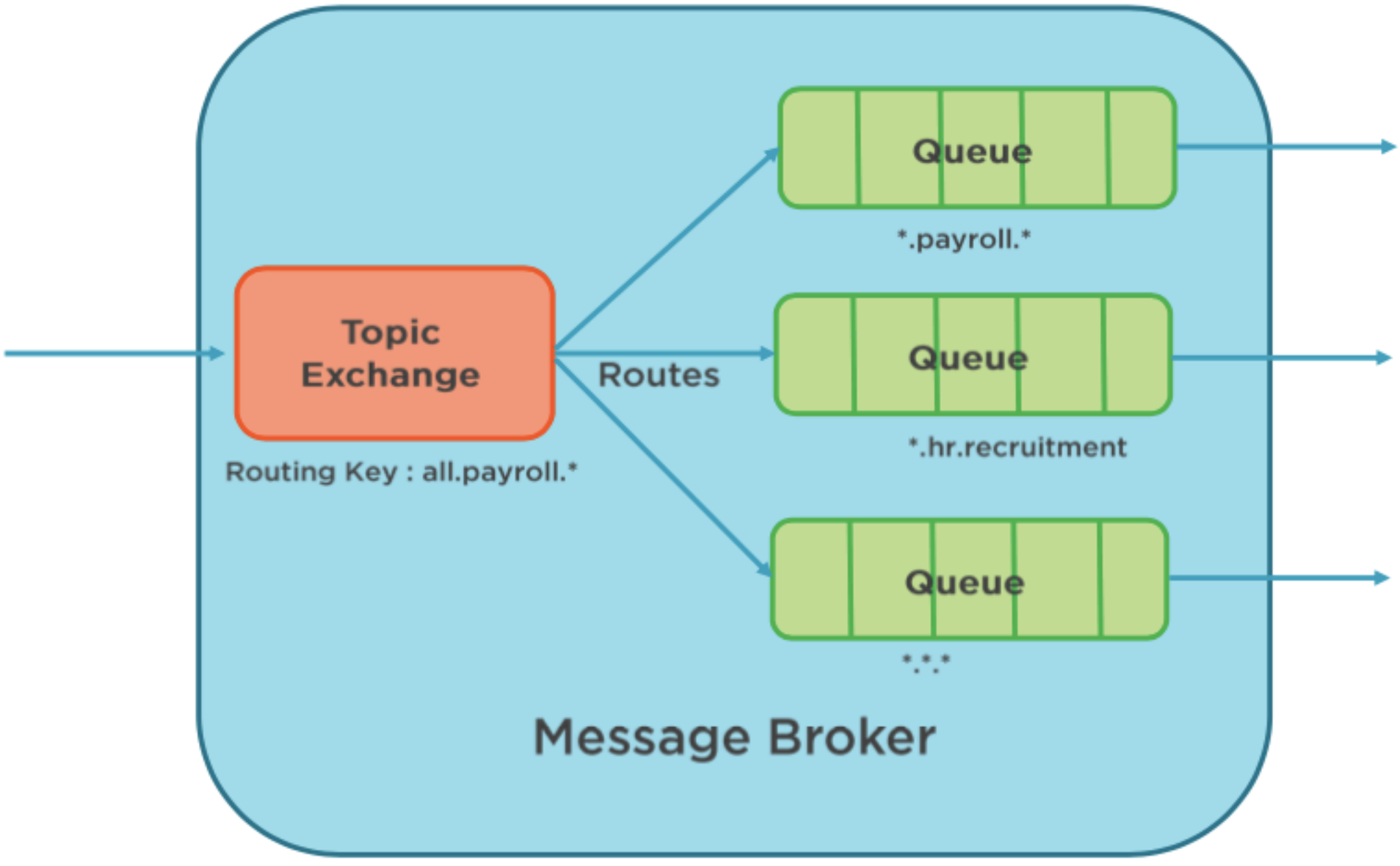
Direct Exchange



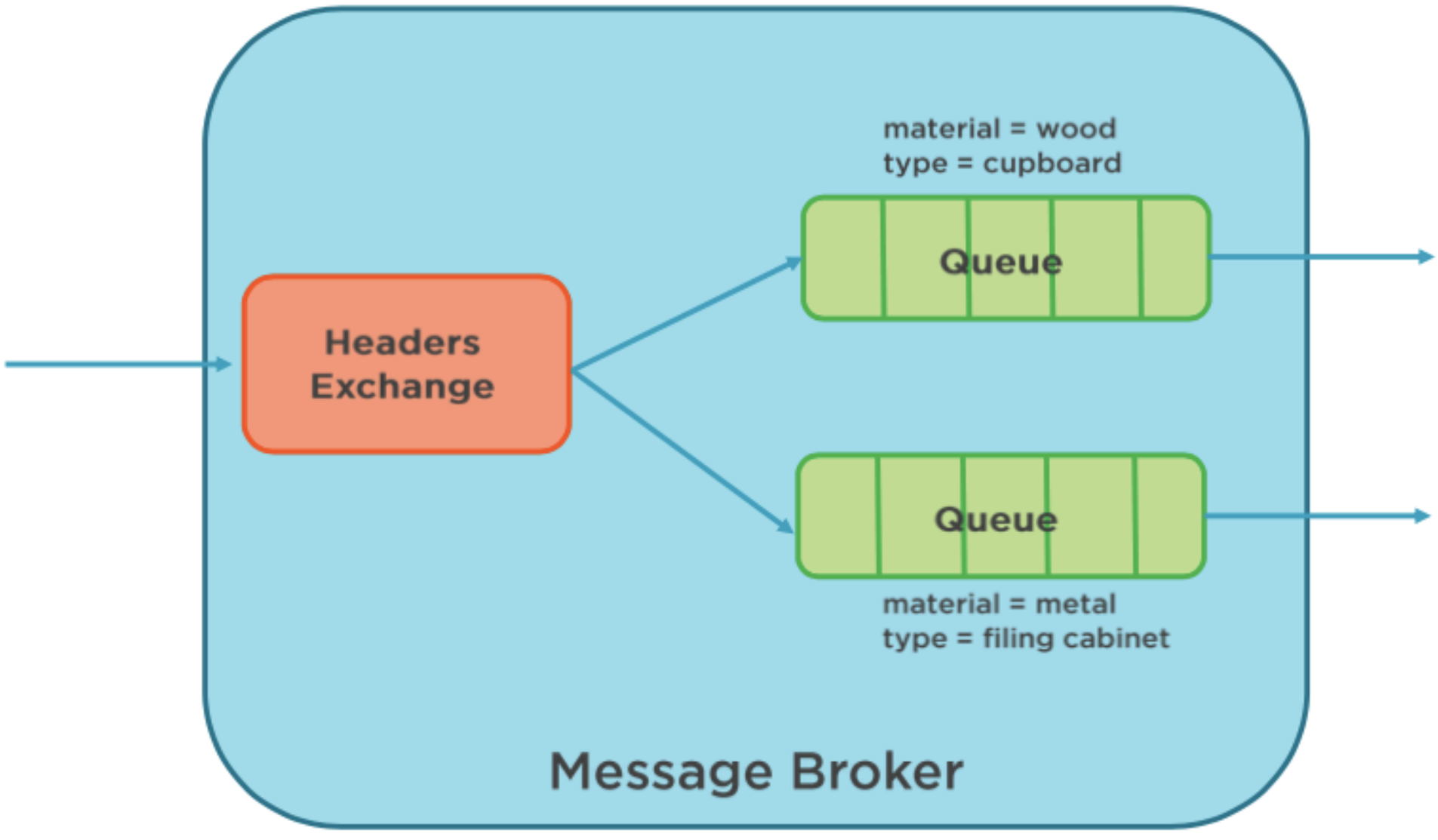
Fanout Exchange



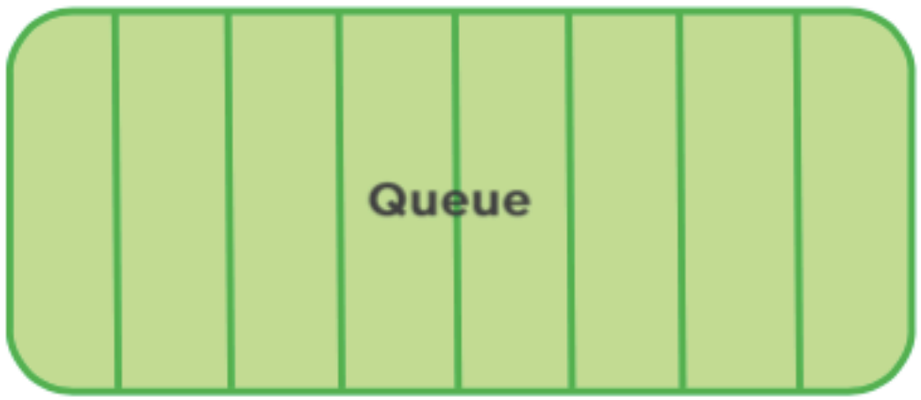
Topic Exchange



Header Exchange

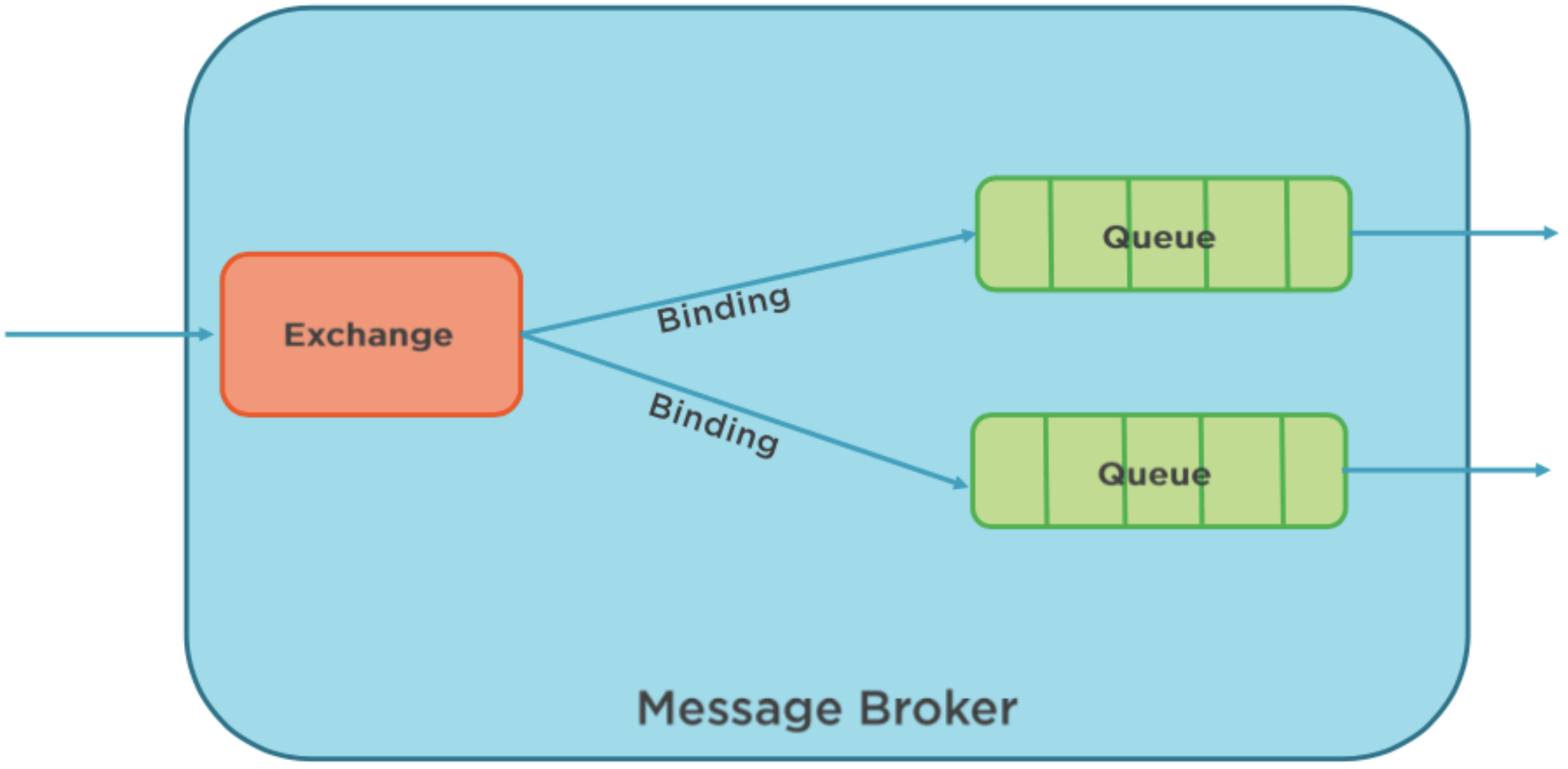


Queues

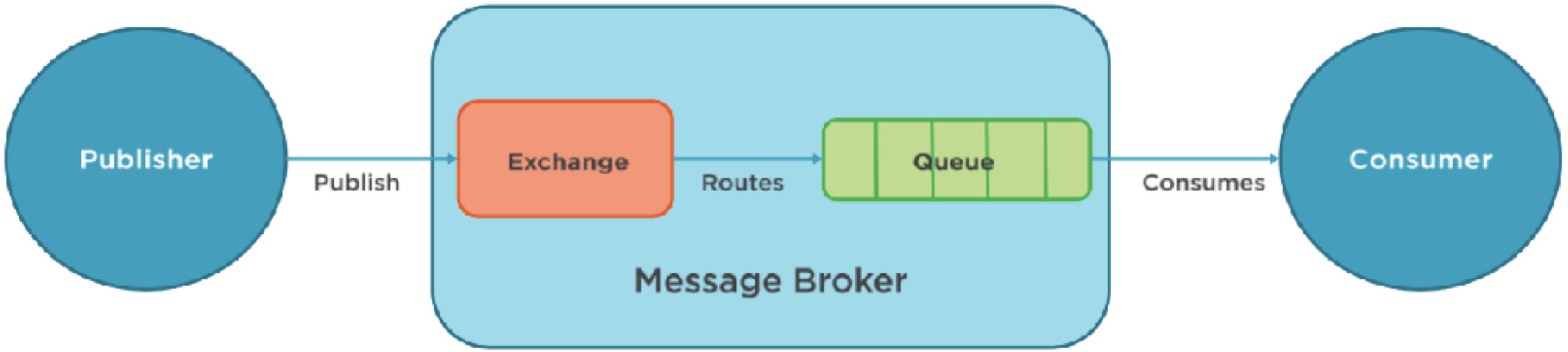


Name	*The name of the queue*
Durable	*Persisting the queue to disk*
Exclusive	*Delete queue when not needed*
Auto Delete	*Queue deleted when consumer unsubscribes*

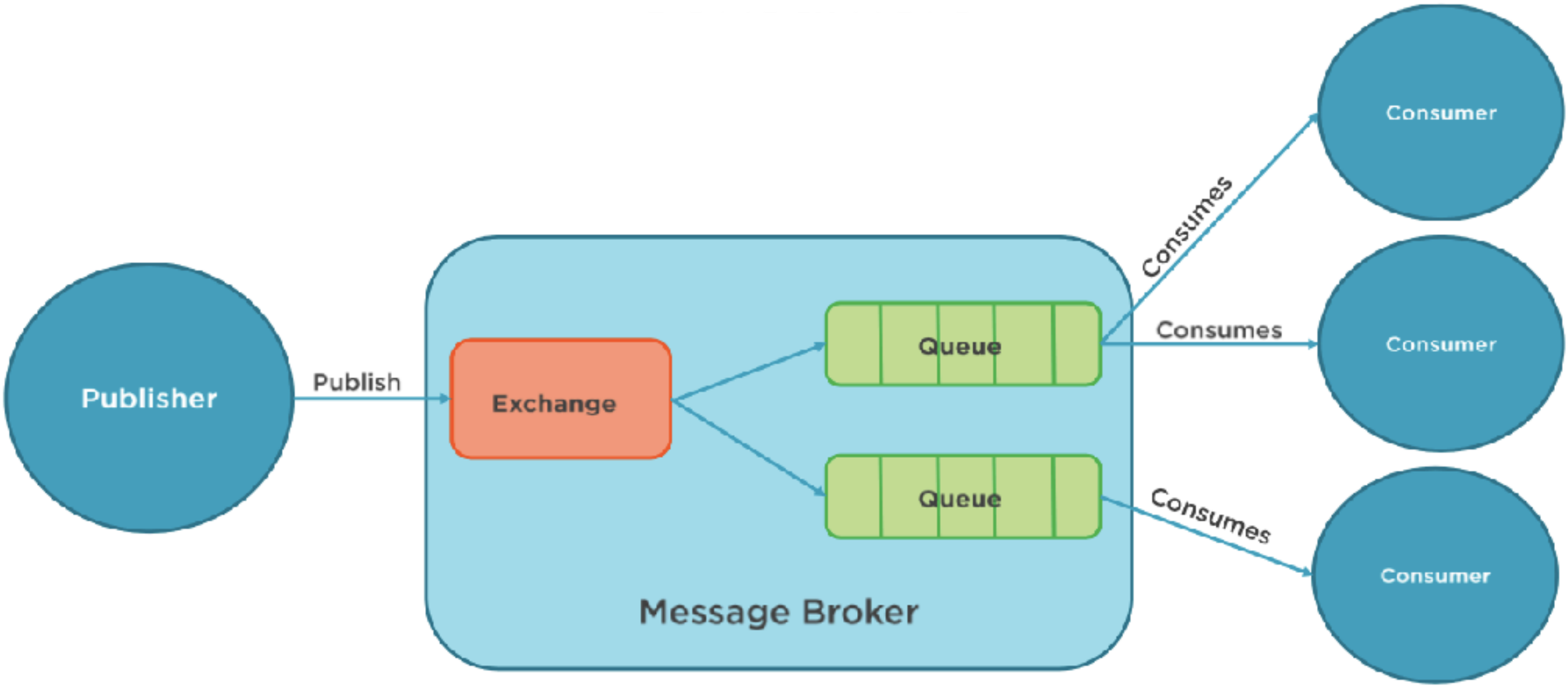
Bindings



Consumers


Publisher
Publish
Exchange
Routes
Queue
Message Broker
Consumes
Consumer

Consumers


Publisher
Publish
Exchange
Queue
Queue
Message Broker
Consumes
Consumes
Consumes
Consumer
Consumer
Consumer

Apache Kafka

HOME

INTRODUCTION

QUICKSTART

USE CASES

DOCUMENTATION

PERFORMANCE

POWERED BY

PROJECT INFO

ECOSYSTEM

CLIENTS

EVENTS

CONTACT US

APACHE

Download

@apachekafka

PUBLISH & SUBSCRIBE

Read and write streams of data like a messaging system.

Learn more »

PROCESS

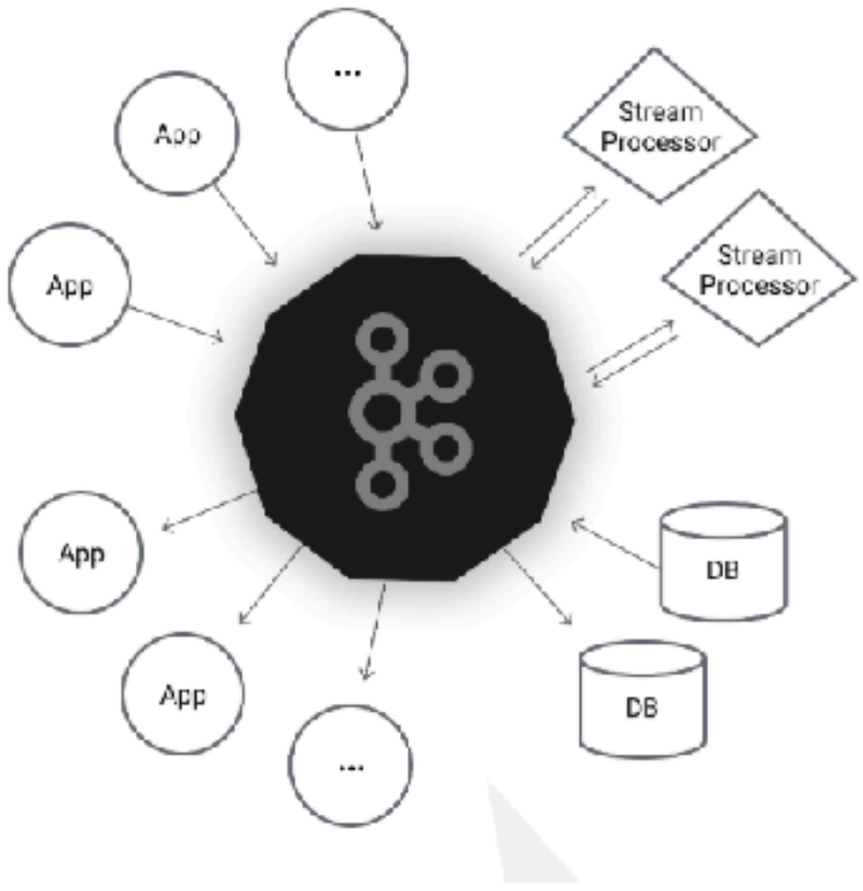
Write scalable stream processing applications that react to events in real-time.

Learn more »

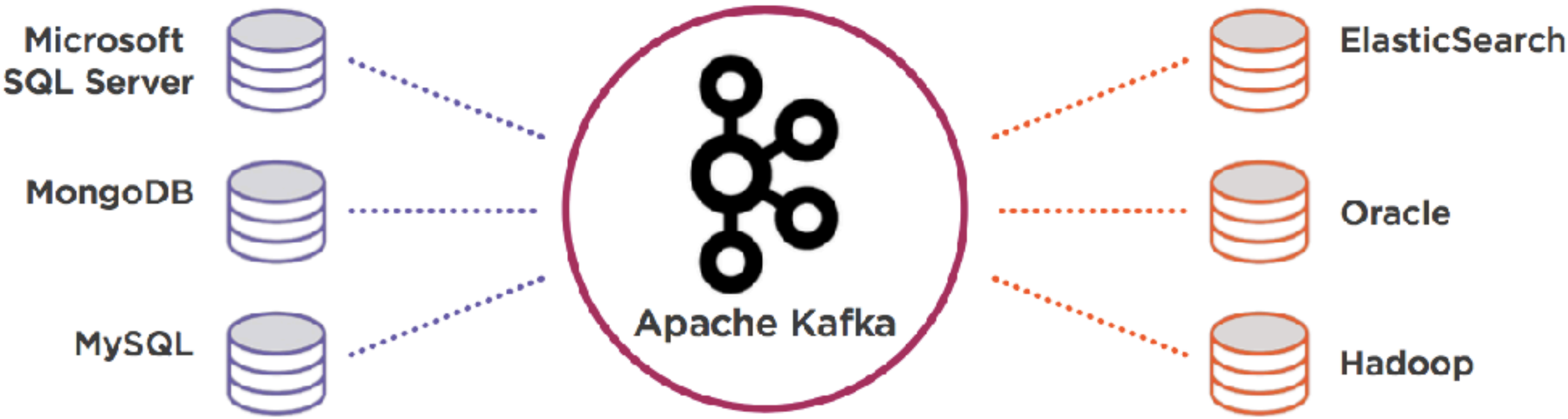
STORE

Store streams of data safely in a distributed, replicated, fault-tolerant cluster.

Learn more »

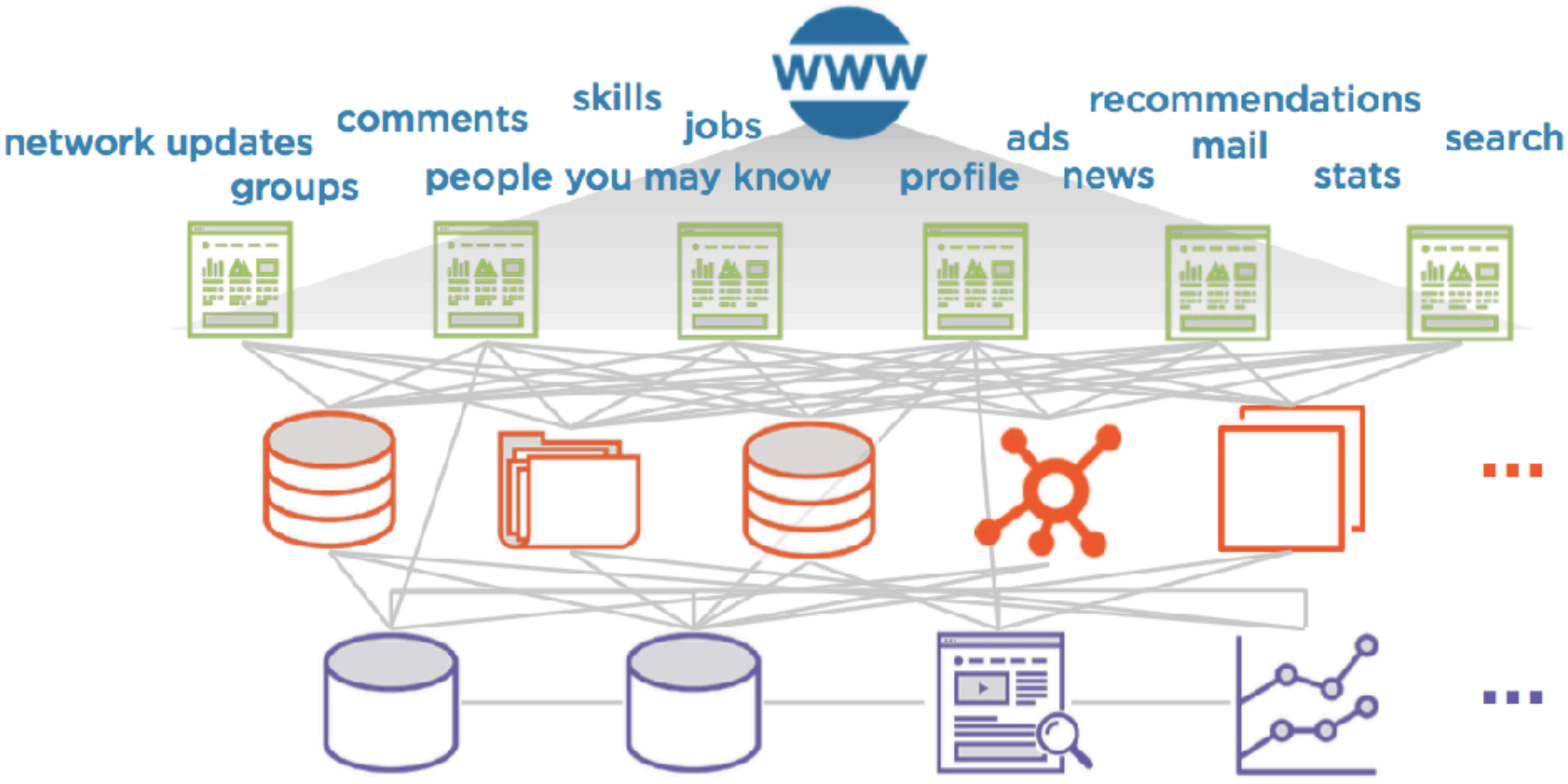


What is Apache Kafka

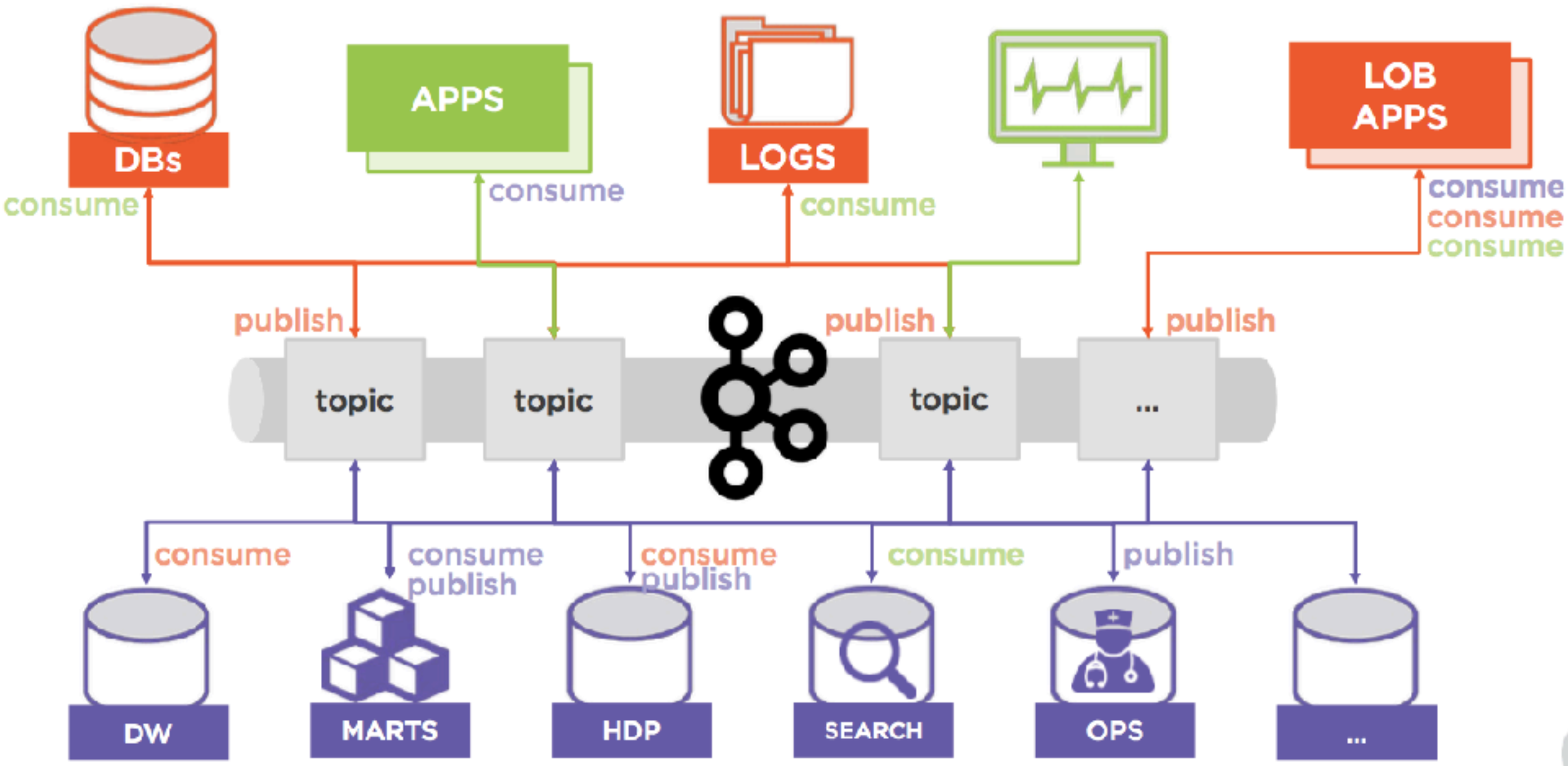


"A high-throughput distributed messaging system."

Pre 2010


www
network updates
comments
skills
jobs
recommendations
ads
mail
search
groups
people you may know
profile
news
stats
...
...

Post 2010


DBs
APPS
LOGS
LOB APPS
consume
consume
consume
consume
consume
consume
publish
publish
publish
topic
topic
topic
...
consume
consume
publish
consume
publish
consume
publish
DW
MARTS
HDP
SEARCH
OPS
...

Timeline of Kafka

2003
LinkedIn Launch

2009
Kafka Inception
Development begins

2010
Initial Kafka Deployment @ LinkedIn

2011
Kafka Open Sourced
Apache Software Foundation

Today
1.1 Trillion messages per day @ LinkedIn

Q&A