



RED HAT 3Scale

API Management Workshop

March, 2021

Tommer Amber, Senior Cloud Security Consultant
Red Hat Professional Services, Israel

Agenda

- ▶ APIs - Background & Types of Web APIs
- ▶ API Management Lifecycle
- ▶ 3Scale - Overview
- ▶ Day Two Deployment Options
- ▶ API Security
- ▶ Automation

BACKGROUND

WHAT IS AN API?

Application Programming Interface



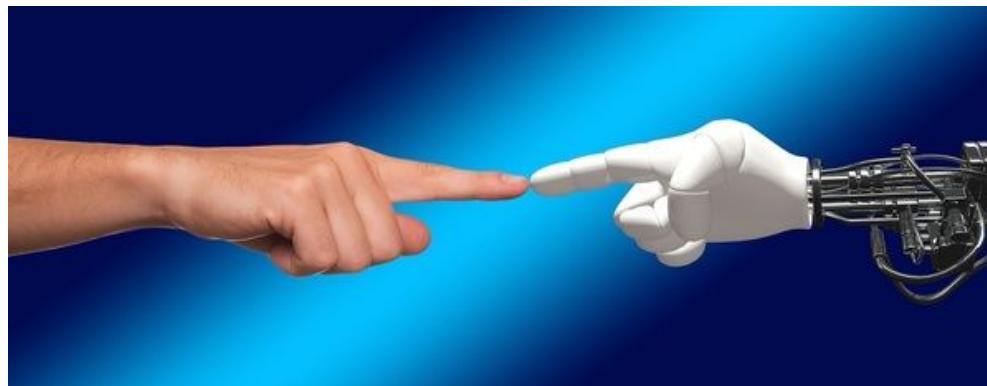
WHAT IS AN API?

"Software component or a system, externally exposed, that **defines** **how** other components or systems can use it"

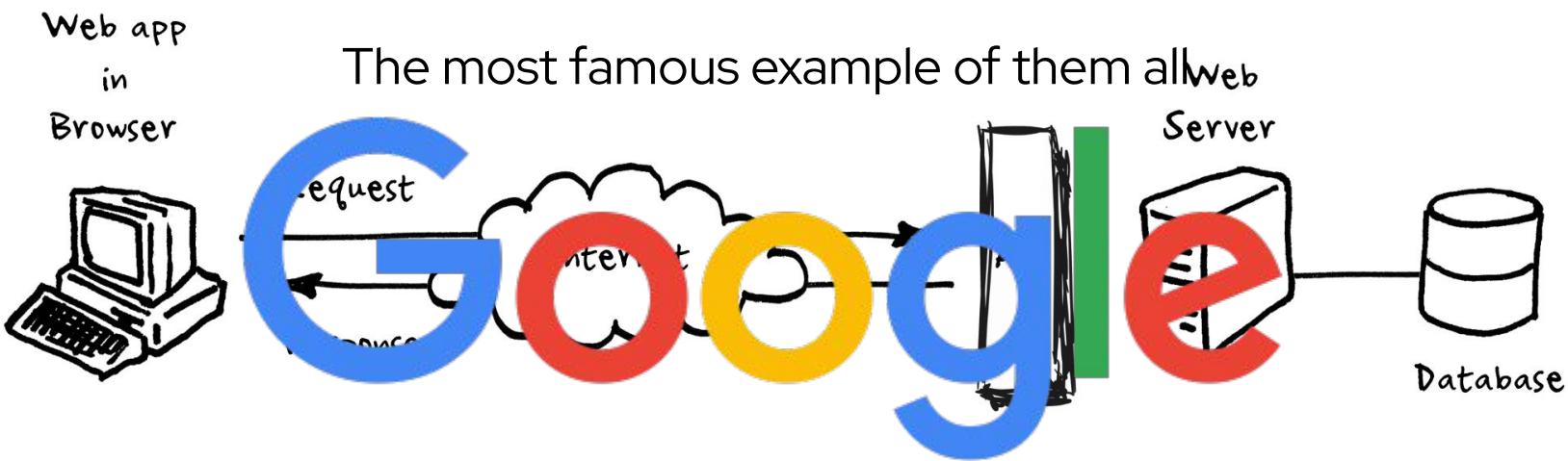
It defines the kinds of calls or requests that can be made, how to make them, the data formats that should be used etc.

WHAT IS AN API?

Machine's API == "I understand requests in the following formats only"

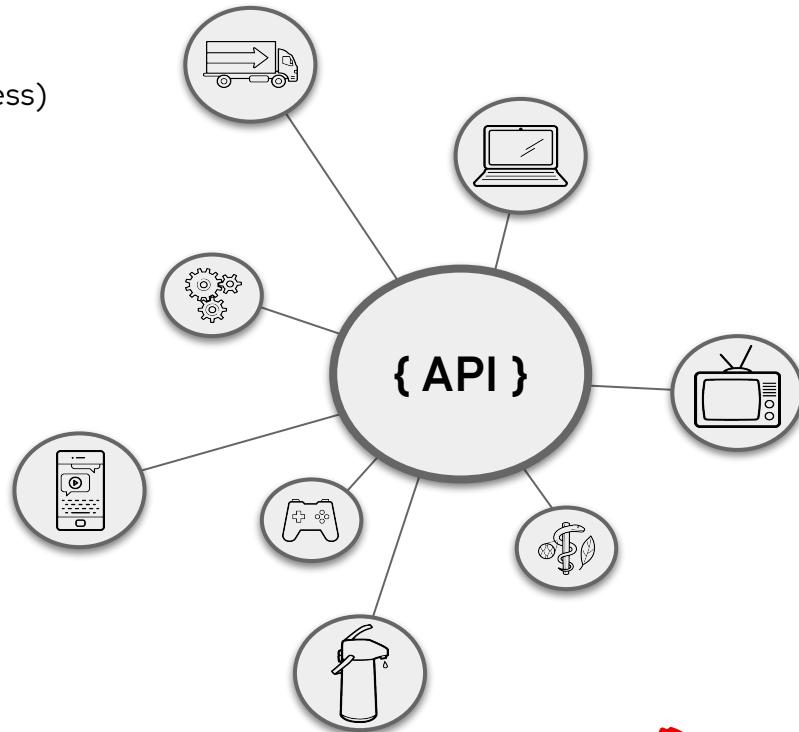


API-exposure architecture - Simplified



APIs Are Everywhere!

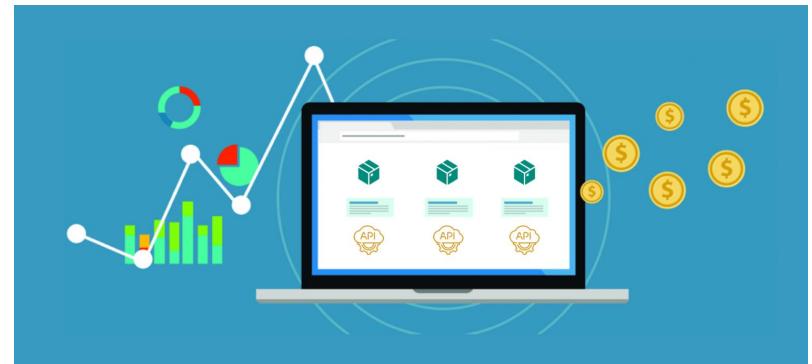
- ▶ Mission critical to every business (or soon every business)
- ▶ Customers and partners depend on it
- ▶ Fundamental to company agility and competitiveness
 - (It is basically a standard nowadays)
- ▶ Web and mobile depend on APIs
 - And not only them



API Economy

API As A Service

- ▶ Supplier & Consumer
 - Customers (Apps functionality)
 - Partners (Develop new products)
- ▶ Bridge between IT vs. Business
- ▶ Forms of consumption
 - C2B - Customer-to-Business (Web, Mobile, Apps)
 - B2B - Business-to-Business (Internally & Externally)
- ▶ *API Access Monetization Question - How?*

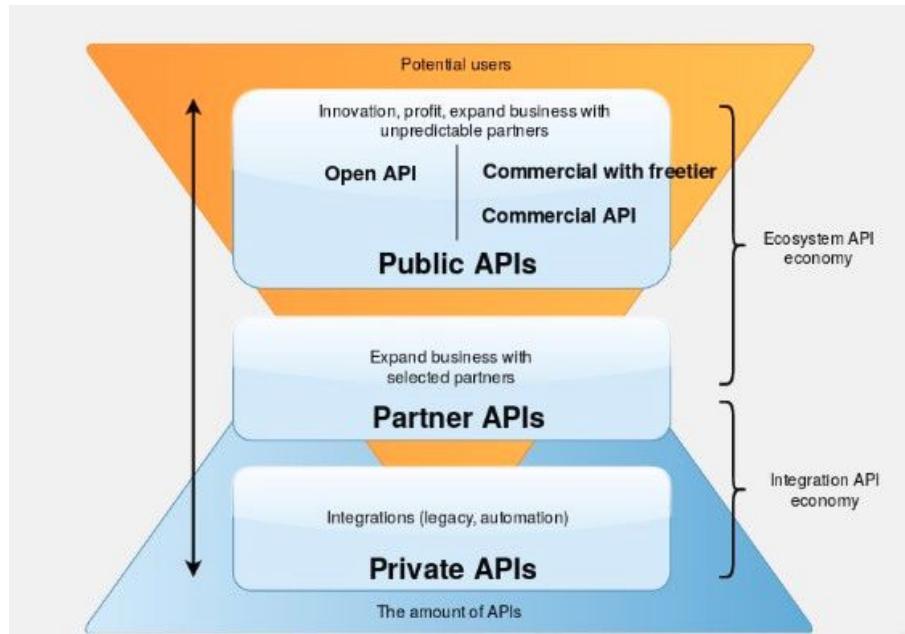


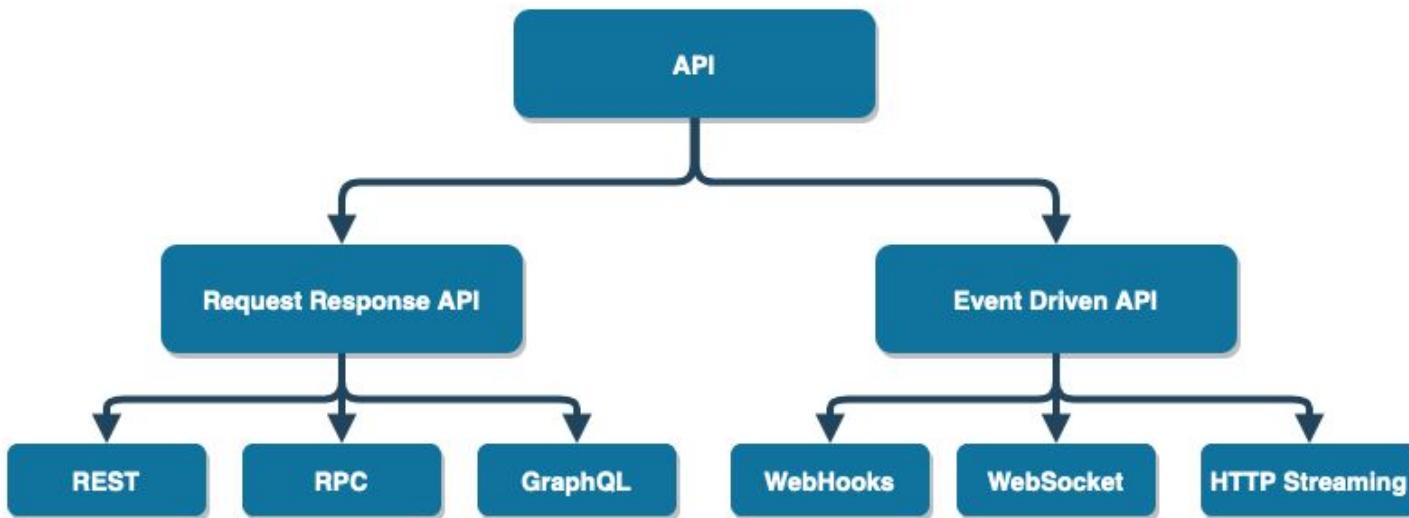
APIs - in depth

<https://rapidapi.com/blog/types-of-apis/>

API Categories

- ▶ Open APIs
 - i.e. GoogleMaps
- ▶ Private APIs
 - Intermediate consumption only
 - Internal DB / Organizational Shared Resource
- ▶ Partner APIs
 - Github
 - Cross organizational products' development (e.g. SaaS)





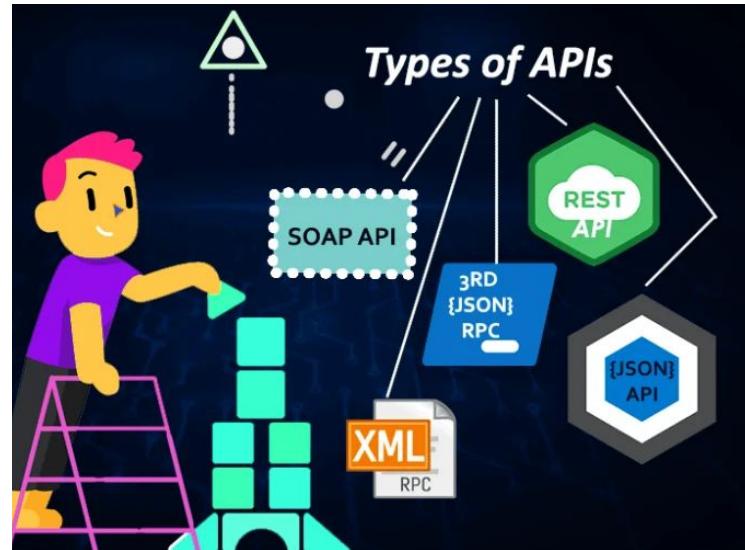
<https://laptrinhx.com/comparing-api-architectural-styles-soap-vs-rest-vs-graphql-vs-rpc-1726803065/>

API TIMELINE



Web (HTTP/S) API Types (Most Popular)

- ▶ RESTful API
 - CURD oriented
 - E.g. Webhook
- ▶ SOAP
 - XML based
- ▶ RPC
 - Application Specific
 - XML vs. JSON
- ▶ gRPC
 - Declarative Docs
- ▶ GraphQL
 - Robust use cases
 - one to many query association



RESTful API

HTTP Verb	Operation	URL Example
GET	Read	GET /api/project/1
POST	Create	POST /api/project
DELETE	Delete	DELETE /api/project/1
PUT	Update	PUT /api/project/1

Small Example

RESTful API

- ▶ REST = **R**epresentational **S**tate **T**ransfer
 - Interactive applications that use Web services
 - OR
 - Non-Interactive automation mechanisms
- ▶ PreDetermined calls (GET, POST, UPDATE, DELETE)
 - payloads lighter
- ▶ REST APIs uses different data formats
 - plain text (raw) , HTML, XML, and JSON sending payload
- ▶ Stateless

Swagger Editor

API Specification Visualization

➤ [Swagger editor](#)

➤ File ⇒ import url ⇒

<https://petstore.swagger.io/v2/swagger.json>

The screenshot shows the Swagger Editor interface. On the left, the raw Swagger JSON code for the Petstore API is displayed. On the right, a preview of the Petstore API is shown, featuring a sidebar with service information and a main area with API operations categorized by resource (Pet, Store).

Swagger Editor

```
swagger: "2.0"
info:
  description: 'This is a sample server Petstore server. You can find out more about Swagger at [http://swagger.io](http://swagger.io) or on [irc.freenode.net, #swagger](http://swagger.io/irc/). For this sample, you can use the api key "special-key" to test the authorization filters.'
  title: Swagger Petstore
  termsOfService: 'http://swagger.io/terms/'
  contact:
    email: apiteam@swagger.io
  license:
    name: Apache 2.0
    url: 'http://www.apache.org/licenses/LICENSE-2.0.html'
host: petstore.swagger.io
basePath: /v2
tags:
  - name: pet
    description: Everything about your Pets
  - name: store
    description: Access to Petstore orders
  - name: user
    description: Operations about user
  - name: store
    description: Find out more about our store
schemes:
  - https
  - http
paths:
  /pet/{petId}/uploadImage:
    post:
      tags:
        - pet
      summary: uploads an image
      description: ''
      operationId: uploadFile
      consumes:
        - multipart/form-data
      produces:
        - application/json
      parameters:
        - name: petId
          in: path
          description: ID of pet to update
          required: true
          type: integer
          format: int64
        - name: additionalMetadata
          in: query
          description: Additional data to pass to server
          required: false
          type: string
          format: string
        - name: file
          in: formData
          description: file to upload
          required: false
          type: file
      responses:
        200:
          description: successful operation
          schema:
            $ref: '#/definitions/ApiResponse'
      security:
        - petstore auth:
          - 'write:pets'
          - 'read:pets'
```

Swagger Petstore

[Basic URL: petstore.swagger.io/v2]

This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on [irc.freenode.net](#) to test the authorization filters.

Terms of service
Contact the developer
Apache 2.0
Find out more about Swagger

Schemes
HTTP

pet Everything about your Pets

Method	Path	Description
POST	/pet/{petId}/uploadImage	uploads an image
POST	/pet	Add a new pet to the store
PUT	/pet	Update an existing pet
GET	/pet/findByStatus	Finds Pets by status
POST	/pet/findByTags	Finds Pets by tags
GET	/pet/{petId}	Find pet by ID
POST	/pet/{petId}	Updates a pet in the store with form data
DELETE	/pet/{petId}	Deletes a pet

store Access to Petstore orders

Method	Path	Description
POST	/store/order	Place an order for a pet

Postman Testing Tool

➤ <https://www.postman.com/>

The screenshot shows the Postman interface with several numbered annotations:

- Annotation 1: A yellow box highlights the "GET" method in the request URL bar.
- Annotation 2: A red circle with the number 2 is placed over the URL field containing "https://jsonplaceholder.typicode.com/users".
- Annotation 3: A red circle with the number 3 is placed over the "Send" button.
- Annotation 4: A red circle with the number 4 is placed over the status bar at the bottom right, which shows "status: 200 OK", "Time: 784 ms", and "Size: 6.27 KB".
- Annotation 5: A yellow box highlights the JSON response body, which displays a list of user objects. One specific user object is highlighted with a yellow box, showing details like id: 1, name: Leanne Graham, and address: Kulas Light, Apt. 556, Gwenborough, 92998-3874.

```
1: {
  "id": 1,
  "name": "Leanne Graham",
  "username": "Bret",
  "email": "Sincere@april.biz",
  "address": {
    "street": "Kulas Light",
    "suite": "Apt. 556",
    "city": "Gwenborough",
    "zipcode": "92998-3874",
    "geo": {
      "lat": "-37.3159",
      "lng": "81.1496"
    }
  },
  "phone": "1-770-736-8031 x56442",
  "website": "hildegard.org",
  "company": {
    "name": "Romaguera-Crona",
    "catchPhrase": "Multi-layered client server neural-net",
    "bs": "harness real-time e-markets"
  }
},
2: {
  "id": 2,
  "name": "Ervin Howell",
  "username": "Antonette"
}
```

SOAP - Simple Object Access Protocol

- ▶ An established web API protocol
 - Can also be implemented for TCP, SMTP etc.
- ▶ (HTTP) Header, XML Body
 - Compatible with schema
- ▶ WSDL - Web Services Definition Language
 - Dictionary of APIs accessible
 - Swagger Equivalent

SOAP Code Example

```
@WebService(endpointInterface="com.redhat.service.Stores")
public class StoresWS implements Stores {

    @Inject
    StoreDAO storeDAO;

    @Override
    public String createStore(Store store) {
        store = new Store(store.getStoreName(),store.getStoreLat(),store.getStoreLong());
        storeDAO.createStore(store);
        return "Store ID:" + store.getStoreID() + " CREATED";
    }

    @Override
    public String deleteStore(int storeID) {
        storeDAO.deleteStore(storeID);
        return "Store ID: " + storeID + " DELETED";
    }

    @Override
    public Store getStore(int storeID) {
        return storeDAO.getStoreById(storeID);
    }

    @Override
    public StoresType getAllStores() {
        StoresType st = new StoresType();
        st.store = storeDAO.getAll();
        return st;
    }
}
```

WSDL Browser

WSDL Browser

Functions

<http://stores-api-user16.apps.47d6.apps.cluster-47d6.47d6.example.opentlc.com/Stor> [Browse WSDL](#)

getAllStores

Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns1="http://www.rhmart.com/Stores/">
  <SOAP-ENV:Body>
    <ns1:getAllStores/>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

[Call function](#)

getAllStores

...

Result:

```
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ns2:getAllStoresResponse
            xmlns:ns2="http://www.rhmart.com/Stores/">
            <Stores>
                <store>
                    <storeID>1</storeID>
                    <storeName>Downtown
                    <Store></storeName>
                    <storeLat>-34.6052704</storeLat>
                    <storeLong>-58.3791766</storeLong>
                </store>
                <store>
                    <storeID>2</storeID>
                    <storeName>EastSide
                    <Store></storeName>
                    <storeLat>-34.5975668</storeLat>
                    <storeLong>-58.3710199</storeLong>
                </store>
            </Stores>
        </ns2:getAllStoresResponse>
    </soap:Body>
</soap:Envelope>
```

SOAP UI

- ▶ Send SOAP requests
 - left - request
 - right - response
- ▶ Better visibility of XML
- ▶ Add header parameters

The screenshot shows the SOAP UI interface with the following details:

- Request Tab:** Displays the SOAP request sent to the URL `https://api-sj-3scale-apicast-staging.apps.nal.openshift.opentlc.com/StoresWS`. The user key is included in the header: `<Header><user key=6779c1c338bc6f7469a7382b4516eba5</Header>`.
- Response Tab:** Displays the XML response received from the server. It includes a list of stores with their IDs, names, latitudes, and longitudes.
- Header Panel:** Shows the SOAPAction header set to `Stores/getAllStores`.
- Table Panel:** Shows the response headers:

Header	Value
Server	openresty/1.11.2.4
Set-Cookie	00e0667ed320ae46e084a856226cd96e=d... 8cc359ab4179ebbe80a2ccc827b5ba67d=c...
#status#	HTTP/1.1 200 OK
Date	Mon, 26 Feb 2018 19:09:41 GMT
Content-Type	text/xml;charset=UTF-8
X-Powered-By	Undertow/1
- Bottom Status Bar:** Shows the response time: `1564ms (501 bytes)`.

RPC

Remote Procedure Call

- ▶ Specifications
 1. Internal API's parameters awareness
 2. Query info transferred in the request's data
 3. Not relying solely on verbs (GET/POST, etc.)

- ▶ Biggest Con of RPC

The sender need to be aware of the exact "serialization" of the payload and every information that relates to the specific method

One typo and you're out

RPC vs. REST

URL Query:

// Request

```
curl -X PUT https://localhost:8080/api/users?name=Piers
```

URL path:

// Request

```
curl -X PUT https://localhost:8080/api/users/Stephen
```

HTTP body:

// Request

```
curl -X PUT https://localhost:8080/api/users { "name": "Josh" }
```

This can be a bit overwhelming, and sometimes we're not sure which one to use; but JSON-RPC is here to help us sort this out since it offers a simpler way:

// Request

```
curl -X POST -d '[{  
    "jsonrpc": "2.0",  
    "method": "addUser",  
    "params": { "name": "Angad" },  
    "id": 1  
}]' https://localhost:8080/api
```

RPC Batch Actions

- ▶ RPC Pro
 - 1. Batch actions
 - 2. Less packets => Lower overhead => Increase performance

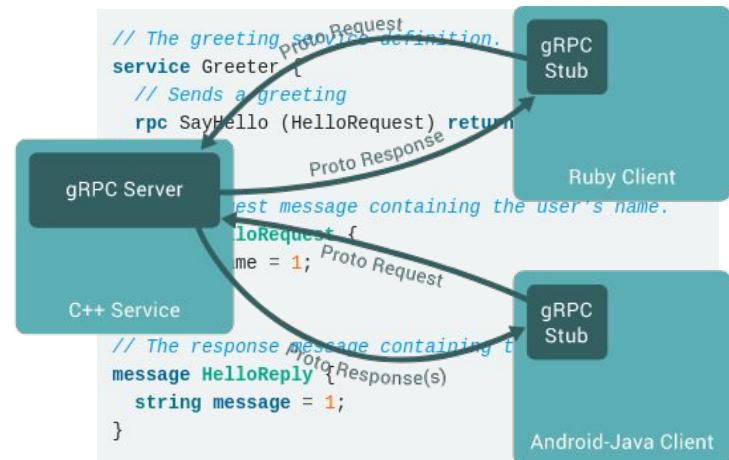
Batched action

```
// Request
curl -X POST -d '[{
    "jsonrpc": "2.0",
    "method": "addUser",
    "params": { "name": "Zalan" },
    "id": 1
}, {
    "jsonrpc": "2.0",
    "method": "addUser",
    "params": { "name": "Dan" },
    "id": 1
}, {
    "jsonrpc": "2.0",
    "method": "addUser",
    "params": { "name": "Edgars" }
}]' https://localhost:8080/api
```

gRPC

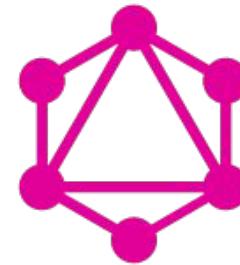


- ▶ Definitions all over!
Solves RPC biggest con
- ▶ 3Scale 2.8+ Supports gRPC
- ▶ Scalable, Fast, Works across multiple languages and platforms
- ▶ <https://grpc.io/docs/what-is-grpc/>



GraphQL

- ▶ The best explanation is an example - <https://graphql.org/>
- ▶ One - to - Many API Query tool
- ▶ Graphql vs. REST - [Medium Blog](#)
- ▶ **Currently not supported by 3Scale (Future Feature)**



Get Your Hands Dirty

Requirements:

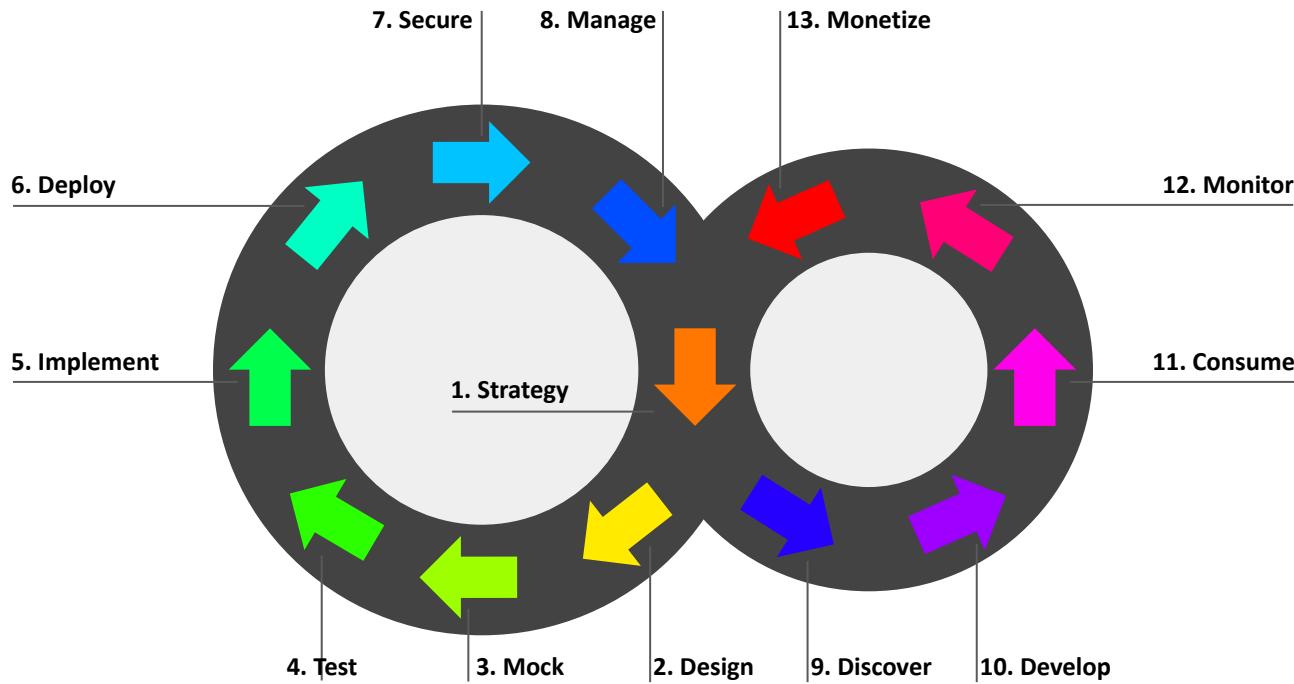
OCP client (oc), python, jq

Optionally use an IDE (such as VS Code) to view files

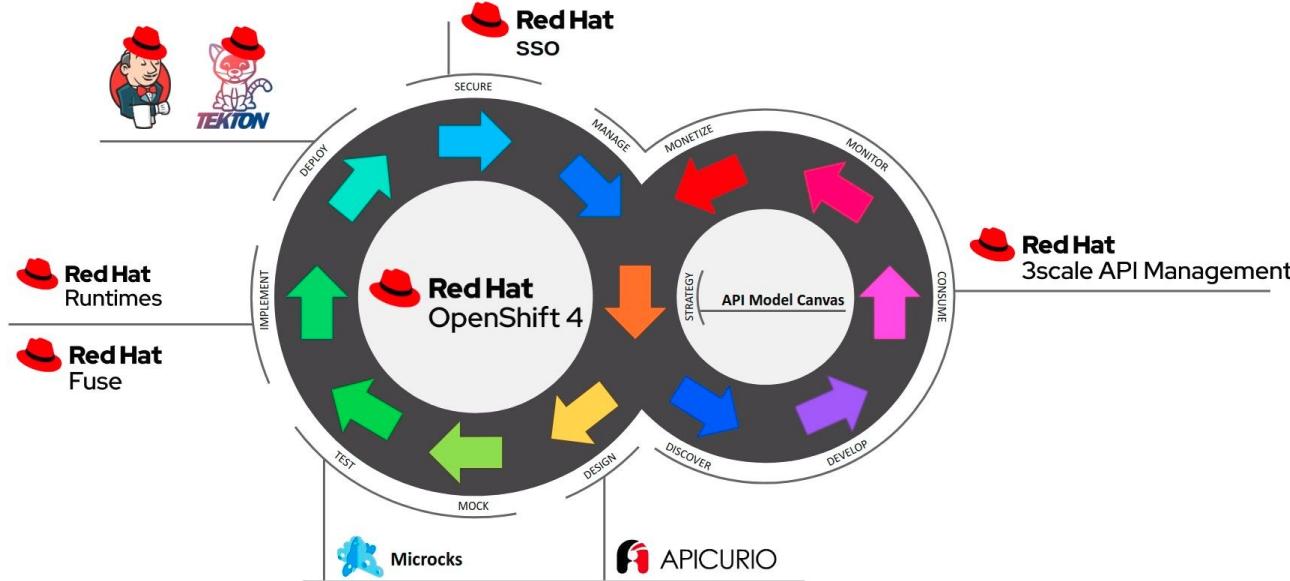
[Exercise 1](#)

RED HAT AGILE INTEGRATION

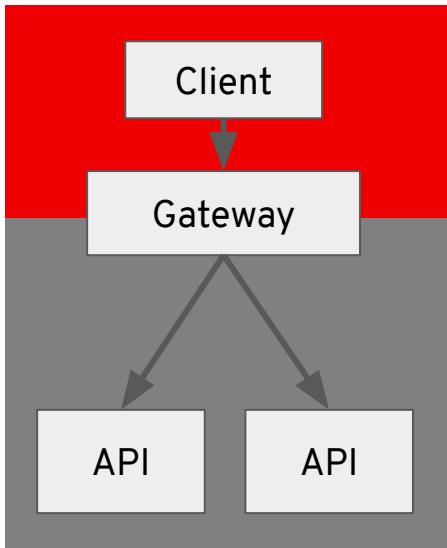
FULL API LIFECYCLE MANAGEMENT



API LIFECYCLE AND STACK - OCP 4.x



Most Common Implementation for API GW



- APIs as a digital access point for your business
- “North-South” service architecture pattern
- Requires traditional API management capabilities
- APIs As A Product

Red Hat 3Scale Overview & Basic Architecture

RED HAT 3SCALE API MANAGEMENT

100% Open Source

Modular

No single point of failure

Quick time-to-value

Highly scalable

Support Multi Tenancy

Support Remote-Implementation Options



My Medium Articles

[API Management Overview](#)

[Installation Guide](#)

API Management Main Disciplines

Visibility

- Analytics
- App Tracking
- User Tracking
- Traffic Alerts
- Engagement
- Developer Support

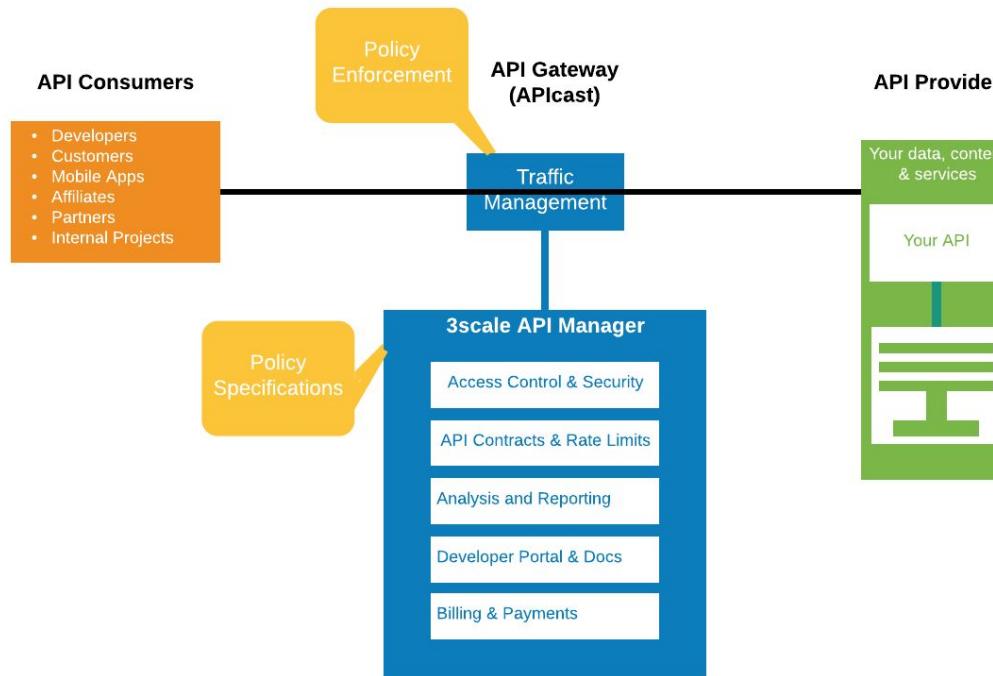
Control

- Security
- Key Management
- Rate Limiting
- Policy Enforcement
- App & User Management
- Provisioning

Flexibility/scalability

- Distributed
- Multi-Department
- Multi-Environment
- Highly Scalable
- Powerful APIs
- Webhooks

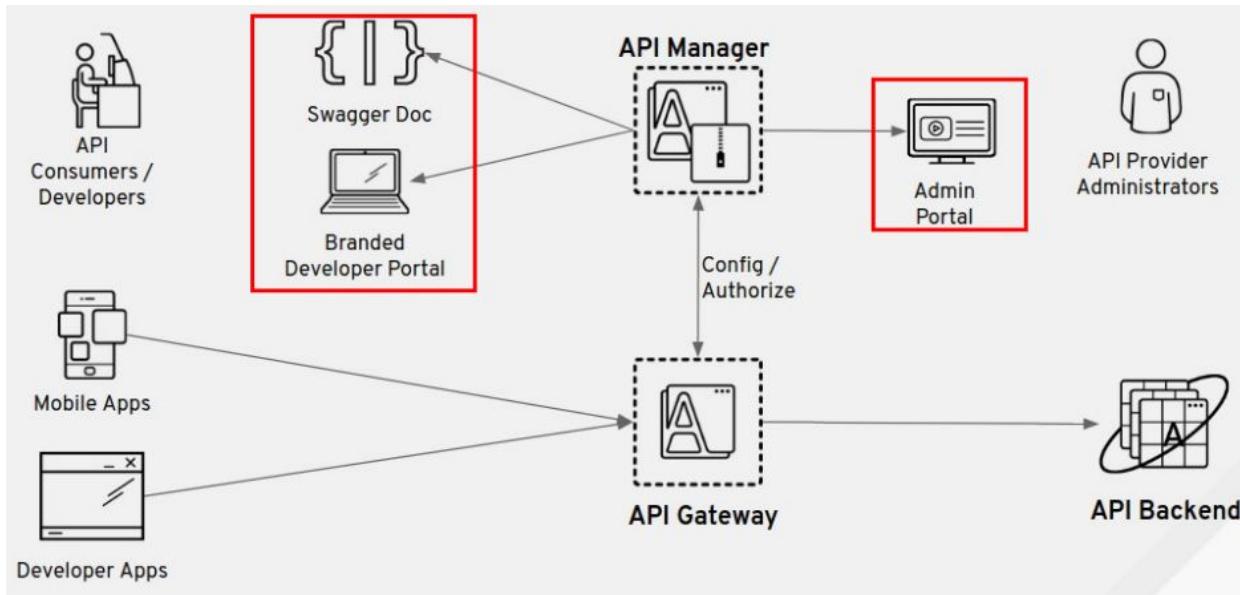
Terminology



Multiple Usage Types

- ▶ Internal Developers
 - Subscription Plans
- ▶ 3Scale Admins
 - Product Management, Deployment & Scaling
- ▶ App Client - Mobile/Browser
 - Account Plans
- ▶ External Companies - Developers
 - Customizable Developer Portal
 - Monetization
 - Also relevant for integration between internal services (East-West)

Complete Diagram



API MANAGEMENT vs. LIFECYCLE



MANAGEMENT

CONTROL

- Security
- Key management
- Rate limiting
- Policy enforcement
- User management
- Provisioning

VISIBILITY

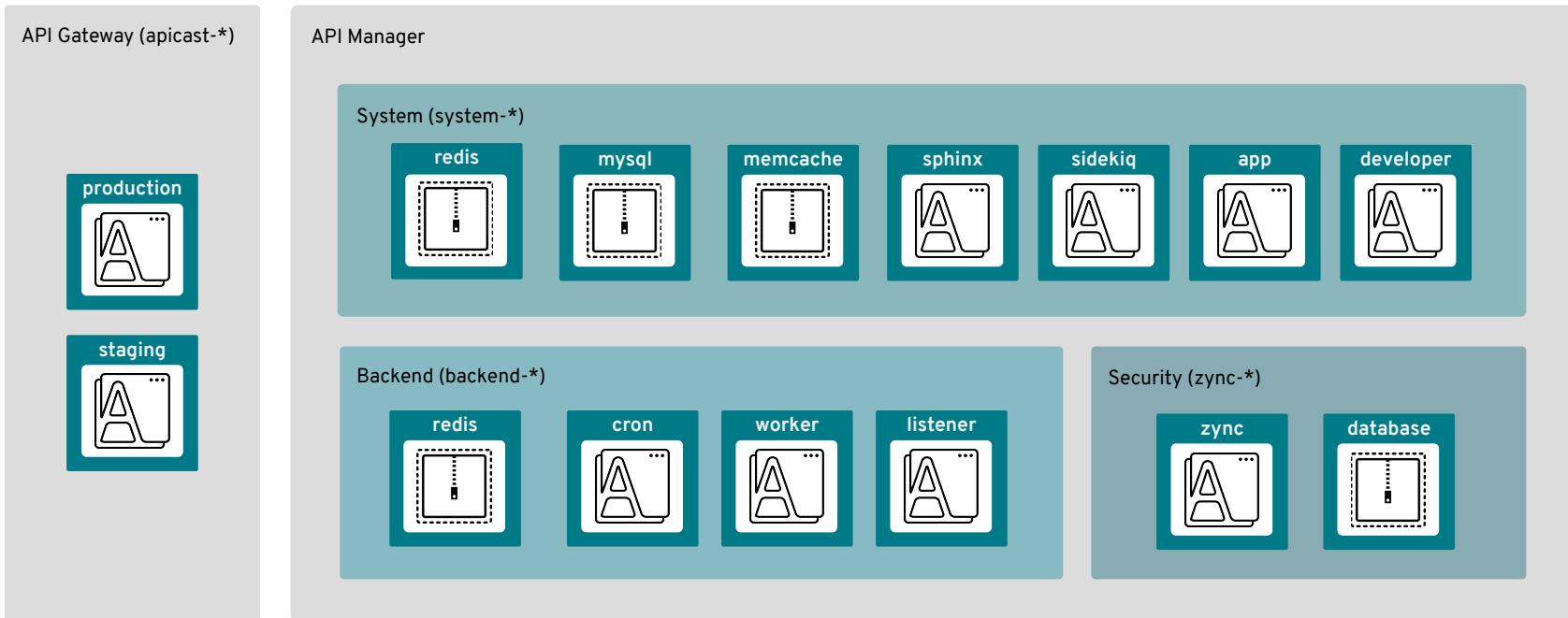
- Analytics
- App tracking
- User tracking
- Traffic alerts
- Engagement
- Developer support



LIFECYCLE

- Testing
- Release
- Versions
- Deployment
- SLA

3Scale Components



RED HAT 3SCALE API MANAGEMENT

- ▶ Storage Requirements
 - 3 or 4 Persistent Volumes
 - 3 RWO PV for Redis and MySQL/Postgres persistence (always needed)
 - 1 RWX PV (amp.yml template) **MUST** be group writable: `chmod g+w`
or
· AWS S3 Bucket for CMS assets (amp-s3.yml template)
- ▶ 10GB storage should be plenty to get started, 30GB for Production
- ▶ PVs should be writeable by OpenShift (Permissions)
 - Owned by root group
 - Proper selinux permissions: `chcon -Rt svirt_sandbox_file_t`
- ▶ [Supported Configuration](#) - Update Periodically

Deploy 3Scale with S3 using Operator

- ▶ [Link to Documentation](#)
- ▶ The RHPDS lab environment runs on AWS managed, which means that it has credentials for the aws bucket for the internal registry, and I used them to deploy 3scale on the same storage
 - oc describe configs.imageregistry.operator.openshift.io cluster | grep "Bucket:"
 - oc describe configs.imageregistry.operator.openshift.io cluster | grep "Region:"
 - oc extract secret/installer-cloud-credentials -n openshift-image-registry --to---
- ▶ **Note!** You won't see a PVC 'system-storage', unlike with the other option (NFS)

```
kind: Secret
metadata:
  creationTimestamp: null
  name: aws-auth
stringData:
  AWS_ACCESS_KEY_ID: AKIA4TELPBRUEPWZQW55
  AWS_SECRET_ACCESS_KEY: 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
  AWS_BUCKET: cluster-3003-f6bqs-image-registry-eu-west-1-sqpcmucqiymddjgiql
  AWS_REGION: eu-west-1
type: Opaque
```

The screenshot shows the Red Hat OpenShift Container Platform web interface. The left sidebar contains a navigation menu with the following items:

- Administrator
- Home
- Operators
- OperatorHub
- Installed Operators** (selected)
- Workloads
 - Pods
 - Deployments
 - DeploymentConfigs
 - StatefulSets
 - Secrets
 - ConfigMaps
- CronJobs
 - Jobs
 - DaemonSets
 - ReplicaSets
 - ReplicationControllers
 - HorizontalPodAutoscalers
- Networking
- Storage
- Builds
- Monitoring
- Compute
- User Management

The main content area displays the configuration for the selected operator, "Installed Operators". It includes the following sections:

- Project:** 3scale-test
- Resource Requirements Enabled:** false
- Monitoring:** A collapsed section.
- System:** A collapsed section.
- Redis Image:** A selected item under System.
- Redis Toleration:** A collapsed section.
- Sphinx Spec:** A collapsed section.
- Redis Affinity:** A collapsed section.
- Memcached Image:** A collapsed section.
- Memcached Toleration:** A collapsed section.
- Redis Persistent Volume Claim:** A collapsed section.
- Sidekiq Spec:** A collapsed section.
- Image:** A collapsed section.
- Database:** A collapsed section.
- App Spec:** A collapsed section.
- File Storage:** A collapsed section.
- Amazon Simple Storage Service:** Deprecated.
- Persistent Volume Claim:** A collapsed section.
- Simple Storage Service:** A collapsed section.
- Configuration Secret Ref ***: LocalObjectReference contains enough information to let you locate the referenced object inside the same namespace.
- Name:** aws-auth (highlighted in blue)

At the bottom, there is a note: "Name of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names> TODO: Add other useful fields. apiVersion, kind, uid?"

APIcast (API Gateway)

NGINX - Web Proxy

Lua - Built-in & Custom Policies

OpenResty - Modules to support in various protocols

Supported API Protocols

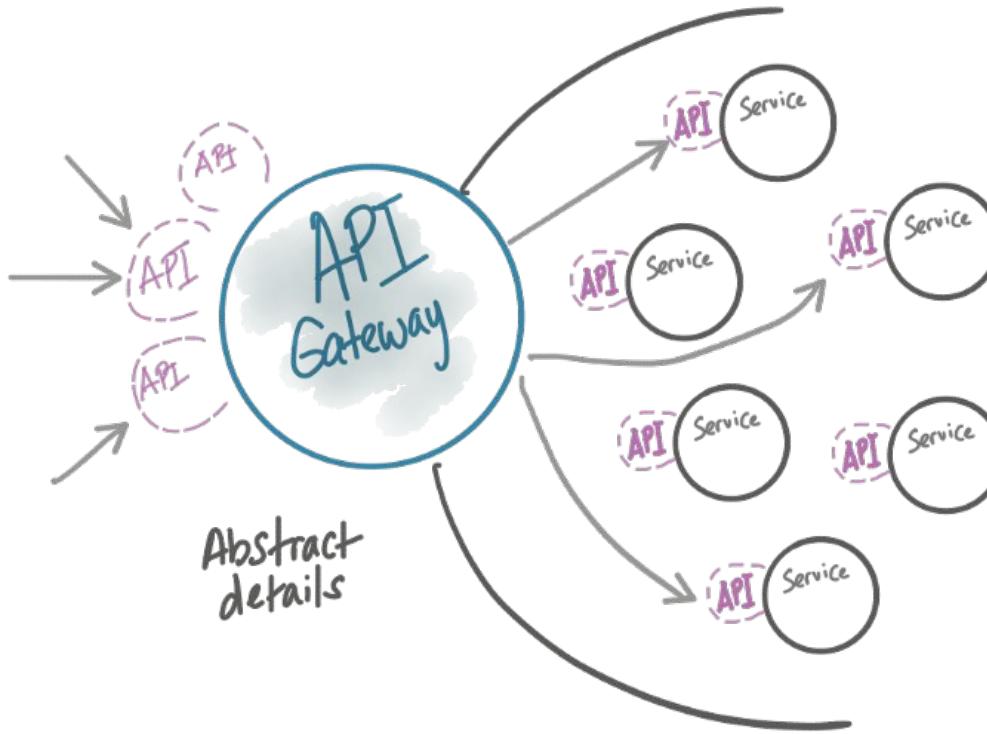
- ▶ REST API (inc. HTTP 2)
- ▶ APICast Policies
 - Simple SOAP (Can be extended with RH Fuse)
 - gRPC
 - WebSocket
- ▶ Graphql, other RPC protocols - Not supported
- ▶ Other protocols (Tcp etc.) - built-in/custom policies, RH-Fuse

3Scale Technical Glossary



- ▶ Backend - Internal Service
- ▶ Product - Externally exposed Service via APICast
- ▶ Method mapping - permit access to specific endpoint-functions
- ▶ Metrics - Counting hits to endpoint-functions

Products & Backends association



API Products & API Backends



API Product public/managed

Business layer

- Public facing ("facade") → URL structures
- Can use multiple API backends
- Sign-up → API creds → AuthN
- App plans = limits and pricing rules
- (Global) metrics/mapping rules
- APIcast policies



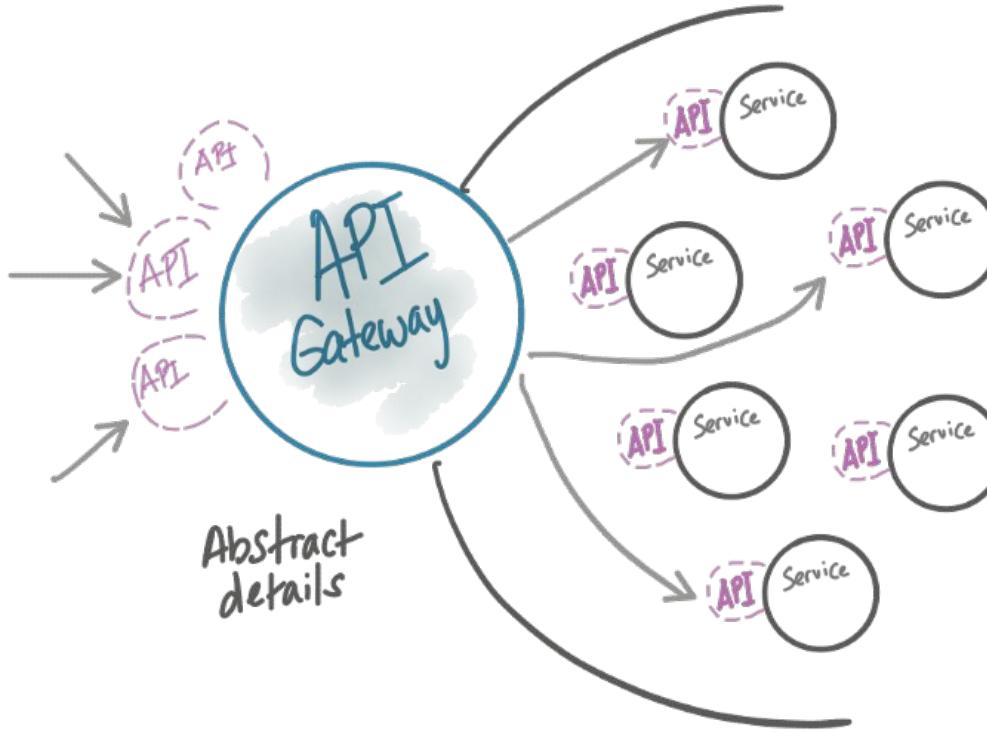
API Backend private

Technical layer

- Private base URL
- Can be used by multiple API products
- (Local) metrics/mapping rules (by method)



Take another look, Just for the sake of it



Mapping Rules 101

Basic: GET /page

Wildcard: GET /frontpage/{version}

Exact Match: GET /wow\$

Passing values: From URL: GET /something?example_value={value} Or From Body:

POST/PUT/DELETE /something?another_example_value={value}

Note! More than one mapping rule can match the request path, but if none matches, the request is discarded with an HTTP 404 status code.

Mapping Rules



Product-level Mapping Rules

- Take precedence – i.e. go on top of all MRs
- Always evaluated, no matter to which Backend the traffic will be routed



Backend-level Mapping Rules

- Evaluated after Product-level MRs
- Evaluated only if the traffic is being routed to the same Backend the Mapping Rule belongs to
- Same set of MRs exists in all Products using the Backend
- The path of the Backend in a given Product is automatically and transparently prepended to each MR of the Backend in that Product

Methods & Metrics



Product-level Methods & Metrics

- “Hits” metric counts hits mapped to “Hits” itself and to its methods + the hits mapped to all Backend-level “Hits” metrics and their methods.

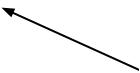


Backend-level Method & Metrics

- Registered in Apisonator as if they belonged to each Product using the Backend
- Automatically and transparently get the ID of the Backend appended to the `system_name` of the metric
- Limits and Pricing rules can be set upon Backend-level metrics in the Application Plans at the Product level

Complete Example - Expose APIs via 3Scale

Starting with the backend and then associating them with the products



DEVELOPER & PARTNER PORTAL

The Schiphol Developer Center Documentation page for the Flight API (public) version V4. It includes sections for Overview, Release Notes, Terms and Conditions, Quickstarts, Flights, and Right Statuses. A detailed 'About the Rest API Flight Information' section provides information on endpoints like /airports, /flights, and /airlines, along with examples and code snippets for making requests.

The AXA Partners API documentation page. It features a 'Quick Start Guide' with instructions for developers to build and monetize their app or website using the AXA-Partners API. A 'Pick your plan' section allows users to choose between Developers (with limited calls and 60 hits per minute) and Partners (with unlimited calls and 1000 hits per minute). A 'TRY IT' section shows a Try-It interface for running requests on a sandbox API, with examples for GET /travel/v1/countries and a response in JSON format.

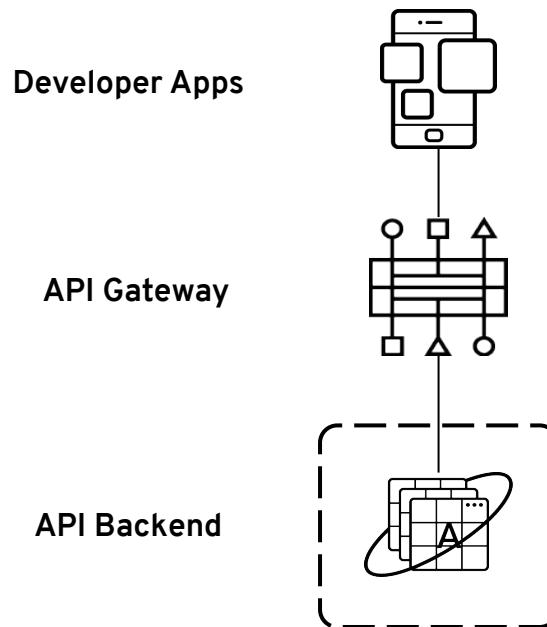
The Yummly Recipe API landing page. It highlights the API as being powered by the world's largest and most powerful recipe search site. A 'Start Your Free Trial' button is prominently displayed. The page also features a video player showing a person using the Yummly app on a smartphone.

The Oxford Dictionaries API documentation page. It features a 'Documentation' section with a grid of user faces, a 'Support' section, and a 'Our Data' section. The main content area shows various ways to use the API, including a 'Dictionary' section for word definitions, a 'LexStats' section for corpus-based statistical data, and a 'Community' section for developer resources. A 'TRY IT' interface is available for testing the API with specific words.



Most Basic & Common Scenario

HTTP/S Web Services Endpoints



Relevant Use-Cases

- ▶ Service is already a HTTP REST / Simple SOAP Endpoint
- ▶ Needs to implement access control and security
- ▶ Direct routing to service implementation

Give it a try

Exercise #2

Features Features Features

Multiple Backends per-APICast

The screenshot shows the Red Hat OpenShift API Catalog interface. At the top, there's a navigation bar with a logo, the text "Product: Coolstore Catalog API", and dropdown menus. Below the navigation bar, the main content area has a title "Backends". A sub-instruction says: "For a Product to work, it needs to have at least one Backend with a Private Base URL (your API). If multiple Backends are added to a Product, each Backend should have a unique Public Path." The main table lists a single backend entry:

Name	Private Base URL	Public Path <small>(?)</small>	Action
Catalog Backend	http://catalog-service.opentlc-mgr-coolstore.svc.cluster.local:8080	/	

A red box highlights the "Add Backend" button in the top right corner of the table header.

Mapping Rules & Wildcards

The screenshot shows a user interface for managing API mapping rules. At the top, there's a header bar with a logo, the text "Product: Coolstore Catalog API", and some icons. Below the header is a search bar labeled "Search for Pattern" and a "Search" button. The main area is titled "Mapping Rules". It has a table with columns: Verb, Pattern, Metric or Method, Last?, Position, and two small icons. A green "Add Mapping Rule" button is located at the top right of the table area.

Verb	Pattern	Metric or Method	Last?	Position	
GET	/▲	1 hits	false	1	
GET	/products	1 get_all_products_metric	false	2	
GET	/product/{itemId}	1 get_product_by_id_metric	false	3	
POST	/product	1 add_new_product_metric	false	4	
GET	/health/readiness	1 health_readiness_check_metric	false	5	
GET	/health/liveness	1 health_liveness_check_metric	false	6	

ActiveDocs

GPTE Coolstore Catalog API

Serves products and prices for retail products

default

Show/Hide | List Operations | Expand Operations

<code>GET</code>	/products	Get a list of products
<code>GET</code>	/product/{itemId}	Get product
<code>POST</code>	/product	Add a new product
<code>GET</code>	/health/readiness	Readiness probe for health check
<code>GET</code>	/health/liveness	Liveness probe for health check

[BASE URL: / , API VERSION: 1.0.0]

3Scale Service Discovery

In-Cluster Services

```
...  
  
apiVersion: v1  
kind: Service  
metadata:  
  annotations:  
    description: The data virtualization services.  
    discovery.3scale.net/port: "8080"  
    discovery.3scale.net/scheme: http  
    openshift.io/generated-by: OpenShiftNewApp  
  creationTimestamp: 2019-02-05T13:13:34Z  
  labels:  
    app: stock-api  
    application: stock-api  
    discovery.3scale.net: "true"  
    template: stock-api  
    xpaas: 1.4.0  
  name: stock-api  
  
...
```

Monitor Usage Per Method

Usage



Rate Limit entire API vs. Rate Limit specific method

The screenshot displays two views of the Red Hat 3scale API Management interface, illustrating different approaches to rate limiting.

Product Level Configuration:

- Metric:** Hits
- Pricing:** 0
- Limits:** 1
- Enabled:** Yes
- Visible:** Yes
- Text only:** Yes
- Period:** 1 minute
- Value:** 5
- Actions:** Edit, Delete

Backend Level Configuration:

- Catalog Backend:** Define method or metric
- Metric:** Hits
- Pricing:** 0
- Limits:** 0
- Enabled:** Yes
- Visible:** Yes
- Text only:** Yes
- Method:** get all products
- Pricing:** 0
- Limits:** 1
- Enabled:** Yes
- Visible:** Yes
- Text only:** Yes
- Period:** 1 hour
- Value:** 100
- Actions:** Edit, Delete

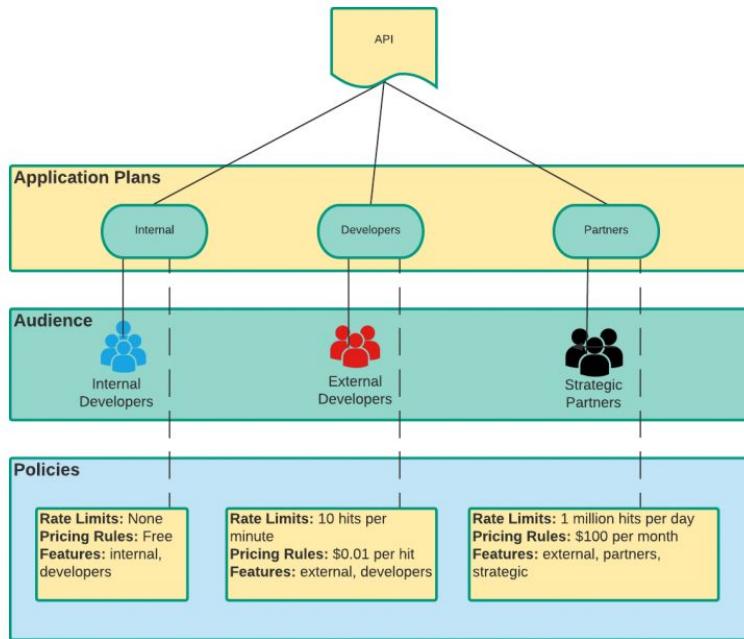
Red Hat Logo: Red Hat logo with the word "Red Hat" next to it.

Prevent Access to Specific Method

Name	Applications	State
Basic	1	published
Premium	2	published

Backend Level	Enabled	Visible	Text only
▼ Catalog Backend (Define method or metric)			
Hits	Pricing (0) Limits (0)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
get all products	Pricing (0) Limits (0)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
get product by id	Pricing (0) Limits (0)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
add new product	Pricing (0) Limits (!)	<input type="checkbox"/>	<input type="checkbox"/>

Policy Separation by Group Association



Policies - Examples

JWT fields Validation

Edit Policy

Cancel

JWT Claim Check

builtin - Allow or deny traffic based on a JWT claim

This Policy allow to block traffic based on a JWT token. To verify any JWT claim can be used and can be compared using plain or liquid filters.

Enabled

RULES

resource_type
How to evaluate 'resource' field

op*
Match operation to compare JWT claim with the provided value. In case that a not a number is in use in numeric comparison, the value will be transformed to 0.

jwt_claim_type*
How to evaluate 'jwt_claim' value

jwt_claim*

Error message
Error message to show to user when traffic is blocked

Remove **Update Policy**

Complete Example Reference

jwt_claim*
String to get JWT claim

value_type
How to evaluate 'value' field

value*
Value to compare the retrieved JWT claim

combine_op

resource*
Resource controlled by the rule. This is the same format as Mapping Rules. This matches from the beginning of the string and to make an exact match you need to use '\$' at the end.

Allowed methods



Policies - Examples

TLS Certification Validation

Edit Policy Cancel

TLS Client Certificate Validation

builtin - Validate certificates provided by the client during TLS handshake (HTTPS).

Validate client certificates against individual certificates and CA certificates.

Enabled

CERTIFICATE WHITELIST

Individual certificates and CA certificates to be whitelisted.

PEM formatted certificate

Certificate including the -----BEGIN CERTIFICATE----- and -----END CERTIFICATE-----

X

+

Remove

Update Policy

Policies - Examples

Multiple policies & Policy Chain order

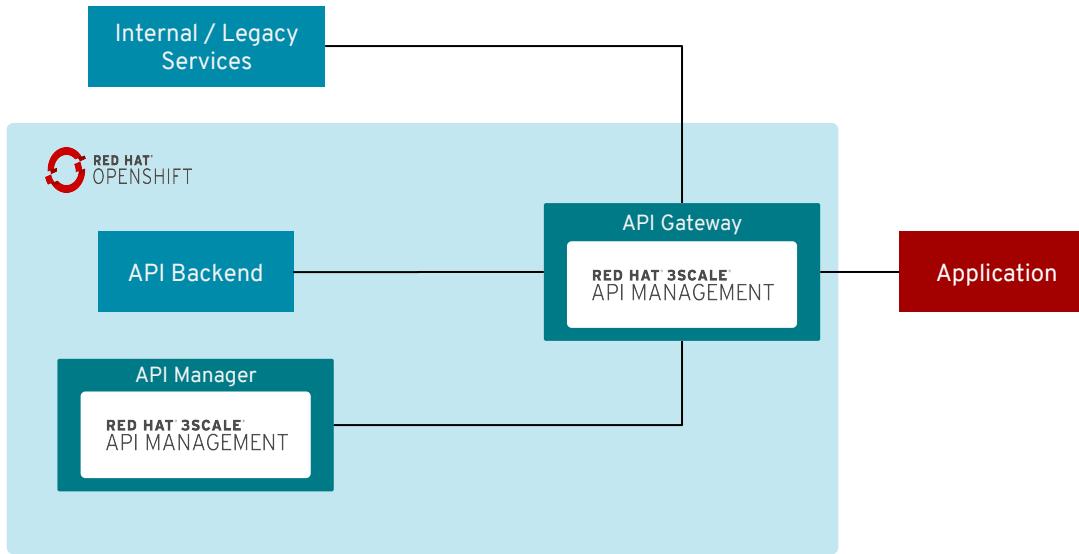
Policy Chain	 Add policy
TLS Client Certificate Validation	
builtin - Validate certificates provided by the client during TLS han...	
IP Check	
builtin - Accepts or denies a request based on the IP.	
JWT Claim Check	
builtin - Allow or deny traffic based on a JWT claim	
3scale APIcast	
builtin - Main functionality of APIcast to work with the 3scale API ...	

Do it yourself

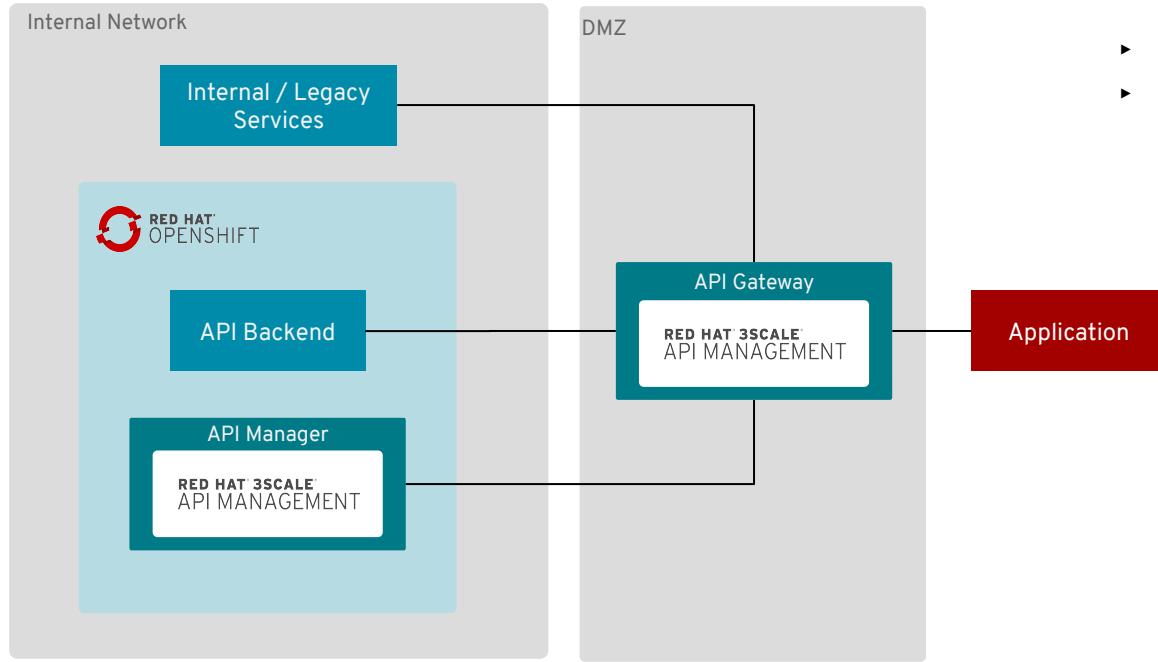
Exercise #3

Day 2 Deployment Options

APICast Internal deployment

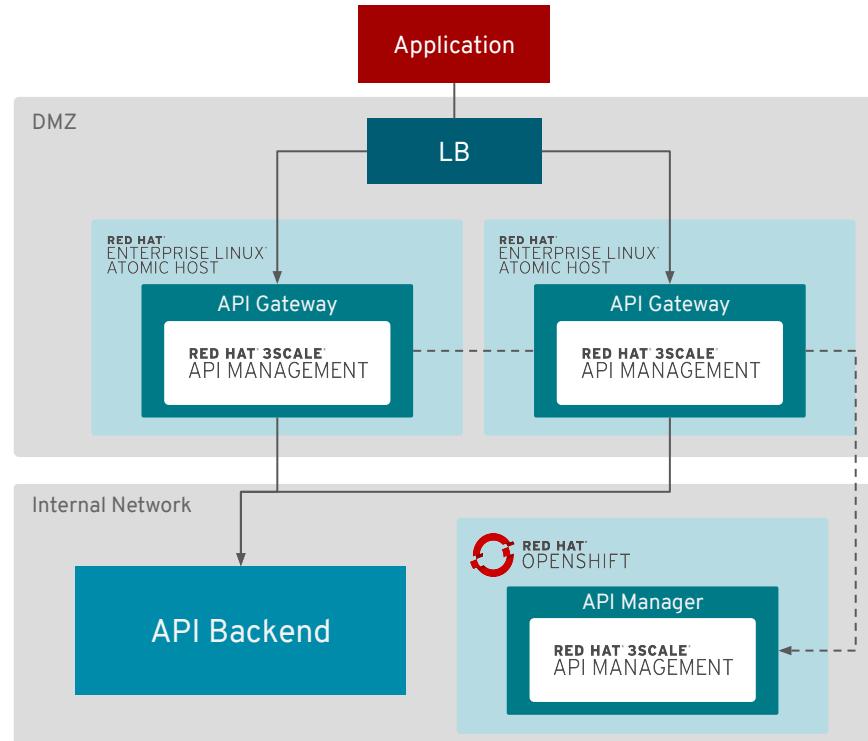


APICast external deployment

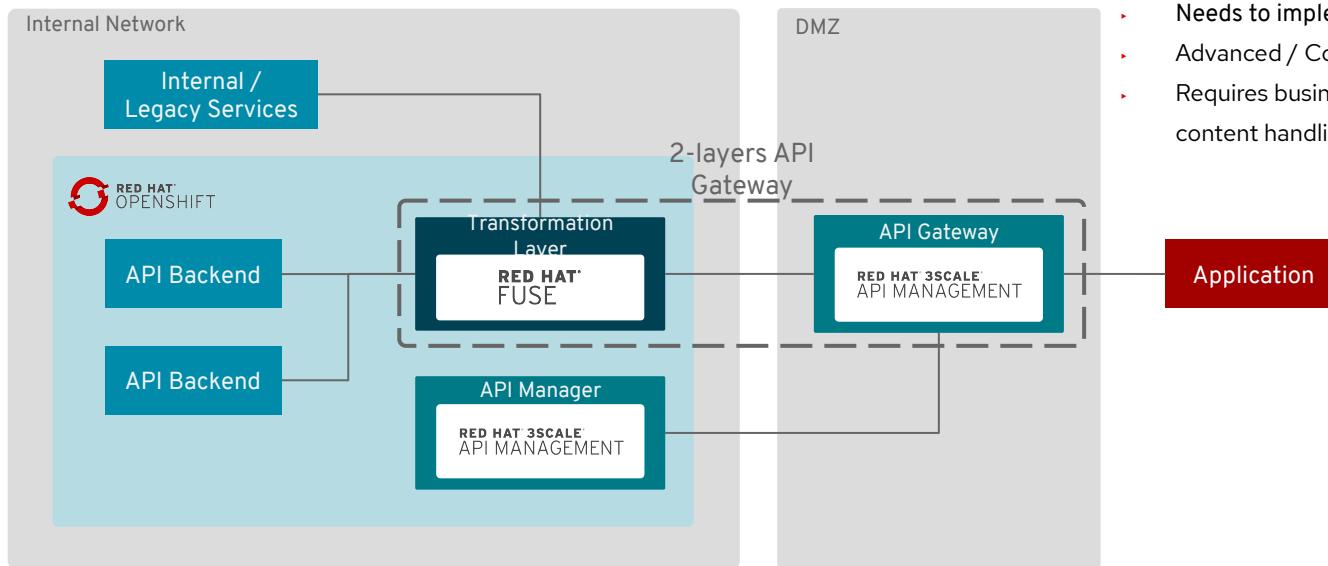


- ▶ AMP - Must Run on OCP
- ▶ APICast - Can run on every Dockerized environment outside of OCP
(Bare Metal / Standard VMs for example)

APICast External & High Availability



Fuse Transformation Layer + 3Scale API Gateway

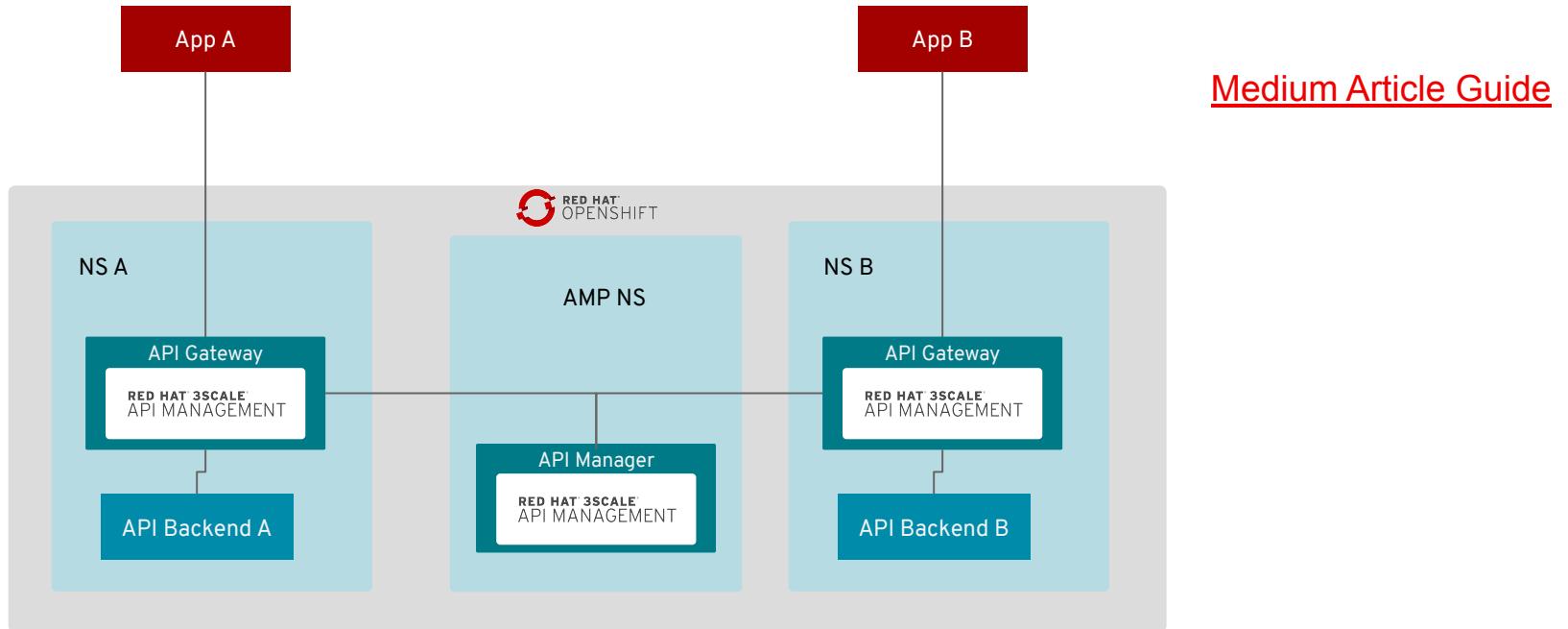


Relevant Cases

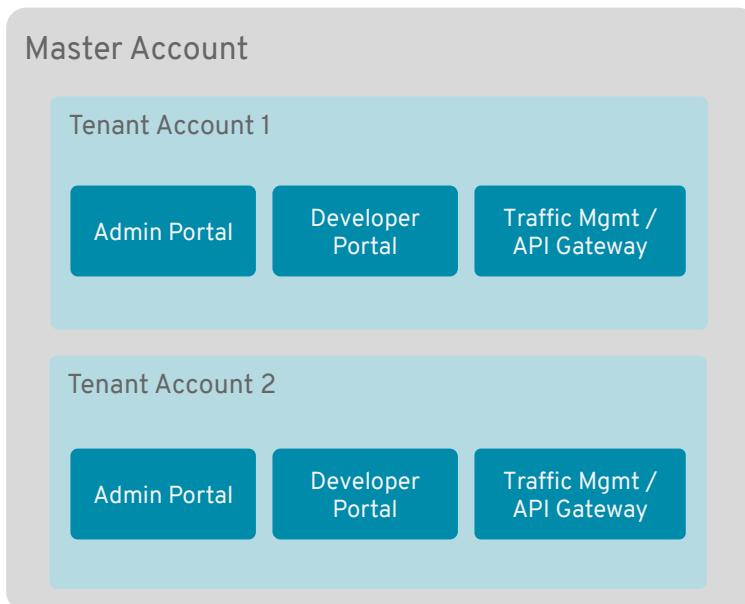
- Service is Complex SOAP Endpoint
- Non-HTTP protocols: TCP, Messaging
- Needs to implement access control and security
- Advanced / Content-based Routing
- Requires business-based transformations / content handling

APICast Self-Managed

APICast Dedicated Per-App, Without risking AMP



Multi-tenant



Master Admin

- Manage Tenants
- Impersonate Tenants

Tenant Admin

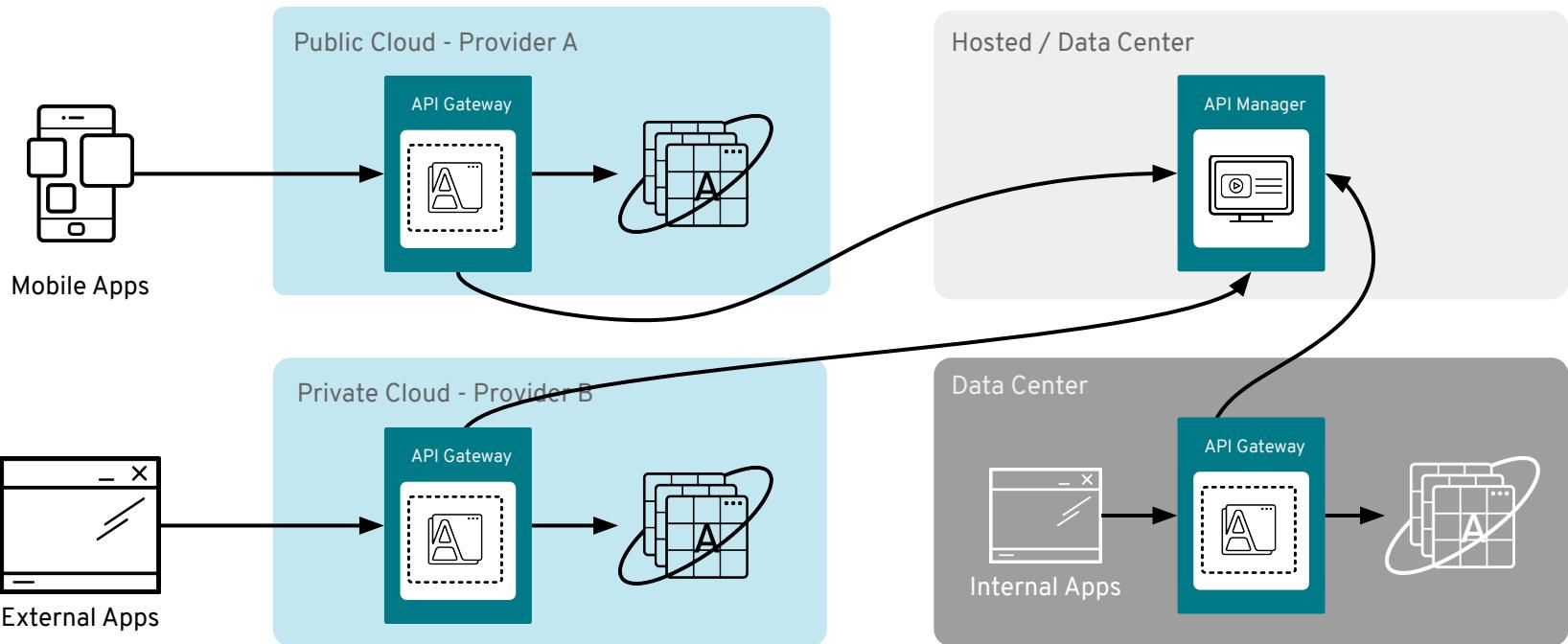
- Manage tenant admins / users
- Access APIs and Admin Portal

Member

- Access given services / sections

[Installation Guide - My Medium Article](#)

Hybrid Deployment



API SECURITY

Main Security Features

- ▶ **Authentication**
 - OpenID Connect / Basic Auth
- ▶ **Authorization** (RBAC)
 - Application Planes & Access Policies
- ▶ **Method mapping**
- ▶ **Built-in & Custom Policies**
 - Drop unwelcome requests by policies rules

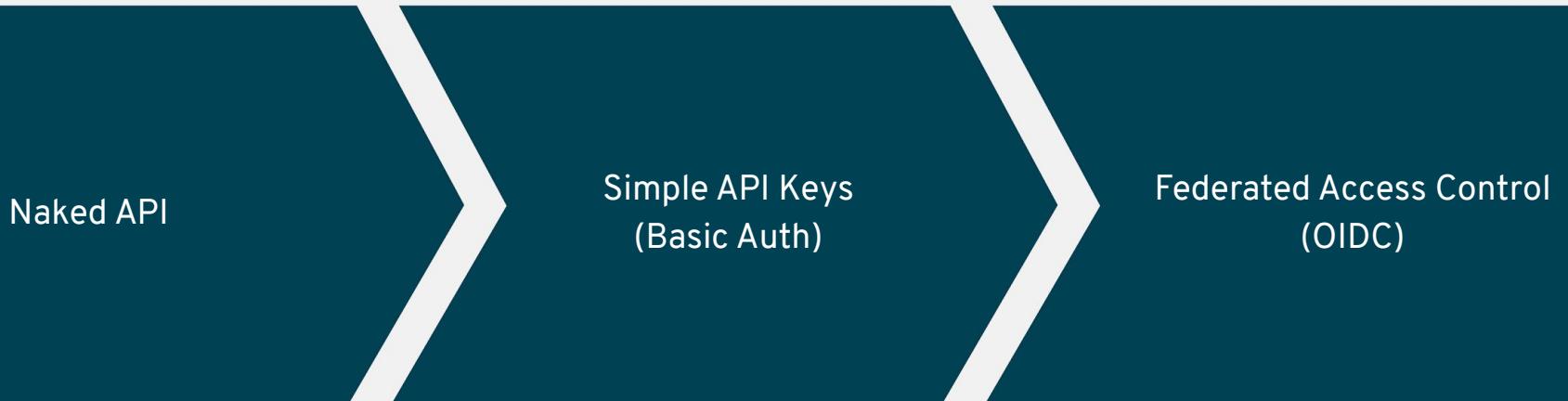
Main Security Features

- **Generate Alerts** for suspicious traffic
- **Rate Limiting**
 - Metrics
- **Advanced Security - Collaboration with other security products**
 - SSO
 - Service Mesh
 - K-WAF



API SECURITY

Evolution of API Security

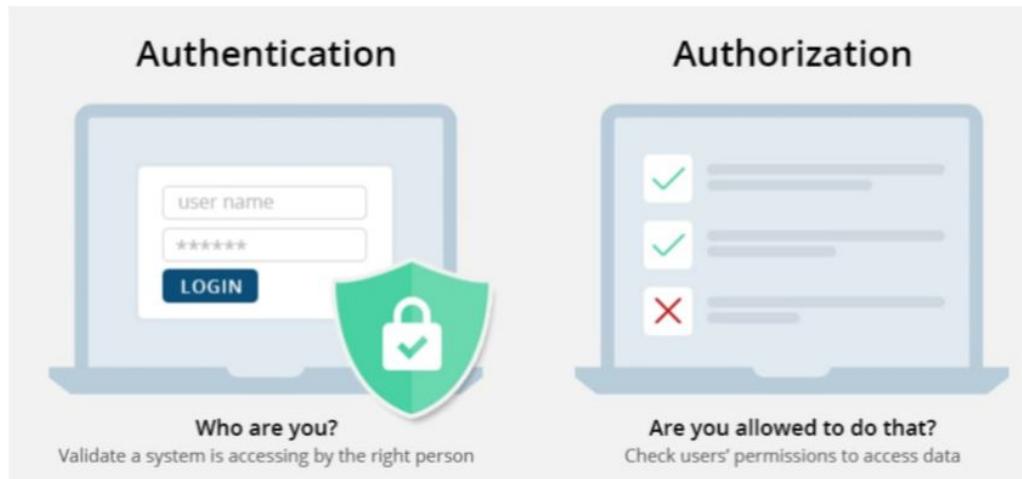


Naked API

Simple API Keys
(Basic Auth)

Federated Access Control
(OIDC)

Auth vs. AuthZ



AUTHENTICATION



API Key: a shared secret used to authenticate a client application. Cannot easily be renewed.

API Key Pair: an identifier + a shared secret used to authenticate a client application. The identifier remains the same during the whole lifetime of the application, the secret can easily be renewed to ensure higher security.



OpenID Connect: a standard protocol to authenticate the client application and the end-user connected on this application. Currently the highest level of security.

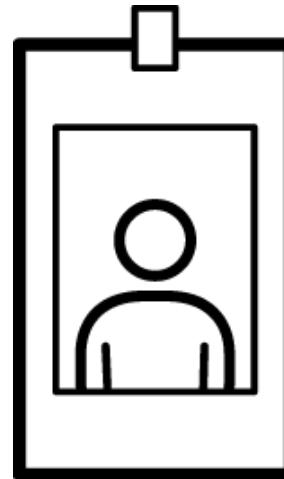
OPENID CONNECT - Auth + AuthZ

- ▶ Built on top of the OAuth 2.0 protocol
- ▶ Stateless, JWT token
- ▶ Allows clients to verify the identity of an end user and obtains basic profile information
- ▶ RESTful HTTP API, using JSON as a data format
- ▶ [My Medium Article – 3Scale & RHSSO \(OpenID Connect Implementation\)](#)



OPENID CONNECT

- ▶ Provides identity information to the application from the Authority Server
- ▶ Base64 encoded - easy to work with.



Name: John Doe

Type: Employee

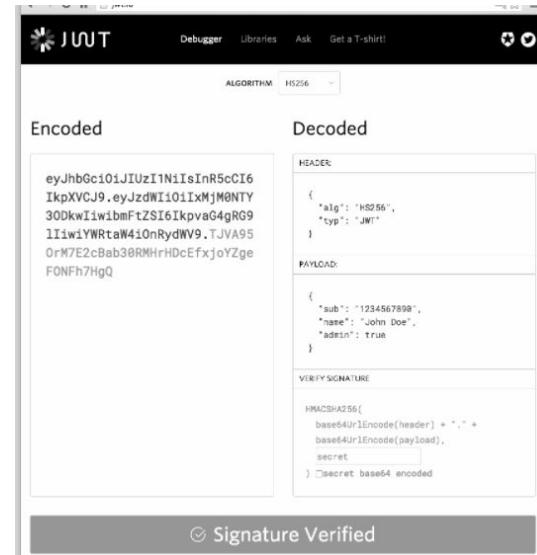
Issued by: Company

Expiration Date:

02-06-2019

JWT (“JOT”) to the Rescue

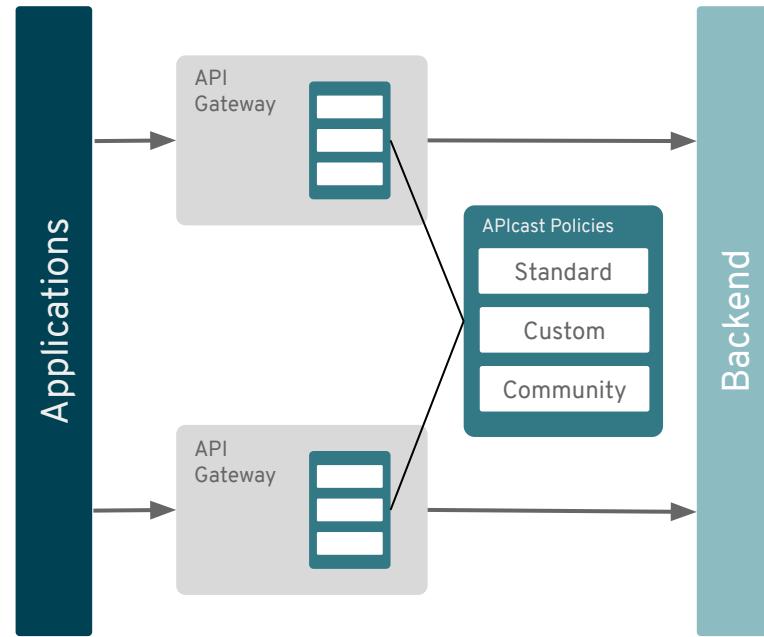
- ▶ Signed by algo and verified by only correct key
- ▶ Contains user identity in form of claims (Private, public, reserved)
- ▶ For OIDC purpose, SSO is widely adopted in consumer/enterprise apps
- ▶ Eliminates the need to look up against a central access control list



3SCALE API GATEWAY POLICIES

Modular Policy Architecture Benefits

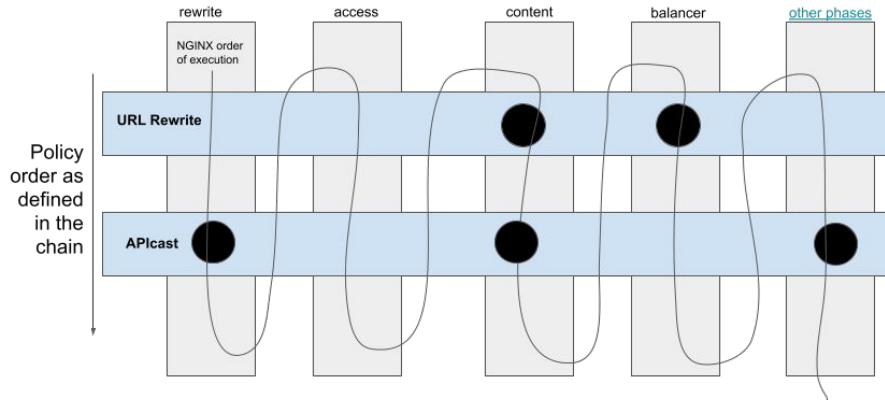
- ▶ Descriptive configuration, not code
- ▶ Add gateway logic with new policies for any phase of the request cycle
- ▶ Better extensibility
- ▶ Improved maintainability
- ▶ Leverage community contributions
- ▶ OOTB policies configurable from UI



Built-in Policies

- ▶ **OAuth Token Introspection:** Executes OAuth 2.0 token introspection for every API call
- ▶ **Upstream:** Allows modify the upstream URL of the request based on its path
- ▶ **Edge Limiting:** Algorithm-based rate limiting, allows global and per service caching
- ▶ **Url Rewriting:** Allows modification of a path request
- ▶ **SOAP:** Adds support for small subset of SOAP
- ▶ **CORS:** Enable Cross-Origin Resource Sharing
- ▶ **HTTP Headers:** Allows control of HTTP request and response headers
- ▶ **Echo:** Prints the request back to the client (status code optional)
- ▶ **Auth Caching:** Control 3scale authorization cache
- ▶ **Anonymous Access:** Provides default credentials for unauthenticated requests
- ▶ **Logging:** Enables / disables access logs per service
- ▶ **Referrer:** Sends the contents of the Referer HTTP header to backend so it can be validated
- ▶ **Metrics:** Enable backend metrics
- ▶ **Batcher:** Caches auth from backend and reports
- ▶ **IP Check:** Accepts or denies a request based on the IP

Policy Chaining



Notes:

Policies can define code for execution in any required phase.

Order of execution is as per the arrow, with processing happening at each phase:

- APIcast's operation in *rewrite* phase will be run BEFORE URL Rewrite operation in *content* phase
- APIcast's operation in *content* phase will be run AFTER URL Rewrite operation in *content* phase.

Custom Policies - in depth

[Link 1](#)

DROP Unwelcome Requests - Examples

Gateway Operations

- ▶ Checks the timestamp for 'expired' token.
- ▶ Checks the client_id is still valid
- ▶ Performs a check on the signature of the JWT using RH SSO public key

DROP Unwelcome Requests - Examples

TLS Strategies (3Scale 2.7+)

- ▶ Passthrough
- ▶ TLS Termination
- ▶ TLS Forward (Re-Encrypt)

TLS Termination Proxy



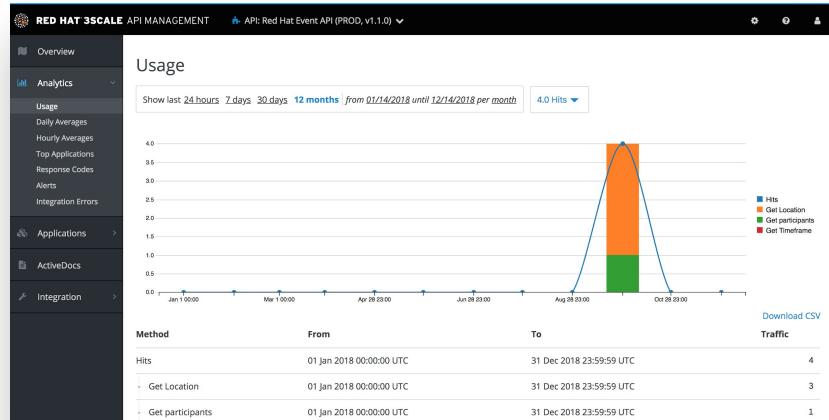
TLS Forward Proxy



ANALYTICS & Alerts

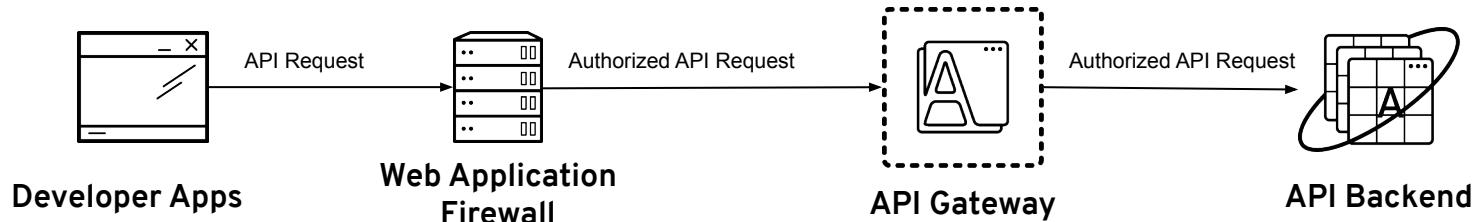
Measure the success of your APIs. Take actions based on numbers

- ▶ Define and present tailored metrics
- ▶ Drill down up to the API Method level
- ▶ Implement business metrics
- ▶ Export data in CSV format



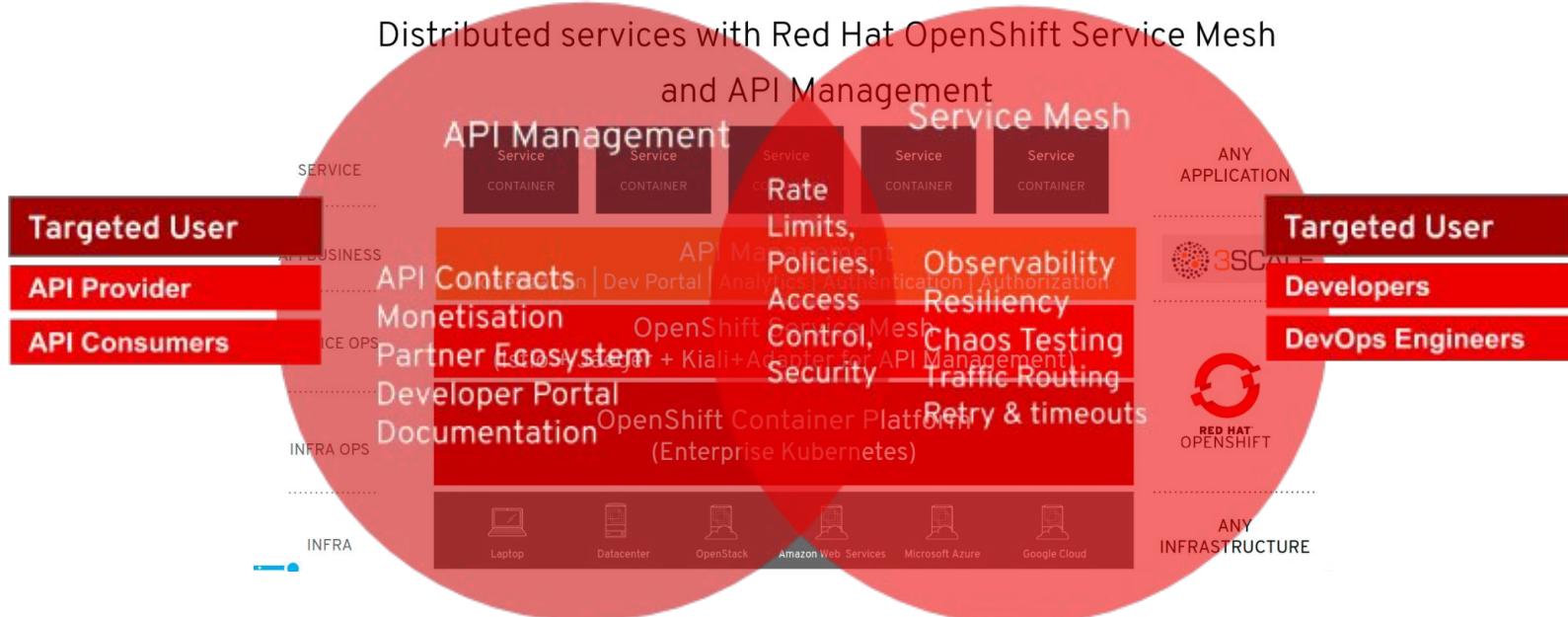
ADVANCED SECURITY

A Web Application Firewall can be used to enhance the security of your APIs



ADVANCED SECURITY

API Management vs. Service Mesh



Automation

Automation Options

- ▶ Pipelines
 - 3Scale Management API
- ▶ 3Scale Toolbox
 - Easier to implement
- ▶ **New!** YAML-oriented management via 3Scale Operator & GitOps (v2.9-2.10)
 - [3Scale-Operator Github Project & RH 3Scale Official documentation](#)
 - [Github complete example](#)
 - [APIManager CRD Reference](#)
 - [Backend CRD Reference](#)
 - [Product CRD Reference](#)
 - [Tenant CRD Reference](#)
 - [Video Demo](#)

External Access to 3Scale (API Calls || Operator)

[Medium Article - Example Demo](#)

```
apiVersion: v1
kind: Secret
metadata:
  name: main-three-scale-tenant #admin
type: Opaque
stringData:
  adminURL: https://3scale-admin.example.com:443
  token: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

```
apiVersion: v1
kind: Secret
metadata:
  name: mytenant
type: Opaque
stringData:
  adminURL: https://my-3scale-tenant.example.com:443
  token: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

3Scale toolbox

- ▶ 3scale toolbox is a set of tools to help you manage your 3scale product.
- ▶ [Official Project Git Repository](#)
- ▶ [Official Red Hat documentation](#)
- ▶ Can be installed on laptop or run on podman/docker container

3Scale Management API

Why work hard when you can automate?

Just do the [Exercise](#) - it covers everything
(requires jq)

Any Questions?

tamber@redhat.com

<https://www.linkedin.com/in/tommeramber/>

<https://medium.com/@tamber>

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.



[linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)



[youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)



[facebook.com/redhatinc](https://www.facebook.com/redhatinc)



twitter.com/RedHat