Enterprise API Management

4 May 2018

Vijay Narayanan
Introducing Application Programming Interfaces (APIs)

Typical Enterprise Level Challenges

API Management Core Tenets

API Contracts

API Catalog

API Gateway

Design Patterns

Conclusion
Wikipedia definition:
“An application programming interface (API) is a set of subroutine definitions, protocols, and tools for building application software. In general terms, it is a set of clearly defined methods of communication between various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer.”

- APIs provide a set of programmatic primitives that can be used in countless new ways to explore opportunities, solve problems

- APIs must be identified, managed, improved, and monitored on a continuous basis to honor Service Level Objectives (SLOs) & Service Level Agreements (SLAs)

- For the purpose of this session – APIs are REST based & accessed over HTTP / HTTPS
What are some common API Management pain points?

- **Internal & External facing APIs developed & managed in fundamentally different ways…**
  - APIs created & managed via multiple technology stacks (Node JS, Spring Boot,)
  - APIs have differences in authentication, authorization, telemetry capture, versioning, scalability levers,…

- **APIs aren’t always managed as a product**
  - Fuzzy definition. Often afterthought, once app is designed. Not always in an authority store
  - Lifecycle management, versioning, migration, testing, etc. are team or tech stack specific

- **API Integrations are point-to-point that force O(n) solutions and increase cost of ownership**
  - Inconsistent controls (e.g. on API contract, security, etc.) that result in high cost
  - Inadequate hooks to measure/improve latency, availability, operability, & scalability

- **Service Level Objectives (SLOs) – e.g. latency, availability, etc. – not always machine readable**
  - Unspecified SLOs result in tribal knowledge & high operational risk
  - Lack of SLOs hurt data driven API evolution – *how do you know if less latency is what clients care about?*

- **Publisher & Consumer tooling gaps vis-à-vis API lifecycle management**
  - Publishers lack tooling for definitions, version management, etc.
  - Consumers lack tooling for discovery, evaluation, SDK generation, etc.
API Platform – Key Elements

Accounting for enterprise, provider, & consumer needs

API Management Elements

1. API Contracts
   Model & Provision APIs as infrastructure-as-code

2. API-as-a-Product
   Manage & Monetize APIs throughout lifecycle

3. API Catalog
   Discover, Evaluate, & Consume APIs

4. API Gateway
   Mediate, Scale, & Secure APIs at runtime

5. Automated Controls
   Monitor, heal, & contain faults to reduce risk

6. API Developer Tools
   Tools to drive developer productivity
API Management Platform must:

1. Adhere to enterprise SDLC standards, processes, & tools
2. Be language and implementation agnostic
3. Be opinionated to drive tenant isolation, scalability, & automation
4. Enforce Information Security requirements
5. Make API definitions discoverable
6. Facilitate API version upgrades & downgrades
7. Availability as a first-class citizen – report, track, & surface availability metrics
8. Surface security, performance, & operational risks
9. Work with internal and/or external cloud providers
10. Reduce blast radius of ill-behaved publishers & consumers
Open-API Specification – roots in the Swagger stack, used heavily by existing customers, tooling ecosystem getting richer

API Contracts authored or generated from IDE and checked into source control

API Contract descriptors provisioned using Infrastructure-as-Code (IaC) provider

Build provisioning time checks and controls to prevent defects at runtime

Post-provisioning lifecycle events published for custom event handling (analytics, risk assessment, controls, legacy integrations)
openapi: "3.0.0"

info:
  version: 1.0.0
  title: Swagger Petstore
  license:
    name: MIT

servers:
  - url: http://petstore.swagger.io/v1

paths:
  /pets:
    get:
      operationId: listPets
      parameters:
        - name: limit
          in: query
          description: How many items to return at one time (max 100)
          required: false
          schema:
            type: integer
            format: int32

responses:
  '200':
    description: An paged array of pets
    headers:
      x-next:
        description: A link to the next page of responses
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/Pets"

Complete examples at: https://github.com/OAI/OpenAPI-Specification/tree/master/examples/v3.0
API Catalog

- Be an authoritative store for API contracts that conform to an open specification

- Surface non-functional Quality of Service attributes of API resources (endpoints)

- Be full-text searchable across both contract definitions and associated metadata

- Support API definition and real-time invocations (for demos / proof of concepts)

- Populated and curated in context of runtime changes to infrastructure

- Surface API versions available and key features

- Support Account Management – i.e. link consumer subscriptions & usage across APIs
**Functionality:**
- Searchable API Catalog for both API publishers & consumers
- Multi-tenant and support custom request and response filters
- Infra-as-Code semantics to define, integrate, & test API contracts
- Self-Service tooling for API Publishers, Consumers, & Admins
- First-class support for API lifecycle & versioning

**Technology:**
- Platform, language, & stack agnostic
- Leverage open-source API Management solutions
- Non-blocking and optimized for minimal latency overhead at runtime
- Will be integrated with firm-specific platform technologies
- Type-safe & IDE friendly abstractions
Traffic Capture & Replay for Safer Upgrades

Using the gateway for graceful versioning

- Deploy multiple versions in production
- Capture trace (request & response)
- Replay traffic against new version
- Compare existing response with new response
- Cut over gracefully based on comparison results
Deploy multiple versions in production

Distribute traffic based on weights across versions

Dial up traffic to new version from 0-100%

Monitor for errors to determine if new version is OK

Ensure adequate capacity when rolling back!
Dealing with realities of multi-tenancy

- Consumers can either due to defects or manual errors trigger high volume of requests during their development / SDLC
- Non-prod traffic is second priority – segregated resources thus can be reassigned to deal with spikes in production traffic
- Independent instances will reduce blast radius, rogue client behavior, etc
Make things possible.
Disclaimer:
Views and opinions expressed are for informational purposes only. They do not constitute a recommendation by Goldman Sachs to buy, sell or hold any security. No part of this material may, without Goldman Sachs’s prior written consent, be (i) copied, photocopied or duplicated in any form, by any means, or (ii) distributed to any person that is an employee, officer, director, or authorized agent of the recipient.

Copyright Note: 2018 Goldman Sachs. All rights reserved.
GREAT INDIAN DEVELOPER SUMMIT 2019

Conference: April 23-26, Bangalore

Register early and get the best discounts!

www.developersummit.com  @greatindiandev  bit.ly/gidslinkedin