
ORACLE®

API 设计，开发，治理与交付

Oracle Code

ORACLE CODE

developer.oracle.com

Kenneth Heung 香瑞鸿
Senior Principal Architect
Asia Pacific Cloud Pursuit
July, 2017

Live for the Code

ORACLE

免责声明 Safe Harbor Statement

以下内容旨在概述产品的总体发展方向。该内容仅供参考，不可纳入任何合同。本演示不承诺提供任何材料、代码或功能，也不应将其作为购买决策的依据。此处所述有关 Oracle 产品的任何特性或功能的开发、发布和时间安排均由 Oracle 自行决定。

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

议程

1. API 经济论 - 这一节假设
2. API 生命周期
3. API 为先 - 设计为先 - 代码为先
4. API 设计 - 发布 - 安全 - 版本

场景设定

- API 的应用已戏剧性地改变
- API 已经成为创新的核心 – 直接影响业务增长和创新
- 增长 + 创新 = 盈利能力
- API 在微服务中广泛地应用

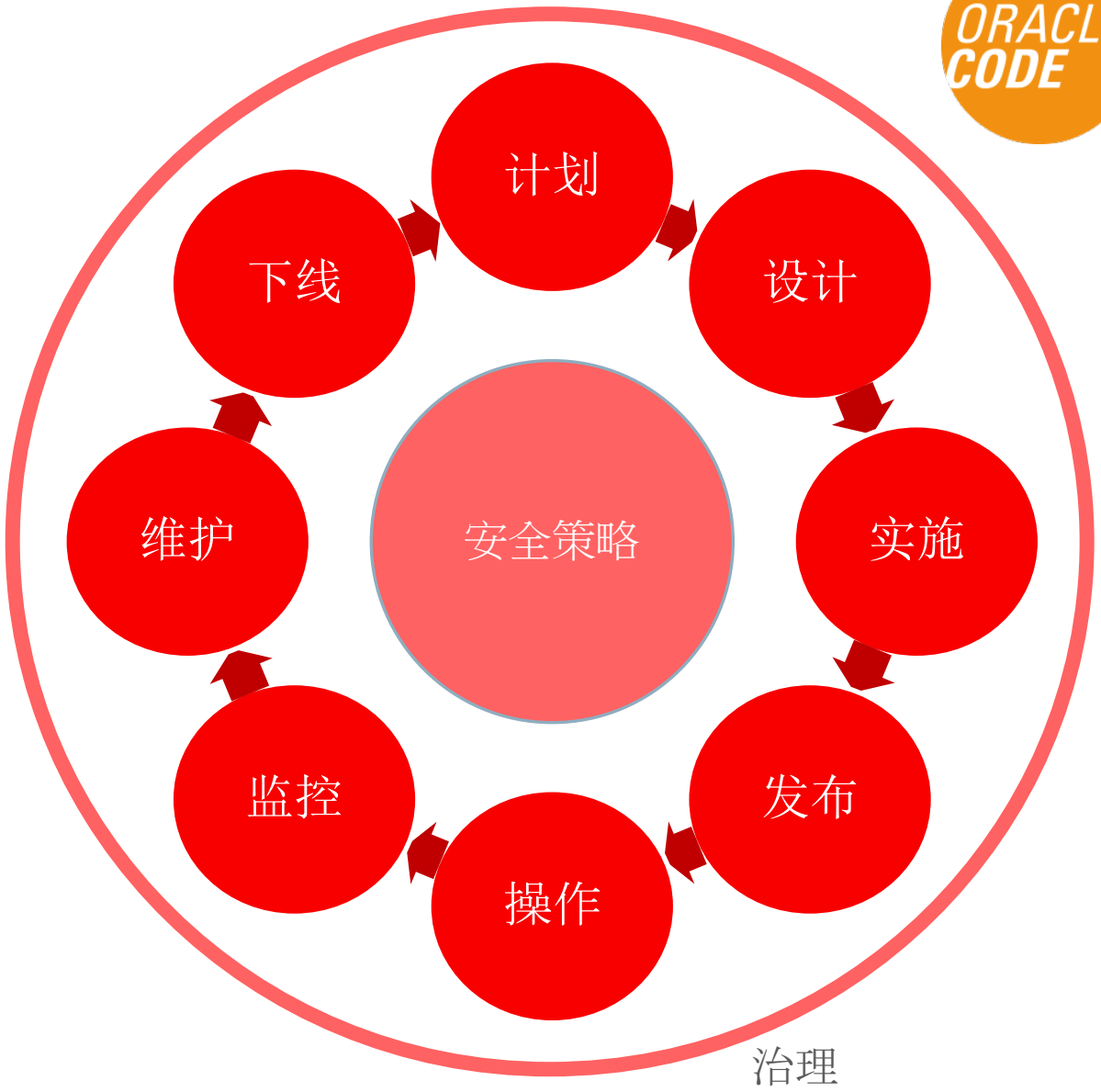
API 经济的 4 大公理

- Fast is better than Slow - 敏捷开发不单只是是一种管理风格 – 更小更快，比又大又慢更好。
- Developers 已成为公司的关键技术决策者，开发者在业务发展方向拥有更大的影响力 - 关注开发者体验 Developer Experience **DX**
- Simplicity – KISS – Keep It Simple, Stupid
- Agile crushes waterfall 一个应用程序需要在一两个月内完成，否则就是失败

Source: Jennifer Riggins
https://www.programmableweb.com/news/how-to-design-great-apis-api-first-design-and-raml/how-to/2015/07/10

API 生命周期管理

- 计划 - 准备 / 语义 / 架构 / 风格
- 设计 - 规范 / 原型 / 模拟服务
- 实施 - 代码开发 / 集成/ 部署 – CI/CD
- 发布 - 文档 / API 消费者-应用开发者 门户
- 操作 - API 消费者发现和使用 API **
- 监控 - +分析 / 反馈 / 迭代 / 性能监控 / 计费
- 维护 - +版本管理
- 下线



实践 API 为先（API-First)

- 计划 Plan - 在正式开始之前，必须决定 API 的目标和 API 真正的样子。定义应用程序域语义，决定 API 架构风格，并制作一个API风格指南 - 同时确保 API 的一致性
- 设计与验证 Design & Validate - 做好计划后进入设计阶段，检测 API 的可行性。即使 API 还没有实施，应该要有工具来制作 API 请求; 这使您便能够验证整个用例，查看 API 将会如何被使用。
- 规范与模拟 Specification & Mock Up - 锁定源代码。 根据您的计划和设计，订定 API 规格。 根据该 API 规范产生成规范文档 - 再通过用例来模拟 API (API mock-up) 。

实践 API 为先 (API-First)

- 测试 Test - 自动生成测试 - API 它就是一个很好的测试界面， API 规范文件方便自动化去测试 API 的所有功能 - 测试 API 中什么应该可以访问的内容，同时也可以测试 API 是否有良好的 DX 及一致性
- 实施 Implement - 让双方 (API 开发者和 API 消费者) 同时参与 - 有了双方的所有信息，我们将真正开始进入 API-First。API 消费者，特别是如 iOS/Android 客户端应用开发者了解如何在 API 之上构建自己的应用 。您甚至可以存储实现API所需的服务器框架的一些部分。
- 操作与建立关系 Operate and engage - 与「客户」接触，得到反馈。学习。 然后再重复。

API 为先 API First

- 从应用及用户开始 (application backward) - 问问自己：你的用户需要什么？ 为需求作适当设计。
- 专注于 干净 整洁 一致性 的 模型/建模
- 在实现 API 之前 确保 API 适用
- 为用户提供各种 工具 来 发现 和 使用 您的 API **

设计为先范式 design first paradigm?

设计为先

- 业务计划被转换为人和机器可读的规范文档，例如 OpenAPI / API Blueprint 文档，再开发代码或透过集成建构 API
- 相对比较着重开发者体验
- 确保良好沟通
- 相对比较适合外部目标受众

代码为先

- 比较传统的方法来构建API
- 基于业务计划，API被直接编码，再生成如 Swagger 文档的人或机器可读文档
- 相对比较快速交付
- 相对比较适合内部API

稍等一下...

Oracle Code 官网对本节的介绍....

API設計，開發，治理與交付

14:10 - 14:55 | 会议厅A (Conference Hall A)

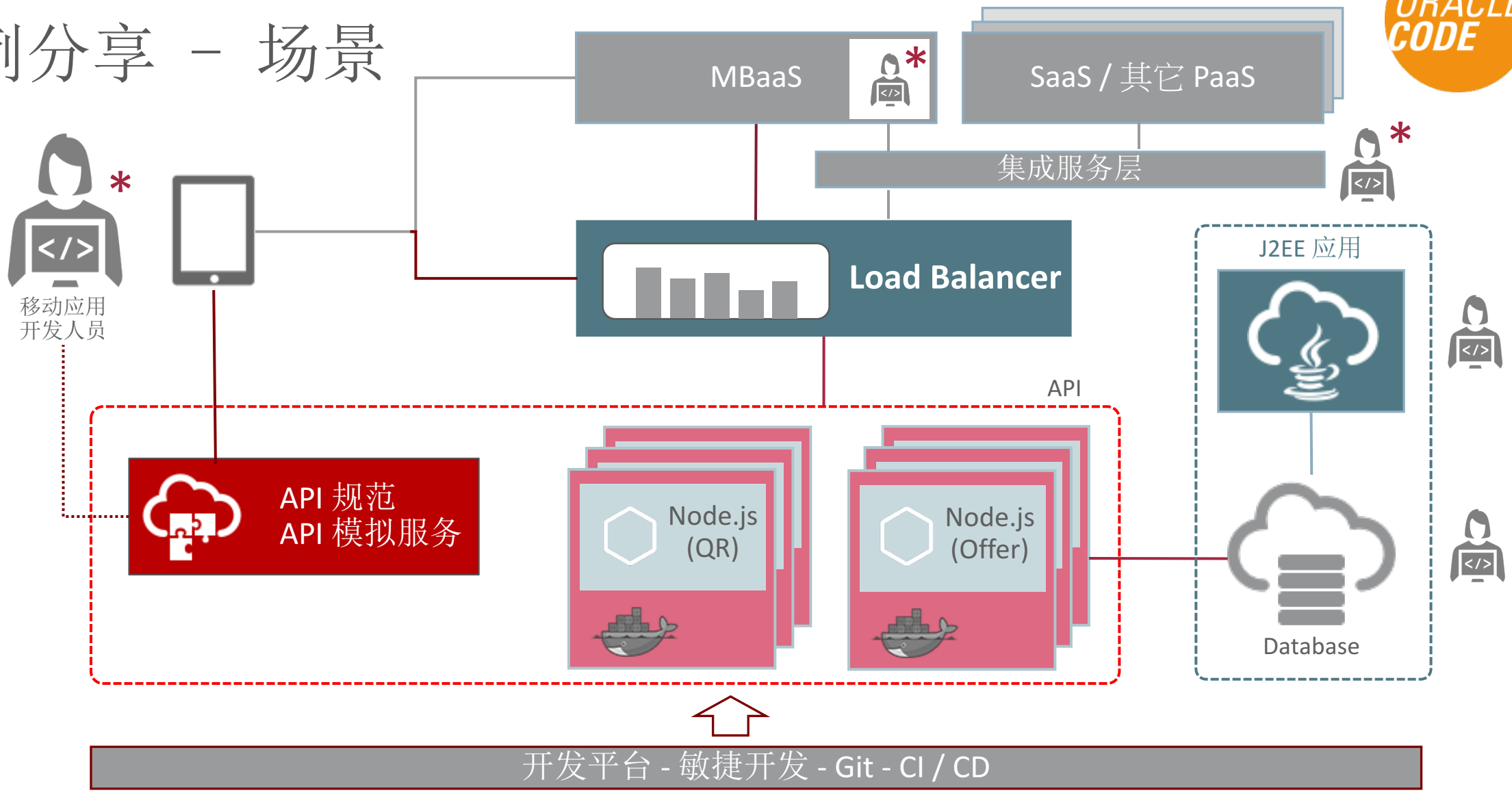
在软件开发中，我们会谈论TTD，RMDD等的开发范例。API开发中的基本概念又会什么？在API开发中，设计为先和开发为先两首之间哪个比较优胜？API经济是一个热门话题。互联网上有很多资源谈论RAML，Swagger，API Blueprint等。事实上，网友总是希望获得更多的信息和经验分享 - (1) 为什么需要API生命周期中实现协作？(2) 我们应该如何在实现协作？本节侧重于用例，并讨论API生命周期管理 - 从设计，实施，安全到管理和监控。

用例？

Kenneth Heung
高级首席架构师
Oracle

Kenneth 是 Oracle 高级首席架构师。他拥有 15 年以上的技术和项目经验，涉及软件开发、IT安全性、ITSM、中间件、SOA和企业集成等多个领域。Kenneth 于 2009 年加入 Oracle，目前专注 InfoSec 和 AppDev 领域。在加入信息技术行业之前，Kenneth 在一所高中担任了七年的计算机教师。

用例分享 – 场景



14

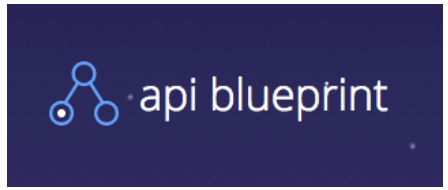
API 设计规范

- 通用的语言去描述 API 以便所有人都能理解
  - 人类可读和机器可读的
  - 无论对于开发人员和非开发人员来说，都是容易理解的
  - 易学 易读 易写
  - 容易调整 / 修改 - 以方便测试和调试API的问题
- 三种常用规范描述
  - OpenAPI https://www.openapis.org/
  - RAML https://raml.org/
  - API Blueprint https://apiblueprint.org

API 常用规范文档

- RESTful API Modeling Language
- 基于 YAML 为蓝本
- RAML 为 MuleSoft, Inc.的注册商标

- 最初为 Swagger
- 从 01-01-2016 开始为 OpenAPI
- JSON and YAML
- 很受欢迎 **



- 由 Apiary 推出
- Markdown 格式
- 以消费者为中心的

RAML / OpenAPI (Swagger) / API Blueprint 例子

```
swagger: "2.0”
info:
  version: 1.0.0
  title: "Cloud Test Drive Microservice Points Management"
  description: Points Management API
host: offer-gse00000000.apaas.em3.oraclecloud.com
basePath: /ptmgt/v1
schemes:
  - https
consumes:
  - application/json
produces:
  - application/json
paths:
  /offers/{id}:
    get:
      description: Returns an offer based on a single ID
      operationId: find an offer by ID
      parameters:
        - name: id
          in: path
          description: ID of an offer to fetch
          required: true
```

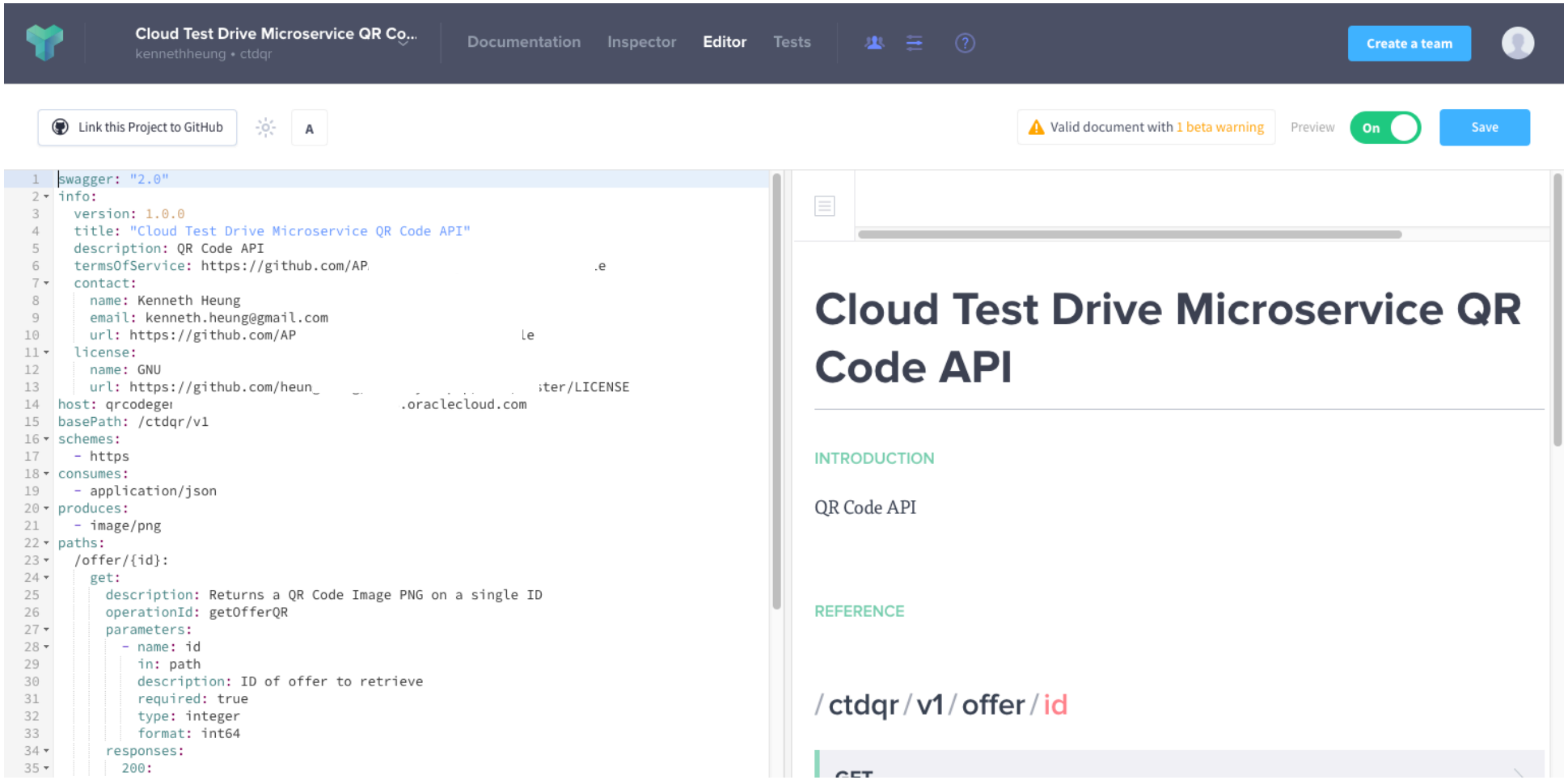
```
{
  "swagger": "2.0",
  "info": {
    "version": "1.0.0",
    "title": "Cloud Test Drive Microservice Points Management",
    "description": "Points Management API",
  },
  "host": "offer-gse00000000.apaas.em3.oraclecloud.com",
  "basePath": "/ptmgt/v1",
  "schemes": [
    "https"
  ],
  "consumes": [
    "application/json"
  ],
  "produces": [
    "application/json"
  ],
  "paths": {
    "/offers/{id}": {
      "get": {
        "description": "Returns an offer based on a single ID",
        "summary": "find an offer by ID",
        "operationId": "find an offer by ID",
        "produces": [
```

RAML / OpenAPI (Swagger) / API Blueprint 例子

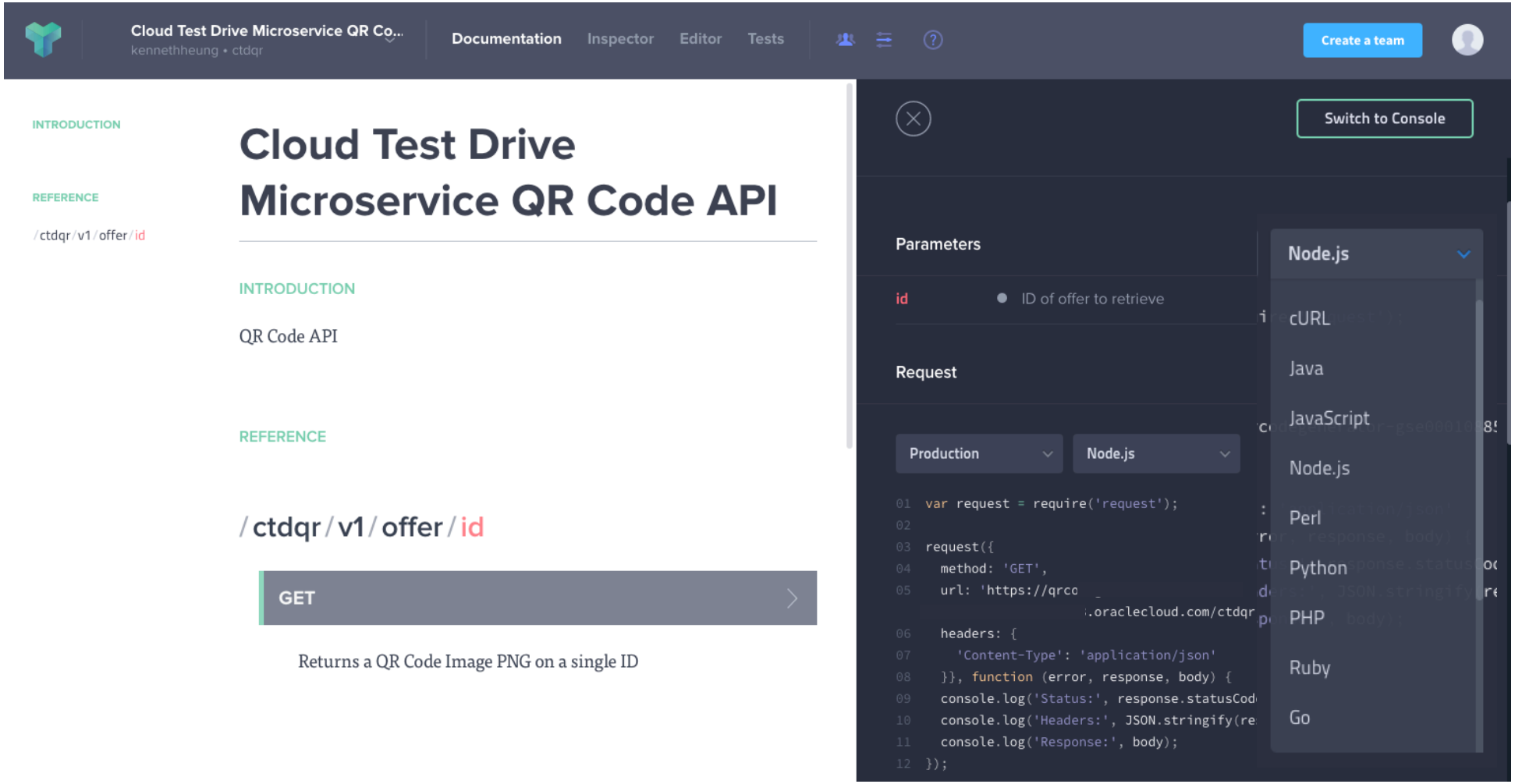
```
#%RAML 1.0
title: Cloud Test Drive Microservice Points Management
version: 1.0
baseUri: https://offer-gse00000000.apaas.em3.oraclecloud.com/ptmgt/v1
baseUriParamaters: {}
documentation:
- title: Cloud Test Drive Microservice Points Management
  content: 'TODO: Add a description'
types:
  Customer:
    displayName: Customer
    type: object


/offers/{id}:
  uriParameters:
    id:
      required: true
      displayName: id
      description: ID of an offer to fetch
      type: integer
      format: int64
  get:
    displayName: find an offer by ID
    description: Returns an offer based on a single ID
```

```
FORMAT: 1A
HOST: https://offer-gse00000000.apaas.em3.oraclecloud.com/ptmgt/v1

# Cloud Test Drive Microservice Points Management
Points Management API

# Offers By Id [/offers/{id}]

+ Parameters
    + id (number, required)

        ID of an offer to fetch{LONG}


## find an offer by ID [GET]
Returns an offer based on a single ID

+ Response 200 (application/json)

        offer response

    + Attributes (Offer)
```

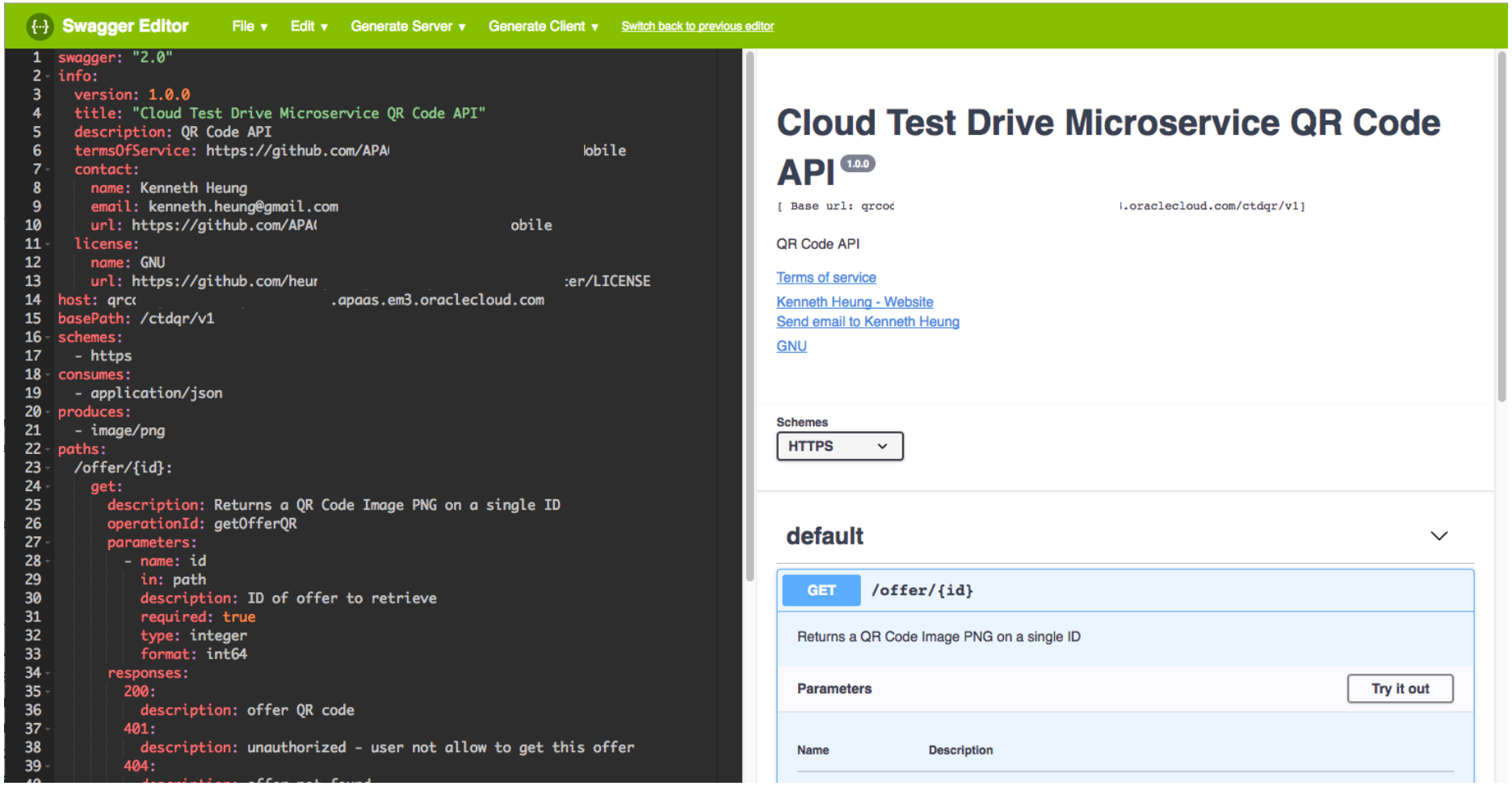
RAML / OpenAPI (Swagger) / API Blueprint 例子



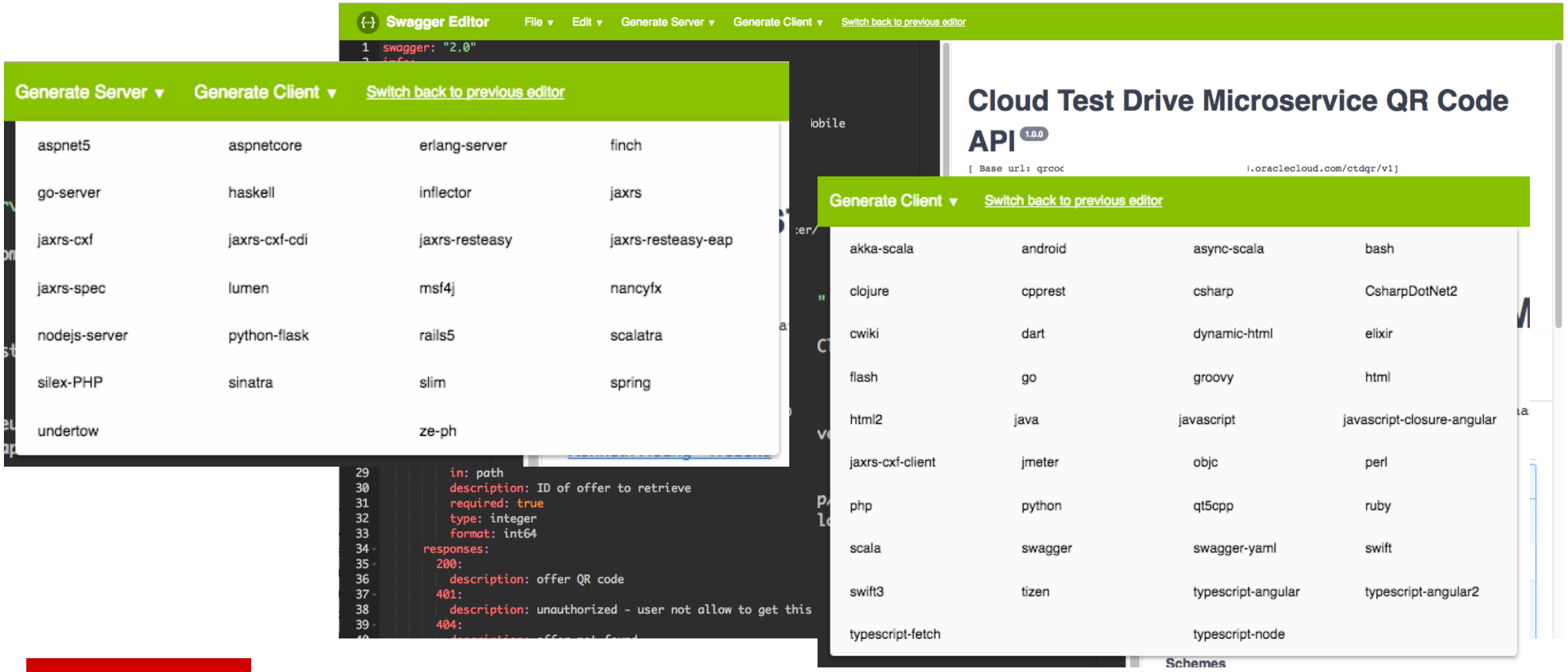
RAML / OpenAPI (Swagger) / API Blueprint 例子



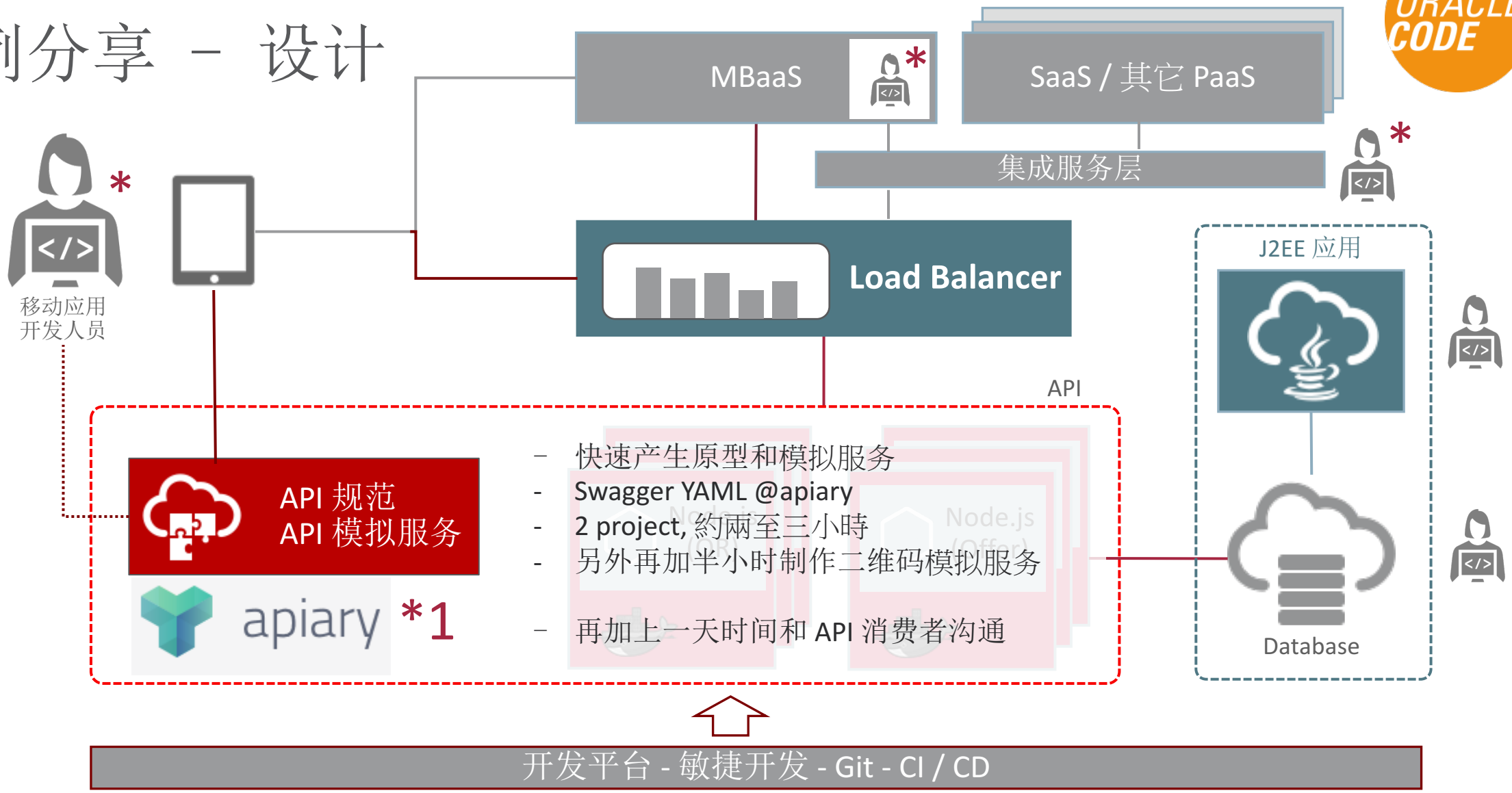
RAML / OpenAPI (Swagger) / API Blueprint 例子



RAML / OpenAPI (Swagger) / API Blueprint 例子



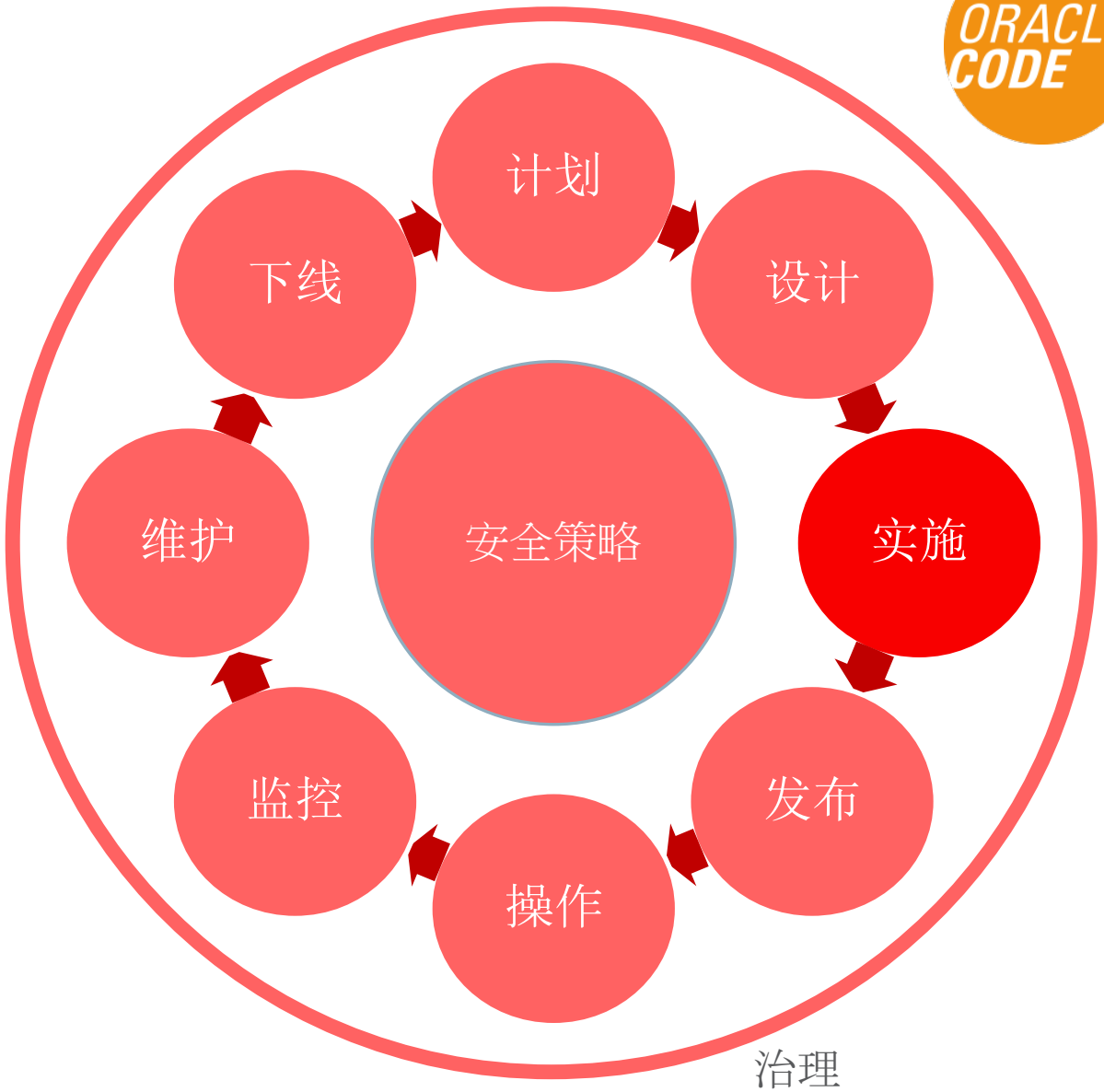
用例分享 – 设计



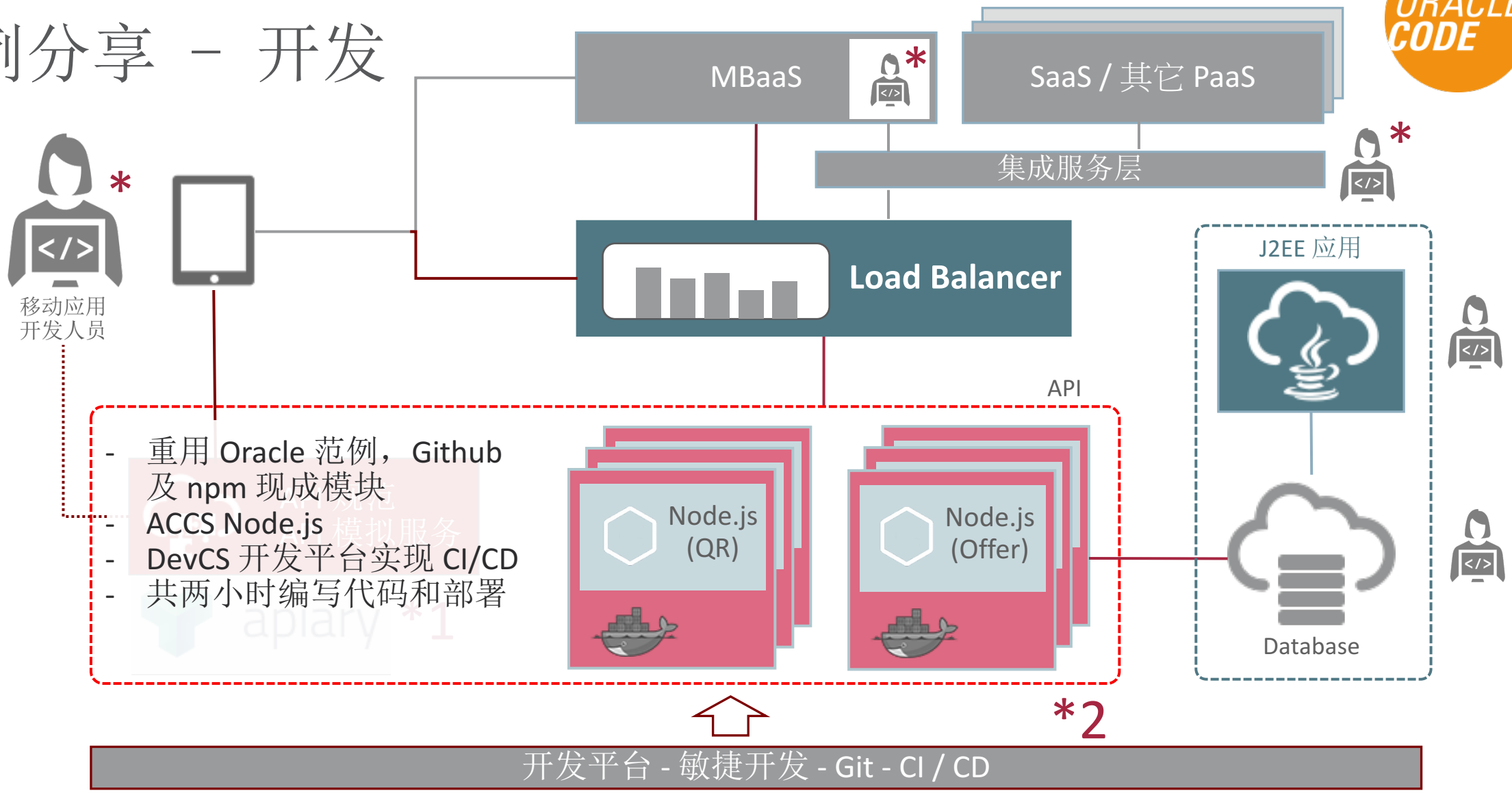
23

API 开发 / 实施

- REUSE 重用
  REUSE 重用
  REUSE 重用
- 微服务 – 多语言
- 敏捷开发
- CI/CD



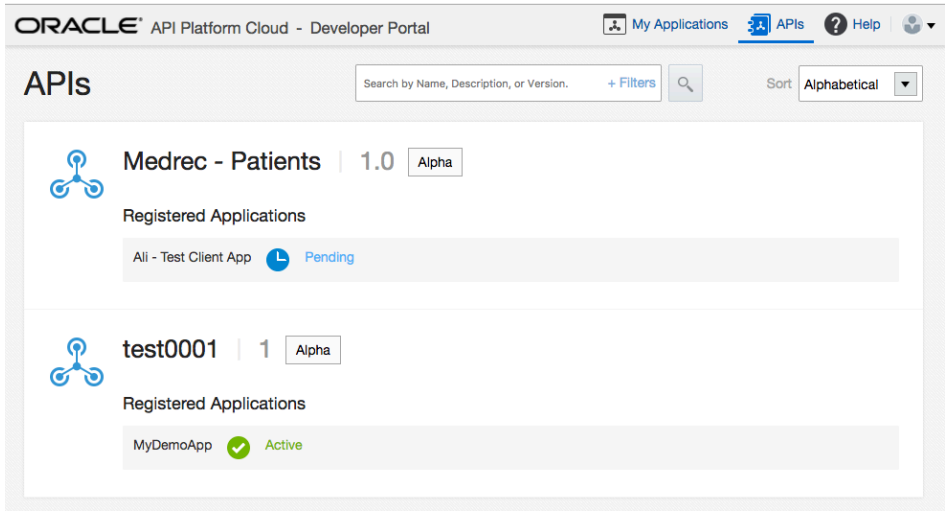
用例分享 – 开发

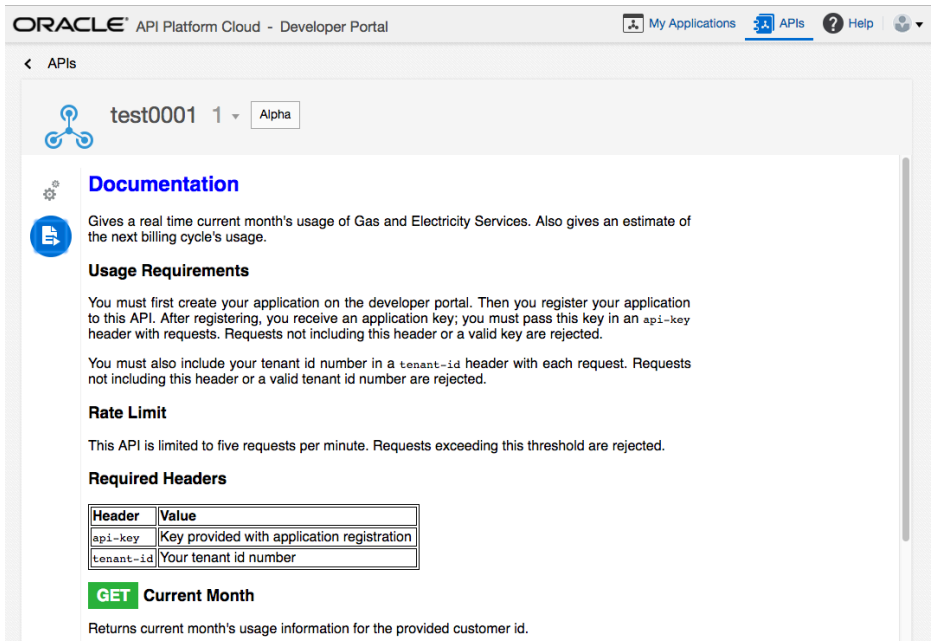


25

API 发布

- 发布规范文档 / API 使用文档
- API - 对内 / 对外?
- API 消费者门户网站 / 应用开发者门户网站



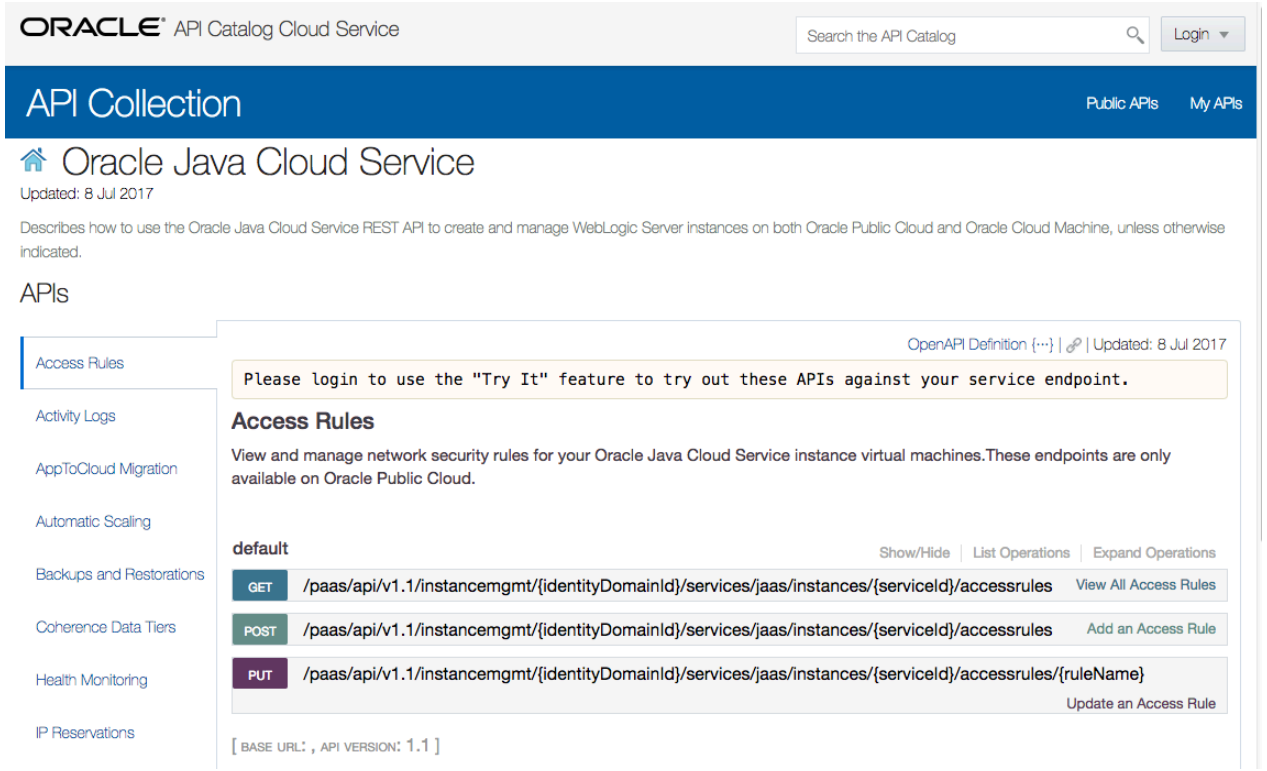


ORACLE CODE

API 门户网站

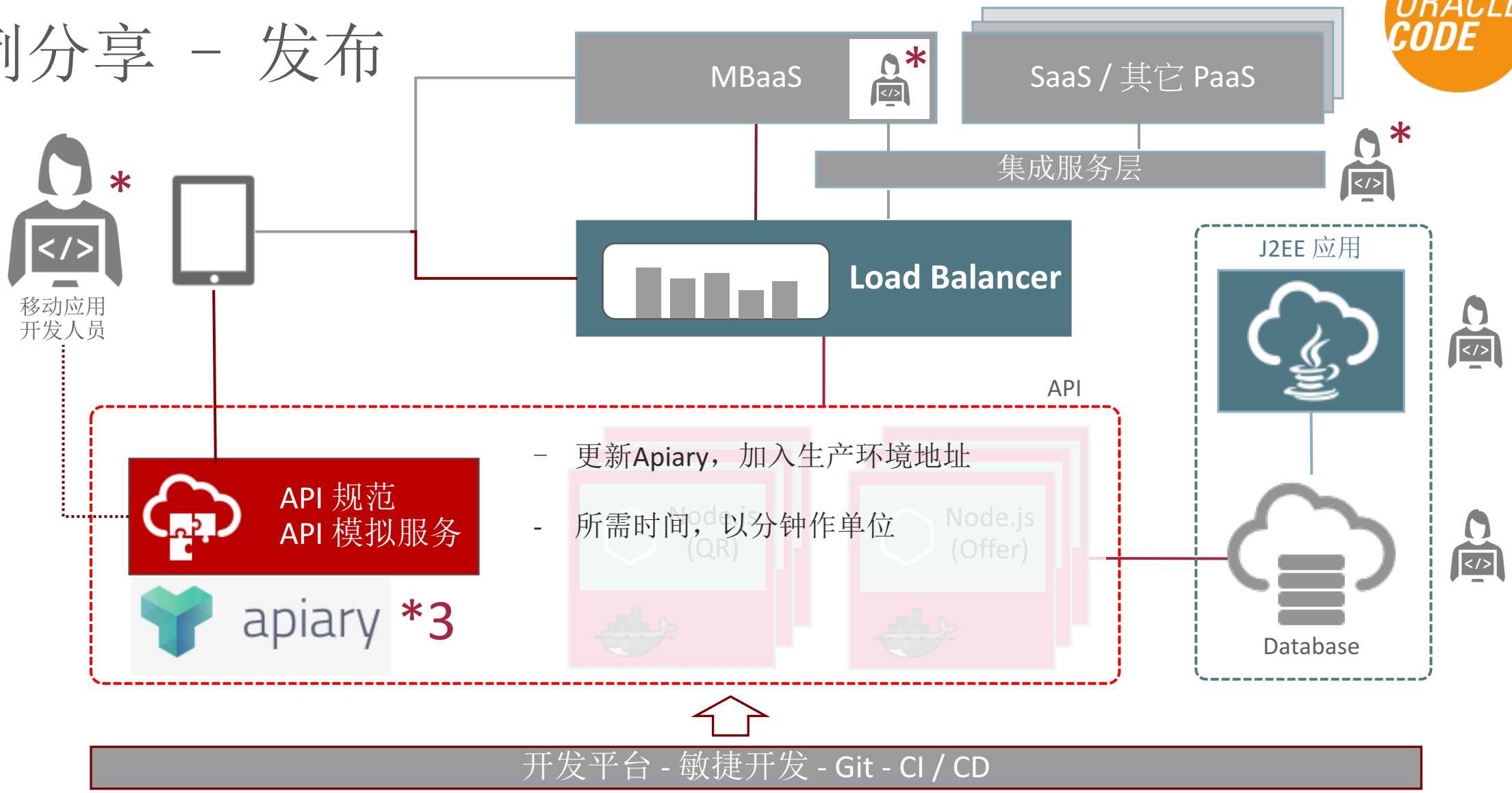
- API 目录 / 菜单/ 搜索
- 查看 API 详细信息 / 文档 / 收费 / 费率限制
- 查看 API 使用范例
- 测试 API / 自动生产源代码
- 注册使用 API / 生产 API 密钥
- 查看应用程序的分析使用

- 目标：开发者体验



ORACLE

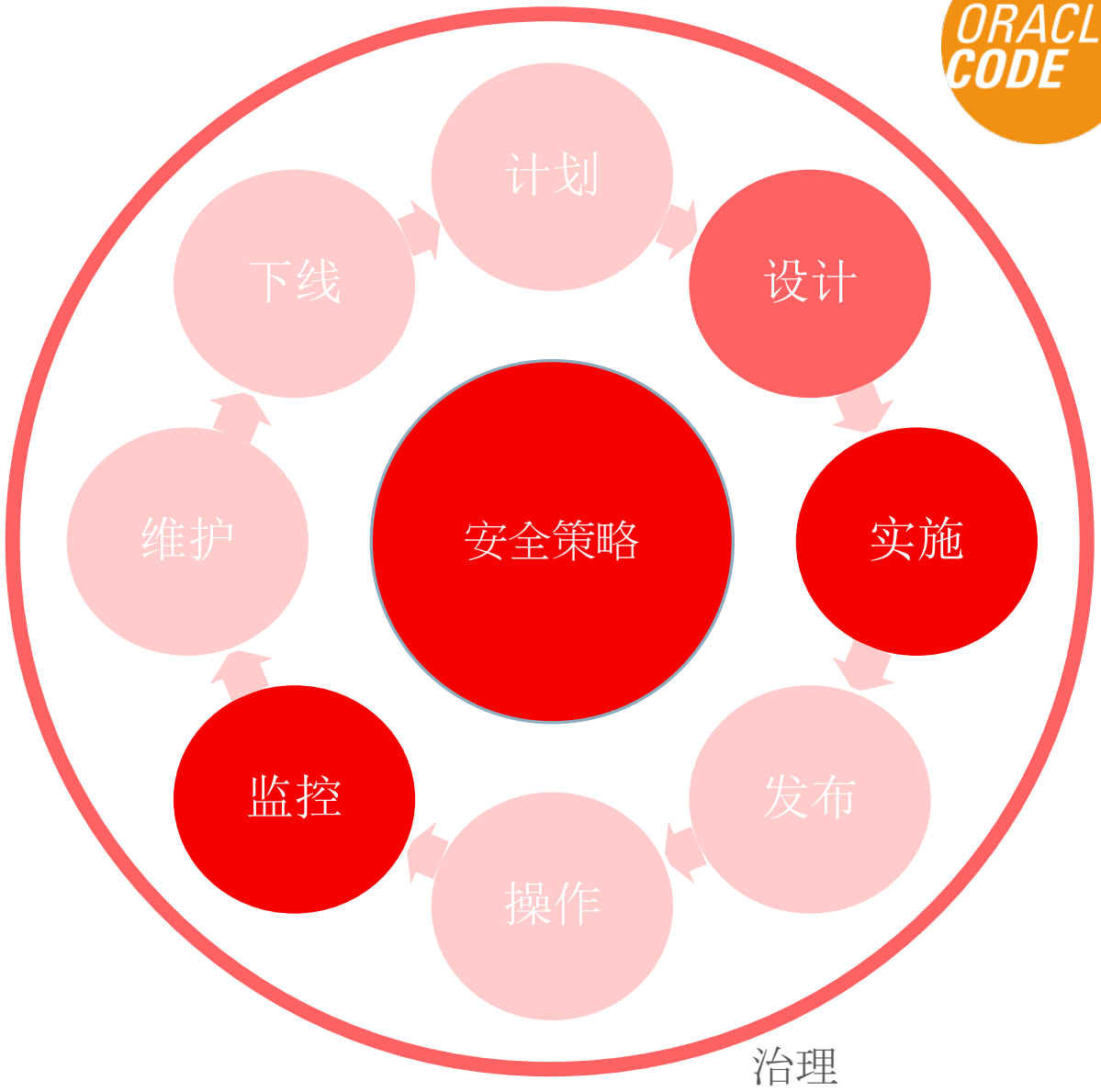
用例分享 – 发布



28

API 安全模型 / 安全策略

- 减少 API 开发人员所需的工作
  - 外部化认证和授权
  - 安全策略不应嵌入在源代码中，而是通过配置实现
- API 开发人员仍然需要实践安全编码的最佳做法
  - 采用正式的安全编码标准
  - 编写代码时已经考虑 OWASP 十大关键应用安全风险
- 使用 API 应用网关来拦截安全攻击 (如SQL注入攻击，代码注入攻击等...)
- 使用监控工具来检测 (高级持续性) 威胁



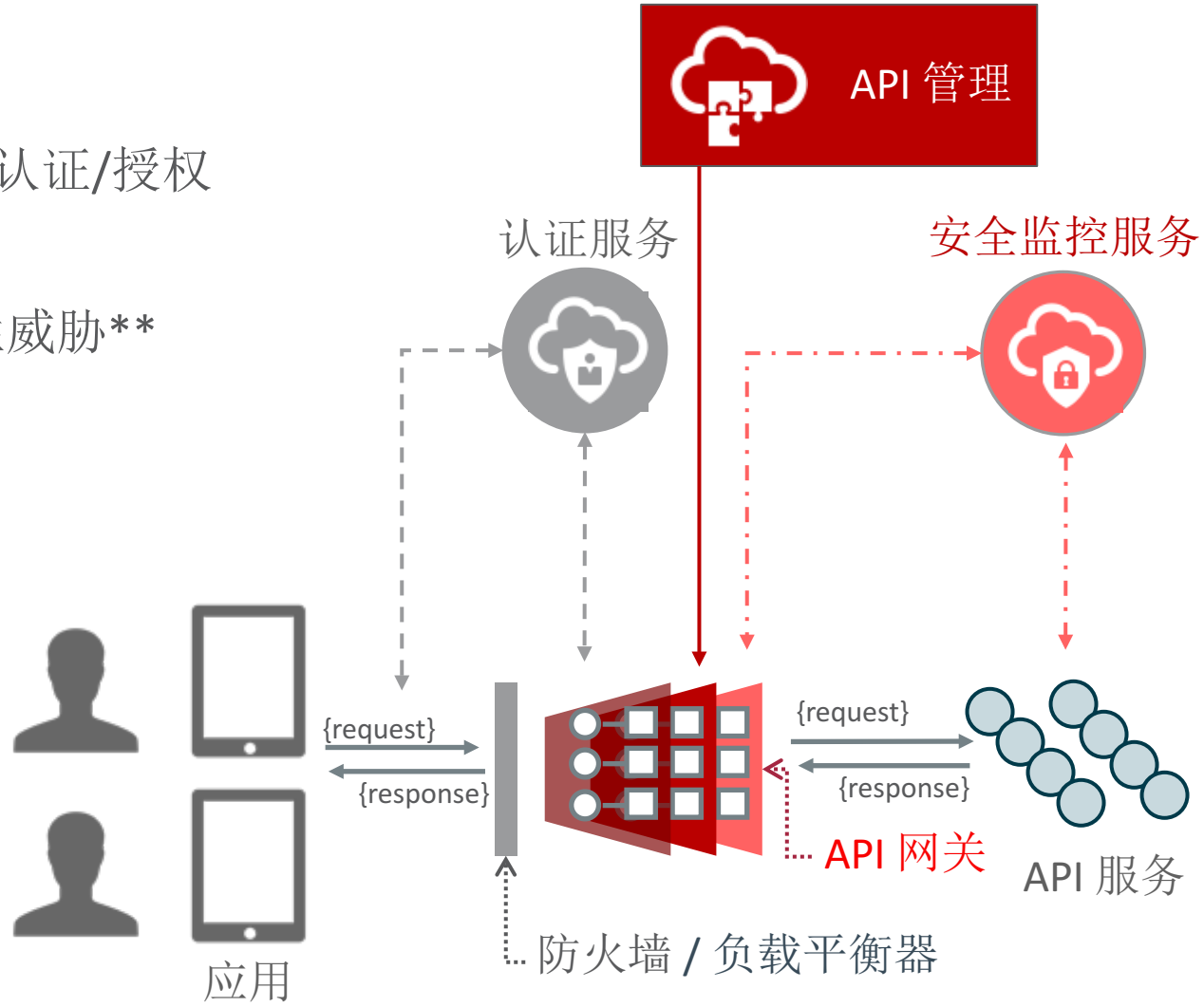
API 安全模型 / 安全策略

- 范例
  - 网关层实现 OAuth2 / OpenID Connect 认证/授权
  - 网关层拦截传统安全攻击
  - 安全监控和分析检测威胁/高级持续性威胁** (Advanced Persistent Threats)

** 高级持续性威胁包含三个要素：高级、长期、威胁。

- 高级强调的是使用复杂精密的恶意软件及技术以利用系统中的漏洞。
- 长期暗指某个外部力量会持续监控特定目标，并从其获取数据。
- 威胁则指人为参与策划的攻击。

高级持续性威胁跟传统攻击不同的地方，主要在于活动十分隐蔽，需要深入的日志分析和比对，从正常流量中分离出异常流量，在该威胁完成任务，窃取数据前检测出来，以便作出相关对应



API 版本

- URI

  curl -X GET http://mydomain.com/api/v1/myapi
  curl -X GET http://mydomain.com/api/v2/myapi

- Query Parameter

  curl -X GET http://mydomain.com/api/myapi?v=1
  curl -X GET http://mydomain.com/api/myapi?v=2

- Hostname

  curl -X GET http://api-v1.mydomain.com/api/myapi
  curl -X GET http://api-v2.mydomain.com/api/myapi

API 版本

- Custom Header

```
curl -X GET -H “X-API-VERSION: 1” \
        http://mydomain.com/api/myapi

curl -X GET -H “X-API-VERSION: 2” \
        http://mydomain.com/api/myapi
```

- Content-Type

```
curl -X GET -H “Accept: application/vnd.myapi.v1+json” \
        http://mydomain.com/api/myapi

curl -X GET -H “Accept: application/vnd.myapi.v2+json” \
        http://mydomain.com/api/myapi
```

API 版本策略

	URI	Query Parameter	Hostname	Custom Header	Media Type
流行度	高	高	低	中	中
例子	Oracle, Twitter, 百度， 腾讯, 微博	Microsoft, Google, AWS.CN, 淘宝	** Facebook	Microsoft, 阿里云	GitHub
		这主要针对 JavaScript API	适合写一些完全不同 API		最接近原始的 RESTful规范
优点	视觉识别	1.视觉识别 2.可以选择性	视觉识别	1.保留原有URI 2.比URI版本法整齐	保留原有URI
缺点	破坏了RESTful合规性	容易混乱资源版本与API版本	破坏开发者体验	较难测试	1.较难测试 2.扭曲HTTP Header 的目的

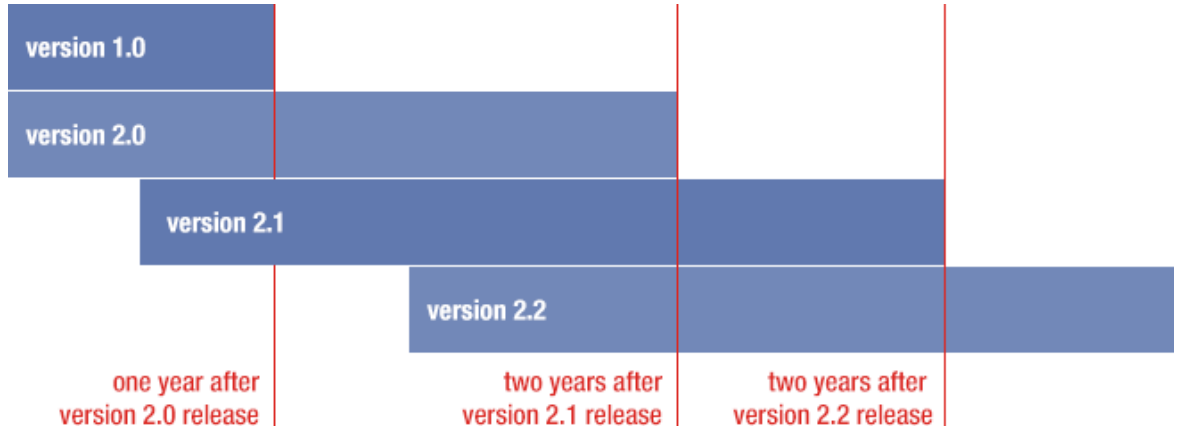
API 版本策略

- 时间表和策略

一旦版本不再可用，对其进行的任何调用将被默认为下一个最旧的可用版本。

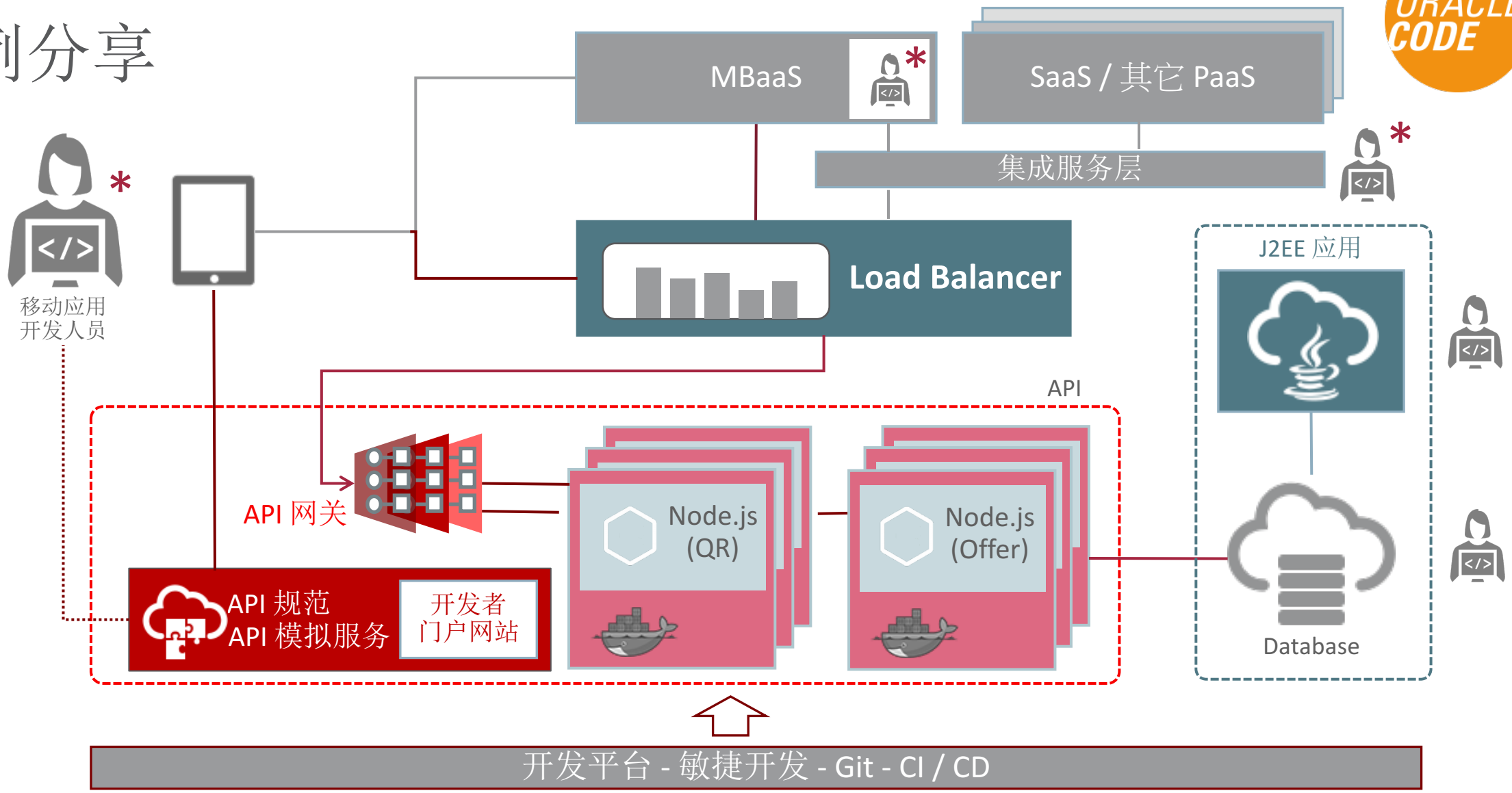
未版本化的呼叫将默认为最旧的可用版本

未版本化的呼叫将默认为最新的可用版本



Source: https://developers.facebook.com/docs/apps/versions

用例分享



35


ORACLE®