AUTOMATION THROUGH APIOPS

API Ops as a next level of automation that includes DevOps and GitOps

Abstract

The paper discusses the API Ops and advantages of it. It also shows how Kong provides the necessary tools to implement APIOps strategy for end-to-end API Management

Pradeep Tallogu

Prowess Software Services

Table of Contents

Introduction

Applications, by nature of its very existence, need constant changes and updates to them to stay relevant, as user needs evolve. More so, for the applications that are used the most. For any changes to applications, the lifecycle invariably consists of working toward design changes, the development work, the testing process, and deployment.

An application goes through certain milestones before it can be deployed into production. Sometimes application development involves starting from scratch, other times it may come as enhancements. In both cases, there is a set of implementation milestones that may be broadly viewed as below

- Design
- Development
- Testing
- Deployment

The goal of this paper is to showcase the opportunities for automation at each of these phases and minimizing the total amount of effort and time to navigate from end-to-end development lifecycle.

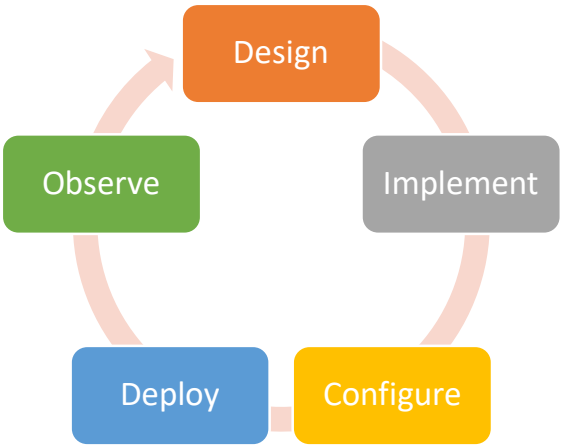
The paper also showcases how this process may be applied to an application that exposes RESTful APIs to consumers.

Achieving quality and scale through APIOps

Many organizations today find themselves in a greater need of APIs than ever before. The process of designing, building, testing, and deployment may involve a combination of manual and automated activities. The goal of DevOps, (or GitOps, or APIOps) is to leverage the use of automation tools as much as possible, thereby improving efficiency and productivity.

DevOps has emerged as the go-to strategy in optimizing the way Developer and IT Operations activities are automated. DevOps is a process that combines the tasks of Developer and IT Operations into a unified process that can be efficient in delivering high-quality application code at high velocity. Before DevOps, deployments depended on a lot of handshakes between developers and the operations teams. The process contained manual activities, reconciliation, that needed a functional subject matter expert, and were prone to human-errors. It also meant that scaling-up for larger number of deployments was difficult.

GitOps is a practice of leveraging the operations performed on GIT repository to trigger the processes defined as part of DevOps. GitOps considers Git as the source of truth for declarative infrastructure and applications.



Lifecycle of an API

The lifecycle of APIs includes a series of milestones

- Design of API Specification
- Implementation of API
- Configure
- Deployment
- Observe

An increasing demand for quality and scale could only be achieved through considerable amount of automation across the various milestones above.

Design of API Specification

API specification is one of the first artifacts that would be developed as part of design-first approach, before the implementation would start. There are multiple considerations for developing the specification

- Open Standards
- In-built validation
- Mock service
- Collaboration

The OpenAPI specification defines a standard interface for RESTful APIs. The interface allows for easy discovery of services and provides for optimal documentation within the specification. Using such standards that is language-agnostic helps with interoperability across systems. It provides a baseline for defining services within the API that may be governed through standard tools.

One of the necessary features for API design tools is the ability to validate and suggest any resolutions for violations as the updates are made. A good spec designer also integrates with a source control repository and allows for collaboration with multiple users. Its integration with backend service for testing and debugging is an added advantage. Some tools also provide for generating a mock service that can be used for validation, testing, and distribution of the mock service URL to potential consuming teams.

API Implementation

API Implementation is the exercise of coding the service details. A service may be implemented on the backend using tools like Java/SpringBoot framework, or other low-code/no-code tools. Some of the considerations in accelerating development are

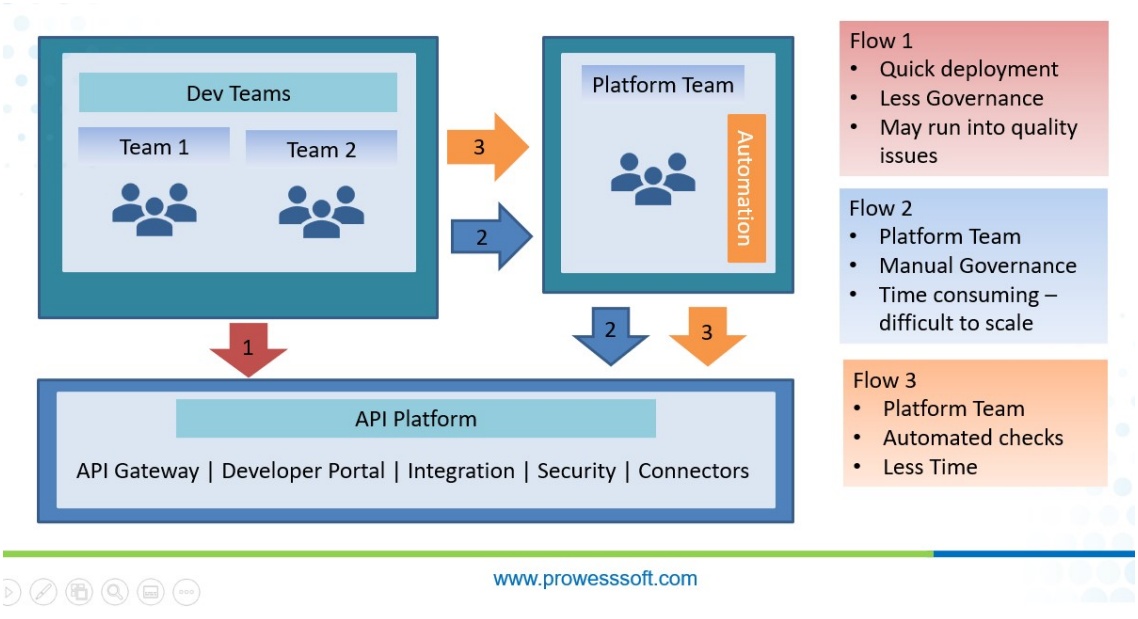
- Out-of-box tools to generate implementation stubs from the API specifications
- Integrated development environment that combines testing and debugging features
- Features to generate Data Access Objects (DAO) as a unified pattern for data operations
- Utilities to implement data validation, transformation, and aggregation
- Features that support Orchestration of services through synchronous and asynchronous calls

Most modern integration tools support these features out-of-box. Along with these, as an added feature, these tools allow for packaging the code to be deployed as microservices, on-prem or on the cloud.

Configuration

The configuration of the API involves working toward generation of scripts that help frontend the API with a Gateway. As part of the service, there would be configuration related to host and port information, endpoints URIs, routing services to backend APIs, configuration of policies, etc. This is an area that is not as much automated as it could possibly be. Some companies are investing heavily into this through APIOps, to close that whitespace.

When there are a smaller number of APIs that a single team is dealing with, there may not be much of a governance overhead. Everyone in the team may know the available APIs, their definition and usage. With larger API landscapes, without standard guidelines, each team may adopt a pattern or standard that works for them, but not necessarily fits into how other teams may discover and use them. To avoid such situations, there is usually a platform team or a governing authority that ensures that configuration is performed as per standard guidelines.



From the diagram above, “Flow 1”, shows how the deployment may be quick without a platform team, it helps with getting the services on the API Platform quicker, but there could be quality issues.

The second flow, “Flow 2” depicts a scenario where each service undergoes a deployment process through a governing authority that is the “Platform Team”. The second flow works for as the team is able to keep up with the growing API needs. With large API communities, it can turn out to be a bottleneck for implementation.

The next flow, “Flow 3”, shows the same route as “Flow 2”, except that the platform team has automated code generation for backend APIs and enabled appropriate checks as part of it.

Deployment

After the configuration for the API is ready, the deployment consists of execution of the generated code on the API Platform. This may be automated through GITOps strategy of using a Pull Request(PR) as the trigger for any changes. The process may also involve a test run to verify and also addition of the service to the developer portal for discovery.

Observe

An API Platform may provide a host of tools that gather appropriate metrics for API usage. It may also provide a customizable dashboard, with options for alerts and notification mechanisms. Post deployment, the observability of an API Platform becomes a great source of feedback into your API usage. It provides insights into the ones that may need to be updated, re-designed, scaled-up, scaled-down, or even retired.

A comprehensive framework of observability tools provides information on the most-used APIs, the usage trends by the application development community, and that can be a valuable input for the next set of APIs or upgrades to the existing ones.