Red Hat

The API owner's manual

7 best practices of successful API teams

Manfred Bortenschlager

Director, Business Development for API-based Integration Solutions and API Management, Red Hat

Andrew Mackenzie

Director of Software Engineering API Management, Red Hat

Jaya Baskaran

Principal Technical Marketing Manager, Red Hat

Greg Pack

Sr. Product Marketing Manager
App Services and API Management Solutions, Red Hat

Red Hat

Contents

Executive summary

Advances in modern application development, connectivity, and infrastructure such as 5G and edge computing continue to reshape how and where business processes happen. While these advances are increasing the speed of business, their benefits would not be possible without modern application programming interfaces (APIs).

Today, APIs have become the digital connective tissue of modern organizations, adding new capabilities to everything from operations and products to partnership strategies. Whether you have an API strategy today or are preparing to launch one, we have created this e-book to give you and your team 7 best practices for successful API programs.

Introduction

Before reading this e-book, consider the key objectives of your API program. Some of the questions asked later may help change, validate, or provide more substance to your objectives.

An effective API program should build on an organization's overarching corporate strategy and contribute to its objectives. You will know you have the makings of a great strategy when you can answer the following 4 questions:

1. Why do we want to implement APIs?
2. Who do we expect to use these APIs?
3. What concrete outcomes do we want to achieve with these APIs?
4. How do we plan to execute the API program to achieve that?

Why do we want to implement APIs?

A comprehensive understanding of their value and purpose is essential when considering APIs. Avoiding common misconceptions is crucial to maximize their potential business value. A prevalent misconception is that APIs are valuable only if users pay for them, which holds true when the API is the product. However, many API projects remain internal, generating diverse metrics like speed to market, reusability, sales, brand awareness, or affiliate referrals.

Five typical API provider use cases include:

1. **Scaling omnichannel.** Extend accessibility through diverse channels like mobile apps, internet of things etc.
2. **Growing your ecosystem.** Foster customer or partner ecosystems for expansion.
3. **Expanding your reach.** Establish broad transaction or content distribution networks.
4. **Powering new business models.** Drive innovation through novel business models.
5. **Generating internal innovation.** Generate innovation within the organization.

Many examples of successful API strategies can be found in modern technology companies, including Amazon, Salesforce, Twilio, and others. But that does not mean that only these leading technology companies can use APIs for monetization, innovation, partnership growth, or agility. Many well-established enterprises have a lot of traditional systems, which will not disappear overnight (probably never). APIs are the perfect glue to integrate with these traditional systems, expose them internally or externally and mash them up with other services to create new value. Any business on a digital transformation journey can tap into the advantages of the API economy by implementing a well-structured API strategy.

This "why" clearly needs to be strong enough that the decision to invest in APIs is an obvious choice for the organization.

Who do we expect to use these APIs?

Understanding who your API end users are will help define your API program success metrics.

- **Internal end users.** APIs can be used by internal teams to access company information for any number of business objectives.
- **External end users.** The same APIs or a subset can be used by external third-party developers in their applications.

This "why" clearly needs to be strong enough that the decision to invest in APIs is an obvious choice for the organization.

What concrete outcomes do we want to achieve with these APIs?

To identify concrete outcomes driven by APIs, consider internal and external organizational views:

- **Internal view.** Use your unique and valuable assets. Organizations possessing distinctive data, like Meta's "likes" data, can offer valuable APIs.
- **External view.** Consider market dynamics, trends, competitors, and customer behaviors. External forces shape business strategies and influence API functionalities.

Market dynamics profoundly affect mapping APIs. Early mapping companies overlooked real-time demands, helping startups like Waze to flourish. Google acquired Waze, integrating its technology into a successful API. Twitter, Reddit, and other giants also embraced APIs for innovation and ecosystem growth.

The right API strategy frequently combines internal assets with external market insights.

How do we plan to execute the API program to achieve that?

The final question, "How do we design the API program to achieve what we want?" is about implementation and execution, which requires meticulous planning of:

1. **Technological choices.** Determine the technology stack for building APIs.
2. **Design and maintenance.** Devise API design and maintenance strategies.
3. **Promotion and marketing.** Promote APIs within the organization and market them externally.
4. **Resource allocation.** Allocate necessary resources for API development and maintenance.
1. **Team composition.** Assemble a capable team for API development and management.
2. **Developer community.** Creating and maintaining a dedicated communication plan for external and internal developers.
3. **Success metrics.** Establish methods to track success against business objectives.

By uncovering the "why," "who," "what," and "how" of APIs, organizations can boost innovation and growth in the evolving API economy. The answers to these questions will differ for each organization and will be influenced by your goals, the strategy you deploy, and also, of critical importance: the API team.

The API team

An API team is typically structured like any other product team. Whether your customers are internal or external, the API team is responsible for building, deploying, operating, and optimizing the infrastructure others depend on.

Just like product teams, API teams can also be diverse. Teams should include a product-centric person who acts as the keeper of strategy and goals, design-focused team members who ensure best practices in API design, engineers who code API technology, and operations folks who ultimately run the API. Over time, you may also have others involved, including support and community team members, API specialists, and security representatives. Your extended API team could also include members of your developer community.

While this team could be a large number of people, some individuals may wear many hats, especially in smaller organizations. It is important to ensure that all stakeholder opinions are represented, even if that consists of having a team member check in on their concerns.

In many cases, API teams are formed temporarily and may belong to different organizational units with different line managers. This structure can make defining a shared vision for the API particularly challenging. For large API programs, different API teams may have to collaborate.

No matter the size of your organization, the 7 best practices described in this e-book will help establish a successful API team , and it may end up including more people than you think.

Best practice #1: Focus relentlessly on the value of the API

API programs must prioritize their core goal of delivering value while avoiding complexity. The value proposition determines user utility, a crucial API success driver. A compelling value proposition is essential for effective marketing. Aligning it with corporate objectives ensures sustainability, allowing established companies to enhance their offerings through APIs.

Alex Osterwalder's model of API value aligns user benefits with API features, creating a pivotal "fit" between your user's needs and API value.[1] Addressing needs, alleviating pain points, and generating value are crucial.

What does this mean in API terms?

In this iterative process, the 1st step describes the jobs your users are trying to complete, like automatically sending urgent communications in an emergency, backing up critical files, and sampling data to detect certain events.

The 2nd step requires identifying particular pain points that affect users before, during, or after trying to get a job done. These include ensuring reliable multiattempt message sending, failure detection, managing multiple messages, location-based message system integration, safeguarding file delivery with minimal bandwidth usage, and real-time correlation of extensive data quantities.

The 3rd step in building a user profile involves outlining potential benefits, like other types of notifications, which create opportunities rather than warn of threats, eliminating other storage equipment if reliability is good enough, and automatically triggering actions based on events.

Moving to the value map, the API's functionality, features, and services should be mapped out, focusing on pain relievers and gain creators. The process forms into concrete examples, like a messaging API that ensures message delivery, a storage synchronization API for efficient version updates, and a data aggregation API that provides configurable data streams.

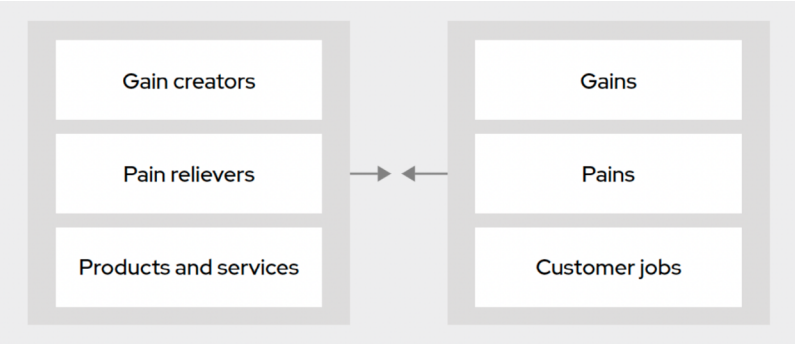

Figure 1. Value proposition canvas

Finally, the API team should complete a clarification exercise to compose several statements that demonstrate the fit between the API and user profile. When you condense and abstract your fit statements into one overarching message, it becomes the value proposition for your APIs. These "fit" statements further solidify API-user alignment, distilling the value proposition. In the case of the messaging API example, this statement might be something like:

> *"Quote from the customer goes here. Be mindful of language expansion when adding sidebar content. Some languages, like German, take up more space than English. To accommodate for translation, leave at least 1/4 of the sidebar empty."*

You may be thinking, "This seems like complete overkill, ours is just an internal API." This reaction is natural, but focusing on value is vital even in internal use cases. A poorly determined value proposition will lead to much time spent educating and pitching the API to other teams. A well-defined value proposition will make the API program a key contributor to the business.

[1] Strategyzer, "Value proposition canvas," Accessed September 2023.

Considerations for API program value

To define your API program's value, consider these key areas:

1. **User identification.** Identify users based on their relationships (customers, partners, and developers), roles, and preferences.
2. **Address pain points and gains.** Referencing the value proposition canvas in Figure 1, will determine user pain points, gains, and critical needs. Measure metric improvements (speed, revenue, and cost) and potential for new opportunities.
3. **Supported use cases.** Use the value proposition canvas to identify effective pain relievers and gain creators. Design your API to cater to these specific use cases.
4. **Future value expansion.** Plan your value proposition with an eye on the future. Anticipate upcoming milestones, trends, or technological innovations for continual value growth.
5. **Internal organizational value.** Evaluate the API's internal benefits and potential value to other teams.

By answering these questions, your API program can establish a clear value proposition and ensure alignment with user needs and organizational objectives.

Best practice #2: Make the business model clear from the beginning

Creating a successful API goes beyond a value proposition alone. It requires pairing the API with a well-defined business model. While recognizing and conveying the API's value is important, its costs must also be balanced against its financial or tangible benefits. In his crowd-created book Business Model Generation, Alex Osterwalder defines the business model of an organization as the way the organization proposes, creates, delivers, and captures value.

Understanding an API business model

The business model canvas dissects a business model's core components, which includes:[2]

1. **Value proposition.** Defines the unique API value.
2. **Revenue streams.** Outlines how the API generates income.
3. **Cost structure.** Associated costs of maintaining the API.
4. **Customer segments.** Target user groups.
5. **Customer relationships.** How the organization engages users.
6. **Channels.** Distribution methods to reach users.
7. **Key partners.** Crucial collaborations for API success.
8. **Key activities.** Essential tasks required for API operation.
9. **Key resources.** Vital assets necessary for API implementation.

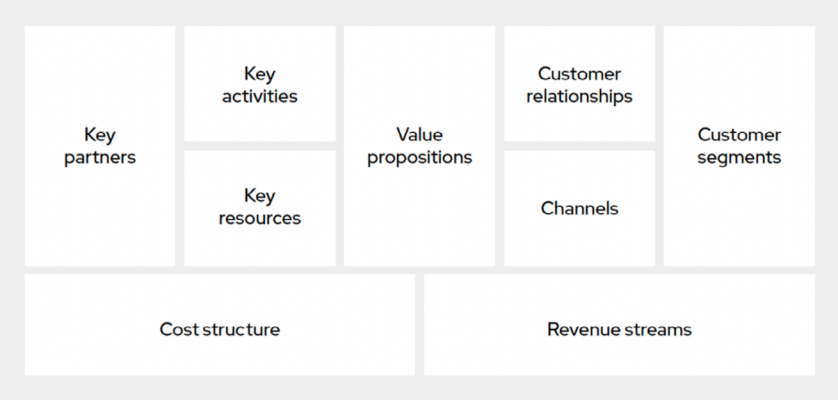

Figure 2. The business model canvas

APIs can open new opportunities, using existing assets. However, while recognizing API value is critical, its costs must be managed. An unsuitable model can escalate costs despite API value.

Considerations for your API business model

To align your business model with API usage, consider 5 vital areas, including:

1. **API value for the organization.** Assess diverse values, not just monetary, by exploring how the API aids the organization in increasing reach or innovation.
2. **Capturing value.** Determine the best way to capture the identified value, minimizing barriers for maximum value realization.
3. **Covering costs.** Recognize API-related costs, often beyond the API team, like engineering or marketing, and plan for their coverage.
4. **Long-term commitment.** Understand that APIs require an ongoing commitment for operation and maintenance beyond the initial investment.
5. **Strategic partnerships.** Identify essential partnerships and use complementary offerings from partners and suppliers during API development and market entry.

Best practice #3: Design and implement with the user in mind

API design encompasses fundamental design principles for a consistent user experience. Achieving a state of "ready to drive" design is the goal, where experienced developers intuitively understand APIs. API design should focus on 2 core areas: simplicity and flexibility.

Simplicity

Simplicity in API design is context-dependent, with designs varying in complexity across use cases. Striking the right balance in API method granularity is essential. Simplicity can be considered on different levels:

1. **Data format.** Choose between XML, JSON, proprietary formats, or combinations.
2. **Method structure.** Methods range from generic to highly specific, often sequenced to achieve specific use cases.
3. **Data model.** The exposed API data model may differ from the underlying one, affecting usability and maintainability.
4. **Authentication.** Different authentication mechanisms have strengths and weaknesses, with the choice dependent on the context.
5. **Usage policies.** Developer rights and quotas should be understandable and manageable.

Flexibility

Balancing simplicity and flexibility is crucial. An overly simplistic API might cater to only specific use cases, limiting its adaptability. To establish flexibility, outline the potential space of operations, including underlying systems and data models. Define a feasible and valuable subset of these operations. To find the right balance between simplicity and flexibility, consider the following:

1. **Expose atomic operations.** Combine atomic operations to cover the entire operational space.
2. **Identify common and valuable use cases.** Design a 2nd layer of meta-operations that combine atomic operations to serve these use cases.

Considerations for API protocol design

While representational state transfer architectural style (REST) remains the dominant standard for API development, API protocols are becoming a lot more diverse. More recent concepts include streaming APIs or WebSockets. Classic web service APIs won't disappear either. The protocol choice should follow this principle: pick the one that is most relevant for the user, then find the right balance between simplicity and flexibility. The infrastructure to deliver REST APIs continues to evolve.

To think through your API design, consider the following areas:

1. **Alignment with use cases.** Design the API to support identified use cases, maintaining flexibility for innovative and less frequent scenarios.
2. **Thoughtful RESTfulness.** While RESTful APIs are state of the art, assess if they genuinely fit your needs, as different architectural styles may be more suitable for specific use cases.
3. **Abstracted data model.** Implement an API with an abstraction layer between it and the data model, avoiding direct database access except when necessary.
4. **Geographic considerations.** Account for nonfunctional aspects such as latency and availability by strategically placing datacenters near your primary user regions.
5. **Integration with products.** Ensure API design harmonizes with other products through coordination or decoupling while maintaining clear internal and external communication of the structure.

API implementation and deployment infrastructure

As organizations move to cloud and hybrid cloud infrastructure for data and applications, API implementation and deployment infrastructures become more diverse, too. APIs may be built based on microservices and mashed up with APIs built as a monolith and everything in between. They may live in containers, virtual machines (VMs), bare metal, or somewhere in a public cloud environment.

As with any other application, having an automated continuous integration and continuous delivery (CI/CD) pipeline is critical to your API life cycle management. Adoption of GitOps and Argo CD can provide centralized API configuration management, automated deployments CD, and empowers collaborative efforts across the API team, leading to swift development and better API product quality.

Best practice #4: Place API operations at the top of the list

The API platform team manages APIs once they are live to make sure that they are accessible and delivered according to developers' expectations. While vendors offer solutions, selecting the right strategy is essential for success. API management involves 2 main functions:

1. Streamlining internal processes to be efficient and reduce cost.
2. Making operations effective to meet the expectations of external developers to the program.

The Building Great APIs Part 1: The Gold Standard and the API operations donut in figure 3 can help you achieve these goals.

API operations donut

The API operations donut can be used to define operations tactics to achieve an organization's API strategy. The inner circle of the donut represents an organization's internal activities and effects, and everything outside of the ring shows external effects.

Figure 3. API operations donut

1. **Dependability.** Ensure API availability through redundancy or quotas and rate limits. These align with business models and prevent downtime.
2. **Flexibility.** Offer technical and business adoption options, such as changing between price plans or cancellations. Keep in mind greater flexibility can translate into higher internal efforts and costs.
3. **Quality.** Maintain consistent adherence to developer expectations using service level agreements (SLAs) and streamlined processes.
4. **Speed.** Attain low latency and high throughput with techniques like throttling and caching, potentially aligned with business models.
5. **Cost.** Optimize developer value while minimizing internal costs without compromising quality.

It should be noted that several vendors provide technical infrastructure for many of these operations challenges—Red Hat included. Using a vendor is often a cost-effective way to address these problems, but the strategy must be thorough.

Essential API management components

Operating an API ecosystem requires a unique set of components to manage effectively. While these components will vary depending on the API strategy, the core 3 components will be the same:

1. **Access control.** Implement authentication and authorization systems to permit access and identify incoming traffic.
2. **Rate limits and usage policies.** Enforce quotas and restrictions on traffic to predict load.
3. **Analytics.** Capture and analyze traffic patterns for monitoring API usage.

Considerations for API operations

Ensuring that your API operations strategy harmonizes with your organization's overall strategies will help you allocate resources according to the API's significance.

To think through your API operations plans, consider the following areas:

1. **Access control.** Define who accesses, performs actions, and enforces limits, ensuring security-focused API usage.
2. **Metrics and alerts.** Gain visibility with analytics, measuring tailored metrics and setting alerts for API performance.
3. **Managing spikes.** Plan infrastructure using access control and policies and establish fallback mechanisms like spike arresting or throttling.
4. **API uptime responsibility.** Clear ownership of API uptime is crucial as it directly links to value generation and capture.
5. **Addressing undesired usage.** Distinguish expected and unexpected undesired usage, managing through proactive operations and terms and conditions.
6. **Communication.** Have a clear communication plan for internal and external developers to inform them of planned downtime, maintenance, and API changes.

Best practice #5: Obsess about developer experience

While developer experience may sound like it is about API design, it goes far beyond that. Think of developer experience as the packaging and delivery of the API rather than the API itself. You can have a wonderfully designed REST or containerized API, but if it is hard to sign up for access, read documents, and test, then you have created a poor developer experience that can significantly effect your API's success.

API success is more than great design

An API designed with simplicity and flexibility is wasted if developers do not engage with the API and eventually adopt it. At the same time, well-thought-out API design considerably impacts developer experience and adoption. Adoption is an essential part of the developer experience.

One of the pioneers in the field of APIs and API management, John Musser described several ways to enhance engagement that still hold true:

1. Clearly define the API's purpose.
2. Offer easy sign-up and free access.
3. Communicate pricing transparently.
4. Provide thorough documentation.

A key metric to improve API design for easy adoption is the time to first hello world (TTFHW) metric, which measures initial API interaction. A broader context is captured by time to first profitable app (TTFPA), emphasizing developer programs. This metric is trickier because being profitable is a matter of definition, depending on your API and business strategy. Considering TTFPA is helpful because it forces you to think about API operations as part of the API program.

Developer experience involves 2 principles: valuable product or service design and easy accessibility. A solid developer engagement program—including a portal, community, evangelists, events, and measurement—enhances engagement. Some examples of typical metrics for the developer portal are page visits, signups, API traffic, or support requests. Events can be measured by the number of attendees, API adoption at hackathons, or leads. It is helpful to create correlations, such as "did a talk at an event trigger more API signups?"

A developer program should include the following elements:

- **Community building.** Events like hackathons foster interactions, promoting adoption.
- **Developer evangelists.** Vital for success, they engage and promote API benefits.
- **Pilot partners and case studies.** Early adopters provide feedback and case studies.
- **Ecosystem partners.** Partnerships amplify adoption synergistically.
- **Measurement.** Metrics align with API objectives, aiding effective management.
- **Communication.** Portals, newsletters, private Slack or Discord servers.

Crafting a developer program tailored to the audience bolsters engagement and experience.

Considerations for evaluating developer experience

Delivering a well-thought-out developer experience can inspire developers to maximize their potential. Here are 6 considerations essential for evaluating developer experience to help build APIs that are not just lines of code but gateways to creativity and productivity.

1. **Value explanation.** Craft a short elevator pitch to convey API value to developers.
2. **TTFHW and TTFPA.** Assess and minimize time to first hello world and time to first profitable application, considering all developer experience elements (like portals).
3. **Onboarding process.** Align onboarding with API use cases, maintaining simplicity for early developer success.
4. **Value provision.** Ensure the API delivers enough value to attract and retain developers.
5. **Developer support.** Prioritize self-service support through documentation, FAQs, and forums, with additional mechanisms for more profound issues.
6. **Unconventional use.** Address support and documentation for developers exploring non-standard use cases.

Best practice #6: Go beyond marketing 101

Marketing APIs to developers can be challenging, especially if the value presented does not match the developer's business or technical needs. APIs should be marketed like any other product, aligning with segmentation, targeting, and positioning (STP). Effective marketing matches the right API with the right developers, moved by API value and STP principles.

1. **Segmentation.** Categorize customers as internal users, partners, end users, or external developers. For effective engagement, segment the vast developer market using the jobs-to-be-done method.
2. **Targeting.** Assess segment attractiveness based on accessibility, substantiality, and differentiation. Select promising segments and tailor marketing tactics accordingly.
3. **Positioning.** Make your API stand out by addressing specific needs, relieving pain points, and providing gains for chosen developer segments.

Prioritizing developer experience in marketing is crucial to success. Strategies like developer evangelists, robust developer portals, hackathons, and other events can create solid relationships with developers.

Considerations for marketing APIs

1. **Target audience.** Prioritize key user groups, evolving your API over stages based on internal users, partners, customers, or the public.
2. **Specialist selection.** Choose experts aligned with your API's value proposition for effective promotion, considering areas like engineering, support, sales, and product management.
3. **Event strategy.** Select event types (horizontal or vertical, global or local, conference or hackathon) based on API goals.
4. **Hackathon suitability.** Assess hackathon relevance (signups, SDK downloads, applications, recruitment, branding) and plan accordingly.
5. **Internal marketing.** Secure support across units, clarify with the marketing department and communicate benefits to product teams and customers.

Best practice #7: Remember API retirement and change management

API advice tends to focus heavily on API design, creation, and operation. However, one of the most critical segments of the API journey that gets overlooked is what happens many months after launch and operation—managing updates to the API, including API deprecation.

Disruptions from abrupt changes can erode confidence and entail significant costs, especially with unknown developers, mobile application approvals, or devices lacking update capabilities.

API changes are typically classified as nonbreaking or breaking. Nonbreaking changes include new methods and enhancements, while breaking changes involve removal, modifications, or total depreciation. Major version numbers and migration plans address breaking changes; minor versions handle nonbreaking ones. It can be challenging to make sure that changes won't break some applications, so we strongly recommend that any changes to the API, even if categorized as nonbreaking, should be rolled out by:

- Providing a test endpoint with the new version before launch.
- Sending an email or other communication to developers informing them of the change and giving timing and details.

Effective communication and transparent contracts are vital. Share problem details, uphold commitments, and outline version support duration. For example, deprecating an API requires a structured approach, including extended notice, addressing media coverage, migration plans, and data export tools if needed.

A migration plan facilitates API updates:

1. Introduce a new version for testing.
2. Notify users about retiring the old version.
3. Assist users during the transition.
4. Retire the old version.

Comprehensive API management includes anticipating updates and retirements, communicating effectively, and maintaining trust through transparent actions.

Considerations for API longevity

1. **Ensuring commitment to your customer guarantee.** This is arguably the most critical for your API program—what level of service stability are you willing to commit to for your users? Define service stability commitment for users. This commitment influences adoption.
2. **Change and breaking change process.** Determine release procedures to uphold the guarantee. Identify involved parties and approval steps.
3. **Change communication.** Detect, document, and convey changes using API definition formats. Ensure compatibility and clear communication.
4. **Version management.** Monitor usage of older versions using developer and user IDs. Establish a retirement process to prevent issues.
5. **Product alignment.** Coordinate API evolution with related product changes, considering customer commitments, and addressing necessary adaptations.

Sustaining your API strategy

These best practices are intended to help define, implement, and enhance your API strategy. The goal is to help you adapt your strategy and uncover new API opportunities. It is likely that many of the questions in the previous sections need to be answered in great detail, and new opportunities or risks may arise over time. For example:

- Are there ways to expand the value of the API to the customer?
- Have we provisioned sufficiently for the future?
- Is the value proposition attractive enough for developers to genuinely engage?

As the IT landscape evolves, so must your API and offerings. Four main drivers of change include:

1. **Industry forces.** New competitors or services could replace your API.
2. **Market forces.** Changing user demands or market segment conditions.
3. **Macroeconomic forces.** Global market shifts affect user budgets.
4. **Trends.** Evolving technology or regulatory mandates.

Creating a lasting API program

An API program that matches strong use cases with potential customers has the potential to successfully launch, grow, and evolve. These successful API programs build room to innovate through planned API flexibility and by proactively addressing potential negative use scenarios through design and operations. It is also critical to ensure your API terms and conditions give you the tools necessary to act against unexpected behavior.

You may need to reevaluate your API strategy if:

- Reliance on unexpected innovation overshadows value. Robust infrastructure is crucial for success alongside this approach.
- Lack of compelling use cases for you and users indicates a need for a different approach.
- Internal doubts about negative behavior persist, signaling insufficient measures or communication.
- Frequent API compromises or misuse reveal misalignment between intended and actual value.

APIs do not exist in a vacuum

The ever-evolving technology landscape has introduced a range of powerful cloud-native application development tools, including Kubernetes, Red Hat® OpenShift®, and Red Hat OpenShift Service Mesh, which synergize to enhance connectivity and expedite the development of exceptional applications. However, amidst these advancements, the foundational principles of a robust API strategy and the importance of an effective API management system remain unwavering. As you embark on this journey, seek an API management solution that not only aligns with your overarching strategy but also demonstrates a strong understanding of, or consistent integration with, these platforms to facilitate the execution of your strategic vision.

Why choose Red Hat to build and manage your APIs?

Red Hat advocates for a strategic API design-first approach to ensure the success of your API program. This approach encompasses the entire API life cycle, from initial planning and design, which includes resource mapping and business scenario modeling, to the management of the API's which includes controlling access, accessing the APIs, and analyzing usage.

To help development teams with these best practices of the API design-first approach you have read about in our API Owner's Manual, Red Hat offers Red Hat 3scale API Management, which is the API management solution purpose-built for hybrid cloud environments. Our distributed, lightweight container-based, cloud-native solution makes it possible to handle large workloads and promotes a security-focused approach and collaboration. Sharing and controlling access to services, resources, applications, and enterprise systems across public and private cloud environments empowers your business in complex ecosystems.

But, like we mentioned above, APIs don't exist in a vacuum and to use some of these best practices, you will need a complete application platform for cloud-native application development. As such, Red Hat 3scale API Management is included in the Red Hat Application Foundations portfolio—a collection of middleware tools that include capabilities such as integration, API design, API governance and registry, streaming and messaging that are designed to work with Red Hat OpenShift Container Platform. By using Red Hat OpenShift and Red Hat Application Foundations together, organizations have a complete cloud-native application platform to deliver new software to users more efficiently and more safely, while also unlocking strategic capabilities and benefits for their organizations.

Elevate your cloud-native application development approach with Red Hat 3scale API Management, Red Hat Application Foundations, and Red Hat OpenShift, and experience the future of streamlined, efficient, and collaborative application building.

Conclusion

We hope these best practices will help guide some choices you make, or at least the questions asked. Your journey to building your API strategy begins with a clear, consistent, and organization-wide understanding of your API's value. But that value and the business objectives for your API are not static. Things change, and your API strategy needs to change with them.

Based on our experience working with our customers and making observations in the API economy, we can summarize some traits of a valuable API program:

- Users adopt an API because it is of value to them. It does an essential job by relieving pain or creating gain.
- An API continues to provide value for users, delivering a value proposition and strategy that changes according to the environment.
- An API is valuable to the organization internally. It does something important and helps the organization achieve significant objectives.
- As many stakeholders as possible (ideally all) are happy.

Learn more about Red Hat 3scale for API management and explore Red Hat's API Management resources.