træfiklabs

Top Five Policies for Runtime API Governance

About the Authors

Ikenna Nwaiwu

is the author of Automating API Delivery and the Principal Consultant at Ikenna Consulting. Ikenna specializes in automating API governance.

Immánuel Fodor

is Product Manager at Traefik Labs with 13+ years of tech industry experience. CKA, CPM, CSM, AWS CP, homelabber, former CTO.

Table of Contents

API governance provides a framework for defining policies and standards to ensure APIs are consistent, compliant, and secure. However, a common misconception about API governance is that it is solely about API design concerns, such as following a specified pagination pattern or ensuring API descriptions adhere to specific linting rules. While these design-time considerations are important, they're only part of the picture. API governance is not just about defining design standards; it's the backbone of ensuring that APIs are consistent, compliant, and secure across their entire lifecycle.

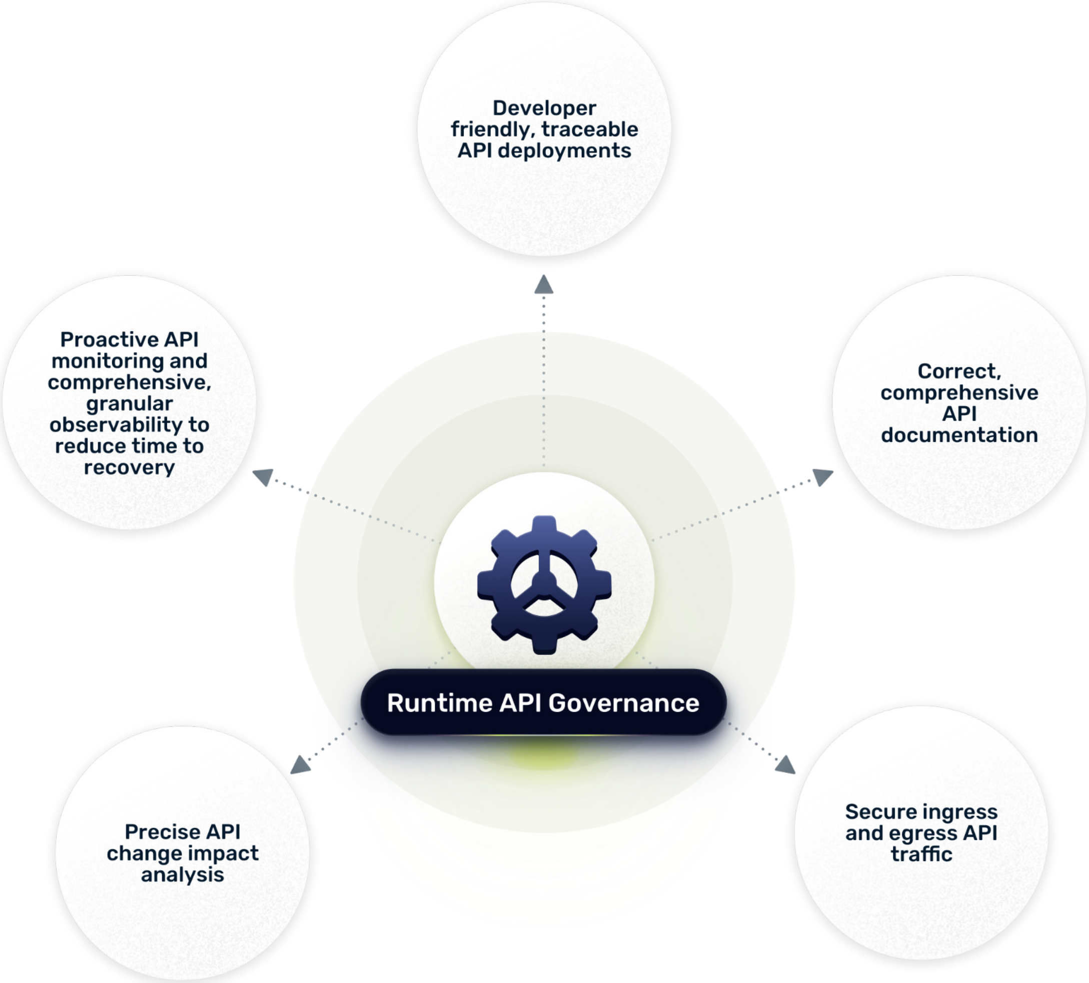
Overlooking runtime governance—like monitoring API performance, managing access controls, and enforcing security protocols—can lead to catastrophic failures in production. For instance, imagine a beautifully designed API that lacked proper runtime governance. Without robust security measures, attackers could exploit it, leading to data breaches and significant financial loss. Or consider an API with no runtime monitoring; a sudden surge in traffic could cause unanticipated outages, bringing down critical services and eroding user trust. Effective API governance addresses design and runtime concerns, ensuring that APIs meet standards during development and perform reliably and securely in the real world.

API management teams are responsible for defining and enforcing runtime API governance. Runtime API governance is the structured framework that manages and enforces policies for API deployment, security, observability, and lifecycle during their operational phase. It ensures APIs are deployed in a traceable and secure manner, with all configurations version-controlled and changes managed through a standardized process. This governance also mandates that live APIs be centrally documented, monitored for performance, and secured via gateways and firewalls. Overall, API governance provides a controlled environment to maintain API integrity, security, and efficiency in real-time operations.

Given how crucial API governance is to an organization's API strategy's success, let's explore the topic further. In this ebook, we discuss five policies we recommend for API teams. They are listed below.

- **Policy 1:**
  Use a developer-friendly, traceable, and easy-to-rollback approach for API deployments.
- **Policy 2:**
  Make comprehensive, correct API documentation available to users.
- **Policy 3:**
  Secure ingress and egress API traffic from malicious and unauthorized access.
- **Policy 4:**
  Provide precise API change impact analysis to minimize service failure and breaking changes.
- **Policy 5:**
  Provide proactive API monitoring and comprehensive, granular observability to reduce time to recovery.

Each policy has one or more actionable standards that teams should adopt to meet these criteria.


Developer friendly, traceable API deployments
Proactive API monitoring and comprehensive, granular observability to reduce time to recovery
Correct, comprehensive API documentation
Runtime API Governance
Precise API change impact analysis
Secure ingress and egress API traffic

Before we dive in, here is a note on the terminology we use in this ebook. By policy, we mean a high-level statement that requires compliance with one or more associated standards. A policy focuses on results and not implementation. On the other hand, by standards, we mean detailed mandatory, recommended, and optional rules used to gauge compliance with a given policy. Standards are specific and measurable, using capitalized keywords from IETF RFC-2119 ('MUST', 'MUST NOT', 'SHOULD', 'SHOULD NOT', 'MAY' etc.) to indicate the level of requirement compliance.

Policy 1
Use a developer-friendly, traceable, and easy-to-rollback approach for API deployments

The DORA metrics, Deployment Frequency (DF), Lead Time for Change (LTC), Change Failure Rate (CFR), and Mean Time to Recovery (MTTR), are widely used to assess a software team's performance. However, there's now an increased emphasis on developer experience and productivity. The State of Developer Experience Report 2024 revealed that **69% of developers are losing 20% or more of their time due to inefficiencies at work**. The survey also explored the key areas that engineering leaders believe will enhance developer productivity and satisfaction. One of the top five areas identified was improving collaboration tooling.

When making changes to your API, it's crucial to have a deployment process that allows for fast deployment, easy experimentation, and the ability to roll back failed deployments and restore service quickly. Such a deployment process requires an automated deployment system to minimize manual errors. A deployment process like this is a key enabler for reducing LTC and improving software delivery performance. Additionally, the deployment process should facilitate seamless collaboration between development and operations teams and provide robust access controls to determine who can make changes. Furthermore, the process should also maintain an auditable history for traceability and accountability.

Here are three standards that can help realize this policy.

Governance Standard 1.1
All API descriptions and configuration MUST be version controlled in Git

Managing API descriptions (like OpenAPI) and API configuration files (like Kubernetes Ingress manifests) in Git enables a workflow that provides a clear audit trail, giving a history of what was changed, when, and who changed it. This traceability makes it easier to troubleshoot changes and roll back a change that introduces a failure with a single Git revert command. It also enables developers to edit the files using editors, CLIs, and any preferred tool.

Governance Metric: The ratio of the organization's APIs whose configuration is under version control.

Governance Standard 1.2
All API configuration changes MUST follow a GitOps approach

GitOps has emerged as the modern way to manage application and infrastructure deployments. As part of the GitOps approach, the application state is managed in Git repositories, which act as the source of truth for the system's desired state. Application state is stored using declarative descriptions in formats like YAML and JSON. When the configuration files are updated, continuous delivery tools synchronize the application's live state with the desired state.

A GitOps approach allows for the fast delivery of API changes. Automated governance checks in the deployment pipeline enable organizations to achieve fast LTC and improved API consistency. Pull requests can be a lightweight governance technique to promote collaboration on changes. Teams following the GitOps approach disallow configuration changes from the application UI, eliminating error-prone TicketOps and ClickOps-based API deployments.

Governance Metric:

- *Number of API configurations managed with GitOps.*
- *API config change review speed: The average time it takes from opening a config change PR to completing a review on it.*

Governance Standard 1.3
The CI/CD pipeline MUST act as the primary change agent, driving all changes, feedback, and collaboration through automated processes, with no manual steps outside the pipeline.

A CI/CD pipeline should include all the necessary steps to validate and deploy working software to production. These steps involve compiling and packaging source code, conducting unit, integration, and acceptance tests, and deploying to progressively higher environments. In this way, the pipeline acts as both a central falsification mechanism and a primary change agent, ensuring that all changes, feedback, and collaboration are processed through automated workflows, with no manual intervention. If a new software change (that is, a commit) presents the hypothesis that "this change complies with standards and does not break working software," then the CI/CD pipeline serves as a falsifiability test to evaluate that hypothesis.

By automating collaboration patterns, the pipeline ensures that feedback loops are consistent and transparent across teams. In this way, the CI/CD pipeline becomes not only an objective evaluator of changes but also a facilitator of cross-team feedback and collaboration. No manual steps should occur outside of the pipeline, ensuring that all changes and collaboration are fully automated and trackable. By automating previously manual processes, the pipeline evolves into an automated knowledge base that outlines the steps required to validate working software.

Such a pipeline has a profound effect on the team's culture. Changing the pipeline changes the culture. Whenever a new standard is introduced, the team must first ask, "Is it possible to build this into the pipeline to solidify our expectations of collaboration from others?" The development team can refer to the pipeline logs when a build fails. The platform and infrastructure teams can use data produced by the pipeline to verify whether the software has been successfully deployed. The security team can review the security scan and testing results from the pipeline to validate a release candidate, and so on.

Governance Metric: % of changes validated through the pipeline vs. not

Policy 2

Make comprehensive and correct API documentation available to users

Large enterprises often struggle with API sprawl, which refers to the uncontrolled proliferation of APIs. API sprawl usually occurs when there is a lack of proper documentation and centralized API governance. Simply put, the organization may not know where all its APIs are located or what they are used for. API sprawl hinders integration and the exploitation of new business opportunities and poses a security risk. This risk can stem from Shadow APIs, which are used without the knowledge of the API governance teams and exist outside the established governance process. Additionally, the security risk can result from Zombie APIs that are no longer actively maintained, supported, or monitored but are still functional and accessible to users.

To combat API sprawl, a comprehensive API discovery mechanism should be implemented. API discovery is about making it easy for anyone in the organization to find any API they need. When people can locate the available APIs within an organization and understand what they do, it helps promote API consistency and reuse and reduces duplication of effort.

API documentation documents (that is, API descriptions like OpenAPI) also have to be comprehensive and accurate. Nothing produces an annoying user experience, and reduces trust in API providers, like available but inaccurate API documentation. Therefore it is important that runtime platforms are able to validate that API documentation matches the runtime traffic.

Governance Standard 2.1
Edge technologies SHOULD detect undocumented APIs

One way to manage API sprawl is to have policies and controls in place to detect undocumented and unmanaged APIs. Detecting APIs can be done by monitoring traffic at the edge. This detection can be one of the first steps for organizations suffering from API sprawl. From there, the organization can collate the documentation and store it in the internal API catalog, which leads to the next standard.

Apart from detecting undocumented APIs, runtime API platforms can also validate the correctness of API documentation by comparing it to run time traffic. This runtime validation mitigates API drift and is a security measure that identifies unexpected and potentially malicious API behaiviour.

Governance Metric:

- *Number of unmanaged APIs detected by edge controls.*
- *Number of API description document defects*

Governance Standard 2.2
All live APIs MUST be documented in a central API catalog

As part of your API deployment pipeline, API documentation should be stored centrally in an internal API catalog or registry. Additionally, a selection of these APIs can be made available to partners or consumers through an external developer portal. The catalog should have a browseable UI for users, but it should also provide API registry features. That is, it should provide the API descriptions for programmatic access by machines.

Governance Metric: The number of APIs in the API catalog.

Governance Standard 2.3
The API management platform SHOULD support self-service onboarding

Along with making APIs discoverable, users should be able to request and obtain access to an API quickly without jumping through administrative hoops. Self-service onboarding of APIs by obtaining an API key through a developer portal is one of the runtime requirements for a great developer experience for API users.

Governance Metric: Number of APIs documented in a self-service developer portal.

Policy 3

Secure ingress and egress API traffic from malicious and unauthorized access

The rapid growth of APIs and API traffic has exponentially expanded the attack surface for malicious actors. In its State of API Security Report 2024, Salt Security states that 95% of respondents in the research experienced security problems in production APIs, with 23% experiencing a breach. 25% of respondents also stated that one of their biggest concerns with their company's API program was that it didn't adequately address runtime or production API security.

It is crucial to protect APIs and invest in API runtime protection to guard against threats resulting from authentication weaknesses, sensitive data exposure, and account takeover or misuse. Public APIs can especially face a host of attack vectors—from SQL and code injection to cross-site scripting (XSS) and other OWASP Top 10 API vulnerabilities. They can also face bot attacks ranging from data scraping and intellectual property theft to account takeover and distributed denial of service (DDoS).

While organizations need to secure the APIs they build, they also consume APIs from third parties. These third-party APIs include APIs of external LLMs, CRM systems, payment processing platforms, messaging APIs, and more. However, the unsafe consumption of APIs is an OWASP API Security Top 10 threat. It arises when an application, such as an API, consumes external, third-party APIs and blindly trusts the data they return without adequately validating or sanitizing the data it receives. This exposes the application to SQL injection, code injection attacks, or the exposure of sensitive information.

Here are a few standards you can implement to achieve this policy in practice.

Governance Standard 3.1

All governed APIs SHOULD be exposed through an API gateway configured with appropriate security controls

API gateways provide control points for runtime API governance, enabling governance teams to enforce common runtime standards such as authentication and authorization, granular API access, rate limiting and throttling, input validation, and monitoring. API gateways can also integrate with an identity provider (IdP) to provide user authentication and authorization.

Governance Metric:

- *The number of APIs with insufficient authentication*
- *The number of governed APIs not exposed through the gateway*

Governance Standard 3.2

All public APIs MUST be protected by a WAF

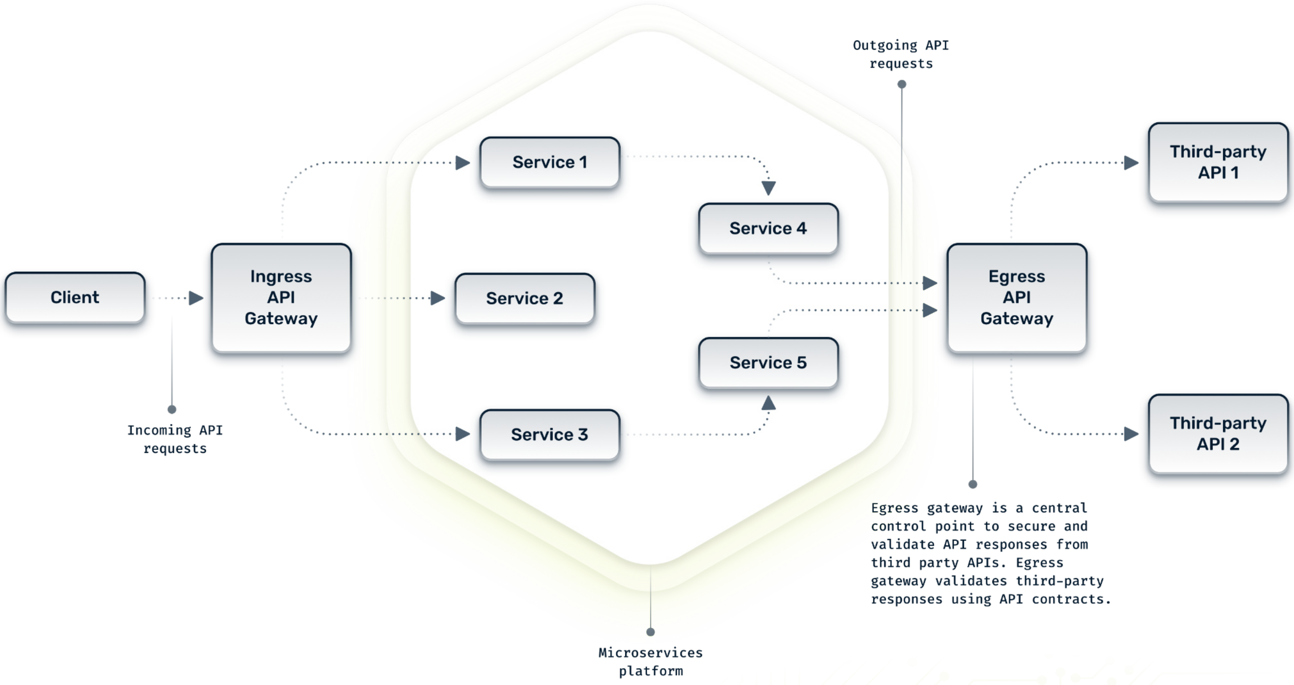
A web application firewall (WAF) actively scans and filters incoming HTTP traffic. It blocks malicious API requests in real-time, typically using a set of threat signatures—pre-defined rules or patterns that identify malicious activity. In addition to maintaining a set of threat signatures, some WAFs use machine learning and behavioral analysis for pre-emptive protection, minimizing the need to update signatures when a new threat is identified continuously. WAFs may integrate with a global network of sensors to provide real-time threat intelligence on emerging threat patterns. WAFs can also provide detailed audit logs and reports required to meet various industry compliance requirements, such as PCI DSS, HIPAA, and GDPR.

Governance Metric: The number of public APIs unprotected by a WAF.

Governance Standard 3.3
Third-party API traffic MUST be validated

While organizations may put a lot of thought into securing the APIs they provide externally (such as encrypting the traffic and routing it through a gateway), they may not pay as much attention to the security of the APIs they consume. That is, API requests to third parties. Developers tend to trust the data from third-party APIs they consume, especially if they are provided by well-known companies. As such, they may adopt weaker security standards for API consumption.

A way to mitigate the security risk from third-party API traffic is to ensure third-party API traffic is encrypted, validated, and sanitized. Organizations can proxy the traffic through an API gateway that acts as an egress API gateway (also called a consumption gateway or reverse gateway). In the egress gateway, teams can use strong API contracts from the third party to validate that the traffic complies with the types, formats, and constraints defined in the contract. The egress gateway can also be a control point for implementing robust error handling and preventing the exposure of sensitive data. This egress gateway pattern is illustrated in the figure below.



Governance Metric: The number of unvalidated third-party APIs.

Policy 4

Provide precise API change impact analysis to minimize service failure and breaking changes

Change impact analysis involves identifying the potential consequences of a change, the components affected, and the possible risks associated with the change. It helps developers identify areas that can be affected by a change so that changes do not introduce system stability, performance, and security issues.

A way to analyze the impact of an API configuration deployment is to perform a static code analysis of the deployment artifacts. This analysis can help build a representation of a change and visualize its impact and any dependencies that may also be affected. For example, a static change analysis check may show that a rate limit change may affect not just one API but more. It may also show that changing a rate limit for a user group affects multiple groups. Such insights help reduce CFR.

Another way to analyze the impact of an API change is by detecting breaking changes, which can disrupt API users. A breaking change is a non-backward compatible change that forces API consumers to change their code to ensure their API integrations keep working.

Let’s look at two standards an organization can use to implement this runtime governance policy.

Governance Standard 4.1

All API configuration changes must pass static checks

One advantage of GitOps is that API configuration is stored in version control. This allows the team to run a static analysis on API configuration in the CI/CD pipeline. The analysis ensures that configuration files are valid and prevents any API misconfiguration from arising. It also helps the dev team understand the impact and scope of their changes on different APIs and API consumer groups.

Governance Metric: The ratio of the organization’s APIs whose configuration is under version control and change impact analysis checks.

Governance Standard 4.2

All APIs must follow the recommended versioning scheme

API versioning helps providers evolve an API and add new versions without breaking existing integrations. Many different versioning schemes exist for REST APIs (such as using the URI path, query parameters, custom headers, content type, and more). Your runtime infrastructure must support the versioning mechanisms that best fit your requirements. As an API provider, defining and adopting a consistent versioning mechanism for your APIs gives users a clear migration path to a new version and helps you manage the lifecycle of the API. Be sure to run breaking change checks before deployments to ensure no breaking changes are introduced.

Governance Metric: The number of APIs that do not follow the versioning scheme.

Policy 5

Provide proactive API monitoring and comprehensive, granular observability to reduce time to recovery

API monitoring involves tracking key metrics of API performance, such as response times and error rates. API Observability goes beyond monitoring a known set of API performance metrics. Observability allows API publishers to troubleshoot novel problems and understand what is happening inside an application. Observability is based on the ability of applications and infrastructure to emit telemetry signals: metrics, logs, and traces (MLT), which API publishers can analyze. API publishers can run distributed traces using telemetry, observing requests as they flow through a distributed system. This tracing gives them visibility into a system's health and enables debugging the behavior of the production system.

API observability should be based on open standards for instrumenting, generating, collecting, and exporting telemetry data. Open observability standards like OpenTelemetry guarantee compatibility with various backend API analytics platforms and prevent vendor lock-in. They provide a way to decouple your observability strategy from your gateway strategy. Open standards also provide teams with a common language for describing cross-application interactions and troubleshooting.

Governance Standard 5.1
Teams MUST provide dashboards and configure alerts to monitor API request rate, error rate, and request latency

Having runtime visibility into an API's performance is essential not only for understanding its usage but also for incident mitigation. API publishers should capture API metrics on uptime, requests per minute, latency, and errors per minute. They should also set up monitoring dashboards and proactive alerts to make API performance information available for the teams that support them.

Governance Metric: The number of APIs not set up for monitoring.

Governance Standard 5.2
APIs SHOULD use OpenTelemetry to publish telemetry data

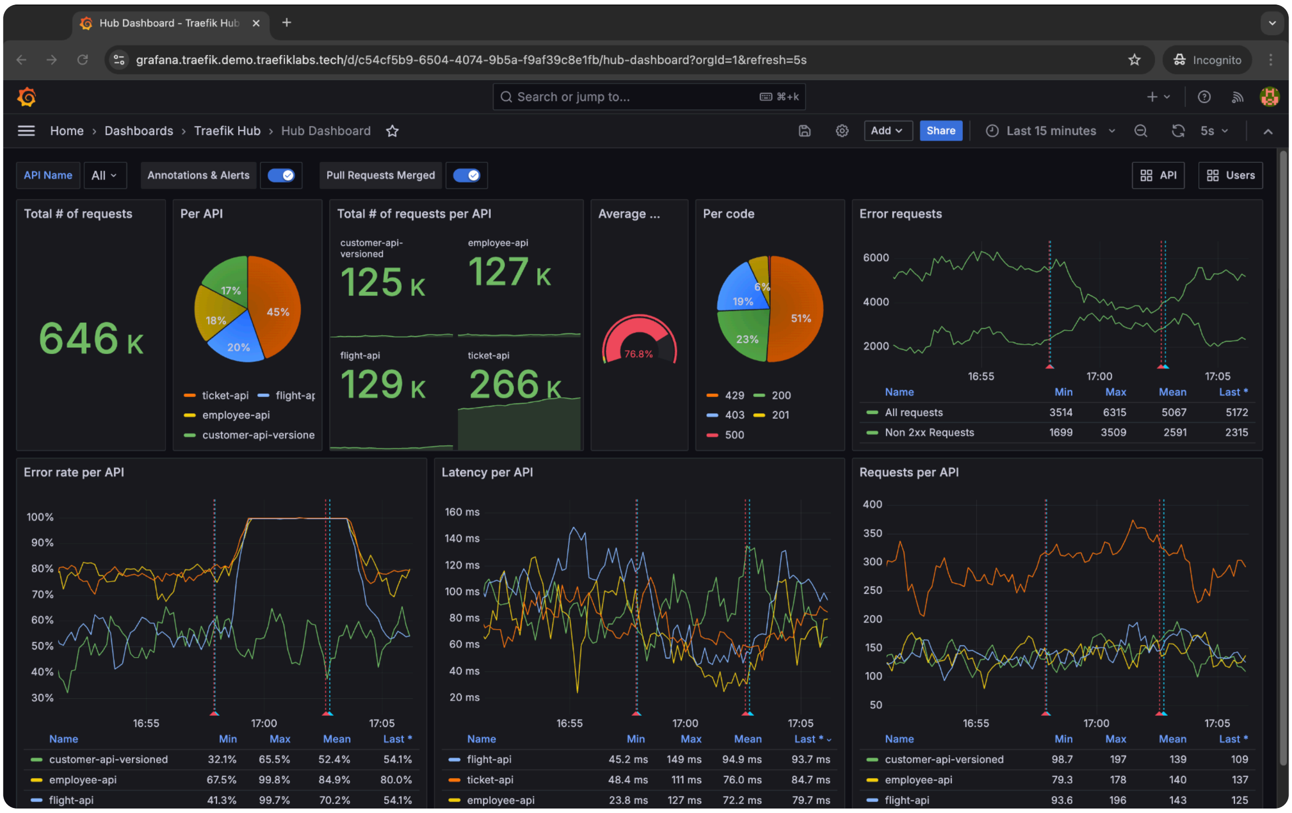
OpenTelemetry is a widely supported open-source observability framework that has become the de facto standard for cloud-native applications. OpenTelemetry is built on rules and conventions known as the OpenTelemetry Protocol (OTLP). This protocol dictates how components related to OpenTelemetry exchange telemetry data between a source system that generates the data and its destination. The OTLP data model consists of traces, which describe the flow of execution; metrics, which offer statistical information on application performance; and logs, which provide detailed information on the application's state.

Governance Metric: The ratio of APIs instrumented with OpenTelemetry.

Governance Standard 5.3
API deployments SHOULD emit telemetry data

Application performance telemetry can be correlated with API deployment events to quickly visualize and investigate any API performance problems resulting from new deployments. Deployment event correlation with API runtime metrics is a powerful incident mitigation technique. This event correlation lets teams see if a deployment introduces problems and roll back the change.

Governance Metric: The ratio of API deployment pipelines that publish telemetry data.



Error requests

Name	Min	Max	Mean	Last *
All requests	3514	6315	5067	5172
Non 2xx Requests	1699	3509	2591	2315

Error rate per API

Name	Min	Max	Mean	Last *
customer-api-versioned	32.1%	65.5%	52.4%	54.1%
employee-api	67.5%	99.8%	84.9%	80.0%
flight-api	41.3%	99.7%	70.2%	54.1%

Latency per API

Name	Min	Max	Mean	Last *
flight-api	45.2 ms	149 ms	94.9 ms	93.7 ms
ticket-api	48.4 ms	111 ms	76.0 ms	84.7 ms
employee-api	23.8 ms	127 ms	72.2 ms	79.7 ms

Requests per API

Name	Min	Max	Mean	Last *
customer-api-versioned	98.7	197	139	109
employee-api	79.3	178	140	137
flight-api	93.6	196	143	125

Implementing Runtime API Governance in Traefik Hub

In Part 1, we discussed the top five policies for runtime API governance. Now, let's look at how Traefik Hub implements these policies. Traefik Hub is a Kubernetes-native API management solution built with scalability, flexibility, and simplicity in mind. Traefik Hub helps route traffic to services in dynamic, microservices-oriented environments.

In the following sections, we will provide examples of configuration implementation in Traefik Hub to support the standards we have discussed. But note that our examples are not exhaustive - Traefik Hub supports many more capabilities and configuration options!

Policy 1: **Use a developer-friendly, traceable, and easy-to-rollback approach for API deployments**

As a Kubernetes-native solution, Traefik Hub supports declarative resource configuration. Traefik Hub configuration can be done through Kubernetes resources like `Ingress` and `Service`, and Traefik Custom Resource Definitions (CRDs) like `IngressRoutes`, `Middleware`, `API`, `APIPortal`, `APIAccess`, `APIVersion`, and `APIRateLimit`. Configuration can be version-controlled in Git and automatically applied to the Kubernetes cluster using GitOps tools like ArgoCD and Flux.

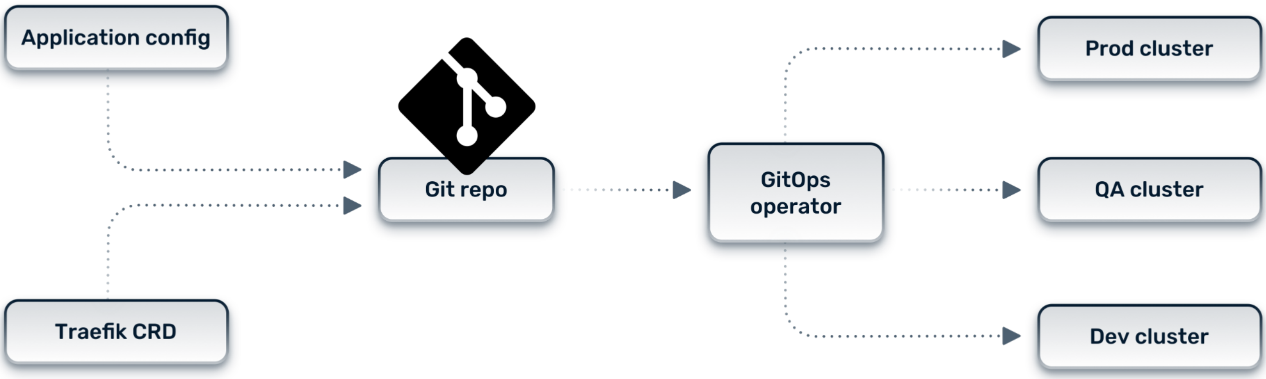
For example, defining external traffic routing to a service involves creating an API CRD and exposing the API with an `IngressRoute`. The following resource definition declares an API called hello-api, and provides the path to its OpenAPI description.

```yaml
---
apiVersion: hub.traefik.io/v1alpha1
kind: API
metadata:
  name: hello-api
  namespace: apps
spec:
  openApiSpec:
    path: /openapi.yaml
```

Next, specify an IngressRoute CRD with routes to connect incoming requests to services that can handle them. The following resource definition declares a router that matches a request with the host `api.example.com` to the hello-api-service.

```yaml
---
apiVersion: traefik.io/v1alpha1
kind: IngressRoute
metadata:
  name: hello-api-ingress
  namespace: apps
  annotations:
    hub.traefik.io/api: hello-api # Name of the API object
spec:
  routes:
  - match: Host(`api.example.com`)
    kind: Rule
    services:
    - name: hello-api-service
      port: 8080
```

Developers can save configurations to a Git repository, and GitOps operators ensure that these configurations are consistently applied across environments. The diagram below illustrates this process.



The declarative nature of Traefik configuration, combined with GitOps workflows, supports advanced and progressive deployment capabilities. Teams can follow blue-green deployments and canary releases using GitOps operators like ArgoCD and FluxCD. If any issues are detected, changes can be rolled back to a stable version by reverting the changes in the Git repository.

TIP: For an in-depth treatment on how to do load-balancing in Traefik and support advanced deployment techniques, see Traefik's Advanced Load Balancing course.

Policy 2: **Make comprehensive and correct API documentation available to users**

Traefik Hub API management includes an API Developer Portal to make API documentation accessible to developers and other API users. In the portal, users can explore available API endpoints, understand their usage, and test them in real time.

The `APIPortal` CRD creates and configures the portal, generating a web interface for browsing the API documentation. The visibility of API documentation depends on user groups. An API will be visible to an API consumer if they belong to a specified group with access to the API, as configured by the APIAccess CRD.

You can create an API developer portal by applying an APIPortal resource as follows:

```yaml
---
apiVersion: hub.traefik.io/v1alpha1
kind: APIPortal
metadata:
  name: my-portal
  namespace: default
spec:
  title: "My Portal" # The title for the Portal
  description: "API documentations" # A short description of the Portal
  trustedUrls:
    - "https://portal.example.com"
  ui:
    logoUrl: https://traefik.io/favicon.png # URL to a picture used as logo
```

To expose the portal, create an `IngressRoute` resource, as demonstrated below:

```yaml
---
apiVersion: traefik.io/v1alpha1
kind: IngressRoute
metadata:
  name: my-portal-ingress
  namespace: traefik-hub
  annotations:
    hub.traefik.io/api-portal: my-portal@default # Reference the APIPortal
spec:
  routes:
  - match: Host(`portal.example.com`)
    kind: Rule
    services:
    - name: apiportal
      port: 9903
```

Policy 3: **Secure all API traffic from malicious and unauthorized access**

With Traefik Hub, you can ensure secure API access using protocols such as Oauth2, OpenID Connect (OIDC), API keys, and JSON Web Tokens (JWTs). Additionally, Traefik integrates with identity providers such as Keycloak and Okta. Traefik also integrates with any other IdP that supports the OIDC protocol to handle user identities and authorize access to the APIs and API portals.

To protect APIs from DDoS attacks or excessive requests from a single client, you can implement rate limits by defining an APIRateLimit resource. Traefik allows you to do this for all APIs or specific user groups. The following declarative config defines a rate limit of 100 requests per minute.

```yaml
---
apiVersion: hub.traefik.io/v1alpha1
kind: APIRateLimit
metadata:
  name: my-rate-limit
  namespace: apps
spec:
  # Rate limit configuration, this config allows 100 requests/minute.
  limit: 100 # 100 requests
  period: 1m # One minute
  groups:
    - support
  apiSelector:
    matchLabels:
      module: crm
```

Using Traefik's Distributed Rate Limit (which is based on the Token Bucket algorithm), you can limit requests over time across your entire cluster rather than just an individual proxy.

In addition, Traefik Proxy v3 includes the Coraza Web Application Firewall (WAF) integrated as a plugin. Coraza is an open-source OWASP project and a high-performance WAF that supports Modsecurity's seclang language and OWASP core rule sets. Enabling this involves two simple steps:

- First, updating Traefik's static configuration that loads the plugin as follows:

```yaml
experimental:
  plugins:
    coraza:
      moduleName: github.com/jcchavezs/coraza-http-wasm-traefik
      version: v0.2.2
```

- Then, updating the WAF CRD as follows:

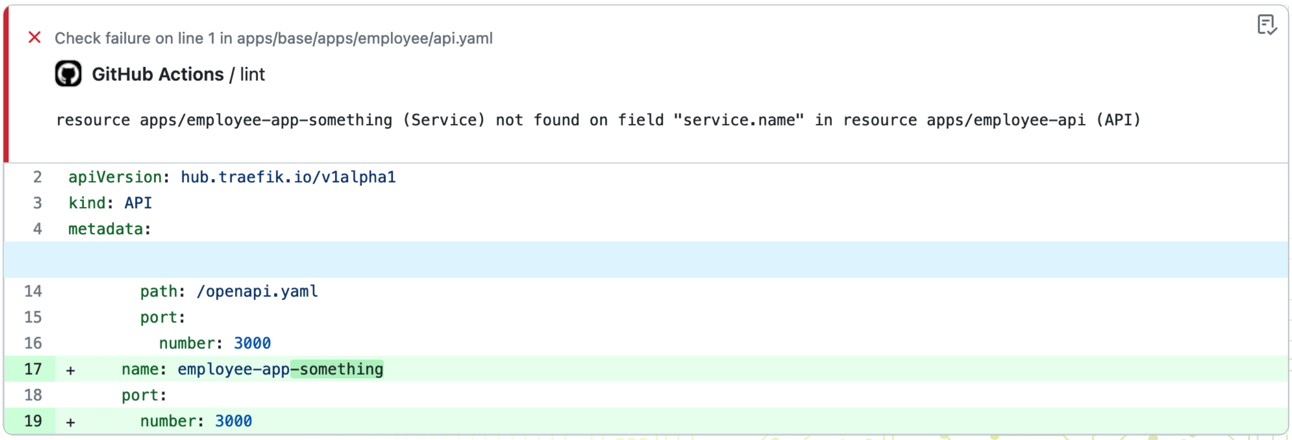
```
---
apiVersion: traefik.io/v1alpha1
kind: Middleware
metadata:
  name: coraza-waf-block-admin-path
  namespace: apps
spec:
  plugin:
    coraza:
      directives:
        - SecRuleEngine On
        - SecDebugLog /dev/stdout
        - SecDebugLogLevel 9
        - SecRule REQUEST_URI "@streq /admin"
"id:101,phase:1,log,deny,status:403"
```

Policy 4: **Provide precise API change impact analysis to minimize service failure and breaking changes**

Traefik Hub helps you understand the impact of API configuration changes by performing static checks on manifest files. It includes a Static Analyzer tool that allows you to check your Traefik Hub manifest files for consistency and quality. The Static Analyzer can lint manifest files and generate differential reports. It can be used as a CLI tool or as part of a CI/CD pipeline. The linter can detect issues such as:

- Childless resources
- Duplicate resources and resource references
- Unknown operation sets
- Orphan resources
- Invalid resource references
- Invalid regular expressions
- Duplicate releases of a given API
- Invalid resource definitions
- Invalid selector definitions

The following is an example of the linting error message displayed on the PR that introduces a change.



× Check failure on line 1 in apps/base/apps/employee/api.yaml

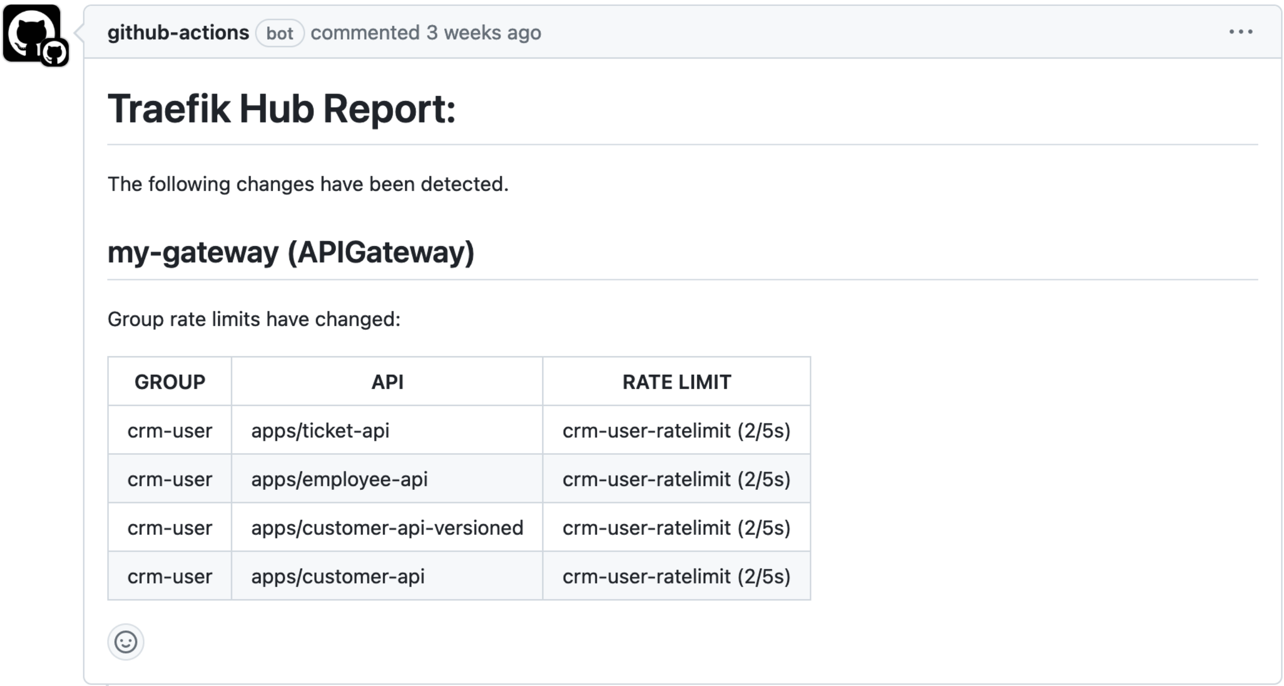
GitHub Actions / lint

```
resource apps/employee-app-something (Service) not found on field "service.name" in resource apps/employee-api (API)
```

```
apiVersion: hub.traefik.io/v1alpha1
kind: API
metadata:

        path: /openapi.yaml
        port:
          number: 3000
+     name: employee-app-something
      port:
+       number: 3000
```

The Static Analyzer tool can also run a diff to generate change reports. For example, a report like the following makes it clear that the rate limit was changed. It shows the user group and APIs affected by the change. If that was not the developer's intention, the developer can take remedial action.



Traefik Hub Report:

The following changes have been detected.

my-gateway (APIGateway)

Group rate limits have changed:

GROUP	API	RATE LIMIT
crm-user	apps/ticket-api	crm-user-ratelimit (2/5s)
crm-user	apps/employee-api	crm-user-ratelimit (2/5s)
crm-user	apps/customer-api-versioned	crm-user-ratelimit (2/5s)
crm-user	apps/customer-api	crm-user-ratelimit (2/5s)

To aid API evolution, Traefik Hub also supports several API versioning schemes. Teams can define and use an API version based on host, URI path, HTTP method, query parameters, media/content type headers, custom headers, client IP, auth user (basic/forward auth), JWT claims, and more. Traefik even allows teams to mix different versioning schemes based on logical expressions.

Policy 5: **Provide proactive API monitoring and comprehensive, granular observability to reduce time to recovery**

Traefik Hub provides metrics and tracing information using the OpenTelemetry (OTel) format, to help teams monitor and gain insights into the traffic flowing through their services and APIs. Traefik Hub has the most comprehensive OTel support in the industry, with over 20 metrics and 15 labels. Additionally, it supports vendor-specific metric systems such as Prometheus, Datadog, InfluxDB, and StatsD.

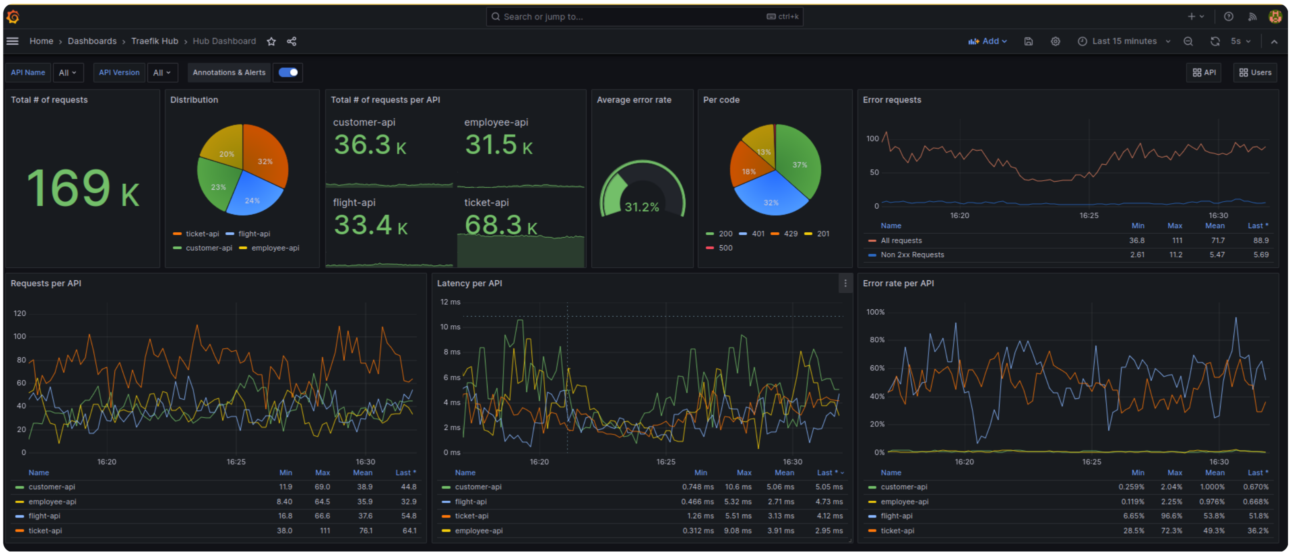
For instance, to set up the OTel metrics, first get the configuration values for the Traefik Hub Helm release:

```
helm get values traefik-hub -n traefik | tail -n +2 > values.yaml
```

Then, modify and apply the configuration values in values.yaml.

```
metrics:
  # Disable Prometheus (enabled by default)
  prometheus: null
  # Enable providing OTel metrics
  otlp:
    enabled: true
    http:
      enabled: true
      endpoint: http://myotlpcollector:4318/v1/metrics
  # Enable providing OTel traces
  tracing:
    enabled: true
    http:
      enabled: true
      endpoint: http://myotlpcollector:4318/v1/traces
```

By integrating with monitoring systems, you can configure alerts and notifications for critical metrics and performance indicators. Traefik can also be connected to pre-built Grafana dashboards to visualize a wide range of API metrics. This allows you to correlate deployment events with incident management, enhancing your ability to monitor and respond to issues effectively and improve MTTR. For more details on Traefik Hub's integration with Grafana, see this webinar.



Error requests

Name	Min	Max	Mean	Last *
All requests	36.8	111	71.7	88.9
Non 2xx Requests	2.61	11.2	5.47	5.69

Requests per API

Name	Min	Max	Mean	Last *
customer-api	11.9	69.0	38.9	44.8
employee-api	8.40	64.5	35.9	32.9
flight-api	16.8	66.6	37.6	54.8
ticket-api	38.0	111	76.1	64.1

Latency per API

Name	Min	Max	Mean	Last *
customer-api	0.748 ms	10.6 ms	5.06 ms	5.05 ms
flight-api	0.466 ms	5.32 ms	2.71 ms	4.73 ms
ticket-api	1.26 ms	5.51 ms	3.13 ms	4.12 ms
employee-api	0.312 ms	9.08 ms	3.91 ms	2.95 ms

Error rate per API

Name	Min	Max	Mean	Last *
customer-api	0.259%	2.04%	1.000%	0.670%
employee-api	0.119%	2.25%	0.976%	0.668%
flight-api	6.65%	96.6%	53.8%	51.8%
ticket-api	28.5%	72.3%	49.3%	36.2%

Additionally, Traefik provides a built-in dashboard that offers a real-time view of the current status of routes, services, middleware, and other components.

GitOps Implementation Example

The first policy we highlighted at the beginning of this article was about providing a developer-friendly, traceable, and easy-to-rollback approach to API deployment. We also mentioned in Standard 1.2 that GitOps was an implementation of this approach. In the Traefik Hub GitOps Tutorial project in GitHub, we provide a hands-on tutorial to demonstrate the power of GitOps for API configuration deployment and change management. It also demonstrates Traefik Hub's integration with API monitoring and observability visualization tools. The tutorial uses the following tools:

Aspect	Tool
API Management	Traefik Hub
Software platform	Kubernetes cluster, running locally in kind or k3d
GitOps Deployment Tool	FluxCD
Git repository	GitHub
Metrics visualization	Grafana
API monitoring and alerting	Prometheus

To work on this project, request a Traefik Hub trial here. Then follow the instructions on the README page, clone the repository, and set up kind and FluxCD. In the tutorial, FluxCD watches your fork of the GitHub repository for changes. When a change to the master branch is detected, it deploys the latest version of the configuration file to the cluster.

Summary

In this article, we introduced five essential policies for effective runtime API governance:

- A developer-friendly API deployment process
- Comprehensive and correct API documentation available to users
- Secure both ingress and egress API traffic,
- Clear API change impact analysis
- Proactive API monitoring and deep observability.

By establishing standards based on these policies, organizations can enforce runtime API governance more effectively. The primary objective of improving runtime API governance is to improve software delivery performance (as measured by the DORA metrics - DR, LTC, CFR, and MTTR). It also aims to enhance the API developer experience for both API producers and API consumers.

To successfully implement these runtime governance policies, it's crucial to leverage a modern API management platform, such as Traefik Hub, which emphasizes dynamic change management and runtime API governance at its core. As a Kubernetes native API management solution, Traefik Hub is optimized for end-to-end API delivery automation which improves operational efficiency. To learn more, get a personalized demo of Traefik Hub today.

træfiklabs