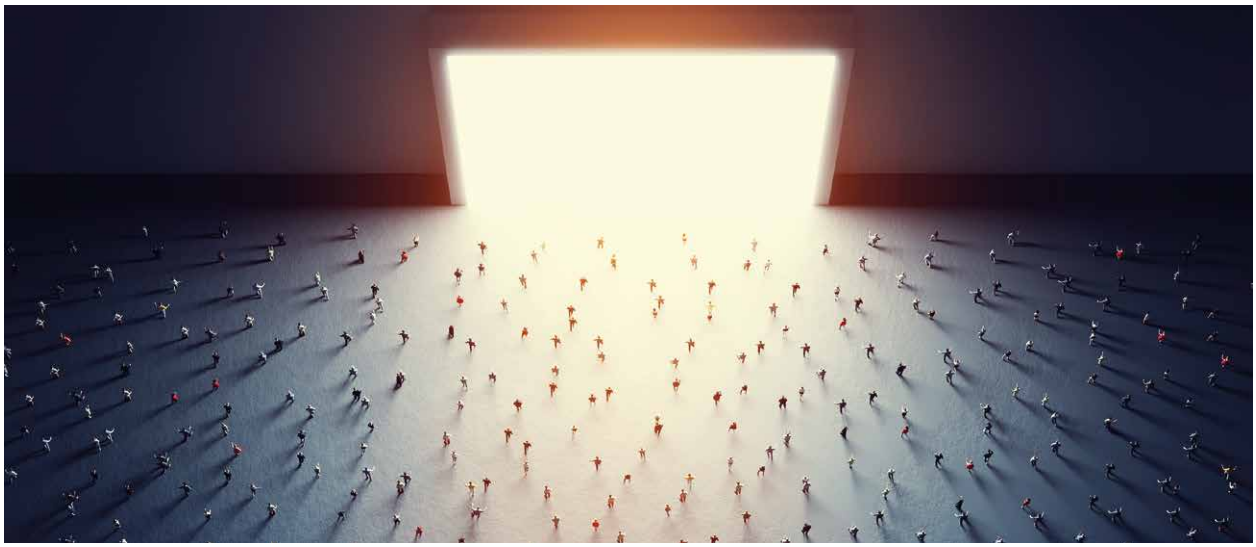

Der Schlüssel zu Sicherheit und Effizienz für moderne Anwendungen

API Gateways in Kubernetes

In diesem Artikel wollen wir zeigen, warum API Gateways im Kubernetes-Umfeld mehr sind als nur ein Einstiegspunkt – wir bieten einen Blick auf die Synergien und Potenziale.

von Sven Bernhardt

Kubernetes, oder kurz K8s, ist heute der De-facto-Industriestandard für die Bereitstellung und Verwaltung von containerisierten Anwendungen. Denn moderne Applikationen werden meist Cloud-Native und im Stil von Microservices-Architekturen entwickelt. Doch wie lassen sich diese in autonome Services aufgeteilten Anwendungen sicher und effizient für Zugriffe von außerhalb eines Kubernetes-Clusters bereitstellen? Ein API Gateway kann hier Abhilfe schaffen und gleichzeitig Entwicklungsteams entlasten sowie Transparenz schaffen. Wie genau? Das sehen wir uns im Folgenden an.

Bevor wir zum eigentlichen Thema kommen, noch einmal kurz der Hinweis, was ein API Gateway ist und warum es eine zentrale Architekturkomponente darstellt. API Gateway ist ein Architekturpattern [1] und zielt auf die Bereitstellung eines zentralen Einstiegspunkts für externe API-Nutzer ab. Grundlegend hat das erstmal wenig mit Technik zu tun. In der konkreten Ausprägung übernimmt es ähnliche Funktionen wie ein Reverse Proxy. Es reicht Anfragen an die passenden Backend-Services weiter und kümmert sich um querschnittliche Funktionen wie TLS-Terminierung, Authentifizierung, Autorisierung, Load Balancing, Caching und Logging.

Für Entwicklerteams bedeutet das mehr Produktivität und Zufriedenheit, weil wiederkehrende technische Aspekte zentral abgedeckt werden und nicht mehr wiederholt implementiert werden müssen. Ein API Gateway unterstützt zudem die Governance, indem es den Aufbau eines zentralen Policy-Managements fördert und für Transparenz bei der API-Bereitstellung und -Verwendung sorgt.

Kubernetes Service Management und die Herausforderung mit Load Balancers

Um Applikationen von außerhalb eines Kubernetes-Clusters aufrufen zu können, wird ein Kubernetes Service vom Typ Load Balancer benötigt. In Cloud-Umgebungen wird hierbei eine Load-Balancer-Instanz gestartet – wenn wir nicht aufpassen, wird im schlimmsten Fall eine Load-Balancer-Instanz pro Applikation gestar-

tet. Das ist eine kostspielige und unpraktische Lösung! Hier setzt das Konzept eines zentralen Reverse Proxys an, der in Kubernetes über einen Load Balancer Service erreichbar gemacht wird und Routing, TLS-Terminierung und Load Balancing übernimmt.

Der Vorteil der in **Abbildung 1** dargestellten Architektur ist, dass nur ein Load Balancer für den Proxy benötigt wird. Die in Kubernetes bereitgestellten Applikationen sind über den Proxy erreichbar.

Kubernetes Ingress und Gateway API – Kubernetes-eigene Konzepte zur Steuerung des Zugriffs

In Kubernetes wurde das Proxy-Konzept weiterentwickelt. Heute wird häufig ein Ingress Controller zur Verwaltung externer Zugriffe auf Services genutzt. Ein typischer Ingress Controller erlaubt Routing auf Basis von Host- und Pfadinformationen und wird Kubernetes-nativ über das Ingress-Objekt [2] konfiguriert (Listing 1).

Seit kurzem steht außerdem das Gateway API zur Verfügung [3], das über HTTP hinaus Protokolle wie gRPC, TCP und UDP unterstützt und mehr Konfigurationsoptionen bietet (Listing 2).

Beide Spezifikationen haben ihre Berechtigung und werden parallel unterstützt – je nach Anwendungsfall kann Ingress oder das Gateway API die passende Lösung sein. Am Markt gängige Ingress Controller unterstützen neben Ingress in der Regel das Gateway API [4].

Abbildung 2 stellt dar, wie sich ein Ingress Controller in Kubernetes integriert und wie entsprechende Regeln über zum Beispiel Ingress- oder *HTTPRoute*-Definitionen konfiguriert werden können.

Wir halten fest: Kubernetes-Applikationen können via Ingress oder Gateway API einfach und effizient für die externe Nutzung verfügbar gemacht werden.

Listing 1

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: httpbin
  namespace: demo
  annotations:
    ingress.kubernetes.io/ssl-redirect: "false"
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: httpbin
            port:
              number: 80
```

API Gateway vs. Ingress Controller: ein Vergleich der Funktionen

Dem aufmerksamen Leser wird es aufgefallen sein: Mit API Gateway und Ingress Controller stehen unterschiedliche Lösungsbausteine zur Verfügung, die grundlegend dasselbe Problem adressieren.

Während Ingress oder das Gateway API die grundlegenden Funktionen wie Load Balancing, Routing und

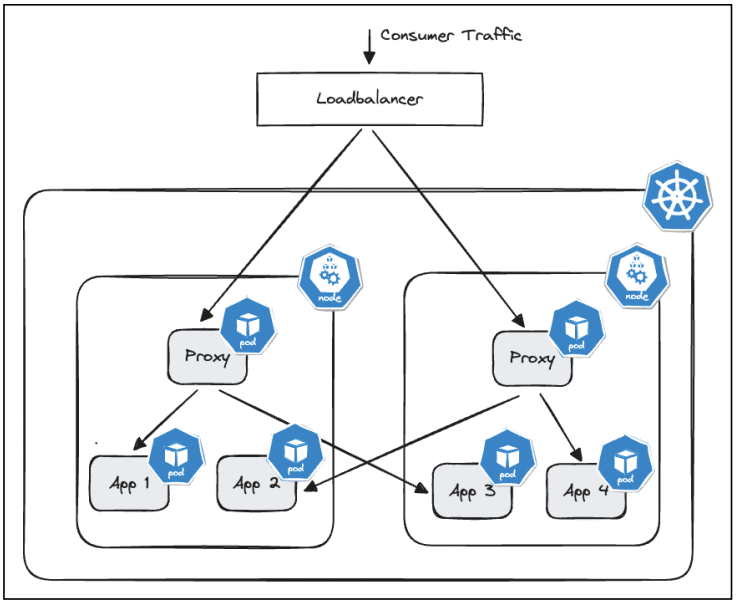

Abb. 1: Reverse-Proxy-Konzept in Kubernetes

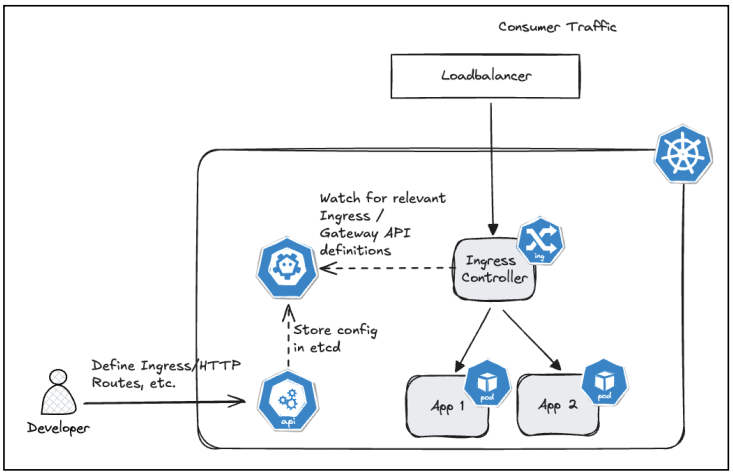

Abb. 2: Ingress Controller in Kubernetes

Listing 2

```
apiVersion: gateway.networking.k8s.io/
v1beta1
kind: HTTPRoute
metadata:
  name: httpbin
  namespace: demo
spec:
  parentRefs:
  - name: nginx-gateway
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /
    backendRefs:
    - name: httpbin
      port: 80
```

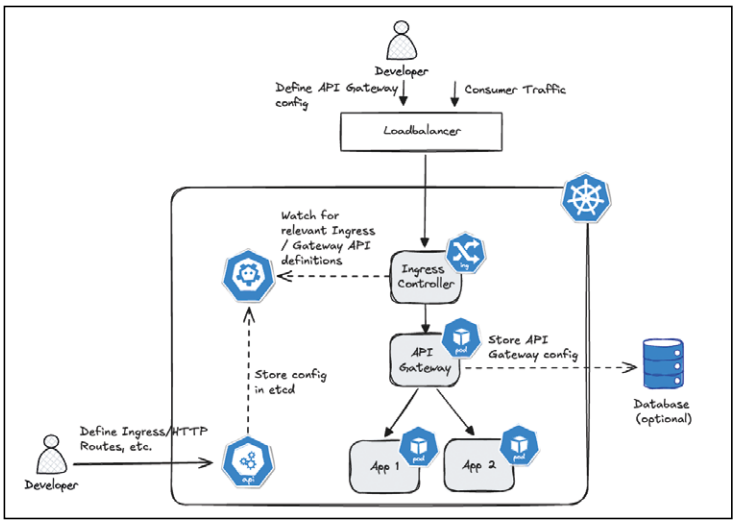

Abb. 3: API Gateway mit agnostischer Architektur in Kubernetes

SSL/TLS-Terminierung abdecken, bietet ein API Gateway umfassendere Fähigkeiten wie Authentifizierung, Autorisierung, Rate Limiting, Caching und Observability. Tabelle 1 zeigt die Unterschiede.

Wie dargestellt, übernimmt ein API Gateway mehr als nur das einfache Routing und kann ein zentraler Baustein für ein umfassendes API-Management werden.

Wer bereits mit Ingress Controllern gearbeitet hat, wird aus eigener Erfahrung wissen, dass die Grenzen technisch verschwimmen. Denn teilweise implementieren Ingress Controller Funktionalitäten, die in Tabelle 1 einem API Gateway zugeschrieben werden. Wichtig ist daher, stets zu wissen, was das Ziel ist.

Die Qual der Wahl

Architekten und Entwickler müssen anhand von Anforderungen und Use Cases entscheiden, welches Konzept oder welche Technologie die bestehenden Herausforderungen am besten löst. Wie immer gibt es an dieser Stelle keine pauschale Lösung. Geht es ausschließlich um die Erreichbarkeit von Kubernetes-Applikationen außerhalb des Clusters und existieren keine weitergehenden Anforderungen an API-Management oder IT-Sicherheit, wird in vielen Fällen ein Ingress Controller ausreichen.

Capability	Ingress Controller	API Gateway
Load Balancing	Ja	Ja
Routing	Ja	Ja
SSL/TLS-Terminierung	Ja	Ja
Authentifizierung	Nein	Ja
Autorisierung	Nein	Ja
Rate Limiting	Nein	Ja
Caching	Nein	Ja
Request-/Response-Validierung	Nein	Ja
Observability	Nein	Ja

Tabelle 1: Gegenüberstellung der Capabilities von Ingress Controller und API Gateway

Viele IT-Systemlandschaften sind heute aber heterogen aufgestellt und basieren auf unterschiedlichen, auch nicht containerbasierten Plattformen. Zudem verteilt sich die Infrastruktur. Der Weg geht von reinen On-Premises-Lösungen hin zu hybriden und Multi-Cloud-Architekturen. APIs existieren ebenfalls bereits, aber häufig ohne jegliche Visibilität und Management.

Um die Gefahr einzudämmen, dass sich die APIs zu einem unkontrolliert wachsenden und undurchsichtigen Dschungel entwickeln, sollten Architekten den Einsatz eines API Gateway in Erwägung ziehen. Und dieses API Gateway sollte die notwendige Flexibilität mitbringen, um den Anforderungen einer verteilten, heterogenen Landschaft gerecht zu werden.

API Gateways und Kubernetes

API Gateways sind heute bereits häufig Bestandteil vieler IT-Systemlandschaften. Viele Unternehmen verwenden heute noch API Gateways, die auf älteren Architekturen basieren, wie etwa das Axway oder Layer 7 API Gateway. Sie stellen uns vor typische Herausforderungen, wie wir sie von Legacy-Applikationen kennen. Es geht also um Themen wie Skalierbarkeit, dynamische Konfiguration, Automatisierbarkeit etc. Zudem sind ältere API Gateways meist ressourcenintensiv, was CPU- und Speichernutzung angeht. Daher stoßen sie in hybriden und Multicloud-Umgebungen häufig an Grenzen. Auch sind sie in der Regel nicht konform mit Cloud-nativen Prinzipien (wie den Twelve-Factor-App-Regeln [5]) und lassen sich in hybriden oder Multi-Cloud-Szenarien schlecht betreiben. Für den Betrieb in dynamischen Umgebungen wie Kubernetes sind diese also nicht geeignet.

Es kann klar gesagt werden: In Zeiten von Containern, Kubernetes und Multi-Cloud ist ein Umdenken nötig und auch zu beobachten. Cloud-native oder sogar -agnostische API Gateways – etwa von Kong, Gravitee.io oder KrakenD – bieten hier Flexibilität und können sogar direkt in Kubernetes betrieben werden. So ein Gateway wird über ein Kubernetes Deployment beschrieben und läuft, wie in **Abbildung 3** dargestellt, in Form von Pods im Cluster. Die Erreichbarkeit erfolgt über Ingress- oder *HTTPRoute*-Definitionen, während Routing- und Policy-Einstellungen über ein Administrations-API gesteuert werden.

Also alles gut, oder?

Nun ja, bis auf einige Einschränkungen … Aus operativer Sicht ist der in **Abbildung 3** dargestellte architektonische Aufbau nicht optimal. Das Gateway ist zwar auf Cloud-Native-Prinzipien aufgebaut, allerdings bedeutet das nicht, dass es auch konform mit den Kubernetes-Mechanismen und Konzepten ist und diese optimal ausnutzt:

- *Betriebliche Komplexität:* Die Konfiguration des API Gateway muss irgendwo gespeichert werden. Entweder wird sie als Datei im Docker-Image hinterlegt, was aus Sicht der dynamischen Konfigurierbarkeit problematisch ist. Alternativ kann auch eine Daten-

bank im Kubernetes-Cluster oder extern bereitgestellt werden. Insgesamt erhöht sich hierdurch die betriebliche Komplexität.

- *Skalierbarkeit:* Die Herausforderungen im Bereich der Konfiguration wirken sich auch negativ auf die Skalierbarkeit aus. Vor allem, wenn eine Datenbank verwendet wird, da diese entsprechend mitskaliert werden muss.
- *Sicherheit:* Der dargestellte Aufbau birgt Sicherheitsrisiken. Da das Administrations-API außerhalb des Clusters verfügbar ist, um das API Gateway konfigurieren zu können, müssen sich Betriebsteams um eine Absicherung kümmern. Weiterhin muss sichergestellt werden, dass Kubernetes-Applikationen nicht am API Gateway vorbei clusterextern verfügbar gemacht werden. Das kann z. B. durch die Einführung eines Admission Controllers wie Kyverno [6] und die Definition entsprechender Regelwerke verhindert werden, erhöht aber ebenfalls die betriebliche Komplexität.

Kubernetes-native API Gateways als zukunftsfähige Lösung

Aber wie können wir die zuvor beschriebenen Herausforderungen lösen? Die Antwort ist: durch die Verwendung Kubernetes-nativer API Gateways! Solche Gateways integrieren sich optimal in das Kubernetes-Ökosystem. Sie benötigen kein separates Administrations-API, da die Konfiguration nativ über Kubernetes erfolgt. Mit Hilfe von Custom Resource Definitions (CRDs) können Policies und Services direkt in Kubernetes beschrieben werden. Der gesamte Lifecycle des API Gateways wird mit Kubernetes-Bordmitteln abgebildet – eine ideale Lösung für GitOps-Ansätze, was auch der Entwicklereffizienz zugutekommt, da das Kubernetes-Ökosystem nicht verlassen werden muss.

Weiterhin vereinfacht ein Kubernetes-natives API Gateway auch die Handhabung im Betrieb, da Standard-Kubernetes-Mechanismen genutzt werden. Werden Kubernetes-Operatoren verwendet, können betriebliche Abläufe nach der initialen Installation automatisiert und somit effektiv unterstützt werden.

Beispiele für Kubernetes-native API Gateways (**Abb. 4**) sind neben anderen Kong, Gravitee.io oder Apache APISIX. Was auffällt, ist: Softwarehersteller, die Cloud-native oder -agnostische API Gateways anbieten, bieten häufig auch Kubernetes-native Pendants an.

API Gateway in Kubernetes – sinnvoll oder nicht?

Abschließend stellt sich die Frage: Ist ein API Gateway für Kubernetes-Applikationen notwendig? Die Antwort lautet: „Es kommt darauf an." Die wahre Stärke eines API Gateway zeigt sich besonders dann, wenn das API-Management strategisch angegangen wird und Sicherheit, Transparenz und Kontrolle gefragt sind.

Ein API Gateway kann Unternehmen helfen, die zunehmende Komplexität in heterogenen IT-Systemlandschaften, die sich zunehmend in Richtung Cloud beziehungsweise Multi-Cloud bewegen, zu bewältigen.

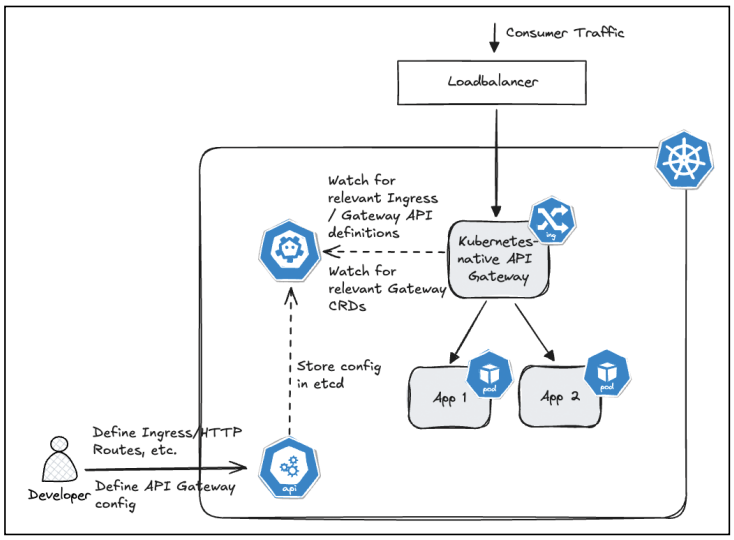

Abb. 4: Kubernetes-natives API Gateway

Gerade in einer Zeit, in der Microservices das bevorzugte Stilmittel sind, wird es wichtiger, Überblick über vorhandene APIs und deren Nutzungsverhalten zu haben, Sicherheitsrichtlinien konsistent durchzusetzen und Kommunikationsverläufe sichtbar zu machen. Kubernetes ist die elementare Plattform, um solche verteilten Applikationsarchitekturen effizient zu betreiben und weiterzuentwickeln. Durch den zunehmenden Einsatz von KI und die Integration von Large Language Models (LLMs) in Applikationen erhöht sich stetig die Anzahl der zu managenden APIs. Das bedeutet, auch die Notwendigkeit für ein konsistentes Management wächst weiter.

Mein Plädoyer daher: Heute schon an morgen denken: Ein API Gateway in Kubernetes könnte der erste Schritt in Richtung eines umfassenden API-Managements sein – und langfristig die Basis für eine stabile und zukunftsfähige IT-Infrastruktur!

Sven Bernhardt ist Chief Architect bei OPITZ CONSULTING mit Fokus auf Modernisierung und Integration bestehender IT-Systeme. Insbesondere am Herzen liegen ihm API-Management und Microservices.

Links & Literatur

[1] https://microservices.io/patterns/apigateway.html

[2] https://kubernetes.io/docs/concepts/services-networking/ingress/

[3] https://gateway-api.sigs.k8s.io

[4] https://gateway-api.sigs.k8s.io/implementations/

[5] https://12factor.net/de/

[6] https://kyverno.io