Simplify Your K8s Connectivity with NGINX Gateway Fabric

Cody Green – Sr. Dir. Global Solution Architects
Mike Stefaniak – Sr. Product Manager

October 22nd, 2024

Agenda

Introduction

Kubernetes Gateway API Overview

Introduction to NGINX Gateway Fabric

Demonstration

Q&A

Introduction

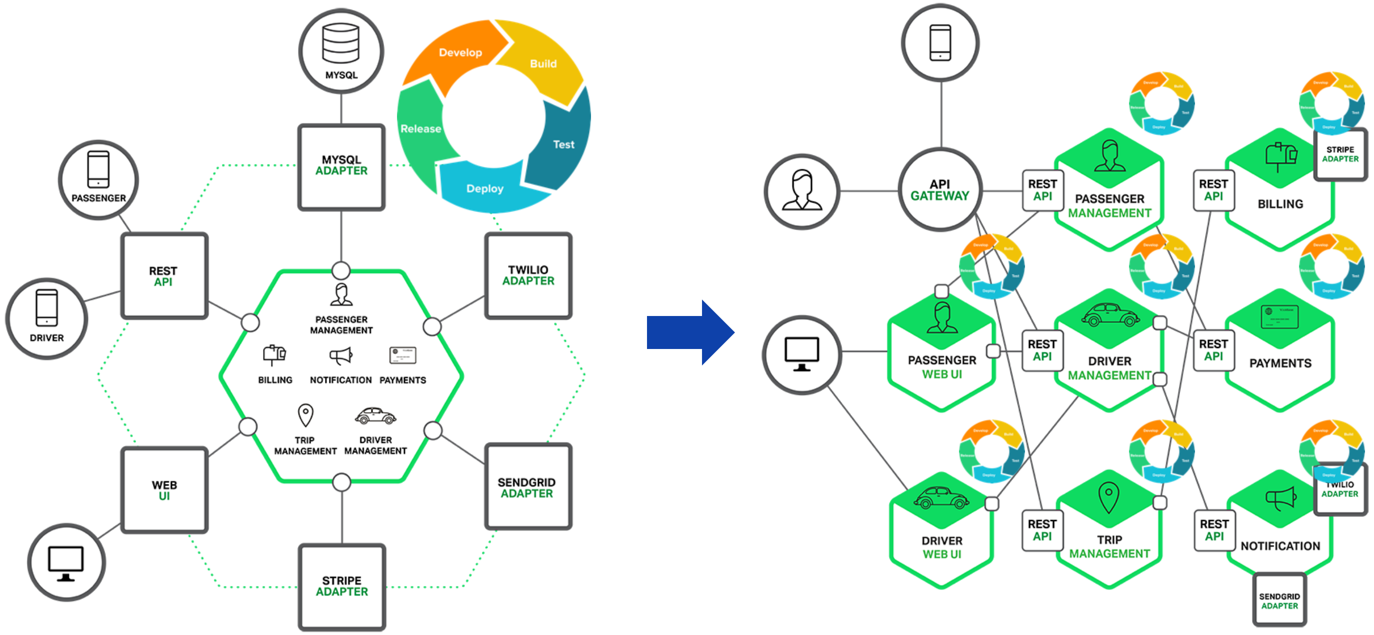
Traffic Management Challenges

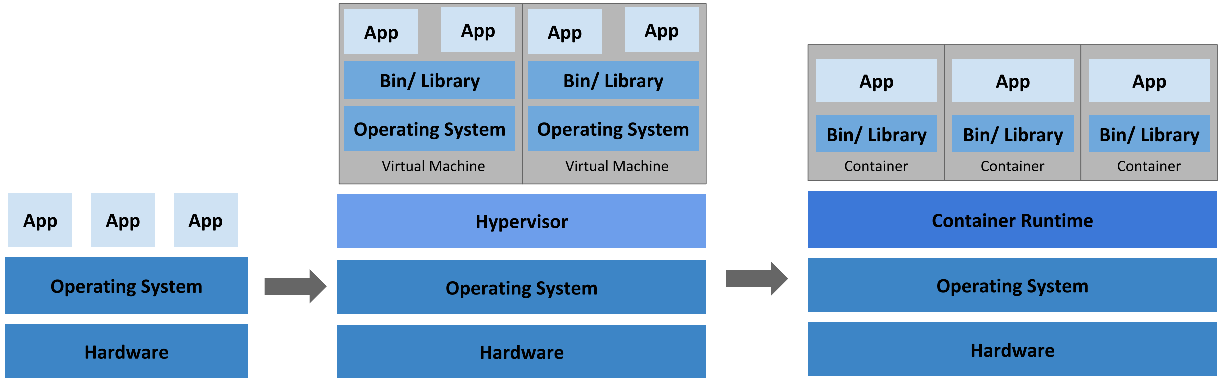
Security

Scalability

Complexity

Application Architecture and Deployment Models

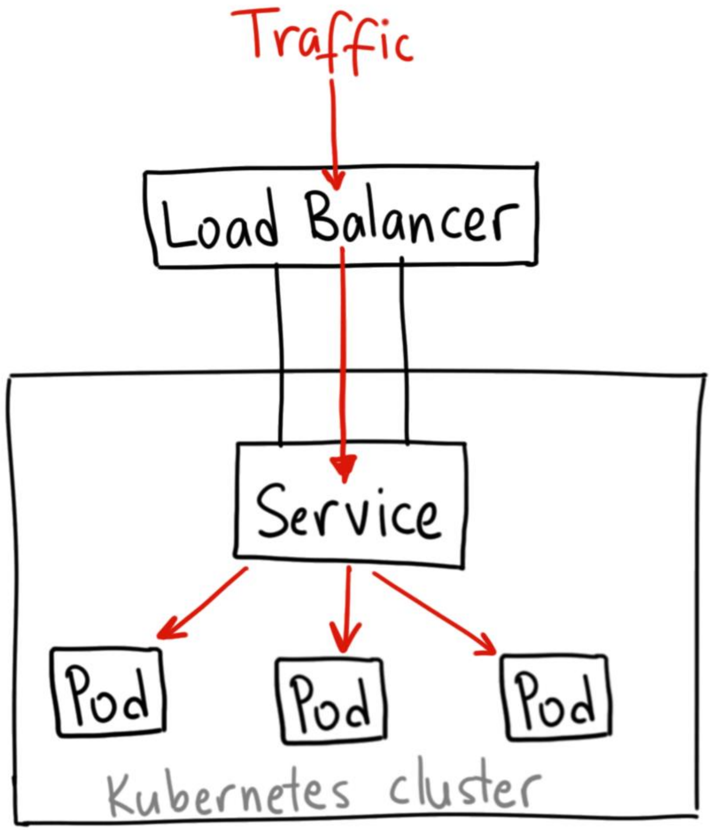




Traditional Deployment **Virtualized Deployment** **Container Deployment**

- Automates deployment and management of containerized workloads and services *at scale*
  - Deployment
  - Scaling
  - Rollouts / Rollbacks
  - Self-healing
- De facto standard for deploying microservices

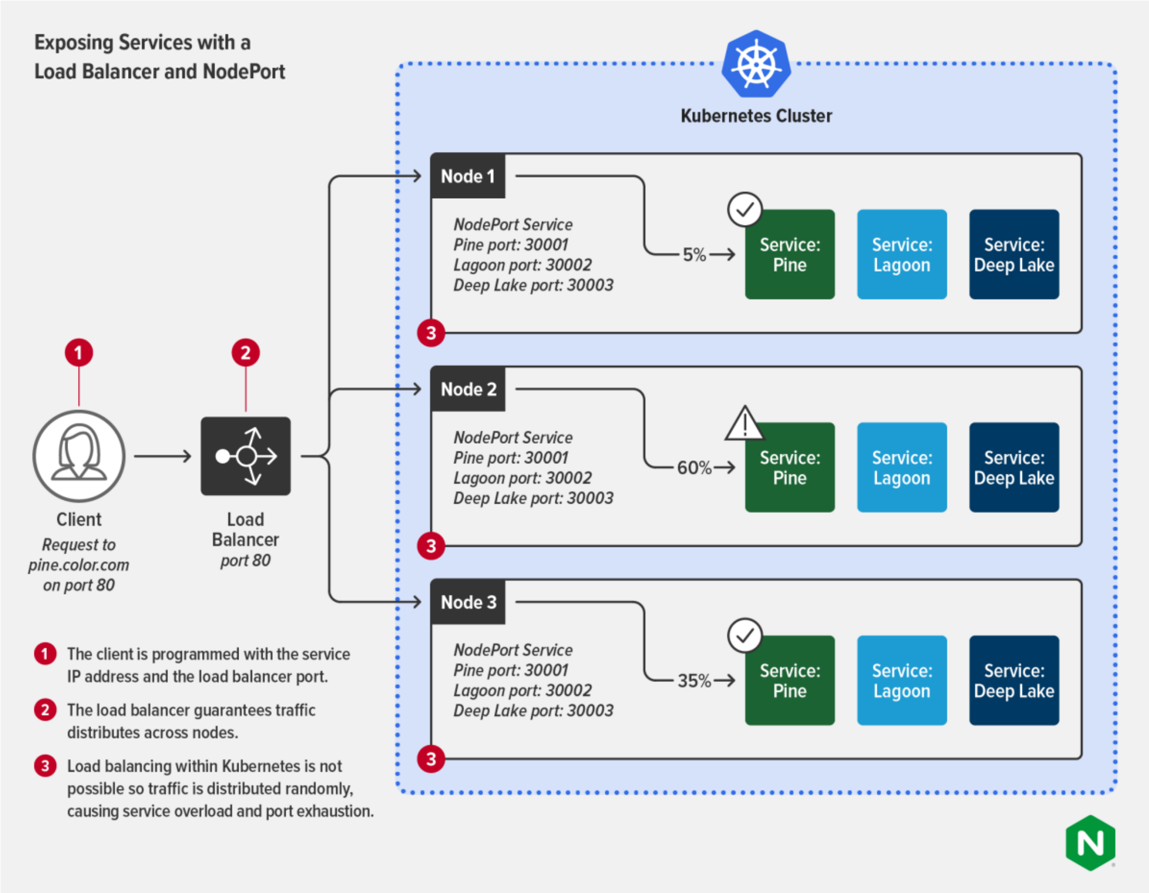
Applications in Kubernetes



- **Pods**
  - Containers run workloads on nodes in a Kubernetes cluster
- **Services**
  - A logical grouping of pods that perform the same function
- **Ingress**
  - How to access a set of Pods via a L7 load balancer (Hostname, URI)

Exposing Apps in Kubernetes

Small, static deployments



Exposing Services with a Load Balancer and NodePort

Kubernetes Cluster

1 Client
Request to pine.color.com on port 80

2 Load Balancer
port 80

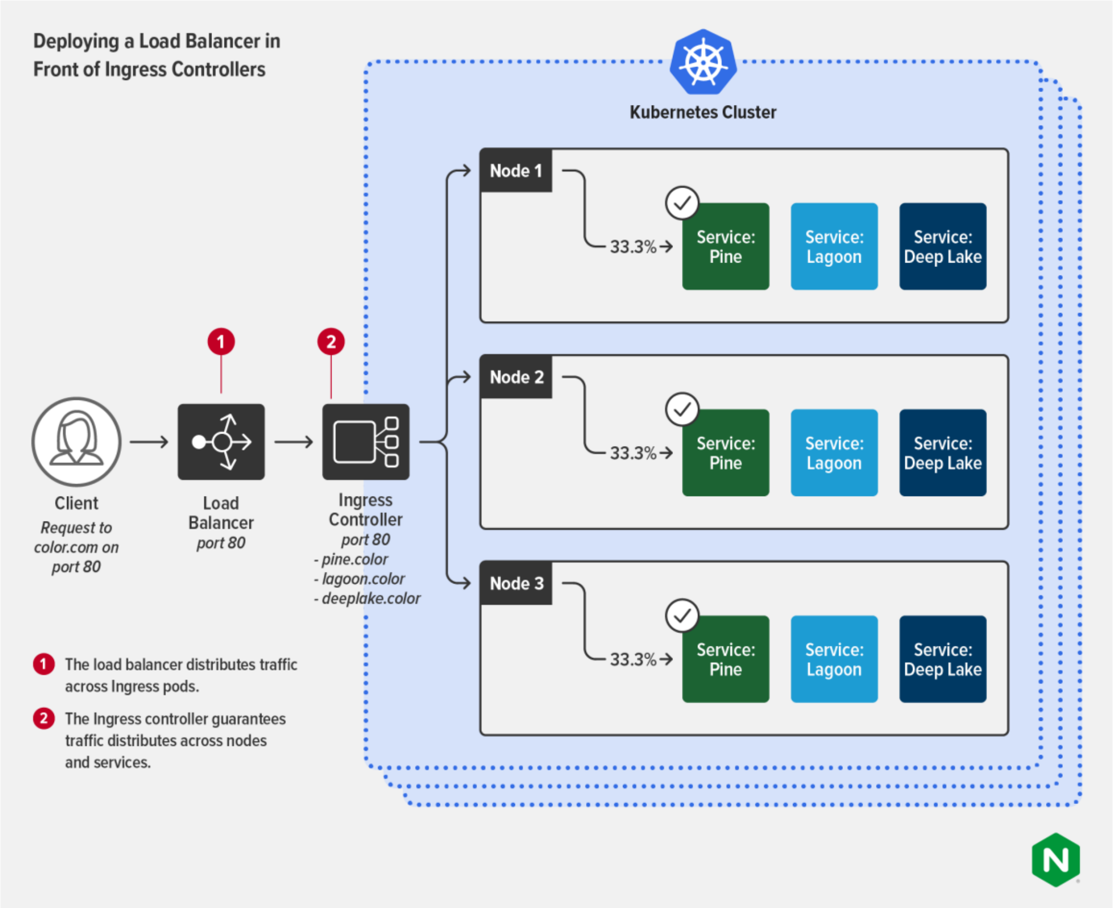
Node 1
NodePort Service
Pine port: 30001
Lagoon port: 30002
Deep Lake port: 30003
5% → Service: Pine ✓ | Service: Lagoon | Service: Deep Lake
3

Node 2
NodePort Service
Pine port: 30001
Lagoon port: 30002
Deep Lake port: 30003
60% → Service: Pine ⚠ | Service: Lagoon | Service: Deep Lake
3

Node 3
NodePort Service
Pine port: 30001
Lagoon port: 30002
Deep Lake port: 30003
35% → Service: Pine ✓ | Service: Lagoon | Service: Deep Lake
3

1. The client is programmed with the service IP address and the load balancer port.
2. The load balancer guarantees traffic distributes across nodes.
3. Load balancing within Kubernetes is not possible so traffic is distributed randomly, causing service overload and port exhaustion.

Scalable, dynamic deployments



Deploying a Load Balancer in Front of Ingress Controllers

Kubernetes Cluster

Client
Request to color.com on port 80

1 Load Balancer
port 80

2 Ingress Controller
port 80
- pine.color
- lagoon.color
- deeplake.color

Node 1
33.3% → Service: Pine ✓ | Service: Lagoon | Service: Deep Lake

Node 2
33.3% → Service: Pine ✓ | Service: Lagoon | Service: Deep Lake

Node 3
33.3% → Service: Pine ✓ | Service: Lagoon | Service: Deep Lake

1. The load balancer distributes traffic across Ingress pods.
2. The Ingress controller guarantees traffic distributes across nodes and services.

Challenges with Running Kubernetes in Production

Across hybrid, multi-cloud environments with disaggregated technologies

Connection timeouts and errors → Poor user experiences

Insufficient visibility into app health and performance → Troubleshooting difficulties and downtime

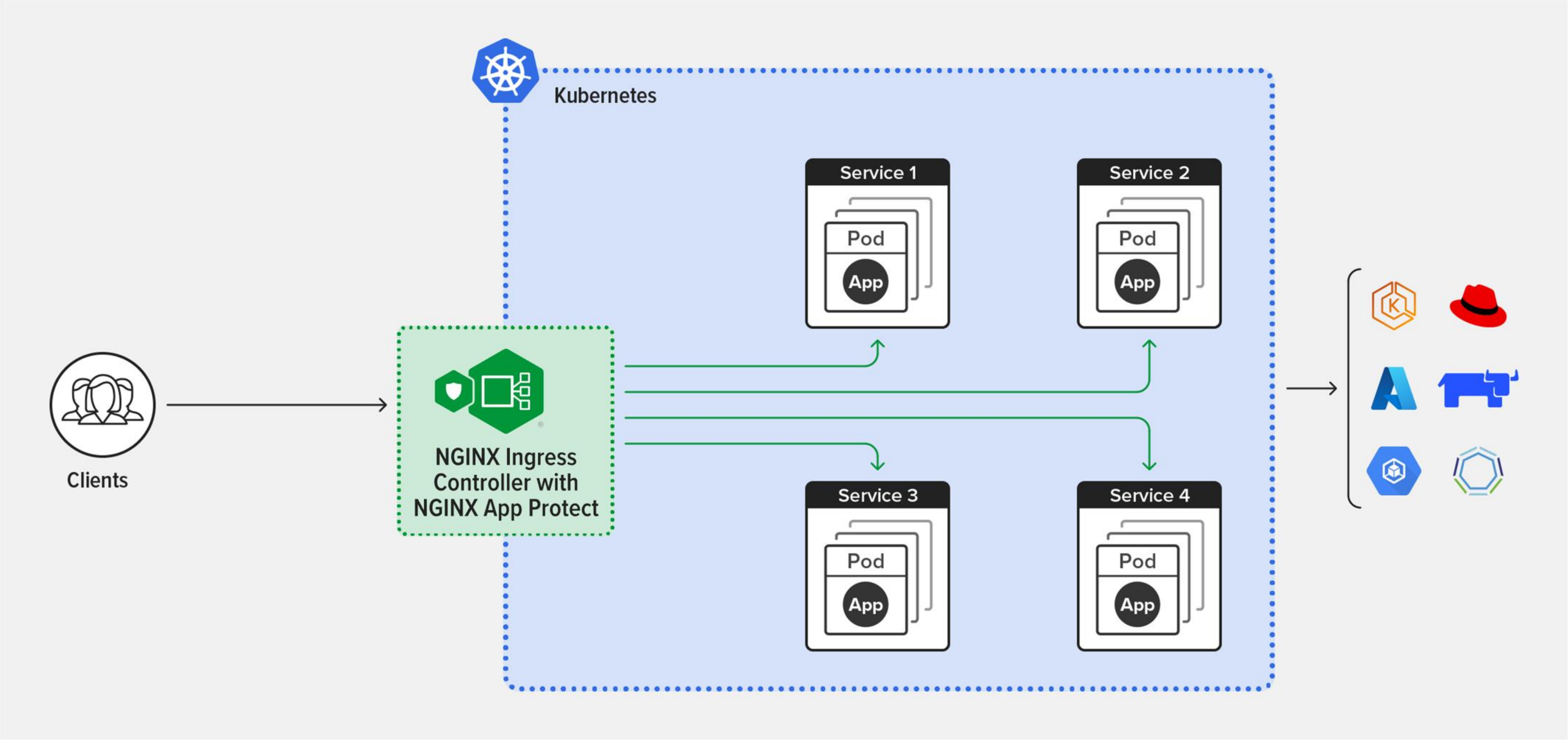
Difficulties with securing distributed app environments → Increased risk of cyberthreat exposure

Limited governance and self-service capabilities → Slow app releases for developers

Increasing complexity and tool sprawl → Hard to operate, manage, and troubleshoot

Unique Ingress Controllers

Companies like NGINX release their own Ingress Controllers to address the challenges



Enterprise-class

Secure end-to-end

Infrastructure-agnostic (Kubernetes-native)

Ingress Controller Limitations

Limited Extensibility

The Ingress API has limited functionality for configuring advanced traffic routing.

This led vendors to implement their own unique API extensions, many in the form of custom resource definitions CRDs), to add needed functionality.

This also plagued new use cases targeted at Ingress controllers.

Governance

The Ingress API does not separate responsibilities to allow platform operation teams and developers to configure only the components relevant to their scope of control.

While NGINX implemented a unique solution to this problem, the lack of a shared standard among Ingress providers remains a significant issue.

Service Mesh

While service meshes like Istio tried to fill some of these gaps, the significant complexity and operational overhead have limited their widespread adoption.

Kubernetes Gateway API Overview

The Gateway API

A Collection of Resources

- Models service networking in Kubernetes
- Works in collaboration with the Service resource

NOT an Implementation

- Only a standard to configure Gateway API features
- An implementation is responsible for all functionality
- The implementation decides what is deployed

The Gateway API

Redefining the Traditional “Ingress” Resource

- Managing "ingress" traffic (North/South)
- In-cluster traffic handled by GAMMA (East/West)

Managed by the SIG-NETWORK Community

- Kubernetes Network Special Interest Group
- Responsible for networking features of Kubernetes
- Competition collaborates to define broad standards

Solving the Problems from the Ingress API

Role Orientation

Problem: Access is all or nothing

Gateway API's Solution:

- Divide API resources by organizational role
- Engineers only deal with resources they care about

Expression

Problem: Annotations everywhere

Gateway API's Solution:

- Core support for traffic policies without custom annotations
- Can change your implementation with the same configuration

Extensibility

Problem: Custom annotations

Gateway API's Solution:

- New features can be built from extension points
- Support levels for optional features:
  - Core
  - Extended
  - Implementation-specific

Role-oriented Object Design

GatewayClass

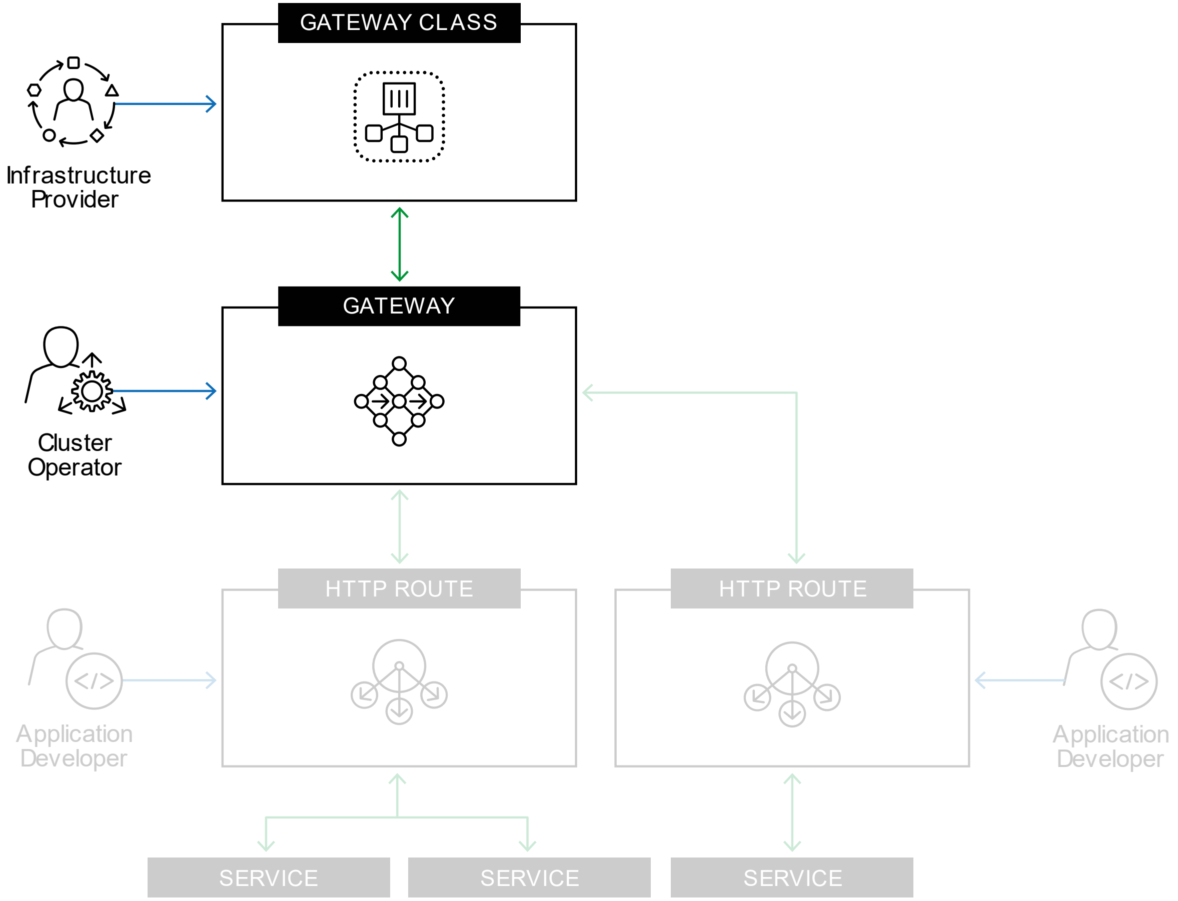
Infrastructure provider

- Defines what implementations are available in the cluster

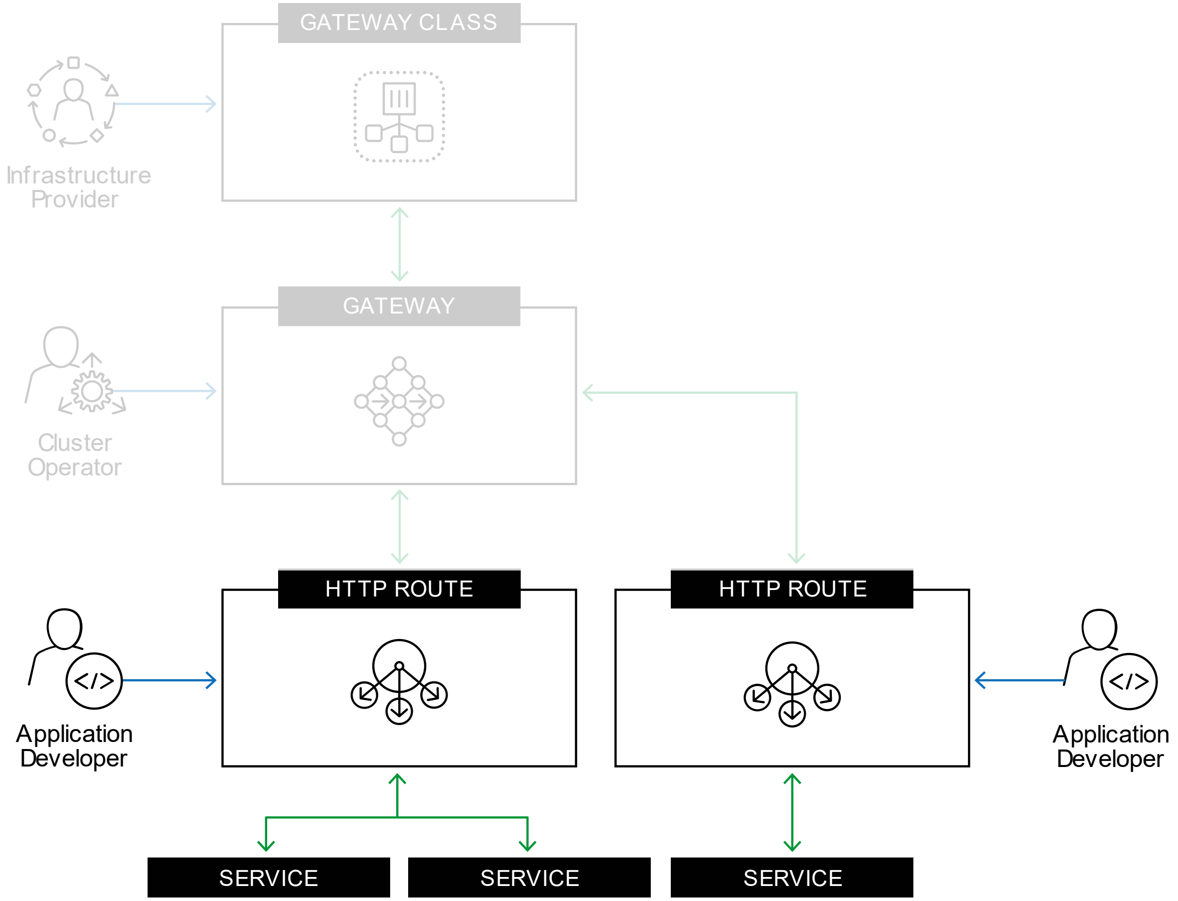
Gateway

Cluster operator

- A definition for some infrastructure to be created



Role-oriented Object Design



Routes (HTTP, TCP, TLS, UDP, gRPC)

Application developer

- Defines routing behavior from a Gateway to an application via Kubernetes services

Role-oriented Object Design

GatewayClass

Infrastructure provider

- Defines what implementations are available in the cluster

Gateway

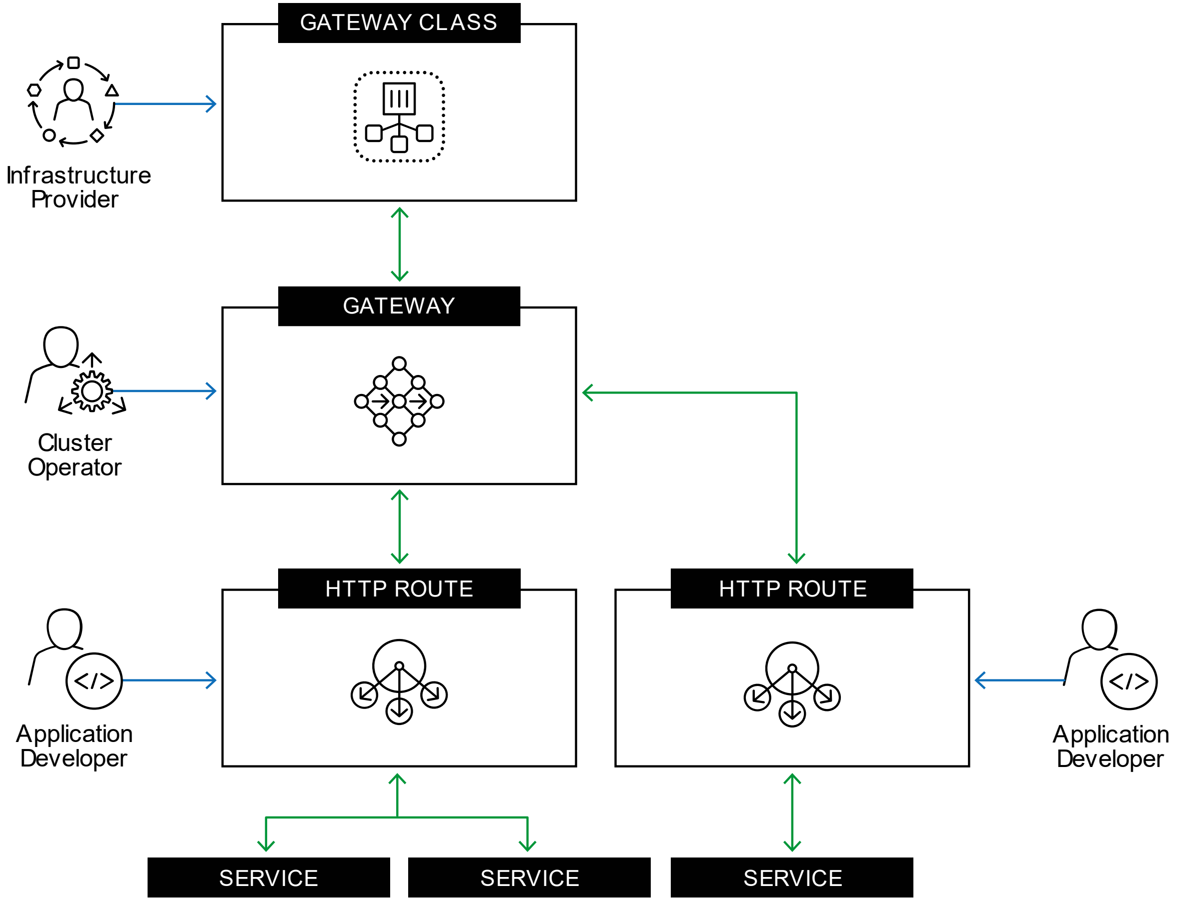
Cluster operator

- A definition for some infrastructure to be created

Routes (HTTP, TCP, TLS, UDP, gRPC)

Application developer

- Defines routing behavior from a Gateway to an application via Kubernetes services



Introduction to NGINX Gateway Fabric

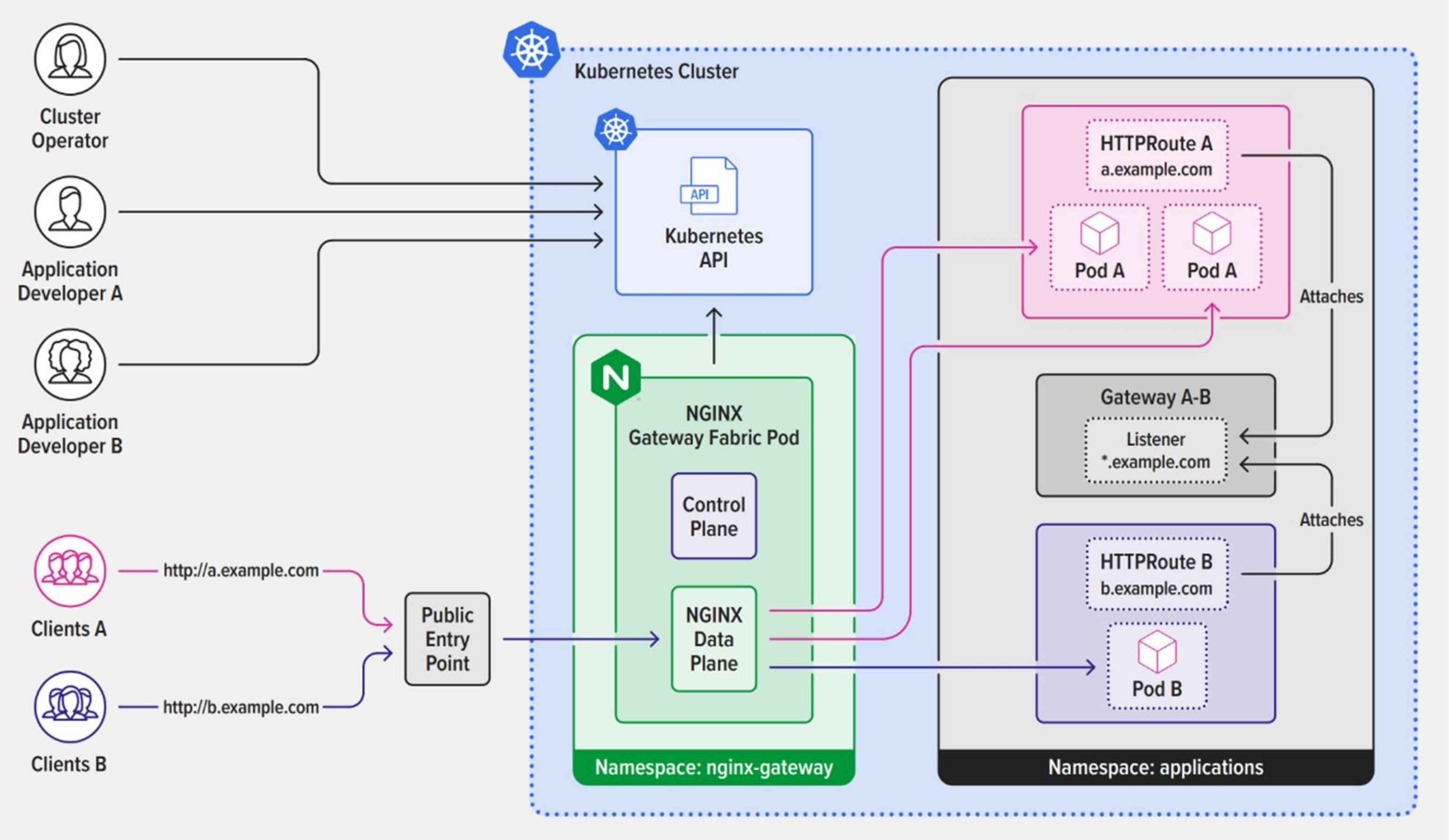
What is NGINX Gateway Fabric?

The Gateway API powered by NGINX

- Optimal performance and reliability
- All traffic processing is handled by native NGINX or official modules.

A Stand-alone Gateway API Product

- All functionality expressed through Gateway API
- NGINX directives via extensions and attachments
- Snippet functionality for any directive not yet “first-class”



Scenarios and Use Cases

NGINX Gateway Fabric leverages core Gateway API features and NGINX capabilities

Attribute-based routing

HTTP redirects, rewrites, header modification

"Zero-downtime" deployment strategies

Observability

Access Control

TLS encryption

Weight-based Load balancing

Roadmap

OpenID Connect

Egress Controls

Secure App to App Traffic

NGINX Snippets

WAF Protection

Demo

NGINX in Kubernetes

The front door to your Kubernetes applications

NGINX Ingress Controller

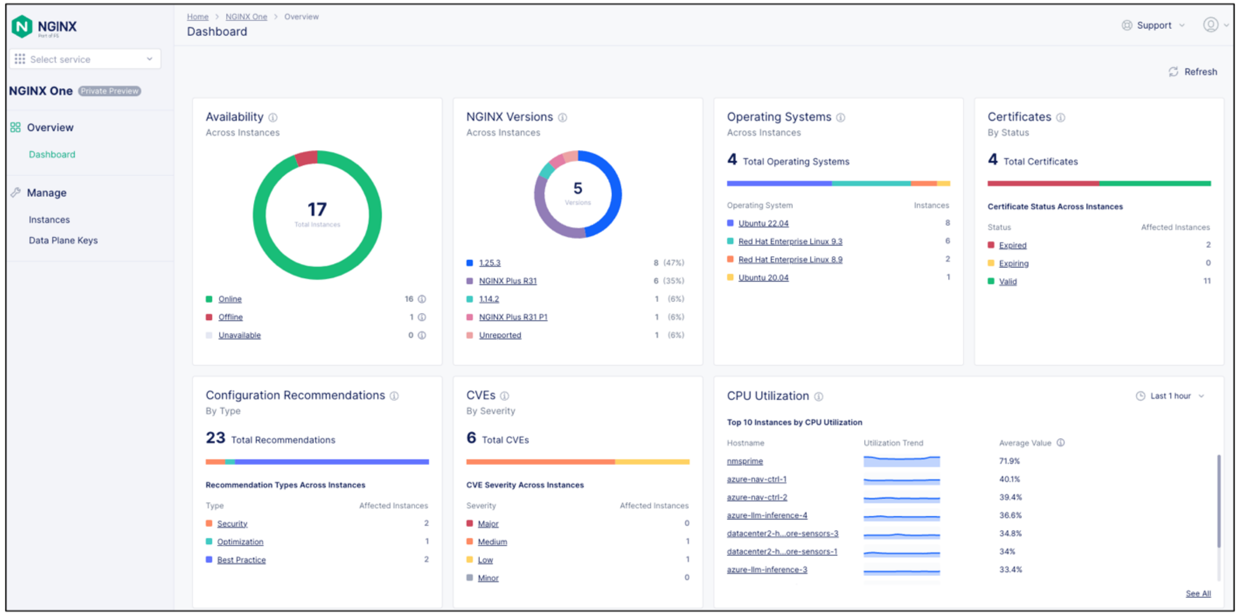
- Implements the Ingress API
- Mature product with 7 years of development
- With community version, near ubiquitous use within Kubernetes clusters

NGINX Gateway Fabric

- Implements the Gateway API
- Next generation of Ingress
- Stand-alone product built for the Gateway API

Manage NGINX instances with a SaaS-based Console or via APIs

- Get actionable insights and recommendations
- Monitor performance, uptime, and security instantly
- Integrate with your preferred tool via our easy-to-use API and OTel



Dashboard

Availability
Across Instances

Online	16
Offline	1
Unavailable	0

NGINX Versions
Across Instances

1.25.3	8	(47%)
NGINX Plus R31	6	(35%)
1.14.2	1	(6%)
NGINX Plus R31 P1	1	(6%)
Unreported	1	(6%)

Operating Systems
Across Instances

4 Total Operating Systems

Operating System	Instances
Ubuntu 22.04	8
Red Hat Enterprise Linux 9.3	6
Red Hat Enterprise Linux 8.9	2
Ubuntu 20.04	1

Certificates
By Status

4 Total Certificates

Certificate Status Across Instances

Status	Affected Instances
Expired	2
Expiring	0
Valid	11

Configuration Recommendations
By Type

23 Total Recommendations

Recommendation Types Across Instances

Type	Affected Instances
Security	2
Optimization	1
Best Practice	2

CVEs
By Severity

6 Total CVEs

CVE Severity Across Instances

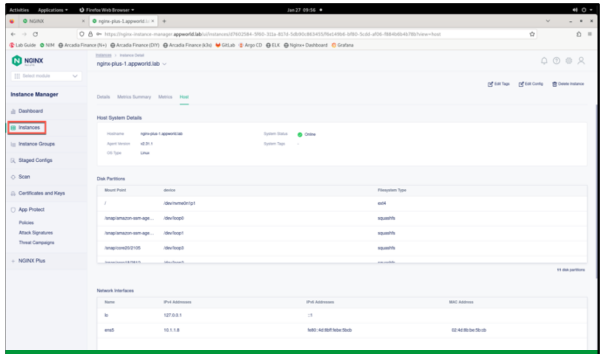
Severity	Affected Instances
Major	0
Medium	1
Low	1
Minor	0

CPU Utilization
Last 1 hour

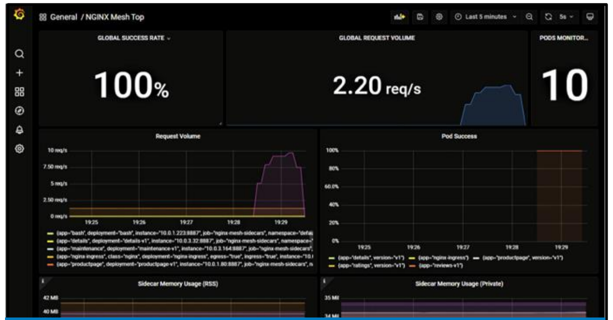
Top 10 Instances by CPU Utilization

Hostname	Utilization Trend	Average Value
nmsprime		71.9%
azure-nav-ctrl-1		40.1%
azure-nav-ctrl-2		39.4%
azure-llm-inference-4		36.6%
datacenter2-h...ore-sensors-3		34.8%
datacenter2-h...ore-sensors-1		34%
azure-llm-inference-3		33.4%

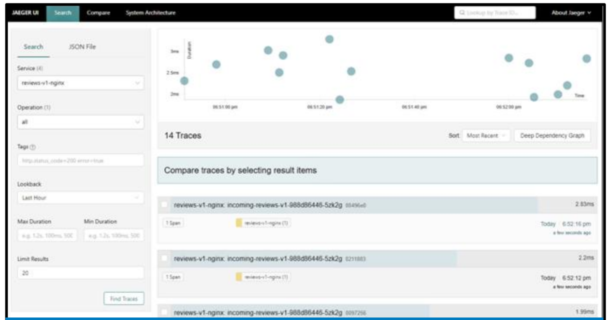
See All



NGINX Instance Manager



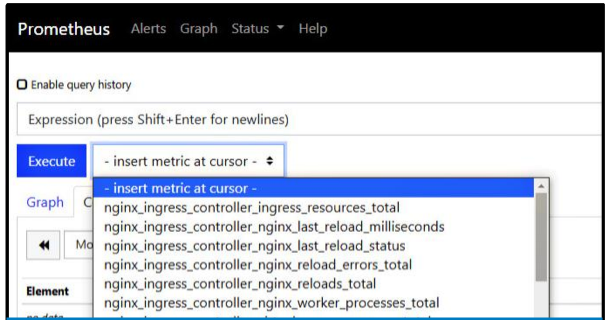
Analytics and visualization



14 Traces

Compare traces by selecting result items

Monitoring and alerting



□ Enable query history

Expression (press Shift+Enter for newlines)

- insert metric at cursor -

nginx_ingress_controller_ingress_resources_total
nginx_ingress_controller_nginx_last_reload_milliseconds
nginx_ingress_controller_nginx_last_reload_status
nginx_ingress_controller_nginx_reload_errors_total
nginx_ingress_controller_nginx_reloads_total
nginx_ingress_controller_nginx_worker_processes_total

Distributed tracing

Q&A