AWS Compute Blog

Architecting for scale with Amazon API Gateway private integrations

by James Beswick | on 26 SEP 2023 | in Amazon API Gateway, Serverless | Permalink | Share

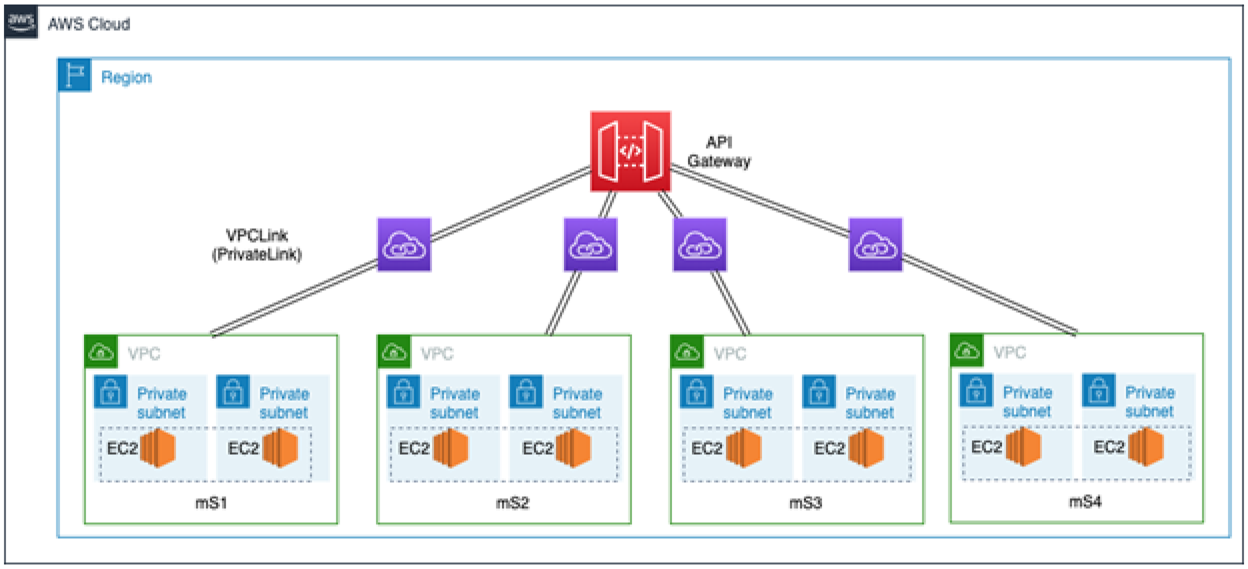
This post is written by Lior Sadan, Sr. Solutions Architect, and Anandprasanna Gaitonde, Sr. Solutions Architect.

Organizations use Amazon API Gateway to build secure, robust APIs that expose internal services to other applications and external users. When the environment evolves to many microservices, customers must ensure that the API layer can handle the scale without compromising security and performance. API Gateway provides various API types and integration options, and builders must consider how each option impacts the ability to scale the API layer securely and performantly as the microservices environment grows.

This blog post compares architecture options for building scalable, private integrations with API Gateway for microservices. It covers REST and HTTP APIs and their use of private integrations, and shows how to develop secure, scalable microservices architectures.

Overview

Here is a typical API Gateway implementation with backend integrations to various microservices:



API Gateway handles the API layer, while integrating with backend microservices running on Amazon EC2, Amazon Elastic Container Service (ECS), or Amazon Elastic Kubernetes Service (EKS). This blog focuses on containerized microservices that expose internal endpoints that the API layer then exposes externally.

To keep microservices secure and protected from external traffic, they are typically implemented within an Amazon Virtual Private Cloud (VPC) in a private subnet, which is not accessible from the internet. API Gateway offers a way to expose these resources securely beyond the VPC through private integrations using VPC link. Private integration forwards external traffic sent to APIs to private resources, without exposing the services to the internet and without leaving the AWS network. For more information, read Best Practices for Designing Amazon API Gateway Private APIs and Private Integration.

The example scenario has four microservices that could be hosted in one or more VPCs. It shows the patterns integrating the microservices with front-end load balancers and API Gateway via VPC links.

While VPC links enable private connections to microservices, customers may have additional needs:

- Increase scale: Support a larger number of microservices behind API Gateway.
- Independent deployments: Dedicated load balancers per microservice enable teams to perform blue/green deployments independently without impacting other teams.
- Reduce complexity: Ability to use existing microservice load balancers instead of introducing additional ones to achieve API Gateway integration
- Low latency: Ensure minimal latency in API request/response flow.

API Gateway offers HTTP APIs and REST APIs (see Choosing between REST APIs and HTTP APIs) to build RESTful APIs. For large microservices architectures, the API type influences integration considerations:

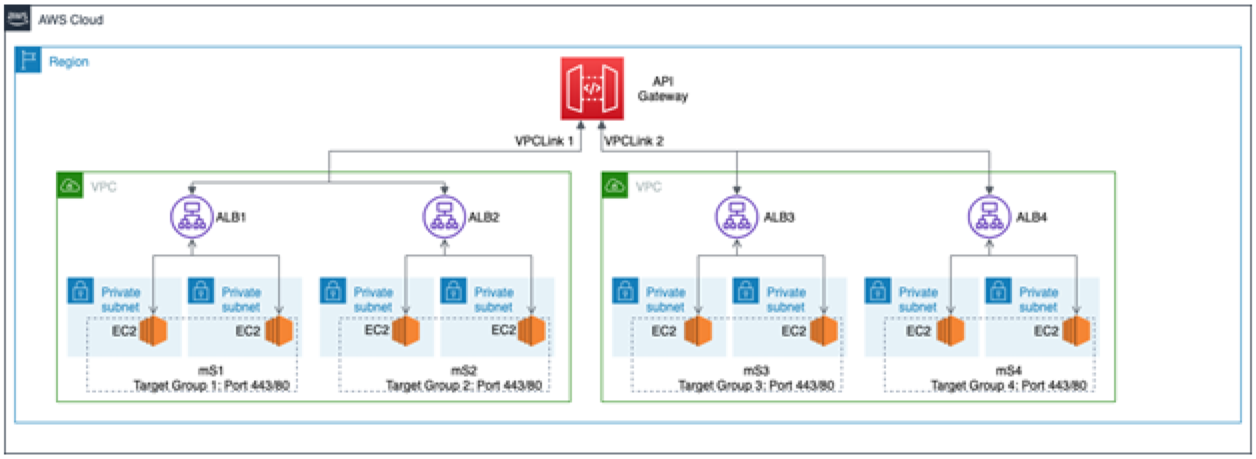
	VPC link supported integrations	**Quota on VPC links per account per Region**
REST API	Network Load Balancer (NLB)	20
HTTP API	Network Load Balancer (NLB), Application Load Balancer (ALB), and AWS Cloud Map	10

This post presents four private integration options taking into account the different capabilities and quotas of VPC link for REST and HTTP APIs:

- **Option 1:** HTTP API using VPC link to multiple NLBs or ALBs.
- **Option 2**: REST API using multiple VPC links.
- **Option 3**: REST API using VPC link with NLB.
- **Option 4:** REST API using VPC link with NLB and ALB targets.

Option 1: HTTP API using VPC link to multiple NLBs or ALBs

HTTP APIs allow connecting a single VPC link to multiple ALBs, NLBs, or resources registered with an AWS Cloud Map service. This provides a fan out approach to connect with multiple backend microservices. However, load balancers integrated with a particular VPC link should reside in the same VPC.


AWS Cloud
Region
API Gateway
VPCLink 1
VPCLink 2
VPC
ALB1
ALB2
ALB3
ALB4
Private subnet
EC2
mS1
Target Group 1: Port 443/80
mS2
Target Group 2: Port 443/80
mS3
Target Group 3: Port 443/80
mS4
Target Group 4: Port 443/80

Two microservices are in a single VPC, each with its own dedicated ALB. The ALB listeners direct HTTPS traffic to the respective backend microservice target group. A single VPC link is connected to two ALBs in that VPC. API Gateway uses path-based routing rules to forward requests to the appropriate load balancer and associated microservice. This approach is covered in Best Practices for Designing Amazon API Gateway Private APIs and Private Integration – HTTP API. Sample CloudFormation templates to deploy this solution are available on GitHub.

You can add additional ALBs and microservices within VPC IP space limits. Use the Network Address Usage (NAU) to design the distribution of microservices across VPCs. Scale beyond one VPC by adding VPC links to connect more VPCs, within VPC link quotas. You can further scale this by using routing rules like path-based routing at the ALB to connect more services behind a single ALB (see Quotas for your Application Load Balancers). This architecture can also be built using an NLB.

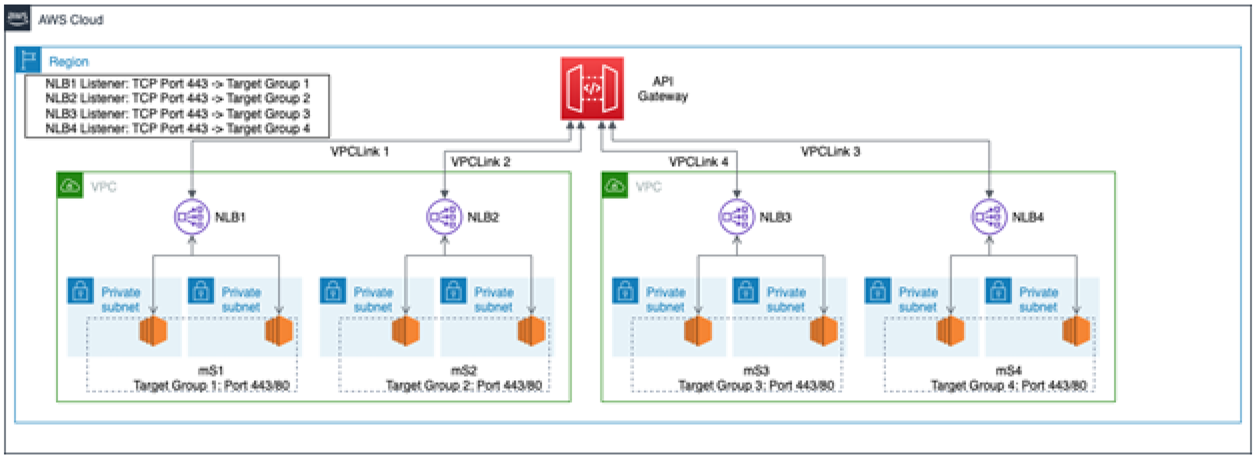
Benefits:

- High degree of scalability. Fanning out to multiple microservices using single VPC link and/or multiplexing capabilities of ALB/NLB.
- Direct integration with existing microservices load balancers eliminates the need for introducing new components and reducing operational burden.
- Lower latency for API request/response thanks to direct integration.
- Dedicated load balancers per microservice enable independent deployments for microservices teams.

Option 2: REST API using multiple VPC links

For REST APIs, the architecture to support multiple microservices may differ due to these considerations:

- NLB is the only supported private integration for REST APIs.
- VPC links for REST APIs can have only one target NLB.


AWS Cloud
Region
NLB1 Listener: TCP Port 443 -> Target Group 1
NLB2 Listener: TCP Port 443 -> Target Group 2
NLB3 Listener: TCP Port 443 -> Target Group 3
NLB4 Listener: TCP Port 443 -> Target Group 4
API Gateway
VPCLink 1
VPCLink 2
VPCLink 4
VPCLink 3
VPC
NLB1
NLB2
NLB3
NLB4
Private subnet
mS1
Target Group 1: Port 443/80
mS2
Target Group 2: Port 443/80
mS3
Target Group 3: Port 443/80
mS4
Target Group 4: Port 443/80

A VPC link is required for each NLB, even if the NLBs are in the same VPC. Each NLB serves one microservice, with a listener to route API Gateway traffic to the target group. API Gateway path-based routing sends requests to the appropriate NLB and corresponding microservice. The setup required for this private integration is similar to the example described in Tutorial: Build a REST API with API Gateway private integration.

To scale further, add additional VPC link and NLB integration for each microservice, either in the same or different VPCs based on your needs and isolation requirements. This approach is limited by the VPC links quota per account per Region.

Benefits:

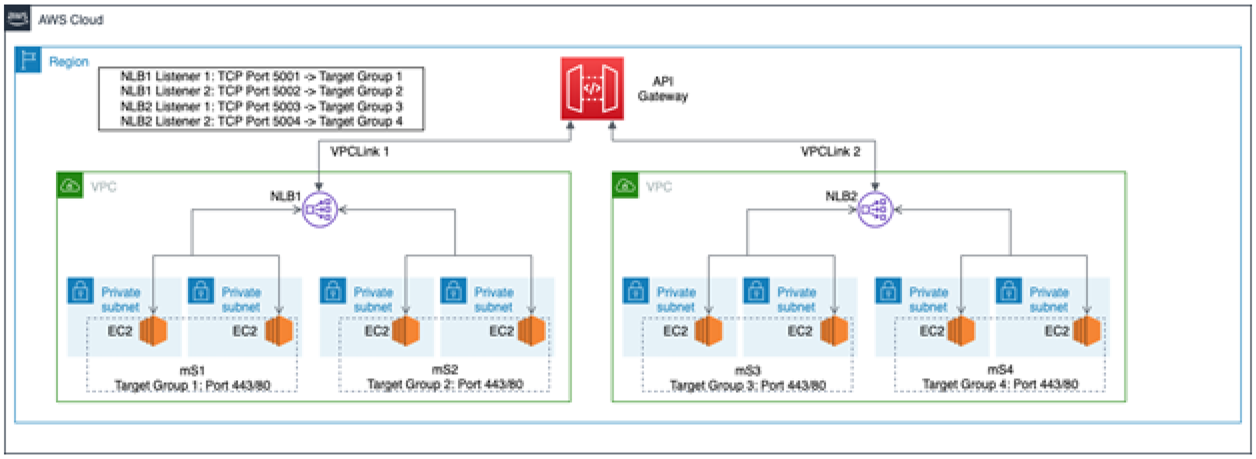
- Single NLB in the request path reduces operational complexity.
- Dedicated NLBs for each enable independent microservice deployments.
- No additional hops in the API request path results in lower latency.

Considerations:

- Limits scalability due to a one-to-one mapping of VPC links to NLBs and microservices limited by VPC links quota per account per Region.

Option 3: REST API using VPC link with NLB

The one-to-one mapping of VPC links to NLBs and microservices in option 2 has scalability limits due to VPC link quotas. An alternative is to use multiple microservices per NLB.


AWS Cloud
Region
NLB1 Listener 1: TCP Port 5001 -> Target Group 1
NLB1 Listener 2: TCP Port 5002 -> Target Group 2
NLB2 Listener 1: TCP Port 5003 -> Target Group 3
NLB2 Listener 2: TCP Port 5004 -> Target Group 4
API Gateway
VPCLink 1
VPCLink 2
VPC
NLB1
NLB2
Private subnet
EC2
mS1
Target Group 1: Port 443/80
mS2
Target Group 2: Port 443/80
mS3
Target Group 3: Port 443/80
mS4
Target Group 4: Port 443/80

A single NLB fronts multiple microservices in a VPC by using multiple listeners, with each listener on a separate port per microservice. Here, NLB1 fronts two microservices in one VPC. NLB2 fronts two other microservices in a second VPC. With multiple microservices per NLB, routing is defined for the REST API when choosing the integration point for a method. You define each service using a combination of selecting the *VPC Link*, which is integrated with a specific NLB, and a specific port that is assigned for each microservice at the NLB Listener and addressed from the *Endpoint URL*.

To scale out further, add additional listeners to existing NLBs, limited by Quotas for your Network Load Balancers. In cases where each microservice has its dedicated load balancer or access point, those are configured as targets to the NLB. Alternatively, integrate additional microservices by adding additional VPC links.

Benefits:

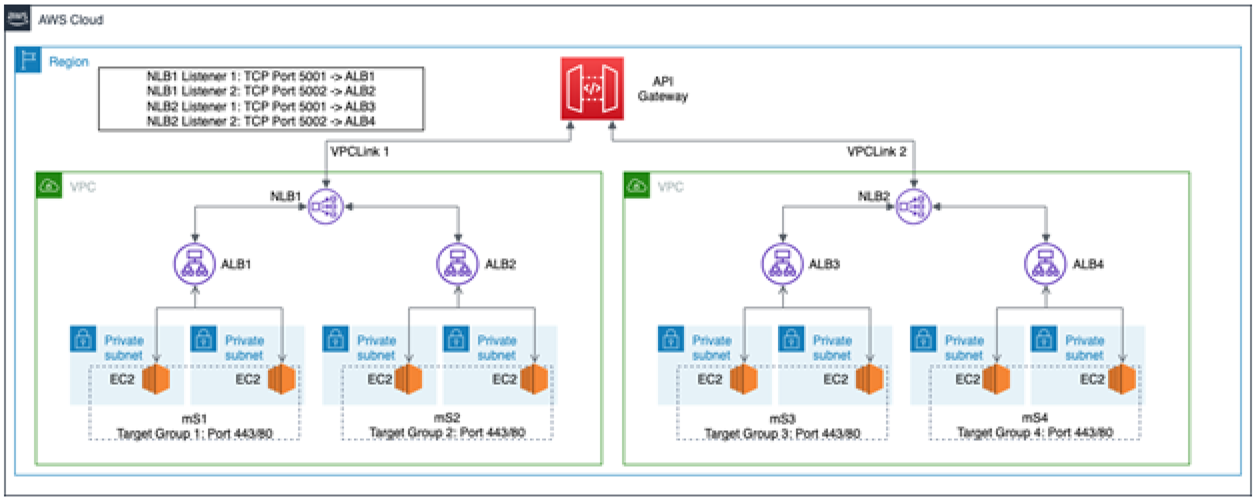
- Larger scalability – limited by NLB listener quotas and VPC link quotas.
- Managing fewer NLBs supporting multiple microservices reduces operational complexity.
- Low latency with a single NLB in the request path.

Considerations:

- Shared NLB configuration limits independent deployments for individual microservices teams.

Option 4: REST API using VPC link with NLB and ALB targets

Customers often build microservices with ALB as their access point. To expose these via API Gateway REST APIs, you can take advantage of ALB as a target for NLB. This pattern also increases the number of microservices supported compared to the option 3 architecture.


AWS Cloud
Region
NLB1 Listener 1: TCP Port 5001 -> ALB1
NLB1 Listener 2: TCP Port 5002 -> ALB2
NLB2 Listener 1: TCP Port 5001 -> ALB3
NLB2 Listener 2: TCP Port 5002 -> ALB4
API Gateway
VPCLink 1
VPCLink 2
VPC
NLB1
NLB2
ALB1
ALB2
ALB3
ALB4
Private subnet
EC2
mS1
Target Group 1: Port 443/80
mS2
Target Group 2: Port 443/80
mS3
Target Group 3: Port 443/80
mS4
Target Group 4: Port 443/80

A VPC link (VPCLink1) is created with NLB1 in a VPC1. ALB1 and ALB2 front-end the microservices mS1 and mS2, added as NLB targets on separate listeners. VPC2 has a similar configuration. Your isolation needs and IP space determine if microservices can reside in one or multiple VPCs.

To scale out further:

- Create additional VPC links to integrate new NLBs.
- Add NLB listeners to support more ALB targets.
- Configure ALB with path-based rules to route requests to multiple microservices.

Benefits:

- High scalability integrating services using NLBs and ALBs.
- Independent deployments per team is possible when each ALB is dedicated to a single microservice.

Considerations:

- Multiple load balancers in the request path can increase latency.

Considerations and best practices

Beyond the scaling considerations of scale with VPC link integration discussed in this blog, there are other considerations:

- Evaluate REST APIs and HTTP APIs capabilities to meet your requirements.
- Choose the optimal load balancer type for your application needs.
- For multi-account architecture reference Building private cross-account APIs using Amazon API Gateway and AWS PrivateLink.
- Avoid exceeding default quotas rapidly and request a quota increase for higher limit requirements.

- Monitor Service Quotas to plan proactively and mitigate risks as your architecture evolves. Consider the use of the Quota Monitor solution for monitoring.
- See Best Practices for Designing Amazon API Gateway Private APIs and Private Integration – Rest API.

Conclusion

This blog explores building scalable API Gateway integrations for microservices using VPC links. VPC links enable forwarding external traffic to backend microservices without exposing them to the internet or leaving the AWS network. The post covers scaling considerations based on using REST APIs versus HTTP APIs and how they integrate with NLBs or ALBs across VPCs.

While API type and load balancer selection have other design factors, it’s important to keep the scaling considerations discussed in this blog in mind when designing your API layer architecture. By optimizing API Gateway implementation for performance, latency, and operational needs, you can build a robust, secure API to expose microservices at scale.

For more serverless learning resources, visit Serverless Land.
TAGS: contributed, serverless