Containers

Integrate Amazon API Gateway with Amazon EKS

by Re Alvarez-Parmar and Vikram Venkataraman | on 12 APR 2021 | in Amazon API Gateway, Amazon Elastic Kubernetes Service, Containers | Permalink | Share

Since 2015, customers have been using Amazon API Gateway to provide scalable and secure entry points for their API services. As customers adopt Amazon Elastic Kubernetes Service (Amazon EKS) to orchestrate their services, they have asked us how they can use API Gateway to expose their microservices running in Kubernetes. This post shows you how to use API gateway to provide external connectivity to your services running in an EKS cluster.

API Gateway is a fully managed service that makes it easy for you to create, publish, maintain, monitor, and secure APIs at any scale. API Gateway provides an entry point to your microservices. It helps you innovate faster by handling common functions such as API throttling, request caching, authorization and access control, monitoring, version management, and security.

Managing API Gateway using AWS Controller for Kubernetes

API Gateway private integrations let you expose services running an EKS cluster to clients outside of your VPC using Network Load Balancers (NLB) and Application Load Balancers (ALB). Currently, customers that use API Gateway to expose their private microservices running in EKS manage their API Gateway configuration separately from their Kubernetes resource definitions. For example, many customers use an infrastructure-as-code tool, like CloudFormation or Terraform, to create API Gateway resources and Helm or a GitOps tool to manage their Kubernetes cluster configuration.

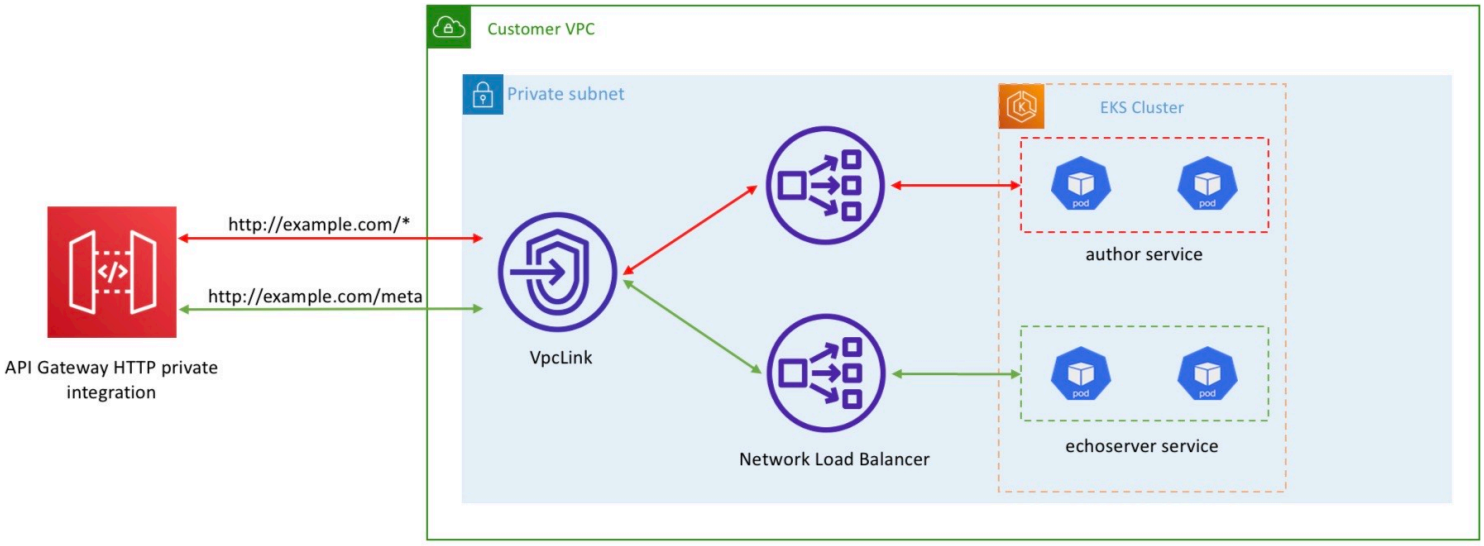
This post will use AWS Controller for Kubernetes (ACK) to create and manage API Gateway resources. ACK is a community-driven project that lets you manage AWS services using the Kubernetes API and tools you are already familiar with, like kubectl. Using ACK, you can create and update AWS service resources, like an S3 bucket or API Gateway API, the same way you create and update a Kubernetes deployment, service, or pod.

> *IMPORTANT Please be sure to read ACK documentation about release versioning and maintenance phases and note that ACK service controllers in the `Preview` maintenance phase are not recommended for production use. Use of ACK controllers in `Preview` maintenance phase is subject to the terms and conditions contained in the AWS Service Terms, particularly the Beta Service Participation Service Terms, and apply to any service controllers in a `Preview` maintenance phase.*

Although this post uses ACK to manage API Gateway, the architecture will remain identical if you choose to create and manage API gateway yourself, either manually or using an infrastructure-as-code tool.

Solution

We will create an EKS cluster with managed nodes. We'll then deploy two sample applications and expose them using an internal Network Load Balancer for each application. Then, we'll create a VpcLink, and create an API Gateway HTTP API with a route for each application.



You will need the following to complete the tutorial:

- AWS CLI version 2
- eksctl
- kubectl
- Helm

Let's start by setting up environment variables required for the solution:

```
Bash
export AGW_AWS_REGION=ap-south-1 <-- Change this to match your region
export AGW_ACCOUNT_ID=$(aws sts get-caller-identity --query 'Account' --output text)
export AGW_EKS_CLUSTER_NAME=eks-ack-apigw
```

Create a new EKS cluster using eksctl:

```
Bash
eksctl create cluster \
  --name $AGW_EKS_CLUSTER_NAME \
  --region $AGW_AWS_REGION \
  --managed
```

Deploy the AWS Load Balancer Controller

Amazon API Gateway HTTP APIs support private integration with NLB and Application Load Balancer (ALB). We'll use two NLBs to distribute traffic to the sample applications. The steps to integrate ALB and NLB with API Gateway are identical. If you choose to use ALB for load balancing, you'll also create an ingress resource and configure routing in ingress instead of API Gateway.

Run the following commands to deploy the AWS Load Balancer Controller into your cluster:

Bash

```
## Associate OIDC provider
eksctl utils associate-iam-oidc-provider \
  --region $AGW_AWS_REGION \
  --cluster $AGW_EKS_CLUSTER_NAME \
  --approve

## Download the IAM policy document
curl -S https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-control

## Create an IAM policy
aws iam create-policy \
  --policy-name AWSLoadBalancerControllerIAMPolicy-APIGWDEMO \
  --policy-document file://iam-policy.json 2> /dev/null

## Create a service account
eksctl create iamserviceaccount \
  --cluster=$AGW_EKS_CLUSTER_NAME \
  --region $AGW_AWS_REGION \
```

Deploy the ACK Controller for API Gateway

The ACK controller for API Gateway will manage API Gateway resources on your behalf. The controller needs IAM permissions to create and update API Gateway resources. We'll create a Kubernetes service account for the controller that has the required permissions.

Create a service account for ACK:

Bash

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-apigateway-ingress-contro

aws iam create-policy \
  --policy-name ACKIAMPolicy \
  --policy-document file://ack-iam-policy.json

eksctl create iamserviceaccount \
  --attach-policy-arn=arn:aws:iam::${AGW_ACCOUNT_ID}:policy/ACKIAMPolicy \
  --cluster=$AGW_EKS_CLUSTER_NAME \
  --namespace=kube-system \
  --name=ack-apigatewayv2-controller \
  --override-existing-serviceaccounts \
```

```
  --region $AGW_AWS_REGION \
  --approve
```

The ACK team has published Helm chart to install ACK on Amazon ECR Public Gallery. Install ACK using Helm:

```
export HELM_EXPERIMENTAL_OCI=1
export SERVICE=apigatewayv2
export RELEASE_VERSION=v0.0.2
export CHART_REPO=oci://public.ecr.aws/aws-controllers-k8s
export CHART_REF="$CHART_REPO/$SERVICE --version $RELEASE_VERSION"

helm chart pull $CHART_REF

# Install ACK

# Please note that the ECR Public Endpoints are available in "us-east-1"
# and "us-west-2". The region value in this command does not need to be
# updated to the region where you are planning to deploy the resources.
# https://docs.aws.amazon.com/general/latest/gr/ecr-public.html#ecr-public-region
aws ecr-public get-login-password --region us-east-1 | helm registry login --userna

# The region value in this command should reflect the region where you
# are planning to deploy the resources
```

Deploy sample applications

We'll deploy two sample applications and create two corresponding Kubernetes services. The `authorservice` service responds with a list of books. The `echoserver` service echoes request metadata. Each of these services will have an associated NLB. Each NLB's listener will correspond to a resource path in API Gateway.

Deploy sample applications:

Bash

```
kubectl apply -f https://github.com/aws-samples/amazon-apigateway-ingress-controller-b
kubectl apply -f https://github.com/aws-samples/amazon-apigateway-ingress-controller-b
```

You can also use a single NLB to expose both applications by creating multiple listeners on the same NLB, and binding target groups associated with the listeners with Kubernetes services as explained here.

Create API Gateway resources

To enable access to a resource in an Amazon Virtual Private Cloud (VPC) through API Gateway, we have to create a VPC Link resource targeted for our VPC and then integrate an API method with a private integration that uses the VpcLink. A VPC link encapsulates connections between API Gateway and targeted VPC resources. When you create a VPC link, API Gateway creates and manages elastic network interfaces for the VPC link in your account.

Create security group for the VPC link:

Bash
```
AGW_VPCLINK_SG=$(aws ec2 create-security-group \
  --description "SG for VPC Link" \
  --group-name SG_VPC_LINK \
  --vpc-id $AGW_VPC_ID \
  --region $AGW_AWS_REGION \
  --output text \
  --query 'GroupId')
```

Create a VPC Link for the internal NLB:

Bash
```
cat > vpclink.yaml<<EOF
apiVersion: apigatewayv2.services.k8s.aws/v1alpha1
kind: VPCLink
metadata:
  name: nlb-internal
spec:
  name: nlb-internal
  securityGroupIDs:
    - $AGW_VPCLINK_SG
  subnetIDs:
    - $(aws ec2 describe-subnets \
         --filter Name=tag:kubernetes.io/role/internal-elb,Values=1 \
         --query 'Subnets[0].SubnetId' \
         --region $AGW_AWS_REGION --output text)
    - $(aws ec2 describe-subnets \
         --filter Name=tag:kubernetes.io/role/internal-elb,Values=1 \
         --query 'Subnets[1].SubnetId' \
         --region $AGW_AWS_REGION --output text)
    - $(aws ec2 describe-subnets \
```

Depending on your AWS Region, you may need to modify the VPC link manifest above to exclude subnets in AZs that don't support VPC link. You can find the offending subnet by checking the log output of the ACK pod.

```
Bash
kubectl -n kube-system logs ack-apigatewayv2-controller-XXXX | grep ERROR
```

```
realvarez@ip-172-31-31-128 ~/my-k8s-cluster/api-gw-ack k -n kube-system logs ack-apigatewayv2-controller-bc7f79f8-ss2z8 | grep ERROR
2021-03-11T23:37:52.838Z ERROR controller-runtime.controller Reconciler error {"controller": "vpclink", "request": "default/
nlb-internal", "error": "BadRequestException: Subnet 'subnet-0e5afd8fc57704203' is in Availability Zone 'cac1-az4' where service is not availa
ble"}
```

For example, in our lab test, the AZ *cac1-az4* (in the Canada central region) did not support VPC link, therefore, we had to remove that subnet or else ACK would fail to create the VPC link.

The VPC link can take a few minutes to become available. You can check its status using AWS CLI:

```
Bash
aws apigatewayv2 get-vpc-links --region $AGW_AWS_REGION
```

```
Items:
- CreatedDate: '2021-03-10T21:01:41+00:00'
  Name: nlb-internal
  SecurityGroupIds:
  - sg-0192330ccd0d1c43f
  SubnetIds:
  - subnet-0b328b70eb14a51ca
  - subnet-0c7e8a90ea11808f8
  - subnet-033e50b917795026f
  Tags: {}
  VpcLinkId: qvm81x
  VpcLinkStatus: AVAILABLE
  VpcLinkStatusMessage: VPC link is ready to route traffic
  VpcLinkVersion: V2
```

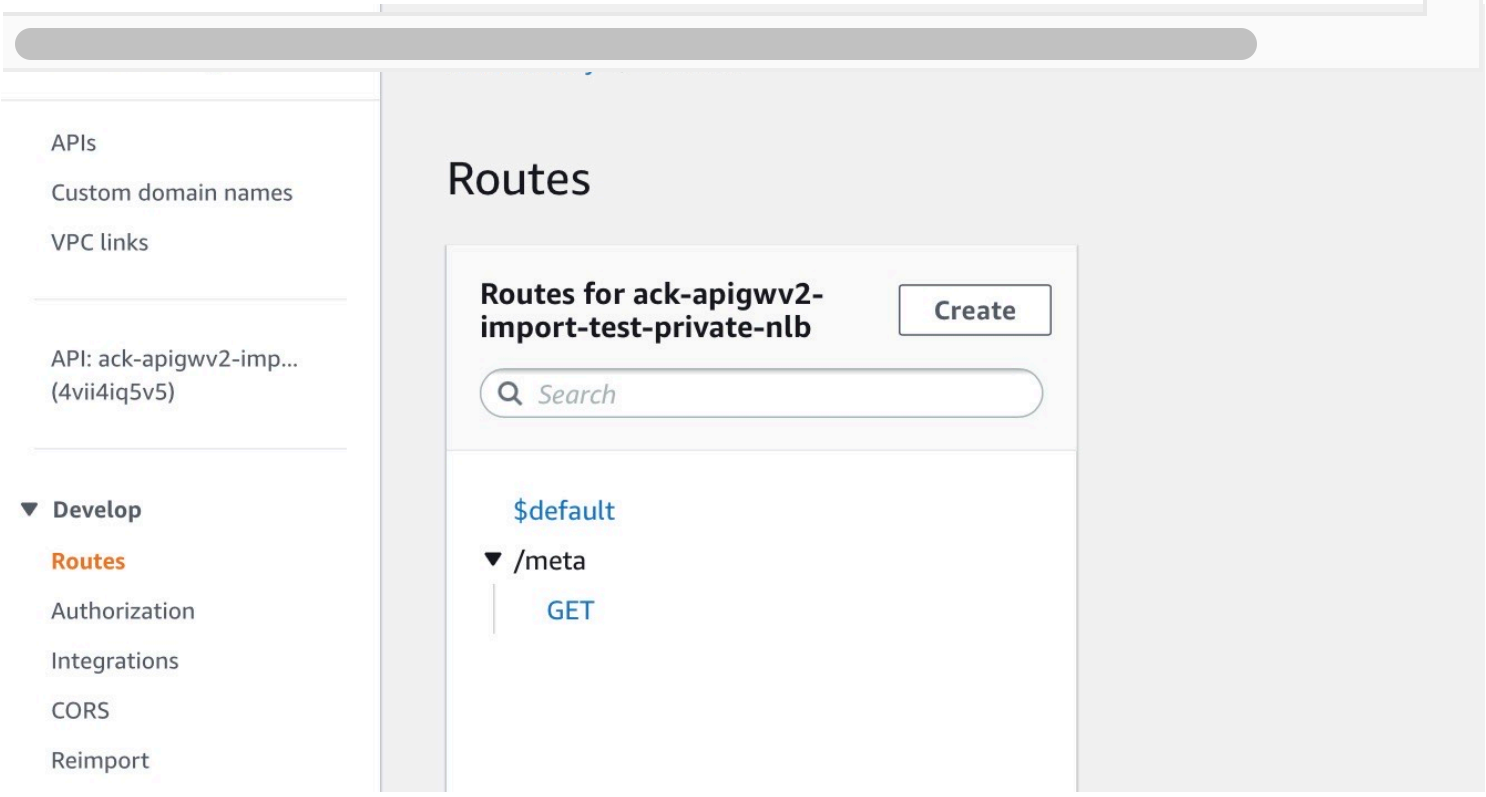
Once the VPC link is available, we can proceed to create an API Gateway API. We'll create a manifest for API configuration that ACK will use to create an API. The `uri` field for each path will map to the ARN of NLB listeners.

Create an API with VPC Link integration:

```
JSON
cat > apigw-api.yaml<<EOF
apiVersion: apigatewayv2.services.k8s.aws/v1alpha1
kind: API
metadata:
  name: apitest-private-nlb
spec:
```

```
  body: '{
              "openapi": "3.0.1",
              "info": {
                "title": "ack-apigwv2-import-test-private-nlb",
                "version": "v1"
              },
              "paths": {
              "/\$default": {
                "x-amazon-apigateway-any-method" : {
                "isDefaultRoute" : true,
                "x-amazon-apigateway-integration" : {
```

ACK will create an API Gateway API and routes based on the definition above. If you go to API Gateway in the AW Management Console, you'll see a new API along with the two routes ACK created. The *$default route* configures Gateway to route to the `authorservice` service whereas /meta route traffic will be routed to the `echoserv` service.



Each route in API Gateway has an associated NLB (or ALB) listener. The $default route maps to the listener of the NLB for the authorservice. Similarly, the /meta maps to the listener of the echoserver NLB

Create a stage:

JSON

```
  echo "
  apiVersion: apigatewayv2.services.k8s.aws/v1alpha1
```

```
  kind: Stage
  metadata:
    name: "apiv1"
  spec:
    apiID: $(kubectl get apis.apigatewayv2.services.k8s.aws apitest-private-nlb -o=jsonpa
    stageName: api
    autoDeploy: true
  " | kubectl apply -f -
```

ACK populates API resources' metadata fields to include the API Endpoint and API ID. You can use kubectl to query this information:

```
Status:
  Ack Resource Metadata:
    Owner Account ID:  $(AWS_ACCOUNT_ID)
  API Endpoint:        https://4vii4iq5v5.execute-api.eu-west-1.amazonaws.com
  API ID:              4vii4iq5v5
```

Invoke API

Let's test the setup by accessing sample applications using the API Gateway API Endpoint. You can use the following command to find the API Endpoint:

Bash

```
kubectl get api apitest-private-nlb -o jsonpath="{.status.apiEndpoint}"
---
https://4vii4iq5v5.execute-api.eu-west-1.amazonaws.com
```

Invoke the `authorservice` service:

Bash

```
curl $(kubectl get api apitest-private-nlb -o jsonpath="{.status.apiEndpoint}")/api/au
```

```
{
    "authors": [
        {
            "author": "Peter Sbarski",
            "id": "Peter-Sbarski",
            "books": "9781617293825",
            "about": ""
        },
        {
            "author": "Ed Robinson",
            "id": "Ed-Robinson",
            "books": "9781788390071",
            "about": "Ed Robinson works as a senior site reliability engineer at Cookpad's global headquarters in Bristol, UK. He has been wor
king with Kubernetes for the last three years, deploying clusters on AWS to deliver resilient and reliable services for global audiences. He i
s a contributor to several open source projects and is a maintainer of Traefik, the modern HTTP reverse proxy designed for containers and micr
oservices."
        },
```


Invoke the `echoserver` service by invoking the `meta` path:

Bash

```
curl $(kubectl get api apitest-private-nlb -o jsonpath="{.status.apiEndpoint}")/api/me
```

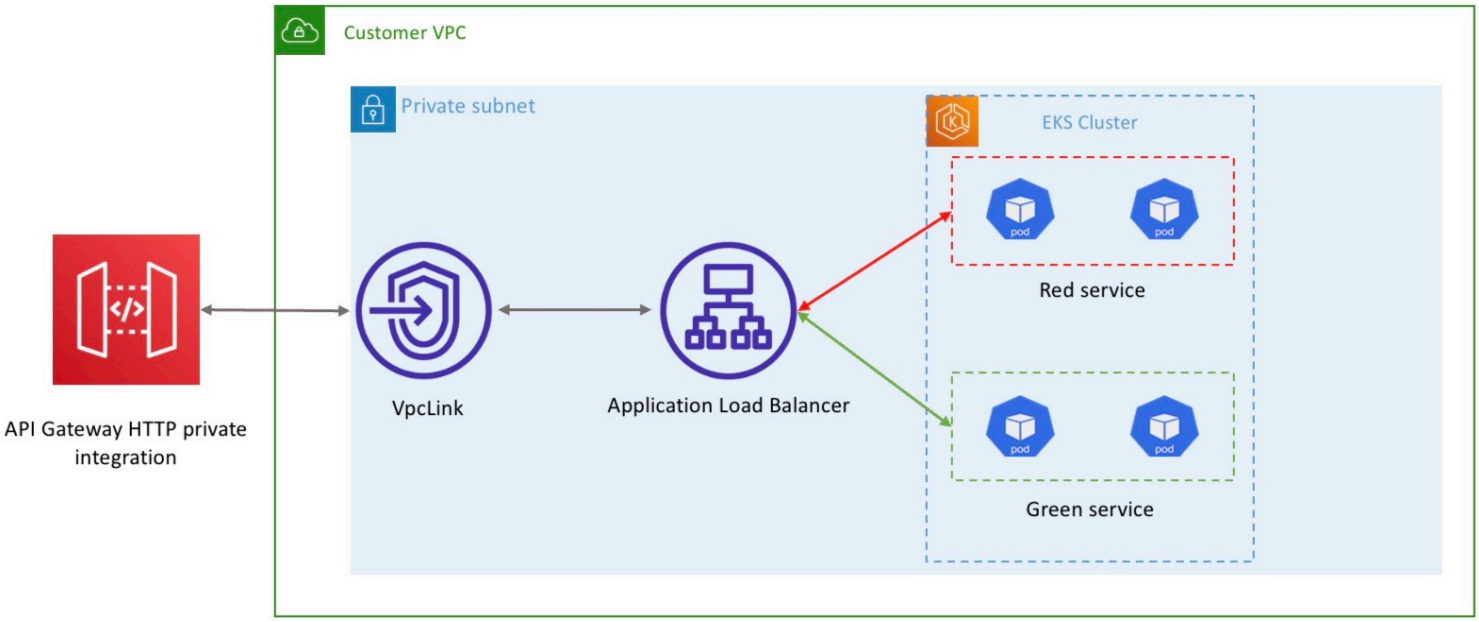
```
CLIENT VALUES:
client_address=192.168.102.48
command=GET
real path=/api/meta
query=nil
request_version=1.1
request_uri=http://4vii4iq5v5.execute-api.eu-west-1.amazonaws.com:8080/api/meta

SERVER VALUES:
server_version=nginx: 1.10.0 - lua: 10001

HEADERS RECEIVED:
accept=*/*
content-length=0
forwarded=for=52.33.188.135;host=4vii4iq5v5.execute-api.eu-west-1.amazonaws.com;proto=https
host=4vii4iq5v5.execute-api.eu-west-1.amazonaws.com
user-agent=curl/7.61.1
via=HTTP/1.1 AmazonAPIGateway
x-amzn-trace-id=Root=1-604bf5d1-441699a21633a7602c901a71
x-forwarded-port=443
```

If you get `{"message":"Service Unavailable"}` error when invoking the API, retry after 30 seconds.

Routing with Application Load Balancer

You can use both NLB and ALB with API Gateway HTTP APIs, API Gateway REST APIs whereas only support private integrations using a NLB. If you use NLB, you'll use API Gateway routes to route traffic to distinct services. If you choose to use an ALB to expose your services, you'll use ALB to route traffic to distinct services.

When using ALB, the API's `$default` route will map to the ALB's listener.



Cleanup

Bash

```
kubectl delete stages.apigatewayv2.services.k8s.aws apiv1
kubectl delete apis.apigatewayv2.services.k8s.aws apitest-private-nlb
kubectl delete vpclinks.apigatewayv2.services.k8s.aws nlb-internal
kubectl delete service echoserver
kubectl delete services authorservice
sleep 10
aws ec2 delete-security-group --group-id $AGW_VPCLINK_SG --region $AGW_AWS_REGION
helm delete aws-load-balancer-controller --namespace kube-system
helm delete ack-apigatewayv2-controller --namespace kube-system
for role in $(aws iam list-roles --query "Roles[?contains(RoleName, \
  'eksctl-eks-ack-apigw-addon-iamserviceaccount')].RoleName" \
  --output text)
do (aws iam detach-role-policy --role-name $role --policy-arn $(aws iam list-attach
done
for role in $(aws iam list-roles --query "Roles[?contains(RoleName, \
  'eksctl-eks-ack-apigw-addon-iamserviceaccount')].RoleName" \
  --output text)
do (aws iam delete-role --role-name $role)
```

Conclusion

This post showed how to use Amazon API Gateway to expose microservices running in your EKS clusters. API Gateway enables you to create an API frontend for your microservices and includes features such as API version management, API key management, authentication and authorization, and DDoS protection. The AWS Controller for

Kubernetes allows you to manage Amazon API Gateway the same way you manage Kubernetes resources like pods, deployments, services, ingresses, and so on.

With ACK, you can define and consume AWS services like API Gateway, Amazon S3, Amazon SNS, Amazon SQS, DynamoDB, and Amazon ECR directly within a Kubernetes cluster. Please see this post to learn more about ACK.