Containers

De-mystifying cluster networking for Amazon EKS worker nodes

by Nathan Taber | on 26 MAR 2020 | in Amazon Elastic Kubernetes Service, Best Practices, Containers, Technical How-to | Permalink | Comments | Share

Running Kubernetes on AWS requires an understanding of both AWS networking configuration and Kubernetes networking requirements. When you use the default Amazon Elastic Kubernetes Service (Amazon EKS) AWS CloudFormation templates to deploy your Amazon Virtual Private Cloud (Amazon VPC) and Amazon EC2 worker nodes, everything typically just works. But small issues in your configuration can result in frustrating errors.

In this blog, we'll review the various ways to configure the Amazon VPC to run EC2 worker nodes for your Kubernetes cluster managed by Amazon EKS. We'll pay particular attention to how to ensure your subnets are properly configured to allow nodes to connect to the cluster control plane.

This blog does not cover pod networking concepts such as the VPC CNI, subnet sizing, or IP address allocation for pods. To learn more about these topics, visit the EKS documentation.

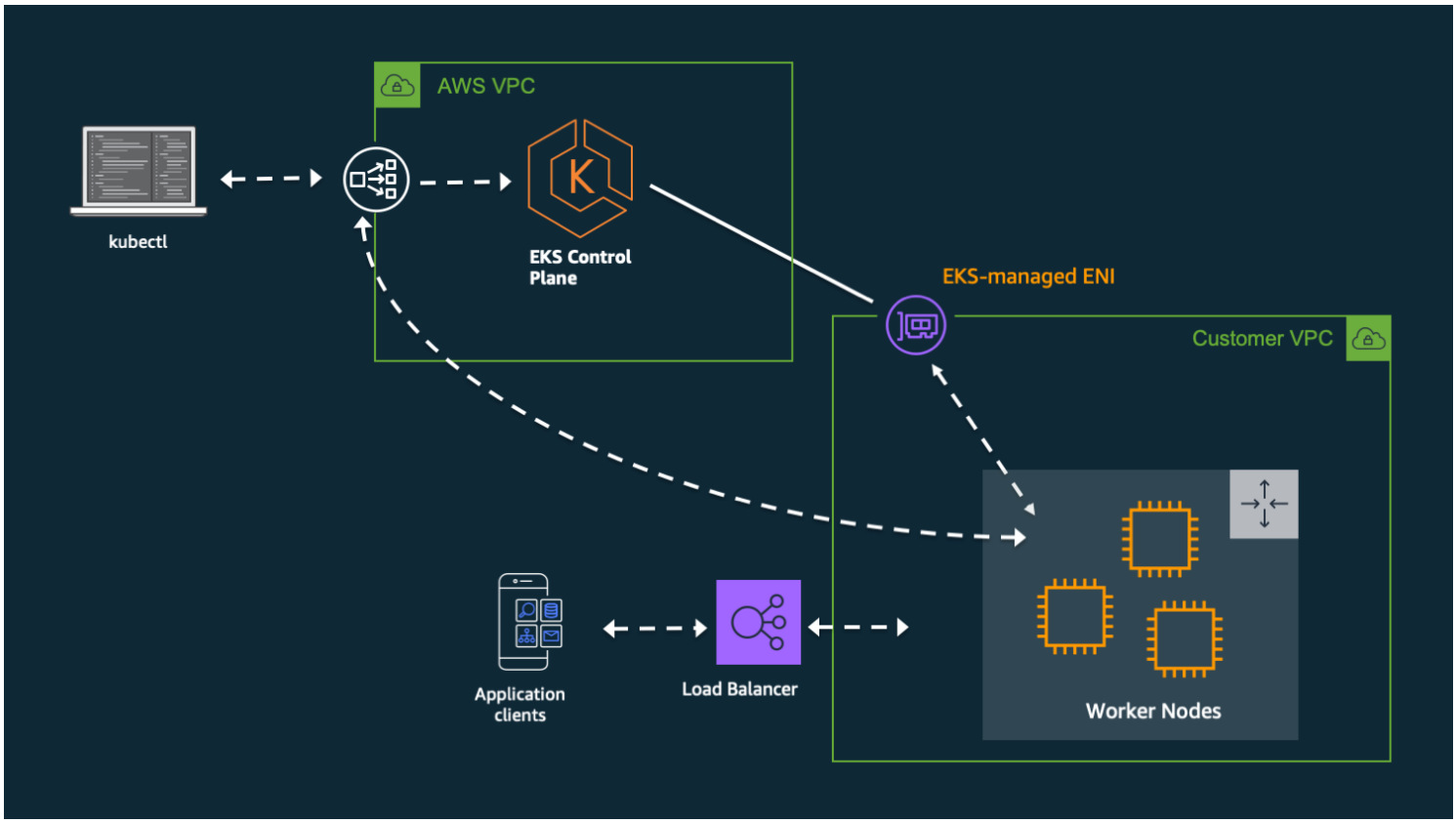
Note – we're changing our VPC and node CloudFormation templates as well as how EKS managed node groups assigns public IP addresses to nodes. Learn more in our blog.

EKS Cluster Architecture

An EKS cluster consists of two VPCs: one VPC managed by AWS that hosts the Kubernetes control plane and a second VPC managed by customers that hosts the Kubernetes worker nodes (EC2 instances) where containers run, as well as other AWS infrastructure (like load balancers) used by the cluster. All worker nodes need the ability to connect to the managed API server endpoint. This connection allows the worker node to register itself with the Kubernetes control plane and to receive requests to run application pods.

The worker nodes connect either to the public endpoint, or through the EKS-managed **elastic network interfaces (ENIs)** that are placed in the subnets that you provide when you create the cluster. The route that worker nodes take to connect is determined by whether you have enabled or disabled the private endpoint for your cluster. Even when the private endpoint is disabled, EKS still provisions ENIs to allow for actions that *originate* from the Kubernetes API server, such as *kubectl exec and logs.*

The diagram below shows this system:



The order of operations for a worker node to come online and start receiving commands from the control plane is:

- EC2 instance starts. Kubelet and the Kubernetes node agent are started as part of the boot process on each instance.
- Kubelet reaches out to the Kubernetes cluster endpoint to register the node. It connects to the public endpoint outside of the VPC or to the private endpoint within the VPC.
- Kubelet receives API commands and sends regular status and heartbeats to the endpoint. When you query the API server (`kubectl get nodes`), you see the latest status that each node's Kubelet has reported back to the API server.

If the node is unable to reach the cluster endpoint, it's unable to register with the control plane and thus unable to receive commands to start or stop pods. If new nodes are unable to connect, this prevents you from being able to use these nodes as part of the cluster.

Networking modes

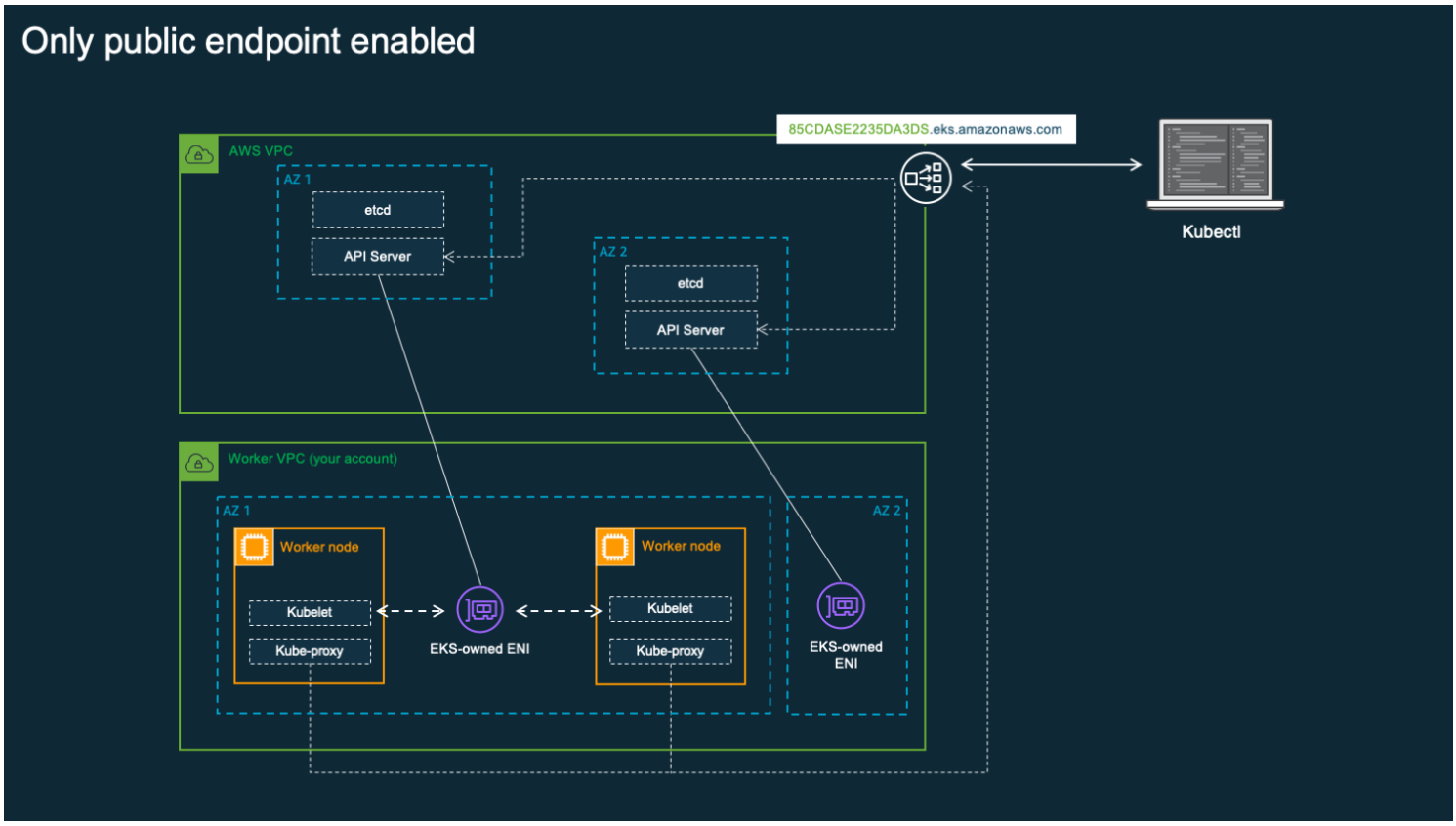
EKS has two ways of controlling access to the cluster endpoint. Endpoint access control lets you configure whether the endpoint is reachable from the public internet or through your VPC. You can enable the public endpoint (default), private endpoint, or both endpoints at the same time. When the public endpoint is enabled, you can also add CIDR restrictions, which allow you to limit the client IP addresses that can connect to the public endpoint.

How your nodes connect to the managed Kubernetes control plane is determined by which endpoint setting you have configured for the cluster. Note, these endpoint settings can be changed anytime through the EKS console or

API.

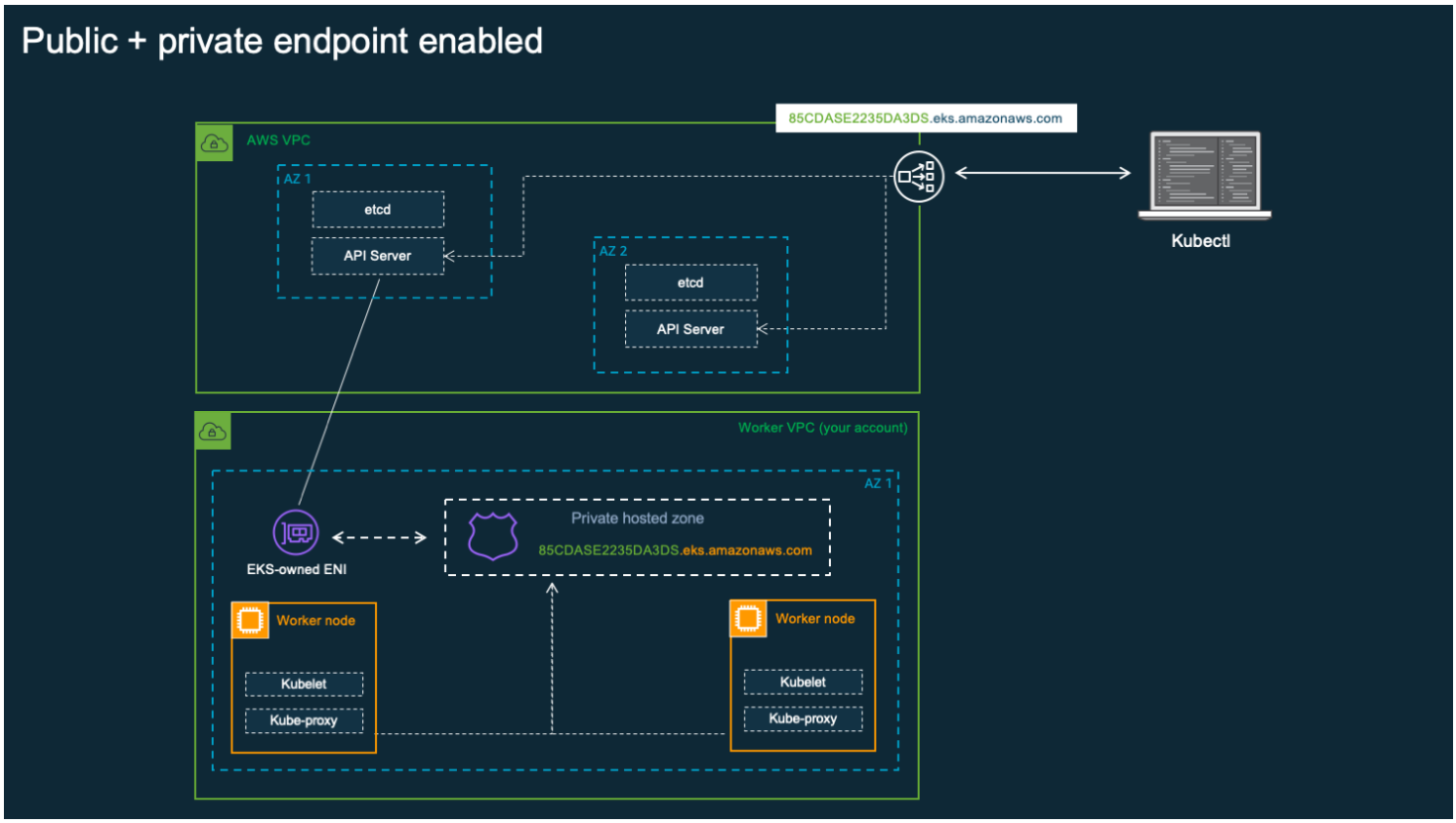
Public endpoint only

This is the default behavior for new Amazon EKS clusters. When only the public endpoint for the cluster is enabled, Kubernetes API requests that originate from within your cluster's VPC (such as worker node to control plane communication) leave the VPC, but not Amazon's network. In order for nodes to connect to the control plane, they must they must have a public IP address and a route to an internet gateway or a route to a NAT gateway where they can use the public IP address of the NAT gateway.



Public and Private endpoints

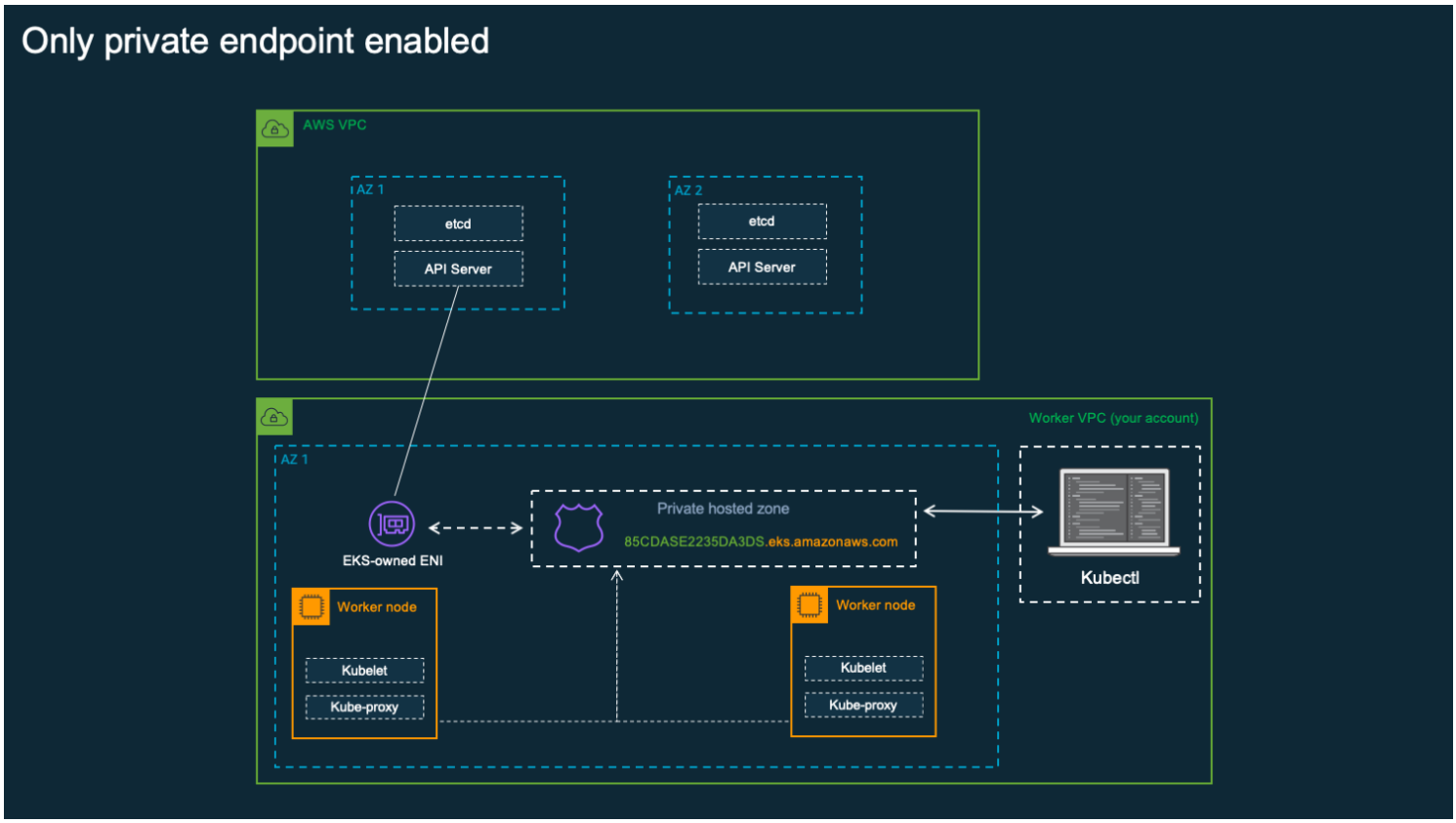
When both the public and private endpoints are enabled, Kubernetes API requests from within the VPC communicate to the control plane via the EKS-managed ENIs within your VPC. Your cluster API server is accessible from the internet.



Private endpoint only

When only the private endpoint is enabled, all traffic to your cluster API server must come from within your cluster's VPC or a connected network. There is no public access to your API server from the internet. Any kubectl commands must come from within the VPC or a connected network. This is typically achieved through using AWS VPN or AWS DirectConnect to your VPC. If you want to restrict access to the endpoint, but don't have AWS VPN or AWS DirectConnect, adding CIDR restrictions to the public endpoint allows you to limit connections to the endpoint without additional networking setup.

For more information on connectivity options, see Accessing a Private Only API Server.



Sidebar: Do my worker nodes need to be in the same subnets as the ones I provided when I started the cluster?

No. Your worker nodes only need to run in the same VPC. They can run in separate subnets from the ones you provided when you started the cluster. This can allow for more IP space or running nodes in multiple availability zones. Subnets you use for your nodes must be tagged appropriately (a step that EKS handles automatically when you create the cluster), so any additional subnets must be manually tagged. To learn more, see the EKS documentation on VPC configuration.

VPC configurations

Now that we have the connection basics down, let's walk through a few of the common scenarios for setting up your cluster networking with Amazon EKS.

In general, your nodes are going to run in either a **public** or a **private** subnet. Whether a subnet is public or private refers to whether traffic within the subnet is routed through an internet gateway. If a subnet is associated with a route table that has a route to an internet gateway, it's known as a **public subnet**. If a subnet is associated with a route table that does not have a route to an internet gateway, it's known as a **private subnet**.

The ability for traffic that originates somewhere else to reach your nodes is called *ingress*. Traffic that originates from the nodes and leaves the network is called *egress*. Nodes with public or elastic IP addresses within a subnet configured with an internet gateway allow ingress from outside of the VPC. Private subnets usually include a NAT gateway, which only allows ingress traffic to the nodes from within the VPC while still allowing traffic *from* the nodes to leave the VPC (*egress*).

There are three typical ways to configure the VPC for your Amazon EKS cluster:

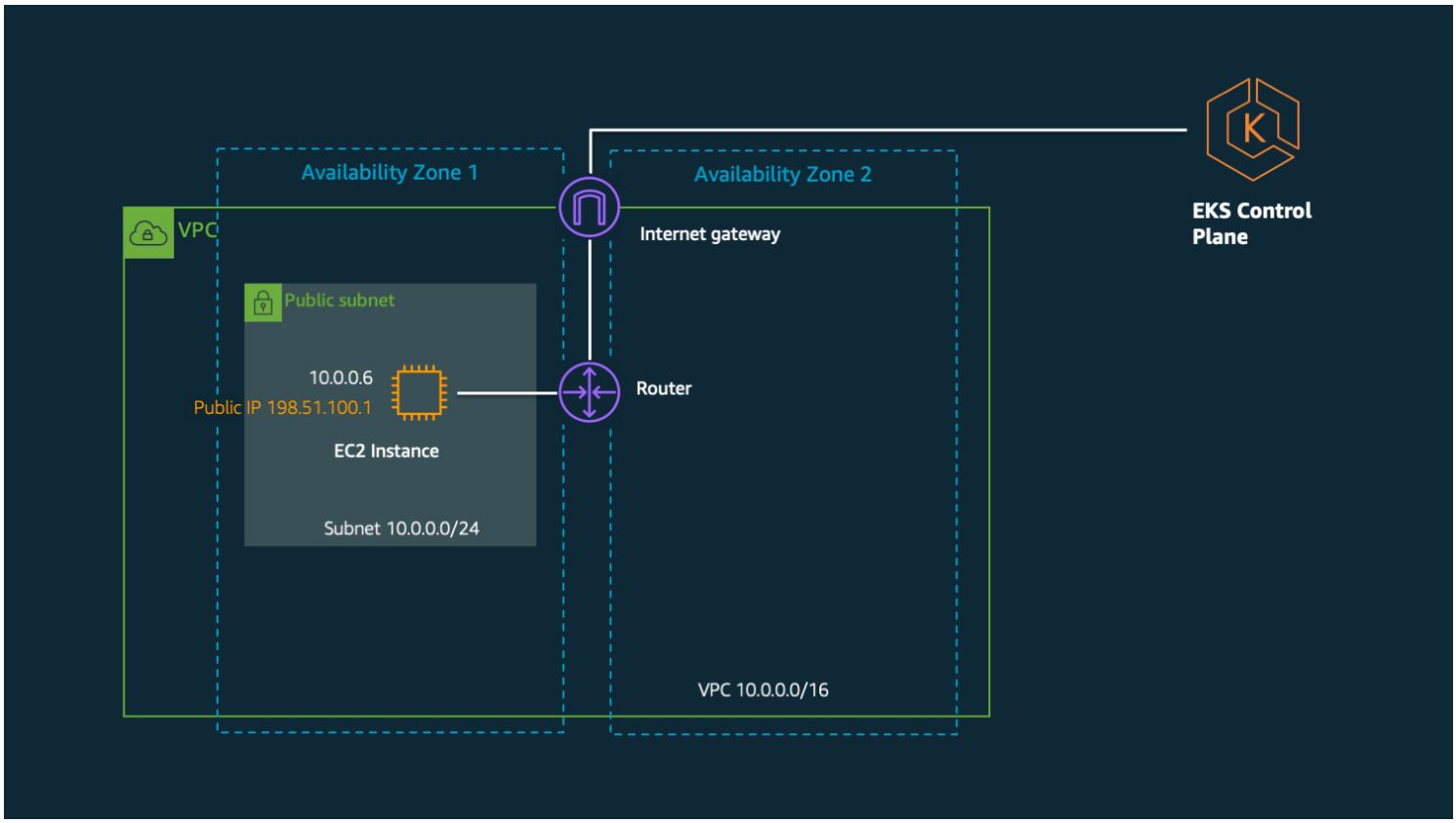
1. Using only public subnets. Nodes and ingress resources (like load balancers) all are instantiated in the same public subnets.
2. Using public and private subnets. Nodes are instantiated in the private subnets and ingress resources (like load balancers) are instantiated in the public subnets.
3. Using only private subnets. Nodes are instantiated in private subnets. There are no public ingress resources as this configuration is only used for workloads that do not need to receive any communications from the public internet.

Public only subnets

This is a simple and straightforward VPC architecture that is good for basic web apps and generally any application where ingress to the nodes from the public internet is not a concern. In this configuration, nodes and ingress resources (like load balancers) all are instantiated in the same public subnets.

Configuration best practices:

- Every subnet should be configured with `mapPublicIpOnLaunch` set to **TRUE** and have a route to an internet gateway.
- Nodes do not need a value for `AssociatePublicIpAddress` (do not include this value in the CFN template or API call)
- The cluster endpoint can be set to enable public, private, or both (public + private)



Learn about this architecture in the Amazon VPC documentation. Start a VPC with the public only subnet configuration using this CloudFormation template:

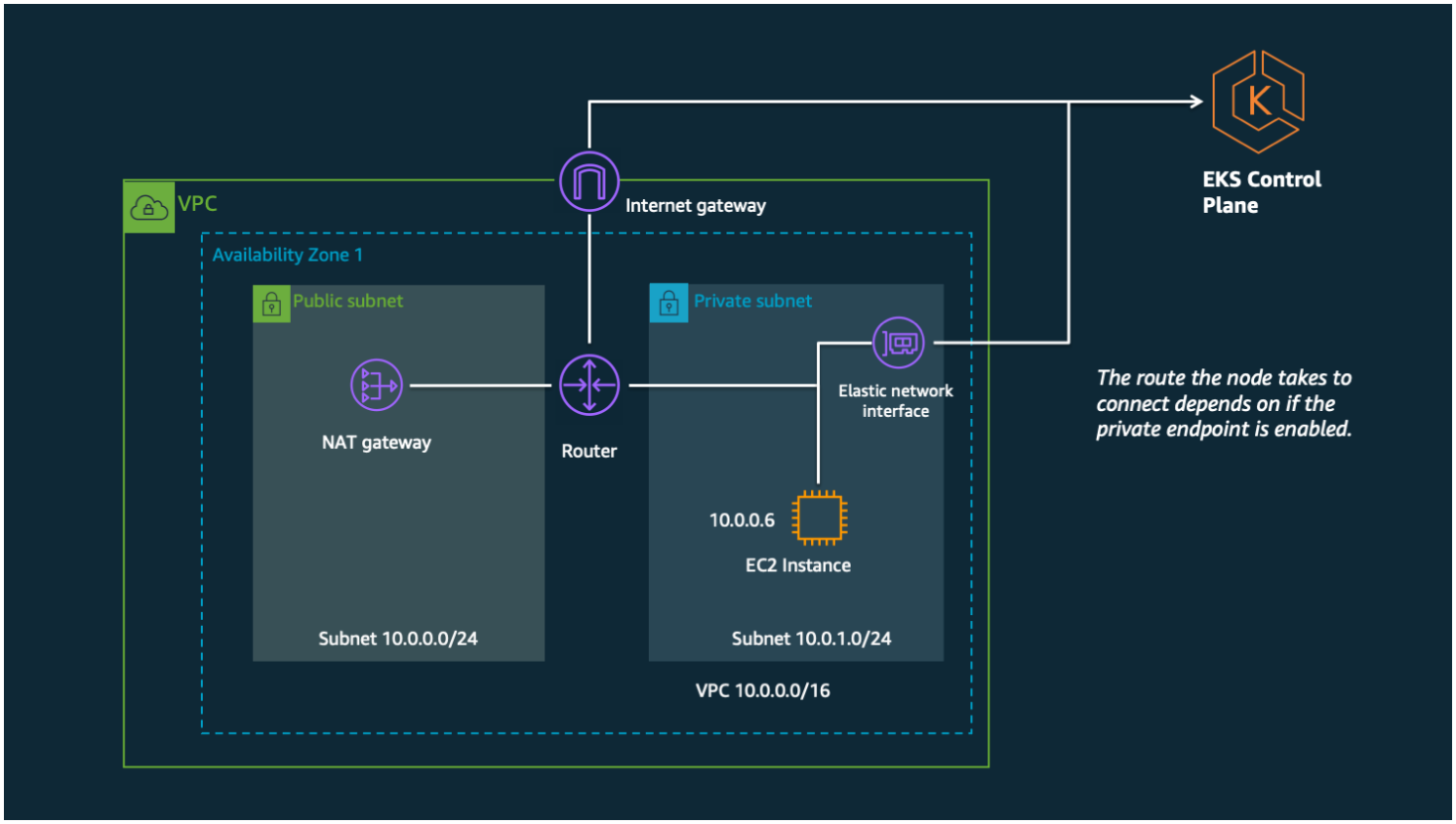
```
https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-03-23/amazon-eks-vpc-sample.yaml
```

Public + Private subnets

This VPC architecture is considered the best practice for common Kubernetes workloads on AWS. In this configuration, nodes are instantiated in the private subnets and ingress resources (like load balancers) are instantiated in the public subnets. This allows for maximum control over traffic to the nodes and works well for a majority of Kubernetes applications.

Configuration best practices:

- Because you are not launching nodes in the public subnets, it's not required to set `mapPublicIpOnLaunch` for public subnets.
- `mapPublicIpOnLaunch` should be set to **FALSE** for the private subnets.
- Nodes do not need a value for `AssociatePublicIpAddress` (do not include this value in the CFN template or API call)
- The cluster endpoint can be set to enable public, private, or both (public + private). Depending on the setting of the cluster endpoint, the node traffic will flow through the NAT gateway or the ENI.



Learn about this architecture in the Amazon VPC documentation. Start a VPC with the public+private subnet configuration using this CloudFormation template:

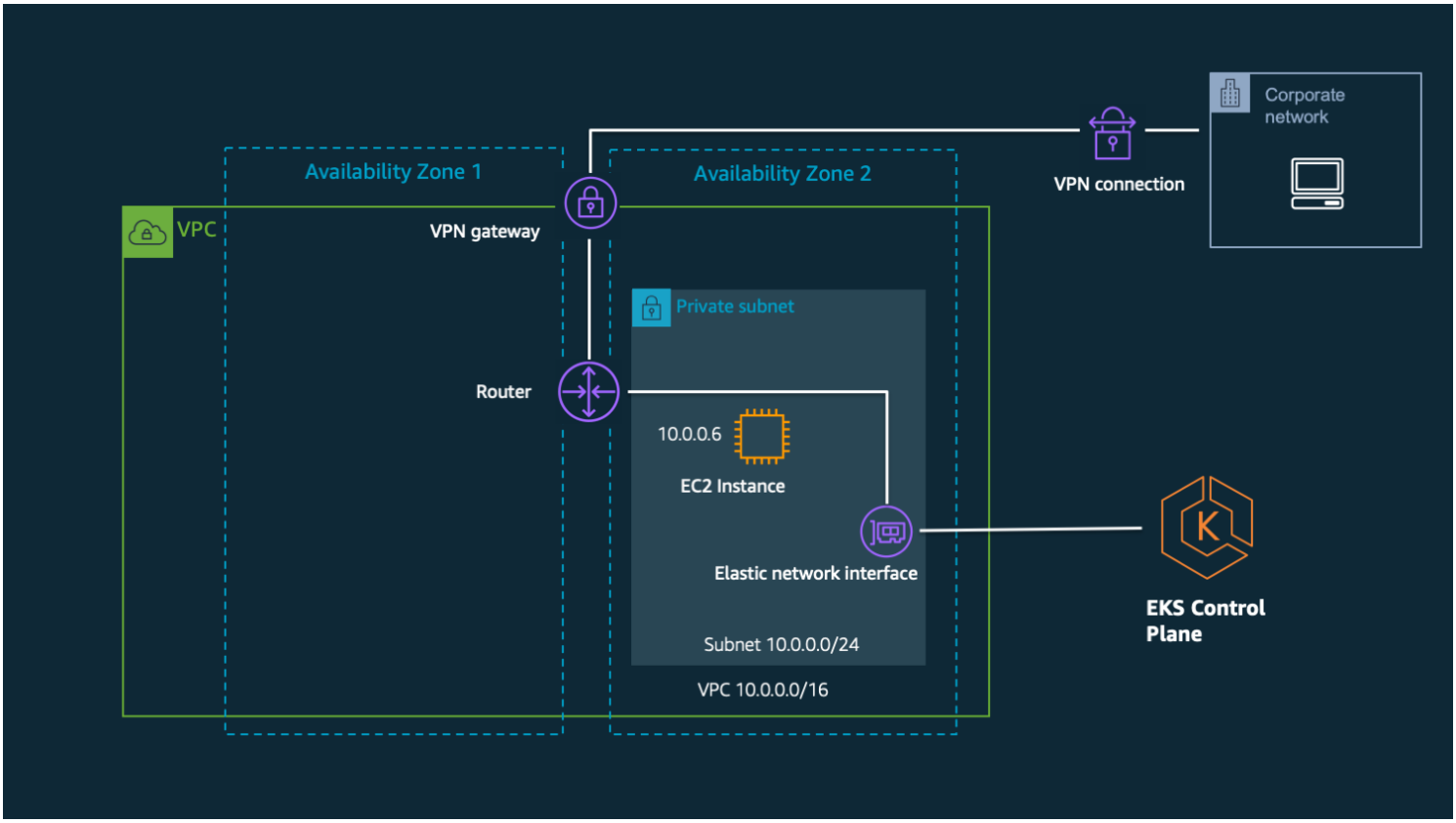
```
https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-03-23/amazon-eks-vpc-private-subnets.yaml
```

Also, note that this is the default VPC configuration for eksctl.

Private only subnets

This is a less common VPC architecture. Use this architecture if your applications do not need to receive communications from the public internet. You'll need to ensure that you have connectivity from your local computer to the private cluster endpoint.

- `mapPublicIpOnLaunch` should be set to **FALSE** for the private subnets.
- Nodes do not need a value for `AssociatePublicIpAddress` (do not include this value in the CFN template or API call)
- Only the private cluster endpoint should be enabled (disable public endpoint). You will need a VPN connection to access your cluster endpoint.
- Enable AWS PrivateLink for EC2 and all of your Amazon ECR and S3 repositories.
- You must set up PrivateLink interface and/or gateway endpoints for your Kubernetes application to be able to reach other AWS services.
- All container images you launch on the cluster must come from ECR repositories with endpoints configured for your VPC. This includes container images for operational tooling such as ClusterAutoscaler and Metrics Server.



Conclusion

After reading this blog, you should now have a better understanding of the options for Amazon EKS VPC architectures. If you want to learn more about VPC configuration, check out the networking section of the Amazon EKS documentation.

— Nate

TAGS: Amazon EKS, Kubernetes, networking, VPC

Comments

7 Comments

4 Prashanth ▾

Join the discussion...

1 Share

Best Newest Oldest

fschroder

4 years ago

Thank you for this article, it's quite useful.
I've a question for the case with private subnets. Is there any guidance regarding how the NACLs for this private subnets should look like?
From experimentation it appears to be working with
Out TCP:443 0/0
In TCP:1024-65535 0/0
Since it's meant to be a private subnet would be good to limit this as much as possible.
Thanks!

1 0 Reply

Mikkel Vesterdahl ↗ fschroder

3 years ago

Hi fschroder.

I've been struggling with the same predicament lately. It seems that limiting inbound traffic in a private subnet, even going as far as to allow the entire VPC CIDR, breaks the communication between worker nodes and the API server. So far my finding is that you'll need to allow 0/0 as well...

I'm assuming that the cause of this is due to the API server being located in the AWS Managed VPC which is inaccessible to us. I have yet to find any official documentation to back up this claim, but it seems to be the only logical explanation.

0 0 Reply

P

Paul Lee

3 years ago edited

Thanks for the article Nathan. I have a question though, when I'm asked to specify a VPC and subnets during the creation of my EKS cluster (on the web console), is that referring to the VPC and subnets where my worker nodes/node groups will be hosted in? Will my control plane reside in those subnets or will it automatically create a separate VPC to host my control plane? Thank you!

0 0 Reply

V

Vijoy Choyi

3 years ago

Well articulated article, Nathan!

o o Reply

Fedor Aredakov

3 years ago

Ahoy, Nathan . Very clear. Thanks!

o o Reply

Nagesh

3 years ago

In the diagram for 'Only public end-point enabled', what do the arrows from kube-proxy to the load balancer signify? kubectl command traffic?

o o Reply

Nagesh

3 years ago

Hi, can the control plane work with more than one Container VPC?

o o Reply