Containers

Leveraging CNI custom networking alongside security groups for pods in Amazon EKS

by Bin Liu and Haofei Feng | on 19 AUG 2022 | in Amazon Elastic Kubernetes Service, Amazon VPC, AWS Transit Gateway, Containers, Technical How-To | Permalink | Share

Introduction

Amazon Elastic Kubernetes Service (Amazon EKS) is a managed service that runs Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes. Amazon EKS supports native virtual private cloud (VPC) networking with the Amazon VPC Container Network Interface (CNI) plugin for Kubernetes. This plugin assigns a private IPv4 or IPv6 address from your VPC to each pod. The VPC CNI allows Kubernetes pods to utilize raw AWS network performance and integrations with other AWS services. In this post, we discuss two features of the VPC CNI plugin: CNI custom networking and security groups for pods. We give a walkthrough of their combined use to provide a scalable, secure architecture for Kubernetes workloads.

CNI custom networking

AWS recommends deploying Amazon EKS clusters into VPCs with large Classless Inter-Domain Routing (CIDR) ranges. This recommendation is based on how VPC CNI allocates IP addresses (one IP per pod) and the limited cluster capacity as the number of workloads in the cluster grows. In scenarios where it may not be possible to use a large CIDR range (such as in environments that share an IP address space with on-premises infrastructure), you can attach a CIDR range from the 100.64.0.0/10 or 198.19.0.0/16 ranges to your VPC. These additional ranges, in conjunction with CNI custom networking, provide additional IP space for Kubernetes pods.

Security groups for pods

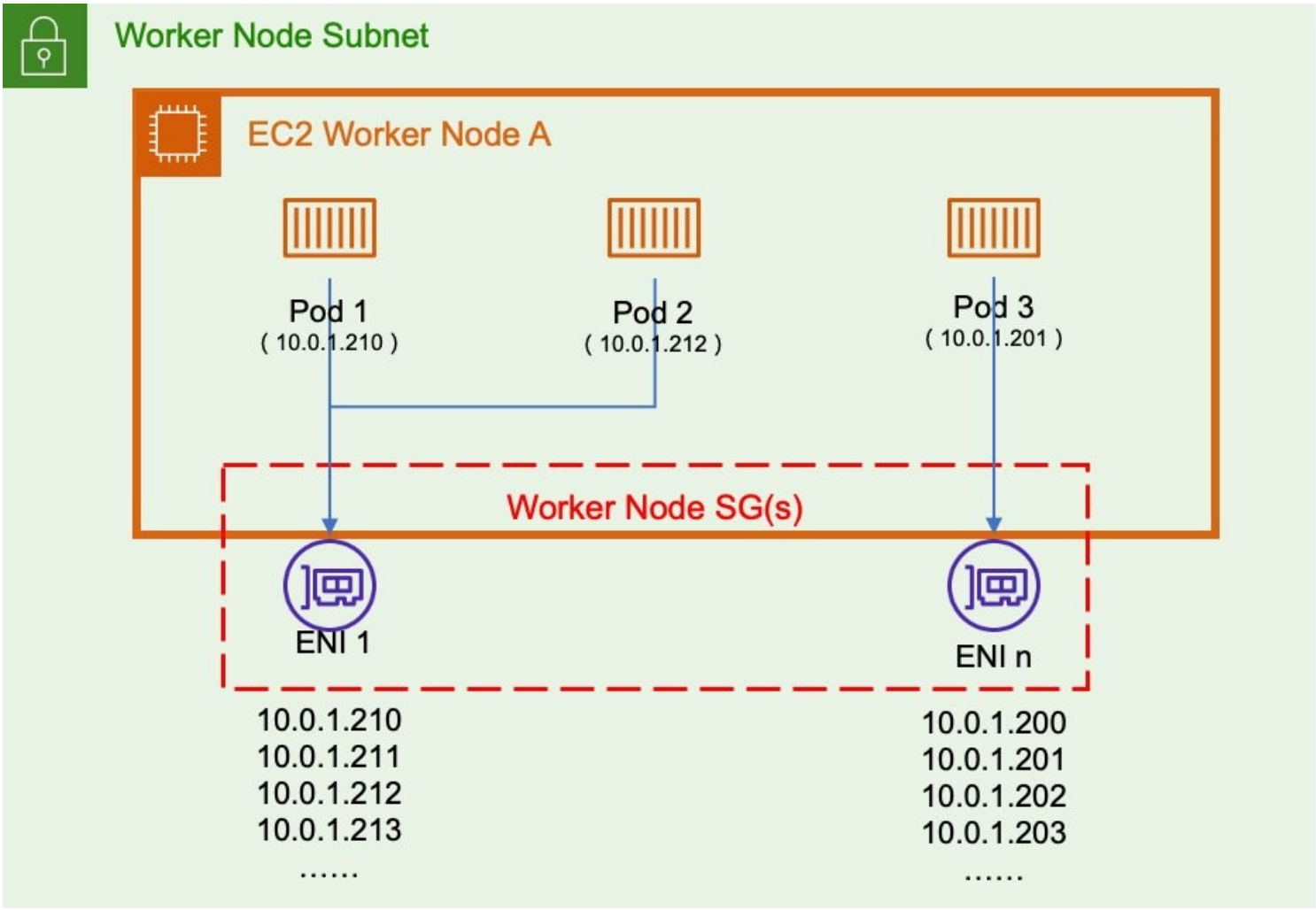
An important concept in container networking is micro-segmentation. This involves the control of which Kubernetes pods can talk to which other pods within the cluster and the control of which pods can talk to dependent services outside of the cluster. When enabled, security group for pods integrate Amazon Elastic Compute Cloud (Amazon EC2) security groups with Kubernetes pods, providing AWS native networking segmentation for Kubernetes workloads. Also, the VPC CNI creates a separate elastic network interface (ENI) for each pod, instead of the default configuration where multiple pods share an ENI. Different security groups can be attached to each Kubernetes pod because each pod has its own ENI. A common example is limiting access to Amazon Relational Database Service (Amazon RDS) databases to pods with a certain a security group. For more information on security groups for pods, see the launch blog post.

VPC CNI traffic flows

To help understand the traffic flow when running CNI custom networking alongside security group for pods, it's important to understand the basics of VPC CNI.

Default behavior without enabling CNI custom networking

In the default VPC CNI configuration, the Amazon EC2 instance is provisioned with one ENI attached to an underlying subnet with the Worker Node Security Group attached. A `primary IP` is attached to the ENI that is used to access the host. As pods are deployed on the Amazon EC2 instance, the VPC CNI attaches `secondary IPs` from the VPC to the same ENI until the limit on that ENI is reached. The VPC CNI provisions a second ENI on the same subnet and fills it with the `secondary IPs`. This process continues until the ENI and secondary IP limits of an instance are reached.

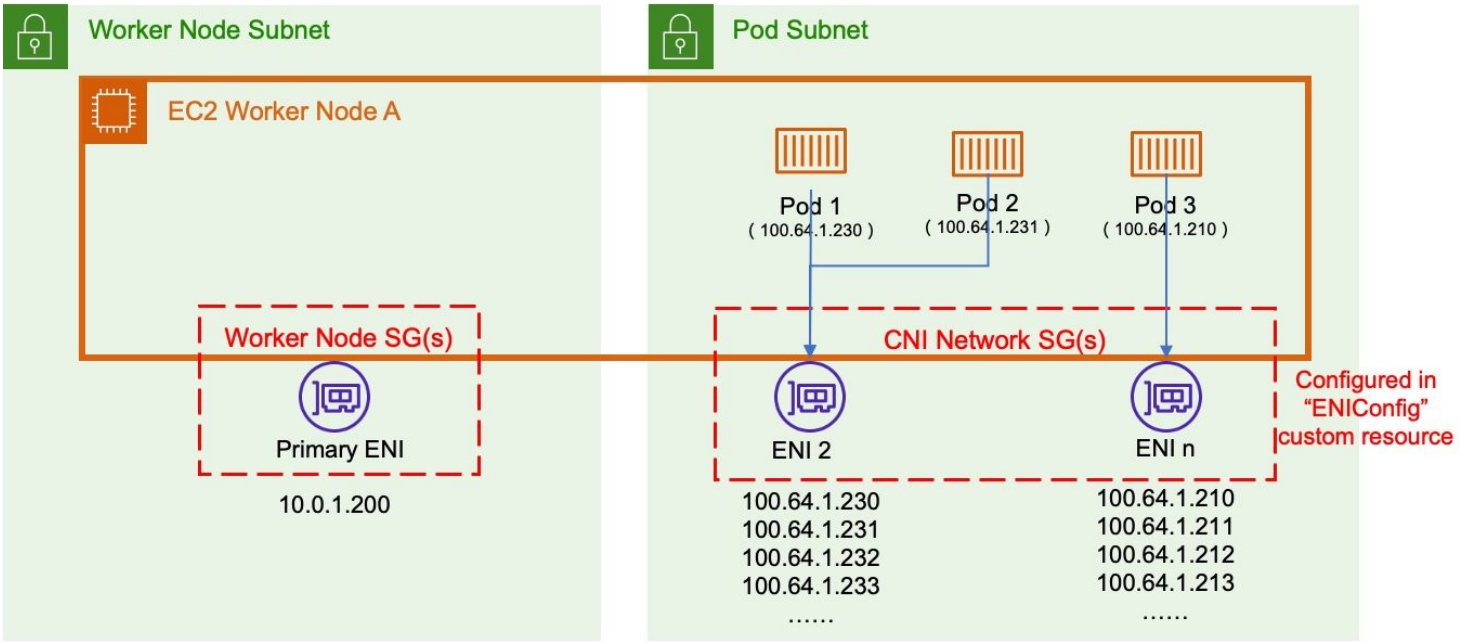


The diagram shows the following:

- Pods share ENIs that are attached to the worker nodes.
- The same security group(s) are attached to all worker node ENIs and apply to all the pods.

Enabling custom networking without security groups for pods

When we enable CNI custom networking, the pods won't be provisioned on the first ENI. Instead, a second ENI with an IP address from the custom CNI CIDR range attaches to the node and the pods attach to this ENI with addresses from the non-routable CIDR range. The number of ENIs and secondary IPs consumed from this non-routable CIDR range are restricted by the limits of each instance.

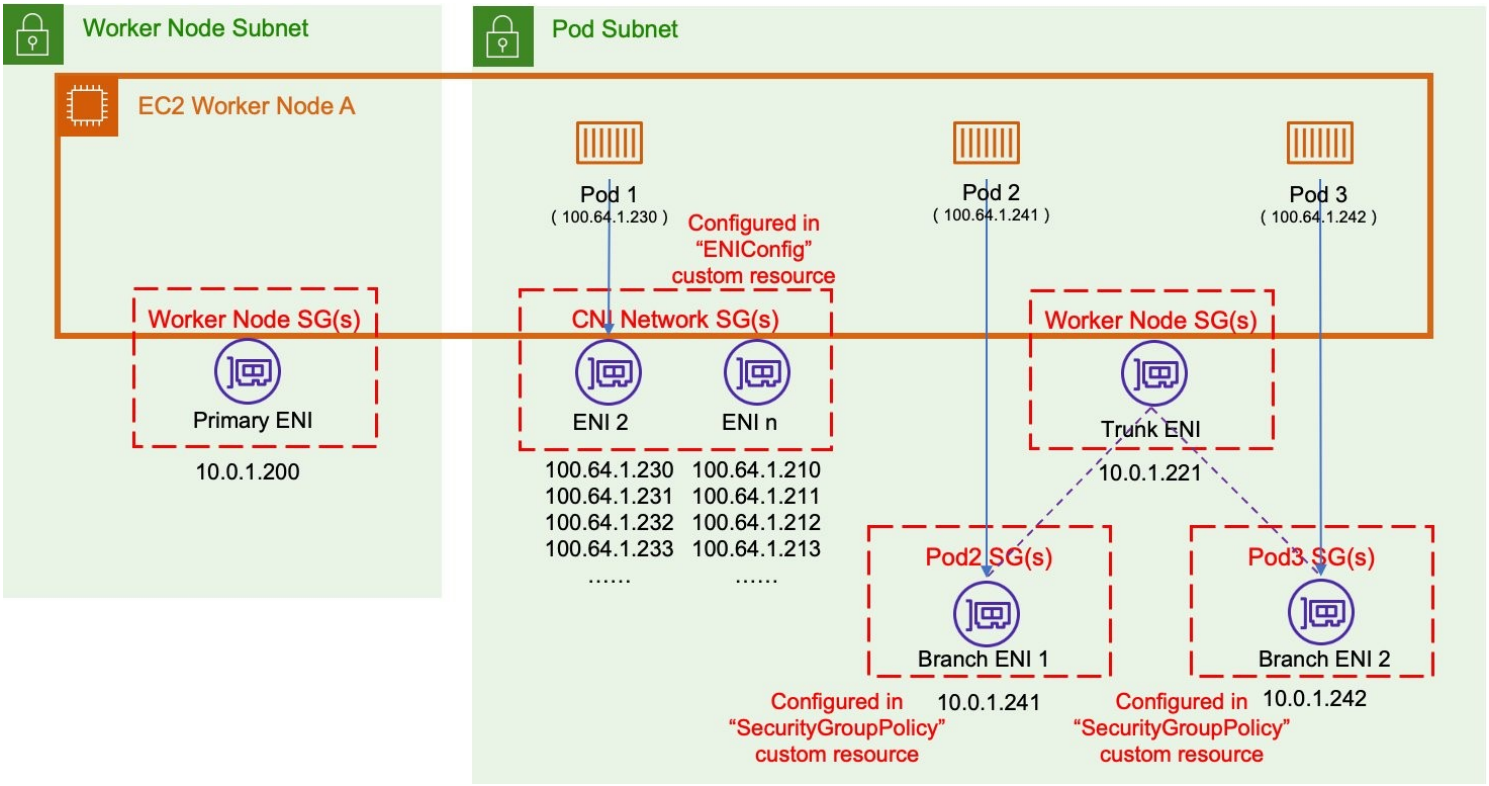


The diagram shows the following:

- The worker node uses one ENI and one IP address from the primary subnet.
- The additional ENIs attached to the worker nodes are provisioned in a different subnet (`100.64 CIDR block`).
- Pods no longer consume IP address from the worker nodes subnet. Pods consume IPs from the non-routable block instead.
- The additional ENIs have their own security group(s) defined within the `ENIConfig` custom resource for each non-routable subnet.
- Custom networking supports Kubernetes Network Policy engines (such as Calico Network Policy) for network segmentation.

Enabling custom networking with security groups for pods

When using a security group for a pod alongside custom networking, logic within the `SecurityGroupPolicy` custom resource decides whether to attach a pod to the shared ENI in the non-routable subnet or whether to give the pod a dedicated branch ENI also in the non-routable subnet.



The diagram shows the following:

- The worker node uses one ENI and one IP address from the primary subnet.
- The worker node uses another ENI as a trunk ENI within the non-routable subnet. With the trunk ENI in place, pods can create branch ENIs and the number of branch ENIs that can be created are based on Amazon EC2 instance types and mapping limits.
- Pods that do not match any `SecurityGroupPolicy Selectors` are launched onto the shared ENI (Pod 1).
- Pods that do match `SecurityGroupPolicy Selectors` are given a branch ENI with their own set of security groups (Pod 2 and Pod 3).

Ingress and egress traffic

With the relevant security group rules established, all Amazon EKS traffic within the VPC could route to each other (Amazon EKS control plane ENIs, worker node primary ENIs, and pod ENIs). However, as the custom networking subnets are not routable outside of the VPC, we need to consider how the routing would work for inbound traffic coming into the cluster and for the workload traffic communicating to dependencies outside of the VPC.

For inbound traffic, we use Kubernetes ingress objects with the AWS Load Balancer Controller. Using the AWS Load Balancer Controller, an Application Load Balancer (ALB) or Network Load Balancer (NLB) could be created into a routable private or public subnet.

If the pods communicate their dependencies outside of the VPC, then we set up NAT to allow the following.

SNAT for pods

By default, when a pod communicates to any `IPv4` address that isn't within a CIDR block that's associated to your VPC, the VPC CNI translates the pod's `IPv4` address to the primary private `IPv4` address of the primary ENI of the node that the pod is running on. With the built-in NAT, pods in the non-routable subnets can talk to dependencies outside of the VPC.

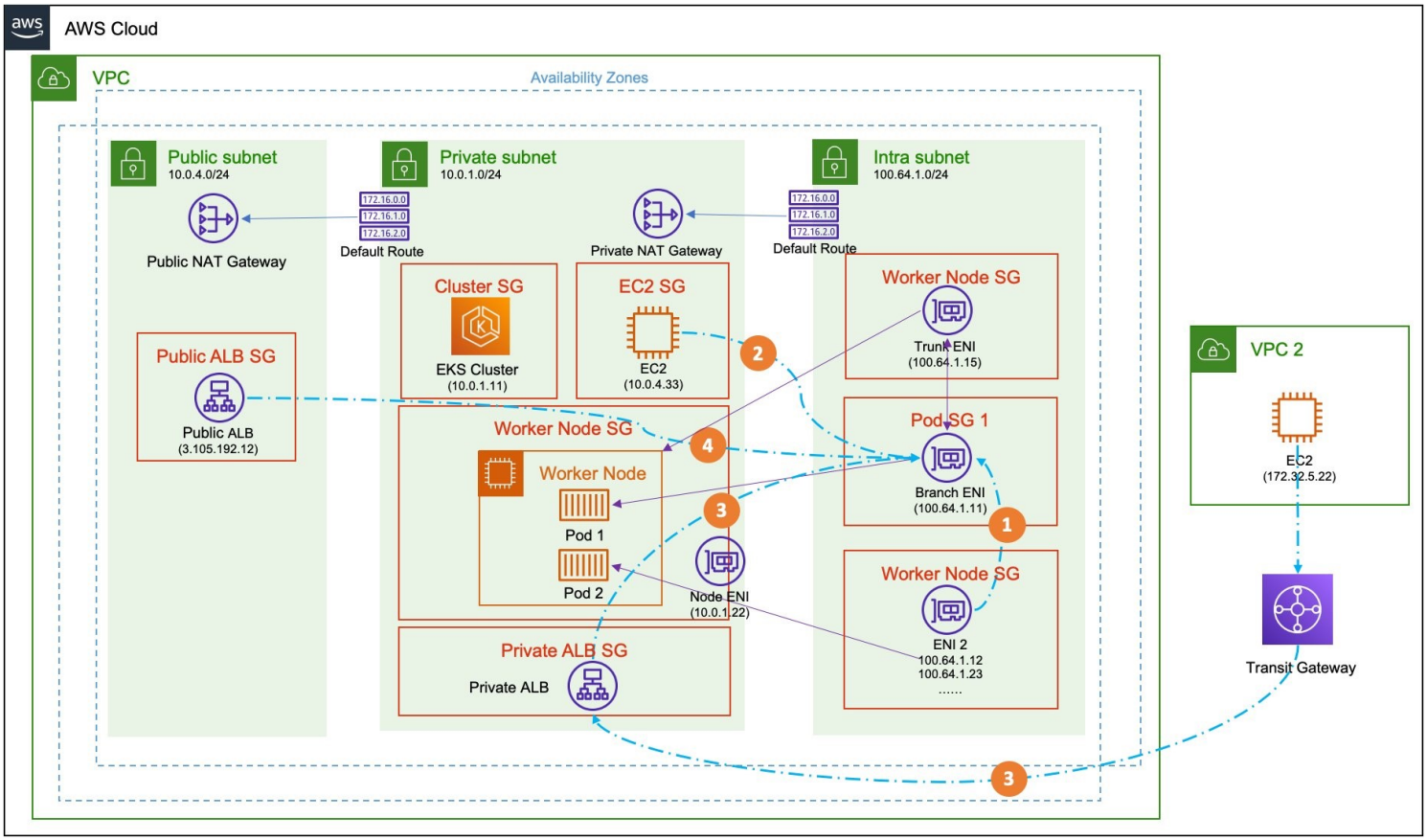
We can also change the default configuration by setting `AWS_VPC_K8S_CNI_EXTERNALSNAT` to `true`, which means an external source network address translation (SNAT), such as a VPC NAT Gateway, is used instead of the pod SNAT done by AWS CNI. More details can be found in this public documentation.

Disable SNAT if you need to: allow inbound communication to your pods from external VPNs, direct connections, external VPCs, and pods that don't access the internet directly through an internet gateway. However, your nodes must run in a private subnet and connect to the internet through an AWS NAT gateway or another external NAT device.

Traffic patterns

The following diagrams highlight the typical Amazon EKS setup with CNI custom networking alongside security groups for pods and some of the traffic patterns.

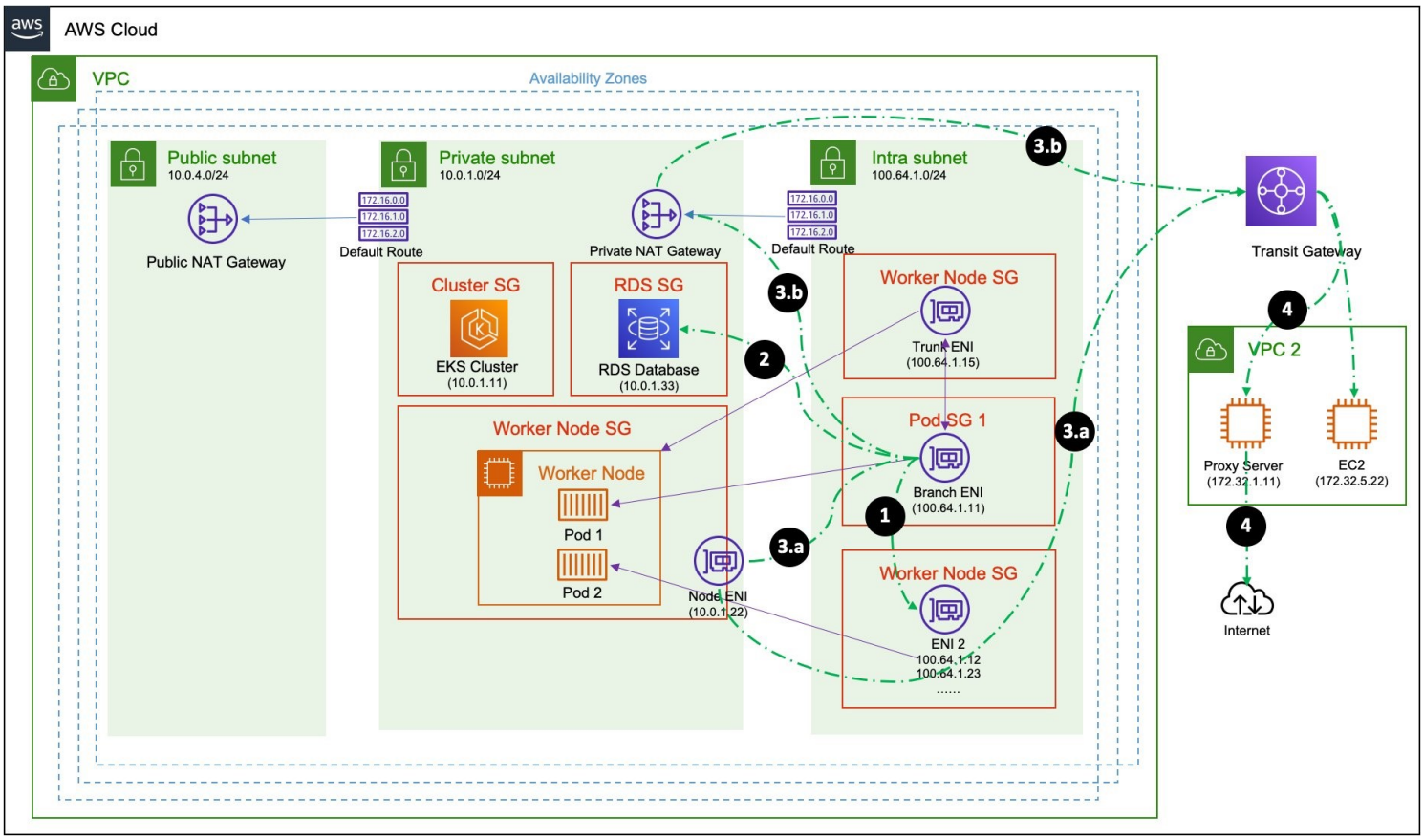
Ingress traffic



1. From pod to pod in the same cluster
2. Private access through Amazon EC2 in the same VPC

3. Private access through private Load Balancer from any connected private network
4. Public access through public Load Balancer sits in the public subnet of the VPC

Egress traffic



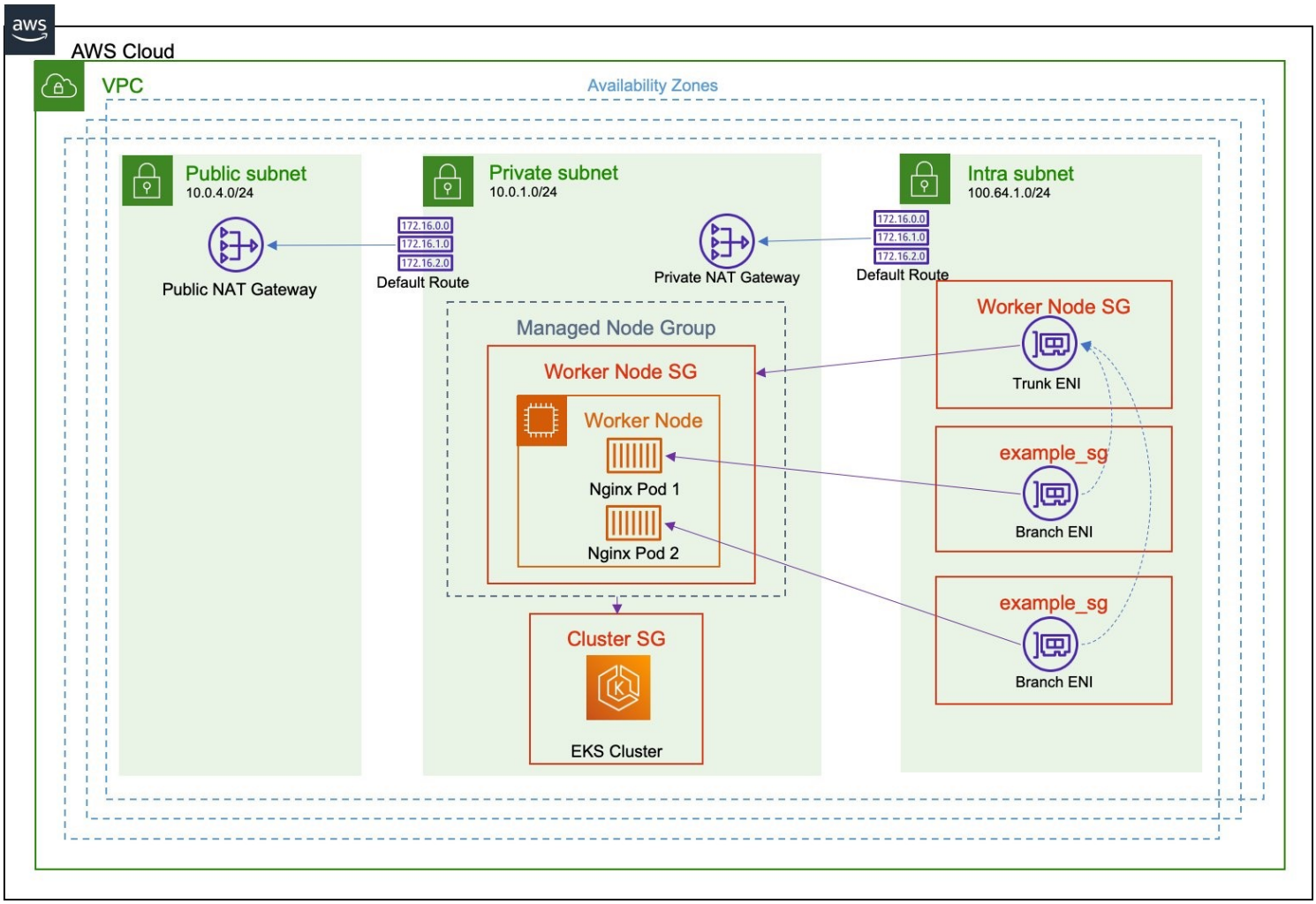
1. From pod to pod in the same cluster
2. From pod to other services in the same VPC, such as Amazon RDS databases
3. From pod to any connected private network through NAT
    1. Pod SNAT by AWS CNI plugin is used
    2. A private NAT gateway is used
4. From pod to internet through proxy server in connected network

In the previous diagrams, different security groups are attached to a pod (branch ENI), worker nodes, the Amazon EKS control plane, Amazon RDS, and ALB to control network inbound and outbound access from them.

Walkthrough

This walkthrough shows you how to automate the setup of an Amazon EKS cluster with CNI custom networking alongside security groups for pods. The sample code for deploying this stack in Terraform is hosted in this GitHub repo.

The example code builds the following resources.



Prerequisites

- An AWS Identity and Access Management (IAM) role with permissions to create Amazon EKS and Amazon EC2 resources
- A Mac or Linux terminal with Terraform and kubectl installed

Build steps

The following code is to trigger the build:

```
$ git clone https://github.com/aws-samples/terraform-cni-custom-network-sample
$ terraform init
$ terraform apply
```

This Terraform module will deploy an Amazon EKS cluster with custom networking and security for pods configured.

The following is a breakdown of what the module deploys:

1. It creates a VPC with public subnets for the public NAT gateway, private subnets for the Amazon EKS cluster, and Amazon EC2 worker nodes, and non-routable intra subnets for Kubernetes pods. A public NAT gateway is created in a public subnet and used as a default gateway for the private subnets. A private NAT gateway is created in one of the private subnets and used as a default gateway for the intra subnets.
2. Create an Amazon EKS cluster in the private subnets with the Kubernetes API server exposed publicly. This can be changed to a private API server endpoint if required.
3. Configure CNI custom networking. Two steps need to be completed to set up CNI custom networking.

First, to set up CNI custom networking, set the required environment variables for the AWS CNI plugin. This is an `aws-node` daemon set deployed into the cluster:

```
kubectl set env daemonset aws-node -n kube-system AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=t
kubectl set env daemonset aws-node -n kube-system ENI_CONFIG_LABEL_DEF=topology.kubern
```

Here `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG` enables CNI custom networking, and `ENI_CONFIG_LABEL_DEF` defines the worker node label that selects which `ENIConfig` to use for each node. The `topology.kubernetes.io/zone` label is automatically set on each node in a managed node group.

Second, set up the `ENIConfig` custom resource configuration that points to the intra subnets. An example of `ENIConfig` for an availability zone is described in the following code:

```
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
 name: "ap-southeast-2a"
spec:
 subnet: "${subnet_a}"
 securityGroups:
 - ${NODE_SG}
```

In this example, we used the availability zone name `ap-southeast-2a` as the name of the `ENIConfig`. For worker nodes launched into the `ap-southeast-2a` availability zone, the `topology.kubernetes.io/zone` label is `ap-southeast-2a` for this particular `ENIConfig`. More on ENIConfig custom resources can be found here.

4. Create an Amazon EKS managed node group. Creating the managed node group after enabling custom networking means the pods don't have to be migrated over to the new subnet. If you enable CNI custom networking after the managed node groups has been created, then all worker nodes need to be replaced to make CNI custom networking effective.

5. Configure a security group policy within the Kubernetes cluster. To set up a security group for a pod, the AWS CNI plugin (`aws-node`) needs to be configured with a new environment variable.

```
kubectl set env daemonset aws-node -n kube-system ENABLE_POD_ENI=true
kubectl patch daemonset aws-node \
  -n kube-system \
  -p '{"spec": {"template": {"spec": {"initContainers": [{"env":[{"name":"DISABLE_TCP_
```

Then we need to create security group policies, which are shown in the following code:

```
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
  name: my-security-group-policy
  namespace: test-namespace
spec:
  podSelector:
    matchLabels:
      role: test-role
  securityGroups:
    groupIds:
      - ${SECURITY_GROUP}
```

With this policy in place, all pods with the `role: test-role` label use a branch ENI with the specified security group(s) attached. More on security group policy can be found here.

6. Launch an example deployment. We create a deployment to create two sample `nignx` pods with the `role: test-role` label. These pods are deployed into the intra subnets and use the security group for pods.

Check the resources

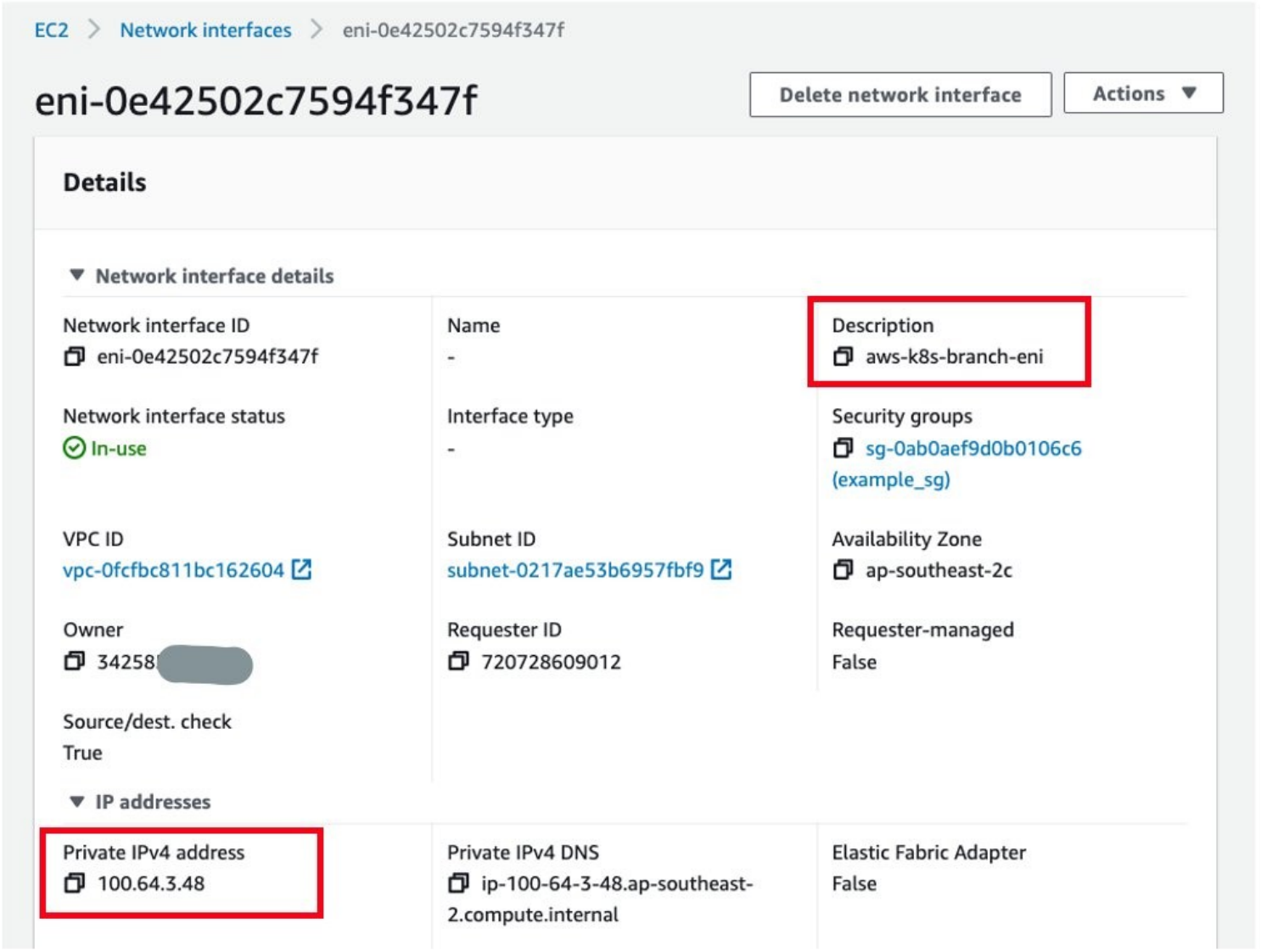
The Terraform build takes about 14 minutes to finish the following command to set up `kubectl` to access your cluster:

```
aws eks update-kubeconfig --name CNICustomNetworkDemoEKS --region <region-name>
```

Then you can access your cluster:

```
$ kubectl get pods -A -o wide
NAMESPACE        NAME                                  READY   STATUS    RESTARTS   AGE
kube-system      aws-node-79th4                        1/1     Running   0          3h3
kube-system      coredns-68f7974869-jlgsc              1/1     Running   0          3h3
kube-system      coredns-68f7974869-qxn22              1/1     Running   0          3h3
kube-system      kube-proxy-pj7vl                      1/1     Running   0          3h3
test-namespace   deployment-example-744b74884-j4jxn    1/1     Running   0          3h2
test-namespace   deployment-example-744b74884-tgx9j    1/1     Running   0          3h1
```

You can see the two `nginx` pods are launched into 100.64 subnet. When you search the IP address of the pods (100.64.3.48 in our case) from the **Network Interfaces** section of the Amazon EC2 console, you see that the ENI has the `aws-k8s-branch-eni` description and the security group with the `example_sg` name is used for the ENI, as shown in the following diagram.



You can add more resources, such as Amazon RDS, into your VPC to further test the specific use cases you are after.

Cleanup

When you are done testing, your resources can be cleaned up by running the following:

```
terraform destroy
```

Conclusion

With custom configurations of the AWS CNI plugin, CNI custom networking is enabled, which gives non-routable 16 IP addresses for each cluster while maintaining the common ingress and egress traffic patterns. In addition, pod security groups use a VPC network for communication and security groups to control inbound and outbound traffic.

Additional resources

The following are some further readings on Amazon EKS pod networking:

- Pod networking documentation
- AWS CNI GitHub repo

TAGS: Amazon EKS, Amazon VPC, Transit Gateway