**Adding Secondary CIDR Range to EKS-Cluster**

Jaswanth Kumar Jonnalagadda

Amazon Web Services

October 2019



© 2019 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without

prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited.  
All trademarks are the property of their owner

Table of Contents

[PreRequisite: 2](#_Toc23278625)

[CONFIGURE CNI: 5](#_Toc23278626)

[Configure Custom Networking: 6](#_Toc23278627)

[CREATE CRDS: 7](#_Toc23278628)

[Launching PODS in secondary CIDR Network: 9](#_Toc23278629)

[Amazon Elastic Container Service for Kubernetes (EKS)](https://aws.amazon.com/eks/) now allows clusters to be created in a Amazon VPC addressed with additional IPv4 CIDR blocks in the 100.64.0.0/10 and 198.19.0.0/16 ranges. This allows customers additional flexibility in configuring the networking for their EKS clusters.

Previously, EKS customers could only create clusters in VPCs that were addressed with [RFC 1918 private IP address ranges](https://docs.aws.amazon.com/vpc/latest/userguide/vpc-ip-addressing.html#vpc-private-ipv4-addresses). This meant customers were often unable to allocate sufficient private IP address space to support the number of Kubernetes pods managed by EKS

Now, customers can create EKS clusters in Amazon VPCs addressed with CIDR blocks in the 100.64.0.0/10 and 198.19.0.0/16 ranges. This gives customers more available IP addresses for their pods managed by Amazon EKS and more flexibility for networking architectures. Additionally, by [adding secondary CIDR blocks to a VPC](https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Subnets.html#vpc-resize) from the 100.64.0.0/10 and 198.19.0.0/16 ranges, in conjunction with the [CNI Custom Networking feature](https://docs.aws.amazon.com/eks/latest/userguide/cni-custom-network.html), it is possible for pods to no longer consume any RFC 1918 IP addresses in a VPC.

In this document, we will walk you through the configuration that is needed so that one can launch the Pod networking on top of secondary CIDRs

# **PreRequisite**:

1. With the following commands you add 100.64.0.0/16 to your EKS cluster VPC

VPC\_ID=$(aws ec2 describe-vpcs --filters Name=tag:Name,Values=<Name of EKS Cluster>\* | jq -r '.Vpcs[].VpcId')

aws ec2 associate-vpc-cidr-block --vpc-id $VPC\_ID --cidr-block 100.64.0.0/16

Sample Output:

{

"CidrBlockAssociation": {

"AssociationId": "vpc-cidr-assoc-0fc2518619e001a5d",

"CidrBlock": "100.64.0.0/16",

"CidrBlockState": {

"State": "associating"

}

},

"VpcId": "vpc-0674822c23b03659e"

}

1. Next step is to create subnets.
   1. We need to know how many subnets we are consuming, with the command

aws ec2 describe-instances --filters "Name=tag: alpha.eksctl.io/cluster-name,Values=<Name of EKS Cluster>\*" –query 'Reservations[\*].Instances[\*].[PrivateDnsName,Tags[?Key==`Name`].Value|[0],Placement.AvailabilityZone,PrivateIpAddress,PublicIpAddress]' --output table

Sample Output:

DescribeInstances

+--------------------------------------------+-----------------------------------------------------+-------------+--------------+--------------+

| ip-10-3-3-229.us-west-2.compute.internal | EFX-CIDR-TST-eks-system-workload-Node | us-west-2b | 10.3.3.229 | 54.213.9.45 |

| ip-10-3-13-241.us-west-2.compute.internal | EFX-CIDR-TST-eks-private-workloads-tenanttwo-Node | us-west-2b | 10.3.13.241 | None |

| ip-10-3-13-214.us-west-2.compute.internal | EFX-CIDR-TST-eks-private-workloads-tenantone-Node | us-west-2b | 10.3.13.214 | None |

| ip-10-3-12-121.us-west-2.compute.internal | EFX-CIDR-TST-eks-private-workloads-tenantone-Node | us-west-2a | 10.3.12.121 | None |

| ip-10-3-12-54.us-west-2.compute.internal | EFX-CIDR-TST-eks-private-workloads-tenanttwo-Node | us-west-2a | 10.3.12.54 | None |

+--------------------------------------------+-----------------------------------------------------+-------------+--------------+--------------+

Here I have 5 instances running on 2 subnets. For the purpose of demo, we will use the same AZ’s and create 2 secondary CIDR subnets. One can customize according to the network requirements.

* 1. Create two subnets with the following commands

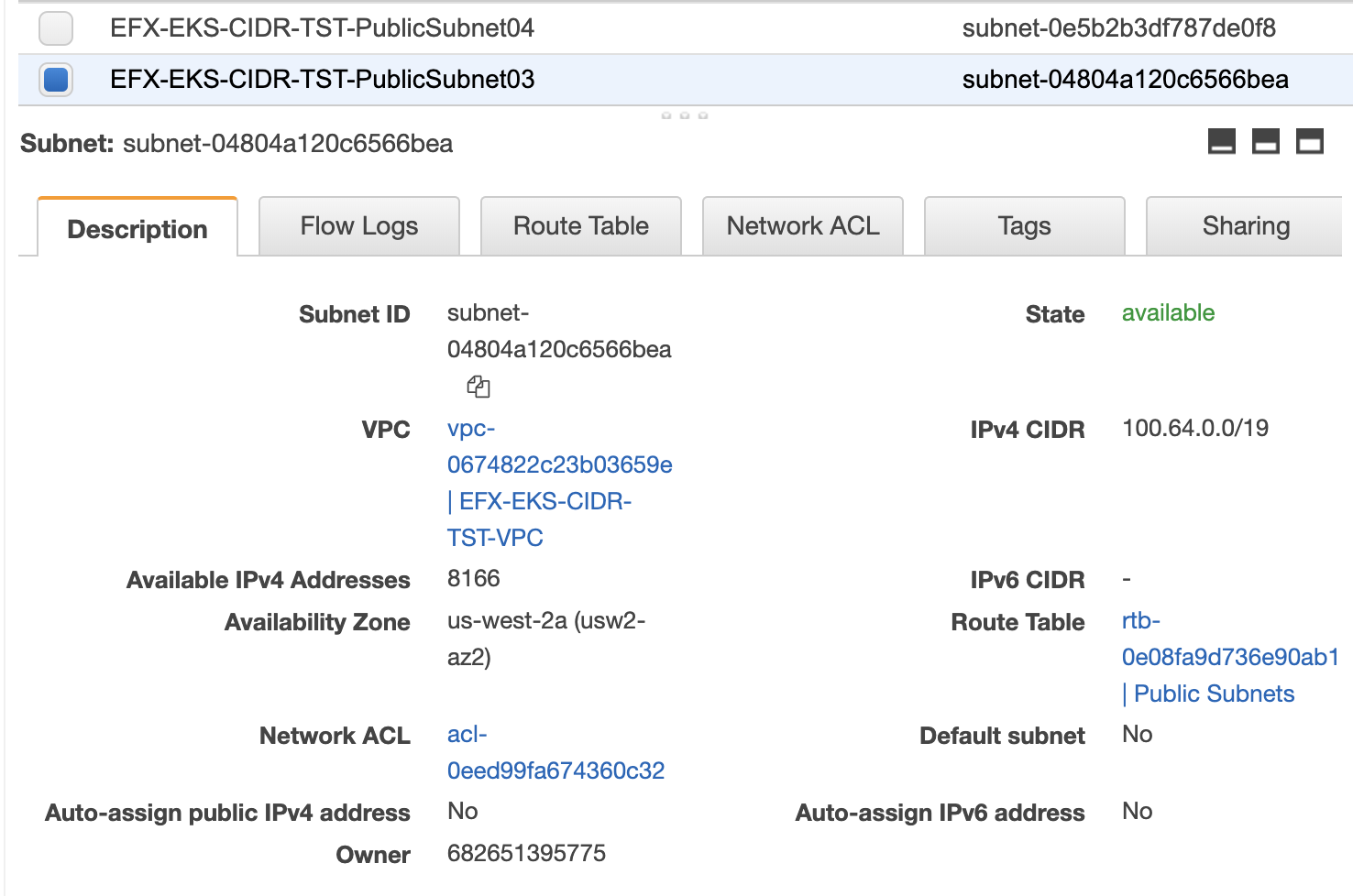
export AZ1=us-west-2a

export AZ2=us-west-2b

CGNAT\_SNET1=$(aws ec2 create-subnet --cidr-block 100.64.0.0/19 --vpc-id $VPC\_ID --availability-zone $AZ1 | jq -r .Subnet.SubnetId)

CGNAT\_SNET2=$(aws ec2 create-subnet --cidr-block 100.64.32.0/19 --vpc-id $VPC\_ID --availability-zone $AZ2 | jq -r .Subnet.SubnetId)

We can login to the AWS console and see two new subnets created.



1. Add necessary tags on created subnets.
   1. Retrieve the tags by querying current subnets

aws ec2 describe-subnets --filters "Name=tag:aws:cloudformation:stack-name,Values=<cf-stack>" "Name=cidr-block,Values=<cidr-block>" --output text

Sample Output:

TAGS aws:cloudformation:logical-id PublicSubnet01

TAGS Name EFX-EKS-CIDR-TST-PublicSubnet01

TAGS kubernetes.io/cluster/EFX-CIDR-TST shared

TAGS kubernetes.io/role/elb 1

TAGS aws:cloudformation:stack-name EFX-EKS-CIDR-TST

TAGS aws:cloudformation:stack-id arn:aws:cloudformation:us-west-2:682651395775:stack/EFX-EKS-CIDR-TST/03f06380-f862-11e9-a2e2-0a134f485a1c

* 1. Add these tags through AWS Console or through AWS CLI.

Using AWS CLI, with the commands

aws ec2 create-tags --resources $CGNAT\_SNET1 --tags Key= Name ,Value= EFX-EKS-CIDR-TST-PublicSubnet03

aws ec2 create-tags --resources $CGNAT\_SNET1 --tags Key=kubernetes.io/cluster/<Name of EKS Cluster>\*,Value=shared

aws ec2 create-tags --resources $CGNAT\_SNET1 --tags Key=kubernetes.io/role/elb,Value=1

Similarly add tags to other created subnets

1. Next step, we need to associate the subnets into the route table. For the demo purpose we will be adding the subnets to public route table that had connectivity to internet gateway
   1. Retrieving the subnet id from existing subnet with the command

SNET1=$(aws ec2 describe-subnets --filters "Name=tag:aws:cloudformation:stack-name,Values=EFX-EKS-CIDR-TST" "Name=cidr-block,Values=10.3.2.0/24" | jq -r .Subnets[].SubnetId)

* 1. Retrieve route table id with the command

RTASSOC\_ID=$(aws ec2 describe-route-tables --filters Name=association.subnet-id,Values=$SNET1 | jq -r .RouteTables[].RouteTableId)

* 1. Associate the route table id to the newly created subnets.

aws ec2 associate-route-table --route-table-id $RTASSOC\_ID --subnet-id $CGNAT\_SNET1

aws ec2 associate-route-table --route-table-id $RTASSOC\_ID --subnet-id $CGNAT\_SNET2

With this we had created entries for the subnets in public route tables

# **CONFIGURE CNI**:

1. Make sure to use the latest CNI version
   1. To view the version, with the command

kubectl describe daemonset aws-node --namespace kube-system | grep Image | cut -d "/" -f 2

Sample Output:

amazon-k8s-cni:v1.5.3

* 1. If the version is less that 1.3 then update to the latest CNI version with the command

kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/config/v1.5.3/aws-k8s-cni.yaml

* 1. Wait for the pods are recycled. We can check the status of pods by using the command

kubectl get pods -n kube-system -w

## **Configure Custom Networking:**

1. Edit aws-node configmap and add AWS\_VPC\_K8S\_CNI\_CUSTOM\_NETWORK\_CFG environment variable to the node container spec and set it to true

With the command we can update the aws-node configmap

Add the bolded lines in the config and save the file.

spec:

containers:

- env:

**- name: AWS\_VPC\_K8S\_CNI\_CUSTOM\_NETWORK\_CFG**

**value: "true"**

- name: AWS\_VPC\_K8S\_CNI\_LOGLEVEL

value: DEBUG

- name: MY\_NODE\_NAME

1. Terminate worker nodes so that Autoscaling launches newer nodes that come bootstrapped with custom network config

NOTE: Following command will restart the worker nodes

With the following command we will be restarting the worker nodes.

INSTANCE\_IDS=(`aws ec2 describe-instances --query 'Reservations[\*].Instances[\*].InstanceId' --filters "Name=tag:alpha.eksctl.io/cluster-name,Values==<Name of EKS Cluster>" --output text` )

for i in "${INSTANCE\_IDS[@]}"

do

echo "Terminating EC2 instance $i ..."

aws ec2 terminate-instances --instance-ids $i

done

Sample Output:

. . .

Terminating EC2 instance i-09570bf643ec67507 ...

{

"TerminatingInstances": [

{

"InstanceId": "i-09570bf643ec67507",

"CurrentState": {

"Code": 32,

"Name": "shutting-down"

},

"PreviousState": {

"Code": 16,

"Name": "running"

}

}

]

}

. . .

Restating might take around 20 minutes depending upon the number of instances and the autoscaling groups

# **CREATE CRDS**:

In this phase we will add custom resources to ENIConfig custom resource definition (CRD). CRD’s are extensions of Kubernetes API that stores collection of API objects of certain kind. In this case, we will store VPC Subnet and SecurityGroup configuration information in these CRD’s so that Worker nodes can access them to configure VPC CNI plugin

EKS cluster when spun up will come up with a default ENIConfig CRD, we can check that with the command

kubectl get crd

Sample Output:

NAME CREATED AT

eniconfigs.crd.k8s.amazonaws.com 2019-10-27T03:54:21Z

If the ENIConfig is missing, we can install it with the command

kubectl apply -f http<s://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/>master/config/v1.3/aws-k8s-cni.yaml

1. Create custom resources for each subnet by replacing the subnet id and security groups

Sample config file

apiVersion: crd.k8s.amazonaws.com/v1alpha1

kind: ENIConfig

metadata:

name: group1-pod-netconfig

spec:

subnet: $SUBNETID1

securityGroups:

- $SECURITYGROUPID1

- $SECURITYGROUPID2

For this demo purpose we will be creating two files for two new subnets we created.

Sample files with actual values

apiVersion: crd.k8s.amazonaws.com/v1alpha1

kind: ENIConfig

metadata:

name: group1-pod-netconfig

spec:

subnet: subnet-04f960ffc8be6865c

securityGroups:

- sg-070d03008bda531ad

- sg-06e5cab8e5d6f16ef

And save the files as group1-pod-netconfig.yaml

1. Apply the created CRD’s with the kubectl command

Kubectl apply -f group1-pod-netconfig.yaml

Kubectl apply -f group2-pod-netconfig.yaml

1. We can view the configs created with the command

kubectl get ENIConfig

Sample Output:

NAME AGE

group1-pod-netconfig 9h

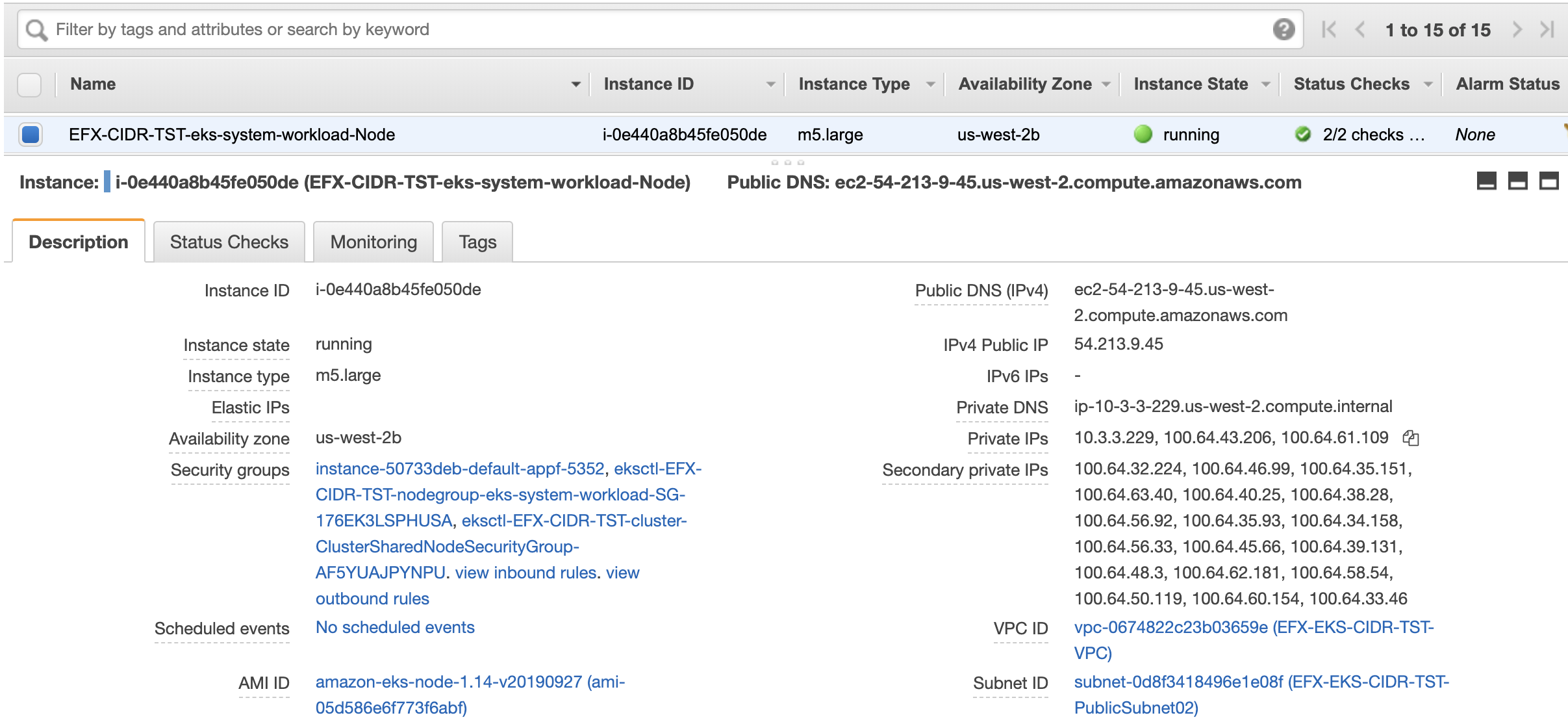
group2-pod-netconfig 9h

1. As the last step will annotate the nodes to take advantage of secondary ips we added.

We will be annotating the nodes with the command

kubectl annotate node <private ip dns of ec2> k8s.amazonaws.com/eniConfig=group1-pod-netconfig

Once annotated we can see secondary ip’s for the annotated nodes as below.



With this we can observe the secondary ip range attached to the worker node.

# **Launching PODS in secondary CIDR Network**:

Once everything is setup we can launch pods in the new ip range.

Let’s launch a nginx pod and see the ip’s the pod is attached with

kubectl run nginx --image=nginx

kubectl scale --replicas=2 deployments/nginx

kubectl expose deployment/nginx --type=NodePort --port 80

With this we deployed a nginx pod with 2 replicas.

View the ip’s attached to the pods with the command:

kubectl get pods -o wide

Sample Output:

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE

nginx-64f497f8fd-k962k 1/1 Running 0 40m **100.64.6.147** ip-192-168-52-113.us-east-2.compute.internal <none>

nginx-64f497f8fd-lkslh 1/1 Running 0 40m **100.64.53.10** ip-192-168-74-125.us-east-2.compute.internal <none>

Here we can observe that new pods are launched in the new CIDR range.

With this demo we can attach secondary IP range to and existing EKS Cluster.