使用EKS Multi-Arch集群实现降本增效Workshop

一、实验环境说明

Github Repo:

> https://github.com/goodbyedavid/EKS-MultiArch

Kubernetes Version 1.26

AddOn Version：

- kube-proxy v1.26.2-eksbuild.1
- CoreDNS v1.9.3-eksbuild.2
- VPC CNI v.12.5-eksbuild.2

Kubectl Version：V-1.26

Eksctl Version : 0.143.0

AWSCLI : 2.11.23

二、实验环境准备

1. 创建EKS集群：

> 开始使用 Amazon EKS – Amazon Web Services Management Console 和 Amazon CLI - Amazon EKS
> 了解如何使用 Amazon Web Services Management Console 和 Amazon CLI 创建您的第一个包含节点的 Amazon EKS 集群。
> https://docs.amazonaws.cn/eks/latest/userguide/getting-started-console.html

2. 创建节点组 - 需要分别创建arm64跟amd64两个节点组，每个节点组各一个实例：

创建托管节点组 - Amazon EKS

本主题介绍了如何启动向 Amazon EKS 集群注册的节点的 Amazon EKS 托管节点组。

https://docs.amazonaws.cn/eks/latest/userguide/create-managed-node-group.html

AMI类型分别选择**Amazon Linux 2(AL2_x86_64)**, **Amazon Linux Arm(AL2_ARM_64)**

3. 安装 Load Balancer Controller组件：

安装 Amazon Load Balancer Controller 附加组件 - Amazon EKS

了解如何将 Amazon Load Balancer Controller 附加组件添加到您的集群。

https://docs.amazonaws.cn/eks/latest/userguide/aws-load-balancer-controller.html

三、创建推送镜像

1. 设置环境变量

```
export AWS_ACCOUNT_ID="$(aws sts get-caller-identity --query Account --output text)"
export REGION="us-west-2"
```

2. 登入ECR服务

```
aws ecr get-login-password --region $REGION | docker login --username AWS --password-stdin $AWS_ACC
OUNT_ID.dkr.ecr.$REGION.amazonaws.com
```

3. 创建ECR仓库

```
aws ecr create-repository --repository-name go-example --region $REGION
```

4. 构建x86_64镜像

```
docker build -f Dockerfile-amd64 -t $AWS_ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/go-example:0.1-am
d64 .
docker push $AWS_ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/go-example:0.1-amd64
```

5. 构建arm64镜像

```
docker build -f Dockerfile-arm64 -t $AWS_ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/go-example:0.1-ar
m64 .
docker push $AWS_ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/go-example:0.1-arm64
```

6. 检查镜像，架构不同

```
docker image inspect $AWS_ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/go-example:0.1-amd64
docker image inspect $AWS_ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/go-example:0.1-arm64
```

7. 通过创建一个docker manifest将两者关联成一个镜像URL

```
docker manifest create $AWS_ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/go-example:0.1 --amend $AWS_AC
COUNT_ID.dkr.ecr.$REGION.amazonaws.com/go-example:0.1-amd64 --amend $AWS_ACCOUNT_ID.dkr.ecr.$REGIO
N.amazonaws.com/go-example:0.1-arm64
```

8. 推送manifest到ECR仓库

```
docker manifest push $AWS_ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/go-example:0.1
```

9. 查看manifest信息

```
docker manifest inspect $AWS_ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/go-example:0.1
```

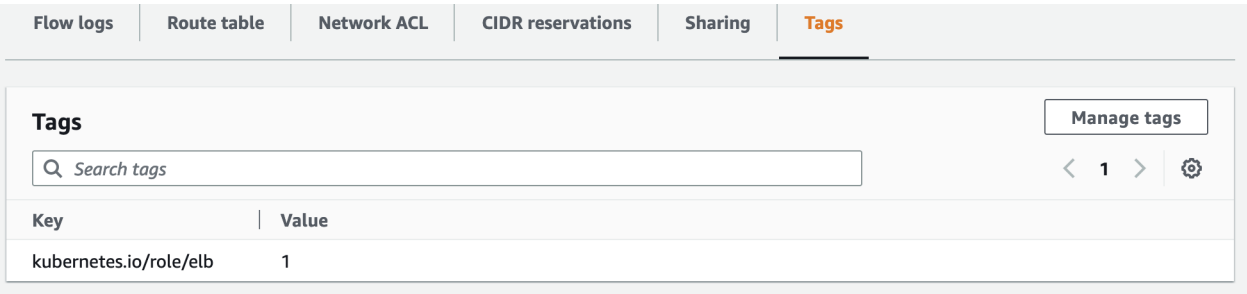
```
{
   "schemaVersion": 2,
   "mediaType": "application/vnd.docker.distribution.manifest.list.v2+json",
   "manifests": [
      {
         "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
         "size": 740,
         "digest": "sha256:73a62668aa18f4ae14acb85d770f5c989b392de23bcd0752f3b528e2d6b1e7b2",
         "platform": {
            "architecture": "amd64",
            "os": "linux"
         }
      },
      {
         "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
         "size": 740,
         "digest": "sha256:637b48ebfd84bcf960b0d9ca073a6b4c48de692cdafcb34c6429c6066c77129c",
         "platform": {
            "architecture": "arm64",
            "os": "linux",
            "variant": "v8"
         }
      }
   ]
}
```


10. 确保VPC中子网配置已经打上kubernetes相关标签：

Amazon EKS VPC and subnet requirements and considerations - Amazon EKS

This is official Amazon Web Services (AWS) documentation for Amazon Elastic Kubernetes Service (Amazon EKS). Amazon EKS is a managed service that makes it easy for you to run Kubernetes on AWS without needing to install and operate your own

https://docs.aws.amazon.com/eks/latest/userguide/network_reqs.html



四、部署验证

前面通过manifest，已经将不同架构的镜像统一成同一个URL，这样我们使用k8s deployment编排时就可以使用`$AWS_ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/go-example:0.1`来统一设置镜像URL而不区分硬件架构，k8s节点在拉取镜像时将自动匹配不同架构的镜像。

1. 确保k8s.yaml中指向您自己的ECR container image

```
containers:
      - name: go-example
        image: $AWS_ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/go-example:0.1
        ports:
        - containerPort: 8000
```

2. 创建k8s workload

```
kubectl create -f k8s.yaml
```

3. 检查pod状态

```
kubectl get pod -o wide
```

```
NAME                          READY   STATUS    RESTARTS   AGE   IP             NODE                                            NOMINATED NODE   READINESS GATES
go-example-7bb7746f55-mc5ml   1/1     Running   0          90m   172.31.46.55   ip-172-31-33-93.us-west-2.compute.internal      <none>           <none>
go-example-7bb7746f55-p9s9w   1/1     Running   0          90m   172.31.8.10    ip-172-31-3-234.us-west-2.compute.internal      <none>           <none>
```

4. 检查node

```
kubectl get node --label-columns beta.kubernetes.io/arch
```

```
NAME                                          STATUS   ROLES    AGE   VERSION               ARCH
ip-172-31-3-234.us-west-2.compute.internal    Ready    <none>   23h   v1.26.4-eks-0a21954   arm64
ip-172-31-33-93.us-west-2.compute.internal    Ready    <none>   23h   v1.26.4-eks-0a21954   amd64
```

5. 查看service配置

```
kubectl describe svc web-svc
```

```
Name:                     web-svc
Namespace:                default
Labels:                   <none>
Annotations:              service.beta.kubernetes.io/aws-load-balancer-attributes: load_balancing.cross_zone.enabled=true
                          service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp
                          service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
                          service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
                          service.beta.kubernetes.io/aws-load-balancer-type: external
Selector:                 app=go-example
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.100.174.230
IPs:                      10.100.174.230
LoadBalancer Ingress:     k8s-default-websvc-86fd064310-6e88778e10948610.elb.us-west-2.amazonaws.com
Port:                     <unset>  80/TCP
TargetPort:               8000/TCP
NodePort:                 <unset>  31976/TCP
Endpoints:                172.31.46.55:8000,172.31.8.10:8000
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

6. 通过关联pod跟node的信息，已经可以看到go-example程序同事运行在不同架构的节点上，同时svc跟loadbalancer已经关联了这两个pod。使用curl命令访问NLB的URL，可以将请求分发到不同的架构节点上的pod。

```
curl k8s-default-websvc-86fd064310-6e88778e10948610.elb.us-west-2.amazonaws.com
```