Prep: Clean up (Do not show)

- Segment07:
  - Detach Cluster Auto scaling / Dynamo policy from Node instance role
  - Kubectl delete –f eks-dynamo.yaml
  - Delete ECR Repo
  - terraform destroy
  - sls remove
- Segment06
  - Remove group from anotherUser
  - Delete dashboard, hpa-demo, meks-admin-service-account, metrics-server, role-binding
- Segment05
  - Kubectl delete –f ngx-volumes.yaml
  - Kubectl delete –f default-backend
  - Kubectl delete –f cert-manager/
  - Kubectl delete –f nginx-ingress-controller
- Segment04
  - Kubectl delete –f dnstest/

Prep: Clean up 2 (Do not show)

- Segment03:
  - Eksctl delete-cluster apr08
  - Remove manual eks cluster nodes
  - Remove manual eks cluster
  - Remove stacks in cloud formation
- Segment02
  - Terraform destroy –auto-approve

Prep: Setup (Do not show)

1. Open Firefox to namecheap and login (for DNS)
2. Open two Chrome browsers: 1 in incognito mode
3. Open Tab to github.com resources
4. Two terminals open in the source directory
5. Setup 2nd user in remote machine.
6. Get some water

Welcome!

Kubernetes on AWS

Poll Question: Level of Expertise with Kubernetes?

1. So good that I should be teaching this class instead of you. I've authored books on the subjects and know all the ins and outs.
2. Pretty good, we run it in production and we've been seen our share of issues.
3. I'm a not quite a beginner, but don't have operational experience with it.
4. I'm just getting started
5. What is Kubernetes and where am I?

Who is this guy?

- Kubernetes since 2015
- Engineer @ Weave Grid
- Portland, OR
- Wife, 4 kids
- Kubernetes Contracting Work
- Type 1 Diabetes Research
- Past:
  - IBM, Cisco, other Small companies

What we cover today

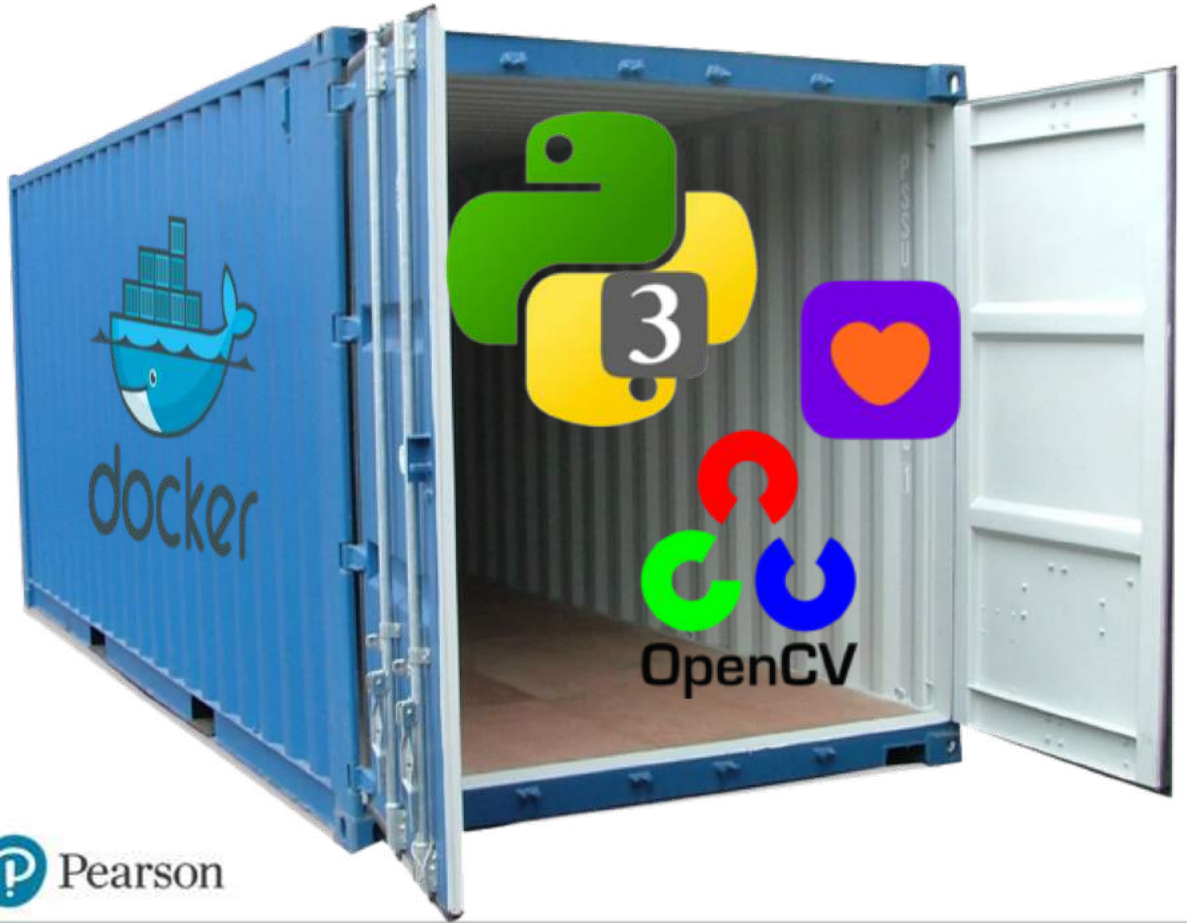
- Segment 1: Introduction
- Segment 2: IAM Configuration
- Segment 3: EKS Installation
- Segment 4: Verification
- Segment 5: Running Applications
- Segment 6: Administrative Tasks
- Segment 7: Additional AWS connections

Kubernetes on AWS

Segment 1: Introduction

What is Kubernetes?

Why Containers?

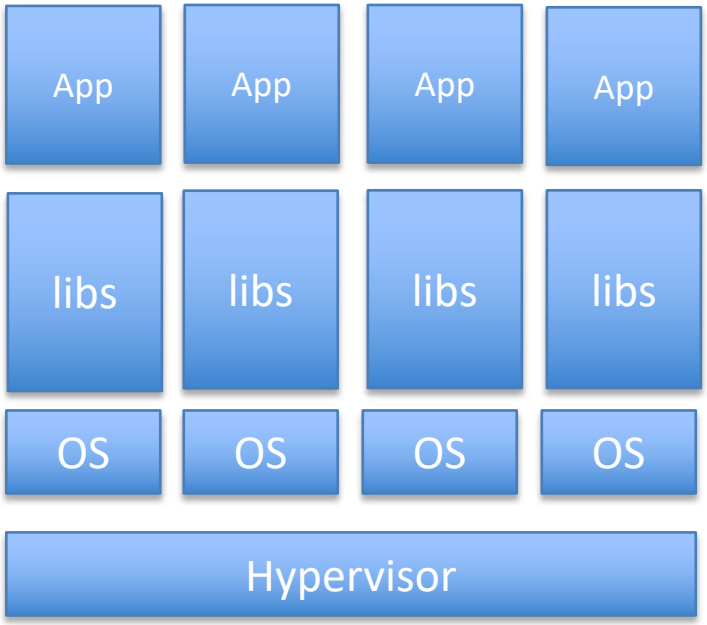


Application ships with dependencies and runtimes

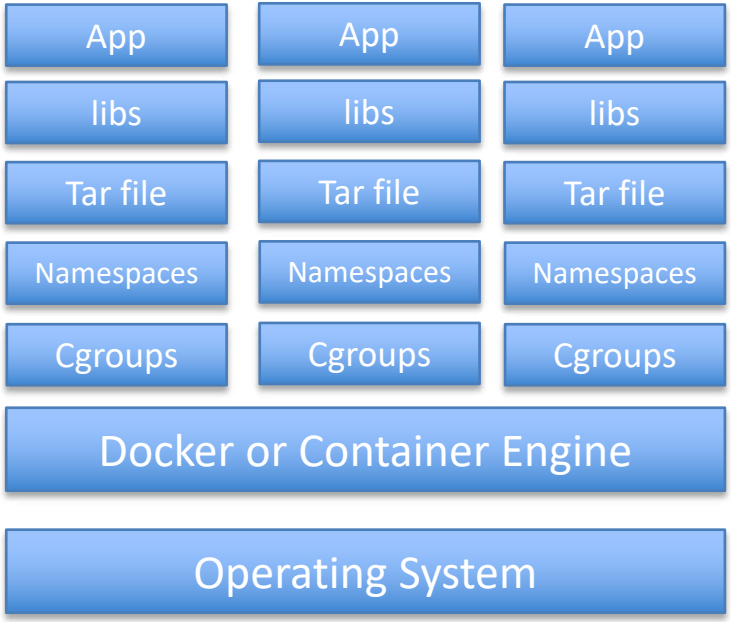
Runs the same in any environment where containers run

Standard way to start applications.

Virtual Machine vs. Container

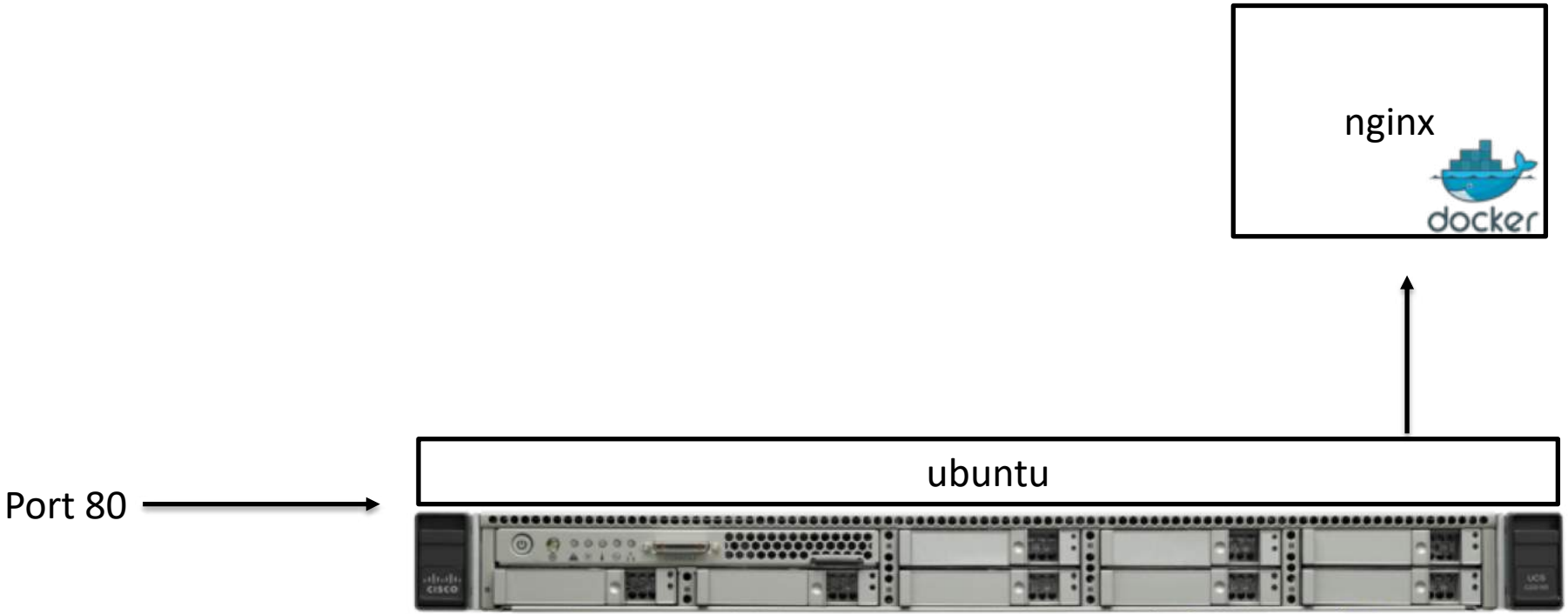


Virtual Machine Stack



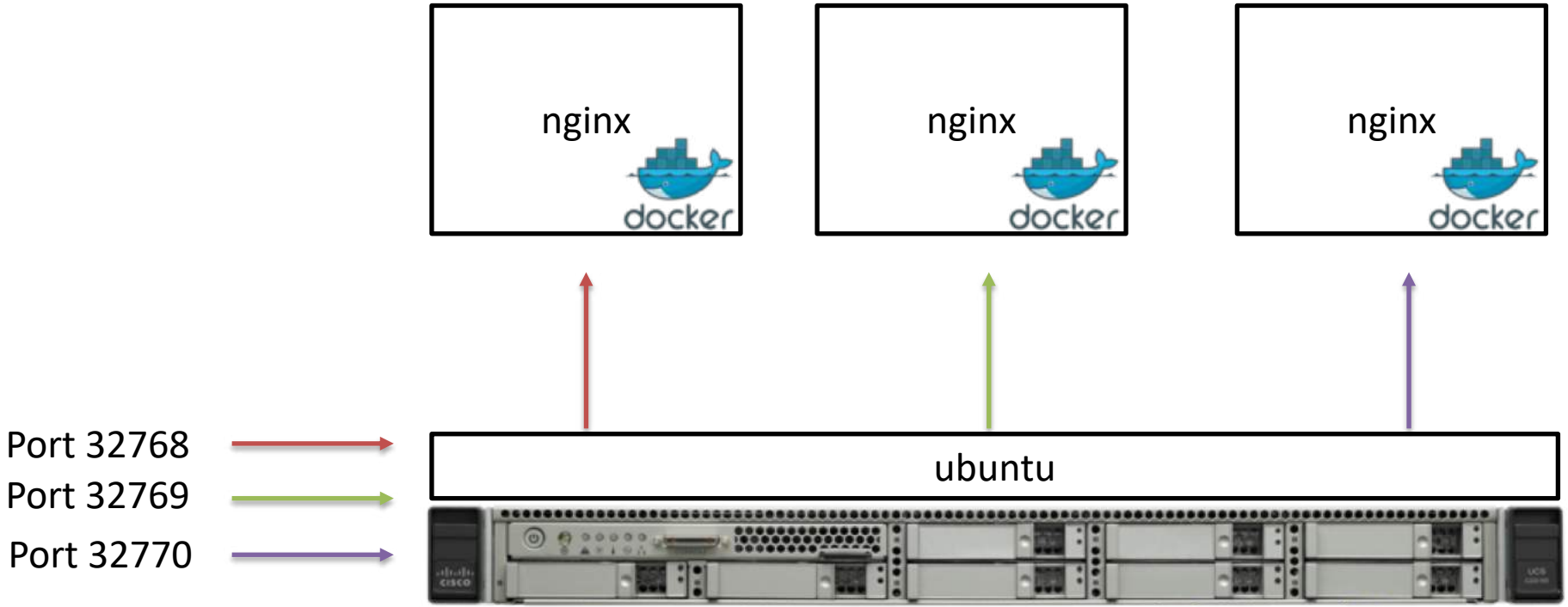
Container Stack

This is easy



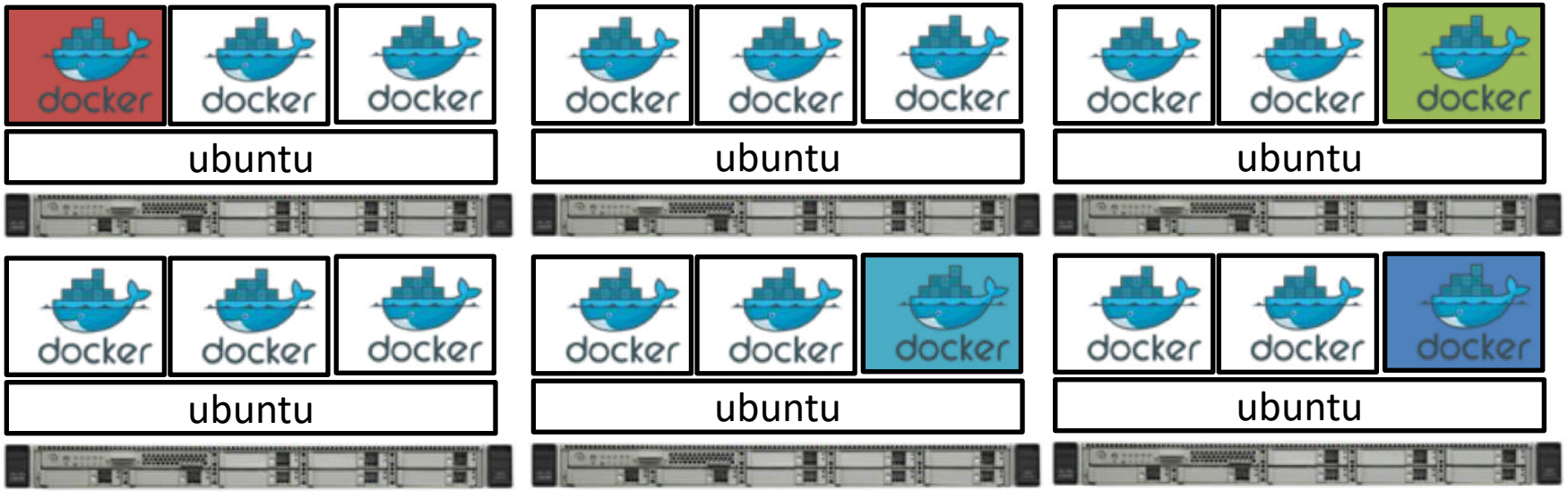

```
docker run -d -p 80:80 nginx
```

This is a little more difficult




```
for i in $(seq 3); do docker run -d -P nginx; done
```

This is starting to become a problem




```
for i in $(seq 6); do for j in $(seq 3); do ssh node0$i docker
                run -d -P app${i}-${j}; done; done
```

Container Management challenges

How do we keep track of which port goes to which container on which host?

How should we efficiently allocate containers to hosts?

Microservices scale horizontally, so how do we map service dependencies?

Applications are frequently updated, container ports are randomized, how do we account for frequent changes?

Kubernetes

Thursday, April 23, 2015

Borg: The Predecessor to Kubernetes

Google has been running containerized workloads in production for more than a decade. Whether it's service jobs like web front-ends and stateful servers, infrastructure systems like Bigtable and Spanner, or batch frameworks like MapReduce and Millwheel, virtually everything at Google runs as a container. Today, we took the wraps off of Borg, Google's long-rumored internal container-oriented cluster-management system, publishing details at the academic computer systems conference Eurosys. You can find the paper here.

Large-scale cluster management at Google with Borg

Abhishek Verma[†] Luis Pedrosa[‡] Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes
Google Inc.

Abstract

Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines.

It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior.

We present a summary of the Borg system architecture and features, important design decisions, a quantitative analysis of some of its policy decisions, and a qualitative examination of lessons learned from a decade of operational experience with it.

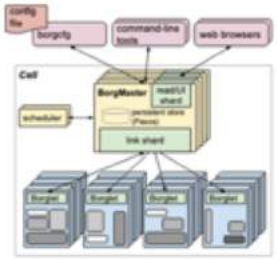


Figure 1: The high-level architecture of Borg. *Only a tiny fraction of the thousands of worker nodes are shown.*

cluding with a set of qualitative observations we have made from operating Borg in production for more than a decade.

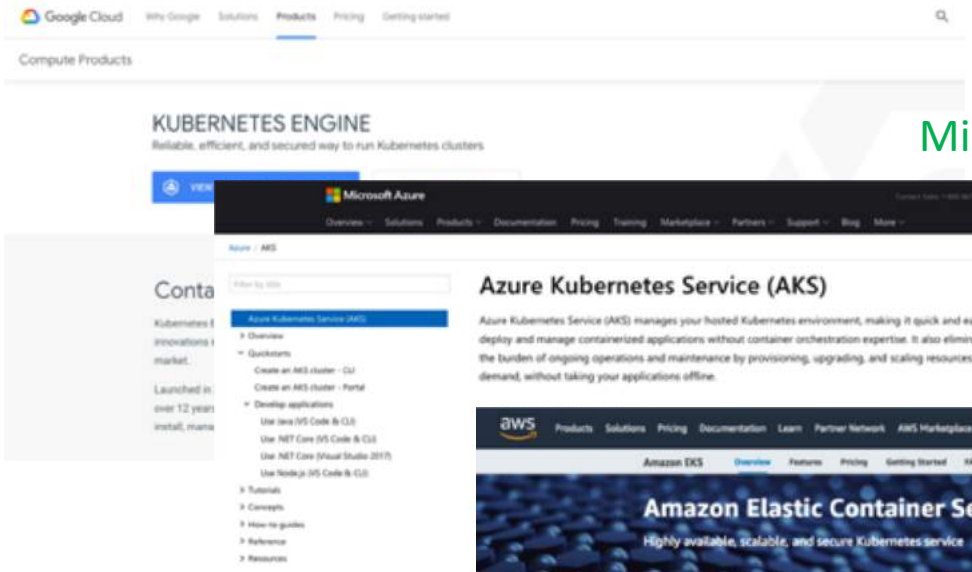
Oh, by the way, we've been doing this for 10 years.

Google

https://ai.google/research/pubs/pub43438

Kubernetes in the Cloud

Google GKE



Microsoft AKS

AWS EKS

Poll Question: Where do you run Containers today?

1. EKS!
2. AKS
3. GKS
4. Bare Metal
5. VMware
6. Instances I manage
7. Just on my own laptop
8. Some other solution: Nomad, Rancher, OpenShift…

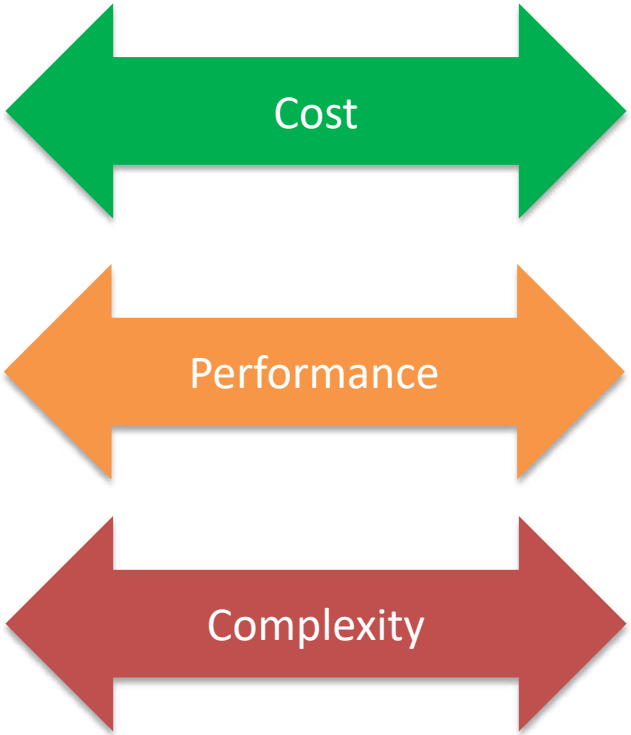
Cloud vs. OnPrem

Cloud

Cloud more expensive if operational costs are not considered.

Can scale up to hundreds of machines so if more performance is needed scale horizontally

Fully managed so many issues such as storage, load balancing, network is all taken care of.



OnPrem

Running in existing datacenter could be less but need more expertise for operational cost.

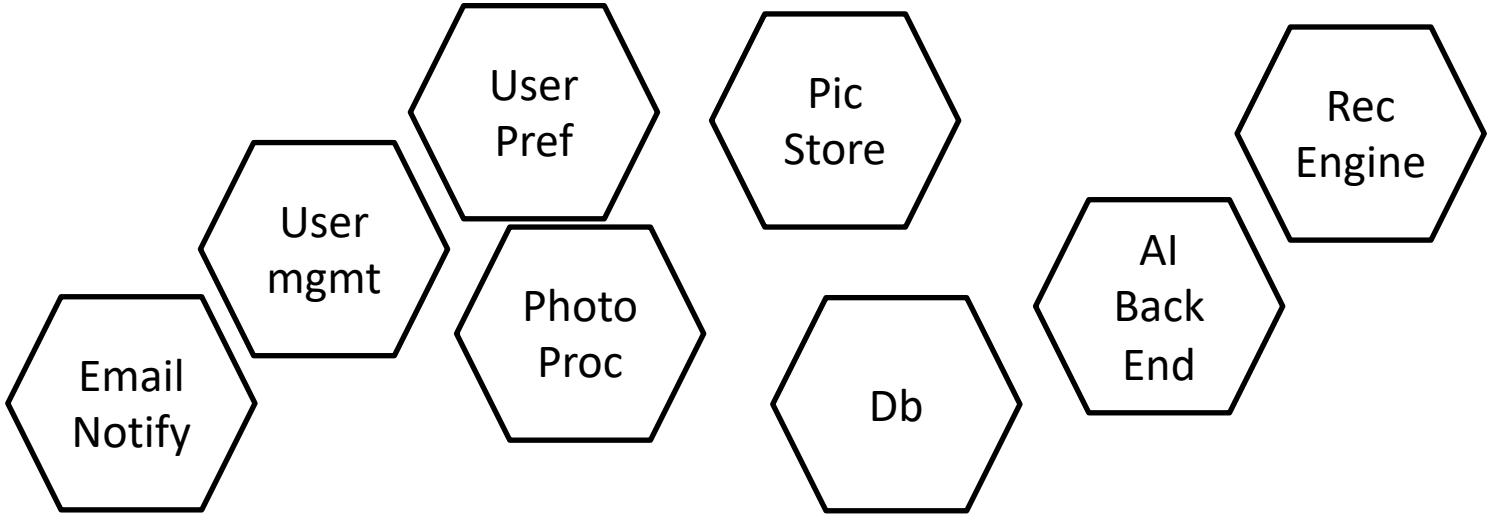
Can use more powerful hardware of any choice including special TPUs/GPUs

Requires deep expertise to run system which may be undifferentiated expertise for business.

The Cloud

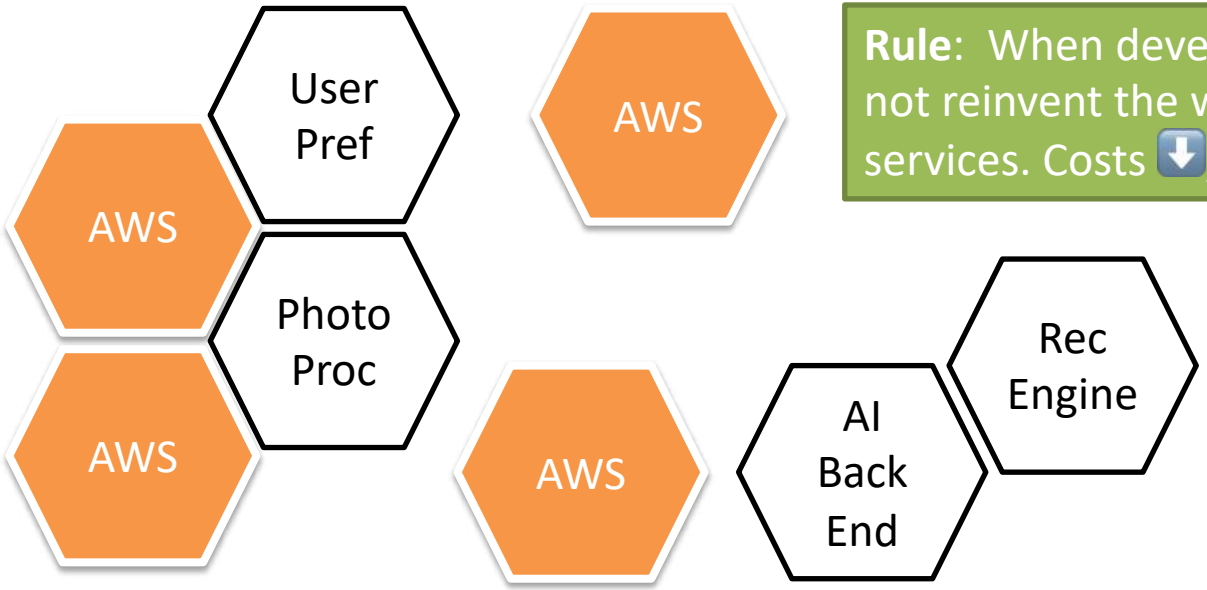
The cloud is not just self service of virtual machines

Microservices architecture has changed application development



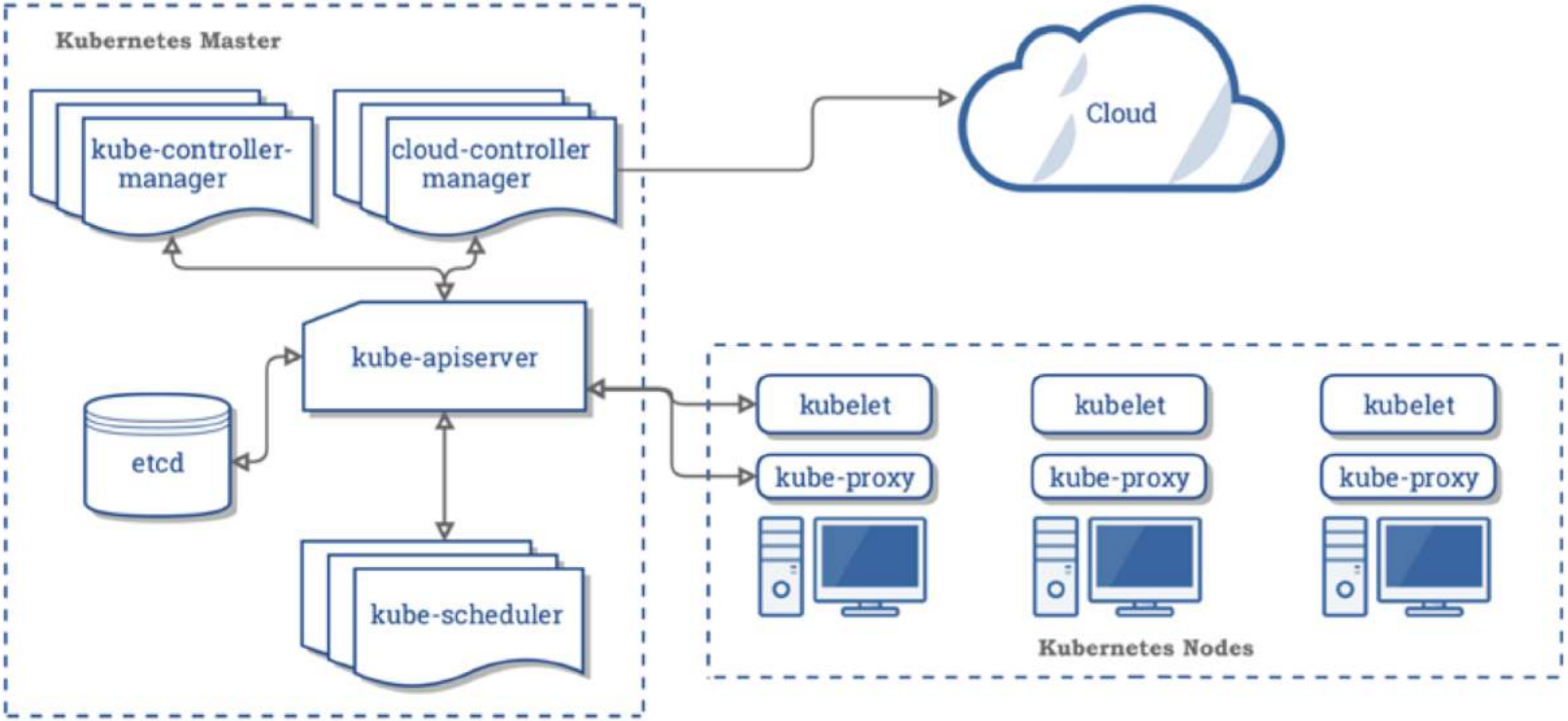
Microservices

Microservices are black boxes to each other.
To increase speed, use existing black boxes



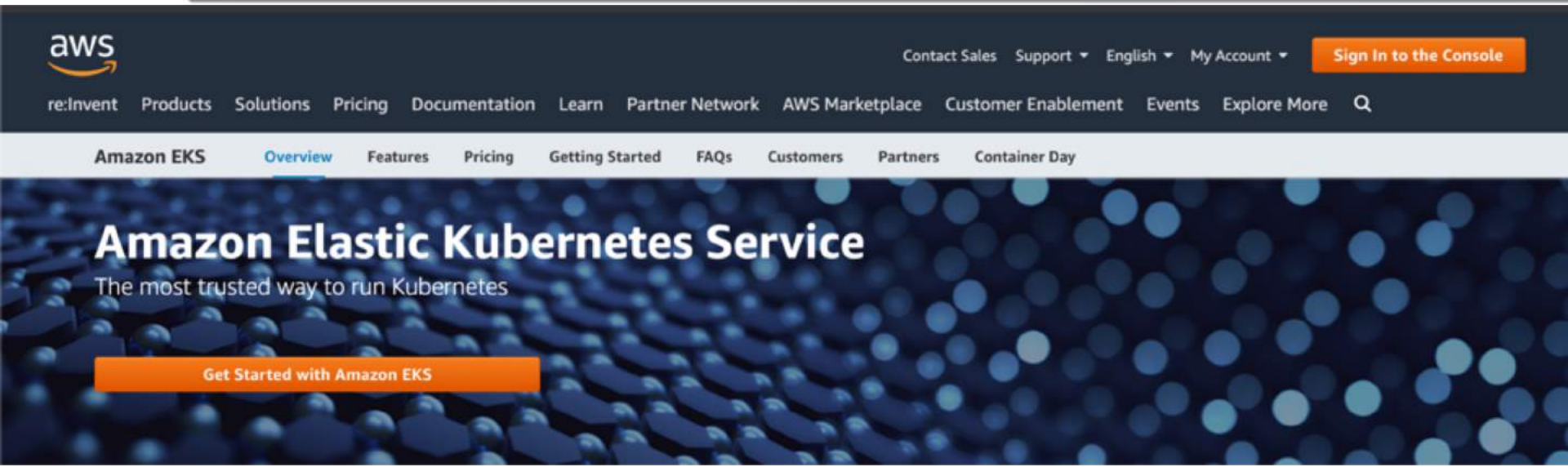
Rule: When developing applications do not reinvent the wheel. Use existing cloud services. Costs ⬇, Feature Velocity ⬆.

Kubernetes Architecture

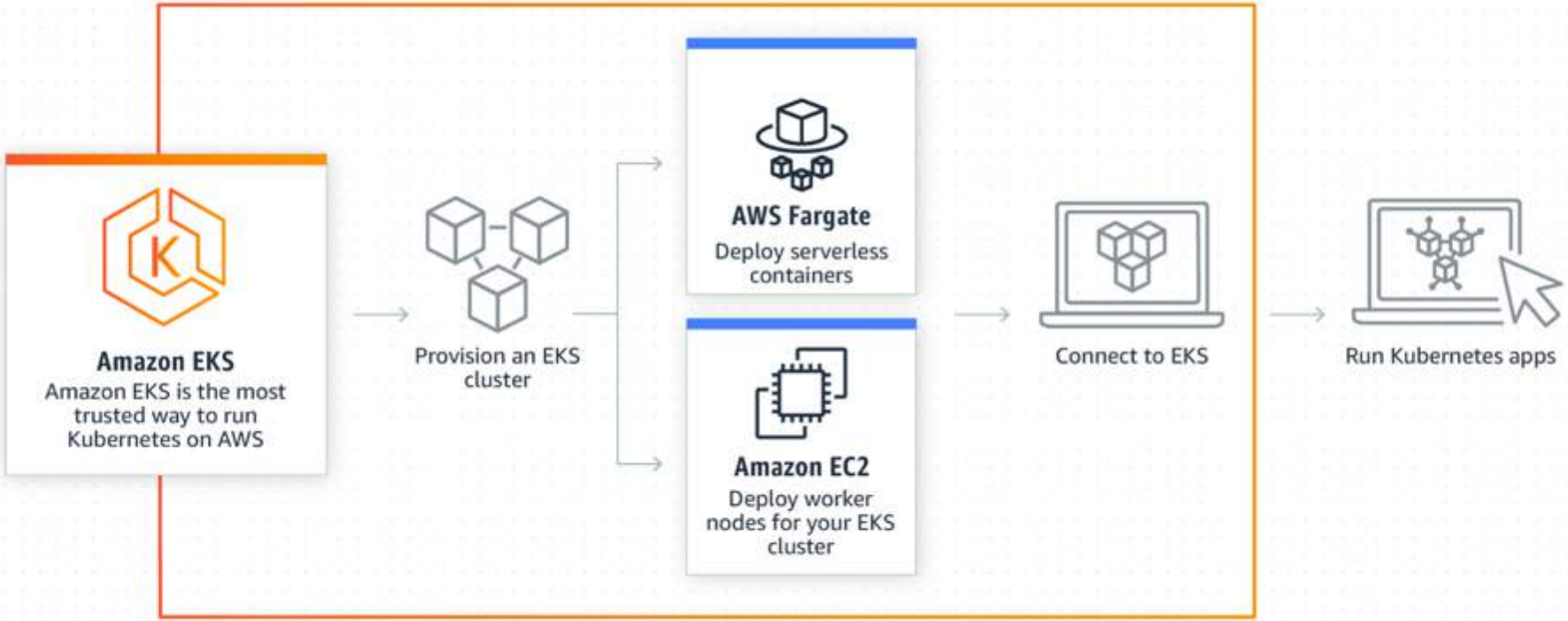


Source: https://kubernetes.io/docs/concepts/overview/components/

What is EKS?


aws
Contact Sales
Support
English
My Account
Sign In to the Console
re:Invent
Products
Solutions
Pricing
Documentation
Learn
Partner Network
AWS Marketplace
Customer Enablement
Events
Explore More
Amazon EKS
Overview
Features
Pricing
Getting Started
FAQs
Customers
Partners
Container Day
Amazon Elastic Kubernetes Service
The most trusted way to run Kubernetes
Get Started with Amazon EKS

What is EKS?



Source: https://aws.amazon.com/eks/

EKS vs. RYO AWS Instances

EKS		RYO
$0.10/hour + 3 x t3.large $0.24 hour	Cost	3 x t3.large = $0.08 x 3 = $0.24/hour, or one master: $0.08 for testing.
Only supported versions can be run: 1.12, 1.13, 1.14 as of Dec 29, 2019	Flexibility	Can run any version, software, plugins, etc.
Control plain managed, redundant, upgrades built in.	Complexity	Requires build your own control plain, manage upgrades, more work.

Other AWS Container Services

ECS

- Proprietary Docker Management service developed first to compete with Kubernetes
- Uses JSON task definition

Fargate

- Container service, no EC2 instances required. Takes place of EC2 layer
- No scaling management
- JSON task definition
- Can be used with EKS and ECS

Benefits of EKS on AWS

- No control plane management
- Built in Load Balancing, Networking, Volume Storage
- Easy to turn off and on
- Integrations with other AWS components to build applications (S3, Cognito, RedShift, RDS, Lambda, etc)

New to EKS (Dec 2020)

- Amazon EKS Distro – an open source Kubernetes distribution used by Amazon EKS
  - https://github.com/aws/eks-distro
- EKS Anywhere (Multicloud)
  - https://aws.amazon.com/eks/eks-anywhere/
- Amazon Proton for CI/CD

Initial laptop setup

- `kubectl`
  - https://github.com/vallard/EKS-Training/blob/master/segment01-intro/kubectl.md
- AWS command line tool
  - https://github.com/vallard/EKS-Training/blob/master/segment02-iam/aws-creds.md
- `eksctl` tool
- Tab completion for aws commands

Kubernetes on AWS

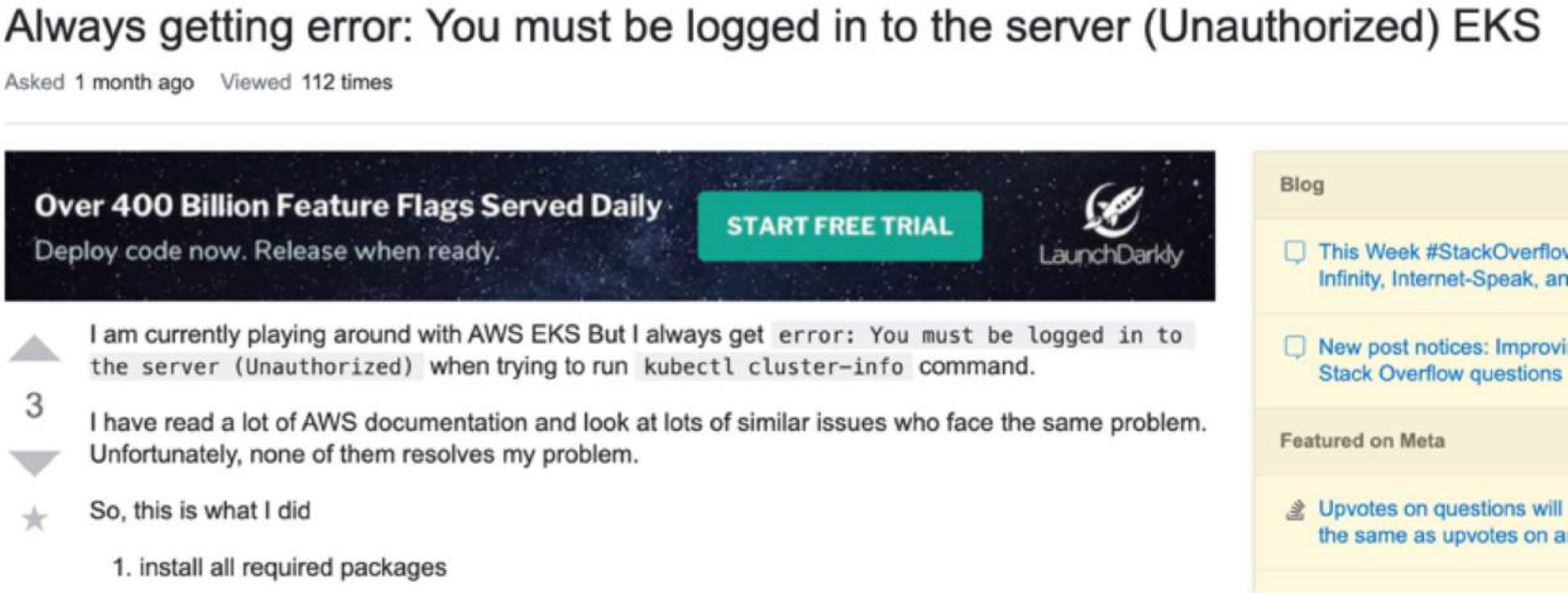
Segment 2: IAM User Accounts

Big Picture

- Create IAM Policies
- Create IAM Group
- Create IAM Roles
- Create IAM User
- Get AWS command line tools
- Login via console and command line

Root Accounts

- Don’t use root accounts
- Use root account to create other users: eksdude



Always getting error: You must be logged in to the server (Unauthorized) EKS

Asked 1 month ago Viewed 112 times

3

I am currently playing around with AWS EKS But I always get `error: You must be logged in to the server (Unauthorized)` when trying to run `kubectl cluster-info` command.

I have read a lot of AWS documentation and look at lots of similar issues who face the same problem. Unfortunately, none of them resolves my problem.

So, this is what I did

1. install all required packages

Blog

This Week #StackOverflow Infinity, Internet-Speak, an

New post notices: Improvi Stack Overflow questions

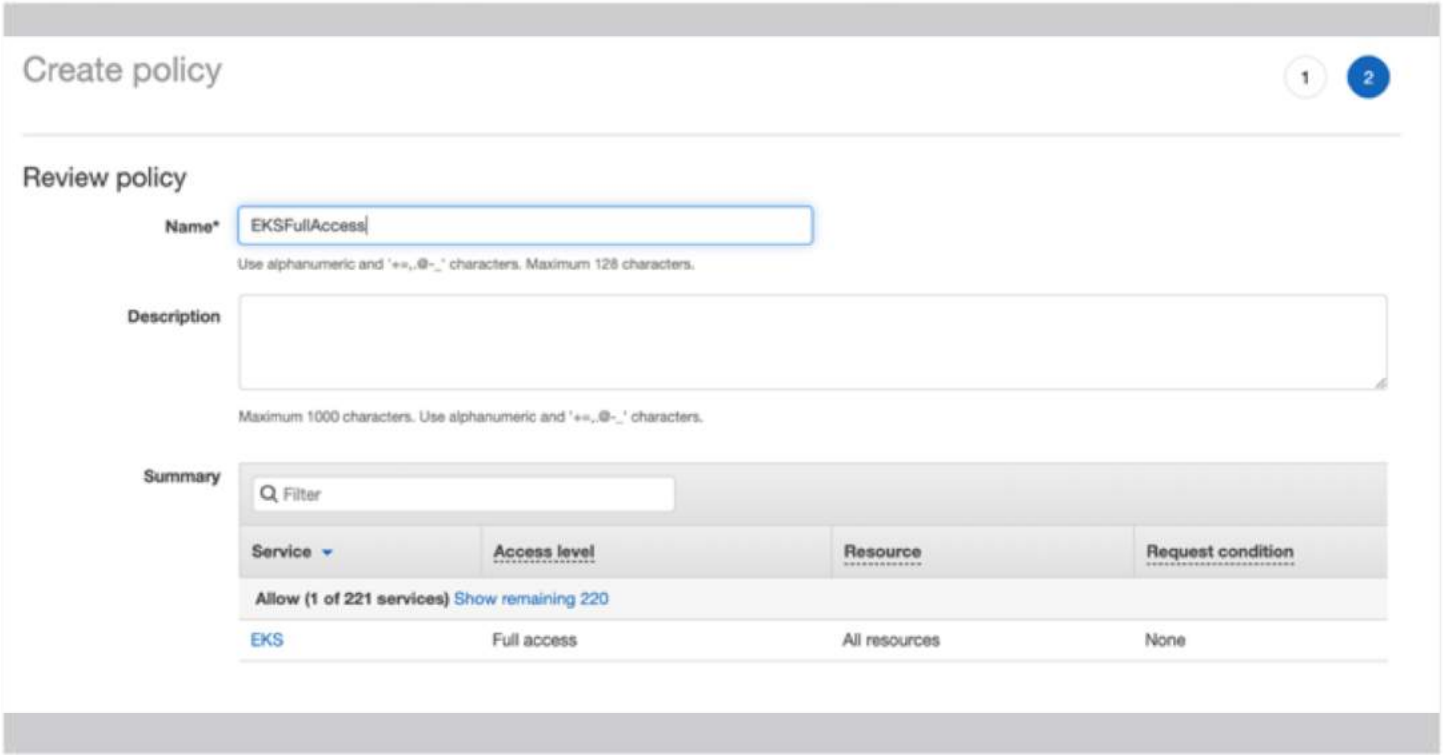
Featured on Meta

Upvotes on questions will the same as upvotes on a

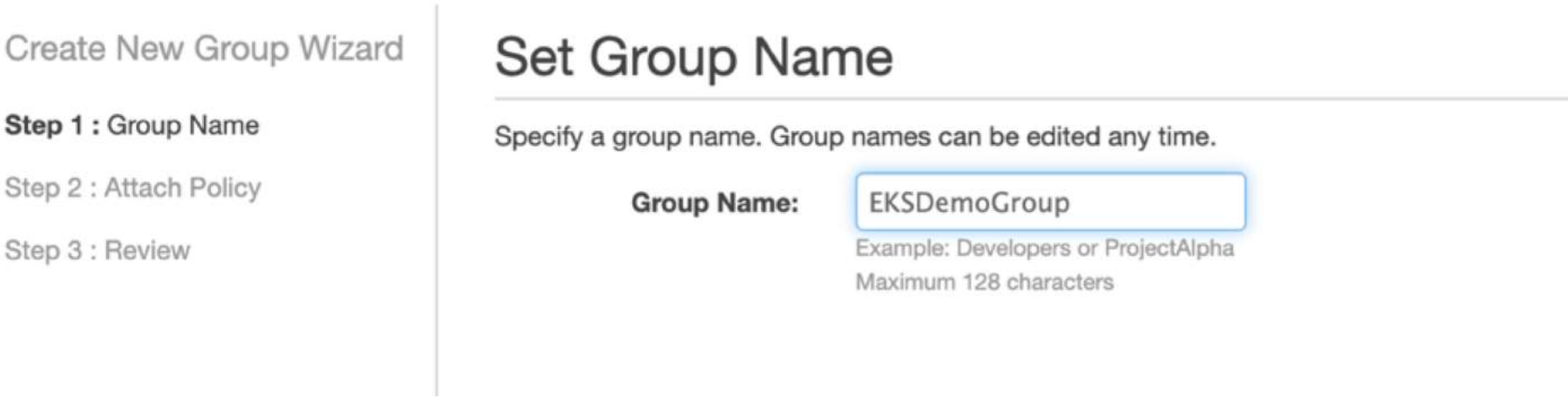
Create EKS Full Access Policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "eks:*",
            "Resource": "*"
        }
    ]
}
```

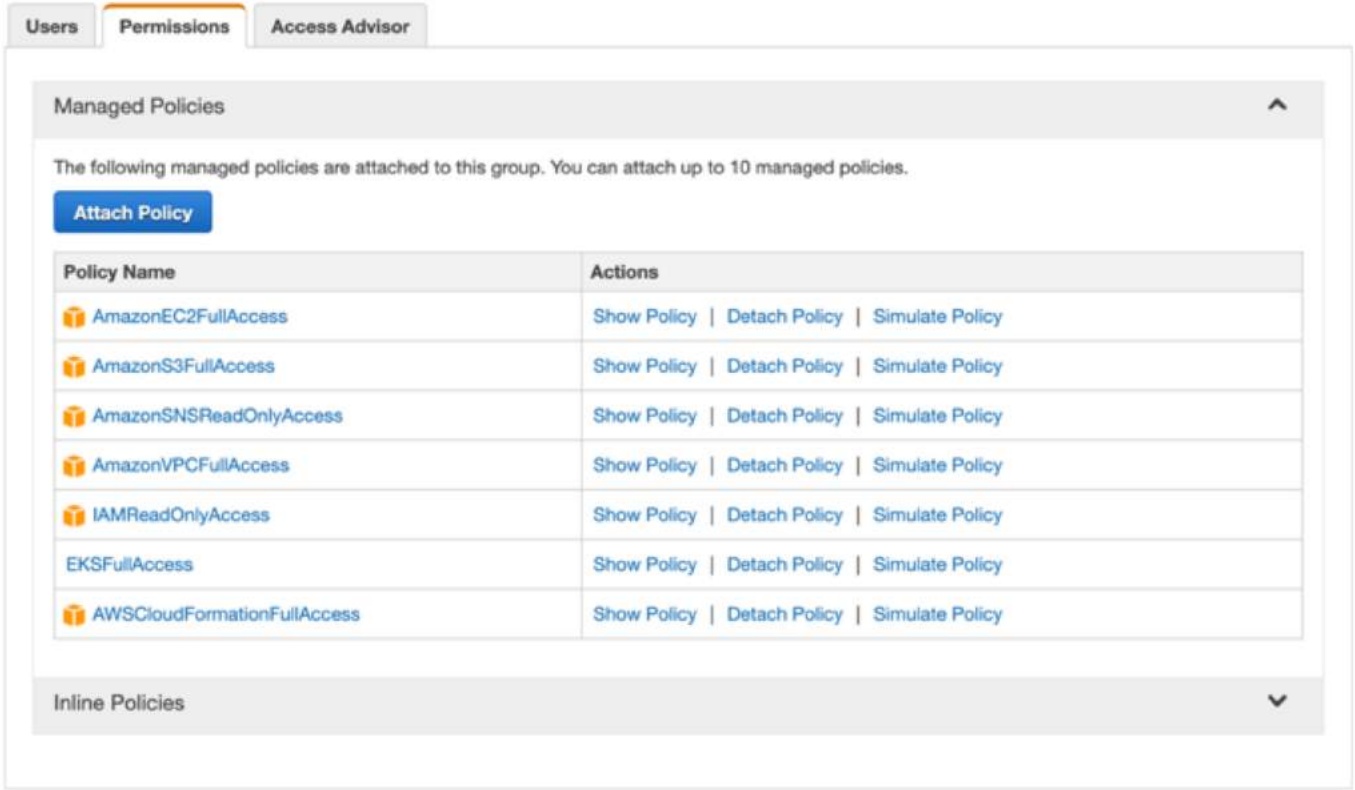
Call the Policy EKSFullAccess



Create New Group

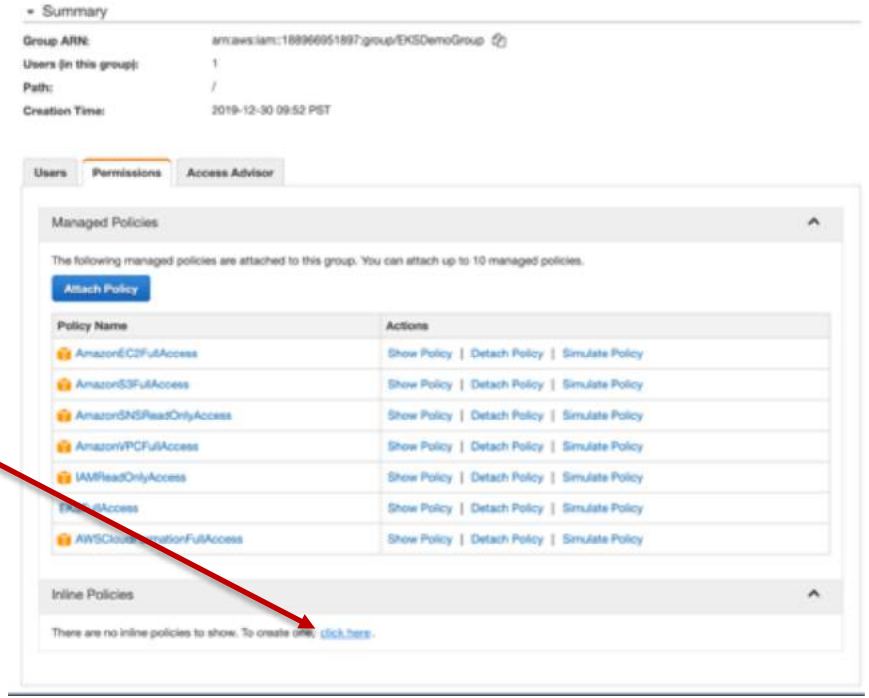


Attach Policies



Attach Inline Policy

- Allow EKS user to pass it's role to the new EKS Service Role



Attach Inline Policy

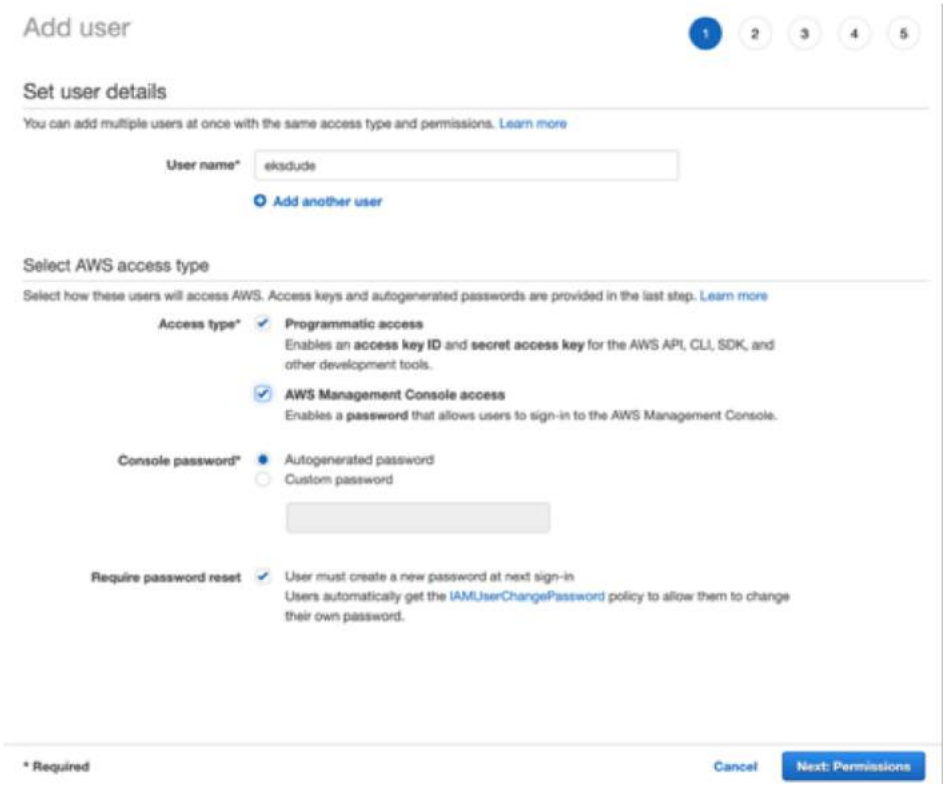
- Pass ability to let EKS clusters modify roles created by this user.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PolicyAtatementToAllowUserToPassAnyRole",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole",
        "iam:CreateServiceLinkedRole",
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:PutRolePolicy",
        "iam:DeleteRolePolicy"
      ],
      "Resource": "arn:aws:iam::188966951897:role/*"
    }
  ]
}
```

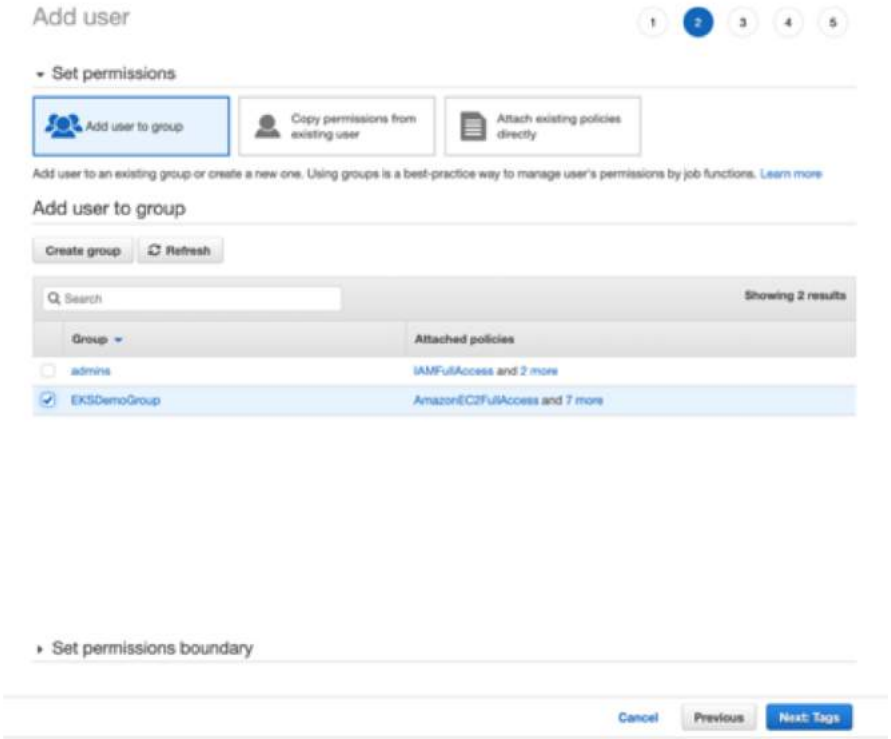
Policy for eksctl

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PolicyAtatementToAllowUserToPassAnyRole",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole",
                "iam:CreateServiceLinkedRole",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:AttachRolePolicy",
                "iam:DetachRolePolicy",
                "iam:PutRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:CreateInstanceProfile"
            ],
            "Resource": "arn:aws:iam::188966951897:role/*"
        }
    ]
}
```

Create a new user



Add user to EKS Group



Make note of Credentials

Add user

1 2 3 4 5

Success

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: https://188966951897.signin.aws.amazon.com/console

Download .csv

		User	Access key ID	Secret access key	Password	Email login instructions
▸	✓	eksdude	AKIASX72O3PMW7LIC4MH	CgRMi+a0QZQCL6ij3RncikjI sphpNulgR4KZdMih Hide	89(#L#L]VE30 Hide	Send email

Pearson

AWS CLI

- Download the AWS CLI tools
  - https://github.com/vallard/EKS-Training/blob/master/segment1/aws-creds.md

Automating with Terraform

- Infrastructure as Code
- Automate consistent workflows
- Build across multiple clouds
- Version control infrastructure


```
terraform init
terraform plan
terraform apply
```

Use the Outputs from Terraform

- User Password for Console Sign in:

```
terraform output password | base64 --decode  | gpg --decrypt | pbcopy
```

- AWS Credentials for CLI:

```
aws configure --profile=eksdude
```

- Access Key:

```
terraform output key | pbcopy
```

- Access Secret:

```
terraform output secret | base64 --decode  | gpg --decrypt | pbcopy
```

Alternate Credential storage

		User	Access key ID	Secret access key	Password	Email login instructions
▸	✓	eksdude	AKIASX72O3PMREGYZ2CQ	********* Show	********* Show	Send email

Add Credentials to ~/.aws/credentials file

```
20 [cr-demouser]
21 aws_access_key_id = AKIASX72O3PMREGYZ2CQ
22 aws_secret_access_key = 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
```

Add configuration to ~/.aws/config file

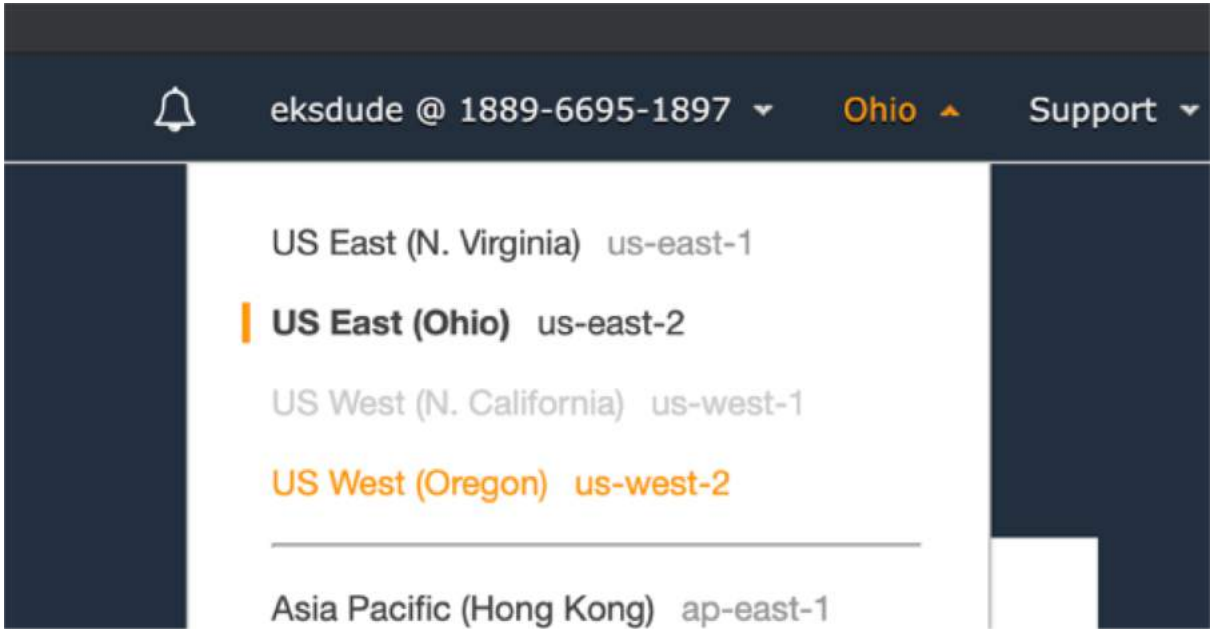
```
 9 [profile cr-demouser]
10 region=us-west-2
11 output=text
```

Set Command Line User

```
export AWS_PROFILE=eksdude
aws eks list-clusters
aws ec2 describe-instances
```

Sign in as new user

Check Proper Region



Kubernetes on AWS

Segment 3: Creating EKS Clusters

Big Picture

- Get `eksctl`
- Create an EKS Cluster with `eksctl`
- Create EKS Service Role
- Create EKS Control Plain manually and with Cloud Formation

Different Modes of Running Clusters

AWS Fargate - Linux

- kube-system and default pods run in Fargate
- Fargate Profile Created

Managed Nodes - Linux

- Linux ec2 instances created
- Cloud Formation Stack for EC2 instances created

Self Managed Nodes - Windows

- For running Windows Container Images
- Requires at least 1 Linux instance created to run system services like coreDNS

You can mix and match instance types in the same cluster

You can use eksctl to create clusters

```
eksctl create cluster \
--name aug05 \
--fargate \
--version 1.17 \
--region us-west-2 \
--nodegroup-name standard-
workers \
--node-type t3.medium \
--nodes 3 \
--nodes-min 1 \
--nodes-max 4 \
--managed
```

Hint: If the version is not supported, upgrade eksctl!

```
brew upgrade eksctl
```

Use eksctl to Create the Cluster

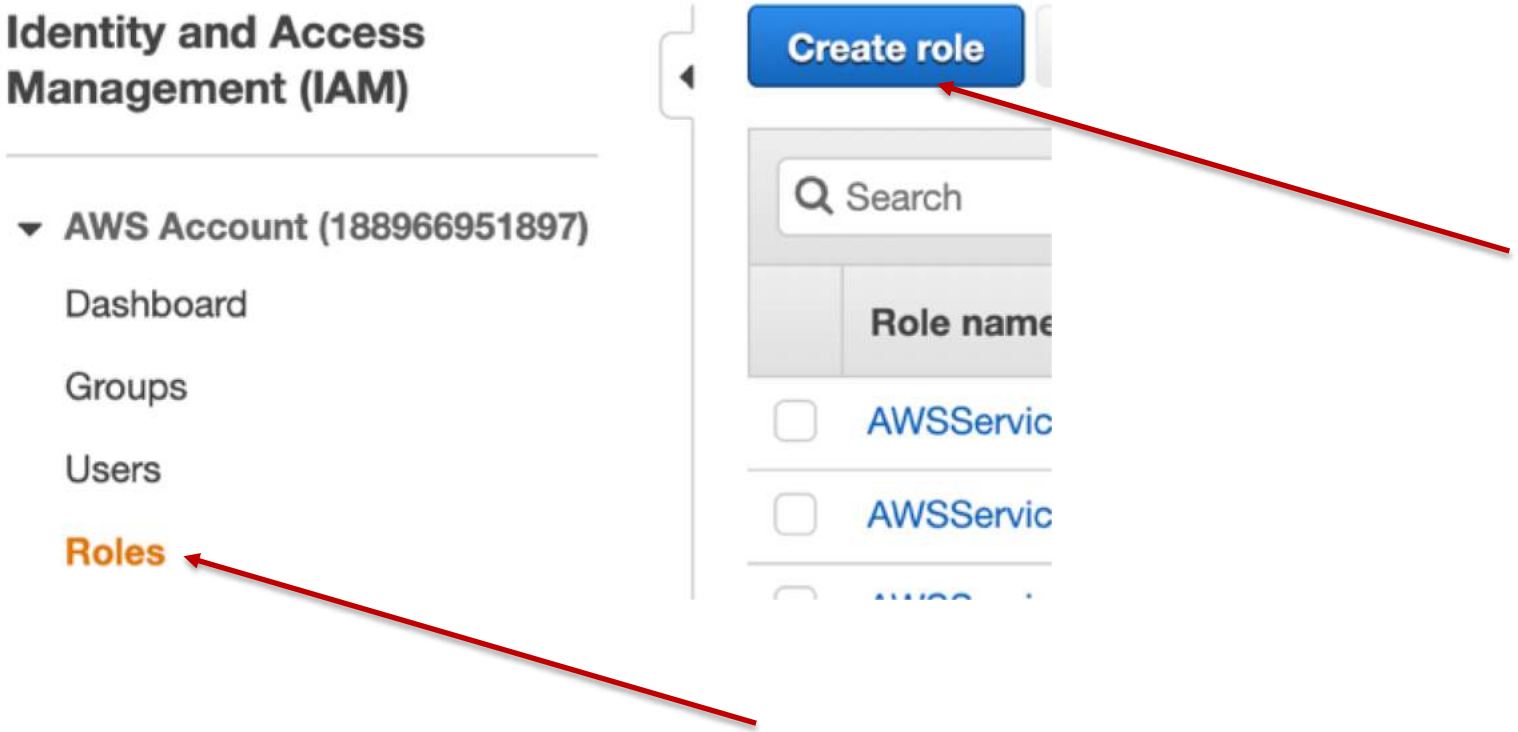
```
vallards-MacBook-Pro:segment03-install vallard$ ./createEKSctlCluster.sh
+ eksctl create cluster --name aug05 --fargate --version 1.17 --region us-west-2 --nodegroup-name standard
-workers --node-type t3.medium --nodes 3 --nodes-min 1 --nodes-max 4 --managed
[ℹ]  eksctl version 0.25.0
[ℹ]  using region us-west-2
[ℹ]  setting availability zones to [us-west-2b us-west-2c us-west-2d]
[ℹ]  subnets for us-west-2b - public:192.168.0.0/19 private:192.168.96.0/19
[ℹ]  subnets for us-west-2c - public:192.168.32.0/19 private:192.168.128.0/19
[ℹ]  subnets for us-west-2d - public:192.168.64.0/19 private:192.168.160.0/19
[ℹ]  using Kubernetes version 1.17
[ℹ]  creating EKS cluster "aug05" in "us-west-2" region with Fargate profile and managed nodes
[ℹ]  will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
[ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --regi
on=us-west-2 --cluster=aug05'
[ℹ]  CloudWatch logging will not be enabled for cluster "aug05" in "us-west-2"
[ℹ]  you can enable it with 'eksctl utils update-cluster-logging --region=us-west-2 --cluster=aug05'
[ℹ]  Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for clust
er "aug05" in "us-west-2"
[ℹ]  2 sequential tasks: { create cluster control plane "aug05", 2 sequential sub-tasks: { create fargate
profiles, create managed nodegroup "standard-workers" } }
[ℹ]  building cluster stack "eksctl-aug05-cluster"
[ℹ]  deploying stack "eksctl-aug05-cluster"
```

```
real    21m32.641s
user    0m0.801s
sys     0m1.179s
```

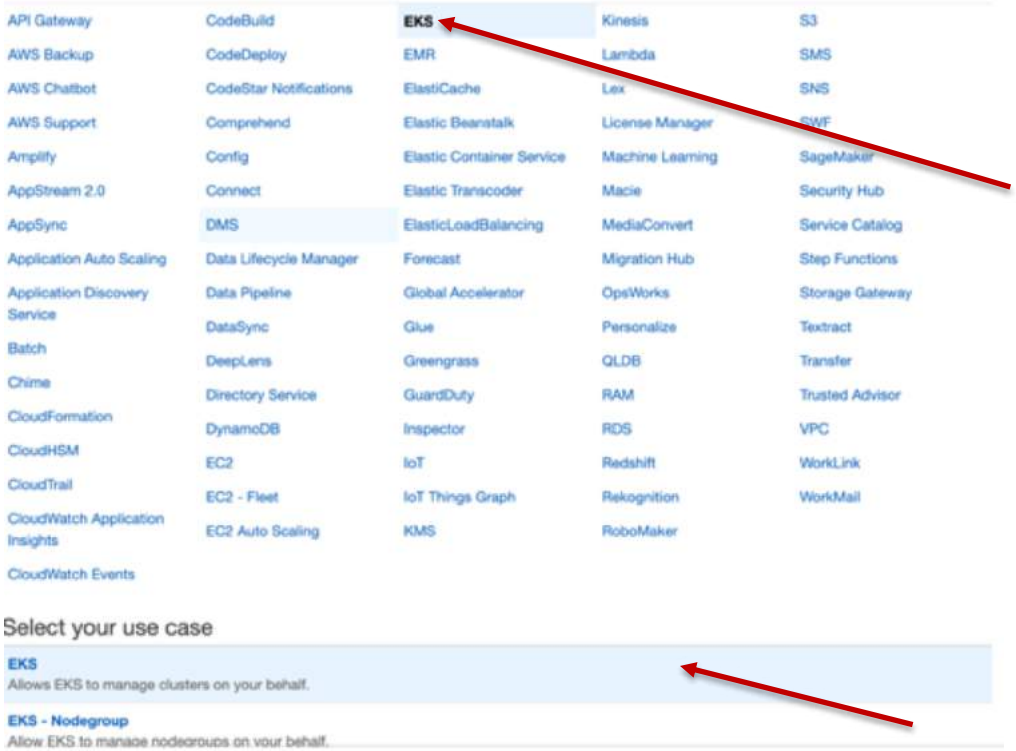
Create cluster manually

- Add EKS Service Role
- CloudFormation to create VPC and entities
- Create EKS Control Plane

Create EKS Service Role

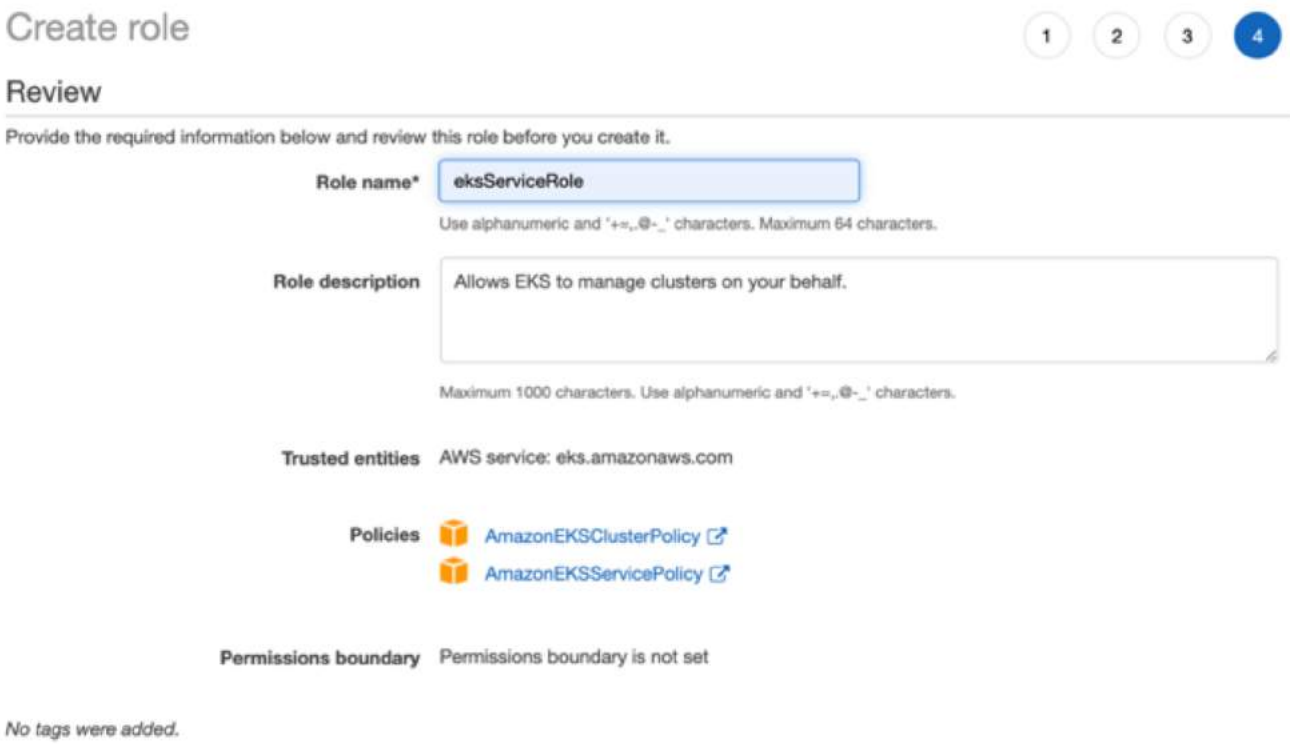


Create EKS Service Role (cont.)



Create EKS Service Role (cont.)

Create EKS Service Role (cont.)



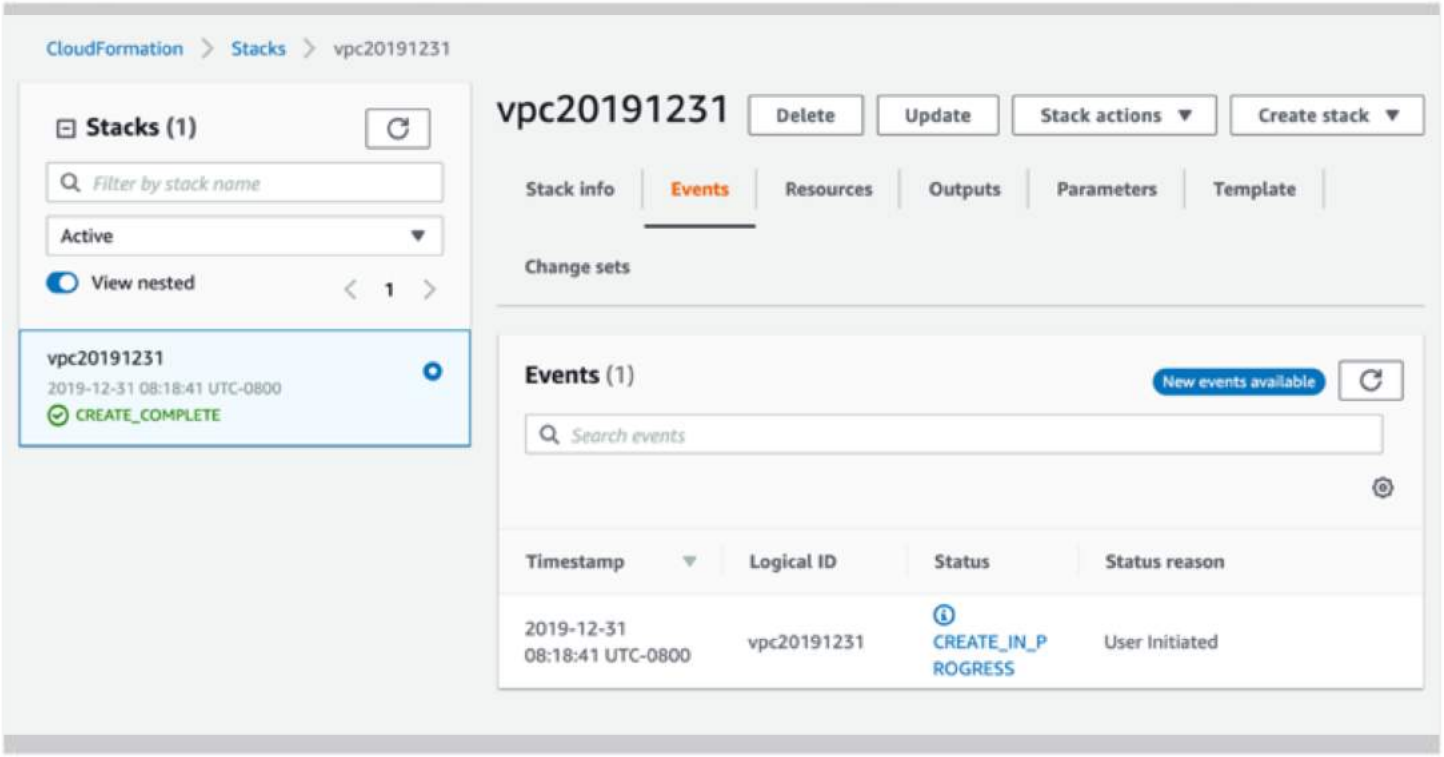
But wait! We already did this in Terraform

```
resource "aws_iam_role" "EKSServiceRole" {
  name = "EKSServiceRole"
  assume_role_policy = <<-EOF
    {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Action": "sts:AssumeRole",
          "Principal": {
            "Service": "eks.amazonaws.com"
          },
          "Effect": "Allow",
          "Sid": ""
        }
      ]
    }
EOF
}

resource "aws_iam_role_policy_attachment" "EKSServiceRoleAttachmentsCluster" {
  role = aws_iam_role.EKSServiceRole.name
  policy_arn  = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
}

resource "aws_iam_role_policy_attachment" "EKSServiceRoleAttachmentsService" {
  role = aws_iam_role.EKSServiceRole.name
  policy_arn  = "arn:aws:iam::aws:policy/AmazonEKSServicePolicy"
}
```

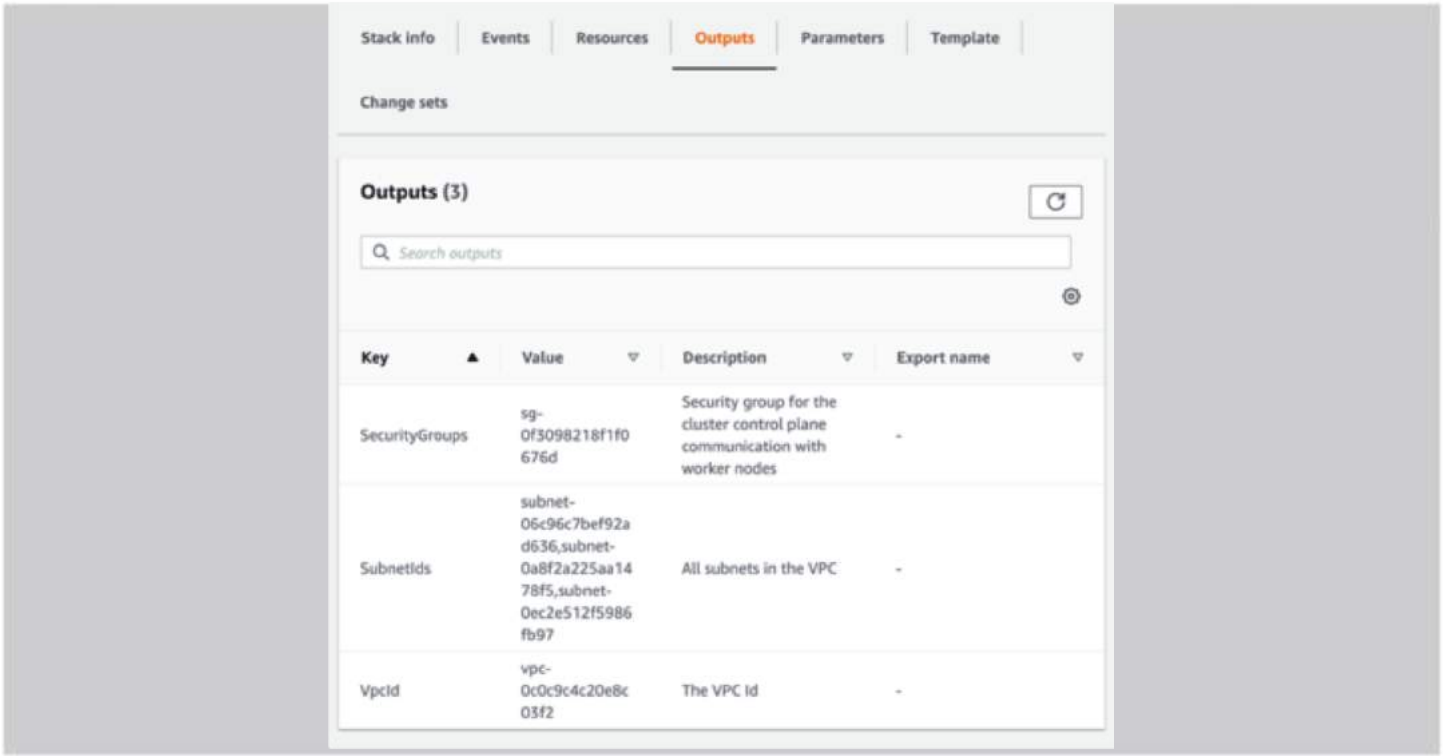
CloudFormation to create Network stack

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Amazon EKS Sample VPC'

Parameters:

  VpcBlock:
    Type: String
    Default: 192.168.0.0/16
    Description: The CIDR range for the VPC. This should be a valid private (RFC 1918) CIDR range.

  Subnet01Block:
    Type: String
    Default: 192.168.64.0/18
    Description: CidrBlock for subnet 01 within the VPC

  Subnet02Block:
    Type: String
    Default: 192.168.128.0/18
    Description: CidrBlock for subnet 02 within the VPC

  Subnet03Block:
    Type: String
    Default: 192.168.192.0/18
    Description: CidrBlock for subnet 03 within the VPC. This is used only if the region has more than 2 AZs.

```

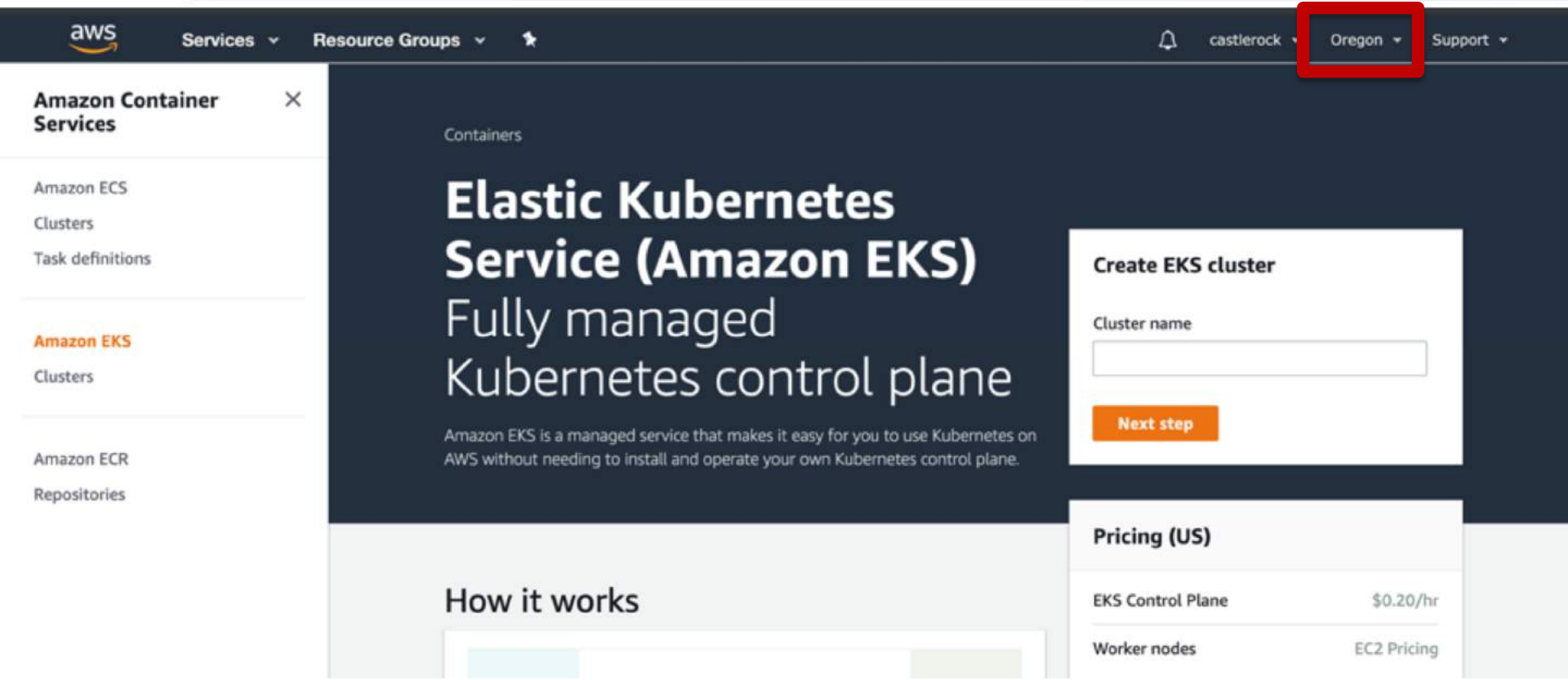
Create VPC with Cloud Formation

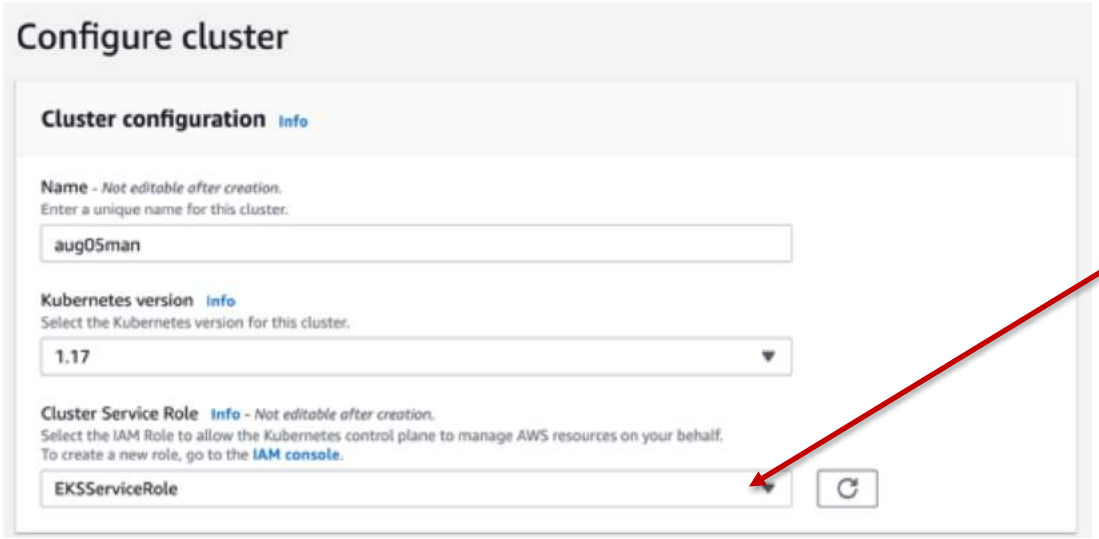


Examine Components of new VPC



Create EKS Control Plane

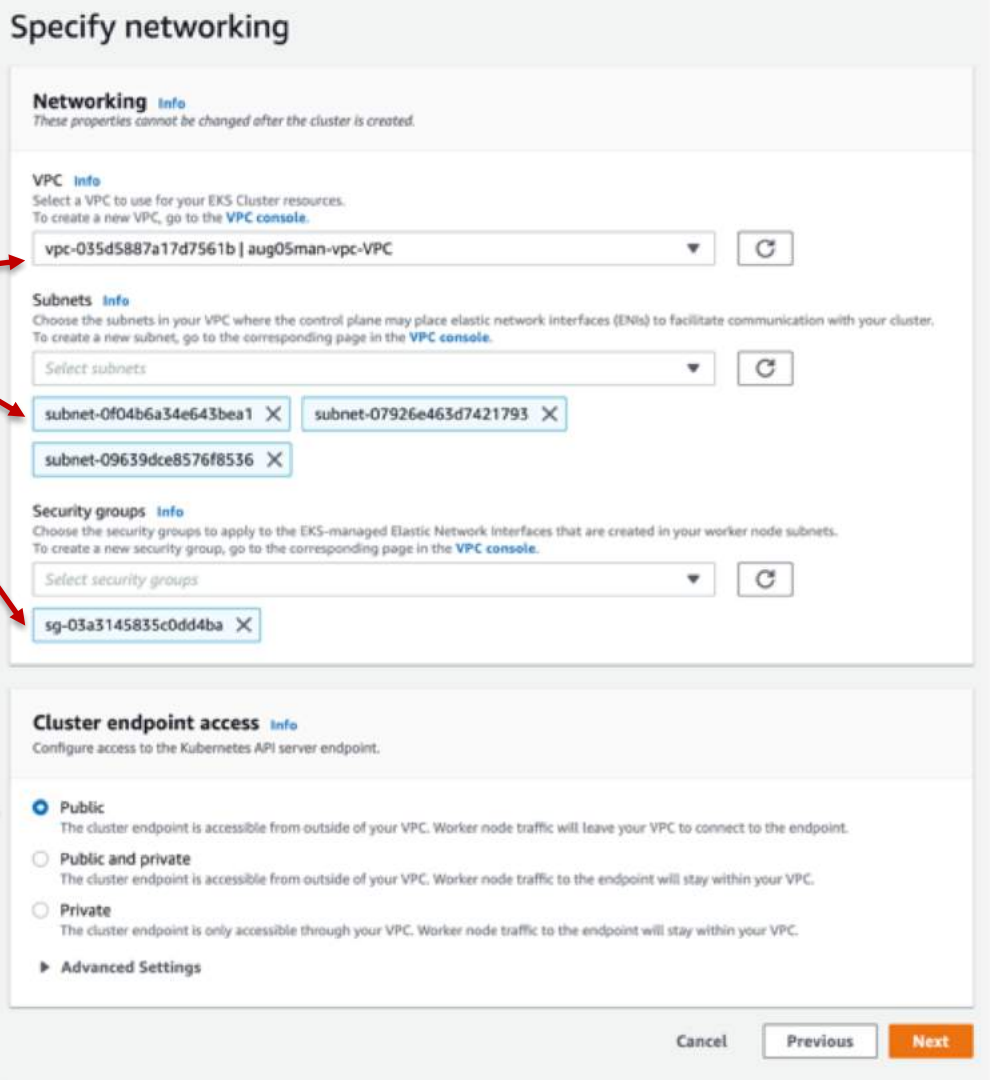




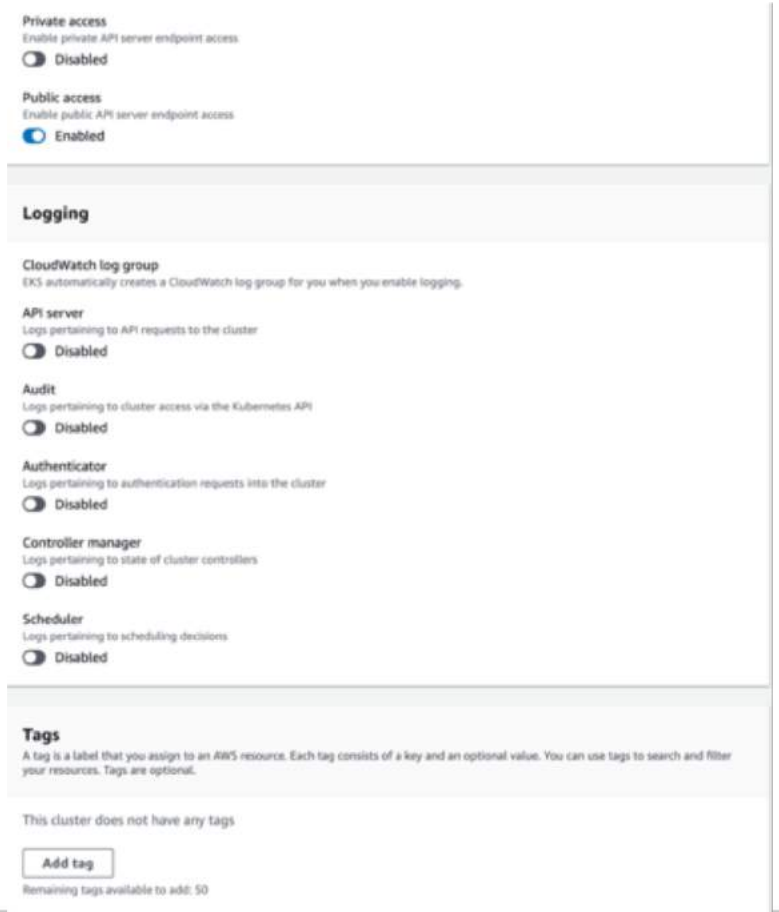
Created by Terraform

Use values created by Cloud Formation

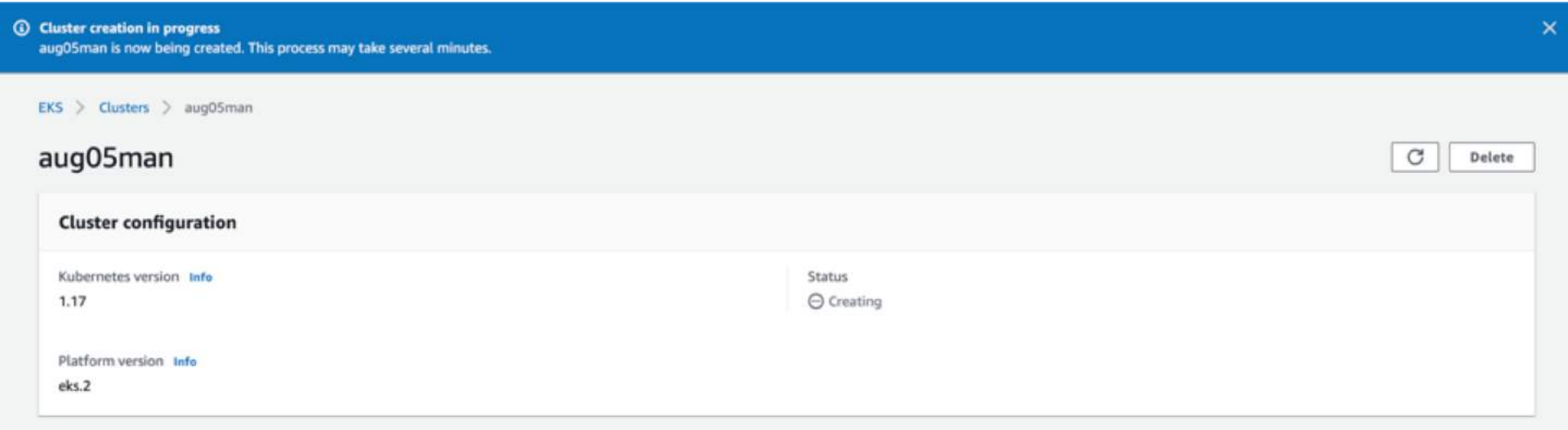
Chose endpoint access



Additional Settings and Options



Ensure cluster is being created



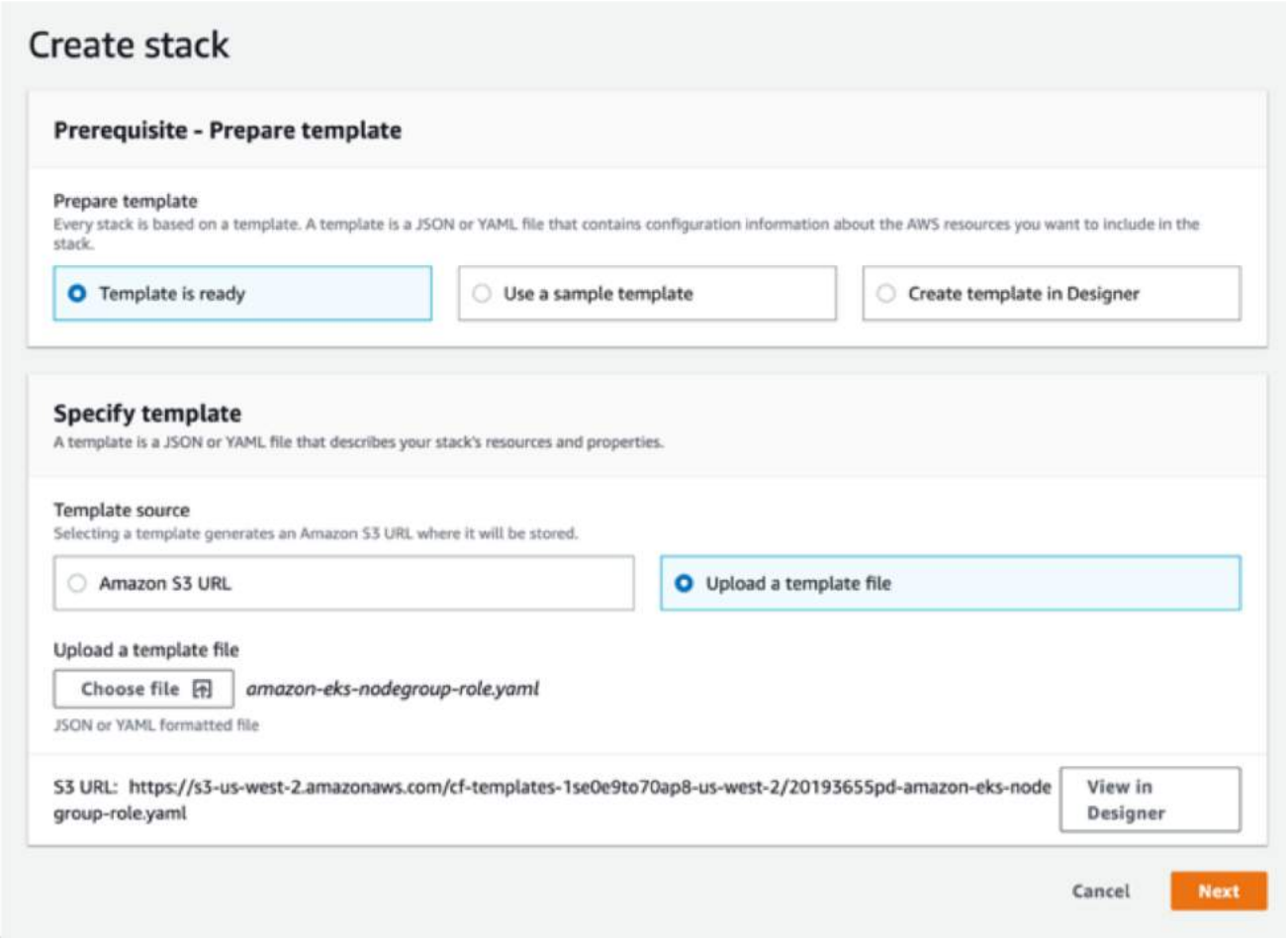
Break Time!

Segment 4: Verifying EKS

Big Picture

- Create Worker Role
- Get kubectl
- Test EKS cluster

Create AWS Role



Specify stack details

Stack name

Stack name

dec3-nodegroup-instance-role

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

Parameters

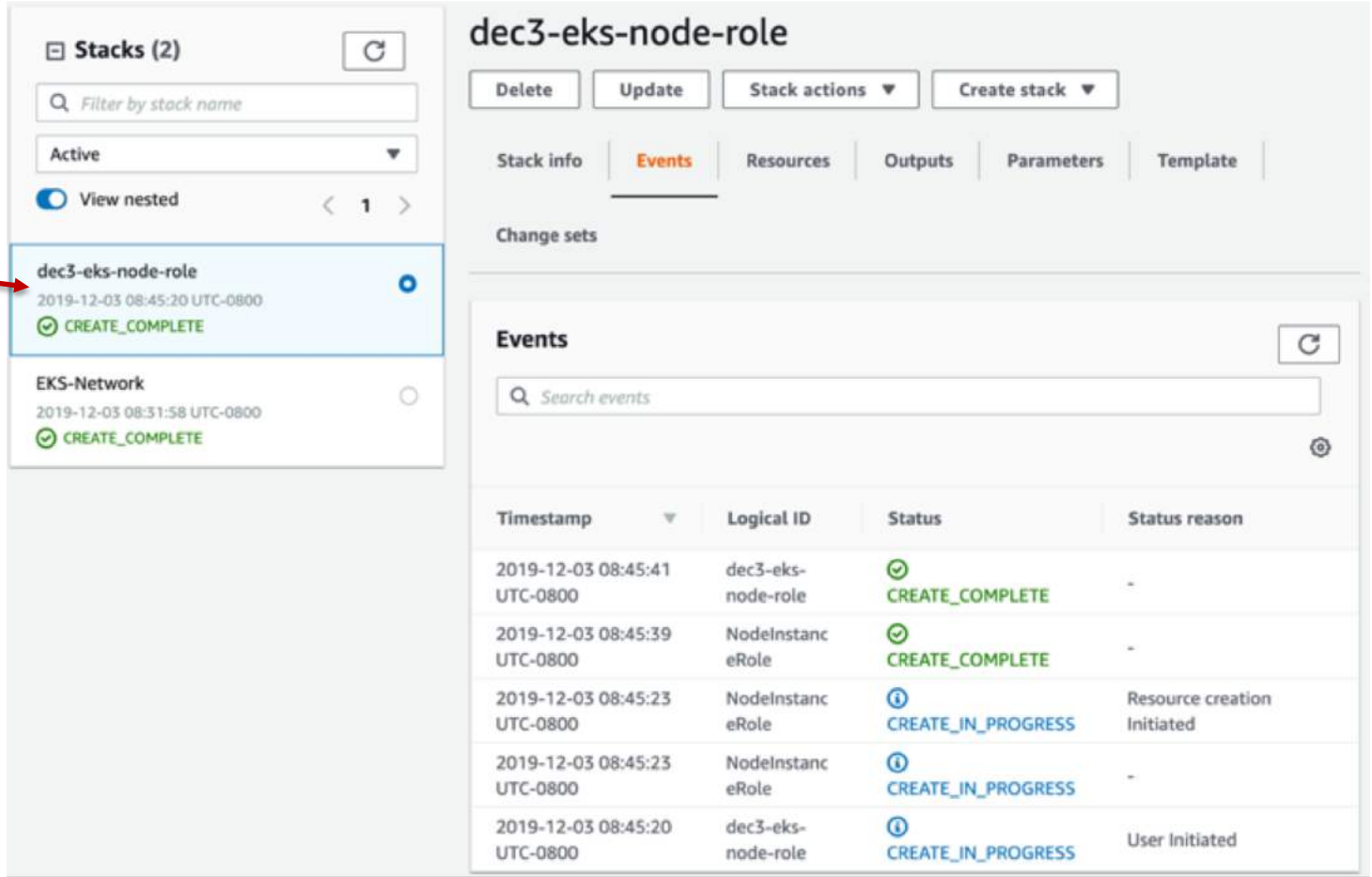
Parameters are defined in your template and allow you to input custom values when you create or update a stack.

No parameters

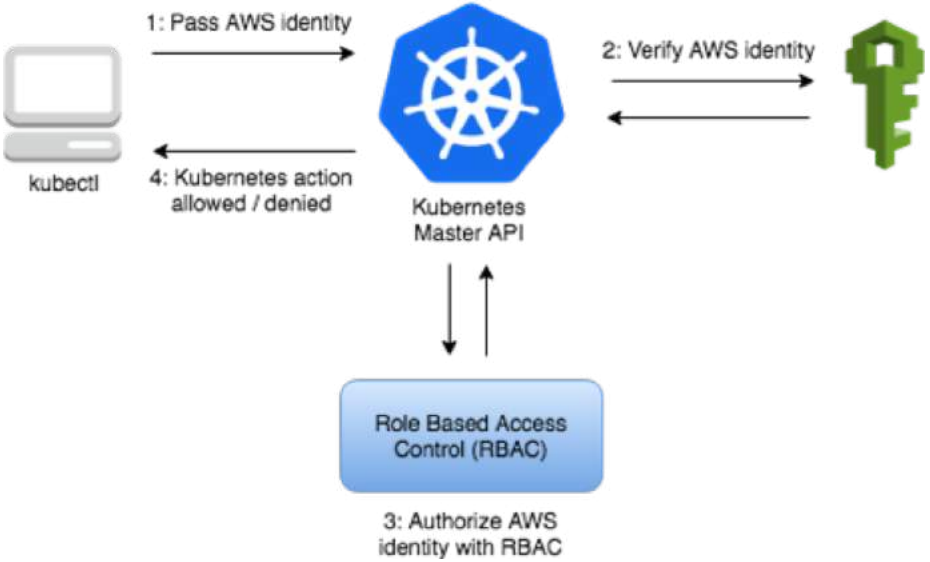
There are no parameters defined in your template

Cancel Previous Next

Accept all defaults



Authentication with EKS



Source: https://docs.aws.amazon.com/eks/latest/userguide/managing-auth.html

Connect to cluster

```
export AWS_PROFILE=eksdude
aws eks list-clusters
aws eks update-kubeconfig –name <cluster-name>
kubectl config get-contexts
kubectl config use-context <new context>
kubectl get svc
```

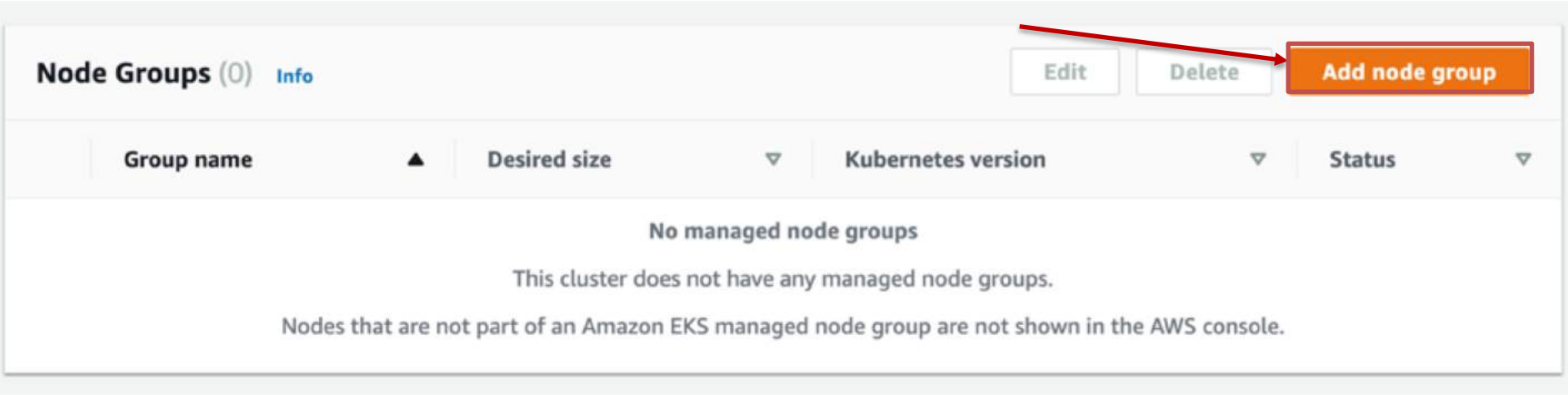
```
vallards-MacBook-Pro:segment2 vallard$ aws eks update-kubeconfig --name dec3
Added new context arn:aws:eks:us-west-2:188966951897:cluster/dec3 to /Users/vallard/.kube/config
vallards-MacBook-Pro:segment2 vallard$ kubectl config get-contexts
CURRENT   NAME                                                  CLUSTER
          AUTHINFO                                              NAMESPACE
          arn:aws:eks:us-east-1:254056899943:cluster/matomo     arn:aws:eks:us-east-1:254056899943:cluste
r/matomo    arn:aws:eks:us-east-1:254056899943:cluster/matomo
*         arn:aws:eks:us-west-2:188966951897:cluster/dec3       arn:aws:eks:us-west-2:188966951897:cluste
r/dec3      arn:aws:eks:us-west-2:188966951897:cluster/dec3
vallards-MacBook-Pro:segment2 vallard$ kubectl config use-context arn:aws:eks:us-west-2:188966951897:cl
uster/dec3
Switched to context "arn:aws:eks:us-west-2:188966951897:cluster/dec3".
```

Interacting

- `eksctl get clusters`
- `aws eks list-clusters`

```
vallards-MacBook-Pro:Code vallard$ kubectl get nodes
No resources found in default namespace.
```

Add Worker Nodes To Console Cluster



Role created from CloudFormation

Configure node group

A node group is a group of EC2 instances that supply compute capacity to your Amazon EKS cluster. You can add multiple node groups to your cluster. Info

Group configuration

These properties cannot be changed after the node group is created.

Name

Assign a unique name for this node group

wg01

Node IAM Role Name Info

Select the IAM Role that will be used by the nodes.

dec31-eksrole-NodeInstanceRole-VZVZ8FBO93GW

Subnets Info

Specify the subnets in your VPC where your nodes will run

Add subnets

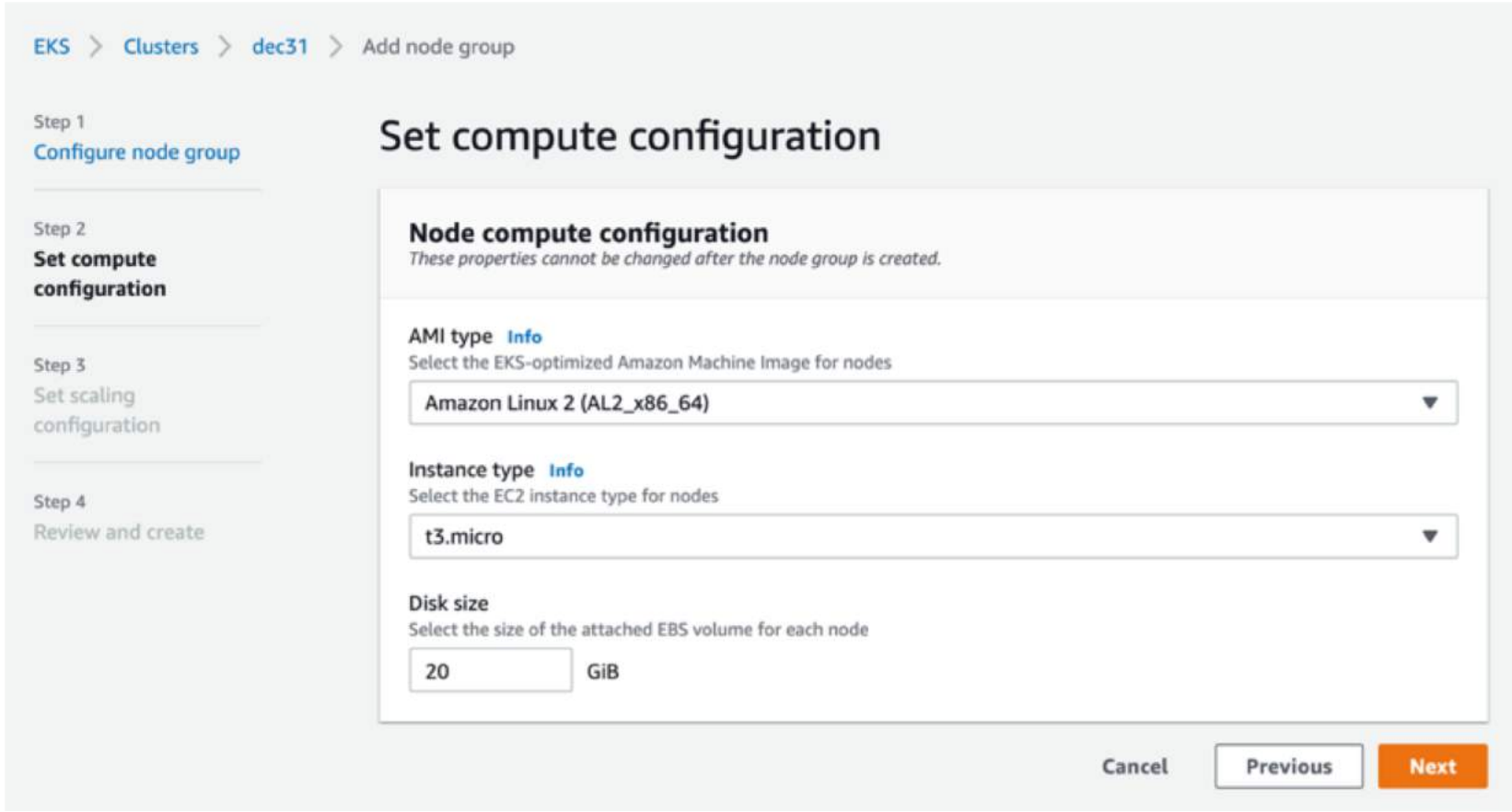
subnet-0c23d7de4a0cae3a9 vpc20191231-Subnet01
us-west-2a 192.168.64.0/18

subnet-0ffed858554051f8a vpc20191231-Subnet03
us-west-2c 192.168.192.0/18

subnet-0742b5586cd70dd4a vpc20191231-Subnet02
us-west-2b 192.168.128.0/18

Allow remote access to nodes Info

Without remote access enabled you will not be able to directly connect to nodes after they are created.


EKS > Clusters > dec31 > Add node group
Step 1
Configure node group
Step 2
Set compute configuration
Step 3
Set scaling configuration
Step 4
Review and create
Set compute configuration
Node compute configuration
These properties cannot be changed after the node group is created.
AMI type Info
Select the EKS-optimized Amazon Machine Image for nodes
Amazon Linux 2 (AL2_x86_64)
Instance type Info
Select the EC2 instance type for nodes
t3.micro
Disk size
Select the size of the attached EBS volume for each node
20
GiB
Cancel
Previous
Next

Verify Nodes

- `kubectl get nodes`
- `kubectl get nodes -o wide`

```
vallards-MacBook-Pro:segment04-verify vallard$ kubectl get nodes
NAME                                           STATUS   ROLES    AGE   VERSION
ip-192-168-12-72.us-west-2.compute.internal    Ready    <none>   54m   v1.14.7-eks-1861c5
ip-192-168-56-142.us-west-2.compute.internal   Ready    <none>   55m   v1.14.7-eks-1861c5
ip-192-168-67-254.us-west-2.compute.internal   Ready    <none>   54m   v1.14.7-eks-1861c5
```

Namespaces

- Logical separations for applications
- Idea is to group different Kubernetes resources such as Pods, Services, Ingress Rules, and Deployments in logical boxes.

```
$ kubectl get ns
```

Pods

```
$ kubectl get pods
$ kubectl get po –all-namespaces
```

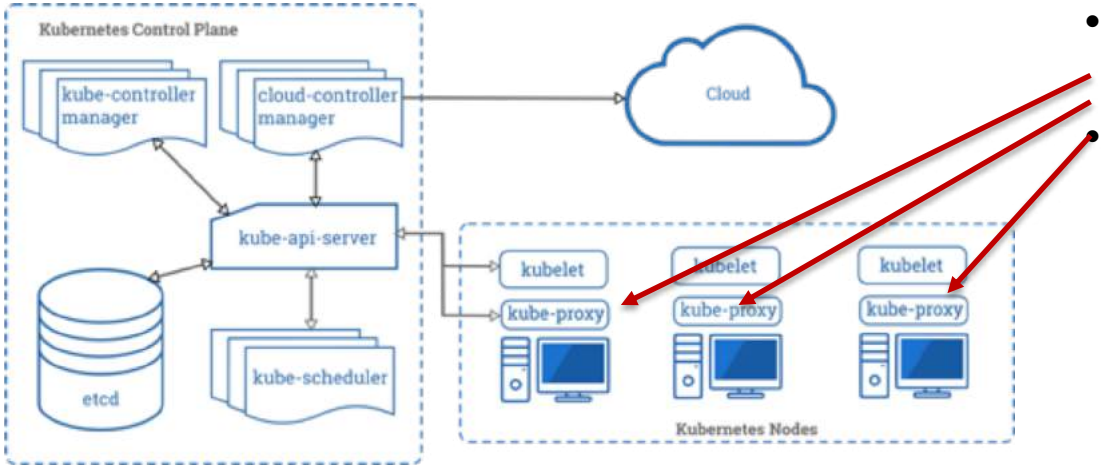
```
NAMESPACE     NAME                       READY   STATUS    RESTARTS   AGE
kube-system   aws-node-6jjnf             1/1     Running   0          52m
kube-system   aws-node-7cdw6             1/1     Running   0          52m
kube-system   aws-node-g6tkd             1/1     Running   0          52m
kube-system   coredns-84549585c-jj6q5    1/1     Running   0          57m
kube-system   coredns-84549585c-sx2jf    1/1     Running   0          57m
kube-system   kube-proxy-76wgk           1/1     Running   0          52m
kube-system   kube-proxy-f7w2p           1/1     Running   0          52m
kube-system   kube-proxy-ppl9p           1/1     Running   0          52m
```

Aws-Node Daemon Set

```
$ kubectl describe ds -n kube-system aws-node
```

```
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:           k8s-app=aws-node
  Service Account:  aws-node
  Containers:
   aws-node:
    Image:      602401143452.dkr.ecr.us-west-2.amazonaws.com/amazon-k8s-cni:v1.5.5
    Port:       61678/TCP
    Host Port:  61678/TCP
    Requests:
      cpu:  10m
    Environment:
      AWS_VPC_K8S_CNI_LOGLEVEL:  DEBUG
      MY_NODE_NAME:               (v1:spec.nodeName)
```

Kube Proxy



- Network Proxy that runs on each node
- Maintains network rules on nodes that allow network communication to Pods

Source: https://kubernetes.io/docs/concepts/overview/components/

What services run by default?

- What is running by default?

Kubernetes service – API service so internal services can communicate with Kubernetes API service

```
vallards-MacBook-Pro:segment03-install vallard$ kubectl get svc --all-namespaces
NAMESPACE     NAME         TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)         AGE
default       kubernetes   ClusterIP   10.100.0.1    <none>        443/TCP         26m
kube-system   kube-dns     ClusterIP   10.100.0.10   <none>        53/UDP,53/TCP   26m
```

DNS service so pods can find other services in the cluster without having to hard code IP addresses

Test kube-dns

```
$ kubectl create -f segment04-verify/dnstest/
$ kubectl exec -it bb8-… -- /bin/sh
$ wget -q -S -O - ngx
$ wget -q -S -O - ngx.default
$ wget -q -S -O - ngx.default.svc
$ wget -q -S -O - ngx.default.svc.cluster.local
```

Segment 5:
Running Applications

Big Picture

- LoadBalancers
- Ingress Controller
- R53 Routing
- TLS
- Persistence

Expose a Service

- Currently our service is inward facing. (No access from outside)
- We can access it remotely by changing the Service interface from `ClusterIP` to `LoadBalancer`
- This automatically creates a load balancer for us
- We can do this for any service we want to expose

```
vallards-MacBook-Pro:test01 vallard$ kubectl get svc
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP                                                              PORT(S)        AGE
kubernetes   ClusterIP      10.100.0.1      <none>                                                                   443/TCP        91m
ngx          LoadBalancer   10.100.49.193   aada23c992d8611ea87060a79e3e7cb4-855121620.us-west-2.elb.amazonaws.com   80:30107/TCP   44m
```

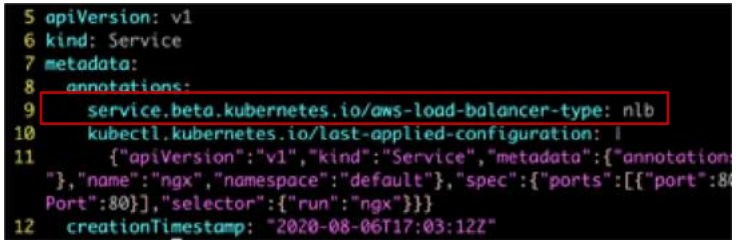
Change to Load balancer

- `kubectl edit svc ngx`
- Change from:
  - `type: ClusterIP`
- To:
  - `type: LoadBalancer`

```
14   name: ngx
15   namespace: default
16   resourceVersion: "14827"
17   selfLink: /api/v1/namespaces/default/services/ngx
18   uid: 283c67cd-00a1-49d4-b410-7b975006c6b0
19 spec:
20   clusterIP: 10.100.45.250
21   ports:
22   - port: 80
23     protocol: TCP
24     targetPort: 80
25   selector:
26     run: ngx
27   sessionAffinity: None
28   type: LoadBalancer
29 status:
30   loadBalancer: {}
```

Load Balancing Types

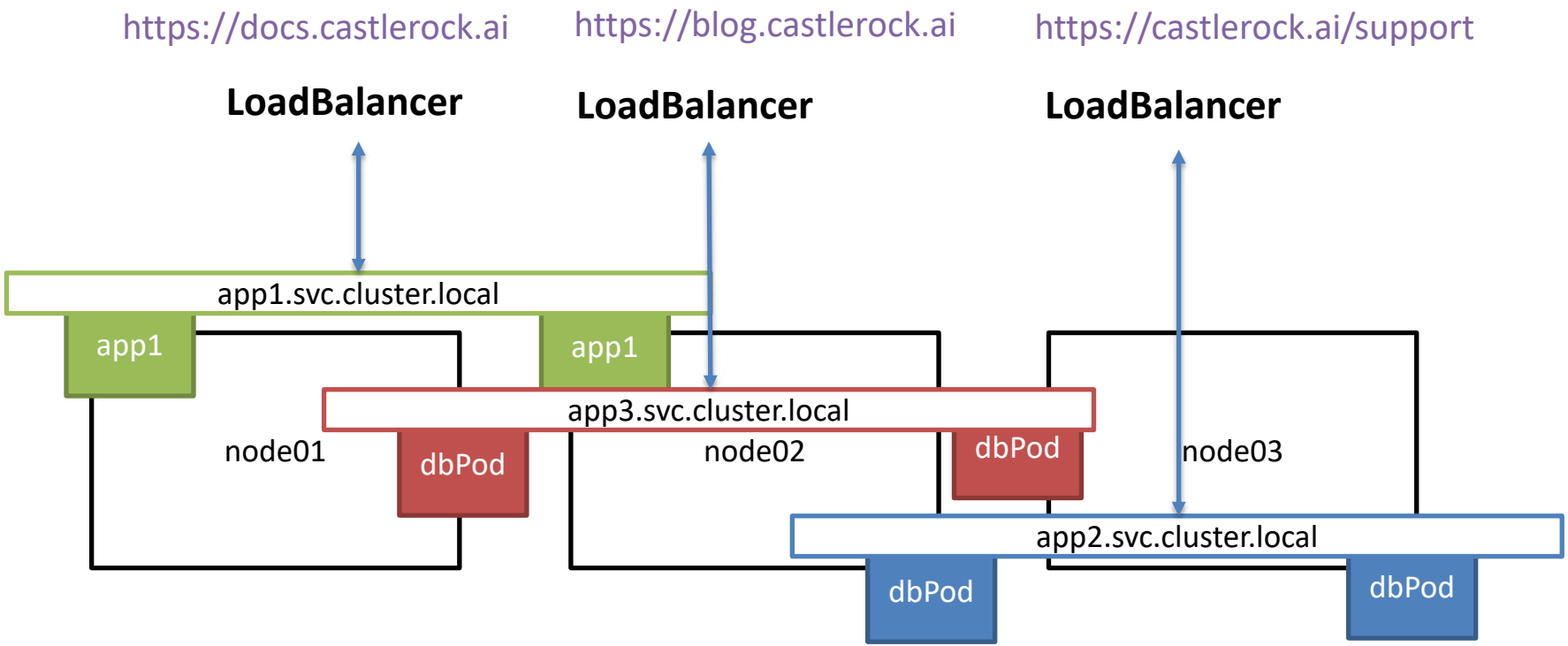
- Classic Load Balancer
  - Default
  - EC2 Classic Support
- Network Load Balancer
  - Newer improved! TCP, TLS, UDP
  - Static IPs: Elastic IP
- Application Load Balancer
  - HTTP/HTTPS protocol support



Multiplexing Load Balancers

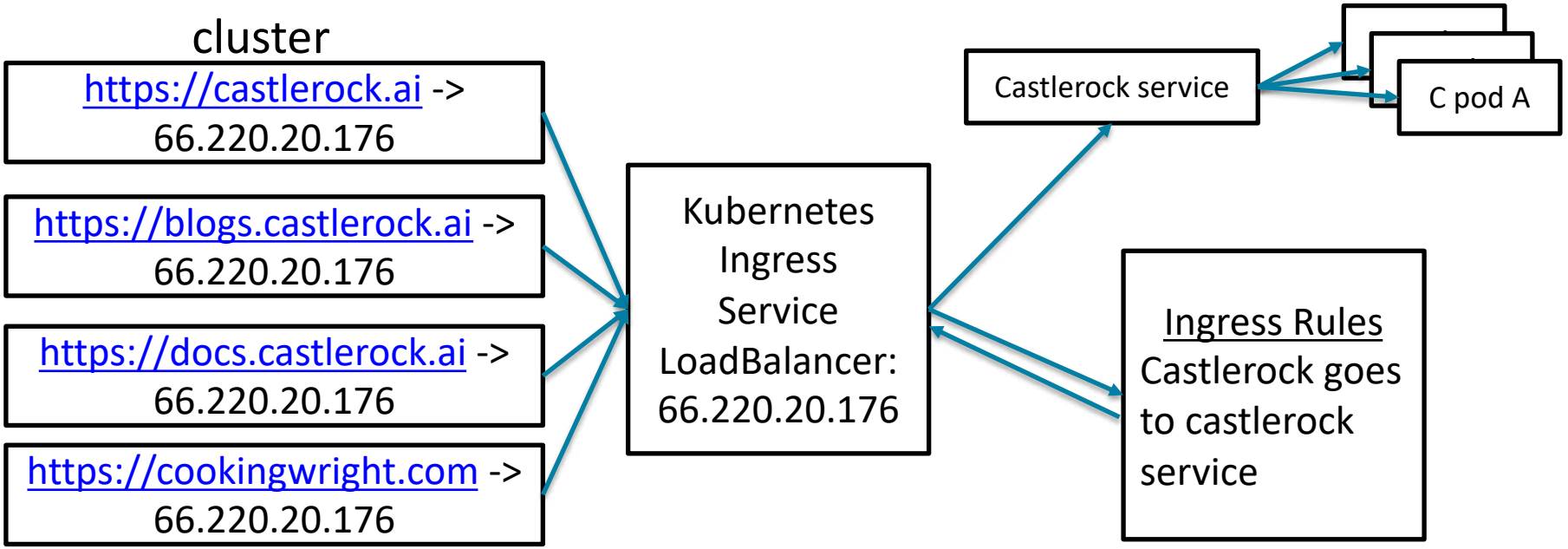
- Cost: $0.0225 / hour => $16/month for each exposed service
- Need one for each service, when traffic may not be that high
- Solution: Use an ingress controller

Too Many External Services?

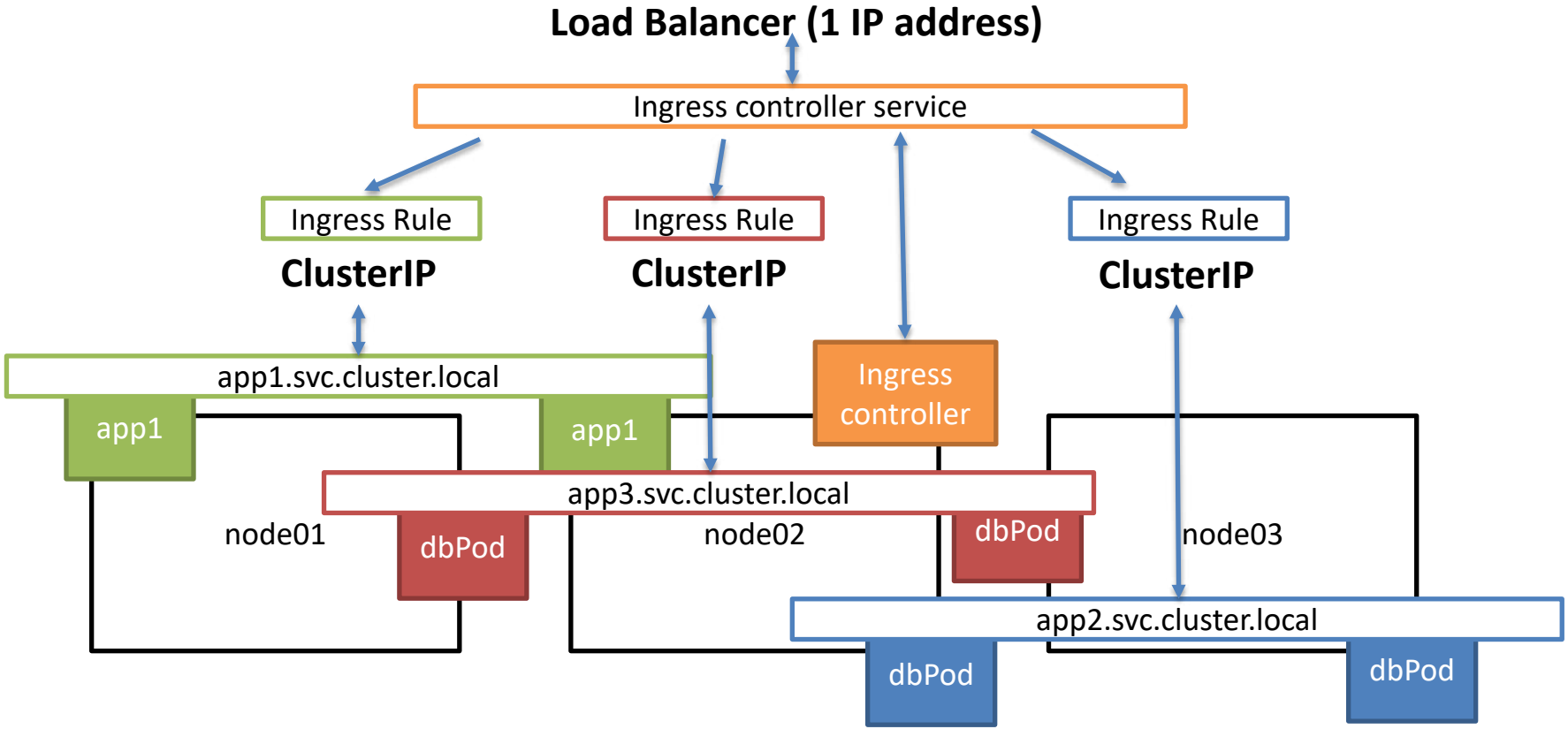


The Need for Ingress Controllers

- Only a fixed amount of external IP addresses
- Might have hundreds of different apps that require URLs in a cluster



Ingress Controller Architecture



Ingress Controller

- A normal Kubernetes Service
- Subscribes to Kubernetes for changes in:
  - Services
  - Ingress rules
- Requires Ingress Rules to route services
- Different Ingress Controllers are available:
  - NGINX
  - HA Proxy
  - Traefik
  - Others

Change ngx back to ClusterIP

- Remove nodePort
- Change type from LoadBalancer to ClusterIP

Ingress Controllers

- Deploy an nginx Load Ingress Controller

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.34.1/deploy/static/provider/aws/deploy.yaml

- Examine our new ingress controller service

```
kubectl get svc –n ingress-nginx
```

- Note the NLB address and open browser

What is in an Ingress Rule?

- Host name: e.g.: www.foo.example.com
- A list of paths /blog/2019-12-11

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: default-backend
spec:
  rules:
  - host: castlerock.ai
    http:
      - paths:
        - backend:
            serviceName: cr-homepage
            servicePort: 80
```

Match Host name ‘castlerock.ai’

Accept all paths

Configure Default Backend

- If traffic hits the ingress controller and no rules are specified then we have a default backend.
- Make your customized 404 page

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: default-backend
spec:
  backend:
    serviceName: default-backend
    servicePort: 80
```

No rules, so will catch all other requests.

Alternative would be to have a ‘page not found by nginx’

Ingress Rule options

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: default-backend
spec:
  rules:
  - host: castlerock.ai
    http:
      - path: /docs
        - backend:
            serviceName: cr-docs
            servicePort: 80
      - path: /blog
        - backend:
           serviceName: wordpress
            servicePort: 80
```

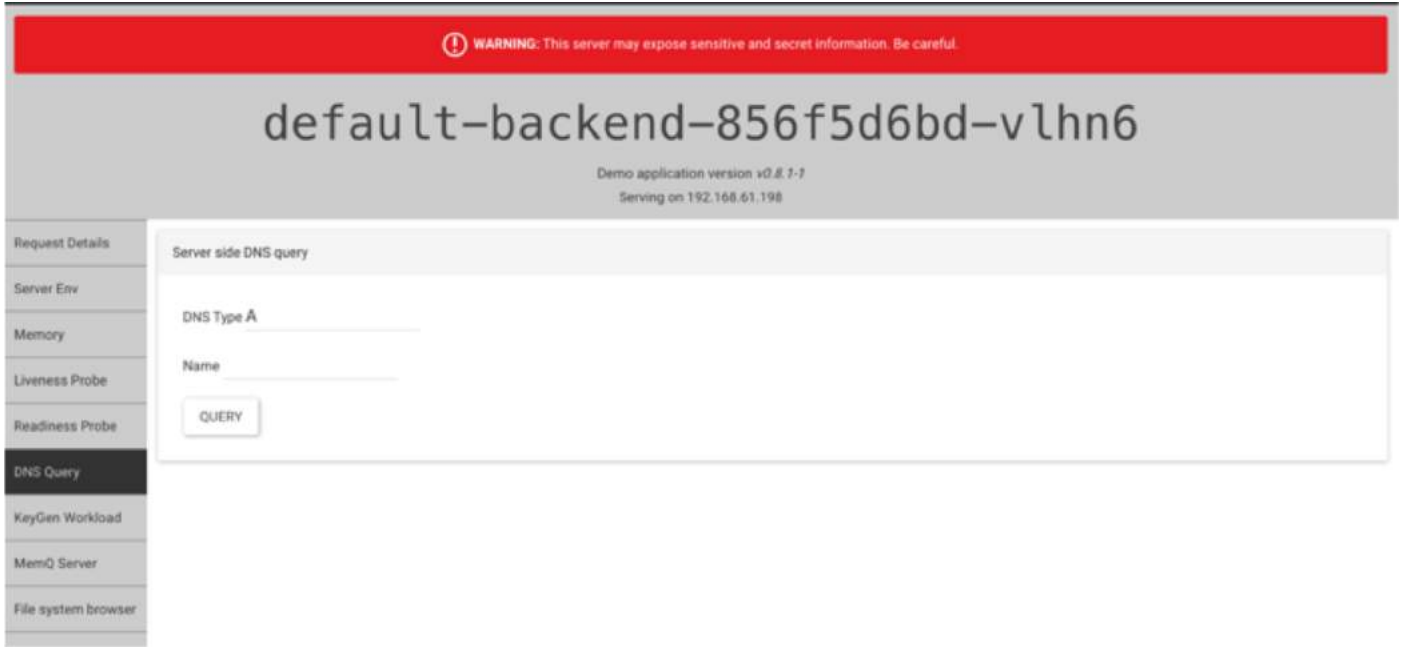
Match Host name ‘castlerock.ai’

Docs route to docs app.

Blog path routes to wordpress service

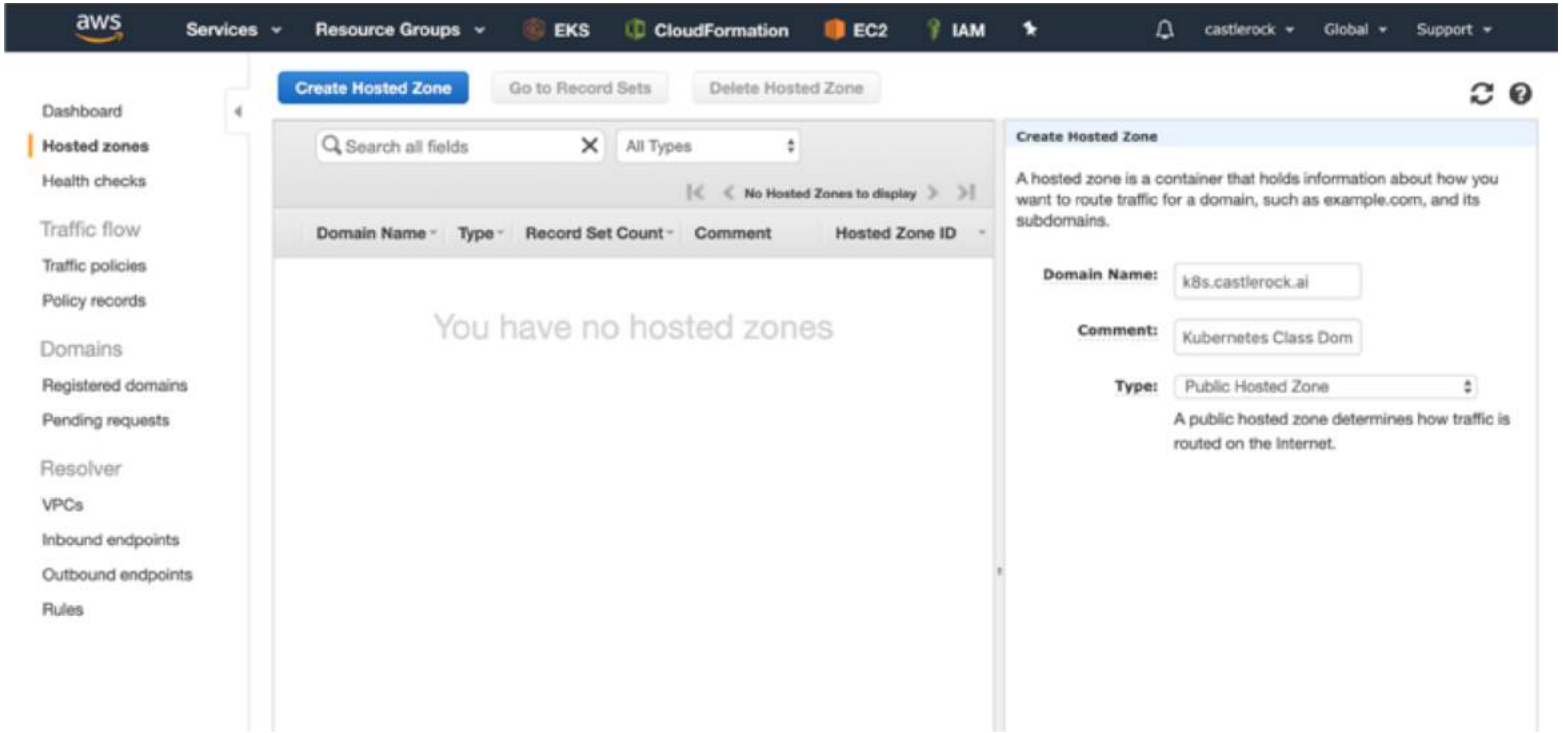
Create Default Backend

- `kubectl apply -f segement05-applications/default-backend.yaml`



Map to a Domain

- Create a new Route 53 Resource



Map to a Domain

- Create a new Route 53 Resource

Name	Type	Value	Evaluate Target Hea
k8s.castlerock.ai.	NS	ns-493.awsdns-61.com. ns-811.awsdns-37.net. ns-1344.awsdns-40.org. ns-1585.awsdns-06.co.uk.	-
k8s.castlerock.ai.	SOA	ns-493.awsdns-61.com. awsdns-hostmaster.amazor	-

Add NS Record to Domain Service

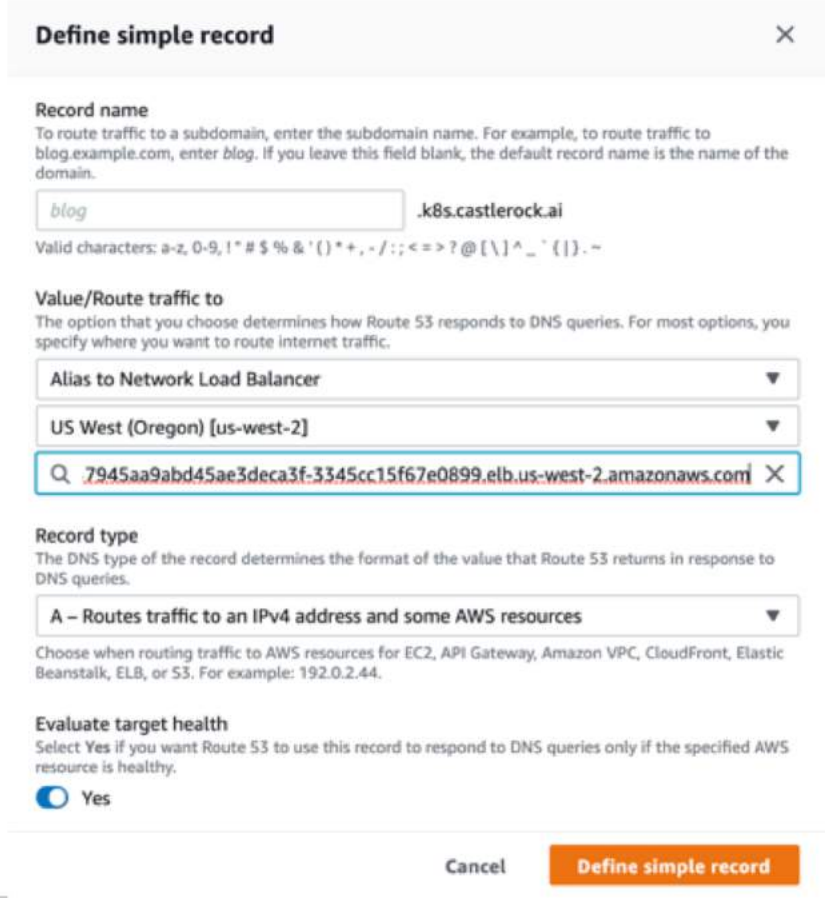
NS Record	k8s	ns-493.awsdns-61.com.	Automatic	
NS Record	k8s	ns-811.awsdns-37.net.	Automatic	
NS Record	k8s	ns-1344.awsdns-40.org.	Automatic	
NS Record	k8s	ns-1585.awsdns-06.co.uk.	Automatic	

ADD NEW RECORD

SHOW LESS

Pearson

Create Alias Record



Map our nginx service to ingress

- `kubectl apply -f ngx-ing.yaml`

Test Name Resolution

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

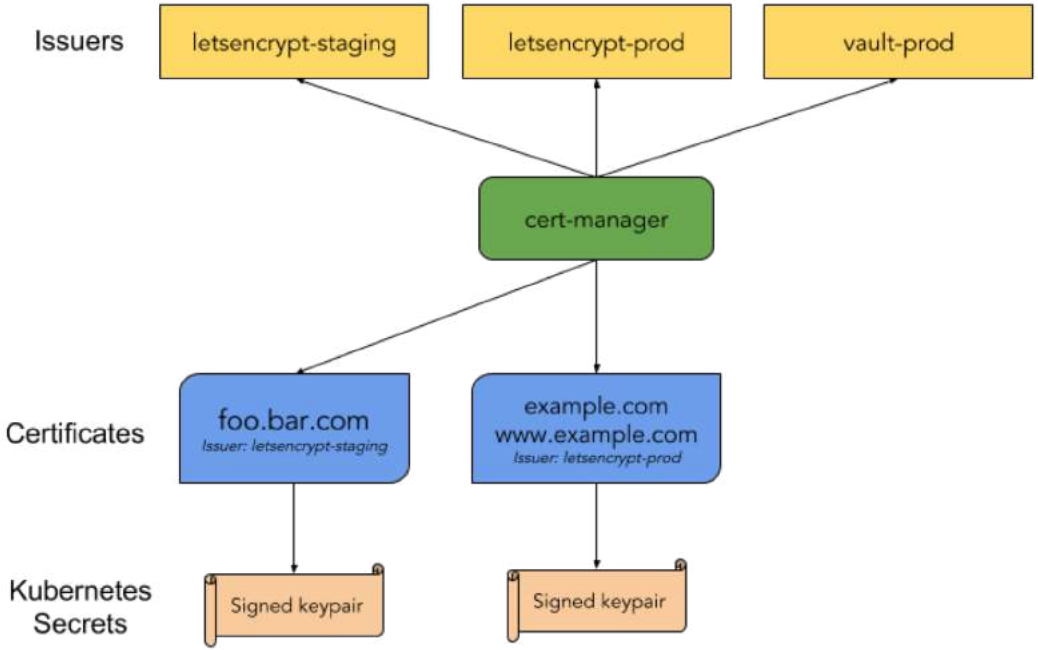
Let's Encrypt

- Free, automated, and open certificate authority (CA) from Internet Security Research Group (ISRG)
- Goal to create a more secure Internet
- Automated service

Let’s Encrypt with Kubernetes

- Cert-manager
  - Open source Kubernetes native certificate manager
  - Works with different issuers including Let’s Encrypt
  - Fast way to get free TLS certifications

Cert-manager Architecture



Source: https://github.com/jetstack/cert-manageraccessed 05/27/2019

Create Manifests

```
kubectl create -f cert-manager.yaml
```

```
vallards-MacBook-Pro:cert-manager vallard$ kubectl create -f cert-manager.yaml --validate=false
customresourcedefinition.apiextensions.k8s.io/certificaterequests.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-manager.io created
serviceaccount/cert-manager-cainjector created
serviceaccount/cert-manager created
serviceaccount/cert-manager-webhook created
clusterrole.rbac.authorization.k8s.io/cert-manager-cainjector created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-cainjector created
role.rbac.authorization.k8s.io/cert-manager-cainjector:leaderelection created
rolebinding.rbac.authorization.k8s.io/cert-manager-cainjector:leaderelection created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-webhook:auth-delegator created
rolebinding.rbac.authorization.k8s.io/cert-manager-webhook:webhook-authentication-reader created
clusterrole.rbac.authorization.k8s.io/cert-manager-webhook:webhook-requester created
role.rbac.authorization.k8s.io/cert-manager:leaderelection created
rolebinding.rbac.authorization.k8s.io/cert-manager:leaderelection created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-issuers created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-clusterissuers created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-certificates created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-orders created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-challenges created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-ingress-shim created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-issuers created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-clusterissuers created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-certificates created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-orders created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-challenges created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-ingress-shim created
clusterrole.rbac.authorization.k8s.io/cert-manager-view created
clusterrole.rbac.authorization.k8s.io/cert-manager-edit created
service/cert-manager created
service/cert-manager-webhook created
deployment.apps/cert-manager-cainjector created
deployment.apps/cert-manager created
```

Validate

```
kubectl get pods --namespace cert-manager
```

```
vallards-MacBook-Pro:cert-manager vallard$ kubectl get pods --namespace cert-manager
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-598bfb5ddb-jn4k4              1/1     Running   0          2m17s
cert-manager-cainjector-594fd9cc45-rr8gd   1/1     Running   0          2m17s
cert-manager-webhook-785ff8fc78-xtsbz      1/1     Running   0          2m17s
```

Create Cluster Issuers

```
kubectl apply -f staging-issuer.yaml
kubectl describe clusterissuer letsencrypt-staging
```

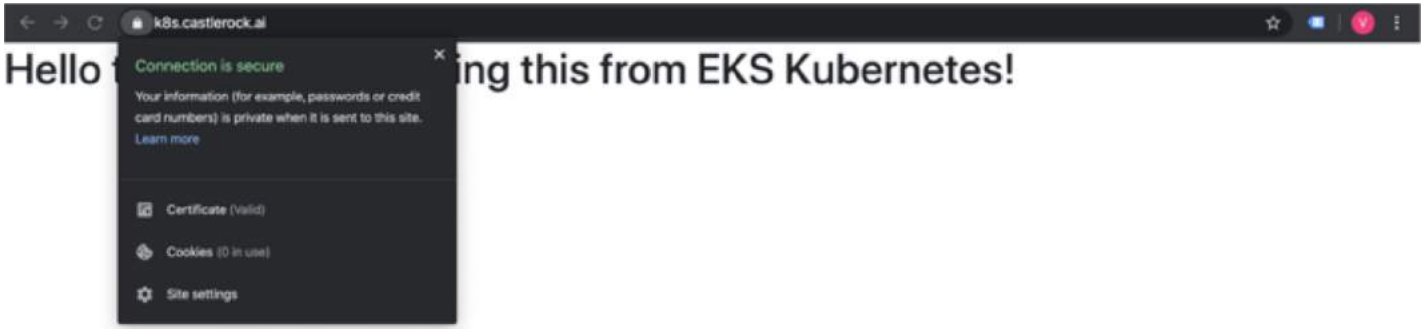
```
kubectl apply -f prod-issuer.yaml
kubectl describe clusterissuer letsencrypt-prod
```


See: https://cert-manager.io/docs/tutorials/acme/ingress/

Apply to ngx

Edit the Ingress, or apply new definition

```
kubectl apply -f ngx-ing.yaml
```



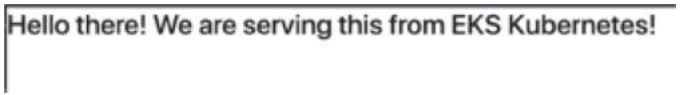
See: https://cert-manager.io/docs/tutorials/acme/ingress/


“Nothing lasts forever,
Even cold November Rain”
- W. Axl Rose

Showing Ephimerality

```
$ cd segment05-applications
$ kubectl cp webpage/index.html ngx-5f858c479c-hgmqp:/usr/share/nginx/html/
```

Look at the page: https://k8s.castlerock.ai



Kill the pod on another screen

```
$ kubectl delete pod ngx-5f85…
```

Look at the page again https://k8s.castlerock.ai

…back to boring nginx welcome screen.

Storage Classes

EBS

volume	gp2	io1	st1	stc1
Description	SSD	SSD fast!	HDD	HDD slow
Use Case	General workloads	Critical, Databases, etc	Big Data, log processing, warehouses	Large volumes infrequently accessed

EFS

FSx for Lustre

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html

Examine Storage classes

- `kubectl edit sc gp2`

```
 5 apiVersion: storage.k8s.io/v1
 6 kind: StorageClass
 7 metadata:
 8   annotations:
 9     kubectl.kubernetes.io/last-applied-configuration: |
10       {"apiVersion":"storage.k8s.io/v1","kind":"StorageClass"
   name":"gp2"},"parameters":{"fsType":"ext4","type":"gp2"},"pro
11     storageclass.kubernetes.io/is-default-class: "true"
12   creationTimestamp: "2020-02-13T17:25:34Z"
13   name: gp2
14   resourceVersion: "169"
15   selfLink: /apis/storage.k8s.io/v1/storageclasses/gp2
16   uid: d7ae0bf8-4e85-11ea-8bba-0a0a49091766
17 parameters:
18   fsType: ext4
19   type: gp2
20 provisioner: kubernetes.io/aws-ebs
21 reclaimPolicy: Delete
22 volumeBindingMode: WaitForFirstConsumer
```

Persistent Volume Claim

How we access

How much storage we want

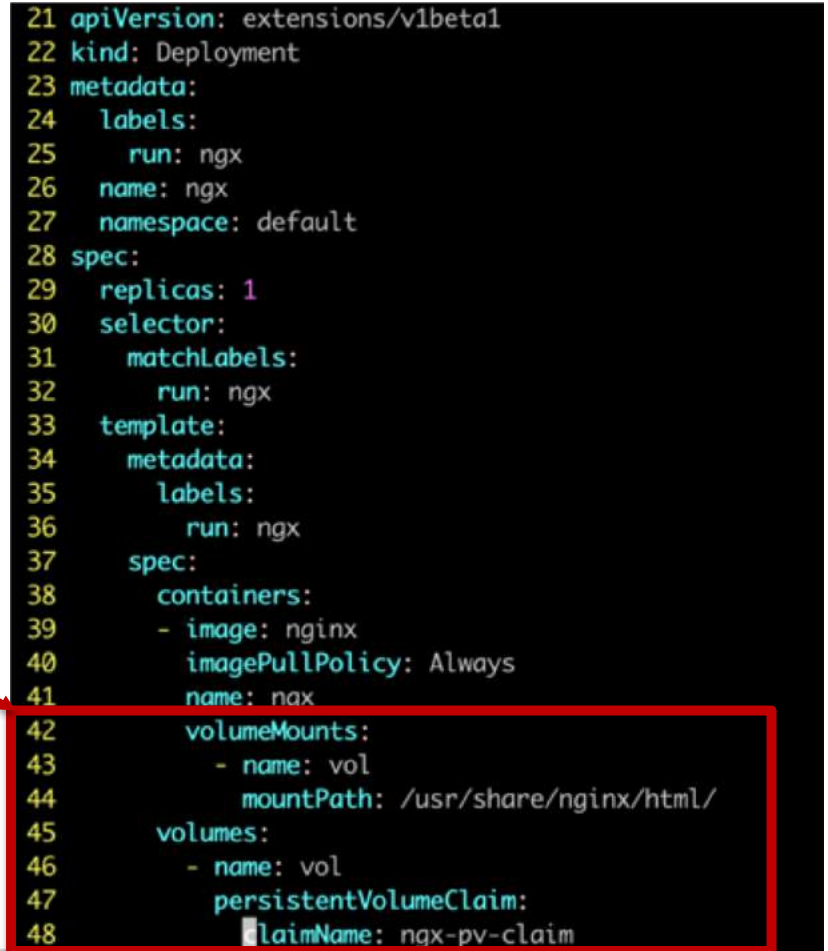
Storage Class we are requesting from

```
 5 apiVersion: v1
 6 kind: PersistentVolumeClaim
 7 metadata:
 8   name: ngx-pv-claim
 9   labels:
10     app: ngx
11 spec:
12   accessModes:
13     - ReadWriteOnce
14   resources:
15     requests:
16       storage: 5Gi
17   storageClassName: gp2
```

Persistent Volume in our Deployment

Append to our existing pod definition

```
kubectl apply -f ngx-volumes.yaml
```



```
21 apiVersion: extensions/v1beta1
22 kind: Deployment
23 metadata:
24   labels:
25     run: ngx
26   name: ngx
27   namespace: default
28 spec:
29   replicas: 1
30   selector:
31     matchLabels:
32       run: ngx
33   template:
34     metadata:
35       labels:
36         run: ngx
37     spec:
38       containers:
39       - image: nginx
40         imagePullPolicy: Always
41         name: nax
42         volumeMounts:
43           - name: vol
44             mountPath: /usr/share/nginx/html/
45       volumes:
46         - name: vol
47           persistentVolumeClaim:
48             claimName: ngx-pv-claim
```

Kubernetes on AWS

Segment 6: Administrative Tasks

Big Picture

- Cluster Autoscaling
- Horizontal Pod Autoscaling
- Kubernetes Dashboard
- Add additional Users
- Cluster Upgrades

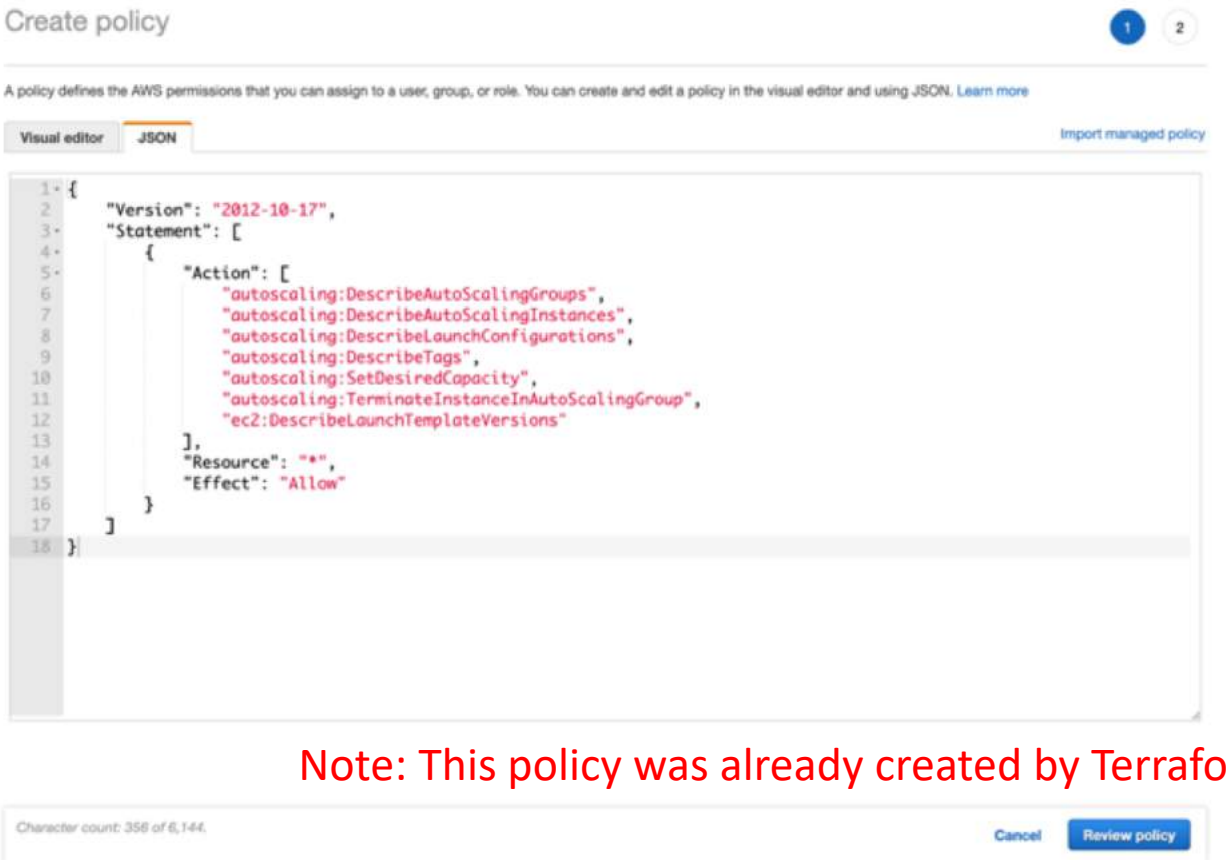
Autoscaling

- Cluster Autoscaling
  - Update the number of nodes in the Kubernetes cluster
- Horizontal Pod Autoscaling
  - Update the number of pods running in a cluster
- Vertical Pod Autoscaling
  - Update the CPU and memory reservations for the pods

Cluster Autoscaling

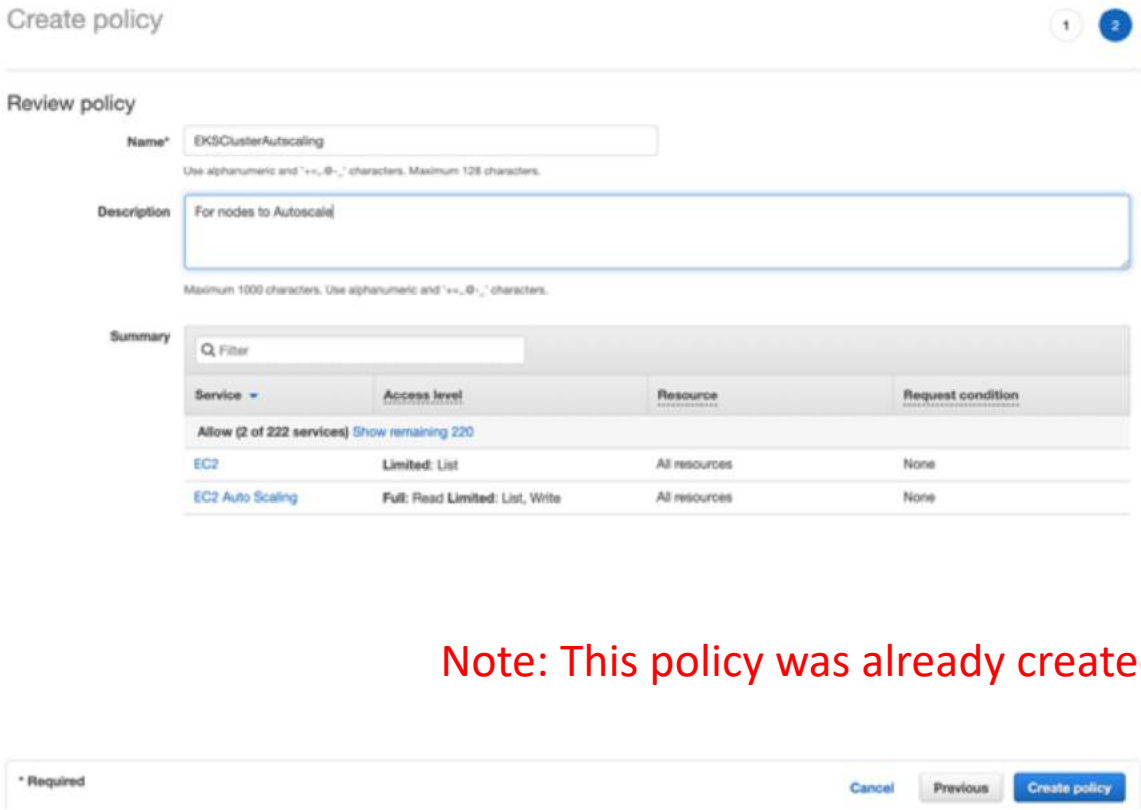
- Create policy for nodes to autoscale
- Apply policy to node role that servers are under
- Install Metrics Server
- Install Autoscaler
- Configure Autoscaler (give it the right cluster name)
- Scale Demo

Cluster Autoscaling



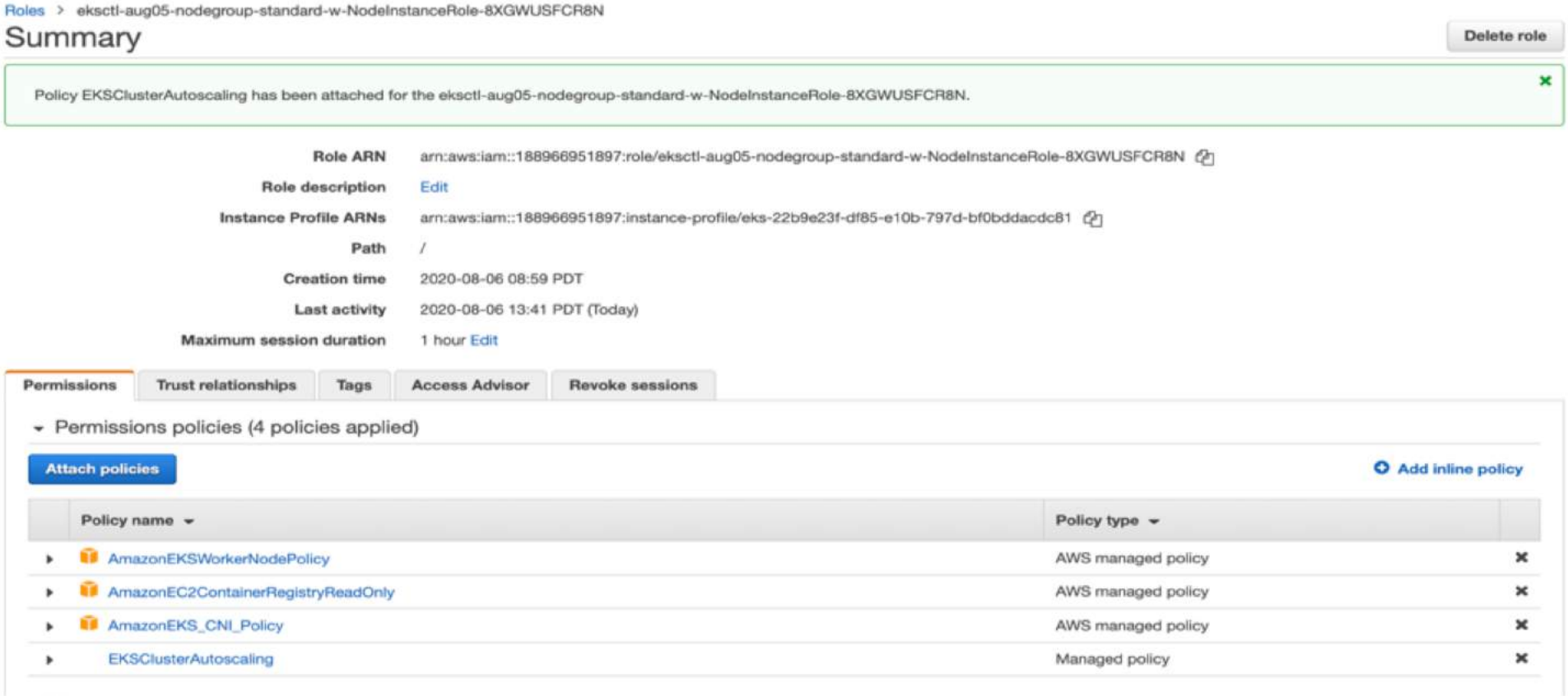
Note: This policy was already created by Terraform

Cluster Autoscaling



Note: This policy was already created by Terraform

Cluster Autoscaling Role Policy



Install Cluster Autoscaler

- Install Metrics Server

```
cd segment06-admin/
kubectl apply -f metrics-server-0.3.6/components.yaml
```

- Install Cluster Autoscaler

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/autoscaler/
master/cluster-
autoscaler/cloudprovider/aws/examples/cluster-
autoscaler-autodiscover.yaml
```

- Allow Annotation

```
kubectl -n kube-system annotate deployment.apps/cluster-
autoscaler cluster-autoscaler.kubernetes.io/safe-to-
evict="false"
```

Edit Autoscaler configuration

```
kubectl -n kube-system edit \
deployment.apps/cluster-autoscaler
```

Update with:

```
...
- --node-group-auto-
discovery=asg:tag=k8s.io/cluster-
autoscaler/enabled,k8s.io/cluster-
autoscaler/dec07
- --skip-nodes-with-system-pods=false
- --balance-similar-node-groups
...
```

Test Cluster Autoscaling

- Check nodes

```
kubectl get nodes
```

- Create Deployment

```
kubectl create deployment autoscaler-demo --image=nginx
```

- Scale Deployment

```
kubectl scale deployment autoscaler-demo --replicas=50
```

- Check nodes and logs

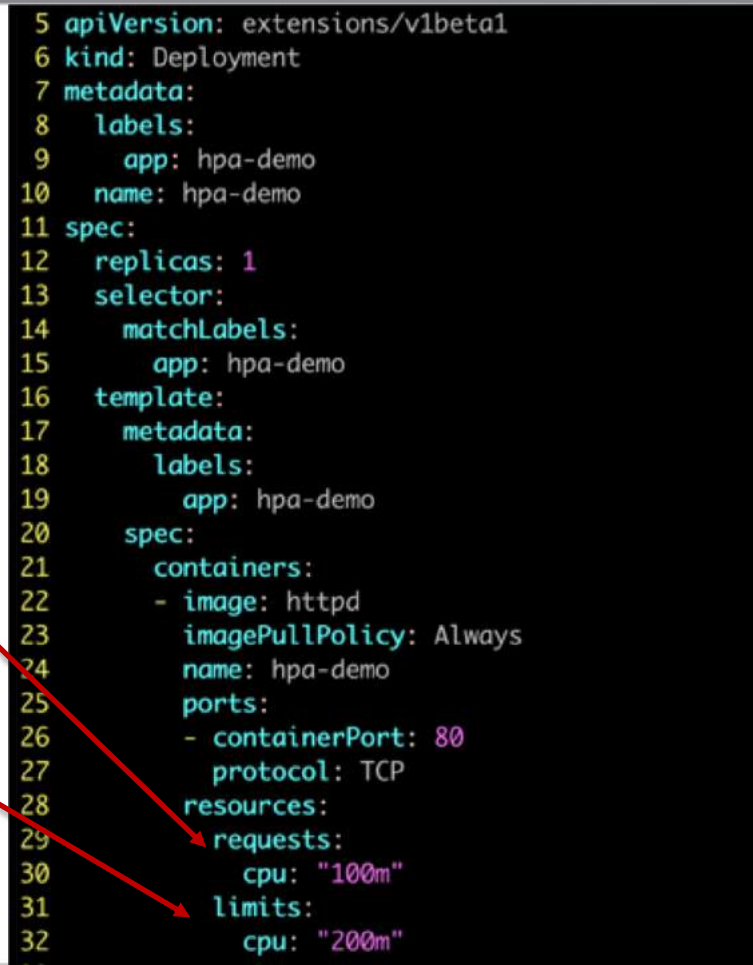
```
kubectl get nodes
kubectl logs -f -n kube-system cluster-autoscaler<tab>
```

Horizontal Pod Autoscaler (HPA)

- Ensure that Metrics Server is deployed
- Create test deployment
- Attach Autoscaler
- Scale load

Add Resource Requests and Limits

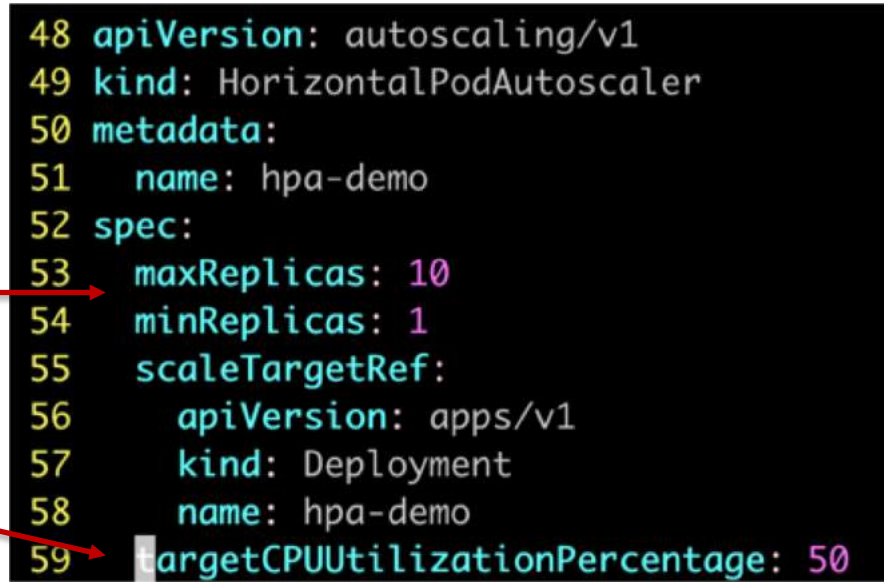
- 100m = 100 millicores
- Limit is the most we will allow for this container (0.2 CPU)



```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: hpa-demo
  name: hpa-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hpa-demo
  template:
    metadata:
      labels:
        app: hpa-demo
    spec:
      containers:
      - image: httpd
        imagePullPolicy: Always
        name: hpa-demo
        ports:
        - containerPort: 80
          protocol: TCP
        resources:
          requests:
            cpu: "100m"
          limits:
            cpu: "200m"
```

Add HPA Policy

- Max replicas 10
- Min replicas 1
- Target 50% CPU utilization
  - (100m = 50m)



```
48 apiVersion: autoscaling/v1
49 kind: HorizontalPodAutoscaler
50 metadata:
51   name: hpa-demo
52 spec:
53   maxReplicas: 10
54   minReplicas: 1
55   scaleTargetRef:
56     apiVersion: apps/v1
57     kind: Deployment
58     name: hpa-demo
59   targetCPUUtilizationPercentage: 50
```

HPA – Create and Test

```
kubectl create -f hpa-demo.yaml

kubectl run apache-bench -i --tty \
    --rm --image=httpd -- ab -n 500000 \
    -c 1000 http://hpa-demo.default.svc.cluster.local/
```

```
vallards-MacBook-Pro:segment06-admin vallard$ kubectl get hpa -w
NAME       REFERENCE                 TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
hpa-demo   Deployment/hpa-demo   198%/50%   1         10        1          116s
hpa-demo   Deployment/hpa-demo   198%/50%   1         10        4          2m1s
hpa-demo   Deployment/hpa-demo   155%/50%   1         10        4          2m47s
hpa-demo   Deployment/hpa-demo   155%/50%   1         10        8          3m2s
```

Kubernetes Dashboard

- Deploy:

```
kubectl apply –f segment06-admin/dashboard.yaml
```

- Create Service Account

```
kubectl apply –f segment06-admin/ eks-admin-service-account.yaml
```

- Get Credentials
  - `kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep eks-admin | awk '{print $1}')`
- Connect
  - `kubectl proxy`

Kubernetes Dashboard

- Create the dashboard and service account

```
cd dashboard/
kubectl apply -f dashboard.yaml

kubectl apply -f eks-admin-service-
account.yaml
```

Get Token:

- `kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep eks-admin | awk '{print $1}')`

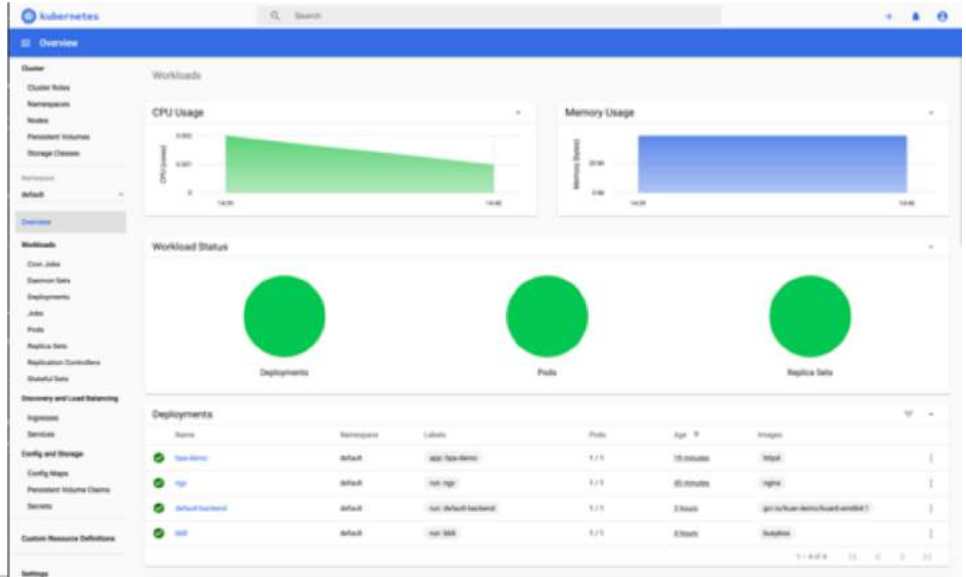
```
Data
====
ca.crt:     1025 bytes
namespace:  11 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJla3MtYWRtaW4tdG9rZW4tbXhramMiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoiZWtzLWFkbWluIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQudWlkIjoiYmY0ZDNhZmMtMTY2ZC00ZTNmLTk0OWUtZDFiNmM5NjZiMjBmIiwic3ViIjoic3lzdGVtOnNlcnZpY2VhY2NvdW50Omt1YmUtc3lzdGVtOmVrcy1hZG1pbiJ9.Sh_sJKuKnKC5V_YV378y1roCLUpAhxR778m_viXQsCEeqUI-21ERAmHdBlj4UYS8GoaXXAII9xLbZDDKzz1OwsZrbIHdEOUIBkkaJBtBTIK5CULvXVJra3ZIjlvRhJtJVbFGBC2-hYmRSN9hZkQz7hRBBLDIVo-olj5Av58TaWHXNH3Flux-7pWy_h9XUXl9bimbH#HKt7gzQeMw2Rqy7ybU5ecAJBKKXji9l5rtr1FNwGm9I6VV_7Qwhp9aSQTzDWAEFWmA8gE0nWu3-haaFlhL1yOGjTBhVoJdQbs47gX6RYB3GoRJKvLWoZ7pu0kaEwtvmTPwezSjZSA139c20Kg
```

Access Dashboard

- Localhost via Proxy:

http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/#!/login

- May take a few minutes before metrics appear.



Add another AWS User

- By default person who created cluster is the only person who has access to it.
- Other users must authenticate with AWS before getting authorization to run kubectl commands. (Might take 15 min)
- Give other user EKS group name to add permissions
- Other user runs:

```
aws eks update-kubeconfig --name aug05
aws sts get-caller-identity
```

- Sends caller identity to cluster owner:

```
root@walrus:~# aws sts get-caller-identity
188966951897    arn:aws:iam::188966951897:user/anotherUser      AIDASX7203PMVFHROSQHV
```

Add another AWS User

```
root@walrus:~# kubectl  get pods
error: You must be logged in to the server (Unauthorized)
```

- Is logged into AWS but not to Kubernetes
- From EKS cluster creater account run:

```
kubectl get cm -n kube-system
kubectl edit cm -n kube-system aws-auth
```

```
mapUsers: |
  - userarn: arn:aws:iam::254056899943:user/judy
    username: judy
    groups:
      - system:masters
```

Limiting Scopes of users

- Create a Role and a Role Binding
- Add User to Role Group in aws-auth Config Map
- Group is "`read-pods-role`" instead of previous "`system:masters`"

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: read-pods-role
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods-role-binding
subjects:
- kind: User
  name: read-pods
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: read-pods-role
  apiGroup: rbac.authorization.k8s.io
```

https://www.freecodecamp.org/news/adding-limited-access-iam-user-to-eks-cluster/

Cluster Upgrades

- Upgrade often, check every 3 months
- Can only migrate one minor version at a time:
  - 1.12 -> 1.13 -> 1.14 -> 1.15 -> 1.16 -> 1.17-> 1.18
- eksctl is easy if you created cluster via eksctl
- Console upgrade process also somewhat simple
  - Upgrade control panel in EKS console
  - Upgrade nodes with cloudformation template

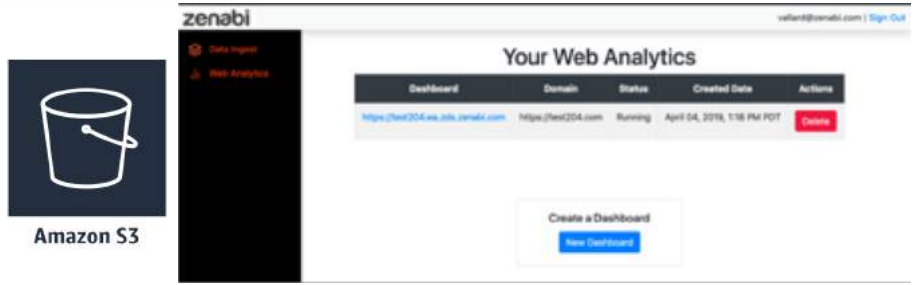
Kubernetes on AWS

Segment 7: Integrations with other AWS Services

Big Picture

- Serverless
- Accessing resources from Kubernetes pods

Serverless Architecture



Amazon S3

Amazon API Gateway

/wa GET

/wa POST

/wa DELETE

AWS Lambda

AWS Lambda

AWS Lambda

Amazon DynamoDB

No Operating Systems

No Containers

No Packaging

Expanding into EKS

kubernetes

Kubernetes: Each instance in own namespace

App SVC	PHP App Pod	Config EBS	Cert
MariaDB SVC	MariaDB Pod	DB EBS	Ingress Rule

/wa GET | /wa POST | /wa DELETE

Data Migration

S3

Pearson

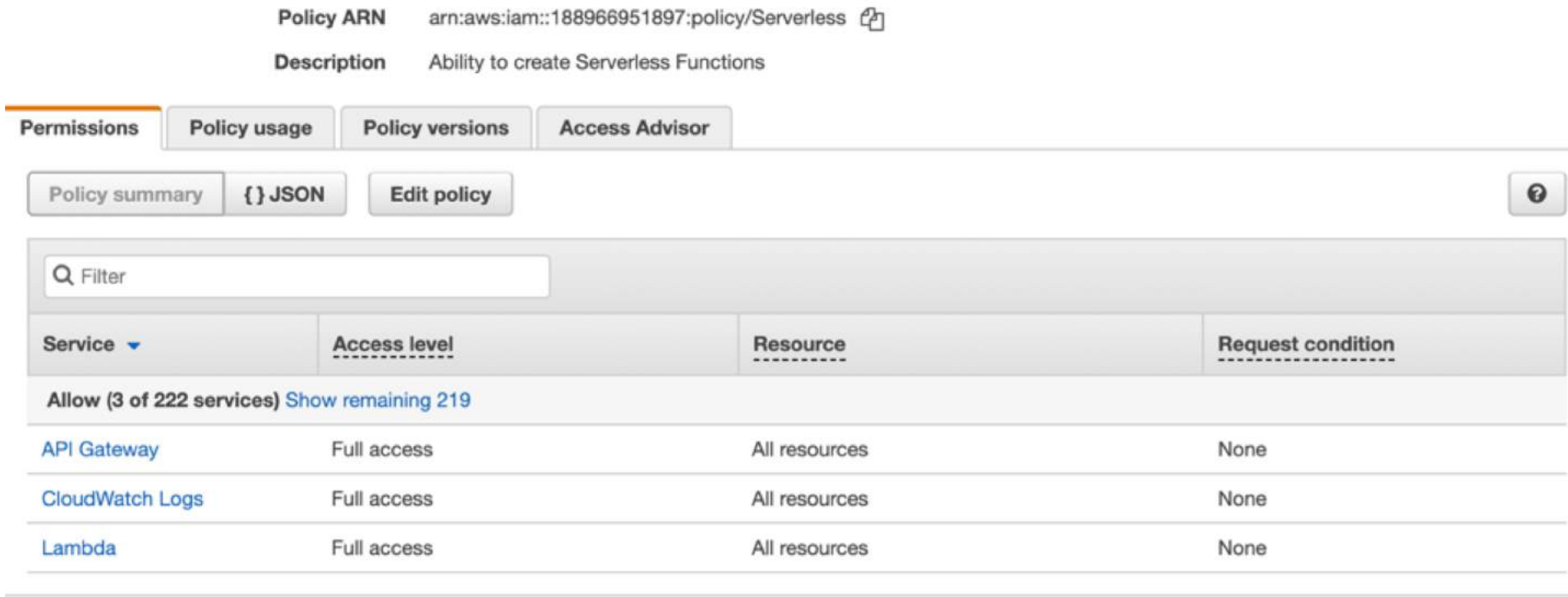
Sample Application

- List the pods in your system

Python code with
Kubernetes library

Create Serverless Policy

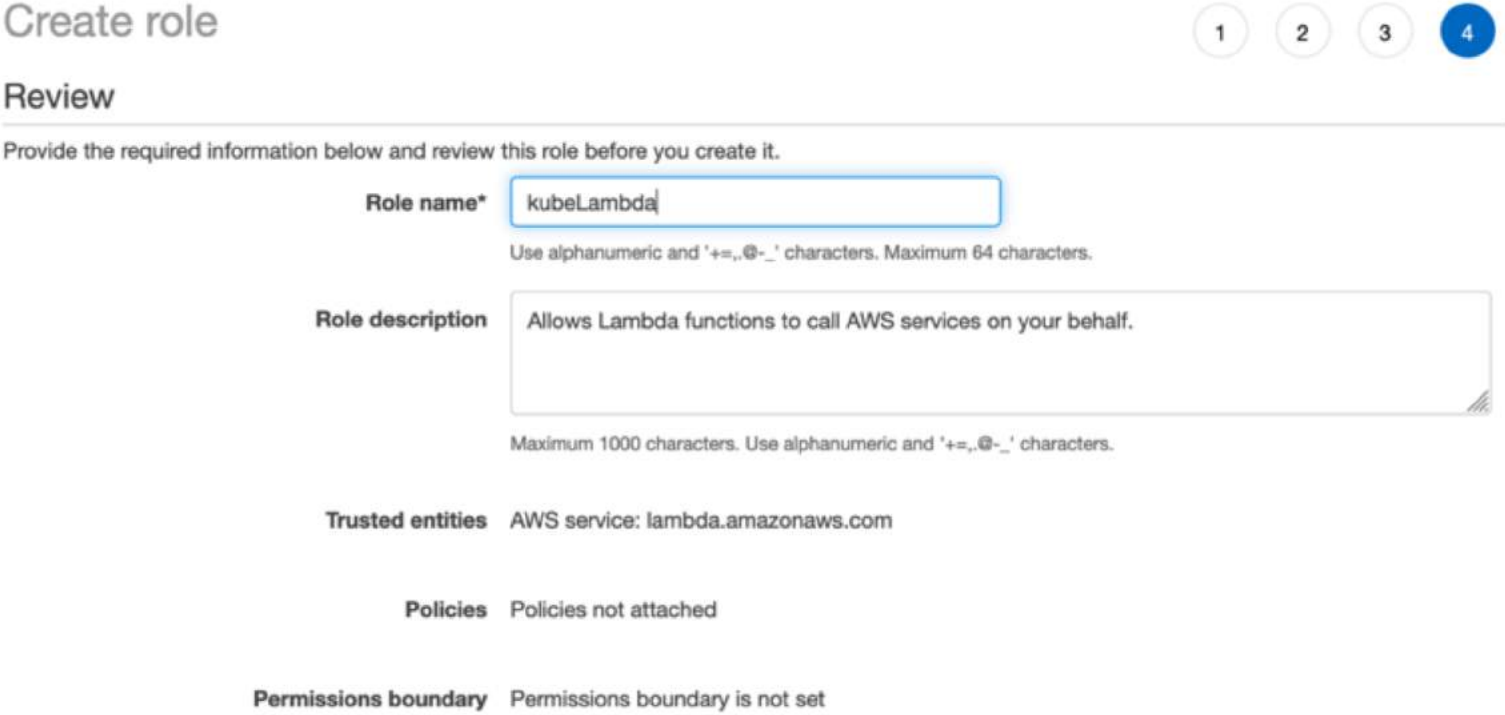
Add Serverless Policy to EKSDemoGroup



Pearson

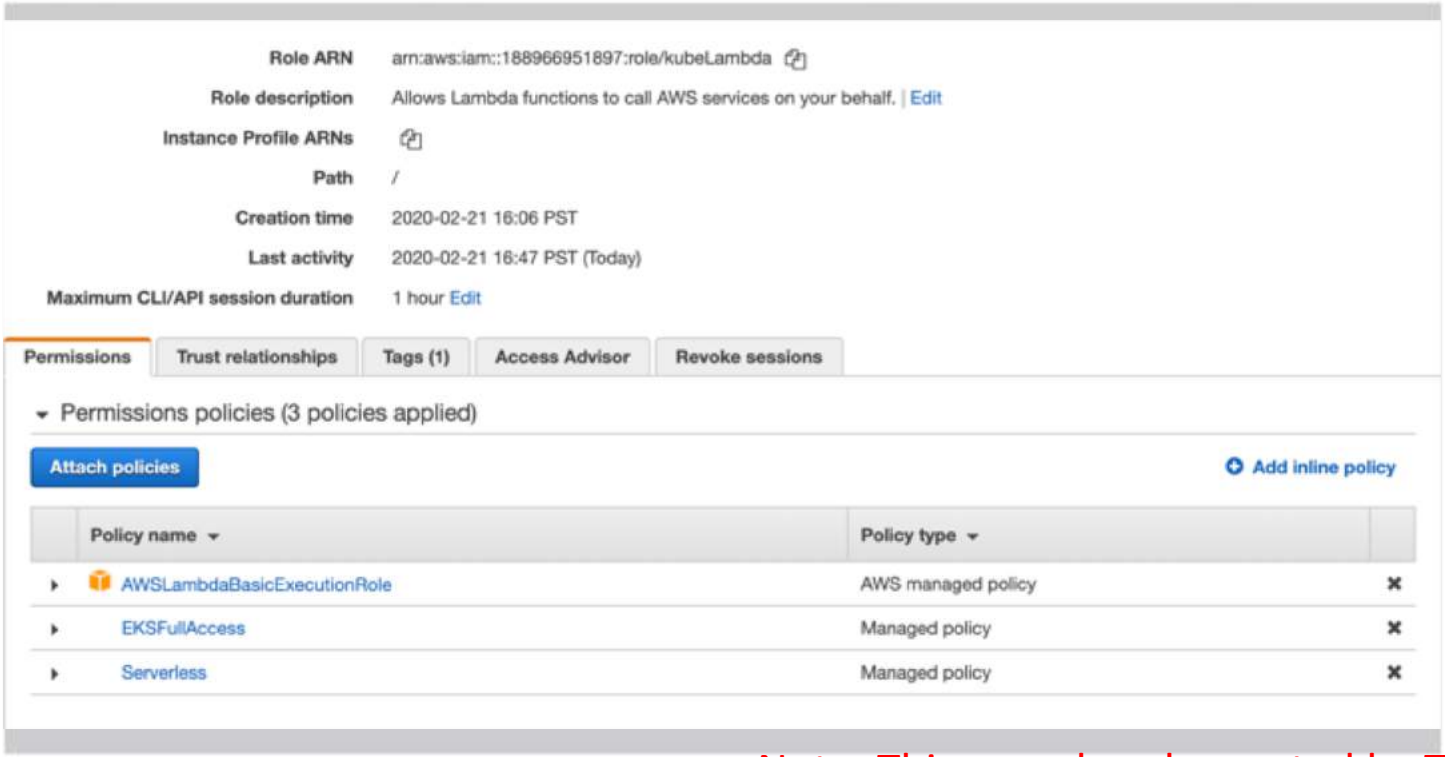
Note: This policy was already created by Terraform

Kube Lambda Role



Note: This policy was already created by Terraform

Add Policies to Role



Note: This was already created by Terraform

Add to aws-auth ConfigMap

```
 7  mapRoles: |
 8    - groups:
 9      - system:bootstrappers
10      - system:nodes
11      rolearn: arn:aws:iam::188966951897:role/eksctl-eksctl-2-18-nodegroup-stan-NodeInstanceRole-1H3V6PKC40V5T
12      username: system:node:{{EC2PrivateDNSName}}
13    - rolearn: arn:aws:iam::188966951897:role/kubeLambda
14      username: kubeLambda
15      groups:
16        - system:masters
17  mapUsers: |
18    - userarn: arn:aws:iam::188966951897:user/anotherUser
19      username: read-pods
20      groups:
21        - read-pods
```

Serverless

- `sls deploy`
- `sls invoke local -f list`
- `sls deploy -f list`
- `sls invoke -f list`
- `sls logs -f list`
- `sls remove`

```
curl https://0qtyabihgc.execute-api.us-west-2.amazonaws.com/dev/list | jq
```

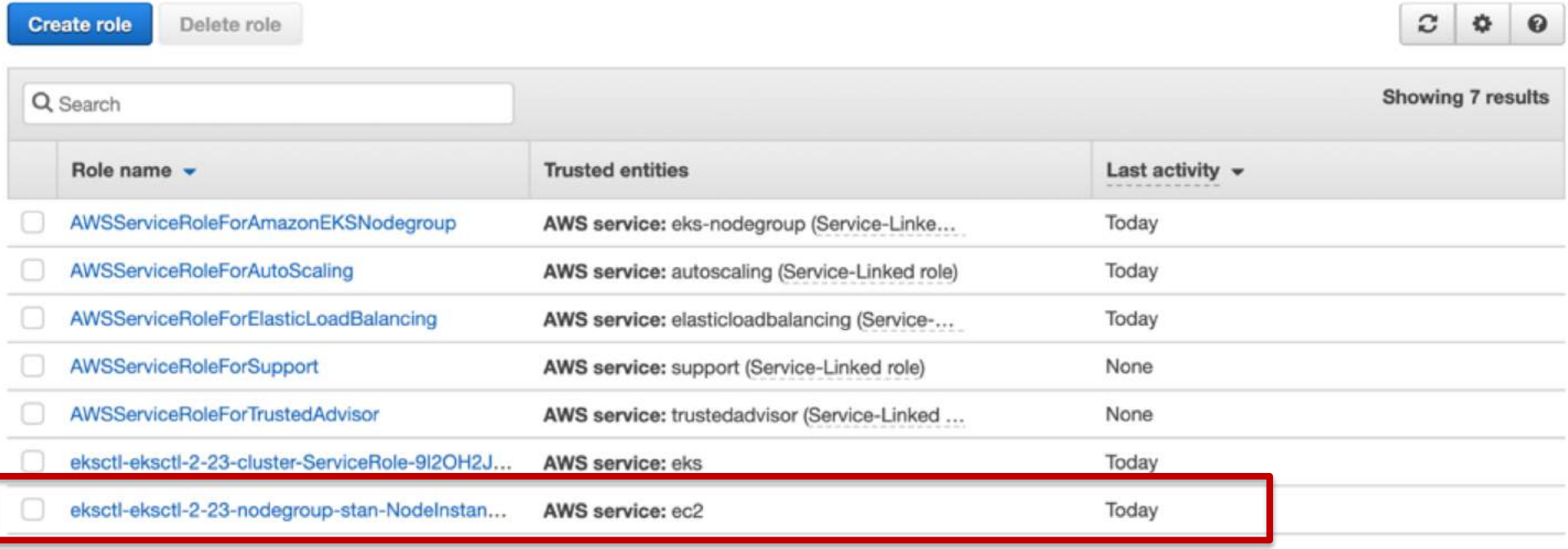
Considerations

- This is **wide open**, we would typically add authentication (Cognito) so not everyone could access and list our pods in our cluster.
- https://serverless-stack.com for inspiration
- Creating Batch, CronJobs may be cheaper than running on servers
- Scale up / Scale Down with HPA and Cluster Autoscaling makes this a good play.

Accessing AWS Resources in Pods

- One of the most valuable reasons to run EKS is access to other AWS resources
- S3, DynamoDB, other databases, Redshift, are all accessible from your applications that run in pods
- Usually IAM policies and roles are the only tricky parts to worry about when writing these applications.

Add permissions to Node Role



Add DynamoDB Policy to node Role

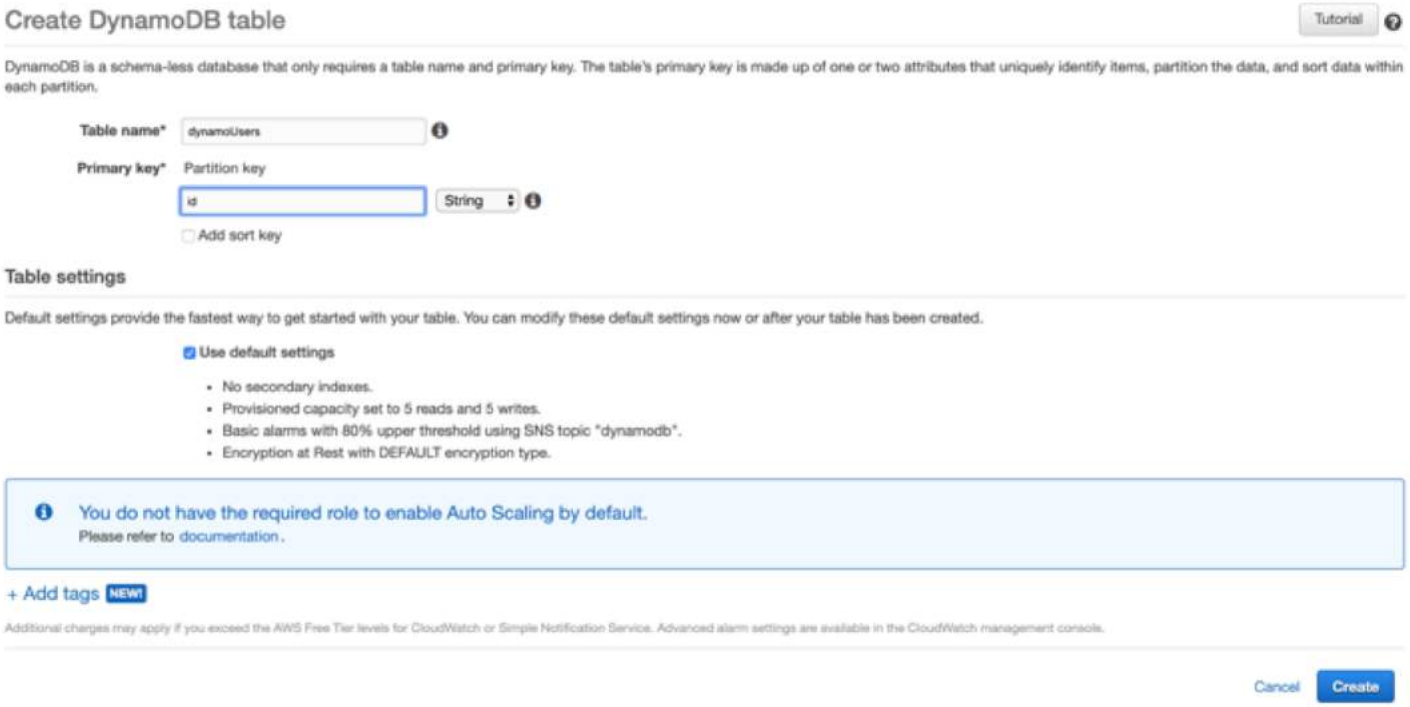
Now applications running in the node get these permissions

Add DynamoDB Policy to node Role

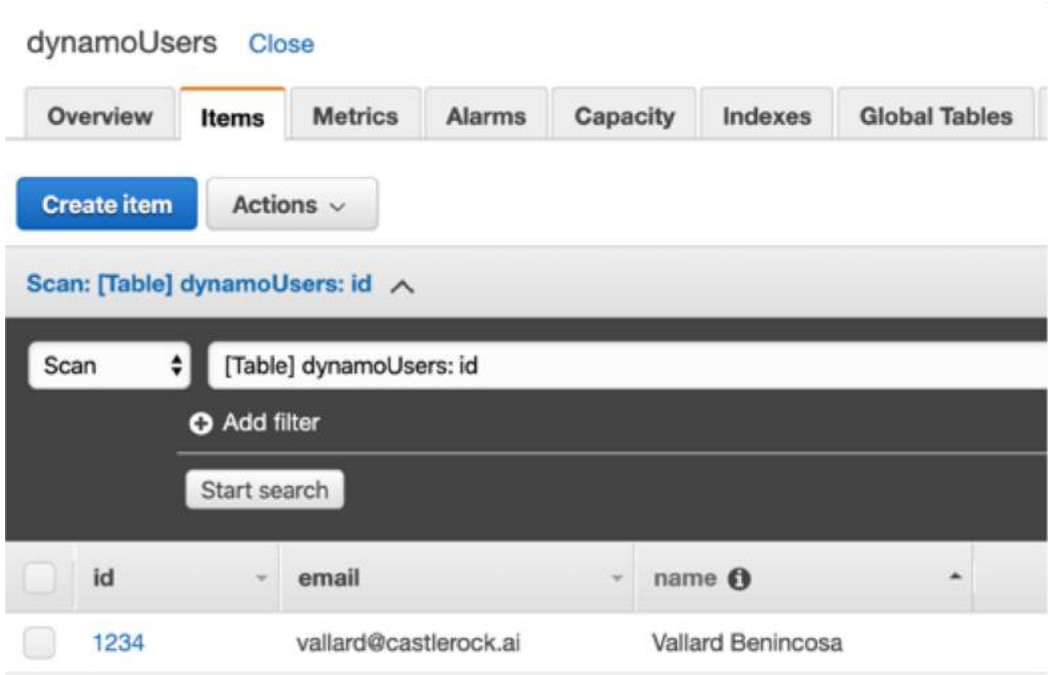
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetShardIterator",
                "dynamodb:Scan",
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:254056899943:table/zds/*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "dynamodb:ListStreams",
            "Resource": "arn:aws:dynamodb:us-east-1:254056899943:table/zds/*"
        }
    ]
}
```

Make a custom policy to limit access to only certain DBs

Create a Dynamo DB table: dynamoUsers

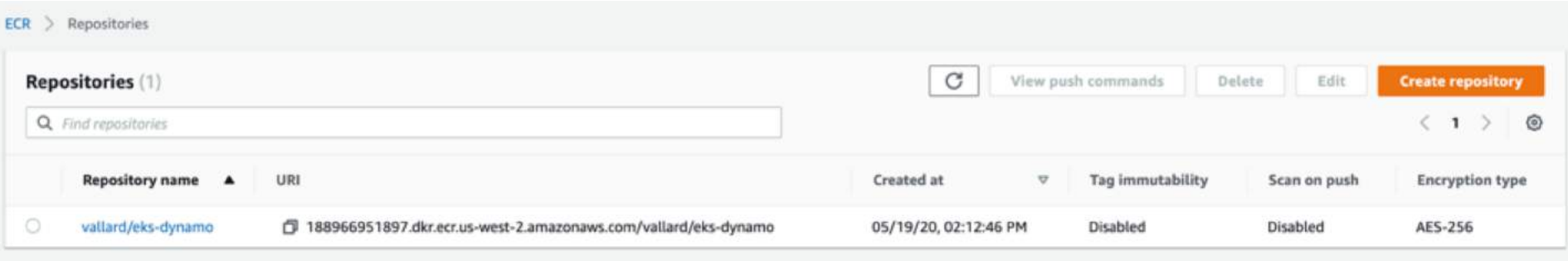


Add a Dynamo Entry to dynamoUsers



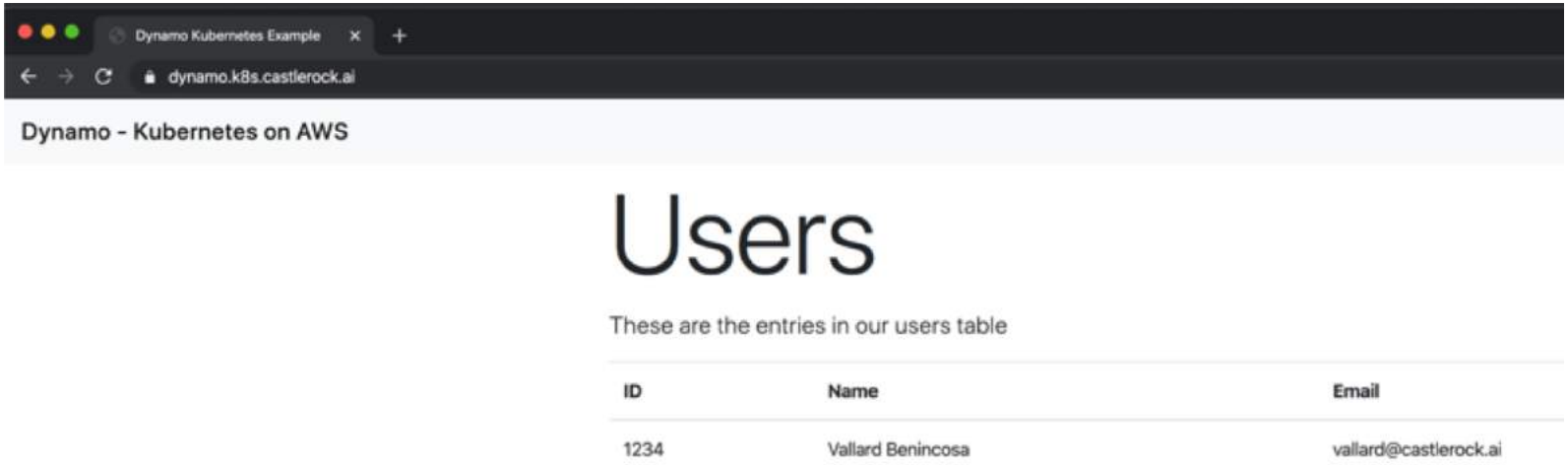
Deploy Sample Application

- Build the Application: `make build`
- Put application in ECR if not created yet



- Push the application: `make push`
- Deploy: `kubectl apply -f eks-dynamo.yaml`

Refresh the Application



Kubernetes on AWS

Conclusion

What we covered! ☺️

- Brief introduction to Kubernetes, Containers, EKS
- IAM Contstructs and Command Line Tools
- Creating EKS Clusters: eksctl & Console
- Verifying and Testing our EKS cluster
- Running Applications in the real world
- Administrative Tasks
- Integrations with other AWS services

What we didn’t cover 😔

- Mulitiple Availability Zones
- Windows Worker nodes
- Fargate for serverless cluster nodes
- App Mesh
- Deep Learning Containers with GPU Instances
- Not too much on Amazon ECR
- Other advanced customizations: Instances, Networking, Volumes

Thank you!

@vallard