Experiences with AWS EKS

Kubernetes and Cloud Native Group Dresden

ALFRED KROHMER

Structure

- Introduction
  - overview / EKS cluster creation
  - reasons for our ongoing migration (and what we had before)
- AWS-specific cluster addons
  - VPC CNI plugin
  - external-dns
  - alb-ingress-controller
  - kube2iam
- Worker node management
  - cluster-autoscaler
- Q&A

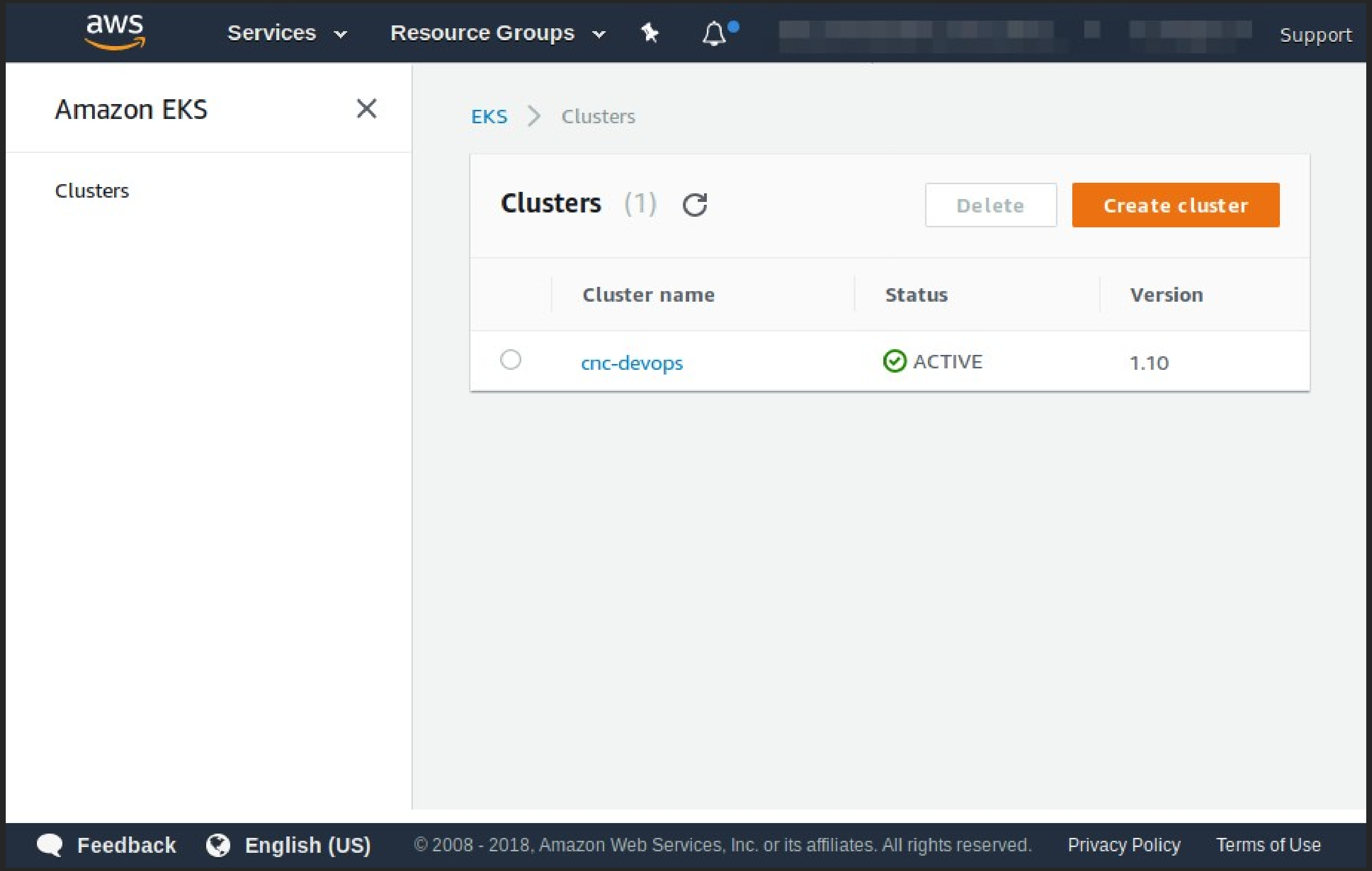
Introduction

- **Overview**
  - EKS = Elastic Kubernetes Service
  - AWS provisions and maintains master nodes
  - you provision and maintain worker nodes
    - “EKS-optimized” AMI by AWS
  - authentication via »AWS IAM Authenticator for Kubernetes«

    https://github.com/kubernetes-sigs/aws-iam-authenticator

Introduction

- **Cluster overview**



	Cluster name	Status	Version
○	cnc-devops	✓ ACTIVE	1.10

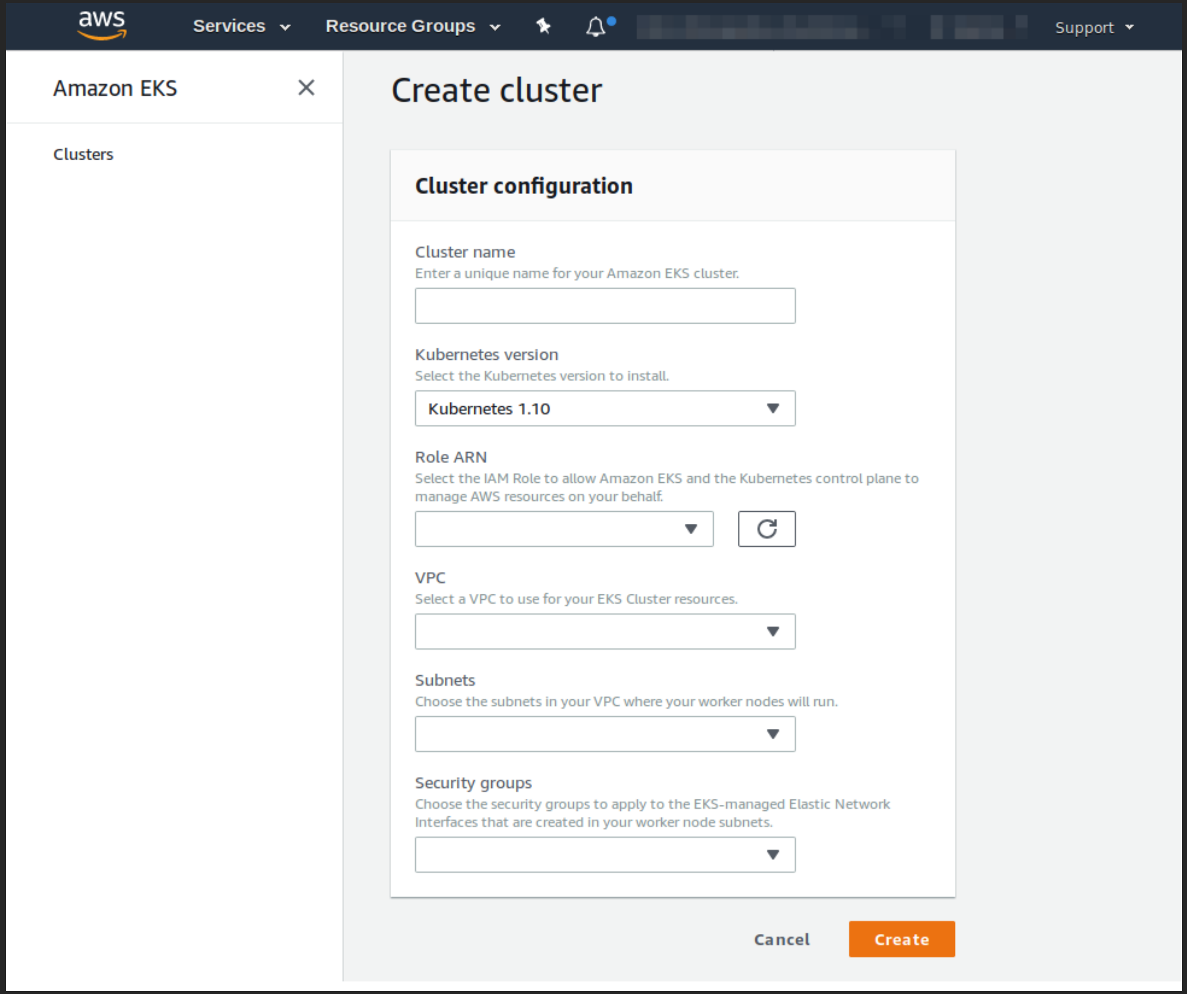
© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Introduction

- **Cluster creation**

  IAM user which creates the cluster is (and stays) Kubernetes admin user (for now)

  → use a headless account to create the cluster



Create cluster

Cluster configuration

Cluster name
Enter a unique name for your Amazon EKS cluster.

Kubernetes version
Select the Kubernetes version to install.
Kubernetes 1.10

Role ARN
Select the IAM Role to allow Amazon EKS and the Kubernetes control plane to manage AWS resources on your behalf.

VPC
Select a VPC to use for your EKS Cluster resources.

Subnets
Choose the subnets in your VPC where your worker nodes will run.

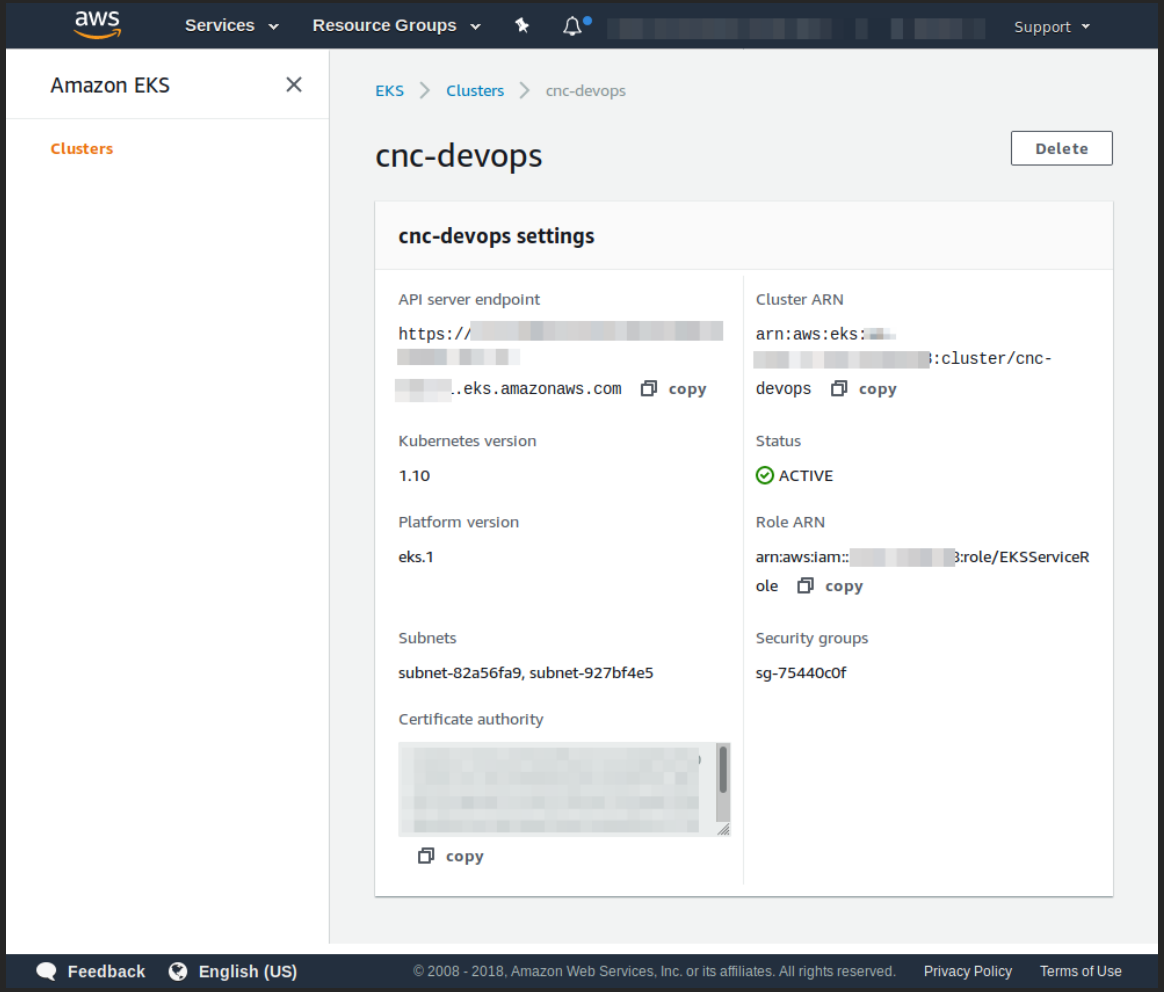
Security groups
Choose the security groups to apply to the EKS-managed Elastic Network Interfaces that are created in your worker node subnets.

Introduction

- **Cluster details**

  → details cannot be modified (for now)

  → “Platform versions” are supposed to be rolled out by AWS



cnc-devops

cnc-devops settings

API server endpoint

https://[redacted].eks.amazonaws.com

Cluster ARN

arn:aws:eks:[redacted]:cluster/cnc-devops

Kubernetes version

1.10

Status

ACTIVE

Platform version

eks.1

Role ARN

arn:aws:iam::[redacted]:role/EKSServiceRole

Subnets

subnet-82a56fa9, subnet-927bf4e5

Security groups

sg-75440c0f

Certificate authority

[redacted]

© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Introduction

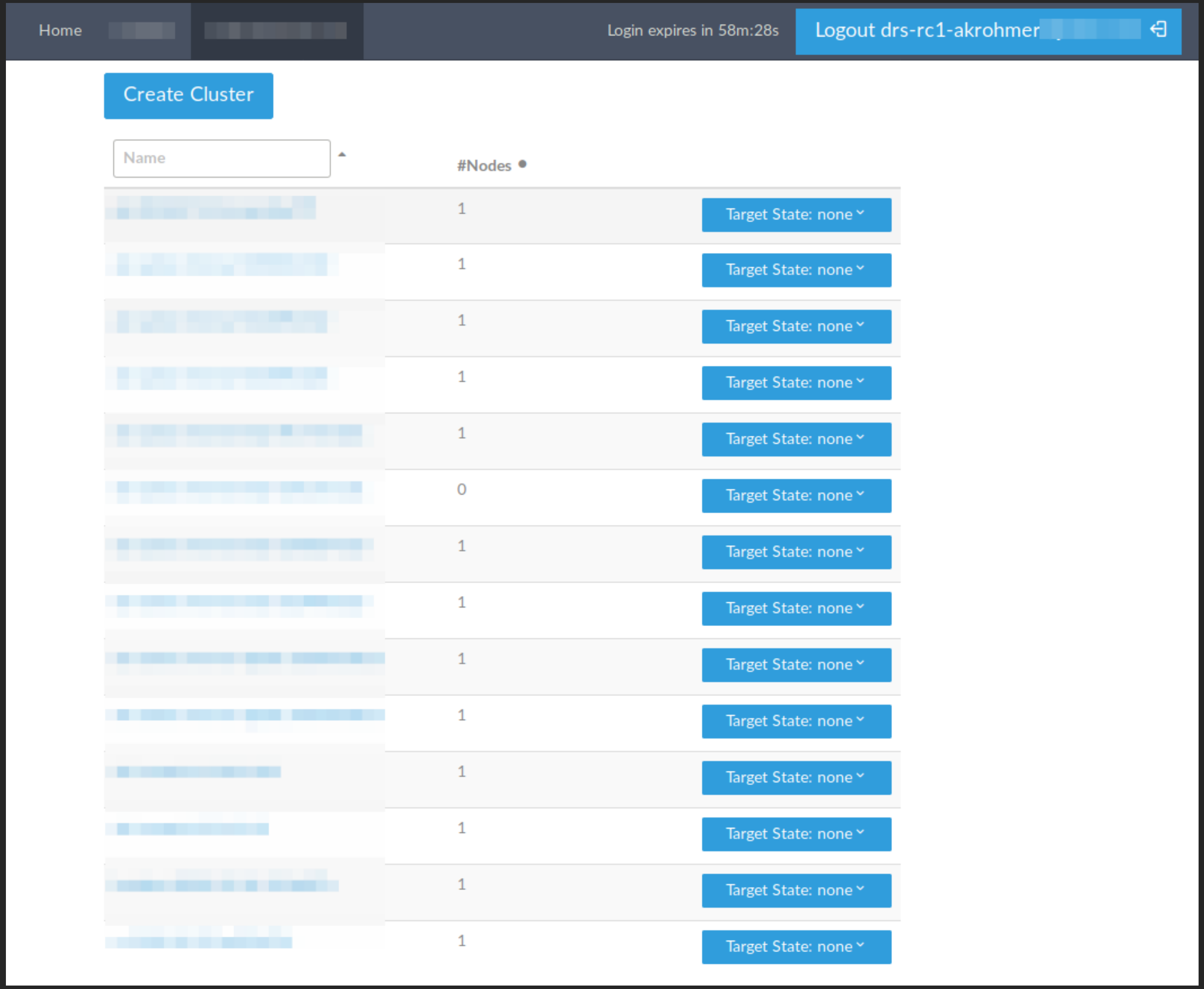
- **Our (ongoing) migration away from**
  - **AWS ECS:**
    - slow (especially with CloudFormation)
    - lack of flexibility
    - vendor lock-in
    - EC2 instances unmaintained
    - Fargate much more expensive then EC2 spot
    - 30s maximum graceful termination period
    - cluster needs to have enough capacity **before** scheduling a task / service

Introduction

- **Our (ongoing) migration away from**
  - **our own container orchestration framework**
    - availability problems (in dev)
    - lack of flexibility
      - one “pod” per host deployments only
    - overly complicated setup with a custom host agent and in-container agent both connecting to ZooKeeper, with frontend UI and REST API
      - starting containers
      - providing application configuration

Introduction

- Our (ongoing) migration away from
  - our own container orchestration framework

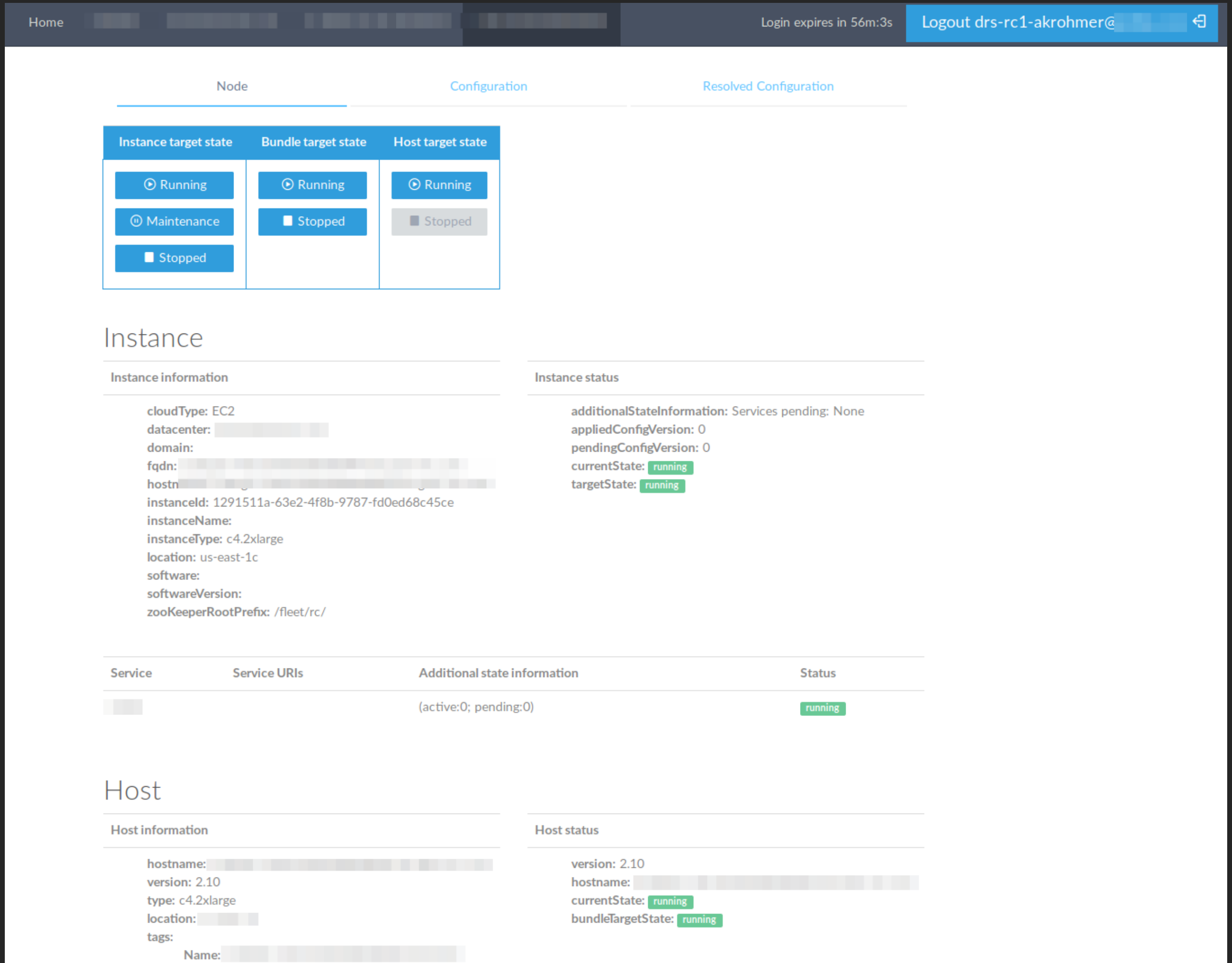


Name	#Nodes	
[redacted]	1	Target State: none
[redacted]	1	Target State: none
[redacted]	1	Target State: none
[redacted]	1	Target State: none
[redacted]	1	Target State: none
[redacted]	0	Target State: none
[redacted]	1	Target State: none
[redacted]	1	Target State: none
[redacted]	1	Target State: none
[redacted]	1	Target State: none
[redacted]	1	Target State: none
[redacted]	1	Target State: none
[redacted]	1	Target State: none
[redacted]	1	Target State: none

Introduction

- **Our (ongoing) migration away from**
  - **our own container orchestration framework**

  all nodes hosted via EC2



Instance

Instance information

cloudType: EC2
datacenter: [redacted]
domain:
fqdn: [redacted]
hostn[redacted]
instanceId: 1291511a-63e2-4f8b-9787-fd0ed68c45ce
instanceName:
instanceType: c4.2xlarge
location: us-east-1c
software:
softwareVersion:
zooKeeperRootPrefix: /fleet/rc/

Instance status

additionalStateInformation: Services pending: None
appliedConfigVersion: 0
pendingConfigVersion: 0
currentState: running
targetState: running

Service	Service URIs	Additional state information	Status
[redacted]		(active:0; pending:0)	running

Host

Host information

hostname: [redacted]
version: 2.10
type: c4.2xlarge
location: [redacted]
tags:
  Name: [redacted]

Host status

version: 2.10
hostname: [redacted]
currentState: running
bundleTargetState: running

AWS-specific cluster addons

- **VPC CNI plugin**
  - CNI plugin for kubelet
  - open source
  - can be used without EKS / with custom Kubernetes installations in EC2

  - https://github.com/aws/amazon-vpc-cni-k8s

AWS-specific cluster addons

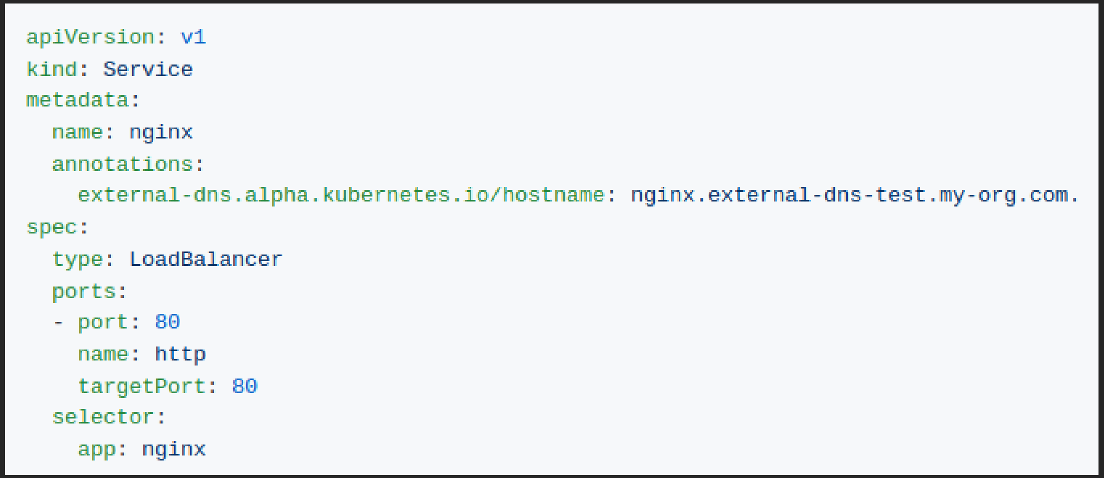
- **VPC CNI plugin**
  - uses AWS VPC as “overlay” network for Kubernetes
    - assigns secondary IPs to EC2 worker nodes
    - routing rules pass traffic through veth pairs to pod network
  - all other EC2 instances in your VPC can connect to your pods directly

AWS-specific cluster addons

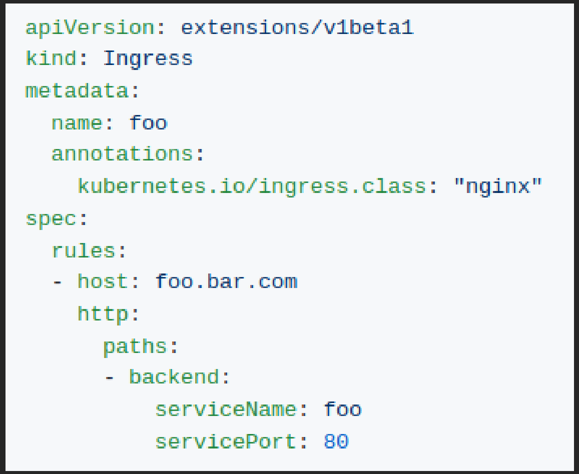
- **VPC CNI plugin**
  - beware of the defaults:
    - allocates as many IPs as possible per instance → IP starvation in small subnets
    - secondary network interfaces: uses SNAT to primary private IP to reach outside of VPC
  - “number of available IPs” is not managed as a resource

AWS-specific cluster addons

- **external-dns**
  - creates Route 53 resource records for your services, ingresses and StatefulSet (or single) pods



```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    external-dns.alpha.kubernetes.io/hostname: nginx.external-dns-test.my-org.com.
spec:
  type: LoadBalancer
  ports:
  - port: 80
    name: http
    targetPort: 80
  selector:
    app: nginx
```



```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: foo
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - backend:
          serviceName: foo
          servicePort: 80
```

  - https://github.com/kubernetes-incubator/external-dns/blob/master/docs/tutorials/aws.md

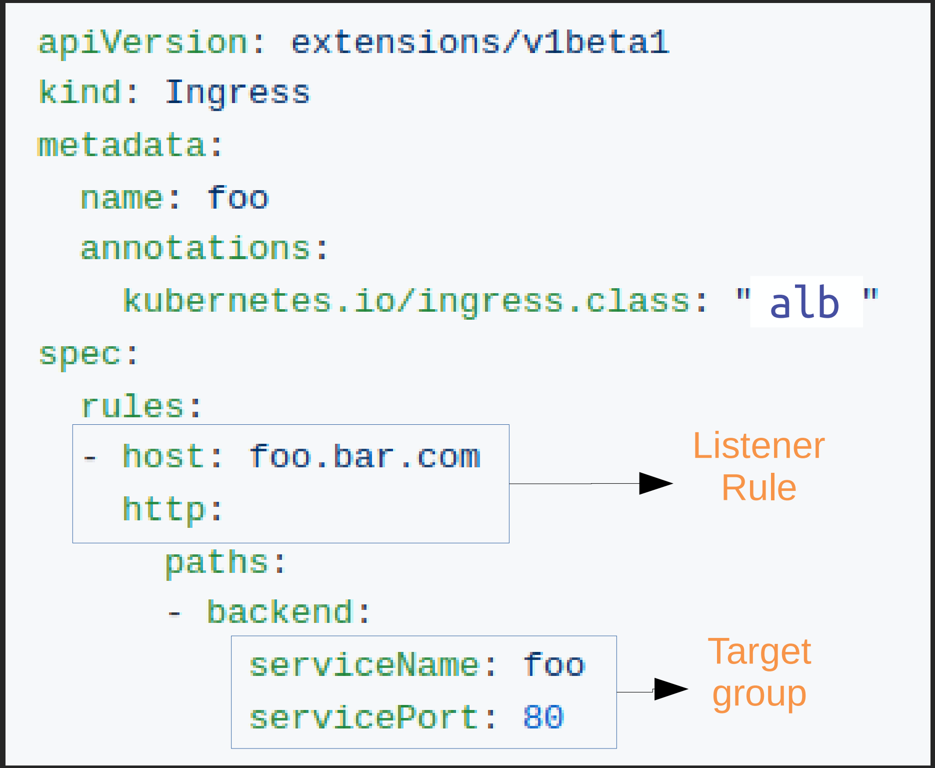
AWS-specific cluster addons

- **external-dns**
  - uses Route 53 TXT records for own storage
  - also supports other cloud providers
  - missing feature for us:
    creating per-pod DNS records

AWS-specific cluster addons

- **alb-ingress-controller**
  - creates AWS Application Load Balancers (ALBs) for your ingresses

    targets are the VPC IPs of your pods



```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: foo
  annotations:
    kubernetes.io/ingress.class: "alb"
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - backend:
          serviceName: foo
          servicePort: 80
```

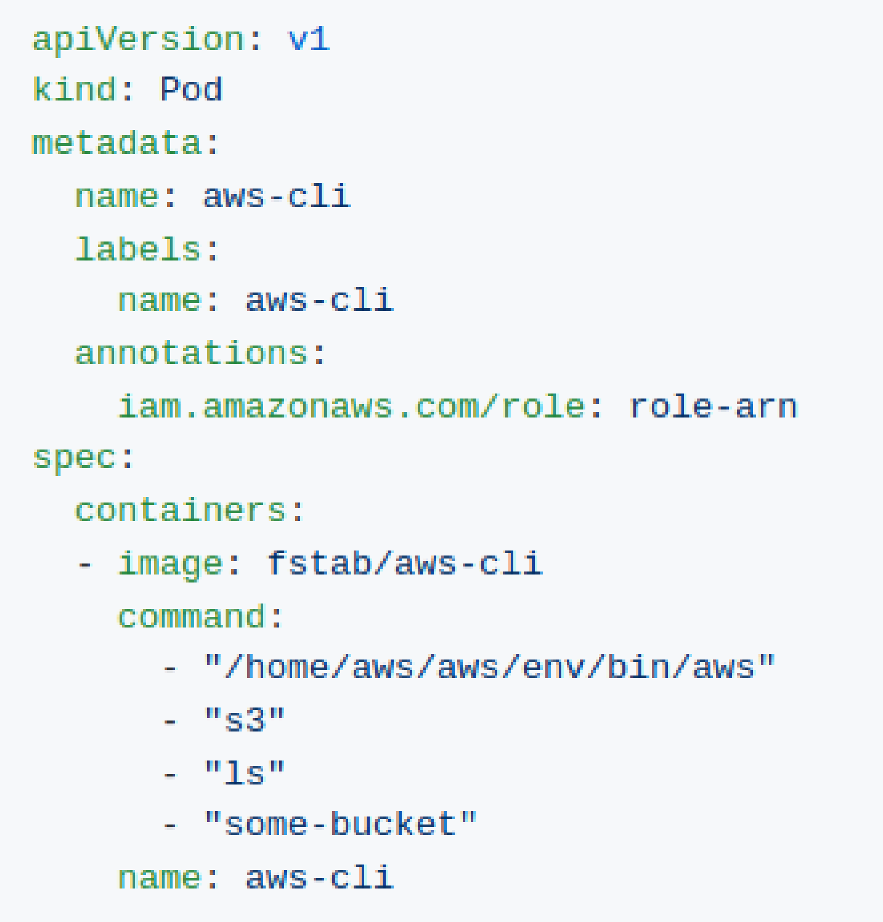
  - https://github.com/kubernetes-sigs/aws-alb-ingress-controller

AWS-specific cluster addons

- **alb-ingress-controller**
  - stops “controlling” an ALB if you add non-existing services
    → can bring down other services behind the same ALB

AWS-specific cluster addons

- **kube2iam**
  - deployed as daemon set
  - intercepts traffic to EC2 metadata service at http://169.254.169.254/ via iptables and serves its own metadata service for pods
  - https://github.com/jtblin/kube2iam



```
apiVersion: v1
kind: Pod
metadata:
  name: aws-cli
  labels:
    name: aws-cli
  annotations:
    iam.amazonaws.com/role: role-arn
spec:
  containers:
  - image: fstab/aws-cli
    command:
      - "/home/aws/aws/env/bin/aws"
      - "s3"
      - "ls"
      - "some-bucket"
    name: aws-cli
```

AWS-specific cluster addons

- **kube2iam**
  - allows to assign IAM roles to your pods
  - whitelist allowed pod roles per namespace via annotation
  - EC2 instance role needs to be allowed to assume all pod roles

Worker node management

- **provisioning auto scaling groups (ASGs)**
  - via CloudFormation
  - one stack per ASG
  - one ASG per
    - instance type
    - launch type (on-demand / spot)
    - availability zone
    - service type (RTC / non-RTC, special kernel settings)

→ huge matrix of ASGs

Worker node management

- **provisioning auto scaling groups (ASGs)**
  - description of the matrix of all required combinations in YAML file
  - read and excuted by python script
  - still missing:
    - automated roll-over of instances
    - automated deletion of unneeded ASGs

Worker node management

- **cluster-autoscaler**
  - another cluster-addon
  - supports multiple cloud providers
  - scales ASGs up & down according to number of pending pods and underutilized nodes
  - can be installed per-namespace to support fine-granular scaling behavior

  - https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler

Worker node management

- **cluster-autoscaler**
    - beware of the defaults:
        - stops working for some time if some actions fail
        - can scale up all ASGs to their maximum number of instances
    - don’t deploy it on spot instances!

Q&A

Thank you.