**AWS EKS** Application Deployment Architecture/ Implementation

*Kiran Sahoo, Jaswanth Kumar Jonnalagadda*

*Amazon Web Services*

*July 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

[Overview: 2](#_Toc18394379)

[End to End Implementation Architecture: 2](#_Toc18394380)

[Amazon EKS: 2](#_Toc18394381)

[Cluster Administration: 2](#_Toc18394382)

[Cluster Creation Automation: 2](#_Toc18394383)

[Cluster Autoscaling: 3](#_Toc18394384)

[Pod Autoscaling 3](#_Toc18394385)

[Configuring Cluster Storage: 3](#_Toc18394386)

[EFS-Provisioner 3](#_Toc18394387)

[Configuring Load Balancer 3](#_Toc18394388)

[Application load balancer: 3](#_Toc18394389)

[Configuring Cluster Multi-Tenancy: 4](#_Toc18394390)

[**Multi-tenancy policy enforcement** 4](#_Toc18394391)

[**Resource quotas** 5](#_Toc18394392)

[**Calico** 5](#_Toc18394393)

[EKS Deployment Automation: 6](#_Toc18394394)

[VPC Creation: 6](#_Toc18394395)

[Cluster Creation: 6](#_Toc18394396)

[Autoscaling: 6](#_Toc18394397)

[**Horizontal POD Autoscaler**: 7](#_Toc18394398)

[**Cluster Autoscaler**: 7](#_Toc18394399)

[ALB Ingress Installer: 7](#_Toc18394400)

[Tenant Authorization: 8](#_Toc18394401)

[Application Deployment Automation: 9](#_Toc18394402)

[Code Pipeline: 9](#_Toc18394403)

[**Architecture**: 9](#_Toc18394404)

# **Overview:**

This document provides a walkthrough of EKS

# **End to End Implementation Architecture**:

# **Amazon EKS**:

Amazon Elastic Container Service for Kubernetes (Amazon EKS) makes it easy to deploy, manage, and scale containerized applications using [**Kubernetes**](https://aws.amazon.com/kubernetes/) on AWS.

Amazon EKS runs the Kubernetes management infrastructure across multiple AWS availability zones to eliminate a single point of failure. Amazon EKS is certified Kubernetes conformant so that one can use existing tooling and plugins from partners and the Kubernetes community. Applications running on any standard Kubernetes environment are fully compatible and can be easily migrated to Amazon EKS.

# **Cluster Administration**:

## Cluster Creation Automation:

Script is being provided for automating creation of EKS cluster. It leverages eksctl. eksctl is a simple CLI tool for creating clusters on EKS . It is written in Go and uses CloudFormation.

Please refer to eksctl.io for further info. A sample eksclustercreation.yaml is attached for reference.

Please refer to Readme.md file of the deployment script for further instructions.

## Cluster Autoscaling:

Cluster auto scaler the default K8s component that can be used to perform scaling nodes in a cluster.

It automatically increases the size of an Auto Scaling group so that pods have a place to run. And it attempts to remove idle nodes, that is, nodes with no running pods.

Please refer to Readme.md file of the deployment script for further instructions in installing cluster auto scaler.

## Pod Autoscaling

HPA use metrics server. Metrics Server is a cluster wide aggregator of resource usage data. These metrics will drive the scaling behavior of the [deployments](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/). Metrics Server usage is part of eks cluster creation script. Please ensure to Have horizontalPodautoscaler

## **Configuring Cluster Storage**:

### EFS-Provisioner

The efs-provisioner allows you to mount EFS storage as PersistentVolumes in kubernetes. It consists of a container that has access to an AWS [EFS](https://aws.amazon.com/efs/) resource. The container reads a configmap which contains the EFS filesystem ID, the AWS region and the name you want to use for your efs-provisioner. This name will be used later when you create a storage class. Please refer to the following link for further info

<https://github.com/kubernetes-incubator/external-storage/tree/master/aws/efs>

## **Configuring Load Balancer**

### Application load balancer:

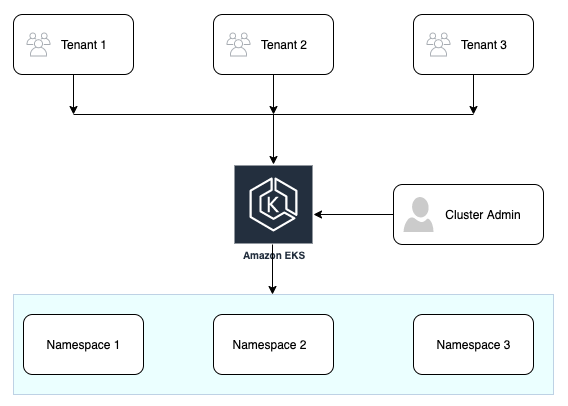
The [AWS ALB Ingress controller](https://github.com/kubernetes-sigs/aws-alb-ingress-controller) is a controller that triggers the creation of an [ALB](https://aws.amazon.com/elasticloadbalancing/features/#Details_for_Elastic_Load_Balancing_Products) and the necessary supporting AWS resources whenever a Kubernetes user declares an Ingress resource on the cluster. The Ingress resource uses the ALB to route HTTP[s] traffic to different endpoints within the cluster.

Please refer to Readme.md file of the deployment script under ALB\_Ingress\_installer

for further instructions.

## **Configuring Cluster Multi-Tenancy**:

Separate each tenant and their Kubernetes resources into their own [namespaces](https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/). Use [policies](https://kubernetes.io/docs/concepts/policy/) to enforce tenant isolation. Policies are usually scoped by namespace and can be used to restrict API access, to constrain resource usage, and to restrict what containers are allowed to do.



### **Multi-tenancy policy enforcement**

IAM roles will be used for managing authentication and authorization of tenants. Tenant’s user account (attached to a namespace) will be mapped to an IAM role in aws-auth file and the tenants account will be mapped to a namespace in RBAC yaml as shown below.

---

kind: Role

apiVersion: rbac.authorization.k8s.io/v1

metadata:

namespace: tenantonenamespace

name: tenantonenamespace-full-right

rules:

- apiGroups: ["\*"]

resources: ["\*"]

verbs: ["\*"]

- apiGroups: ["\*"]

resources: ["\*"]

verbs: ["\*"]

---

kind: RoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: tenantonenamespace-full-right-binding

namespace: tenantonenamespace

subjects:

- kind: User

name: admin-tenantonenamespace

apiGroup: rbac.authorization.k8s.io

roleRef:

kind: Role

name: tenantonenamespace-full-right

apiGroup: rbac.authorization.k8s.io

### **Resource quotas**

Resource quotas manage the amount of resources used by the objects in a namespace. You can set quotas in terms of CPU and memory usage, or in terms of object counts. Resource quotas let you ensure that no tenant uses more than its assigned share of cluster resources.

**Dedicated nodes with taints and tolerations**

Node taints are another way to control workload scheduling. You can use node taints to reserve specialized Nodes for use by certain tenants.

To dedicate a [node pool](https://cloud.google.com/kubernetes-engine/docs/concepts/node-pools) to a certain tenant, apply a taint with effect: "NoSchedule" to the node pool. Then only pods with a corresponding toleration can be scheduled to nodes in the node pool.

Please refer to the eksclustercreation.yaml under ClusterCreationInstaller folder for declarative way of adding Taints to Nodes.

### **Calico**

Calico enables networking and network policy in Kubernetes clusters

# **EKS Deployment Automation**:

With the help of the attached scripts following steps can be automated.

1. VPC creation compatible for EKS
2. EKS cluster creation
3. Enabling autoscaling
4. Deploying Ingress controller
5. Tenant Authorization

## **VPC Creation**:

Create a VPC that is compatible for EKS deployment with the script **EKS-VPC.yaml**

Upload the script in CloudFormation console and provide all the necessary parameters to create the VPC that is compatible with EKS.

## **Cluster Creation**:

Update the **eksClusterCreation.yml** file with cluster name, region, VPC id, Subnet id’s and respective CIDR ranges.

Run the **eksclusterinstaller.py** script to create the cluster.

**cmd**: python eksclusterinstaller.py

This script will deploy an EKS cluster with suggested name and in the suggested region with 4 private worker nodes and one regular node. The nodes are deployed with necessary taints and tolerations.

## **Autoscaling**:

**Automatic scaling in K8s comes in two forms**:

1. **Horizontal Pod Autoscaler**
2. **Cluster Autoscaler**

Metrics Server is a cluster-wide aggregator of resource usage data. These metrics will drive the scaling behavior of the deployments. We will deploy the metrics server using Helm configured in a previous module.

### **Horizontal POD Autoscaler**:

HPA scales the pods in a deployment or replica set. It is implemented as a K8s API resource and a controller. The controller manager queries the resource utilization against the metrics specified in each HorizontalPodAutoscaler definition. It obtains the metrics from either the resource metrics API (for per-pod resource metrics), or the custom metrics API.

Metrics Server and HPA are deployed as a part of **eksclusterinstaller.py** script.

### **Cluster Autoscaler**:

Cluster Autoscaler is the default K8s component that can be used to perform pod scaling as well as scaling nodes in a cluster. It automatically increases the size of an Auto Scaling group so that pods have a place to run. And it attempts to remove idle nodes, that is, nodes with no running pods.

Automate cluster autoscaling with the **eksclusterautoscale\_installer.py** script.

Update the files cluster\_autoscaler\_tenantone.yml, cluster\_autoscaler\_tenanttwo.yml files with the respective cluster autoscaling groups from AWS console. Update the **asg\_param.ini** file with workernodes instance profile names and cluster autoscaler files

Once the parameter file is updated run the python script with the following command  
**cmd:** python eksclusterautoscale\_installer.py

This python script will attach autoscaling policy to the worker nodes instance profile, applies autoscaling policies for the worker nodes.

## **ALB Ingress Installer**:

The AWS ALB Ingress Controller for Kubernetes is a controller that triggers the creation of an Application Load Balancer and the necessary supporting AWS resources whenever an Ingress resource is created on the cluster with the kubernetes.io/ingress.class: alb annotation. The Ingress resource uses the ALB to route HTTP or HTTPS traffic to different endpoints within the cluster.

Automated deployment of alb ingress controller can be achieved with **alb\_ingress\_contoller\_installer.py** script.

Update the **ingress\_param.ini** with workernodes instance profile names and save the file.

Also update the **alb-ingress-controller.yaml** file with EKS cluster name, VPC id, region and save the file.

Run the python script to deploy the ingress controller with the command

**cmd:** python alb\_ingress\_contoller\_installer.py

This script will attach the IAM policy necessary to enable ingress deployment, apply RBAC for the cluster necessary to enable ingress ALB and deploy the alb ingress controller.

## **Tenant Authorization**:

This script with authorize a tenant with an IAM role. i.e., to have a tenant deploy an application in the respective namespace only but not into any other namespace.

Tenant Isolation is achieved through namespaces.

This script creates an IAM role for authentication and a RBAC role to scope the API calls allowed. The IAM role is mapped to the user in aws-auth and kube-config file

Please ensure the following prerequisites are achieved before running the script

1) Namespace should exist for the tenant.

2) Update the tenant\_RBAC.yaml file with the namespace,username and role name.

3) Update the RABC rules in tenant\_RBAC.yaml as needed by the tenant application.

3) Update the account\_id property in tenant.ini file with the account number of the tenant

After updating the tenant.ini and Tenant\_RBAC.yaml files run the python script **Tenant\_Authorization.py** with the command

**cmd:** python Tenant\_Authorization.py

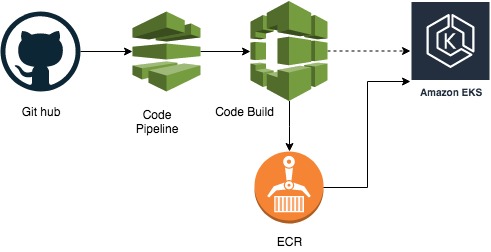
Script will create an IAM role with the same name of namespace, attach an IAM policy to the IAM role created, add account as trusted entity, Applying Role and RoleBinding to the cluster, update configmap/aws-auth with the new Role and attached the role to aws-auth configmap

# **Application Deployment Automation:**

## **Code Pipeline**:

AWS Code Pipeline is a fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates. Code Pipeline automates the build, test, and deploy phases of your release process every time there is a code change, based on the release model you define. This enables you to rapidly and reliably deliver features and updates

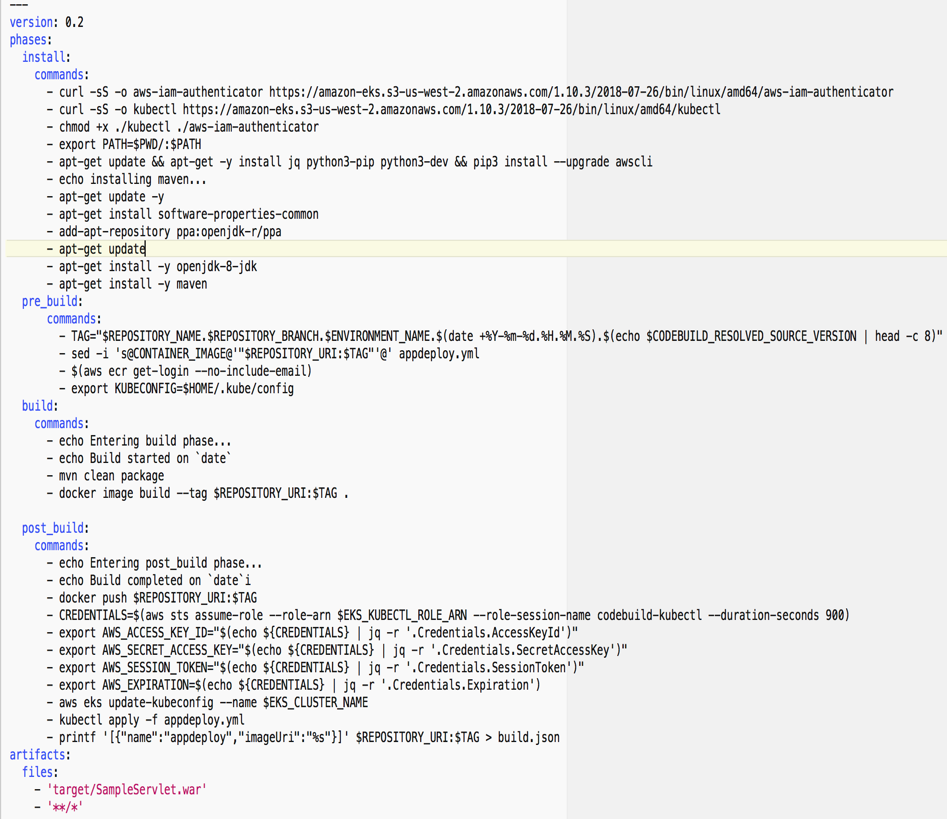
### **Architecture**:



To move the application from code repository to EKS cluster, implement AWS code pipeline integrated with Code Build and ECR. One can deploy a complete code pipeline with a cloud formation [template](https://github.com/jonnalagadda35153/eksworkingapp/blob/master/codepipeline.yaml).

**NOTE**: For the template please replace the placeholders with appropriate parameter values

To build the docker image from the code repository and push the built image to ECR repository to deploy the same on the deployed EKS cluster we integrate code build with [**builspec.yml**](https://github.com/jonnalagadda35153/eksworkingapp/blob/master/buildspec.yml) file



Part of script to download all the dependencies and command build the application

Commands to be executed before building the image

Commands to build the docker image form the app build

Command after build, pushing the image to ECR, deploying the app updates on to the cluster

**NOTE:**

1. Provide the namespace role created as a part of tenant authorization to be used for the code pipeline. So that a tenant will deploy the application in its respective namespace.
2. Update **“EKS\_KUBECTL\_ROLE\_ARN”** in the post build phase to be actual role arn of the tenant authorization role.