**AWS EKS** Application Deployment Architecture/ Implementation

*Jaswanth Kumar/Kiran Sahoo*

*Amazon Web Services*

*Sep 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: 3](#_Toc18403581)

[Amazon EKS: 3](#_Toc18403582)

[Cluster Administration: 3](#_Toc18403583)

[Cluster Creation Automation: 3](#_Toc18403584)

[Cluster Autoscaling: 3](#_Toc18403585)

[Pod Autoscaling 4](#_Toc18403586)

[Configuring Cluster Storage: 4](#_Toc18403587)

[EFS-Provisioner 4](#_Toc18403588)

[Configuring Load Balancer 4](#_Toc18403589)

[Application load balancer: 4](#_Toc18403590)

[Configuring Cluster Multi-Tenancy: 5](#_Toc18403591)

[Tenant Authorization 5](#_Toc18403592)

[Resource quotas 6](#_Toc18403593)

[Dedicated nodes For Tenants with NodeSelectors /Taints and Tolerations 6](#_Toc18403594)

[Calico 7](#_Toc18403595)

[Application Deployment Automation: 7](#_Toc18403596)

[Code Pipeline: 7](#_Toc18403597)

[Architecture: 8](#_Toc18403598)

# **Overview:**

## **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.

This document captures the following

1. Scripts and instructions for Automating provisioning of EKS Cluster
2. Scripts and instructions for Auto scaling (Cluster and Pod Autoscaler)
3. Scripts and instructions for Deploying ALB
4. Scripts and instructions for Configuring Cluster Storage (EFS)
5. Tenant Authentication/Authorization using IAM roles
6. Tenant Isolation using Namespaces/IAMRoles/Taint&Tolerations/NodeSelectors
7. Deployment Automation of Tenant Code using Code Pipeline/GitHub
8. Implement POD Network Policy using Calico

# **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 under Cluster\_Creation\_Installer 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 under Cluster\_Auto\_Scaling\_installer folder 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 deployment is part of eks cluster creation script.

Please ensure to have HorizontalPodAutoscaler in the deployment yaml. Please refer to the sample deployment files under Deployment\_Files folder for further information.

# **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>

Sample yaml file is provided for creating Persistent Volumes under Deployment\_Files folder.

# **Configuring Load Balancer**

## Application load balancer:

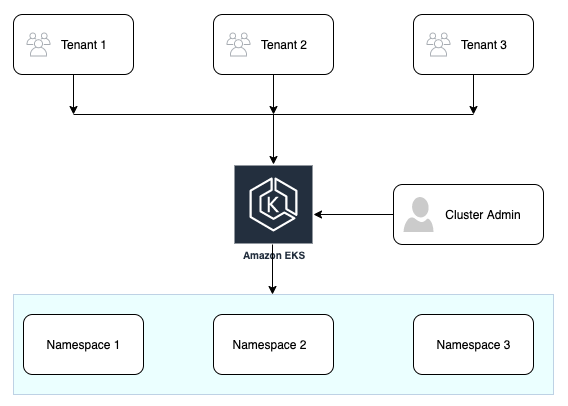
The [AWS ALB Ingress controller](https://github.com/kubernetes-sigs/aws-alb-ingress-controller) is a controller that triggers the creation of an [ALB](https://aws.amazon.com/elasticloadbalancing/features/#Details_for_Elastic_Load_Balancing_Products) and the necessary supporting AWS resources whenever 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.



## **Tenant Authorization**

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 user 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

Please refer to folder Tenant\_Authorization\_Scripts for instructions and scripts for automating Tenant Authorization/Authentication.

## **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. Please refer to sample tenantnamespace yaml under deployment\_files for defining resource quotas.

**Dedicated nodes For Tenants with NodeSelectors /Taints and Tolerations**

Node selectors can be used to assign pods to specific node. Please refer to cluster creation yaml for defining node labels and example deployment file for assigning pods to nodes based on labels.

Node taints are another way to control workload scheduling. 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 and sample deployment yaml for defining Tolerations.

## **Calico**

Calico enables networking and network policy in Kubernetes clusters. Calico provides the following advantages when running in Amazon Web Services (AWS):

* **Network Policy for Containers**: Calico provides fine-grained network security policy for individual containers.
* **No Overlays**: Within each VPC subnet Calico doesn’t need an overlay, which means high performance networking for your containers.

Apply the Calico manifest from the [aws/amazon-vpc-cni-k8s GitHub project](https://github.com/aws/amazon-vpc-cni-k8s" \t "_blank). This creates the daemon sets in the kube-system namespace.

* kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/config/v1.2/calico.yaml

A sample Networking policy for restricting access to Pods is shown below.

kind: NetworkPolicy

apiVersion: networking.k8s.io/v1

metadata:

name: default-deny

spec:

podSelector:

matchLabels: {}

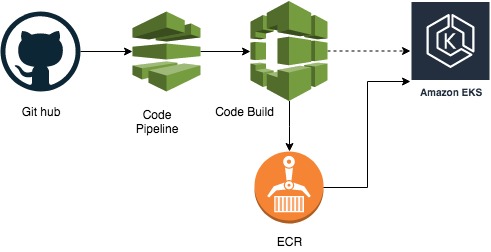
Further info can be found under <https://docs.projectcalico.org/v3.7/reference/public-cloud/aws>

# **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

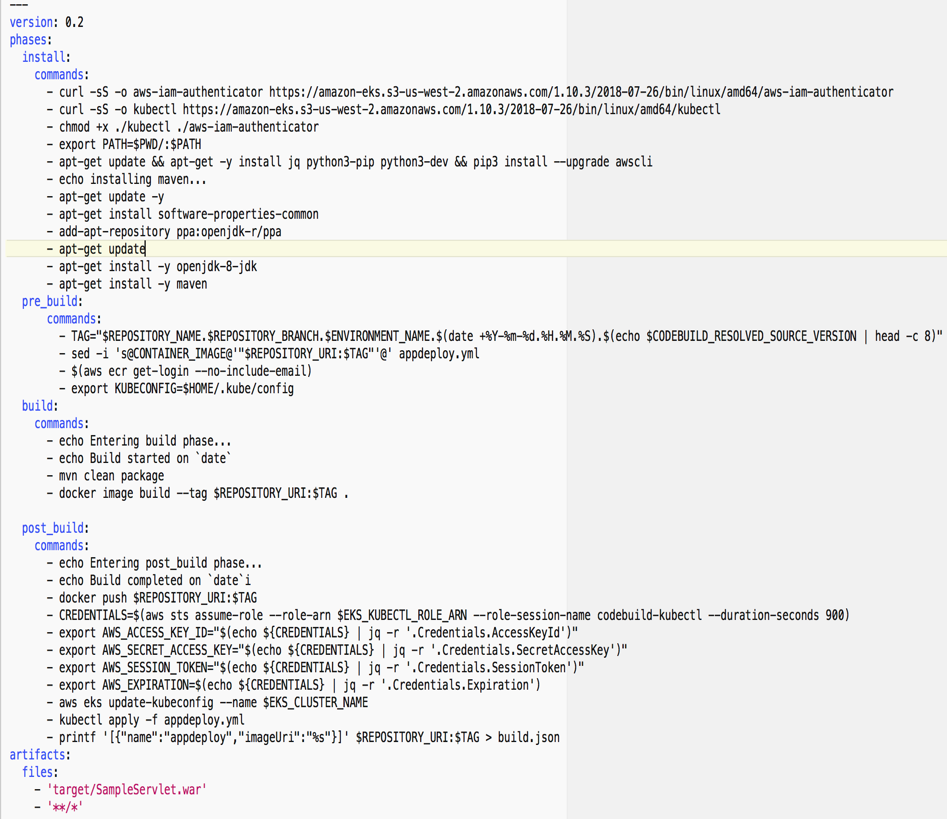
## **Architecture**:



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. This is being provided as part of the code release

**NOTE**: For the CF Template please replace the placeholders with appropriate parameter value .

**buildspec.yml.** file will build the docker image from the tenant github code repository ,push the built image to ECR repository and deploy the same on the deployed EKS cluster.



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.
3. Scripts/CF template and instructions is provided as part of the Code release and can be found Deployment\_Automation folder.