AWS EKS Kubernetes - Masterclass | DevOps, Microservices

Kalyan Reddy Daida
Docker - Fundamentals

• Docker Registry or Docker Hub
  • A Docker registry stores Docker images.
  • Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default.
  • We can even run our own private registry.
  • When we use the docker pull or docker run commands, the required images are pulled from our configured registry.
  • When we use the docker push command, our image is pushed to our configured registry.

Docker Registry (Docker Hub)

Docker Daemon

Docker Host

Containers

Images

Docker Client (My Desktop or Docker Host)

- docker pull nginxdemos/hello
- docker run -p 82:80 -d nginxdemos/hello
Kubernetes - Imperative & Declarative

Kubernetes Fundamentals

Imperative

- kubectl
- Pod
- ReplicaSet
- Deployment
- Service

Declarative

- YAML & kubectl
- Pod
- ReplicaSet
- Deployment
- Service
EKS Storage
EBS CSI Driver

Drawbacks of EBS CSI for MySQL DB
- Complex setup to achieve HA
- Complex Multi-Az support for EBS
- Complex Master-Master MySQL setup
- Complex Master-Slave MySQL setup
- No Automatic Backup & Recovery
- No Auto-Upgrade MySQL

EKS Cluster

REST API
POD
ReplicaSet
Deployment (UserMgmt)

UserMgmt – NodePort Service

MySQL – ClusterIP Service

POD
ReplicaSet
Deployment (mysql)

Storage Class
Persistent Volume Claim
Config Map
Deployment

NodePort Service
Deployment
Environment Variables
StatefulSets
Environment Variables
Volumes
Volume Mounts
ClusterIP Service


AWS Elastic Block Store - EBS

Users
AWS RDS Database

用户

EKS Cluster

 deployment

 ReplicaSet

 Deployment (UserMgmt)

 MySQL – ExternalName Service

 NoePort Service

 Deployment

 Environment Variables

 ExternalName Service

 Amazon RDS

 High Availability

 Backup & Recovery

 Read Replicas

 Metrics & Monitoring

 Automatic Upgrades

 Multi-AZ Support

AWS EKS Network Design
With EKS Workload & RDS Database
AWS EKS Network Design With EKS Workload RDS & ELB Classic Load Balancer

StackSimplify
© Kalyan Reddy Daida
AWS EKS Network Design

With

EKS Workload

RDS & ELB

Network Load Balancer
AWS EKS Network Design With EKS Workload RDS & ELB Application Load Balancer & Route53 Ingress & External-DNS

Users

Amazon Route 53

HTTPS URLs
https://apps.kubeoncloud.com/usermgmt/users

AWS Certificate Manager

AWS EKS

Network Design

With

EKS Workload

RDS & ELB

Application Load Balancer & Route53

Ingress & External-DNS

HTTP → HTTPS

User Management – Application Load Balancer Service (Ingress)

/app1/*

/app2/*

Ingress /*

NAT gateway

EC2 Worker Node-1

app1- NodePort SVC

app2- NodePort SVC

Pod

ReplicaSet

Deployment: app1

Deployment: app2

Pod

ReplicaSet

Deployment: UMS

MySQL – External Name Service

EKS Private NodeGroup

Amazon RDS DB

Amazon RDS DB

SSL Redirect

external-dns

AWS Certificate Manager

Amazon Route 53

HTTPS URLS

https://apps.kubeoncloud.com/usermgmt/users

HTTP → HTTPS

User Management – Node Port Service

EKS Private NodeGroup

Amazon RDS DB

SSL

StackSimplify

© Kalyan Reddy Daida
EKS Deployment Options - Mixed

AWS Cloud

VPC

EKS Cluster

Auto Scaling group

EC2 Instance

EKS Public Managed Node Group

EKS Private Managed Node Group

Public subnet

Availability Zone: us-east-1a

Private subnet

Availability Zone: us-east-1b

Public subnet

Availability Zone: us-east-1a

Private subnet

Availability Zone: us-east-1b

EKS Fargate

Fargate Profile

Fargate EC2 Instance

EKS Fargate Profiles
Stages in Release Process

- **Source**
  - Continuous integration

- **Build**

- **Test**

- **Production**
  - Continuous delivery
  - Continuous deployment
  - Infrastructure as code
AWS Developer Tools or Code Services

Source:
- AWS CodeCommit

Build:
- AWS CodeBuild

Test:
- AWS CodeBuild + Third Party

Deploy:
- CodeBuild
- kubectl

Monitor:
- AWS CloudWatch
  - Container Insights

AWS CodePipeline
AWS Developer Tools or Code Services

Source: AWS CodeCommit
Build: AWS CodeBuild
Test: AWS CodeBuild + Third Party
Deploy: CodeBuild + kubectl
Monitor: AWS CloudWatch Container Insights

Continuous integration
Continuous delivery
Continuous deployment
Microservices

User Management Microservice
- Create User API
- List Users API
- Delete User API
- Health Status API

Notification Microservice
- Send Notification API
- Health Status API

SMTP Server

API Developer
Or
API User

Postman Client

End User

Users DB

Email
Microservices Deployment on AWS EKS

AWS Cloud

VPC

Public subnet

Availability Zone: us-east-1a

NAT gateway

Private subnet

EC2 Worker Node - 1

EC2 Worker Node - 2

Amazon RDS DB

Ingress

Application Load Balancer

UMS – Ingress Service

EKS Cluster

End User

Email

Simple Email Service (SES)

AWS Certificate Manager

Amazon Route 53

Users

https://ums.kubeoncloud.com/usermgmt/user

ums.kubeoncloud.com

services.kubeoncloud.com

Microservices Deployment on AWS EKS

Public subnet

Availability Zone: us-east-1a

NAT gateway

Private subnet

EC2 Worker Node

MySQL – External Name Service

Notification – ClusterIP Service

UMS – NodePort Service

Notification Microservice Deployment

NS Pod

SMTP – External Name Service

EKS Managed Node Groups

EC2 Worker Node

MySQL – External Name Service

Notification – ClusterIP Service

NS Pod

smtp – External Name Service

UMS – NodePort Service

MySQL – External Name Service

Notification – ClusterIP Service

NS Pod
Kubernetes – DaemonSets

Worker Node - 1

Worker Node - 2

Kubernetes Cluster

AWS X-Ray

UMS LoadBalancer Service

UMS Deployment

 DaemonSet

XRay ClusterIP Service

XRay pod

ReplicaSet

UMS POD

UMS POD

UMS POD

UMS POD

© Kalyan Reddy Daida

StackSimplify
AWS X-Ray – Service Map

Client

User-Management-Microservice
AWS::EKS::Container
avg. 4ms
10 t/min

V1-Notification-Microservice
AWS::EKS::Container
avg. 9ms
2 t/min
AWS X-Ray - Traces

Timeline | Raw data
--- | ---
Method | Response | Duration | Age | ID
GET | 200 | 11.0 ms | 3.5 min (2020-07-14 09:38:42 UTC) | 1-5f0d7d22-372ae5e68602f9376576f3b

Trace Map

Client → User-Management-Microservice → V1-Notification-Microservice

<table>
<thead>
<tr>
<th>Name</th>
<th>Res.</th>
<th>Duration</th>
<th>Status</th>
<th>Duration (ms)</th>
<th>Details</th>
</tr>
</thead>
</table>
| User-Management-Microservice | 200 | 11.0 ms | ✔ | 11.0 ms | GET services.kubeoncloud.com/usermgmt/notification...
| getNotificationAppInfo | - | 9.0 ms | ✔ | 9.0 ms | Remote: GET ...:8096/notification/vray
| notification-clusterip-service | 200 | 6.0 ms | ✔ | 6.0 ms | GET notification-clusterip-service:8096/notification...
| V1-Notification-Microservice | 200 | 3.0 ms | ✔ | 3.0 ms | GET notification-clusterip-service:8096/notification...
| notificationAppVersionService | - | 0.0 ms | ✔ | 0.0 ms |
Microservices – Canary Deployments

Client -> User-Management-Microservice (avg. 18ms, 19 t/min) -> V1-Notification-Microservice (avg. 4ms, 7 t/min) -> V2-Notification-Microservice (avg. 5ms, 7 t/min)
How HPA works?

1. Query for Metrics
2. Calculate the Replica’s
3. Scale the app to desired replicas

Metrics Server
Pod - 1
Pod - 2
Pod - N

Horizontal Pod Autoscaler
Kubernetes Cluster

Deployment
ReplicaSet
Replication Controller
StatefulSet

Application

This control loop is executed every 15 seconds

15 secs
We can control multiple AWS services from the command line and automate them through scripts.

- `eksctl` is used for creating & deleting clusters on AWS EKS.
- We can even create, autoscale and delete node groups.
- We can even create fargate profiles using `eksctl`.
- In short, it is VERY VERY POWERFUL tool for managing EKS clusters on AWS.

`kubectl` allows us to control Kubernetes clusters and objects.
AWS EKS Cluster
AWS EKS – Core Objects

**EKS Cluster**

- **EKS Control Plane**
  - Contains Kubernetes Master components like etcd, kube-apiserver, kube-controller. It’s a managed service by AWS

- **Worker Nodes & Node Groups**
  - Group of EC2 Instances where we run our Application workloads

- **Fargate Profiles (Serverless)**
  - Instead of EC2 Instances, we run our Application workloads on Serverless Fargate profiles

- **VPC**
  - With AWS VPC we follow secure networking standards which will allow us to run production workloads on EKS.
How does EKS work?

1. Provision an EKS cluster
   - EKS automatically deploys Kubernetes masters

2. Deploy worker nodes
   - Add worker nodes to your EKS cluster

3. Connect to EKS
   - Point your favorite Kubernetes tooling at your EKS cluster

4. Run Kubernetes apps
   - Deploy your Kubernetes applications to your EKS cluster
# EKS Cluster – Core Objects Detailed

**EKS Control Plane**

1. EKS runs a single tenant Kubernetes control plane for each cluster, and control plane infrastructure is **not shared** across clusters or AWS accounts.
2. This control plane consists of at least two API server nodes and three etcd nodes that run across three Availability Zones within a Region.
3. EKS **automatically detects and replaces unhealthy** control plane instances, restarting them across the Availability Zones within the Region as needed.

---

**Worker Nodes & Node Groups**

1. Worker machines in Kubernetes are called nodes. These are EC2 Instances.
2. EKS worker nodes run in our AWS account and connect to our cluster's control plane via the **cluster API server endpoint**.
3. A node group is **one or more EC2 instances** that are deployed in an EC2 Autoscaling group.
4. All instances in a node group must
   1. Be the **same instance type**
   2. Be running the **same AMI**
   3. Use the **same EKS worker node IAM role**
Fargate Profiles

1. AWS Fargate is a technology that provides on-demand, right-sized compute capacity for containers.
2. With Fargate, we no longer have to provision, configure, or scale groups of virtual machines to run containers.
3. Each pod running on Fargate has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another pod.
4. AWS specially built Fargate controllers that recognizes the pods belonging to fargate and schedules them on Fargate profiles.
5. We will see more in our Fargate learning section.

VPC

1. EKS uses AWS VPC network policies to restrict traffic between control plane components to within a single cluster.
2. Control plane components for a EKS cluster cannot view or receive communication from other clusters or other AWS accounts, except as authorized with Kubernetes RBAC policies.
3. This secure and highly-available configuration makes EKS reliable and recommended for production workloads.
Kubernetes Architecture
Kubernetes - Architecture

Master

- Kube Controller Manager
- Cloud Controller Manager
- kube-apiserver
- etcd

Worker Node

- Kubelet
- Kube-Proxy
- Container Runtime (Docker)

Worker Node

- Kubelet
- Kube-Proxy
- Container Runtime (Docker)
### Kubernetes Architecture - Master

- **kube-apiserver**
  - It acts as *front end* for the Kubernetes control plane. It *exposes* the Kubernetes API
  - Command line tools (like `kubectl`), Users and even Master components (scheduler, controller manager, etcd) and Worker node components like (Kubelet) everything *talk* with API Server.

- **etcd**
  - Consistent and highly-available *key value store* used as Kubernetes’ *backing store* for all cluster data.
  - It *stores* all the masters and worker node information.

- **kube-scheduler**
  - Scheduler is responsible for distributing containers across multiple nodes.
  - It watches for newly created Pods with no assigned node, and selects a node for them to run on.
Kubernetes Architecture - Master

- **kube-controller-manager**
  - Controllers are responsible for noticing and responding when nodes, containers or endpoints go down. They make decisions to bring up new containers in such cases.
  - **Node Controller**: Responsible for noticing and responding when nodes go down.
  - **Replication Controller**: Responsible for maintaining the correct number of pods for every replication controller object in the system.
  - **Endpoints Controller**: Populates the Endpoints object (that is, joins Services & Pods)
  - **Service Account & Token Controller**: Creates default accounts and API Access for new namespaces.
**cloud-controller-manager**
- A Kubernetes control plane component that embeds *cloud-specific control logic*.
- It only runs controllers that are *specific* to your cloud provider.
- **On-Premise** Kubernetes clusters will not have this component.
- **Node controller:** For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- **Route controller:** For setting up *routes* in the underlying cloud infrastructure
- **Service controller:** For creating, updating and deleting cloud provider *load balancer*
Kubernetes Architecture – Worker Nodes

- **Kubelet**
  - Kubelet is the *agent* that runs on every node in the cluster.
  - This agent is *responsible* for making sure that containers are running in a Pod on a node.

- **Kube-Proxy**
  - It is a *network proxy* that runs on each node in your cluster.
  - It maintains *network rules* on nodes.
  - In short, these network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

- **Container Runtime**
  - Container Runtime is the *underlying software* where we run all these Kubernetes components.
  - We are using Docker, but we have other runtime options like rkt, container-d etc.
Kubernetes - Architecture

Master
- Kube Controller Manager
- Cloud Controller Manager
- kube-apiserver
- etcd

Worker Node
- Kubelet
- Kube-Proxy
- Container Runtime (Docker)

Worker Node
- Kubelet
- Kube-Proxy
- Container Runtime (Docker)
EKS Kubernetes - Architecture

Master
- EKS Controller Manager
- Fargate Controller Manager
- kube-apiserver
- etcd
- kube-scheduler

Worker Node - 1
- Kubelet
- Kube-Proxy
- Container Runtime (Docker)

Worker Node - 2
- Kubelet
- Kube-Proxy
- Container Runtime (Docker)

EKS Control Plane

EKS Node Group

© Kalyan Reddy Daida
Kubernetes Fundamentals
Pod, ReplicaSet, Deployment & Service
A POD is a single instance of an Application. A POD is the smallest object, that you can create in Kubernetes.

A ReplicaSet will maintain a stable set of replica Pods running at any given time. In short, it is often used to guarantee the availability of a specified number of identical Pods.

A Deployment runs multiple replicas of your application and automatically replaces any instances that fail or become unresponsive. Rollout & rollout changes to applications. Deployments are well-suited for stateless applications.

A service is an abstraction for pods, providing a stable, so-called virtual IP (VIP) address. In simple terms, service sits in front of a POD and acts as a load balancer.
Kubernetes - Imperative & Declarative

Kubernetes Fundamentals

Imperative
- kubectl
  - Pod
  - ReplicaSet
  - Deployment
  - Service

Declarative
- YAML & kubectl
  - Pod
  - ReplicaSet
  - Deployment
  - Service
Kubernetes

POD
With Kubernetes our core goal will be to deploy our applications in the form of containers on worker nodes in a k8s cluster.

Kubernetes does not deploy containers directly on the worker nodes.

Container is encapsulated in to a Kubernetes Object named POD.

A POD is a single instance of an application.

A POD is the smallest object that we can create in Kubernetes.
Kubernetes - POD

- PODs generally have one to one relationship with containers.
- To scale up we **create** new POD and to scale down we **delete** the POD.
Kubernetes – PODs

- We cannot have multiple containers of same kind in a single POD.
- Example: Two NGINX containers in single POD serving same purpose is not recommended.
Kubernetes – Multi-Container Pods

• We can have multiple containers in a single POD, provided they are not of same kind.

• Helper Containers (Side-car)
  • Data Pullers: Pull data required by Main Container
  • Data pushers: Push data by collecting from main container (logs)
  • Proxies: Writes static data to html files using Helper container and Reads using Main Container.

• Communication
  • The two containers can easily communicate with each other easily as they share same network space.
  • They can also easily share same storage space.

• Multi-Container Pods is a rare use-case and we will try to focus on core fundamentals.
Kubernetes

PODs

Demo
Kubernetes

Services - NodePort
Kubernetes – Service - NodePort

• We can **expose an application** running on a set of **PODs** using different types of Services available in k8s.
  • ClusterIP
  • NodePort
  • LoadBalancer

• **NodePort Service**
  • To access our application **outside of k8s cluster**, we can use NodePort service.
  • Exposes the Service on each **Worker Node's IP** at a static port (nothing but NodePort).
  • A **ClusterIP** Service, to which the **NodePort** Service routes, is **automatically** created.
  • Port Range **30000-32767**

http://<Worker-Node-IP>:<NodePort>
Kubernetes

POD & NodePort Service Demo
Kubernetes
ReplicaSets
Kubernetes - ReplicaSets

- High Availability or Reliability
- Scaling
- Load Balancing
- Labels & Selectors
Kubernetes – ReplicaSet

- A ReplicaSet’s purpose is to maintain a stable set of replica Pods running at any given time.
- If our application crashes (any pod dies), replicaset will recreate the pod immediately to ensure the configured number of pods running at any given time.

Reliability
Or
High Availability

![Diagram of Kubernetes ReplicaSet]

- **POD N**
- **POD N**
- **POD N**

Kubernetes Cluster

Worker Node

ReplicaSet
• Load Balancing

• To avoid overloading of traffic to single pod we can use load balancing.

• Kubernetes provides pod load balancing out of the box using Services for the pods which are part of a ReplicaSet.

• Labels & Selectors are the key items which ties all 3 together (Pod, ReplicaSet & Service), we will know in detail when we are writing YAML manifests for these objects.
Kubernetes – ReplicaSet

• Scaling

• When load become too much for the number of existing pods, Kubernetes enables us to easily scale up our application, adding additional pods as needed.

• This is going to be seamless and super quick.
Kubernetes
ReplicaSets Demo
Kubernetes
Deployments
Kubernetes – Deployments

- Worker Node
- Kubernetes Cluster
- POD
- ReplicaSet
- Deployment
- Service
Kubernetes - Deployment

- Create a Deployment to rollout a ReplicaSet
- Updating the Deployment
- Rolling Back a Deployment
- Scaling a Deployment
- Pausing and Resuming a Deployment
- Deployment Status
- Clean up Policy
- Canary Deployments
Kubernetes Deployments Demo
Kubernetes

Services
Kubernetes - Services

- **ClusterIP**
  - Used for communication between applications inside k8s cluster (Example: Frontend application accessing backend application)

- **NodePort**
  - Used for accessing applications outside of k8s cluster using Worker Node Ports (Example: Accessing Frontend application on browser)

- **LoadBalancer**
  - Primarily for Cloud Providers to integrate with their Load Balancer services (Example: AWS Elastic Load Balancer)

- **Ingress**
  - Ingress is an advanced load balancer which provides Context path based routing, SSL, SSL Redirect and many more (Example: AWS ALB)

- **externalName**
  - To access externally hosted apps in k8s cluster (Example: Access AWS RDS Database endpoint by application present inside k8s cluster)
Services

Kubernetes Cluster

Frontend App – NodePort or LoadBalancer or Ingress Service

Deployment (app=frontend)

Backend App - ClusterIP Service

Deployment (app=Backend)

DB – ExternalName Service

AWS Cloud

AWS RDS Database

Users
Services Demo

Users

http://<workernode-public-ip>:<NodePort>/hello
Kubernetes

YAML Basics
YAML Basics

- YAML is **not a Markup Language**
- YAML is used to **store information** about different things
- We can use YAML to **define key, Value pairs** like variables, lists and objects
- YAML is very similar to **JSON** (Javascript Object Notation)
- YAML primarily focuses on **readability** and **user friendliness**
- YAML is designed to be **clean and easy to read**
- We can define YAML files with two different extensions
  - abc.yml
  - abc.yaml
YAML Basics

- YAML Comments
- YAML Key Value Pairs
- YAML Dictionary or Map
- YAML Array / Lists
- YAML Spaces
- YAML Document Separator
EKS Storage

- **In-Tree EBS Provisioner**
  - Legacy
  - Will be deprecated soon

- **EBS CSI Driver**
  - Latest & Greatest available today & in Beta release & ready for production use
  - As on today, **not supported** on AWS EKS Fargate (Serverless)
  - Allows EKS Clusters to **manage lifecycle** of EBS Volumes for persistent storage, EFS File systems & FSx for Luster File systems
  - Supported for k8s 1.14 & later

- **EFS CSI Driver**

- **FSx for Luster CSI**
  - Supported for k8s 1.16 & later

**CSI** means Container Storage Interface
AWS Elastic Block Store - Introduction

• EBS provides block level storage volumes for use with EC2 & Container instances.

• We can mount these volumes as devices on our EC2 & Container instances.

• EBS volumes that are attached to an instance are exposed as storage volumes that persist independently from the life of the EC2 or Container instance.

• We can dynamically change the configuration of a volume attached to an instance.

• AWS recommends EBS for data that must be quickly accessible and requires long-term persistence.

• EBS is well suited to both database-style applications that rely on random reads and writes, and to throughput-intensive applications that perform long, continuous reads and writes.
EKS Storage

EBS CSI Driver

- AWS Elastic Block Store - EBS
- MySQL – ClusterIP Service
- Storage Class
- Persistent Volume Claim
- Config Map
- Deployment
- Environment Variables
- Volumes
- Volume Mounts
- ClusterIP Service

© Kalyan Reddy Daida
EKS Storage
EBS CSI Driver

- Users
- NodePort Service
- Deployment
- Environment Variables

- AWS Elastic Block Store - EBS
- Storage Class
- Persistent Volume Claim
- Config Map
- Deployment

- MySQL - ClusterIP Service
- MySQL
- POD ReplicaSet
- Deployment (mysql)

- Deployment (UserMgmt)
- UserMgmt - NodePort Service
- REST API
- POD ReplicaSet

- Environment Variables
- Volumes
- Volume Mounts
- ClusterIP Service

Users can access the UserMgmt service through the following URL:

AWS EKS Storage
EBS CSI Driver
Important k8s Concepts for Application Deployments
EKS Storage
EBS CSI Driver
Probes

Liveness Probe
Kubelet uses liveness probes to know when to restart a container
Liveness probes could catch a deadlock, where an application is running, but unable to make progress and restarting container helps in such state

Readiness Probe
Kubelet uses readiness probes to know when a container is ready to accept traffic
When a Pod is not ready, it is removed from Service load balancers based on this readiness probe signal.

Startup Probe
Kubelet uses startup probes to know when a container application has started
Firstly this probe disables liveness & readiness checks until it succeeds ensuring those pods don’t interfere with app startup.
This can be used to adopt liveness checks on slow starting containers, avoiding them getting killed by the kubelet before they are up and running.

Options to define Probes
Check using Commands
/bin/sh –c nc -z localhost 8095
Check using HTTP GET Request
httpget path:/health-status
Check using TCP
tcpSocket Port: 8095
Kubernetes
Namespaces
Namespaces - Introduction

- Namespaces are also called Virtual clusters in our physical k8s cluster
- We use this in environments where we have many users spread across multiple teams or projects
- Clusters with tens of users ideally don’t need to use namespaces
- Benefits
  - Creates isolation boundary from other k8s objects
  - We can limit the resources like CPU, Memory on per namespace basis (Resource Quota).
Namespaces

Namespace Manifest - Declarative

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: dev3
```

Namespace Manifest - Imperative

```
kubectl create namespace dev3
```
Limit Range Manifest

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: default-cpu-mem-limit-range
  namespace: dev3
spec:
  limits:
  - default:
      memory: "512Mi"
      cpu: "500m"
  defaultRequest:
    memory: "256Mi"
    cpu: "300m"
  type: Container
```

Namespace: dev

Namespace: qa

Namespace: staging
Resource Quota

Resource Quota Manifest

```yaml
apiVersion: v1
type: ResourceQuota
metadata:
  name: ns-resource-quota
  namespace: dev
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
    pods: "5"
    configmaps: "5"
    persistentvolumeclaims: "5"
    secrets: "5"
    services: "5"
```
AWS EKS & RDS Database
EKS Storage
EBS CSI Driver

Drawbacks of EBS CSI for MySQL DB
- Complex setup to achieve HA
- Complex Multi-Az support for EBS
- Complex Master-Master MySQL setup
- Complex Master-Slave MySQL setup
- No Automatic Backup & Recovery
- No Auto-Upgrade MySQL

Users


EKS Cluster

NodePort Service
Deployment
Environment Variables
StatefulSets
Environment Variables
Volumes
Volume Mounts
ClusterIP Service

Deployment
Deployment (mysql)
Deployment (UserMgmt)
POD ReplicaSet
POD ReplicaSet
POD
MySQL – ClusterIP Service
MySQL – NodePort Service
REST API
AWS Elastic Block Store - EBS

Storage Class
Persistent Volume Claim
Config Map
Deployment

StackSimplify
© Kalyan Reddy Daida
EKS Cluster

POD

ReplicaSet

Deployment (UserMgmt)

UserMgmt – NodePort Service

MySQL – ExternalName Service

NoePort Service

Deployment

Environment Variables

ExternalName Service

High Availability

Backup & Recovery

Read Replicas

Metrics & Monitoring

Automatic Upgrades

Multi-AZ Support

AWS

RDS Database

Users


Amazon RDS

High Availability

Backup & Recovery

Read Replicas

Metrics & Monitoring

Automatic Upgrades

Multi-AZ Support
AWS EKS Network Design With EKS Workload & RDS Database
AWS
Elastic Load Balancing
Overview
Elastic Load Balancing

- Classic Load Balancer
- Network Load Balancer
- Application Load Balancer

https://aws.amazon.com/elasticloadbalancing/features/#compare
AWS EKS & RDS & ELB
Classic Load Balancer
AWS EKS Network Design
With EKS Workload
RDS & ELB
Classic Load Balancer
Elastic Load Balancing & Classic Load Balancer

Network Load Balancer & Application Load Balancer

AWS EKS & RDS & ELB

Network Load Balancer
AWS EKS Network Design
With EKS Workload
RDS & ELB
Network Load Balancer

*NLB DNS URL*

*Public subnet*

*Private subnet*

*EC2 Worker Node 1*

*EC2 Worker Node 2*

*ReplicaSet Pod*

*MySQL – External Name Service*

*AWS Cloud VPC*

*Availability Zone: us-east-1a*

*Availability Zone: us-east-1b*

*Network Load Balancer Service*

*User Management – Network Load Balancer Service*

*NAT gateway*

*Amazon RDS DB*

*EKS Private NodeGroup*
AWS EKS & RDS & ELB

Application Load Balancer
How Ingress Works?
### k8s ClusterRole

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app.kubernetes.io/name: alb-ingress-controller
    name: alb-ingress-controller
rules:
  - apiGroups:
      - v1
    resources:
      - configmaps
      - endpoints
      - events
      - ingresses
      - ingresses/status
      - services
      - pods/status
    verbs:
      - create
      - get
      - list
      - update
      - watch
      - patch
  - apiGroups:
      - v1
    resources:
      - nodes
      - pods
      - secrets
      - services
      - namespaces
    verbs:
      - get
      - list
      - watch
```

### k8s ServiceAccount

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/name: alb-ingress-controller
    name: alb-ingress-controller
    namespace: kube-system
```

### k8s ClusterRoleBinding

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    app.kubernetes.io/name: alb-ingress-controller
    name: alb-ingress-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: alb-ingress-controller
subjects:
  - kind: ServiceAccount
    name: alb-ingress-controller
    namespace: kube-system
```

### AWS IAM Policy

```json
```
AWS EKS
RDS & ELB
Application Load Balancer
Ingress Controller Basics
AWS EKS Network Design With EKS Workload RDS & ELB Application Load Balancer

- Public subnet
- Application Load Balancer
- User Management – Application Load Balancer Service (Ingress)
- NAT gateway
- EC2 Worker Node-1
- EC2 Worker Node-2
- MySQL – External Name Service
- Amazon RDS DB
- Deployment: UMS

User Management – Node Port Service

Ingress

Availability Zone: us-east-1a

Availability Zone: us-east-1b

Users

http://ALB-DNS-URL/usermgmt/users
AWS EKS
RDS & ELB
Application Load Balancer
Ingress Context Path based Routing
AWS EKS Network Design

With

EKS Workload
RDS & ELB
Application Load Balancer

Context path based Routing
AWS EKS

RDS & ELB

Application Load Balancer

Ingress SSL
**AWS EKS Network Design**

With

EKS Workload

RDS & ELB

Application Load Balancer

Ingress SSL

**HTTP URLs**

http://ssldemo.kubeoncloud.com/usermgmt/users

http://ssldemo.kubeoncloud.com/app1/index.html

http://ssldemo.kubeoncloud.com/app2/index.html

**HTTPS URLs**

https://ssldemo.kubeoncloud.com/usermgmt/users

https://ssldemo.kubeoncloud.com/app1/index.html

https://ssldemo.kubeoncloud.com/app2/index.html
AWS EKS
RDS & ELB
Application Load Balancer & Route53
Ingress & External-DNS
AWS EKS
Fargate Profiles
Serverless
What is Fargate?

- Fargate is a **Serverless compute platform** for containers on AWS
- Fargate provides **on-demand, right-sized compute capacity** for containers
- EKS integrates Kubernetes with Fargate by using **controllers** that are built by AWS using the **upstream, extensible model** provided by Kubernetes.
- These controllers run as part of the **EKS managed Kubernetes control plane** and are responsible for **scheduling** native Kubernetes pods onto Fargate.
- The **Fargate controllers** include a **new scheduler** that runs alongside the **default Kubernetes scheduler** in addition to several mutating and validating admission controllers.
- When we start a pod that **meets the criteria for running on Fargate**, the Fargate controllers running in the cluster recognize, update, and **schedule the pod onto Fargate**.
AWS EKS on Fargate

Bring existing pods
- We don’t need to change our existing pods
- Fargate works with existing workflows and services that run on Kubernetes

Production Ready
- Launch pods easily.
- Easily run pods across Azs for HA
- Each pod runs in an isolated compute environment

Rightsized and Integrated
- Only pay for resources you need to run your pods
- Includes native AWS integrations for networking and security
We can deploy EC2 Managed Node Groups in both public and private subnets of a VPC.
EKS Deployment Options – Only Fargate

Pods running on Fargate are only supported on private subnets.
EKS Deployment Options - Mixed

AWS Cloud

EKS Cluster

VPC

Availability Zone: us-east-1a
- Public subnet
- EC2 Instance
- Auto Scaling group
- EKS Public Managed Node Group

Availability Zone: us-east-1b
- Public subnet
- EC2 Instance
- Auto Scaling group
- EKS Public Managed Node Group

Availability Zone: us-east-1a
- Public subnet
- EC2 Instance
- Auto Scaling group
- EKS Private Managed Node Group

Availability Zone: us-east-1b
- Public subnet
- EC2 Instance
- Auto Scaling group
- EKS Private Managed Node Group

EKS Managed Node Groups

EKS Cluster

EKS Fargate

Fargate EC2 Instance
- Fargate Profile

EKS Fargate Profiles

© Kalyan Reddy Daida
### EKS Fargate vs Managed vs Unmanaged Nodes

<table>
<thead>
<tr>
<th></th>
<th>Fargate</th>
<th>Managed nodes</th>
<th>Unmanaged nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Units of work</strong></td>
<td>Pod</td>
<td>Pod and EC2</td>
<td>Pod and EC2</td>
</tr>
<tr>
<td><strong>Unit of charge</strong></td>
<td>Pod</td>
<td>EC2</td>
<td>EC2</td>
</tr>
<tr>
<td><strong>Host lifecycle</strong></td>
<td>There is no visible host</td>
<td>AWS (SSH is allowed)</td>
<td>Customer</td>
</tr>
<tr>
<td><strong>Host AMI</strong></td>
<td>There is no visible host</td>
<td>AWS vetted AMIs</td>
<td>Customer BYO</td>
</tr>
<tr>
<td><strong>Host : Pods</strong></td>
<td>1 : 1</td>
<td>1 : many</td>
<td>1 : many</td>
</tr>
</tbody>
</table>
EKS Fargate Considerations

• There are many considerations we need to be aware of before we decide our Kubernetes workloads to run on Fargate.

• Documentation Link

• https://docs.aws.amazon.com/eks/latest/userguide/fargate.html
AWS EKS

Fargate Profiles

Basics
EKS Deployment Options - Mixed

- **AWS Cloud**
- **VPC**
  - Availability Zone: us-east-1a
    - Public subnet
      - EC2 Instance
    - Private subnet
      - EC2 Instance
  - Availability Zone: us-east-1b
    - Public subnet
      - EC2 Instance
    - Private subnet
      - EC2 Instance

- **EKS Cluster**
- **EKS Managed Node Groups**
  - EKS Public Managed Node Group
    - Auto Scaling group
  - EKS Private Managed Node Group
    - Auto Scaling group

- **EKS Fargate**
- **EKS Fargate Profiles**

© Kalyan Reddy Daida

StackSimplify
Fargate Profiles can be deployed to EKS Cluster only when we have at least one private subnet.
AWS EKS Fargate Profiles Basics

Users

Amazon Route 53

NAT gateway

AWS Certificate Manager

Fargate Profiles

AWS EKS

Ingress

SSL Redirect

external-dns

Application Load Balancer Service (Ingress)

User Management

Application Load Balancer

Private subnet

Fargate Node-1

Fargate Node-2

ReplicaSet

Deployment: app1

Pod

app1- NodePort Service

Fargate Profile: fp-dev

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*

Ingress

Amazon Route 53

https://fpdev.kubeoncloud.com/app1/index.html

AWS Certificate Manager

fpdev.kubeoncloud.com

App1

ReplicaSet

N

Pod

N

Pod

Deployment: app1

SSL

NAT gateway

/*
Ingress with Cross Namespaces is not supported as on today.

- **Private subnet**
  - App1 - NodePort Service
  - NS: ns-app1

- **Public subnet**
  - App1 - Ingress
    - /app1/*

- **Private subnet**
  - App2 - NodePort Service
  - NS: ns-app2

- **Public subnet**
  - /app2/*

- **Public subnet**
  - /ums/*

- **Private subnet**
  - /ums/*

- **Public subnet**
  - /ums/*

- **Availability Zone: us-east-1a**
  - EKS Managed Node Groups
  - EKS Fargate

- **Availability Zone: us-east-1b**
  - EKS Fargate Profiles
  - EKS Cluster

- **MySQL – ExternalName Service**
  - NS: ns-ums

- **Amazon RDS DB**
  - App1 - NodePort Service
  - NS: ns-app1

- **Amazon Route 53**
  - App2 - NodePort Service
  - NS: ns-app2

- **Auto Scaling group**
  - EKS Private Managed Node Group

- **EKS Fargate**
  - UMS - NodePort Service
  - NS: ns-ums

- **Users**
  - app1.kubeoncloud.com
  - app2.kubeoncloud.com
  - ums.kubeoncloud.com

- **EKS Deployment – Mixed – Ingress with Cross Namespaces**
AWS EKS

ECR

Elastic Container Registry

Fargate Profiles

Elastic Container Registry
Elastic Container Registry - ECR

• Elastic Container Registry (ECR) is a **fully-managed** Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images.

• ECR is integrated with Elastic Kubernetes Service (EKS), simplifying our development to production workflow.

• ECR **eliminates** the need to operate our own container repositories or worry about scaling the underlying infrastructure.

• ECR hosts our images in a **highly available** and scalable architecture, allowing us to reliably deploy containers for our applications.

• Integration with **AWS Identity and Access Management (IAM)** provides resource-level control of each repository.

• With Amazon ECR, there are **no upfront fees** or commitments. We pay only for the amount of data you store in your repositories and data transferred to the Internet.
Elastic Container Registry - ECR

• Benefits
  • Full managed
  • Secure
  • Highly Available
  • Simplified Workflow
How ECR Works?

Developer

Push Images

Docker Image

Elastic Container Registry - ECR

Pull Images

Pull Images and Run Containers

Docker Container

Elastic Container Service - ECS

Docker Container

Elastic Kubernetes Service - EKS

Docker Container

On-Premise
AWS EKS

&

AWS Developer Tools
StackSimplify

Kalyan Reddy Daida

Stages in Release Process

Source
- Check-in source code
- Peer review new code
- Pull Request process

Build
- Compile Code & build artifacts (war, jar, container images, Kubernetes manifest files)
- Unit Tests

Test
- Integration tests with other systems.
- Load Testing
- UI Tests
- Security Tests
- Test Environments (Dev, QA and Staging)

Production
- Deployment to production environments
- Monitor code in production to quickly detect errors
Stages in Release Process

- Source
- Build
- Test
- Production

- Continuous integration
- Continuous delivery
- Continuous deployment
- Infrastructure as code
Continuous Integration

- Automatically kick off a new release when new code is checked-in
- Build and test code in a consistent, repeatable environment
- Continually have an artifact ready for deployment
Continuous Delivery

- Automatically deploy new changes to staging environments for testing
- Deploy to production safely without affecting customers
- Deliver to customers faster
- Increase deployment frequency, and reduce change lead time and change failure rate
AWS Developer Tools or Code Services

Source
- AWS CodeCommit

Build
- AWS CodeBuild

Test
- AWS CodeBuild + Third Party

Deploy
- CodeBuild
- kubectl

Monitor
- AWS CloudWatch
- Container Insights

AWS CodePipeline
AWS Developer Tools or Code Services

Source
- AWS CodeCommit

Build
- AWS CodeBuild

Test
- AWS CodeBuild + Third Party

Deploy
- CodeBuild
- kubectl

Monitor
- AWS CloudWatch Container Insights

Continuous integration

Continuous delivery
- AWS CodePipeline

Continuous deployment
AWS CodeCommit
AWS CodeCommit - Introduction

- **Version Control Service** hosted by AWS
- We can privately store and manage documents, **source code**, and binary files
- **Secure & highly scalable**
- Supports standard functionality of **Git** (CodeCommit supports Git versions 1.7.9 and later.)
- Uses a **static user name and password** in addition to standard SSH.
CodeCommit – Integration with AWS Services

![Diagram showing integration of AWS services with CodeCommit](image-url)
CodeCommit - Steps

Developer

Local Git Repo

push

AWS Cloud

AWS CodeCommit
AWS CodeBuild
CodeBuild - Introduction

• CodeBuild is a **fully managed** build service in the cloud.
• Compiles our **source code**, runs **unit tests**, and produces **artifacts** that are ready to deploy.
• Eliminates the need to provision, manage, and scale our own build servers.
• It provides **prepackaged build environments** for the most popular programming languages and build tools such as Apache Maven, Gradle, and many more.
• We can also customize build environments in CodeBuild to use our own build tools.
• **Scales automatically** to meet peak build requests.
How to run CodeBuild?

How CodeBuild works?

1. AWS Management Console
2. Build project
3. Source code
4. Build environment
5. Amazon SNS, Amazon S3
6. Amazon CloudWatch Logs
7. AWS CodeBuild
8. AWS CodeBuild
CodeBuild - Steps

1. Developer commits code changes to the Local Git Repo.
2. Code is pushed to AWS CodeCommit.
3. AWS CodeBuild is triggered to build the code.
4. The build is uploaded to Simple Storage Service (S3).
AWS CodePipeline
CodePipeline - Introduction

• AWS CodePipeline is a continuous delivery service to model, visualize, and automate the steps required to release your software.

• Benefits
  • We can automate our release processes.
  • We can establish a consistent release process.
  • We can speed up delivery while improving quality.
  • Supports external tools integration for source, build and deploy.
  • View progress at a glance
  • View pipeline history details.
null
Continuous Delivery

Stack Simplify
Kalyan Reddy Daida
© Amazon
What are Microservices?
What are Microservices?

- **Microservices** - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable
  - Organized around business capabilities
  - Owned by a small team
Microservices - Benefits

- **Developer independence**: Small teams work in parallel and can iterate faster than large teams.
- **Isolation and resilience**: If a component dies, you spin up another while and the rest of the application continues to function.
- **Scalability**: Smaller components take up fewer resources and can be scaled to meet increasing demand of that component only.
- **Lifecycle automation**: Individual components are easier to fit into continuous delivery pipelines and complex deployment scenarios not possible with monoliths.
- **Relationship to the business**: Microservice architectures are split along business domain boundaries, increasing independence and understanding across the organization.
AWS EKS
Microservices Deployment

Elastic Load Balancing
Classic Load Balancer
Network Load Balancer
Application Load Balancer
Certificate Manager
Route53
Elastic Block Store
Amazon RDS

Fargate Profiles
Elastic Container Registry
Code Commit
Code Build
Code Pipeline
Simple Email Service
Microservices

User Management Microservice
- Create User API
- List Users API
- Delete User API
- Health Status API

Notification Microservice
- Send Notification API
- Health Status API

API Developer
Or
API User

Postman Client

End User

Email

Users DB

SMTP Server
Microservices Deployment on AWS EKS

- **AWS Cloud**
  - VPC
    - Public subnet
      - NAT gateway
    - Private subnet
      - Amazon RDS DB
  - EKS Cluster
    - Availability Zone: us-east-1a
    - EKS Managed Node Groups
      - EC2 Worker Node
      - Notification Microservice Deployment
      - Notification - ClusterIP Service
      - Notification - NodePort Service
      - MySQL - External Name Service
      - SMTP - External Name Service
      - UMS - Ingress Service
      - UMS - NodePort Service
      - Ingress Application Load Balancer
      - Amazon Route 53
      - Users
      - End User
      - Simple Email Service (SES)
      - EKS Application Load Balancer
      - https://ums.kubeoncloud.com/usermgmt/user
      - ums.kubeoncloud.com/services.kubeoncloud.com

© Kalyan Reddy Daida
StackSimplify
AWS EKS
Microservices Distributed Tracing
AWS X-Ray
• AWS X-Ray helps analyse and debug distributed applications built using microservices architecture.

• With X-Ray, we can understand how our application and its underlying services are performing to identify and troubleshoot the root cause of performance issues and errors.

• X-Ray provides an end-to-end view of requests as they travel through our application and shows a map of our application’s underlying components.

• We can also use X-Ray to analyse applications in development and in production, from simple three-tier applications to complex microservices applications consisting of thousands of services.
AWS X-Ray - Benefits

• Review request behavior
• Discover application issues
• Improve application performance
• Ready to use with AWS
• Designed for a variety of applications
AWS X-Ray – How it works?

How it works

**AWS X-Ray**
AWS X-Ray helps you analyze and debug modern applications built using microservices and serverless architecture and quantify customer impact.

**Collect traces**
Collect data about the request from each of the underlying application services it passes through.

**Record traces**
X-Ray combines the data gathered from each service into singular units called traces.

**View service map**
View the service map to see trace data such as latencies, HTTP statuses, and metadata for each service.

**Analyze issues**
Drill into the service showing unusual behavior to identify the root issue.
A **DaemonSet** ensures that all (or some) Nodes run a copy of a Pod.

- As nodes are **added** to the cluster, Pods are added to them.
- As nodes are **removed** from the cluster, those Pods are garbage collected.
- Deleting a DaemonSet will clean up the Pods it created.

Some typical uses of a DaemonSet are:

- running a **logs collection daemon** on every node (Example: fluentd)
- running a **node monitoring daemon** on every node (Example: cloudwatchagent)
- running an **application trace collection daemon** on every node (Example: AWS X-Ray)

In a **simple case**, one DaemonSet, covering all nodes, would be used for each type of daemon.

A **more complex setup** might use **multiple DaemonSets for a single type of daemon**, but with different flags and/or different memory and cpu requests for different hardware types.
Kubernetes – DaemonSets

- Kubernetes Cluster
- Worker Node - 1
- Worker Node - 2

- UMS Deployment
- ReplicaSet
- UMS Pod

- XRay ClusterIP Service
- XRay Pod

- UMS LoadBalancer Service

- AWS X-Ray
AWS X-Ray – Service Map

Client → **User-Management-Microservice** (avg. 4ms, 10 t/min) → **V1-Notification-Microservice** (avg. 9ms, 2 t/min)
AWS X-Ray - Traces
## AWS X-Ray - Traces

### Segment - User-Management-Microservice

<table>
<thead>
<tr>
<th>Overview</th>
<th>Resources</th>
<th>Annotations</th>
<th>Metadata</th>
<th>Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment ID</td>
<td>722d02f36eb066c4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parent ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>User-Management-Microservice</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Origin</td>
<td>AWS::EKS::Container</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start time</td>
<td>2020-07-14 09:38:42.647 (UTC)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>End time</td>
<td>2020-07-14 09:38:42.658 (UTC)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duration</td>
<td>11.0 ms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In progress</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Errors &amp; Faults</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fault</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request &amp; Response</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request url</td>
<td><a href="http://services.kubeoncloud.com/umgmt/notification-xray">http://services.kubeoncloud.com/umgmt/notification-xray</a></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request method</td>
<td>GET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request user_agent</td>
<td>PostmanRuntime/7.25.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request client_ip</td>
<td>49.206.222.64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request x_forwarded_for</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### CloudWatch Logs

```json
[
  {
    "log_group": "/aws/containerinsights//application"
  }
]
```

### EC2

- Availability zone: us-east-1b
- Instance ID: i-0247e0ae1264716b3
- Instance size: t3.medium
- Ami ID: ami-0b33458af4b31b8f

### X-Ray

- SDK version: 2.6.1
- SDK: X-Ray for Java

### Eks

- Pod: usermgmt-microservice-7598cb7f7cf-tkwck
- Containerid: 1eabf8732ab8303eb24a138b31f963566d5d113cd571ea965f5b7ad2dd9e34a
What are Canary Deployments?

- Canaries means **incremental** rollouts.
- With canaries, the **new version** of the application is slowly deployed to the Kubernetes cluster while getting a very small amount of **live traffic**.
- In short, a **subset of live users** are connecting to the **new version** while the rest are still using the **previous version**.
- Using canaries, we can detect **deployment issues very early** while they effect only a small subset of users.
- If we **encounter any issues with a canary**, the production version is still present, and **all traffic can simply be reverted to it**.
Microservices – Canary Deployments

Client → User-Management-Microservice

User-Management-Microservice

→ avg. 4 ms
→ 7 t/min
→ V1-Notification-Microservice
→ AWS::EKS::Container

→ avg. 5 ms
→ 7 t/min
→ V2-Notification-Microservice
→ AWS::EKS::Container

avg. 18 ms
19 t/min
Canary Deployments *out of box* on Kubernetes

**Downside**
We need to incrementally increase pods based on percentage distribution we need for canary.
Canary Deployments on Kubernetes with AWS App Mesh
AWS EKS
Autoscaling
Horizontal Pod Autoscaler
In a very simple note Horizontal Scaling means **increasing and decreasing the number of Replicas (Pods)**

HPA **automatically scales** the number of pods in a deployment, replication controller, or replica set, stateful set based on that resource's **CPU utilization**.

This can help our applications **scale out to meet increased demand or scale in when resources are not needed**, thus freeing up your worker nodes for other applications.

When we set a **target CPU utilization percentage**, the HPA scales our application in or out to try to meet that **target**.

HPA needs **Kubernetes metrics server** to verify CPU metrics of a pod.

*We do not need to deploy or install the HPA on our cluster to begin scaling our applications, its out of the box available as a **default** Kubernetes API resource.*
How HPA works?

1. Query for Metrics
2. Calculate the Replica’s
3. Scale the app to desired replicas

This control loop is executed every 15 seconds.
How is HPA configured?

HPA requires

- Scaling Metric: CPU Utilization
- Target Value - CPU = 50%
- Min Replicas = 2
- Max Replicas = 10

kubectl autoscale deployment demo-deployment --cpu-percent=50 --min=1 --max=10
AWS EKS
Autoscaling
Vertical Pod Autoscaler
Vertical Pod Autoscaler – VPA - Introduction

• VPA automatically adjusts the CPU and memory reservations for our pods to help "right size" our applications.
• This adjustment can improve cluster resource utilization and free up CPU and memory for other pods.

• Benefits
  • Cluster nodes are used efficiently, because Pods use exactly what they need.
  • Pods are scheduled onto nodes that have the appropriate resources available.
  • We don't have to run time-consuming benchmarking tasks to determine the correct values for CPU and memory requests.
  • Maintenance time is reduced, because the autoscaler can adjust CPU and memory requests over time without any action on your part.
Every pod submitted to the k8s cluster goes through this webhook automatically which checks whether a VerticalPodAutoscaler object is referencing this pod or one of its parents (a ReplicaSet, a Deployment, etc.)

Connects to the metrics-server in the cluster, fetches historical and current usage data (CPU and memory) for each VPA-enabled pod and generates recommendations for scaling up or down the requests and limits of these pods.

Runs every 1 minute. If a pod is not running in the calculated recommendation range, it evicts the currently running version of this pod, so it can restart and go through the VPA admission webhook which will change the CPU and memory settings for it, before it can start.
AWS EKS
Autoscaling
Cluster Autoscaler
Cluster Autoscaler - Introduction

- **Cluster Autoscaler** is a tool that **automatically adjusts** the size of a Kubernetes cluster when one of the following conditions is true:
  - There are pods that **failed to run** in the cluster due to **insufficient resources**.
  - There are nodes in the cluster that have been **underutilized** for an extended period of time and their pods can be placed on other existing nodes.
  - The Cluster Autoscaler **modifies** our worker node groups so that they **scale out** when we need more resources and **scale in** when we have underutilized resources.
Container Insights

• A fully managed observability service for monitoring, troubleshooting and alarming on our containerized applications.
• Container Insights to collect, aggregate, and summarize metrics and logs from our containerized applications and microservices.
• The metrics include utilization for resources such as CPU, memory, disk, and network.
• It also provides diagnostic information, such as container restart failures, to help us isolate issues and resolve them quickly.
• We can also set CloudWatch alarms on metrics that Container Insights collects.
• The metrics that Container Insights collects are available in CloudWatch automatic dashboards.
• We can analyze and troubleshoot container performance and logs data with CloudWatch Logs Insights.
Container Map

Container Resources

Performance Dashboards

Log Groups

Log Insights

Alarms

Developer or Operations User

CloudWatch

CloudWatch Agent DaemonSet

Fluentd DaemonSet

App Pod

App Pod

CW pod

CW pod

Fluentd pod

Fluentd pod

NGINX Deployment

ReplicaSet

Worker Node - 1

Worker Node - 2

Kubernetes Cluster

Load Balancer Service

Container Insights
CloudWatch Container Insights Map
Automatic Performance Dashboard

CPU Utilization

Memory Utilization

Network

Performance monitoring

EKS Pods

ECS Clusters

ECS Instances

ECS Services

ECS Tasks

EKS Clusters

EKS Namespaces

EKS Nodes

EKS Services

Reserved CPU Comp...

Reserved Memory Comp...

© Kalyan Reddy Daida

StackSimplify
THANK YOU