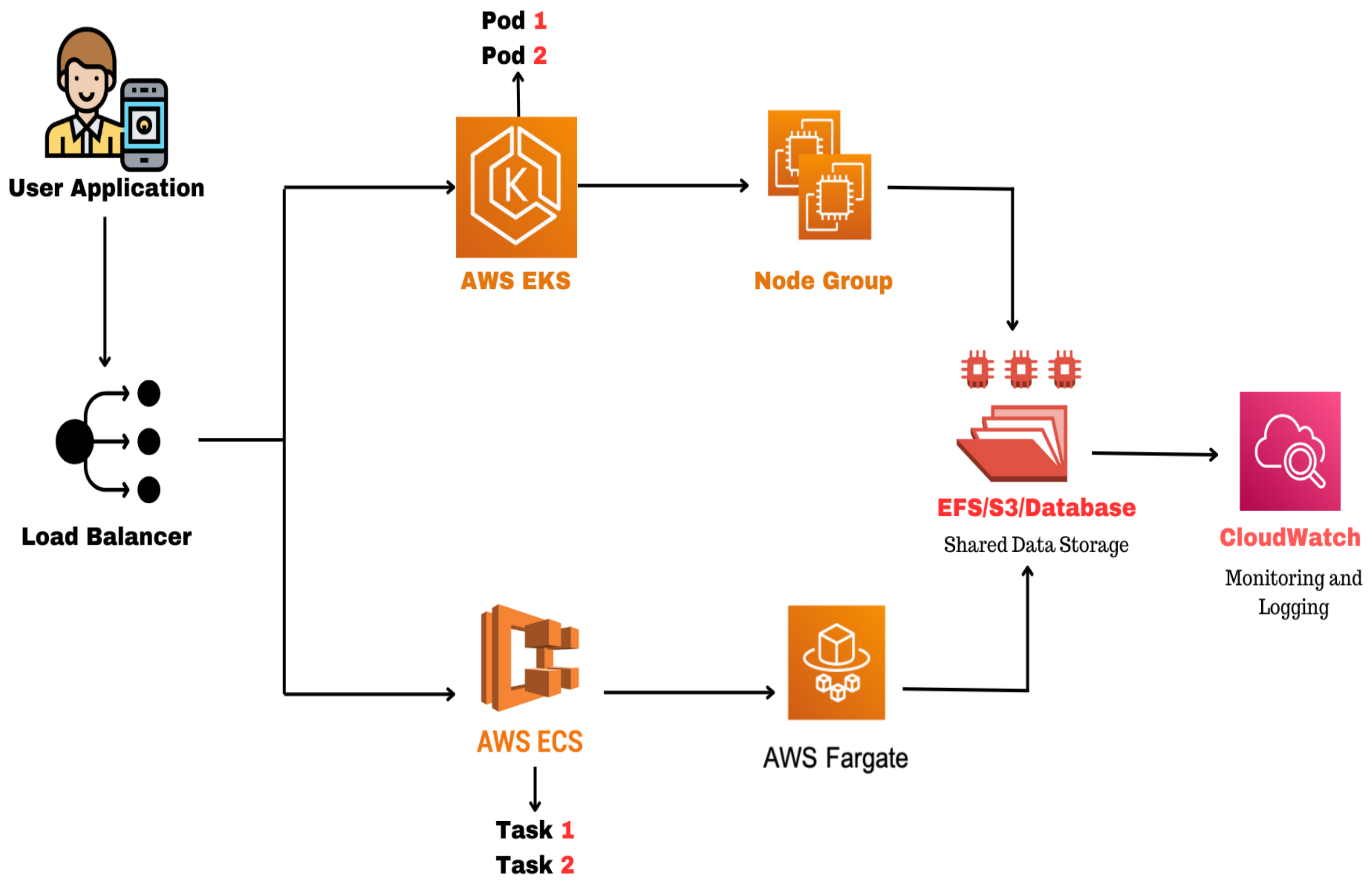
DevOps Shack

ECS and EKS: Orchestrating Containers on AWS

I. Elastic Container Service (ECS)

1. Introduction to Amazon ECS

Amazon Elastic Container Service (ECS) is a fully managed container orchestration service offered by AWS. It simplifies the deployment, management, and scaling of containerized applications, leveraging Docker containers. ECS integrates seamlessly with other AWS services like IAM, Elastic Load Balancing, and CloudWatch, making it ideal for a variety of workloads, including microservices, batch jobs, and machine learning.



Why Use Amazon ECS?

1. **Flexibility**: ECS supports both EC2-based and serverless (AWS Fargate) deployments.
2. **Simplified Management**: No need to maintain complex control plane infrastructure.
3. **Seamless AWS Integration**: Integrates with CloudWatch, Secrets Manager, and more.
4. **Cost Efficiency**: Pay only for what you use with Fargate or optimize costs with EC2.
5. **High Availability**: ECS automatically distributes tasks across availability zones.

2. Core Concepts of ECS

2.1 Clusters

Clusters are logical groupings of ECS resources, either EC2 instances or Fargate tasks. All tasks and services are deployed within a cluster.

2.2 Task Definitions

A task definition is a blueprint for ECS tasks, including container image, CPU, memory, ports, and networking details. It supports multiple container definitions.

2.3 Services

Services ensure that a specified number of tasks are continuously running. They manage Auto Scaling and integration with load balancers.

2.4 Launch Types

- **EC2 Launch Type**: Runs tasks on EC2 instances you manage.
- **Fargate Launch Type**: Serverless execution without managing infrastructure.

3. Setting Up Amazon ECS

3.1 Prerequisites

1. **AWS CLI**: Install and configure the AWS CLI.

```
aws configure
```

2. **Docker**: Install Docker to build and test images.

```
sudo apt update && sudo apt install -y docker.io
```

3. **IAM Role**: Create an IAM role with the **AmazonECSTaskExecutionRolePolicy**.

3.2 Creating an ECS Cluster

Step 1: Using the AWS Console

1. Navigate to the **ECS Dashboard**.
2. Click on **Clusters** > **Create Cluster**.
3. Select either EC2 or Fargate launch type.
4. Provide cluster details and create the cluster.

Step 2: Using the AWS CLI

```
aws ecs create-cluster --cluster-name my-cluster
```

3.3 Building and Pushing Docker Images

Step 1: Create a Dockerfile

```
FROM python:3.9
COPY app.py /app/app.py
WORKDIR /app
RUN pip install flask
CMD ["python", "app.py"]
```

Step 2: Test Locally

Build and run the image:

```
docker build -t flask-app .
docker run -d -p 5000:5000 flask-app
```

Step 3: Push to ECR

1. Authenticate Docker with ECR:

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin <account-id>.dkr.ecr.us-west-2.amazonaws.com
```

2. Create a repository:

```
aws ecr create-repository --repository-name flask-app
```

3. Push the image:

```
docker tag flask-app:latest <account-id>.dkr.ecr.us-west-2.amazonaws.com/flask-app:latest
```

docker push <account-id>.dkr.ecr.us-west-2.amazonaws.com/flask-app:latest

3.4 Creating a Task Definition

Step 1: JSON Definition

```
{
  "family": "flask-task",
  "containerDefinitions": [
    {
      "name": "flask-container",
      "image": "<account-id>.dkr.ecr.us-west-2.amazonaws.com/flask-app:latest",
      "memory": 512,
      "cpu": 256,
      "portMappings": [
        {
          "containerPort": 5000,
          "hostPort": 5000
        }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/flask-app",
          "awslogs-region": "us-west-2",
          "awslogs-stream-prefix": "ecs"
        }
      }
    }
  ]
}
```

Step 2: Register Task Definition

```
aws ecs register-task-definition --cli-input-json file://task-definition.json
```

3.5 Deploying a Service

Step 1: Create a Service

```
aws ecs create-service --cluster my-cluster \
  --service-name flask-service \
  --task-definition flask-task \
  --desired-count 2 \
  --launch-type FARGATE \
  --network-configuration 'awsvpcConfiguration={subnets=[subnet-123],securityGroups=[sg-456],assignPublicIp="ENABLED"}'
```

Step 2: Updating the Service

Scale the service to 5 tasks:

```
aws ecs update-service --cluster my-cluster \
 --service-name flask-service --desired-count 5
```

4. Monitoring ECS

4.1 CloudWatch Logs

Enable CloudWatch logging in the task definition:

```
"logConfiguration": {
 "logDriver": "awslogs",
 "options": {
   "awslogs-group": "/ecs/flask-app",
   "awslogs-region": "us-west-2",
   "awslogs-stream-prefix": "ecs"
 }
}
```

4.2 ECS Metrics

Monitor metrics like CPU and memory usage:

```
aws cloudwatch get-metric-data --metric-name CPUUtilization \
 --namespace AWS/ECS --dimensions Name=ClusterName,Value=my-cluster
```

5. ECS Auto Scaling

1. **Task Auto Scaling**: Adjust the number of running tasks dynamically.
2. **Cluster Auto Scaling**: Automatically increase or decrease EC2 instances.

Step 1: Configure Auto Scaling

Create scaling policies to trigger scaling based on metrics like CPU usage.

6. Networking in ECS

6.1 Load Balancing

Use an Application Load Balancer (ALB) to route traffic to ECS tasks.

```
aws elbv2 create-load-balancer --name flask-alb --subnets subnet-123 subnet-456
```

6.2 Security Groups

Define ingress and egress rules to control traffic.

7. Cost Optimization with ECS

1. **Fargate for Burst Workloads**: Use Fargate for unpredictable workloads to save costs.
2. **Spot Instances**: Use EC2 Spot Instances for cost-effective deployments.
3. **Auto Scaling**: Dynamically scale resources to match demand.

II. ELASTIC KUBERNATES SERVICE(EKS)

1. Introduction

Amazon Elastic Kubernetes Service (EKS) is a managed Kubernetes service provided by AWS. It simplifies the process of deploying, managing, and scaling Kubernetes clusters in the cloud. EKS removes the operational burden of maintaining the Kubernetes control plane, providing a reliable and secure environment for running containerized workloads.

Whether you're migrating from an on-premise Kubernetes deployment or starting fresh in the cloud, EKS provides flexibility, scalability, and deep integration with other AWS services.

Why Use Amazon EKS?

1. **Ease of Kubernetes Management**: AWS handles upgrades, patching, and scaling for the control plane.
2. **Flexibility**: EKS supports hybrid and multi-cloud environments through EKS Anywhere and AWS Outposts.
3. **Security**: Integrated with IAM for access control and AWS VPC for secure networking.
4. **Performance**: Offers seamless integration with AWS EC2, Fargate, and Auto Scaling.

2. Key Features of Amazon EKS

Managed Control Plane

AWS manages the highly available and secure Kubernetes control plane, allowing you to focus on deploying workloads.

- Automatic scaling for API servers and etcd.
- Multi-AZ redundancy ensures no single point of failure.

Integration with AWS Services

- **IAM**: Control access to Kubernetes resources.
- **CloudWatch**: Monitor and log your cluster's activity.
- **ECR**: Manage container images natively.

Support for Fargate

Run serverless Kubernetes workloads with AWS Fargate, eliminating the need to manage worker nodes.

3. Core Concepts in Amazon EKS

1. Kubernetes Cluster Architecture

- **Control Plane**: Hosted and managed by AWS, consisting of Kubernetes API servers, etcd, and core controllers.

- **Worker Nodes**: Hosted on EC2 or Fargate. These nodes run your containerized applications.

2. Networking

EKS uses the Amazon VPC Container Network Interface (CNI) plugin to assign private IPs from your VPC to Kubernetes pods.

- **Security Groups**: Control traffic at the instance and cluster level.
- **Load Balancers**: Use AWS ALB, NLB, or CLB for external traffic.

3. EKS Add-ons

EKS provides managed add-ons like:

- CoreDNS for service discovery.
- kube-proxy for communication between pods and nodes.
- Amazon VPC CNI for networking.

4. Detailed Steps for Setting Up Amazon EKS

4.1 Prerequisites

1. **AWS CLI**
   Install AWS CLI and configure credentials:

```
aws configure
```

2. **kubectl**
   Install Kubernetes CLI to interact with the cluster:

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

```
chmod +x kubectl
```

```
sudo mv kubectl /usr/local/bin/
```

3. **eksctl**
   Simplifies cluster setup:

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/download/latest_release/eksctl_$(uname -s)_$(uname -m).tar.gz" | tar xz -C /tmp
```

```
sudo mv /tmp/eksctl /usr/local/bin
```

4.2 Step-by-Step Cluster Creation

Step 1: Create an EKS Cluster

```
eksctl create cluster --name my-cluster --region us-west-2 \
  --nodegroup-name standard-workers --node-type t3.medium \
  --nodes 3 --nodes-min 1 --nodes-max 5 --managed
```

- **--name**: The name of your cluster.
- **--region**: AWS region for the cluster.
- **--nodegroup-name**: Name of the worker node group.

Step 2: Verify Cluster Creation

Check if the cluster is active:

```
kubectl get svc
```

Step 3: Deploy a Sample Application

Create a deployment YAML file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
```

```
  metadata:
    labels:
      app: nginx
  spec:
    containers:
    - name: nginx
      image: nginx:1.21
      ports:
      - containerPort: 80
```

Apply the deployment:

```
kubectl apply -f nginx-deployment.yaml
```

4.3 Scaling Kubernetes Workloads

Horizontal Pod Autoscaler (HPA)

Automatically scale pods based on CPU utilization:

```
kubectl autoscale deployment nginx-deployment \
  --cpu-percent=50 --min=1 --max=10
```

Cluster Autoscaler

Enable Cluster Autoscaler to dynamically add/remove EC2 instances:

1. Add IAM policy for the worker node:

```
eksctl utils associate-iam-oidc-provider \
  --region us-west-2 --cluster my-cluster --approve
```

2. Deploy Cluster Autoscaler:

```
kubectl apply -f cluster-autoscaler.yaml
```

5. Networking in EKS

Ingress with AWS ALB

1. Install the ALB Ingress Controller:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/main/docs/install/v2_2_0/full.yaml
```

2. Define an Ingress Resource:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
spec:
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: nginx-service
            port:
              number: 80
```

6. Observability in EKS

Logging with CloudWatch

Deploy the CloudWatch agent:

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-monitoring/main/cloudwatch-agent-daemonset.yaml
```

7. Monitoring with Prometheus and Grafana

Deploy Prometheus:

```
kubectl apply -f prometheus.yaml
```

Deploy Grafana and connect it to Prometheus:

```
kubectl apply -f grafana.yaml
```

Securing Amazon EKS

1. **RBAC**:
   Configure roles and permissions for Kubernetes users and services:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

2. **Pod Security Policies**:
   Limit container privileges:

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
```

```
spec:
  privileged: false
```

Differences Between Amazon ECS and Amazon EKS

Feature	Amazon ECS	Amazon EKS
Definition	A fully managed container orchestration service by AWS that works natively with AWS services.	A managed Kubernetes service that allows you to run Kubernetes clusters on AWS.
Underlying Technology	AWS proprietary container orchestration.	Kubernetes, an open-source container orchestration tool.
Complexity	Simpler to set up and manage, especially for AWS-native workloads.	More complex due to Kubernetes' steep learning curve.
Portability	Not portable across other cloud providers or on-premise environments.	Highly portable—can run Kubernetes workloads across multiple clouds or on-premises.
Management	Requires minimal management; AWS handles most tasks.	Requires more configuration and maintenance, even with AWS-managed nodes.
Integration	Tight integration with AWS services like CloudWatch, IAM, and Load Balancers.	Kubernetes-native but integrates with AWS services via custom controllers.
Flexibility	Less flexible than Kubernetes but easier to use.	Extremely flexible, supporting custom configurations and third-party integrations.
Networking	Simplified networking with built-in support for tasks and services.	Kubernetes networking model with CNI plugins like AWS VPC CNI.

Feature	**Amazon ECS**	**Amazon EKS**
Scaling	Auto Scaling for ECS tasks and services.	Kubernetes-native scaling mechanisms (e.g., HPA, Cluster Autoscaler).
Cost	AWS Fargate makes ECS highly cost-efficient with serverless containers.	EKS has additional control plane costs ($0.10/hour per cluster), but Fargate is also available.
Community Support	Limited community support; AWS documentation is the primary resource.	Large, active Kubernetes community with open-source tools and resources.

When to Use Amazon ECS

Amazon ECS is ideal for scenarios where:

1. **AWS-Centric Workloads**: If your applications are built entirely on AWS and require tight integration with AWS services like IAM, CloudWatch, and ALB/ELB.
2. **Simpler Workflows**: For teams that want a managed container service with minimal setup and management overhead.
3. **Serverless Containers**: If you're leveraging AWS Fargate to run containers without managing the underlying infrastructure.
4. **Cost Efficiency**: ECS on Fargate is cost-effective for small-to-medium workloads or teams without dedicated Kubernetes expertise.
5. **Quicker Deployment**: For startups or teams that want to deploy containerized applications quickly without learning Kubernetes.

When to Use Amazon EKS

Amazon EKS is best suited for scenarios where:

1. **Kubernetes Expertise**: If your team has expertise in Kubernetes or is already using Kubernetes for on-premise or other cloud deployments.
2. **Multi-Cloud or Hybrid Workloads**: For workloads that need portability across AWS, other clouds, or on-premises environments.
3. **Advanced Orchestration Needs**: If your applications require advanced Kubernetes features like custom controllers, CRDs, or complex networking setups.
4. **Scalability**: For large-scale applications that require Kubernetes-native scaling and auto-healing capabilities.
5. **Open-Source Ecosystem**: If you plan to leverage the Kubernetes ecosystem of tools, plugins, and integrations.
6. **Container Standardization**: If you need to follow Kubernetes as the industry-standard container orchestration framework.

Choosing Between ECS and EKS

The choice between ECS and EKS depends on several factors, including:

- **Team Expertise**: ECS is simpler for teams without container orchestration experience, while EKS requires Kubernetes expertise.
- **Use Case**: ECS is better for AWS-native workloads and simpler architectures. EKS is ideal for complex, multi-cloud, or Kubernetes-based workloads.
- **Cost Considerations**: ECS on Fargate may be cheaper for small workloads. EKS incurs additional costs for the Kubernetes control plane but provides greater flexibility.
- **Operational Overhead**: ECS minimizes management effort, while EKS requires more hands-on administration.
- **Scalability**: Both services scale well, but EKS offers more granular control over scaling with Kubernetes-native tools.

Use Both ECS and EKS Together

In certain scenarios, organizations may use both ECS and EKS to leverage the best of both worlds. For instance:

- **ECS for Simpler Services**: Use ECS for straightforward applications or AWS-native workloads where simplicity and ease of management are key.
- **EKS for Complex Workloads**: Use EKS for Kubernetes-native applications or workloads requiring advanced orchestration.

This hybrid approach ensures operational efficiency while providing flexibility for different types of workloads.

Conclusion: ECS and EKS

Amazon ECS and Amazon EKS represent two of the most powerful container orchestration platforms provided by AWS, catering to diverse needs and workloads in the container ecosystem. Both services simplify the deployment and management of containerized applications, offering scalability, high availability, and deep integrations with the broader AWS ecosystem.

- **Amazon ECS** excels in its seamless AWS integration, low operational overhead, and simplicity for organizations that want a managed container service without diving into the complexities of Kubernetes. With options to run containers on EC2 or AWS Fargate, ECS offers flexibility for cost optimization and workload-specific deployments. It’s particularly well-suited for teams already committed to the AWS ecosystem who want a straightforward, efficient solution for container orchestration.
- **Amazon EKS**, on the other hand, empowers teams that prioritize Kubernetes' extensive ecosystem and flexibility. EKS bridges the gap between AWS services and Kubernetes' robust capabilities, allowing teams to run Kubernetes-native workloads at scale while leveraging AWS's security, networking, and operational tools. It is ideal for enterprises seeking multi-cloud portability or already invested in Kubernetes expertise.