Definitive Guide: EKS with Fargate, ALB, and CloudWatch

This document provides a complete, step-by-step guide to creating a resilient Amazon EKS cluster from scratch. The final architecture runs application workloads on serverless AWS Fargate in private subnets, exposed securely to the internet via an Application Load Balancer (ALB). A dedicated, low-cost EC2 node group is used to run specialized components like the AWS Load Balancer Controller and monitoring agents.

Phase 1: Prerequisites & Tool Installation

These steps prepare your local environment (e.g., CloudShell).

1. Install/Verify Prerequisite Tools
   Ensure you have the AWS CLI and kubectl installed. Then, install helm.
   ```
   # Install Helm (the package manager for Kubernetes)
   curl -fsSL -o get_helm.sh
   https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
   chmod 700 get_helm.sh
   ./get_helm.sh
   ```

2. Set Environment Variables
   This avoids repetition and errors. Use your own desired values.
   ```
   export AWS_REGION="<your-aws-region>"
   export CLUSTER_NAME="<your-cluster-name>"
   export AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account
   --output text)
   ```

Phase 2: Networking Infrastructure (Manual AWS CLI)

Create a resilient, multi-AZ VPC to house the cluster.

1. **Create the VPC:**
   ```
   VPC_ID=$(aws ec2 create-vpc --cidr-block 10.0.0.0/16 --query 'Vpc.VpcId'
   --output text)
   aws ec2 create-tags --resources $VPC_ID --tags
   Key=Name,Value=${CLUSTER_NAME}-VPC
   ```

2. **Create Public and Private Subnets:**
   ```
   # Public Subnet 1
   PUB_SUBNET_1_ID=$(aws ec2 create-subnet --vpc-id $VPC_ID --cidr-block
   10.0.1.0/24 --availability-zone ${AWS_REGION}a --query 'Subnet.SubnetId'
   ```

```
--output text)
aws ec2 create-tags --resources $PUB_SUBNET_1_ID --tags
Key=Name,Value=${CLUSTER_NAME}-PublicSubnet-A

# Public Subnet 2
PUB_SUBNET_2_ID=$(aws ec2 create-subnet --vpc-id $VPC_ID --cidr-block
10.0.2.0/24 --availability-zone ${AWS_REGION}b --query 'Subnet.SubnetId'
--output text)
aws ec2 create-tags --resources $PUB_SUBNET_2_ID --tags
Key=Name,Value=${CLUSTER_NAME}-PublicSubnet-B

# Private Subnet 1
PRIV_SUBNET_1_ID=$(aws ec2 create-subnet --vpc-id $VPC_ID --cidr-block
10.0.3.0/24 --availability-zone ${AWS_REGION}a --query 'Subnet.SubnetId'
--output text)
aws ec2 create-tags --resources $PRIV_SUBNET_1_ID --tags
Key=Name,Value=${CLUSTER_NAME}-PrivateSubnet-A

# Private Subnet 2
PRIV_SUBNET_2_ID=$(aws ec2 create-subnet --vpc-id $VPC_ID --cidr-block
10.0.4.0/24 --availability-zone ${AWS_REGION}b --query 'Subnet.SubnetId'
--output text)
aws ec2 create-tags --resources $PRIV_SUBNET_2_ID --tags
Key=Name,Value=${CLUSTER_NAME}-PrivateSubnet-B
```

3. **Create Internet and NAT Gateways:**

```
# Internet Gateway
IGW_ID=$(aws ec2 create-internet-gateway --query
'InternetGateway.InternetGatewayId' --output text)
aws ec2 attach-internet-gateway --vpc-id $VPC_ID --internet-gateway-id
$IGW_ID
aws ec2 create-tags --resources $IGW_ID --tags
Key=Name,Value=${CLUSTER_NAME}-IGW

# Elastic IP and NAT Gateway
EIP_ALLOC_ID=$(aws ec2 allocate-address --domain vpc --query 'AllocationId'
--output text)
NAT_GW_ID=$(aws ec2 create-nat-gateway --subnet-id $PUB_SUBNET_1_ID
--allocation-id $EIP_ALLOC_ID --query 'NatGateway.NatGatewayId' --output text)
```

```
aws ec2 create-tags --resources $NAT_GW_ID --tags Key=Name,Value=${CLUSTER_NAME}-NAT-GW
```

4. **Configure Route Tables:**

```
# Public Route Table
PUB_RT_ID=$(aws ec2 create-route-table --vpc-id $VPC_ID --query 'RouteTable.RouteTableId' --output text)
aws ec2 create-route --route-table-id $PUB_RT_ID --destination-cidr-block 0.0.0.0/0 --gateway-id $IGW_ID
aws ec2 associate-route-table --subnet-id $PUB_SUBNET_1_ID --route-table-id $PUB_RT_ID
aws ec2 associate-route-table --subnet-id $PUB_SUBNET_2_ID --route-table-id $PUB_RT_ID

# Private Route Table
PRIV_RT_ID=$(aws ec2 create-route-table --vpc-id $VPC_ID --query 'RouteTable.RouteTableId' --output text)
aws ec2 create-route --route-table-id $PRIV_RT_ID --destination-cidr-block 0.0.0.0/0 --nat-gateway-id $NAT_GW_ID
aws ec2 associate-route-table --subnet-id $PRIV_SUBNET_1_ID --route-table-id $PRIV_RT_ID
aws ec2 associate-route-table --subnet-id $PRIV_SUBNET_2_ID --route-table-id $PRIV_RT_ID
```

5. Tag Public Subnets for ALB Discovery:
   This is a critical step that allows the AWS Load Balancer Controller to automatically find these subnets.

```
aws ec2 create-tags --resources $PUB_SUBNET_1_ID $PUB_SUBNET_2_ID --tags Key=kubernetes.io/role/elb,Value=1
```

Phase 3: IAM Roles and Policies

1. **Create EKS Cluster Role:**

```
cat > trust-policy-cluster.json <<EOF
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": { "Service": "eks.amazonaws.com" }, "Action": "sts:AssumeRole" } ] }
EOF
aws iam create-role --role-name ${CLUSTER_NAME}-ClusterRole --assume-role-policy-document file://trust-policy-cluster.json
aws iam attach-role-policy --role-name ${CLUSTER_NAME}-ClusterRole
--policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
```

2. **Create EC2 Node Group Role:**

```
cat > trust-policy-nodes.json <<EOF
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": { "Service":
"ec2.amazonaws.com" }, "Action": "sts:AssumeRole" } ] }
EOF
aws iam create-role --role-name ${CLUSTER_NAME}-NodeRole
--assume-role-policy-document file://trust-policy-nodes.json
aws iam attach-role-policy --role-name ${CLUSTER_NAME}-NodeRole
--policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
aws iam attach-role-policy --role-name ${CLUSTER_NAME}-NodeRole
--policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
aws iam attach-role-policy --role-name ${CLUSTER_NAME}-NodeRole
--policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
```

Phase 4: EKS Cluster Creation

1. Create the Cluster:
   This can take 10-15 minutes.

```
CLUSTER_ROLE_ARN=$(aws iam get-role --role-name
${CLUSTER_NAME}-ClusterRole --query 'Role.Arn' --output text)
aws eks create-cluster --name $CLUSTER_NAME --role-arn
$CLUSTER_ROLE_ARN --resources-vpc-config
subnetIds=$PUB_SUBNET_1_ID,$PUB_SUBNET_2_ID,$PRIV_SUBNET_1_ID,$PRIV_S
UBNET_2_ID
```

2. Configure kubectl:
   Wait for the cluster status to become ACTIVE, then run this command.

```
aws eks update-kubeconfig --name $CLUSTER_NAME
```

3. Create OIDC Provider:
   This is essential for allowing pods to assume IAM roles (IRSA).

```
OIDC_URL=$(aws eks describe-cluster --name $CLUSTER_NAME --query
"cluster.identity.oidc.issuer" --output text | sed -e 's/^https:\/\///')
THUMBPRINT=$(echo | openssl s_client -servername $OIDC_URL -connect
$OIDC_URL:443 2>/dev/null | openssl x509 -fingerprint -noout | sed 's/://g' | awk
-F= '{print $2}')
```

```
aws iam create-open-id-connect-provider --url https://$OIDC_URL --client-id-list
sts.amazonaws.com --thumbprint-list $THUMBPRINT
```

Phase 5: Compute Setup (EC2 & Fargate)

1. **Create the EC2 Node Group:**

   ```
   NODE_ROLE_ARN=$(aws iam get-role --role-name
   ${CLUSTER_NAME}-NodeRole --query 'Role.Arn' --output text)
   aws eks create-nodegroup \
     --cluster-name $CLUSTER_NAME \
     --nodegroup-name ${CLUSTER_NAME}-ec2-nodes \
     --instance-types t3.small \
     --subnets $PUB_SUBNET_1_ID,$PUB_SUBNET_2_ID \
     --node-role $NODE_ROLE_ARN \
     --scaling-config minSize=2,maxSize=2,desiredSize=2 \
     --labels eks.amazonaws.com/compute-type=ec2 \
     --taints
   key=eks.amazonaws.com/compute-type,value=ec2,effect=NO_SCHEDULE
   ```

2. **Create the Fargate Profile:**

   ```
   cat > trust-policy-fargate.json <<EOF
   { "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": { "Service":
   "eks-fargate-pods.amazonaws.com" }, "Action": "sts:AssumeRole" } ] }
   EOF
   aws iam create-role --role-name ${CLUSTER_NAME}-FargateRole
   --assume-role-policy-document file://trust-policy-fargate.json
   aws iam attach-role-policy --role-name ${CLUSTER_NAME}-FargateRole
   --policy-arn arn:aws:iam::aws:policy/AmazonEKSFargatePodExecutionRolePolicy
   FARGATE_ROLE_ARN=$(aws iam get-role --role-name
   ${CLUSTER_NAME}-FargateRole --query 'Role.Arn' --output text)

   aws eks create-fargate-profile \
     --cluster-name $CLUSTER_NAME \
     --fargate-profile-name ${CLUSTER_NAME}-fargate-profile \
     --pod-execution-role-arn $FARGATE_ROLE_ARN \
     --selectors namespace=default \
     --subnets $PRIV_SUBNET_1_ID,$PRIV_SUBNET_2_ID
   ```

Phase 6: AWS Load Balancer Controller Setup

1. **Create IAM Policy and Role for the Controller:**

```
curl -o iam_policy.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.5.0/docs/install/iam_policy.json
aws iam create-policy --policy-name AWSLoadBalancerControllerIAMPolicyFor${CLUSTER_NAME} --policy-document file://iam_policy.json
POLICY_ARN=$(aws iam list-policies --query "Policies[?PolicyName=='AWSLoadBalancerControllerIAMPolicyFor${CLUSTER_NAME}'].Arn" --output text)

OIDC_PROVIDER_ARN="arn:aws:iam::${AWS_ACCOUNT_ID}:oidc-provider/${OIDC_URL}"
cat > trust-policy-controller.json <<EOF
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": { "Federated": "${OIDC_PROVIDER_ARN}" }, "Action": "sts:AssumeRoleWithWebIdentity", "Condition": { "StringEquals": { "${OIDC_URL}:sub": "system:serviceaccount:kube-system:aws-load-balancer-controller" } } } ] }
EOF
aws iam create-role --role-name ${CLUSTER_NAME}-ALB-Controller-Role --assume-role-policy-document file://trust-policy-controller.json
aws iam attach-role-policy --role-name ${CLUSTER_NAME}-ALB-Controller-Role --policy-arn ${POLICY_ARN}
```

2. **Install the Controller using Helm:**

```
CONTROLLER_ROLE_ARN=$(aws iam get-role --role-name ${CLUSTER_NAME}-ALB-Controller-Role --query 'Role.Arn' --output text)
helm repo add eks https://aws.github.io/eks-charts
helm repo update
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --set clusterName=$CLUSTER_NAME \
  --set serviceAccount.create=true \
  --set serviceAccount.name=aws-load-balancer-controller \
  --set serviceAccount.annotations."eks\.amazonaws\.com/role-arn"=$CONTROLLER_RO
```

```
LE_ARN \
  --set nodeSelector."eks\.amazonaws\.com/compute-type"=ec2 \
  --set
tolerations[0].key="eks.amazonaws.com/compute-type",tolerations[0].operator="
Exists",tolerations[0].effect="NoSchedule"
```

Phase 7: Deploying a Sample Application

1. **Create nginx-app.yaml:**

```
cat > nginx-app.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:latest
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: default
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  namespace: default
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: nginx-service
            port:
              number: 80
EOF
```

2. **Apply the Manifest**

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

Phase 8: CloudWatch Monitoring Integration

1. **Add Permissions to Node Role:**

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
  --role-name ${CLUSTER_NAME}-NodeRole
```

2. **Deploy and Configure the CloudWatch Agent:**

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/quickstart/cwagent-fluentd-quickstart.yaml
sed -i "s/{{cluster_name}}/$CLUSTER_NAME/g;s/{{region_name}}/$AWS_REGION/g" cwagent-fluentd-quickstart.yaml
kubectl apply -f cwagent-fluentd-quickstart.yaml
```

3. **Patch the Agent DaemonSets:**

```
kubectl patch daemonset cloudwatch-agent -n amazon-cloudwatch -p '{"spec":{"template":{"spec":{"nodeSelector":{"eks.amazonaws.com/compute-type":"ec2"},"tolerations":[{"key":"eks.amazonaws.com/compute-type","operator":"Exists","effect":"NoSchedule"}]}}}}'
kubectl patch daemonset fluentd-cloudwatch -n amazon-cloudwatch -p '{"spec":{"template":{"spec":{"nodeSelector":{"eks.amazonaws.com/compute-type":"ec2"},"tolerations":[{"key":"eks.amazonaws.com/compute-type","operator":"Exists","effect":"NoSchedule"}]}}}}'
```

Phase 9: Final Verification

1. Get the Load Balancer Address:
   It may take a few minutes for the address to be provisioned.

```
kubectl get ingress nginx-ingress
```

2. Test the Connection:
   Use the address from the command above.

```
curl http://<your-load-balancer-dns-name>
```