AWS EKS Overview: Advanced

Amazon Elastic Kubernetes Service (EKS) is a managed Kubernetes service that makes it easy to run Kubernetes on AWS without needing to install and operate your own Kubernetes control plane. Here's a more advanced look at **Auto-scaling** and **EKS Best Practices**, followed by a step-by-step example.

---

1. Auto-scaling in AWS EKS

Auto-scaling in EKS involves scaling both the Kubernetes resources (pods) and the infrastructure (nodes). This allows for efficient management of workloads and optimal resource utilization.

Types of Auto-scaling in EKS:

- **Horizontal Pod Autoscaler (HPA):** Scales the number of pods based on CPU utilization or other custom metrics.
- **Cluster Autoscaler (CA):** Automatically adjusts the size of the Kubernetes cluster by scaling nodes in response to unschedulable pods.
- **Karpenter (Node Autoscaling):** An alternative to Cluster Autoscaler, which can help with scaling nodes faster and more flexibly.
- **Vertical Pod Autoscaler (VPA):** Adjusts resource requests and limits for running pods based on their usage.

Step-by-Step Example: Configuring Auto-scaling

Step 1: Enable HPA (Horizontal Pod Autoscaler)

- Make sure that the metrics server is installed. Use the following command to deploy it:

```
kubectl apply -f
https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

- Create a Deployment (example: NGINX):

yaml
```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

- Create an HPA to scale the deployment based on CPU usage:

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

- View the status of the HPA:

bash
```
kubectl get hpa
```

Step 2: Enable Cluster Autoscaler

- Tag your ASG (Auto Scaling Group) with the following key-value pair to let the Cluster Autoscaler manage it:

bash

```
Key: k8s.io/cluster-autoscaler/enabled
Value: true
```

- Install the Cluster Autoscaler in your EKS cluster using the following YAML file:

bash
```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/autoscaler/master/cluster-autoscaler/cloudprovider/aws/examples/cluster-autoscaler-autodiscover.yaml
```

- After deployment, configure it with your ASG name:

bash
```
kubectl set env deployment/cluster-autoscaler \
    -n kube-system \
    ASG_NAME=<your-auto-scaling-group-name>
```

- Make sure to edit the deployment to set the correct scaling parameters.

Step 3: Monitor the Auto-scaling

- You can monitor the scaling activities by inspecting your logs, such as:

bash
```
kubectl logs -f deployment/cluster-autoscaler -n kube-system
```

---

2. Best Practices for AWS EKS

To ensure optimal performance, cost-effectiveness, and security, follow these best practices:

Security Best Practices:

- **IAM Roles for Service Accounts:** Use IAM roles for service accounts to limit permissions for your pods, preventing over-permissioning.

- **Network Segmentation:** Use VPC networking, security groups, and network policies to isolate workloads.
- **Pod Security Policies:** Ensure that you have policies in place to enforce best practices, such as non-root users.
- **Secrets Management:** Use AWS Secrets Manager or Kubernetes secrets for managing sensitive data.

Scaling and Performance Best Practices:

- **Cluster Autoscaler and HPA:** Use both the Cluster Autoscaler for scaling nodes and HPA for scaling pods to ensure efficient resource utilization.
- **Right-sizing Nodes:** Choose the right instance types for your workloads, and leverage spot instances where possible for cost optimization.
- **Use Managed Node Groups:** Managed node groups simplify scaling, upgrading, and managing worker nodes.
- **Efficient Load Balancing:** Use AWS ALB Ingress Controller or NGINX ingress controller to efficiently distribute traffic.

Monitoring and Logging Best Practices:

- **Prometheus and Grafana:** Use these tools to monitor cluster health, resource utilization, and performance metrics.
- **AWS CloudWatch Logs and Metrics:** Enable CloudWatch logs and use log groups to store EKS cluster logs (e.g., kubelet logs).
- **Fluentd for Log Aggregation:** Use Fluentd to aggregate logs from the cluster to CloudWatch or another logging system.

Step-by-Step Example: EKS with Auto-scaling and Best Practices

Step 1: Create an EKS Cluster

You can use `eksctl` to create an EKS cluster:

bash

```
eksctl create cluster \
  --name eks-cluster \
  --version 1.27 \
  --region us-west-2 \
  --nodegroup-name standard-workers \
  --node-type t3.medium \
```

```
--nodes 3 \
--nodes-min 1 \
--nodes-max 4 \
--managed
```

This creates a cluster with managed node groups, with auto-scaling enabled between 1 to 4 nodes.

Step 2: Deploy a Sample Application with Auto-scaling

- Deploy the application (as shown earlier with the NGINX example) and configure both HPA and Cluster Autoscaler.

Step 3: Monitoring and Observability

- Set up Prometheus and Grafana to monitor your EKS cluster:
  - Install Prometheus and Grafana using Helm:
    bash
    ```
    helm repo add prometheus-community
    https://prometheus-community.github.io/helm-charts
    helm repo update
    helm install prometheus
    prometheus-community/prometheus
    helm install grafana grafana/grafana
    ```

- Configure CloudWatch logs:
  - Ensure you have CloudWatch enabled for monitoring the cluster and the logs are flowing correctly.

Step 4: Implement Security Practices

- Implement IAM roles for service accounts:

yaml
```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: eks-service-account
```

```
  annotations:
    eks.amazonaws.com/role-arn:
arn:aws:iam::ACCOUNT_ID:role/EKS-ServiceRole
```

- Apply Network Policies to restrict pod communications:

yaml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-ingress
spec:
  podSelector:
    matchLabels:
      app: myapp
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: nginx
```

By following these steps and best practices, you'll have an EKS cluster that scales automatically, is secure, and provides the necessary observability tools to manage it efficiently.