Create IAM Service Role

You work for a company called Globomantics which is looking to move their existing EC2 workload to Kubernetes. You've been tasked with creating a Kubernetes cluster using Amazon EKS and testing it for this purpose.

1. Click on the **Open AWS console** button to the right of this text, then use the credentials provided to log in to AWS.
2. Open the **Services** list from the toolbar and navigate to the **IAM** service.
3. Select **Roles** under **Access management** from the left sidebar.
4. Click the blue **Create role** button to create a new role.
5. Click on the **EKS** use case, then select **EKS - Cluster**.
6. Click the blue **Next: Permissions** button.

   The **AmazonEKSClusterPolicy** AWS managed policy will automatically be selected for the role.

7. Click the blue **Next: Tags** button, then click the blue **Next: Review** button.
8. Enter `eks-cluster-role` for the **Role name**.
9. Click the blue **Create role** button to finish creating the role.
10. Click the blue **Create role** button to create a new role.
11. Click on the **EC2** use case, then select **EC2**.
12. Click the blue **Next: Permissions** button.
13. Search for `AmazonEKSWorkerNodePolicy` and check the left checkbox to select the policy.

    This policy will allow the EC2 worker nodes to access information about the EC2 environment they're running in as well as their EKS cluster.

14. Search for `AmazonEC2ContainerRegistryReadOnly` and select it.

    This policy will allow the worker nodes to access any container images stored in ECR.

15. Search for `AmazonEKS_CNI_Policy` and select it.

    This policy will allow the Amazon VPC CNI plugin to manage the EC2 network interfaces and IP addresses to handle multiple pods on a single node.

    Note: While adding this policy to the node's role is sufficient for now, it's usually a good idea to create a new role specifically for the `aws-node` Kubernetes service account in the cluster. (See Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts)

16. Click the blue **Next: Tags** button, then click the blue **Next: Review** button.
17. Enter `eks-node-role` for the **Role name**.
18. Click the blue **Create role** button to finish creating the role.

You should now see the new `eks-cluster-role` and `eks-node-role` roles in the list. In the next challenge, you'll use these roles to create the EKS cluster which will in turn assume the role to access other AWS services.

Create EKS Cluster

Now that the service role has been created, you can create the cluster.

1. Open the **Services** list from the toolbar and navigate to the **Elastic Kubernetes Service** service.

   AWS EKS is a service which allows you to run a Kubernetes cluster in AWS without needing to manually manage Kubernetes nodes. It can manage EC2 instances for you or run on AWS Fargate, and it integrates with other AWS services such as IAM for authentication, ECR for container image storage, ELB for load balancing, and VPC for networking.

2. Select **Clusters** under **Amazon EKS** from the left sidebar.
3. Click the orange **Create cluster** button.
4. Enter `globomantics` in the **Name** field.
5. Click on the **Kubernetes version** dropdown and select the highest version.

   The default EKS Kubernetes version isn't always the latest available version. Unless you have a specific need to run an older version, it's a good idea to select the latest version.

6. Click on the **Cluster Service Role** dropdown and select the `eks-cluster-role` IAM role you created in the previous challenge.
7. Click the orange **Next** button.

   The default VPC and subnets should automatically be selected. The chosen VPC will be used for all the cluster's resources, and the selected subnets will only be used for the Kubernetes control plane resources. Worker nodes can be run in the same or different subnets as the control plane. See Cluster VPC considerations for additional considerations when deciding which subnets to use.

   The security groups chosen will be used for the control plane, however newer Kubernetes and EKS versions will automatically create a security group for you, so this can be left blank.

   You'll leave **Cluster endpoint access** on the default of **Public** so you can easily connect `kubectl` later, though clusters could also be created in a private mode which would prevent external access outside of the VPC.

   EKS includes a few default Kubernetes plugins under **Networking add-ons**. You can leave them all on their default versions.

    - **Amazon VPC CNI** is a plugin which manages network interfaces and IP addresses of EC2 worker nodes.
    - **CoreDNS** is a DNS server which provides name resolution for the cluster.
    - **kube-proxy** is used to provide external network access to services inside the Kubernetes cluster.
8. Click the orange **Next** button.

    EKS can be configured to log details about many different parts of the control plane here. This can be useful for debugging potential permission issues as well as auditing in the future.

9. Click the orange **Next** button.
10. Click the orange **Create** button.
11. Click the refresh button until you see a status of **Active**.

    Note: The cluster creation process will take several minutes to complete.

In the next challenge, you'll create a node group to manage worker nodes in the cluster.

Create Node Group

Now that the EKS cluster has been created, you have to create a node group to manage worker nodes in the cluster. Without these worker nodes, there would be nowhere for the pods to run.

1. Ensure you're on the **Configuration** tab, then Click on the **Compute** subtab.
2. Click the **Add Node Group** button.
3. Enter `nodes` for the **Name** field.

    If you plan on having multiple types of worker nodes, you would want to name the node groups appropriately and configure Kubernetes labels and taints to allow for proper scheduling of pods on the nodes. This is useful if you have different workloads which are optimized to run with different amounts of memory, CPU usage, disk speed, or network speed. Having different node groups allows for the easy management of the nodes for each of these unique workloads.

4. Click on the **Node IAM Role** dropdown and select the `eks-node-role` IAM role you created earlier.
5. Click the orange **Next** button.

    Here you'll be able to configure the details of the EC2 instances which act as Kubernetes worker nodes. You'll leave most of these set to their defaults for now. If you were creating a cluster for a specific use case which required very large resource limits on pods, you may have to use a larger instance type. Alternatively, if you plan on running a lot of smaller pods, you may be

beneficial to use a smaller instance type. You might even consider running on some or all instances with a **Capacity type** of **Spot** to reduce cost.

6. Click on the **X** in the corner of the **t3.medium** instance type.
7. Click on the **Instance types** dropdown and select **t3.micro**.
8. Click on the orange **Next** button.

   The subnets should default to using all the same subnets that were selected when you created the cluster.

9. Switch off the **Allow remote access to nodes** option.

   This option allows you to configure if and how you'll be able to directly connect to the EC2 instances. You won't be doing this for these nodes since it would require creating an **SSH key pair**.

10. Click on the orange **Next** button.
11. Review the details of our node group, then click the orange **Create** button.
12. Click the refresh button until you see a **Status** of **Active**.

    Note: This will take several minutes to create the node group and start the two worker nodes.

13. Click on the **Nodes** tab.

    Here you should see two nodes with a **Status** of **Ready**.

In the next challenge, you'll connect kubectl to this cluster and try to create pods in these nodes.

Connect Kubectl

Now that the cluster has been created and there are multiple worker nodes available, you can finally connect kubectl to it and create some pods.

1. If you don't have the kubectl CLI tool installed, follow the Kubernetes documentation to install kubectl.
2. Likewise, if you don't have the AWS CLI tool installed, follow the AWS documentation to install the AWS CLI.
3. Locate the provided **CLI CREDENTIALS**, then run the following scripts in a terminal after substituting the correct `${Access Key ID}` and `${Secret Access Key}` values:

   Linux or macOS

   ```
   export AWS_ACCESS_KEY_ID=${Access Key ID} export
   AWS_SECRET_ACCESS_KEY=${Secret Access Key} export
   AWS_DEFAULT_REGION=us-west-2 export KUBECONFIG=temporary-
   kubeconfig
   ```

Windows Command Prompt

```
setx AWS_ACCESS_KEY_ID ${Access Key ID} setx
AWS_SECRET_ACCESS_KEY ${Secret Access Key} setx
AWS_DEFAULT_REGION us-west-2 setx KUBECONFIG temporary-kubeconfig
```

PowerShell

```
$Env:AWS_ACCESS_KEY_ID="${Access Key ID}"
$Env:AWS_SECRET_ACCESS_KEY="${Secret Access Key}"
$Env:AWS_DEFAULT_REGION="us-west-2" $Env:KUBECONFIG="temporary-
kubeconfig"
```

These environment variables will authenticate the AWS CLI with the provided AWS account.

4. Run `aws eks update-kubeconfig --name globomantics` in the terminal.

   This command will generate a `kubectl` configuration to connect to the `globomantics` cluster and merge it with the existing configuration. Since `KUBECONFIG` is set to a temporary config file name, the configuration will be saved there instead of modifying the global `kubectl` configuration on your system.

   You should now see a `temporary-kubeconfig` file in the same directory as you ran the command in. This configuration file contains all the information `kubectl` needs to authenticate with the cluster in combination with the AWS CLI credential environment variables you set.

   Note: Under normal circumstances, the `aws eks update-kubeconfig` command would include a `--role-arn` value which would cause `kubectl` to assume a role which grants it access to the Kubernetes cluster. The `kubectl` configuration can also be configured to use a specific AWS CLI profile from your AWS CLI `.aws/credentials` file so you don't have to manually export credentials as you did here. (See Create a kubeconfig for Amazon EKS for details.)

5. Run `kubectl cluster-info`.

   If the authentication was successful, you should be shown the endpoint information of the cluster.

   Note: If you receive an error, make sure you have successfully installed the kubectl CLI tool by following all the steps for your preferred operating system.

6. Run `kubectl get node`.

   This command will show you the two EC2 worker nodes running in the cluster.

7. Run `kubectl run --rm -it globomantics --image=alpine`.

You'll now be presented with the shell of an Alpine Linux pod running inside the cluster.

8. Run `ip addr`.

   This command will show the pod's `172.16/12` (`172.16.0.1 - 172.31.255.254`) IP address which will be different from the main IP address of the EC2 node it's running on thanks to the VPC CNI plugin.

9. Run `cat /etc/resolv.conf`.

   The `nameserver` in this file will point to the IP of the CoreDNS server which EKS is running for you in the cluster.

10. Type `exit` to disconnect from the pod.

You should now have finished creating and testing a new pod in the cluster you created. EKS offers many advantages over manually managed Kubernetes clusters. Many of the normal Kubernetes maintenance tasks are simpler and can be automated. Security through IAM allows you to integrate with your existing AWS access control. Managing redundancy across availability zones is done automatically.