EXPERIMENT 10

Aim: To study and Implement orchestration using kubernetes.

Objectives:
To understand the steps to deploy Kubernetes Cluster on local systems, deploy applications on Kubernetes, creating a Service in Kubernetes, develop Kubernetes configuration files in YAML and creating a deployment in Kubernetes using YAML

Theory:
Container orchestration is the automation of much of the operational effort required to run containerized workloads and services. This includes a wide range of things software teams need to manage a container's lifecycle, including provisioning, deployment, scaling (up and down), networking, load balancing and more.
Why do we need container orchestration?
Because containers are lightweight and ephemeral by nature, running them in production can quickly become a massive effort. Particularly when paired with microservices—which typically each run in their own containers—a containerized application might translate into operating hundreds or thousands of containers, especially when building and operating any large-scale system.
This can introduce significant complexity if managed manually. Container orchestration is what makes that operational complexity manageable for development and operations— or DevOps—because it provides a declarative way of automating much of the work. This makes it a good fit for DevOps teams and culture, which typically strive to operate with much greater speed and agility than traditional software teams.
Container orchestration is key to working with containers, and it allows organizations to unlock their full benefits. It also offers its own benefits for a containerized environment, including:

- Simplified operations: This is the most important benefit of container orchestration and the main reason for its adoption. Containers introduce a large amount of complexity that can quickly get out of control without container orchestration to manage it.
- Resilience: Container orchestration tools can automatically restart or scale a container or cluster, boosting resilience.
- Added security: Container orchestration's automated approach helps keep containerized applications secure by reducing or eliminating the chance of human error.

Containers are a method of building, packaging and deploying software. They are similar to but not the same thing as virtual machines (VMs). One of the primary differences is that containers are isolated or abstracted away from the underlying operating system and infrastructure that they

run on. In the simplest terms, a container includes both an application's code and everything that code needs to run properly.

Because of this, containers offer many benefits, including:

- Portability: One of the biggest benefits of containers is that they're built to run in any environment. This makes containerized workloads easier to move between different cloud platforms, for example, without having to rewrite large amounts of code to ensure it will execute properly, regardless of the underlying operating system or other factors. This also boosts developer productivity, since they can write code in a consistent manner without worrying about its execution when deployed to different environments—from a local machine to an on-premises server to a public cloud.
- Application development: Containers can speed up application development and deployments, including changes or updates over time. This is particularly true with containerized microservices. This is an approach to software architecture that entails breaking up a larger solution into smaller parts. Those discrete components (or microservices) can then be deployed, updated or retired independently, without having to update and redeploy the entire application.
- Resource utilization and optimization: Containers are lightweight and ephemeral, so they consume fewer resources. You can run many containers on a single machine, for example.

What is Kubernetes container orchestration?

Kubernetes is a popular open source platform for container orchestration. It enables developers to easily build containerized applications and services, as well as scale, schedule and monitor those containers. While there are other options for container orchestration, such as Apache Mesos or Docker Swarm, Kubernetes has become the industry standard. Kubernetes provides extensive container capabilities, a dynamic contributor community, the growth of cloud-native application development and the widespread availability of commercial and hosted Kubernetes tools. Kubernetes is also highly extensible and portable, meaning it can run in a wide range of environments and be used in conjunction with other technologies, such as service meshes.

In addition to enabling the automation fundamental to container orchestration, Kubernetes is considered highly declarative. This means that developers and administrators use it to essentially describe how they want a system to behave, and then Kubernetes executes that desired state in dynamic fashion.

What is multi-cloud container orchestration?

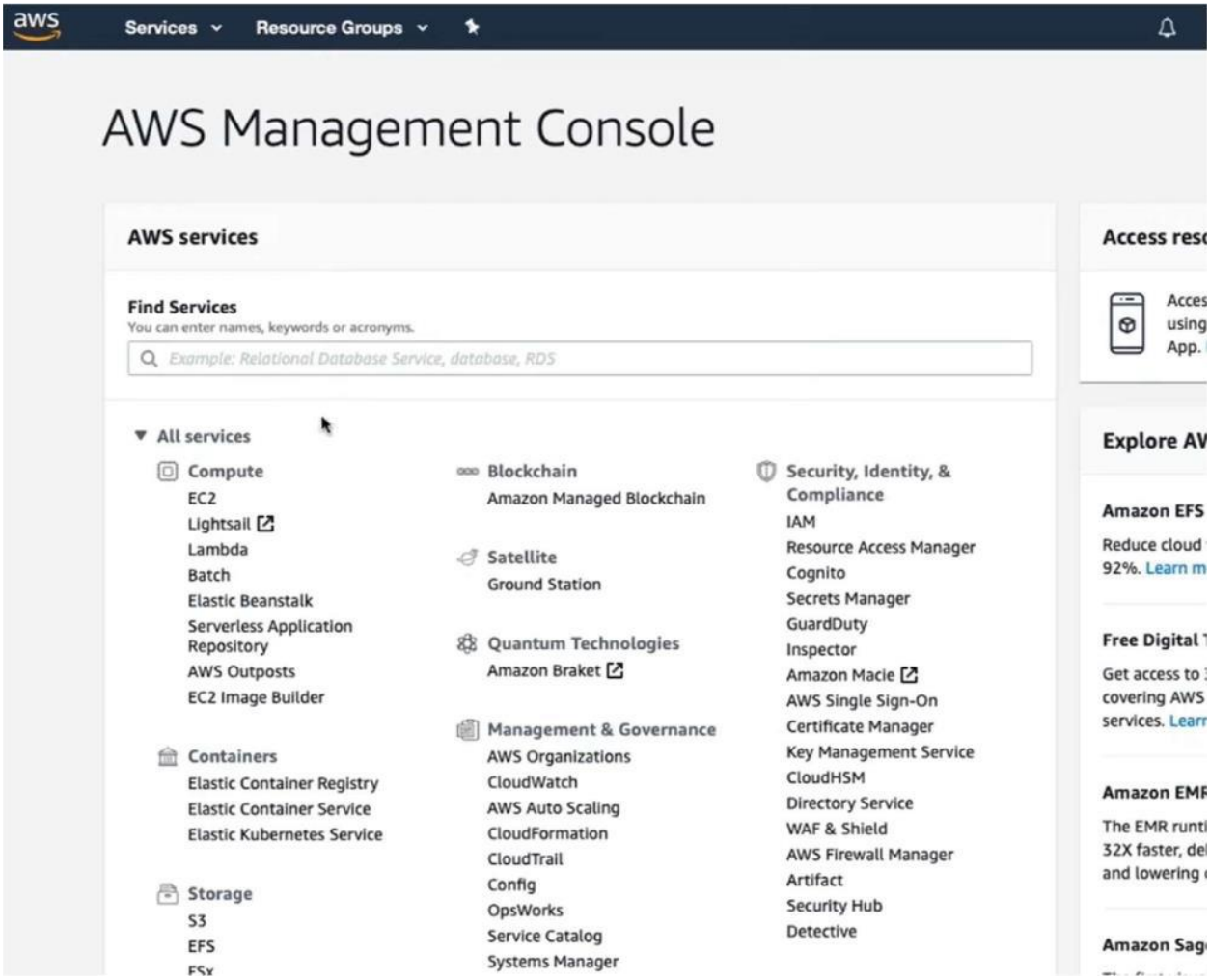
In the most basic sense, the term "multi-cloud" refers to an IT strategy of using two or more cloud services from two or more providers. In the context of containers and orchestration, multi-cloud usually means the use of two or more cloud infrastructure platforms, including public and private clouds, for running applications. Multi-cloud container orchestration, then, refers to the use of an orchestration tool to operate containers across multi-cloud infrastructure environments—instead of running containers in a single cloud environment.

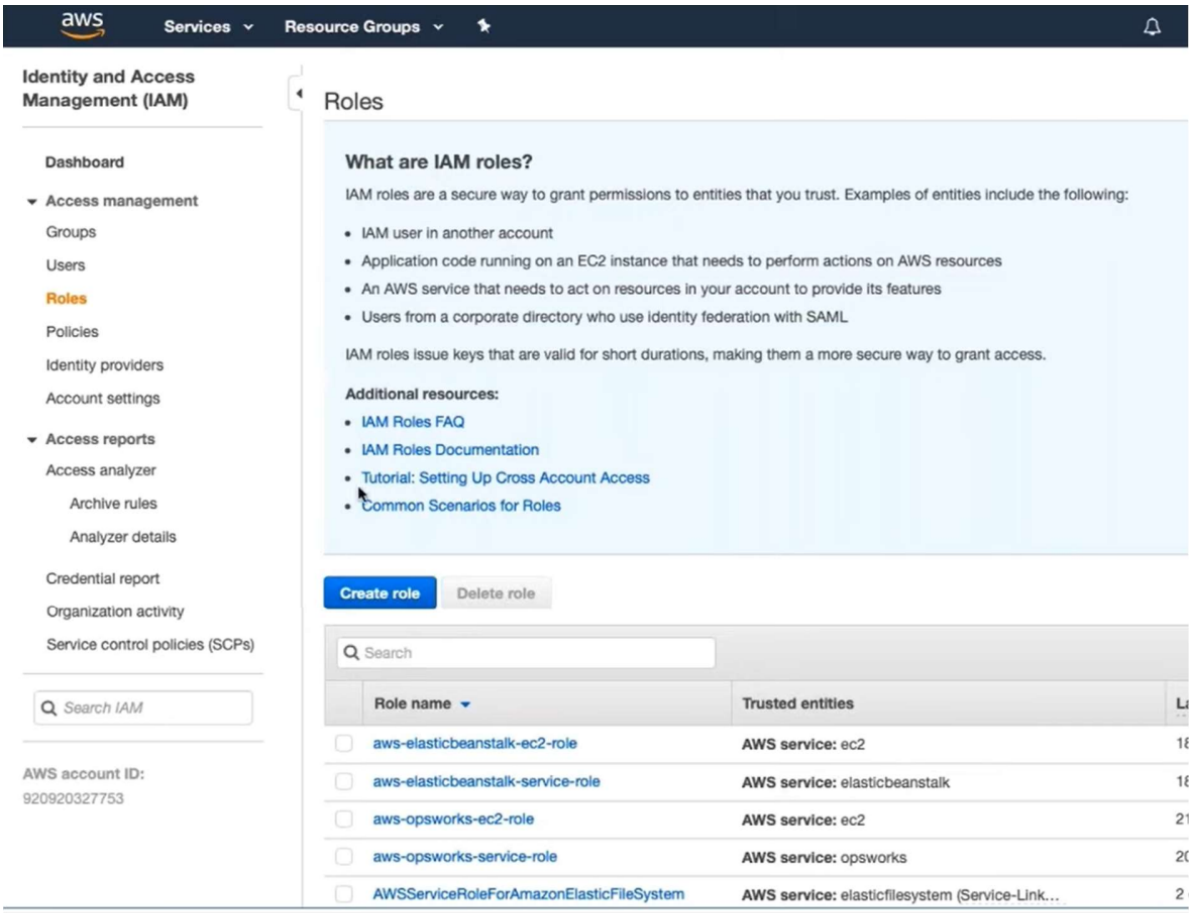
Software teams pursue multi-cloud strategies for different reasons, but the benefits can include infrastructure cost optimization, flexibility and portability (including reducing vendor lock-in), and scalability (such as dynamically scaling out a cloud from an on-premises environment when necessary.) Multi-cloud environments and containers go hand-in-hand because of the latter's portable, "run anywhere" nature.

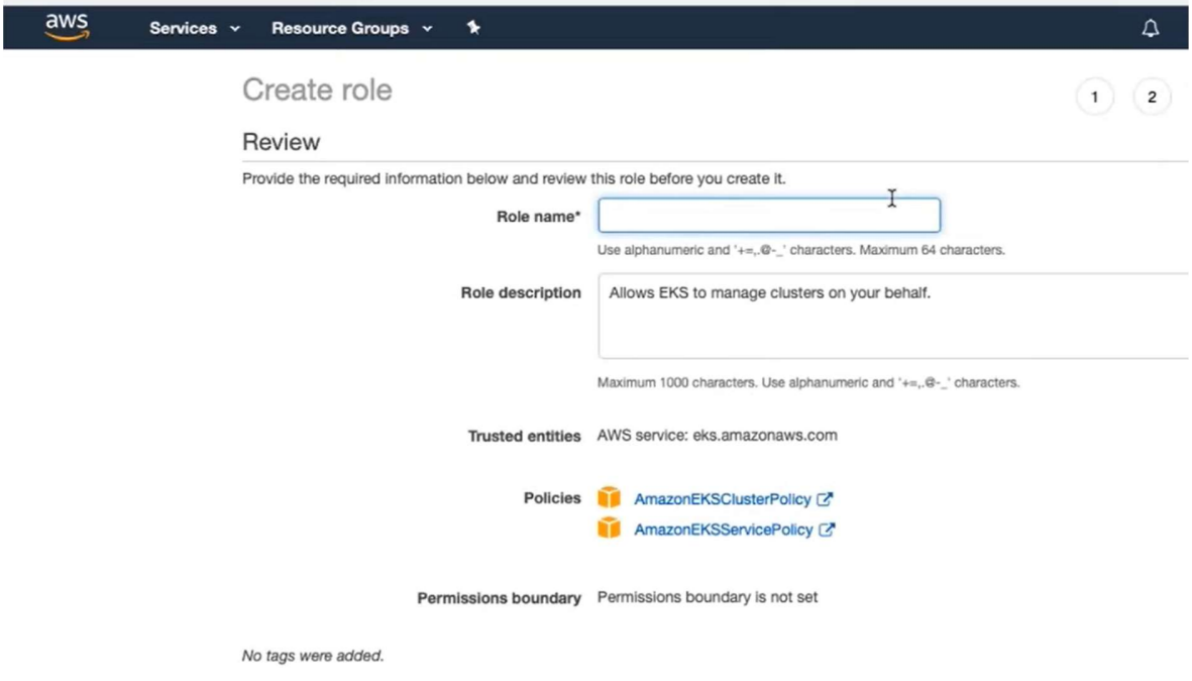
Container orchestration versus Docker

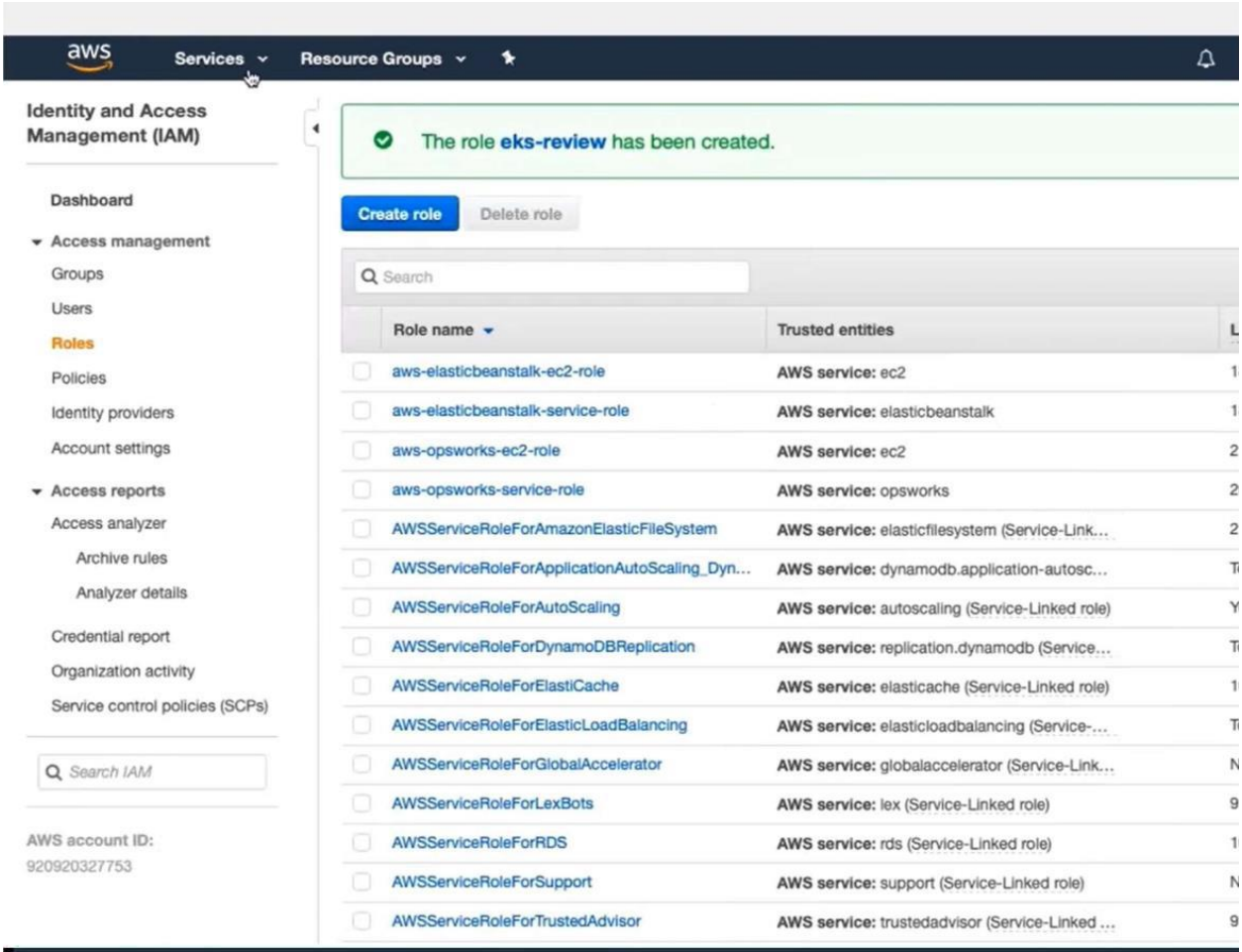
Docker is a specific platform for building containers, including the Docker Engine container runtime, whereas container orchestration is a broader term referring to automation of any container's lifecycle. Docker also includes Docker Swarm, which is the platform's own container orchestration tool that can automatically start Docker containers.

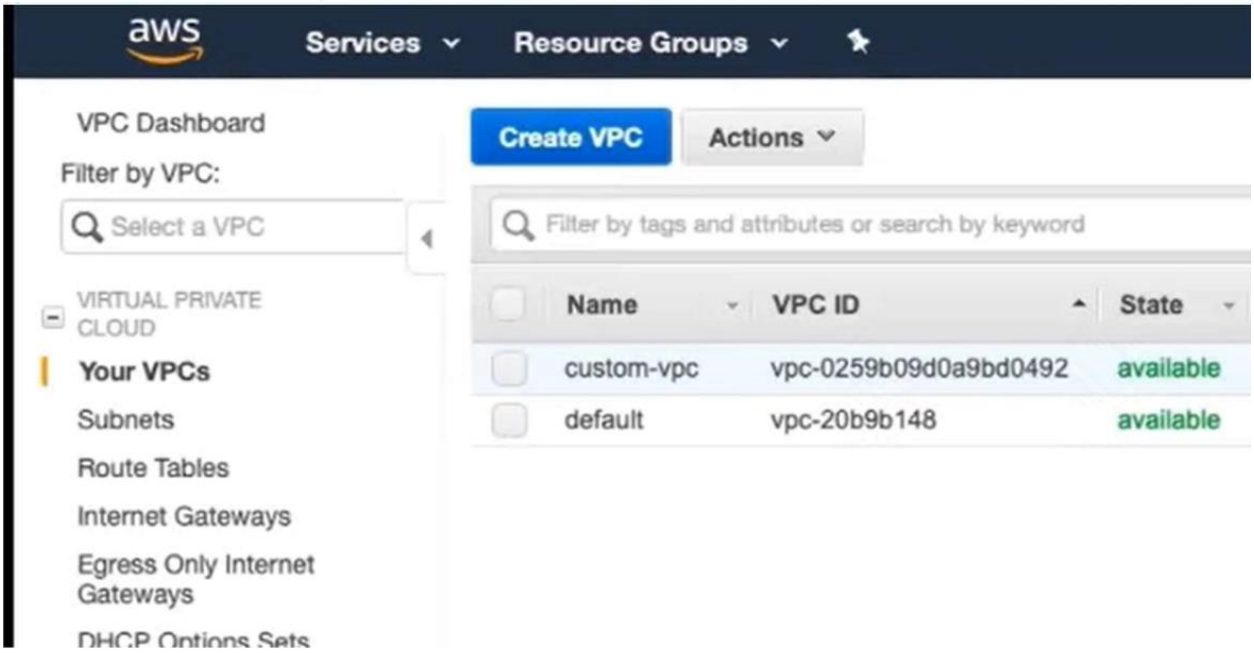
Implementation and Output:

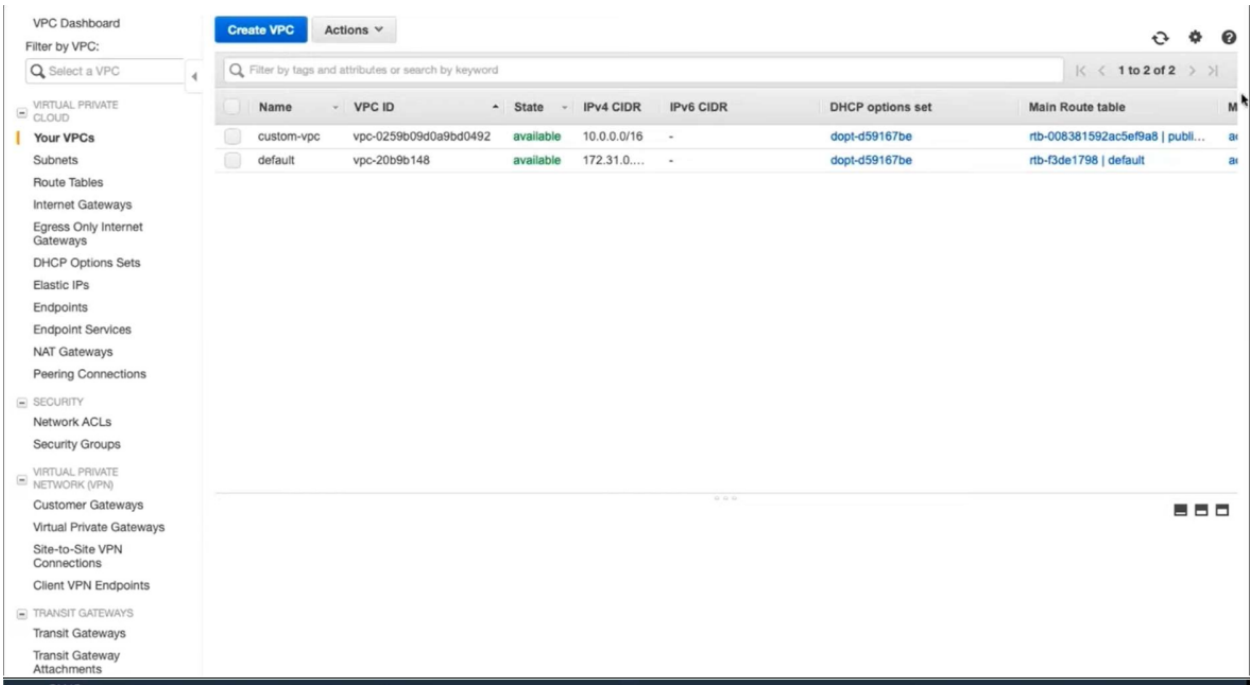


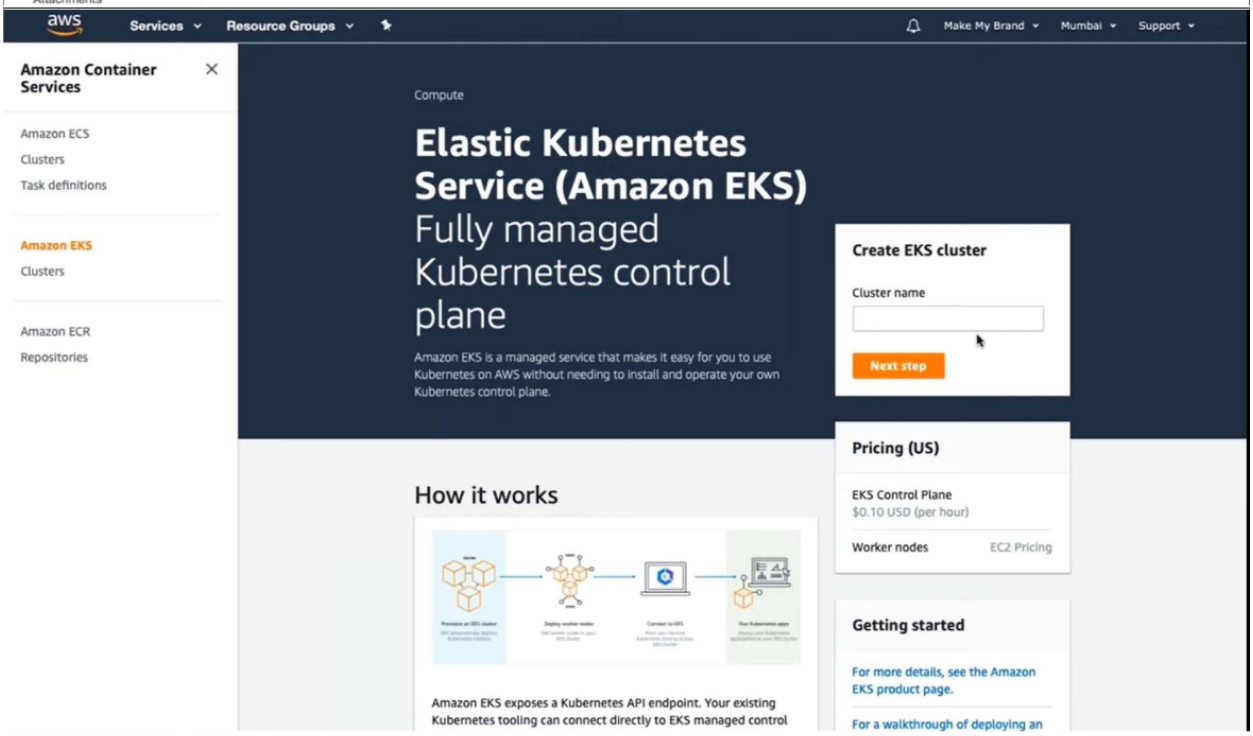

aws
Services
Resource Groups
Identity and Access Management (IAM)
Dashboard
Access management
Groups
Users
Roles
Policies
Identity providers
Account settings
Access reports
Access analyzer
Archive rules
Analyzer details
Credential report
Organization activity
Service control policies (SCPs)
Search IAM
AWS account ID:
920920327753
Roles
What are IAM roles?
IAM roles are a secure way to grant permissions to entities that you trust. Examples of entities include the following:
IAM user in another account
Application code running on an EC2 instance that needs to perform actions on AWS resources
An AWS service that needs to act on resources in your account to provide its features
Users from a corporate directory who use identity federation with SAML
IAM roles issue keys that are valid for short durations, making them a more secure way to grant access.
Additional resources:
IAM Roles FAQ
IAM Roles Documentation
Tutorial: Setting Up Cross Account Access
Common Scenarios for Roles
Create role
Delete role
Search
Role name
Trusted entities
aws-elasticbeanstalk-ec2-role
AWS service: ec2
aws-elasticbeanstalk-service-role
AWS service: elasticbeanstalk
aws-opsworks-ec2-role
AWS service: ec2
aws-opsworks-service-role
AWS service: opsworks
AWSServiceRoleForAmazonElasticFileSystem
AWS service: elasticfilesystem (Service-Link...

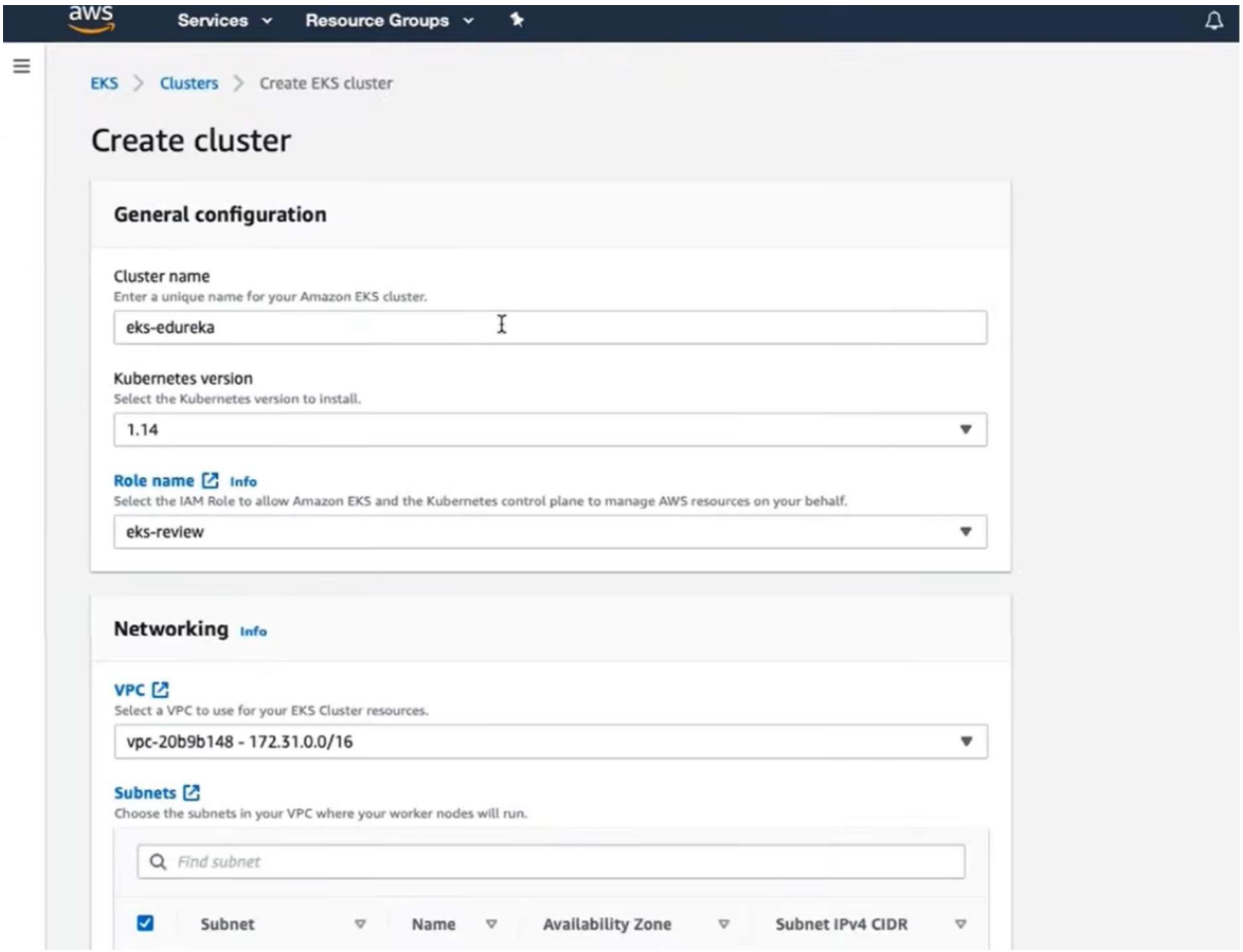

aws
Services
Resource Groups
Create role
1
2
Review
Provide the required information below and review this role before you create it.
Role name*
Use alphanumeric and '+=,.@-_' characters. Maximum 64 characters.
Role description
Allows EKS to manage clusters on your behalf.
Maximum 1000 characters. Use alphanumeric and '+=,.@-_' characters.
Trusted entities
AWS service: eks.amazonaws.com
Policies
AmazonEKSClusterPolicy
AmazonEKSServicePolicy
Permissions boundary
Permissions boundary is not set
No tags were added.

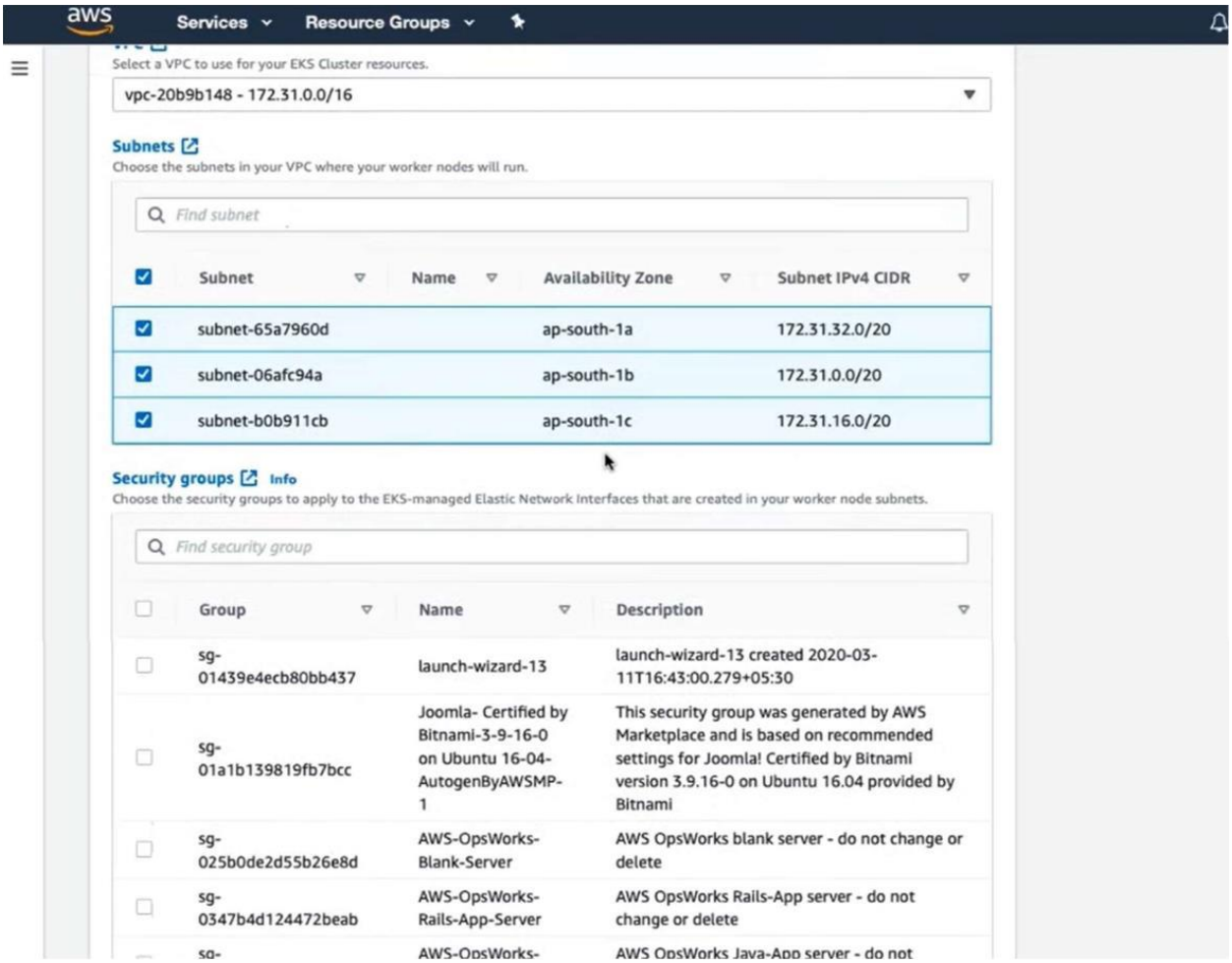

aws
Services
Resource Groups
Identity and Access Management (IAM)
Dashboard
Access management
Groups
Users
Roles
Policies
Identity providers
Account settings
Access reports
Access analyzer
Archive rules
Analyzer details
Credential report
Organization activity
Service control policies (SCPs)
Search IAM
AWS account ID:
920920327753
The role eks-review has been created.
Create role
Delete role
Search
Role name
Trusted entities
aws-elasticbeanstalk-ec2-role
AWS service: ec2
aws-elasticbeanstalk-service-role
AWS service: elasticbeanstalk
aws-opsworks-ec2-role
AWS service: ec2
aws-opsworks-service-role
AWS service: opsworks
AWSServiceRoleForAmazonElasticFileSystem
AWS service: elasticfilesystem (Service-Link...
AWSServiceRoleForApplicationAutoScaling_Dyn...
AWS service: dynamodb.application-autosc...
AWSServiceRoleForAutoScaling
AWS service: autoscaling (Service-Linked role)
AWSServiceRoleForDynamoDBReplication
AWS service: replication.dynamodb (Service...
AWSServiceRoleForElastiCache
AWS service: elasticache (Service-Linked role)
AWSServiceRoleForElasticLoadBalancing
AWS service: elasticloadbalancing (Service-...
AWSServiceRoleForGlobalAccelerator
AWS service: globalaccelerator (Service-Link...
AWSServiceRoleForLexBots
AWS service: lex (Service-Linked role)
AWSServiceRoleForRDS
AWS service: rds (Service-Linked role)
AWSServiceRoleForSupport
AWS service: support (Service-Linked role)
AWSServiceRoleForTrustedAdvisor
AWS service: trustedadvisor (Service-Linked ...

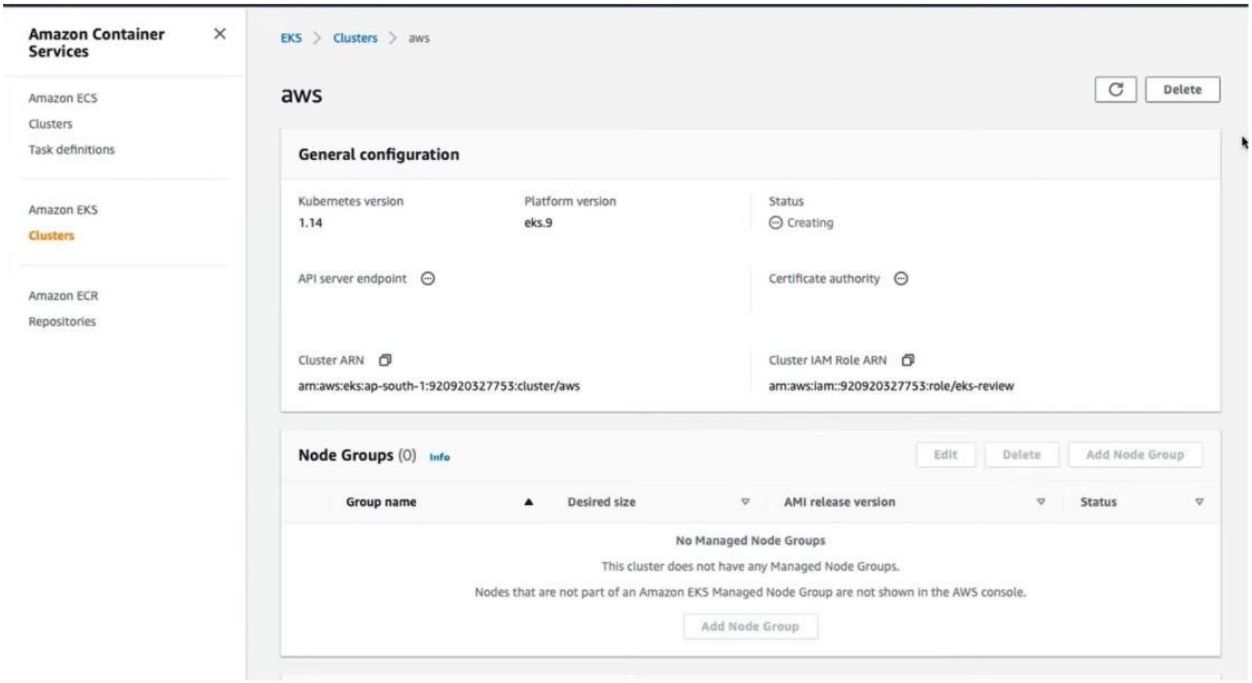

aws
Services
Resource Groups
VPC Dashboard
Filter by VPC:
Select a VPC
VIRTUAL PRIVATE CLOUD
Your VPCs
Subnets
Route Tables
Internet Gateways
Egress Only Internet Gateways
DHCP Options Sets
Create VPC
Actions
Filter by tags and attributes or search by keyword
Name
VPC ID
State
custom-vpc
vpc-0259b09d0a9bd0492
available
default
vpc-20b9b148
available

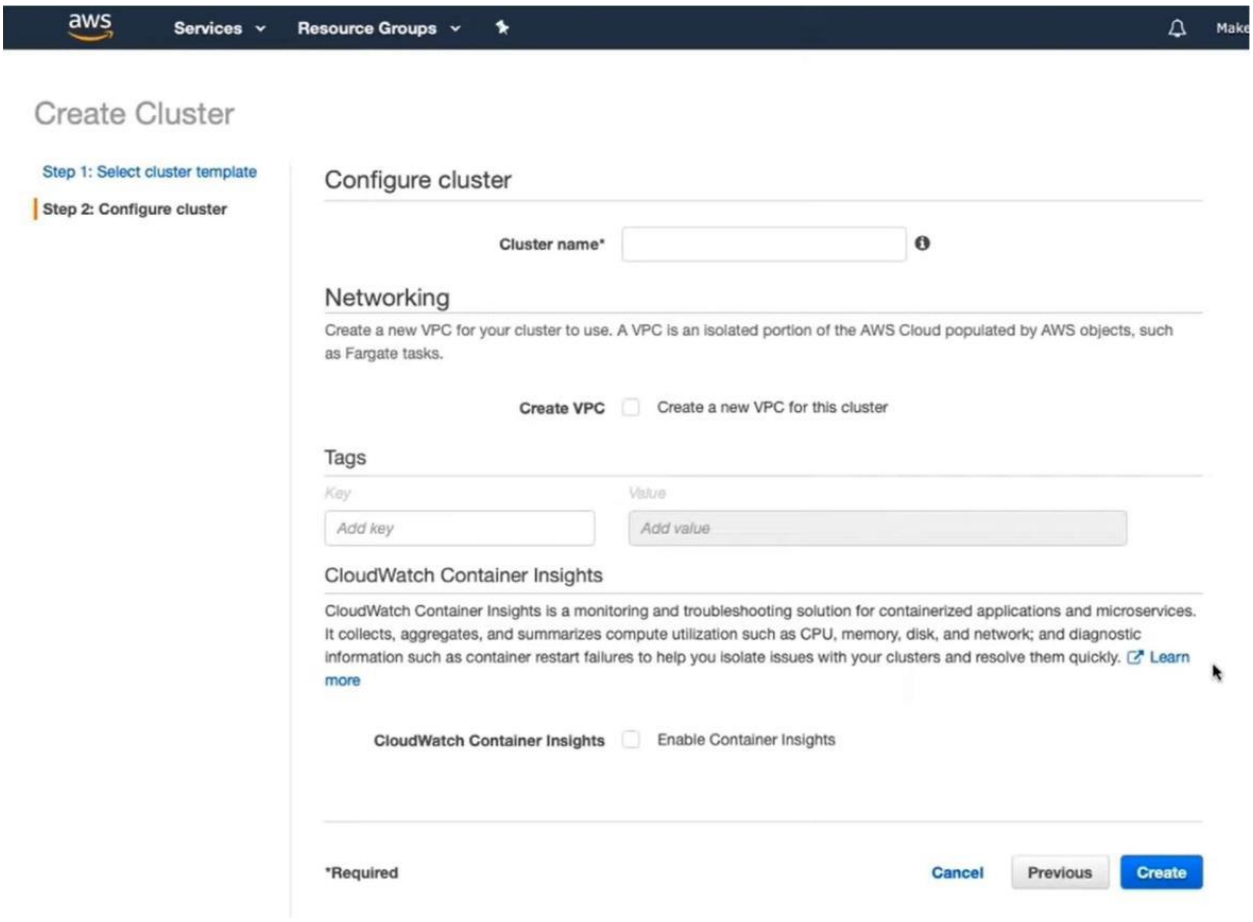

VPC Dashboard
Filter by VPC:
Select a VPC
VIRTUAL PRIVATE CLOUD
Your VPCs
Subnets
Route Tables
Internet Gateways
Egress Only Internet Gateways
DHCP Options Sets
Elastic IPs
Endpoints
Endpoint Services
NAT Gateways
Peering Connections
SECURITY
Network ACLs
Security Groups
VIRTUAL PRIVATE NETWORK (VPN)
Customer Gateways
Virtual Private Gateways
Site-to-Site VPN Connections
Client VPN Endpoints
TRANSIT GATEWAYS
Transit Gateways
Transit Gateway Attachments
Create VPC
Actions
Filter by tags and attributes or search by keyword
1 to 2 of 2
Name
VPC ID
State
IPv4 CIDR
IPv6 CIDR
DHCP options set
Main Route table
custom-vpc
vpc-0259b09d0a9bd0492
available
10.0.0.0/16
-
dopt-d59167be
rtb-008381592ac5ef9a8 | publi...
default
vpc-20b9b148
available
172.31.0....
-
dopt-d59167be
rtb-f3de1798 | default


aws
Services
Resource Groups
Make My Brand
Mumbai
Support
Amazon Container Services
Amazon ECS
Clusters
Task definitions
Amazon EKS
Clusters
Amazon ECR
Repositories
Compute
Elastic Kubernetes Service (Amazon EKS)
Fully managed Kubernetes control plane
Amazon EKS is a managed service that makes it easy for you to use Kubernetes on AWS without needing to install and operate your own Kubernetes control plane.
Create EKS cluster
Cluster name
Next step
Pricing (US)
EKS Control Plane
$0.10 USD (per hour)
Worker nodes
EC2 Pricing
Getting started
For more details, see the Amazon EKS product page.
For a walkthrough of deploying an
How it works
Amazon EKS exposes a Kubernetes API endpoint. Your existing Kubernetes tooling can connect directly to EKS managed control


aws
Services
Resource Groups
EKS > Clusters > Create EKS cluster
Create cluster
General configuration
Cluster name
Enter a unique name for your Amazon EKS cluster.
eks-edureka
Kubernetes version
Select the Kubernetes version to install.
1.14
Role name Info
Select the IAM Role to allow Amazon EKS and the Kubernetes control plane to manage AWS resources on your behalf.
eks-review
Networking Info
VPC
Select a VPC to use for your EKS Cluster resources.
vpc-20b9b148 - 172.31.0.0/16
Subnets
Choose the subnets in your VPC where your worker nodes will run.
Find subnet
Subnet
Name
Availability Zone
Subnet IPv4 CIDR


Services
Resource Groups
Select a VPC to use for your EKS Cluster resources.
vpc-20b9b148 - 172.31.0.0/16
Subnets
Choose the subnets in your VPC where your worker nodes will run.
Find subnet
Subnet
Name
Availability Zone
Subnet IPv4 CIDR
subnet-65a7960d
ap-south-1a
172.31.32.0/20
subnet-06afc94a
ap-south-1b
172.31.0.0/20
subnet-b0b911cb
ap-south-1c
172.31.16.0/20
Security groups
Info
Choose the security groups to apply to the EKS-managed Elastic Network Interfaces that are created in your worker node subnets.
Find security group
Group
Name
Description
sg-01439e4ecb80bb437
launch-wizard-13
launch-wizard-13 created 2020-03-11T16:43:00.279+05:30
sg-01a1b139819fb7bcc
Joomla- Certified by Bitnami-3-9-16-0 on Ubuntu 16-04-AutogenByAWSMP-1
This security group was generated by AWS Marketplace and is based on recommended settings for Joomla! Certified by Bitnami version 3.9.16-0 on Ubuntu 16.04 provided by Bitnami
sg-025b0de2d55b26e8d
AWS-OpsWorks-Blank-Server
AWS OpsWorks blank server - do not change or delete
sg-0347b4d124472beab
AWS-OpsWorks-Rails-App-Server
AWS OpsWorks Rails-App server - do not change or delete


Amazon Container Services
Amazon ECS
Clusters
Task definitions
Amazon EKS
Clusters
Amazon ECR
Repositories
EKS > Clusters > aws
aws
Delete
General configuration
Kubernetes version
1.14
Platform version
eks.9
Status
Creating
API server endpoint
Certificate authority
Cluster ARN
arn:aws:eks:ap-south-1:920920327753:cluster/aws
Cluster IAM Role ARN
arn:aws:iam::920920327753:role/eks-review
Node Groups (0)
Info
Edit
Delete
Add Node Group
Group name
Desired size
AMI release version
Status
No Managed Node Groups
This cluster does not have any Managed Node Groups.
Nodes that are not part of an Amazon EKS Managed Node Group are not shown in the AWS console.
Add Node Group



Conclusion: In conclusion, the experimentation and implementation of orchestration using Kubernetes have provided valuable insights into the capabilities and benefits of this powerful tool. Through this experiment, we have gained a deeper understanding of how Kubernetes efficiently manages containerized applications, automates deployment, scales resources dynamically, and ensures high availability.

The results of the experiment demonstrate that Kubernetes simplifies the management of complex containerized environments, streamlines the deployment process, and enhances the overall efficiency of the system. By leveraging Kubernetes' features such as pods, services, deployments, and namespaces, we have been able to achieve better resource utilization, improved scalability, and enhanced fault tolerance.

Furthermore, the experiment has highlighted the importance of proper planning, configuration, and monitoring when implementing Kubernetes orchestration. Effective utilization of Kubernetes requires careful consideration of factors such as networking, storage, security, and application architecture.