

For VRT setup we need to follow below steps:

1. Create EKS Cluster with below command:

```
eksctl create cluster --name bmi-demo --vpc-cidr 10.50.0.0/16 --region ap-south-1  
--nodegroup-name application-worker-node --node-type r5a.large --nodes 3 --nodes-min 3  
--nodes-max 6 --node-volume-size 40 --node-volume-type gp3 --enable-ssm --managed  
--node-private-networking --ssh-access=true --ssh-public-key=demi-demo --profile bmi-eks
```

```
rupam.saini@scriptbox:~$ eksctl create cluster --name bmi-demo --vpc-cidr 10.50.0.0/16 --region ap-south-1 --nodegroup-name application-worker-node --node-type r5a.large --nodes 3 --nodes-min 3 --nodes-max 6 --node-volume-size 40 --node-volume-type gp3 --enable-ssm --managed --node-private-networking --ssh-access=true --ssh-public-key=demi-demo --profile bmi-eks  
2022-12-19 16:05:52 [i] SSM is now enabled by default; 'ssh.enableSSM' is deprecated and will be removed in a future release  
2022-12-19 16:05:53 [i] eksctl version 0.122.0  
2022-12-19 16:05:53 [i] using region ap-south-1  
2022-12-19 16:05:53 [i] setting availability zones to [ap-south-1a ap-south-1c ap-south-1b]  
2022-12-19 16:05:53 [i] subnets for ap-south-1a - public:10.50.0.0/19 private:10.50.96.0/19  
2022-12-19 16:05:53 [i] subnets for ap-south-1c - public:10.50.32.0/19 private:10.50.128.0/19  
2022-12-19 16:05:53 [i] subnets for ap-south-1b - public:10.50.64.0/19 private:10.50.160.0/19  
2022-12-19 16:05:53 [i] nodegroup "application-worker-node" will use "" [AmazonLinux2/1.23]  
2022-12-19 16:05:53 [i] using EC2 key pair %!q(*string=<nil>)  
2022-12-19 16:05:53 [i] using Kubernetes version 1.23  
2022-12-19 16:05:53 [i] creating EKS cluster "bmi-demo" in "ap-south-1" region with managed nodes  
2022-12-19 16:05:53 [i] will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup  
2022-12-19 16:05:53 [i] if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=ap-south-1 --cluster=bmi-demo'  
2022-12-19 16:05:53 [i] Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "bmi-demo" in "ap-south-1"  
2022-12-19 16:05:53 [i] CloudWatch logging will not be enabled for cluster "bmi-demo" in "ap-south-1"  
2022-12-19 16:05:53 [i] you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=ap-south-1 --cluster=bmi-demo'  
2022-12-19 16:05:53 [i]  
2 sequential tasks: { create cluster control plane "bmi-demo",  
  2 sequential sub-tasks: {  
    wait for control plane to become ready,  
    create managed nodegroup "application-worker-node",  
  }  
}  
2022-12-19 16:05:53 [i] building cluster stack "eksctl-bmi-demo-cluster"  
2022-12-19 16:05:54 [i] deploying stack "eksctl-bmi-demo-cluster"  
2022-12-19 16:06:24 [i] waiting for CloudFormation stack "eksctl-bmi-demo-cluster"  
2022-12-19 16:06:54 [i] waiting for CloudFormation stack "eksctl-bmi-demo-cluster"  
2022-12-19 16:07:54 [i] waiting for CloudFormation stack "eksctl-bmi-demo-cluster"  
2022-12-19 16:08:55 [i] waiting for CloudFormation stack "eksctl-bmi-demo-cluster"
```

```
rupam.saini@scriptbox:~$ eksctl create cluster --name bmi-demo --vpc-cidr 10.50.0.0/16 --region ap-south-1  
--nodegroup-name application-worker-node --node-type r5a.large --nodes 3 --nodes-min 3  
--nodes-max 6 --node-volume-size 40 --node-volume-type gp3 --enable-ssm --managed  
--node-private-networking --ssh-access=true --ssh-public-key=demi-demo --profile bmi-eks  
2022-12-19 16:05:52 [i] SSM is now enabled by default; 'ssh.enableSSM' is deprecated and will be removed in a future release  
2022-12-19 16:05:53 [i] eksctl version 0.122.0  
2022-12-19 16:05:53 [i] using region ap-south-1  
2022-12-19 16:05:53 [i] setting availability zones to [ap-south-1a ap-south-1c ap-south-1b]  
2022-12-19 16:05:53 [i] subnets for ap-south-1a - public:10.50.0.0/19 private:10.50.96.0/19  
2022-12-19 16:05:53 [i] subnets for ap-south-1c - public:10.50.32.0/19 private:10.50.128.0/19  
2022-12-19 16:05:53 [i] subnets for ap-south-1b - public:10.50.64.0/19 private:10.50.160.0/19  
2022-12-19 16:05:53 [i] nodegroup "application-worker-node" will use "" [AmazonLinux2/1.23]  
2022-12-19 16:05:53 [i] using EC2 key pair %!q(*string=<nil>)  
2022-12-19 16:05:53 [i] using Kubernetes version 1.23  
2022-12-19 16:05:53 [i] creating EKS cluster "bmi-demo" in "ap-south-1" region with managed nodes  
2022-12-19 16:05:53 [i] will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup  
2022-12-19 16:05:53 [i] if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=ap-south-1 --cluster=bmi-demo'  
2022-12-19 16:05:53 [i] Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "bmi-demo" in "ap-south-1"  
2022-12-19 16:05:53 [i] CloudWatch logging will not be enabled for cluster "bmi-demo" in "ap-south-1"  
2022-12-19 16:05:53 [i] you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=ap-south-1 --cluster=bmi-demo'  
2022-12-19 16:05:53 [i]  
2 sequential tasks: { create cluster control plane "bmi-demo",  
  2 sequential sub-tasks: {  
    wait for control plane to become ready,  
    create managed nodegroup "application-worker-node",  
  }  
}  
2022-12-19 16:05:53 [i] building cluster stack "eksctl-bmi-demo-cluster"  
2022-12-19 16:05:54 [i] deploying stack "eksctl-bmi-demo-cluster"  
2022-12-19 16:06:24 [i] waiting for CloudFormation stack "eksctl-bmi-demo-cluster"  
2022-12-19 16:06:54 [i] waiting for CloudFormation stack "eksctl-bmi-demo-cluster"  
2022-12-19 16:07:54 [i] waiting for CloudFormation stack "eksctl-bmi-demo-cluster"  
2022-12-19 16:08:55 [i] waiting for CloudFormation stack "eksctl-bmi-demo-cluster"
```

- replace the EKS cluster name with dmi-demo on command and create Node group for custom application.

```
eksctl create nodegroup --cluster=dmi-demo --region=ap-south-1 --managed
--name=custom-apps-nodegroup --instance-types=r5.large,t3.medium,t3a.large,t3a.xlarge
--nodes-min=2 --nodes-max=8 --node-volume-type gp3 --node-volume-size 15 --asg-access
--enable-ssm --managed --node-private-networking --ssh-access=true --ssh-public-key=demi-demo
--profile dmi-eks
```

```
rupam.saini@scriptbox: $ eksctl create nodegroup --cluster=dmi-demo --region=ap-south-1 --managed --name=custom-apps-nodegroup --instance-types=r5.large,t3.medium,t3a.large,t3a.xlarge --nodes-min=2 --nodes-max=8 --node-volume-type gp3 --node-volume-size 15 --asg-access --enable-ssm --managed --node-private-networking --ssh-access=true --ssh-public-key=demi-demo --profile dmi-eks
2022-12-19 16:34:58 [i] will use version 1.23 for new nodegroup(s) based on control plane version
2022-12-19 16:34:58 [!] SSM is now enabled by default; 'ssh.enableSSM' is deprecated and will be removed in a future release
2022-12-19 16:34:59 [i] nodegroup "custom-apps-nodegroup" will use "" [AmazonLinux2/1.23]
2022-12-19 16:35:00 [i] using EC2 key pair %!q(*string=<nil>)
2022-12-19 16:35:00 [i] 1 existing nodegroup(s) (application-worker-node) will be excluded
2022-12-19 16:35:00 [i] 1 nodegroup (custom-apps-nodegroup) was included (based on the include/exclude rules)
2022-12-19 16:35:00 [i] will create a CloudFormation stack for each of 1 managed nodegroups in cluster "dmi-demo"
2022-12-19 16:35:00 [i]
2 sequential tasks: { fix cluster compatibility, 1 task: { 1 task: { create managed nodegroup "custom-apps-nodegroup" } } }
2022-12-19 16:35:00 [i] checking cluster stack for missing resources
2022-12-19 16:35:00 [i] cluster stack has all required resources
2022-12-19 16:35:01 [i] building managed nodegroup stack "eksctl-dmi-demo-nodegroup-custom-apps-nodegroup"
2022-12-19 16:35:01 [i] deploying stack "eksctl-dmi-demo-nodegroup-custom-apps-nodegroup"
2022-12-19 16:35:01 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-custom-apps-nodegroup"
2022-12-19 16:35:32 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-custom-apps-nodegroup"
2022-12-19 16:36:26 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-custom-apps-nodegroup"
2022-12-19 16:37:29 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-custom-apps-nodegroup"
2022-12-19 16:38:10 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-custom-apps-nodegroup"
2022-12-19 16:39:08 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-custom-apps-nodegroup"
2022-12-19 16:39:08 [i] no tasks
2022-12-19 16:39:08 [✓] created 0 nodegroup(s) in cluster "dmi-demo"
2022-12-19 16:39:08 [i] nodegroup "custom-apps-nodegroup" has 2 node(s)
2022-12-19 16:39:08 [i] node "ip-10-50-155-253.ap-south-1.compute.internal" is ready
2022-12-19 16:39:08 [i] node "ip-10-50-179-180.ap-south-1.compute.internal" is ready
2022-12-19 16:39:08 [i] waiting for at least 2 node(s) to become ready in "custom-apps-nodegroup"
2022-12-19 16:39:08 [i] nodegroup "custom-apps-nodegroup" has 2 node(s)
2022-12-19 16:39:08 [i] node "ip-10-50-155-253.ap-south-1.compute.internal" is ready
2022-12-19 16:39:08 [i] node "ip-10-50-179-180.ap-south-1.compute.internal" is ready
2022-12-19 16:39:08 [✓] created 1 managed nodegroup(s) in cluster "dmi-demo"
2022-12-19 16:39:09 [i] checking security group configuration for all nodegroups
```

3. Create Tools stack nodegroup for Elasticsearch, logstash, Kafka.

```
eksctl create nodegroup --cluster=dmi-demo --region=ap-south-1 --managed --name=tools-elk  
--instance-types=r5a.large --nodes-min=2 --nodes-max=8 --node-volume-type gp3  
--node-volume-size 15 --asg-access --enable-ssm --managed --node-private-networking  
--ssh-access=true --ssh-public-key=demi-demo --profile dmi-eks
```

```
rupam.saini@scriptbox: $ eksctl create nodegroup --cluster=dmi-demo --region=ap-south-1 --managed --name=tools-elk --instance-types=r5a.large --nodes-min=2 --nodes-max=8 --node-volume-type gp3 --node-volume-size 15 --asg-access --enable-ssm --managed --node-private-networking --ssh-access=true --ssh-public-key=demi-demo --profile dmi-eks  
2022-12-19 16:51:40 [i] will use version 1.23 for new nodegroup(s) based on control plane version  
2022-12-19 16:51:40 [!] SSM is now enabled by default; 'ssh.enableSSM' is deprecated and will be removed in a future release  
2022-12-19 16:51:41 [i] nodegroup "tools-elk" will use "" [AmazonLinux2/1.23]  
2022-12-19 16:51:41 [i] using EC2 key pair %!q(*string=<nil>)  
2022-12-19 16:51:42 [i] 2 existing nodegroup(s) (application-worker-node,custom-apps-nodegroup) will be excluded  
2022-12-19 16:51:42 [i] 1 nodegroup (tools-elk) was included (based on the include/exclude rules)  
2022-12-19 16:51:42 [i] will create a CloudFormation stack for each of 1 managed nodegroups in cluster "dmi-demo"  
2022-12-19 16:51:42 [i]  
2 sequential tasks: { fix cluster compatibility, 1 task: { 1 task: { create managed nodegroup "tools-elk" } }  
}  
2022-12-19 16:51:42 [i] checking cluster stack for missing resources  
2022-12-19 16:51:42 [i] cluster stack has all required resources  
2022-12-19 16:51:43 [i] building managed nodegroup stack "eksctl-dmi-demo-nodegroup-tools-elk"  
2022-12-19 16:51:43 [i] deploying stack "eksctl-dmi-demo-nodegroup-tools-elk"  
2022-12-19 16:51:43 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-elk"  
2022-12-19 16:52:13 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-elk"  
2022-12-19 16:52:54 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-elk"  
2022-12-19 16:54:06 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-elk"  
2022-12-19 16:55:26 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-elk"  
2022-12-19 16:55:26 [i] no tasks  
2022-12-19 16:55:26 [✓] created 0 nodegroup(s) in cluster "dmi-demo"  
2022-12-19 16:55:26 [i] nodegroup "tools-elk" has 2 node(s)  
2022-12-19 16:55:26 [i] node "ip-10-50-107-80.ap-south-1.compute.internal" is ready  
2022-12-19 16:55:26 [i] node "ip-10-50-132-35.ap-south-1.compute.internal" is ready  
2022-12-19 16:55:26 [i] waiting for at least 2 node(s) to become ready in "tools-elk"  
2022-12-19 16:55:26 [i] nodegroup "tools-elk" has 2 node(s)  
2022-12-19 16:55:26 [i] node "ip-10-50-107-80.ap-south-1.compute.internal" is ready  
2022-12-19 16:55:26 [i] node "ip-10-50-132-35.ap-south-1.compute.internal" is ready  
2022-12-19 16:55:26 [✓] created 1 managed nodegroup(s) in cluster "dmi-demo"  
2022-12-19 16:55:27 [i] checking security group configuration for all nodegroups  
2022-12-19 16:55:27 [i] all nodegroups have up-to-date cloudformation templates
```

4. Create Node group for Consul service monitoring.

```
eksctl create nodegroup --cluster=dmi-demo --region=ap-south-1 --managed  
--name=consul-monitoring --instance-types=r5a.large --nodes-min=2 --nodes-max=8  
--node-volume-type gp3 --node-volume-size 15 --asg-access --enable-ssm --managed  
--node-private-networking --ssh-access=true --ssh-public-key=demi-demo --profile dmi-eks
```

```
rupam.saini@scriptbox: $ eksctl create nodegroup --cluster=dmi-demo --region=ap-south-1 --managed --name=consul-monitoring --instance-types=r5a.large --nodes-min=2 --nodes-max=8 --node-volume-type gp3 --node-volume-size 15 --asg-access --enable-ssm --managed --node-private-networking --ssh-access=true --ssh-public-key=demi-demo --profile dmi-eks  
2022-12-19 17:00:44 [i] will use version 1.23 for new nodegroup(s) based on control plane version  
2022-12-19 17:00:44 [!] SSM is now enabled by default; `ssh.enableSSM` is deprecated and will be removed in a future release  
2022-12-19 17:00:46 [i] nodegroup "consul-monitoring" will use "" [AmazonLinux2/1.23]  
2022-12-19 17:00:46 [i] using EC2 key pair %!q(*string=<nil>)  
2022-12-19 17:00:46 [i] 3 existing nodegroup(s) (application-worker-node,custom-apps-nodegroup,tools-elk) will be excluded  
2022-12-19 17:00:46 [i] 1 nodegroup (consul-monitoring) was included (based on the include/exclude rules)  
2022-12-19 17:00:46 [i] will create a CloudFormation stack for each of 1 managed nodegroups in cluster "dmi-demo"  
2022-12-19 17:00:46 [i]  
2 sequential tasks: { fix cluster compatibility, 1 task: { 1 task: { create managed nodegroup "consul-monitoring" } } }  
2022-12-19 17:00:46 [i] checking cluster stack for missing resources  
2022-12-19 17:00:47 [i] cluster stack has all required resources  
2022-12-19 17:00:47 [i] building managed nodegroup stack "eksctl-dmi-demo-nodegroup-consul-monitoring"  
2022-12-19 17:00:47 [i] deploying stack "eksctl-dmi-demo-nodegroup-consul-monitoring"  
2022-12-19 17:00:47 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-consul-monitoring"  
2022-12-19 17:01:18 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-consul-monitoring"  
2022-12-19 17:02:14 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-consul-monitoring"  
2022-12-19 17:03:33 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-consul-monitoring"  
2022-12-19 17:05:09 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-consul-monitoring"  
2022-12-19 17:05:09 [i] no tasks  
2022-12-19 17:05:09 [✓] created 0 nodegroup(s) in cluster "dmi-demo"  
2022-12-19 17:05:09 [i] nodegroup "consul-monitoring" has 2 node(s)  
2022-12-19 17:05:09 [i] node "ip-10-50-123-195.ap-south-1.compute.internal" is ready  
2022-12-19 17:05:09 [i] node "ip-10-50-176-110.ap-south-1.compute.internal" is ready  
2022-12-19 17:05:09 [i] waiting for at least 2 node(s) to become ready in "consul-monitoring"  
2022-12-19 17:05:09 [i] nodegroup "consul-monitoring" has 2 node(s)  
2022-12-19 17:05:09 [i] node "ip-10-50-123-195.ap-south-1.compute.internal" is ready  
2022-12-19 17:05:09 [i] node "ip-10-50-176-110.ap-south-1.compute.internal" is ready  
2022-12-19 17:05:09 [✓] created 1 managed nodegroup(s) in cluster "dmi-demo"  
2022-12-19 17:05:10 [i] checking security group configuration for all nodegroups  
2022-12-19 17:05:10 [i] all nodegroups have up-to-date cloudformation templates
```

5. Create Node Group for redis tools.

```
eksctl create nodegroup --cluster=dmi-demo --region=ap-south-1 --managed --name=tools-redis  
--instance-types=r5a.large --nodes-min=2 --nodes-max=8 --node-volume-type gp3  
--node-volume-size 15 --asg-access --enable-ssm --managed --node-private-networking  
--ssh-access=true --ssh-public-key=demi-demo --profile dmi-eks
```

```
rupam.saini@scriptbox:~$ eksctl create nodegroup --cluster=dmi-demo --region=ap-south-1 --managed --name=tools-redis --instance-types=r5a.large --nodes-min=2  
--nodes-max=8 --node-volume-type gp3 --node-volume-size 15 --asg-access --enable-ssm --managed --node-private-networking --ssh-access=true --ssh-public-key=demi-demo --profile dmi-eks  
2022-12-19 17:03:33 [i] will use version 1.23 for new nodegroup(s) based on control plane version  
2022-12-19 17:03:33 [!] SSM is now enabled by default; `ssh.enableSSM` is deprecated and will be removed in a future release  
2022-12-19 17:03:34 [i] nodegroup "tools-redis" will use "" [AmazonLinux2/1.23]  
2022-12-19 17:03:34 [i] using EC2 key pair %!q(*string=<nil>)  
2022-12-19 17:03:35 [i] 4 existing nodegroup(s) (application-worker-node,consul-monitoring,custom-apps-nodegroup,tools-elk) will be excluded  
2022-12-19 17:03:35 [i] 1 nodegroup (tools-redis) was included (based on the include/exclude rules)  
2022-12-19 17:03:35 [i] will create a CloudFormation stack for each of 1 managed nodegroups in cluster "dmi-demo"  
2022-12-19 17:03:35 [i]  
2 sequential tasks: { fix cluster compatibility, 1 task: { 1 task: { create managed nodegroup "tools-redis" } } }  
2022-12-19 17:03:35 [i] checking cluster stack for missing resources  
2022-12-19 17:03:35 [i] cluster stack has all required resources  
2022-12-19 17:03:36 [i] building managed nodegroup stack "eksctl-dmi-demo-nodegroup-tools-redis"  
2022-12-19 17:03:36 [i] deploying stack "eksctl-dmi-demo-nodegroup-tools-redis"  
2022-12-19 17:03:36 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-redis"  
2022-12-19 17:04:06 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-redis"  
2022-12-19 17:05:06 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-redis"  
2022-12-19 17:06:37 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-redis"  
2022-12-19 17:08:17 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-redis"  
2022-12-19 17:08:18 [i] no tasks  
2022-12-19 17:08:18 [✓] created 0 nodegroup(s) in cluster "dmi-demo"  
2022-12-19 17:08:18 [i] nodegroup "tools-redis" has 2 node(s)  
2022-12-19 17:08:18 [i] node "ip-10-50-137-122.ap-south-1.compute.internal" is ready  
2022-12-19 17:08:18 [i] node "ip-10-50-182-210.ap-south-1.compute.internal" is ready  
2022-12-19 17:08:18 [i] waiting for at least 2 node(s) to become ready in "tools-redis"  
2022-12-19 17:08:18 [i] nodegroup "tools-redis" has 2 node(s)  
2022-12-19 17:08:18 [i] node "ip-10-50-137-122.ap-south-1.compute.internal" is ready  
2022-12-19 17:08:18 [i] node "ip-10-50-182-210.ap-south-1.compute.internal" is ready  
2022-12-19 17:08:18 [✓] created 1 managed nodegroup(s) in cluster "dmi-demo"  
2022-12-19 17:08:19 [i] checking security group configuration for all nodegroups  
2022-12-19 17:08:19 [i] all nodegroups have up-to-date cloudformation templates
```

6. Create Node group for Nginx Ingress controller setup.

```
eksctl create nodegroup --cluster=dmi-demo --region=ap-south-1 --managed --name=tools-nginx  
--instance-types=r5a.large --nodes-min=2 --nodes-max=8 --node-volume-type gp3  
--node-volume-size 15 --asg-access --enable-ssm --managed --node-private-networking  
--ssh-access=true --ssh-public-key=demi-demo --profile dmi-eks
```

```
rupam.saini@scriptbox: $ eksctl create nodegroup --cluster=dmi-demo --region=ap-south-1 --managed --name=tools-nginx --instance-types=r5a.large --nodes-min=2  
--nodes-max=8 --node-volume-type gp3 --node-volume-size 15 --asg-access --enable-ssm --managed --node-private-networking --ssh-access=true --ssh-public-key=d  
mi-demo --profile dmi-eks  
2022-12-19 17:06:39 [i] will use version 1.23 for new nodegroup(s) based on control plane version  
2022-12-19 17:06:39 [!] SSM is now enabled by default; `ssh.enableSSM` is deprecated and will be removed in a future release  
2022-12-19 17:06:40 [i] nodegroup "tools-nginx" will use "" [AmazonLinux2/1.23]  
2022-12-19 17:06:40 [i] using EC2 key pair %!q(*string=<nil>)  
2022-12-19 17:06:41 [i] 5 existing nodegroup(s) (application-worker-node,consul-monitoring,custom-apps-nodegroup,tools-elk,tools-redis) will be excluded  
2022-12-19 17:06:41 [i] 1 nodegroup (tools-nginx) was included (based on the include/exclude rules)  
2022-12-19 17:06:41 [i] will create a CloudFormation stack for each of 1 managed nodegroups in cluster "dmi-demo"  
2022-12-19 17:06:41 [i]  
2 sequential tasks: { fix cluster compatibility, 1 task: { 1 task: { create managed nodegroup "tools-nginx" } }  
}  
2022-12-19 17:06:41 [i] checking cluster stack for missing resources  
2022-12-19 17:06:42 [i] cluster stack has all required resources  
2022-12-19 17:06:42 [i] building managed nodegroup stack "eksctl-dmi-demo-nodegroup-tools-nginx"  
2022-12-19 17:06:43 [i] deploying stack "eksctl-dmi-demo-nodegroup-tools-nginx"  
2022-12-19 17:06:43 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-nginx"  
2022-12-19 17:07:13 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-nginx"  
2022-12-19 17:08:13 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-nginx"  
2022-12-19 17:09:55 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-nginx"  
2022-12-19 17:11:19 [i] waiting for CloudFormation stack "eksctl-dmi-demo-nodegroup-tools-nginx"  
2022-12-19 17:11:19 [i] no tasks  
2022-12-19 17:11:19 [✓] created 0 nodegroup(s) in cluster "dmi-demo"  
2022-12-19 17:11:20 [i] nodegroup "tools-nginx" has 2 node(s)  
2022-12-19 17:11:20 [i] node "ip-10-50-113-234.ap-south-1.compute.internal" is ready  
2022-12-19 17:11:20 [i] node "ip-10-50-144-27.ap-south-1.compute.internal" is ready  
2022-12-19 17:11:20 [i] waiting for at least 2 node(s) to become ready in "tools-nginx"  
2022-12-19 17:11:20 [i] nodegroup "tools-nginx" has 2 node(s)  
2022-12-19 17:11:20 [i] node "ip-10-50-113-234.ap-south-1.compute.internal" is ready  
2022-12-19 17:11:20 [i] node "ip-10-50-144-27.ap-south-1.compute.internal" is ready  
2022-12-19 17:11:20 [✓] created 1 managed nodegroup(s) in cluster "dmi-demo"  
2022-12-19 17:11:20 [i] checking security group configuration for all nodegroups  
2022-12-19 17:11:20 [i] all nodegroups have up-to-date cloudformation templates
```

Once above EKS cluster and Node group created then add label to nodegroups:-

1.click on consul-monitoring nodegroup and click on edit button in right corner.

Key: **vrt-consul**

Value: **true**

EKS > Clusters > dmi-demo > Node group: consul-monitoring

consul-monitoring

Node group configuration [Info](#)

Kubernetes version 1.23	AMI type Info AL2_x86_64	Launch template eksctl-dmi-demo-nodegroup-consul-monitoring	Status Active
AMI release version Info 1.23.13-20221112	Instance types r5a.large	Launch template version 1	Disk size Specified in launch template

Details Nodes Health issues (0) **Kubernetes labels** Update config Kubernetes taints Update history Tags

Kubernetes labels (3)

Key	Value
alpha.eksctl.io/cluster-name	dmi-demo
alpha.eksctl.io/nodegroup-name	consul-monitoring
vrt-consul	true

2. Go to the kubernetes labels section and add labels like in the image below. The label is vrt-consul you have to change the label name in another nodegroup. **For example, for an tools-elk node the label is vrt-elk.** Like below screenshots.

Kubernetes labels Info

Key	Value	
alpha.eksctl.io/cluster-name	dmi-demo	Remove label
alpha.eksctl.io/nodegroup-name	consul-monitoring	Remove label
vrt-consul	true	Remove label

[Add label](#)

Remaining labels available to add: 47

Kubernetes labels (6)

<input type="text"/> Filter by key or value		< 1 >
Key	Value	▼
alpha.eksctl.io/cluster-name	dmi-demo	▼
alpha.eksctl.io/nodegroup-name	tools-elk	▼
vrt-zookeeper	true	▼
vrt-redis	true	▼
vrt-kafka	true	▼
vrt-elk	true	▼

- [Details](#)
- [Nodes](#)
- [Health issues 0](#)
- [**Kubernetes labels**](#)
- [Update config](#)
- [Kubernetes taints](#)
- [Update history](#)
- [Tags](#)

Kubernetes labels (3)

<input type="text"/> Filter by key or value		< 1 >
Key	Value	▼
alpha.eksctl.io/cluster-name	dmi-demo	▼
alpha.eksctl.io/nodegroup-name	tools-nginx	▼
vrt-nginx	true	▼

- [Details](#)
- [Nodes](#)
- [Health issues 0](#)
- [**Kubernetes labels**](#)
- [Update config](#)
- [Kubernetes taints](#)
- [Update history](#)
- [Tags](#)

Kubernetes labels (3)

<input type="text"/> Filter by key or value		< 1 >
Key	Value	▼
alpha.eksctl.io/cluster-name	dmi-demo	▼
alpha.eksctl.io/nodegroup-name	tools-redis	▼
vrt-redis	true	▼

3. Go to the kubernetes taint section and add taints like in the image below and hit save button.

The screenshot shows a 'Kubernetes taints' configuration page. It has three main fields: 'Key' (containing 'key'), 'Value' (containing 'persistTool'), and 'Effect' (set to 'NoSchedule'). Below these fields is a 'Remove taint' button. At the bottom left is an 'Add taint' button, and at the bottom center is a message: 'Remaining taints available to add: 49'.

Key	Value	Effect	
key	persistTool	NoSchedule	▼ Remove taint

Add taint
Remaining taints available to add: 49

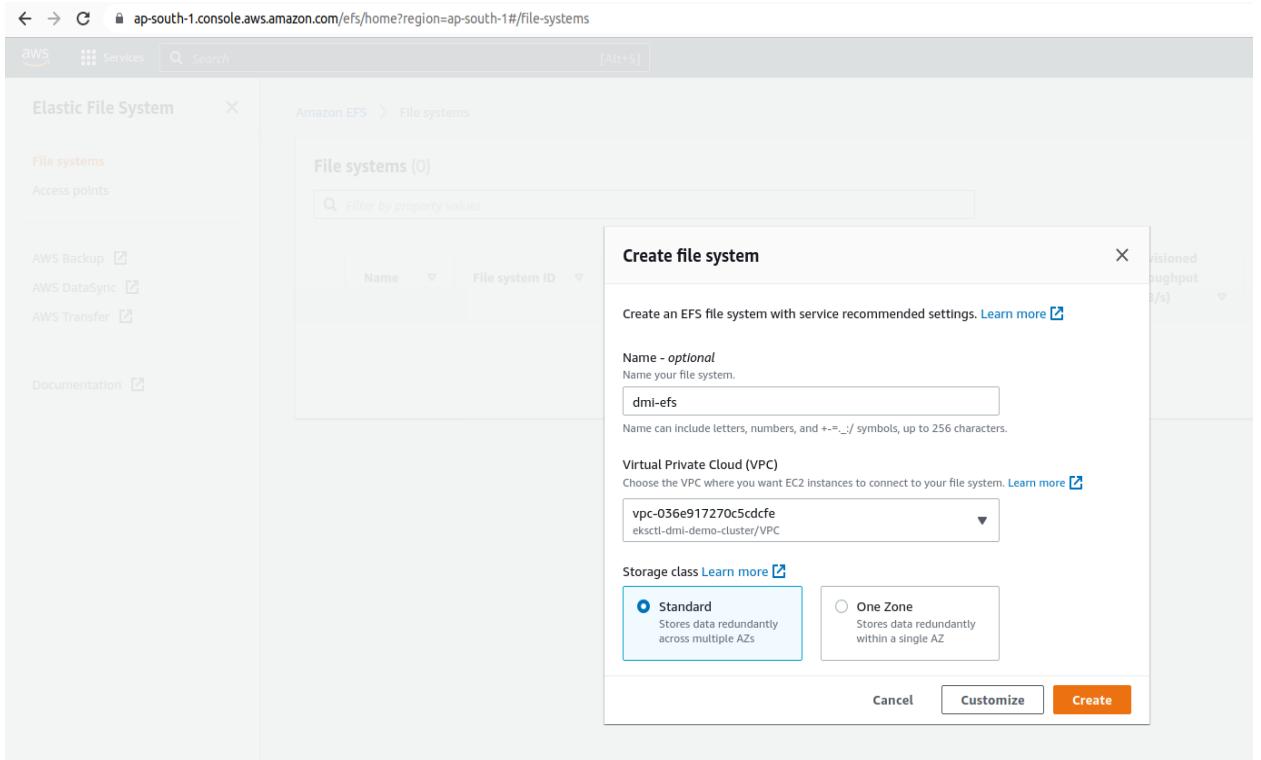
4. Follow this process for all node-groups except application node and custom application node.

Create EFS:

Stateful applications like (Elasticsearch, Logstash, Kafka, PostgreSQL, Consul, Zookeeper) store the data. In which data may be lost during restart pods.

To prevent this, We keeps data on NFS storage, Which is available on AWS service as a EFS.

1. EFS for persistence volume and persistent volume claim for deploying the tools.
2. Go to your aws console and search for EFS and create EFS and select your cluster vpc.



Jumpbox:

1. Create an ec2 instance into your cluster vpc public subnet.(search your vpc in cluster networking section)
2. Now login to your ec2 instance and create an efs folder on home directory.

```
ubuntu@ip-10-50-90-216:~$ mkdir efs
ubuntu@ip-10-50-90-216:~$ ls
efs
ubuntu@ip-10-50-90-216:~$ █
```

3. Now create a security group in your cluster vpc and allow the nfs inbound rule with your vpc cidr.

The screenshot shows the 'Create security group' page in the AWS EC2 console. The 'Basic details' section includes fields for 'Security group name' (dmi-efs-sg), 'Description' (dmi-efs-sg), and 'VPC' (vpc-036e917270c5cdcf). The 'Inbound rules' section contains one rule: Type: NFS, Protocol: TCP, Port range: 2049, Source: Custom (10.50.0.0/16), Description: dmi-eks-vpc. The 'Outbound rules' section is empty.

4. Now go to your aws console and go to the EFS dashboard.click on your EFS name. And go to the network section and add the security group which you created earlier.

The screenshot shows the AWS EFS console interface. On the left, there's a sidebar with navigation links like 'File systems' and 'AWS Backup'. The main content area displays the 'General' tab for the file system 'dmi-efs (fs-03465cb72cfbea23e)'. Key details shown include:

- Performance mode:** General Purpose
- Encryption:** Enabled
- File system state:** Available
- DNS name:** fs-03465cb72cfbea23e.efs.ap-south-1.amazonaws.com
- Availability zone:** Standard

Below the general tab, there are tabs for 'Metered size', 'Monitoring', 'Tags', 'File system policy', 'Access points', 'Network', and 'Replication'. The 'Metered size' tab is selected, showing a pie chart indicating 6.00 kB total size, all in Standard / One Zone.

- Add your security group in the networking section and click the save button.

The screenshot shows the 'Network access' section of the AWS EFS console. It includes:

- Virtual Private Cloud (VPC):** A dropdown menu showing 'vpc-036e917270c5dcfe' and 'eksctl-arni-demo-cluster/VPC'.
- Mount targets:** A table listing three availability zones with their corresponding subnet IDs, IP addresses, and security groups assigned.

Availability zone	Subnet ID	IP address	Security groups
ap-south-1a	subnet-05fb346fe170bfe6e	10.50.102.38	sg-051ef7721b4ba8c27 dmi-efs-ig
ap-south-1b	subnet-00873d5a77938bc6b	10.50.165.211	sg-051ef7721b4ba8c27 dmi-efs-ig
ap-south-1c	subnet-001ce7bbc13a0498f	10.50.130.242	sg-051ef7721b4ba8c27 dmi-efs-ig

At the bottom, there are 'Add mount target' and 'Save' buttons.

6. Now go to your jumpbox security group and allow NFS port 2049 on inbound rule with your EKS cluster VPC CIDR.

The screenshot shows the AWS EC2 console under the 'Security Groups' section. A specific security group, 'sg-0d4611371a0a31e9a - launch-wizard-1', is selected. The 'Edit inbound rules' tab is active. There are two rules listed:

- sgr-07aa5a76b9a1e64b7**: Type: NFS, Protocol: TCP, Port range: 2049, Source: Custom (CIDR 10.50.0.0/16), Description: dmi-vpc cidr.
- sgr-0c032af1921491ba1**: Type: SSH, Protocol: TCP, Port range: 22, Source: Custom (CIDR 0.0.0.0/0), Description: (empty).

At the bottom right, there are 'Cancel', 'Preview changes', and 'Save rules' buttons. The status bar at the bottom indicates 'Feedback Looking for language selection? Find it in the new Unified Settings' and '© 2022, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences'.

7. Now again go to your efs and click on your efs and in the right corner click on the **Attach** option.

The screenshot shows the AWS EFS console under the 'File systems' section. A specific file system, 'dmi-efs (fs-03465cb72cfbea23e)', is selected. A modal dialog box titled 'Attach' is open, providing instructions for mounting the EFS file system:

- Mounting via DNS:** The radio button is selected, and the command is: `sudo mount -t efs -o tls fs-03465cb72cfbea23e:/ efs`.
- Mounting via IP:** An alternative method using IP address.
- Using the EFS mount helper:** A command: `sudo mount -t efs -o tls fs-03465cb72cfbea23e.efs.ap-south-1.amazonaws.com:/ efs`.
- Using the NFS client:** Another command: `sudo mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport fs-03465cb72cfbea23e.efs.ap-south-1.amazonaws.com:/ efs`.

Below the dialog, a table lists mounted volumes across different availability zones:

	Availability zone	Mount target ID	Subnet ID	Mount target state	IP address	Network interface ID	Security groups
	ap-south-1a	fsmt-0941bf046fb160355	subnet-05fb346fe170bfe6e	Available	10.50.102.38	eni-0e4ef5e449a27c1c2	sg-07c6e0448f01e96d8 (default)
	ap-south-1b	fsmt-071fbe4a5f6a4043b	subnet-00873d5a77938bc8	Available	10.50.165.211	eni-006c0d001f713d212	sg-07c6e0448f01e96d8 (default)

8. Now copy the command “using the nfs” client and paste on your jumpbox home directory and hit enter.

9. If you find “**mount: /home/ubuntu/efs: bad option; for several filesystems (e.g. nfs, cifs) you might need a /sbin/mount.<type> helper program.**”

this error then runs below command and tries again.

sudo apt update

sudo apt install nfs-common

10. After this process you can verify your mount volume using the “**df -HT**” command and output should be like this.

```
ubuntu@ip-10-50-90-216: ~ $ df -HT
Filesystem           Type  Size  Used Avail Use% Mounted on
/dev/root            ext4  11G   1.9G  8.4G  18% /
tmpfs               tmpfs 496M    0  496M  0% /dev/shm
tmpfs               tmpfs 199M  865k  198M  1% /run
tmpfs               tmpfs  5.3M    0  5.3M  0% /run/lock
/dev/nvme0n1p15      vfat  110M  5.5M  104M  5% /boot/efi
tmpfs               tmpfs 100M  4.1k  100M  1% /run/user/1000
fs-03465cb72cfbea23e.efs.ap-south-1.amazonaws.com:/ nfs4  9.3E    0  9.3E  0% /home/ubuntu/efs
```

11. You can check your efs id at the end of output in above image.

12. Now we have to create directories which are used for PV or PVC.

13. Create directories for consul like in image below and assign permission to then “**sudo chmod 777**”.

```
ubuntu@ip-10-50-90-216: ~/efs/consul$ ll
total 20
drwxr-xr-x 5 root root 6144 Dec 21 05:06 .
drwxr-xr-x 5 root root 6144 Dec 21 05:08 ..
drwxrwxrwx 2 root root 6144 Dec 21 05:06 consul-0/
drwxrwxrwx 2 root root 6144 Dec 21 05:06 consul-1/
drwxrwxrwx 2 root root 6144 Dec 21 05:06 consul-2/
```

14. Create all directories like this.

```
ubuntu@ip-10-50-90-216: ~/efs/elasticsearch$ ll
total 32
drwxr-xr-x 8 root root 6144 Dec 21 05:07 .
drwxr-xr-x 5 root root 6144 Dec 21 05:08 ..
drwxrwxrwx 2 root root 6144 Dec 21 05:07 elasticsearch-0/
drwxrwxrwx 2 root root 6144 Dec 21 05:07 elasticsearch-1/
drwxrwxrwx 2 root root 6144 Dec 21 05:07 elasticsearch-2/
drwxrwxrwx 2 root root 6144 Dec 21 05:07 elasticsearch-3/
drwxrwxrwx 2 root root 6144 Dec 21 05:07 elasticsearch-master-0/
drwxrwxrwx 2 root root 6144 Dec 21 05:07 elasticsearch-master-1/
```

```
ubuntu@ip-10-50-90-216:~/efs/kafka$ ll
total 24
drwxr-xr-x 6 root root 6144 Dec 21 05:12 .
drwxr-xr-x 5 root root 6144 Dec 21 05:08 ../
drwxrwxrwx 2 root root 6144 Dec 21 05:12 kafka-0/
drwxrwxrwx 2 root root 6144 Dec 21 05:12 kafka-1/
drwxrwxrwx 2 root root 6144 Dec 21 05:12 kafka-2/
drwxrwxrwx 2 root root 6144 Dec 21 05:12 kafka-3/
```

```
ubuntu@ip-10-50-90-216:~/efs/logging-agent$ ll
total 20
drwxr-xr-x 5 root root 6144 Dec 21 05:13 .
drwxr-xr-x 6 root root 6144 Dec 21 05:13 ../
drwxrwxrwx 2 root root 6144 Dec 21 05:13 logging-agent-0/
drwxrwxrwx 2 root root 6144 Dec 21 05:13 logging-agent-1/
drwxrwxrwx 2 root root 6144 Dec 21 05:13 logging-agent-2/
```

```
ubuntu@ip-10-50-90-216:~/efs/postgres$ ll
total 12
drwxr-xr-x 3 root root 6144 Dec 21 05:15 .
drwxr-xr-x 8 root root 6144 Dec 21 05:17 ../
drwxrwxrwx 2 root root 6144 Dec 21 05:15 postgres-0/
```

```
ubuntu@ip-10-50-90-216:~/efs/redis$ ll
total 20
drwxr-xr-x 5 root root 6144 Dec 21 05:18 .
drwxr-xr-x 8 root root 6144 Dec 21 05:17 ../
drwxrwxrwx 2 root root 6144 Dec 21 05:18 redis-0/
drwxrwxrwx 2 root root 6144 Dec 21 05:18 redis-1/
drwxrwxrwx 2 root root 6144 Dec 21 05:18 redis-2/
```

15. To install kubectl on jumpbox . follow this [Document](#).
16. Then you have to install awscli using this command. “**`sudo apt install awscli`**”.
17. Now configure your aws credentials using “aws configure” command.
18. After configure your aws credentials you have to update your cluster in jumpbox using below command.
`aws eks update-kubeconfig --name "your-cluster-name" --region "aws-region"`
 Replace “your-cluster-name” and “aws-region” with your requirements.
 And your output is like the image below.

```
ubuntu@ip-10-50-90-216:~$ aws eks update-kubeconfig --name dmi-demo --region ap-south-1
Added new context arn:aws:eks:ap-south-1:377211472778:cluster/dmi-demo to /home/ubuntu/.kube/config
```

19. After these steps you can verify your cluster with “kubectl get ns” command.

```
ubuntu@ip-10-50-90-216:~$ kubectl get ns
NAME          STATUS   AGE
default        Active   42h
kube-node-lease  Active   42h
kube-public    Active   42h
kube-system    Active   42h
```

20. Now create a new namespace with any name you want.

```
kubectl create ns "namespace-name"
```

```
ubuntu@ip-10-50-90-216:~$ kubectl get ns
NAME          STATUS   AGE
default        Active   43h
dmi           Active   8s
kube-node-lease  Active   43h
kube-public    Active   43h
kube-system    Active   43h
```

21. Now you have to install aws CSI and EBS driver using this [link](#).
22. After completing this step you can verify your driver using.
“**kubectl get pods -n kube-system**” and output should be like this.
23. Check these pods on your output .

ebs-csi-controller-988ff97c9-fr722	6/6	Running	0	33m	
ebs-csi-controller-988ff97c9-kjfb4	6/6	Running	0	33m	
ebs-csi-node-2v6d2	3/3	Running	0	34m	
ebs-csi-node-674px	3/3	Running	0	34m	
ebs-csi-node-9b8tn	3/3	Running	0	34m	
ebs-csi-node-9k88x	3/3	Running	0	34m	
ebs-csi-node-bzwmn	3/3	Running	0	34m	
ebs-csi-node-cd9zg	3/3	Running	0	34m	
ebs-csi-node-djh4q	3/3	Running	0	34m	
ebs-csi-node-dzdzk	3/3	Running	0	34m	
ebs-csi-node-jcvzz	3/3	Running	0	34m	
ebs-csi-node-knvsh	3/3	Running	0	34m	
ebs-csi-node-mnh6j	3/3	Running	0	34m	
ebs-csi-node-s5kwt	3/3	Running	0	34m	
ebs-csi-node-x8xvr	3/3	Running	0	34m	
efs-csi-controller-7f49c7865-lvxh9	3/3	Running	0	26m	
efs-csi-controller-7f49c7865-qc5ms	3/3	Running	0	26m	
efs-csi-node-2bwxl	3/3	Running	0	26m	
efs-csi-node-6ktcm	3/3	Running	0	26m	
efs-csi-node-6wqgc	3/3	Running	0	26m	
efs-csi-node-bnwzd	3/3	Running	0	26m	
efs-csi-node-g9k9l	3/3	Running	0	26m	
efs-csi-node-jqn5f	3/3	Running	0	26m	
efs-csi-node-mc8gk	3/3	Running	0	26m	
efs-csi-node-qzbtd	3/3	Running	0	26m	
efs-csi-node-vxstx	3/3	Running	0	26m	
efs-csi-node-wb8wp	3/3	Running	0	26m	
efs-csi-node-wz2lc	3/3	Running	0	26m	
efs-csi-node-zcrxn	3/3	Running	0	26m	
efs-csi-node-zlfgn	3/3	Running	0	26m	
total	66/66	Running	0	26m	

24. After verifying these you have to deploy the pv and pvc form attach file.
With **kubectl apply -f <filename> -n <namespace>**

```
ubuntu@ip-10-50-90-216:~/vrt-manifest-yaml/vrt-pvc$ kubectl apply -f . -n dmi
persistentvolume/efs-consul-0 created
persistentvolumeclaim/data-consul-0 created
persistentvolume/efs-consul-1 created
persistentvolumeclaim/data-consul-1 created
persistentvolume/efs-consul-2 created
persistentvolumeclaim/data-consul-2 created
persistentvolume/efs-consul-3 created
persistentvolumeclaim/data-consul-3 created
persistentvolume/efs-consul-4 created
persistentvolumeclaim/data-consul-4 created
persistentvolume/efs-elasticsearch-data-0 created
persistentvolumeclaim/data-elasticsearch-data-0 created
persistentvolume/efs-elasticsearch-data-1 created
persistentvolumeclaim/data-elasticsearch-data-1 created
persistentvolume/efs-elasticsearch-data-2 created
persistentvolumeclaim/data-elasticsearch-data-2 created
persistentvolume/efs-elasticsearch-data-3 created
persistentvolumeclaim/data-elasticsearch-data-3 created
persistentvolume/efs-elasticsearch-master-0 created
persistentvolumeclaim/data-elasticsearch-master-0 created
persistentvolume/efs-elasticsearch-master-1 created
persistentvolumeclaim/data-elasticsearch-master-1 created
persistentvolume/efs-kafka-0 created
persistentvolumeclaim/data-kafka1-0 created
persistentvolume/efs-kafka-1 created
persistentvolumeclaim/data-kafka1-1 created
persistentvolume/efs-kafka-2 created
persistentvolumeclaim/data-kafka1-2 created
persistentvolume/efs-kafka-3 created
persistentvolumeclaim/data-kafka1-3 created
persistentvolume/efs-kafka-4 created
```

Helm chart:-

1. Now you have to deploy the helm chart from the given yaml file and below commands.

"helm repo add bitnami <https://charts.bitnami.com/bitnami>"

"helm install consul -f consul.yaml bitnami/consul -n dmi --version 10.7.7"

Run these commands where you store the yaml files .

Your output should be like this.

```
ubuntu@ip-10-50-90-216:~/vrt-manifest-yaml/helm-charts$ helm install consul -f consul.yaml bitnami/consul -n vrt --version 10.7.7 -n dmi
NAME: consul
LAST DEPLOYED: Wed Dec 21 07:23:22 2022
NAMESPACE: dmi
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: consul
CHART VERSION: 10.7.7
APP VERSION: 1.12.3

** Please be patient while the chart is being deployed **

Consul can be accessed within the cluster on port 8300 at consul-headless.dmi.svc.cluster.local

In order to access to the Consul Web UI:

1. Get the Consul URL by running these commands:

    kubectl port-forward --namespace dmi svc/consul-ui 80:80
    echo "Consul URL: http://127.0.0.1:80"

2. Access ASP.NET Core using the obtained URL.

Please take into account that you need to wait until a cluster leader is elected before using the Consul Web UI.

In order to check the status of the cluster you can run the following command:

    kubectl exec -it consul-0 -- consul members

Furthermore, to know which Consul node is the cluster leader run this other command:
```

helm install zookeeper -f zookeeper.yaml bitnami/zookeeper -n dmi --version 10.0.1

```
ubuntu@ip-10-50-90-216:~/vrt-manifest-yaml/helm-charts$ helm install zookeeper -f zookeeper.yaml bitnami/zookeeper -n dmi --version 10.0.1
NAME: zookeeper
LAST DEPLOYED: Wed Dec 21 07:34:30 2022
NAMESPACE: dmi
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: zookeeper
CHART VERSION: 10.0.1
APP VERSION: 3.8.0

** Please be patient while the chart is being deployed **

ZooKeeper can be accessed via port 2181 on the following DNS name from within your cluster:

    zoo1.dmi.svc.cluster.local

To connect to your ZooKeeper server run the following commands:

    export POD_NAME=$(kubectl get pods --namespace dmi -l "app.kubernetes.io/name=zookeeper,app.kubernetes.io/instance=zookeeper,app.kubernetes.io/component=zookeeper" -o jsonpath="{.items[0].metadata.name}")
    kubectl exec -it $POD_NAME -- zkCli.sh

To connect to your ZooKeeper server from outside the cluster execute the following commands:

    kubectl port-forward --namespace dmi svc/zoo1 2181:2181 &
    zkCli.sh 127.0.0.1:2181
```

helm install kafka -f kafka.yaml bitnami/kafka -n dmi --version 18.0.3

```
ubuntu@ip-10-50-90-216:~/vrt-manifest-yaml/helm-charts$ helm install kafka -f kafka.yaml bitnami/kafka -n dmi --version 18.0.3
NAME: kafka
LAST DEPLOYED: Wed Dec 21 07:36:03 2022
NAMESPACE: dmi
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: kafka
CHART VERSION: 18.0.3
APP VERSION: 3.2.0

** Please be patient while the chart is being deployed **

Kafka can be accessed by consumers via port 9092 on the following DNS name from within your cluster:

    kafka1.dmi.svc.cluster.local

Each Kafka broker can be accessed by producers via port 9092 on the following DNS name(s) from within your cluster:

    kafka1-0.kafka1-headless.dmi.svc.cluster.local:9092
    kafka1-1.kafka1-headless.dmi.svc.cluster.local:9092
    kafka1-2.kafka1-headless.dmi.svc.cluster.local:9092

To create a pod that you can use as a Kafka client run the following commands:

    kubectl run kafka1-client --restart='Never' --image docker.io/bitnami/kafka:3.2.0-debian-11-r12 --namespace dmi --command -- sleep infinity
    kubectl exec --tty -i kafka1-client --namespace dmi -- bash

PRODUCER:
    kafka-console-producer.sh \
```

helm install redis -f redis.yaml bitnami/redis -n dmi --version 17.0.6

```
ubuntu@ip-10-50-90-216:~/vrt-maintest-yaml/helm-charts$ helm install redis -f redis.yaml bitnami/redis -n dmi --version 17.0.6
NAME: redis
LAST DEPLOYED: Wed Dec 21 07:37:56 2022
NAMESPACE: dmi
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: redis
CHART VERSION: 17.0.6
APP VERSION: 7.0.4

** Please be patient while the chart is being deployed **

Redis&reg; can be accessed via port 6379 on the following DNS name from within your cluster:

    redis.dmi.svc.cluster.local for read only operations

For read/write operations, first access the Redis&reg; Sentinel cluster, which is available in port 26379 using the same domain name above.

To get your password run:

    export REDIS_PASSWORD=$(kubectl get secret --namespace dmi redis -o jsonpath=".data.redis-password" | base64 -d)

To connect to your Redis&reg; server:

1. Run a Redis&reg; pod that you can use as a client:

    kubectl run --namespace dmi redis-client --restart='Never' --env REDIS_PASSWORD=$REDIS_PASSWORD --image docker.io/bitnami/redis:7.0.4-debian-11-r4 --comm
and -- sleep infinity
```

helm install elasticsearch -f elasticsearch.yaml bitnami/elasticsearch -n dmi --version 18.2.6

Now you have to install aws alb ingress controller with below documentation.

1. Setup IAM for ServiceAccount

Create IAM OIDC provider
eksctl utils associate-iam-oidc-provider \
--region <aws-region> \
--cluster <your-cluster-name> \
--approve

2. Create an IAM policy called AWSLoadBalancerControllerIAMPolicy

```
aws iam create-policy \
--policy-name AWSLoadBalancerControllerIAMPolicy \
--policy-document file://iam-policy.json
```

```

ubuntu@ip-10-50-90-216:~$ aws iam create-policy --policy-name AWSLoadBalancerControllerIAMPolicy --policy-document file://iam-policy.json
POLICY arn:aws:iam::REDACTED:policy/AWSLoadBalancerControllerIAMPolicy      0      2023-01-04T07:23:20+00:00      v1      True   /      0
ubuntu@ip-10-50-90-216:~$ eksctl create iamserviceaccount \
> --cluster=dmi-demo \
> --namespace=dmi \
> --name=aws-load-balancer-controller \
> --attach-policy-arn=arn:aws:iam::REDACTED:policy/AWSLoadBalancerControllerIAMPolicy \
> --approve
2023-01-04 07:28:07 [!] 1 iamserviceaccount (dmi/aws-load-balancer-controller) was included (based on the include/exclude rules)
2023-01-04 07:28:07 [!] serviceaccounts that exist in Kubernetes will be excluded, use --override-existing-serviceaccounts to override
2023-01-04 07:28:07 [!] 1 task:
  2 sequential sub-tasks:
    create IAM role for serviceaccount "dmi/aws-load-balancer-controller",
    create serviceaccount "dmi/aws-load-balancer-controller",
  } 2023-01-04 07:28:07 [!] building iamserviceaccount stack "eksctl-dmi-demo-addon-iamserviceaccount-dmi-aws-load-balancer-controller"
2023-01-04 07:28:08 [!] deploying stack "eksctl-dmi-demo-addon-iamserviceaccount-dmi-aws-load-balancer-controller"
2023-01-04 07:28:08 [!] waiting for CloudFormation stack "eksctl-dmi-demo-addon-iamserviceaccount-dmi-aws-load-balancer-controller"
2023-01-04 07:28:38 [!] waiting for CloudFormation stack "eksctl-dmi-demo-addon-iamserviceaccount-dmi-aws-load-balancer-controller"
2023-01-04 07:29:23 [!] waiting for CloudFormation stack "eksctl-dmi-demo-addon-iamserviceaccount-dmi-aws-load-balancer-controller"
2023-01-04 07:29:23 [!] created serviceaccount "dmi/aws-load-balancer-controller"
ubuntu@ip-10-50-90-216:~$ k9s
ubuntu@ip-10-50-90-216:~$ k9s
ubuntu@ip-10-50-90-216:~$ helm install aws-load-balancer-controller eks/aws-load-balancer-controller --set clusterName=dmi-demo -n dmi --set serviceAccount.create=false --set serviceAccount.name=aws-load-balancer-controller
NAME: aws-load-balancer-controller
LAST DEPLOYED: Wed Jan  4 07:30:32 2023
NAMESPACE: dmi
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
AWS Load Balancer controller installed!
ubuntu@ip-10-50-90-216:~$ k9s

```

3. Download IAM policy for the AWS Load Balancer Controller

```
curl -o iam-policy.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balance
```

```

ubuntu@ip-10-50-90-216:~$ curl -o iam-policy.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/main/docs/install/iam_policy.json
% Total    % Received % Xferd  Average Speed   Time     Time     Current
          Dload  Upload Total   Spent    Left Speed
100  7617  100  7617    0     0  15840      0 --:--:-- --:--:-- 15868

```

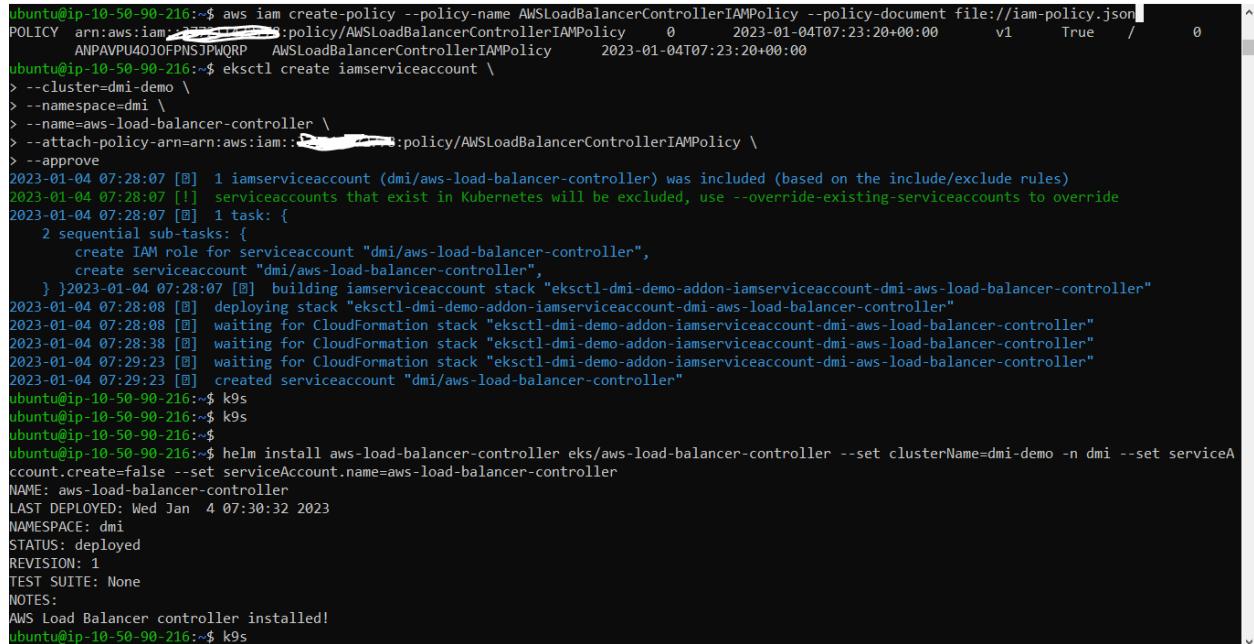
Take note of the policy ARN that is returned

4. Create a IAM role and ServiceAccount for the Load Balancer controller, use the ARN from the step above

```
eksctl create iamserviceaccount \
--cluster=<cluster-name> \
--namespace=kube-system \
--name=aws-load-balancer-controller \
--attach-policy-arn=arn:aws:iam:<AWS_ACCOUNT_ID>:policy/AWSLoadBalancerControllerIAMPolicy \
--approve
```

5. Now Install Helm chart for ALB Ingress controller:

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller --set
clusterName=dmi-demo -n dmi --set serviceAccount.create=false --set
serviceAccount.name=aws-load-balancer-controller
```



The terminal window shows the command being run and its output. It starts with creating an IAM policy, then using eksctl to create a service account, and finally running the helm install command. The output includes logs from eksctl showing the creation of a CloudFormation stack and the deployment of the controller.

```
ubuntu@ip-10-50-90-216:~$ aws iam create-policy --policy-name AWSSLoadBalancerControllerIAMPolicy --policy-document file://iam-policy.json
POLICY arn:aws:iam::██████████:policy/AWSLoadBalancerControllerIAMPolicy          0   2023-01-04T07:23:20+00:00      v1     True   /
ANPAVPU4OJOFPNSJPWQRP  AWSLoadBalancerControllerIAMPolicy      2023-01-04T07:23:20+00:00
ubuntu@ip-10-50-90-216:~$ eksctl create iamserviceaccount \
> --cluster=dmi-demo \
> --namespace=dmi \
> --name=aws-load-balancer-controller \
> --attach-policy-arm=arn:aws:iam::██████████:policy/AWSLoadBalancerControllerIAMPolicy \
> --approve
2023-01-04 07:28:07 [!] 1 iamserviceaccount (dmi/aws-load-balancer-controller) was included (based on the include/exclude rules)
2023-01-04 07:28:07 [!]  serviceaccounts that exist in Kubernetes will be excluded, use --override-existing-serviceaccounts to override
2023-01-04 07:28:07 [!] 1 task:
  2 sequential sub-tasks:
    create IAM role for serviceaccount "dmi/aws-load-balancer-controller",
    create serviceaccount "dmi/aws-load-balancer-controller",
  }2023-01-04 07:28:07 [!]  building iamserviceaccount stack "eksctl-dmi-demo-addon-iamserviceaccount-dmi-aws-load-balancer-controller"
2023-01-04 07:28:08 [!]  deploying stack "eksctl-dmi-demo-addon-iamserviceaccount-dmi-aws-load-balancer-controller"
2023-01-04 07:28:08 [!]  waiting for CloudFormation stack "eksctl-dmi-demo-addon-iamserviceaccount-dmi-aws-load-balancer-controller"
2023-01-04 07:28:38 [!]  waiting for CloudFormation stack "eksctl-dmi-demo-addon-iamserviceaccount-dmi-aws-load-balancer-controller"
2023-01-04 07:29:23 [!]  waiting for CloudFormation stack "eksctl-dmi-demo-addon-iamserviceaccount-dmi-aws-load-balancer-controller"
2023-01-04 07:29:23 [!]  created serviceaccount "dmi/aws-load-balancer-controller"
ubuntu@ip-10-50-90-216:~$ k9s
ubuntu@ip-10-50-90-216:~$ k9s
ubuntu@ip-10-50-90-216:~$ helm install aws-load-balancer-controller eks/aws-load-balancer-controller --set clusterName=dmi-demo -n dmi --set serviceA
ccount.create=false --set serviceAccount.name=aws-load-balancer-controller
NAME: aws-load-balancer-controller
LAST DEPLOYED: Wed Jan  4 07:30:32 2023
NAMESPACE: dmi
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
AWS Load Balancer controller installed!
ubuntu@ip-10-50-90-216:~$ k9s
```

And your output should be like above images

Ingres:

Now you have to create an ingress rule with the given `ingres.yaml` file. But if you want your custom ingres rule you have to add rules in `ingres` file.

```
- path: /<path>/
  pathType: Prefix
  backend:
    service:
      name: <name>
      port:
        number: <portnumber>
```

Please change the values of `<path>` `<name>` `<portnumber>` according to your need. And you have to change annotation in `ingres.yaml` file

annotations:

```
kubernetes.io/ingress.class: alb
alb.ingress.kubernetes.io/listen-ports: '[{"HTTP": 80}, {"HTTPS": 443}]'
alb.ingress.kubernetes.io/action.ssl-redirect: '{"Type": "redirect", "RedirectConfig": {"Protocol": "HTTPS", "Port": 443, "StatusCode": "HTTP 301"}'}
```

```

alb.ingress.kubernetes.io/scheme: internet-facing
alb.ingress.kubernetes.io/target-type: ip
alb.ingress.kubernetes.io/subnets: <subnetsids>
alb.ingress.kubernetes.io/certificate-arn: <certificatearn>
alb.ingress.kubernetes.io/ssl-policy: ELBSecurityPolicy-TLS-1-2-Ext-2018-06

```

Replace these two lines <subnetsids> lines with <certificatearn> with your subnet and certificate arn. And apply these yaml on jumpbox.

Kubectl apply -f ingres.yaml -n <namespace>

And check the load balancer in your aws account.if it is created or not.

Create

RDS Setup:

Now you have to create rds for our custom applications.please find below steps to create rds.

- 1.Sign in to the AWS Management Console and open the Amazon RDS.
2. Create RDS Subnet group.

The screenshot shows the AWS RDS dashboard. On the left, a sidebar menu includes 'Dashboard', 'Databases', 'Query Editor', 'Performance insights', 'Snapshots', 'Exports in Amazon S3', 'Automated backups', 'Reserved instances', 'Proxies', 'Subnet groups' (which is highlighted with a yellow box), 'Parameter groups', 'Option groups', 'Custom engine versions', and 'Events'. The main content area has a heading 'Resources' with a 'Refresh' button. It lists various RDS resources: DB Instances (5/20), DB Clusters (0/40), Reserved instances (0/20), Snapshots (48), and DB Cluster (0/100). To the right, there's a 'Recommended for you' section with links to 'Amazon RDS Backup and Restore using AWS Backup', 'Time-Series Tables in PostgreSQL', and 'Build RDS Operational Tasks'. At the bottom, there's a search bar and a toolbar with various icons.

The screenshot shows the 'Subnet groups' page under the RDS service. The sidebar is identical to the previous dashboard screenshot. The main area shows a table titled 'Subnet groups (4)'. The table has columns for 'Name', 'Description', 'Status', and 'VPC'. At the top right of the table, there is a red box highlighting the 'Create DB subnet group' button. The table also includes a search bar and navigation controls.

Amazon RDS

RDS > Subnet groups > Create DB subnet group

Create DB subnet group

To create a new subnet group, give it a name and a description, and choose an existing VPC. You will then be able to add subnets related to that VPC.

Subnet group details

Name
You won't be able to modify the name after your subnet group has been created.
 Must contain from 1 to 255 characters. Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.

Description

VPC
Choose a VPC identifier that corresponds to the subnets you want to use for your DB subnet group. You won't be able to choose a different VPC identifier after your subnet group has been created.

Add subnet and Availability Zones and choose private subnets of EKS cluster based VPC.

Amazon RDS

Dashboard
Databases
Query Editor
Performance insights
Snapshots
Exports in Amazon S3
Automated backups
Reserved instances
Proxies

Subnet groups
Parameter groups
Option groups
Custom engine versions

Events

Add subnets

Availability Zones
Choose the Availability Zones that include the subnets you want to add.
 ap-south-1a
 ap-south-1b
 ap-south-1c

Select subnets

Subnets selected (0)

Availability zone	Subnet ID	CIDR block
No subnets added to this group		

Create

Amazon RDS

Dashboard
Databases
Query Editor
Performance insights
Snapshots
Exports in Amazon S3
Automated backups
Reserved instances
Proxies

Subnet groups
Parameter groups
Option groups
Custom engine versions

Events

Subnets
Choose the subnets that you want to add. The list includes the subnets in the selected Availability Zones.

Subnets selected (3)

Availability zone	Subnet ID	CIDR block
ap-south-1c	subnet-f980ec82	172.31.16.0/20
ap-south-1b	subnet-d9fca695	172.31.0.0/20
ap-south-1a	subnet-56a5513d	172.31.32.0/20

Create

3. Create a Parameter group for PostgreSQL database to customize your DB parameters according to your requirements.

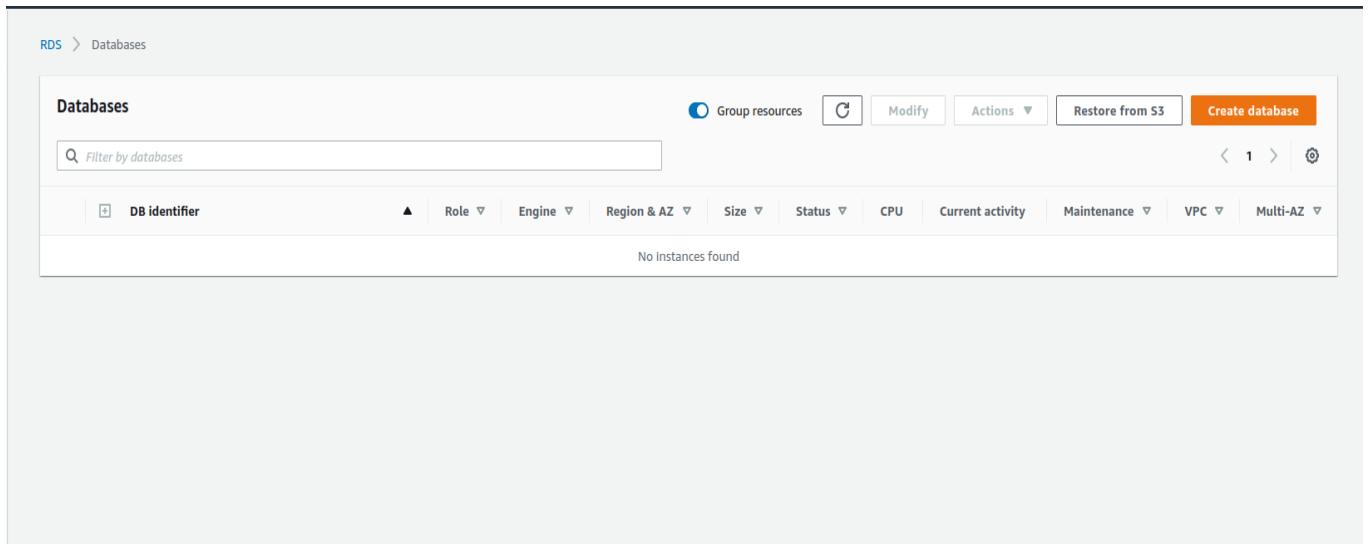
Select the Postgres as a DB Engine and version with 12.

The screenshot shows the 'Create parameter group' dialog in the Amazon RDS console. On the left, a sidebar menu lists various RDS services: Dashboard, Databases, Query Editor, Performance insights, Snapshots, Exports in Amazon S3, Automated backups, Reserved instances, Proxies, Subnet groups, **Parameter groups** (which is selected), Option groups, Custom engine versions, and Events. The main 'Create parameter group' dialog has a 'Parameter group details' section with a note: 'To create a parameter group, choose a parameter group family, then name and describe your parameter group'. Below this is a 'Parameter group family' dropdown containing a list of DB families: aurora-mysql5.7, oracle-se2-cdb-19, oracle-se2-cdb-21, aurora5.6, postgres10, postgres11, **postgres12** (highlighted in yellow), postgres13, postgres14, sqlserver-ee-12.0, sqlserver-ee-13.0, and sqlserver-ee-14.0. At the bottom right of the dialog are 'Cancel' and 'Create' buttons. The status bar at the bottom of the page includes 'Feedback', 'Looking for language selection? Find it in the new Unified Settings', '© 2023, Amazon Web Services India Private Limited or its affiliates.', 'Privacy', 'Terms', and 'Cookie preferences'.

This screenshot shows the 'Create parameter group' dialog with the following filled fields: 'Parameter group family' set to 'postgres12', 'Type' set to 'DB Parameter Group', 'Group name' set to 'dmi-demo-custom-params', and 'Description' set to 'dmi-demo-custom-params'. The rest of the interface is identical to the previous screenshot, including the sidebar menu and the bottom status bar.

Once you select the all option click on Create for parameter group.

- 4.In the upper-right corner of the Amazon RDS console, choose the AWS Region in which you want to create the DB instance.
- 5.Now click on the create option on the right side .



6. .And select these options like in the image below.

RDS > Create database

Create database

Choose a database creation method Info

Standard create
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy create
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Engine options

Engine type Info

Amazon Aurora 

MySQL 

MariaDB 

PostgreSQL 

Oracle 

Microsoft SQL Server 

Engine Version
PostgreSQL 12.11-R1 ▾

7. .And select templates option according to your need.

Templates

Choose a sample template to meet your use case.

Production
Use defaults for high availability and fast, consistent performance.

Dev/Test
This instance is intended for development use outside of a production environment.

Free tier
Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS.
[Info](#)

8.select multi A-Z instance.

Availability and durability

Deployment options [Info](#)
The deployment options below are limited to those supported by the engine you selected above.

- Multi-AZ DB Cluster - new**
Creates a DB cluster with a primary DB instance and two readable standby DB instances, with each DB instance in a different Availability Zone (AZ). Provides high availability, data redundancy and increases capacity to serve read workloads.
- Multi-AZ DB Instance**
Creates a primary DB instance and a standby DB instance in a different AZ. Provides high availability and data redundancy, but the standby DB instance doesn't support connections for read workloads.
- Single DB instance**
Creates a single DB instance with no standby DB instances.

9.Now you have to select a master username or password for login.

Settings

DB instance identifier [Info](#)
Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

database-1

The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

▼ Credentials Settings

Master username [Info](#)
Type a login ID for the master user of your DB instance.

postgres

1 to 16 alphanumeric characters. First character must be a letter.

Manage master credentials in AWS Secrets Manager
Manage master user credentials in Secrets Manager. RDS can generate a password for you and manage it throughout its lifecycle.

Auto generate a password
Amazon RDS can generate a password for you, or you can specify your own password.

Master password [Info](#)

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote), "(double quote) and @ (at sign).

Confirm master password [Info](#)

10.select instance configuration and storage according to your need.

Instance configuration

The DB instance configuration options below are limited to those supported by the engine that you selected above.

DB Instance class [Info](#)

Standard classes (includes m classes)
 Memory optimized classes (includes r and x classes)
 Burstable classes (includes t classes)

db.m6g.large

2 vCPUs 8 GiB RAM Network: 4,750 Mbps

Include previous generation classes

Storage

Storage type [Info](#)

Provisioned IOPS SSD (io1)
Flexibility in provisioning I/O

Allocated storage
400 GIB
The minimum value is 100 GiB and the maximum value is 65,536 GiB

Provisioned IOPS [Info](#)
3000 IOPS
The minimum value is 1,000 IOPS and the maximum value is 80,000 IOPS. The storage size ratio must be between 0.1 and 1000. The storage size ratio is the ratio between allocated storage and the Provisioned IOPS.

ⓘ Your actual IOPS might vary from the amount that you provisioned based on your database workload and instance type. [Learn more](#)

Storage autoscaling [Info](#)
Provides dynamic scaling support for your database's storage based on your application's needs.

Enable storage autoscaling
Enabling this feature will allow the storage to increase after the specified threshold is exceeded.

Maximum storage threshold [Info](#)
Charges will apply when your database autoscales to the specified threshold
1000 GIB
The minimum value is 440 GiB and the maximum value is 65,536 GiB

11. In the next connectivity section you have to select your vpc which is your eks vpc and subnets. Please no in public access.

The screenshot shows the 'Connectivity' section of the AWS RDS configuration interface. It includes fields for Compute resource, Network type, VPC selection, DB Subnet group, and Public access.

Compute resource
Choose whether to set up a connection to a compute resource for this database. Setting up a connection will automatically change connectivity settings so that the compute resource can connect to this database.

Don't connect to an EC2 compute resource
Don't set up a connection to a compute resource for this database. You can manually set up a connection to a compute resource later.

Connect to an EC2 compute resource
Set up a connection to an EC2 compute resource for this database.

Network type [Info](#)
To use dual-stack mode, make sure that you associate an IPv6 CIDR block with a subnet in the VPC you specify.

IPv4
Your resources can communicate only over the IPv4 addressing protocol.

Dual-stack mode
Your resources can communicate over IPv4, IPv6, or both.

Virtual private cloud (VPC) [Info](#)
Choose the VPC. The VPC defines the virtual networking environment for this DB instance.

Default VPC (vpc-0b239071c215d0f4c) ▾
Only VPCs with a corresponding DB subnet group are listed.

DB Subnet group [Info](#)
Choose the DB subnet group. The DB subnet group defines which subnets and IP ranges the DB instance can use in the VPC that you selected.

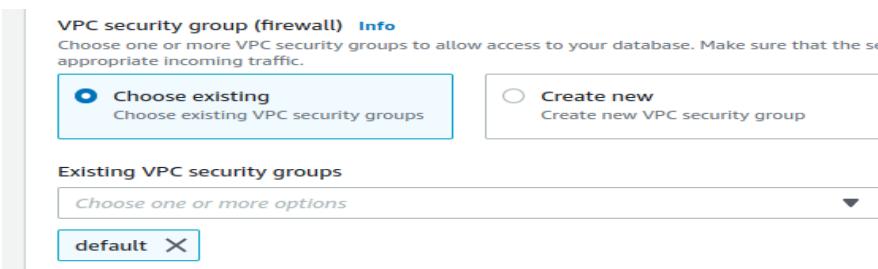
default ▾

Public access [Info](#)

Yes
RDS assigns a public IP address to the database. Amazon EC2 instances and other resources outside of the VPC can connect to your database. Resources inside the VPC can also connect to the database. Choose one or more VPC security groups that specify which resources can connect to the database.

No
RDS doesn't assign a public IP address to the database. Only Amazon EC2 instances and other resources inside the VPC can

12. Create a security group for your RDS.



13. Now click on the create RDS option.