

import pulumi

import pulumi\_aws as aws

import os

# Create a VPC

vpc = aws.ec2.Vpc(

'nodejs-db-vpc',

cidr\_block='10.0.0.0/16',

enable\_dns\_support=True,

enable\_dns\_hostnames=True,

tags={'Name': 'nodejs-db-vpc'}

)

# Create public and private subnets

public\_subnet = aws.ec2.Subnet(

'nodejs-public-subnet',

vpc\_id=vpc.id,

cidr\_block='10.0.1.0/24',

map\_public\_ip\_on\_launch=True,

availability\_zone='ap-southeast-1a', # Update with your desired AZ

tags={'Name': 'nodejs-public-subnet'}

)

private\_subnet = aws.ec2.Subnet(

'db-private-subnet',

vpc\_id=vpc.id,

cidr\_block='10.0.2.0/24',

map\_public\_ip\_on\_launch=False,

availability\_zone='ap-southeast-1a', # Update with your desired AZ

tags={'Name': 'db-private-subnet'}

)

# Create an Internet Gateway

internet\_gateway = aws.ec2.InternetGateway(

'nodejs-db-internet-gateway',

vpc\_id=vpc.id,

tags={'Name': 'nodejs-db-internet-gateway'}

)

# Create NAT Gateway for private subnet

elastic\_ip = aws.ec2.Eip('nat-eip')

nat\_gateway = aws.ec2.NatGateway(

'nat-gateway',

allocation\_id=elastic\_ip.id,

subnet\_id=public\_subnet.id,

tags={'Name': 'nodejs-db-nat-gateway'}

)

# Create public Route Table

public\_route\_table = aws.ec2.RouteTable(

'public-route-table',

vpc\_id=vpc.id,

routes=[

aws.ec2.RouteTableRouteArgs(

cidr\_block='0.0.0.0/0',

gateway\_id=internet\_gateway.id,

)

],

tags={'Name': 'nodejs-public-route-table'}

)

# Create private Route Table

private\_route\_table = aws.ec2.RouteTable(

'private-route-table',

vpc\_id=vpc.id,

routes=[

aws.ec2.RouteTableRouteArgs(

cidr\_block='0.0.0.0/0',

nat\_gateway\_id=nat\_gateway.id,

)

],

tags={'Name': 'db-private-route-table'}

)

# Associate route tables with subnets

public\_route\_table\_association = aws.ec2.RouteTableAssociation(

'public-route-table-association',

subnet\_id=public\_subnet.id,

route\_table\_id=public\_route\_table.id

)

private\_route\_table\_association = aws.ec2.RouteTableAssociation(

'private-route-table-association',

subnet\_id=private\_subnet.id,

route\_table\_id=private\_route\_table.id

)

# Create security group for Node.js application

nodejs\_security\_group = aws.ec2.SecurityGroup(

'nodejs-security-group',

vpc\_id=vpc.id,

description="Security group for Node.js application",

ingress=[

# SSH access

aws.ec2.SecurityGroupIngressArgs(

protocol='tcp',

from\_port=22,

to\_port=22,

cidr\_blocks=['0.0.0.0/0'], # Consider restricting to your IP

),

# Node.js application port

aws.ec2.SecurityGroupIngressArgs(

protocol='tcp',

from\_port=3000,

to\_port=3000,

cidr\_blocks=['0.0.0.0/0'],

),

],

egress=[

aws.ec2.SecurityGroupEgressArgs(

protocol='-1',

from\_port=0,

to\_port=0,

cidr\_blocks=['0.0.0.0/0'],

)

],

tags={'Name': 'nodejs-security-group'}

)

# Create security group for MySQL database

db\_security\_group = aws.ec2.SecurityGroup(

'db-security-group',

vpc\_id=vpc.id,

description="Security group for MySQL database",

ingress=[

# SSH access from Node.js subnet

aws.ec2.SecurityGroupIngressArgs(

protocol='tcp',

from\_port=22,

to\_port=22,

cidr\_blocks=[public\_subnet.cidr\_block],

),

# MySQL access from Node.js subnet

aws.ec2.SecurityGroupIngressArgs(

protocol='tcp',

from\_port=3306,

to\_port=3306,

cidr\_blocks=[public\_subnet.cidr\_block],

),

],

egress=[

aws.ec2.SecurityGroupEgressArgs(

protocol='-1',

from\_port=0,

to\_port=0,

cidr\_blocks=['0.0.0.0/0'],

)

],

tags={'Name': 'db-security-group'}

)

# Create EC2 Instance for DB

db = aws.ec2.Instance(

'db-server',

instance\_type='t2.micro',

ami='ami-01811d4912b4ccb26', # Update with correct Ubuntu AMI ID

subnet\_id=private\_subnet.id,

key\_name="db-cluster",

vpc\_security\_group\_ids=[db\_security\_group.id],

tags={'Name': 'db-server'}

)

# Update the path of the script to the path of the script in the root directory

with open('/root/code/script/check-mysql.sh', 'r') as file:

print("Reading From script...\n")

mysql\_check\_script = file.read()

def generate\_nodejs\_user\_data(db\_private\_ip):

return f'''#!/bin/bash

exec > >(tee /var/log/user-data.log) 2>&1

# Update system and install dependencies

apt-get update

apt-get upgrade -y

apt-get install -y netcat-openbsd

# Install Node.js

curl -fsSL https://deb.nodesource.com/setup\_18.x | bash -

apt-get install -y nodejs

# Create script directory

mkdir -p /usr/local/bin

# Set environment variable for DB IP

echo "DB\_PRIVATE\_IP={db\_private\_ip}" >> /etc/environment

# Create MySQL check script

cat > /usr/local/bin/check-mysql.sh << 'EOL'

{mysql\_check\_script}

EOL

chmod +x /usr/local/bin/check-mysql.sh

'''

# Update your Pulumi EC2 instance configurations

nodejs = aws.ec2.Instance(

'nodejs-server',

instance\_type='t2.micro',

ami='ami-01811d4912b4ccb26', # Update with correct Ubuntu AMI ID

subnet\_id=public\_subnet.id,

key\_name="db-cluster",

vpc\_security\_group\_ids=[nodejs\_security\_group.id],

associate\_public\_ip\_address=True,

user\_data=pulumi.Output.all(db.private\_ip).apply(

lambda args: generate\_nodejs\_user\_data(args[0])

),

user\_data\_replace\_on\_change=True,

tags={'Name': 'nodejs-server'}

)

# Export Public and Private IPs

pulumi.export('nodejs\_public\_ip', nodejs.public\_ip)

pulumi.export('nodejs\_private\_ip', nodejs.private\_ip)

pulumi.export('db\_private\_ip', db.private\_ip)

# Export the VPC ID and Subnet IDs for reference

pulumi.export('vpc\_id', vpc.id)

pulumi.export('public\_subnet\_id', public\_subnet.id)

pulumi.export('private\_subnet\_id', private\_subnet.id)

# Create config file

def create\_config\_file(all\_ips):

config\_content = f"""Host nodejs-server

HostName {all\_ips[0]}

User ubuntu

IdentityFile ~/.ssh/db-cluster.id\_rsa

Host db-server

ProxyJump nodejs-server

HostName {all\_ips[1]}

User ubuntu

IdentityFile ~/.ssh/db-cluster.id\_rsa

"""

config\_path = os.path.expanduser("~/.ssh/config")

with open(config\_path, "w") as config\_file:

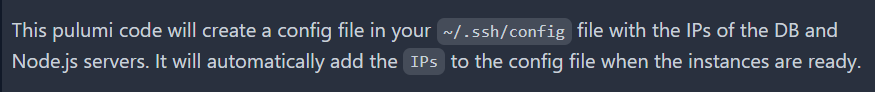
config\_file.write(config\_content)

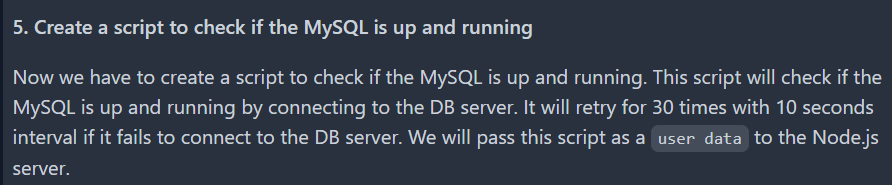
# Collect the IPs for all nodes

all\_ips = [nodejs.public\_ip, db.private\_ip]

# Create the config file with the IPs once the instances are ready

pulumi.Output.all(\*all\_ips).apply(create\_config\_file)





#!/bin/bash

DB\_HOST="$DB\_PRIVATE\_IP"

DB\_PORT=3306

MAX\_RETRIES=30

RETRY\_INTERVAL=10

check\_mysql() {

nc -z "$DB\_HOST" "$DB\_PORT"

return $?

}

retry\_count=0

while [ $retry\_count -lt $MAX\_RETRIES ]; do

if check\_mysql; then

echo "Successfully connected to MySQL at $DB\_HOST:$DB\_PORT"

exit 0

fi

echo "Attempt $((retry\_count + 1))/$MAX\_RETRIES: Cannot connect to MySQL at $DB\_HOST:$DB\_PORT. Retrying in $RETRY\_INTERVAL seconds..."

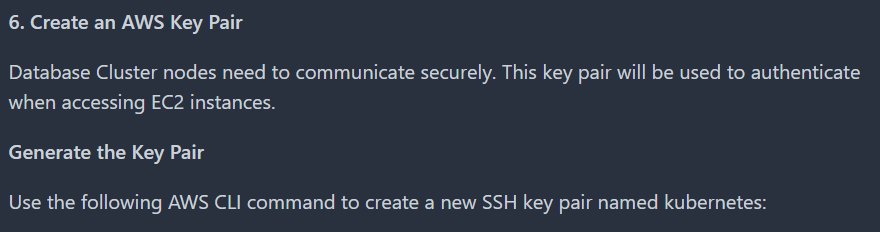
sleep $RETRY\_INTERVAL

retry\_count=$((retry\_count + 1))

done

echo "Failed to connect to MySQL after $MAX\_RETRIES attempts"

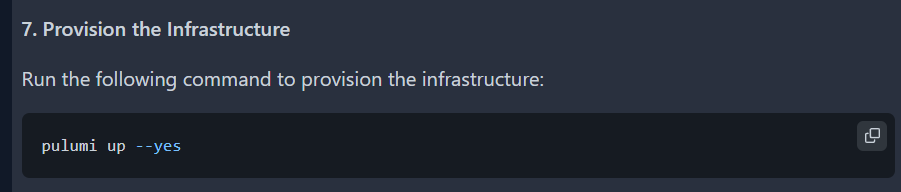
exit 1

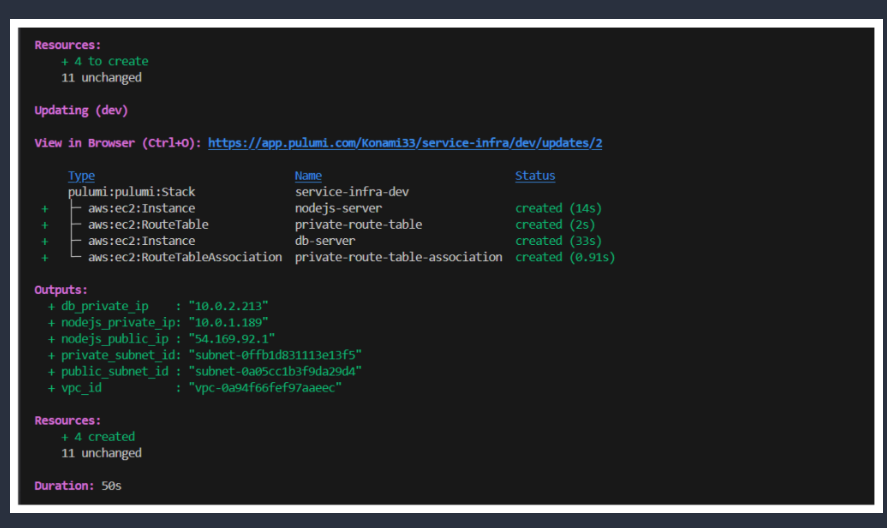


cd ~/.ssh/

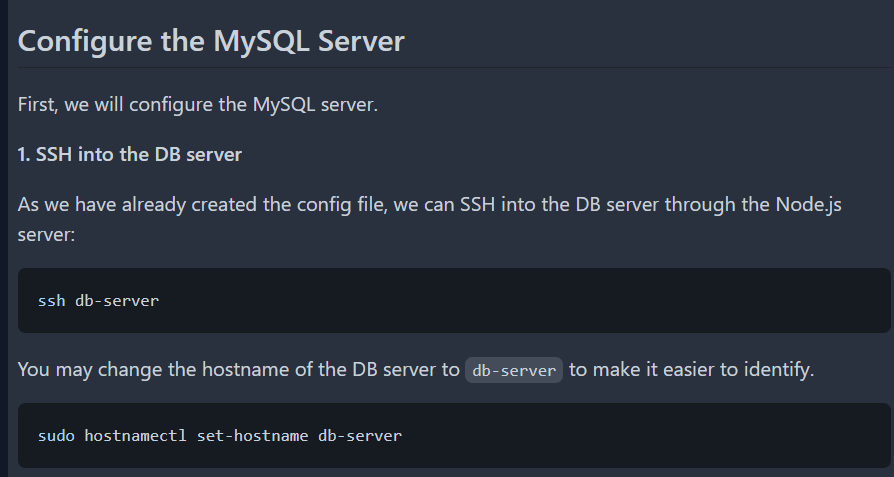
aws ec2 create-key-pair --key-name db-cluster --output text --query 'KeyMaterial' > db-cluster.id\_rsa

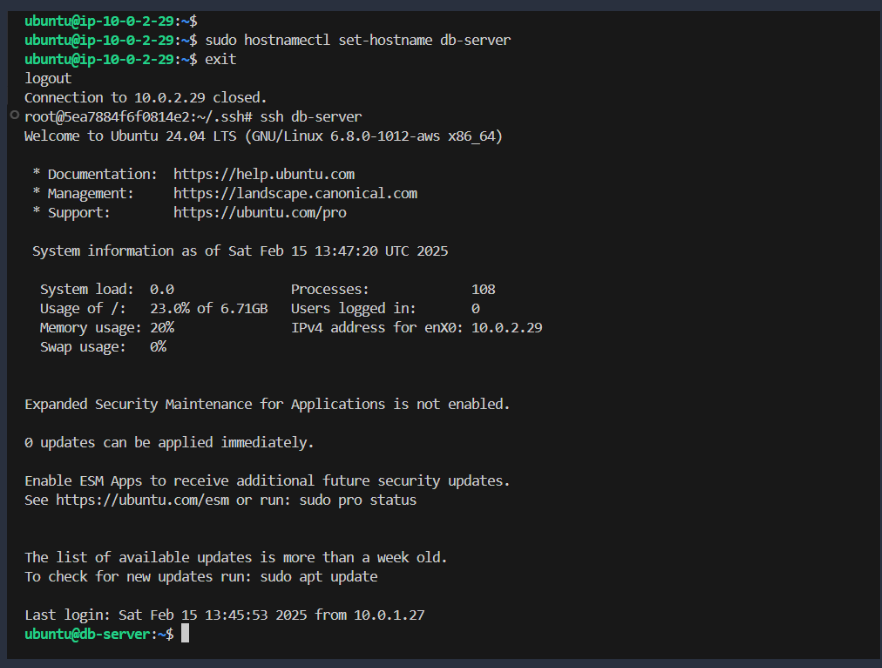
chmod 400 db-cluster.id\_rsa

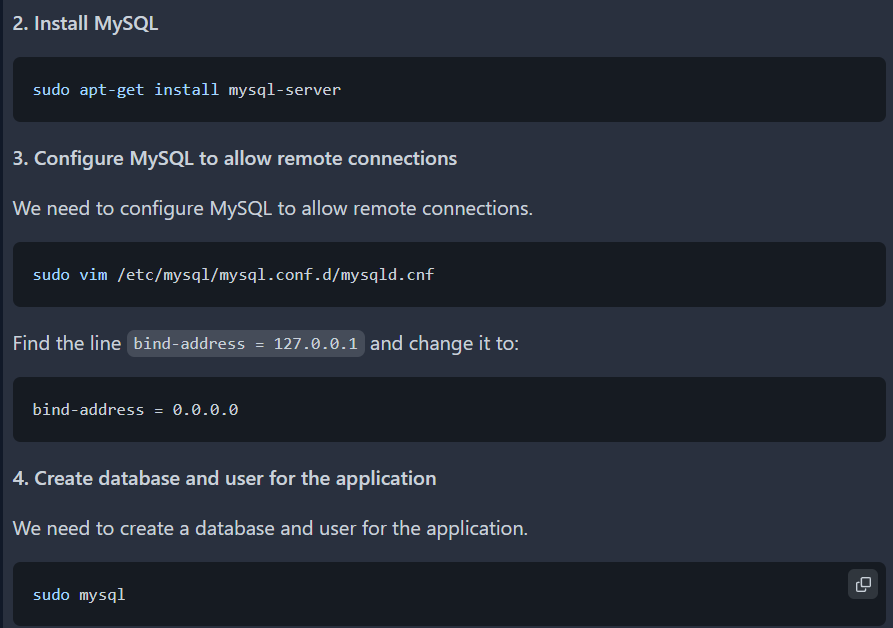


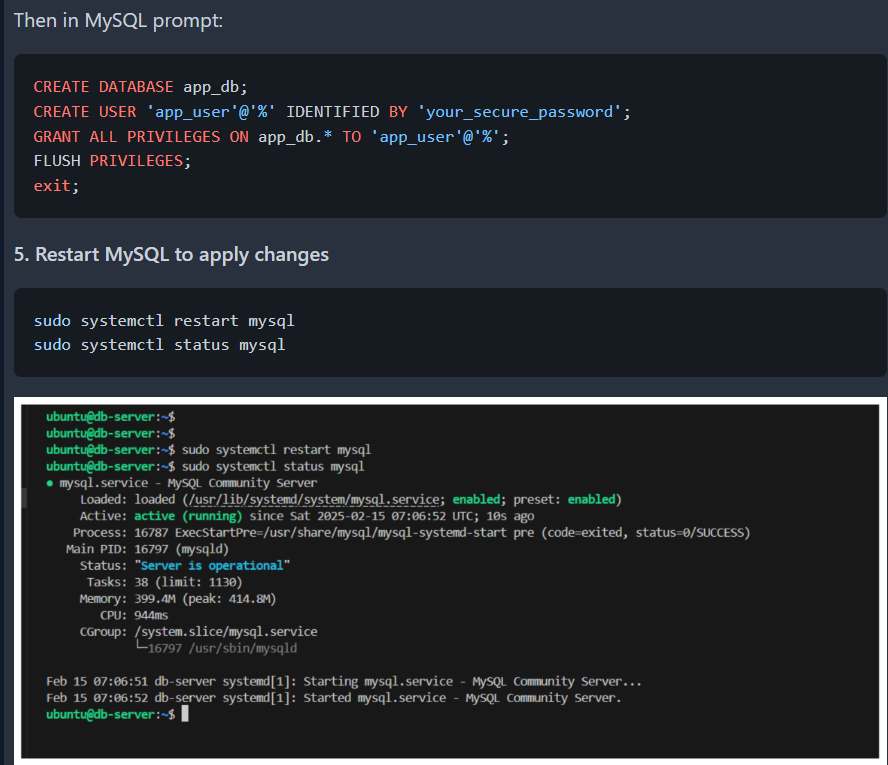












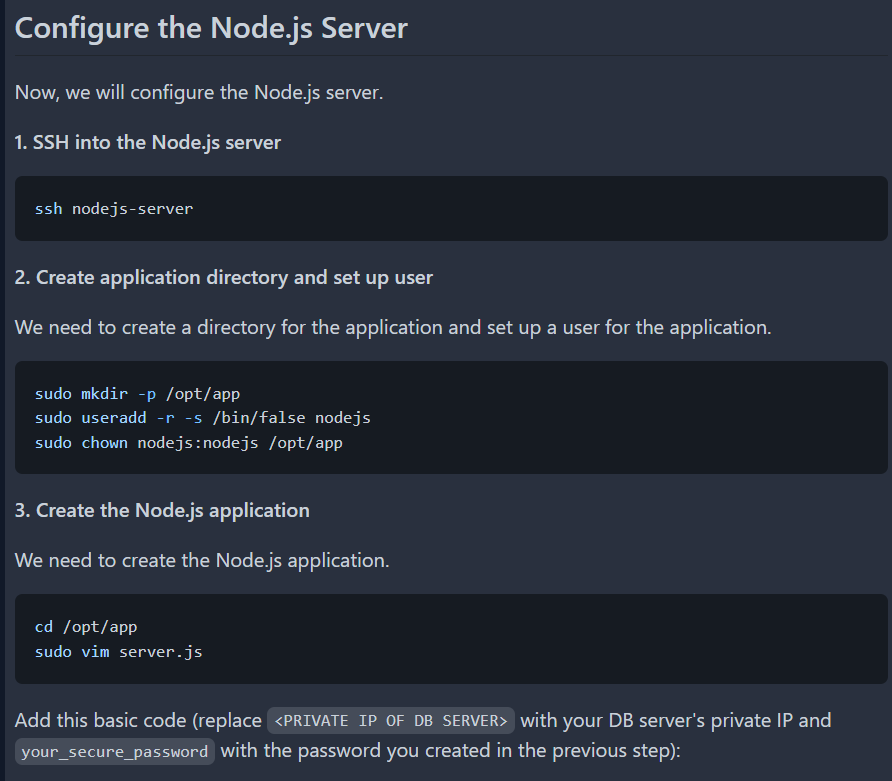
CREATE DATABASE app\_db;

CREATE USER 'app\_user'@'%' IDENTIFIED BY 'your\_secure\_password';

GRANT ALL PRIVILEGES ON app\_db.\* TO 'app\_user'@'%';

FLUSH PRIVILEGES;

exit;



const express = require('express');

const mysql = require('mysql2');

const app = express();

const pool = mysql.createPool({

host: '<PRIVATE IP OF DB SERVER>', // Replace with your DB private IP

user: 'app\_user',

password: 'your\_secure\_password',

database: 'app\_db',

waitForConnections: true,

connectionLimit: 10

});

app.get('/', (req, res) => {

pool.query('SELECT 1', (err, results) => {

if (err) {

res.status(500).send('Database connection failed');

return;

}

res.send('Application is running!');

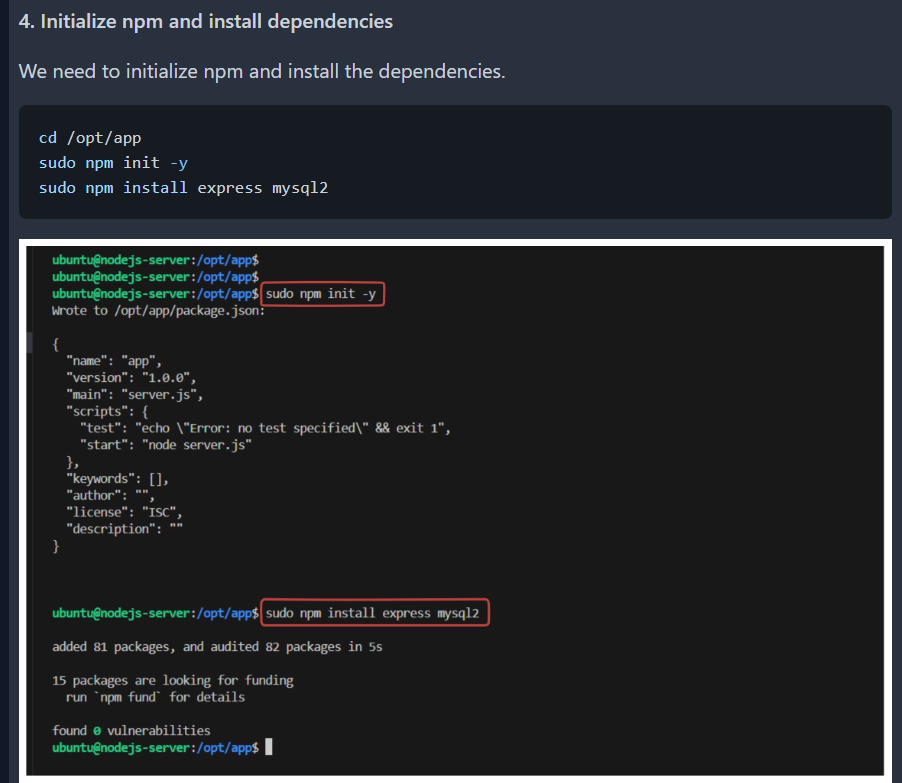
});

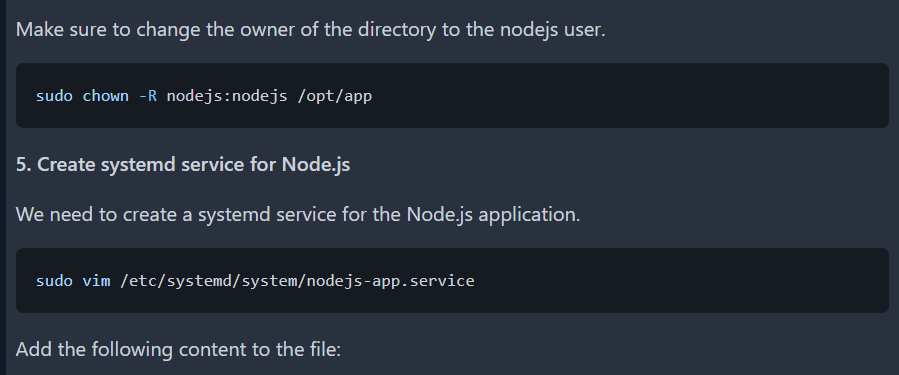
});

app.listen(3000, () => {

console.log('Server running on port 3000');

});





[Unit]

Description=Node.js Application

After=mysql-check.service

Requires=mysql-check.service

[Service]

Type=simple

User=nodejs

WorkingDirectory=/opt/app

ExecStart=/usr/bin/node server.js

Restart=always

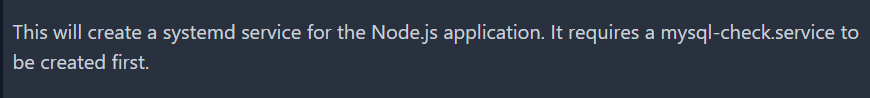
RestartSec=10

StandardOutput=journal

StandardError=journal

[Install]

WantedBy=multi-user.target





[Unit]

Description=MySQL Availability Check

After=network.target

[Service]

Type=oneshot

EnvironmentFile=/etc/environment

ExecStart=/usr/local/bin/check-mysql.sh

RemainAfterExit=yes

[Install]

WantedBy=multi-user.target

