NextCloud Project on AWS

Level: Basic

Phase 1: Setting Up the Basic AWS Infrastructure

Create an AWS Account:

If you don't already have one, go to the AWS website (https://aws.amazon.com/) and sign up for an account. You'll need to provide an email address, phone number, and credit card information. AWS offers a free tier that you can leverage for this basic setup, but be mindful of the limitations.

Explanation: This is the fundamental first step to access AWS services. The free tier allows you to experiment with many services within certain usage limits for the first 12 months.

Access the AWS Management Console:

Once your account is set up, log in to the AWS Management Console (https://console.aws.amazon.com/). This is a web-based interface for managing your AWS resources.

Explanation: The AWS Management Console provides a graphical interface to interact with all the AWS services. You'll use this to create and manage your virtual machine.

Launch an EC2 Instance (Virtual Machine):

Navigate to EC2: In the AWS Management Console, find the "EC2" service (you can search for it in the search bar). EC2 stands for Elastic Compute Cloud and is AWS's virtual machine service.

Launch Instance: Click on "Instances" in the left-hand menu and then click the "Launch instances" button.

Choose an Amazon Machine Image (AMI): An AMI is a template that contains the operating system, application server, and applications required to launch your instance. For a basic Nextcloud setup, we'll choose a standard Linux distribution:

Search for and select "Ubuntu Server". Choose a recent, stable LTS (Long Term Support) version (e.g., Ubuntu Server 22.04 LTS). These are well-documented and have a large community for support.

Explanation: AMIs streamline the process of creating a VM by providing pre-configured operating systems. Ubuntu Server is a popular choice for server applications due to its stability and ease of use.

Choose an Instance Type: This determines the hardware configuration of your VM (CPU, memory, storage, network performance). For a basic setup, a t2.micro instance is often sufficient and falls within the free tier eligibility (check the free tier documentation for the latest details).

Explanation: Instance types are categorized based on their intended use. t2.micro is a low-cost option suitable for development and testing or low-traffic applications.

Configure Instance Details (You can usually leave these as default for a basic setup):

Network: You'll likely be launching into your default VPC (Virtual Private Cloud). A VPC is a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network that you define.

Subnet: Choose a public subnet within your VPC. A public subnet has a route to the internet.

Auto-assign Public IP: Ensure this is enabled so your instance gets a public IP address that you can use to connect to it.

Explanation: VPC provides network isolation. Subnets are subdivisions within your VPC. A public IP address is necessary to access your VM from the internet.

Add Storage: The default storage (root volume) is usually sufficient for a basic Nextcloud installation. You can increase the size if you anticipate storing a lot of data directly on the instance (though we might explore external storage later).

Explanation: This is the disk space allocated to your VM's operating system and applications.

Configure Security Group: A security group acts as a virtual firewall for your instance, controlling inbound and outbound traffic.

Click "Create a new security group".

Security Group Name: Give it a descriptive name (e.g., nextcloud-sg).

Description: Add a brief description (e.g., "Security group for Nextcloud instance").

Add Rules:

SSH: Add a rule with Type "SSH", Protocol "TCP", Port Range "22", and Source "My IP" (for security, it's best to restrict SSH access to your current IP address) or "Anywhere" (if you have a dynamic IP or are unsure, but this is less secure). SSH is used to connect to your instance via a terminal.

HTTP: Add a rule with Type "HTTP", Protocol "TCP", Port Range "80", and Source "Anywhere". This allows web traffic to reach your Nextcloud installation.

HTTPS: Add a rule with Type "HTTPS", Protocol "TCP", Port Range "443", and Source "Anywhere". This will be needed later for secure access to Nextcloud.

Explanation: Security groups are essential for controlling network access to your instance. Only traffic matching the defined rules will be allowed.

Review and Launch: Review your instance configuration and click "Launch".

Select an Existing Key Pair or Create a New Key Pair: A key pair consists of a public key (stored by AWS) and a private key (that you download and keep secure). This is used for securely connecting to your instance via SSH.

If you don't have one, select "Create a new key pair", give it a name (e.g., nextcloud-key), download the .pem file, and store it in a secure location. You will need this file to connect to your instance.

Important: If you lose your private key, you won't be able to SSH into your instance (unless you follow a complex recovery process).

Click "Launch Instances".

Phase 2: Installing Nextcloud on the EC2 Instance

Connect to Your EC2 Instance via SSH:

Open a Terminal (Linux or macOS) or use PuTTY (Windows).

Navigate to the directory where you saved your private key (.pem file).

Use the chmod command to ensure your private key file is not publicly accessible (Linux/macOS):

Bash

chmod 400 your-private-key.pem

Replace your-private-key.pem with the actual name of your file.

Connect using the ssh command:

Bash

ssh -i "your-private-key.pem" ubuntu@your-instance-public-ip

Replace your-private-key.pem with the path to your key file and your-instance-public-ip with the public IP address of your EC2 instance (you can find this in the EC2 Management Console).

Explanation: SSH is a secure protocol used to establish a command-line connection to your remote server. The -i flag specifies the private key to use for authentication, ubuntu is the default username for Ubuntu AMIs, and @your-instance-public-ip specifies the server's address.

Update the System Packages:

Once connected to your instance, it's good practice to update the package lists and upgrade any outdated packages:

Bash

sudo apt update

sudo apt upgrade -y

Explanation: apt update refreshes the list of available packages from the repositories. apt upgrade -y downloads and installs the latest versions of all installed packages, and the -y flag automatically confirms the installation.

Install the Necessary Software:

Nextcloud requires a web server (Apache or Nginx), PHP, and a database (MySQL/MariaDB or PostgreSQL). Let's install Apache, PHP, and MariaDB, which is a common and straightforward setup:

Bash

sudo apt install apache2 mariadb-server libapache2-mod-php php-cli php-fpm php-json php-mysql php-curl php-mbstring php-xml php-zip php-gd php-imagick

Explanation: This command uses apt to install the Apache web server (apache2), the MariaDB database server (mariadb-server), the PHP command-line interpreter (php-cli), PHP-FPM (FastCGI Process Manager for better performance with Apache), and various essential PHP modules that Nextcloud relies on.

Download and Install Nextcloud:

Navigate to the Apache web server's document root directory:

Bash

cd /var/www/html/

Download the latest stable version of Nextcloud. Go to the official Nextcloud download page (https://nextcloud.com/install/#install-script) and find the link for the "Web installer". We'll use wget to download it:

Bash

sudo apt install wget

sudo wget https://download.nextcloud.com/server/releases/latest.tar.bz2

(Note: Replace latest.tar.bz2 with the actual filename if the link points to a specific version.)

Extract the Nextcloud files:

Bash

sudo tar -xjvf latest.tar.bz2

This will create a directory named nextcloud in /var/www/html/.

Set the correct ownership and permissions for the Nextcloud directory:

Bash

sudo chown -R www-data:www-data /var/www/html/nextcloud/

sudo chmod -R 755 /var/www/html/nextcloud/

Explanation: chown changes the owner and group of the files to www-data, which is the user that Apache runs under. chmod changes the file permissions to allow Apache to read and write necessary files.

Configure Apache for Nextcloud:

Enable the necessary Apache modules:

Bash

sudo a2enmod rewrite

sudo a2enmod headers

sudo a2enmod env

sudo a2enmod dir

sudo a2enmod mime

sudo systemctl restart apache2

Explanation: These Apache modules provide important functionalities that Nextcloud relies on, such as URL rewriting, setting HTTP headers, and handling different file types.

(Optional but Recommended) Create an Apache virtual host configuration file for Nextcloud. This helps in managing your web server configuration if you host multiple websites.

Bash

sudo nano /etc/apache2/sites-available/nextcloud.conf

Add the following content (replace your_instance_public_ip with the actual IP or a domain name if you have one):

Apache

<VirtualHost *:80>

ServerAdmin webmaster@localhost

DocumentRoot /var/www/html/nextcloud/

ServerName your_instance_public_ip

<Directory /var/www/html/nextcloud/> Options +FollowSymlinks AllowOverride All Require all granted </Directory>

ErrorLog ${APACHE_LOG_DIR}/error.log

CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>

```

Save and close the file (Ctrl+X, then Y, then Enter in `nano`).

Enable the new virtual host and disable the default one:

```bash

sudo a2ensite nextcloud.conf

sudo a2dissite 000-default.conf

sudo systemctl restart apache2

```

 * **Explanation:** Virtual hosts allow you to configure different websites or applications on the same server. This configuration tells Apache where to find the Nextcloud files and how to handle requests for your instance's IP address.

Configure the Database (MariaDB):

Secure your MariaDB installation:

Bash

sudo mysql_secure_installation

Follow the prompts to set a root password, remove anonymous users, disallow remote root login, and remove the test database.

Log in to the MariaDB shell:

Bash

sudo mysql -u root -p

Enter the root password you set in the previous step.

Create a database and user for Nextcloud:

SQL

CREATE DATABASE nextcloud;

CREATE USER 'nextclouduser'@'localhost' IDENTIFIED BY 'your_strong_password';

GRANT ALL PRIVILEGES ON nextcloud.* TO 'nextclouduser'@'localhost' IDENTIFIED BY 'your_strong_password';

FLUSH PRIVILEGES;

EXIT;

Replace 'your_strong_password' with a strong, unique password.

Explanation: These commands create a dedicated database for Nextcloud and a user with the necessary permissions to access and manage it.

Phase 3: Finalizing the Nextcloud Setup via the Web Interface

Access the Nextcloud Web Installer:

Open your web browser and navigate to the public IP address of your EC2 instance (e.g., http://your-instance-public-ip). If you configured a virtual host with a domain name, use that instead.
```

You should see the Nextcloud setup wizard.

Complete the Setup:

Create an Admin Account: Enter your desired username and password for the Nextcloud administrator.

Database Configuration:

Select "MySQL/MariaDB".

Database user: nextclouduser

Database password: The strong password you set for the nextclouduser in MariaDB.

Database name: nextcloud

Database host: localhost

Click "Install".

Log in to Your New Nextcloud Instance:

Once the installation is complete, you will be redirected to the login page. Use the admin credentials you created to log in.

Congratulations! You now have a basic Nextcloud instance running on AWS. You can start uploading files and exploring its features.

Next Steps (Basic Level Enhancements):

Secure with HTTPS: Obtain an SSL/TLS certificate (e.g., using Let's Encrypt) and configure Apache to use HTTPS for secure access.

Basic Backups: Explore simple backup strategies for your instance and Nextcloud data.

User Management: Learn how to create and manage users and groups within Nextcloud.

This detailed walkthrough should provide a solid foundation for your Nextcloud project on AWS at the basic level. As you become more comfortable, you can explore the intermediate and expert-level concepts we discussed earlier to further enhance your setup. Remember to consult the official Nextcloud and AWS documentation for more in-depth information on each step.

Phase 4: Transitioning to Managed Services (Intermediate Level)

Our primary goals for this level are to:

- **Move the database to Amazon RDS (Relational Database Service).** This offloads database management tasks.
- **Consider using Amazon S3 for scalable storage.** This provides better durability and scalability for your files.

Part 1: Launching and Configuring an Amazon RDS Instance

1. **Navigate to RDS in the AWS Management Console:** Search for and select "RDS".
2. **Create a Database:** Click on "Databases" in the left-hand menu and then "Create database".
3. **Choose a Database Creation Method:** Select "Standard create".
4. **Choose a Database Engine:** Select **"MySQL"** or **"MariaDB"** (whichever you used in the basic setup for compatibility).
5. **Choose a Template:** For a non-production, intermediate setup, you can choose "Dev/Test".
6. **Configure Settings:**
    - **DB instance identifier:** Give your database instance a name (e.g., nextcloud-db).
    - **Master username:** Choose a username for the administrative user of your RDS instance (e.g., admin).
    - **Master password:** Set a strong password for the master user. **Remember this password!**
    - **Confirm password:** Re-enter the password.
7. **Instance Configuration:**
    - **DB instance class:** Choose an instance type suitable for your needs. db.t3.micro or db.t2.micro are often sufficient for intermediate testing and fall within potential free tier eligibility (check the latest free tier documentation).
    - **Storage:** Allocate an appropriate amount of storage (e.g., 20-50 GB) based on your anticipated database size. Ensure "Enable auto scaling" is checked for storage to automatically increase if needed.
    - **Availability & Durability:** For an intermediate setup, you can leave "Multi-AZ deployment" as "Create a standby instance" for increased availability in case of a failure in one Availability Zone.
8. **Connectivity:**
    - **Virtual private cloud (VPC):** Ensure it's in the same VPC as your EC2 instance.
    - **Subnet group:** You can use the default.
    - **Publicly accessible:** Choose **"No"**. It's generally more secure to keep your database private and only accessible from within your VPC. We'll configure security groups to allow access from your EC2 instance.
    - **VPC security groups:** Choose "Create new security group". Give it a name (e.g., nextcloud-db-sg). This security group will control access to your RDS instance.
9. **Database options:**
    - **Initial database name:** Enter nextcloud (the same name you used in the basic setup).
10. **Backup:** Configure your backup settings as desired.

11. **Monitoring:** Enable enhanced monitoring if you want more detailed performance metrics.
12. **Maintenance:** Configure maintenance windows as needed.
13. **Create database:** Click "Create database".

Part 2: Configuring the RDS Security Group

1. **Navigate to VPC Security Groups:** In the AWS Management Console, search for and select "VPC", then click on "Security Groups" in the left-hand menu.
2. **Find the Security Group for Your RDS Instance:** Look for the security group you created in the RDS setup (e.g., nextcloud-db-sg).
3. **Edit Inbound Rules:** Select the security group and click on the "Inbound rules" tab, then "Edit inbound rules".
4. **Add a Rule:**
    - **Type:** Select "MySQL/Aurora" (for MySQL) or "MariaDB" (for MariaDB).
    - **Protocol:** TCP
    - **Port Range:** 3306 (default port for MySQL/MariaDB)
    - **Source:** Select "Custom" and enter the **private IP address** of your EC2 instance. Alternatively, for more flexibility within your VPC, you can select the security group of your EC2 instance. To do this, start typing the name of your EC2 instance's security group and select it when it appears.
    - **Description:** Add a description (e.g., "Allow MySQL/MariaDB access from Nextcloud EC2 instance").
5. **Save Rules:** Click "Save rules".

Part 3: Connecting Your Nextcloud Instance to the RDS Database

1. **SSH into your EC2 instance:** Follow the same steps as in Phase 2, Step 1.
2. **Edit the Nextcloud Configuration File:** Open the Nextcloud configuration file:

Bash

sudo nano /var/www/html/nextcloud/config/config.php

3. **Modify the Database Settings:** Look for the 'dbhost', 'dbuser', and 'dbpassword' lines and update them with the information for your RDS instance:
    - 'dbhost' => 'your-rds-endpoint', Replace your-rds-endpoint with the **Endpoint** of your RDS instance. You can find this in the RDS Management Console by selecting your database and looking at the "Connectivity & security" tab. It will be in the format your-instance-name.xxxxxxxxxxxx.your-region.rds.amazonaws.com.

- 'dbuser' => 'admin', Replace admin with the master username you created for your RDS instance.
- 'dbpassword' => 'your-rds-master-password', Replace your-rds-master-password with the master password you set for your RDS instance.

4. **Save and Close the File:** (Ctrl+X, then Y, then Enter in nano).
5. **Restart Apache:**

Bash

sudo systemctl restart apache2

Now, your Nextcloud instance should be using the managed RDS database. You can verify this by logging into Nextcloud and ensuring everything is working as expected. If you encounter issues, double-check the RDS endpoint, username, password, and the security group rules.

Part 4: Considering Amazon S3 for Scalable Storage (Optional but Recommended)

Using S3 for storage offers significant scalability and durability benefits. Here's how to start exploring this:

1. **Create an S3 Bucket:**
   - Navigate to the S3 service in the AWS Management Console.
   - Click "Create bucket".
   - Choose a globally unique bucket name.
   - Select the AWS Region that is geographically closest to you or where your EC2 instance is located.
   - Configure other bucket settings as needed (e.g., object ownership, block public access). For Nextcloud private storage, it's generally recommended to keep "Block all public access" enabled.
   - Click "Create bucket".
2. **Install the Nextcloud S3 Backend App:**
   - Log in to your Nextcloud instance as an administrator.
   - Click on your profile picture in the top right corner and go to "Apps".
   - In the app store, search for "External Storage Support".
   - Install and enable this app.
3. **Configure S3 as External Storage in Nextcloud:**
   - Go to your profile picture again and click on "Settings" -> "Administration" -> "External Storage".

- Click "Add storage" and choose "Amazon S3".
- Fill in the required details:
  - **Bucket:** The name of your S3 bucket.
  - **Hostname:** s3.your-region.amazonaws.com (replace your-region with the AWS Region where your bucket is located, e.g., s3.us-east-1.amazonaws.com).
  - **Region:** Your AWS Region (e.g., us-east-1).
  - **Access Key ID:** Your AWS access key ID.
  - **Secret Access Key:** Your AWS secret access key.
  - **Path/Folder:** You can specify a folder within the bucket if you want to isolate Nextcloud's data. Leave it blank to use the root of the bucket.
  - **Enable Previews:** (Optional)
  - **Enable Big File Chunking:** (Recommended for large files).
- **Important Security Note:** It's **highly recommended** to use **IAM Roles for EC2 instances** instead of hardcoding access keys and secret keys directly in Nextcloud. This is a more secure way to grant your EC2 instance permissions to access S3. To do this:
  - Create an IAM Role with permissions to access your S3 bucket (e.g., AmazonS3FullAccess or a more restrictive custom policy).
  - Attach this IAM Role to your EC2 instance.
  - In the Nextcloud S3 configuration, you can often leave the Access Key ID and Secret Access Key fields blank, and Nextcloud will attempt to use the instance's IAM Role. Consult the Nextcloud External Storage documentation for specific instructions on using IAM Roles.
- Click the checkmark to save the configuration.

You can now configure this S3 storage as either a primary storage location for new users or as an additional external storage that users can access. Migrating existing data from the local EC2 instance storage to S3 requires careful planning and execution (you might use tools like s3cmd or the AWS CLI).

Intermediate Level Enhancements:

- **HTTPS with ACM and ELB (Optional):** For a more robust setup, you could consider putting your EC2 instance behind an Elastic Load Balancer (ELB) and using AWS Certificate Manager (ACM) to manage SSL/TLS certificates. This adds a layer of security and can improve availability.
- **Basic Backups of RDS:** Configure automated backups for your RDS instance.
- **Monitoring with CloudWatch:** Start monitoring basic metrics of your EC2 and RDS instances using AWS CloudWatch.

Phase 5: Embracing Kubernetes and Infrastructure as Code (Expert/Advanced Level)

Our primary goals for this level are to:

- **Containerize Nextcloud:** Package Nextcloud and its dependencies into Docker containers.
- **Orchestrate with Kubernetes (EKS):** Deploy and manage Nextcloud using Amazon Elastic Kubernetes Service.
- **Manage Infrastructure with Terraform:** Define and provision all AWS resources (EKS cluster, RDS, S3, etc.) using Terraform.

Part 1: Containerizing Nextcloud with Docker

1. **Create Dockerfiles:** You'll need Dockerfiles to define how your Nextcloud containers will be built. This typically involves:
    - **Nextcloud Web Server (e.g., Apache with PHP-FPM):**

Dockerfile

```
FROM php:8.2-apache


# Install necessary PHP extensions

RUN docker-php-ext-install -j$(nproc) bcmath ctype curl gd imagick intl json ldap mbstring mysqli opcache pdo_mysql redis smbclient zip


# Enable Apache modules

RUN a2enmod rewrite headers env dir mime


# Install required system packages

RUN apt-get update && apt-get install -y --no-install-recommends \
   libmagickwand-dev --no-install-recommends && rm -rf /var/lib/apt/lists/*


# Download and extract Nextcloud

ARG NEXTCLOUD_VERSION=latest

RUN curl -o /tmp/nextcloud.tar.bz2 -L
https://download.nextcloud.com/server/releases/$NEXTCLOUD_VERSION.tar.bz2 \
   && tar -xjvf /tmp/nextcloud.tar.bz2 -C /var/www/html --strip-components=1 \
```

```
    && rm /tmp/nextcloud.tar.bz2

# Set file permissions
RUN chown -R www-data:www-data /var/www/html/
RUN chmod -R 755 /var/www/html/

# Copy Nextcloud configuration (optional - can be managed via environment variables/configmaps)
# COPY config/config.php /var/www/html/config/

# Set the document root for Apache
<VirtualHost *:80>
    DocumentRoot /var/www/html
    <Directory /var/www/html/>
        Options Indexes FollowSymlinks
        AllowOverride All
        Require all granted
    </Directory>
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

- You might also consider a separate **PHP-FPM container** if you want to decouple the PHP processing from the web server for better scalability. This would involve an Nginx web server container acting as a reverse proxy.

2. **Build Docker Images:**
    - Navigate to the directory containing your Dockerfile and build the image:

```
Bash
docker build -t your-dockerhub-username/nextcloud-web:latest .
```

Replace your-dockerhub-username with your Docker Hub username (or your AWS ECR repository URI later).

3. **Push Images to a Container Registry:**

- **Docker Hub:** If you're starting, you can push to Docker Hub:

Bash

```
docker login

docker push your-dockerhub-username/nextcloud-web:latest
```

- **AWS Elastic Container Registry (ECR):** For a production setup, ECR is recommended:

Bash

```
aws ecr create-repository --repository-name nextcloud-web --region your-aws-region

aws ecr get-login-password --region your-aws-region --profile your-aws-profile | docker login --username AWS --password-stdin your-aws-account-id.dkr.ecr.your-aws-region.amazonaws.com

docker tag your-dockerhub-username/nextcloud-web:latest your-aws-account-id.dkr.ecr.your-aws-region.amazonaws.com/nextcloud-web:latest

docker push your-aws-account-id.dkr.ecr.your-aws-region.amazonaws.com/nextcloud-web:latest
```

(Replace placeholders with your AWS details.)

Part 2: Orchestrating with Amazon EKS using Terraform

We'll use Terraform to provision the necessary EKS cluster and related resources.

1. **Install Terraform:** Follow the instructions on the Terraform website (https://www.terraform.io/downloads).
2. **Configure AWS Provider:** Create a provider.tf file:

Terraform

```
terraform {
 required_providers {
  aws = {
   source  = "hashicorp/aws"
   version = "~> 5.0" # Or the latest version
  }
  kubernetes = {
   source  = "hashicorp/kubernetes"
   version = "~> 2.0" # Or the latest version
  }
  helm = {
```

```
    source  = "hashicorp/helm"
    version = "~> 2.0" # Or the latest version
   }
 }
}

provider "aws" {
 region = "your-aws-region" # e.g., "us-east-1"
 # profile = "your-aws-profile" # If you're using an AWS CLI profile
}

provider "kubernetes" {
 host                   = module.eks.cluster_endpoint
 token                  = data.aws_eks_cluster_auth.cluster.token
 cluster_ca_certificate = base64decode(module.eks.cluster_certificate_authority_data)
}

provider "helm" {
 kubernetes {
   host                   = module.eks.cluster_endpoint
   token                  = data.aws_eks_cluster_auth.cluster.token
   cluster_ca_certificate = base64decode(module.eks.cluster_certificate_authority_data)
 }
}
```

3. **Define EKS Cluster in eks.tf:**

```
Terraform
module "eks" {
 source  = "terraform-aws-modules/eks/aws"
 version = "~> 19.0" # Or the latest version
```

```
  cluster_name    = "nextcloud-eks"
  cluster_version = "1.28" # Or your desired Kubernetes version
  vpc_id          = "your-vpc-id" # Replace with your VPC ID
  subnet_ids      = ["your-subnet-id-1", "your-subnet-id-2"] # Replace with your public subnet IDs

  node_groups = {
    default = {
      desired_capacity = 2
      max_capacity     = 4
      min_capacity     = 1
      instance_type    = "t3.medium" # Choose an appropriate instance type
      subnets          = ["your-private-subnet-id-1", "your-private-subnet-id-2"] # Use private subnets for worker nodes
      # You can configure scaling policies, launch templates, etc., here
    }
  }

  map_roles = [
    {
      rolearn  = "arn:aws:iam::your-aws-account-id:role/your-admin-eks-role" # Replace with your IAM role ARN for EKS admin access
      username = "eks-admin"
      groups   = ["system:masters"]
    }
  ]
}

data "aws_eks_cluster_auth" "cluster" {
```

```
  name = module.eks.cluster_name
}
```

(Replace all the your-... placeholders with your actual AWS resource IDs and desired configurations.)

4. **Define RDS Instance in rds.tf (if you haven't already):** Ensure your RDS instance is provisioned via Terraform for consistency. The configuration would be similar to what you did manually but defined in Terraform.
5. **Define S3 Bucket in s3.tf (if you haven't already):** Provision your S3 bucket using Terraform.
6. **Apply Terraform Configuration:**

Bash

```
terraform init
terraform plan
terraform apply -auto-approve
```

This will create your EKS cluster, RDS instance (if defined), and S3 bucket.

Part 3: Deploying Nextcloud to EKS with Kubernetes Manifests (YAML)

1. **Create Kubernetes Deployment (nextcloud-deployment.yaml):**

YAML

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nextcloud-web
spec:
  replicas: 2 # Start with 2 replicas for availability
  selector:
    matchLabels:
      app: nextcloud-web
  template:
    metadata:
      labels:
        app: nextcloud-web
```

```
  spec:
    containers:
      - name: nextcloud-web
        image: your-aws-account-id.dkr.ecr.your-aws-region.amazonaws.com/nextcloud-web:latest
        ports:
          - containerPort: 80
        env:
          - name: NEXTCLOUD_DB_HOST
            value: "your-rds-endpoint" # RDS Endpoint
          - name: NEXTCLOUD_DB_NAME
            value: "nextcloud"
          - name: NEXTCLOUD_DB_USER
            value: "admin" # RDS Master Username
          - name: NEXTCLOUD_DB_PASSWORD
            valueFrom:
              secretKeyRef:
                name: nextcloud-db-credentials
                key: password
          - name: NEXTCLOUD_TRUSTED_DOMAINS
            value: "your-load-balancer-dns,your-domain.com" # Add your domain(s)
        volumeMounts:
          - name: nextcloud-data
            mountPath: /var/www/html/data
    volumes:
      - name: nextcloud-data
        persistentVolumeClaim:
          claimName: nextcloud-pvc
```

2. **Create Kubernetes Service (nextcloud-service.yaml):** To expose Nextcloud via a Load Balancer.

YAML

```
apiVersion: v1
kind: Service
metadata:
  name: nextcloud-loadbalancer
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: alb
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing # Or internal
    alb.ingress.kubernetes.io/scheme: internet-facing # For Ingress (alternative)
spec:
  selector:
    app: nextcloud-web
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

3. **Create Kubernetes Persistent Volume Claim (nextcloud-pvc.yaml):** For persistent storage (consider using EFS or S3 via the Nextcloud app).

```
YAML
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nextcloud-pvc
spec:
  accessModes:
    - ReadWriteOnce # Or ReadWriteMany if using EFS
  resources:
    requests:
      storage: 100Gi # Adjust size as needed
```

storageClassName: "gp2" # Or your desired StorageClass (consider AWS EBS CSI driver)

4. **Create Kubernetes Secret (nextcloud-secrets.yaml):** To store sensitive information.

YAML

```
apiVersion: v1
kind: Secret
metadata:
  name: nextcloud-db-credentials
type: Opaque
data:
  password: $(echo -n "your-rds-master-password" | base64)
```

Replace "your-rds-master-password" with your actual RDS password.

5. **Apply Kubernetes Manifests:**

Bash

```
kubectl apply -f nextcloud-secrets.yaml
kubectl apply -f nextcloud-pvc.yaml
kubectl apply -f nextcloud-deployment.yaml
kubectl apply -f nextcloud-service.yaml
```

(You'll need kubectl configured to interact with your EKS cluster. Follow the AWS documentation on how to configure kubectl for your EKS cluster.)

Part 4: Configuring S3 as Persistent Storage within Nextcloud (Recommended)

As discussed in the intermediate level, using S3 via the Nextcloud "External Storage Support" app is a highly scalable and durable solution. You would install and configure this app within your containerized Nextcloud environment, ensuring your EKS worker nodes have the necessary IAM permissions to access the S3 bucket (using IAM Roles for Service Accounts - IRSA).

Advanced Considerations at the Expert Level:

- **Helm Charts:** Package your Kubernetes deployments using Helm for easier management and upgrades.
- **Ingress Controller:** Use an Ingress controller (like Nginx Ingress or AWS Load Balancer Controller) for more sophisticated routing and TLS termination.
- **Horizontal Pod Autoscaler (HPA):** Automatically scale the number of Nextcloud pods based on CPU utilization or other metrics.

- **Cluster Autoscaler:** Automatically scale the number of worker nodes in your EKS cluster based on the resource needs of your pods.
- **Monitoring and Logging:** Integrate Prometheus and Grafana for detailed monitoring and set up centralized logging with Elasticsearch, Fluentd, and Kibana (EFK stack) or AWS CloudWatch Logs.
- **CI/CD Pipelines:** Automate the building and deployment of your Docker images and Kubernetes manifests using tools like Jenkins, GitLab CI, or GitHub Actions.
- **Security Best Practices:** Implement network policies within Kubernetes, use secrets management solutions, and regularly audit your container images and infrastructure for vulnerabilities.