Report on CLI Design for Guided AWS Lab

Objective

The objective of this report is to provide an overview of a CLI-based approach for managing a VPC peering connection between two VPCs, configuring route tables, and managing AWS resources like log groups This design aligns with the steps necessary to implement a guided lab focusing on automating VPC networking configurations using AWS CLI

Step-by-Step CLI Design

1 Creating a VPC Peering Connection

The first step is to create a peering connection between two VPCs A VPC peering connection allows you to route traffic between two VPCs in the same or different regions

Command:

```
bash

aws ec2 create-vpc-peering-connection \
    --vpc-id <Requester-VPC-ID> \
    --peer-vpc-id <Accepter-VPC-ID> \
    --peer-region <Region>
```

Example:

```
bash

aws ec2 create-vpc-peering-connection \
    --vpc-id vpc-0c6f86f8cfb0fcd5 \
    --peer-vpc-id vpc-05e2eb3d4701556f0 \
    --peer-region us-east-1
```

2 Accepting the VPC Peering Connection

After creating the peering request, it needs to be accepted by the accepter VPC

Command:

bash

```
aws ec2 accept-vpc-peering-connection \
    --vpc-peering-connection-id <Peering-Connection-ID>
```

Example:

bash

```
aws ec2 accept-vpc-peering-connection \
    --vpc-peering-connection-id pcx-06dda4d948cea7df1
```

3 Adding Routes to Route Tables

Once the VPC peering connection is active, you need to add routes to the route tables in both VPCs to allow traffic to flow between the CIDR blocks

Command (for both VPCs):

bash

```
aws ec2 create-route \
    --route-table-id <Route-Table-ID> \
    --destination-cidr-block <Destination-CIDR> \
    --vpc-peering-connection-id <Peering-Connection-ID>
```

Example (for Requester VPC):

bash

```
aws ec2 create-route \
    --route-table-id rtb-0ece108eee6357fd7 \
    --destination-cidr-block 10500/16 \
```

```
    --vpc-peering-connection-id pcx-06dda4d948cea7df1
```

Example (for Accepter VPC):

```
bash

aws ec2 create-route \
    --route-table-id rtb-0b26db77bf3ca4b56 \
    --destination-cidr-block 10000/16 \
    --vpc-peering-connection-id pcx-06dda4d948cea7df1
```

4 Verifying Route Tables

After configuring the routes, it's essential to verify the routes to ensure that the traffic is correctly configured

Command:

```
bash

aws ec2 describe-route-tables --route-table-id <Route-Table-ID>
```

5 Creating a CloudWatch Log Group

Log groups in AWS CloudWatch allow you to monitor and troubleshoot your AWS infrastructure You can create a log group using the CLI

Command:

```
bash

aws logs create-log-group --log-group-name <Log-Group-Name>
```

Example: Create Flow Log Shared Vpc

```
bash
```

.

```
aws logs create-log-group --log-group-name My-VPC-Logs

Caws ec2 create-flow-logs \
  --resource-ids <Shared_VPC_ID> \
  --resource-type VPC \
  --traffic-type ALL \
  --log-destination-type cloud-watch-logs \
  --log-group-name ShareVPCFlowLogs \
  --deliver-logs-permission-arn arn:aws:iam::<AWS_Account_ID>:role/vpc-flow-logs-Role \
```

Make sure to replace `<AWS_Account_ID>` with your actual AWS account ID.

View the Flow Logs in CloudWatch

To verify that the flow logs are created, use:

```
bash

aws logs describe-log-groups --log-group-name-prefix ShareVPCFlowLogs
```

These steps will allow you to create the VPC peering connection, configure routing, and set up VPC Flow Logs via the AWS CLI. Let me know if you want further assistance with any specific step!

To create a CloudWatch log group using the AWS CLI, you can use the `aws logs create-log-group` command. Here's the syntax:

Command to Create a Log Group

```
bash

aws logs create-log-group --log-group-name <Log_Group_Name>
```

Example: To create a log group named `ShareVPCFlowLogs`:

```
aws logs create-log-group --log-group-name ShareVPCFlowLogs
```

This will create a CloudWatch Logs group named `ShareVPCFlowLogs`, which you can use for VPC Flow Logs or any other CloudWatch logging purposes.

6 Tagging Resources (Optional)

AWS encourages tagging for resource management You can add tags to the VPCs, route tables, and other resources for easier identification and management

Command:

```
bash

aws ec2 create-tags --resources <Resource-ID> --tags
Key=<Key>,Value=<Value>
```

Example:

```
bash

aws ec2 create-tags --resources vpc-0c6f86f8cfb0fcd5 --tags
Key=Environment,Value=Lab
```

Summary

This CLI-based design simplifies and automates the configuration of VPC peering and routing in a guided AWS lab The key steps include:

- Creating and accepting a VPC peering connection between two VPCs
- Configuring route tables to enable communication between the VPCs
- Verifying route tables to ensure traffic is properly routed
- Creating CloudWatch log groups for monitoring and logging purposes
- Optionally tagging AWS resources for improved resource management

Benefits of CLI-Based Design

- **Efficiency**: Automates tasks that would otherwise require manual configuration via the AWS Management Console
- **Scalability**: CLI scripts can be reused in various AWS regions or accounts, making it easier to replicate the same infrastructure
- **Automation**: These commands can be integrated into scripts, reducing manual intervention and human errors in cloud management
- **Version Control**: CLI commands/scripts can be version-controlled and documented for future reference

This design is ideal for lab environments and scenarios where multiple students or users need to configure VPC peering connections and manage AWS resources consistently