Midterm Project-Chord, Part 2: Auto-Scaling on AWS Cloud

> Please be mindful of avoiding extra charges when using your AWS account:
> [How to Avoid](#)

The design of the Chord protocol allows for easy scability. In Part 2 of the midterm project, you will be leveraging AWS infrastructure and Chord node implementation from Part 1 to build a distributed file system that can automatically scale out on the AWS cloud. In this system, a cloud node (computing unit) will be running a Chord node and a file server. Multiple Chord nodes must be deployed on individual cloud machines respectively.

There are several ways to deploy the Chord-based distributed file system with auto-scaling capabilities on the cloud, and an example architecture is provided. You may adopt a different architecture design as long as it meets the requirements, but it is highly recommended to discuss your design with TAs beforehand.

This document provides an operational guide for creating a **basic** auto-scaling file system, which serves as a foundation for developing an automatically scalable Chord-based distributed file system. The process of building the architecture can be divided into two phases. In the first phase, you will run a file server on a single AWS instance without enabling auto-scaling. In the second phase, your distributed file system should automatically scale new EC2 instances running new file servers, based on the disk usage of each instance.

Once completed, you should be able to integrate Chord nodes into the system yourself. The final architecture should resemble the diagram below:
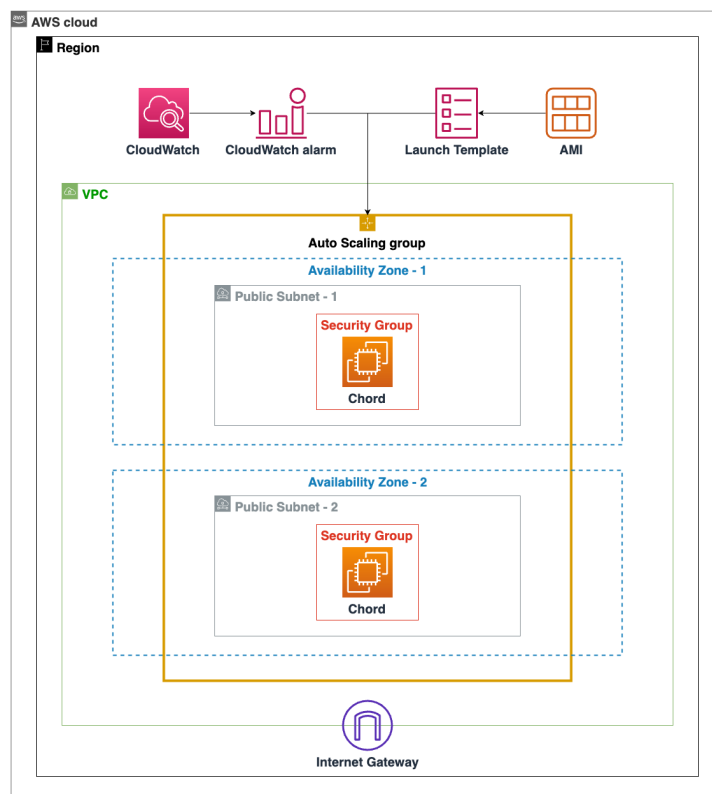
Prerequisite

1. Newly created (in less than 12 months) AWS account
   or
2. AWS academy account (with $100 free credit for each user)

Also, download and extract the assignment data:

```
$ wget ntu.im/IM5057/chord-part-2.tar.gz
$ tar zxvf chord-part-2.tar.gz
```

The `chord-part-2` folder contains the following directories:

```
chord-part-2/
|
+-- chord
|
+-- download.py
|
+-- upload.py
|
+-- test_download.py
|
+-- test_upload.py
```

The `chord` file is a compiled binary that meets the requirements in Part 1 of the midterm project. You may use this for your Chord nodes if you did not finish your implementation.

`upload.py`, `download.py`, `test_upload.py` and `test_download.py` are Python scripts for uploading and downloading files to/from the file server `uploadserver`, which will be introduced later.

Phase 1: Running the First Node on AWS Cloud

In phase 1, you will be creating some services using the AWS console and run a simple file server on a single cloud node.

> To prevent additional costs, follow these two optional steps:
>
> **Create Budget Alart**
>
> - AWS Billing > Budgets > Create budget
>
> **Enable Free Tier Usage Alart**
>
> - AWS Billing > Billing preferences > Receive Free Tier Usage Alerts

Create IAM Role

- Go to IAM > Roles > Create role
- Create a new policy and allow IAM:PassRole in the policy visual editor

- Create a new role for your service
- Attach predefined Policy EC2:AmazonEC2FullAccess and the policy you created

Create Security Group

- Go to EC2 > Network & Security > Security Groups > Create security group

Modify Security Group Inbound Rules

- Go to EC2 > Network & Security > Security Groups > Select target security group > Inbound Rules > Edit inbound rules

You should:

1. Add all TCP inbound rules from yourself (security group), **not from 0.0.0.0**, to allow communications between nodes in same security group. All instances should be configured in the same security group.
2. Add ssh and TCP from your IP or 140.112.0.0/16 to allow your local computer to access the EC2 instances (using VPN). **You are not recommended to allow connections from all IPs to your instances.**

Create Free Tier EC2 Instances

EC2 > Instances > Instances > Launch instances
Important instance specifications:

- OS image: Amazon Linux 2
- Instance type: t2.micro (under free tier)
- Keypair: create or attach one
- IAM role: attach the role you just create
- Security group: attach the security group you just create

Connect EC2 instance through SSH

- Go to EC2 > Instances > Select target intance > Connect > SSH Client

> Wait until the EC2 is fully initialized and ready.

Install Amazon CloudWatch Agent

```
$ sudo yum install amazon-cloudwatch-agent -y
```

Install and Run File Server

We will be using Python `uploadserver` as a simple file server. To install the file server:

```
$ sudo yum install -y python3-pip python3 python3-setuptools -y
$ pip3 install uploadserver
```

To start the file server listening on port 5058:

```
$ python3 -m uploadserver 5058
```

Install some Additional Requirements

```
$ pip3 install boto3 ec2_metadata
```

(Optional) Launch CloudWatch Wizard

Launch CloudWatch Wizard to generate metric config file:

```
$ sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard
```

Enable CloudWatch Agent

Create the metric config file and start the CloudWatch agent. You may need to modify the config file even if you created with the Wizard. Please refer to **Cloudwatch Agent** and **Cloudwatch Agent Config Detail** for more information.

In your config file:

- set `agent.metrics_collection_interval` to 60 seconds
- add `AutoScalingGroupName` to `metrics.append_dimensions`
- add `used_percent` at `metrics.disk.measurement` to collect `disk_used_percent` for each node

After finishing your config file, start the CloudWatch agent:

```
$ sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config
-m ec2 -s -c file:<your config file>
```

(Optional) CloudWatch Verification

It may take some time for the metrics to be displayed on the CloudWatch dashboard. To verify your CloudWatch connection, you may use telnet:

```
$ sudo yum install telnet -y
$ telnet monitoring.ap-northeast-1.amazonaws.com 443
```

Check CloudWatch Metrics

- Go to CloudWatch > Metrics > Browse > CWAgent > device, fstype, path, then select desired metrics
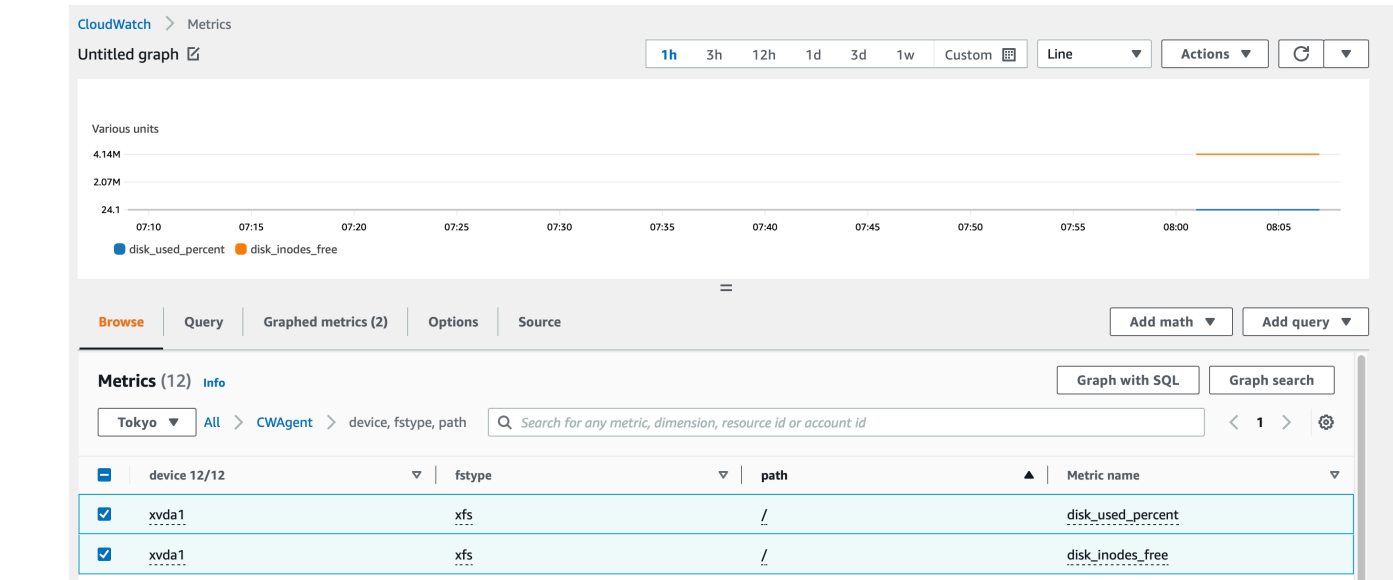
> Remember to change the metric period to one minute in the tab "Graphed metrics".

Test your File Server from Local or Another EC2 instance

After successfully running `uploadserver` on your EC2 instances, you can use the `test_upload.py` and `test_download.py` scripts for testing the access to your `uploadserver`. For example, you may use

```
$ ./test_upload.py a.txt 10.10.10.10
```

to upload the `a.txt` file onto the file server running on 10.10.10.10. You can also check if the `a.txt` file appears on your 10.10.10.10 instance.

Similarly, you may use

```
$ ./test_download.py a.txt 10.10.10.10
```

to download the `a.txt` file from the file server running on 10.10.10.10.

Phase 2: Enabling Auto-scaling

In this phase, we will use the instance created in part 1 to build the auto-scaling file system.

Create AMI

First, we need to create a new Amazon Machine Image (AMI) based on the instance we created in part 1.

- Go to EC2 > Instances > Instances > Select target intance > Actions > Image and templates > Create image

Launch an Auto-scaling Group

- EC2 > Instances > Launch Templates > Create launch template

Create Launch Configuration

1. Select the AMI you created in the previous step.
2. Modfiy the template details, and select the same instance specification as in phase 1.
3. Select the Security Group, IAM, keypair, etc.
4. Add User data and make sure it runs successfully when new instance is created.

An example of User data:

```
#!/bin/sh
/bin/echo "Hello World" >> /tmp/testfile.txt
sudo yum install -y python3-pip python3 python3-setuptools -y
/usr/bin/python3 -m pip install boto3 ec2_metadata uploadserver
/usr/bin/python3 -m uploadserver 5058 --directory /home/ec2-user/files
--//--
```

The command

```
$ /usr/bin/python3 -m uploadserver 5058 --directory /home/ec2-user/files
```

runs the `uploadserver` listening on port 5058 and uses `/home/ec2-user/files` as the file directory.

> Please keep an eye on the source version when modifying the launch template.

> Note that cloud-init will run the user data only at the time when the instance is created, and the log will be stored in `/var/log/cloud-init-output.log`.

Create Auto-scaling Group

Go to EC2 > Auto Scaling Group > Create

- name, desired launch template
- VPC, AZ
- no load balancer, reduce Health check grace period, optional settings
- group size:
    - min=1, max=3, desire=2 for frugality
    - min=2, max=5, desire=3 for better experience if you have free credits :)
- Scaling policies: you can attach it later

> If you are going to create/delete the auto scaling group multiple times, it is recommended to write the `create_auto_scaling_group.py` script and run it.

Create CloudWatch Alarm and Attach it to Auto-scaling Group

- Go to EC2 > Auto Scaling Group
- Go to Cloudwatch > Alarms > Create alarm

> If you are going to create and attach alarms multiple times, it is recommended to write the `create_scaling_policy.py` script and run it.

Your file system is now set up to achieve auto-scaling on AWS.

Requirements

After deploying the file servers and configuring auto-scaling metrics, you are required to integrate Chord nodes into the system.

When a new cloud node is initialized, in addition to launching a `uploadserver` file server, the cloud node should also start a Chord node, and make the Chord node join the existing Chord ring system. The Chord nodes should listen on port 5057, and the `uploadserver` file server should listen on port 5058.

> To join the existing Chord ring system, you can use AutoScaling.Client.describe_auto_scaling_groups to find IDs of all running instances in the Auto-scaling Group, and then use EC2.Client.describe_instances to obtain their public IP addresses.

You can use the `upload.py` and `download.py` scripts for testing the correctness of your Chord-based distributed file system. Different from `test_upload.py` and `test_download.py`, `upload.py` and `download.py` compute the hash of the file name, and find the specific cloud node for storing the file by sending a `"find_successor"` RPC request, before uploading and downloading a file.

To upload an `a.txt` file onto the file server, we can run the `upload.py` script by providing the file name and the URL(IP address) of an arbitrary cloud node as arguments:

```
$ ./upload.py a.txt 10.10.10.10
Hash of a.txt is 607204990
Uploading file to http://20.20.20.20
```

In this example, the IP 10.10.10.10 is the IP address of a random cloud node in the deployed system. However, the IP is not necessary the IP of the file server for uploading the file.

The script first calculates the 32-bit hash of the file name (in this case, 607204990). After this, the script calls the `"find_successor"` RPC of the Chord node 10.10.10.10:5057, and gets the Node information of the cloud node for uploading the file (in this case, 20.20.20.20). Finally, the script uploads the file to the file server 20.20.20.20:5058. Similarly, the `download.py` script calculates the hash of the file name, get the Node information by calling the `"find_successor"` RPC, and finally download the file from the file server on the cloud node.

1. Auto-Scaling (50%)

Your system should be capable of automatically adding new cloud nodes when the storage usage hits the threshold.

Please note that if your system failed to meet the Auto-Scaling requirement, you will not receive credit for the next part.

2. Integrating Chord Nodes (50%)

A Chord node should be started together with the file server on each of the cloud nodes. The Chord node should be able to join each other and together form a Chord ring. Also, you should be able to upload and download files to/from your distributed file system using the provided `upload.py` and `download.py` scripts on your local machine.

You are required to demonstrate specific functionalities to the TA for grading. Parts 2 and 3 of the midterm project will be evaluated together during the demonstration.

Appendix: Friendly Reminders

Remember to clean up your AWS account and stop unused resources. Here are some tips:

- stop/terminate EC2
- deregister AMI
- delete Auto scaling group
- delete Launch Template

- delete Additional CloudWatch metrices
- delete all the unused resourses
- for more information, please check out: [find unactive services to avoid unexpected charges](#)