Minecraft Server with CloudWatch Monitoring

Project Overview

This project creates a Minecraft server with real-time player activity monitoring. It combines AWS services to display player logins, logouts, and other server events on a web dashboard.

What You'll Build

1. **Minecraft Server** - A Java Edition Minecraft server running on AWS Lightsail
2. **Monitoring System** - A CloudWatch-powered log collection and processing pipeline
3. **Web Dashboard** - A simple web interface showing real-time player activity

Architecture Overview

This solution uses these AWS microservices:

- **AWS Lightsail** hosts the Minecraft server and generates log files
- **CloudWatch Agent** installed on the Lightsail instance collects these logs
- **CloudWatch Logs** stores and manages server log data
- **Lambda Function** processes log events when triggered by CloudWatch
- **EC2 Web Server** receives processed log data and displays it on a web interface
- **VPC** provides secure networking between all components

Prerequisites

Software Requirements

- Minecraft Java Edition client (to test server connection)
- Web browser to access AWS Console
- SSH client (built into browser for Lightsail)

Cost

- Lightsail instance: ~$5/month

- EC2 t2.micro: ~$8.50/month
- Lambda: Free tier includes 1M free requests per month
- CloudWatch: First 5GB of logs ingested is free

Knowledge

- Basic understanding of AWS services
- Basic Linux command line knowledge
- Familiarity with editing text files in terminal (using nano editor)

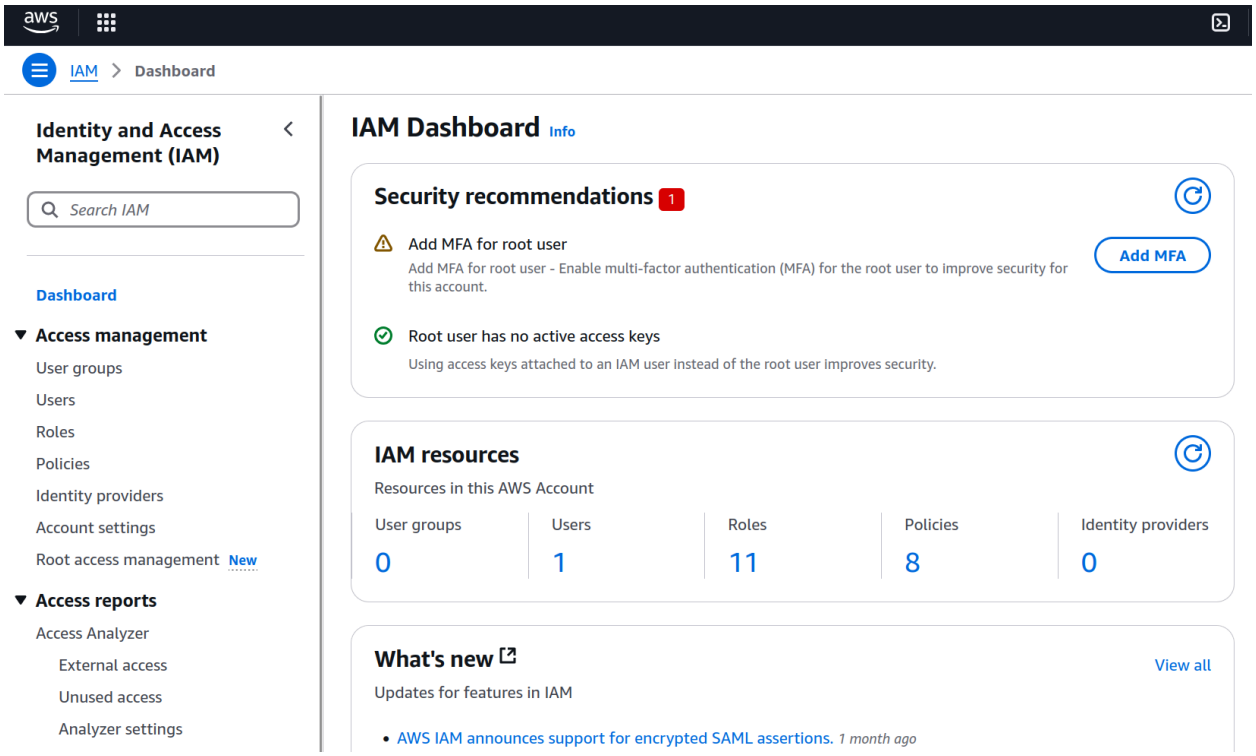
Pre-Installation Steps

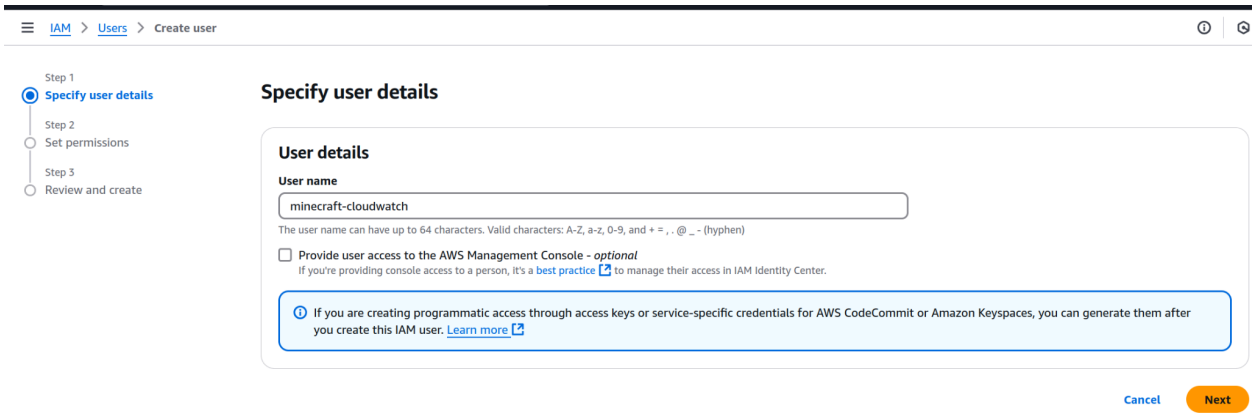
- Make sure you are on us-east-1 (virginia)
- You can only copy paste in lightsail terminal using right click to paste

Project Start

Creating user for cloud watch with permissions

Go to users tab and create a user





Attach policies to them

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. Learn more

Permissions options

- Add user to group — Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.
- Copy permissions — Copy all group memberships, attached managed policies, and inline policies from an existing user.
- Attach policies directly — Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1343)

Choose one or more policies to attach to your new user.

Create policy

Filter by Type

cloudwatchlogsf | All types | 1 match | 1

Policy name	Type	Attached entities
CloudWatchLogsFullAccess	AWS managed	1

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. Learn more

Permissions options

- Add user to group — Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.
- Copy permissions — Copy all group memberships, attached managed policies, and inline policies from an existing user.
- Attach policies directly — Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (2/1343)

Choose one or more policies to attach to your new user.

Create policy

Filter by Type

cloudwatchage | All types | 2 matches | 1

Policy name	Type	Attached entities
CloudWatchAgentAdminPolicy	AWS managed	0
CloudWatchAgentServerPolicy	AWS managed	1

▶ **Set permissions boundary** - *optional*

Make sure these are the policies at the end

Permissions summary

Name	Type	Used as
CloudWatchAgentServerPolicy	AWS managed	Permissions policy
CloudWatchLogsFullAccess	AWS managed	Permissions policy

Tags - *optional*

Tags are key-value pairs you can add to AWS resources to help identify, organize, or search for resources. Choose any tags you want to associate with this user.

No tags associated with the resource.

Add new tag

You can add up to 50 more tags.

Cancel Previous Create user

Create an access key

Summary

ARN
arn:aws:iam::831926625893:user/minecraft-cloudwatch2

Created
March 22, 2025, 01:12 (UTC-04:00)

Console access
Disabled

Last console sign-in
-

Access key 1
Create access key

Use case

- ○ Command Line Interface (CLI)
  You plan to use this access key to enable the AWS CLI to access your AWS account.
- ○ Local code
  You plan to use this access key to enable application code in a local development environment to access your AWS account.
- ◉ Application running on an AWS compute service
  You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.
- ○ Third-party service
  You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.
- ○ Application running outside AWS
  You plan to use this access key to authenticate workloads running in your data center or other infrastructure outside of AWS that needs to access your AWS resources.
- ○ Other
  Your use case is not listed here.

⚠ **Alternative recommended**
Assign an IAM role to compute resources like EC2 instances or Lambda functions to automatically supply temporary credentials to enable access. Learn more

Confirmation

☐ I understand the above recommendation and want to proceed to create an access key.

Cancel **Next**

Create a descriptive name if you want
Copy these two keys somewhere!

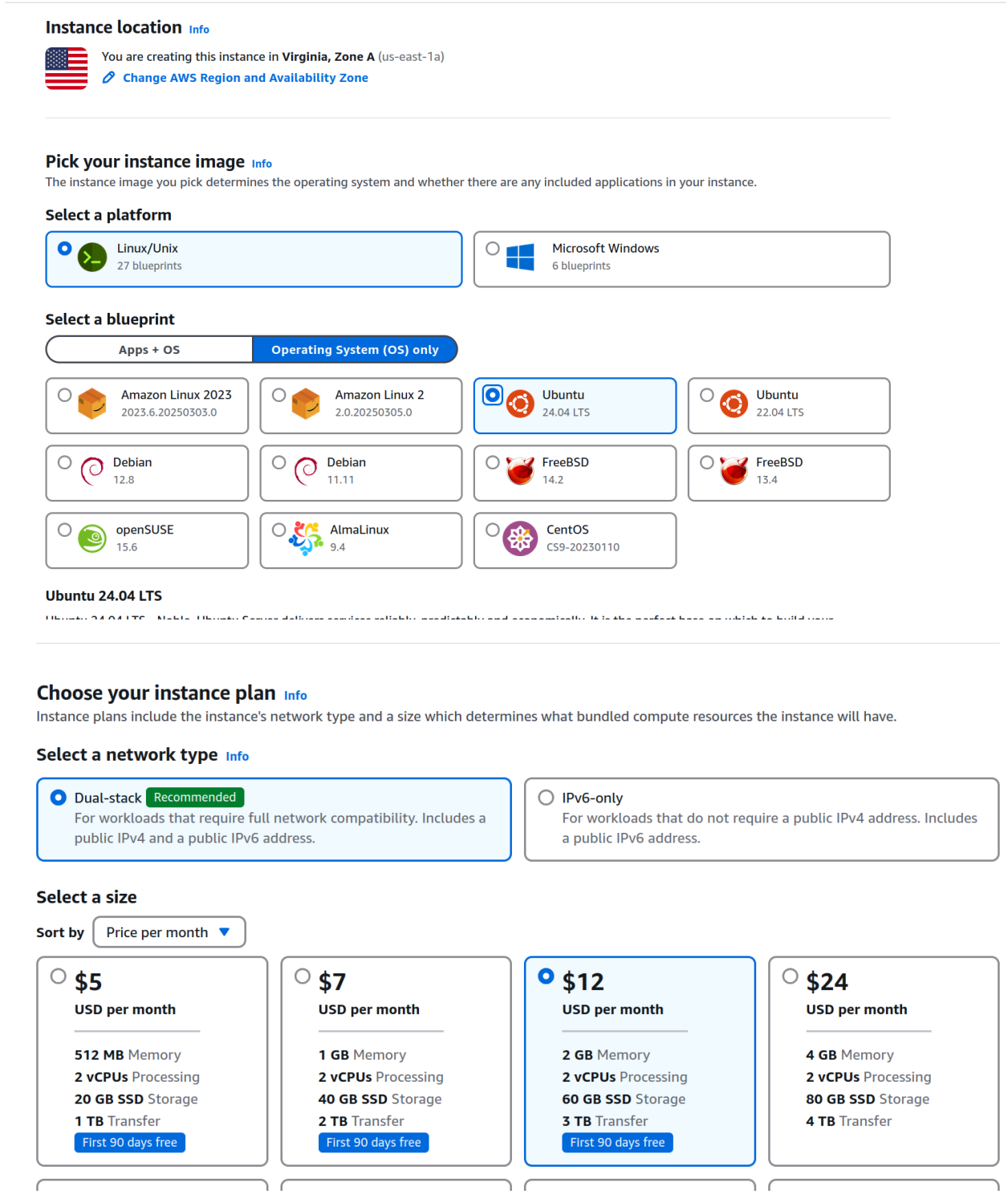
Retrieve access keys Info

Access key
If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key	Secret access key
AKIA4DMVQ4ZS2GWO4W6Q	LXoG45xmIq2S5moGWWDm8fxiui/jVXg9mskn7sDy Hide

Create Lightsail instance

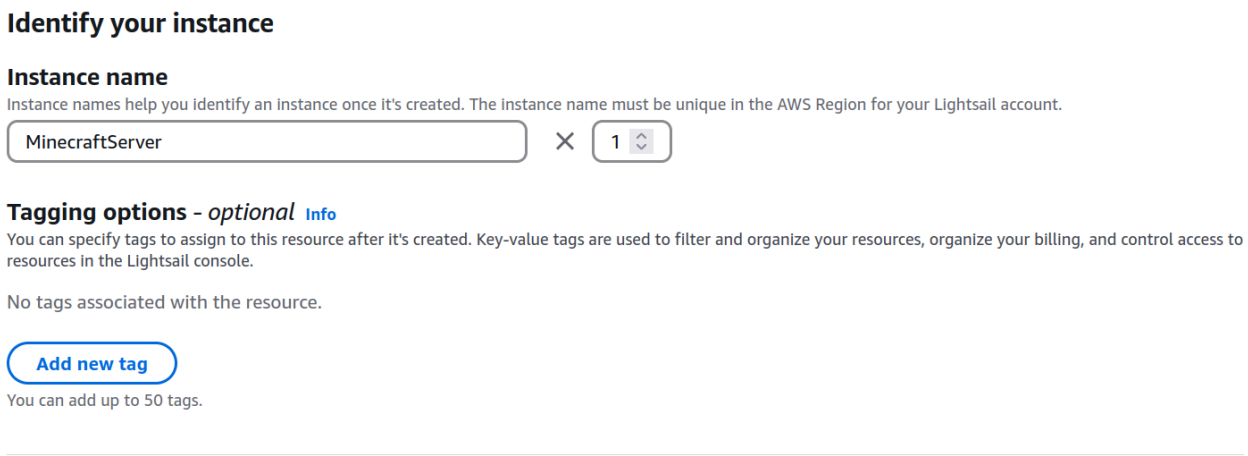
Make sure you are using us-east-1a and using Ubuntu 24.04



Start up script

```
apt-get update
```

```
cd /home/ubuntu/
wget
https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-
cloudwatch-agent.deb
dpkg -i -E ./amazon-cloudwatch-agent.deb
apt install -y tmux
apt install -y default-jre
mkdir minecraft-server
cd minecraft-server
wget
https://piston-data.mojang.com/v1/objects/4707d00eb834b446575d89a61a11b5d54
8d8c001/server.jar
java -Xmx1G -Xms1G -jar server.jar nogui
```



Adding Cloudwatch agent to Lightsail Instance

sudo nano /opt/aws/amazon-cloudwatch-agent/bin/config.json

```
{
  "agent": {
      "metrics_collection_interval": 60,
      "run_as_user": "root"
  },
  "logs": {
      "logs_collected": {
```

```
      "files": {
      "collect_list": [
            {
            "file_path": "/home/ubuntu/minecraft-server/logs/latest.log",
            "log_group_name": "minecraft-server-logs",
            "log_stream_name": "{instance_id}-minecraft",
            "retention_in_days": 14
            }
      ]
      }
      }
  }
}
```

Ctrl+x, y, then enter to save

```
sudo nano /opt/aws/amazon-cloudwatch-agent/etc/common-config.toml
```

Copy paste this at the end, after all the comments

```
[credentials]
shared_credential_profile = "minecraft-cloudwatch"
shared_credential_file = "/home/ubuntu/.aws/credentials"
```

```
mkdir ~/.aws/
nano ~/.aws/credentials
```

paste this in, replacing with the values you copied earlier

```
[minecraft-cloudwatch]
aws_access_key_id = YOUR_ACCESS_KEY_ID
aws_secret_access_key = YOUR_SECRET_ACCESS_KEY
region = us-east-1
```

now paste this whole thing in, all one line

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a
fetch-config -m ec2 -s -c
file:/opt/aws/amazon-cloudwatch-agent/bin/config.json
```

Check if its running properly

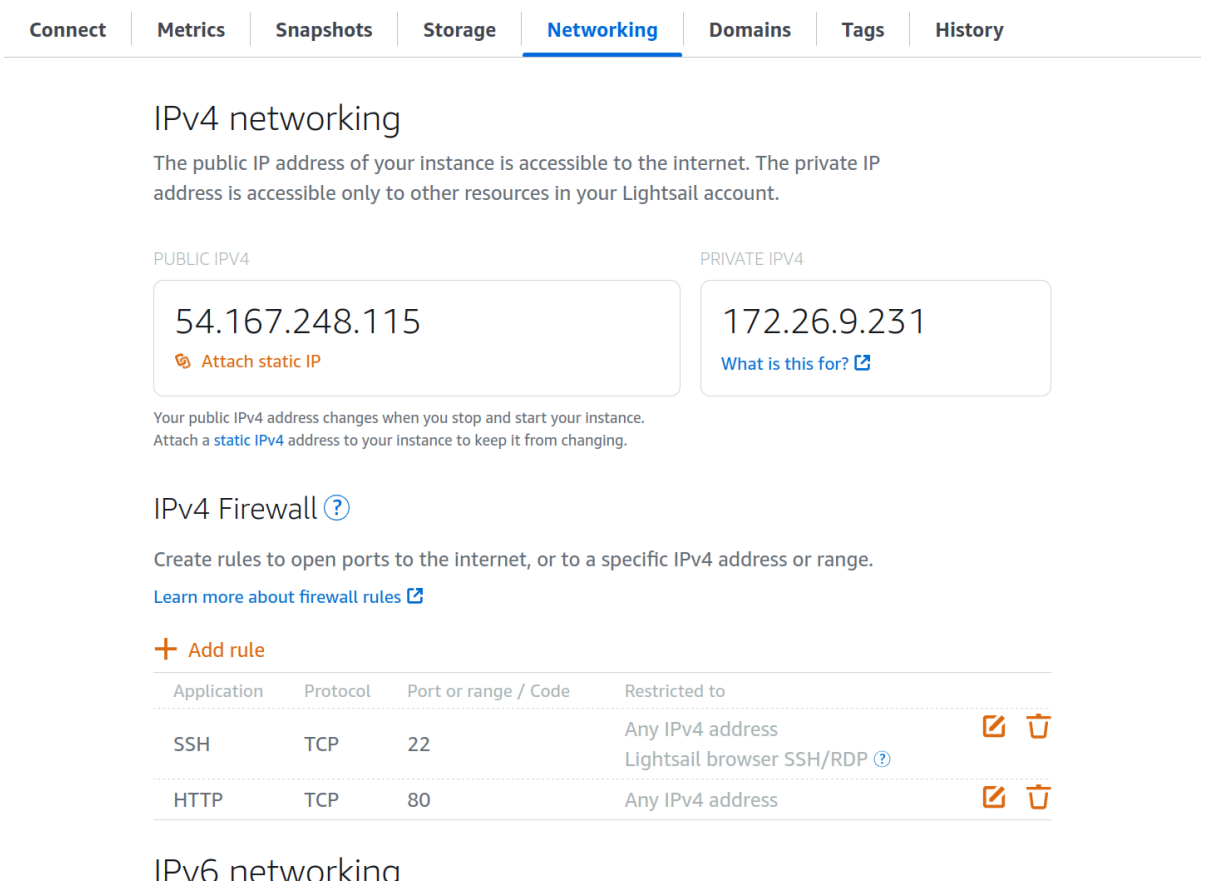
```
sudo systemctl status amazon-cloudwatch-agent
```

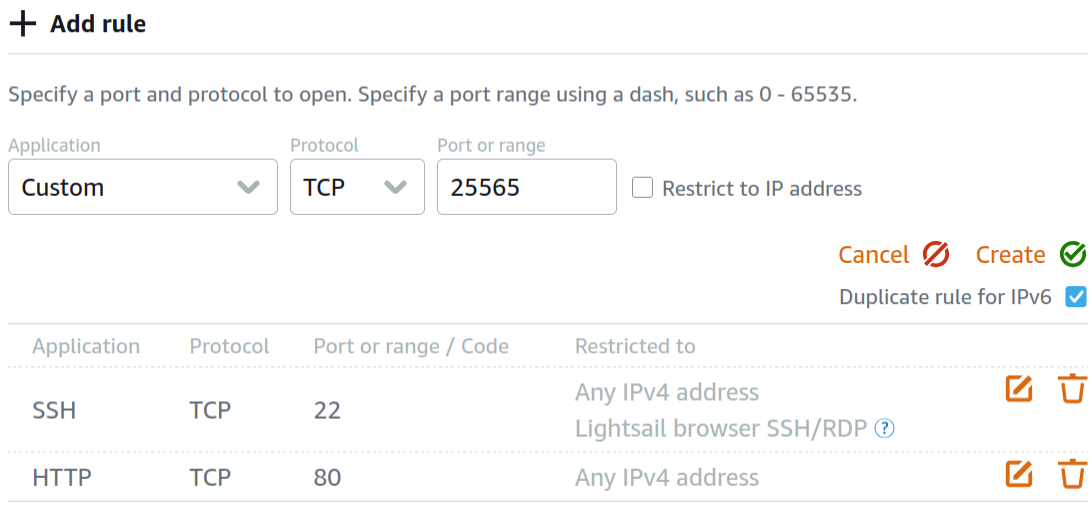
```
nt.service.
ubuntu@ip-172-26-9-231:~/.aws$ sudo systemctl status amazon-cloudwatch-agent.service
● amazon-cloudwatch-agent.service - Amazon CloudWatch Agent
     Loaded: loaded (/etc/systemd/system/amazon-cloudwatch-agent.service; enabled; preset: enabled)
     Active: active (running) since Sat 2025-03-22 06:05:36 UTC; 32s ago
   Main PID: 3965 (amazon-cloudwat)
      Tasks: 8 (limit: 2274)
     Memory: 32.1M (peak: 32.4M)
        CPU: 332ms
     CGroup: /system.slice/amazon-cloudwatch-agent.service
             └─3965 /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent -config /opt/aws/amazon-cloudwa

Mar 22 06:05:36 ip-172-26-9-231 start-amazon-cloudwatch-agent[3970]: 2025/03/22 06:05:36 Reading json config fi
Mar 22 06:05:36 ip-172-26-9-231 start-amazon-cloudwatch-agent[3970]: /opt/aws/amazon-cloudwatch-agent/etc/amazo
Mar 22 06:05:36 ip-172-26-9-231 start-amazon-cloudwatch-agent[3970]: 2025/03/22 06:05:36 Reading json config fi
Mar 22 06:05:36 ip-172-26-9-231 start-amazon-cloudwatch-agent[3970]: 2025/03/22 06:05:36 I! Valid Json input sc
Mar 22 06:05:36 ip-172-26-9-231 start-amazon-cloudwatch-agent[3970]: I! Detecting run_as_user...
Mar 22 06:05:36 ip-172-26-9-231 start-amazon-cloudwatch-agent[3970]: Got Home directory: /root
Mar 22 06:05:36 ip-172-26-9-231 start-amazon-cloudwatch-agent[3970]: I! Set home dir Linux: /root
Mar 22 06:05:36 ip-172-26-9-231 start-amazon-cloudwatch-agent[3970]: I! SDKRegionWithCredsMap region:  us-east-
Mar 22 06:05:36 ip-172-26-9-231 start-amazon-cloudwatch-agent[3970]: 2025/03/22 06:05:36 Configuration validati
Mar 22 06:05:36 ip-172-26-9-231 start-amazon-cloudwatch-agent[3965]: I! Detecting run_as_user...
```

Should be active (running)

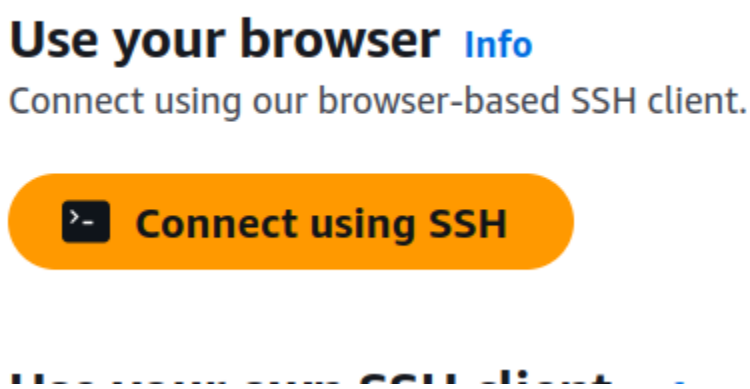
Start Minecraft Server

First open up port 25565 for Minecraft





Now use browser connect to ssh




```
tmux new -s minecraft
cd ~/minecraft-server
```

```
sudo nano eula.txt
```

change eula=true
ctrl+x and press Y and then enter to save
then do

```
sudo java -Xmx1G -Xms1G -jar server.jar nogui
```

press ctrl+b and then press d

Try joining your minecraft server by using your lightsail instance ip and port 25565

restart the cloudwatch amazon agent. This is because a new log file has been made, so it must be restarted

```
sudo systemctl restart amazon-cloudwatch-agent
```

Check this to make sure there are no errors

```
sudo tail -n 100
/var/log/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.log
```

Create a VPC for the other services

VPC settings

Resources to create Info

Create only the VPC resource or the VPC and other networking resources.

- ○ VPC only
- ◉ VPC and more

Name tag auto-generation Info

Enter a value for the Name tag. This value will be used to auto-generate Name tags for all resources in the VPC.

☑ Auto-generate

MinecraftLogServerVPC

IPv4 CIDR block Info

Determine the starting IP and the size of your VPC using CIDR notation.

10.0.0.0/16 65,536 IPs

CIDR block size must be between /16 and /28.

IPv6 CIDR block Info

- ◉ No IPv6 CIDR block
- ○ Amazon-provided IPv6 CIDR block

Tenancy Info

Default

Number of Availability Zones (AZs) Info

Choose the number of AZs in which to provision subnets. We recommend at least two AZs for high availability.

1 | 2 | 3

▶ **Customize AZs**

Number of public subnets Info

The number of public subnets to add to your VPC. Use public subnets for web applications that need to be publicly accessible over the internet.

0 | 1

NAT gateways ($) Info

Choose the number of Availability Zones (AZs) in which to create NAT gateways. Note that there is a charge for each NAT gateway

None | In 1 AZ | 1 per AZ

VPC endpoints Info

Endpoints can help reduce NAT gateway charges and improve security by accessing S3 directly from the VPC. By default, full access policy is used. You can customize this policy at any time.

None | S3 Gateway

DNS options Info

- [x] Enable DNS hostnames
- [x] Enable DNS resolution

▶ **Additional tags**

Creating EC2 Web Server

Go to AMI catalog and search up the AMI ami-04b2888da26fcb333 in the catalog

▼ **Images**

AMIs

AMI Catalog

AMI Catalog

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

AMIs

Create Template with AMI | Launch Instance with AMI

ami-04b2888da26fcb333

Quick Start AMIs (0)	My AMIs (0)	AWS Marketplace AMIs (6496)	Community AMIs (500)
Commonly used AMIs	Created by me	AWS & trusted third-party AMIs	Published by anyone

Refine results

Clear all filters

- Operating system
- Linux/Unix
  - All Linux/Unix
  - Amazon Linux
  - CentOS
  - Debian
  - Fedora
  - Gentoo
  - macOS
  - openSUSE
  - Other Linux
  - Red Hat

ami-04b2888da26fcb333 (1+ filtered, 500+ unfiltered)

1 ...

Loading AMIs

Stop loading AMIs

Community AMIs
Community AMIs contain all AMIs that are public, therefore anyone can publish an AMI and it will show in this catalog. This catalog can also contain paid products. When using community AMIs it is best practice to ensure you know and trust the publisher before launching an AMI.

aws

MinecraftLogWebServer
ami-04b2888da26fcb333
OwnerAlias: – Platform: – Architecture: x86_64 Owner: 831926625893 Publish date: 2025-03-21
Root device type: ebs Virtualization: hvm ENA enabled: Yes Boot mode: uefi-preferred

Select

Select it and launch an instance

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps belov

Name and tags Info

Name

MinecraftLogWebServer

Add additional tags

Application and OS Images (Amazon Machine Image) Info

Instance type Info | Get advice

Instance type

t2.micro Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Windows base pricing: 0.0162 USD per Hour
On-Demand Ubuntu Pro base pricing: 0.0134 USD per Hour
On-Demand SUSE base pricing: 0.0116 USD per Hour On-Demand RHEL base pricing: 0.026 USD per Hour
On-Demand Linux base pricing: 0.0116 USD per Hour

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

Key pair (login) Info

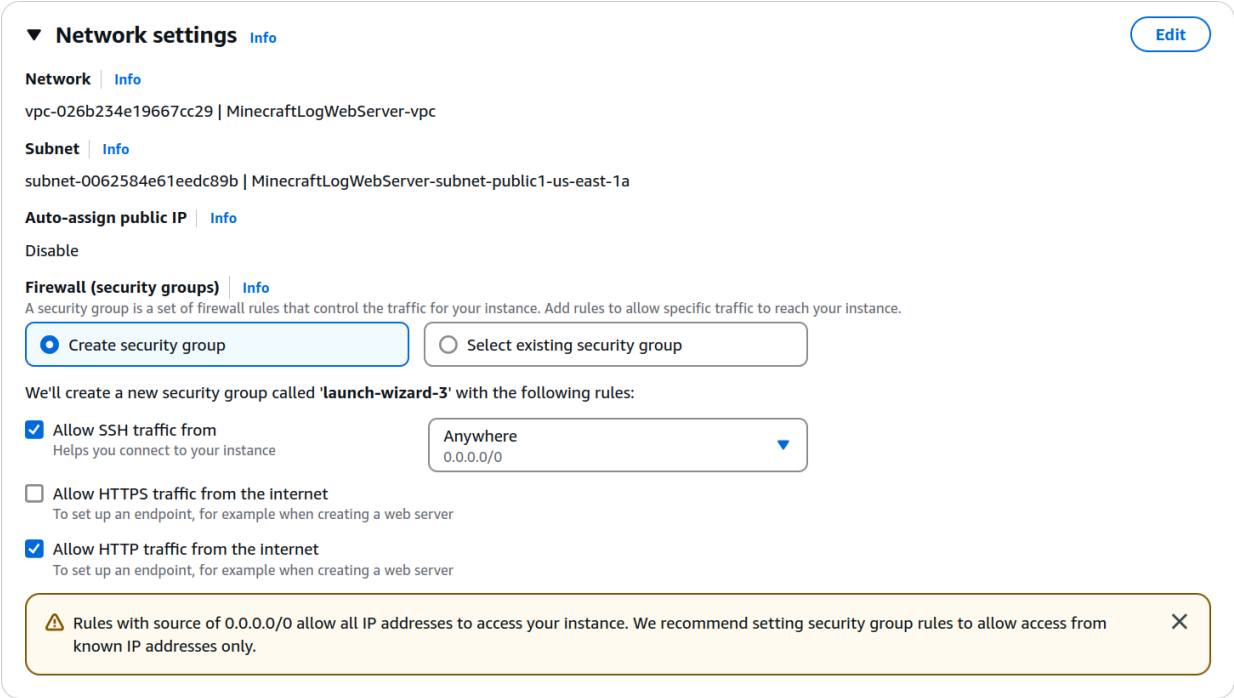
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

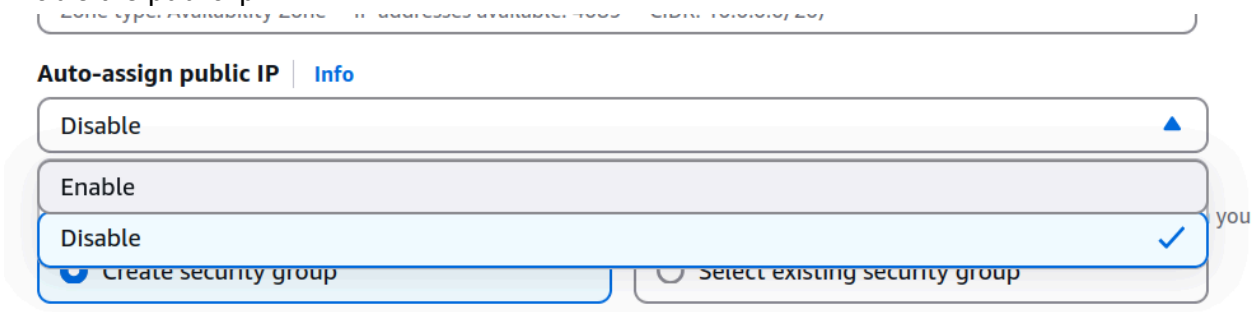
Proceed without a key pair (Not recommended) Default value

Create new key pair

Allow HTTP Traffic. Then Click edit to further edit network settings



Enable the public ip



Create the instance and finish. Record the internal DNS somewhere of the instance.

Creating a lambda function to send information to EC2

Basic information

Function name
Enter a name that describes the purpose of your function.

SendDataToEC2

A function with the same name already exists in your account.

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Node.js 22.x

Architecture Info
Choose the instruction set architecture you want for your function code.

- (●) x86_64
- ( ) arm64

Permissions Info

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▸ **Change default execution role**

By default, Lambda encrypts the .zip file archive using an AWS owned key.

☐ **Enable function URL** Info
Use function URLs to assign HTTP(S) endpoints to your Lambda function.

☐ **Enable tags** Info
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources, track your AWS costs, and enforce attr
access control.

☑ **Enable VPC** Info
Connect your function to a VPC to access private resources during invocation.

VPC
Choose a VPC for your function to access.

vpc-026b234e19667cc29 (10.0.0.0/16)

☐ Allow IPv6 traffic for dual-stack subnets
You can allow outbound IPv6 traffic to subnets that have both IPv4 and IPv6 CIDR blocks.

Subnets
Select the VPC subnets for Lambda to use to set up your VPC configuration.

Choose subnets

subnet-0062584e61eedc89b (10.0.0.0/20) us-east-1a ✕
Name: MinecraftLogWebServer-subnet-public1-us-east-1a

You must select between 1 and 16 Subnets.

⚠ We recommend that you choose at least 2 subnets for Lambda to run your functions in high availability mode.

Security groups
Choose the VPC security groups for Lambda to use to set up your VPC configuration. The table below shows the inbound and outbound rules for the security groups that you choose.

Choose security groups

sg-02d5e3bb03576a44a (default) ✕
default VPC security group

You must select between 1 and 5 Security groups.

Inbound rules | Outbound rules

Edit the following variables for your function:

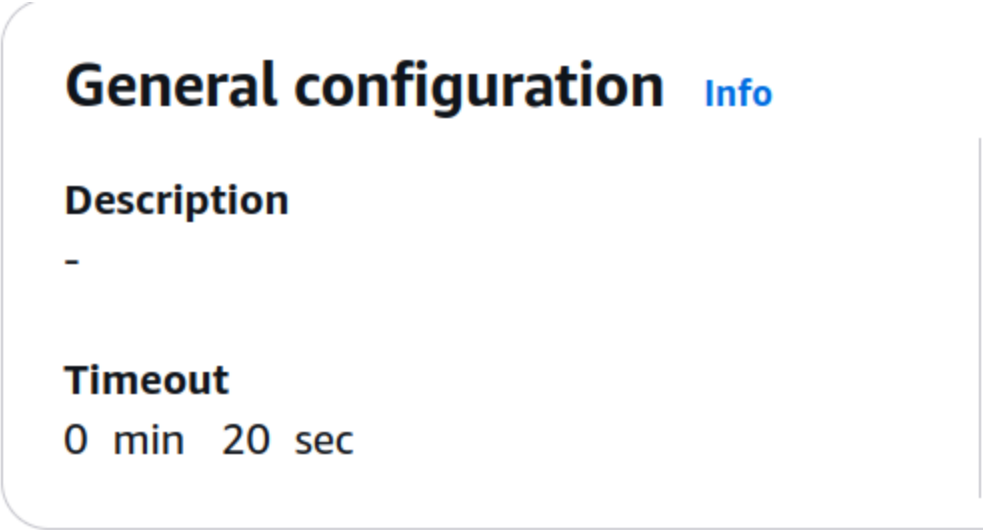
- TargetIP (Should be the internal DNS of the EC2 Instance)

```
import * as https from 'node:https';
import * as http from 'node:http';
import * as zlib from 'node:zlib';

/**
* Lambda function that receives CloudWatch events and forwards them to an
external API
*
* @param {Object} event - The CloudWatch event object
* @returns {Object} Response object with status code and body
*/
export const handler = async (event) => {
 try {
   console.log('Received event:', JSON.stringify(event, null, 2));

   // Configuration for the external API
   const targetIP = 'ip-10-0-1-67.ec2.internal';
   const targetPort = 80; // HTTP port
   const targetPath = '/log';
   const useHttps = false;

   // Extract and decode the CloudWatch Logs data
   if (event.awslogs && event.awslogs.data) {
     // Decode base64 and decompress
     const compressed = Buffer.from(event.awslogs.data, 'base64');
     const decompressed = zlib.gunzipSync(compressed).toString('utf-8');
     const logData = JSON.parse(decompressed);

     // Process log events
     if (logData.logEvents && logData.logEvents.length > 0) {
       for (const logEvent of logData.logEvents) {
         // Send each log message as plain text
         const logMessage = logEvent.message;

         await sendHttpRequest(
           targetIP,
           targetPort,
           targetPath,
           logMessage, // Send the raw log message as plain text
           useHttps
         );
       }
     }
   }
```

```
    // Return a success response
    return {
      statusCode: 200,
      body: JSON.stringify({
        message: 'Successfully forwarded CloudWatch logs'
      })
    };
  } catch (error) {
    console.error('Error processing CloudWatch event:', error);

    // Return an error response
    return {
      statusCode: 500,
      body: JSON.stringify({
        message: 'Error processing CloudWatch event',
        errorDetails: error.message
      })
    };
  }
};

/**
* Sends an HTTP request to the specified endpoint
*
* @param {string} host - The target host/IP
* @param {number} port - The target port
* @param {string} path - The target path
* @param {string} data - The data to send in the request body
* @param {boolean} useHttps - Whether to use HTTPS or HTTP
* @returns {Promise<Object>} The response data
*/
function sendHttpRequest(host, port, path, data, useHttps = false) {
  return new Promise((resolve, reject) => {
    // Configure the request options
    const options = {
      hostname: host,
      port: port,
      path: path,
      method: 'POST',
      headers: {
        'Content-Type': 'text/plain',
        'Content-Length': Buffer.byteLength(data)
      }
    };
```

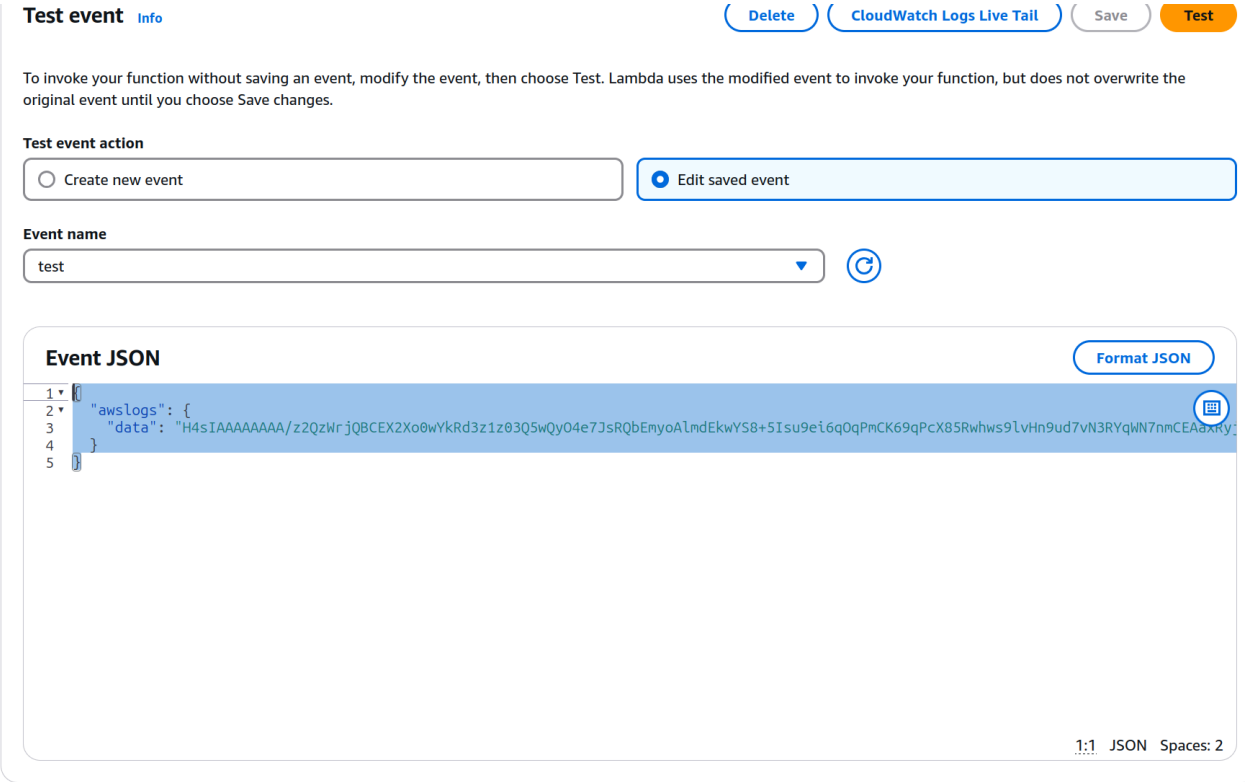
```
  // Choose the appropriate protocol (HTTP or HTTPS)
  const protocol = useHttps ? https : http;

  // Create and send the request
  const req = protocol.request(options, (res) => {
    let responseData = '';

    // Collect the response data
    res.on('data', (chunk) => {
      responseData += chunk;
    });

    // Resolve the promise when the response is complete
    res.on('end', () => {
      try {
        const parsedData = responseData ? JSON.parse(responseData) : {};
        resolve({
          statusCode: res.statusCode,
          headers: res.headers,
          body: parsedData
        });
      } catch (e) {
        resolve({
          statusCode: res.statusCode,
          headers: res.headers,
          body: responseData
        });
      }
    });
  });

  // Handle request errors
  req.on('error', (error) => {
    reject(error);
  });

  // Send the request data
  req.write(data);
  req.end();
 });
}
```

Change the timeout of the function to 20 seconds



Creating test cases for lambda

Write a test case for your lambda function and test it



Try this test code. This is what cloudwatch will send to the lambda function:

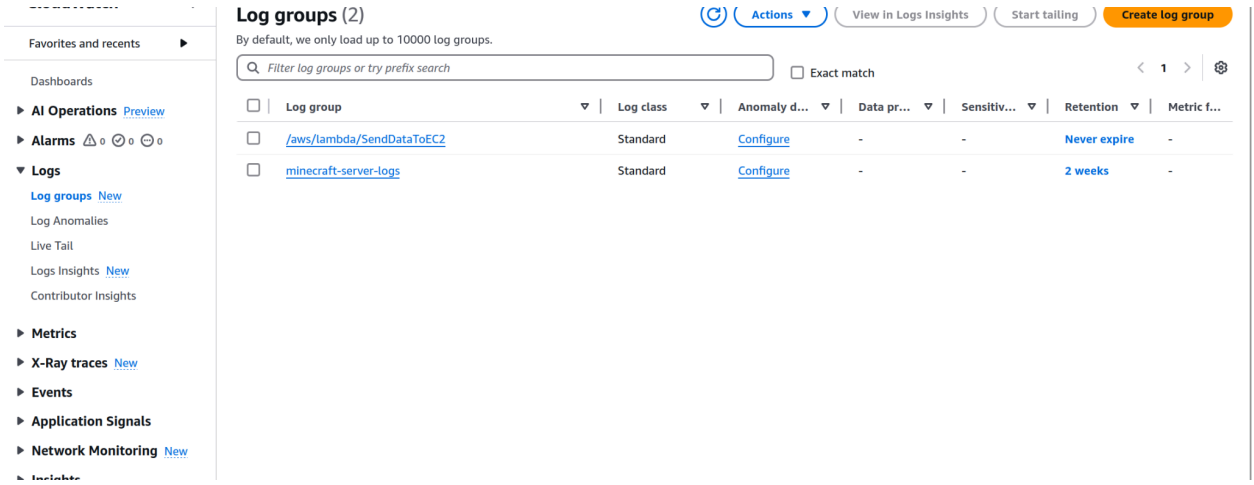
```
{
  "awslogs": {
      "data":
"H4sIAAAAAAAA/z2QzWrjQBCEX2Xo0wYkRd3z1z03Q5wQyO4e7JsRQbEmyoAlmdEkwYS8+5Isu9
ei6qOqPmCK69qPcX85Rwhws9lvHn9ud7vN3RYqWN7nmCEAaxRyjiyLhgpOy3iXl9czBJjSHI+5f
y71GvNbzPVpGde/ll3JsZ8gQKrbwZqoB+be2IgYTf0/BxWsr0/rMadzSct8m04l5hXCAR6WcYwZ
um/Y9i3O5Uv+gDRAAM1sRbR15AW1dtISkVjRBgVbdi2j11bQMXlyTE7Ie2QLFZQ0xbX00xkCekN
Wi/fCqKt/X0CAQ+uCkaClU4fd9zBVXnLsh+v7X7e/u6D2y7QcU7nQ4dpjQ8Y31FKDGCxZ4k6dvs
oPKs3qPZUXFeeSykWlQVnvtOqL+lEjNUQO0XmxGq2vFJumrVTdNiSO2HhmYbZWtL+Cz+7zDw/NN
WGtAQAA"
  }
}
```


Now Run it in the test

Sending Logs from cloudwatch to lambda

Going to log streams

Go to cloudwatch dashboard and go to log groups. Click on minecraft-server-logs

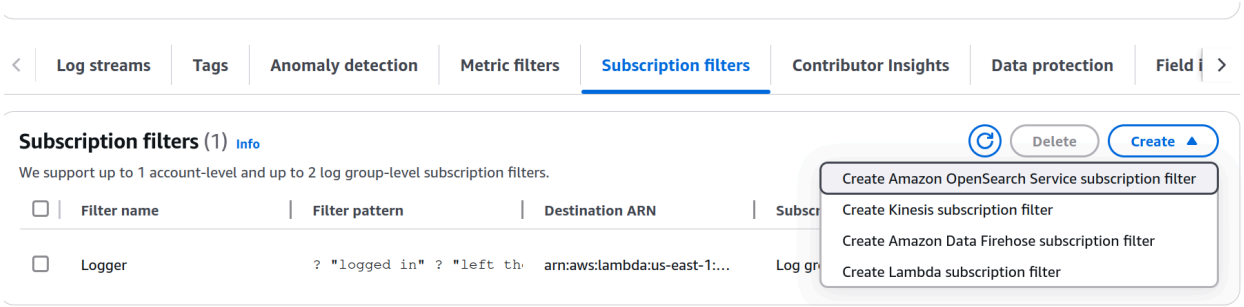


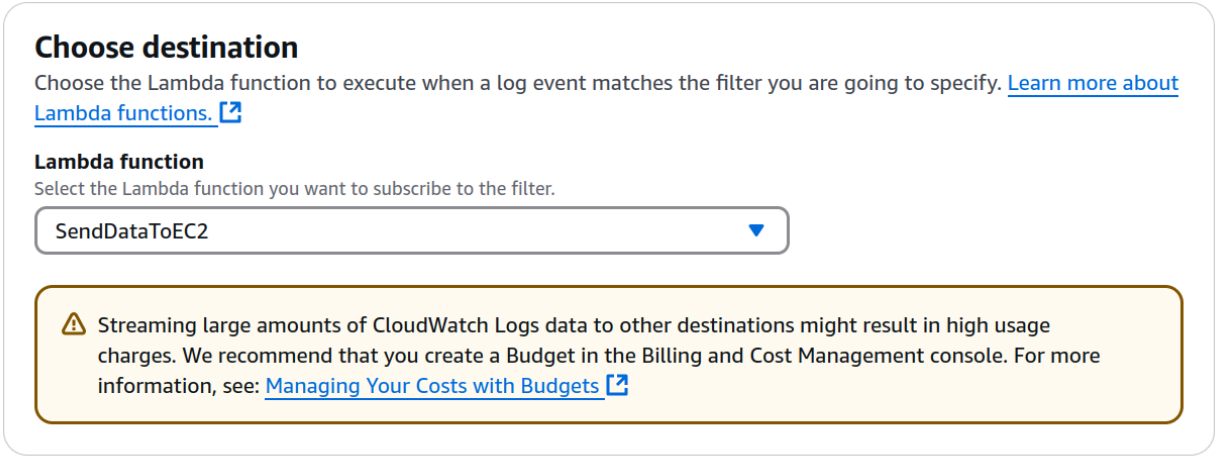
There should now be a logstream called {lightsail instance}-minecraft. Make sure its there.

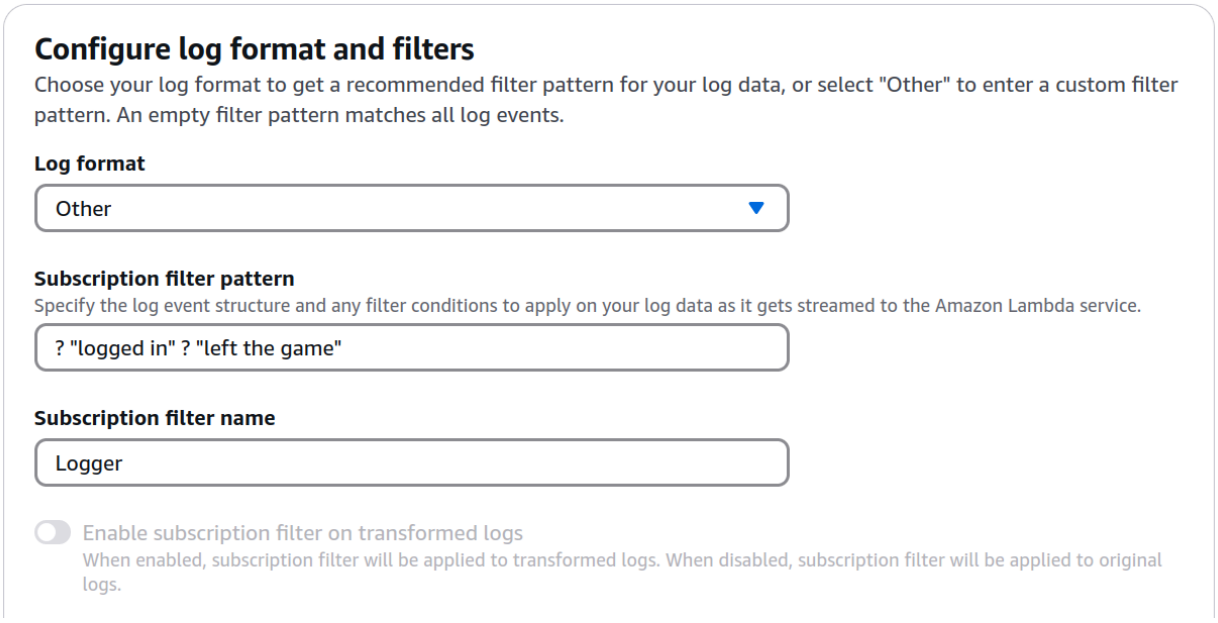
Creating a subscription filter

The subscription filter will send the data to the lambda function itself

Go to subscription filters and create a lambda filter







Test the pattern out with this

```
2025-03-22T06:15:36.977Z
[06:11:30] [User Authenticator #1/INFO]: UUID of player Tomocity2 is
787b9319-d89f-4498-b98e-d7557bc10ed6
2025-03-22T06:15:36.977Z
```

```
[06:11:30] [Server thread/INFO]: Tomocity2[/71.247.203.11:44402] logged in
with entity id 134 at (21.180475311874886, 75.0, 21.98323143604005)
2025-03-22T06:15:42.188Z
[06:11:30] [Server thread/INFO]: Tomocity2 joined the game
2025-03-22T06:15:50.264Z
[06:15:50] [Server thread/INFO]: Tomocity2 lost connection: Disconnected
2025-03-22T06:15:52.521Z
[06:15:50] [Server thread/INFO]: Tomocity2 left the game
2025-03-22T06:15:53.023Z
[06:15:52] [User Authenticator #2/INFO]: UUID of player Tomocity2 is
787b9319-d89f-4498-b98e-d7557bc10ed6
2025-03-22T06:15:53.023Z
[06:15:52] [Server thread/INFO]: Tomocity2[/71.247.203.11:55440] logged in
with entity id 804 at (7.638552196096684, 72.0, 48.305932660813625)
2025-03-22T06:15:57.977Z
[06:15:52] [Server thread/INFO]: Tomocity2 joined the game
```

start streaming

Start the python web server

Connect to your EC2 and do
python3 server.py

All done!

Time to try it out

Connect to the public EC2 url. Don’t hit open address, because it will use https. Instead copy the public dns and do http:///{dns} in your browser

Public IPv4 DNS

ec2-3-237-255-163.compute-1.amazonaws.com |

open address

Now try to connect to the server

If it works, it should show up in 3-10 seconds