Introduction

The Ether Cattle Initiative has produced a set of CloudFormation templates for those wishing to run a master / replica geth cluster on AWS. This document has three sections. Section 1 is a simple walkthrough of how to deploy and manage a cluster. Section 2 discusses the elements of the CloudFormation stack, and what happens under the hood. Section 3 covers general troubleshooting of the cluster.

This document assumes a basic level of familiarity with AWS, and general systems administration skills.

1. Setup

Prerequisites

Before you get started, you must have:

- An AWS Account
- A pre-created VPC where the resources can be deployed
- A pre-created subnet for deploying VMs
- An IAM user with administrative privileges to deploy the CloudFormation stack
- Check out the Git repository of CloudFormation templates from: https://github.com/openrelayxyz/ethercattle-deployment

Generating Your First Chaindata Snapshot

Before you can stand up a cluster, you need a snapshot of the Ethereum blockchain data. Your masters and replicas will both be created from the same initial snapshot. Creating this initial snapshot takes 36-48 hours at the time of this writing, and that is likely to increase as the blockchain continues to grow.

The snapshot_generator.yaml CloudFormation template will generate this initial snapshot with minimal effort on your part.

1. Navigate to the CloudFormation Dashboard
2. Click “Create Stack”
3. Select “Template is Ready” and “Upload a Template File”
4. Pick “Choose File” and choose “snapshot_generator.yaml” from your Git checkout
5. Click Next
6. Enter a stack name. We recommend “ethercattle-snapshot”
7. Fill in the following parameters:

- **Instance Type**: The type of EC2 instance to use to sync your Ethereum node. The least expensive node we have found that will successfully sync mainnet is the default, **m5a.large**. If you are syncing a testnet, you may be able to use a smaller instance. If you want the syncing process to go faster, an instance type with more CPUs may help.
- **Disk Size**: If you are syncing the Ethereum Mainnet, you will need at least 250 GB. If you are syncing a testnet, such as Goerli, you can lower this.
- **SSH Key Pair**: This is the name of an SSH Key Pair you can use to SSH into the instances created by this stack. If you need help setting up an SSH key pair, see: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html
- **S3GethBucketName**: The S3 bucket containing the necessary Geth binaries and Lambda Functions for this stack. The default is a publicly shared bucket managed by the OpenRelay team, but if you wish to host your own you may point to it here.
- **Chaindata Snapshot ID**: If you already have a chaindata snapshot and wish to bring it up to date, you can enter the snapshot id here. In most cases, this field will be left blank, and the snapshot will sync from an empty disk.
- **Geth Flags**: These are command-line flags for the geth service that will create your snapshot. If you are creating a mainnet snapshot, nothing is needed in this field. If you want to create a snapshot for a testnet, you can add the appropriate flags here.
- **VPC ID**: The VPC ID of the VPC you want to deploy to.
- **VPC Base IP**: The first two octets of your VPC's IP range. For example, if the subnet you indicated earlier has an IP range of 10.42.19.0, this field would be "10.42"
- **Subnet**: The subnet ID for a valid subnet on your VPC. This should be a public subnet that assigns public IPs to instances (necessary for connecting to peers).
- **Notification Email**: If you wish to be notified when the snapshot is complete, provide an email address here and the stack will subscribe you to an SNS topic to receive notifications when the snapshot is complete. If you do not provide an email address, no notifications will be sent. If you do provide an email address, you will receive a verification email, which you must confirm before you will receive notifications.

8. Once the above form fields are completed, click "Next"
9. On the Stack Options page, you can add tags, stack policies, or notifications. If you don't know what this means, you can just click "Next"
10. The Stack Review page will show you the options you just filled out. At the bottom of this page is a "Capabilities" section. You will need to check the box that says "**I acknowledge that AWS CloudFormation might create IAM resources.**" (The IAM resources will enable an EC2 instance created by the stack to take a snapshot).
11. Click "Create Stack"

12. This will launch an EC2 instance, which will connect to peers and download the blockchain. It will take several days to get in sync with the blockchain - this is simply a reality of running a full node, and not specific to Ether Cattle. When it is complete, it will create a snapshot of its disk and shutdown. To monitor progress, you can SSH to the EC2 instance and run:
    ```
    tail -f /var/log/cloud-init-output.log
    ```
    Which will show geth's progress.
13. When it is complete, the EC2 > Volumes > Snapshots dashboard will have the snapshot created by this instance. It will have a tag "clusterid = snapshotter" and the name "Snapshotter-chaindata-$timestamp", where $timestamp indicates when the snapshot completed. You will use the Snapshot ID for this snapshot in later steps.
14. You may now delete the CloudFormation Stack you created. It will leave the snapshot intact, and the other resources are no longer needed.

Deploying Cluster Infrastructure

Ether Cattle uses Kafka for communicating between master and replica nodes, and uses an Application Load Balancer to pool replica nodes together. If you intend to run multiple Ether Cattle clusters for high availability, you can use the same Kafka cluster and load balancer for multiple Ether Cattle clusters. To simplify this we have separated out an Infrastructure Stack, which provides Kafka and a Load Balancer, and a Replica Cluster stack, which provides the master and replica nodes.

WARNING

While the Infrastructure Stack's Kafka cluster is redundant and spans three availability zones, it is not especially well structured for maintenance and updates. If you have true high-availability requirements for your cluster, we recommend either developing in-house expertise in running Kafka, or outsourcing to a managed Kafka provider such as AWS MSK, Confluent, or Eventador.

To deploy the infrastructure stack:

1. Navigate to the CloudFormation Dashboard
2. Click "Create Stack"
3. Select "Template is Ready" and "Upload a Template File"
4. Pick "Choose File" and choose "infrastructure.yaml" from your Git checkout
5. Click Next
6. Enter a stack name. We recommend "ethercattle-infra"
7. Fill in the following parameters
    - **Cluster ID**: An identifier for this infrastructure cluster. If you deploy multiple infrastructure stacks, this must be unique to avoid resource naming conflicts.

- **SSH Key Pair**: This is the name of an SSH Key Pair you can use to SSH into the instances created by this stack. If you need help setting up an SSH key pair, see: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html
- **Kafka Instance Type**: The type of EC2 instance to run the Kafka nodes on. The Ether Cattle use case is not especially taxing on Kafka, and t3.small instances are suitable for a small number of clusters.
- **Kafka Count**: The number of Kafka servers to run. This will generally be 3.
- **Kafka Disk Size**: The amount of disk to provision for use in Kafka. We recommend 500 GB.
- **Kafka Disk Type**: The type of EBS volume to use for Kafka. We recommend st1, which is high-throughput HDD volumes. Note that because Kafka is optimized for sequential reads and writes, there is little to gain from using SSDs in Kafka.
- **VPC ID**: The VPC ID of the subnet you wish to deploy to. Note that the CloudFormation stack will deploy a public subnet in each availability zone, and set up routing tables between them.
- **Internet Gateway ID**: The ID of an internet gateway on the VPC you specified.
- **VPC Base IP**: The first two octets of your VPC's IP range. For example, if the VPC you indicated earlier has an IP range of 10.42.0.0, this field would be "10.42"
- **Unused IP Octet (3)**: The CloudFormation template will deploy three public subnets on your VPC (in three different availability zones). If you specified a VPC Base IP of "10.42", and "160" for an IP octet, you would get a subnet covering "10.42.160.0/24". You need to provide three octets available for creating /24 subnets on the specified VPC.

8. Once the above form fields are completed, click "Next"
9. On the Stack Options page, you can add tags, stack policies, or notifications. If you don't know what this means, you can just click "Next"
10. The Stack Review page will show you the options you just filled out. At the bottom of this page is a "Capabilities" section. You will need to check the box that says "**I acknowledge that AWS CloudFormation might create IAM resources.**" (The IAM resources will enable nodes to log to CloudWatch, and interact with the Elastic Container Service).
11. Click "Create Stack"
12. This will launch EC2 instances, join them to an ECS cluster, and deploy both Apache ZooKeeper and Kafka to the cluster. It will also create an internal Application Load Balancer, which, initially, will have no instances connected to it.
13. Once the status of your CloudFormation Stack shows as "Create Complete", you are ready to deploy a Replica Cluster against this stack.

Deploying A Cluster

Now that you have the necessary infrastructure to support a cluster, and you have a chaindata snapshot to launch your master and replicas from, you are ready to launch your first replica cluster.

To deploy the infrastructure stack:

1. Navigate to the CloudFormation Dashboard
2. Click “Create Stack”
3. Select “Template is Ready” and “Upload a Template File”
4. Pick “Choose File” and choose “replica_cluster.yaml” from your Git checkout
5. Click Next
6. Enter a stack name. We recommend “ethercattle-c1”
7. Fill in the following parameters:
    - **Infrastructure Cloudformation Stack**: The CloudFormation stack name you specified in step 6 of the previous section (we suggested “ethercattle-infra”). This will determine the subnets instances get deployed to, the Kafka brokers to connect to, the load balancer to register with, etc.
    - **External Target Group**: If you want replicas to be available under a different load balancer than the one from the infrastructure stack you deployed in the previous section, you can specify the ARN of a target group you want replicas to join here. Otherwise, leave this blank.
    - **Use Standard Target Group**: If you specified a value in the **External Target Group** field, you may not want replicas in this cluster to connect to the standard target group associated with the infrastructure stack. For example, if you wanted to run a Goerli cluster on the same Kafka brokers as a mainnet cluster, you could specify an alternative load balancer above, and set this to 0 to avoid having replicas from this stack join the mainnet loadbalancer.
    - **SNS Topic For Alarms**: If you already have an SNS topic where you wish to be notified of issues, you can specify it here, and any alarms will be sent to that topic. If you do not provide an SNS topic, no notifications will be sent. You might, for example, use this to send alarm notifications to Slack or other chat providers (see https://medium.com/cohealo-engineering/how-set-up-a-slack-channel-to-be-an-aws-sns-subscriber-63b4d57ad3ea**).**
    - **Notification Email Address**: If you wish to be notified of issues via email, provide an email address here and we will set up a subscription to the topic alarms are broadcast to. If you do not provide an email address, no notifications will be sent. If you do provide an email address, you will receive a verification email, which you must confirm before you will receive notifications.

- **SSH Key Pair**: This is the name of an SSH Key Pair you can use to SSH into the instances created by this stack. If you need help setting up an SSH key pair, see: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html
- **Unique Kafka Topic Name**: A name for the Kafka topic for this cluster. This **must** be unique across the clusters running on the same infrastructure stack, and **should** be unique globally.
- **Unique Network ID**: An identifier for the network this cluster represents. You can choose this identifier, but *should* use the same identifier for all clusters connecting to the same network.
- **S3 Geth Bucket**: The S3 bucket containing the necessary Geth binaries and Lambda Functions for this stack. The default is a publicly shared bucket managed by the OpenRelay team, but if you wish to host your own you may point to it here.
- **Chaindata Snapshot ID**: The snapshot id of the snapshot created by the first stack. If you have run other clusters, you could use a recent snapshot id generated by another cluster.
- **Master Instance Type**: The EC2 instance type the master will run on. The least expensive instance type we have found that will run a master reliably is an m5a.large instance, so this is the default.
- **Master Extra Geth Flags**: If you want to add extra flags to the master's Geth process, add them here. Most commonly, this would be used if you wanted to run on a network other than mainnet. In most cases, this should be left blank.
- **Disk Size**: The amount of disk to provision for the chaindata folder on replica and master nodes. This must be greater than or equal to the size of the chaindata snapshot. For mainnet, this must be at least 250 GB, but for testnets or private chains it may be smaller.
- **Replica Extra Geth Flags:** If you want to add extra flags to the replicar's Geth process, add them here. Most commonly, this would be used if you wanted to run on a network other than mainnet. In most cases, this should be left blank.
- **Replica Disk Type**: The type of disk to be used on replica servers. By default this is st1. If you anticipate that your replica will be under heavy load, you may want to change this to gp2.
- **Replica Target Capacity**: The number of replica instances to run.
- **Replica On Demand Percentage**: The percentage of replica instances that should be On Demand instances, as opposed to Spot Instances. This can be anywhere from 100 (for all on-demand instances) to 0 (for all spot instances). Spot instances are likely to be terminated abruptly, but are much cheaper. Because replicas can be replaced fairly quickly, it is fairly safe to use spot instances for this use case.

8. Once the above form fields are completed, click "Next"
9. On the Stack Options page, you can add tags, stack policies, or notifications. If you don't know what this means, you can just click "Next"

10. The Stack Review page will show you the options you just filled out. At the bottom of this page is a “Capabilities” section. You will need to check the box that says “**I acknowledge that AWS CloudFormation might create IAM resources.**” (The IAM resources will enable a wide range of tasks, from launching instances to creating and cleaning up snapshots.).
11. Click “Create Stack”
12. This will launch master and replica servers in autoscaling groups, add a variety of metrics collection and alarms, set up daily tasks to take snapshots for replica servers, as well as recurring tasks to clean up those snapshots. The full depth of this stack will be discussed in Section 2.
13. Once the CloudFormation stack shows a status of “Create Complete”, you should have a master server and replicas running. The replicas will not join the load balancer until the master is caught up with the Ethereum network - how long that takes will depend on the age of the snapshot you launched from.

High Availability

A single Ether Cattle cluster provides several operational benefits over running conventional Ethereum nodes, but the master server is still a single point of failure. Using data stored in Kafka, the master can recover much more quickly than a node that needed to sync from peers, but that can still lead to a period of time where the replicas are serving stale data.

To achieve high availability requires multiple clusters with independent masters and their own replicas. Multiple replica clusters can share a high-availability Kafka cluster. The following formula can be used to determine the statistical availability of a cluster:

$$a = 1 - \left(1 - \frac{mtbf}{mttr + mtbf}\right)^N$$

Where:

- *mtbf* - Mean Time Between Failures - The average amount of time between failures of a master server
- *mttr* - Mean Time To Recovery - The average amount of time it takes to replace a master server after a failure
- *N* - The number of independently operating clusters

The values of *mtbf* and *mttr* will depend on your operational environment. With our AWS CloudFormation templates, we have established an *mttr* of 45 minutes when snapshotting daily. We have not gathered enough data to establish a *mtbf*, but with two independent clusters and a 45 minute *mttr*, EC2’s regional SLA becomes the bounding factor of availability if the *mtbf* is greater than two weeks.

This formula focuses only on the availability of masters - it assumes that each master has multiple independent replicas. If a master only has a single replica, that will hurt the *mtbf* of the cluster as a whole.

Deleting a Cluster

If you are done with a cluster, you can delete it. This will remove all resources created by the CloudFormation stack, including autoscaling groups (and corresponding instances and attached volumes), security groups, IAM roles & policies, lambda functions, log groups, alarms, and dashboards. It **will not** remove any EBS Snapshots, as these are not tracked by the CloudFormation stack. You will want to delete those manually. Additionally, CloudWatch metrics will continue to be available according to their retention period.

Additionally, because of dependencies between your infrastructure stack and cluster stacks, you need to delete all clusters based on an infrastructure stack before you will be able to delete the infrastructure stack itself.

Upgrade Process

We recommend against upgrading individual clusters. Software updates may change the on-disk format or the log message format, and having inconsistencies between the master and replicas could cause serious problems.

Instead, we recommend standing up a new cluster when you wish to upgrade.

1. Note the snapshot id from your existing cluster.
2. Go through the steps of Deploying a Cluster, using the snapshot id from the cluster you are replacing.
3. Wait for the new replicas to be in sync with the network and connected to the load balancer.
4. Delete the old cluster's CloudFormation stack as described in Deleting a Cluster.

This will ensure zero-downtime upgrades, without any issues synchronizing updates between the master, replicas, and the snapshotting process.

Expected Monthly Costs

The table below shows expected monthly costs for a single Ether Cattle Cluster deployed on an infrastructure stack. Items with check marks in the "Scale" column can be expected to increase roughly linearly with each Ether Cattle Cluster you add, while other items should be able to support multiple clusters on the cost indicated below.

Item	**Quantit**	**Cost Per Unit**	**Total**	**Note**	**Scale**

	y	(monthly)			
t3.small	3	$15	$45	Kafka Hosts	
m5a.large	1	$62	$62	Master Server	✔
t3.micro	0.125	$7.50	$1	Snapshot Servers (about 3 hours daily)	✔
Misc. Spot	2	$15	$30	Replica Hosts	✔
st1 Storage	1500 GB	$.045	$67.50	Kafka Storage	
gp2 storage	900 GB	$.10	$90	Host root volumes, chaindata storage	✔
snapshots	1 TB	$.05	$50	Snapshots of chaindata volumes	✔
Cloudwatch Metrics	60	$.30	$18		✔
Cloudwatch Alarms	5	$0.10	$.50		✔
Cloudwatch Dashboards	2	$3	$6	1 Infrastructure, 1 cluster	✔
Cloudwatch Logs	15 GB	$.50	$7.50		✔
Application Load Balancer	1	$33	$33		
Regional Data Transfer	2.5 TB	$.01	$25	Inter-AZ Communication between Kafka brokers and Cluster	✔
Transfer out to Internet	125 GB	$.09	$11.25	Master syncing with network	✔

The total cost for this configuration is around $450 per month. Note that you could get savings by making instance reservations for 1 m5a.large and 3 t3.small servers, which will be running on a regular basis. Note that your first month may be slightly higher, due to the costs of making an initial snapshot. These costs assume fairly minimal traffic to replicas - costs for load balancers, regional data transfer, and logging will increase for high volume clusters.

2. Under The Hood

The CloudFormation templates for an Ether Cattle Cluster have a lot of moving parts. This section of the document is intended to give you a good understanding of what is involved in an operating cluster.

The Master

The Ether Cattle master is mostly a conventional Geth node, but uses Kafka to keep a log of everything it writes to its underlying database. That Kafka log will be used by replicas to be able to serve the same information as the master.

The master is run through an autoscaling group, though that auto-scaling group will only ever run one master at a time. In the event that a master fails, it can be terminated, and the autoscaler will replace the instance automatically. The master’s chaindata folder will start from the Snapshot ID provided in the CloudFormation template.

On startup, the master first attempts to sync from the Kafka topic it will eventually write to. This ensures that it is starting from the same place its replicas are starting before connecting to peers. On the first startup for a cluster there will be nothing available from Kafka, but on subsequent runs it may take a few minutes for the master to sync from Kafka before it starts syncing with peers.

If you terminate an existing master and it must resume from a snapshot that is 24 hours old, it typically takes about 45 minutes to sync with Kafka and then catch up from peers on the network. By comparison, a traditional Geth node would take around 3.5 hours to catch up with network syncing only from peers.

The Ether Cattle CloudFormation template has an optimization to improve startup time. When starting a new EC2 instance with a volume derived from a snapshot, there is a period of high read latency for the snapshotted volume. Without this optimization, it would take several hours before the EC2 volume’s read latency was low enough to keep up with the Ethereum blockchain. When a new instance starts up, it is created with a Provisioned IOPS disk, giving it much better read performance. After the master finishes syncing from Kafka, that volume is modified from a Provisioned IOPS disk to a standard SSD volume (gp2). The provisioned IOPS cost a little bit extra to get the master up quickly, but once the volume modification is complete we see no issues with the master keeping up with the blockchain.

The main master process runs with geth’s “--gcmode=archive” flag enabled. By default, Geth keeps much of the state trie in memory, flushing it to disk every few minutes. Since only information written to disk gets sent to replicas, we must have Geth write to disk on a continuous basis to make sure replicas have current information. This means that the disk utilization will grow at a faster pace than might be the case on a standard Geth node (around 25 GB per week of growth).

Replicas

Replicas run a variant of the standard Geth node that does not rely on peer-to-peer communication, and serves everything directly from disk. They pull the master’s write log from Kafka, and write that information to their local database. Once in sync with the blockchain, replicas will start serving RPC requests on port 8545.

Like masters, replicas are also started from an autoscaling group, and also start from the snapshot id provided to the CloudFormation template. At the time of this writing, the autoscaler will run the number of replicas indicated by the “Replica Target Capacity” parameter of the template, but this could be configured to autoscale based on a variety of metrics.

In the CloudFormation configuration, replicas will only start serving RPC requests when the following conditions are met:

- The replica must be in sync with Kafka
- The most recent block received from the master must be less than 45 seconds old

Additionally, if either of the following conditions are met, the replica will shutdown:

- The replica has not received a message from the master in over 62 seconds (the master has a 30 second heartbeat, so this means it missed at least 2 heartbeats).
- The latest block from the master is over 4 minutes old.

In those situations, systemd will restart the replica, and it will resume serving RPC requests once in sync with Kafka and having a block less than 45 seconds old.

Regular Snapshots

A critical piece of running an Ether Cattle cluster is having frequent snapshots for starting new instances. This allows you to scale up the number of replicas to increase capacity, and replace failed masters and replicas.
The Kafka server, by default, has a 7 day retention period for write logs. When starting a new master or replica, it is critical that the chaindata snapshot comes from within that 7 day retention period, or it will not be possible for the server to sync up with Kafka. Thus, we need to make sure that we have snapshots more recent than Kafka’s retention period. The CloudFormation template includes a snapshotting process that runs once daily to ensure snapshots are available.

- A CloudWatch Event Rule is scheduled to execute a lambda function once daily. Its first run will be 24 hours after you deploy the CloudFormation stack.
- The lambda function will launch a t3.micro EC2 instance with a chaindata volume derived from the CloudFormation stack's latest snapshot.
- The EC2 instance will pull the latest data from Kafka, then the replica process terminates.
- The EC2 instance creates a snapshot from the volume it just synced with Kafka. This snapshot will have a “clusterid” tag that corresponds to the Kafka topic you indicated in the CloudFormation stack, and a “name” tag of “$clusterid-chaindata-$timestamp”.
- Once the snapshot is 100% complete, the EC2 instance will update the CloudFormation template parameter with the new snapshot. This means any new masters or replicas started by their respective autoscalers will launch with this version of the snapshot.
- The EC2 instance terminates.

This process can take a couple of hours, but runs behind the scenes. As an administrator, you generally don't need to worry about the process, so much as making sure that the snapshots are created and the CloudFormation stack is updated on a regular basis.

In addition to the daily process that takes snapshots, every hour a Lambda function executes to clean up older snapshots. By default, it will keep the four most recent completed snapshots, and delete anything older.

Monitoring

The CloudFormation stack sets up several CloudWatch metrics, as well as the necessary infrastructure to populate those metrics. General system metrics are collected by the AWS CloudWatch agent, which is installed on each machine. Application-specific metrics are logged by the application, sent to CloudWatch logs via the journald-cloudwatch-logs daemon, sent to a Lambda function using a CloudWatch subscription filter, and the Lambda function parses the log messages to create CloudWatch Metrics.

Several of these metrics have alarms associated with them. Each alarm is sent to two or three SNS topic:

- **Aggregated Notifications**: An SNS topic that receives all alarm notifications. If you provided a **Notification Email** address parameter, it will be subscribed to this channel. You can create additional subscriptions to this channel.
- **Alarm SNS Topic**: If you provided an **Alarm SNS Topic** parameter, all alarm triggers will be sent to that channel.
- **Alarm-Specific Topics**: Each alarm has a corresponding SNS topic, with no subscribers. These are created for your convenience, and you can add your own subscriptions to them, but as of now they are unused.

The metrics collected by the CloudFormation stack are:

Master

- **Disk Utilization**: Monitoring disk utilization on the master is essential. Both masters and replicas will increase in disk utilization over time, and will eventually need to be increased. We have alarms at 95% disk utilization, which should be a couple days warning based on typical growth patterns. As the Master and all Replicas use disk at effectively the same rate, all generally need to be updated at the same time.
- **Memory**: Memory usage on the master should be fairly constant. If you have 8 GB of RAM and the configuration described above, the master should stay consistently below 75% RAM utilization. We have alarms set at this threshold, and if it stays in an alarming state for an extended period, we recommend rebooting your master server.
- **CPU**: CPU utilization on the master will be a bit spikey - when new blocks come in it will ramp up to validate the block, then settle back down. With two modern CPUs, the CPU utilization will be around 60% averaged out over time. We have an alarm set at 85%, and recommend rebooting the master if it stays above 85% for an extended period.
- **Peer Count**: A critical metric for master health is its number of peers. If a master has no peers, it will not receive new block information. By default, a master will establish connections with 25 peers. We have an alarm configured to trigger if the master is below 10 peers. If a master drops to zero peers and stops syncing blocks, we recommend restarting the master.
- **Block Age**: Every time it processes a block (or group of block, during the initial sync) geth logs a message about the processed blocks. If the processed block is older than 60 seconds, Geth will include the age of that block. We track the block age as a metric.
- **Block Number**: Because block age metrics are only available when blocks are greater than 60 seconds old, we also track how frequently new blocks are reported. If no new blocks have been received in 120 seconds, follow the same process you would if you were seeing high block ages reported.

Replicas

- **Disk Utilization**: Both masters and replicas will increase in disk utilization over time, and will eventually need to be increased. We have alarms at 95% disk utilization, which should be a couple days warning based on typical growth patterns. As the Master and all Replicas use disk at effectively the same rate, all generally need to be updated at the same time.
- **Memory**: Replica RAM utilization depends primarily on the volume of RPC traffic the replica is serving. A replica that is serving only a handful of requests will use less than 1 GB of RAM. A replica under very heavy load will trend up with the load. We have alarms at 80% RAM utilization, and recommend scaling out your cluster of replicas to help share the RPC traffic (at the time of this writing, we do not have autoscaling triggers from this alarm).

- **CPU**: Replica CPU utilization depends primarily on the volume of RPC traffic the replica is serving. A replica that is serving only a handful of requests will have a nearly idle CPU. A replica under heavy load will trend up with the load. We have an alarm at 75% CPU utilization and recommend scaling out your cluster of replicas to help share the RPC traffic (at the time of this writing, we do not have autoscaling triggers from this alarm).
- **Block Number**: Every 30 seconds, the replica will log a message with several pieces of information, including the latest block number. If this number does not increase regularly, there is likely a problem with either the master, or communication between the master and replica.
- **Block Age**: Every 30 seconds, the replica will log a message with several pieces of information, including the latest block age. If the block age exceeds `--replica.block.age` the replica will shutdown, so you may want to monitor and be alerted before that happens.
- **Block Lag**: Block lag is a computed metric found by comparing

  ```
  master.blockNumber - replica.blockNumber
  ```

  If this number is higher than 2, that might indicate that the replica is not receiving information from the master. If it stays high or increases steadily, check that Kafka is functioning properly, and try restarting the replica.
- **Replica Offset Age**: Every 30 seconds, the replica will log a message with several pieces of information, including how long it has been since it last got a message from the master. If this number exceeds `--replica.offset.age` the replica will shutdown, so you may want to monitor and be alerted before that happens. Generally this happens if either the master has crashed or Kafka has become unavailable.

Kafka

- **Disk Utilization**: A single Ether Cattle topic with snappy compression enabled and a default 7 day retention period will use about 40 GB of RAM on an ongoing basis. If you are running multiple Ether Cattle clusters on the same Kafka cluster, this will scale accordingly. We recommend leaving plenty of buffer between your expected usage and your disk limits, and monitoring closely to ensure you don’t run out of disk.
- **Memory**: Kafka memory utilization is dependent on the volume of data in Kafka topics and the number of active subscribers. With two Ether Cattle clusters, each with two subscribers, we see around 1.5 GB of RAM utilization. We have an alarm on memory utilization around 85%, and if that threshold is exceeded for an extended period, increase the RAM for your Kafka servers.
- **CPU**: The CPU utilization for Kafka in an Ether Cattle cluster is exceptionally low. We recommend tracking it and scaling to larger instances if it is consistently high, but this is unlikely to be a bottleneck for your cluster.

3. Troubleshooting

This section explores the variety of different problem scenarios you may encounter.

Getting an Initial Snapshot is Taking Forever

Slow initial syncs are an unfortunate fact of life with Geth. See this discussion for more details: https://github.com/ethereum/go-ethereum/issues/15001#issuecomment-426918953. In general, it will get there eventually, but it takes a long time. If you think it may not be running, SSH to the snapshot instance and `tail -f /var/log/cloud-init-output.log` to see what is happening.

An Alarm is Going Off

When you configured your cluster, you should have set up notifications, either via E-Mail or a pre-defined SNS topic so that the alarms will be brought to your attention. In these cases, we recommend that you refer to the monitoring section for the specific metric that is alarming, to find recommendations for how to handle that alarm.

The Disk Is Full

As noted previously, after creating an initial snapshot, you can expect your cluster's disk utilization to increase by about 25 GB per week. Eventually, you will run out of provisioned disk. In this case, you have a few options:

- Deploy a new cluster. In the **Disk Size** parameter, put in a bigger number. This will stand up a new cluster with bigger disks. Once the new cluster's replicas have joined the load balancer, you can delete the old cluster and you have more disk available.
- If you want to avoid deploying a new cluster, you can modify your existing cluster. First, update the CloudFormation stack providing a larger number for the disk size. This will modify your autoscaling groups so that new instances will have disks that size, but it will not replace the ones that are already deployed. In the EC2 Dashboard in the Volumes tab, you can identify the volumes associated with a stack by their Name Tags; they will be tagged as KafkaTopic-Master and KafkaTopicReplica, given the value of KafkaTopic that you used for this stack. Find the volumes for your cluster, click "Actions > Modify Volume" and increase the volume size. At this point, you will need to SSH into each instance and run `resize2fs /dev/nvme1n1` to resize the filesystem to match the volume.

Eventually though, the size of these disks may seem like it's getting out of hand. The snapshot volumes that started out around 250 GB will be 1 TB after about 7 months. When they exceed your threshold for acceptable volume sizes, we recommend starting back at the “Generating your first Chaindata Snapshot” process, and deploying a new cluster from the generated snapshot.

Responses are Slow

First, check CPU and RAM. If your instances are at the limits of CPU and RAM, the best answer is probably to add more instances. You can do this by updating your CloudFormation stack and increasing the ReplicaTargetCapacity parameter. If the CPU is mostly idle and RAM usage is not heavy, read on.

Replicas serve responses out of their on-disk LevelDB database. This makes them sensitive to the performance of the underlying disks. If your are encountering performance issues, you might try:

- Switching from st1 volumes to gp2 volumes
- If you are already on gp2 volumes, make them bigger. Bigger volumes on AWS have higher throughput, even if you don’t need the storage space.
- Switch to using instance types with more RAM. The Linux Kernel will cache as much of the database as it can in RAM, giving better general performance.

Replicas Are Behind the Network

In general, your replicas should reflect the state of your master within a few dozen milliseconds. If your master is not able to keep up with the network, update your replica cluster stack updating the **MasterInstanceType** parameter to a bigger instance type. This will update the autoscaling group for the master, but will not replace the master instance. You can force it to update the instance by terminating the master instance.

The Load Balancer is Serving 502 Responses

If your replicas lag too far behind the network, they will eventually shut down and wait for the master to catch up. If your master is not running properly, terminate the instance and let the autoscaler replace it. We have seen that periodically masters will lose many of their peers simultaneously - this seems to be a part of operating on a peer-to-peer network. You can help mitigate this by adding `--maxpeers=50` to the **Master Extra Flags** parameter in your CloudFormation stack, but handling extra peers may also require you increase the size of your master.

We also recommend running at least two replica clusters if you have high availability requirements. The replicas from both clusters will join the same load balancer, so if one master loses peers and cannot keep up with the network, requests will be routed to replicas from the other load balancer. This also helps mitigate risks relating to infrastructure issues within AWS.

My Question isn’t Answered Here

We’re sorry! We’ve tried to head off the issues we’re aware of, but if you’ve run into something not covered here, please visit us on Gitter here: https://gitter.im/ethercattle-initiative/community and we will try to help get you situated.