READ MODE · ANSWER FIRST

ClickHouse Realtime Ingestion Workers

An interactive, visual explanation of ClickHouse Realtime Ingestion Workers, from setup through the complete system.

CHEAT SHEET · 6 ESSENTIAL IDEAS

The whole story in 6 lines

ClickHouse Realtime Ingestion Workers becomes easier to reason about when every stage is connected as one system.

1. Partitions let many workers ingest in parallel while each lane still keeps its own order.
2. Batching trades a tiny wait for much better insert efficiency.
3. Workers protect ClickHouse by separating clean column blocks from bad rows before insertion.
4. An insert is only safe to acknowledge after ClickHouse has accepted the block according to the chosen durability path.
5. Healthy ingestion creates parts at a rate ClickHouse can merge without drowning the table.
6. Safe ingestion commits offsets only after success and slows the source when ClickHouse needs room.

STAGE 1 OF 8

Setup

Welcome. We are going to follow one stream of events as ingestion workers move it into ClickHouse. These workers are like a loading crew at a warehouse. They keep trucks moving, but they also wait for the warehouse to confirm each safe handoff.

An event is one incoming fact, like a page view, payment, sensor reading, or log line. A topic partition is an ordered lane of those events. Workers can split lanes across the pool, but inside one lane the order stays stable.

A worker is the process that does the repeated work: claim a lane, read some events, shape the rows and send an insert. A batch is the group of rows it collects before sending. Without batching, every tiny event would become its own expensive insert request.

An insert block is the column shaped chunk ClickHouse receives. Instead of one row at a time, ClickHouse sees vectors of values. After the block lands, a MergeTree table writes a physical piece on disk called a part.

Backpressure is the slow down signal. If ClickHouse is busy merging parts, if the queue is growing, or if inserts are timing out, workers should poll less aggressively. That protects the database from being flooded by more work than it can finish.

Over the next six stages we will build the whole loop: source lanes, batch building, parsing, insertion, part creation and recovery. Each stage answers one question about how the handoff stays fast without losing track of source offsets. Now let us start with the source lanes.

STAGE 2 OF 8

Source Lanes

02 SOURCE LANES

We begin at the stream source. Events do not arrive as one giant global line. They are divided into ordered partitions, which gives the ingestion system several lanes to read from at the same time.

Workers claim partitions from the group. Each worker becomes responsible for one or more lanes, like checkout clerks each taking a register. Try the Partitions control and watch how adding lanes changes the assignment pressure on the same pool.

Each worker polls its assigned lane and remembers an offset. The offset is just a bookmark that says how far this worker has read. It is easy to read ahead, but the bookmark should move only after ClickHouse has safely accepted the rows.

Switch Skew to Hot lane. One partition now fills faster than the others, which means one worker may become busier even though the whole system still has parallel lanes. This is why balanced partition keys matter in real ingestion systems.

The worker pool is not trying to turn everything into one giant queue. That would lose the benefit of partitions. Instead, it preserves order inside each lane while letting different lanes move independently.

The key takeaway: partitions are the safe unit of parallelism. They let workers scale out without scrambling order inside a lane. Next we will zoom into one worker and see how it turns polled events into a ClickHouse sized batch.

STAGE 3 OF 8

Batch Builder

Now that workers own lanes, let us zoom into one worker loop. The worker polls events, appends rows to a buffer and asks a practical question: is this buffer big enough or old enough to send?

Rows may arrive one at a time, but ClickHouse is built to ingest blocks. A block shares fixed work such as parsing the request, planning the insert and writing metadata across many rows. That is why the buffer matters.

Use the Workers control. More workers means more buffers can fill at the same time, which can raise throughput. It also means more possible inserts hitting ClickHouse, so each worker still needs a thoughtful flush rule.

Change Batch size to Small. The block leaves earlier, so individual events wait less time. The tradeoff is that ClickHouse now receives more insert requests, which can create more overhead and more parts.

With a larger batch, the worker waits a little longer and sends a fuller block. That can feel less realtime at the single event level, but it is often better for overall throughput because the fixed cost is shared by more rows.

The lesson is that realtime ingestion does not mean one insert per event. It means bounded waiting plus efficient blocks. Next we will open the batch and see how raw events become typed ClickHouse columns.

STAGE 4 OF 8

Parse And Shape

We just built a batch, but a batch is still just raw event data. ClickHouse cannot store random bytes as table columns. The worker has to decode each event and turn it into typed values that match the table schema.

The decoder reads the event format and extracts fields. JSON is flexible and human readable, but it usually costs more CPU to parse. Use the Format control to compare that text heavy path with a compact Proto style path.

The validator checks required fields and types before the block reaches ClickHouse. This is a protective gate. If one row is malformed, the worker should isolate that row instead of poisoning the entire insert.

Turn on Bad rows. The invalid row splits into the error path while clean rows keep moving forward. This is the same idea as a factory quality check: reject the bad item without stopping the whole line.

Clean rows are reshaped into columns. Instead of sending each event as a mini document, the worker builds vectors such as time, user, path and value. That shape matches how ClickHouse reads, compresses and stores data efficiently.

The takeaway: workers do more than forward bytes. They protect the database by turning messy stream events into clean, typed column blocks. Next we will send that block across the ClickHouse insert boundary.

STAGE 5 OF 8

Insert Path

05 INSERT PATH

We now have a clean column block. This is the moment the worker hands work to ClickHouse. The important question is not just whether the network send happened, but whether ClickHouse accepted the block safely.

In direct mode, the block moves straight toward the table write path. In async mode, ClickHouse can place rows in a server side buffer first, then flush them later as a larger insert. That can reduce insert overhead when many small inserts arrive.

Switch Insert mode to Async. Notice that the buffer lights up before the table. That means the worker may receive a faster response, while ClickHouse still controls when buffered rows become a real table insert.

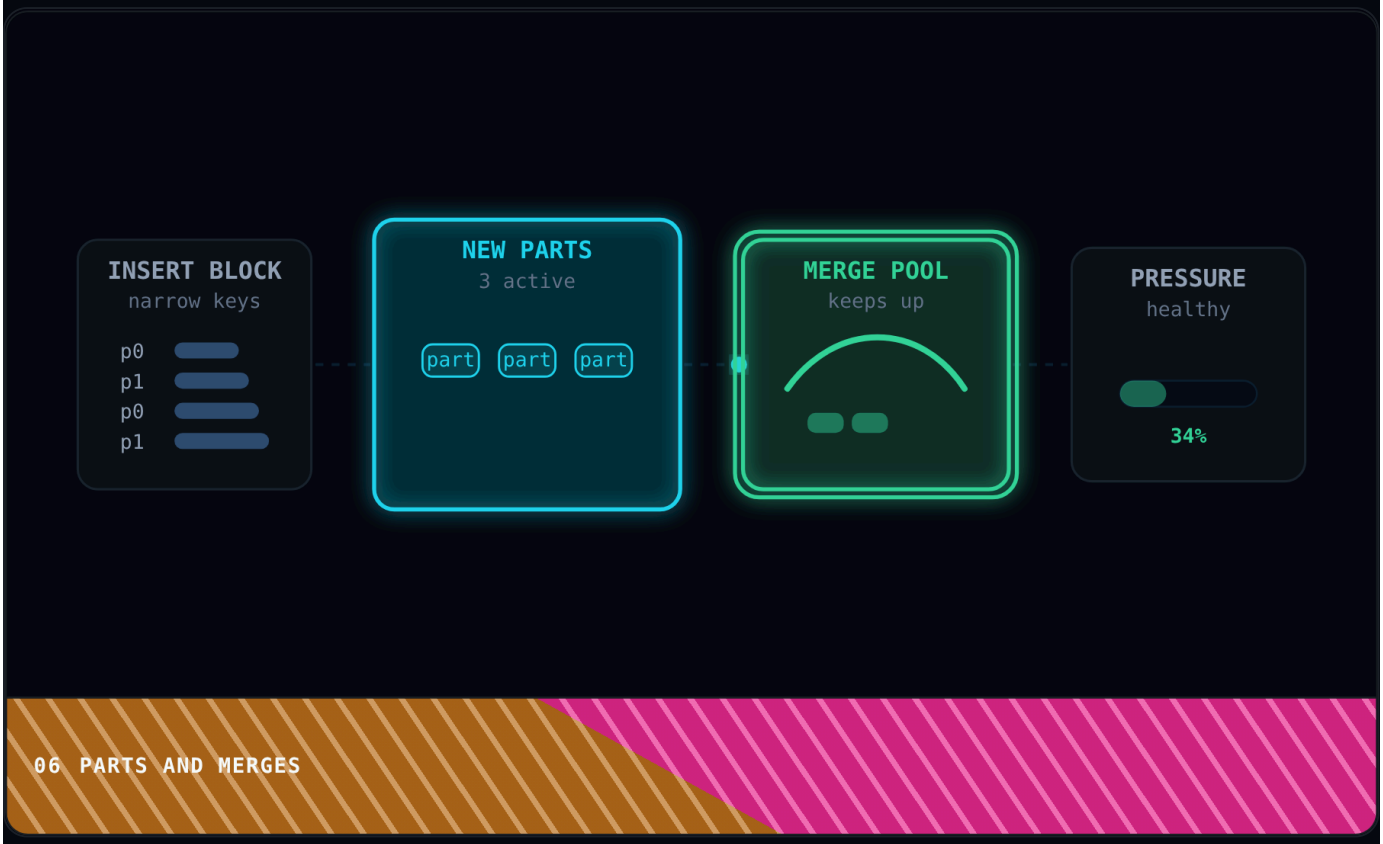
Use the Replicas control. With one replica, the path is simple. With more replicas, the same logical insert may need to reach more storage targets before the system treats the handoff as safe.

Once ClickHouse accepts the block, the worker can treat this insert attempt as successful. This is the boundary that matters for source safety. The offset bookmark should move only after this acceptance point, not when the worker merely starts sending.

The key idea is that the worker waits for the database boundary, not just the network send. That rule prevents the source from skipping rows after a failed insert. Next we will see what ClickHouse writes on disk after accepting the block.

STAGE 6 OF 8

Parts And Merges



06 PARTS AND MERGES

After ClickHouse accepts a block, a MergeTree table writes physical parts. A part is like a sealed folder of sorted column files. New inserts create new folders instead of rewriting the whole table.

Rows are split by table partition before they become parts. Change Partition spread to Wide and notice that one insert block can create more new parts. A wider spread means the same batch touches more table partitions.

New parts are immutable, which means ClickHouse does not rewrite existing parts for every insert. This keeps inserts fast. The tradeoff is that many small parts create cleanup work in the background.

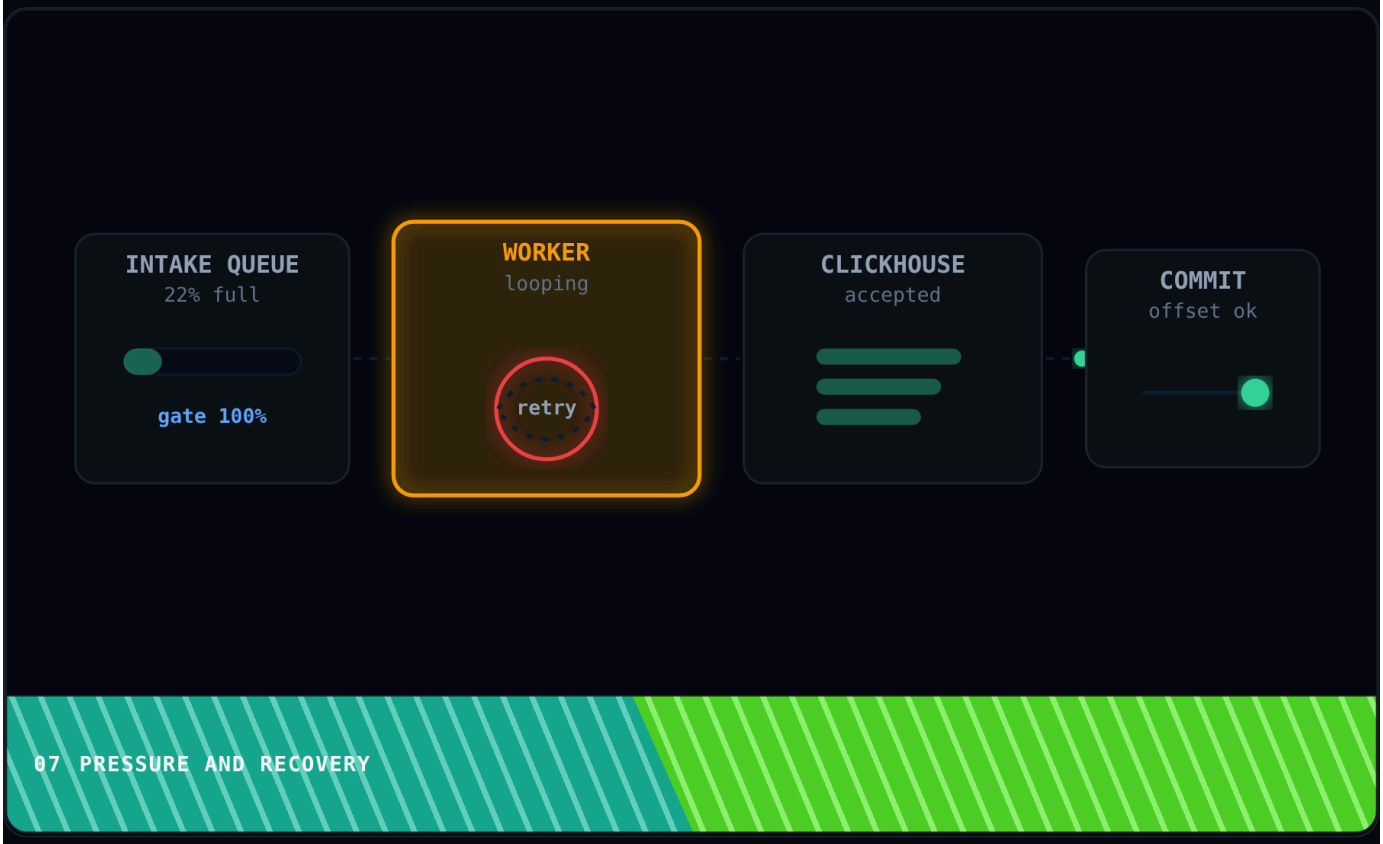
Background merges combine small parts into larger sorted parts. This is ClickHouse tidying the table while ingestion continues. The merge pool is not optional decoration, it is what keeps read performance from falling apart over time.

Switch Merge load to Busy. Parts now pile up faster than merges can clean them up, so the warning meter grows. That is a signal for the ingestion workers: slow down, build larger batches, or reduce partition spread.

The takeaway: worker batch size affects ClickHouse long after the insert request returns. Healthy ingestion is not only about sending rows quickly, it is also about creating parts at a rate ClickHouse can merge. Next we will connect that pressure back to retries, offsets and backpressure.

STAGE 7 OF 8

Pressure And Recovery



07 PRESSURE AND RECOVERY

We have followed the block all the way to disk parts. Now we close the worker loop. The worker must decide whether to read more, retry the same rows, or commit the source offset.

When pressure is low, the queue stays small and workers keep polling. Change Backpressure to High and watch the intake gate tighten. The system is deliberately choosing stability over maximum immediate intake.

If ClickHouse rejects the insert or the request times out, the worker should keep the same source offset. Use the Failure control to make the retry loop appear. The same rows are attempted again instead of being skipped.

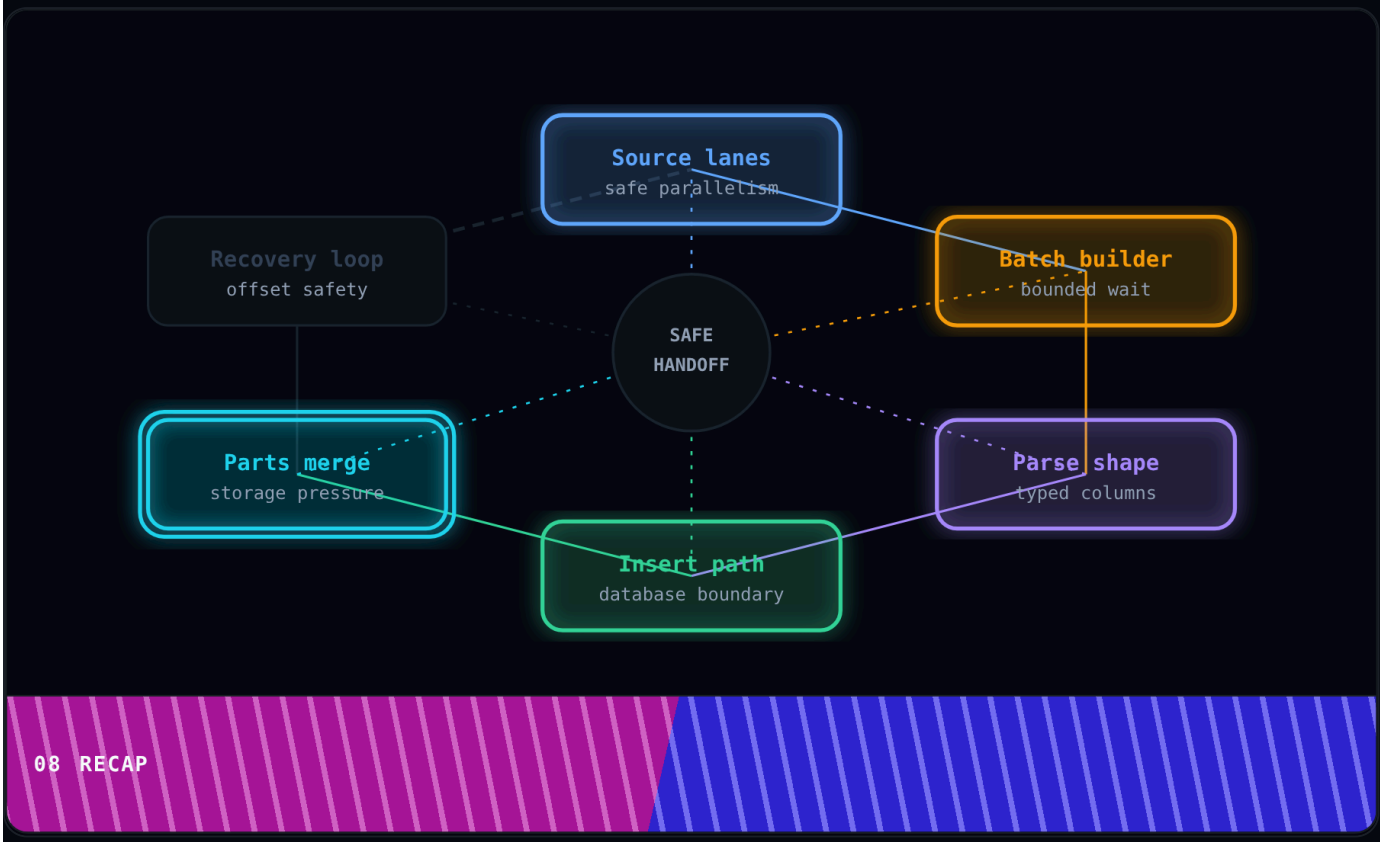
The important rule is that failure does not move the bookmark. This can create a duplicate attempt, so real systems also use idempotent design, deduplication keys, or careful insert semantics. The visual point is the same: never mark source rows as done before the database handoff succeeds.

After ClickHouse accepts the insert, the worker commits the offset. That commit says these source rows are now safely handed off. From this point forward, the source can advance without losing the rows that were just inserted.

This completes the ingestion loop: claim lanes, batch rows, shape columns, insert safely, manage parts and only then move offsets. Every stage exists to keep the stream moving without lying about what has been safely stored. Now let us step back and see the whole picture together.

STAGE 8 OF 8

Recap



08 RECAP

We started with source lanes. Partitions gave the worker pool safe parallelism: several lanes can move at once, while each lane keeps its own order. That is the foundation for scaling ingestion without turning the stream into chaos.

Then we learned why workers batch. A small wait creates larger blocks, which usually makes ClickHouse inserts healthier. This is the first major tradeoff: lower per event latency versus better throughput and fewer tiny parts.

Next we shaped data. Workers decoded raw events, rejected bad rows and built column blocks that match the table schema. This is where the worker protects ClickHouse from messy input instead of pushing every problem downstream.

Then the block crossed the ClickHouse boundary. The worker waits for ClickHouse to accept the insert before it treats the batch as safe. That acceptance point is what separates a sent request from a durable handoff.

After that, ClickHouse wrote parts and merged them in the background. Part pressure is why batch size matters even after an insert succeeds. If workers create tiny parts too quickly, the merge pool becomes the next bottleneck.

Finally, pressure and recovery closed the loop. Offsets move only after success and backpressure slows intake when ClickHouse needs room. The worker is always balancing two goals: keep data flowing and never claim rows are safe too early.

The unified takeaway: realtime ingestion is a relay with feedback. Workers keep the stream moving, but they advance the source only when ClickHouse has safely accepted the handoff. That is the core pattern behind reliable high throughput ingestion.