Your Fun Guide to Cloudflare Workers Analytics Engine: Unleash the Power of Custom Serverless Insights!

1. Blast Off with Analytics: Introducing Cloudflare Workers Analytics Engine!

- What's all the buzz about? A quick, engaging intro.
  Imagine deploying a sleek, super-fast Cloudflare Worker. It could be personalizing user experiences, acting as a smart API gateway, or even powering a full-stack application right from the edge. Now, the crucial question arises: how does one truly understand what's happening inside this Worker, beyond a simple "it ran successfully"? What if the goal is to track which features users adore, whether that A/B test is a hit, or to pinpoint performance bottlenecks for specific user segments – all without bogging down the Worker or incurring massive analytics bills? This is precisely where Cloudflare Workers Analytics Engine (WAE) steps into the limelight, poised to become an indispensable tool for developers seeking custom serverless analytics.1 WAE is engineered to provide "unlimited-cardinality analytics at scale," allowing for versatile data exploration directly from Workers, sidestepping common performance and cost hurdles.2
- What WAE is (and what it isn't – comparing with general Workers/Zone metrics).
  At its core, Cloudflare Workers Analytics Engine is a service enabling developers to dispatch custom, structured data events, known as "data points," directly from their Cloudflare Workers. These data points are not just arbitrary logs; they are well-defined packets containing dimensions (referred to as blobs), numerical metrics (doubles), and a special index field crucial for intelligent data handling and sampling. Once these data points are sent, they can be queried using a familiar SQL-like API, offering a powerful way to analyze custom application behavior.1 It's like having a personal, programmable analytics database, optimized for the serverless, edge-computing paradigm.
  It is vital to differentiate WAE from other analytics offerings within the Cloudflare ecosystem. Many developers are acquainted with "Workers metrics" and "Zone analytics" available through the Cloudflare dashboard.[4] These general metrics are excellent for monitoring operational health. "Workers metrics" provide insights into request counts, CPU time, memory usage, and error rates for individual Workers, answering questions like, "Is my Worker performing efficiently?" or "What's the error rate for this specific Worker function?".[4] "Zone analytics" offer a broader view, aggregating data for all Workers operating within a particular zone, useful for understanding overall traffic patterns and performance at the zone level.[4]
  WAE, in contrast, is tailored for *custom, application-level* analytics that the

developer defines and instruments.[2] It addresses questions specific to the application's logic, such as, "How many users clicked the 'subscribe' button after viewing the new pricing page today?", "What is the average processing time for image uploads larger than 5MB?", or "Which product categories are most frequently viewed by users in the 'EMEA' region?". The developer designs the events, the data they carry, and the resulting analytical queries.
A significant technical differentiator for WAE is its capability to handle "unlimited cardinality".[1] In practical terms, this means an application can have a vast number of unique values for its analytical dimensions – for instance, millions of distinct user IDs, product SKUs, A/B test variant names, or session identifiers – without the analytics system performance degrading or costs escalating uncontrollably. This is a substantial advantage, as many traditional metrics systems, including popular ones like Prometheus, can face challenges and become economically unviable when the number of unique label combinations (which translates to cardinality) grows excessively large.[1] WAE's architecture is designed to manage this complexity inherently.
The distinction between WAE and general operational metrics is fundamental. Operational metrics are provided by the platform about the platform's execution of code. WAE allows developers to generate telemetry *from within* their application code about the application's internal state, business logic, and user interactions. This shift from platform-provided observability to developer-defined, application-specific analytics is a key characteristic of WAE, enabling a much deeper and more tailored understanding of how serverless functions are being used and are performing in the context of the application's goals.[2]

- Why it's a game-changer for your Workers.
  Workers Analytics Engine offers several compelling advantages that can transform how developers approach analytics for their serverless applications:
  - **Deep, Custom Visibility**: WAE liberates developers from the constraints of generic, platform-level metrics. It empowers them to instrument their applications to capture precisely the events and data points that are most relevant to their specific business logic, user flows, and performance indicators.[2]
  - **Edge-Friendly Performance**: A critical design consideration for any service interacting with edge functions is performance. WAE excels here. The writeDataPoint() API call within a Worker is engineered to be extremely lightweight and, crucially, non-blocking. This means the Worker can dispatch an analytics event without waiting for an acknowledgment from the WAE backend, ensuring that the primary task of serving the user request or executing application logic is not delayed. This preserves the low-latency

  benefits of edge computing.[3]
  - **Cost-Effective for Rich Data**: Collecting detailed, high-cardinality custom analytics can often be prohibitively expensive with other solutions. WAE is designed to ingest large volumes of such data in a cost-effective manner. This efficiency is largely attributable to its sophisticated, built-in sampling mechanisms, which intelligently reduce the volume of stored data while preserving analytical integrity.[1] This encourages developers to be more comprehensive in their instrumentation.
  - **Unlocks New Possibilities**: The capabilities of WAE extend beyond simple charting and dashboarding. It serves as a foundational building block for more advanced applications. For instance, developers can construct their own usage-based billing systems by tracking granular consumption metrics per customer. It enables the creation of personalized analytics dashboards for end-users of a SaaS platform. Furthermore, WAE can be employed to implement custom security logic, such as identifying and responding to anomalous event patterns in real-time.[1]

2. The Magic Under the Hood: How WAE Actually Works

- The big picture: Architecture overview.
  Workers Analytics Engine is not a superficial addition to Cloudflare's offerings; it is architecturally rooted in the same robust, highly scalable analytics infrastructure that Cloudflare itself relies on to process an immense volume of events from its global network – on the order of tens of millions of events per second.1 This underlying foundation provides WAE with proven reliability and performance.
  The system can be conceptualized through several key components and flows:
  1. **Worker Binding**: Integration with WAE begins in the Worker's configuration file (wrangler.toml or wrangler.jsonc). Developers declare an analytics_engine_datasets binding, which creates a direct, optimized communication channel from their Worker code to a specific WAE dataset.[3] This binding manifests as an environment variable within the Worker's runtime.
  2. **writeDataPoint() API**: Inside the Worker, custom analytics events are dispatched by calling the writeDataPoint() method on the aforementioned binding. This method accepts a structured payload containing the analytical data.[3]
  3. **Ingestion & Sampling Backend**: This is the core of WAE's processing power. When data points arrive at the WAE backend, they undergo an intelligent ingestion process. A critical part of this process is the application of

sophisticated sampling techniques to manage data volume while preserving analytical value. The sampled data is then stored in a highly optimized time-series format, designed for rapid querying.[1]

4. **SQL API**: To retrieve and analyze the stored data, WAE provides a powerful HTTP-based API that allows users to execute queries using SQL syntax. This offers a flexible and familiar interface for data exploration and integration with external tools.[2]

- The art of sampling: Making sense of massive data.
  Sampling is a cornerstone of WAE's ability to handle vast amounts of data efficiently and cost-effectively. Instead of attempting to store and query every single event, which can be impractical and expensive, WAE intelligently selects a representative subset of the data.2
  - **Why Sample?**: The primary motivations for sampling are performance and cost. Storing and querying petabytes of raw event data is a significant engineering challenge. Sampling reduces the data footprint, leading to faster query responses and lower storage costs, making detailed analytics accessible even for high-traffic applications.[5]
  - **Weighted Adaptive Sampling**: WAE employs more than just simple random sampling. It utilizes "weighted adaptive sampling".[5] This implies that the sampling process can be influenced by certain characteristics of the data and can adjust dynamically.
  - **Equitable Sampling – The Fairness Doctrine**: A key characteristic of WAE's sampling is its "equitable" nature, particularly concerning the index field specified in writeDataPoint().[5] The system aims to ensure that data associated with each unique index value receives a fair representation in the sampled dataset. This is highly beneficial in multi-tenant scenarios or when analyzing per-user data. For example, if customer_id is used as the index, a very active customer generating millions of events might have their data sampled. In contrast, a less active customer generating only a few events is more likely to have all their events stored. This prevents the data from smaller entities from being drowned out by high-volume entities, ensuring their activity remains visible.[8]
  - **Adaptive Bit Rate (ABR) Sampling at Read Time – The "Google Maps" Analogy**: When a query is executed, especially one spanning a long time range or potentially touching a vast number of data points, WAE employs Adaptive Bit Rate (ABR) sampling at read time. Cloudflare uses the analogy of online mapping services like Google Maps to explain this [5]:
    - When viewing a map zoomed out to an entire continent, a lower-resolution image is displayed. Each pixel on the screen represents a large

geographical area.
    - As one zooms into a specific city or street, the map service provides higher-resolution imagery, where each pixel covers a much smaller area, revealing more detail.
    - Similarly, WAE adjusts the "resolution" or granularity of the data it scans based on the scope and complexity of the query. For queries over shorter time periods or smaller data subsets, it might use more detailed (less sampled) data. For queries over extended periods or very large datasets, it might use data that has a higher sample interval (more summarized) to ensure that results are returned quickly and efficiently.[5] This mechanism helps maintain consistent query performance regardless of the underlying data volume.
  - **The _sample_interval Field**: This field is crucial for interpreting query results from sampled data. When data is queried from WAE, each row returned includes a _sample_interval field. This integer value indicates how many original, unsampled events that particular row represents due to the sampling process. For instance, if a row has a _sample_interval of 100, it signifies that this single row in the result set effectively stands for 100 similar raw events that occurred. Consequently, to obtain accurate counts of original events, SQL queries should use SUM(_sample_interval) rather than COUNT(*) or COUNT(some_field).[5]
  - **Impact on Accuracy**: While sampling inherently means not every single event is stored, WAE's sampling mechanisms are designed to provide highly reliable and statistically meaningful results for most analytical queries, especially when the index field is chosen thoughtfully to align with common query patterns.[11] The accuracy of an aggregated result depends not just on the _sample_interval of individual rows but also on the total number of sampled rows read and aggregated to compute the result. Cloudflare has indicated plans to expose a "margin of error" alongside query results in the future, which would provide users with a quantitative measure of confidence in the sampled data.[11] However, it's important to note that accurate unique counts of fields *not* included in the index might be difficult or impossible to obtain, and very rare values of such fields might not be observable.[5]
  - **When Does Sampling Kick In?**: There isn't a fixed, universal threshold for when sampling begins. It is dynamic and depends significantly on the rate of data being written to a specific index. For very high-traffic indexes (Cloudflare's internal systems, for example, might see this trigger above approximately 100 data points per second per index), sampling will occur at write time.[11] WAE is also designed to handle bursty traffic gracefully; an

equalization process occurs every few seconds to normalize differences and manage spikes for any given index.[11]

- How data is stored and processed.
  The data points sent to WAE are structured events. Each data point consists of:
  - blobs: An array of strings, serving as categorical dimensions or labels for filtering and grouping (e.g., event type, user segment, product category).[3]
  - doubles: An array of numbers, representing the numerical metrics to be aggregated (e.g., page load time, transaction value, error count).[3]
  - indexes: A crucial array containing a single string value, which acts as the primary key for equitable sampling and often for joining data logically (e.g., user_id, session_id, customer_id).[3]

This structured data is ingested and stored by WAE for a period of **three months**.[12] If longer-term retention is required, users would need to implement a process to periodically export the data from WAE and store it in an alternative archival system. The storage system within WAE is highly optimized for time-series analytics. This means that queries filtered or aggregated over time ranges (e.g., "show me all events from last Tuesday" or "trend of daily active users over the past month") are designed to be exceptionally fast.[1] This is a notable improvement over some traditional database systems that can become slow and inefficient when performing analytical queries over extended date ranges.[1]The choice of the index field is a critical design decision when using WAE. Because WAE's equitable sampling is fundamentally tied to this index, and because querying across many distinct index values simultaneously can lead to performance degradation or highly sampled (low-resolution) data, careful consideration is needed.[5] The index is more than just a filterable field; it directly influences data quality, accuracy, and query performance. If an application has multiple, equally important dimensions for high-granularity analysis without a clear single primary entity key, developers might need to consider writing the same logical event to multiple WAE datasets, each tailored with a different index to optimize for specific query patterns.[11] This strategy, however, introduces its own complexities and would increase the number of data points written. This suggests WAE's sweet spot is for analytics that primarily revolve around a central entity, such as a customer, user, session, or a specific resource identifier.Furthermore, WAE's architecture, with its non-blocking write API and write-time sampling for high-volume indexes, clearly prioritizes write performance and overall system scalability.[3] This ensures that the act of collecting analytics does not impede the primary functionality of the Cloudflare Worker. The fixed three-month data retention period also contributes to Cloudflare's ability to manage storage at scale.[12] These design choices mean WAE is optimized for capturing high-volume, potentially bursty custom event data from the edge where aggregated trend analysis over recent timeframes is the primary goal. It is not, by itself, an exhaustive audit log guaranteeing capture of every single event with perfect fidelity, especially for events within heavily sampled indexes.

3. Let's Get Practical: Writing and Reading Your First Analytics

- Setting the stage: Configuration (wrangler.toml/.jsonc).
  Before a Cloudflare Worker can send data to Workers Analytics Engine, a connection needs to be established. This is achieved by defining an analytics_engine_datasets binding within the Worker's configuration file, typically wrangler.toml (or wrangler.jsonc if using JSON configuration).3 This binding acts as a conduit, making a specific WAE dataset accessible to the Worker code via an environment variable.
  Here is an example of how this configuration looks in a wrangler.toml file:
  Ini, TOML
  ```
  # wrangler.toml
  name = "my-analytics-worker"
  main = "src/index.ts" # or.js
  compatibility_date = "2023-10-26" # Use a recent compatibility date

  [[analytics_engine_datasets]]
  binding = "MY_ANALYTICS" # This is the variable name used in Worker code (e.g.,
  env.MY_ANALYTICS)
  dataset = "user_actions"   # A descriptive name for this WAE dataset (like a table name)
  ```

  In this configuration, MY_ANALYTICS will be the object available in the Worker's env parameter, providing the writeDataPoint() method. The dataset property, "user_actions", is the logical name for this collection of analytics data within WAE.[3]
- Writing custom events: writeDataPoint() in action.
  With the binding configured, the Worker can now send custom events to WAE using the writeDataPoint() method on the binding object.3 This method takes a single object argument containing the structured event data.
  The payload for writeDataPoint() consists of three key properties:
  - **blobs (Array of strings)**: These are the categorical dimensions of the event. They provide context and are used for filtering and grouping data during analysis. Examples include event names, page types, user segments, or product categories. A maximum of 20 blob strings can be included, and their total combined size must not exceed 5120 bytes.[3]
  - **doubles (Array of numbers)**: These are the numerical metrics associated with the event. These values can be aggregated (e.g., summed, averaged) during queries. Examples include counts, durations, scores, or monetary values. A maximum of 20 double values can be included.[3]
  - **indexes (Array with a SINGLE string)**: This field is critical for WAE's

sampling mechanism. It should contain exactly one string, which acts as the sampling key. This key is often a unique identifier like a user_id, session_id, or customer_id, guiding how WAE performs equitable sampling. The index string must not exceed 96 bytes in length.[3]

A crucial aspect of using writeDataPoint() currently is the **order consistency** for blobs and doubles. When querying data, blob1 will always refer to the first string provided in the blobs array, double1 to the first number in the doubles array, and so on. Therefore, it is essential to maintain a consistent order and meaning for these array elements across all Worker code that writes to the same dataset to ensure data integrity.[3] Cloudflare has indicated plans to allow named fields in the binding definition in the future, which would simplify this and reduce the risk of ordering errors.[1]The writeDataPoint() method is designed to be **non-blocking**. It returns immediately, allowing the Worker to continue processing its primary task without waiting for the analytics data to be persisted. This means await is not necessary when calling this method.[3]Consider an example where a Worker tracks product views on an e-commerce site:JavaScript

```javascript
// In your Worker's fetch handler (or any other relevant logic):
// Assuming 'env.MY_ANALYTICS' is the binding from wrangler.toml

async function handleRequest(request, env) {
  //... logic to determine product_sku, category, user_id, session_duration...
  const productSku = "electronics-phone-XYZ123";
  const productCategory = "Mobile Phones";
  const userId = "user_abcdef";
  const sessionDurationSeconds = 123.45;

  env.MY_ANALYTICS.writeDataPoint({
    blobs:, // event_type, category, sku
    doubles:, // view_count (always 1 for this event), user_session_duration
    indexes: [userId] // Sample by user ID
  });

  //... continue with serving the request...
  return new Response("Product page processed");
}
```

In this scenario, each product view event records the type of event, the product's category and SKU, a count of 1 for the view, the duration of the user's session at that point, and uses the userId for sampling.

- Querying your treasure trove: Using the SQL API from a Worker (or externally). Once data is flowing into WAE, it can be retrieved and analyzed using the WAE SQL API. This API provides a flexible way to access the custom analytics data.
  - **API Endpoint**: The SQL API is accessible via an HTTP endpoint: https://api.cloudflare.com/client/v4/accounts/<YOUR_ACCOUNT_ID>/analytics_

engine/sql.[9] Replace <YOUR_ACCOUNT_ID> with the actual Cloudflare account ID.

- **Authentication**: Requests to the SQL API must be authenticated using a Bearer token. This token should be generated from the Cloudflare dashboard and must have the Account Analytics: Read permission. For security, when querying from a Worker, this token should be stored as a secret.[9]
- **Request Method and Body**: SQL queries are submitted as the body of an HTTP POST request to the API endpoint.[9] The Content-Type header can be application/sql or text/plain.
- **Response Format**: The API responds with JSON-formatted data. This JSON object includes metadata about the returned columns (names and types), an array of data rows, and a count of the rows returned.[9]

Here's an example of how a Worker might query WAE to generate a simple report:JavaScript

```javascript
// In a Worker designed to fetch and display analytics summaries:
// Assume env.ACCOUNT_ID and env.WAE_API_TOKEN are configured as environment variable and
secret respectively.

async function getAnalyticsSummary(env, datasetName) {
  const query = `
   SELECT
     blob1 AS event_type,      -- Assuming blob1 is 'event_type'
     blob2 AS category,        -- Assuming blob2 is 'category'
     SUM(_sample_interval) AS total_events,
     AVG(double1) AS average_value_metric  -- Assuming double1 is some relevant metric
   FROM ${datasetName}
   WHERE timestamp > NOW() - INTERVAL '7' DAY -- Analyze data from the last 7 days
   GROUP BY event_type, category
   ORDER BY total_events DESC
   LIMIT 20;
  `;

  const API_ENDPOINT =
`https://api.cloudflare.com/client/v4/accounts/${env.ACCOUNT_ID}/analytics_engine/sql`;

  const response = await fetch(API_ENDPOINT, {
    method: "POST",
    headers: {
      "Authorization": `Bearer ${env.WAE_API_TOKEN}`,
      "Content-Type": "application/sql"
    },
    body: query,
  });

  if (!response.ok) {
    const errorText = await response.text();
    console.error(`Error querying WAE (${response.status}): ${errorText}`);
    throw new Error(`Failed to fetch analytics: ${response.statusText}`);
  }

  const queryResult = await response.json();
  return queryResult.data; // This is an array of result objects
}

export default {
  async fetch(request, env, ctx) {
    try {
      // 'user_actions' should match the 'dataset' name in wrangler.toml
      const summary = await getAnalyticsSummary(env, "user_actions");
      return new Response(JSON.stringify(summary, null, 2), {
        headers: { "Content-Type": "application/json" }
      });
    } catch (e) {
      return new Response(e.message, { status: 500 });
    }
  }
};
```
This Worker defines a function getAnalyticsSummary that constructs a SQL query to find the top 20 event types and categories by total events in the last 7 days, along with an average of a numerical metric. It then makes an authenticated POST request to the WAE SQL API and returns the results as JSON.The SQL dialect supported by WAE allows for standard filtering using WHERE clauses, grouping with GROUP BY, ordering with ORDER BY, and various aggregate functions like SUM(), AVG(), MIN(), MAX(), and COUNT(). When counting events, it's crucial to use SUM(_sample_interval) to account for WAE's sampling and get an accurate estimate of the original event volume.[9]**Table 3.1: writeDataPoint() Parameters Deep Dive**

Parameter	**Type**	**Purpose**	**Example (blobs, doubles, indexes)**	**Key Considerations & Limits**
blobs	Array of Strings	Categorical dimensions for filtering, grouping, and adding context to your events.	["user_login", "success", "europe-west"] (event_name, status, region)	Max 20 blobs. Total size of all blob strings combined: 5120 bytes. Order is critical and must

				be consistent for all writes to the same dataset. [3]
doubles	Array of Numbers	Numerical values associated with the event, used for aggregations (sum, average, min, max, etc.).	[1, 25.50, 3] (login_count, time_taken_seconds, attempt_number)	Max 20 doubles. Order is critical and must be consistent. [3]
indexes	Array of Strings (must contain exactly one string)	The primary key used for equitable sampling. This determines how WAE groups and samples your data to ensure fairness and performance.	["customer_id_abc123"] or ["session_id_xyz789"]	Max 1 index string. Max length 96 bytes. This choice is *fundamental* to data accuracy and query performance. [3]

The current design of `writeDataPoint()`, relying on ordered arrays for `blobs` and `doubles` without explicit field names in the API call itself, places a notable responsibility on the developer.[1, 3] This "implied schema" means that the first element in the `blobs` array will be queried as `blob1`, the second as `blob2`, and so on. Similarly for `doubles`. If a team of developers is working on an application, or if a single developer is writing to the same WAE dataset from multiple Workers or different parts of a complex Worker, meticulous care must be taken to ensure that the order and semantic meaning of these positional fields are identical everywhere. An accidental transposition of two `blob` values or a change in what `double3` represents in one part of the code could lead to significant data integrity issues, making query results misleading or nonsensical. This underscores the need for robust internal documentation of the chosen "schema" for each dataset, and potentially the use of shared constants, helper functions, or even TypeScript types/interfaces within the application code to enforce this structural consistency. The planned future enhancement to allow named `blobs` and `doubles` directly in the binding configuration [1] will be a significant improvement, mitigating this risk and enhancing the developer experience, especially for larger projects.

4. Superpowers Unlocked: Cool Things WAE Enables

Workers Analytics Engine is more than just a way to count clicks; it's a versatile tool that unlocks sophisticated capabilities for serverless applications.

- Custom analytics for your own customers.1
  One of the most powerful applications of WAE is the ability to expose tailored analytics dashboards to the end-users of a platform or SaaS product. If a service is built on Cloudflare Workers, WAE can track how individual customers or tenants are utilizing that service. By using the customer's unique identifier as the index when writing data points, it becomes straightforward to query and retrieve analytics specific to that customer. This data can then be presented to them within their own control panel or dashboard, offering valuable insights into their usage, performance, or any other metrics relevant to the service provided. This transforms analytics from an internal tool into a customer-facing feature.
- Building usage-based billing systems.1
  This is a prime use case where WAE truly shines, enabling businesses to move beyond flat-rate subscriptions to more equitable and flexible pricing models. The core concept involves meticulously tracking billable events generated by customers and then aggregating this data to calculate charges.
  1. **Track Billable Events**: Within the Cloudflare Worker that handles customer requests or actions, each time a billable event occurs (e.g., an API call is made, a certain amount of data is processed, a specific feature is invoked), a data point is written to a dedicated WAE dataset.
     JavaScript
     ```javascript
     // Example: Customer 'tenant_alpha' processes a 10MB file
     env.BILLING_EVENTS.writeDataPoint({
      blobs: ["file_processing", "large_file_tier"], // action_type, service_tier
      doubles: [10.0, 1], // megabytes_processed, event_count
      indexes: ["tenant_alpha"] // CRITICAL: Index by customer/tenant ID
     });
     ```
     This example logs the action, potentially a service tier, the amount processed, and an event count, all indexed by the tenant's ID.
  2. **The Role of index**: As emphasized in the Cloudflare documentation for usage-based billing [8], using the customer_id (or tenant ID) as the index is fundamental. WAE's equitable sampling is applied per index, meaning that

high-volume activity from one customer will not disproportionately affect the sampling (and thus accuracy) of data from lower-volume customers.
3. **blobs for Detail**: The blobs array can store contextual information such as the specific feature used, the API endpoint called, or the type of resource consumed. This allows for detailed, itemized billing.
4. **doubles for Quantity**: The doubles array should record the quantifiable aspect of the usage, such as the number of API calls (often 1 per event), bytes transferred, CPU seconds consumed, or any other relevant metric.
5. **End-of-Cycle Aggregation**: At the conclusion of each billing cycle (e.g., monthly), SQL queries are executed against the WAE dataset for each customer individually. Querying one customer at a time is recommended for optimal accuracy due to how sampling works.[8]
   ```sql
   SQL
   -- SQL to calculate total usage for 'tenant_alpha' for a specific billing period
   SELECT
     index1 AS tenant_id,
     blob1 AS action_type,
     SUM(_sample_interval * double1) AS total_megabytes_processed,
     SUM(_sample_interval * double2) AS total_processing_events
   FROM billing_dataset_name -- Replace with your actual dataset name
   WHERE index1 = 'tenant_alpha'
     AND timestamp >= '2023-08-01 00:00:00' -- Start of billing period
     AND timestamp < '2023-09-01 00:00:00'  -- End of billing period
   GROUP BY tenant_id, action_type;
   ```
   This query would provide the total megabytes processed and the number of processing events for tenant_alpha, adjusted for any sampling, broken down by action_type. This data then feeds into the billing system to generate invoices.

- Gaining deep insights into service health and user behavior on a per-user/customer basis.1
  WAE allows for a granular understanding of service performance and user interaction that goes far beyond aggregated metrics. Developers can track error rates experienced by specific users, monitor the performance of critical functions within a Worker segmented by customer tiers, or analyze feature adoption patterns at an individual user level. For example, if a particular user reports slow performance, WAE data (indexed by user ID and including performance doubles like processing time) can help verify and diagnose the issue. This capability enables proactive customer support, identification of users struggling with certain features, and a nuanced understanding of how different customer

segments engage with the application.

- Instrumenting frequently called code paths without performance impact or overwhelming external systems.1
  The non-blocking nature of the writeDataPoint() API call is a significant advantage.3 It allows developers to insert detailed instrumentation into critical or frequently executed sections of their Worker code without introducing latency that would affect the user experience. This is invaluable for:
  - **Debugging Complex Flows**: Logging entry/exit points of functions or intermediate state variables during complex, multi-step operations.
  - **Frequency Analysis**: Tracking how often specific conditions are met or particular code branches are executed, which can inform optimization efforts or reveal unexpected behavior.
  - **A/B Testing Internal Logic**: When testing different internal algorithms or data transformation strategies, WAE can log the outcomes and performance characteristics of each variant, associated with a variant identifier in the blobs. WAE is designed to handle the potentially high volume of events generated by such detailed instrumentation, a load that might overwhelm traditional logging systems or prove too costly with third-party analytics services not optimized for high-frequency, custom event streams from globally distributed edge functions.

The ability of WAE to provide custom, scalable analytics directly within the serverless environment represents a significant step in democratizing data-driven development. Traditionally, setting up such systems was a complex undertaking, often requiring dedicated data engineering teams and substantial infrastructure.[1] WAE, by abstracting this complexity behind simple APIs and integrating directly with the Worker runtime, lowers this barrier significantly. This empowers developers, even in small teams or as individuals, to incorporate sophisticated analytics into their applications from the outset. This fostering of a data-driven approach at the edge allows for more rapid iteration, better-informed product decisions, and a deeper understanding of application performance and user engagement, directly benefiting the end-users and the business.

5. Adventures in Serverless Land: WAE & Friends (Case Studies & Fun Implementations)

Workers Analytics Engine truly comes alive when seen in action, especially when it collaborates with other Cloudflare serverless technologies. These examples illustrate its versatility and power.

- **Official Examples & Demos (from Cloudflare):**
  - **shrty.dev - The Mighty URL Shortener**: A well-showcased example from

Cloudflare, shrty.dev demonstrates a practical application combining several Cloudflare services.[14]

  - **Cloudflare Workers KV**: Serves as the high-performance database, storing the crucial mapping between the generated short codes and their corresponding long URLs. KV's low-latency global reads are ideal for ensuring fast redirections when a short link is accessed.[13]
  - **Workers Analytics Engine**: This is the engine behind the analytics dashboard for shrty.dev. Each time a short link is successfully resolved and the user is redirected, an event is dispatched to WAE. This event likely includes the short code, a timestamp, and potentially anonymized request data like geographic origin. This data allows shrty.dev to track click counts, identify popular links, and analyze traffic patterns.[14]
  - **Cloudflare Workers AI (Optional Enhancement)**: To add a layer of sophistication, shrty.dev can leverage Workers AI. This could be for automatically generating memorable or context-aware short slugs, or even for providing a natural language interface for administrators to manage their links through a chat-like interaction.[14]
  - *Key Takeaway*: This example perfectly illustrates a common and powerful pattern: KV for fast, operational data storage and retrieval; WAE for time-series analytics on the usage and interaction with that data; and Workers AI for adding intelligent features.
- **Cloudflare's Internal Dogfooding - R2 Analytics**: In a strong testament to WAE's capabilities, Cloudflare's own R2 object storage team utilizes WAE to monitor the R2 service itself. They track metrics such as the number of read and write operations, unique users initiating these requests, total bytes transferred, and the latency of various R2 operations.[1]
  - *Key Takeaway*: This internal adoption underscores WAE's scalability and reliability. If it's robust enough for Cloudflare to monitor a core infrastructure service like R2, it's well-equipped for demanding user applications. It also shows WAE's flexibility in tracking not just application-level events but also infrastructure-like performance metrics.
- **Custom Security with WAE - A Fail2Ban-like Implementation**: WAE can be employed for custom security event tracking and response. For instance, a Worker could log login attempts (recording details like IP address, username, timestamp, and success/failure status) to WAE. Another Worker, perhaps a scheduled one, could then periodically query WAE for suspicious patterns, such as an excessive number of failed login attempts from a single IP address within a short timeframe. Upon detecting such activity, the Worker could use Cloudflare's firewall APIs to temporarily block the offending IP.[1]

        - *Key Takeaway*: This demonstrates that WAE's utility extends beyond typical business or performance analytics into the realm of custom, real-time security monitoring and automated response based on observed event patterns.
- **Community Projects & Inspirations:**
    - **Counterscale - Your Own Privacy-First Google Analytics Alternative**: Counterscale is an open-source project that enables users to self-host a web analytics solution built entirely on the Cloudflare stack. It notably uses Workers Analytics Engine as its sole "database".[18]
        - A Cloudflare Worker collects page views and other user interaction metrics from websites, writing these events to a WAE dataset. The Counterscale dashboard then queries this WAE dataset via the SQL API to render analytics charts and reports.
        - The project aims for minimal operational costs, leveraging Cloudflare's generous free tiers.
        - *Key Takeaway*: Counterscale showcases WAE's capability to serve as the primary data store for complete analytics applications. It's particularly appealing for users who prioritize data ownership, privacy, and cost-efficiency.
    - **Chickadee - Another Privacy-Focused Analytics Tool**: Similar in spirit to Counterscale, Chickadee is another community-driven example of a privacy-respecting web analytics tool that leverages WAE for its backend.[18] A distinguishing feature mentioned is its ability to allow setting user IDs (with appropriate consent) for tracking user retention.
- Hypothetical Fun Projects (Illustrating WAE with other Cloudflare tech):
  These illustrative scenarios further highlight how WAE can be combined with other Cloudflare serverless components:
    - **WAE & Cloudflare D1 (Relational Data + Event Analytics):**
        - *Scenario*: Consider a multiplayer online game where player profiles, scores, achievements, and inventory items are stored in Cloudflare D1, Cloudflare's serverless SQL database.[9]
        - *WAE Use*: Track significant in-game events by writing to WAE. For example: env.GAME_ANALYTICS.writeDataPoint({ blobs:, doubles: [1, playerLevel], indexes: [playerId] }). This logs an achievement unlock, the player's level at the time, and is indexed by player ID. Other events could include "rare_item_looted", "boss_defeated_time", "level_completed_duration".
        - *Benefit*: This allows for analysis of achievement popularity, player progression rates, correlation between item drops and player levels, or

average times to complete certain objectives, all without putting heavy analytical load on the D1 database which holds the current game state. D1 manages the canonical state, while WAE tracks the historical trends and frequencies of discrete events.

- **WAE & Cloudflare R2 (Object Storage + Custom Access Analytics):**
  - *Scenario*: A platform for hosting and sharing large media files, such as educational videos or podcasts, using Cloudflare R2 for storage.[13]
  - *WAE Use*: When a Cloudflare Worker serves a file from an R2 bucket, it also logs access details to WAE: env.R2_ACCESS_LOGS.writeDataPoint({ blobs:, doubles:, indexes: [userIdOrFileId] }).
  - *Benefit*: This enables granular analysis of download trends per file type, geographic access patterns, bandwidth consumption per user or per file, or download speeds experienced by different user tiers. This level of custom, application-aware analytics can be more detailed than R2's built-in operational metrics.
- **WAE & Cloudflare Queues (Background Processing + Job Performance Analytics):**
  - *Scenario*: An application that utilizes Cloudflare Queues for asynchronous background job processing, such as image resizing, sending email notifications, or generating complex reports.[2]
  - *WAE Use*: The consumer Worker that processes jobs from the Queue, upon completing (or failing) a job, writes an event to WAE: env.QUEUE_JOB_METRICS.writeDataPoint({ blobs:, doubles:, indexes: [queueName] }).
  - *Benefit*: This allows tracking of key performance indicators for background jobs, such as average processing times per job type, success/failure rates, and retry counts. This provides application-level insights into the efficiency and reliability of the queuing system, complementing the operational metrics provided by Queues itself.[33]
- **WAE & Cloudflare Workers AI**: Beyond shrty.dev, one could analyze the usage patterns of different AI models invoked via Workers AI, track token consumption per user for specific AI tasks, or A/B test the effectiveness of different prompts by logging the input parameters and output quality scores (if measurable) to WAE.[14]
- **WAE & Cloudflare Durable Objects**: For applications using Durable Objects for stateful coordination, WAE can track the frequency of specific state transitions, the duration of critical operations within an Object, or patterns of interaction with an Object, providing insights into its lifecycle and usage

characteristics over time.[13]

The common thread in these examples is WAE's role in providing a custom, scalable analytics layer that complements the primary function of other Cloudflare services. This ability to combine specialized services is a hallmark of the Cloudflare developer platform. Each service (KV for key-value, D1 for SQL, R2 for objects, Queues for messaging, WAE for custom analytics) is optimized for its specific domain. WAE becomes particularly valuable by offering a unified way to gather operational and business intelligence across these diverse components, all orchestrated by Cloudflare Workers. This encourages a composable, microservices-like architecture within the serverless paradigm, with WAE serving as a powerful, cross-cutting tool for observability and insight generation. This level of detailed, per-user or per-event tracking is crucial for understanding feature adoption, user engagement funnels, and identifying points of friction, all of which are vital for product-led growth strategies and for building a deep, nuanced understanding of how systems perform under real-world conditions.

6. Making Data Dance: Visualizing Your WAE Insights

Collecting custom analytics is only half the journey; the other half is visualizing and interpreting that data to extract meaningful insights. Workers Analytics Engine primarily relies on its SQL API for external data access, which opens up several avenues for visualization.

- Using the Cloudflare Dashboard for WAE?
  Cloudflare provides a custom dashboards feature within its main dashboard, allowing users to build their own views of analytics data.36 However, the available datasets for these custom dashboards primarily include "HTTP Requests" and "Security Events".36 Datasets from Workers Analytics Engine are not explicitly listed as direct, selectable sources for building charts within this native Cloudflare custom dashboard feature.36 While general console.log() messages from Workers (which could, in theory, include stringified WAE event details if a developer chose to log them there) can be viewed and filtered in the "Workers Logs" section of the dashboard 34, this is distinct from querying and visualizing the structured datasets stored within WAE itself. The primary and recommended method for querying WAE data for external visualization is its SQL API.10
- Grafana: The Go-To for WAE Visualization.
  Grafana has emerged as a popular choice for visualizing WAE data, largely due to its flexibility in connecting to various data sources, including those accessible via HTTP/JSON APIs.

- **Official Integration Path**: Cloudflare's documentation explicitly mentions the ability to query WAE from Grafana, indicating this as a supported and intended use case.[2] The mechanism for this is the WAE SQL API.
- **Community Examples & Setup**:
  - A notable community project, markdembo/cloudflare_analytics_engine_demo on GitHub [38], provides a docker-compose setup that includes a Grafana instance pre-configured to query WAE. This demo showcases connecting Grafana to the WAE SQL API endpoint. The connection is typically configured using Grafana's generic JSON API data source (or a similar plugin that can make HTTP requests and parse JSON responses), with environment variables in the Grafana setup defining the WAE SQL API endpoint and the necessary authentication bearer token.
  - To set up Grafana for WAE based on such examples [38]:
    1. Deploy a Grafana instance (Docker is a common method).
    2. Within Grafana, add a new data source. A generic "JSON API" data source or a plugin supporting custom HTTP requests is often suitable.
    3. Configure this data source to point to your WAE SQL API endpoint: https://api.cloudflare.com/client/v4/accounts/<YOUR_ACCOUNT_ID>/analytics_engine/sql.
    4. Ensure the data source configuration includes the required Authorization: Bearer <YOUR_API_TOKEN> header for authenticated requests.
    5. In Grafana dashboard panels, write SQL queries directly. These queries will be sent to the WAE SQL API. Remember to use WAE-specific considerations like SUM(_sample_interval) for accurate event counts.
- **Distinction from General Cloudflare-Grafana Integration**: It's important to note that Grafana Cloud also offers a general "Cloudflare integration".[39] This official Grafana Cloud integration is primarily designed to ingest *account and zone analytics* (overall requests, bandwidth, CPU utilization, etc.) from Cloudflare. It typically works by using a Prometheus exporter that Cloudflare provides for these general metrics. It provides pre-built dashboards for "Cloudflare Geomap overview," "Cloudflare worker overview," and "Cloudflare zone overview".[39] This is a different mechanism and targets different data than visualizing custom WAE datasets via the WAE SQL API.

- Datadog, Splunk, Looker, New Relic, Sentinel: How do they fit?
  These major observability and SIEM platforms have robust integrations with Cloudflare, but these integrations primarily focus on ingesting general Cloudflare

logs and metrics, often through Cloudflare Logpush.41 Logpush allows for the streaming of detailed logs (HTTP requests, firewall events, DNS queries, etc.) to these external systems.

- **Datadog**: The standard Datadog integration uses the Cloudflare Analytics API for general zone metrics and relies on Logpush for detailed log ingestion.[41] There is no explicit, out-of-the-box Datadog connector documented for directly querying the WAE SQL API.[41]
- **Splunk**: The Cloudflare App for Splunk is designed to work with data sent via Logpush.[44] Again, a direct WAE SQL API connector is not explicitly mentioned.[53]
- **Looker**: The documented Looker integration tutorial involves using Logpush to send Cloudflare logs to Google Cloud Storage (GCS), then importing them into Google BigQuery. Looker then connects to BigQuery as its data source.[43] This is not a direct connection to the WAE SQL API.
- **New Relic**: Integration primarily utilizes Logpush for sending Cloudflare logs to New Relic.[45]
- **Microsoft Sentinel**: The integration path involves Logpush to Azure Blob Storage, from which a Microsoft-provided connector ingests the logs into Sentinel.[47]

To get WAE data into these platforms, several approaches could be considered:

1. **Indirectly via Logpush (Less Ideal for Structured WAE Data)**: If WAE data points (or summaries derived from them) are console.log()-ed from the Worker, these log messages could be captured by Logpush. However, this means sending potentially unstructured or less optimally structured data, rather than querying the rich, structured WAE dataset directly.
2. **Custom Ingestion via SQL API (Most Robust)**: The most flexible and robust method would be to create an intermediary service. This could be another Cloudflare Worker (perhaps on a schedule) or a process running elsewhere. This service would periodically query the WAE SQL API for the desired data, transform it if necessary, and then send it to Datadog, Splunk, etc., using their respective data ingestion APIs. This is a custom solution but offers full control.
3. **Generic API/DB Connectors**: Some of these platforms might offer generic "SQL database" connectors or "JSON API" data source types. It might be possible to configure these to query the WAE SQL API directly, similar to the Grafana approach. This would require investigation on a per-tool basis to check compatibility with WAE's authentication and API specifics.

- Exporting data via the SQL API for custom visualization.
  Because the WAE SQL API returns data in JSON format 9, it's straightforward to

write custom scripts (e.g., using Python with the requests library, or Node.js with fetch) to programmatically retrieve analytics data. This raw JSON data can then be processed and fed into any custom charting library (like D3.js, Chart.js, Plotly.js for web-based visualizations) or imported into business intelligence tools or data analysis environments (like Jupyter notebooks with Pandas) that can consume JSON or tabular data. This offers maximum flexibility for bespoke reporting and visualization needs.

Table 6.1: Visualization & Integration Options for WAE Data

Tool/Method	**Primary Connection to WAE**	**How it Works for WAE Data**	**Use Cases for WAE**	**Notes**
Grafana	WAE SQL API (typically via a JSON API data source plugin)	Grafana is configured to make HTTP POST requests with SQL queries to the WAE SQL API endpoint and visualizes the JSON response.	Building custom dashboards, time-series charts, and alerts based directly on WAE datasets.	Community demos like markdembo/cloudflare_analytics_engine_demo exist.[38] This is distinct from the general Cloudflare-Grafana Cloud integration which uses a Prometheus exporter for account/zone metrics.[39]
Cloudflare Dashboard	Indirect (Potentially via Workers Logs) / Not Directly for WAE Datasets	WAE datasets are not listed as direct selectable sources for the Cloudflare custom dashboard feature.[36] console.log() output from Workers, which *could* include	Limited for direct, structured WAE dataset visualization. Primarily for operational Worker logs.	The WAE SQL API is the main interface for querying WAE datasets.

		WAE event data if manually logged, can be viewed in Workers Logs.[37]		
Datadog, Splunk, Looker, New Relic, Sentinel	Primarily Logpush for general Cloudflare Logs; Custom solution needed for WAE SQL API data	These platforms have strong integrations for general Cloudflare logs (HTTP, security, etc.) via Logpush.[41] For WAE data, a custom pipeline would typically be needed: query WAE SQL API -> transform -> ingest into the platform using its API. Some might offer generic API/DB connectors that could potentially be adapted.	Advanced log correlation, SIEM integration, broader observability if WAE data is custom ingested alongside other telemetry.	Standard integrations focus on non-WAE data. WAE SQL API offers flexibility for building custom data pipelines to these tools.
Custom Script/Application (e.g., Python, Node.js)	WAE SQL API	Scripts make HTTP POST requests to the WAE SQL API, retrieve JSON results, and then process/visualize using any preferred libraries (D3.js, Pandas, etc.) or export to other systems.	Custom reports, data export for long-term archival, bespoke visualizations, feeding data into data warehouses or machine learning pipelines.	Offers maximum control over data processing, presentation, and integration with other systems.

The consistent theme across external data access for WAE is the central role of its SQL API.[2, 9, 10] While Cloudflare provides many integrations for its broader suite of services (often leveraging Logpush for high-volume raw logs), the WAE SQL API is the specific, standardized gateway for pulling structured, custom analytics data. This API-first approach empowers users to connect WAE to a diverse range of tools, either through direct API calls or by building simple intermediary services, ensuring that the valuable insights captured by WAE can be utilized wherever they are most needed.

7. Know Before You Go: Limitations & Why You Might *Not* Use WAE

While Workers Analytics Engine is a powerful tool, it's essential to understand its limitations and the scenarios where it might not be the optimal choice.

- Data Point & Query Limits:
  WAE operates with specific constraints on data ingestion and querying:
  - **Writes per Invocation**: A single Cloudflare Worker invocation can make a maximum of 25 writeDataPoint() calls to WAE.[12] This encourages batching or summarizing data if many potential events occur within one invocation.
  - **Blobs**: Each writeDataPoint() call can include up to 20 blobs (string dimensions). The total size of all blob strings in a single call must not exceed 5120 bytes.[12]
  - **Doubles**: Up to 20 doubles (numerical metrics) can be included in each writeDataPoint() call.[12]
  - **Index**: Only one index string is permitted per writeDataPoint() call, and its length is limited to 96 bytes.[12]
  - **General Worker Limits**: Beyond WAE-specific limits, standard Cloudflare Workers limits regarding CPU time per request, memory usage, request/response body sizes, and daily/burst request rates also apply to the Worker executing the writeDataPoint() calls.[54] These can indirectly affect how much analytics processing can be done.
- Data Retention:
  Data written to Workers Analytics Engine is stored for a period of three months.12 This makes WAE suitable for operational analytics, trend analysis over recent periods, and debugging. However, it is not designed for long-term archival of raw analytics data. If retention beyond three months is required, users must implement a system to periodically export data from WAE (via the SQL API) and store it in another system like R2, BigQuery, or a traditional data warehouse.
- Sampling Considerations:

The sampling mechanisms in WAE, while crucial for performance and cost-effectiveness, have implications for data analysis:

  - **Accuracy for Rare Events**: If an event is extremely infrequent and happens to fall within an index that is being heavily sampled due to other high-volume activity for that same index, the rare event might be missed entirely, or its counted occurrences might be less precise.[5]
  - **Complex Queries Across Many Indices**: Executing queries that span a large number of distinct index values simultaneously can trigger more aggressive Adaptive Bit Rate (ABR) sampling at read time. This can result in lower-resolution (more heavily sampled) data being returned to ensure query performance, potentially obscuring finer details.[11]
  - **Unique Counts on Non-Indexed Fields**: Obtaining accurate unique counts of values in blob or double fields that are *not* part of the index can be challenging or impossible with sampled data.[5] The index is the primary basis for ensuring representative sampling.

- **When WAE Might Not Be The Best Fit:**
  - **Need for 100% Accurate Log of Every Single Event**: If the application requires a guaranteed, verbatim record of every single event for strict audit compliance, financial transaction logging, or critical traceability, WAE's sampling model might not be suitable. In such cases, a traditional logging system pushing to a dedicated log aggregator, or writing critical transaction details directly to a database like Cloudflare D1 or R2 (and accepting the associated performance and cost implications for high-volume writes) might be more appropriate.
  - **Very Long-Term Data Retention Needed Natively**: As mentioned, WAE's three-month retention window means it's not a solution for indefinite data archival.
  - **Primary Need is Basic Operational Metrics**: If the primary goal is to monitor fundamental Worker health – such as uptime, overall request volume, CPU/memory usage, and basic error rates – the built-in Workers metrics available in the Cloudflare dashboard are often simpler to use and may be sufficient.[4] WAE is for custom, application-specific insights beyond these.
  - **Highly Complex, Multi-Dimensional Exploratory Analysis without a Clear Primary Index**: If the analytical needs involve frequent, ad-hoc queries across many different dimensions, and there isn't a natural, dominant primary index (like user_id or session_id) that aligns with most queries, users might find WAE's sampling behavior challenging. This could lead to less detailed results or necessitate writing data to multiple WAE datasets, each optimized with a different index.[11]

- **Alternatives for Different Needs:**
    - **Cloudflare General Metrics/Logs**: For basic Worker operational health and platform-level logging.[4]
    - **Cloudflare Logpush to SIEM/Log Management Systems**: For comprehensive, raw log analysis of HTTP requests, firewall events, etc., sent to tools like Splunk, Datadog, or Sentinel.[41]
    - **Direct Storage from Worker (D1, R2, KV)**: For scenarios requiring full control over data, no sampling, and different consistency or storage models (e.g., storing critical state in D1, large objects in R2, configuration in KV). This comes with its own design considerations for schema, performance, and cost, especially for high-volume writes from Workers.[13]
    - **Third-party Serverless Analytics Platforms**: General cloud providers like AWS (Lambda with CloudWatch Logs/Metrics/X-Ray), Google Cloud (Cloud Functions with Operations Suite), and Azure (Functions with Application Insights) offer their own integrated observability and analytics solutions for serverless functions.[57] These provide different feature sets, pricing structures, and levels of integration with their respective ecosystems.
- Pricing:
  WAE's pricing model is designed to be straightforward, based on two primary metrics: data points written and read queries executed.6
    - **Data Points Written**: Each call to writeDataPoint() in a Worker counts as one data point written.
    - **Read Queries**: Each HTTP request made to the WAE SQL API counts as one read query. Cloudflare emphasizes that, unlike some other platforms, the cost per data point or per query is the same regardless of the number of dimensions (cardinality) or the complexity/size of the data within that point/query.[1]
    - **Workers Free Plan**: Includes 100,000 data points written per day and 10,000 read queries per day.
    - **Workers Paid Plan**: Includes 10 million data points written per month (with additional points charged at $0.25 per million) and 1 million read queries per month (with additional queries charged at $1.00 per million).[6] (Note: Pricing details are subject to change; always refer to the latest Cloudflare documentation.)

Table 7.1: Cloudflare Workers Analytics Engine - Key Limits

Limit Type	Free Plan	Paid Plan	Details / Source

Data Points Written (per Worker invocation)	25	25	12
Data Points Written (total)	100,000/day	10 million/month (then $0.25/million)	6
Read Queries (total)	10,000/day	1 million/month (then $1.00/million)	6
Blobs per writeDataPoint	20 (total 5120 bytes)	20 (total 5120 bytes)	12
Doubles per writeDataPoint	20	20	12
Indexes per writeDataPoint	1 (max 96 bytes)	1 (max 96 bytes)	12
Data Retention	3 months	3 months	12

The design choices inherent in WAE—such as the limit on data points per invocation, the fixed retention period, and the reliance on sampling—create specific trade-offs. These limits encourage developers to be thoughtful about the data they send, potentially batching information into fewer, richer data points. The three-month retention positions WAE as a tool for operational and recent-trend analysis rather than a long-term data archive.[12] Sampling is fundamental to its cost and performance at scale but means it isn't a verbatim record of every event.[5] Therefore, WAE is a specialized instrument excelling at high-cardinality, custom, time-series analytics from the edge. For use cases demanding exhaustive audit logging, very long-term data storage, or scenarios where every single event must be captured with perfect fidelity, WAE should be complemented by, or in some cases replaced with, other tools like traditional logging systems, direct database writes for critical data, or archival to long-term storage like R2.

8. The Horizon: What's Next for Workers Analytics Engine?

- **General Availability (GA) Status**: Workers Analytics Engine is Generally Available (GA), a milestone announced as part of Cloudflare's broader initiative to simplify and enhance stateful application development on its platform.[25] This GA announcement was a highlight of Developer Week 2024 [60], signaling its production-readiness. Earlier blog posts, such as one from November 2022 detailing its internal use by Cloudflare teams (like the R2 team), provide context on its development journey and capabilities even before GA.[1]
- **Recent & Future Developments:**
  - The GA announcement emphasized a straightforward pricing model based on data points written and read queries, with a key benefit being no cost penalty for high cardinality or the complexity of individual data points/queries.[1] This is designed to make costs predictable and encourage rich data collection.
  - A significant planned enhancement, mentioned in earlier introductions of WAE, is the ability for developers to **name their blobs and doubles directly in the binding configuration**.[1] This would be a major improvement for developer experience and data integrity, as it would remove the current reliance on strict positional ordering in the writeDataPoint arrays and allow for more self-documenting and robust code. This is a key future development to anticipate.
  - Cloudflare is also investing in a broader observability platform. The "Workers Logs" feature, which allows viewing console.log messages, errors, and exceptions from Workers directly in the Cloudflare dashboard with advanced filtering and querying capabilities [34], is part of this initiative. While Workers Logs are distinct from WAE datasets, they are part of the same ecosystem aimed at providing comprehensive telemetry and insight into Worker behavior. The goal is to help developers easily correlate different types of telemetry data.
- **Focus on Developer Experience**: Cloudflare consistently message their commitment to improving the developer experience on their platform. WAE is a component of this strategy, designed to make custom, scalable analytics accessible and easy to integrate for developers building on Workers.[1]
  The General Availability of WAE, alongside D1 (serverless SQL database) and Hyperdrive (database accelerator), signals Cloudflare's strong commitment to supporting full-stack, stateful applications at the edge.[25] As developers increasingly build more complex applications on Cloudflare Workers—applications that manage state, interact with databases, and execute intricate business logic—the need for sophisticated, custom analytics to understand and optimize these applications grows proportionally. WAE is positioned to meet this need, providing the analytics piece of this evolving stateful edge puzzle. Its GA status

implies production-readiness and ongoing support from Cloudflare. Consequently, WAE is likely to see continued development, deeper integrations with other Cloudflare services, and further refinements to its usability and feature set, mirroring the overall maturation of stateful capabilities on the Workers platform.

9. Wrapping Up: Your WAE Journey Begins!

Cloudflare Workers Analytics Engine emerges as a remarkably potent and developer-friendly tool for anyone looking to gain deep, custom insights into their serverless applications. It successfully abstracts the immense complexity of building and maintaining a scalable, high-cardinality analytics system, offering a simple API for both writing and querying custom event data directly from the edge.

The core power of WAE lies in its ability to handle vast amounts of custom, structured telemetry with impressive performance, largely thanks to its sophisticated sampling mechanisms and architecture built upon Cloudflare's own battle-tested infrastructure. This translates into tangible benefits:

- **Unprecedented Visibility**: Go beyond generic metrics and understand the specific behaviors, patterns, and performance characteristics that matter to *your* application and *your* users.
- **Enabling New Business Models**: Seamlessly implement usage-based billing by accurately tracking granular consumption per customer.
- **Actionable Per-User Insights**: Diagnose issues, understand feature adoption, and personalize experiences by analyzing data at an individual user or customer level.
- **Safe and Performant Instrumentation**: Add detailed logging to critical code paths without fear of degrading your Worker's performance or overwhelming traditional systems.

While it's important to be mindful of its limitations—such as the three-month data retention, the nuances of its sampling model, and specific data point constraints—WAE offers a compelling solution for a wide array of analytics needs. It shines particularly when combined with other Cloudflare serverless technologies like KV, D1, R2, and Queues, allowing for a holistic view of complex, distributed applications.

The journey into Workers Analytics Engine is an invitation to experiment. Start with simple use cases: track a new feature's adoption, monitor the performance of a critical API endpoint, or A/B test a change within your Worker. As familiarity grows, explore more advanced scenarios like building custom dashboards with Grafana or integrating WAE data into broader business intelligence workflows.

Cloudflare's extensive documentation and active developer community are valuable resources for further learning and troubleshooting. By leveraging Workers Analytics Engine, developers can unlock a new level of understanding and control over their serverless applications, driving innovation and delivering better experiences from the edge.

Works cited

1. Introducing Workers Analytics Engine - The Cloudflare Blog, accessed May 20, 2025, https://blog.cloudflare.com/workers-analytics-engine/
2. Workers Analytics Engine - Cloudflare Docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/analytics-engine/
3. Get started with Workers Analytics Engine - Cloudflare Docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/analytics-engine/get-started/
4. Metrics and analytics · Cloudflare Workers docs, accessed May 20, 2025, https://developers.cloudflare.com/workers/observability/metrics-and-analytics/
5. Sampling with Workers Analytics Engine · Cloudflare Analytics docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/analytics-engine/sampling/
6. Workers Analytics Engine — Pricing · Cloudflare Analytics docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/analytics-engine/pricing/
7. Example projects · Cloudflare Analytics docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/analytics-engine/recipes/
8. Usage-based billing · Cloudflare Analytics docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/analytics-engine/recipes/usage-based-billing-for-your-saas-product/
9. Querying Workers Analytics Engine from a Worker - Cloudflare Docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/analytics-engine/worker-querying/
10. Workers Analytics Engine SQL API · Cloudflare Analytics docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/analytics-engine/sql-api/
11. Workers Analytics Engine FAQs · Cloudflare Analytics docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/faq/wae-faqs/
12. Workers Analytics Engine — Limits · Cloudflare Analytics docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/analytics-engine/limits/
13. cloudflare-docs/src/content/docs/workers/platform/storage-options.mdx at production - GitHub, accessed May 20, 2025, https://github.com/cloudflare/cloudflare-docs/blob/production/src/content/docs/workers/platform/storage-options.mdx
14. Demos and architectures - Workers AI - Cloudflare Docs, accessed May 20, 2025, https://developers.cloudflare.com/workers-ai/guides/demos-architectures/

15. Tutorials · Cloudflare Workers KV docs, accessed May 20, 2025, https://developers.cloudflare.com/kv/tutorials/
16. Tutorials · Cloudflare Workers AI docs, accessed May 20, 2025, https://developers.cloudflare.com/workers-ai/guides/tutorials/
17. Choosing a data or storage product. - Workers - Cloudflare Docs, accessed May 20, 2025, https://developers.cloudflare.com/workers/platform/storage-options/
18. Self-hosted Web Analytics on a Cloudflare Worker : r/selfhosted - Reddit, accessed May 20, 2025, https://www.reddit.com/r/selfhosted/comments/1k6xisx/selfhosted_web_analytics_on_a_cloudflare_worker/
19. classroomio/analytics: Scalable web analytics you run yourself on Cloudflare - GitHub, accessed May 20, 2025, https://github.com/classroomio/analytics
20. Query a database - D1 - Cloudflare Docs, accessed May 20, 2025, https://developers.cloudflare.com/d1/best-practices/query-d1/
21. Connect to databases · Cloudflare Workers docs, accessed May 20, 2025, https://developers.cloudflare.com/workers/databases/connecting-to-databases/
22. Getting started · Cloudflare D1 docs, accessed May 20, 2025, https://developers.cloudflare.com/d1/get-started/
23. Metrics and analytics - D1 - Cloudflare Docs, accessed May 20, 2025, https://developers.cloudflare.com/d1/observability/metrics-analytics/
24. cloudflare/workers-for-platforms-example - GitHub, accessed May 20, 2025, https://github.com/cloudflare/workers-for-platforms-example
25. Making state easy with D1 GA, Hyperdrive, Queues and Workers Analytics Engine updates, accessed May 20, 2025, https://blog.cloudflare.com/making-full-stack-easier-d1-ga-hyperdrive-queues/
26. Securely access and upload assets with Cloudflare R2 · Cloudflare Workers docs, accessed May 20, 2025, https://developers.cloudflare.com/workers/tutorials/upload-assets-with-r2/
27. Use R2 from Workers - Cloudflare Docs, accessed May 20, 2025, https://developers.cloudflare.com/r2/api/workers/workers-api-usage/
28. Overview · Cloudflare R2 docs, accessed May 20, 2025, https://developers.cloudflare.com/r2/
29. Custom access control for files in R2 using D1 and Workers · Cloudflare Developer Spotlight, accessed May 20, 2025, https://developers.cloudflare.com/developer-spotlight/tutorials/custom-access-control-for-files/
30. Bindings (env) - Workers - Cloudflare Docs, accessed May 20, 2025, https://developers.cloudflare.com/workers/runtime-apis/bindings/
31. Overview · Cloudflare Queues docs, accessed May 20, 2025, https://developers.cloudflare.com/queues/
32. Cloudflare Workflows is now GA: production-ready durable execution, accessed May 20, 2025, https://blog.cloudflare.com/workflows-ga-production-ready-durable-execution/
33. Metrics · Cloudflare Queues docs, accessed May 20, 2025, https://developers.cloudflare.com/queues/observability/metrics/

34. Workers Logs - Cloudflare Docs, accessed May 20, 2025, https://developers.cloudflare.com/workers/observability/logs/
35. Tutorials · Cloudflare Workers docs, accessed May 20, 2025, https://developers.cloudflare.com/workers/tutorials/
36. Custom dashboards · Cloudflare Analytics docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/dashboards/
37. Builder Day 2024: 18 big updates to the Workers platform - The Cloudflare Blog, accessed May 20, 2025, https://blog.cloudflare.com/builder-day-2024-announcements/
38. markdembo/cloudflare_analytics_engine_demo: A full demo setup for the workers analytics engine - GitHub, accessed May 20, 2025, https://github.com/markdembo/cloudflare_analytics_engine_demo
39. Cloudflare monitoring made easy | Grafana Labs, accessed May 20, 2025, https://grafana.com/solutions/cloudflare/monitor/
40. Cloudflare integration | Grafana Cloud documentation, accessed May 20, 2025, https://grafana.com/docs/grafana-cloud/monitor-infrastructure/integrations/integration-reference/integration-cloudflare/
41. Cloudflare - Datadog Docs, accessed May 20, 2025, https://docs.datadoghq.com/integrations/cloudflare/
42. Datadog - Analytics - Cloudflare Docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/analytics-integrations/datadog/
43. Looker · Cloudflare Analytics docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/analytics-integrations/looker/
44. Splunk · Cloudflare Analytics docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/analytics-integrations/splunk/
45. New Relic - Tech Partners - Cloudflare, accessed May 20, 2025, https://www.cloudflare.com/partners/technology-partners/new-relic/
46. Enable Logpush to New Relic · Cloudflare Logs docs, accessed May 20, 2025, https://developers.cloudflare.com/logs/get-started/enable-destinations/new-relic/
47. Microsoft (Azure Sentinel) - Tech Partners - Cloudflare, accessed May 20, 2025, https://www.cloudflare.com/partners/technology-partners/microsoft/azure-sentinel/
48. Sentinel · Cloudflare Analytics docs, accessed May 20, 2025, https://developers.cloudflare.com/analytics/analytics-integrations/sentinel/
49. Re: Cloudflare app for Splunk integration with Spl, accessed May 20, 2025, https://community.splunk.com/t5/Dashboards-Visualizations/Cloudflare-app-for-Splunk-integration-with-Splunk-Cloud-Help/m-p/741868/highlight/true
50. Re: Cloudflare app for Splunk integration with Splunk Cloud. Help!, accessed May 20, 2025, https://community.splunk.com/t5/Dashboards-Visualizations/Cloudflare-app-for-Splunk-integration-with-Splunk-Cloud-Help/m-p/741866
51. Cloudflare (Preview) (using Azure Functions) connector for Microsoft ..., accessed May 20, 2025, https://learn.microsoft.com/en-us/azure/sentinel/data-connectors/cloudflare

52. Cloudflare App for Splunk - Splunkbase, accessed May 20, 2025, https://splunkbase.splunk.com/app/4501
53. Cloudflare App for Splunk | Splunkbase, accessed May 20, 2025, https://splunkbase.splunk.com/app/4501/
54. Limits · Cloudflare Workers docs, accessed May 20, 2025, https://developers.cloudflare.com/workers/platform/limits/
55. Limits & pricing · Cloudflare Workers docs, accessed May 20, 2025, https://developers.cloudflare.com/workers/ci-cd/builds/limits-and-pricing/
56. Overview · Cloudflare Workers docs, accessed May 20, 2025, https://developers.cloudflare.com/workers/
57. Top Cloudflare Workers Competitors & Alternatives 2025 | Gartner Peer Insights, accessed May 20, 2025, https://www.gartner.com/reviews/market/cloud-application-platforms/vendor/cloudflare/product/cloudflare-workers/alternatives
58. Best Cloudflare Workers Alternatives & Competitors - SourceForge, accessed May 20, 2025, https://sourceforge.net/software/product/Cloudflare-Workers/alternatives
59. Compare Cloudflare Workers vs. Serverless in 2025 - Slashdot, accessed May 20, 2025, https://slashdot.org/software/comparison/Cloudflare-Workers-vs-Serverless/
60. Developer Week 2024 | Cloudflare, accessed May 20, 2025, https://www.cloudflare.com/developer-week-2024/updates/
61. Cloudflare Workers, accessed May 20, 2025, https://blog.cloudflare.com/en-us/tag/workers/page/6/