Fine-Grained Durable WebAssembly on Cloudflare

Architecture, Recovery Semantics, and Economic Limits

Anonymous author

15 November 2026

Abstract

A serverless runtime can migrate a WebAssembly continuation without preserving its process. We describe Trace Mesh, a cross-account architecture that runs bounded guest slices in Cloudflare Containers, stores checkpoints and portable compiled traces in R2, and fences the publication of progress with leases in a SQLite Durable Object. A QEMU failure experiment exercises recovery and trace reuse, including exact 64-bit state. Local experiments measure hot trace execution and the serialization of full checkpoints. At the shipped quantum of 2,048 instructions and one tick per minute, a million-instruction sequential job takes approximately eight hours on one executor. For a job with 1 MiB of guest memory, a billion instructions create approximately 1.024 TB of retained snapshots even if memory never changes. Removing the timer bottleneck exposes the metered checkpoint path. At ten successful slices per second for a 30-day month, standard R2 checkpoint writes alone exceed a $50 budget. We give latency and cost conditions under which the design fails and identify the changes that would alter those conditions. Cloudflare prices are documented inputs to the model. Execution and serialization measurements are local. **Keywords.** WebAssembly, tracing JIT, serverless computing, durable execution, checkpointing, cost model.

1 Introduction

A client that submits a WebAssembly binary expects progress to survive process eviction, a host restart, or migration across deployment accounts. The executable state includes a guest stack and control position, whereas a serverless deployment coordinates short-lived invocations. A durable runtime must therefore preserve an application-level continuation. Trace Mesh addresses this requirement with an explicit guest state, fuel-limited execution, and an immutable checkpoint store. An optimization tier compiles recurring paths to portable Wasm traces. This makes trace reuse possible after the original executor disappears.

The central systems question is whether frequent checkpointing remains practical when control, computation, and storage use separately metered services. This work makes four contributions. First, it specifies the scheduler, checkpoint boundary, and cross-account trust model of an implemented runtime. Second, it formalizes version fencing and its relation to replay. Third, it derives throughput, retention, and price bounds for the implemented configuration and a faster event-driven configuration. Fourth, it reports a recovery experiment and a local microbenchmark that separates hot guest execution from JSON serialization. The economic claim is conditional on the size and cadence of committed state, the deployment configuration, and the operator's budget and latency requirements.

The relevant unit is a committed slice. Independent guest jobs may run concurrently. Within one sequential job, operation $i{+}1$ consumes the continuation produced by operation i. A faster trace compiler reduces the compute portion of a slice, making fixed costs for serialization, wakeup, and durable publication proportionally larger.

2 Runtime architecture

Submission and scheduling. The client submits a Core Wasm binary, an export name, and integer arguments to the coordinator over HTTPS. The coordinator Worker stores the binary in R2 under a content digest and indexes the queued job in a SQLite-backed Durable Object. A claim records an executor identity, random lease token, expiration, and current checkpoint version. The executor receives the binary, the latest checkpoint when present, and available portable trace artifacts. An executor in another account connects through the same HTTPS API. The implementation uses one coordinator Durable Object, so executor distribution leaves one authoritative coordination domain.

Execution and migration. A Python interpreter validates and executes a restricted Core Wasm profile. The supported types are `i32` and `i64`. The implementation supports structured control, direct calls, integer globals, and one memory with a 2 MiB application limit. The durable continuation records the program position, operand and call frames, locals, globals, linear memory, and metering state. Large integer bit patterns are serialized as decimal-string tags so they survive JavaScript number conversion. I/O features such as WASI imports are outside the guest profile. The interpreter supports `i64` execution. The trace compiler specializes eligible `i32`

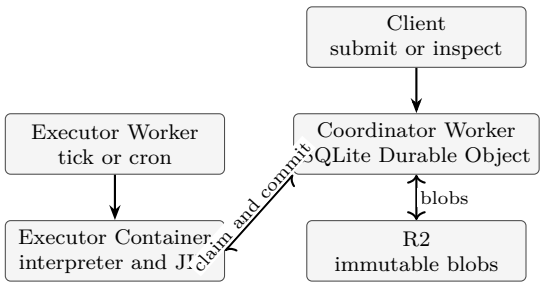


Figure 1: One coordinator authorizes checkpoint versions. The execution Container and coordinator may be in different accounts. The durable path carries guest state and portable trace artifacts.

paths, including helpers called from an `i64` frame [16, 17].

Warm Node/V8 processes cache compiled trace instances. Executors publish a portable trace plan and generated Wasm, which another executor can validate and load into its own engine. A guard failure reconstructs the exact pre-branch guest state and returns control to the interpreter. Completed traces and checkpoints encode guest-level state explicitly. Cloudflare Containers may stop or restart, and their default disk state is ephemeral [3, 4]. The durable parts of a job reside outside that disk. Fig. 1 shows the control and data paths.

Commit and dispatch. The executor sends the checkpoint and trace artifacts to the coordinator, which writes the blobs before publication. A second lease check runs inside a SQLite transaction after the R2 writes. The coordinator advances the authoritative checkpoint pointer and releases the lease in that transaction. Unpublished blobs are possible after an interrupted or stale attempt. The reference implementation retains superseded and orphaned checkpoints. R2 documents strong consistency for direct object operations, and Durable Object storage supplies the transactional boundary for pointer updates [7, 8].

The release uses one cron tick per minute per executor, or an authenticated manual tick. Each tick claims at most one slice, and the default slice budget is 2,048 guest instructions. The executor sets a 90-second inactivity timeout, so consecutive scheduled checks can keep a Container warm while the queue is empty. The 30-second default lease starts at claim, before the checkpoint and trace fetches, and covers all steps through commit. A legal instruction quantum may still expire if its elapsed time exceeds this lease. The test deployment shares one bearer credential across the coordinator API. Issuing separate submitter identities would require a gateway with job-level authorization. The analysis assumes trusted executors.

3 Recovery semantics

Write $\mathcal{S}$ for the set of validated guest configurations, s_0 for a given job's initial configuration, and δ for its interpreter step relation, restricted here to deterministic integer execution. Terminal return and trap configurations have no outgoing transitions. Given remaining fuel q, an admissible trace applied to s must yield a pair (s', j) with $0 \le j \le q$ and $s' = \delta^j(s)$, with matching terminal state and memory effects. At a guard failure, the return position is the first instruction requiring interpreter execution. The next interpreter step can then restore progress. The reference compiler's guards and reconstruction are checked by differential tests and the recovery demonstration. A general proof of the compiler is outside the evaluated artifact.

Let v denote the committed checkpoint version and ℓ the current lease identifier. A valid commit checks (v, ℓ), the assigned node, the running status, and the expiration against the same coordinator's transactional record. The commit transition raises v by one and clears ℓ. Publication is distinct from the earlier object upload.

Proposition 1 (One publication per version)**.** *At most one commit can advance a particular job from version v to version $v + 1$. If every committed slice satisfies the interpreter-prefix property, the visible checkpoints form a prefix sequence of the original interpreter run.*

Proof. Two accepted transitions from v must both read v inside the serialized transaction. The first accepted transition changes the record to $v + 1$ and clears its lease. Any subsequent transition carrying v fails the predicate. For correctness, the base checkpoint denotes s_0. Each accepted checkpoint at $v + 1$ extends the run denoted by the checkpoint at v with the interpreter-equivalent number of steps. Induction on the version gives the prefix property, including a possible terminal version. □

A failed attempt can leave its newly uploaded blob in R2, and a timeout can produce duplicated work. The publication invariant concerns checkpoints visible through the job index. It does not establish progress during service unavailability or a bound on repeated failures. Once an expired lease can be reclaimed and a later attempt completes, the unfinished slice may be replayed from the last committed checkpoint. Import-free integer execution makes this replay observable only in cost and elapsed time. Host side effects would require an explicit replay or idempotency protocol [12, 13].

4 Latency, metering, and feasible operating regions

Let a sequential guest execute $N > 0$ Wasm instructions, with a budget of q instructions in each slice. In the failure-free model it commits $k = \lceil N/q \rceil$ checkpoints, including the terminal one. Let M be the size in bytes

of its allocated linear memory, assumed constant in the examples. The present implementation serializes every byte as two hexadecimal characters. The checkpoint size is $b = 2M + \beta$, where β includes the continuation, execution counters, and JSON framing. Each subsequent committed version of this job has a different step counter, so content-addressed object names do not deduplicate its sequence of checkpoints. A failed or superseded attempt may also leave a blob outside the committed history. The calculations below omit these blobs and the nonnegative term β.

4.1 A bound for serial checkpoint dependencies

Let T be the time from the first admissible launch to the committed terminal result. With one executor doing one slice per cron event and a cron period of Δ seconds,

$$T \geq (k-1)\Delta. \tag{1}$$

This bound assumes the first launch is immediate, skips no tick, and omits the duration of the final slice. Additional executors can run independent jobs concurrently. For a single sequential job, a later slice needs the previous committed continuation. Additional executors can make more launch opportunities available, but the bound for one fixed cron schedule cannot be divided by the number of Cloudflare accounts without specifying the dispatch schedule.

For event-driven dispatch, let r be the compute-only rate in guest instructions per second. Let δ be a lower bound on the non-overlapped scheduling, restore, and commit time per slice along the critical path. Independent operations within a slice can be pipelined. The quantity δ includes only the work that remains serial after such overlap. Then

$$T \geq N/r + k\delta. \tag{2}$$

Under this model, a target relative efficiency $\eta_0 \in (0, 1)$ with respect to the compute-only execution N/r requires

$$q \geq \frac{\eta_0}{1-\eta_0}\, r\delta. \tag{3}$$

Indeed, $(N/r)/T \leq (1 + r\delta/q)^{-1}$, using $k \geq N/q$. A deadline $D > N/r$ likewise requires $k \leq (D - N/r)/\delta$ when $\delta > 0$. These are necessary conditions. A positive δ limits how much a faster trace compiler can improve whole-job efficiency at a fixed quantum.

The current executor defaults to $q = 2{,}048$, accepts at most 100,000 instructions per slice, and uses a 30-second coordinator lease by default. Under the constant-rate model, a full slice can commit before expiry only if $q/r + \delta < 30$ seconds. In practice the relevant time includes restore and commit tails. A larger quantum also increases the maximum work replayed after an interrupted uncommitted slice, from at most q instructions per failed attempt.

Table 1: Single-executor schedule and retained bytes at the default quantum with 1 MiB of guest memory. GB denotes 10^9 bytes.

Instructions	Slices	Time bound	Stored GB
10^6	489	8.13 h	1.03
10^8	48,829	33.91 d	102.40
10^9	488,282	339.08 d	1,024.00

Table 1 instantiates (1) for a 60-second schedule and 1 MiB of guest memory. Retained bytes count only the lossless memory encoding in each checkpoint. For example, 10^8 guest instructions with $q = 100{,}000$ would instead create exactly 1,000 checkpoints and at least 2.10 GB of snapshot objects, with roughly a 49-fold reduction in both checkpoint count and memory encoding volume. This alternative still needs a lower-latency scheduler for latency-sensitive work.

Equation (3) also gives a latency budget to test in a deployed experiment. At $\eta_0 = 0.8$, a workload capable of $r = 10^6$ guest instructions per second can spend at most 0.512 ms of non-overlapped overhead per default slice. An assumed 20 ms overhead would require at least 80,000 instructions per slice at that rate. At $r = 10^7$ and the same overhead, the required quantum rises to 800,000, outside the current executor's configured range. The rates and durations in this sensitivity calculation are hypotheses for deployment tests.

A separate deadline example takes $N = 10^9$, $r = 10^8$ guest instructions per second, $\delta = 25$ ms, and $D = 60$ seconds. Equation (2) permits at most 2,000 slices, hence requires $q \geq 500{,}000$. Even the largest implemented quantum of 100,000 has $k = 10{,}000$ and a bound of 260 seconds. The assumed compute rate and overhead are synthetic. They give a testable exclusion condition pending a live cross-account measurement.

4.2 Durable operation costs

Let A, R, and G be the monthly counts of R2 Class A and Class B operations and the stored volume in GB-months. Let $x_+ = \max(x, 0)$, and let $\mathcal{P}_{\mathrm{R2}}$ be the Standard storage charge in US dollars with the documented monthly allowances and rounding [6].

$$\begin{aligned}\mathcal{P}_{\mathrm{R2}}(A, R, G) = {} & 4.50\big(\lceil A/10^6 \rceil - 1\big)_+ \\ & + 0.36\big(\lceil R/10^6 \rceil - 10\big)_+ \\ & + 0.015\big(\lceil G \rceil - 10\big)_+. \end{aligned} \tag{4}$$

The storage term uses the mean of daily peak volumes. An end-of-day crossing of 10 GB of live data and a monthly consumption of 10 GB-months are distinct events.

A completed job contributes at least k checkpoint writes and $2k - 1$ reads in the shipped protocol. The coordinator loads the module at every claim and loads the prior checkpoint after the first slice. Cached trace fetches, status requests, the uploaded module, and retries can add

operations. The job also makes at least $2k$ coordinator requests for claim and commit. Workers and Durable Objects meter those requests under their respective products, and Durable Objects may incur duration and SQL storage charges [1, 9]. These costs accrue to the coordinator account. Container activity is metered in the execution accounts [2].

A throughput target of 10 new successful commits per second for 30 days generates 25.92 million distinct checkpoint objects. If one R2 account holds these checkpoints and has its entire included allowance available, the 24.92 million Class A writes over that allowance round up to 25 million billable writes. Checkpoint writes alone cost $112.50, so a $50 monthly control-plane budget is infeasible at this target under the shipped protocol. This holds with immediate reclamation of old checkpoints and with zero prices for reads and execution. The threshold concerns the number of durable commits, independent of the guest's instruction mix. More generally, a monthly control-plane budget B can cover at most $10^6(1 + \lfloor B/4.50 \rfloor)$ such commits under this checkpoint-write lower bound. For $B = \$50$ this means at most 12 million per 30-day period, or about 4.63 per second continuously. Other charges and preexisting usage reduce the budget available for checkpoints.

As a sensitivity case, take 100 independent jobs with $N = 10^9$, $q = 2{,}048$, and $M = 1$ MiB and assume event-driven dispatch completes them within one billing period. Assume distinct checkpoint histories across jobs, Standard storage, and no competing use of the included allowances. The cohort produces 48,828,200 checkpoints. Checkpoint writes and baseline reads alone incur at least $216.00 and $31.68 in R2 operation charges in the creation month. The memory encodings occupy 102,400.16 GB. If that cohort remains stored throughout a subsequent month, its minimum storage charge is $1,535.87 for that month under (4). This omits creation-month storage, Durable Objects, Workers, Containers, and all overhead bytes. These figures describe two different billing windows and require no assumption about the shape of the creation month's storage ramp.

For a sustained unique snapshot arrival of at least $V > 0$ GB each month with zero reclamation, the accumulated volume from prior months grows by at least V per billing period. Holding n complete prior cohorts throughout a month gives a storage charge of at least $0.015(nV - 10)_+$ dollars. Any fixed positive operating budget is eventually exceeded. This follows from monotone retention even if compute, scheduling, and operation costs are set to zero. A retention policy changes the conclusion by bounding the number of live checkpoint histories. A dirty-page scheme changes write amplification if individual unchanged pages are reused. For example, a 1% or 10% dirty fraction of 1 MiB emits roughly 10 or 102 KiB of changed memory if the fractions are already measured at the stored-page granularity, before manifests and compaction. These savings describe a different storage layout and leave the number of durable commits intact.

Table 2: Local warm execution. Times are medians in ms, with eleven samples per cell.

Guest instructions	Interpreter	Warm JIT + IPC
2,048	1.727	0.429
16,384	14.471	0.675
100,000	91.504	0.790

5 Measurements

Failure and migration. The experiment ran separate Linux executors under QEMU x86-64 TCG with a host-side HTTP scheduler. Executor A reached version 1 after 196 guest instructions and published one hot-loop trace. A subsequent executor process was killed while holding the next lease. The coordinator paused and resumed the job, clearing the stale lease while retaining version 1. Executor B restored a mixed `i32/i64` continuation, used the shared trace for 2,810 guest instructions, took a guard side exit, and committed the terminal state. The run ended at 3,017 instructions with the exact unsigned result 9223373287854703674. The full terminal VM state matched interpreter execution. The test also checked cancellation and rejection of stale commits. A separate base experiment exercised timeout-based lease recovery. These tests demonstrate portable state and recovery in QEMU. They provide no estimates of Cloudflare network latency, CPU throughput, or billed usage.

Warm execution and snapshot encoding. The benchmark used a Linux x86-64 host reporting an Intel Xeon Platinum 8573C, Python 3.12.14, and Node 24.22.1 with V8 13.6. The guest is an integer accumulator with a 12-instruction hot cycle. Its generated trace is 425 bytes. For each instruction budget, eleven samples measured Python interpretation and an already-warm V8 trace, the latter including local process IPC. The benchmark checks exact state equivalence per trial. The timed region excludes interpreter and JIT setup, recording, compilation, network activity, and checkpoint serialization. Table 2 reports medians. These results characterize a loop with unusually effective tracing.

The checkpoint benchmark separately times JSON encoding and decoding with guest memory represented as the same hexadecimal string used by the runtime. It also times one SHA-256 digest of the encoded blob. Memory construction, HTTP, R2, authorization, and Durable Object transactions are outside these timed segments. At 1 MiB of guest memory the checkpoint is 2,097,491 bytes and its local encode-plus-decode medians sum to 10.634 ms, roughly 25 times the warm 2,048-instruction trace time. At 2 MiB, the serialized checkpoint is 4,194,643 bytes. Table 3 separates the three stages. Composing local medians illustrates relative magnitude and cannot

Table 3: Local full snapshots. Times are medians in ms.

Guest memory	JSON bytes	Encode	Decode	Digest
0 B	339	0.013	0.012	0.001
64 KiB	131,411	0.444	0.239	0.094
1 MiB	2,097,491	6.383	4.251	1.746
2 MiB	4,194,643	12.768	7.782	3.305

establish an end-to-end latency percentile.

6 Pricing consequences and design alternatives

Container use and the fixed subscription. As of the study date, Workers Paid starts at $5 per account per month, and the Containers allowances cover 25 GiB-hours of memory, 375 vCPU-minutes and 200 GB-hours of disk. The Basic instance provisions 1 GiB of memory and 4 GB of disk and permits one-quarter of a vCPU. A continuously awake instance reaches the monthly memory allowance after 25 hours and the disk allowance after 50 hours. At sustained full quarter-vCPU usage it also reaches the CPU allowance after 25 hours. RAM and disk use are billed by provisioned size while awake, and compute by active CPU use [2, 1]. Over 720 awake hours in an otherwise unused account, these rates yield approximately $11.93 at negligible CPU use and $24.44 at continuous quarter-vCPU use, including that account's subscription. The interval excludes other services and transfer. The account bill may be quite modest even when a job's time to completion is poor.

At ten successful slices per second, claim and commit generate at least twenty coordinator Durable Object requests per second. In an otherwise unused account the one-million-request monthly allowance lasts at most 13.9 hours. At least one checkpoint write per slice consumes the one-million Class A allowance in at most 27.8 hours. An executor limited to once-a-minute cron attempts at most 43,200 slices in 30 days, so the dispatch schedule slows the consumption of request allowances. These figures locate the start of metered overage. Hard capacity limits and separately configured limits govern rejection [9, 6, 1].

Storage retention and transfer. At the margin after allowances, retaining a 2 MiB checkpoint for one month costs about $0.00003146 under the published R2 Standard storage rate. The fully utilized Basic Container costs approximately $0.028008 per extra awake hour, including its quarter-vCPU, RAM and disk. The retained object therefore costs about four seconds of saturated Basic runtime in this comparison. This comparison uses marginal rates. Actual monthly R2 storage charges use the average of the daily peaks and rounding at the account level [6, 2]. Checkpoint writes and storage are independent expense terms.

R2 lists no egress charge for its own outgoing transfers, and Containers publish regional outbound allowances and overage rates. Their classifications apply to different services. The executor sends its checkpoint to a coordinator HTTPS endpoint before R2 writes it. The billing classification of that hop was not observed. If the route is metered as North America or Europe Container egress after its allowance, the listed $0.025/GB adds about $0.00005243 per 2 MiB checkpoint after the inclusion is exhausted. Ten such checkpoints per second for 30 days emit about 54.36 TB before headers or retries. If all of that outbound volume lands in one such billing bucket with 1 TB included, the illustrative egress overage is about $1,334. The R2 write bound is independent of this estimate. The egress estimate is conditional on classification, billing region and account placement [2, 6].

Feasible changes. Removing periodic polling from the critical path allows a submitted job or completed slice to dispatch the next wakeup at once. A durable dispatch intent plus periodic reconciliation would cover a lost notification or expired lease. An at-least-once delivery service can wake existing Workers without changing the publication invariant [10]. A larger execution quantum reduces the number of checkpoint barriers, at the expense of more lost compute on eviction and the need for a defensible lease bound. A warm actor could run many in-memory quanta per durable checkpoint. At that point scheduling fuel and the durability interval become separate parameters.

Binary encoding would remove hexadecimal's factor of two. Dirty-page checkpoints can reduce effective bytes further when a workload changes a small part of memory, provided each immutable manifest has a complete and published chain of dependencies. Periodic compaction bounds reconstruction cost. Reachability-based garbage collection can remove superseded checkpoints only after accounting for in-flight publication and retained audit roots. Trace and module fetches can also be amortized across sticky execution on a warm node. These changes address different terms in the cost model. They do not remove the possibility of lost work between durable checkpoints.

7 Related work and scope

A formal semantics of serverless execution must account for instance reuse, termination, and retries. Jangda et al. characterize when these behaviors can be abstracted away by a simpler semantics [11]. Burckhardt et al. prove a corresponding abstraction for durable workflows whose progress is recovered by record and replay [12]. Trace Mesh places the durable boundary inside the guest machine and serializes an explicit continuation. This accommodates instruction-level suspension for its supported Wasm profile at the expense of a state transfer on every committed slice. Beldi extends log-based fault

tolerance to transactional operations, inter-function invocation, and garbage collection [13]. Its treatment of external effects and retention illustrates how much a production implementation would have to add to the prototype's deterministic guest model.

The cost of moving intermediate state has been studied independently of checkpoint recovery. Pocket provides an elastic storage tier for ephemeral intermediate data in serverless analytics [14]. ExCamera demonstrates that fine-grained cloud execution can be effective when the algorithm exposes substantial parallel work and limits the work on the serial path [15]. These results delimit the present claim. Trace Mesh assigns each committed successor to one serial lease chain per job. Independent jobs can exploit horizontal capacity. A single job's recovery checkpoints still have to cross the critical path in version order.

Validity and alternatives. The provider prices and limits describe the public plan at a recorded date. The latency and billing models describe stated workload regimes. They leave cold starts, cross-account transfer latency, and overload behavior for measurement in a paid deployment. Local timing measures the implemented software on the test host. Cloudflare latency and billing classification of the executor-to-coordinator hop require provider measurements. R2 egress and Container network egress are separate pricing categories. If the latter applies to that hop, its regional allowance and marginal rate must be included with the reported byte volume [6, 2].

Checkpointing fewer times trades lower coordination cost against a longer replay window. An incremental checkpoint design could preserve frequent fencing with less transferred state when writes are sparse. It would also need an atomic manifest, a recovery policy for chains of deltas, and garbage collection. A resident executor can hold an in-memory continuation between checkpoints and amortize local JIT initialization. Cloudflare's beta Container snapshots preserve files tied to one image version. Recovering a guest from such a snapshot would still require serialized Wasm state and an explicit resumption step [5]. Each alternative changes the assumptions behind the cost model. The observed and derived results apply to the evaluated full-snapshot, single-coordinator design in the stated workload regimes.

8 Conclusion

The implementation demonstrates that compute-process lifetimes can be decoupled from WebAssembly job lifetimes. Its current fine-grained configuration fails practical latency targets by construction, and simply dispatching more frequently can push checkpoint writes beyond a modest budget before other service charges. Full memory snapshots and indefinite retention can make storage scale with elapsed instruction count even when useful guest memory is unchanged. The appropriate feasibility claim is a region defined by checkpoint size, cadence, recovery precision, compute rate, retained history, and account allowances. Coarser durable boundaries or small state can fall inside that region. The endpoint for a high-performance design is to make that region wider and verify it with live, itemized Cloudflare measurements.

Artifact availability

The accompanying source archive contains the manuscript, the measured timing samples, the benchmark script, and the QEMU verification outputs. It also includes the exact Trace Mesh runtime release used in this analysis. Its SHA-256 digest is 92f7d0671057cdc5b5c2b983e00492e2 35633a9113e23b9812f09cc54adc9746. The source archive includes a published-price calculator and instructions for reproducing the local tests.

References

[1] Cloudflare. Workers pricing. https://developers.cloudflare.com/workers/platform/pricing/. Accessed November 15, 2026.

[2] Cloudflare. Containers pricing. https://developers.cloudflare.com/containers/platform/pricing/. Accessed November 15, 2026.

[3] Cloudflare. Lifecycle of a Container. https://developers.cloudflare.com/containers/concepts/architecture/. Accessed November 15, 2026.

[4] Cloudflare. Containers frequently asked questions. https://developers.cloudflare.com/containers/faq/. Accessed November 15, 2026.

[5] Cloudflare. Use snapshots. https://developers.cloudflare.com/containers/guides/snapshots/. Accessed November 15, 2026.

[6] Cloudflare. R2 pricing. https://developers.cloudflare.com/r2/pricing/. Accessed November 15, 2026.

[7] Cloudflare. R2 consistency model. https://developers.cloudflare.com/r2/reference/consistency/. Accessed November 15, 2026.

[8] Cloudflare. SQLite-backed Durable Object storage. https://developers.cloudflare.com/durable-objects/api/sqlite-storage-api/. Accessed November 15, 2026.

[9] Cloudflare. Durable Objects pricing. https://developers.cloudflare.com/durable-objects/platform/pricing/. Accessed November 15, 2026.

[10] Cloudflare. Queues delivery guarantees. https://developers.cloudflare.com/queues/reference/delivery-guarantees/. Accessed November 15, 2026.

[11] A. Jangda, D. Pinckney, Y. Brun, and A. Guha. Formal foundations of serverless computing. *Proc. ACM Program. Lang.*, 3(OOPSLA), Article 149, 2019. https://doi.org/10.1145/3360575.

[12] S. Burckhardt et al. Durable Functions: Semantics for stateful serverless. *Proc. ACM Program. Lang.*, 5(OOPSLA), Article 133, 2021. https://doi.org/10.1145/3485510.

[13] H. Zhang, A. Cardoza, P. B. Chen, S. Angel, and V. Liu. Fault-tolerant and transactional stateful serverless workflows. *14th USENIX Symposium on Operating Systems Design and Implementation*, 2020, 1187–1204. https://www.usenix.org/conference/osdi20/presentation/zhang-haoran.

[14] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, and C. Kozyrakis. Pocket: Elastic ephemeral storage for serverless analytics. *13th USENIX Symposium on Operating Systems Design and Implementation*, 2018, 427–444. https://www.usenix.org/conference/osdi18/presentation/klimovic.

[15] S. Fouladi et al. Encoding, fast and slow: Low-latency video processing using thousands of tiny threads. *14th USENIX Symposium on Networked Systems Design and Implementation*, 2017, 363–376. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi.

[16] WebAssembly Community Group. WebAssembly specification, Core 3.0. https://webassembly.github.io/spec/core/. Accessed November 15, 2026.

[17] A. Gal et al. Trace-based just-in-time type specialization for dynamic languages. *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2009, 465–478. https://www.cs.williams.edu/~freund/cs434-exemplar/gal-trace.pdf.