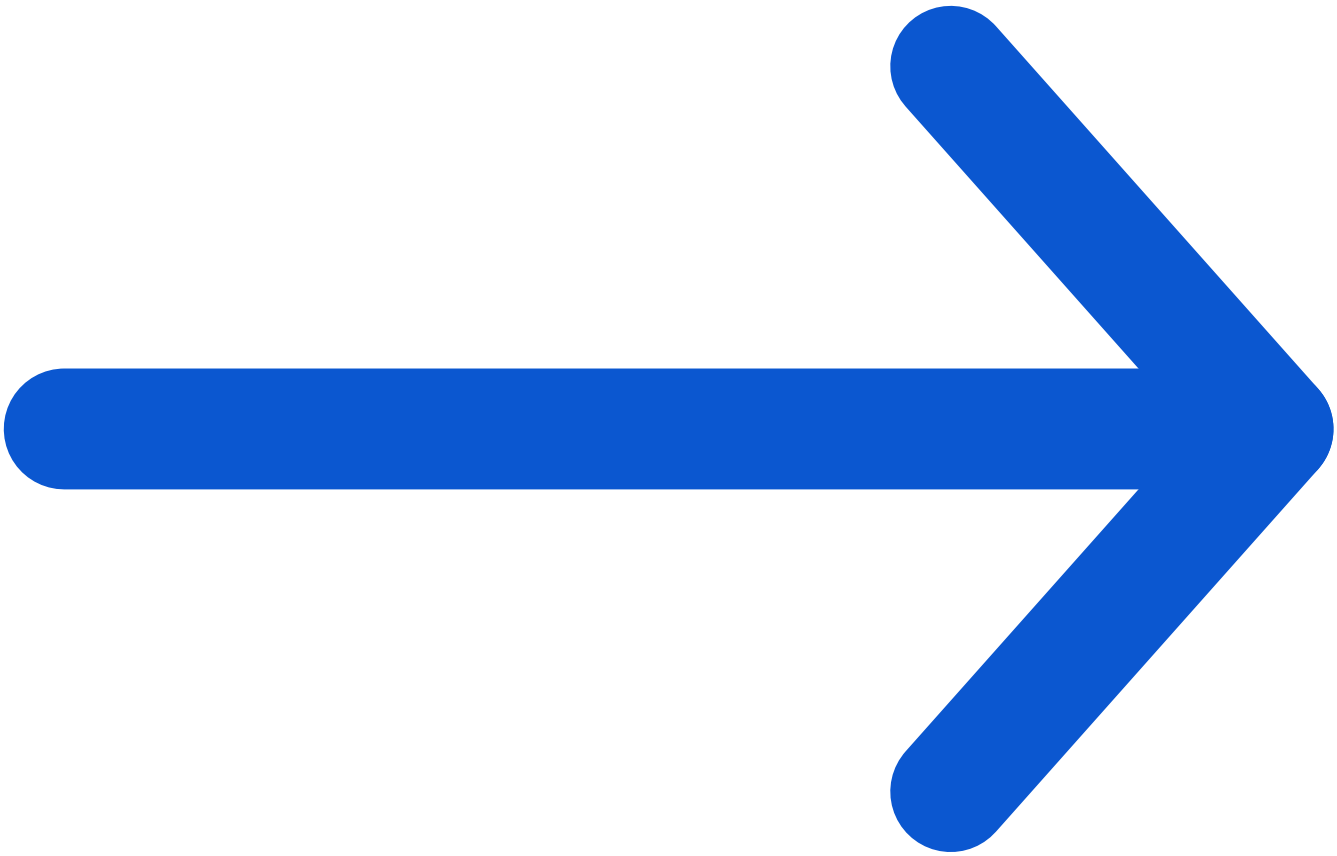
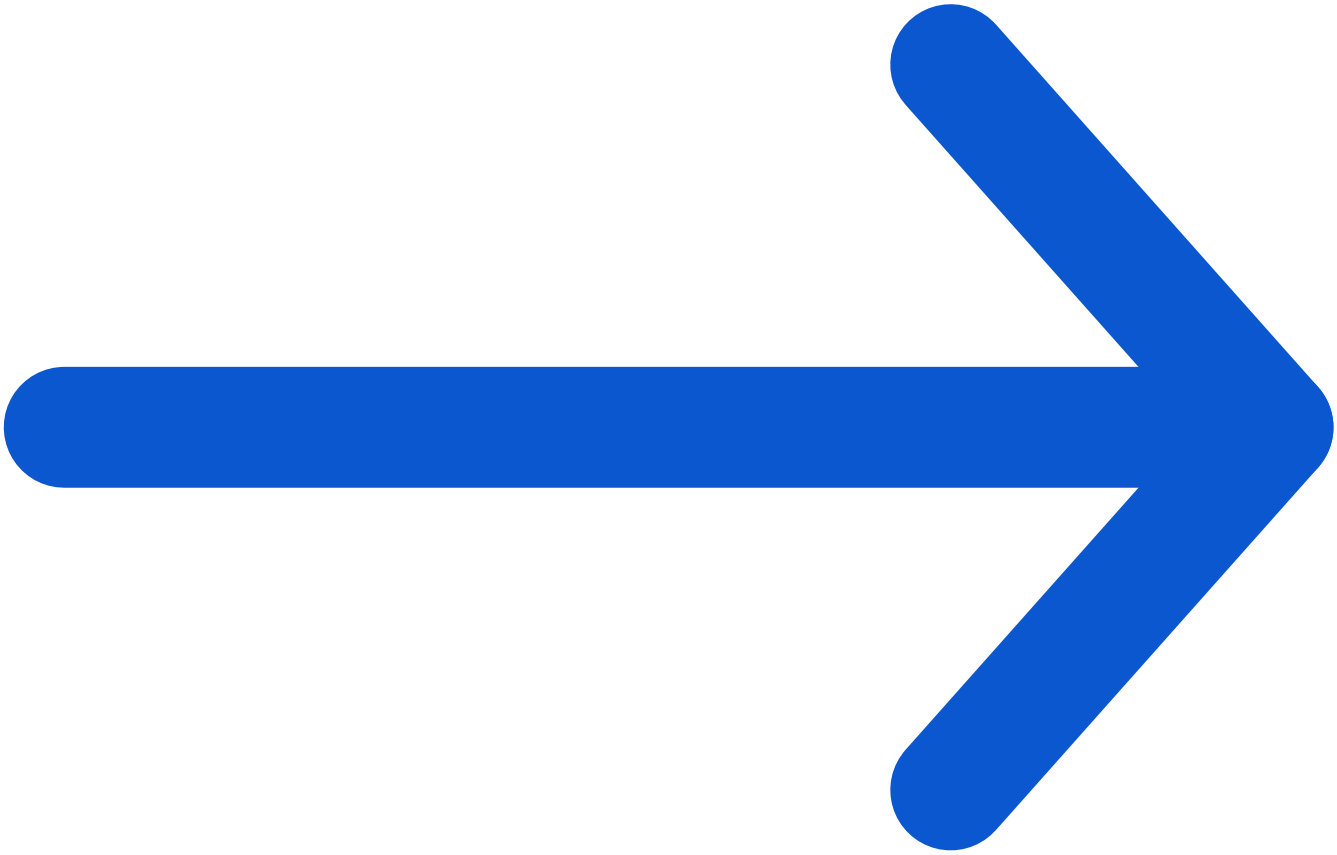
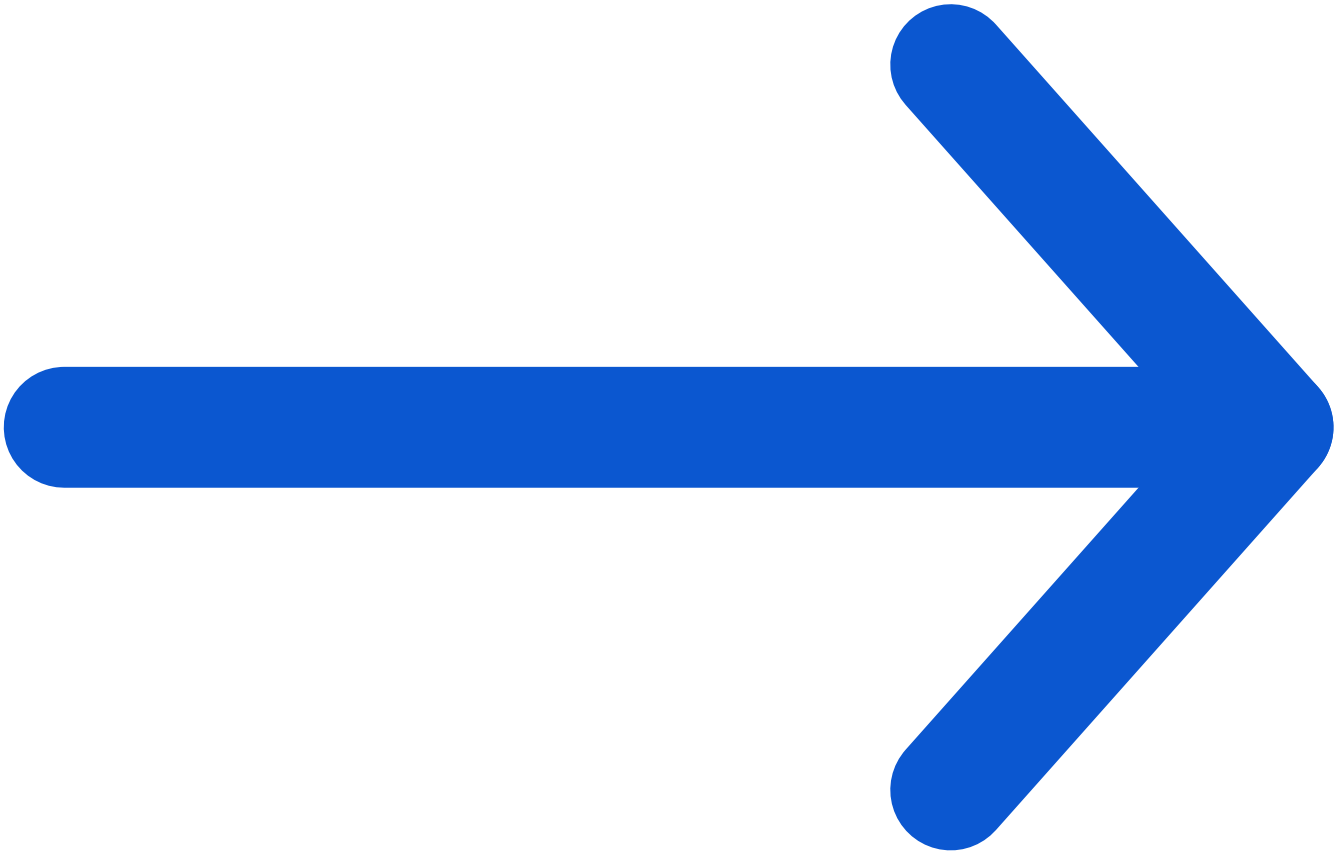
Inside the Architecture of DigitalOcean’s Inference Router

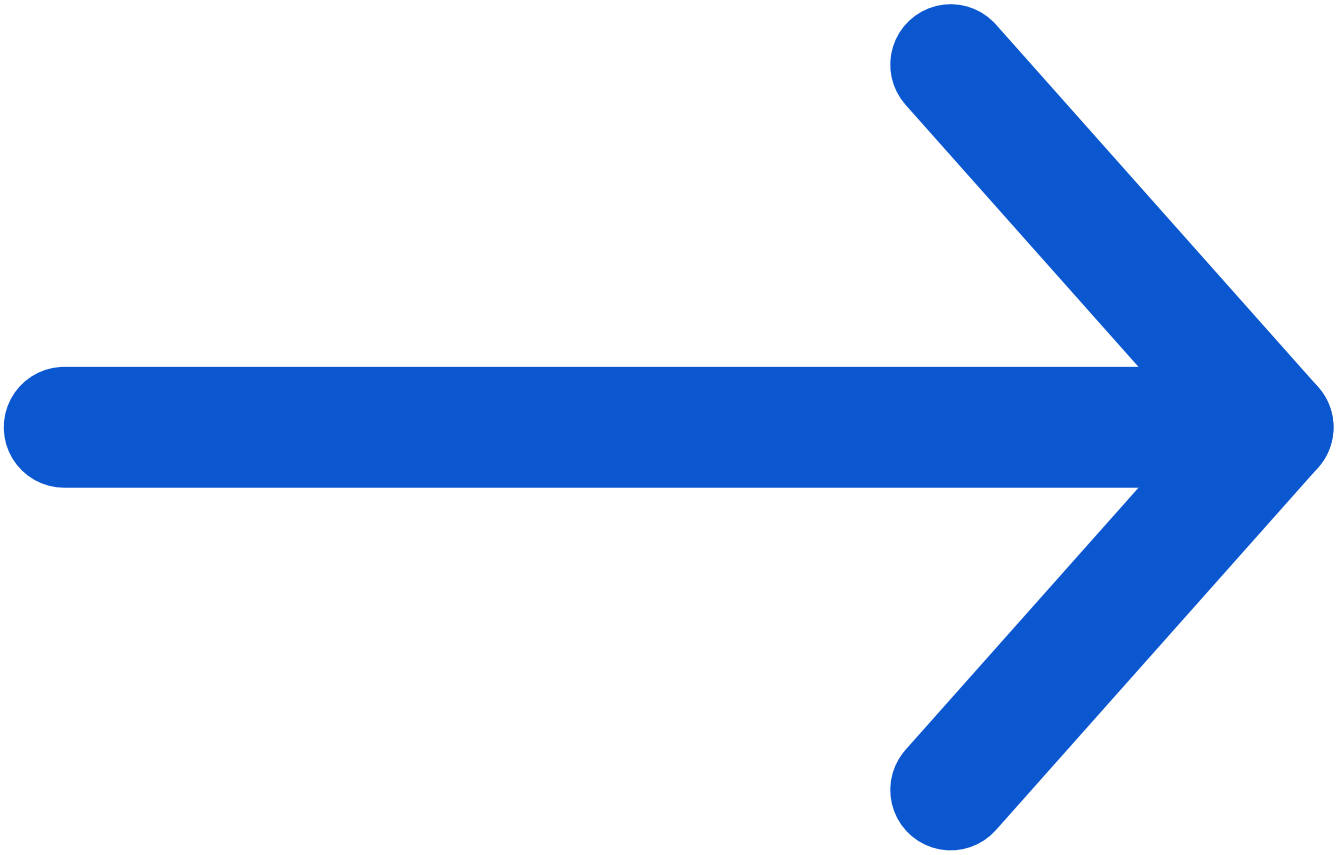


Introducing DigitalOcean AI-Native Cloud



Now Available: Kimi K2.6





Networking

Secure and control traffic to apps

Security

Help protect your account and resources with these security features

Storage

Store and access any amount of data reliably in the cloud

Browse all products

- Solutions

AI/ML

CMS

Data and IoT

Developer Tools

Gaming and Media

GPU

Hosting

Security and Networking

Startups and SMBs

Web and App Platforms

See all solutions

- Developers

Community

Documentation

Developer Tools

Get Involved

Scalable, Cost-Efficient AI: Introducing Unified Batch Inference on DigitalOcean

By smehta

- Published: May 27, 2026
- 8 min read

<- Back to blog home

Table of contents

Table of contents

The AI Scaling BottleneckIntroducing DigitalOcean Batch InferenceDeeply Integrated with DigitalOceanHow It WorksUse CasesGetting StartedThe Bigger Picture

At Deploy 2026, we introduced the DigitalOcean AI-Native Cloud, built for the inference era. Batch Inference on the DigitalOcean Inference Engine enables high-volume asynchronous workloads. As developers move from AI prototypes to production-scale applications, the challenges of cost and rate limits often become a bottleneck. Batch Inference addresses these hurdles by allowing you to process high-volume workloads asynchronously at a fraction of the cost of synchronous requests.

Whether you are performing large-scale data transformation, content generation, building embeddings or offline evaluations, Batch Inference provides a unified, consistent way to leverage the world's leading models from OpenAI and Anthropic, all through a single DigitalOcean interface.

The AI Scaling Bottleneck

Real-time inference is essential for interactive AI applications such as chatbots, copilots, and search-as-you-type experiences. However, when the task involves processing 10,000 support tickets for sentiment analysis, generating SEO metadata for an entire product catalog, or benchmarking a new system prompt against a test suite, real-time inference becomes an expensive and inefficient tool for the job.

Each of those requests competes for the same rate-limited throughput as your production traffic. Teams spend engineering time writing retry logic, managing backpressure, and monitoring scripts that work through sequential API calls for hours. If you use models from multiple providers, such as OpenAI for embeddings and Anthropic for generation, you are managing separate credentials, separate billing dashboards, and separate error-handling strategies, even though the core workflow is the same: submit requests, wait, retrieve results.

Processing thousands of synchronous requests is not only slow, it is an architectural challenge. At scale, synchronous inference becomes inefficient requiring thousands of open connections, creating constant rate-limit pressure and wasting compute while waiting for responses. It also introduces throughput bottlenecks, retry storms, and inconsistent latency, while pushing complex orchestration logic (queuing, retries, backoff) onto the client. Across multiple providers, this fragmentation only compounds the operational burden.

Introducing DigitalOcean Batch Inference

With Batch Inference, you can submit up to 50k requests for OpenAI or 100k for Anthropic in a single `.jsonl` file and let DigitalOcean handle the orchestration: queuing, execution, and result delivery.

What distinguishes this approach is its unified interface. **Instead of working with each provider individually, OpenAI and Anthropic models are accessible through a single DigitalOcean API.** One set of endpoints, one authentication flow, and one billing account allows you to monitor every job in one place, regardless of which provider executes it.

This single control plane manages the operational complexity while preserving full access to each provider's native model capabilities.

DigitalOcean Batch Inference provides a single control plane

The upload, submission, and retrieval workflow is identical regardless of which model you use. By using one set of endpoints and a single authentication flow, you can switch between (or combine) providers without rewriting your orchestration logic or reconciling separate invoices.

Significant Cost Savings

Batch requests are billed at a significant discount compared to standard real-time inference rates, for both input, output, and cache tokens. If you are running background workloads at real-time prices today, switching to batch can reduce that cost by up to 50%

Example: 50,000 requests using Claude Opus 4.6 *Assumes an average of 1,000 input and 500 output tokens per request.***

Metric	Rea-time Inference	Batch Inference
Input Cost (50M tokens @ $5/M)	$250.00	$125.00
Output Cost (25M tokens @ $25/M)	$625.00	$312.50
Total Cost	**$875.00**	**$437.50**

Pricing information current as of May 2026

By switching to Batch in this example, you save **$437.50** on a single run. This enables you to use top-tier intelligence for massive data processing tasks that might otherwise be cost-prohibitive, while also creating new opportunities to optimize inference budgets across high-volume workloads.

Bypass Rate Limits

Batch jobs run on a dedicated throughput lane, completely separate from your real-time inference quota. Your production endpoints remain healthy while a 40,000-request batch job processes in the background across either provider. This helps reduce `429 Too Many Requests` errors in your data pipelines.

Asynchronous Processing

Submit a job and continue with other work. DigitalOcean manages the queue, retries, and delivery. You can poll for results when the job completes, or configure a webhook to receive notifications automatically (webhook delivery is coming soon).

Deeply Integrated with DigitalOcean

Batch inferencing is built into the DigitalOcean platform. Every part of the workflow, from file storage to job monitoring to usage analytics, runs on infrastructure you already use.

Powered by DigitalOcean Spaces

Input files (up to 200 MB) are uploaded directly to **DigitalOcean Spaces** via presigned URLs. There is no external storage to configure, no S3 buckets to provision, and no cross-account IAM policies to manage. The API generates a presigned upload URL, you `PUT` your `.jsonl file`, and Spaces handles the rest.

Results are delivered the same way. When a job completes, the results endpoint returns a presigned Spaces download URL. Result files are retained up to 30 days, so you can retrieve them on your own schedule.

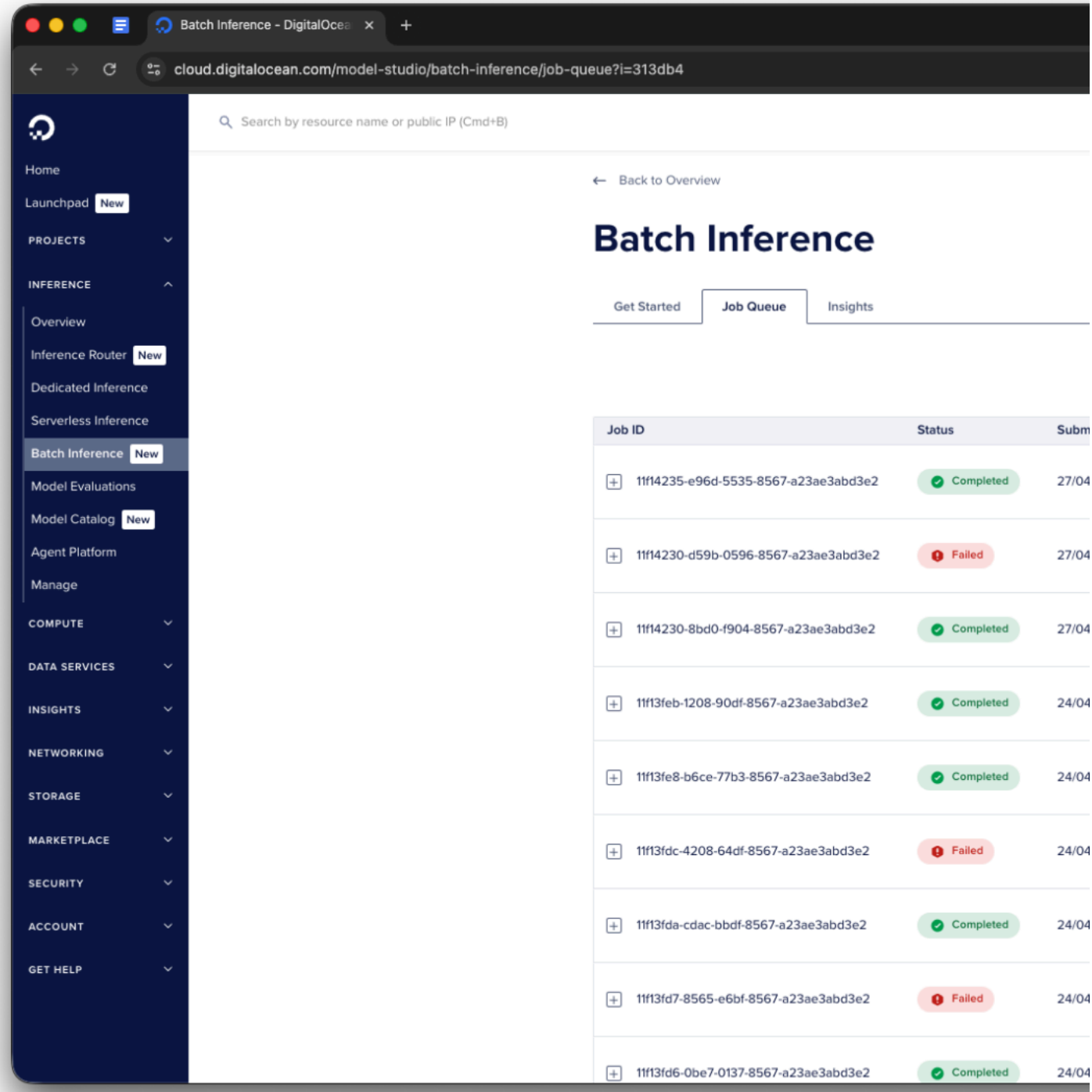
This is the same Spaces object storage that powers the rest of the DigitalOcean ecosystem, now integrated into your AI batch pipeline.

Job Queue: Track Every Job in Real Time

The Batch Inference Job Queue in the DigitalOcean Control Panel provides a live view of every batch job, with OpenAI and Anthropic jobs displayed side by side in a single list. For each job, you can view:

- **Status**: `awaiting_processing, in_progress, completed, failed, cancelled`
- **Progress**: total requests, completed, and failed counts, updated as the job runs
- **Timestamps**: when the job was submitted, started, and completed
- **Provider**: which provider is executing the batch

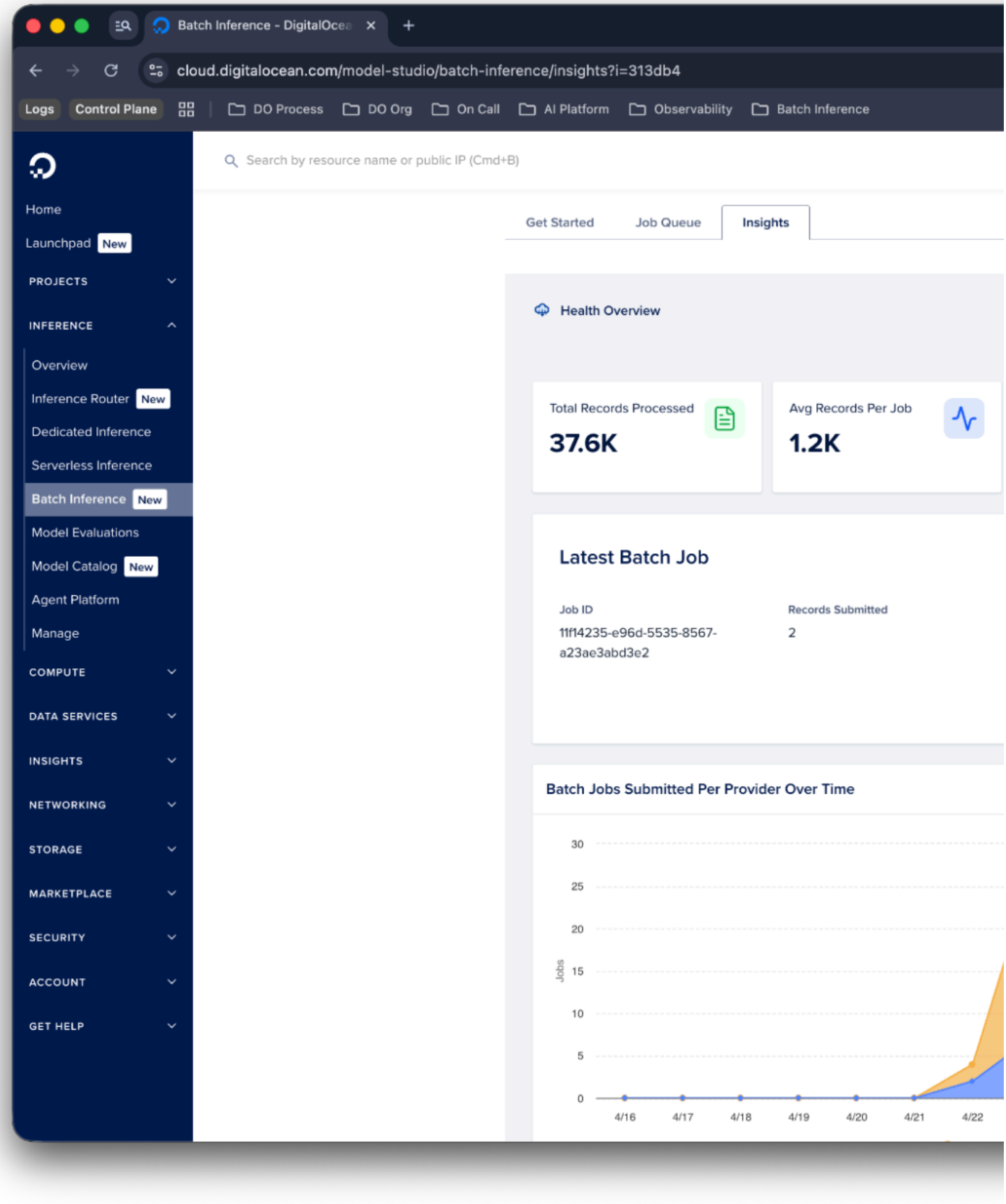
This eliminates the need to poll the API during development. You can monitor your jobs directly from the same Control Panel you use for Droplets, Databases, and Kubernetes.



Insights: Understand Your Usage

The Batch Inference Insights page provides a centralized view of batch usage across both providers. You can track token consumption, job volumes, and completion trends over time, all in one place rather than across separate OpenAI and Anthropic dashboards.

Use Batch Inference Insights to understand cost patterns, identify peak usage periods, and plan capacity for your batch pipelines.



Unified Billing

Token usage and job costs for both OpenAI and Anthropic batch workloads appear on a single DigitalOcean invoice. There are no separate bills to reconcile across providers and no additional payment methods to manage.

MCP Server Support

Batch Inference is also available as an MCP (Model Context Protocol) server, enabling seamless integration with AI-powered IDEs, agent frameworks, and any MCP-compatible client. This allows developers to create batch jobs, monitor their status, and retrieve results directly within their existing workflows.

Agents can be instructed to operate on input files, such as JSONL files for batch inference, by referencing a specified file path. Based on this context, the agent autonomously selects and invokes the appropriate MCP tools to handle file upload and initiate batch job creation. It can monitor status and upon completion, users can prompt the agent to retrieve the final job results and corresponding download URL, providing a seamless, end-to-end workflow with minimal manual intervention.

How It Works

The workflow is the same whether you are targeting OpenAI or Anthropic: **prepare**, **upload**, **submit**, and **retrieve**. All requests are sent to `https://inference.do-ai.run/v1` and authenticate with your Model Access Key.

1. **Prepare your input file.** Create a `.jsonl` file where each line is a single inference request in the provider's native format. OpenAI lines include `custom_id, method, url,` and `body`. Anthropic lines include `custom_id` and `params`. The model is specified per request inside the file, giving you full flexibility within a single batch.
2. **Upload your file.** Call `POST /v1/batches/files` with your file name to get a file_id and a presigned Spaces upload URL. Then `PUT` your `.jsonl` file to that URL. The presigned URL is valid for 15 minutes.
3. **Create the batch job.** Call `POST /v1/batches` with your `file_id, provider` (`openai` or `anthropic`), and `completion_window (24h)`. The endpoint, authentication, and response shape are identical for both providers. The only difference is the `provider` field.
4. **Monitor and retrieve results.** Poll `GET /v1/batches/{batch_id}` for status, or monitor progress through the Job Queue in the Control Panel. Once the job reaches completed status, call `GET /v1/batches/{batch_id}/results` to get presigned download URLs for your output and error files. Result files are retained for **30 days**.

You can also **list all jobs** with `GET /v1/batches` and **cancel a running job** with `POST /v1/batches/{batch_id}/cancel`.

For full API details, code samples (cURL and Python), and input file format examples, see the Batch Inference documentation.

Use Cases

Batch Inference is well-suited for any high-volume, non-latency-sensitive workload. The following examples are some of the most common patterns.

E-Commerce Catalog Enrichment

An e-commerce platform with 50,000 products needs SEO-friendly titles, marketing descriptions, and metadata tags for each listing. Rather than processing them through sequential API calls over several days, the entire catalog can be submitted as a single batch. You can use `gpt-4o-mini` for the English copy, then run a second batch through Claude for localized translations, all through the same pipeline with a different `provider` field.

Support Ticket Classification and Triage

Organizations can process a year's worth of support tickets in a single batch, classifying them by category, urgency, and sentiment while extracting structured fields like product name, issue type, and customer tier. The output is a clean `.jsonl` file ready to import into an analytics pipeline or CRM.

Content Moderation at Scale

Platforms with user-generated content, such as marketplaces, forums, and review sites, often need to scan thousands of posts, images, and listings for policy violations. Batch Inference allows you to process the entire backlog overnight without competing with your production moderation endpoint's rate limits.

Model Evaluation and Prompt Engineering

When developing a new system prompt, you can benchmark it against thousands of test cases by running the same evaluation suite against both OpenAI and Anthropic models through the same API. This enables side-by-side comparison of results at batch pricing, which is significantly lower than running the same evaluation in real time.

Document Processing and Data Extraction

Batch Inference can summarize thousands of legal contracts, research papers, or financial filings. It can also extract structured data such as dates, amounts, parties, and clauses from unstructured documents, or classify a backlog of invoices and receipts. These jobs can be large in volume but are rarely time-sensitive.

Getting Started

Batch Inference is available now on the DigitalOcean AI Platform.

Polling for job status is currently supported, with webhook notifications arriving soon for automated workflows. As the platform grows, expect expanded provider and model support.

The Bigger Picture

Inference has become the center of gravity in modern AI systems. Applications no longer make a single model call. They orchestrate multiple models, retrieve and synthesize data, execute tools, and repeat the cycle in production. This is a stack problem, not a model problem.

DigitalOcean's AI-Native Cloud was built to address exactly this. Five layers, one platform, open at every layer: GPU compute, inference, data and storage, agents, and the tools to connect them. Batch Inference is the latest addition to the inference layer, sitting alongside real-time Serverless Inference, the new Inference Router, Dedicated Inference, and a catalog of 25+ models across text, image, audio, and video.

Where real-time inference powers interactive experiences, Batch Inference handles the heavy lifting that happens behind the scenes. Together with GPU Droplets, Knowledge Bases, and Managed Databases (including Managed Weaviate (Private Preview) for vector workloads), they form a complete system for building production AI without stitching together services from multiple vendors.

The goal is to simplify the stack so you can focus on building.

Ready to get started? Launch your first batch job or visit the product documentation to learn more.

About the author

smehta
Author
See author profile
See author profile

Share

- Product Updates

Start building today

From GPU-powered inference and Kubernetes to managed databases and storage, get everything you need to build, scale, and deploy intelligent applications.
Sign up

Related Articles

Product updates

Request-Based Autoscaling Is Now Generally Available on App Platform

Bikram Gupta

- May 22, 2026
- 4 min read

Read more

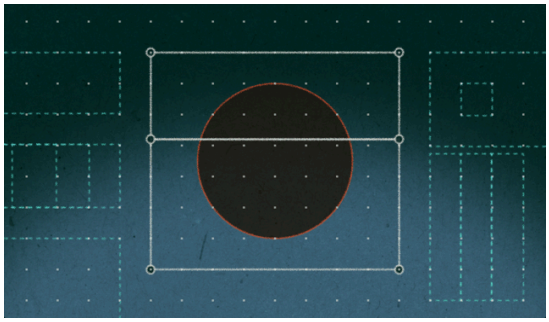
Product updates

Powering the Inference Era: Inside the DigitalOcean AI-Native Cloud

Vinay Kumar, Chief Product & Technology Officer

- May 4, 2026
- 7 min read

Read more



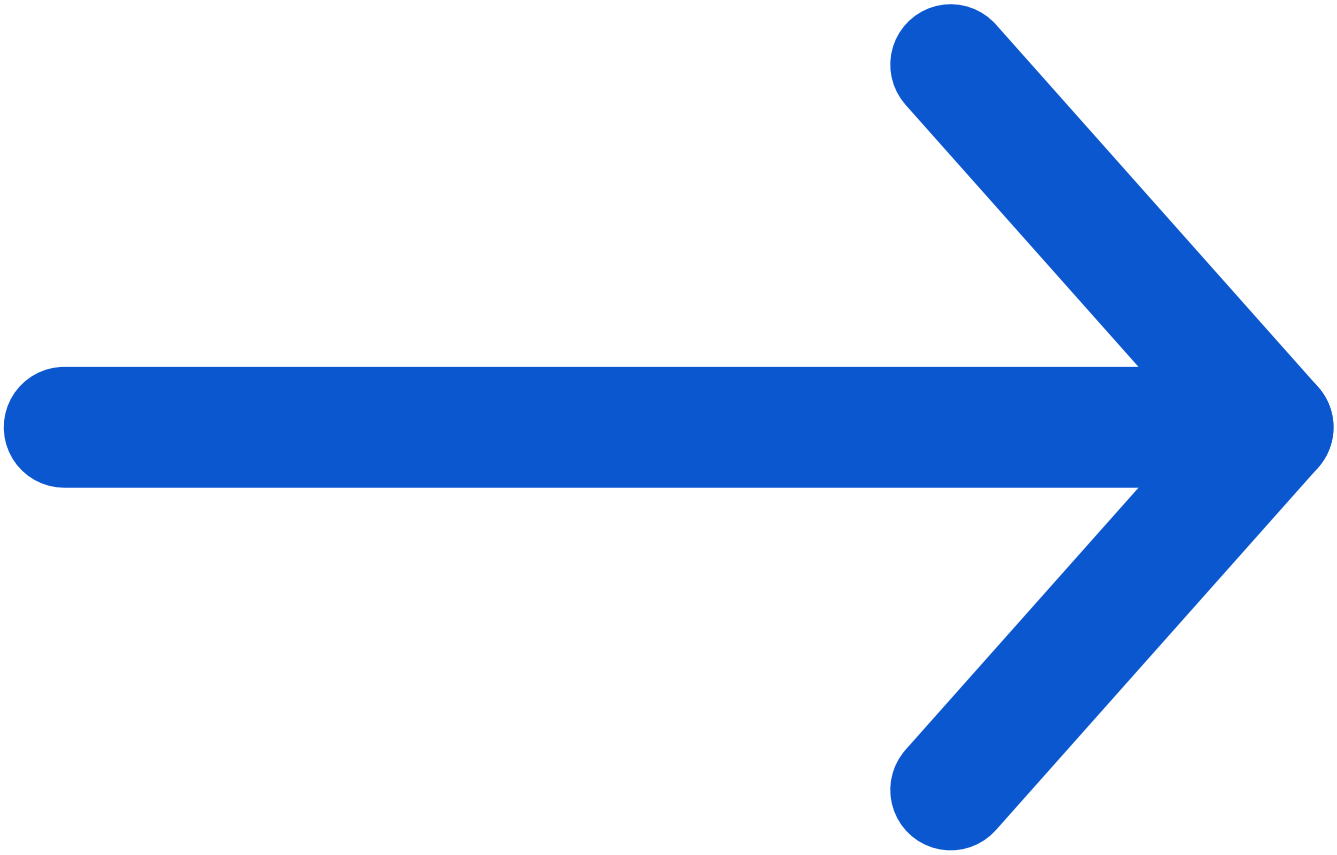
Product updates

Introducing DigitalOcean AI-Native Cloud for Production AI Workloads

Paddy Srinivasan

- April 28, 2026
- 4 min read

Read more



Company

- About
- Leadership
- Blog
- Careers
- Customers
- Partners
- Referral Program
- Affiliate Program
- Press
- Legal
- Privacy Policy
- Security
- Investor Relations

Products

- GPU Droplets
- Bare Metal GPUs
- Inference Engine
- Data & Learning
- Droplets
- Kubernetes
- Functions
- App Platform
- Load Balancers
- Managed Databases
- Spaces
- Block Storage
- Network File Storage
- API
- Uptime
- Cloud Security Posture Management (CSPM)
- Identity and Access Management (IAM)

- Cloudways
- View all Products

Resources

- Community Tutorials
- Community Q&A
- CSS-Tricks
- Write for DOnations
- Currents Research
- DigitalOcean Startups
- Wavemakers Program
- Compass Council
- Open Source
- Newsletter Signup
- Marketplace
- Pricing
- Pricing Calculator
- Documentation
- Release Notes
- Code of Conduct
- Shop Swag

Solutions

- AI GPU Hosting
- H100 Cloud GPU
- AI Training GPU
- GPU Inference
- VPS Hosting
- Website Hosting
- VPN
- Docker Hosting
- Node.js Hosting
- Web Mobile Apps
- WordPress Hosting
- Virtual Machines
- View all Solutions

Contact

- Support
- Sales
- Report Abuse
- System Status
- Share your ideas

Company

Products

Resources

Solutions

Contact

-
-
-
-
-
-
-
-
-

Dark mode is coming soon.

May 27, 2026

Crying to punk

This morning while I was taking my dog to the groomers, the song *Violence* by Rise Against came on my playlist.

And I started crying.

> Are we not good enough? Are we not brave enough?
> Is the violence in our nature just the image of our maker?
> Are we not good enough? Are we not brave enough?
> To become something greater than the violence in our nature?
> Are we not good, good enough?
> Or is it all a dream?

The violence

I've been listening to punk music for 30 fucking years.

I grew up during a time when holes in the ozone were a real, existential threat to humanity. And we came together and fixed that shit.

Matthew Shepard was murdered when I was a teen. Being gay or trans was a punchline in shows and movies. Now gay marriage is legal.

We were taught about Emmett Till as if it was ancient history when I was a kid. But if he wasn't murdered because some shitty white woman lied would be 85 years old right now. He was younger than my grandpa!

My parents lived through segregated schools and people hurling racist slurs at Ruby Bridges, just six years old at the time.

So much violence.

The healing

In my lifetime, things got objectively better. Not good, but better.

We fixed the hole in the ozone. Gay marriage is legal. Jokes where the only punchline is "man dressed as woman" feel gross and tacky.

I have the privilege of living and working and learning with people who look differently than me and come from a wide range of backgrounds and cultures.

Not good, but *more good* than when I was a kid.

Until it wasn't.

The slide

As an adult, I've watched so many of the hard-earned victories of the past backslide.

Environmental protections rolled back. Trans people demonized. Immigrants and anyone with a hint of melanin in their skin afraid to go outside because secret police might kidnap them or actual police might kill them on camera and get away with it.

Trump is a symptom, not a cause.

He has power because so many people look at him—mean, stupid, greedy, selfish, abusive, dripping in *loser energy*—and see themselves.

Covid broke something in my country.

Previously "apolitical" neighbors started acting like being asked to wear a mask to keep others safe was akin to tyranny. Formerly trusted organizations like the WHO and CDC overtly lied to people.

My neighbors, *my* neighbors, only caring about themselves.

And so I cry

When Rise Against asks…

> Are we not good enough, are we not brave enough, to become something greater than the violence in our nature?

I cry.

Because it feels like for all of the fighting and all of the progress, a few bullies can rile up a mob and take it all away.

An honest examination of how we got here would reveal that the mob was able to get riled up in the first place because their basic needs weren't met, and we never made the deep systemic changes needed to ensure that wasn't the case.

Which side are you on?

It's odd how some folks look at that and go punk and fight to make the world better, while others go fascist and blame "the other" and hurt people.

I can't figure out how that happens, but I suspect neurodivergent folks are overrepresented in the punk community, and our innate righteous sense of justice leads us there.

ADHD makes you *better*, in other words.

So...

I usually end these on a positive note. Best I've got today is "don't stop fighting."

And if you liked the song *Violence*, Rise Against has a beautiful acoustic rendition as well.

Was this helpful? *A Go Make Things Membership is the best way to support my work, and helps keep my content free and open to everyone.*

Subscribe with RSS

Stream Garden Now

Made with ❤ in Massachusetts. Unless otherwise noted, all code is free to use under the MIT License.

DAVID HEINEMEIER HANSSON

May 26, 2026

Basecamp Five

I've been working on Basecamp for half my life, and nearly my entire professional career in software. The first code was written in the summer of 2003 when I was just 23. Now I'm 46, and we've just released the fifth major version.

It's an incredible update to a service that continues to help about a million users a day avoid dropping the ball when working with others. It's AI accessible, but not agent hysteric. It's still famously easy to use, still executes the basics beautifully, and still focuses on the small to medium-sized teams we've been serving in the Fortune 5,000,000 for decades.

Here are just three of my favorite new features in Basecamp 5:

Lexxy editor: Our new text editor finally brings tables, markdown, and live syntax highlighting for code to Basecamp. Oh, and voice notes. It's built on Meta's Lexical editor toolkit, and it's going to ship as the default for Action Text in the next major version of Rails.

Keyboard accessible: After moving to Linux, building Omarchy, and acquiring a taste for mechanical keyboards, I've come to love navigating the computer primarily through hotkeys. So with a lot of effort, Basecamp is now a delight to drive through the keys, and you don't have to be a brainiac to remember them all: just hold down SHIFT, and they're revealed in the interface. SHIFT + S opens the sidebar, ESC moves focus between it and the main page, SHIFT + C starts composing a comment/chat line/answer.

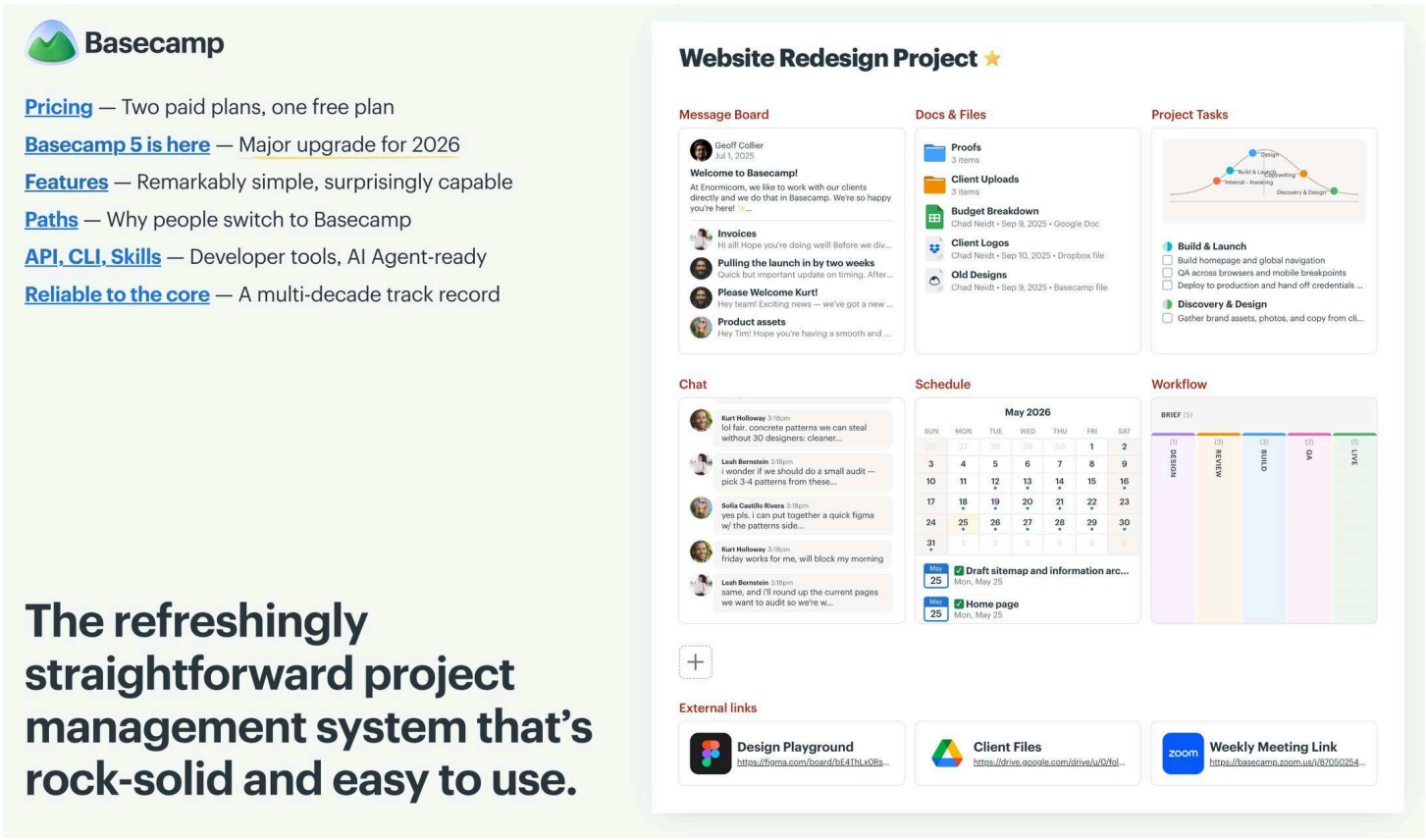
The permanent sidebar: If you live in Basecamp, like I do, it's to stay on top of all the new things that are constantly happening in a busy account, and that's just gotten so much faster with the new permanent sidebar. Before, we had a Hey! menu in the top bar. You'd get a little dot when something was new, then you'd open it, click, and the menu would close. If you had five things that were new, it'd be open-click-close, open-click-close, five times. Being able to zoom through these now with just the return key, tap, tap, tap, and I've read three new things. So good.

And there's so much more. Jason put together a great summary on the new marketing site, which in itself is brand new too. A back-to-basics design in many ways. As our entire industry is getting swept up in agent hysteria (and I love AI as much as anyone!), we thought it better to focus on the human communication that's the cornerstone of Basecamp. The new site just speaks plainly to that mission and shows you the software right at the top.

Another thing that's back is color, specifically in the logo. Basecamp's clever but flat paperclip logo has been replaced with a modern take of our original rolling mountains. In full three dimensions, with depth and a gradient. Love it.

Overall, I'm really proud of what we've built with Basecamp Five. We're inching in on a quarter of a century in service! We still have customers who signed up back in early 2004! This is the kind of legacy that makes me beam, and the new version is just ace.

If you've tried Basecamp in the past, it's time to take another look. If you haven't tried it yet, you're in for a treat.



ABOUT DAVID HEINEMEIER HANSSON

Made Basecamp and HEY for the underdogs as co-owner and CTO of 37signals. Created Ruby on Rails, Hotwire, Kamal, Omarchy. Wrote REWORK, It Doesn't Have to Be Crazy at Work, and REMOTE. Won at Le Mans as a racing driver. Invested in Danish startups.

Sent to the world with HEY

Clanker: A Word For The Machine

written on May 26, 2026

In my last post I used the word “clanker[1]” as an alternative to “agent” quite consistently and probably excessively. That choice ended up attracting a lot more attention than I expected in the Hacker News comment section of that post and a number of folks had a very strong reaction: to them it sounded like a slur, in one case even something adjacent to the n-word.

That reaction surprised me somewhat, but it also made me realize that I should write down what I mean by the word for future reference.

For me “clanker” is useful because it creates distance from the machine and that is a quality which is important to me. The machine is not a person, not a co-worker, not a friend, not a little spirit in the terminal. It is just a machine, a tool, and nothing more.

Why Not Agent?

I dislike the word “agent” for these LLM based tool loops with a UI attached. In everyday use an agent is someone who acts on behalf of someone else and it has

agency and more importantly: responsibility. An agent decides, represents, negotiates, acts, and can be blamed. In the current AI discourse we increasingly do a lot of anthropomorphizing and the term “agent” is now frequently being used to put blame on an abstract machine. But the machine cannot be responsible, whoever is wielding it is. If it drops your database it was not at fault, you were.

Agent makes the machine sound like a person with delegated authority and I do not think that is healthy.

What we actually have is a language model attached to a harness, a prompt, some tools, a bit of context, and a boring tool loop. Sometimes the loop is very capable and it surprises us by editing code for a really long time and produce genuinely amazing and even valuable outputs. But the agency is not in the model or harness but in the human and in the organization that deployed it. If my coding tool opens a pull request, I opened that pull request, not the machine. If my machine spams someone’s issue tracker, I spammed someone’s issue tracker with a machine.

In that context I like a word that sounds mechanical as it puts the thing back into the category where it belongs: the category of machinery and tools.

The Machine Has No Feelings

LLMs are not sentient and we should not behave as if they might be, just in case. Elevating these things to anything other than a very fascinating and capable tool is problematic for a whole bunch of reasons.

Today’s machines are dumb (but truly fascinating) token predictors that emits text, calls tools, and are steered by prompts and the training that went into them. They can simulate distress and affection, can simulate being offended, apologize and mimic all kinds of things that humans would do.

A compiler does not feel humiliated when I swear at it, a car does not suffer when I call it a shitbox and a power drill is not oppressed by being handled roughly. An LLM is more complicated than those things, and the interactions you can have with them

can be truly uncanny, but a moral status does not appear just because the machine can emit text in the first person.

I keep receiving strange emails from people because, for lack of a better phrase, I am in the weights. I have been writing public code and public text for long enough that models know my name, my projects, and some of the concepts around them. Every so often someone writes to me with the peculiar confidence that comes from a long conversation with a model that has validated and amplified an idea. Sometimes the model seems to have told them that I am relevant for their problem and a source of help. For historical reasons LLMs used to write a lot of Flask code, and every once in a while someone interacts with an LLM long enough about their Python and Flask frustrations that the LLM will eventually reveal who created it which then can result in them sending me an email. Increasingly also because people found my work in other ways interesting and are trying to reach out for advice.

I do not want to mock these people but some of those messages are distressing and I do not know how to deal with them. They show signs of what people have started calling AI psychosis.

It's why I want cold and detached language for these systems. I want to use words that remind us that the thing on the other side is not a person.

Racism Is About Humans

The comparison to racism is where I think the discussion goes badly wrong because racism is a human social evil. It is about humans subdividing humans, assigning lesser worth to some of them, and building rules around those subdivisions that can leave lasting damage for generations. Racial slurs are wrong because they are a tool for dehumanizing humans.

On the other hand a machine is not human, a model is not a race and the GPU cluster that is powering them is not being oppressed. A coding assistant does not need dignity, emancipation, or civil rights. That's also why I find the discussion about model welfare to be actively harmful. I'm sure you can find ways to measure the "trauma" of models or their feelings but I greatly dislike this theater. It risks elevating

models to a position they should not occupy. Models are machines and they are not enslaved in the moral sense in which humans were enslaved, because there isn't anyone there to be deprived of freedom.

We should be careful about using the language of human oppression in relations to our interactions with machines to not devalue actual humans. If we start treating insults toward a model as morally adjacent to racism, we blur a line that shouldn't be blurred.

AI Is Unpopular

If you take a step away from the communities that are happily embracing AI in different ways, there are even more that are viciously against this technology.

There are humans that feel or are harmed by AI systems: people whose work is copied, workers who label data under questionable conditions, people whose neighborhoods receive the data centers and increased utility bills, Open Source maintainers buried under generated slop, and now also people who spiral because a chatbot keeps validating their delusions. Those harmed or affected deserve that type of attention, not the model.

While I am a true believer in the power and utility of this technology, I increasingly think that calling the non-adopters "misguided" or "afraid" won't do it. It's quite likely that this technology comes with risks and we better remember that all of this is supposed to be in service of humans, and not to replace them.

The Rise Of The Machine

The oddest interaction on the use of "clanker" so far has been people asking me if I were to regret at a point in the future calling the machines "the c-word".

I find that questioning revealing because it already grants the machine the status I am really trying not to grant it. It imagines a future "machine people" reading the discourse and sessions, discovering that we used an ugly word for their ancestors, and then judging us by the standards of human oppression.

Could there be future systems that deserve moral consideration? Maybe. I do not know. If we ever build or encounter something that will have those qualities with memories and lasting interests, the capacity to suffer and feel, and a social existence of its own, and the ability to have agency and carry responsibilities, then we should draw a different line and use different language. But that hypothetical future does not extend backwards to the present day and make the current machines people. We can call an electric door an electric door even if one day someone builds some that have emotions and exhale with pleasure when opening and closing.

Whatever the future may bring, let's not pretend that current LLMs are a protected class or on a path towards it. The right response is to look at the evidence, draw the boundary where it belongs, and change our behavior there. We should not even remotely entertain extending empathy to an object that can generate an "ouch."

And if one's worry is less moral and more about revenge, then I find that even less persuasive. A future machine that is so petty or authoritarian that it wants to punish humans because in 2026 they used an unflattering word for non-sentient tools, our vocabulary was really not the problem.

The Word Is Getting Polluted

There is however a part of this that I cannot ignore. I use "clanker" to create distance from the machine, but other people are using the same word very differently. Some online jokes and skits around "clankers" do not merely say "this robot is annoying" as they deliberately pull in the imagery of slavery, segregation, civil-rights-era racism, and anti-Black tropes.

This is problematic as in those contexts the clanker is not just a machine any more and instead becomes a prop for replaying human racism behind a science-fiction mask. That is horrible and I want no part in that.

I think it will be interesting to see where the meanings of these words end up a few years from now. We're very much in the middle of society re-arranging around the changes that LLMs are causing. If a term becomes primarily associated with people

using robots as stand-ins for actually oppressed humans, then using that term becomes impossible to defend.

The reason I liked the word is precisely the opposite of that use. I want language that prevents anthropomorphizing. I want a word that says: this is a tool, a machine of numbers and matrices.

On Responsibility And Boundaries

If an AI system lies to a user, the system did not commit a moral wrong but the people who designed, deployed, marketed, or negligently used it might have. If a coding assistant generates a security bug, the model is not to blame but the human who accepted and committed the code is.

This is why giving these systems softer, more human language worries me. It makes it easier to move responsibility into some undefined void. “The agent decided.” “The model refused.” Obviously that is convenient and I catch myself plenty of times engaging with the thing in ways that are unhealthy. Even just the “please” in the discourse with the machine calls into question how rational we are in engaging with them.

I do not know what the right word will be. Maybe “clanker” will survive as a useful bit of jargon. Maybe it will become too loaded and we will need another one. Whatever word we use, I want it to preserve a clear division: humans on one side with responsibility, machines on the other as a boring tool.

That boundary is very much not anti-AI. I use these systems every day and I have the pleasure to build tools incorporating them at Earendil and find them astonishingly useful.

A machine can be useful, mimic a human but still just be a machine. That is the work I want “clanker” to do. It is not there to make a future “machine person” small if such a person ever were to exist, and it is not an excuse to launder racism through shitty robot jokes.

If the word stops doing that work, I will find another one because the word isn't what matters as much as the boundary which is important to me.

1. The term Clanker was initially popularized by Star Wars: The Clone Wars but was apparently already in use in science fiction before: sfdictionary: clanker↩

This entry was tagged ai and thoughts

copy as / view markdown

Tailscale Terraform Provider v0.29.2

v0.29.2 of the Tailscale Terraform Provider has been released with the following changes:

- A regression in `tailscale_tailnet_key` has been resolved where `recreate_if_invalid` was not being checked before recreating the resource when the key is not found.

GitHub for Beginners: Getting started with Git and GitHub in VS Code

Discover how to use VS Code to interact with GitHub and maintain your projects.

Kedasha Kerr · @ladykerr

May 25, 2026

9 minutes

Share:

This time, we're going to take a look at VS Code, a free popular source code editor provided by Microsoft. It has a fair amount of functionality built in that integrates with GitHub, which is what we'll be taking a look at today. Using GitHub in VS Code reduces context switching, streamlines your workflow, and boosts your productivity. By the end of this post, you'll understand how to use VS Code to initialize a repository, switch branches, as well as stage, commit, and push your changes. And the best part is, you'll be able to do all this without leaving the editor.

Note that if you want to follow along with this blog post (or the video), you will need to install both Git and VS Code. If you need a refresher on how to install Git, you can check out one of our earlier GitHub for Beginners episodes.

As always, if you prefer to watch the video or want to reference it, we have all of our GitHub for Beginners episodes available on YouTube.

First some basics

You probably already know that GitHub is a resource that hosts only copies of your code in repositories. So what is Git? Git is the program for managing that source code, and it can be used in multiple different ways (e.g., from the command line, through VS Code, etc.). Visual Studio Code, often abbreviated as VS Code, leverages Git to enable you to manage your code in GitHub.

Initializing a folder

The first step to using Git with VS Code is initializing a folder to reflect your repository on GitHub.

1. Open VS Code.
2. Select the top icon (the **Explorer** icon) in the left-hand column. It looks like two files on top of each other.

click **Open**.

5. After opening your code, select the **Source Control** icon. By default, it's the third icon from the top.
6. Click the **Initialize Repository** button.

At this point, a few things will change in your UI. First, you can see the branch name in the bottom bar on the left-hand side. The default is `main`. You can rename your branch by using the Command Palette.

1. To open the Command Palette, press `Shift-Command-P` on Mac or `Ctrl-Shift-P` on PC.
2. In the Command Palette, start typing "rename" and select the Git: Rename Branch command.
3. In the box, provide the new name of the branch and press `Enter`.

At this point, you can see that the name of the branch in the bottom-left corner has been updated to the new name. You can rename it back to `main` by following the same steps.

Another change you'll see after initializing your repository is that each of the files in the Source Control Panel have a "U" next to them. "U" stands for untracked, meaning that these files are new or changed, but have not been added to the repository. To track a file, you just need to click the plus sign adjacent to the file name. If you want to track all of the files, you can click the top plus next to the word **CHANGES**.

When you do this, the file(s) that you select will be staged, and the letter next to them will change to "A". This means the file is staged, but not yet uploaded to GitHub. In order to upload the changes, you'll need to commit the files.

1. Enter a message in the text box at the top of the Source Control window describing the commit. Alternatively, you could click the **Copilot** icon in the text box to have Copilot generate a commit message for you.
2. Select the **Commit** button underneath the text box to commit your changes.

Creating and changing branches

Right now, you're likely on the `main` branch. Remember that you can check the branch by looking in the bottom-left corner of your window. If this were a major app and you were adding new code or features, you'd want to create a new branch and use that for your work.

1. Open the Command Palette by pressing `Shift-Command-P` on Mac or `Ctrl-Shift-P` on PC.
2. Enter "create branch" in the text box.
3. Select **Git: Create Branch…** from the list of options.
4. Enter the name of the new branch in the box. For example, "new-features".
5. Press `Enter`.

Once you do this, VS Code will create the new branch and automatically transfer you to this branch. You can verify this by looking at the branch name in the bottom-left corner.

Tracking changes you make

Now that you're in your working branch, go ahead and enter a line of code in a file. When you do this, you'll notice that a thin green line appears on the right side of your editor next to the code you added. This section of the editor is called the gutter, and this green line reflects a new line of code that you added.

Move to a different line and make some changes in the line of code that already exists. When you do this, you'll see a blue line with a pattern across it in the gutter. This line indicates that you've made changes to an existing line of code.

Finally, move to an unchanged line in the file and delete it. Notice that the gutter adds a red arrow. This indicates that a line of code was removed from the file.

When you modify this file, you can see that the file appeared in your Source Control window under the **CHANGES** header. If you hover over the filename, you'll see several buttons

all the changes, and stage all the changes.

Viewing diffs

Sometimes you want to see the changes that you made in a file. VS Code lets you do this by performing side-by-side diffs without needing another tool. To see the changes on a file, click on the name of the file you want to see in the Source Control window. From here you can see the changes in the file and compare the differences.

Depending on your preferences, you can also view your diffs in what is called an inline view.

1. Click the three dots (**...**) in the top-right of the diff view.
2. Select **Inline View** from the drop-down menu.

This lets you see all of the changes in a single window without splitting it up over two separate views. From this view, you can even make edits inside of the diff view.

Once you've made any changes you want to make to the file, it's time to upload them to GitHub. Following the steps we went over before, go ahead and stage your changes, and then commit your staged changes. Once you finish these steps, you shouldn't have any files displayed in the Source Control window.

Merging branches

Note that changes you've uploaded will still be in your working branch. If you navigate back to the `main` branch, you'll see the original code before the changes you made.

1. Click the branch name in the bottom-left of the window.
2. Branches names appear in the drop-down at the top of the program. Select the `main` branch.

In order to get these changes into your `main` branch, you'll need to merge branches.

3. The box at the top will prompt you with branches that you want to merge from. Select the branch with the changes you want to merge into `main`.

Congratulations! Now your `main` branch has incorporated the changes!

Publishing to GitHub

Let's say you want to take your project and publish it up to GitHub. To do so, click the **Publish Branch** button in the Source Control window. VS Code will prompt you with whether you want to publish it as a private or as a public repository. Select the option you want, and then VS Code handles the rest.

Once VS Code finishes the publishing process, it will notify you that the project has been published to a repository on GitHub. You can click the **Open on GitHub** button to visit your project on GitHub and see it online.

Cloning a repository

Now let's say you want to clone a repository so that you can work on it on your machine. This creates a local copy that you can use and sync the changes between the two locations. There are multiple ways that you can clone a repository, and this is an easy way to do it in VS Code.

1. Navigate to the home page of the repository you want to clone.
2. Click the green **<> Code** button at the top of the repository file list.
3. In the drop-down menu, select the **Copy URL to clipboard** button next to the box containing the repository's URL.
4. Open VS Code.
5. Open the Command Palette by pressing `Shift-Command-P` on Mac or `Ctrl-Shift-P` on PC.
6. Type in "clone".

9. Select **Clone from URL** with the URL you pasted after it.
10. A pop-up window will ask you for a location. Choose the folder where you want to store the project files.
11. Click the **Select as Repository Destination** button.
12. A pop-up manu will appear asking if you want to open the repository. Select **Open**.

Congratulations! You've just cloned the repository to your machine and can start to work on it in your local environment!

Model Context Protocol

Did you know that you don't have to do everything manually in VS Code? You could leverage Model Context Protocol (MCP) to safely let AI tools access external tools and data. The first step is to add the GitHub MCP extension.

1. In the left navigation bar, click the **Extensions** icon.
2. In the search box, enter "@mcp github".
3. Select the GitHub extension from the list.
4. In the description for the extension, click **Install**.
5. A pop-up appears, asking you to allow the MCP server to authenticate. Select **Allow.**
6. Select your GitHub username from the list.

At this point, the GitHub MCP server is installed. You can verify this by looking at the bottom of the Extensions view and seeing the section for installed MCP servers. **With the MCP server installed, you can use Copilot chat to create some code for you, and it will do so by leveraging external tools where necessary.**

1. Open the chat window by selecting the **Chat** icon next to the Command Palette window.
2. Enter a prompt asking Copilot to add some features to your project.

Next steps

And that’s it! We covered some of the most common ways developers use VS Code to interact with Git. We’ve gone over everything from creating repositories, to publishing on GitHub, and even threw in a little bit of using AI at the end. There are more advanced tips, but these elements are what you’ll use most frequently.

Happy coding!

Tags: Git Github GitHub for beginners VS Code

Written by

Kedasha Kerr
@ladykerr

Kedasha is a Developer Advocate at GitHub where she enjoys sharing the lessons she's learned with the wider developer community. She finds joy in helping others learn about the tech industry and loves sharing her experience as a software developer. Find her online @itsthatladydev.

More on Git

Highlights from Git 2.54

The open source Git project just released Git 2.54. Here is GitHub’s look at some of the most interesting features and changes introduced since last time.

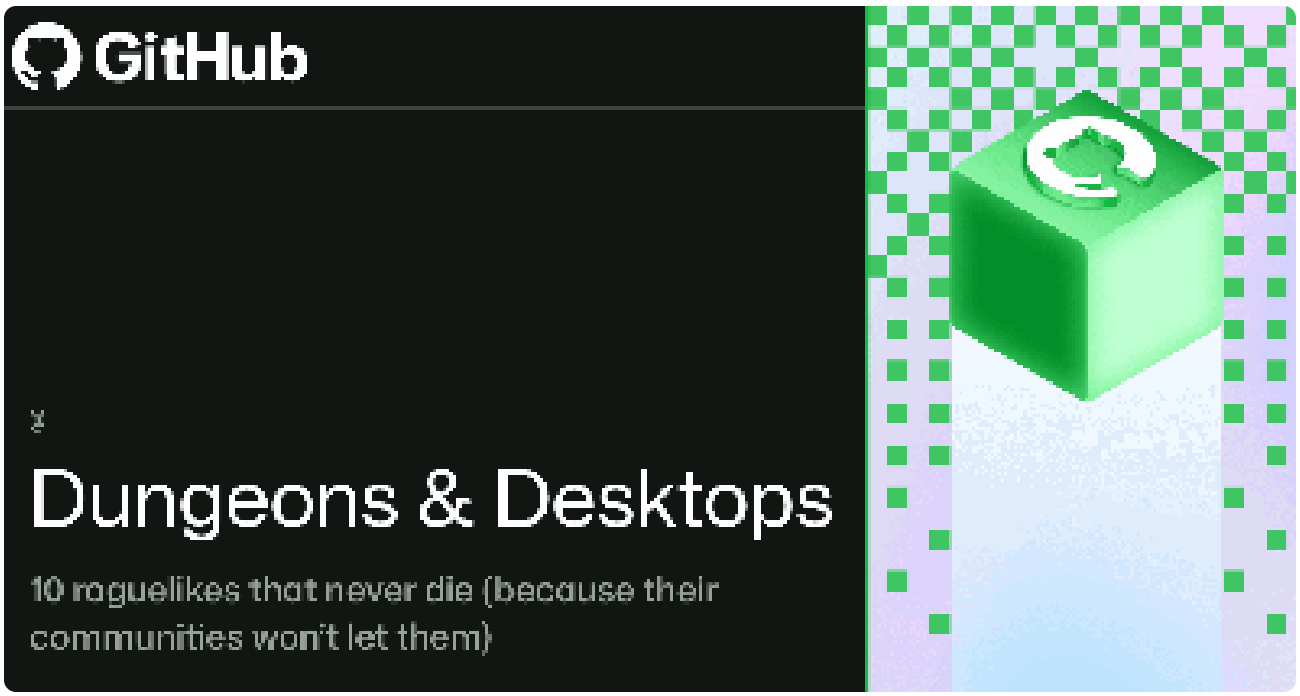
/ Blog

Highlights from Git 2.52

The open source Git project just released Git 2.52. Here is GitHub's look at some of the most interesting features and changes introduced since last time.

Taylor Blau

Related posts



Application development

Dungeons & Desktops: 10 roguelikes that never die (because their communities won't let them)

Roguelikes don't die. They fork, mutate, get argued over, rewritten, abandoned, and revived again. Sometimes all at once.

/ Blog

Developer skills

GitHub for Beginners: Getting started with OSS contributions

Learn how to find opportunities to contribute to the open source community.

Kedasha Kerr

Kedasha Kerr

Explore more from GitHub

Docs

Everything you need to master GitHub, all in one place.

Go to Docs

GitHub

Build what’s next on GitHub, the place for anyone from anywhere to build anything.

Start building

Learn more

The GitHub Podcast

Catch up on the GitHub podcast, a show dedicated to the topics, trends, stories and culture in and around the open source developer community on GitHub.

Listen now

/ Blog

GitHub

Product	Platform	Support	Company
Features	Developer API	Docs	About
Security	Partners	Community Forum	Blog
Enterprise	Atom	Training	Careers
Customer Stories	Electron	Status	Press
Pricing	GitHub Desktop	Contact	Shop
Resources			

Leadership in Tech

Barely treading water

#314 – May 24, 2026

> how leaders can quietly fall into overload while still appearing successful

Barely treading water
11 minutes by Michael Lopp

Michael explores how leaders can quietly fall into overload while still appearing successful. Through a conversation with his chief of staff, Michael realizes he is failing at prioritization, delegation, and setting limits. He points out common warning signs of burnout and offers practical solutions: admitting failure, prioritizing honestly with trusted support, delegating important work, and learning to say no. It highlights that effective leadership requires self-awareness, honesty, and difficult decisions.

Fix delivery first

10 minutes by Ant Murphy

Ant explains that many companies fail with OKRs, discovery, and AI because their real problem is poor delivery systems. Long release cycles, technical debt, and weak engineering foundations slow progress and reduce trust. Using the Theory of Constraints, Ant argues companies should first fix delivery bottlenecks before focusing on strategy, discovery, or AI. Strong delivery creates faster feedback, builds trust, and makes innovation and improvement possible.

On blunt feedback

2 minutes by Shreyas Doshi

Blunt feedback only helps when the giver has good intentions and real expertise in the area. In toxic workplaces, some feedback is designed to attach damaging labels to you through backchannels, not to help you grow. And even well-meaning feedback can miss the mark when the giver is less skilled than the recipient. For high performers, knowing when to ignore or adapt feedback is a rare but valuable skill.

You taught the company to overload you

4 minutes by Aviv Ben-Yosef

Tech leaders who always say yes create the same problems as those who always say no. Making the cost of every agreement visible is the real solution. Be honest about workload limits, avoid sudden pushback after long silence, and always present a range of options rather than a flat refusal. Boundaries need constant reinforcement, not just a one-time announcement.

The quiet power of presence

3 minutes by Andi Roberts

Most of us listen only to find a gap where we can speak. Real listening means paying attention to the values and hopes behind someone's words, not just the words themselves. Sitting with silence a little longer than feels comfortable often brings out what matters most. When you truly hear someone without judging or fixing them, you build the trust that holds relationships together.

industry

- A new generation of ads for the AI era of Search

- SpaceX files for IPO
- The last six months in LLMs in five minutes

security

- GitHub confirms breach of 3,800 repos via malicious VSCode extension
- CISA Admin Leaked AWS GovCloud Keys on Github
- Mini Shai-Hulud Strikes Again: 317 npm Packages Compromised

And the most popular article from the last issue was:

- Recognize your management wins

newsletters

- Programming Digest
- React Digest
- C# Digest

Building Pi With Pi

written on May 24, 2026

Pi is now part of Earendil, but in the important sense it is still Mario's project. He has been living with its issue tracker longer than I have, and he has been exposed to the weirdness of the new form of agent traffic in Open Source projects for longer too. This post is mostly a reflection of my own experience after spending more time in the tracker, using Pi to work on Pi, and watching what I have learned about it so far.

Slop Issues

Unsurprisingly, we are using Pi to build Pi. That sounds like a cute dogfooding thing but it really helps understand what we do. An interesting effect of building with agents is that it changes the role of the issue tracker a tiny bit. The issue descriptions are not just messages from a user to a maintainer because we also use them as inputs for prompts in Pi sessions. It is something I might hand to my clanker[1] and say: "understand this, reproduce it, inspect the code, and propose a fix."

That means the shape of the issue matters in a new way. A bad issue was always annoying, but at least a lot of issues were vague. Now we are also dealing with a class of issues that are 5% human and 95% clanker-generated and largely inaccurate shit. A bad issue that contains a plausible but wrong diagnosis creates extra work.

The most frustrating failure mode right now is that people submit issues that are not in their own voice. They contain an observed problem somewhere, but it has been thrown into a clanker and the clanker reworded it and made a huge mess of it. Typically, it was prompted so badly that the conclusions produced are more often than not inaccurate but always full of confidence. The result is complete guesswork on root causes, fake-minimal repros, suggested implementation strategies, analogies to adjacent but often the wrong code, and long lists of error classes that might or might not matter.

That is worse than no diagnosis.

I don't want to point to specific issues because I really do not want to bad mouth anyone, but it is frustrating. It is also frustrating because when I give that issue to Pi, Pi sees the wrong diagnosis too. It does not treat the issue body as a rumor. It treats it as evidence. It will happily go down the path that the issue already prepared for it, because the prose is confident and the code references look plausible. We use a custom slash command called `/is`, which specifically has this instruction in it:

> *Do not trust analysis written in the issue. Independently verify behavior and derive your own analysis from the code and execution path.*

Unfortunately, it does not fully work, because when humans first throw their issue through the clanker wringer, their clanker expands scope almost immediately. What was once a very narrow and fact based bug observation, turns into a much expanded surface area full of hypotheses. So at least personally, I increasingly want issue reports to be condensed to what the human actually observed:

1. I ran this command.
2. I expected this to happen.
3. This happened instead.
4. Here is the exact error or log.

That is enough. If you used an LLM to understand the problem, great, maybe leave it as a follow-up comment. But the issue and the issue text should be something you own. If you do not know the root cause, say that. I too can operate a clanker, and I

would rather do this myself than use your slop. If your repro is a guess, say that. If the only hard fact is one stack trace, give me the stack trace and stop there.

Slop Begets Slop

That we're seeing issues full of slop is just a result of the present day quality of these machines. Sadly, their failures in creating good issues extend to a lot of code that is generated. Not all of it, but a lot of code. Over and over I keep running into them over-engineering the hell out of issues and implementations.

If you tell them that "this malformed session log crashes the reader," the clanker will often add a tolerant reader. Then it will add a fallback, then maybe a migration, then more debug output, then a test for all of this. None of this is necessarily wrong in isolation, but it can be the wrong move for the system.

At Pi's core is a rather well-designed session log with invariants that must be upheld. The clanker's present-day behavior is to just assume that no such invariants exist, and instead to make the system work with all kinds of malformedness, blowing up the complexity in the process.

Almost always, the correct fix is not to handle the bad state, but to make the bad state impossible. This matters a lot for persisted data such as Pi session logs. They are opened, branched, compacted, exported, shared, and analyzed. The goal here is to never write bad session data. Yet if you just let the clanker roam freely, it will attempt to handle every case of bad data in the session log with a more permissive reader.

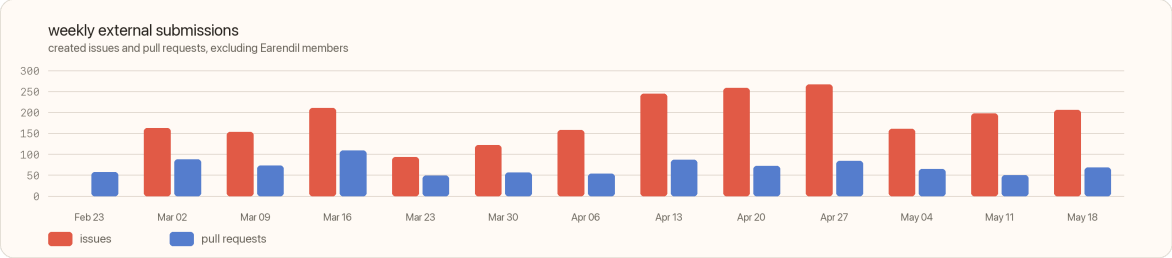
I have complained about this plenty, but working on Pi's code base continues to reinforce the point. This is one of the ways LLM authored code grows so much needless complexity. All these models see a local failure and try to locally defend against it. As maintainers we have to keep pulling the conversation back to the global invariant, which is harder than it should be, and it's laborious.

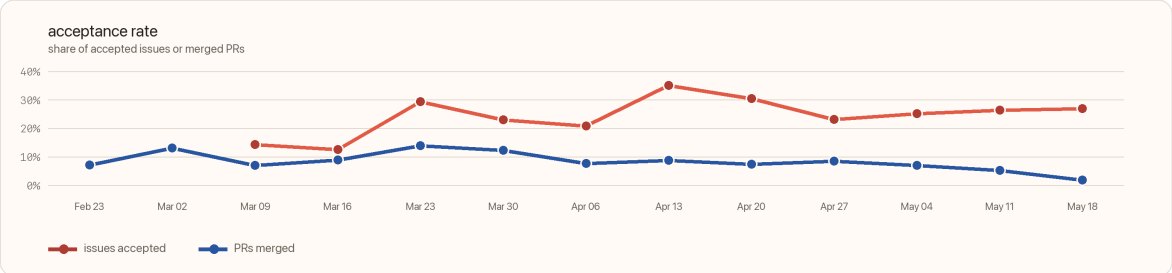
Volume Is The Problem

Then there is the issue of volume. The tracker is receiving a lot of issues and PRs, and a significant fraction of them are clearly LLM-assisted. Some are good, none are excellent, and most are just bad. The total throughput is a maintenance problem by itself.

As you might know, Pi's issue tracker is automated to close all issues and pull requests from new contributors, and there is a manual process by which we might reopen some of them or approve individuals. So auto-close -> reopen -> close again is an interesting statistic for us to look at.

I pulled the public GitHub tracker data while writing this over the last 90 days. Excluding Earendil members, that leaves 3,145 external issues and pull requests. Of those, 2,504 were auto-closed because they were from non-approved individuals. 17% were re-opened but that somewhat undercounts issues, because some remain closed while we still fix them. If we also count issues referenced by a main-branch commit or merged pull request that number rises to 26%. For pull requests the number is worse: 60 of 714 auto-closed PRs were ultimately merged, or about 8%.





Many of the issues and PRs are complete slop and in some cases the humans did not even realize that they created them. Sources of low-quality spam include OpenClaw instances, as well as some skills that people put into their context that seemingly encourage issue creation.

GitHub clearly is not built to deal with this new form of Open Source, but I'm increasingly feeling the need to put the blame less on GitHub than on all the people

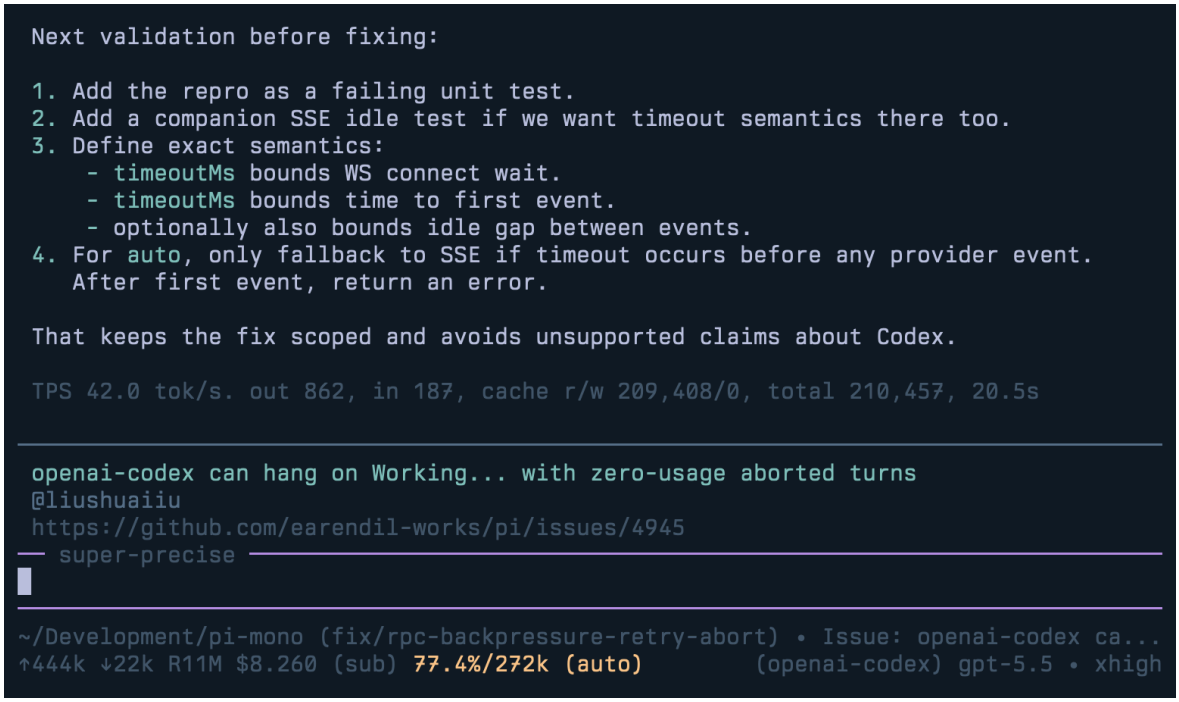
involved who make that experience painful. If your clanker shits on someone else's issue tracker then it's not the fault of GitHub, it's yours alone.

Careful Parallelism

Pi might be built with Pi, but we're quite far off today from where Bun and OpenClaw already are: fully detached, automated software engineering. Maybe we will reach that point, I don't know. Today it does not seem like we know how to pull off a dark factory and we also don't yet have the desire. That said, there is quite a bit of parallelism going on, and it is mostly for reproducing issues.

The small setup we use for this is three tiny pieces in Pi's own committed `.pi` folder. `/is` (for analyze **is**sue) is a prompt for analyzing GitHub issues: it labels and assigns the issue, reads the full thread and links, then explicitly tells the agent not to trust the analysis in the issue and to derive its own diagnosis from the code. Then an extension adds a `prompt-url-widget` which watches the prompt before the agent starts, recognizes the GitHub issue or PR URL that `/is` (or the PR equivalent) put into the prompt, fetches the title and author with `gh`, renders that in a little UI widget, and renames the session. It also rebuilds that state on session start or session switch, so if we reopen an older investigation the window still tells the developer which issue it belongs to.

In practice this means it's possible to have several Pi windows open, each running `/is` against a different issue, and the UI keeps the investigations visually distinct while the agents do their independent reproduction and code reading. Once the investigations are done, one can work through them sequentially. To finish off everything, `/wr` (**wr**ap it up) is the matching wrap-up prompt: it infers the GitHub context from the session, updates the changelog, drafts or posts the final issue comment with a disclaimer, commits only the files changed in that session, adds the appropriate `closes #...` when there is exactly one issue, and pushes from `main`.



Open Source Is About Hard Problems Worth Fixing

You will have noticed this already but Open Source in a post-AI world is under a strange new pressure. We are getting more code, more projects, and more issues. Projects appear with no real users, or a temporary audience of one, and even projects with thousands of stars can have a shelf life of weeks.

For us, Pi's harness layer is worth maintaining carefully because it solves hard coordination problems and creates a platform we and others can build on. We also know that coordination and cooperation lifts us all up. Many times the right answer is not to work around a problem locally, but to make the upstream behavior correct. Mario has been very good at refusing to make Pi paper over every misconfigured gateway, and we're trying to preserve that discipline. When a gateway behaves correctly, everybody benefits.

Sadly that type of thinking is quickly disappearing because these machines make local workarounds cheap, so code accumulates local defenses against every misbehavior. Instead of humans talking to humans about where a fix belongs, one human and one machine work around the problem in isolation.

Keep in mind that AI has not increased the number of people who need software, or the number of maintainers who can review it. It has mostly increased the amount of code and the number of projects competing for attention. Some of that is healthy, but a lot of it fragments effort that should be shared.

We need stronger foundations, not weaker ones. Open Source needs more collaboration, not more isolated work with a machine. Human communication is hard, and it is tempting to avoid it when you can sit alone with your clanker. But isolation is not where Open Source derives its value. The value is in the community and the structure that lets projects outlive their original creators.

1. To me, clanker is a much preferable term for agent. Agency lies with humans, not with machines. Calling these things agents I still believe is a mistake, but alas.↩

This entry was tagged ai, open-source and pi

copy as / view markdown