



Getting started

Jules is an experimental coding agent that helps you fix bugs, add documentation, and build new features. It integrates with GitHub, understands your codebase, and works **asynchronously** — so you can move on while it handles the task.

This guide will walk you through setting up Jules and running your first task.

Login

- 1 Visit jules.google.com
- 2 Sign in with your Google account.
- 3 Accept the privacy notice (one-time).

Connect GitHub

Jules needs access to your repositories in order to work.

- 1 Click **Connect to GitHub account**.
- 2 Complete the login flow.
- 3 Choose *all* or specific repos that you want to connect to Jules.
- 4 You will be redirected back to Jules. If not, try refreshing the page.

Once connected, you'll see a **repo selector** where you can select the repo you want Jules to work with, and a prompt input box.

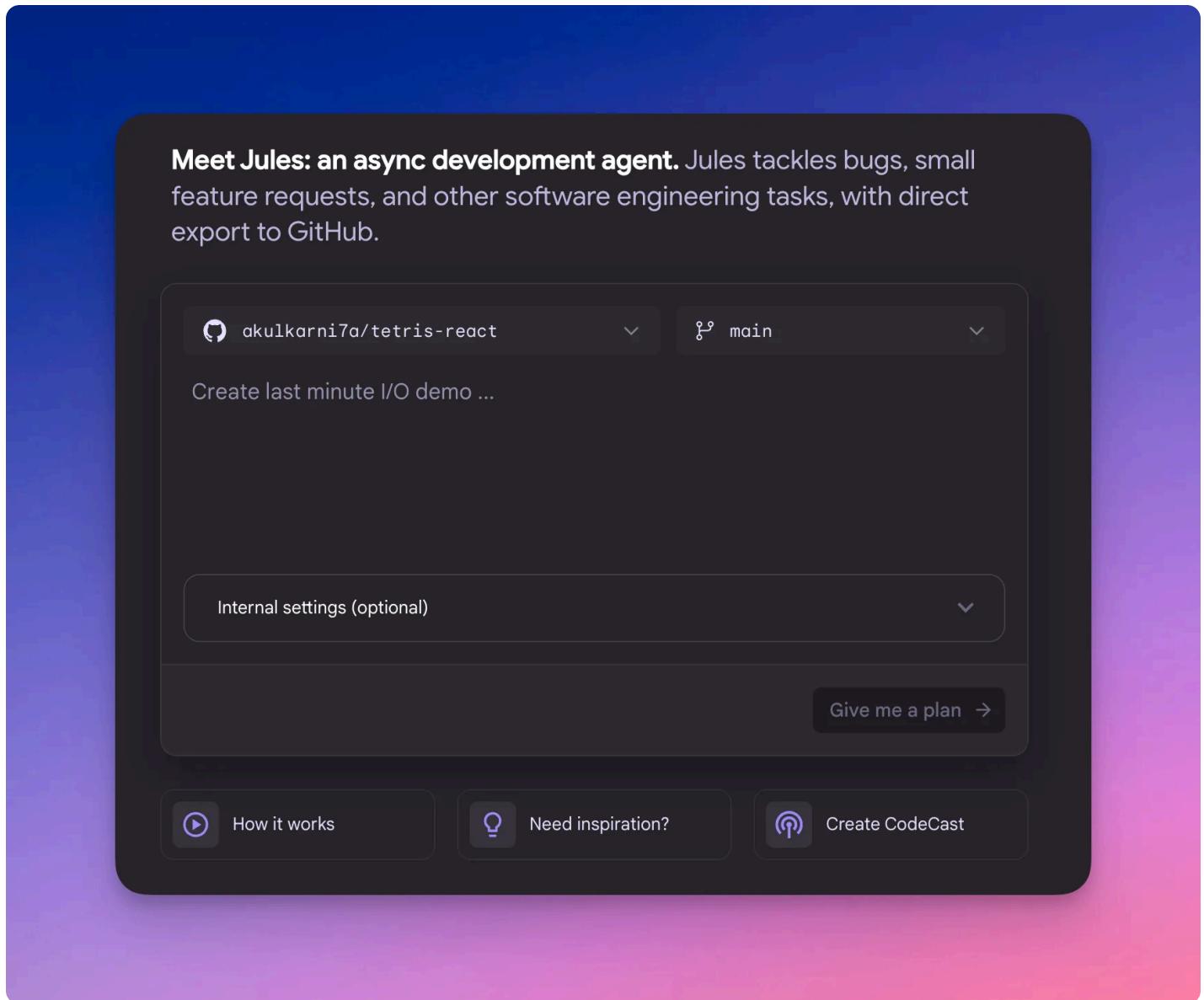
Starting your first task

Jules runs in a virtual machine where it clones your code, installs dependencies, and modifies files.

- 1 Pick a repository from the repo selector.

- 2** Choose the branch you want Jules to work on. The default branch will be selected already.
You do not have to modify this unless you want Jules to work on a specific branch.
- 3** Write a clear, specific prompt. For example, Add a test for "parseQueryString function in utils.js"
- 4** (Optional) Add environment setup scripts.
- 5** Click **Give me a plan**

Once you submit a task, Jules will generate a plan. You can review and approve it before any code changes are made.



Include AGENTS.md file

Jules now automatically looks for a file named AGENTS.md in the root of your repository. This file can describe the agents or tools in your codebase, such as what they do, how to interact with them, or any input and output conventions. Jules uses this file to better understand your code and generate more relevant plans and completions.

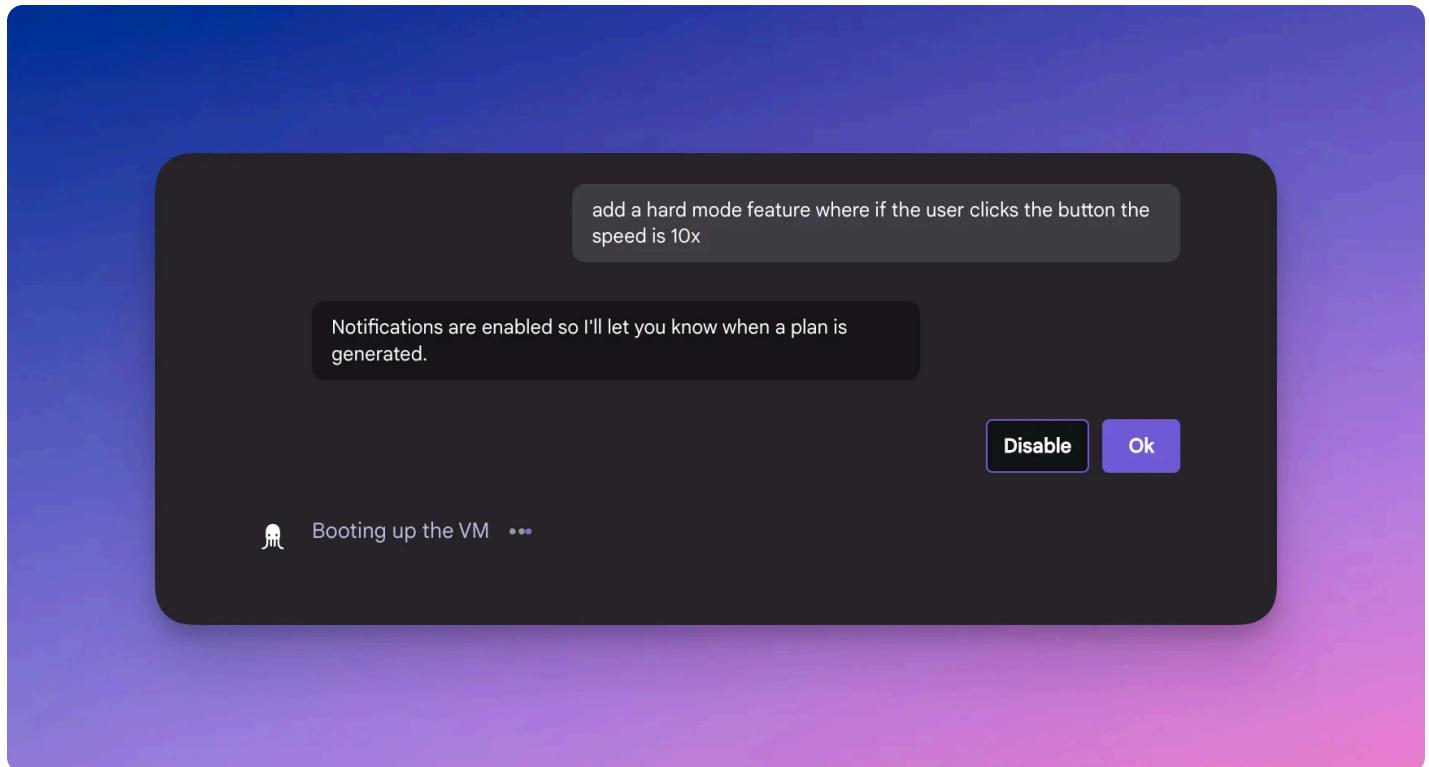
Tip: Keep AGENTS.md up to date. It helps Jules and your teammates work with your repo more effectively.

Enabling notifications

You are free to leave Jules while it is running. To stay informed:

- 1 When prompted, enable browser notifications.
- 2 Go to **Settings —> notifications** at any time to enable or disable notifications.

You'll be notified when the task completes or needs your input.



What's next?

- [Running a task with Jules](#) - Full walkthrough

- [Environment setup](#) - Make Jules smarter about your project
- [Reviewing plans & feedback](#) - how to approve and integrates

Want to dive into real-world use cases? Check out the [Jules Awesome Prompts repo](#).



Environment setup

Jules runs each task inside a secure, short-lived virtual machine (VM). This lets it clone your repository, install dependencies, and run tests.

To make sure Jules can do its job effectively, you can provide **setup scripts** that prepare the environment.

What's preinstalled?

Every Jules VM runs Ubuntu Linux and includes many popular developer tools out of the box:

- Node.js
- Python
- Go
- Java
- Rust

The latest versions can be seen here:

Environment check output

----- Python -----

- ✓ python3: Python 3.12.11
- ✓ python: Python 3.12.11
- ✓ pip: pip 25.1.1 from /home/jules/.pyenv/versions/3.12.11/lib/python3.12/site-packages/pip (python 3.12)
- ✓ pipx: 1.4.3
- ✓ poetry: Poetry (version 2.1.3)
- ✓ uv: uv 0.7.13
- ✓ black: black, 25.1.0 (compiled: yes)
- ✓ mypy: mypy 1.16.1 (compiled: yes)
- ✓ pytest: pytest 8.4.0
- ✓ ruff: ruff 0.12.0
- ✓ pyenv: available
 - system
 - 3.10.18
 - 3.12.11 (set by /home/jules/.pyenv/version)

----- NodeJS -----

- ✓ node: v22.16.0 *
 - v18.20.8 *
 - v20.19.2 *
 - v22.16.0 *
- system *
- ✓ nvm: available
- ✓ npm: 11.4.2
- ✓ yarn: 1.22.22
- ✓ pnpm: 10.12.1
- ✓ eslint: v9.29.0
- ✓ prettier: 3.5.3
- ✓ chromedriver: ChromeDriver 137.0.7151.70
 - (dfa4dc56b2ahb56eb2a14cad006ea5e68c60d5de-refs/branch-heads/7151@{#1875})

----- Java -----

- ✓ java: openjdk version "21.0.7" 2025-04-15
 - OpenJDK Runtime Environment (build 21.0.7+6-Ubuntu-0ubuntu124.04)
 - OpenJDK 64-Bit Server VM (build 21.0.7+6-Ubuntu-0ubuntu124.04, mixed mode, sharing)
- ✓ maven: Apache Maven 3.9.10 (5f519b97e9448438d878815739f519b2eade0a91d)
- ✓ gradle: Gradle 8.8

----- Go -----

✓ go: go version go1.24.3 linux/amd64

----- Rust -----

✓ rustc: rustc 1.87.0 (17067e9ac 2025-05-09)

✓ cargo: cargo 1.87.0 (99624be96 2025-05-06)

----- C/C++ Compilers -----

✓ clang: Ubuntu clang version 18.1.3 (1ubuntu1)

✓ gcc: gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0

✓ cmake: cmake version 3.28.3

✓ ninja: 1.11.1

✓ conan: Conan version 2.17.0

----- Docker -----

✓ docker: Docker version 28.2.2, build e6534b4

✓ docker: Docker Compose version v2.36.2

----- Other Utilities -----

✓ awk: GNU Awk 5.2.1, API 3.2, PMA Avon 8-g1, (GNU MPFR 4.2.1, GNU MP 6.3.0)

✓ curl: curl 8.5.0 (x86_64-pc-linux-gnu) libcurl/8.5.0 OpenSSL/3.0.13 zlib/1.3
brotli/1.1.0 zstd/1.5.5 libidn2/2.3.7 libpsl/0.21.2 (+libidn2/2.3.7)
libssh/0.10.6/openssl/zlib nghttp2/1.59.0 librtmp/2.3 OpenLDAP/2.6.7

✓ git: git version 2.49.0

✓ grep: grep (GNU grep) 3.11

✓ gzip: gzip 1.12

✓ jq: jq-1.7

✓ make: GNU Make 4.3

✓ rg: ripgrep 14.1.0

✓ sed: sed (GNU sed) 4.9

✓ tar: tar (GNU tar) 1.35

✓ tmux: tmux 3.4

✓ yq: yq 0.0.0

You can check installed versions by adding commands like node -v to your setup script and clicking **Run to Validate**.

To view all preinstalled tools, you can use this command in your setup script and click **Run to Validate**.

```
set +x; . /opt/environment_summary.sh
```

Add a setup script

To help it install dependencies and run tests:

- 1 Click on the repo on the left sidebar
- 2 Select Configuration at the top
- 3 In the “Initial Setup” window, enter the commands needed to install dependencies and prep your project For example:

```
npm install  
npm run test
```

Test your setup script

Click **Run to Validate** to check that your setup script works.

Validation tips

- You can check installed versions by adding commands like node -v to your setup script and clicking **Run to Validate**.
- Always include commands to install packages, run linters, or execute tests.
- Use the validation button to catch errors early.
- Keep your setup lightweight and fast
- Jules runs a clean environment each time.



Running Tasks with Jules

Once you've logged in and connected GitHub, you're ready to start coding with Jules. This guide walks through the key steps of running a task — from selecting a repo to writing your prompt and setting up notifications.

Choose a repo and branch

Jules needs a repo and branch to work on. After logging in:

- 1 Open the **repo selector** dropdown.
- 2 Select the repository you'd like Jules to work on.
- 3 Choose the branch you want to base your changes on.

Jules remembers your last-used repo, so you'll always see the last used repo in the repo selector.

Write a clear prompt

Jules works best when your prompt is specific and scoped. Use plain language — no need for perfect grammar or code.

Good prompts

- Add a loading spinner while `fetchUserProfile` runs
- Fix the 500 error while submitting the feedback form
- Document the `useCache` hook with JSDoc

Avoid

- Fix everything
- Optimize code
- Make this better

If Jules needs more clarity, it will ask for feedback before writing code.

Watching Jules work

Once the plan is approved, Jules will start coding.

You will see:

- An **activity feed** as each step completes
- Inline explanations of each change
- A **mini diff** preview for each file

Use the **diff editor** for a full view across all files.

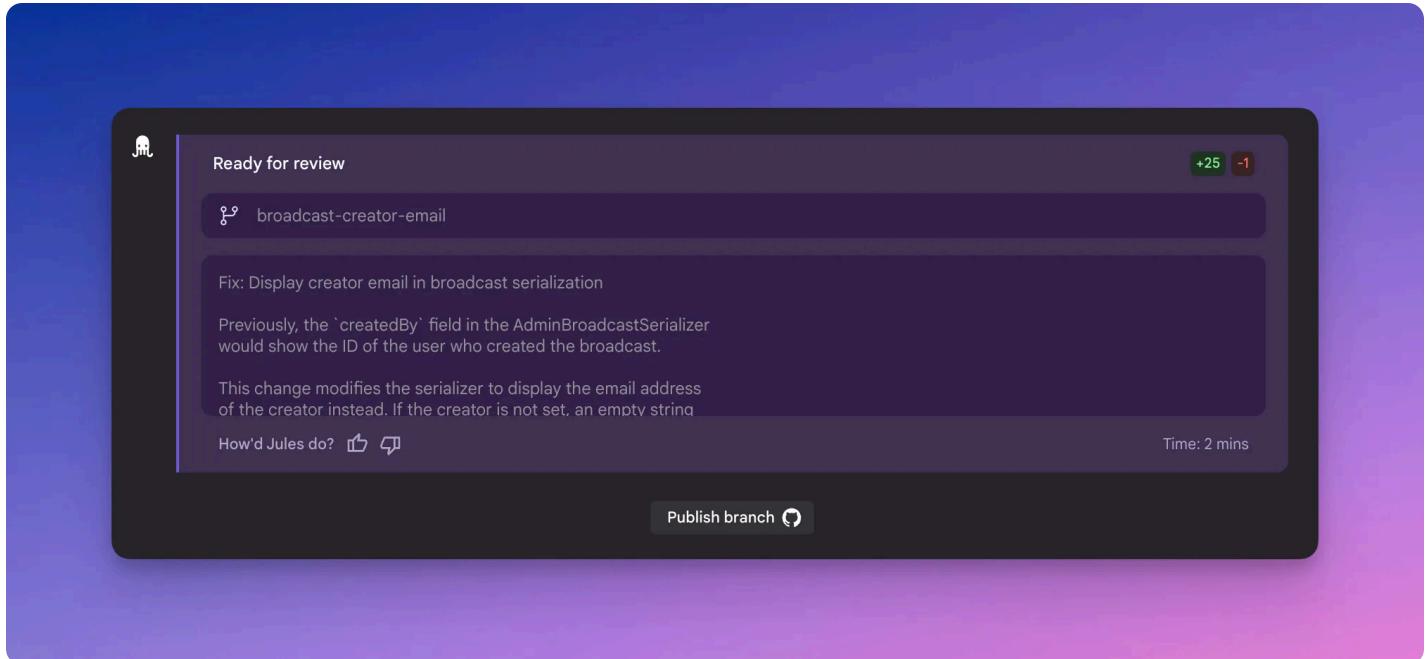
Final summary & branch creation

When Jules finishes a task, it provides a summary of everything it accomplished.

- **Files changed**
- **Total runtime**
- **Lines of code added/changed/removed**
- **Option to create branch and commit message**

You can click **Create branch** to push the changes. Note that:

- You are the branch owner
- Jules appears as the commit author
- You can open a PR from this branch in GitHub



Giving feedback mid-task

You can send feedback to Jules while it's working:

- Type directly into the chat box
- Ask Jules to change its approach, revise code, or clarify logic
- Jules will respond and, if needed, replan or revise the task

You can intervene at any time, you're in control.

Pausing Jules

You can pause Jules at any time by clicking the “**pause**”.

When Jules is paused it won't do any work, and will wait for your next set of instructions. You can prompt it again, unpause it, or delete the task.



Reviewing plans & giving feedback

Once you start a task in Jules, it generates a **plan** before writing any code. This lets you know the direction Jules will take while it works on the task. From here, you can also iterate with Jules on the plan before any code is written.

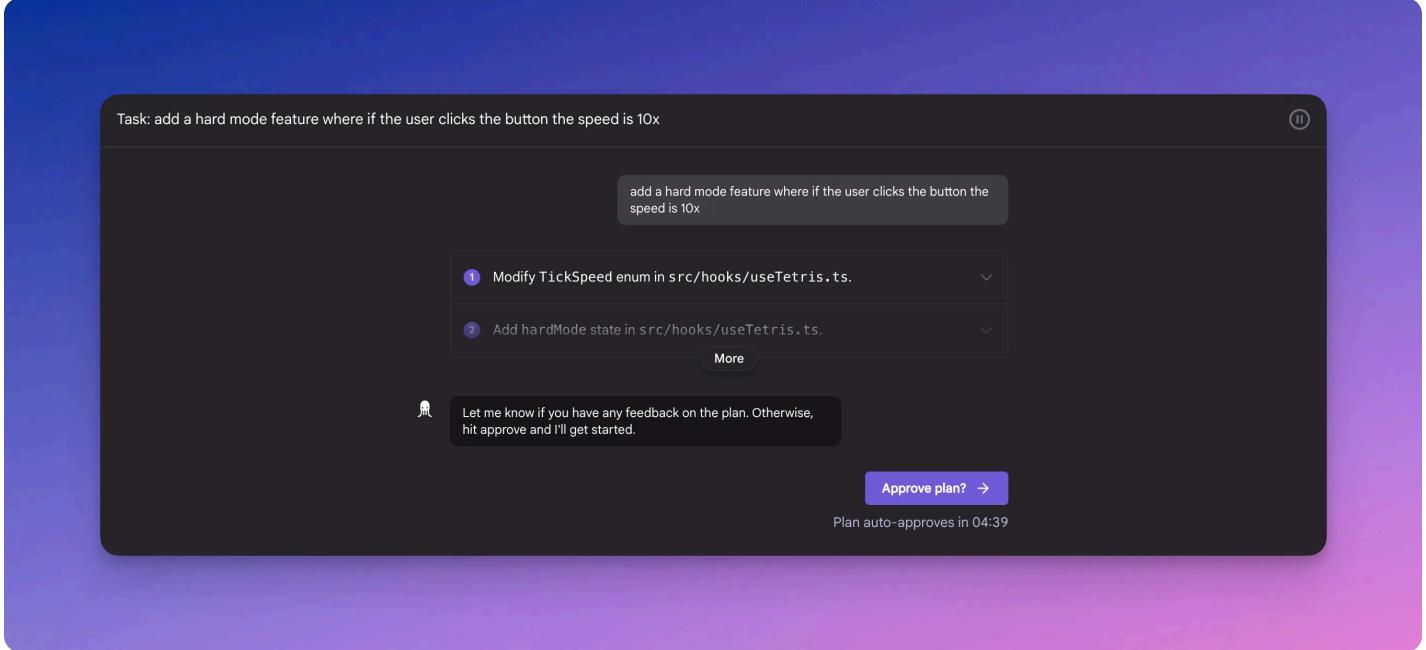
Reviewing the plan

After setup (cloning your repo, initializing the VM, and installing dependencies), Jules will present its plan.

You'll see:

- A natural language description of what Jules intends to direction
 - Step-by-step breakdowns
 - Any assumptions or setup steps
- 1 Open each step by expanding them to read through each one.
 - 2 Give feedback using the chat input if you need.
 - 3 Click **approve plan** when you're ready for Jules to begin executing and writing code.

Note: If you navigate away, Jules will eventually auto-approve the plan, which is set on a timer. So there is no need to babysit.



Giving feedback

At any point, you can use the chat box to talk to Jules.

- Ask it to revise a step
- Point out something it missed
- Clarify your original request
- Answer its questions, or provide additional information Jules may need

Jules will respond and update its plan if needed.

Task: When serializing a broadcast object, include the email address of the creator instead of j [1]

I've created a plan to address the issue. Please review it and let me know if you have any feedback. Otherwise, let me know if you approve, and I'll get started.

Plan approved

Modified AdminBroadcastSerializer to include the creator's email address.

Updated src/sentry/api/serializers/models/broadcast.py

actually, instead of returning None lets return a empty string

Added a test for the AdminBroadcastSerializer changes.

Updated tests/sentry/api/serializers/test_broadcast.py

Okay, I'll change the serializer to return an empty string instead of None when created_by_id is not present. I'll also update the test accordingly.

1. Modify AdminBroadcastSerializer in src/sentry/api/serializers/models/broadcast.py.

2. Update the test for the change in tests/sentry/api/serializers/test_broadcast.py.

Working ...

Modify AdminBroadcastSerializer in src/sentry/api/serializers/models/broadcast.py.

Change the createdBy field to use the email address of the user instead of the ID.

- This will involve changing obj.created_by_id to obj.created_by_id.email.
- If created_by_id is None, the createdBy field should be an empty string.

Talk to Jules →

```

broadcast.py  test_broadcast.py
```

```

7 7 @register(Broadcast)
8 8 class BroadcastSerializer(Serializer):
9 9     def get_attrs(self, item_list, user, **kwargs):
10 10         if not user.is_authenticated:
11 11             seen = set()
12 12         else:
13 13             seen = set(
14 14                 BroadcastSeen.objects.filter(broadcast__in=item_list, user_i
15 15                     "broadcast", flat=True
16 16             )
17 17         )
18 18
19 19     return {item: {"seen": item.id in seen} for item in item_list}
20 20
21 21 def serialize(self, obj, attrs, user, **kwargs):
22 22     return {
23 23         "id": str(obj.id),
24 24         "message": obj.message,
25 25         "title": obj.title,
26 26         "link": obj.link,
27 27         "cto": obj.cto,
28 28         "mediaUrl": obj.media_url,
29 29         "isActive": obj.is_active,
30 30         "dateCreated": obj.date_added,
31 31         "dateExpires": obj.date_expires,
32 32         "hasSeen": attrs["seen"],
33 33         "category": obj.category,
34 34     }
35 35
36 36
37 37 class AdminBroadcastSerializer(BroadcastSerializer):
38 38     def get_attrs(self, item_list, user, **kwargs):
39 39         attrs = super().get_attrs(item_list, user)
40 40         counts = dict(
41 41             BroadcastSeen.objects.filter(broadcast__in=item_list)
42 42                 .values("broadcast")
43 43                     .distinct()
44 44                     .annotate(user_count=Count("broadcast"))
45 45                     .values_list("broadcast", "user_count")
46 46         )
47 47
48 48     for item in attrs:
49 49         attrs[item]["user_count"] = counts.get(item.id, 0)
50 50
51 51
52 52     def serialize(self, obj, attrs, user, **kwargs):
53 53         context = super().serialize(obj, attrs, user)
54 54         context["userCount"] = attrs["user_count"]
55 55         context["createdBy"] = obj.created_by_id.id if obj.created_by_id.els
56 56         context["createdBy"] = obj.created_by_id.email if obj.created_by_id
57 57

```



Reviewing code changes

Once you approve a plan, Jules begins working inside a virtual machine. As it completes steps, you'll see an activity log with Jules' thoughts and code updates. This page walks you through what to look for and how to respond.

Activity feed

As Jules works, you'll see a real-time **activity feed** that logs:

- Each step it completes
- Descriptions of what it did
- Any outputs or errors encountered
- Requests for additional information or feedback

This feed gives you insight into Jules' decision-making and progress.

Code diffs

When Jules changes code, you'll see a **mini diff** directly in the feed. For a more complete view:

- On the right pane, you can view a full screen expanded **diff editor** to see all changes across files
- The **diff editor** only shows files where it modified or added code.
- Expand the diff editor to full screen, or drag the left sidebar of the diff editor to slide it to your preferred width

You can download and copy code that Jules has written from the download and copy icons located in the top right of the diff editor panel. When you copy code, only the updated code will save to your clip board (not the full diff).

This is your central hub for understanding the scope of the changes Jules made to your repo.

The screenshot shows a developer's terminal interface with Jules AI integrated. The terminal window has a dark theme with syntax-highlighted code. On the left, a sidebar displays a task log with messages from Jules:

- "I've created a plan to address the issue. Please review it and let me know if you have any feedback. Otherwise, let me know if you approve, and I'll get started."
- "Plan approved! 🚀"
- "Modified AdminBroadcastSerializer to include the creator's email address."
- "Updated src/sentry/api/serializers/models/broadcast.py" (with a note: "actually, instead of returning None lets return a empty string")
- "Added a test for the AdminBroadcastSerializer changes."
- "Updated tests/sentry/api/serializers/test_broadcast.py" (with a note: "Okay, I'll change the serializer to return an empty string instead of None when created_by_id is not present. I'll also update the test accordingly.")
- Two numbered items: "1. Modify AdminBroadcastSerializer in src/sentry/api/serializers/models/broadcast.py." and "2. Update the test for the change in tests/sentry/api/serializers/test_broadcast.py."
- A "Working ..." status indicator.
- A "Talk to Jules" button at the bottom.

The right side of the terminal shows the actual code being edited, with line numbers and syntax highlighting. The code is a Python file containing serializers for broadcast objects.

```

broadcast.py  test_broadcast.py

7 7 @register(Broadcast)
8 8 class BroadcastSerializer(Serializer):
9 9     def get_attrs(self, item_list, user, **kwargs):
10 10         if not user.is_authenticated:
11 11             seen = set()
12 12         else:
13 13             seen = set(
14 14                 BroadcastSeen.objects.filter(broadcast__in=item_list, user_i
15 15                     "broadcast", flat=True
16 16             )
17 17         )
18 18
19 19     return {item: {"seen": item.id in seen} for item in item_list}
20 20
21 21 def serialize(self, obj, attrs, user, **kwargs):
22 22     return {
23 23         "id": str(obj.id),
24 24         "message": obj.message,
25 25         "title": obj.title,
26 26         "link": obj.link,
27 27         "cta": obj.cta,
28 28         "mediaUrl": obj.media_url,
29 29         "isActive": obj.is_active,
30 30         "dateCreated": obj.date_added,
31 31         "dateExpires": obj.date_expires,
32 32         "hasSeen": attrs["seen"],
33 33         "category": obj.category,
34 34     }
35 35
36 36
37 37 class AdminBroadcastSerializer(BroadcastSerializer):
38 38     def get_attrs(self, item_list, user, **kwargs):
39 39         attrs = super().get_attrs(item_list, user)
40 40         counts = dict(
41 41             BroadcastSeen.objects.filter(broadcast__in=item_list)
42 42                 .values("broadcast")
43 43                     .distinct()
44 44                     .annotate(user_count=Count("broadcast"))
45 45                 .values_list("broadcast", "user_count")
46 46         )
47 47
48 48         for item in item_list:
49 49             attrs[item]["user_count"] = counts.get(item.id, 0)
50 50
51 51
52 52
53 53
54 54
55 55 -      55 +      55 +
56 56
57 57

```

Interactive feedback

You can interact with Jules in real-time through the chat box:

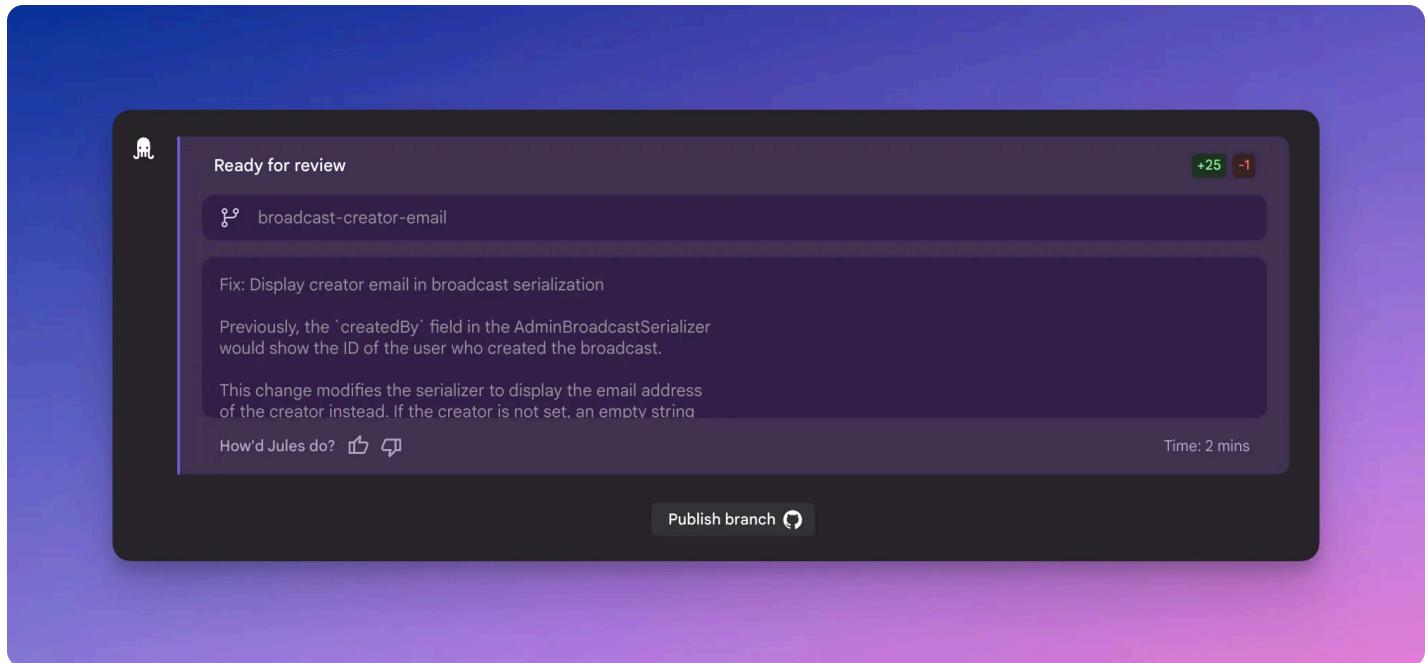
- Ask it to revise logic or naming
- Request additional tests or cleanup
- Give corrections like “return an empty string instead of None”

Task summary

When the task completes, Jules provides a final summary which includes:

- ✓ Files changed

- ⌚ **Total runtime**
- ➕ **Lines of code added/changed**
- 🌿 **Branch name and commit message**



Pushing to GitHub

Click **Create branch** to push Jules' changes to GitHub:

- Jules will appear as a commit author on the branch
- If you choose to open a PR, you will be the PR author

Once pushed, you can continue editing the branch, review it as a GitHub PR, or delete it.



Managing tasks and repos

To start a new task:

- 1 Click the Jules icon to return to the home screen and a blank prompt input
- 2 Use the + button in the top navigation on the task view
- 3 From within a specific repo view, click **new task**

You can launch multiple tasks at a time to run them simultaneously. If you are viewing an existing task, or repo page when you kick off a new task, that task will load and begin running in the background. You will have to navigate to the task to check for updates, approve the plan, etc.

Each task runs in its own virtual machine and maintains its own logs, environment setup, and code changes.

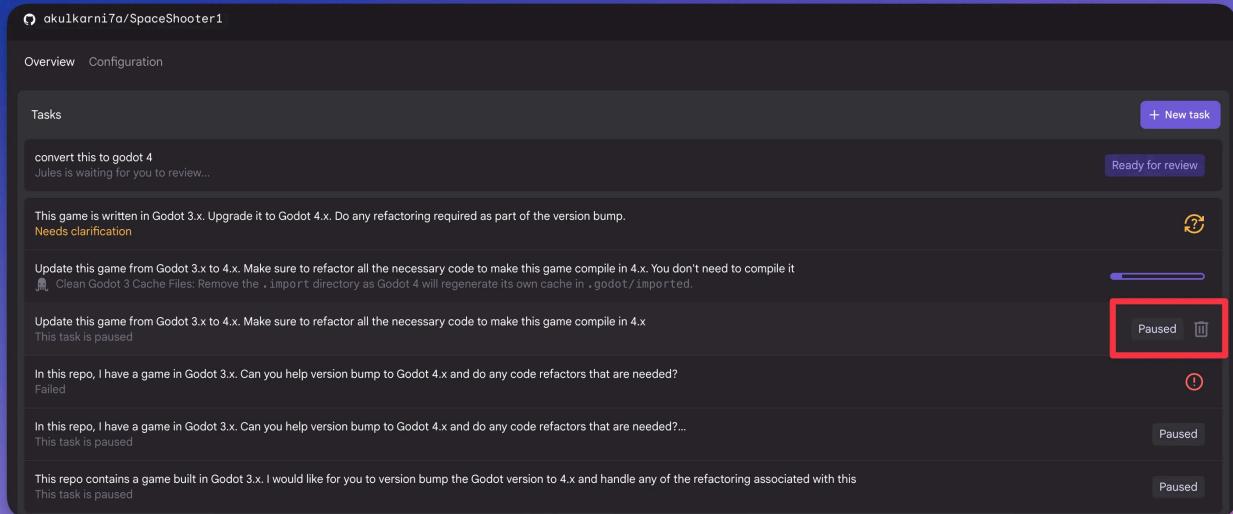
Pausing or deleting tasks

Sometimes you need to stop Jules mid-task or clean up old runs.

- To **pause** a task: click the **pause**.
- To **delete** a task: hover over the task in the repo view and click the **trash icon**.

Paused tasks can be resumed later; deleted tasks are removed permanently.

Task: Can you upgrade this to the latest LTS version of Node.js (i think its v22) and then compile it using yarn, yarn dev to verify 



Managing repositories

Jules can only access repositories you explicitly allow through GitHub. In the future, Jules will work with more version control systems.

Selecting a repo

If you've already granted Jules access to all your repos:

- Use the dropdown when starting a task to choose any authorized repo
- Enroll new from the left sidebar
- Search for your repo using the **repo selector**



akulkarni7a/-Archive-Pixel...



Granting access to more repos

To give Jules access to more of your repositories:

- 1 Go to github.com
- 2 Click your profile photo —> **settings**
- 3 In the left sidebar, click **applications**
- 4 Find **Google Labs Jules** and click **configure**
- 5 Under **repository access**, select additional repos
- 6 Click **save**

You can also authorize new repositories via the **repo selector** in Jules. Open the repo selector and scroll to the bottom to click on **+Add repository**. This will take you to GitHub, where you can select additional repositories to grant access for Jules.



Repo view

The repo view gives you a focused workspace for interacting with a specific repository. From here, you can view task history, manage running tasks, and launch features like Codecasts (coming soon).

Accessing the repo view

There are two ways to open a repo view:

- 1 Click on any repository name from the sidebar task list
- 2 Select a repo from the task picker, then click into it after starting a task

This opens a view scoped to that repository, including task history and available actions.

Viewing tasks by repo

Inside the repo view, you'll see a list of all tasks associated with that repo:

- **Running** tasks are listed at the top
- **Completed** tasks include full logs and diffs
- **Failed** or **waiting** tasks are clearly labeled

You can click on any task to reopen it, review the plan, or continue feedback.

Starting a new task

From the repo view, you can start a new task scoped to that repository:

- Click **new tasks** in the top right corner
- This will preselect the repo and branch
- Enter your prompt and optimal setup script as usual

Launch a Codecast

Codecasts are short, audio summaries of recent activity in your repo. They're an audio changelog for your repository.

In the repo view, you can generate a Codecast by clicking **Generate Today's Codecast**.

Codecasts currently include all commits (not just Jules activity) from the past few days. Like many features in Jules, Codecasts is still evolving.



Errors and failures

Even though Jules automates many parts of the development process, things can go wrong such as environment misconfigurations to task failures. This page covers how Jules reports errors, what it does automatically, and how you can debug issues.

How errors are reported

Jules surfaces errors in two key places:

- The **activity feed**, where the step failed is logged
- A **notification badge** (red dot) in the UI

These errors can show up whether the task has failed completely or simply requires your intervention.

Automatic retry behavior

Jules will attempt to retry failed steps automatically when possible. For example:

- Network hiccups
- Transient install errors
- Slow dependency resolutions

After multiple retries, if the problem persists, the task will be marked as **Failed**.

Common causes of failure

The most frequent issues are:

- Incomplete or missing **environment setup scripts**
- Prompts that are too vague or overly broad

- Repos with unusual or nonstandard build systems
- Long-running processes (like `npm run dev`) included in setup

Debugging environment setup

You can retry a task by:

- Clicking **rerun** from the task summary view
- Modifying your setup script or prompt before restarting the task

Make sure to address any specific feedback from the failure logs.



Limits and usage

Jules is currently in **beta** and available free of charge during this period. We're actively improving the service and learning how developers use Jules and where it provides the most value.

Usage data during this beta will help us:

- Improve task quality and system performance
- Understand real-world workflows
- Inform future pricing models

| [Learn more](#) about how your data is used to improve Jules.

While Jules is free of charge today, we do expect to introduce pricing in the future. We'll share more details as the product matures. Our focus is on building the right developer experience and learning from early adopters.

Task limits

Each user is subject to the following default limits:

- **5 concurrent tasks**
- **60 total tasks per day**
- **5 codecasts per day**

If you try to exceed these limits, Jules will notify you and prevent new task creation until your quota resets.

What happens at the limit

When you hit a limit:

- The **new task** button will be disabled

- A tooltip or error message will explain the reason
- You can continue reviewing or managing existing tasks
- Task history and feedback are unaffected

Requesting higher limits

If you're part of a larger team, or actively using Jules in your daily development flow, we'd love to hear from you.

To request increased limits:

[→ Fill out the request form](#)

You'll be asked to share:

- How you're using Jules
- Repositories or workflows it supports
- Your ideal task volume

We prioritize access for developers actively contributing feedback and pushing Jules to its limits.



FAQ

What is Jules?

Jules is a software coding agent that helps you fix bugs, add documentation, update your app, and implement new features. It integrates with GitHub and works asynchronously — meaning you can submit a task, go do something else, and return when it's done. Jules is currently in Public Beta.

Is Jules free of charge?

Yes, for now, Jules is free of charge. Jules is in beta and available without payment while we learn from usage. In the future, we expect to introduce pricing, but our focus right now is improving the developer experience.

How does Jules work under the hood?

Each task runs in a fresh virtual machine where Jules clones your repo, installs dependencies, and makes changes based on your prompt. You can provide setup scripts to ensure your project builds and tests correctly.

How does Jules run code, and what should I know about security?

When you run code in Jules, it's executed in a secure, cloud-based virtual machine (VM) with internet access. While this gives you powerful tools to test, build, and debug in context, it's important to treat the environment with the same security precautions you would for any public or shared compute surface. If you're not sure whether something is safe to run, we recommend reviewing it carefully (including non-code components). Jules is a large language model based system which operates on both the code and non-code files in a repository.

You are responsible for the code you run. That means:

- **Don't commit secrets** (like API keys, tokens, or credentials) to your repo, especially if you're pulling code into Jules from a Git provider.
- **Avoid known security vulnerabilities** in your dependencies or scripts. Consider following [GitHub's Quickstart](#) for securing your repository if you're using GitHub.
- **Be cautious with third-party packages or shell commands** that could compromise your system, your data, or others.

Can Jules run long-lived commands like `npm run dev`?

No. Long-running processes like dev servers or watch scripts aren't currently supported in setup scripts. Use discrete install/test commands instead.

What languages does Jules support?

Jules is language agnostic but works best with projects that use:

- JavaScript/TypeScript
- Python
- Go
- Java
- Rust

Support depends on what's installed on the VM and the clarity of your environment setup script.

Can I leave Jules while its working?

Yes! Jules is designed to be asynchronous. You can leave the app after submitting a task. Make sure to [enable notifications](#) so you'll be alerted when a plan is ready or a task completes.

How do I provide feedback or report a bug?

Click the [feedback](#) button in the bottom left of the Jules UI. No account or tracking system required — feedback goes straight to the team.

What happens if a task fails?

Jules will retry automatically. If it continues to fail, it will mark the task as failed and notify you. Common causes include broken setup scripts or vague prompts.

You can revise and rerun the task once you've addressed the issue.

How many tasks can I run?

Limits are listed [here](#).

Can I change which repos Jules has access to?

Yes. Go to your GitHub settings:

- 1 Click your profile —> **settings**
- 2 Select **applications** in the sidebar
- 3 Find **Google Labs Jules** and click **configure**
- 4 Adjust repository access

Then refresh Jules and your new repos will appear. [Learn more about managing tasks and repositories.](#)

Does Jules train on private repos?

No. Jules does **not** train on private repository content. Privacy is a core principle for Jules, and we do not use your private repositories to train models. [Learn more](#) about how your data is used to improve Jules.



Changelog

Jules Agent Update: Faster, Smarter, More Reliable

June 20, 2025



Changelog

AGENTS.md support,
better test habits, no punting

We've shipped a big upgrade to the Jules agent under the hood.

What's new:

- **Smarter context.** Jules reads from AGENTS.md if it's in your repo.
- **Improved performance.** Tasks now complete faster—no numbers to share just yet, but you'll feel it.
- **Significantly reduced punting.** We tightened the loop to keep Jules moving forward.
- **More reliable setup.** If you've added an environment setup script, Jules now runs it consistently.

- **Better test habits.** Jules is more likely to write and run tests on its own.

Check out the [Getting Started](#) guide to learn more about AGENTS.md support.

Modernized base environment and updated toolchains

June 18, 2025



Changelog

```
> jules --upgrade

“node”: 18.19.1 → 22.16.0
“python”: 3.12.3 → 3.12.11
“rust”: 1.75.0 → 1.87.0
```

We've overhauled the Jules development environment to move beyond the default Ubuntu 24.04 LTS packages. This includes:

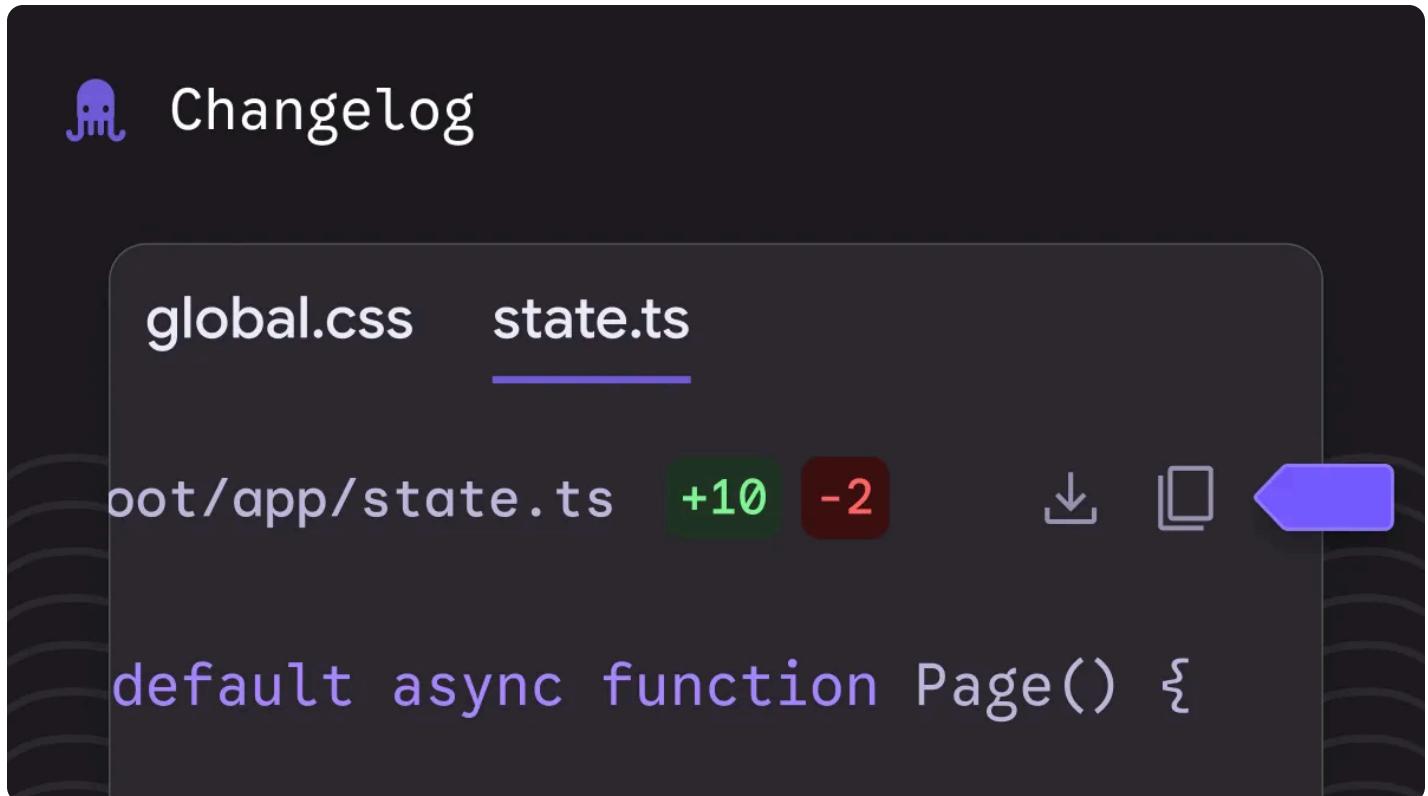
- Explicitly installing newer versions of key toolchains like Rust, Node, and Python, addressing long-standing version issues.
- Adding finer-grained control over installation steps via custom scripts instead of relying solely on apt.
- Introducing support for multiple runtimes, improved isolation, and version pinning to reduce drift and better match developer expectations.

These changes unblock several issues developers encountered with outdated dependencies and improve alignment with modern project requirements.

[Read about the Jules environment setup to learn more about what's pre-installed.](#)

Customization and Efficiency Enhancements

June 6, 2025



Performance upgrades: Enjoy a smoother, faster Jules experience with recent under-the-hood improvements.

Quickly copy and download code: New copy and download buttons are now available in the code view pane, making it easier to grab your code directly from Jules.

Stay focused with task modals: Initiate multiple tasks seamlessly through a new modal option, allowing you to keep your context and workflow intact. [Learn more](#) about kicking off tasks.

Adjustable code panel: Customize your workspace by adjusting the width of the code panel to your preferred viewing experience.

[Check out the docs](#) to learn more about how to download code that Jules writes.

A faster, smoother and more reliable Jules

May 30, 2025

This week, our focus has been on improving reliability, fixing our GitHub integration, and scaling capacity.

Here's what's we shipped:

- Updated our limits to 60 tasks per day, 5 concurrent.
- We substantially improved the reliability of the GitHub sync. Export to GitHub should also be fixed on previously created tasks.
- We've decreased the number of failure cases by 2/3

Learn more [about usage limits](#).

Improving Stability

May 22, 2025

We've been heads down improving stability and fixing bugs—big and small—to make Jules faster, smoother, and more reliable for you.

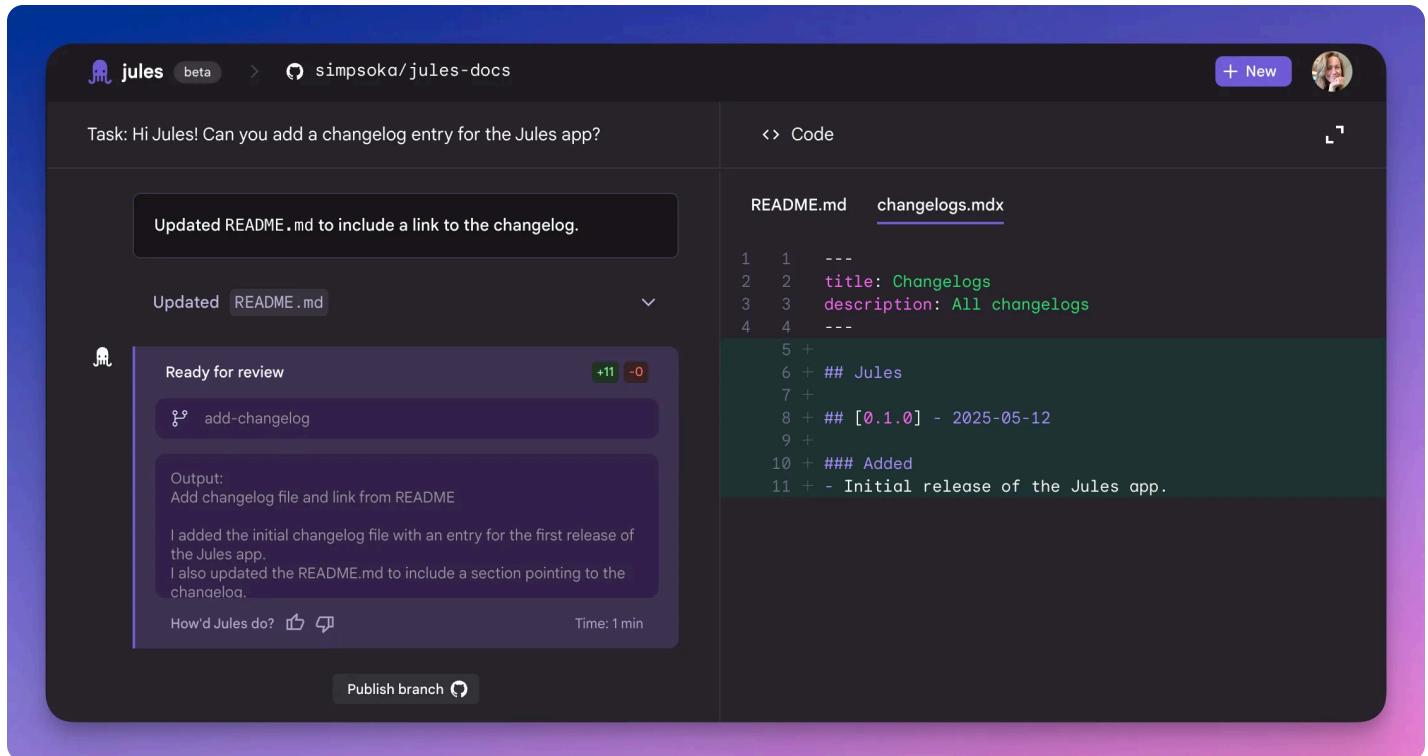
Here's what's fixed:

- Upgraded our queuing system and added more compute to reduce wait times during peak usage
- Publish Branch button is now part of the summary UI in the activity feed so it's easier to find
- Bug fixes for task status and mobile

[Learn more](#) about how to publish a branch on GitHub.

Jules is here

May 19, 2025



Today, we're launching [Jules](#), a new AI coding agent.

Jules helps you move faster by working asynchronously on tasks in your GitHub repo. It can fix bugs, update dependencies, migrate code, and add new features.

Once you give Jules a task, it spins up a fresh dev environment in a VM, installs dependencies, writes tests, makes the changes, runs the tests, and opens a pull request. Jules shows its work as it makes progress, so you never have to guess what code it's writing, or what it's thinking.

What Jules can do today

- Fix bugs with test verified patches
- Handle version bumps and dependency upgrades
- Perform scoped code transformations
- Migrate code across languages or frameworks
- Ship isolated, scoped, features
- Open PRs with runnable code and test results

[Get started with the Jules documentation](#), and visit jules.google.com to run your first Jules task.