**What is an Actor?**[**​**](https://docs.apify.com/platform/actors#what-is-an-actor)

Actors are serverless programs running in the cloud. They can perform anything from simple actions (such as filling out a web form or sending an email) to complex operations (such as crawling an entire website or removing duplicates from a large dataset). Actor runs can be as short or as long as necessary. They could last seconds, hours, or even infinitely.

**Actors are containers**[**​**](https://docs.apify.com/platform/actors#actors-are-containers)

A single isolated Actor consists of source code and various settings. You can think of an Actor as a cloud app or service that runs on the Apify platform. The run of an Actor is not limited to the lifetime of a single HTTP transaction. It can run for as long as necessary, even forever.

Basically, Actors are programs packaged as [Docker images](https://hub.docker.com/), which accept a well-defined JSON input, perform an action, and optionally produce an output.

**Actors in Store**

[**Apify Store**](https://apify.com/store)**is home to hundreds of public Actors available to the Apify community. It's the easiest way for you to start with Apify.**

**Pricing models**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#pricing-models)

All Actors in [Apify Store](https://apify.com/store) fall into one of the four pricing models:

1. [**Rental**](https://docs.apify.com/platform/actors/running/actors-in-store#rental-actors) - to continue using the Actor after the trial period, you must rent the Actor from the developer and pay a flat monthly fee in addition to the costs associated with the platform usage that the Actor generates.
2. [**Pay per result**](https://docs.apify.com/platform/actors/running/actors-in-store#pay-per-result) - you do not pay for platform usage the Actor generates and instead just pay for the results it produces.
3. [**Pay per event**](https://docs.apify.com/platform/actors/running/actors-in-store#pay-per-event) - you can run the Actor and you do not pay for platform usage the Actor generates. Instead you pay for the specific events the Actor creator defines, such as for generating a single result or starting the Actor.
4. [**Pay per usage**](https://docs.apify.com/platform/actors/running/actors-in-store#pay-per-usage) - you can run the Actor and you pay for the platform usage the Actor generates.

**Rental Actors**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#rental-actors)

Rental Actors are Actors for which you have to pay a recurring fee to the developer after your trial period ends. This empowers the developer to dedicate more time and effort to their Actors, thus ensuring they are of the *highest quality* and receive *ongoing maintenance*.

Most rental Actors have a *free trial* period. The length of the trial is displayed on each Actor's page.

fter a trial period, a flat monthly *Actor rental* fee is automatically subtracted from your prepaid platform usage in advance for the following month. Most of this fee goes directly to the developer and is paid on top of the platform usage generated by the Actor. You can read more about our motivation for releasing rental Actors in [this blog post](https://blog.apify.com/make-regular-passive-income-developing-web-automation-actors-b0392278d085/) from Apify's CEO Jan Čurn.

**Rental Actors - Frequently Asked Questions**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#rental-actors---frequently-asked-questions)

**Can I run rental Actors via API or the Apify client?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#can-i-run-rental-actors-via-api-or-the-apify-client)

Yes, when you are renting an Actor, you can run it using either our [API](https://docs.apify.com/api/v2), [JavaScript](https://docs.apify.com/api/client/js) or [Python](https://docs.apify.com/api/client/python) clients as you would do with private or free public Actors.

**Do I pay platform costs for running rental Actors?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#do-i-pay-platform-costs-for-running-rental-actors)

Yes, you will pay normal [platform usage costs](https://apify.com/pricing) on top of the monthly Actor rental fee. The platform costs work exactly the same way as for free public Actors or your private Actors. You should find estimates of the cost of usage in each individual rental Actor's README ([see an example](https://apify.com/compass/crawler-google-places#how-much-will-it-cost)).

**Do I need an Apify paid plan to use rental Actors?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#do-i-need-an-apify-paid-plan-to-use-rental-actors)

You don't need a paid plan to start a rental Actor's free trial. Just activate the trial, and you are good to go. After that, you will need to subscribe to one of [Apify's paid plans](https://apify.com/pricing) in order to keep renting the Actor and continue using it.

**When will I be charged for the Actor rental?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#when-will-i-be-charged-for-the-actor-rental)

You always prepay the Actor rental for the following month. The first payment happens when the trial expires, and then recurs monthly. When you open the Actor in the Apify Console, you will see when the next rental payment is due, and you will also receive a notification when it happens.

*Example*: You activate a 7-day trial of an Actor at *noon of April 1, 2021*. If you don't turn off auto-renewal, you will be charged at *noon on April 8, 2021*, then *May 8, 2021*.

**How am I charged for Actor rental?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#how-am-i-charged-for-actor-rental)

The rental fee for an Actor is automatically subtracted from your prepaid platform usage, similarly to, e.g. [compute units](https://docs.apify.com/platform/actors/running/usage-and-resources). If you don't have enough usage prepaid, you will need to cover any overage in the next invoice.

**Will I be automatically charged at the end of the free trial?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#will-i-be-automatically-charged-at-the-end-of-the-free-trial)

If you have an [Apify paid plan](https://apify.com/pricing), the monthly rental fee will be automatically subtracted from your plan's prepaid usage at the end of your free trial, and you will be able to run the Actor for another month. If you are not subscribed to any of [Apify's paid plans](https://apify.com/pricing), you will need to subscribe to one in order to continue using the Actor after the trial has ended.

**Can I cancel my Actor rental?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#can-i-cancel-my-actor-rental)

*You can cancel the Actor rental* during your trial or any time after that so you don't get charged when your current Actor rental period expires. You can always turn it back on later if you want.

**Where can I see how much I have paid for Actor rental?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#where-can-i-see-how-much-i-have-paid-for-actor-rental)

Since Actor rental fees are paid from prepaid platform usage, these fees conceptually belong under platform usage.

You can find the breakdown of how much you have been charged for rental Actors in the **Actors** tab, which you will find within the **Current period** tab in the [Billing](https://console.apify.com/billing) section.

P**ay per result**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#pay-per-result)

When you run an Actor that is *paid per result*, you pay for the successful results that an Actor returns when you run it, and you are not charged for the underlying platform usage.

**Estimation simplified**

This makes it transparent and easy to estimate upfront costs. If you have any feedback or would like to ask something, please join our [Discord](https://discord.gg/qkMS6pU4cF) community and let us know!

**Pay per result Actors - Frequently Asked Questions**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#pay-per-result-actors---frequently-asked-questions)

**How do I know an Actor is paid per result?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#how-do-i-know-an-actor-is-paid-per-result)

When you try the Actor on the platform, you will see that the Actor is paid per result next to the Actor name.

**Do I need to pay a monthly rental fee to run the Actor?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#do-i-need-to-pay-a-monthly-rental-fee-to-run-the-actor)

No, the Actor is free to run. You only pay for the results.

**What happens when I interact with the dataset after the run finishes?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#what-happens-when-i-interact-with-the-dataset-after-the-run-finishes)

Under the **pay per result** model, all platform costs generated *during the run of an Actor* are not charged towards your account; you pay for the results instead. After the run finishes, any interactions with the default dataset storing the results, such as reading the results or writing additional data, will incur the standard platform usage costs. But do not worry, in the vast majority of cases, you only want to read the result from the dataset and that costs near to nothing.

**Do I pay for the storage of results on the Apify platform?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#do-i-pay-for-the-storage-of-results-on-the-apify-platform)

You will still be charged for the timed storage of the data in the same fashion as with any other Actor. You can always decide to delete the dataset to reduce your costs after you export the data from the platform. By default, any unnamed dataset will be automatically removed after your data retention period, so usually, this is nothing to worry about.

**Can I set a cap on how many results an Actor should return?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#can-i-set-a-cap-on-how-many-results-an-actor-should-return)

You can set a limit on how many items an Actor should return and the amount you will be charged in Options on the Actor detail page in the section below the Actor input.

**Can I publish an Actor that is paid per result?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#can-i-publish-an-actor-that-is-paid-per-result)

Yes, you can publish an Actor that is paid per result.

**Where do I see how much I was charged for the pay per result Actors?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#where-do-i-see-how-much-i-was-charged-for-the-pay-per-result-actors)

You can see the overview of how much you have been charged for Actors paid by result on your invoices and in the [Usage tab](https://console.apify.com/billing) of the Billing section in Console. It will be shown there as a separate service.

On the top of that, you can see how much you have been charged for a specific run in the detail of that run and also in the overview table showing all runs.

**Pay per event**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#pay-per-event)

Pay per event Actor pricing model is very similar to the pay per result model. You still do not pay the underlying platform usage. Instead of results, you pay for specific events defined by the creator of the Actor. These events will vary between Actors, and will always be described, together with their pricing, on each Actor. Example events might be producing a single result, doing a unit piece of work (e.g. uploading a file) or starting an Actor.

**Pay per event Actors - Frequently Asked Questions**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#pay-per-event-actors---frequently-asked-questions)

**How do I know Actor is paid per events?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#how-do-i-know-actor-is-paid-per-events)

You will the that the Actor is paid per events next to the Actor name.

**Do I need to pay a monthly rental fee to run the Actor?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#do-i-need-to-pay-a-monthly-rental-fee-to-run-the-actor-1)

No, you only pay for the events.

**What happens when I interact with the dataset after the run finishes?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#what-happens-when-i-interact-with-the-dataset-after-the-run-finishes-1)

You would still pay for all interactions after the Actor run finishes, same as for pay per result Actors.

**Do I pay for the storage of results on the Apify platform?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#do-i-pay-for-the-storage-of-results-on-the-apify-platform-1)

You would still pay for the long term storage of results, same as for pay per result Actors.

**Where do I see how much I was charged for the pay per result Actors?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#where-do-i-see-how-much-i-was-charged-for-the-pay-per-result-actors-1)

Similarly to pay per result Actors, you can see how much you have been charged on your invoices, and on the [Usage tab](https://console.apify.com/billing) of the Billing section in the Console.

**Can I put a cap on a cost of a single Actor run?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#can-i-put-a-cap-on-a-cost-of-a-single-actor-run)

Yes, when starting an Actor run, you can define the maximum limit on the cost of that run. When the Actor reaches the defined limit, it should terminate gracefully. Even if it didn't, for any reason, and kept producing results, we make always sure you are never charged more that your defined limit.

**How do I raise a dispute if the charges for an Actor seem off?**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#how-do-i-raise-a-dispute-if-the-charges-for-an-actor-seem-off)

Please, in such a case, do not hesitate to contact the Actor author or our support team. If you suspect a bug in the Actor, you can also always create an issue on the Actor detail in the Apify Console.

**Pay per usage**[**​**](https://docs.apify.com/platform/actors/running/actors-in-store#pay-per-usage)

When you use a pay per usage Actor, you are only charged for the platform usage that the runs of this Actor generate. [Platform usage](https://docs.apify.com/platform/actors/running/usage-and-resources) includes components such as compute units, operations on [storages](https://docs.apify.com/platform/storage), and usage of [residential proxies](https://docs.apify.com/platform/proxy/residential-proxy) or [SERPs](https://docs.apify.com/platform/proxy/google-serp-proxy).

**Estimating Actor usage cost**

With this model, it's very easy to see how many platform resources each Actor run consumed, but it is quite difficult to estimate their usage beforehand. The best way to find the costs of free Actors upfront is to try out the Actor on a limited scope (for example, on a small number of pages) and evaluate the consumption. You can easily do that using our [free plan](https://apify.com/pricing).

**nput and output**

**Configure your Actor's input parameters using Apify Console, locally or via API. Access parameters in key-value stores from your Actor's code.**

**Input**[**​**](https://docs.apify.com/platform/actors/running/input-and-output#input)

Each Actor accepts input, which tells it what to do. You can run an Actor using the [Apify Console](https://console.apify.com/) UI, then configure the input using the autogenerated UI:

When running an Actor using the [API](https://docs.apify.com/api/v2) you can pass the same input as the JSON object. In this case, the corresponding JSON input looks as follows:

{  
 "maxRequestsPerCrawl": 10,  
 "proxy": {  
 "useApifyProxy": true  
 },  
 "startUrl": "https://apify.com"  
}

**Options - Build, Timeout, and Memory**[**​**](https://docs.apify.com/platform/actors/running/input-and-output#options---build-timeout-and-memory)

As part of the input, you can also specify run options such as [Build](https://docs.apify.com/platform/actors/development/builds-and-runs/builds), Timeout, and [Memory](https://docs.apify.com/platform/actors/running/usage-and-resources) for your Actor run

| **Option** | **Description** |
| --- | --- |
| Build | Tag or number of the build to run (e.g. **latest** or **1.2.34**). |
| Timeout | Timeout for the Actor run in seconds. Zero value means there is no timeout. |
| Memory | Amount of memory allocated for the Actor run, in megabytes. |

**Output**[**​**](https://docs.apify.com/platform/actors/running/input-and-output#output)

While the input object provides a way to instruct Actors, an Actor can also generate an output, usually stored in its default [Dataset](https://docs.apify.com/platform/storage/dataset), but some additional files might be stored in its [Key-value store](https://docs.apify.com/platform/storage/key-value-store). Always read the Actor's README to learn more about its output.

For more details about storages, visit the [Storage](https://docs.apify.com/platform/storage) section.

You can quickly access the Actor's output from the run detail page:

and to access all the data associated with the run, see the **Storage** tab, where you can explore the Actor's default [Dataset](https://docs.apify.com/platform/storage/dataset), [Key-value store](https://docs.apify.com/platform/storage/key-value-store), and [Request queue](https://docs.apify.com/platform/storage/request-queue):

**Runs and builds**

**Learn about Actor builds and runs, their lifecycle, sharing, and data retention policy.**

**Builds**[**​**](https://docs.apify.com/platform/actors/running/runs-and-builds#builds)

An Actor is a combination of source code and various settings in a Docker container. To run, it needs to be built. An Actor build consists of the source code built as a Docker image, making the Actor ready to run on the Apify platform.

**What is Docker image?**

A Docker image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings. For more information visit Docker's [site](https://www.docker.com/resources/what-container/).

With every new version of an Actor, a new build is created. Each Actor build has its number (for example, **1.2.34**), and some builds are tagged for easier use (for example, *latest* or *beta*). When running an Actor, you can choose what build you want to run by selecting a tag or number in the run options.

Each build may have different features, input, or output. By fixing the build to an exact version, you can ensure that you won't be affected by a breaking change in a new Actor version. However, you will lose updates.

**Runs**[**​**](https://docs.apify.com/platform/actors/running/runs-and-builds#runs)

When you start an Actor, an Actor run is created. An Actor run is a Docker container created from the build's Docker image with dedicated resources (CPU, memory, disk space). For more on this topic, see [Usage and resources](https://docs.apify.com/platform/actors/running/usage-and-resources).

Each run has its own (default) [storages](https://docs.apify.com/platform/storage) assigned, which it may but not necessarily need to use:

* [Key-value store](https://docs.apify.com/platform/storage/key-value-store) containing the input and enabling Actor to store other files.
* [Dataset](https://docs.apify.com/platform/storage/dataset) enabling Actor to store the results.
* [Request queue](https://docs.apify.com/platform/storage/request-queue) to maintain a queue of URLs to be processed.

What's happening inside of an Actor is visible on the Actor run log in the Actor run detail:

**Origin**[**​**](https://docs.apify.com/platform/actors/running/runs-and-builds#origin)

Both **Actor runs** and **builds** have the **Origin** field indicating how the Actor run or build was invoked, respectively. The origin is displayed in Apify Console and available via [API](https://docs.apify.com/api/v2/actor-run-get) in the meta.origin field.

| **Name** | **Origin** |
| --- | --- |
| DEVELOPMENT | Manually from Apify Console in the Development mode (own Actor) |
| WEB | Manually from Apify Console in "normal" mode (someone else's Actor or task) |
| API | From [Apify API](https://docs.apify.com/api) |
| CLI | From [Apify CLI](https://docs.apify.com/cli/) |
| SCHEDULER | Using a schedule |
| WEBHOOK | Using a webhook |
| ACTOR | From another Actor run |
| STANDBY | From [Actor Standby](https://docs.apify.com/platform/actors/running/standby) |

**Lifecycle**[**​**](https://docs.apify.com/platform/actors/running/runs-and-builds#lifecycle)

Each run and build starts with the initial status **READY** and goes through one or more transitional statuses to one of the terminal statuses.

Terminal states

Transitional states

RUNNING

TIMING-OUT

ABORTING

SUCCEEDED

FAILED

TIMED-OUT

ABORTED

READY

| **Status** | **Type** | **Description** |
| --- | --- | --- |
| READY | initial | Started but not allocated to any worker yet |
| RUNNING | transitional | Executing on a worker machine |
| SUCCEEDED | terminal | Finished successfully |
| FAILED | terminal | Run failed |
| TIMING-OUT | transitional | Timing out now |
| TIMED-OUT | terminal | Timed out |
| ABORTING | transitional | Being aborted by the user |
| ABORTED | terminal | Aborted by the user |

**Aborting runs**[**​**](https://docs.apify.com/platform/actors/running/runs-and-builds#aborting-runs)

You can abort runs with the statuses **READY**, **RUNNING**, or **TIMING-OUT** in two ways:

* *Immediately* - this is the default option. The Actor process is killed immediately with no grace period.
* *Gracefully* - the Actor run receives a signal about aborting via the aborting event and is granted a 30-second window to finish in-progress tasks before getting aborted. This is helpful in cases where you plan to resurrect the run later because it gives the Actor a chance to persist its state. When resurrected, the Actor can restart where it left off.

You can abort a run in Apify Console using the **Abort** button or via API using the [Abort run](https://docs.apify.com/api/v2#/reference/actor-runs/abort-run/abort-run) endpoint.

**Resurrection of finished run**[**​**](https://docs.apify.com/platform/actors/running/runs-and-builds#resurrection-of-finished-run)

Any Actor run in a terminal state, i.e., run with status **FINISHED**, **FAILED**, **ABORTED**, and **TIMED-OUT**, might be resurrected back to a **RUNNING** state. This is helpful in many cases, for example, when the timeout for an Actor run was too low or in case of an unexpected error.

The whole process of resurrection looks as follows:

* Run status will be updated to **RUNNING**, and its container will be restarted with the same storage (the same behavior as when the run gets migrated to the new server).
* Updated duration will not include the time when the Actor was not running.
* Timeout will be counted from the point when this Actor run was resurrected.

Resurrection can be performed in Apify Console using the **resurrect** button or via API using the [Resurrect run](https://docs.apify.com/api/v2#/reference/actors/resurrect-run) API endpoint.

**Settings adjustments**

You can also adjust timeout and memory or change Actor build before the resurrection. This is especially helpful in case of an error in the Actor's source code as it enables you to:

1. Abort a broken run
2. Update the Actor's code and build the new version
3. Resurrect the run using the new build

**Data retention**[**​**](https://docs.apify.com/platform/actors/running/runs-and-builds#data-retention)

Apify securely stores your ten most recent runs indefinitely, ensuring your records are always accessible. All **Actor runs** beyond the latest ten are deleted along with their default storages (Key-value store, Dataset, Request queue) after the data retention period based on your [subscription plan](https://apify.com/pricing).

**Actor builds** are deleted only when they are *not tagged* and have not been used for over 90 days.

**Usage and resources**

**Learn about your Actors' memory and processing power requirements, their relationship with Docker resources, minimum requirements for different use cases and its impact on the cost.**

**Resources**[**​**](https://docs.apify.com/platform/actors/running/usage-and-resources#resources)

[Actors](https://docs.apify.com/platform/actors) run in [Docker containers](https://www.docker.com/resources/what-container/), which have a [limited amount of resources](https://phoenixnap.com/kb/docker-memory-and-cpu-limit) (memory, CPU, disk size, etc). When starting, the Actor needs to be allocated a certain share of those resources, such as CPU capacity that is necessary for the Actor to run.

Assigning an Actor a specific **Memory** capacity, also determines the allocated CPU power and its disk size.

Check out the [Limits](https://docs.apify.com/platform/limits) page for detailed information on Actor memory, CPU limits, disk size and other limits.

**Memory**[**​**](https://docs.apify.com/platform/actors/running/usage-and-resources#memory)

When invoking an Actor, the caller must specify the memory allocation for the Actor run. The memory allocation must follow these requirements:

* It must be a power of 2.
* The minimum allowed value is 128MB
* The maximum allowed value is 32768MB
* Acceptable values include: 128MB, 256MB, 512MB, 1024MB, 2048MB, 4096MB, 8192MB, 16384MB, and 32768MB

Additionally, each user has a certain total limit of memory for running Actors. The sum of memory allocated for all running Actors and builds needs to be within this limit, otherwise the user cannot start a new Actor. For more details, see [limits](https://docs.apify.com/platform/limits).

**CPU**[**​**](https://docs.apify.com/platform/actors/running/usage-and-resources#cpu)

The CPU allocation for an Actor is automatically computed based on the assigned memory, following these rules:

* For every 4096MB of memory, the Actor receives one full CPU core
* If the memory allocation is not a multiple of 4096MB, the CPU core allocation is calculated proportionally
* Examples:
  + 512MB = 1/8 of a CPU core
  + 1024MB = 1/4 of a CPU core
  + 8192MB = 2 CPU cores

**CPU usage spikes**

Sometimes, you see the Actor's CPU use go over 100%. This is not unusual. To help an Actor start up faster, it is allocated a free CPU boost. For example, if an Actor is assigned 1GB (25% of a core), it will temporarily be allowed to use 100% of the core, so it gets started quicker.

**Disk**[**​**](https://docs.apify.com/platform/actors/running/usage-and-resources#disk)

The Actor has hard disk space limited by twice the amount of memory. For example, an Actor with 1024MB of memory will have 2048MB of disk available.

**Requirements**[**​**](https://docs.apify.com/platform/actors/running/usage-and-resources#requirements)

Actors built with [Crawlee](https://crawlee.dev/) use autoscaling. This means that they will always run as efficiently as they can based on the allocated memory. If you double the allocated memory, the run should be twice as fast and consume the same amount of compute units (1 \* 1 = 0.5 \* 2).

A good middle ground is 4096MB. If you need the results faster, increase the memory (bear in mind the [next point](https://docs.apify.com/platform/actors/running/usage-and-resources#maximum-memory), though). You can also try decreasing it to lower the pressure on the target site.

Autoscaling only applies to solutions that run multiple tasks (URLs) for at least 30 seconds. If you need to scrape just one URL or use Actors like [Google Sheets](https://apify.com/lukaskrivka/google-sheets) that do just a single isolated job, we recommend you lower the memory.

If the Actor doesn't have this information, or you want to use your own solution, just run your solution like you want to use it long term. Let's say that you want to scrape the data **every hour for the whole month**. You set up a reasonable memory allocation like 4096MB, and the whole run takes 15 minutes. That should consume 1 CU (4 \* 0.25 = 1). Now, you just need to multiply that by the number of hours in the day and by the number of days in the month, and you get an estimated usage of 720 (1 \* 24 \* 30) CUs monthly.

**Estimating usage**

Check out our article on [estimating consumption](https://help.apify.com/en/articles/3470975-how-to-estimate-compute-unit-usage-for-your-project) for more details.

**Memory requirements**[**​**](https://docs.apify.com/platform/actors/running/usage-and-resources#memory-requirements)

Each use case has its own memory requirements. The larger and more complex your project, the more memory/CPU power it will require. Some examples which have minimum requirements are:

* Actors using [Puppeteer](https://pptr.dev/) or [Playwright](https://playwright.dev/) for real web browser rendering require at least 1024MB of memory.
* Large and complex sites like [Google Maps](https://apify.com/compass/crawler-google-places) require at least 4096MB for optimal speed and [concurrency](https://crawlee.dev/api/core/class/AutoscaledPool#minConcurrency).
* Projects involving large amount of data in memory.

**Maximum memory**[**​**](https://docs.apify.com/platform/actors/running/usage-and-resources#maximum-memory)

Apify Actors are most commonly written in [Node.js](https://nodejs.org/en/), which uses a [single thread process](https://dev.to/arealesramirez/is-node-js-single-threaded-or-multi-threaded-and-why-ab1). Unless you use external binaries such as the Chrome browser, Puppeteer, Playwright, or other multi-threaded libraries you will not gain more CPU power from assigning your Actor more than 4096MB of memory because Node.js cannot use more than 1 core.

In other words, giving a [Cheerio-based crawler](https://apify.com/apify/cheerio-scraper) 16384MB of memory (4 CPU cores) will not improve its performance, because these crawlers cannot use more than 1 CPU core.

**Multi-threaded Node.js configuration**

It's possible to [use multiple threads in Node.js-based Actor](https://dev.to/reevranj/multiple-threads-in-nodejs-how-and-what-s-new-b23) with some configuration. This can be useful if you need to offload a part of your workload.

**Usage**[**​**](https://docs.apify.com/platform/actors/running/usage-and-resources#usage)

When you run an Actor it generates platform usage that's charged to the user account. Platform usage comprises four main parts:

* **Compute units**: CPU and memory resources consumed by the Actor.
* **Data transfer**: The amount of data transferred between the web, Apify platform, and other external systems.
* **Proxy costs**: Residential or SERP proxy usage.
* **Storage operations**: Read, write, and other operations performed on the Key-value store, Dataset, and Request queue.

**hat is a compute unit**[**​**](https://docs.apify.com/platform/actors/running/usage-and-resources#what-is-a-compute-unit)

A compute unit (CU) is the unit of measurement for the resources consumed by Actor runs and builds. You are charged for using Actors based on CU consumption.

For example, running an Actor with1024MB of allocated memory for 1 hour will consume 1 CU. The cost of this CU depends on your subscription plan.

You can check each Actor run's exact CU usage in the run's details.

**Compute unit calculation**[**​**](https://docs.apify.com/platform/actors/running/usage-and-resources#compute-unit-calculation)

CUs are calculated by multiplying two factors:

* **Memory** (MB) - The size of the allocated server for your Actor or task run.
* **Duration** (hours) - The duration for which the server is used (Actor or task run). For example, if your run took 6 minutes, you would use 0.1 (hours) as the second number to calculate CUs. The minimum granularity is a second.

Example: *1024MB memory x 1 hour = 1 CU*

**What determines consumption**[**​**](https://docs.apify.com/platform/actors/running/usage-and-resources#what-determines-consumption)

The factors that influence resource consumption, in order of importance, are:

* *Browser vs. Plain HTTP*: Launching a browser (e.g., [Puppeteer](https://pptr.dev/)/[Playwright](https://playwright.dev/)) is resource-intensive and slower compared to working with plain HTML ([Cheerio](https://cheerio.js.org/)). Using Cheerio can be up to *20 times* faster.
* *Run size and frequency*: Large runs can use full resource scaling and are not subjected to repeated Actor start-ups (as opposed to many short runs). Whenever possible, opt for larger batches.
* *Page type*: Heavy pages, such as Amazon or Facebook will take more time to load regardless whether you use a browser or Cheerio. Large pages can take up to *3 times* more resources to load and parse than average pages.

**Development in web IDE**

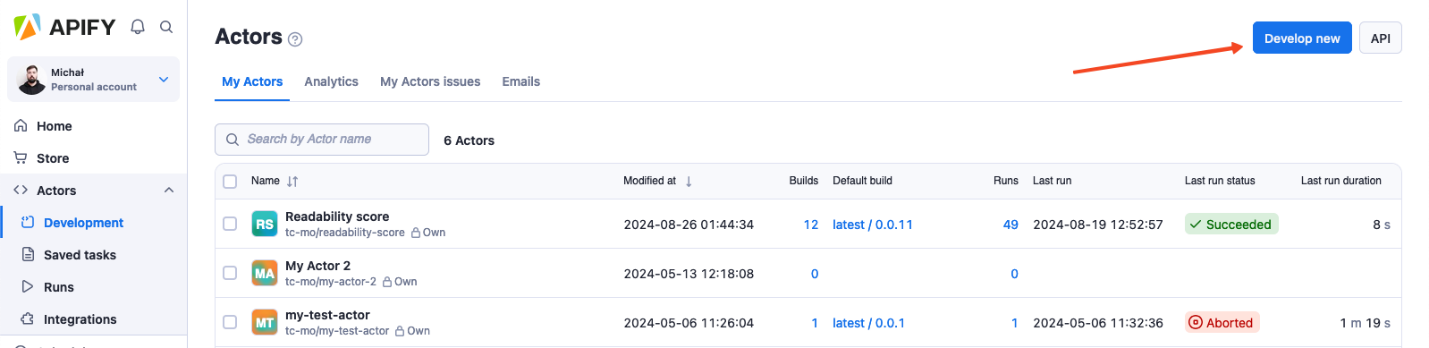
**Create your first Actor using the web IDE in Apify Console.**

**Create the Actor**[**​**](https://docs.apify.com/platform/actors/development/quick-start/web-ide#create-the-actor)

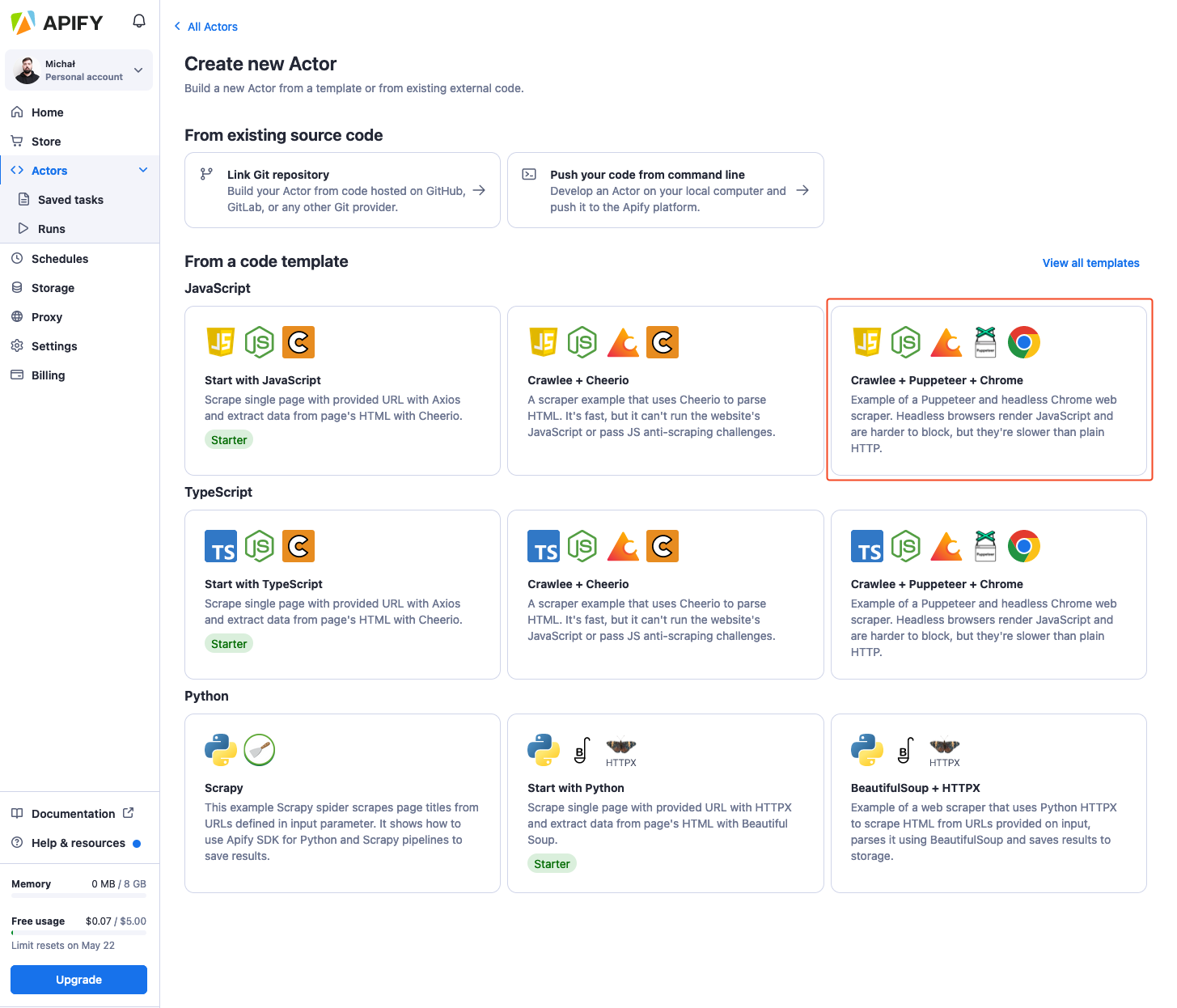
**Prerequisites**

To use Web IDE, you will need an Apify account. You can [sign-up for a free account](https://console.apify.com/sign-up) on the Apify website.

After you sign in to [Apify Console](https://console.apify.com/), navigate to the [**Development**](https://console.apify.com/actors/development/my-actors) subsection. Then, click the **Develop new** button at the top right corner of the page.



You will be redirected to a page containing various Actor development templates for popular languages such as JavaScript, TypeScript, and Python. These templates provide boilerplate code and a preconfigured environment tailored to specific use cases. You can choose the template that best suits your technology stack. For demonstration purposes, let's choose **Crawlee + Puppeteer + Chrome**.



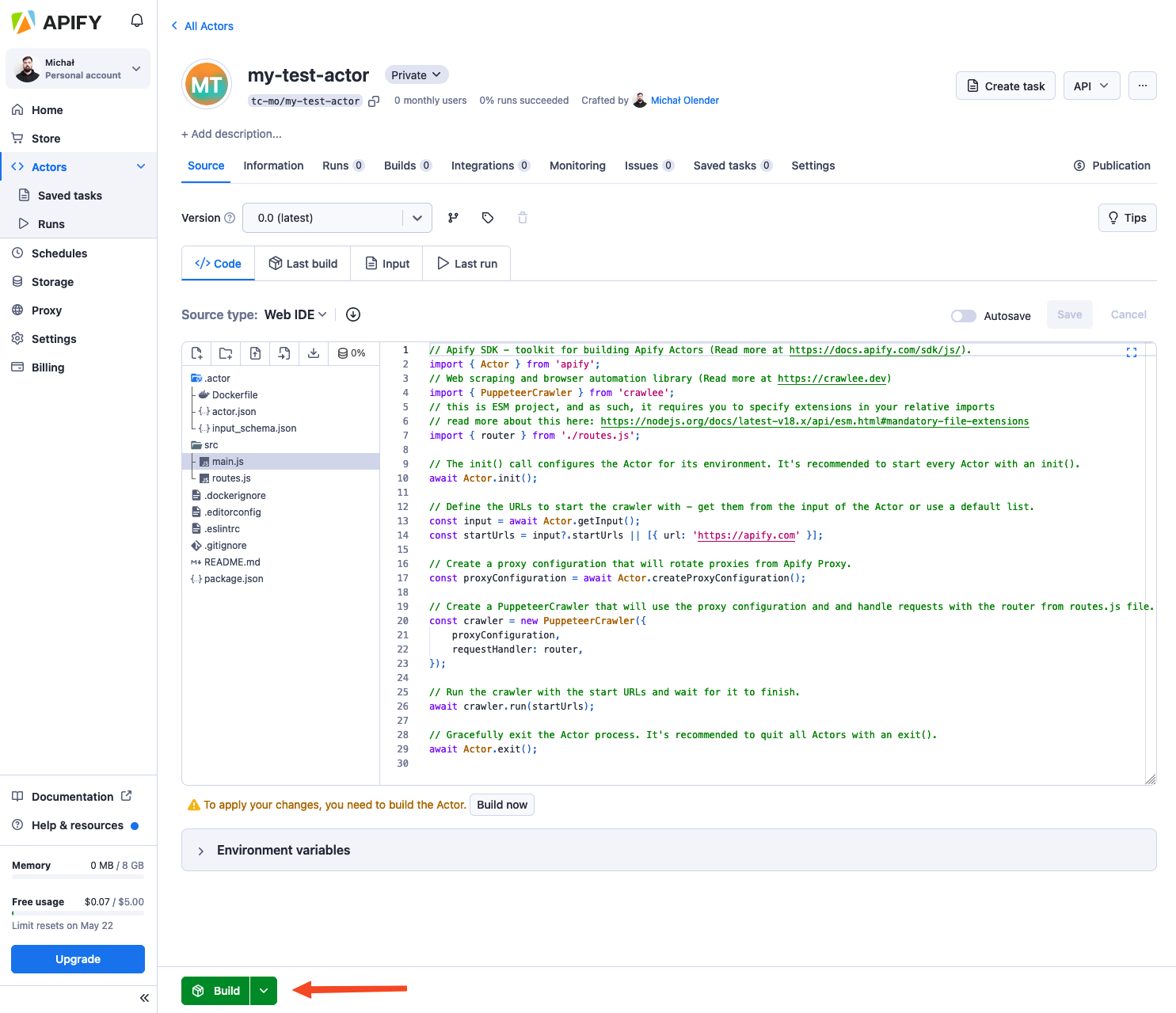
After choosing the template your Actor will be automatically named and you will be redirected to its page.

**Explore the source code**[**​**](https://docs.apify.com/platform/actors/development/quick-start/web-ide#explore-the-source-code)

The provided boilerplate code utilizes the [Apify SDK](https://docs.apify.com/sdk/js/) combined with [Crawlee](https://crawlee.dev/), Apify's popular open-source Node.js web scraping library. By default the code performs a recursive crawl of the [apify.com](https://apify.com/) website, but you can change it to a website of your choosing.

**Crawlee**

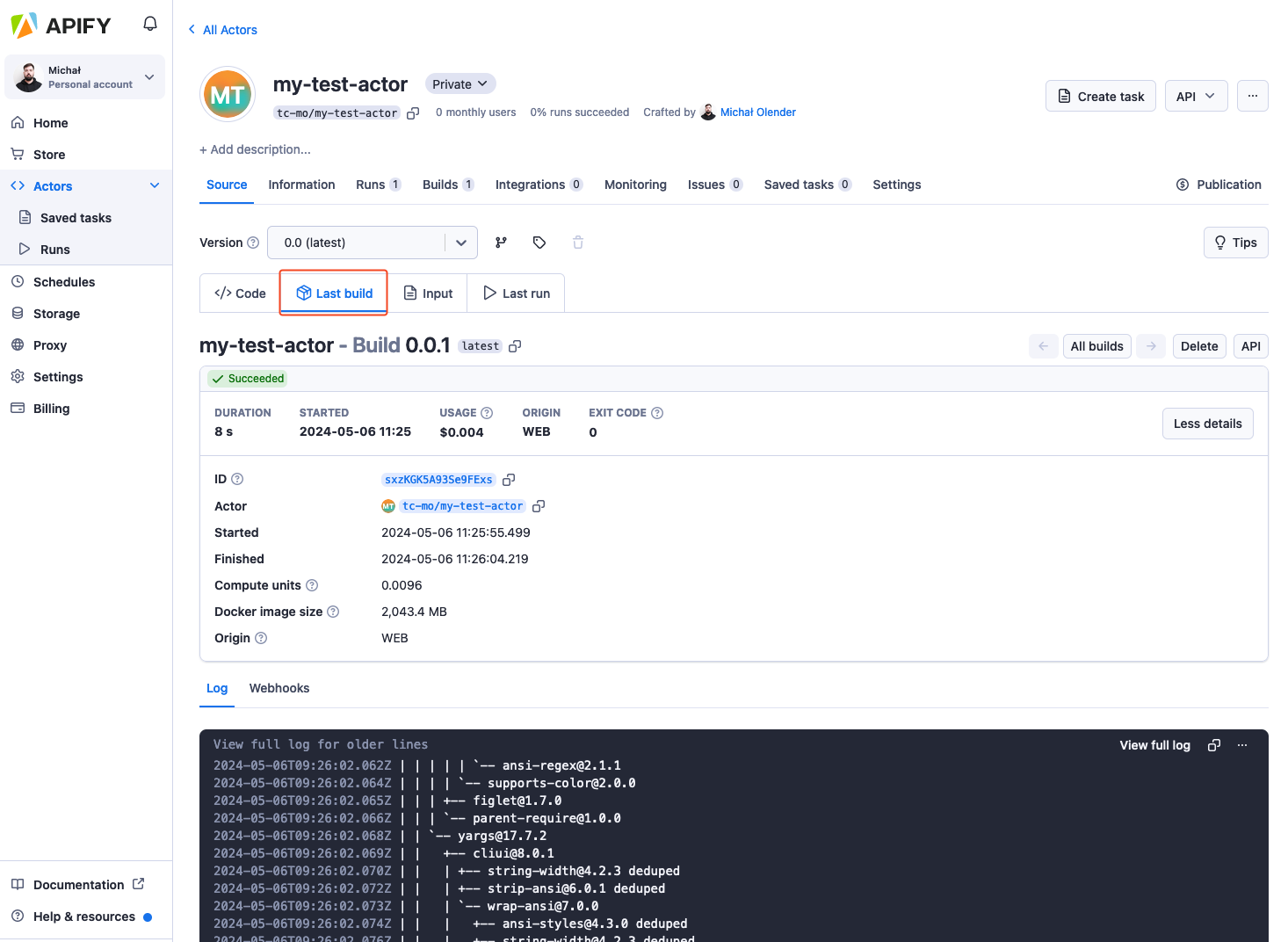
[Crawlee](https://crawlee.dev/) is an open-source Node.js library designed for web scraping and browser automation. It helps you build reliable crawlers quickly and efficiently.



**Build the Actor**[**​**](https://docs.apify.com/platform/actors/development/quick-start/web-ide#build-the-actor)

To run your Actor, you need to build it first. Click the **Build** button below the source code to start the build process.

Once the build has been initiated, the UI will transition to the **Last build** tab, displaying the progress of the build and the Docker build log.



**Actor creation flow**

The UI includes four tabs:

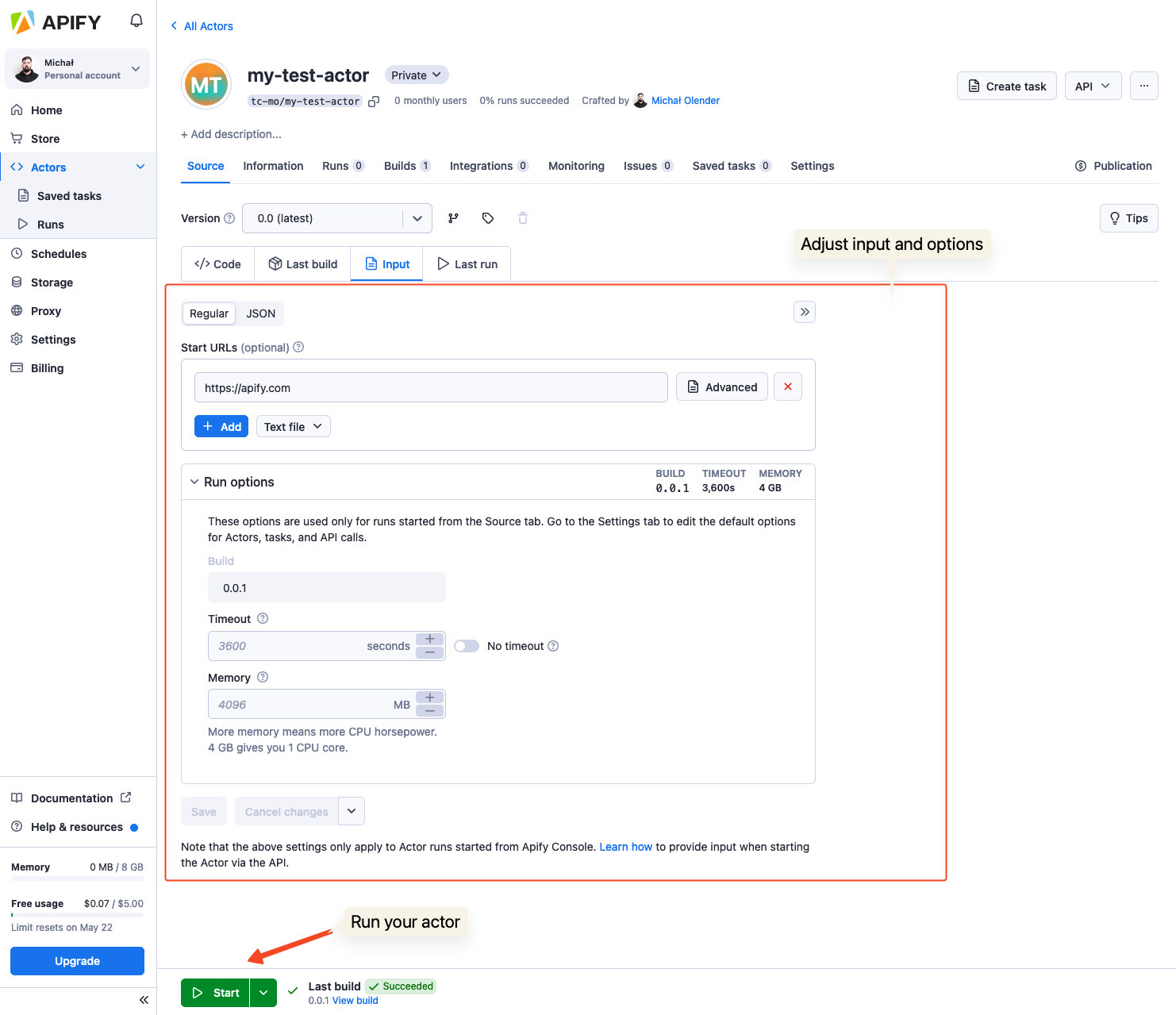
* **Code**
* **Last build**
* **Input**
* **Last Run**

This represents the Actor creation flow, where you first build the Actor from the source code. Once the build is successful, you can provide input parameters and initiate an Actor run.

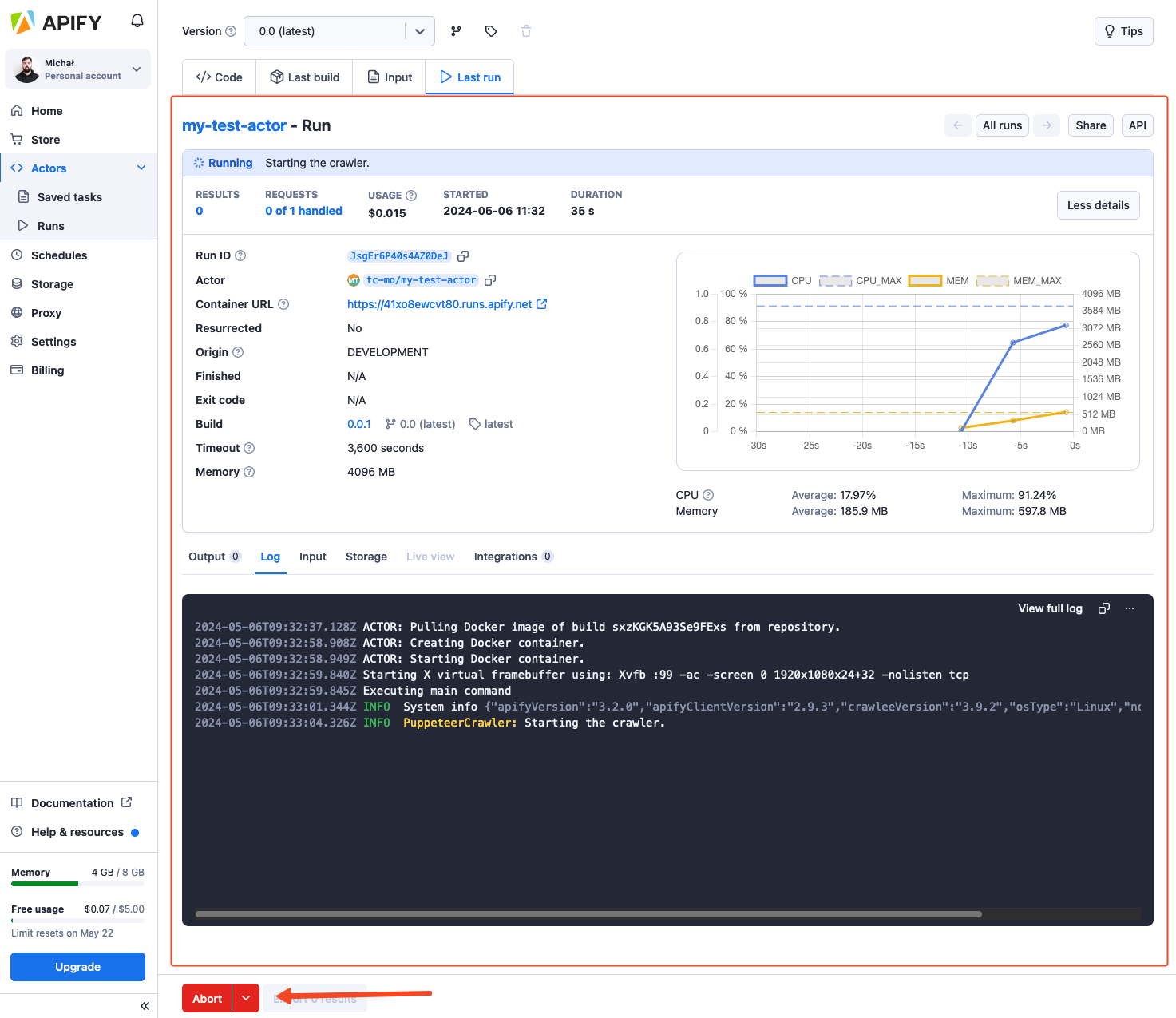
**Run the Actor**[**​**](https://docs.apify.com/platform/actors/development/quick-start/web-ide#run-the-actor)

Once the Actor is built, you can look at its input, which consists of one field - **Start URL**, the URL where the crawling starts. Below the input, you can adjust the **Run options**:

* **Build**
* **Timeout**
* **Memory limit**



To initiate an Actor run, click the **Start** button at the bottom of the page. Once the run is created, you can monitor its progress and view the log in real-time. The **Output** tab will display the results of the Actor's execution, which will be populated as the run progresses. You can abort the run at any time using the **Abort** button.



**Pull the Actor**[**​**](https://docs.apify.com/platform/actors/development/quick-start/web-ide#pull-the-actor)

To continue development locally, you can pull the Actor's source code to your local machine.

**Prerequisites**

Install [apify-cli](https://docs.apify.com/cli/) :

* **macOS/Linux**
* **Other platforms**

brew install apify-cli

To pull your Actor, you need to:

1. Log in to the Apify platform

apify login

1. Pull your Actor using the following command:

apify pull your-actor-name

Or with a specific version:

apify pull your-actor-name --version [version\_number]

As your-actor-name, you can use either:

* + The unique name of the Actor (e.g., apify/hello-world)
  + The ID of the Actor (e.g., E2jjCZBezvAZnX8Rb)

You can find both by clicking on the Actor title at the top of the page, which will open a new window containing the Actor's unique name and ID.

**Iterate & customize**[**​**](https://docs.apify.com/platform/actors/development/quick-start/web-ide#iterate--customize)

After pulling the Actor's source code to your local machine, you can modify and customize it to match your specific requirements. Leverage your preferred code editor or development environment to make the necessary changes and enhancements.

Once you've made the desired changes, you can push the updated code back to the Apify platform for deployment & execution, leveraging the platform's scalability and reliability.

To learn more about the Apify platform's features and best practices for Actor development:

* Continue to the next chapter of this section for in-depth guidance and examples
* Visit the [Apify Academy](https://docs.apify.com/academy) to access a comprehensive collection of tutorials, documentation, and learning resources.

**actor.json**

**Learn how to write the main Actor configuration in the .actor/actor.json file.**

Your main Actor configuration is in the actor/actor.json file at the root of your Actor's directory. This file links your local development project to an Actor on the Apify platform. It should include details like the Actor's name, version, build tag, and environment variables. Make sure to commit this file to your Git repository.

For example, the .actor/actor.json file can look like this:

* **Full actor.json**
* **Minimal actor.json**

{  
 "actorSpecification": 1, *// always 1*  
 "name": "name-of-my-scraper",  
 "version": "0.0",  
 "buildTag": "latest",  
 "minMemoryMbytes": 256,  
 "maxMemoryMbytes": 4096,  
 "environmentVariables": {  
 "MYSQL\_USER": "my\_username",  
 "MYSQL\_PASSWORD": "@mySecretPassword"  
 },  
 "usesStandbyMode": false,  
 "dockerfile": "./Dockerfile",  
 "readme": "./ACTOR.md",  
 "input": "./input\_schema.json",  
 "storages": {  
 "dataset": "./dataset\_schema.json"  
 }  
}

**Reference**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/actor-json#reference)

**Deployment metadata**

Actor name, version, buildTag, and environmentVariables are currently only used when you deploy your Actor using the [Apify CLI](https://docs.apify.com/cli) and not when deployed, for example, via GitHub integration. There, it serves for informative purposes only.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| actorSpecification | Required | The version of the Actor specification. This property must be set to 1, which is the only version available. |
| name | Required | The name of the Actor. |
| version | Required | The version of the Actor, specified in the format [Number].[Number], e.g., 0.1, 0.3, 1.0, 1.3, etc. |
| buildTag | Optional | The tag name to be applied to a successful build of the Actor. If not specified, defaults to latest. Refer to the [builds](https://docs.apify.com/platform/actors/development/builds-and-runs/builds) for more information. |
| environmentVariables | Optional | A map of environment variables to be used during local development. These variables will also be applied to the Actor when deployed on the Apify platform. For more details, see the [environment variables](https://docs.apify.com/cli/docs/vars) section of Apify CLI documentation. |
| dockerfile | Optional | The path to the Dockerfile to be used for building the Actor on the platform. If not specified, the system will search for Dockerfiles in the .actor/Dockerfile and Dockerfile paths, in that order. Refer to the [Dockerfile](https://docs.apify.com/platform/actors/development/actor-definition/dockerfile) section for more information. |
| dockerContextDir | Optional | The path to the directory to be used as the Docker context when building the Actor. The path is relative to the location of the actor.json file. This property is useful for monorepos containing multiple Actors. Refer to the [Actor monorepos](https://docs.apify.com/platform/actors/development/deployment/source-types#actor-monorepos) section for more details. |
| readme | Optional | The path to the README file to be used on the platform. If not specified, the system will look for README files in the .actor/README.md and README.md paths, in that order of preference. Check out [Apify Marketing Playbook to learn how to write a quality README files](https://apify.notion.site/How-to-create-an-Actor-README-759a1614daa54bee834ee39fe4d98bc2) guidance. |
| input | Optional | You can embed your [input schema](https://docs.apify.com/platform/actors/development/actor-definition/input-schema) object directly in actor.json under the input field. You can also provide a path to a custom input schema. If not provided, the input schema at .actor/INPUT\_SCHEMA.json or INPUT\_SCHEMA.json is used, in this order of preference. |
| changelog | Optional | The path to the CHANGELOG file displayed in the Information tab of the Actor in Apify Console next to Readme. If not provided, the CHANGELOG at .actor/CHANGELOG.md or CHANGELOG.md is used, in this order of preference. Your Actor doesn't need to have a CHANGELOG but it is a good practice to keep it updated for published Actors. |
| storages.dataset | Optional | You can define the schema of the items in your dataset under the storages.dataset field. This can be either an embedded object or a path to a JSON schema file. [Read more](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema) about Actor dataset schemas. |
| minMemoryMbytes | Optional | Specifies the minimum amount of memory in megabytes required by the Actor to run. Requires an *integer* value. If both minMemoryMbytes and maxMemoryMbytes are set, then minMemoryMbytes must be equal or lower than maxMemoryMbytes. Refer to the [Usage and resources](https://docs.apify.com/platform/actors/running/usage-and-resources#memory) for more details about memory allocation. |
| maxMemoryMbytes | Optional | Specifies the maximum amount of memory in megabytes required by the Actor to run. It can be used to control the costs of run, especially when developing pay per result Actors. Requires an *integer* value. Refer to the [Usage and resources](https://docs.apify.com/platform/actors/running/usage-and-resources#memory) for more details about memory allocation. |
| usesStandbyMode | Optional | Boolean specifying whether the Actor will have [Standby mode](https://docs.apify.com/platform/actors/development/programming-interface/standby) enabled. |

**Actor input schema specification**

**Learn how to define and validate a schema for your Actor's input with code examples. Provide an autogenerated input UI for your Actor's users.**

The Actor input schema serves three main purposes:

* It ensures the input data supplied to the Actor adhere to specified requirements and validation rules.
* It is used by the Apify platform to generate a user-friendly interface for configuring and running your Actor.
* It simplifies invoking your Actors from external systems by generating calling code and connectors for integrations.

To define an input schema for an Actor, set input field in the .actor/actor.json file to an input schema object (described below), or path to a JSON file containing the input schema object. For backwards compatibility, if the input field is omitted, the system looks for an INPUT\_SCHEMA.json file either in the .actor directory or the Actor's top-level directory—but note that this functionality is deprecated and might be removed in the future. The maximum allowed size for the input schema file is 500 kB.

When you provide an input schema, the Apify platform will validate the input data passed to the Actor on start (via the API or Apify Console) to ensure compliance before starting the Actor. If the input object doesn't conform the schema, the caller receives an error and the Actor is not started.

**Validation aid**

You can use our [visual input schema editor](https://apify.github.io/input-schema-editor-react/) to guide you through the creation of the INPUT\_SCHEMA.json file.

To ensure the input schema is valid, here's a corresponding [JSON schema file](https://github.com/apify/apify-shared-js/blob/master/packages/input_schema/src/schema.json).

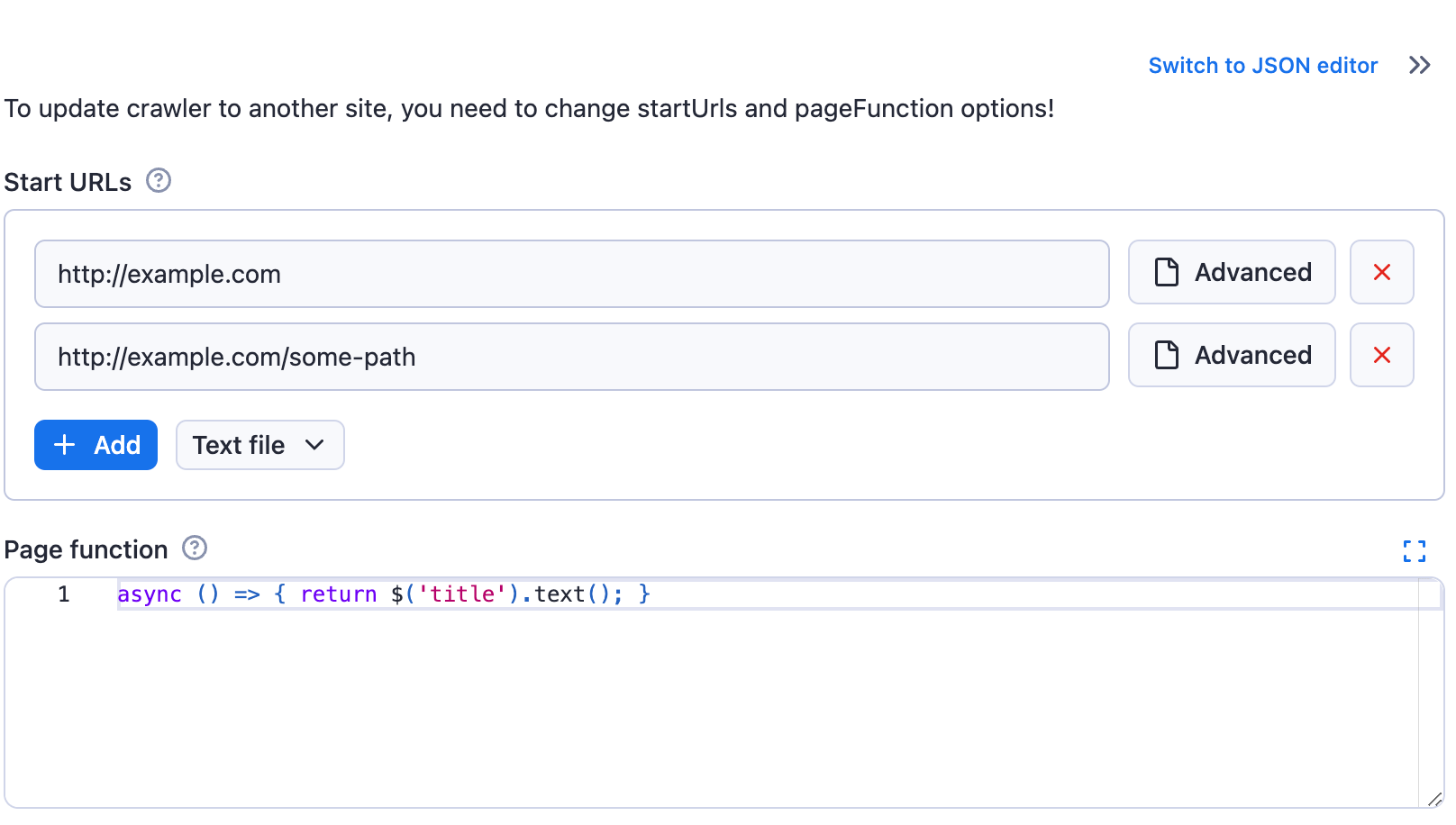
You can also use the [apify validate-schema](https://docs.apify.com/cli/docs/reference#apify-validate-schema-path) command in the Apify CLI.

**Example**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/specification/v1#example)

Imagine a simple web crawler that accepts an array of start URLs and a JavaScript function to execute on each visited page. The input schema for such a crawler could be defined as follows:

{  
 "title": "Cheerio Crawler input",  
 "description": "To update crawler to another site, you need to change startUrls and pageFunction options!",  
 "type": "object",  
 "schemaVersion": 1,  
 "properties": {  
 "startUrls": {  
 "title": "Start URLs",  
 "type": "array",  
 "description": "URLs to start with",  
 "prefill": [  
 { "url": "http://example.com" },  
 { "url": "http://example.com/some-path" }  
 ],  
 "editor": "requestListSources"  
 },  
 "pageFunction": {  
 "title": "Page function",  
 "type": "string",  
 "description": "Function executed for each request",  
 "prefill": "async () => { return $('title').text(); }",  
 "editor": "javascript"  
 }  
 },  
 "required": ["startUrls", "pageFunction"]  
}

The generated input UI will be:



If you switch the input to the **JSON** display using the toggle, then you will see the entered input stringified to JSON, as it will be passed to the Actor:

{  
 "startUrls": [  
 {  
 "url": "http://example.com"  
 },  
 {  
 "url": "http://example.com/some-path"  
 }  
 ],  
 "pageFunction": "async () => { return $('title').text(); }"  
}

**Structure**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/specification/v1#structure)

{  
 "title": "Cheerio Crawler input",  
 "type": "object",  
 "schemaVersion": 1,  
 "properties": { */\* define input fields here \*/* },  
 "required": []  
}

| **Property** | **Type** | **Required** | **Description** |
| --- | --- | --- | --- |
| title | String | Yes | Any text describing your input schema. |
| description | String | No | Help text for the input that will be displayed above the UI fields. |
| type | String | Yes | This is fixed and must be set to string object. |
| schemaVersion | Integer | Yes | The version of the input schema specification against which your schema is written. Currently, only version 1 is out. |
| properties | Object | Yes | This is an object mapping each field key to its specification. |
| required | String | No | An array of field keys that are required. |

**Input schema differences**

Even though the structure of the Actor input schema is similar to JSON schema, there are some differences. We cannot guarantee that JSON schema tooling will work on input schema documents. For a more precise technical understanding of the matter, feel free to browse the code of the [@apify/input\_schema](https://github.com/apify/apify-shared-js/tree/master/packages/input_schema/src) package.

**Fields**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/specification/v1#fields)

Each field of your input is described under its key in the inputSchema.properties object. The field might have integer, string, array, object, or boolean type, and its specification contains the following properties:

| **Property** | **Value** | **Required** | **Description** |
| --- | --- | --- | --- |
| type | One of   * string * array * object * boolean * integer | Yes | Allowed type for the input value. Cannot be mixed. |
| title | String | Yes | Title of the field in UI. |
| description | String | Yes | Description of the field that will be displayed as help text in Actor input UI. |
| default | Must match type property. | No | Default value that will be used when no value is provided. |
| prefill | Must match type property. | No | Value that will be prefilled in the Actor input interface. |
| example | Must match type property. | No | Sample value of this field for the Actor to be displayed when Actor is published in Apify Store. |
| sectionCaption | String | No | If this property is set, then all fields following this field (this field included) will be separated into a collapsible section with the value set as its caption. The section ends at the last field or the next field which has the sectionCaption property set. |
| sectionDescription | String | No | If the sectionCaption property is set, then you can use this property to provide additional description to the section. The description will be visible right under the caption when the section is open. |

**Prefill vs. default vs. required**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/specification/v1#prefill-vs-default-vs-required)

Here is a rule of thumb for whether an input field should have a prefill, default, or be required:

* **Prefill** - Use for fields that don't have a reasonable default. The provided value is prefilled for the user to show them an example of using the field and to make it easy to test the Actor (e.g., search keyword, start URLs). In other words, this field is only used in the user interface but does not affect the Actor functionality and API.
* **Required** - Use for fields that don't have a reasonable default and MUST be entered by the user (e.g., API token, password).
* **Default** - Use for fields that MUST be set for the Actor run to some value, but where you don't need the user to change the default behavior (e.g., max pages to crawl, proxy settings). If the user omits the value when starting the Actor via any means (API, CLI, scheduler, or user interface), the platform automatically passes the Actor this default value.
* **No particular setting** - Use for purely optional fields where it makes no sense to prefill any value (e.g., flags like debug mode or download files).

In summary, you can use each option independently or use a combination of **Prefill + Required**, but the combinations **Prefill + Default** or **Default + Required** don't make sense to use.

**Additional properties**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/specification/v1#additional-properties)

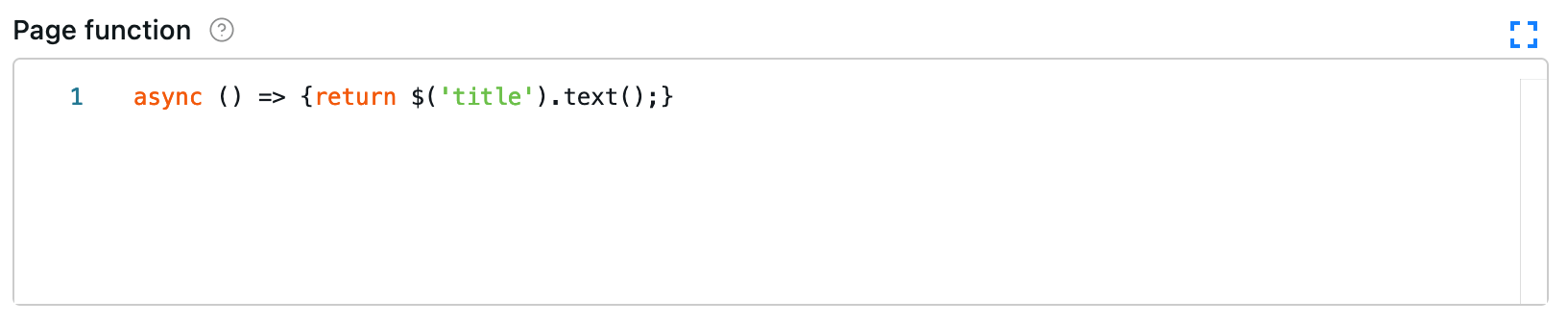
Most types also support additional properties defining, for example, the UI input editor.

**String**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/specification/v1#string)

Example of a code input:

{  
 "title": "Page function",  
 "type": "string",  
 "description": "Function executed for each request",  
 "editor": "javascript",  
 "prefill": "async () => { return $('title').text(); }"  
}

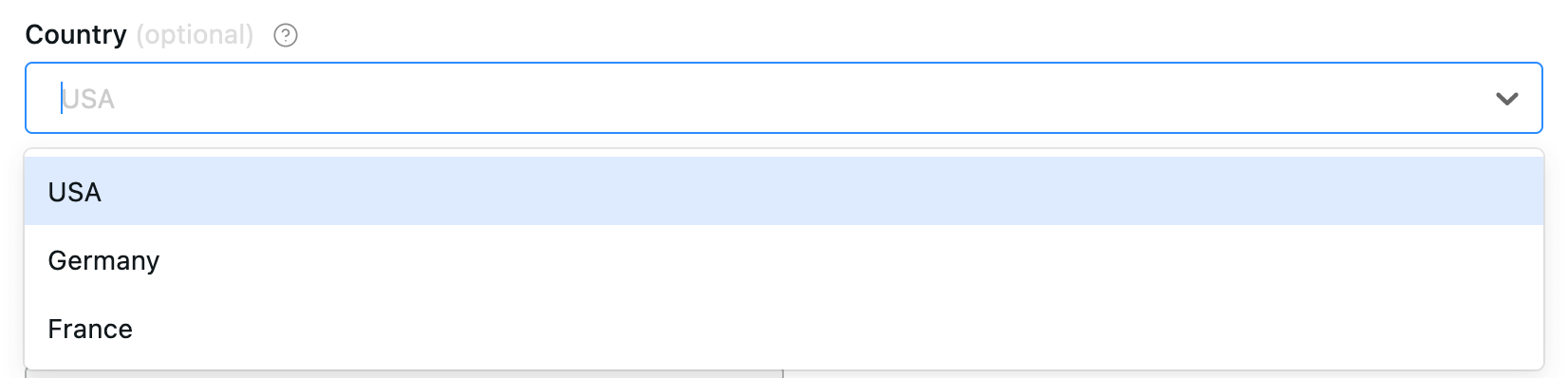
Rendered input:



Example of country selection using a select input:

{  
 "title": "Country",  
 "type": "string",  
 "description": "Select your country",  
 "editor": "select",  
 "default": "us",  
 "enum": ["us", "de", "fr"],  
 "enumTitles": ["USA", "Germany", "France"]  
}

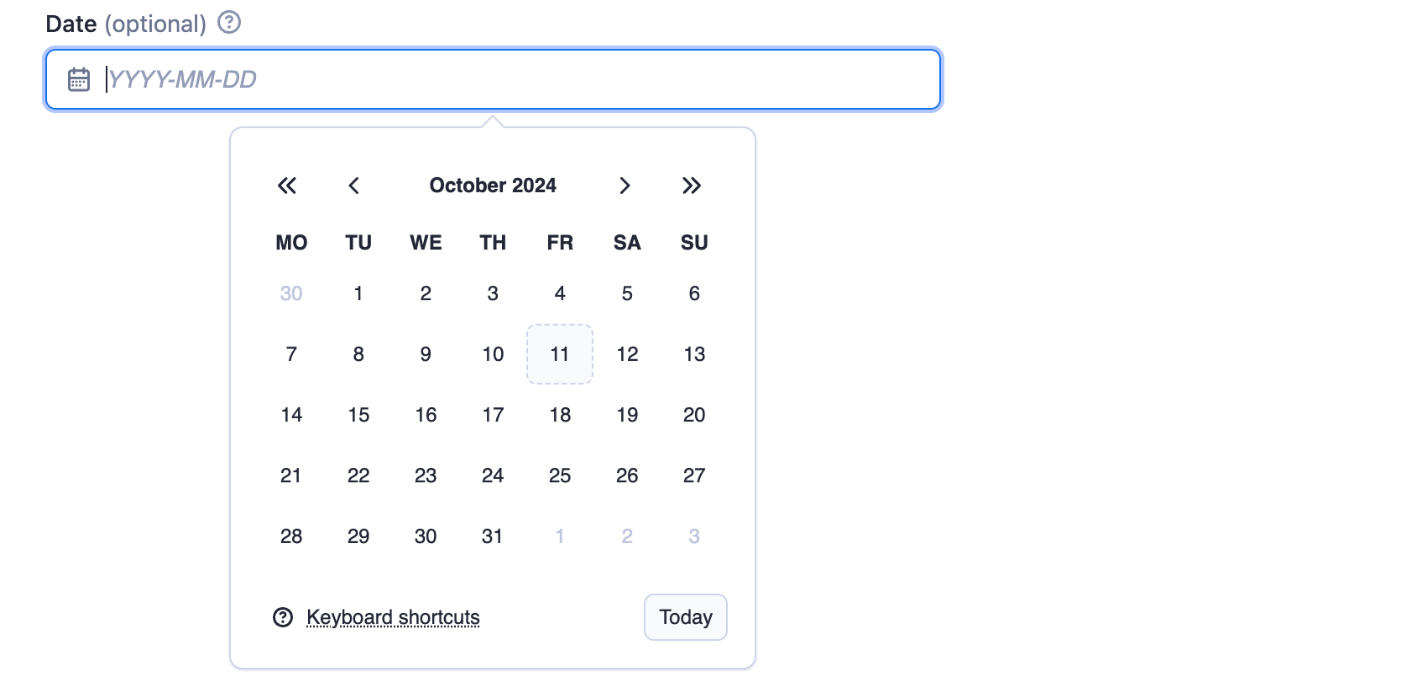
Rendered input:



Example of date selection using absolute and relative datepicker editor:

{  
 "absoluteDate": {  
 "title": "Date",  
 "type": "string",  
 "description": "Select absolute date in format YYYY-MM-DD",  
 "editor": "datepicker",  
 "pattern": "^(\\d{4})-(0[1-9]|1[0-2])-(0[1-9]|[12]\\d|3[01])$"  
 },  
 "relativeDate": {  
 "title": "Relative date",  
 "type": "string",  
 "description": "Select relative date in format: {number} {unit}",  
 "editor": "datepicker",  
 "dateType": "relative",  
 "pattern": "^(\\d+)\\s\*(day|week|month|year)s?$"  
 },  
 "anyDate": {  
 "title": "Any date",  
 "type": "string",  
 "description": "Select date in format YYYY-MM-DD or {number} {unit}",  
 "editor": "datepicker",  
 "dateType": "absoluteOrRelative",  
 "pattern": "^(\\d{4})-(0[1-9]|1[0-2])-(0[1-9]|[12]\\d|3[01])$|^(\\d+)\\s\*(day|week|month|year)s?$"  
 }  
}

The absoluteDate property renders a date picker that allows selection of a specific date and returns string value in YYYY-MM-DD format. Validation is ensured thanks to pattern field. In this case the dateType property is omitted, as it defaults to "absolute".



The relativeDate property renders an input field that enables the user to choose the relative date and returns the value in {number} {unit} format, for example "2 days". The dateType parameter is set to "relative" to restrict input to relative dates only.



The anyDate property renders a date picker that accepts both absolute and relative dates. The Actor author is responsible for parsing and interpreting the selected date format.



Properties:

| **Property** | **Value** | **Required** | **Description** |
| --- | --- | --- | --- |
| editor | One of   * textfield * textarea * javascript * python * select * datepicker * hidden | Yes | Visual editor used for the input field. |
| pattern | String | No | Regular expression that will be used to validate the input. If validation fails, the Actor will not run. |
| minLength | Integer | No | Minimum length of the string. |
| maxLength | Integer | No | Maximum length of the string. |
| enum | [String] | Required if editor is select | Using this field, you can limit values to the given array of strings. Input will be displayed as select box. |
| enumTitles | [String] | No | Titles for the enum keys described. |
| nullable | Boolean | No | Specifies whether null is an allowed value. |
| isSecret | Boolean | No | Specifies whether the input field will be stored encrypted. Only available with textfield and textarea editors. |
| dateType | One of   * absolute * relative * absoluteOrRelative | No | This property, which is only available with datepicker editor, specifies what date format should visual editor accept (The JSON editor accepts any string without validation.).   * absolute value enables date input in YYYY-MM-DD format. To parse returned string regex like this can be used: ^(\d{4})-(0[1-9]|1[0-2])-(0[1-9]|[12]\d|3[01])$.      * relative value enables relative date input in {number} {unit} format. Supported units are: days, weeks, months, years.  The input is passed to the Actor as plain text (e.g., "3 weeks"). To parse it, regex like this can be used: ^(\d+)\s\*(day|week|month|year)s?$.      * absoluteOrRelative value enables both absolute and relative formats and user can switch between them. It's up to Actor author to parse a determine actual used format - regexes above can be used to check whether the returned string match one of them.   Defaults to absolute. |

**Regex escape**

When using escape characters \ for the regular expression in the pattern field, be sure to escape them to avoid invalid JSON issues. For example, the regular expression https:\/\/(www\.)?apify\.com\/.+ would become https:\\/\\/(www\\.)?apify\\.com\\/.+.

**Advanced date and time handling**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/specification/v1#advanced-date-and-time-handling)

While the datepicker editor doesn't support setting time values visually, you can allow users to handle more complex datetime formats and pass them via JSON. The following regex allows users to optionally extend the date with full ISO datetime format or pass hours and minutes as a relative date:

"pattern": "^(\\d{4})-(0[1-9]|1[0-2])-(0[1-9]|[12]\\d|3[01])(T[0-2]\\d:[0-5]\\d(:[0-5]\\d)?(\\.\\d+)?Z?)?$|^(\\d+)\\s\*(minute|hour|day|week|month|year)s?$"

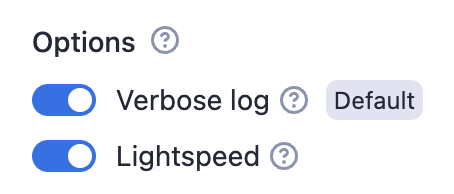
When implementing time-based fields, make sure to explain to your users through the description that the time values should be provided in UTC. This helps prevent timezone-related issues.

**Boolean**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/specification/v1#boolean)

Example options with group caption:

{  
 "verboseLog": {  
 "title": "Verbose log",  
 "type": "boolean",  
 "description": "Debug messages will be included in the log.",  
 "default": true,  
 "groupCaption": "Options",  
 "groupDescription": "Various options for this Actor"  
 },  
 "lightspeed": {  
 "title": "Lightspeed",  
 "type": "boolean",  
 "description": "If checked then actors runs at the  
 speed of light.",  
 "prefill": true  
 }  
}

Rendered input:



Properties:

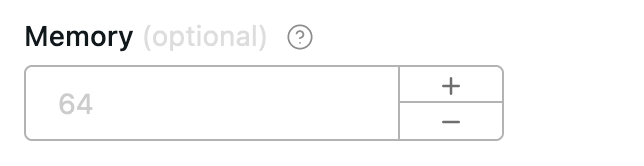
| **Property** | **Value** | **Required** | **Description** |
| --- | --- | --- | --- |
| editor | One of   * checkbox * hidden | No | Visual editor used for the input field. |
| groupCaption | String | No | If you want to group multiple checkboxes together, add this option to the first of the group. |
| groupDescription | String | No | Description displayed as help text displayed of group title. |
| nullable | Boolean | No | Specifies whether null is an allowed value. |

**Integer**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/specification/v1#integer)

Example:

{  
 "title": "Memory",  
 "type": "integer",  
 "description": "Select memory in megabytes",  
 "default": 64,  
 "maximum": 1024,  
 "unit": "MB"  
}

Rendered input:



Properties:

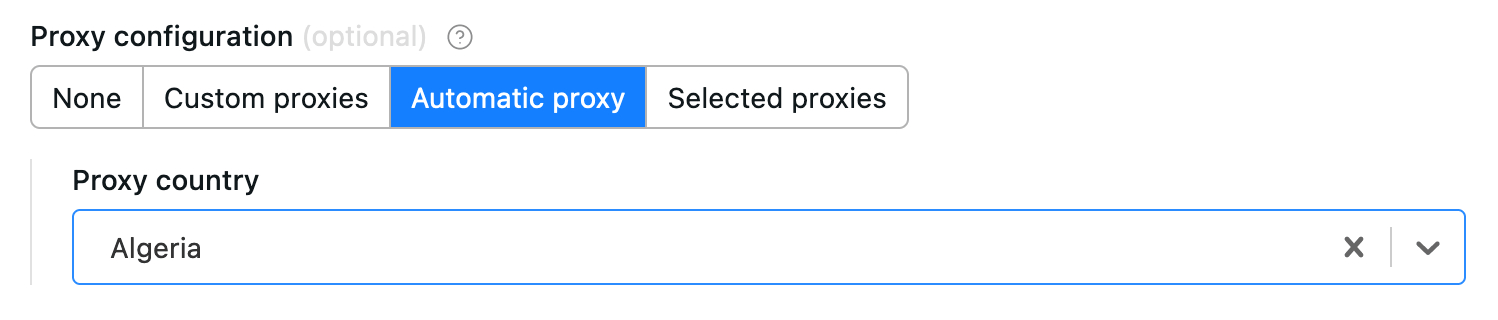
| **Property** | **Value** | **Required** | **Description** |
| --- | --- | --- | --- |
| editor | One of:   * number * hidden | No | Visual editor used for input field. |
| maximum | Integer | No | Maximum allowed value. |
| minimum | Integer | No | Minimum allowed value. |
| unit | String | No | Unit displayed next to the field in UI, for example *second*, *MB*, etc. |
| nullable | Boolean | No | Specifies whether null is an allowed value. |

**Object**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/specification/v1#object)

Example of proxy configuration:

{  
 "title": "Proxy configuration",  
 "type": "object",  
 "description": "Select proxies to be used by your crawler.",  
 "prefill": { "useApifyProxy": true },  
 "editor": "proxy"  
}

Rendered input:



The object where the proxy configuration is stored has the following structure:

{  
 *// Indicates whether Apify Proxy was selected.*  
 "useApifyProxy": Boolean,  
  
 *// Array of Apify Proxy groups. Is missing or null if*  
 *// Apify Proxy's automatic mode was selected*  
 *// or if proxies are not used.*  
 "apifyProxyGroups": String[],  
  
 *// Array of custom proxy URLs.*  
 *// Is missing or null if custom proxies were not used.*  
 "proxyUrls": String[],  
}

Example of a black box object:

{  
 "title": "User object",  
 "type": "object",  
 "description": "Enter object representing user",  
 "prefill": {  
 "name": "John Doe",  
 "email": "janedoe@gmail.com"  
 },  
 "editor": "json"  
}

Rendered input:



Properties:

| **Property** | **Value** | **Required** | **Description** |
| --- | --- | --- | --- |
| editor | One of   * json * proxy * hidden | Yes | UI editor used for input. |
| patternKey | String | No | Regular expression that will be used to validate the keys of the object. |
| patternValue | String | No | Regular expression that will be used to validate the values of object. |
| maxProperties | Integer | No | Maximum number of properties the object can have. |
| minProperties | Integer | No | Minimum number of properties the object can have. |
| nullable | Boolean | No | Specifies whether null is an allowed value. |

**Array**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/specification/v1#array)

Example of request list sources configuration:

{  
 "title": "Start URLs",  
 "type": "array",  
 "description": "URLs to start with",  
 "prefill": [{ "url": "https://apify.com" }],  
 "editor": "requestListSources"  
}

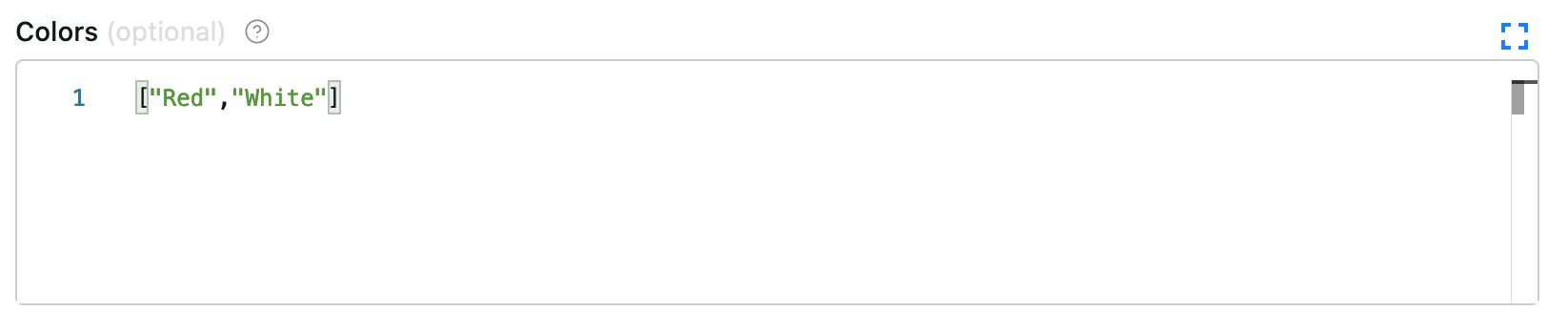
Rendered input:



Example of an array:

{  
 "title": "Colors",  
 "type": "array",  
 "description": "Enter colors you know",  
 "prefill": ["Red", "White"],  
 "editor": "json"  
}

Rendered input:



Properties:

| **Property** | **Value** | **Required** | **Description** |
| --- | --- | --- | --- |
| editor | One of   * json * requestListSources * pseudoUrls * globs * keyValue * stringList * select * hidden | Yes | UI editor used for input. |
| placeholderKey | String | No | Placeholder displayed for key field when no value is specified. Works only with keyValue editor. |
| placeholderValue | String | No | Placeholder displayed in value field when no value is provided. Works only with keyValue and stringList editors. |
| patternKey | String | No | Regular expression that will be used to validate the keys of items in the array. Works only with keyValue editor. |
| patternValue | String | No | Regular expression that will be used to validate the values of items in the array. Works only with keyValue and stringList editors. |
| maxItems | Integer | No | Maximum number of items the array can contain. |
| minItems | Integer | No | Minimum number of items the array can contain. |
| uniqueItems | Boolean | No | Specifies whether the array should contain only unique values. |
| nullable | Boolean | No | Specifies whether null is an allowed value. |
| items | object | No | Specifies format of the items of the array, useful mainly for multiselect (see below) |

Usage of this field is based on the selected editor:

* requestListSources - value from this field can be used as input for the [RequestList](https://crawlee.dev/api/core/class/RequestList) class from Crawlee.
* pseudoUrls - is intended to be used with a combination of the [PseudoUrl](https://crawlee.dev/api/core/class/PseudoUrl) class and the [enqueueLinks()](https://crawlee.dev/api/core/function/enqueueLinks) function from Crawlee.

Editor type requestListSources supports input in formats defined by the [sources](https://crawlee.dev/api/core/interface/RequestListOptions#sources) property of [RequestListOptions](https://crawlee.dev/api/core/interface/RequestListOptions).

Editor type globs maps to the Crawlee's [GlobInput](https://crawlee.dev/api/core#GlobInput) used by the [UrlPatterObject](https://crawlee.dev/api/core#UrlPatternObject).

Editor type select allows the user to pick items from a select, providing multiple choices. Please check this example of how to define the multiselect field:

{  
 "title": "Multiselect field",  
 "description": "My multiselect field",  
 "type": "array",  
 "editor": "select",  
 "items": {  
 "type": "string",  
 "enum": ["value1", "value2", "value3"],  
 "enumTitles": ["Label of value1", "Label of value2", "Label of value3"]  
 }  
}

To correctly define options for multiselect, you need to define the items property and then provide values and (optionally) labels in enum and enumTitles properties.

**Resource type**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/specification/v1#resource-type)

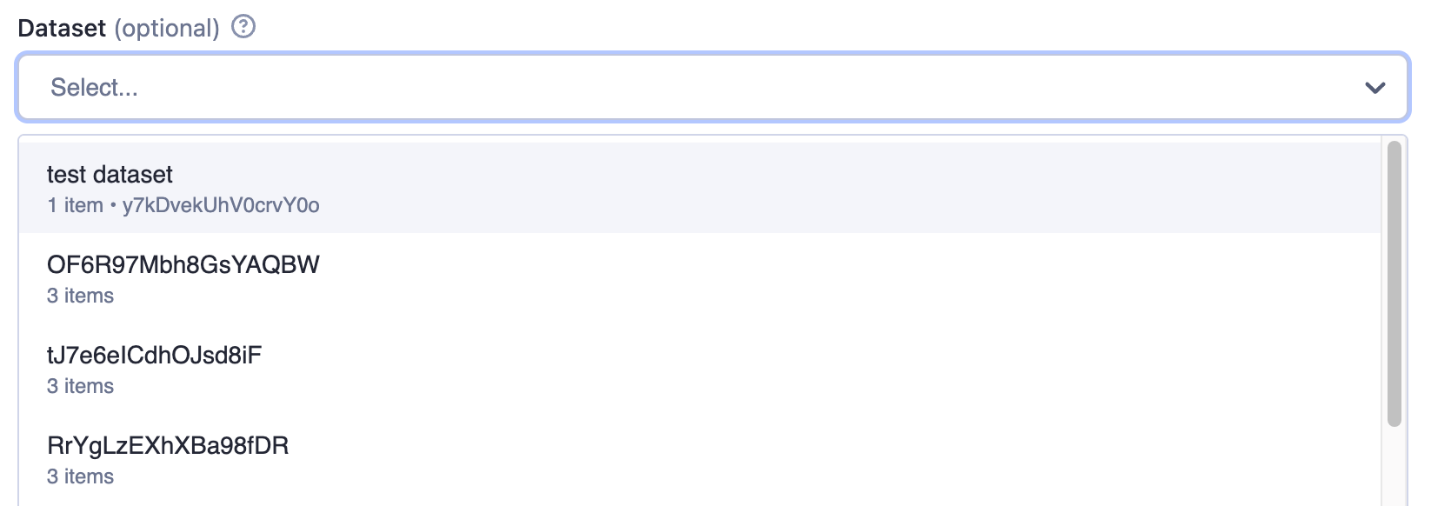
Resource type identifies what kind of Apify Platform object is referred to in the input field. For example, the Key-value store resource type can be referred to using a string ID. Currently, it supports storage resources only, allowing the reference of a Dataset, Key-Value Store or Request Queue.

For Actor developers, the resource input value is a string representing the storage ID. The type of the property is either string or array. In case of array (for multiple resources) the return value is an array of IDs. In the user interface, a picker is provided for easy selection, where users can search and choose from their own storages or those they have access to.

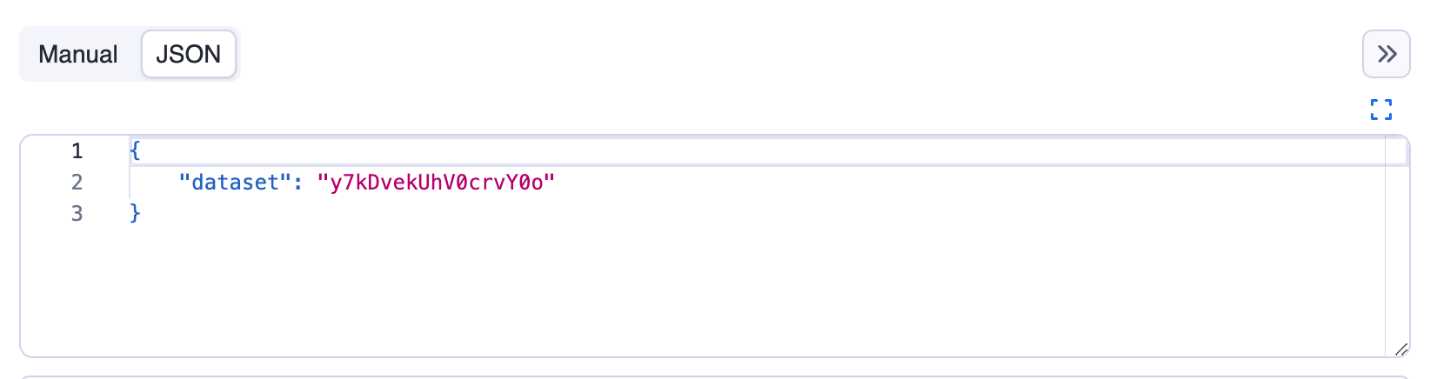
Example of a Dataset input:

{  
 "title": "Dataset",  
 "type": "string",  
 "description": "Select a dataset",  
 "resourceType": "dataset"  
}

Rendered input:



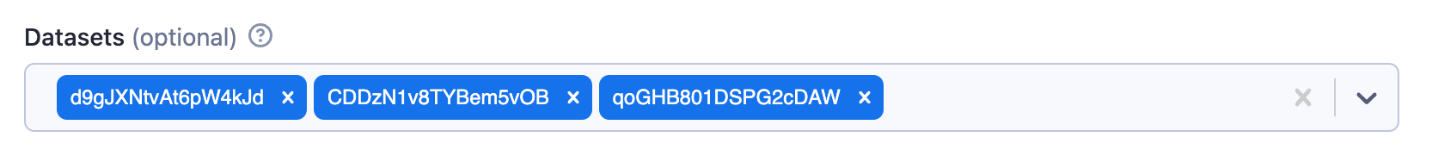
The returned value is resource reference, in this example it's the dataset ID as can be seen in the JSON tab:



Example of multiple datasets input:

{  
 "title": "Datasets",  
 "type": "array",  
 "description": "Select multiple datasets",  
 "resourceType": "dataset"  
}

Rendered input:



Properties:

| **Property** | **Value** | **Required** | **Description** |
| --- | --- | --- | --- |
| type | One of   * string * array | Yes | Specifies the type of input - string for single value or array for multiple values |
| editor | One of   * resourcePicker * hidden | No | Visual editor used for the input field. Defaults to resourcePicker. |
| resourceType | One of   * dataset * keyValueStore * requestQueue | Yes | Type of Apify Platform resource |
| minItems | Integer | No | Minimum number of items the array can contain. Only for type: array |
| maxItems | Integer | No | Maximum number of items the array can contain. Only for type: array |

**Secret input**

**Learn about making some Actor input fields secret and encrypted. Ideal for passing passwords, API tokens, or login cookies to Actors.**

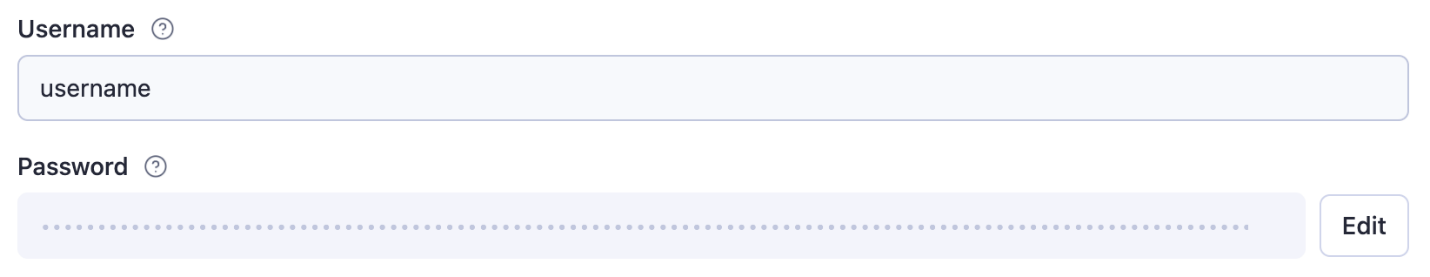
The secret input feature lets you mark specific input fields of an Actor as sensitive. When you save the Actor's input configuration, the values of these marked fields get encrypted. The encrypted input data can only be decrypted within the Actor. This provides an extra layer of security for sensitive information like API keys, passwords, or other confidential data.

**How to set a secret input field**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/secret-input#how-to-set-a-secret-input-field)

To make an input field secret, you need to add a "isSecret": true setting to the input field in the Actor's [input schema](https://docs.apify.com/platform/actors/development/actor-definition/input-schema), like this:

{  
 *// ...*  
 "properties": {  
 *// ...*  
 "password": {  
 "title": "Password",  
 "type": "string",  
 "description": "A secret, encrypted input field",  
 "editor": "textfield",  
 "isSecret": true  
 },  
 *// ...*  
 },  
 *// ...*  
}

The editor for this input field will then turn into a secret input, and when you edit the field value, it will be stored encrypted.



**Type restriction**

This is only available for string inputs, and the editor type is limited to textfield or textarea.

**Read secret input fields**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/secret-input#read-secret-input-fields)

When you read the Actor input through Actor.getInput(), the encrypted fields are automatically decrypted (starting with the [apify package](https://www.npmjs.com/package/apify) version 3.1.0).

> await Actor.getInput();  
{  
 username: 'username',  
 password: 'password'  
}

If you read the INPUT key from the Actor run's default key-value store directly, you will still get the original, encrypted input value.

> await Actor.getValue('INPUT');  
{  
 username: 'username',  
 password: 'ENCRYPTED\_VALUE:Hw/uqRMRNHmxXYYDJCyaQX6xcwUnVYQnH4fWIlKZL2Vhtq1rZmtoGXQSnhIXmF58+DjKlMZpTlK2zN3YUXk1ylzU6LfXyysOG/PISAfwm27FUgy3IfdgMyQggQ4MydLzdlzefX0mPRyixBviRcFhRTC+K7nK9lkATt3wJpj91YAZm104ZYkcd5KmsU2JX39vxN0A0lX53NjIenzs3wYPaPYLdjKIe+nqG9fHlL7kALyi7Htpy91ZgnQJ1s9saJRkKfWXvmLYIo5db69zU9dGCeJzUc0ca154O+KYYP7QTebJxqZNQsC8EH6sVMQU3W0qYKjuN8fUm1fRzyw/kKFacQ==:VfQd2ZbUt3S0RZ2ciywEWYVBbTTZOTiy'  
}

**Encryption mechanism**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/secret-input#encryption-mechanism)

The encryption mechanism used for encrypting the secret input fields is the same dual encryption as in [PGP](https://en.wikipedia.org/wiki/Pretty_Good_Privacy#/media/File:PGP_diagram.svg). The secret input field is encrypted using a random key, using the aes-256-gcm cipher, and then the key is encrypted using a 2048-bit RSA key.

The RSA key is unique for each combination of user and Actor, ensuring that no Actor can decrypt input intended for runs of another Actor by the same user, and no user can decrypt input runs of the same Actor by a different user. This isolation of decryption keys enhances the security of sensitive input data.

During Actor execution, the decryption keys are passed as environment variables, restricting the decryption of secret input fields to occur solely within the context of the Actor run. This approach prevents unauthorized access to sensitive input data outside the Actor's execution environment.

**Dataset Schema Specification**

**Learn how to define and present your dataset schema in an user-friendly output UI.**

The dataset schema defines the structure and representation of data produced by an Actor, both in the API and the visual user interface.

**Example**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema#example)

Let's consider an example Actor that calls Actor.pushData() to store data into dataset:

main.js

import { Actor } from 'apify';  
*// Initialize the JavaScript SDK*  
await Actor.init();  
  
*/\*\**  
 *\* Actor code*  
 *\*/*  
await Actor.pushData({  
 numericField: 10,  
 pictureUrl: 'https://www.google.com/images/branding/googlelogo/2x/googlelogo\_color\_92x30dp.png',  
 linkUrl: 'https://google.com',  
 textField: 'Google',  
 booleanField: true,  
 dateField: new Date(),  
 arrayField: ['#hello', '#world'],  
 objectField: {},  
});  
  
  
*// Exit successfully*  
await Actor.exit();

To set up the Actor's output tab UI using a single configuration file, use the following template for the .actor/actor.json configuration:

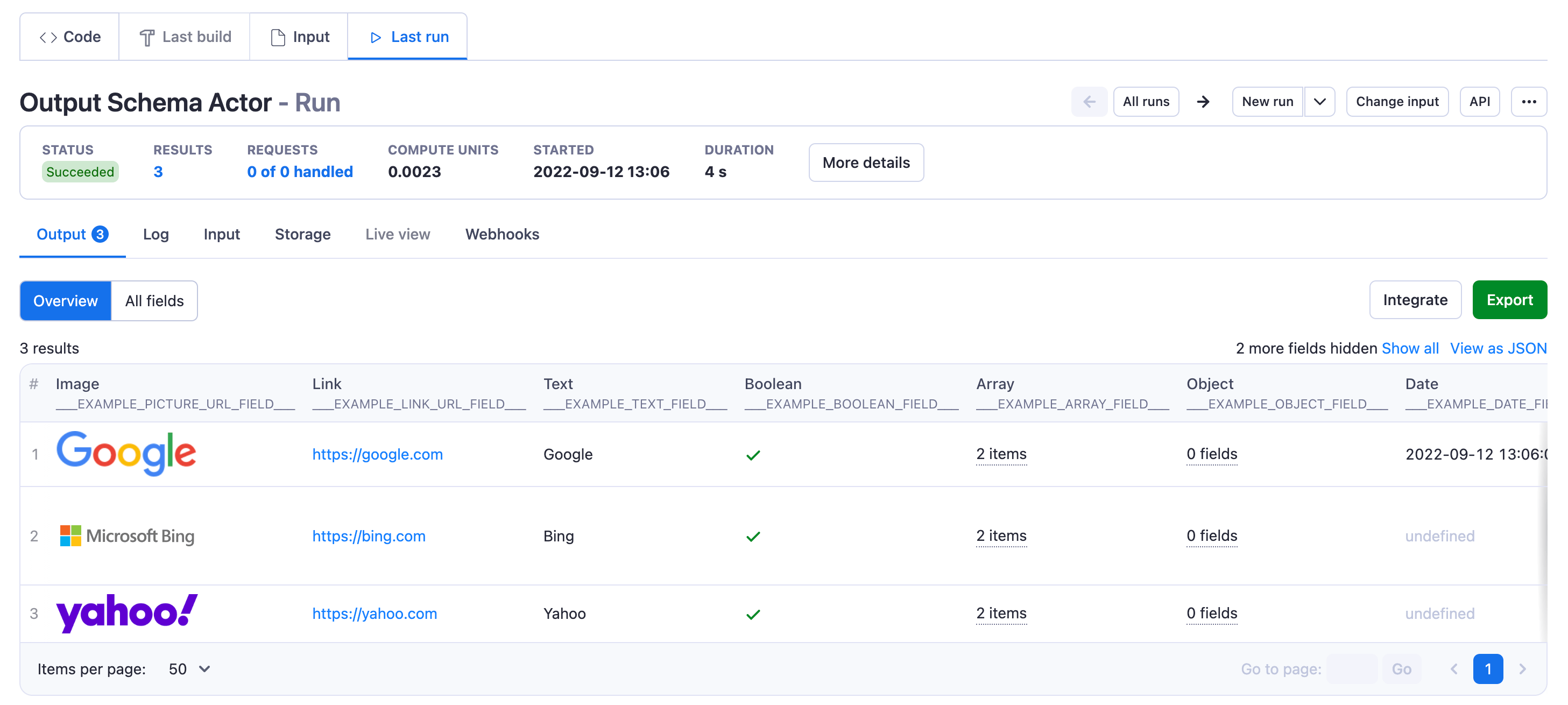
.actor/actor.json

{  
 "actorSpecification": 1,  
 "name": "Actor Name",  
 "title": "Actor Title",  
 "version": "1.0.0",  
 "storages": {  
 "dataset": {  
 "actorSpecification": 1,  
 "views": {  
 "overview": {  
 "title": "Overview",  
 "transformation": {  
 "fields": [  
 "pictureUrl",  
 "linkUrl",  
 "textField",  
 "booleanField",  
 "arrayField",  
 "objectField",  
 "dateField",  
 "numericField"  
 ]  
 },  
 "display": {  
 "component": "table",  
 "properties": {  
 "pictureUrl": {  
 "label": "Image",  
 "format": "image"  
 },  
 "linkUrl": {  
 "label": "Link",  
 "format": "link"  
 },  
 "textField": {  
 "label": "Text",  
 "format": "text"  
 },  
 "booleanField": {  
 "label": "Boolean",  
 "format": "boolean"  
 },  
 "arrayField": {  
 "label": "Array",  
 "format": "array"  
 },  
 "objectField": {  
 "label": "Object",  
 "format": "object"  
 },  
 "dateField": {  
 "label": "Date",  
 "format": "date"  
 },  
 "numericField": {  
 "label": "Number",  
 "format": "number"  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
}

The template above defines the configuration for the default dataset output view. Under the views property, there is one view titled *Overview*. The view configuration consists of two main steps:

1. transformation - set up how to fetch the data.
2. display - set up how to visually present the fetched data.

The default behavior of the Output tab UI table is to display all fields from transformation.fields in the specified order. You can customize the display properties for specific formats or column labels if needed.



**Structure**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema#structure)

Output configuration files need to be located in the .actor folder within the Actor's root directory.

You have two choices of how to organize files within the .actor folder.

**Single configuration file**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema#single-configuration-file)

.actor/actor.json

{  
 "actorSpecification": 1,  
 "name": "this-is-book-library-scraper",  
 "title": "Book Library scraper",  
 "version": "1.0.0",  
 "storages": {  
 "dataset": {  
 "actorSpecification": 1,  
 "fields": {},  
 "views": {  
 "overview": {  
 "title": "Overview",  
 "transformation": {},  
 "display": {}  
 }  
 }  
 }  
 }  
}

**Separate configuration files**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema#separate-configuration-files)

.actor/actor.json

{  
 "actorSpecification": 1,  
 "name": "this-is-book-library-scraper",  
 "title": "Book Library scraper",  
 "version": "1.0.0",  
 "storages": {  
 "dataset": "./dataset\_schema.json"  
 }  
}

.actor/dataset\_schema.json

{  
 "actorSpecification": 1,  
 "fields": {},  
 "views": {  
 "overview": {  
 "title": "Overview",  
 "transformation": {},  
 "display": {}  
 }  
 }  
}

Both of these methods are valid so choose one that suits your needs best.

**Handle nested structures**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema#handle-nested-structures)

The most frequently used data formats present the data in a tabular format (Output tab table, Excel, CSV). If your Actor produces nested JSON structures, you need to transform the nested data into a flat tabular format. You can flatten the data in the following ways:

* Use transformation.flatten to flatten the nested structure of specified fields. This transforms the nested object into a flat structure. e.g. with flatten:["foo"], the object {"foo": {"bar": "hello"}} is turned into {"foo.bar": "hello"}. Once the structure is flattened, it's necessary to use the flattened property name in both transformation.fields and display.properties, otherwise, fields might not be fetched or configured properly in the UI visualization.
* Use transformation.unwind to deconstruct the nested children into parent objects.
* Change the output structure in an Actor from nested to flat before the results are saved in the dataset.

**Dataset schema structure definitions**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema#dataset-schema-structure-definitions)

The dataset schema structure defines the various components and properties that govern the organization and representation of the output data produced by an Actor. It specifies the structure of the data, the transformations to be applied, and the visual display configurations for the Output tab UI.

**DatasetSchema object definition**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema#datasetschema-object-definition)

| **Property** | **Type** | **Required** | **Description** |
| --- | --- | --- | --- |
| actorSpecification | integer | true | Specifies the version of dataset schema structure document. Currently only version 1 is available. |
| fields | JSONSchema compatible object | true | Schema of one dataset object. Use JsonSchema Draft 2020–12 or other compatible formats. |
| views | DatasetView object | true | An object with a description of an API and UI views. |

**DatasetView object definition**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema#datasetview-object-definition)

| **Property** | **Type** | **Required** | **Description** |
| --- | --- | --- | --- |
| title | string | true | The title is visible in UI in the Output tab and in the API. |
| description | string | false | The description is only available in the API response. |
| transformation | ViewTransformation object | true | The definition of data transformation applied when dataset data is loaded from Dataset API. |
| display | ViewDisplay object | true | The definition of Output tab UI visualization. |

**ViewTransformation object definition**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema#viewtransformation-object-definition)

| **Property** | **Type** | **Required** | **Description** |
| --- | --- | --- | --- |
| fields | string[] | true | Selects fields to be presented in the output. The order of fields matches the order of columns in visualization UI. If a field value is missing, it will be presented as **undefined** in the UI. |
| unwind | string | false | Deconstructs nested children into parent object, For example, with unwind:["foo"], the object {"foo": {"bar": "hello"}} is transformed into {"bar": "hello"}. |
| flatten | string[] | false | Transforms nested object into flat structure. For example, with flatten:["foo"] the object {"foo":{"bar": "hello"}} is transformed into {"foo.bar": "hello"}. |
| omit | string | false | Removes the specified fields from the output. Nested fields names can be used as well. |
| limit | integer | false | The maximum number of results returned. Default is all results. |
| desc | boolean | false | By default, results are sorted in ascending based on the write event into the dataset. If desc:true, the newest writes to the dataset will be returned first. |

**ViewDisplay object definition**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema#viewdisplay-object-definition)

| **Property** | **Type** | **Required** | **Description** |
| --- | --- | --- | --- |
| component | string | true | Only the table component is available. |
| properties | Object | false | An object with keys matching the transformation.fields and ViewDisplayProperty as values. If properties are not set, the table will be rendered automatically with fields formatted as strings, arrays or objects. |

**ViewDisplayProperty object definition**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema#viewdisplayproperty-object-definition)

| **Property** | **Type** | **Required** | **Description** |
| --- | --- | --- | --- |
| label | string | false | In the Table view, the label will be visible as the table column's header. |
| format | One of   * text * number * date * link * boolean * image * array * object | false | Describes how output data values are formatted to be rendered in the Output tab UI. |

**Dataset validation**

**Specify the dataset schema within the Actors so you can add monitoring and validation at the field level.**

To define a schema for a default dataset of an Actor run, you need to set fields property in the dataset schema.

**info**

The schema defines a single item in the dataset. Be careful not to define the schema as an array, it always needs to be a schema of an object.

Schema configuration is not available for named datasets or dataset views.

You can either do that directly through actor.json:

.actor.json

{  
 "actorSpecification": 1,  
 "storages": {  
 "dataset": {  
 "actorSpecification": 1,  
 "fields": {  
 "$schema": "http://json-schema.org/draft-07/schema#",  
 "type": "object",  
 "properties": {  
 "name": {  
 "type": "string"  
 }  
 },  
 "required": ["name"]  
 },  
 "views": {}  
 }  
 }  
}

Or in a separate file linked from the .actor.json:

.actor.json

{  
 "actorSpecification": 1,  
 "storages": {  
 "dataset": "./dataset\_schema.json"  
 }  
}

dataset\_schema.json

{  
 "actorSpecification": 1,  
 "fields": {  
 "$schema": "http://json-schema.org/draft-07/schema#",  
 "type": "object",  
 "properties": {  
 "name": {  
 "type": "string"  
 }  
 },  
 "required": ["name"]  
 },  
 "views": {}  
}

**important**

Dataset schema needs to be a valid JSON schema draft-07, so the $schema line is important and must be exactly this value or it must be omitted:

"$schema": "http://json-schema.org/draft-07/schema#"

**Dataset validation**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema/validation#dataset-validation)

When you define a schema of your default dataset, the schema is then always used when you insert data into the dataset to perform validation (we use [AJV](https://ajv.js.org/)).

If the validation succeeds, nothing changes from the current behavior, data is stored and an empty response with status code 201 is returned.

If the data you attempt to store in the dataset is *invalid* (meaning any of the items received by the API fails validation), *the entire request will be discarded*, The API will return a response with status code 400 and the following JSON response:

{  
 "error": {  
 "type": "schema-validation-error",  
 "message": "Schema validation failed",  
 "data": {  
 "invalidItems": [{  
 "itemPosition": "<array index in the received array of items>",  
 "validationErrors": "<Complete list of AJV validation error objects>"  
 }]  
 }  
 }  
}

The type of the AJV validation error object is [here](https://github.com/ajv-validator/ajv/blob/master/lib/types/index.ts#L86).

If you use the Apify JS client or Apify SDK and call pushData function you can access the validation errors in a try catch block like this:

* **Javascript**
* **Python**

try {  
 const response = await Actor.pushData(items);  
} catch (error) {  
 if (!error.data?.invalidItems) throw error;  
 error.data.invalidItems.forEach((item) => {  
 const { itemPosition, validationErrors } = item;  
 });  
}

**Examples of common types of validation**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema/validation#examples-of-common-types-of-validation)

Optional field (price is optional in this case):

{  
 "$schema": "http://json-schema.org/draft-07/schema#",  
 "type": "object",  
 "properties": {  
 "name": {  
 "type": "string"  
 },  
 "price": {  
 "type": "number"  
 }  
 },  
 "required": ["name"]  
}

Field with multiple types:

{  
 "price": {  
 "type": ["string", "number"]  
 }  
}

Field with type any:

{  
 "price": {  
 "type": ["string", "number", "object", "array", "boolean"]  
 }  
}

Enabling fields to be null :

{  
 "name": {  
 "type": "string",  
 "nullable": true  
 }  
}

Define type of objects in array:

{  
 "comments": {  
 "type": "array",  
 "items": {  
 "type": "object",  
 "properties": {  
 "author\_name": {  
 "type": "string"  
 }  
 }  
 }  
 }  
}

Define specific fields, but allow anything else to be added to the item:

{  
 "$schema": "http://json-schema.org/draft-07/schema#",  
 "type": "object",  
 "properties": {  
 "name": {  
 "type": "string"  
 }  
 },  
 "additionalProperties": true  
}

See [json schema reference](https://json-schema.org/understanding-json-schema/reference) for additional options.

You can also use [conversion tools](https://www.liquid-technologies.com/online-json-to-schema-converter) to convert an existing JSON document into it's JSON schema.

**Dataset field statistics**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema/validation#dataset-field-statistics)

When you configure the dataset fields schema, we generate a field list and measure the following statistics:

* **Null count:** how many items in the dataset have the field set to null
* **Empty count:** how many items in the dataset are undefined , meaning that for example empty string is not considered empty
* **Minimum and maximum**
  + For numbers, this is calculated directly
  + For strings, this field tracks string length
  + For arrays, this field tracks the number of items in the array
  + For objects, this tracks the number of keys
  + For booleans, this tracks whether the boolean was set to true. Minimum is always 0, but maximum can be either 1 or 0 based on whether at least one item in the dataset has the boolean field set to true.

You can use them in [monitoring](https://docs.apify.com/platform/monitoring#alert-configuration).

**Dockerfile**

**Learn about the available Docker images you can use as a base for your Apify Actors. Choose the right base image based on your Actor's requirements and the programming language you're using.**

When developing an [Actor](https://docs.apify.com/platform/actors) on the Apify platform, you can choose from a variety of pre-built Docker images to serve as the base for your Actor. These base images come with pre-installed dependencies and tools, making it easier to set up your development environment and ensuring consistent behavior across different environments.

**Base Docker images**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dockerfile#base-docker-images)

Apify provides several Docker images that can serve as base images for Actors. All images come in two versions:

* latest - This version represents the stable and production-ready release of the base image.
* beta - This version is intended for testing new features. Use at your own risk.

**Pre-cached Docker images**

All Apify Docker images are pre-cached on Apify servers to speed up Actor builds and runs. The source code for generating these images is available in the [apify-actor-docker](https://github.com/apify/apify-actor-docker) repository.

**Node.js base images**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dockerfile#nodejs-base-images)

These images come with Node.js (versions 16, 18, 20, or 22) the [Apify SDK for JavaScript](https://docs.apify.com/sdk/js), and [Crawlee](https://crawlee.dev/) preinstalled. The latest tag corresponds to the latest LTS version of Node.js.

| **Image** | **Description** |
| --- | --- |
| [actor-node](https://hub.docker.com/r/apify/actor-node/) | Slim Alpine Linux image with only essential tools. Does not include headless browsers. |
| [actor-node-puppeteer-chrome](https://hub.docker.com/r/apify/actor-node-puppeteer-chrome/) | Debian image with Chromium, Google Chrome, and the [puppeteer](https://github.com/puppeteer/puppeteer) library. |
| [actor-node-playwright-chrome](https://hub.docker.com/r/apify/actor-node-playwright-chrome/) | Debian image with Chromium, Google Chrome, and the [playwright](https://github.com/microsoft/playwright) library. |
| [actor-node-playwright-firefox](https://hub.docker.com/r/apify/actor-node-playwright-firefox/) | Debian image with Firefox and the [playwright](https://github.com/microsoft/playwright) library . |
| [actor-node-playwright-webkit](https://hub.docker.com/r/apify/actor-node-playwright-webkit/) | Ubuntu image with WebKit and the [playwright](https://github.com/microsoft/playwright) library. |
| [actor-node-playwright](https://hub.docker.com/r/apify/actor-node-playwright/) | Ubuntu image with [playwright](https://github.com/microsoft/playwright) and all its browsers (Chromium, Google Chrome, Firefox, WebKit). |

See the [Docker image guide](https://docs.apify.com/sdk/js/docs/guides/docker-images) for more details.

**Python base images**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dockerfile#python-base-images)

These images come with Python (version 3.8, 3.9, 3.10, 3.11, or 3.12) and the [Apify SDK for Python](https://docs.apify.com/sdk/python) preinstalled. The latest tag corresponds to the latest Python 3 version supported by the Apify SDK.

| **Image** | **Description** |
| --- | --- |
| [actor-python](https://hub.docker.com/r/apify/actor-python) | Slim Debian image with only the Apify SDK for Python. Does not include headless browsers. |
| [actor-python-playwright](https://hub.docker.com/r/apify/actor-python-playwright) | Debian image with [playwright](https://github.com/microsoft/playwright) and all its browsers. |
| [actor-python-selenium](https://hub.docker.com/r/apify/actor-python-selenium) | Debian image with [selenium](https://github.com/seleniumhq/selenium), Google Chrome, and [ChromeDriver](https://developer.chrome.com/docs/chromedriver/). |

**Custom Dockerfile**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/dockerfile#custom-dockerfile)

Apify uses Docker to build and run Actors. If you create an Actor from a template, it already contains an optimized Dockerfile for the given use case.

To use a custom Dockerfile, you can either:

* Reference it from the dockerfile field in .actor/actor.json,
* Store it in .actor/Dockerfile or Dockerfile in the root directory (searched in this order of preference).

If no Dockerfile is provided, the system uses the following default:

FROM apify/actor-node:20  
  
COPY package\*.json ./  
  
RUN npm --quiet set progress=false \  
 && npm install --only=prod --no-optional \  
 && echo "Installed NPM packages:" \  
 && (npm list --only=prod --no-optional --all || true) \  
 && echo "Node.js version:" \  
 && node --version \  
 && echo "NPM version:" \  
 && npm --version  
  
COPY . ./

For more information about Dockerfile syntax and commands, see the [Dockerfile reference](https://docs.docker.com/reference/dockerfile/).

**Custom base images**

While apify/actor-node is a base Docker image provided by Apify, you can use other Docker images as the base for your Actors.  
However, using the Apify images has some performance advantages, as they are pre-caches on Apify servers.

By default, Apify base Docker images with the Apify SDK and Crawlee start your Node.js application the same way as npm start, i.e, by running the command specified in package.json under scripts - start. The default package.json is similar to:

{  
 "description": "Anonymous Actor on the Apify platform",  
 "version": "0.0.1",  
 "license": "UNLICENSED",  
 "main": "main.js",  
 "scripts": {  
 "start": "node main.js"  
 },  
 "dependencies": {  
 "apify": "^3.0.0",  
 "crawlee": "^3.0.0"  
 },  
 "repository": {}  
}

This means the system expects the source code to be in main.js by default. If you want to override this behavior, use a custom package.json and/or Dockerfile.

The Apify Actor's source code placement is defined by its [Dockerfile](https://docs.apify.com/platform/actors/development/actor-definition/dockerfile). If you have created the Actor from one of Apify's [templates](https://apify.com/templates) then it's by convention placed in the /src directory.

You have the flexibility to choose any programming language, technologies, and dependencies (such as Chrome browser, Selenium, Cypress, or others) for your projects. The only requirement is to define a Dockerfile that builds the image for your Actor, including all dependencies and your source code.

**Example setup**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/source-code#example-setup)

Let's take a look at the example JavaScript Actor's source code. The following Dockerfile:

FROM apify/actor-node:20  
  
COPY package\*.json ./  
  
RUN npm --quiet set progress=false \  
 && npm install --omit=dev --omit=optional \  
 && echo "Installed NPM packages:" \  
 && (npm list --omit=dev --all || true) \  
 && echo "Node.js version:" \  
 && node --version \  
 && echo "NPM version:" \  
 && npm --version \  
 && rm -r ~/.npm  
  
COPY . ./  
  
CMD npm start --silent

This Dockerfile does the following tasks:

1. Builds the Actor from the apify/actor-node:20 base image.

FROM apify/actor-node:20

1. Copies the package.json and package-lock.json files to the image.

COPY package\*.json ./

1. Installs the npm packages specified in package.json, omitting development and optional dependencies.

RUN npm --quiet set progress=false \  
 && npm install --omit=dev --omit=optional \  
 && echo "Installed NPM packages:" \  
 && (npm list --omit=dev --all || true) \  
 && echo "Node.js version:" \  
 && node --version \  
 && echo "NPM version:" \  
 && npm --version \  
 && rm -r ~/.npm

1. Copies the rest of the source code to the image

COPY . ./

1. Runs the npm start command defined in package.json

CMD npm start --silent

**Optimized build cache**

By copying the package.json and package-lock.json files and installing dependencies before the rest of the source code, you can take advantage of Docker's caching mechanism. This approach ensures that dependencies are only reinstalled when the package.json or package-lock.json files change, significantly reducing build times. Since the installation of dependencies is often the most time-consuming part of the build process, this optimization can lead to substantial performance improvements, especially for larger projects with many dependencies.

**package.json**[**​**](https://docs.apify.com/platform/actors/development/actor-definition/source-code#packagejson)

The package.json file defines the npm start command:

{  
 "name": "getting-started-node",  
 "version": "0.0.1",  
 "type": "module",  
 "description": "This is an example of an Apify Actor.",  
 "dependencies": {  
 "apify": "^3.0.0"  
 },  
 "devDependencies": {},  
 "scripts": {  
 "start": "node src/main.js",  
 "test": "echo \"Error: oops, the Actor has no tests yet, sad!\" && exit 1"  
 },  
 "author": "It's not you; it's me",  
 "license": "ISC"  
}

**Basic commands**

**Learn how to use basic commands of the Apify SDK for both JavaScript and Python.**

This page covers essential commands for the Apify SDK in JavaScript & Python. These commands are designed to be used within a running Actor, either in a local environment or on the Apify platform.

**Initialize your Actor**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/basic-commands#initialize-your-actor)

Before using any Apify SDK methods, initialize your Actor. This step prepares the Actor to receive events from the Apify platform, sets up machine and storage configurations, and clears previous local storage states.

* **JavaScript**
* **Python**

Use the init() method to initialize your Actor. Pair it with exit() to properly terminate the Actor. For more information on exit(), go to [Exit Actor](https://docs.apify.com/platform/actors/development/programming-interface/basic-commands#exit-actor).

import { Actor } from 'apify';  
  
await Actor.init();  
console.log('Actor starting...');  
*// ...*  
await Actor.exit();

Alternatively, use the main() function for environments that don't support top-level awaits. The main() function is syntax-sugar for init() and exit(). It will call init() before it executes its callback and exit() after the callback resolves.

import { Actor } from 'apify';  
  
Actor.main(async () => {  
 console.log('Actor starting...');  
 *// ...*  
});

**Get input**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/basic-commands#get-input)

Access the Actor's input object, which is stored as a JSON file in the Actor's default key-value store. The input is an object with properties. If the Actor defines the input schema, the input object is guaranteed to conform to it.

* **JavaScript**
* **Python**

import { Actor } from 'apify';  
  
await Actor.init();  
  
const input = await Actor.getInput();  
console.log(input);  
*// prints: {'option1': 'aaa', 'option2': 456}*  
  
await Actor.exit();

Usually, the file is called INPUT, but the exact key is defined in the ACTOR\_INPUT\_KEY environment variable.

**Key-value store access**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/basic-commands#key-value-store-access)

Use the [Key-value store](https://docs.apify.com/platform/storage/key-value-store) to read and write arbitrary files

* **JavaScript**
* **Python**

import { Actor } from 'apify';  
  
await Actor.init();  
  
*// Save object to store (stringified to JSON)*  
await Actor.setValue('my\_state', { something: 123 });  
  
*// Save binary file to store with content type*  
await Actor.setValue('screenshot.png', buffer, { contentType: 'image/png' });  
  
*// Get a record from the store (automatically parsed from JSON)*  
const value = await Actor.getValue('my\_state');  
  
*// Access another key-value store by its name*  
const store = await Actor.openKeyValueStore('screenshots-store');  
await store.setValue('screenshot.png', buffer, { contentType: 'image/png' });  
  
await Actor.exit();

**Push results to the dataset**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/basic-commands#push-results-to-the-dataset)

Store larger results in a [Dataset](https://docs.apify.com/platform/storage/dataset), an append-only object storage

Note that Datasets can optionally be equipped with the schema that ensures only certain kinds of objects are stored in them.

* **JavaScript**
* **Python**

import { Actor } from 'apify';  
  
await Actor.init();  
  
*// Append result object to the default dataset associated with the run*  
await Actor.pushData({ someResult: 123 });  
  
await Actor.exit();

**Exit Actor**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/basic-commands#exit-actor)

When an Actor's main process terminates, the Actor run is considered finished. The process exit code determines Actor's final status:

* Exit code 0: Status SUCCEEDED
* Exit code not equal to 0: Status FAILED

By default, the platform sets a generic status message like *Actor exit with exit code 0*. However, you can provide more informative message using the SDK's exit methods.

**Basic exit**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/basic-commands#basic-exit)

Use the exit() method to terminate the Actor with a custom status message:

* **JavaScript**
* **Python**

import { Actor } from 'apify';  
  
await Actor.init();  
*// ...*  
*// Actor will finish with 'SUCCEEDED' status*  
await Actor.exit('Succeeded, crawled 50 pages');

**Immediate exit**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/basic-commands#immediate-exit)

To exit immediately without calling exit handlers:

* **JavaScript**
* **Python**

import { Actor } from 'apify';  
  
await Actor.init();  
*// ...*  
*// Exit right away without calling `exit` handlers at all*  
await Actor.exit('Done right now', { timeoutSecs: 0 });

**Failed exit**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/basic-commands#failed-exit)

To indicate a failed run:

* **JavaScript**
* **Python**

import { Actor } from 'apify';  
  
await Actor.init();  
*// ...*  
*// Actor will finish with 'FAILED' status*  
await Actor.exit('Could not finish the crawl, try increasing memory', { exitCode: 1 });

**Preferred exit methods**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/basic-commands#preferred-exit-methods)

The SDK provides convenient methods for exiting Actors:

* Use exit() with custom messages to inform users about the Actor's achievements or issues.
* Use fail() as a shortcut for exit() when indicating an error. It defaults to an exit code of 1 and emits the exit event, allowing components to perform cleanup or state persistence.
* The exit() method also emits the exit event, enabling cleanup or state persistence.

Example of a failed exit using a shorthand method:

* **JavaScript**
* **Python**

import { Actor } from 'apify';  
  
await Actor.init();  
*// ...*  
*// Or nicer way using this syntactic sugar:*  
await Actor.fail('Could not finish the crawl, try increasing memory');

**Exit event handlers (JavaScript only)**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/basic-commands#exit-event-handlers-javascript-only)

In JavaScript, you can register handlers for the exit event:

* **JavaScript**
* **Python**

import { Actor } from 'apify';  
  
await Actor.init();  
  
*// Register a handler to be called on exit.*  
*// Note that the handler has `timeoutSecs` to finish its job.*  
Actor.on('exit', ({ statusMessage, exitCode, timeoutSecs }) => {  
 *// Perform cleanup...*  
});  
  
await Actor.exit();

**Actor environment variables**

**Learn how to provide your Actor with context that determines its behavior through a plethora of pre-defined environment variables set by the Apify platform.**

**System environment variables**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/environment-variables#system-environment-variables)

Apify sets several system environment variables for each Actor run. These variables provide essential context and information about the Actor's execution environment.

Here's a table of key system environment variables:

| **Environment Variable** | **Description** |
| --- | --- |
| ACTOR\_ID | ID of the Actor. |
| ACTOR\_FULL\_NAME | Full technical name of the Actor, in the format owner-username/actor-name. |
| ACTOR\_RUN\_ID | ID of the Actor run. |
| ACTOR\_BUILD\_ID | ID of the Actor build used in the run. |
| ACTOR\_BUILD\_NUMBER | Build number of the Actor build used in the run. |
| ACTOR\_BUILD\_TAGS | A comma-separated list of tags of the Actor build used in the run. Note that this environment variable is assigned at the time of start of the Actor and doesn't change over time, even if the assigned build tags change. |
| ACTOR\_TASK\_ID | ID of the Actor task. Empty if Actor is run outside of any task, e.g. directly using the API. |
| ACTOR\_EVENTS\_WEBSOCKET\_URL | Websocket URL where Actor may listen for [events](https://docs.apify.com/platform/actors/development/programming-interface/system-events) from Actor platform. |
| ACTOR\_DEFAULT\_DATASET\_ID | Unique identifier for the default dataset associated with the current Actor run. |
| ACTOR\_DEFAULT\_KEY\_VALUE\_STORE\_ID | Unique identifier for the default key-value store associated with the current Actor run. |
| ACTOR\_DEFAULT\_REQUEST\_QUEUE\_ID | Unique identifier for the default request queue associated with the current Actor run. |
| ACTOR\_INPUT\_KEY | Key of the record in the default key-value store that holds the [Actor input](https://docs.apify.com/platform/actors/running/input-and-output#input). |
| ACTOR\_MAX\_PAID\_DATASET\_ITEMS | For paid-per-result Actors, the user-set limit on returned results. Do not exceed this limit. |
| ACTOR\_MAX\_TOTAL\_CHARGE\_USD | For pay-per-event Actors, the user-set limit on run cost. Do not exceed this limit. |
| APIFY\_HEADLESS | If **1**, web browsers inside the Actor should run in headless mode (no windowing system available). |
| APIFY\_IS\_AT\_HOME | Contains **1** if the Actor is running on Apify servers. |
| ACTOR\_MEMORY\_MBYTES | Size of memory allocated for the Actor run, in megabytes. Can be used to optimize memory usage or finetuning of low-level external libraries. |
| APIFY\_PROXY\_PASSWORD | Password for accessing Apify Proxy services. This password enables the Actor to utilize proxy servers on behalf of the user who initiated the Actor run. |
| APIFY\_PROXY\_PORT | TCP port number to be used for connecting to the Apify Proxy. |
| APIFY\_PROXY\_STATUS\_URL | URL for retrieving proxy status information. Appending ?format=json to this URL returns the data in JSON format for programmatic processing. |
| ACTOR\_STANDBY\_URL | URL for accessing web servers of Actor runs in the [Actor Standby](https://docs.apify.com/platform/actors/development/programming-interface/standby) mode. |
| ACTOR\_STARTED\_AT | Date when the Actor was started. |
| ACTOR\_TIMEOUT\_AT | Date when the Actor will time out. |
| APIFY\_TOKEN | API token of the user who started the Actor. |
| APIFY\_USER\_ID | ID of the user who started the Actor. May differ from the Actor owner. |
| ACTOR\_WEB\_SERVER\_PORT | TCP port for the Actor to start an HTTP server on. This server can be used to receive external messages or expose monitoring and control interfaces. The server also receives messages from the [Actor Standby](https://docs.apify.com/platform/actors/development/programming-interface/standby) mode. |
| ACTOR\_WEB\_SERVER\_URL | Unique public URL for accessing the Actor run web server from the outside world. |
| APIFY\_API\_PUBLIC\_BASE\_URL | Public URL of the Apify API. May be used to interact with the platform programmatically. Typically set to api.apify.com. |
| APIFY\_DEDICATED\_CPUS | Number of CPU cores reserved for the actor, based on allocated memory. |
| APIFY\_DISABLE\_OUTDATED\_WARNING | Controls the display of outdated version warnings. Set to 1 to suppress notifications about updates. |
| APIFY\_WORKFLOW\_KEY | Identifier used for grouping related runs and API calls together. |
| APIFY\_META\_ORIGIN | Specifies how an Actor run was started. Possible values are [here](https://docs.apify.com/platform/actors/running/runs-and-builds#origin) |
| APIFY\_SDK\_LATEST\_VERSION | Specifies the most recent release version of the Apify SDK for JavaScript. Used for checking for updates. |
| APIFY\_INPUT\_SECRETS\_KEY\_FILE | Path to the secret key used to decrypt [Secret inputs](https://docs.apify.com/platform/actors/development/actor-definition/input-schema/secret-input). |
| APIFY\_INPUT\_SECRETS\_KEY\_PASSPHRASE | Passphrase for the input secret key specified in APIFY\_INPUT\_SECRETS\_KEY\_FILE. |

**Date format**

All date-related variables use the UTC timezone and are in [ISO 8601](https://en.wikipedia.org/wiki/ISO_8601) format (e.g., *2022-07-13T14:23:37.281Z*).

**Access environment variables**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/environment-variables#access-environment-variables)

You can access environment variables in your code as follows:

* **JavaScript**
* **Python**

In Node.js, use the process.env object:

console.log(process.env.APIFY\_USER\_ID);

**Use the Configuration class**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/environment-variables#use-the-configuration-class)

For more convenient access to Actor configuration, use the [Configuration](https://docs.apify.com/sdk/js/reference/class/Configuration) class

* **JavaScript**
* **Python**

import { Actor } from 'apify';  
  
await Actor.init();  
  
*// get current token*  
const token = Actor.config.get('token');  
*// use different token*  
Actor.config.set('token', 's0m3n3wt0k3n');  
  
await Actor.exit();

**Custom environment variables**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/environment-variables#custom-environment-variables)

Actor owners can define custom environment variables to pass additional configuration to their Actors. To set custom variables:

1. Go to your Actor's **Source** page in the Apify Console
2. Navigate to the **Environment variables** section.
3. Add your custom variables.

For sensitive data like API keys or passwords, enable the **Secret** option. This encrypt the value and redacts it from logs to prevent accidental exposure.

**Build-time variables**

Custom environment variables are set during the Actor's build process and cannot be changed for existing builds. For more information, check out the [Builds](https://docs.apify.com/platform/actors/development/builds-and-runs/builds) page.

**Build-time environment variables**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/environment-variables#build-time-environment-variables)

You can also use environment variables during the Actor's build process. In this case, they function as Docker build arguments. To use them in your Dockerfile, include ARG instruction:

ARG MY\_BUILD\_VARIABLE  
RUN echo $MY\_BUILD\_VARIABLE

**Insecure build variables**

Build-time environment variables are not suitable for secrets, as they are not encrypted.

By leveraging environment variables effectively, you can create more flexible and configurable Actors that adapt to different execution contexts and user requirements.

**Status messages**

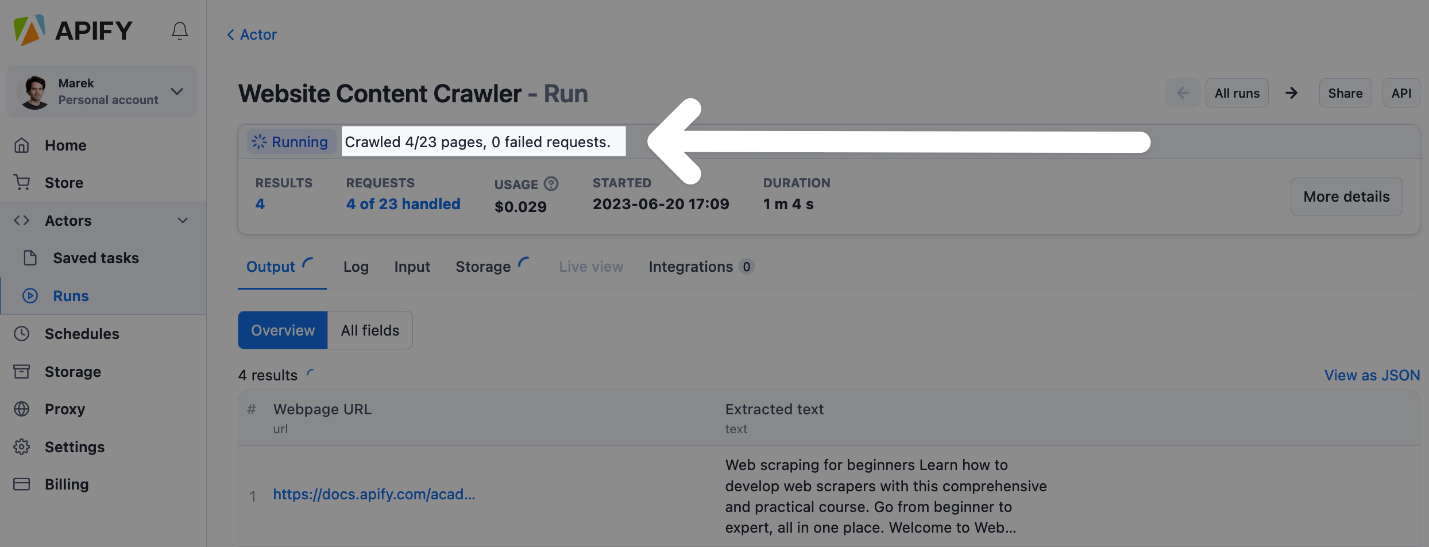
**Learn how to use custom status messages to inform users about an Actor's progress.**

Each Actor run has a status, represented by the status field. The following table describes the possible values:

| **Status** | **Type** | **Description** |
| --- | --- | --- |
| READY | initial | Started but not allocated to any worker yet |
| RUNNING | transitional | Executing on a worker |
| SUCCEEDED | terminal | Finished successfully |
| FAILED | terminal | Run failed |
| TIMING-OUT | transitional | Timing out now |
| TIMED-OUT | terminal | Timed out |
| ABORTING | transitional | Being aborted by user |
| ABORTED | terminal | Aborted by user |

**Status messages**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/status-messages#status-messages)

In addition to the status, each Actor run has a status message (the statusMessage field). This message informs users about the Actor's current activity, enhancing the user experience.



**Exit status message**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/status-messages#exit-status-message)

When an Actor exits, the status message is set to either:

* A default text (e.g., *Actor finished with exit code 1*)
* A custom message (see the [exit](https://docs.apify.com/platform/actors/development/programming-interface/basic-commands#exit-actor) method for details)

**Update status message**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/status-messages#update-status-message)

To keep users informed during the Actor's execution, update the status message periodically. Use the following code to set a status message:

* **JavaScript**
* **Python**

import { Actor } from 'apify';  
  
await Actor.init();  
  
*// ...*  
await Actor.setStatusMessage('Crawled 45 of 100 pages');  
  
await Actor.exit();

**System events in Apify Actors**

**Learn about system events sent to your Actor and how to benefit from them.**

**Understand system events**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/system-events#understand-system-events)

Apify's system notifies Actors about various events, such as:

* Migration to another server
* Abort operations triggered by another Actor
* CPU overload

These events help you manage your Actor's behavior and resources effectively.

**System events**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/system-events#system-events)

The following table outlines the system events available:

| **Event name** | **Payload** | **Description** |
| --- | --- | --- |
| cpuInfo | { isCpuOverloaded: Boolean } | Emitted approximately every second, indicating whether the Actor is using maximum available CPU resources. |
| migrating | { timeRemainingSecs: Float } | Signals that the Actor will soon migrate to another worker server on the Apify platform. |
| aborting | N/A | Triggered when a user initiates a graceful abort of an Actor run, allowing time for cleanup. |
| persistState | { isMigrating: Boolean } | Emitted at regular intervals (default: *60 seconds*) to notify Apify SDK components to persist their state. |

**How system events work**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/system-events#how-system-events-work)

Actors receive system events through a WebSocket connection. The address is specified by the ACTOR\_EVENTS\_WEBSOCKET\_URL environment variable. Messages are sent in JSON format with the following structure:

{  
 *// Event name*  
 name: String,  
  
 *// Time when the event was created, in ISO format*  
 createdAt: String,  
  
 *// Optional object with payload*  
 data: Object,  
}

**Virtual events**

Some events like persistState, are generated virtually at the Actor SDK level, not sent via WebSocket.

**Handle system events**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/system-events#handle-system-events)

To work with system events in your Actor, use the following methods:

* **JavaScript**
* **Python**

import { Actor } from 'apify';  
  
await Actor.init();  
  
*// Add event handler*  
Actor.on('cpuInfo', (data) => {  
 if (data.isCpuOverloaded) console.log('Oh no, we need to slow down!');  
});  
  
*// Remove all handlers for a specific event*  
Actor.off('systemInfo');  
  
*// Remove a specific event handler*  
Actor.off('systemInfo', handler);  
  
await Actor.exit();

By utilizing these system events, you can create more robust and efficient Actors that respond dynamically to changes in their environment.

**Container web server**

**Learn about how to run a web server inside your Actor to enable communication with the outside world through both UI and API.**

Each Actor run is assigned a unique URL (e.g. kmdo7wpzlshygi.runs.apify.net) that allows HTTP access to an optional web server running inside the Actor's Docker container. This feature enhances your Actor's capabilities by enabling external communication.

**Using Actors as an API**

The container web server provides a way how to connect to one specific Actor run. To enable using your Actor as an API, with a pre-defined hostname, load balancing and autoscaling, check out [Actor Standby](https://docs.apify.com/platform/actors/development/programming-interface/standby).

**Access the container URL**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/container-web-server#access-the-container-url)

You can find the container URL in three locations:

* In the web application, on the Actor run details page as the **Container URL** field.
* In the API as the containerUrl property of the [Run object](https://docs.apify.com/api/v2#/reference/actors/run-object/get-run).
* In the Actor run's container as the ACTOR\_WEB\_SERVER\_URL environment variable.

**Set up the web server**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/container-web-server#set-up-the-web-server)

The web server inside the container must listen on the port specified by the ACTOR\_WEB\_SERVER\_PORT environment variable (typically: *4321*). To use a different port:

1. Go to your Actor version configuration
2. Define the ACTOR\_WEB\_SERVER\_PORT environment variable with your desired port number.

Check out [Custom environment variables](https://docs.apify.com/platform/actors/development/programming-interface/environment-variables) for more details.

**Example: Start a simple web server**[**​**](https://docs.apify.com/platform/actors/development/programming-interface/container-web-server#example-start-a-simple-web-server)

* **JavaScript**
* **Python**

Here's how to start a basic web server in your Actor using Express.js:

*// npm install express*  
import { Actor } from 'apify';  
import express from 'express';  
  
await Actor.init();  
  
const app = express();  
const port = process.env.ACTOR\_WEB\_SERVER\_PORT;  
  
app.get('/', (req, res) => {  
 res.send('Hello world from Express app!');  
});  
  
app.listen(port, () => console.log(`Web server is listening  
 and can be accessed at  
 ${process.env.ACTOR\_WEB\_SERVER\_URL}!`));  
  
*// Let the Actor run for an hour*  
await new Promise((r) => setTimeout(r, 60 \* 60 \* 1000));  
  
await Actor.exit();

**Deployment**

**Learn how to deploy your Actors to the Apify platform and build them.**

Deploying an Actor involves uploading your [source code](https://docs.apify.com/platform/actors/development/actor-definition) and [building](https://docs.apify.com/platform/actors/development/builds-and-runs/builds) it on the Apify platform. Once deployed, you can run and scale your Actor in the cloud.

**Deploy using Apify CLI**[**​**](https://docs.apify.com/platform/actors/development/deployment#deploy-using-apify-cli)

The fastest way to deploy and build your Actor is by using the [Apify CLI](https://docs.apify.com/cli). If you've completed one of the tutorials from the [academy](https://docs.apify.com/academy), you should have already have it installed. If not, follow the [Apify CLI installation instructions](https://docs.apify.com/cli/docs/installation).

To deploy your Actor using Apify CLI:

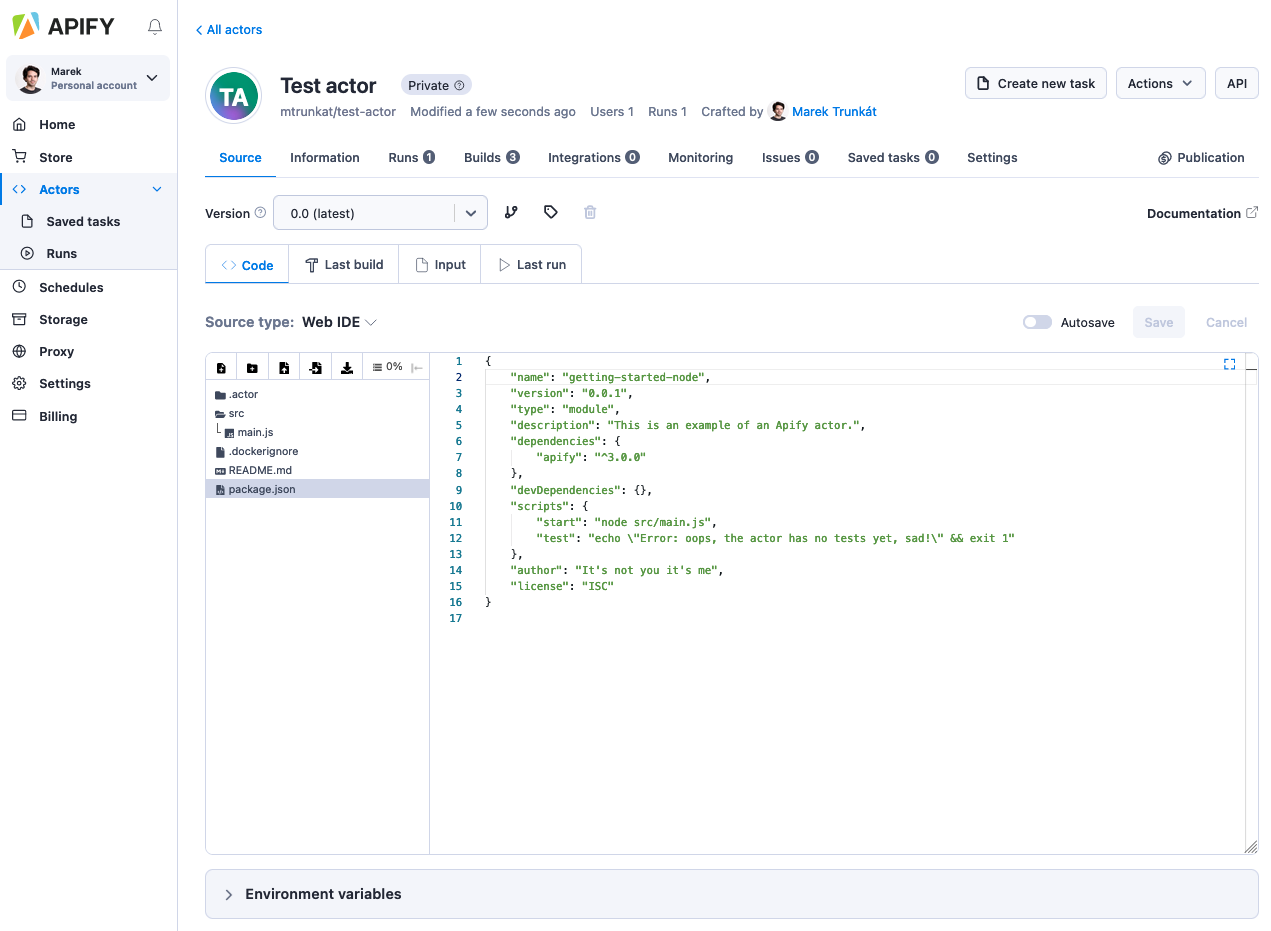
1. Log in to your Apify account:

apify login

1. Navigate to the directory of your Actor on your local machine.
2. Deploy your Actor by running:

apify push

When you deploy using the CLI, your source code is uploaded as "multiple source files" and is visible and editable in the Web IDE.



**Source files size limit**

The CLI deploys code as multiple source files up to 3 MB. Beyond that, it deploys as a Zip file.

**Pull an existing Actor**[**​**](https://docs.apify.com/platform/actors/development/deployment#pull-an-existing-actor)

You can also pull an existing Actor from the Apify platform to your local machine using apify pull command

apify pull [ACTORID]

This command fetches the Actor's files to your current directory. If the Actor is defined as a Git repository, it will be cloned, for Actors defined in the Web IDE, the command will fetch the files directly.

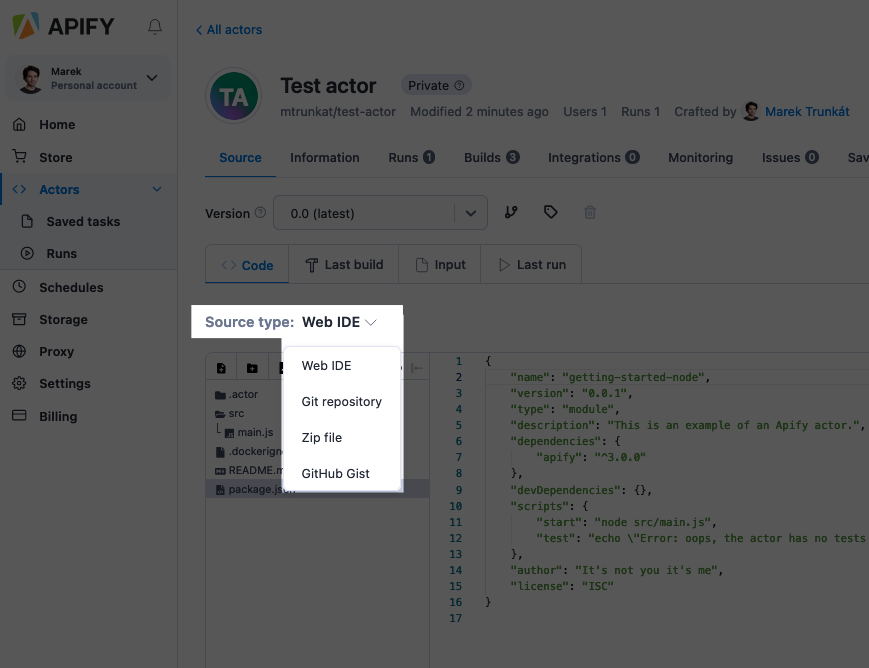
You can specify a particular version of the Actor to pull by using the --version flag:

apify pull [ACTORID] --version=1.2

If you don't provide the ACTORID, the command will update the Actor in the current directory based on its name in the .actor/actor.json file.

**Alternative deployment methods**[**​**](https://docs.apify.com/platform/actors/development/deployment#alternative-deployment-methods)

To deploy using other methods, first create the Actor manually through Apify CLI or Apify Console, then change its source type:



You can link your Actor to a Git repository, Gist, or a Zip file.

For more information on alternative source types, check out next chapter.

**Source types**

**Learn about Apify Actor source types and how to deploy an Actor from GitHub using CLI or Gist.**

This section explains the various sources types available for Apify Actors and how to deploy an Actor from GitHub using CLI or Gist. Apify Actors supports four source types:

* [Web IDE](https://docs.apify.com/platform/actors/development/deployment/source-types#web-ide)
* [Git repository](https://docs.apify.com/platform/actors/development/deployment/source-types#git-repository)
  + [Private repositories](https://docs.apify.com/platform/actors/development/deployment/source-types#private-repositories)
    - [How to configure deployment keys](https://docs.apify.com/platform/actors/development/deployment/source-types#how-to-configure-deployment-keys)
  + [Actor monorepos](https://docs.apify.com/platform/actors/development/deployment/source-types#actor-monorepos)
* [Zip file](https://docs.apify.com/platform/actors/development/deployment/source-types#zip-file)
* [GitHub Gist](https://docs.apify.com/platform/actors/development/deployment/source-types#github-gist)

**Web IDE**[**​**](https://docs.apify.com/platform/actors/development/deployment/source-types#web-ide)

This is the default option when your Actor's source code is hosted on the Apify platform. It offers quick previews and updates to your source code, easy file and directory browsing, and direct testing of the [INPUT\_SCHEMA.json](https://docs.apify.com/platform/actors/development/actor-definition/input-schema) on the Apify platform.

A Dockerfile is mandatory for all Actors. When using the default NodeJS Dockerfile, you'll typically need main.js for your source code and package.json for [NPM](https://www.npmjs.com/) package configurations.

For more information on creating custom Dockerfiles or using Apify's base images, refer to the [Dockerfile](https://docs.apify.com/platform/actors/development/actor-definition/dockerfile#custom-dockerfile) and [base Docker images](https://docs.apify.com/platform/actors/development/actor-definition/dockerfile#base-docker-images) documentation.

**Git repository**[**​**](https://docs.apify.com/platform/actors/development/deployment/source-types#git-repository)

Hosting your Actor's source code in a Git repository allows for multiple files and directories, a custom Dockerfile for build process control, and a user description fetched from README.md. Specify the repository location using the **Git URL** setting with https, git, or ssh protocols.

To deploy an Actor from GitHub, set the **Source Type** to **Git repository** and enter the GitHub repository URL in the **Git URL** field. You can optionally specify a branch or tag by adding a URL fragment (e.g., #develop).

To use a specific directory, add it after the branch/tag, separated by a colon (e.g., #develop:some/dir)

**GitHub integration**

You can easily set up an integration where the Actor is automatically rebuilt on every commit to the Git repository. For more details, see [GitHub integration](https://docs.apify.com/platform/integrations/github).

**Private repositories**[**​**](https://docs.apify.com/platform/actors/development/deployment/source-types#private-repositories)

When using a private Git repository for your Actor's source code, you need to configure a deployment key. This key grants Apify secure, read-only access to your repository.

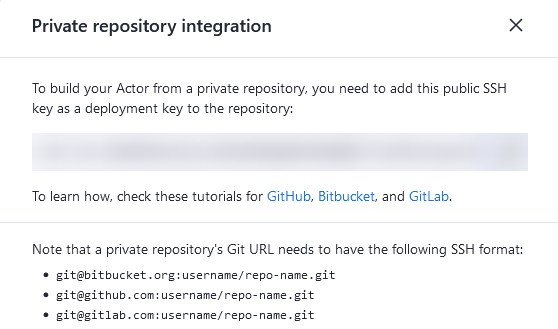
**What are deployment keys**

Deployment keys are unique SSH keys that allow Apify to clone and build your Actor from a private repository.

**How to configure deployment keys**[**​**](https://docs.apify.com/platform/actors/development/deployment/source-types#how-to-configure-deployment-keys)

To configure the deployment key for your Actor's source code you need to:

1. In your Actor's settings, find the **Git URL** input field
2. Click the **deployment key** link below the input field
3. Follow the instruction to add the key to your Git hosting service.



By using deployment keys, you enable secure, automated builds of your Actor from private repositories.

**Key usage limitations**

Remember that each key can only be used once per Git hosting service (GitHub, Bitbucket, GitLab)

**Actor monorepos**[**​**](https://docs.apify.com/platform/actors/development/deployment/source-types#actor-monorepos)

To manage multiple Actors in a single repository, use the dockerContextDix property in the [Actor definition](https://docs.apify.com/platform/actors/development/actor-definition/actor-json) to set the Docker context directory (if not provided then the repository root is used). In the Dockerfile, copy both the Actor's source and any shared code into the Docker image.

To enable sharing Dockerfiles between multiple Actors, the Actor build process passes the ACTOR\_PATH\_IN\_DOCKER\_CONTEXT build argument to the Docker build. It contains the relative path from dockerContextDir to the directory selected as the root of the Actor in the Apify Console (the "directory" part of the Actor's git URL).

For an example, see the [apify/actor-monorepo-example](https://github.com/apify/actor-monorepo-example) repository. To build Actors from this monorepo, you would set the source URL (including branch name and folder) as https://github.com/apify/actor-monorepo-example#main:actors/javascript-actor and https://github.com/apify/actor-monorepo-example#main:actors/typescript-actor respectively.

**Zip file**[**​**](https://docs.apify.com/platform/actors/development/deployment/source-types#zip-file)

Actors can also use source code from a Zip archive hosted on an external URL. This option supports multiple files and directories, allows for custom Dockerfile, and uses README.md for the Actor description. If not using a [custom Dockerfile](https://docs.apify.com/platform/actors/development/actor-definition/dockerfile#custom-dockerfile), ensure your main file is named main.js.

**Automatic use of ZIP file**

This source type is used automatically when you are using Apify-CLI and the source size exceeds 3MB.

**GitHub Gist**[**​**](https://docs.apify.com/platform/actors/development/deployment/source-types#github-gist)

For smaller projects, GitHub Gist offers a simpler alternative to full Git repositories or hosted Zip files. To use a GitHub Gist, create your Gist at <https://gist.github.com/>, set the **Source type** to **GitHub Gist**, and paste the Gist URL in the provided field.

Like other source types, Gists can include multiple files, directories, and a custom Dockerfile. The Actor description is taken from README.md.

By understanding these source types, you can choose the most appropriate option for hosting and deploying your Apify Actors. Each type offers unique advantages, allowing you to select the best fit for your project's size, complexity, and collaboration needs.

**Continuous integration for Actors**

**Learn how to set up automated builds, deploys, and testing for your Actors using GitHub Actions or Bitbucket Pipelines.**

Automating your Actor development process can save time and reduce errors, especially for projects with multiple Actors or frequent updates. Instead of manually pushing code, building Actors, and running tests, you can automate these steps to run whenever you push code to your repository.

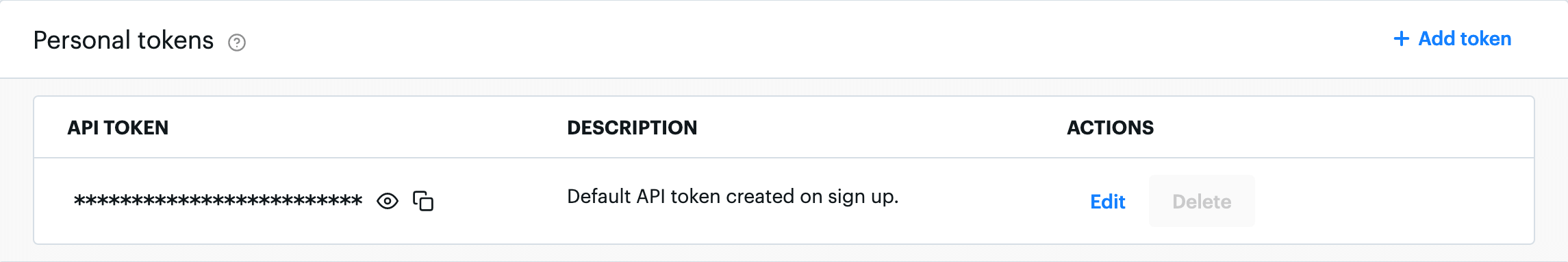
You can automate Actor builds and tests using your Git repository's automated workflows like [GitHub Actions](https://github.com/features/actions) or [Bitbucket Pipelines](https://www.atlassian.com/software/bitbucket/features/pipelines).

This article focuses on GitHub, but [we also have a guide for Bitbucket](https://help.apify.com/en/articles/6988586-setting-up-continuous-integration-for-apify-actors-on-bitbucket).

**Set up automated builds and tests**[**​**](https://docs.apify.com/platform/actors/development/deployment/continuous-integration#set-up-automated-builds-and-tests)

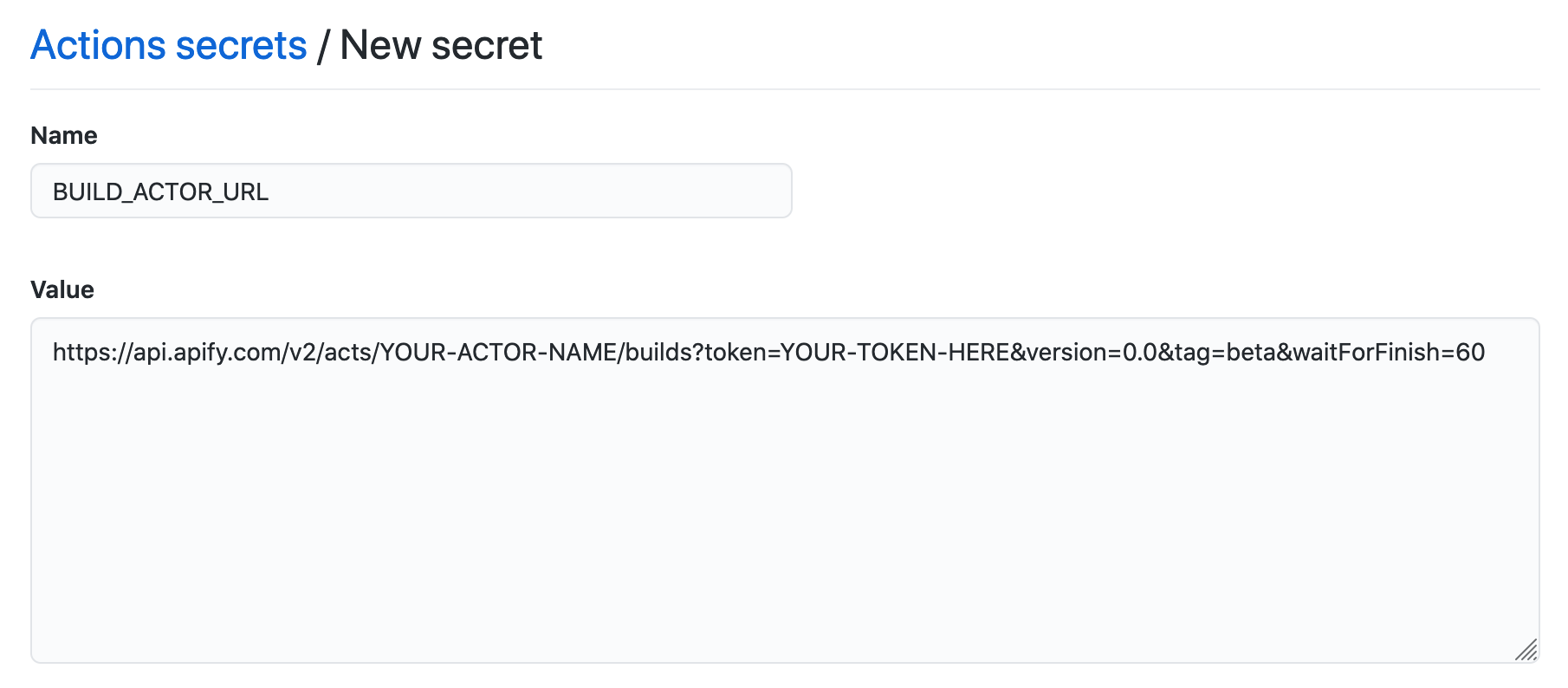
To set up automated builds and tests for your Actors you need to:

1. Create a GitHub repository for your Actor code.
2. Get your Apify API token from the [Apify Console](https://console.apify.com/settings/integrations)



1. Add your Apify token to GitHub secrets
   1. Go to your repository > Settings > Secrets > New repository secret
   2. Name the secret & paste in your token
2. Add the Builds Actor API endpoint URL to GitHub secrets
   1. Use this format:

https://api.apify.com/v2/acts/YOUR-ACTOR-NAME/builds?token=YOUR-TOKEN-HERE&version=0.0&tag=beta&waitForFinish=60



* 1. Name the secret

1. Create GitHub Actions workflow files:
   1. In your repository, create the .github/workflows directory
   2. Add latest.yml and beta.yml files with the following content
   3. **latest.yml**
   4. **beta.yml**

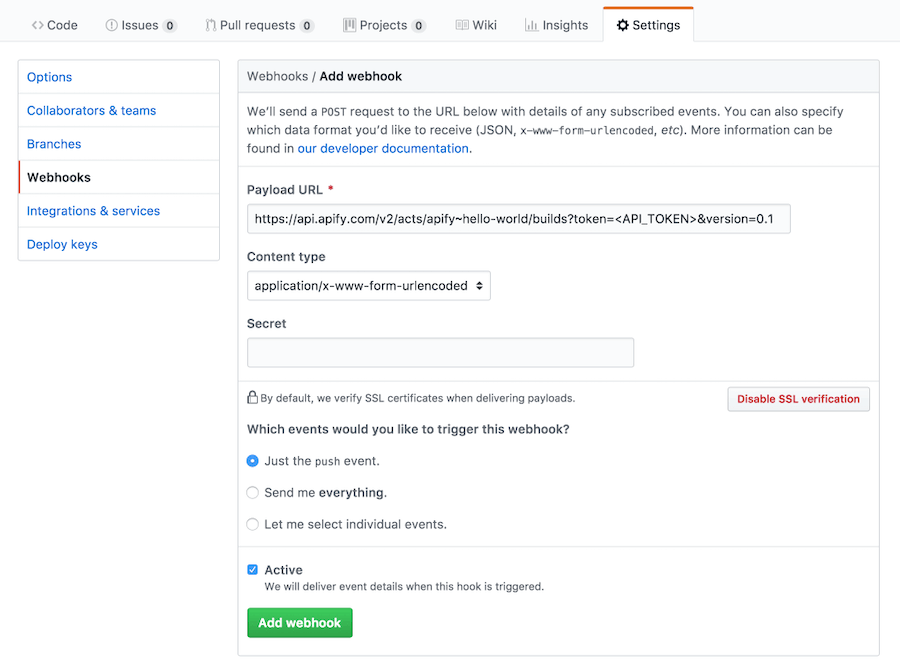
name: Test and build latest version  
on:  
 push:  
 branches:  
 - master  
 - main  
jobs:  
 test:  
 runs-on: ubuntu-latest  
 steps:  
 *# Install dependencies and run tests*  
 - uses: actions/checkout@v2  
 - run: npm install && npm run test  
 *# Build latest version*  
 - uses: distributhor/workflow-webhook@v1  
 env:  
 webhook\_url: ${{ secrets.LATEST\_BUILD\_URL }}  
 webhook\_secret: ${{ secrets.APIFY\_TOKEN }}

With this setup, pushing to the main or master branch builds a new latest version.

**GitHub integration**[**​**](https://docs.apify.com/platform/actors/development/deployment/continuous-integration#github-integration)

To set up automatic builds from GitHub:

1. Go to your Actor's detail page and coy the Build Actor API endpoint URL from the API tab.
2. In your GitHub repository, go to Settings > Webhooks > Add webhook.
3. Paste the API URL into the Payload URL field.



Now your Actor will automatically rebuild on every push to the GitHub repository.

**Monetize your Actor**

**Learn how you can monetize your web scraping and automation projects by publishing Actors to users in Apify Store.**

Apify Store allows you to monetize your web scraping and automation projects by publishing them as paid Actors. This guide explains the available pricing models and how to get started.

**Pricing models**[**​**](https://docs.apify.com/platform/actors/publishing/monetize#pricing-models)

Actors in Apify Store can be published under one of the following pricing models:

* 1. **Free**: Users can run the Actor without any additional charges beyond the platform usage costs generated by the Actor.
  2. **Rental**: Users pay for the platform usage costs. However, after a trial period, they need to pay a flat monthly fee to the developer to continue using the Actor.
  3. **Pay per result (PPR)**: Users don't pay for the platform usage costs. Instead, they pay the developer based on the number of results produced by the Actor.
  4. **Pay per event (PPE)**: Users don't pay for the platform usage cost the Actor generates. Instead, they pay based on specific events that are programmatically triggered from the Actor's source code. These events are defined by the developer and can include actions such as generating a single result or starting an Actor.

**Rental pricing model**[**​**](https://docs.apify.com/platform/actors/publishing/monetize#rental-pricing-model)

With the rental model, you can specify a free trial period and a monthly rental price. After the trial, users with an [Apify paid plan](https://apify.com/pricing) can continue using your Actor by paying the monthly fee. You can receive 80% of the total rental fees collected each month.

Example - rental pricing model

**Pay-per-result pricing model**[**​**](https://docs.apify.com/platform/actors/publishing/monetize#pay-per-result-pricing-model)

In this model, you set a price per 1,000 results. Users are charged based on the number of results your Actor produces. Your profit is calculated as 80% of the revenue minus platform usage costs. The formula is:

(0.8 \* revenue) - costs = profit

**Pay-per-result unit pricing for cost computation**[**​**](https://docs.apify.com/platform/actors/publishing/monetize#pay-per-result-unit-pricing-for-cost-computation)

| **Service** | **Unit price** |
| --- | --- |
| Compute unit | **$0.4** / CU |
| Residential proxies | **$13** / GB |
| SERPs proxy | **$3** / 1,000 SERPs |
| Data transfer - external | **$0.20** / GB |
| Data transfer - internal | **$0.05** / GB |
| Dataset - reads | **$0.0004** / 1,000 reads |
| Dataset - writes | **$0.005** / 1,000 writes |
| Key-value store - reads | **$0.005** / 1,000 reads |
| Key-value store - writes | **$0.05** / 1,000 writes |
| Key-value store - lists | **$0.05** / 1,000 lists |
| Request queue - reads | **$0.004** / 1,000 reads |
| Request queue - writes | **$0.02** / 1,000 writes |

Only revenue and cost for Apify customers on paid plans are taken into consideration when computing your profit. Users on free plans are not reflected there.

Read more about Actors on Apify Store and different pricing models from the perspective of your users in the [Store documentation](https://docs.apify.com/platform/actors/running/actors-in-store).

Example - pay-per-result pricing model

**Best practices for PPR Actors**[**​**](https://docs.apify.com/platform/actors/publishing/monetize#best-practices-for-ppr-actors)

To ensure profitable operation:

* 1. Set memory limits in your [actor.json](https://docs.apify.com/platform/actors/development/actor-definition/actor-json) file to control platform usage costs
  2. Implement the ACTOR\_MAX\_PAID\_DATASET\_ITEMS check to prevent excess result generation
  3. Test your Actor with various result volumes to determine optimal pricing

**Pay-per-event pricing model**[**​**](https://docs.apify.com/platform/actors/publishing/monetize#pay-per-event-pricing-model)

The pay-per-event pricing model offers a flexible monetization option for Actors on Apify Store. Unlike pay per result, PPE allows you to charge users based on specific events triggered programmatically by your Actor's code.

**PPE vs. PPR**[**​**](https://docs.apify.com/platform/actors/publishing/monetize#ppe-vs-ppr)

Unlike PPR, which charges based on the number of results produced, PPE lets you define pricing for individual events. You can charge for specific events directly from your Actor by calling the PPE charging API. Common events include Actor start, dataset item creation, and external API calls.

For a detailed comparison of pricing models, refer to our [user-facing documentation](https://docs.apify.com/platform/actors/running/actors-in-store).

**How is profit computed**[**​**](https://docs.apify.com/platform/actors/publishing/monetize#how-is-profit-computed)

Your profit is calculated as follows:

profit = (0.8 \* revenue) - platform costs

where:

* 1. *Revenue*: The amount charged for events via the PPE [API](https://docs.apify.com/api/v2/post-charge-run) or through [JS](https://docs.apify.com/sdk/js/reference/class/Actor#charge)/[Python](https://docs.apify.com/sdk/python/reference/class/Actor#charge) SDK. You receive 80% of this revenue.
  2. *Platform costs*: The underlying platform usage costs for running the Actor, are calculated using the same unit pricing as PPR.

Only paid user activity is included in profit calculations.

**How to set pricing for PPE**[**​**](https://docs.apify.com/platform/actors/publishing/monetize#how-to-set-pricing-for-ppe)

* 1. *Understand your costs*: Analyze resource usage (e.g CPU, memory, proxies, external APIs) and identify cost drivers
  2. *Define clear events*: break your Actor's functionality into measurable, chargeable events.
  3. *Common use cases*:
     1. *For scraping*: combine Actor start and dataset items pricing to reflect setup and per-result cost.
     2. *Beyond scraping*: Account for integrations with external systems or external API calls.
  4. *External API costs*: Account for additional processing costs.
  5. *Test your pricing*: Run your Actor and analyze cost-effectiveness using a special dataset.
  6. *Communicate value*: Ensure pricing reflects the value provided and is competitive.

**Best practices for PPE Actors**[**​**](https://docs.apify.com/platform/actors/publishing/monetize#best-practices-for-ppe-actors)

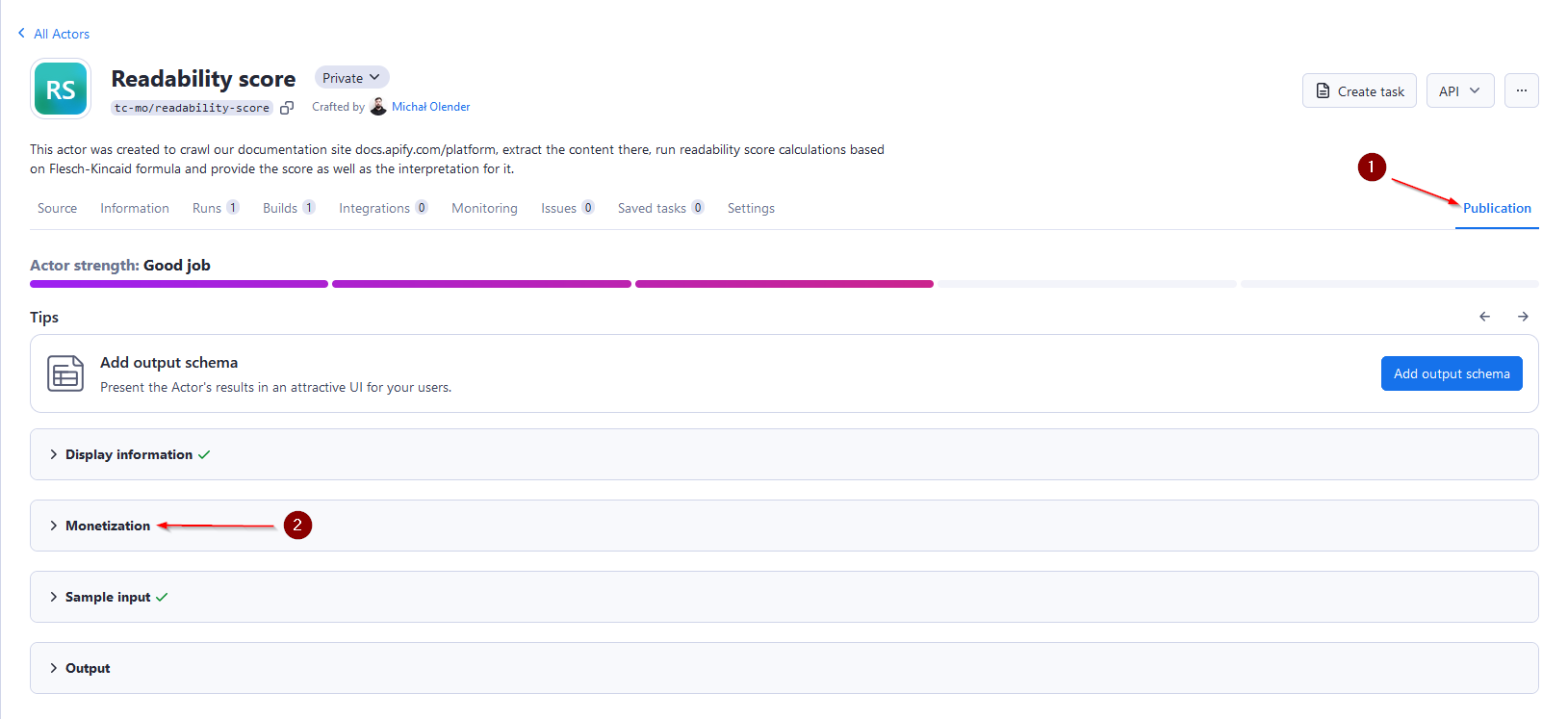
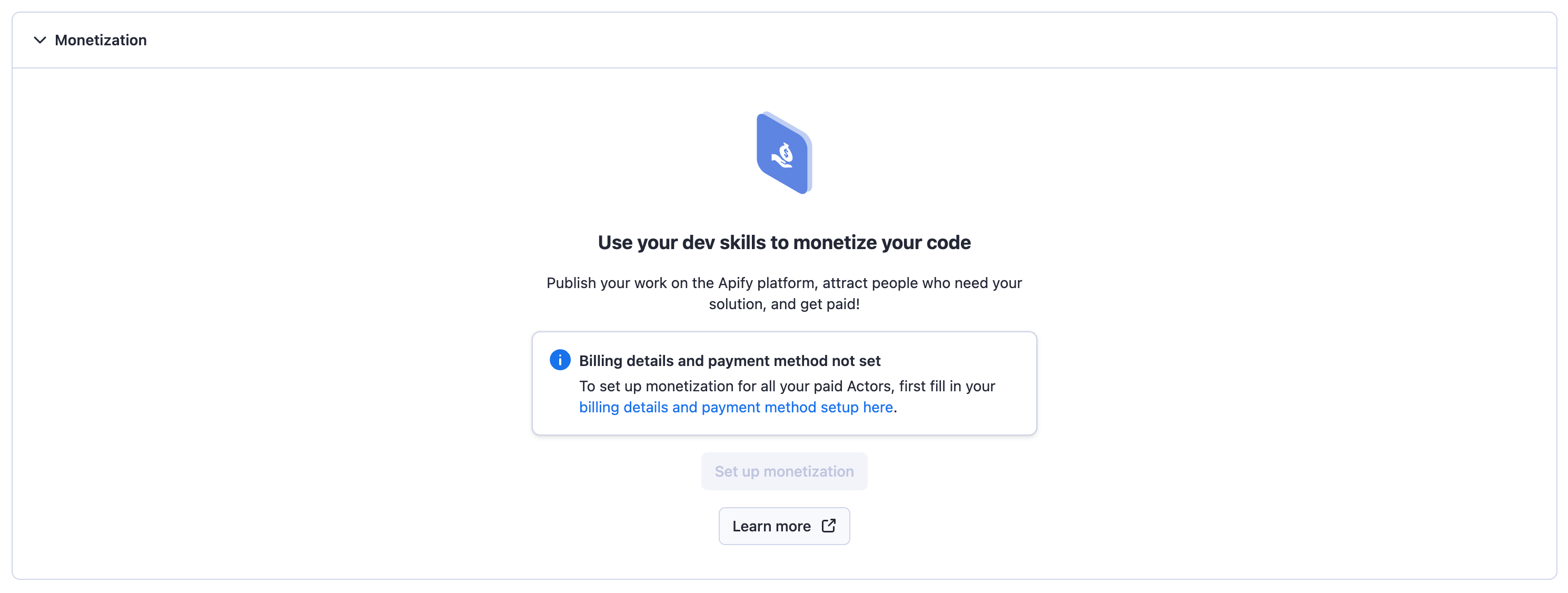
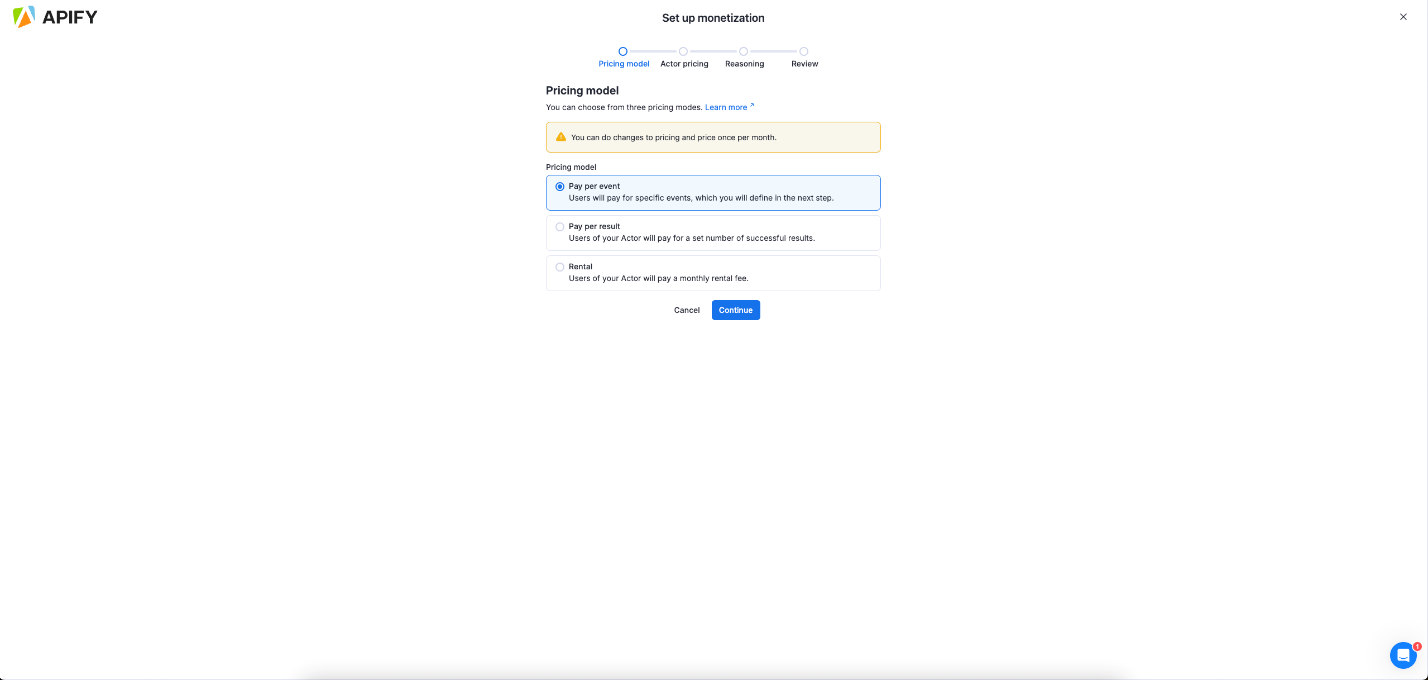
* 1. Set memory limits in your [actor.json](https://docs.apify.com/platform/actors/development/actor-definition/actor-json) file to control platform usage costs
  2. Avoid duplicate charges by not calling the start Actor event during migration.
  3. Ensure users are not charged beyond their set maximum. The [ChargingManager](https://github.com/metalwarrior665/actor-charge-manager-poc) or our SDKs ([JS](https://docs.apify.com/sdk/js/reference/class/ChargingManager) and [Python](https://docs.apify.com/sdk/python/reference/class/ChargingManager)) can handle this logic.
  4. Use idempotency keys in API calls to prevent double charges.
  5. Try to limit the number of events. Fewer events make it easier for users to understand your pricing.
  6. Try to make your event have tangible artifacts that users can see and understand (this might not be possible when using external APIs) i.e:
     1. Get a record
     2. Save it to a dataset
     3. etc.

**ChargingManager usage**

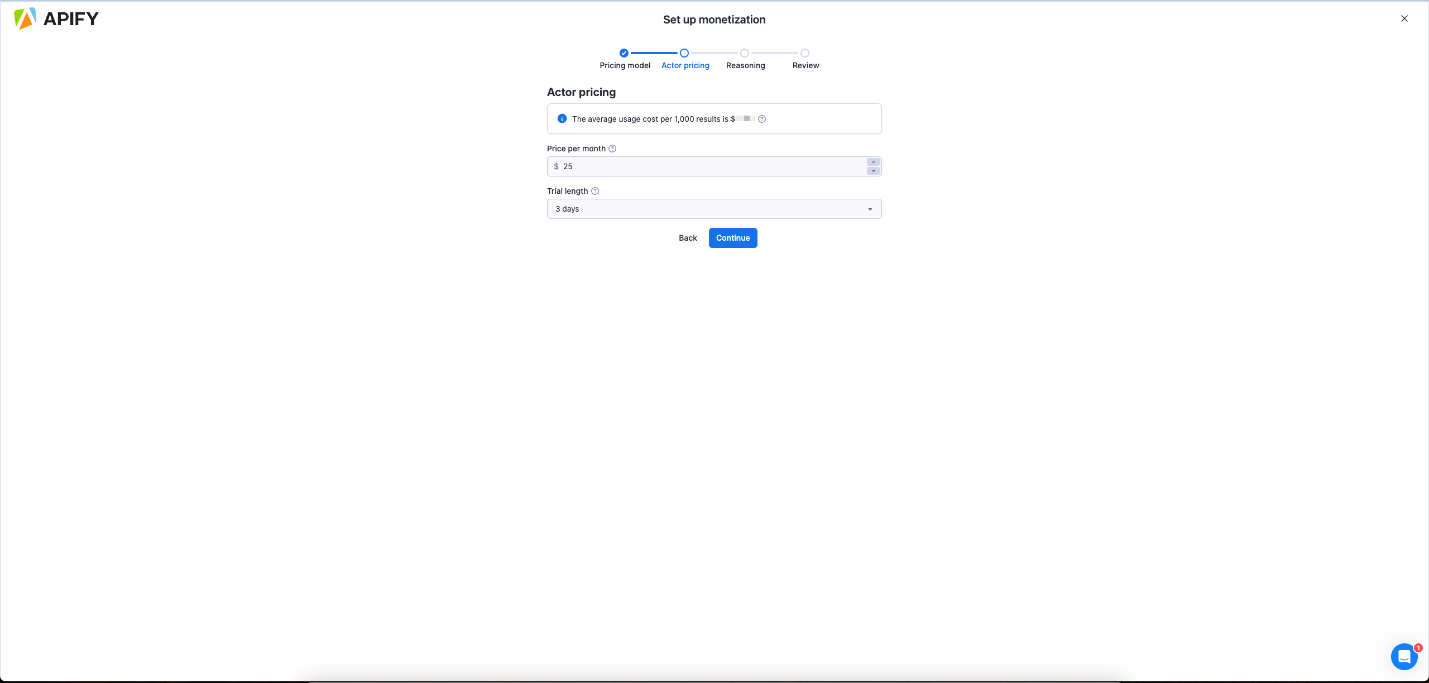
Use our [ChargingManager](https://github.com/metalwarrior665/actor-charge-manager-poc) or SDKs ([JS](https://docs.apify.com/sdk/js/reference/class/ChargingManager) and, [Python](https://docs.apify.com/sdk/python/reference/class/ChargingManager) or use [apify actor charge](https://docs.apify.com/cli/docs/next/reference#apify-actor-charge-eventname) when using our Apify CLI) to simplify PPE implementation into your Actor. This tool can handle pricing, usage tracking, idempotency keys, API errors, and, event charging via an API.

You can also choose not to use it, but then you must handle API integration and possible edge cases manually. You can use ChargingManager code as a reference.

**Setting up monetization**[**​**](https://docs.apify.com/platform/actors/publishing/monetize#setting-up-monetization)

Navigate to your [Actor page](https://console.apify.com/actors?tab=my) in Apify Console, choose the Actor that you want to monetize, and select the Publication tab.Open the Monetization section and complete your billing and payment details.Choose the pricing model for your Actor.Follow the monetization wizard to configure your pricing model.

* 1. **Rental**
  2. **Pay-per-result**
  3. **Pay-per-event**



**Changing monetization**[**​**](https://docs.apify.com/platform/actors/publishing/monetize#changing-monetization)

You can change the monetization setting of your Actor by using the same wizard as for the setup in the **Monetization** section of your Actor's **Publication** tab. Any changes made to an already published Actor will take *14 days* to come into effect, so that the users of your Actor have time to prepare.

**Frequency of monetization adjustments**

Be aware that you can change the monetization setting of each Actor only once per month. For further information and guidelines, please refer to our [Terms & Conditions](https://apify.com/store-terms-and-conditions)

**Payouts and analytics**[**​**](https://docs.apify.com/platform/actors/publishing/monetize#payouts-and-analytics)

Payout invoices are generated automatically on the 14th of each month. Review your invoice in the **Settings > Payout** section within one week. If not approved by the 20th, the system will auto-approve on the 21st.

Track your Actor's performance through:

* 1. The payout section for financial records
  2. [Actor Analytics](https://console.apify.com/actors/development/analytics) (**Actors > Development > Analytics**) for usage statistics

**Promoting your Actor**[**​**](https://docs.apify.com/platform/actors/publishing/monetize#promoting-your-actor)

Create serach-engine-optimized descriptions and README files to improve search engine visibility. Share your Actor on multiple channels:

* 1. Post on Reddit, Quora, and social media platforms
  2. Create tutorial videos demonstrating key features
  3. Publish articles about your Actor on relevant websites
  4. Consider creating a product showcase on platforms like Product Hunt

Remember to tag Apify in your social media posts for additional exposure. Effective promotion can significantly impact your Actor's success, differentiating between those with many paid users and those with few to none.

**Schedules**

**Learn how to automatically start your Actor and task runs and the basics of cron expressions. Set up and manage your schedules from Apify Console or via API.**

Schedules allow you to run your Actors and tasks at specific times. You schedule the run frequency using [cron expressions](https://docs.apify.com/platform/schedules#cron-expressions).

**Timezone & Daylight Savings Time**

Schedules allow timezone settings and support daylight saving time shifts (DST).

You can set up and manage your Schedules using:

* [Apify Console](https://console.apify.com/schedules)
* [Apify API](https://docs.apify.com/api/v2#/reference/schedules)
* [JavaScript API client](https://docs.apify.com/api/client/js/reference/class/ScheduleClient)
* [Python API client](https://docs.apify.com/api/client/python/reference/class/ScheduleClient)

When scheduling a new Actor or task run, you can override its input settings using a JSON object similarly to when invoking an Actor or task using the [Apify REST API](https://docs.apify.com/api/v2#/reference/schedules/).

**Events Startup Variability**

In most cases, scheduled events are fired within one second of their scheduled time.  
However, runs can be delayed because of a system overload or a server shutting down.

Each schedule can be associated with a maximum of *10* Actors and *10* Actor tasks.

**Setting up a new schedule**[**​**](https://docs.apify.com/platform/schedules#setting-up-a-new-schedule)

Before setting up a new schedule, you should have the [Actor](https://docs.apify.com/platform/actors) or [task](https://docs.apify.com/platform/actors/running/tasks) you want to schedule prepared and tested.

To schedule an Actor, you need to have run it at least once before. To run the Actor, navigate to the Actor's page through [Apify Console](https://console.apify.com/store), where you can configure and initiate the Actor's run with your preferred settings by clicking the **Start** button. After this initial run, you can then use Schedules to automate future runs.

**Name Length**

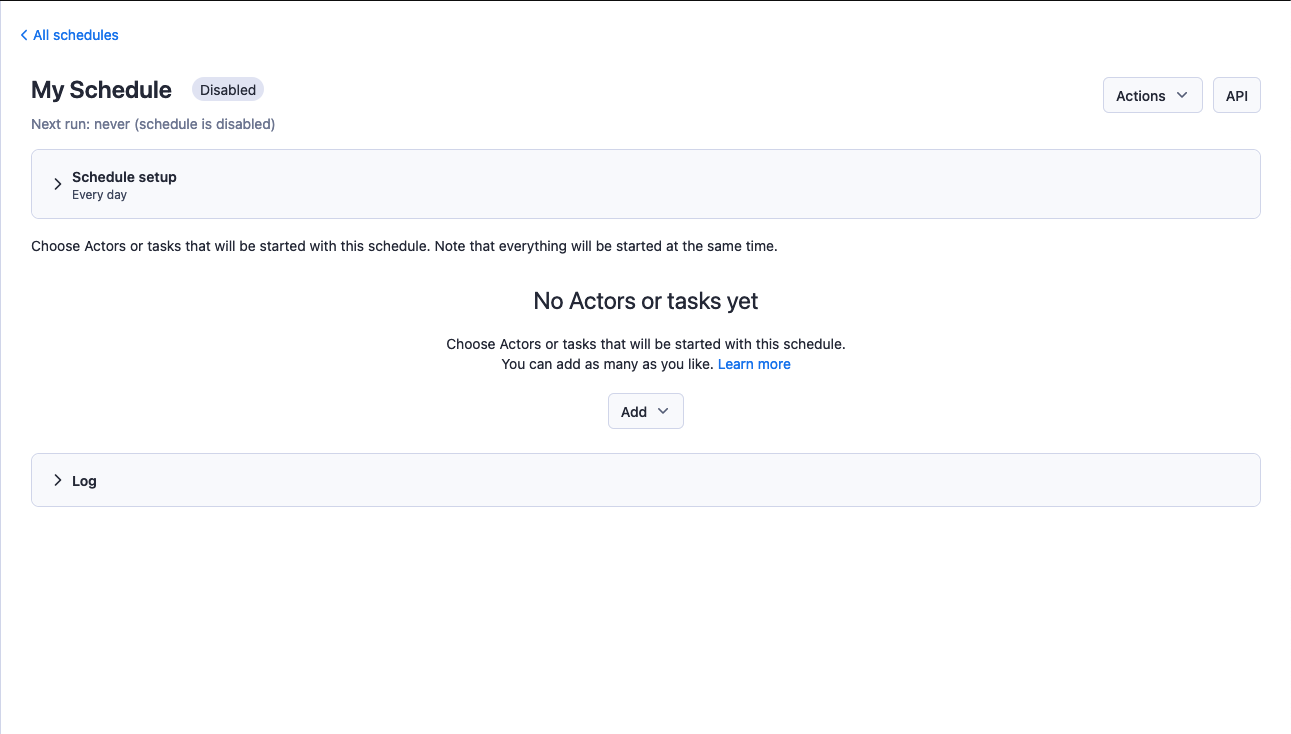
Your schedule's name should be 3–63 characters long.

**Apify Console**[**​**](https://docs.apify.com/platform/schedules#apify-console)

In [Apify Console](https://console.apify.com/schedules), click on the **Schedules** in the navigation menu, then click the **Create new** button.

Click on the name (by default it is **My Schedule**), there you can change its name, add a description, as well as check its *Unique name* or *ID*.

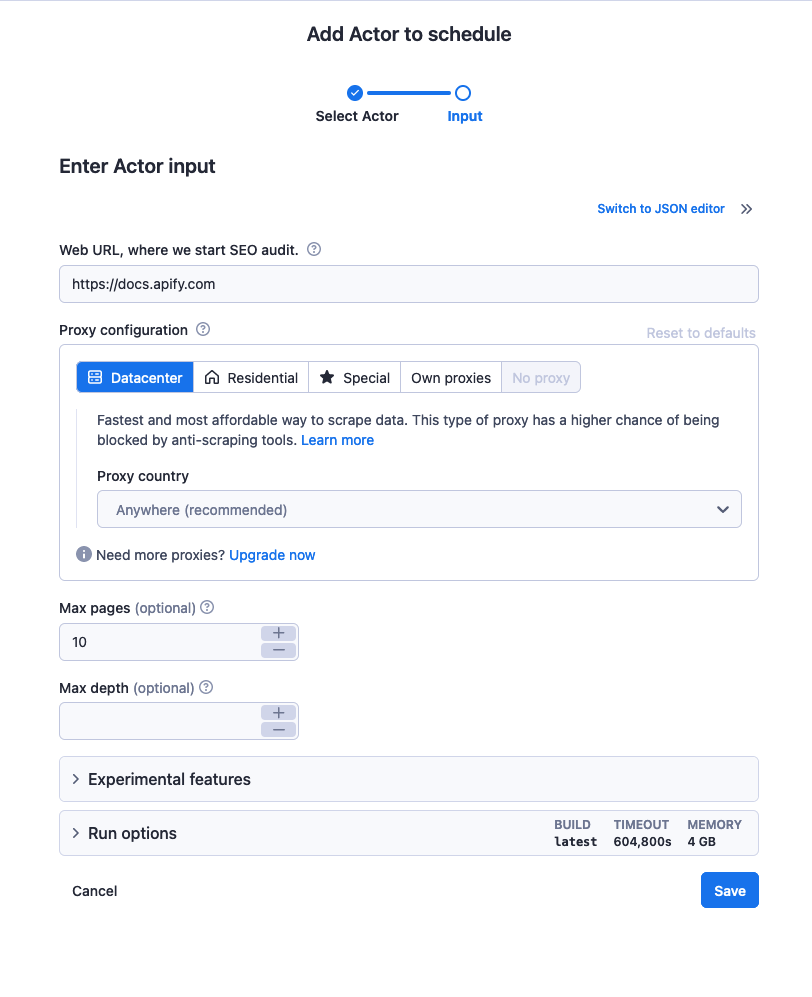
You can adjust how often your Actor or task runs using the [schedule setup tool](https://docs.apify.com/platform/schedules#schedule-setup). You can find it by clicking on the **Schedule setup** card.



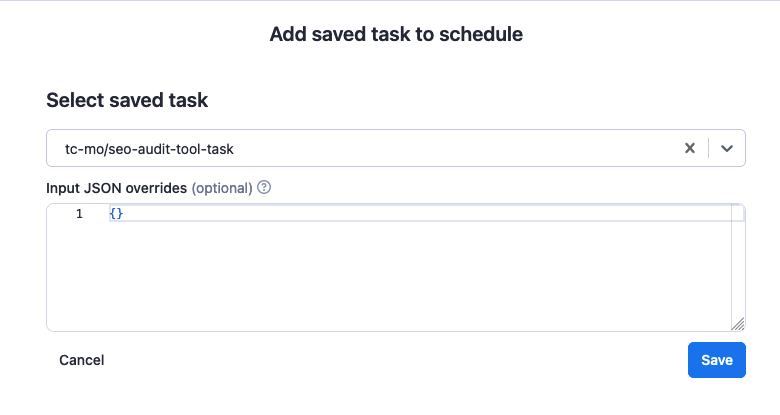
Next, you'll need to give the schedule something to run. This is where the Actor or task you prepared earlier comes in. Click on the **Add** dropdown and select whether you want to schedule an Actor or task.

If you're scheduling an Actor run, you'll be able to specify the Actor's [input](https://docs.apify.com/platform/actors/running/input-and-output) and running options like [build](https://docs.apify.com/platform/actors/development/builds-and-runs/builds), timeout, [memory](https://docs.apify.com/platform/actors/running/usage-and-resources). The **timeout** value is specified in seconds; a value of *0* means there is no timeout, and the Actor runs until it finishes.

If you don't provide an input, then the Actor's default input is used. If you provide an input with some fields missing, the missing fields are filled in with values from the default input. If input options are not provided, the default options values are used.



If you're scheduling a task, just select the task you prepared earlier using the drop-down. If you need to override the task's input, you can pass it as a JSON object in the **Input JSON overrides** field.



To add more Actors or tasks, just repeat the process.

Now, all you need to do is click **Save & activate** and let the scheduler take care of running your jobs on time.

For integrations, you can also add a [webhook](https://docs.apify.com/platform/integrations/webhooks) to your tasks, which will notify you (or perform an action of your choice) every time the task runs.

**Apify API**[**​**](https://docs.apify.com/platform/schedules#apify-api)

To create a new [schedule](https://docs.apify.com/api/v2#/reference/schedules) using the Apify API, send a POST request to the [create schedule](https://docs.apify.com/api/v2#/reference/schedules/schedules-collection/create-schedule) endpoint.

You can find your [secret API token](https://docs.apify.com/platform/integrations) under the [Integrations](https://console.apify.com/account?tab=integrations) tab of your Apify account settings.

**API authentication recommendations**

When providing your API authentication token, we recommend using the request's Authorization header, rather than the URL ([more info](https://docs.apify.com/api/v2#/introduction/authentication)).

In the POST request's payload should be a JSON object specifying the schedule's name, your [user ID](https://console.apify.com/account#/integrations), and the schedule's *actions*.

The following JSON object creates a schedule which runs an SEO audit of the Apify domain once a month.

{  
 "name": "apify-domain-monthly-seo-audit",  
 "userId": "7AxwNO4kCDZxsMHip",  
 "isEnabled": true,  
 "isExclusive": true,  
 "cronExpression": "@monthly",  
 "timezone": "UTC",  
 "description": "A monthly audit of the Apify domain's SEO",  
 "actions": [  
 {  
 "type": "RUN\_ACTOR\_TASK",  
 "actorTaskId": "6rHoK2zjYJkmYhSug",  
 "input": {  
 "startUrl": "https://apify.com"  
 }  
 }  
 ]  
}

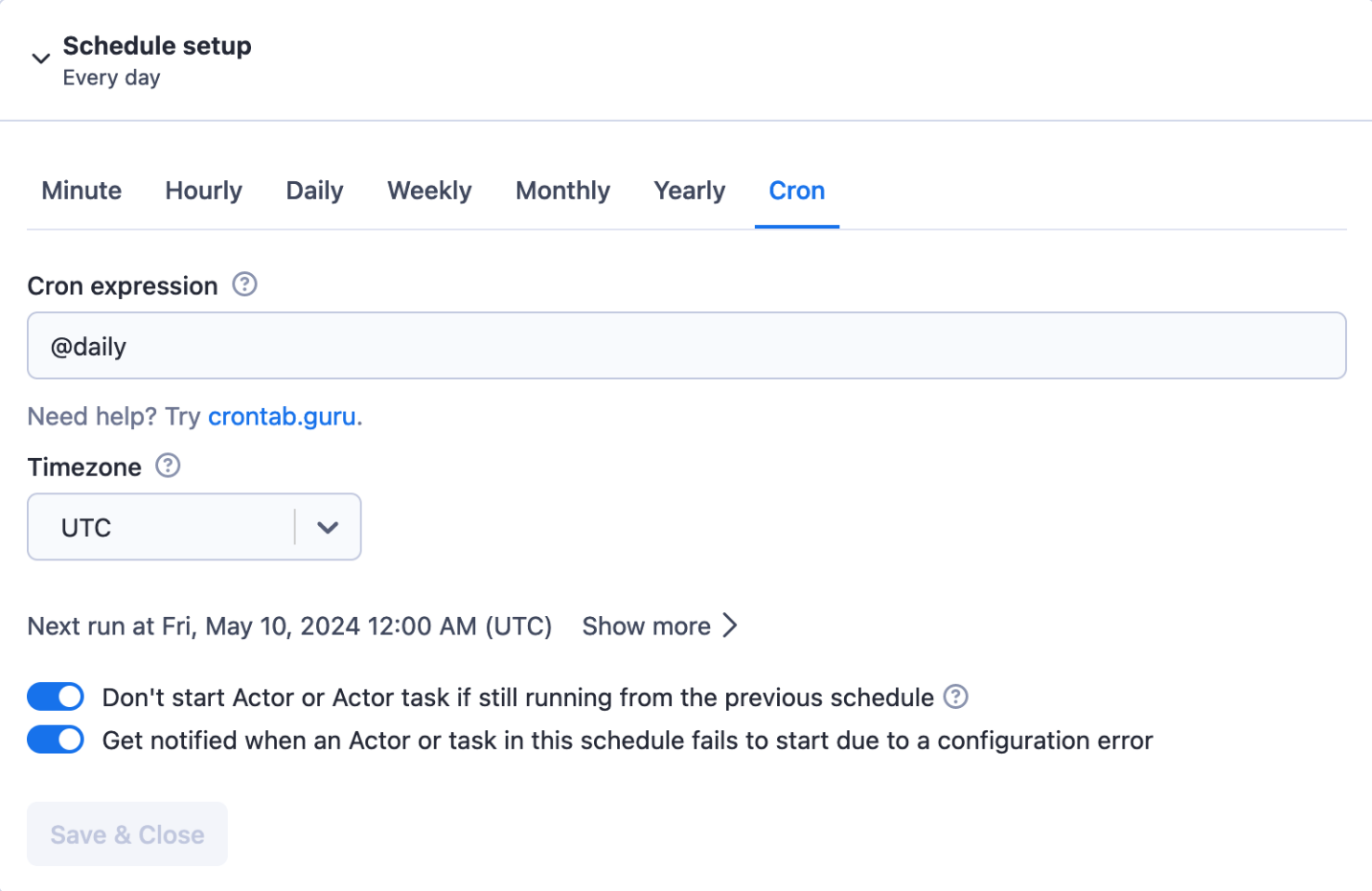
If the request is successful, you will receive a 201 [HTTP response code](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status) and a JSON object in the response body containing the details of your new schedule. If you receive an error (4\*\* code), you will need to check your API token, user ID, or POST request body.

You can add multiple Actor and task runs to a schedule with a single POST request. Simply add another object with the run's details to the **actions** array in your POST request's payload object.

For more information, refer to the [schedules](https://docs.apify.com/api/v2#/reference/schedules/schedule-object/get-schedule) section in our API documentation.

**Schedule setup**[**​**](https://docs.apify.com/platform/schedules#schedule-setup)

The schedule setup tool uses [cron expressions](https://en.wikipedia.org/wiki/Cron#CRON_expression) to specify run times. If you're familiar with how to use them and need a specific run schedule, you can dive right in. If not, don't worry - the setup tool has a visual custom schedule builder that provides a similar level of control as cron expressions, though it's not quite as powerful.



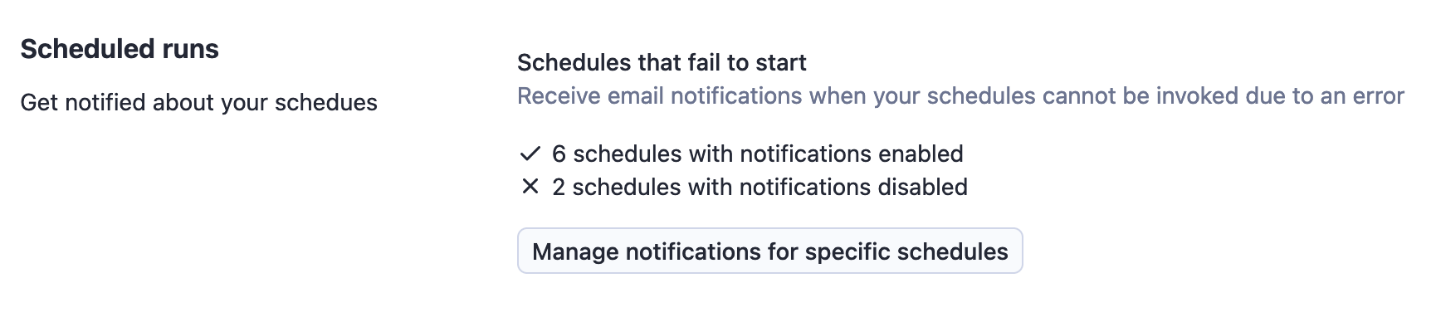
The **Next runs** section shows when the next run will be, if you click on **Show more** button it will expand and show you the next five runs. You can use this live feedback to experiment until you find the correct configuration.

You can find more information and examples of cron expressions on [crontab.guru](https://crontab.guru/). For additional and non-standard characters, see [this](https://en.wikipedia.org/wiki/Cron#CRON_expression) Wikipedia article.

**Notifications**[**​**](https://docs.apify.com/platform/schedules#notifications)

The schedule setup tool allows you to control the schedule's notifications. All schedules have notifications enabled by default. When one of the scheduled Actors or tasks fails to start due to being configured incorrectly, you will receive an email notification.

If you want to manage the notifications for your schedules in bulk, you can do this from the [Notifications](https://console.apify.com/settings/notifications) settings tab. As long as you have 15 schedules or less, you can manage their notifications all at once by clicking the **Manage notifications for specific schedules** button.



**Cron expressions**[**​**](https://docs.apify.com/platform/schedules#cron-expressions)

A cron expression has the following structure:

| **Position** | **Field** | **Values** | **Wildcards** | **Optional** |
| --- | --- | --- | --- | --- |
| 1 | second | 0 - 59 | , - \* / | yes |
| 2 | minute | 0 - 59 | , - \* / | no |
| 3 | hour | 0 - 23 | , - \* / | no |
| 4 | day of month | 1 - 31 | , - \* / | no |
| 5 | month | 1 - 12 | , - \* / | no |
| 6 | day of week | 0 - 7 (0 or 7 is Sunday) | , - \* / | no |

For example, the expression 30 5 16 \* \* 1 will start an Actor at 16:05:30 every Monday.

The minimum interval between runs is 10 seconds; if your next run is scheduled sooner than 10 seconds after the previous run, the next run will be skipped.

**Examples of cron expressions**[**​**](https://docs.apify.com/platform/schedules#examples-of-cron-expressions)

* 0 8 \* \* \* - every day at 8 AM.
* 0 0 \* \* 0 - every 7 days (at 00:00 on Sunday).
* \*/3 \* \* \* \* - every 3rd minute.
* 0 0 1 \*/2 \* - every other month (at 00:00 on the first day of month, every 2nd month).

Additionally, you can use the following shortcut expressions:

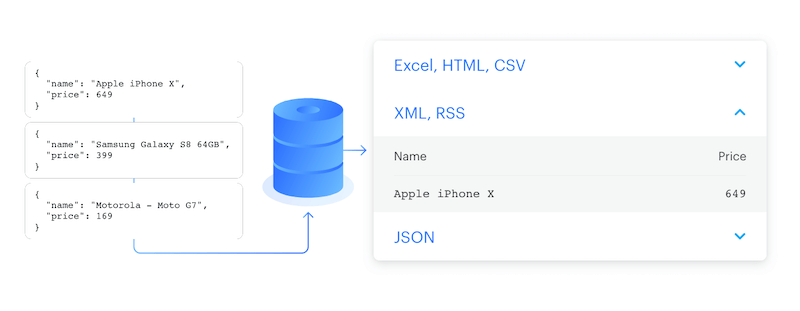
* @yearly = 0 0 1 1 \* - once a year, on Jan 1st at midnight.
* @monthly = 0 0 1 \* \* - once a month, on the 1st at midnight.
* @weekly = 0 0 \* \* 0 - once a week, on Sunday at midnight.
* @daily = 0 0 \* \* \* - run once a day, at midnight.
* @hourly = 0 \* \* \* \* - on the hour, every hour.

**Usage**

**Learn how to effectively use Apify's storage options. Understand key aspects of data retention, rate limiting, and secure sharing.**

**Dataset**[**​**](https://docs.apify.com/platform/storage/usage#dataset)

[Dataset](https://docs.apify.com/platform/storage/dataset) storage allows you to store a series of data objects, such as results from web scraping, crawling, or data processing jobs. You can export your datasets in JSON, CSV, XML, RSS, Excel, or HTML formats.



**Key-value store**[**​**](https://docs.apify.com/platform/storage/usage#key-value-store)

The [key-value store](https://docs.apify.com/platform/storage/key-value-store) is ideal for saving data records such as files, screenshots of web pages, and PDFs or for persisting your Actor's state. The records are accessible under a unique name and can be written and read quickly.

**Request queue**[**​**](https://docs.apify.com/platform/storage/usage#request-queue)

[Request queues](https://docs.apify.com/platform/storage/request-queue) allow you to dynamically maintain a queue of URLs of web pages. You can use this when recursively crawling websites: you start from initial URLs and add new links as they are found while skipping duplicates.

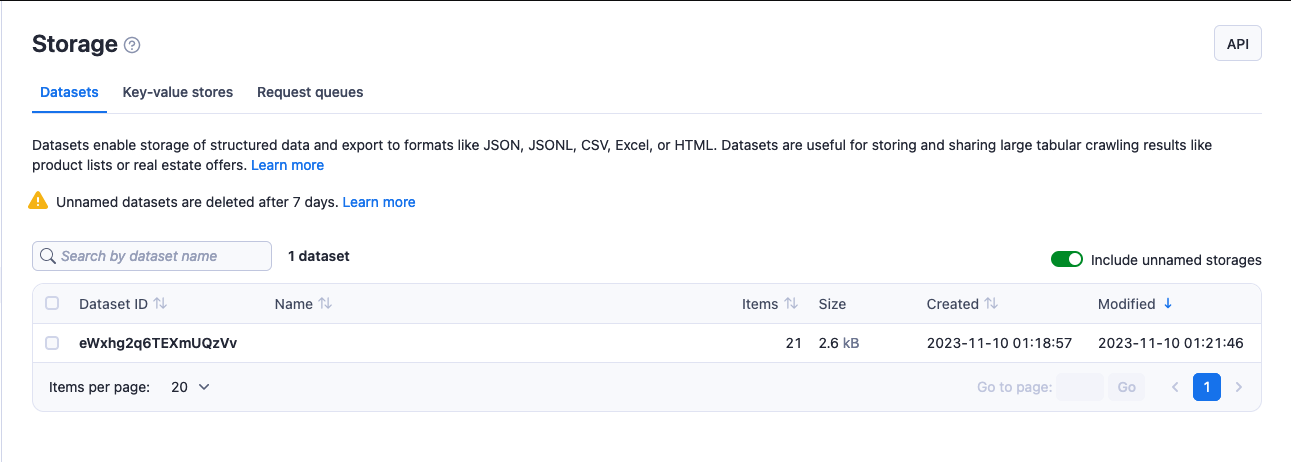
**Basic usage**[**​**](https://docs.apify.com/platform/storage/usage#basic-usage)

You can access your storage in several ways:

* [Apify Console](https://console.apify.com/storage) - provides an easy-to-use interface.
* [Apify API](https://docs.apify.com/api/v2#/reference/key-value-stores) - to access your storages programmatically.
* [API clients](https://docs.apify.com/api) - to access your storages from any Node.js/Python application.
* [Apify SDKs](https://docs.apify.com/sdk) - when building your own JavaScript/Python Actor.

**Apify Console**[**​**](https://docs.apify.com/platform/storage/usage#apify-console)

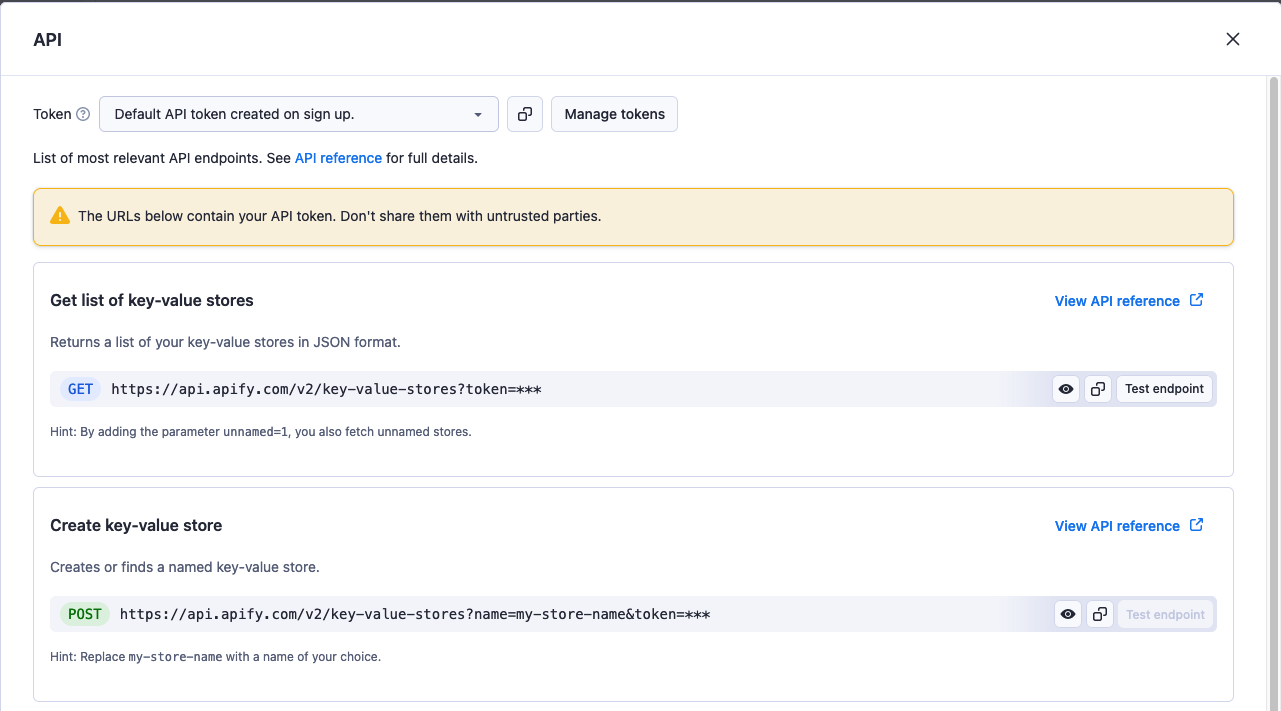
To access your storages via Apify Console, navigate to the [**Storage**](https://console.apify.com/storage) section in the left-side menu. From there, you can click through the tabs to view your key-value stores, datasets, and request queues, and you can click on the **API** button in the top right corner to view related API endpoints. To view a storage, click its **ID**.



Use the **Include unnamed storages** checkbox to either display or hide unnamed storages. By default Apify Console will display them.

You can edit your store's name by clicking on the **Actions** menu and selecting **Rename**.

Additionally, you can quickly share the contents and details of your storage by selecting **Share** under the **Actions** menu and providing either email, username or user ID.



These URLs link to API *endpoints*—the places where your data is stored. Endpoints that allow you to *read* stored information do not require an [authentication token](https://docs.apify.com/api/v2#/introduction/authentication). Calls are authenticated using a hard-to-guess ID, allowing for secure sharing. However, operations such as *update* or *delete* require the authentication token.

Never share a URL containing your authentication token, to avoid compromising your account's security.  
If the data you want to share requires a token, first download the data, then share it as a file.

**Apify API**[**​**](https://docs.apify.com/platform/storage/usage#apify-api)

The [Apify API](https://docs.apify.com/api/v2#/reference/key-value-stores) allows you to access your storages programmatically using [HTTP requests](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods) and easily share your crawling results.

In most cases, when accessing your storages via API, you will need to provide a store ID, which you can do in the following formats:

* WkzbQMuFYuamGv3YF - the store's alphanumerical ID if the store is unnamed.
* ~store-name - the store's name prefixed with tilde (~) character if the store is named (e.g. ~ecommerce-scraping-results)
* username~store-name - username and the store's name separated by a tilde (~) character if the store is named and belongs to a different account (e.g. janedoe~ecommerce-scraping-results). Note that in this case, the store's owner needs to grant you access first.

For read (GET) requests, it is enough to use a store's alphanumerical ID, since the ID is hard to guess and effectively serves as an authentication key.

With other request types and when using the username~store-name, however, you will need to provide your secret API token in your request's [Authorization](https://docs.apify.com/api/v2#/introduction/authentication) header or as a query parameter. You can find your token on the [Integrations](https://console.apify.com/account?tab=integrations) page of your Apify account.

For further details and a breakdown of each storage API endpoint, refer to the [API documentation](https://docs.apify.com/api/v2#/reference/datasets).

**Apify API Clients**[**​**](https://docs.apify.com/platform/storage/usage#apify-api-clients)

The Apify API Clients allow you to access your datasets from any Node.js or Python application, whether it's running on the Apify platform or externally.

You can visit [API Clients](https://docs.apify.com/api) documentations for more information.

**Apify SDKs**[**​**](https://docs.apify.com/platform/storage/usage#apify-sdks)

The Apify SDKs are libraries in JavaScript or Python that provide tools for building your own Actors.

* JavaScript SDK requires [Node.js](https://nodejs.org/en/) 16 or later.
* Python SDK requires [Python](https://www.python.org/downloads/release/python-380/) 3.8 or above.

**Rate limiting**[**​**](https://docs.apify.com/platform/storage/usage#rate-limiting)

All API endpoints limit their rate of requests to protect Apify servers from overloading. The default rate limit for storage objects is *30 requests per second*. However, there are exceptions limited to *200 requests per second* per storage object, including:

* [Push items](https://docs.apify.com/api/v2#/reference/datasets/item-collection/put-items) to dataset.
* CRUD ([add](https://docs.apify.com/api/v2#/reference/request-queues/request-collection/add-request), [get](https://docs.apify.com/api/v2#/reference/request-queues/request-collection/get-request), [update](https://docs.apify.com/api/v2#/reference/request-queues/request-collection/update-request), [delete](https://docs.apify.com/api/v2#/reference/request-queues/request-collection/delete-request)) operations of *request queue* requests.

If a client exceeds this limit, the API endpoints respond with the HTTP status code 429 Too Many Requests and the following body:

{  
 "error": {  
 "type": "rate-limit-exceeded",  
 "message": "You have exceeded the rate limit of ... requests per second"  
 }  
}

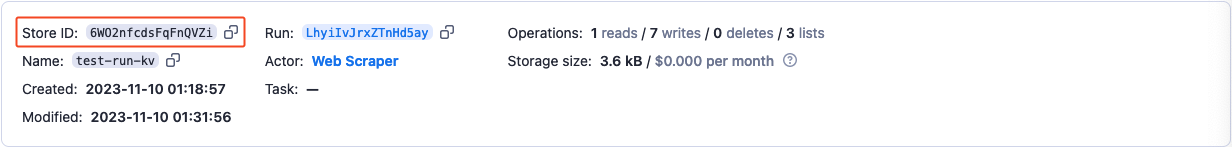
Go to the [API documentation](https://docs.apify.com/api/v2#/introduction/rate-limiting) for details and to learn what to do if you exceed the rate limit.

**Data retention**[**​**](https://docs.apify.com/platform/storage/usage#data-retention)

Apify securely stores your ten most recent runs indefinitely, ensuring your records are always accessible. Unnamed datasets and runs beyond the latest ten will be automatically deleted after 7 days unless otherwise specified. Named datasets are retained indefinitely.

**Preserving your storages**[**​**](https://docs.apify.com/platform/storage/usage#preserving-storages)

To ensure indefinite retention of your storages, assign them a name. This can be done via Apify Console or through our API. First, you'll need your store's ID. You can find it in the details of the run that created it. In Apify Console, head over to your run's details and select the **Dataset**, **Key-value store**, or **Request queue** tab as appropriate. Check that store's details, and you will find its ID among them.



Find and open your storage by clicking the ID, click on the **Actions** menu, choose **Rename**, and enter its new name in the field. Your storage will now be preserved indefinitely.

To name your storage via API, get its ID from the run that generated it using the [Get run](https://docs.apify.com/api/v2#/reference/actor-runs/run-object-and-its-storages/get-run) endpoint. You can then give it a new name using the Update \[storage\] endpoint. For example, [Update dataset](https://docs.apify.com/api/v2#/reference/datasets/dataset/update-dataset).

Our SDKs and clients each have unique naming conventions for storages. For more information check out documentation:

* [SDKs](https://docs.apify.com/sdk)
* [API Clients](https://docs.apify.com/api)

**Named and unnamed storages**[**​**](https://docs.apify.com/platform/storage/usage#named-and-unnamed-storages)

The default storages for an Actor run are unnamed, identified only by an *ID*. This allows them to expire after 7 days (or longer on paid plans) conserving your storage space. If you want to preserve a storage, [assign it a name](https://docs.apify.com/platform/storage/usage#preserving-storages), and it will be retained indefinitely.

Storages' names can be up to 63 characters long.

Named and unnamed storages are identical in all aspects except for their retention period. The key advantage of named storages is their ease in identifying and verifying the correct store.

For example, storage names janedoe~my-storage-1 and janedoe~web-scrape-results are easier to tell apart than the alphanumerical IDs cAbcYOfuXemTPwnIB and CAbcsuZbp7JHzkw1B.

**Sharing**[**​**](https://docs.apify.com/platform/storage/usage#sharing)

You can grant [access rights](https://docs.apify.com/platform/collaboration) to others Apify users to view or modify your storages. Check the [full list of permissions](https://docs.apify.com/platform/collaboration/list-of-permissions).

**Sharing storages between runs**[**​**](https://docs.apify.com/platform/storage/usage#sharing-storages-between-runs)

Storage can be accessed from any [Actor](https://docs.apify.com/platform/actors) or [task](https://docs.apify.com/platform/actors/running/tasks) run, provided you have its *name* or *ID*. You can access and manage storages from other runs using the same methods or endpoints as with storages from your current run.

[Datasets](https://docs.apify.com/platform/storage/dataset) and [key-value stores](https://docs.apify.com/platform/storage/key-value-store) support concurrent use by multiple Actors. Thus, several Actors or tasks can simultaneously write data to a single dataset or key-value store. Similarly, multiple runs can read data from datasets and key-value stores at the same time.

[Request queues](https://docs.apify.com/platform/storage/request-queue), on the other hand, only allow multiple runs to add new data. A request queue can only be processed by one Actor or task run at any one time.

When multiple runs try to write data to a storage simultaneously, the order of data writing cannot be controlled. Data is written as each request is processed.  
Similar principle applies in key-value stores and request queues, when a delete request for a record precedes a read request for the same record, the read request will fail.

**Deleting storages**[**​**](https://docs.apify.com/platform/storage/usage#deleting-storages)

Named storages are only removed upon your request.  
You can delete storages in the following ways:

* [Apify Console](https://console.apify.com/storage) - using the **Actions** button in the store's detail page.
* [JavaScript SDK](https://docs.apify.com/sdk/js) - using the .drop() method of the [Dataset](https://docs.apify.com/sdk/js/api/apify/class/Dataset#drop), [Key-value store](https://docs.apify.com/sdk/js/api/apify/class/KeyValueStore#drop), or [Request queue](https://docs.apify.com/sdk/js/api/apify/class/RequestQueue#drop) class.
* [Python SDK](https://docs.apify.com/sdk/python) - using the .drop() method of the [Dataset](https://docs.apify.com/sdk/python/reference/class/Dataset#drop), [Key-value store](https://docs.apify.com/sdk/python/reference/class/KeyValueStore#drop), or [Request queue](https://docs.apify.com/sdk/python/reference/class/RequestQueue#drop) class.
* [JavaScript API client](https://docs.apify.com/api/client/js) - using the .delete() method in the [dataset](https://docs.apify.com/api/client/js/reference/class/DatasetClient), [key-value store](https://docs.apify.com/api/client/js/reference/class/KeyValueStoreClient), or [request queue](https://docs.apify.com/api/client/js/reference/class/RequestQueueClient) clients.
* [Python API client](https://docs.apify.com/api/client/python) - using the .delete() method in the [dataset](https://docs.apify.com/api/client/python#datasetclient), [key-value store](https://docs.apify.com/api/client/python/reference/class/KeyValueStoreClient), or [request queue](https://docs.apify.com/api/client/python/reference/class/RequestQueueClient) clients.
* [API](https://docs.apify.com/api/v2#/reference/key-value-stores/store-object/delete-store) using the - Delete [store] endpoint, where [store] is the type of storage you want to delete.

**Dataset**

**Store and export web scraping, crawling or data processing job results. Learn how to access and manage datasets in Apify Console or via API.**

Dataset storage enables you to sequentially save and retrieve data. A unique dataset is automatically created and assigned to each Actor run when the first item is stored.

Typically, datasets comprise results from web scraping, crawling, and data processing jobs. You can visualize this data in a table, where each object is forming a row and its attributes are represented as columns. You have the option to export data in various formats, including JSON, CSV, XML, Excel, HTML Table, RSS or JSONL.

Named datasets are retained indefinitely.  
Unnamed datasets expire after 7 days unless otherwise specified.  
> [Learn more](https://docs.apify.com/platform/storage/usage#named-and-unnamed-storages)

Dataset storage is *append-only* - data can only be added and cannot be modified or deleted once stored.

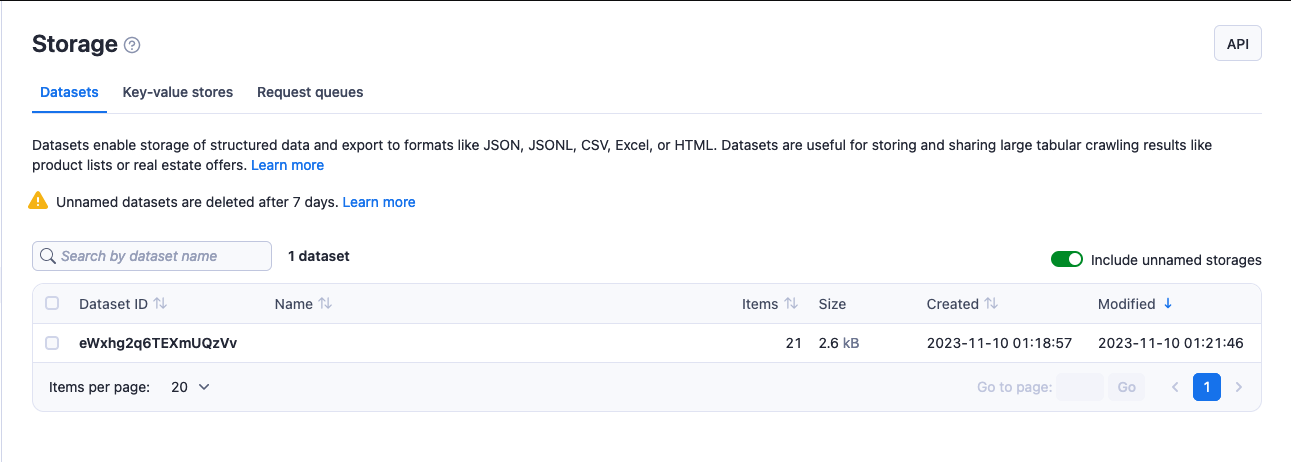
**Basic usage**[**​**](https://docs.apify.com/platform/storage/dataset#basic-usage)

You can access your datasets in several ways:

* [Apify Console](https://console.apify.com/) - provides an easy-to-understand interface.
* [Apify API](https://docs.apify.com/api/v2) - to access your datasets programmatically.
* [Apify API client](https://docs.apify.com/api) - to access your datasets from any Node.js/Python application.
* [Apify SDKs](https://docs.apify.com/sdk) - when building your own JavaScript/Python Actor.

**Apify Console**[**​**](https://docs.apify.com/platform/storage/dataset#apify-console)

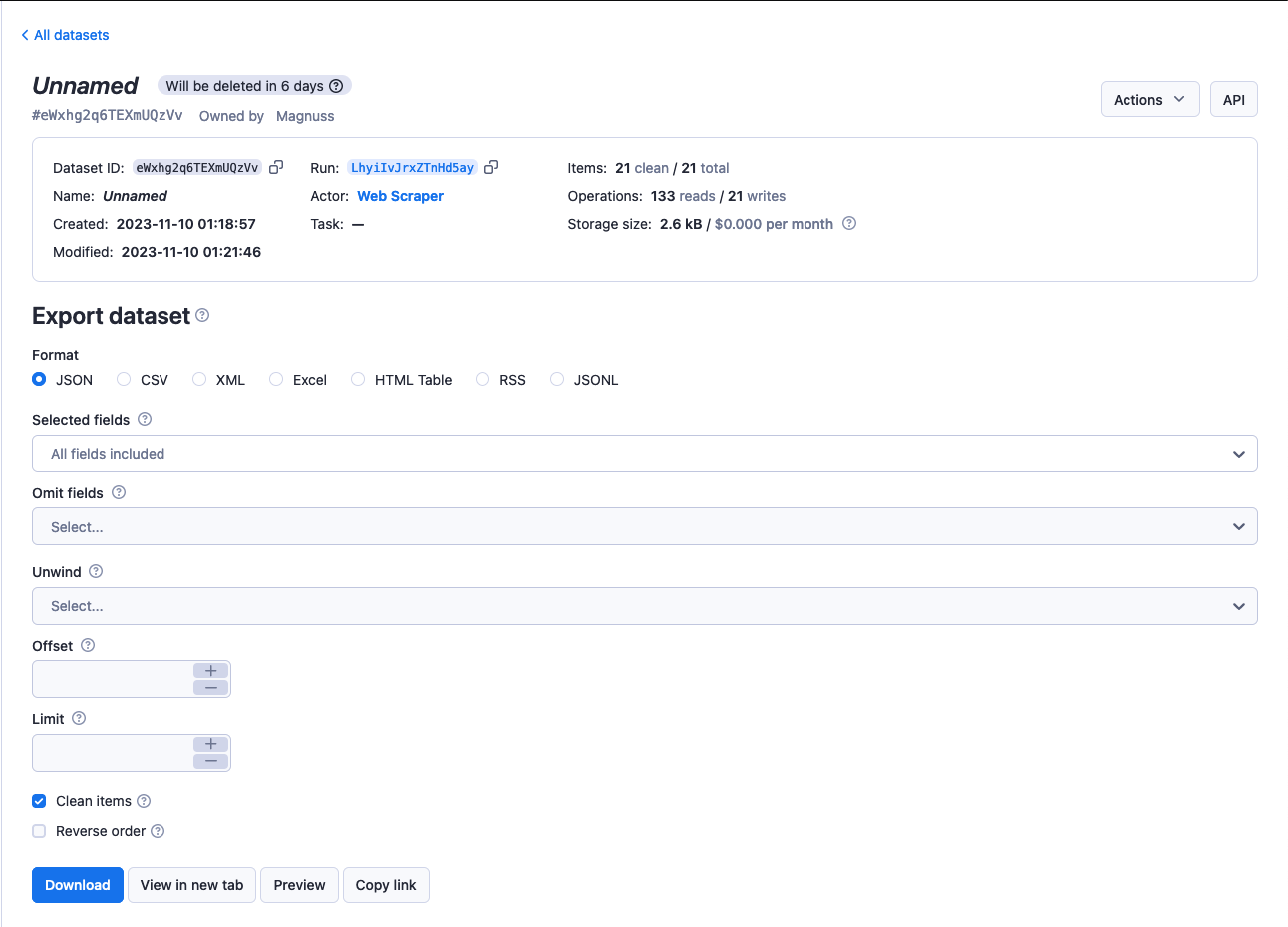
In [Apify Console](https://console.apify.com/), you can view your datasets in the [Storage](https://console.apify.com/storage) section under the [Datasets](https://console.apify.com/storage?tab=datasets) tab.



To view or download a dataset:

1. Click on its **Dataset ID**.
2. Select the format & configure other options if desired in **Export dataset** section.
3. Click **Download**.

Utilize the **Actions** menu to modify the dataset's name, which also affects its [retention period](https://docs.apify.com/platform/storage/usage#data-retention), and to adjust [access rights](https://docs.apify.com/platform/collaboration). The **API** button allows you to explore and test the dataset's [API endpoints](https://docs.apify.com/api/v2#/reference/datasets).



**Apify API**[**​**](https://docs.apify.com/platform/storage/dataset#apify-api)

The [Apify API](https://docs.apify.com/api/v2#/reference/datasets) enables you programmatic access to your datasets using [HTTP requests](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods).

If you are accessing your datasets using the username~store-name [store ID format](https://docs.apify.com/platform/storage), you will need to use your secret API token. You can find the token (and your user ID) on the [Integrations](https://console.apify.com/account#/integrations)tab of **Settings** page of your Apify account.

When providing your API authentication token, we recommend using the request's Authorization header, rather than the URL. ([More info](https://docs.apify.com/platform/integrations/api#authentication)).

To retrieve a list of your datasets, send a GET request to the [Get list of datasets](https://docs.apify.com/api/v2#/reference/datasets/dataset-collection/get-list-of-datasets) endpoint.

https://api.apify.com/v2/datasets

To get information about a dataset such as its creation time and item count, send a GET request to the [Get dataset](https://docs.apify.com/api/v2#/reference/datasets/dataset/get-dataset) endpoint.

https://api.apify.com/v2/datasets/{DATASET\_ID}

To view a dataset's data, send a GET request to the [Get dataset items](https://docs.apify.com/api/v2#/reference/datasets/item-collection/get-items) Apify API endpoint.

https://api.apify.com/v2/datasets/{DATASET\_ID}/items

Control the data export by appending a comma-separated list of fields to the fields query parameter. Likewise, you can also omit certain fields using the omit parameter.

If you fill both omit and field parameters with the same value, then >omit parameter will take precedence and the field is excluded from the >results.

In addition, you can set the format in which you retrieve the data using the ?format= parameter. The available formats are json, jsonl, csv, html, xlsx, xml and rss. The default value is json.

To retrieve the hotel and cafe fields, you would send your GET request to the URL below.

https://api.apify.com/v2/datasets/{DATASET\_ID}/items?format=json&fields=hotel%2Ccafe

Use %2C instead of commas for URL encoding, as %2C represent a comma. For more on URL encoding check out [this page](https://www.url-encode-decode.com/)

To add data to a dataset, issue a POST request to the [Put items](https://docs.apify.com/api/v2#/reference/datasets/item-collection/put-items) endpoint with the data as a JSON object payload.

https://api.apify.com/v2/datasets/{DATASET\_ID}/items

API data push to a dataset is capped at *200 requests per second* to avoid overloading our servers.

Example payload:

[  
 {  
 "foo": "bar"  
 },  
 {  
 "foo": "hotel"  
 },  
 {  
 "foo": "cafe"  
 }  
]

For further details and a breakdown of each storage API endpoint, refer to the [API documentation](https://docs.apify.com/api/v2#/reference/datasets).

**Apify API Clients**[**​**](https://docs.apify.com/platform/storage/dataset#apify-api-clients)

**JavaScript API client**[**​**](https://docs.apify.com/platform/storage/dataset#javascript-api-client)

The [JavaScript API client](https://docs.apify.com/api/client/js/reference/class/DatasetClient) (apify-client) enables you access to your datasets from any Node.js application, whether hosted on the Apify platform or externally.

After importing and initiating the client, you can save each dataset to a variable for easier access.

const myDatasetClient = apifyClient.dataset('jane-doe/my-dataset');

You can then use that variable to [access the dataset's items and manage it](https://docs.apify.com/api/client/js/reference/class/DatasetClient).

When using the [.listItems()](https://docs.apify.com/api/client/js/reference/class/DatasetClient#listItems) method, if you fill both omit and field parameters with the same value, then omit parameter will take precedence and the field is excluded from the results.

Check out the [JavaScript API client documentation](https://docs.apify.com/api/client/js/reference/class/DatasetClient) for [help with setup](https://docs.apify.com/api/client/js/docs) and more details.

**Python API client**[**​**](https://docs.apify.com/platform/storage/dataset#python-api-client)

The [Python API client](https://docs.apify.com/api/client/python/reference/class/DatasetClient) (apify-client) enables you access to your datasets from any Python application, whether it is running on the Apify platform or externally.

After importing and initiating the client, you can save each dataset to a variable for easier access.

my\_dataset\_client = apify\_client.dataset('jane-doe/my-dataset')

You can then use that variable to [access the dataset's items and manage it](https://docs.apify.com/api/client/python/reference/class/DatasetClient).

When using the [.list\_items()](https://docs.apify.com/api/client/python/reference/class/DatasetClient#list_items) method, if you fill both omit and field parameters with the same value, then omit parameter will take precedence and the field is excluded from the results.

Check out the [Python API client documentation](https://docs.apify.com/api/client/python/reference/class/DatasetClient) for [help with setup](https://docs.apify.com/api/client/python/docs/overview/introduction) and more details.

**Apify SDKs**[**​**](https://docs.apify.com/platform/storage/dataset#apify-sdks)

**JavaScript SDK**[**​**](https://docs.apify.com/platform/storage/dataset#javascript-sdk)

When working with a JavaScript [Actor](https://docs.apify.com/platform/actors), the [JavaScript SDK](https://docs.apify.com/sdk/js/docs/guides/result-storage#dataset) is an essential tool, especially for dataset management. It simplifies the tasks of storing and retrieving data, seamlessly integrating with the Actor's workflow. Key features of the SDK include the ability to append data, retrieve what is stored, and manage dataset properties effectively. Central to this functionality is the [Dataset](https://docs.apify.com/sdk/js/reference/class/Dataset) class. This class allows you to determine where your data is stored - locally or in the Apify cloud. To add data to your chosen datasets, use the [pushData()](https://docs.apify.com/sdk/js/reference/class/Dataset#pushData) method.

Additionally the SDK offers other methods like [getData()](https://docs.apify.com/sdk/js/reference/class/Dataset#getData), [map()](https://docs.apify.com/sdk/js/reference/class/Dataset#map), and [reduce()](https://docs.apify.com/sdk/js/reference/class/Dataset#reduce). For practical applications of these methods, refer to the [example](https://docs.apify.com/sdk/js/docs/examples/map-and-reduce) section.

If you have chosen to store your dataset locally, you can find it in the location below.

{APIFY\_LOCAL\_STORAGE\_DIR}/datasets/{DATASET\_ID}/{INDEX}.json

DATASET\_ID refers to the dataset's *name* or *ID*. The default dataset will be stored in the *default* directory.

To add data to the default dataset, you can use the example below:

*// Import the JavaScript SDK into your project*  
import { Actor } from 'apify';  
  
await Actor.init();  
*// ...*  
  
*// Add one item to the default dataset*  
await Actor.pushData({ foo: 'bar' });  
  
*// Add multiple items to the default dataset*  
await Actor.pushData([{ foo: 'hotel' }, { foo: 'cafe' }]);  
  
*// ...*  
await Actor.exit();

It's crucial to use the await keyword when calling pushData(), to ensure data storage completes before the Actor process terminates.

If you want to use something other than the default dataset, e.g. a dataset that you share between Actors or between Actor runs, you can use the [Actor.openDataset()](https://docs.apify.com/sdk/js/reference/class/Actor#openDataset) method.

import { Actor } from 'apify';  
  
await Actor.init();  
*// ...*  
  
*// Save a named dataset to a variable*  
const dataset = await Actor.openDataset('some-name');  
  
*// Add data to the named dataset*  
await dataset.pushData({ foo: 'bar' });  
  
*// ...*  
await Actor.exit();

Utilize the fields option in the [getData()](https://docs.apify.com/sdk/js/reference/class/Dataset#getData) method to specify which data fields to retrieve. This option accepts an array of fields names (string) to include in your results.

import { Actor } from 'apify';  
  
await Actor.init();  
*// ...*  
  
const dataset = await Actor.openDataset();  
  
*// Only get the 'hotel' and 'cafe' fields*  
const hotelAndCafeData = await dataset.getData({  
 fields: ['hotel', 'cafe'],  
});  
  
*// ...*  
await Actor.exit();

Check out the [JavaScript SDK documentation](https://docs.apify.com/sdk/js/docs/guides/result-storage#dataset) and the Dataset class's [API reference](https://docs.apify.com/sdk/js/reference/class/Dataset) for details on managing datasets with the JavaScript SDK.

**Python SDK**[**​**](https://docs.apify.com/platform/storage/dataset#python-sdk)

For Python [Actors](https://docs.apify.com/platform/actors), the [Python SDK](https://docs.apify.com/sdk/python/docs/concepts/storages#working-with-datasets) is essential. The dataset is represented by a [Dataset](https://docs.apify.com/sdk/python/reference/class/Dataset) class. You can use this class to specify whether your data is stored locally or in the Apify cloud and push data to the datasets of your choice using the [push\_data()](https://docs.apify.com/sdk/python/reference/class/Dataset#push_data) method. For further data manipulation you could also use other methods such as [get\_data()](https://docs.apify.com/sdk/python/reference/class/Dataset#get_data), [map()](https://docs.apify.com/sdk/python/reference/class/Dataset#map) and [reduce()](https://docs.apify.com/sdk/python/reference/class/Dataset#reduce).

For datasets stored locally, the data is located at the following path:

{APIFY\_LOCAL\_STORAGE\_DIR}/datasets/{DATASET\_ID}/{INDEX}.json

The DATASET\_ID refers to the dataset's *name* or *ID*. The default dataset will be stored in the *default* directory.

To add data to the default dataset, you can use the example below:

from apify import Actor  
  
async def main():  
 async with Actor:  
 *# Add one item to the default dataset*  
 await Actor.push\_data({'foo': 'bar'})  
  
 *# Add multiple items to the default dataset*  
 await Actor.push\_data([{'foo': 'hotel'}, {'foo': 'cafe'}])

If you want to use something other than the default dataset, e.g. a dataset that you share between Actors or between Actor runs, you can use the [Actor.open\_dataset()](https://docs.apify.com/sdk/python/reference/class/Actor#open_dataset) method.

from apify import Actor  
  
async def main():  
 async with Actor:  
 *# Save a named dataset to a variable*  
 dataset = await Actor.open\_dataset(name='some-name')  
  
 *# Add data to the named dataset*  
 await dataset.push\_data({'foo': 'bar'})

Utilize the fields option in the [get\_data()](https://docs.apify.com/sdk/python/reference/class/Dataset#get_data) method to specify which data fields to retrieve. This option accepts an array of fields names (string) to include in your results.

from apify import Actor  
  
async def main():  
 async with Actor:  
 dataset = await Actor.open\_dataset()  
  
 *# Only get the 'hotel' and 'cafe' fields*  
 hotel\_and\_cafe\_data = await dataset.get\_data(fields=['hotel', 'cafe'])

For more information, visit our [Python SDK documentation](https://docs.apify.com/sdk/python/docs/concepts/storages#working-with-datasets) and the Dataset class's [API reference](https://docs.apify.com/sdk/python/reference/class/Dataset) for details on managing datasets with the Python SDK.

**Hidden fields**[**​**](https://docs.apify.com/platform/storage/dataset#hidden-fields)

Fields in a dataset that begin with a # are treated as hidden. You can exclude these fields when downloading data by using either skipHidden=1 or clean=1 in your query parameters. This feature is useful for excluding debug information from the final dataset output.

The following example demonstrates a dataset record with hidden fields, including HTTP response and error details.

{  
 "url": "https://example.com",  
 "title": "Example page",  
 "data": {  
 "foo": "bar"  
 },  
 "#error": null,  
 "#response": {  
 "statusCode": 201  
 }  
}

Data excluding hidden fields, termed as "clean" data, can be downloaded from the [Apify Console](https://console.apify.com/storage?tab=datasets) using the **Clean items** option. Alternatively, you can download it via API by applying clean=true or clean=1 as [URL parameters](https://docs.apify.com/api/v2#/reference/datasets/item-collection/get-items).

**XML format extension**[**​**](https://docs.apify.com/platform/storage/dataset#xml-format-extension)

In XML and RSS export formats, object property name are converted into XML tags, and their corresponding values are represented as children of these tags.

For example, the JavaScript object:

{  
 name: 'Rashida Jones',  
 address: [  
 {  
 type: 'home',  
 street: '21st',  
 city: 'Chicago',  
 },  
 {  
 type: 'office',  
 street: null,  
 city: null,  
 },  
 ],  
}

becomes the following XML snippet:

<name>Rashida Jones</name>  
<address>  
 <type>home</type>  
 <street>21st</street>  
 <city>Chicago</city>  
</address>  
<address>  
 <type>office</type>  
 <street/>  
 <city/>  
</address>

In a JavaScript object, if a property is named @, its sub-properties are exported as attributes of the corresponding parent XML element. Additionally, when the parent XML element lacks child elements, its value is sourced from a property named # in the JavaScript Object.

For example, the following JavaScript object:

{  
 address: [  
 {  
 '@': {  
 type: 'home',  
 },  
 street: '21st',  
 city: 'Chicago',  
 },  
 {  
 '@': {  
 type: 'office',  
 },  
 '#': 'unknown',  
 },  
 ],  
}

will be transformed to the following XML snippet:

<address *type*="home">  
 <street>21st</street>  
 <city>Chicago</city>  
</address>  
<address *type*="office">unknown</address>

This feature is also useful when customizing your RSS feeds generated for various websites.

By default, the whole result is wrapped in an <items/> element, while each page object is contained in an <item/> element. You can change this using the xmlRoot and xmlRow URL parameters when retrieving your data with a GET request.

**Sharing**[**​**](https://docs.apify.com/platform/storage/dataset#sharing)

You can grant [access rights](https://docs.apify.com/platform/collaboration) to your dataset through the **Share** button under the **Actions** menu. For more details, check the [full list of permissions](https://docs.apify.com/platform/collaboration/list-of-permissions).

**Sharing datasets between runs**[**​**](https://docs.apify.com/platform/storage/dataset#sharing-datasets-between-runs)

You can access a dataset from any [Actor](https://docs.apify.com/platform/actors) or [task](https://docs.apify.com/platform/actors/running/tasks) run as long as you know its *name* or *ID*.

To access a dataset from another run using the [Apify SDK](https://docs.apify.com/sdk), open it using the same method as you would with any other dataset.

* **JavaScript**
* **Python**

import { Actor } from 'apify';  
  
await Actor.init();  
  
const otherDataset = await Actor.openDataset('old-dataset');  
*// ...*  
  
await Actor.exit();

In the [JavaScript API client](https://docs.apify.com/api/client/js/reference/class/DatasetClient) as well as in [Python API client](https://docs.apify.com/api/client/python/reference/class/DatasetClient) , you can access a dataset using its client. Once you've opened the dataset, you can read its contents and add new data in the same manner as you would for a dataset from your current run.

* **JavaScript**
* **Python**

const otherDatasetClient = apifyClient.dataset('jane-doe/old-dataset');

The same applies for the [Apify API](https://docs.apify.com/platform/storage/dataset#apify-api) - you can use [the same endpoints](https://docs.apify.com/platform/storage/dataset#apify-api) as you would normally do.

See the [Storage overview](https://docs.apify.com/platform/storage/usage#sharing-storages-between-runs) for details on sharing storages between runs.

**Limits**[**​**](https://docs.apify.com/platform/storage/dataset#limits)

* Data storage formats that use tabulation (like HTML, CSV, and EXCEL) are limited to a maximum of *3000* columns. Data exceeding this limit will not be retrieved.
* The pushData()method is constrained by the receiving API's size limit. It accepts objects with JSON size under *9MB*. While individual objects within an array must not exceed *9MB*, the overall size has no restriction.
* The maximum length for dataset names is 63 characters.

**Rate limiting**[**​**](https://docs.apify.com/platform/storage/dataset#rate-limiting)

The rate limit for pushing data to a dataset through the [API](https://docs.apify.com/api/v2#/reference/datasets/item-collection/put-items) is capped at *200 requests per second* for each dataset, a measure to prevent overloading Apify servers.

**Key-value store**

**Store anything from Actor or task run results, JSON documents, or images. Learn how to access and manage key-value stores from Apify Console or via API.**

The key-value store is simple storage that can be used for storing any kind of data. It can be JSON or HTML documents, zip files, images, or strings. The data are stored along with their [MIME content type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types).

Each Actor run is assigned its own key-value store when it is created. The store contains the Actor's input, and, if necessary, other data such as its output.

Key-value stores are mutable–you can both add entries and delete them.

Named key-value stores are retained indefinitely.  
Unnamed key-value stores expire after 7 days unless otherwise specified.  
> [Learn more](https://docs.apify.com/platform/storage/usage#named-and-unnamed-storages)

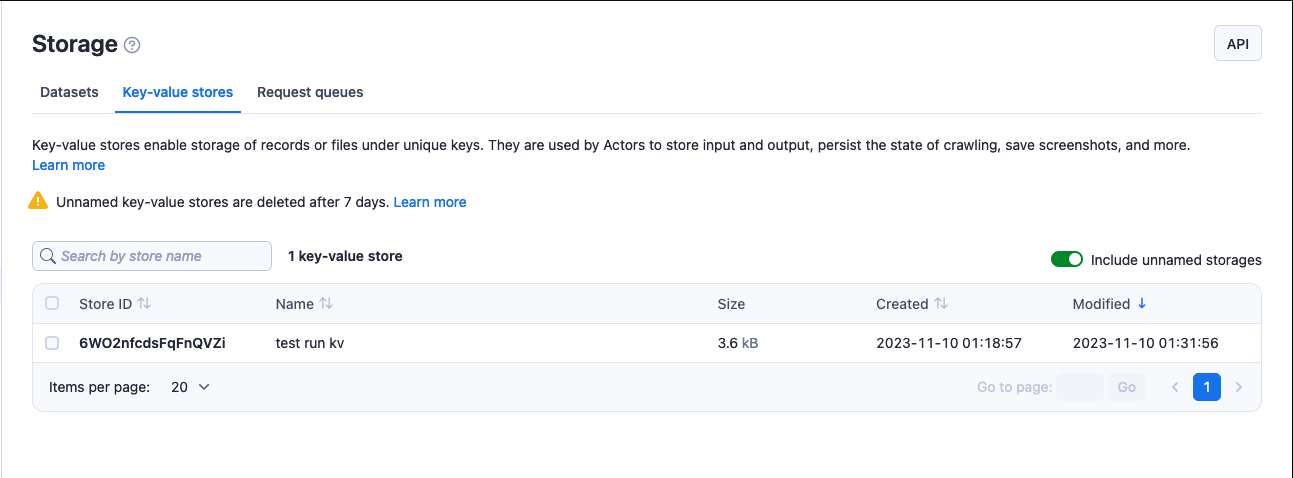
**Basic usage**[**​**](https://docs.apify.com/platform/storage/key-value-store#basic-usage)

You can access key-value stores through several methods

* [Apify Console](https://console.apify.com/) - provides an easy-to-understand interface.
* [Apify API](https://docs.apify.com/api/v2) - for accessing your key-value stores programmatically.
* [Apify API clients](https://docs.apify.com/api) - to access your key-value stores from any Node.js/Python application.
* [Apify SDKs](https://docs.apify.com/sdk) - when building your own JavaScript/Python Actor.

**Apify Console**[**​**](https://docs.apify.com/platform/storage/key-value-store#apify-console)

In [Apify Console](https://console.apify.com/), you can view your key-value stores in the [Storage](https://console.apify.com/storage) section under the [Key-value stores](https://console.apify.com/storage?tab=keyValueStores) tab.



To view a key-value store's content, click on its **Store ID**. Under the **Actions** menu, you can rename your store (and, in turn extend its [retention period](https://docs.apify.com/platform/storage/usage#named-and-unnamed-storages)) and grant [access rights](https://docs.apify.com/platform/collaboration) using the **Share** button. Click on the **API** button to view and test a store's [API endpoints](https://docs.apify.com/api/v2#/reference/key-value-stores).



**Apify API**[**​**](https://docs.apify.com/platform/storage/key-value-store#apify-api)

The [Apify API](https://docs.apify.com/api/v2#/reference/key-value-stores) enables you programmatic access to your key-value stores using [HTTP requests](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods).

If you are accessing your datasets using the username~store-name [store ID format](https://docs.apify.com/platform/storage), you will need to use your secret API token. You can find the token (and your user ID) on the [Integrations](https://console.apify.com/account#/integrations) tab of **Settings** page of your Apify account.

When providing your API authentication token, we recommend using the request's Authorization header, rather than the URL. ([More info](https://docs.apify.com/platform/integrations/api#authentication)).

To retrieve a list of your key-value stores, send a GET request to the [Get list of key-value stores](https://docs.apify.com/api/v2#/reference/key-value-stores/store-collection/get-list-of-key-value-stores) endpoint.

https://api.apify.com/v2/key-value-stores

To get information about a key-value store such as its creation time and item count, send a GET request to the [Get store](https://docs.apify.com/api/v2#/reference/key-value-stores/store-object/get-store) endpoint.

https://api.apify.com/v2/key-value-stores/{STORE\_ID}

To get a record (its value) from a key-value store, send a GET request to the [Get record](https://docs.apify.com/api/v2#/reference/key-value-stores/key-collection/get-record) endpoint.

https://api.apify.com/v2/key-value-stores/{STORE\_ID}/records/{KEY\_ID}

To add a record with a specific key in a key-value store, send a PUT request to the [Put record](https://docs.apify.com/api/v2#/reference/key-value-stores/record/put-record) endpoint.

https://api.apify.com/v2/key-value-stores/{STORE\_ID}/records/{KEY\_ID}

Example payload:

{  
 "foo": "bar",  
 "fos": "baz"  
}

To delete a record, send a DELETE request specifying the key from a key-value store to the [Delete record](https://docs.apify.com/api/v2#/reference/key-value-stores/record/delete-record) endpoint.

https://api.apify.com/v2/key-value-stores/{STORE\_ID}/records/{KEY\_ID}

For further details and a breakdown of each storage API endpoint, refer to the [API documentation](https://docs.apify.com/api/v2#/reference/key-value-stores).

**Apify API Clients**[**​**](https://docs.apify.com/platform/storage/key-value-store#apify-api-clients)

**JavaScript API client**[**​**](https://docs.apify.com/platform/storage/key-value-store#javascript-api-client)

The Apify [JavaScript API client](https://docs.apify.com/api/client/js/reference/class/KeyValueStoreClient) (apify-client) enables you to access your key-value stores from any Node.js application, whether hosted on the Apify platform or externally.

After importing and initiating the client, you can save each key-value store to a variable for easier access.

const myKeyValStoreClient = apifyClient.keyValueStore(  
 'jane-doe/my-key-val-store',  
);

You can then use that variable to [access the key-value store's items and manage it](https://docs.apify.com/api/client/js/reference/class/KeyValueStoreClient).

Check out the [JavaScript API client documentation](https://docs.apify.com/api/client/js/reference/class/KeyValueStoreClient) for [help with setup](https://docs.apify.com/api/client/js/docs) and more details.

**Python API client**[**​**](https://docs.apify.com/platform/storage/key-value-store#python-api-client)

The Apify [Python API client](https://docs.apify.com/api/client/python/reference/class/KeyValueStoreClient) (apify-client) allows you to access your key-value stores from any Python application, whether it is running on the Apify platform or externally.

After importing and initiating the client, you can save each key-value store to a variable for easier access.

my\_key\_val\_store\_client = apify\_client.key\_value\_store('jane-doe/my-key-val-store')

You can then use that variable to [access the key-value store's items and manage it](https://docs.apify.com/api/client/python/reference/class/KeyValueStoreClient).

Check out the [Python API client documentation](https://docs.apify.com/api/client/python/reference/class/KeyValueStoreClient) for [help with setup](https://docs.apify.com/api/client/python/docs/overview/introduction) and more details.

**Apify SDKs**[**​**](https://docs.apify.com/platform/storage/key-value-store#apify-sdks)

**JavaScript SDK**[**​**](https://docs.apify.com/platform/storage/key-value-store#javascript-sdk)

When working with a JavaScript [Actor](https://docs.apify.com/platform/actors), the [JavaScript SDK](https://docs.apify.com/sdk/js/docs/guides/result-storage#key-value-store) is an essential tool, especially for key-value store management. The primary class for this purpose is the [KeyValueStore](https://docs.apify.com/sdk/js/reference/class/KeyValueStore). This class allows you to decide whether your data will be stored locally or in the Apify cloud. For data manipulation, it offers the [getValue()](https://docs.apify.com/sdk/js/reference/class/KeyValueStore#getValue) and [setValue()](https://docs.apify.com/sdk/js/reference/class/KeyValueStore#setValue) methods to retrieve and assign values, respectively.

Additionally, you can iterate over the keys in your store using the [forEachKey()](https://docs.apify.com/sdk/js/reference/class/KeyValueStore#forEachKey) method.

Every Actor run is linked to a default key-value store that is automatically created for that specific run. If you're running your Actors and opt to store data locally, you can easily supply the [input](https://docs.apify.com/platform/actors/running/input-and-output) by placing an *INPUT.json* file in the corresponding directory of the default key-value store. This method ensures that you Actor has all the necessary data readily available for its execution.

You can find *INPUT.json* and other key-value store files in the location below.

{APIFY\_LOCAL\_STORAGE\_DIR}/key\_value\_stores/{STORE\_ID}/{KEY}.{EXT}

The default key-value store's ID is *default*. The {KEY} is the record's *key* and {EXT} corresponds to the record value's MIME content type.

To manage your key-value stores, you can use the following methods. See the KeyValueStore class's [API reference](https://docs.apify.com/sdk/js/reference/class/KeyValueStore) for the full list.

import { Actor } from 'apify';  
  
await Actor.init();  
*// ...*  
  
*// Get the default input*  
const input = await Actor.getInput();  
  
*// Open a named key-value store*  
const exampleStore = await Actor.openKeyValueStore('my-store');  
  
*// Read a record in the exampleStore storage*  
const value = await exampleStore.getValue('some-key');  
  
*// Write a record to exampleStore*  
await exampleStore.setValue('some-key', { foo: 'bar' });  
  
*// Delete a record from exampleStore*  
await exampleStore.setValue('some-key', null);  
  
*// ...*  
await Actor.exit();

Note that JSON is automatically parsed to a JavaScript object, text data returned as a string and other data is returned as binary buffer.

import { Actor } from 'apify';  
  
await Actor.init();  
*// ...*  
  
*// Get input of your Actor*  
const input = await Actor.getInput();  
const value = await Actor.getValue('my-key');  
  
*// ...*  
await Actor.setValue('OUTPUT', imageBuffer, { contentType: 'image/jpeg' });  
  
*// ...*  
await Actor.exit();

The Actor.getInput() method is not only a shortcut to Actor.getValue('INPUT'); it is also compatible with [Actor.metamorph()](https://docs.apify.com/platform/actors/development/programming-interface/metamorph). This is because a metamorphed Actor run's input is stored in the *INPUT-METAMORPH-1* key instead of *INPUT*, which hosts the original input.

Check out the [JavaScript SDK documentation](https://docs.apify.com/sdk/js/docs/guides/result-storage#key-value-store) and the KeyValueStore class's [API reference](https://docs.apify.com/sdk/js/reference/class/KeyValueStore) for details on managing your key-value stores with the JavaScript SDK.

**Python SDK**[**​**](https://docs.apify.com/platform/storage/key-value-store#python-sdk)

For Python [Actor](https://docs.apify.com/platform/actors), the [Python SDK](https://docs.apify.com/sdk/python/docs/concepts/storages#working-with-key-value-stores) is essential. The key-value store is represented by a [KeyValueStore](https://docs.apify.com/sdk/python/reference/class/KeyValueStore) class. You can use this class to specify whether your data is stored locally or in the Apify cloud. For further data manipulation it offers [get\_value()](https://docs.apify.com/sdk/python/reference/class/KeyValueStore#get_value) and [set\_value()](https://docs.apify.com/sdk/python/reference/class/KeyValueStore#set_value) methods to retrieve and assign values, respectively.

Every Actor run is linked to a default key-value store that is automatically created for that specific run. If you're running your Actors and opt to store data locally, you can easily supply the [input](https://docs.apify.com/platform/actors/running/input-and-output) by placing an *INPUT.json* file in the corresponding directory of the default key-value store. This method ensures that you Actor has all the necessary data readily available for its execution.

You can find *INPUT.json* and other key-value store files in the location below.

{APIFY\_LOCAL\_STORAGE\_DIR}/key\_value\_stores/{STORE\_ID}/{KEY}.{EXT}

The default key-value store's ID is *default*. The {KEY} is the record's *key* and {EXT} corresponds to the record value's MIME content type.

To manage your key-value stores, you can use the following methods. See the KeyValueStore class [documentation](https://docs.apify.com/sdk/python/reference/class/KeyValueStore) for the full list.

from apify import Actor  
from apify.storages import KeyValueStore  
  
async def main():  
 async with Actor:  
 *# Open a named key-value store*  
 example\_store: KeyValueStore = await Actor.open\_key\_value\_store(name='my-store')  
  
 *# Read a record in the example\_store storage*  
 value = await example\_store.get\_value('some-key')  
  
 *# Write a record to example\_store*  
 await example\_store.set\_value('some-key', {'foo': 'bar'})  
  
 *# Delete a record from example\_store*  
 await example\_store.set\_value('some-key', None)

Note that JSON is automatically parsed to a Python dictionary, text data returned as a string and other data is returned as binary buffer.

from apify import Actor  
  
async def main():  
 async with Actor:  
 value = await Actor.get\_value('my-key')  
 *# ...*  
 image\_buffer = ... *# Get image data*  
 await Actor.set\_value(key='OUTPUT', value=image\_buffer, content\_type='image/jpeg')

The Actor.get\_input() method is not only a shortcut to Actor.get\_value('INPUT'); it is also compatible with [Actor.metamorph()](https://docs.apify.com/platform/actors/development/programming-interface/metamorph). This is because a metamorphed Actor run's input is stored in the *INPUT-METAMORPH-1* key instead of *INPUT*, which hosts the original input.

Check out the [Python SDK documentation](https://docs.apify.com/sdk/python/docs/concepts/storages#working-with-key-value-stores) and the KeyValueStore class's [API reference](https://docs.apify.com/sdk/python/reference/class/KeyValueStore) for details on managing your key-value stores with the Python SDK.

**Compression**[**​**](https://docs.apify.com/platform/storage/key-value-store#compression)

Previously, when using the [Put record](https://docs.apify.com/api/v2#/reference/key-value-stores/record/put-record) endpoint, every record was automatically compressed with Gzip before being uploaded. However, this process has been updated. *Now, records are stored exactly as you upload them.* This change means that it is up to you whether the record is stored compressed or uncompressed.

You can compress a record and use the [Content-Encoding request header](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Encoding) to let our platform know which compression it uses. We recommend compressing large key-value records to save storage space and network traffic.

*Using the*[*JavaScript SDK*](https://docs.apify.com/sdk/js/reference/class/KeyValueStore#setValue)*or our*[*JavaScript API client*](https://docs.apify.com/api/client/js/reference/class/KeyValueStoreClient#setRecord)*automatically compresses your files.* We advise utilizing the JavaScript API client for data compression prior to server upload and decompression upon retrieval, minimizing storage costs.

**Sharing**[**​**](https://docs.apify.com/platform/storage/key-value-store#sharing)

You can grant [access rights](https://docs.apify.com/platform/collaboration) to your key-value store through the **Share** button under the **Actions** menu. For more details check the [full list of permissions](https://docs.apify.com/platform/collaboration/list-of-permissions).

**Sharing key-value stores between runs**[**​**](https://docs.apify.com/platform/storage/key-value-store#sharing-key-value-stores-between-runs)

You can access a key-value store from any [Actor](https://docs.apify.com/platform/actors) or [task](https://docs.apify.com/platform/actors/running/tasks) run as long as you know its *name* or *ID*.

To access a key-value store from another run using the [Apify SDK](https://docs.apify.com/sdk), open it using the same method as you would do with any other store.

* **JavaScript**
* **Python**

import { Actor } from 'apify';  
  
await Actor.init();  
  
const otherStore = await Actor.openKeyValueStore('old-store');  
*// ...*  
  
await Actor.exit();

In the [JavaScript API client](https://docs.apify.com/api/client/js/reference/class/KeyValueStoreClient) as well as in [Python API client](https://docs.apify.com/api/client/python/reference/class/KeyValueStoreClient), you can access a store using its client. Once you've opened a store, read and manage its contents like you would do with a key-value store from your current run.

* **JavaScript**
* **Python**

const otherStoreClient = apifyClient.keyValueStore('jane-doe/old-store');

The same applies for the [Apify API](https://docs.apify.com/platform/storage/key-value-store#apify-api) - you can use [the same endpoints](https://docs.apify.com/platform/storage/key-value-store#apify-api) as you would normally do.

Check out the [Storage overview](https://docs.apify.com/platform/storage/usage#sharing-storages-between-runs) for details on sharing storages between runs.

**Data consistency**[**​**](https://docs.apify.com/platform/storage/key-value-store#data-consistency)

Key-value storage uses the [AWS S3](https://aws.amazon.com/s3/) service. According to the [S3 documentation](https://aws.amazon.com/s3/consistency/), it provides *strong read-after-write* consistency.

**Limits**[**​**](https://docs.apify.com/platform/storage/key-value-store#limits)

* The maximum length for key of key-value store is 63 characters

**Request queue**

**Queue URLs for an Actor to visit in its run. Learn how to share your queues between Actor runs. Access and manage request queues from Apify Console or via API.**

Request queues enable you to enqueue and retrieve requests such as URLs with an [HTTP method](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods) and other parameters. They prove essential not only in web crawling scenarios but also in any situation requiring the management of a large number of URLs and the addition of new links.

The storage system for request queues accommodates both breadth-first and depth-first crawling strategies, along with the inclusion of custom data attributes. This system enables you to check if certain URLs have already been encountered, add new URLs to the queue, and retrieve the next set of URLs for processing.

Named request queues are retained indefinitely.  
Unnamed request queues expire after 7 days unless otherwise specified.  
> [Learn more](https://docs.apify.com/platform/storage/usage#named-and-unnamed-storages)

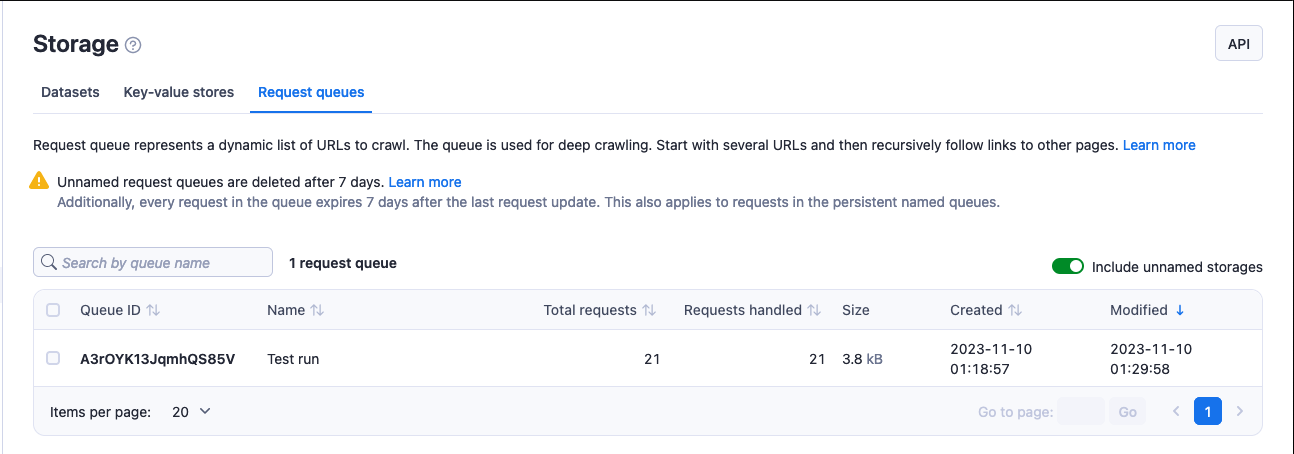
**Basic usage**[**​**](https://docs.apify.com/platform/storage/request-queue#basic-usage)

You can access your request queues in several ways:

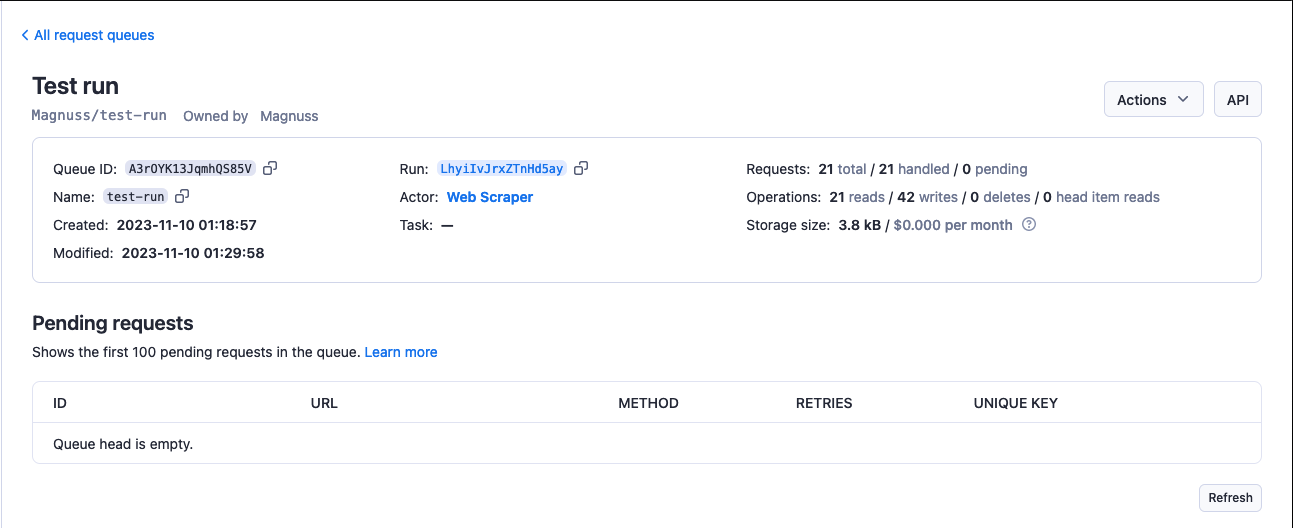
* [Apify Console](https://console.apify.com/) - provides an easy-to-understand interface.
* [Apify API](https://docs.apify.com/api/v2) - for accessing your request queues programmatically.
* [Apify API clients](https://docs.apify.com/api) - to access your request queues from any Node.js application.
* [Apify SDK](https://docs.apify.com/sdk) - when building your own JavaScript Actor.

**Apify Console**[**​**](https://docs.apify.com/platform/storage/request-queue#apify-console)

In the [Apify Console](https://console.apify.com/), you can view your request queues in the [Storage](https://console.apify.com/storage) section under the [Request queues](https://console.apify.com/storage?tab=requestQueues) tab.



To view a request queue, click on its **Queue ID**. Under the **Actions** menu, you can rename your queue's name (and, in turn, its [retention period](https://docs.apify.com/platform/storage/usage#named-and-unnamed-storages)) and [access rights](https://docs.apify.com/platform/collaboration) using the **Share** button. Click on the **API** button to view and test a queue's [API endpoints](https://docs.apify.com/api/v2#/reference/request-queues).



**Apify API**[**​**](https://docs.apify.com/platform/storage/request-queue#apify-api)

The [Apify API](https://docs.apify.com/api/v2#/reference/request-queues) allows you programmatic access to your request queues using [HTTP requests](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods).

If you are accessing your datasets using the username~store-name [store ID format](https://docs.apify.com/platform/storage), you will need to use your secret API token. You can find the token (and your user ID) on the [Integrations](https://console.apify.com/account#/integrations) page of your Apify account.

When providing your API authentication token, we recommend using the request's Authorization header, rather than the URL. ([More info](https://docs.apify.com/platform/integrations/api#authentication)).

To get a list of your request queues, send a GET request to the [Get list of request queues](https://docs.apify.com/api/v2#/reference/request-queues/store-collection/get-list-of-request-queues) endpoint.

https://api.apify.com/v2/request-queues

To get information about a request queue such as its creation time and item count, send a GET request to the [Get request queue](https://docs.apify.com/api/v2#/reference/request-queues/queue/get-request-queue) endpoint.

https://api.apify.com/v2/request-queues/{QUEUE\_ID}

To get a request from a queue, send a GET request to the [Get request](https://docs.apify.com/api/v2#/reference/request-queues/request/get-request) endpoint.

https://api.apify.com/v2/request-queues/{QUEUE\_ID}/requests/{REQUEST\_ID}

To add a request to a queue, send a POST request with the request to be added as a JSON object in the request's payload to the [Add request](https://docs.apify.com/api/v2#/reference/request-queues/request-collection/add-request) endpoint.

https://api.apify.com/v2/request-queues/{QUEUE\_ID}/requests

Example payload:

{  
 "uniqueKey": "http://example.com",  
 "url": "http://example.com",  
 "method": "GET"  
}

To update a request in a queue, send a PUT request with the request to update as a JSON object in the request's payload to the [Update request](https://docs.apify.com/api/v2#/reference/request-queues/request/update-request) endpoint. In the payload, specify the request's ID and add the information you want to update.

https://api.apify.com/v2/request-queues/{QUEUE\_ID}/requests/{REQUEST\_ID}

Example payload:

{  
 "id": "dnjkDMKLmdlkmlkmld",  
 "uniqueKey": "http://example.com",  
 "url": "http://example.com",  
 "method": "GET"  
}

When adding or updating requests, you can optionally provide a clientKey parameter to your request. It must be a string between 1 and 32 characters in length. This identifier is used to determine whether the queue was accessed by [multiple clients](https://docs.apify.com/platform/storage/request-queue#sharing). If clientKey is not provided, the system considers this API call to come from a new client. See the hadMultipleClients field returned by the [Get head](https://docs.apify.com/api/v2#/reference/request-queues/queue-head/get-head) operation for details.

Example: client-abc

For further details and a breakdown of each storage API endpoint, refer to the [API documentation](https://docs.apify.com/api/v2#/reference/key-value-stores).

**Apify API Clients**[**​**](https://docs.apify.com/platform/storage/request-queue#apify-api-clients)

**JavaScript API client**[**​**](https://docs.apify.com/platform/storage/request-queue#javascript-api-client)

The Apify [JavaScript API client](https://docs.apify.com/api/client/js/reference/class/RequestQueueClient) (apify-client) enables you to access your request queues from any Node.js application, whether it is running on the Apify platform or externally.

After importing and initiating the client, you can save each request queue to a variable for easier access.

const myQueueClient = apifyClient.requestQueue('jane-doe/my-request-queue');

You can then use that variable to [access the request queue's items and manage it](https://docs.apify.com/api/client/js/reference/class/RequestQueueClient).

Check out the [JavaScript API client documentation](https://docs.apify.com/api/client/js/reference/class/RequestQueueClient) for [help with setup](https://docs.apify.com/api/client/js/docs) and more details.

**Python API client**[**​**](https://docs.apify.com/platform/storage/request-queue#python-api-client)

The Apify [Python API client](https://docs.apify.com/api/client/python) (apify-client) allows you to access your request queues from any Python application, whether it's running on the Apify platform or externally.

After importing and initiating the client, you can save each request queue to a variable for easier access.

my\_queue\_client = apify\_client.request\_queue('jane-doe/my-request-queue')

You can then use that variable to [access the request queue's items and manage it](https://docs.apify.com/api/client/python/reference/class/RequestQueueClient).

Check out the [Python API client documentation](https://docs.apify.com/api/client/python/reference/class/RequestQueueClient) for [help with setup](https://docs.apify.com/api/client/python/docs/overview/introduction) and more details.

**Apify SDKs**[**​**](https://docs.apify.com/platform/storage/request-queue#apify-sdks)

**JavaScript SDK**[**​**](https://docs.apify.com/platform/storage/request-queue#javascript-sdk)

When working with a JavaScript [Actor](https://docs.apify.com/platform/actors), the [JavaScript SDK](https://docs.apify.com/sdk/js/docs/guides/request-storage#request-queue) is an essential tool, especially for request queue management. The primary class for this purpose is the [RequestQueue](https://docs.apify.com/sdk/js/reference/class/RequestQueue) class. Use this class to decide whether your data is stored locally or in the Apify cloud.

If you are building a JavaScript [Actor](https://docs.apify.com/platform/actors), you will be using the [JavaScript SDK](https://docs.apify.com/sdk/js/docs/guides/request-storage#request-queue). The request queue is represented by a [RequestQueue](https://docs.apify.com/sdk/js/reference/class/RequestQueue) class. You can use the class to specify whether your data is stored locally or in the Apify cloud and [enqueue new URLs](https://docs.apify.com/sdk/js/reference/class/RequestQueue#addRequests).

Every Actor run is automatically linked with a default request queue, initiated upon adding the first request. This queue is primarily utilized for storing URLs to be crawled during the particular Actor run, though its use is not mandatory. For enhanced flexibility, you can establish named queues. These named queues offer the advantage of being shareable across different Actors or various Actor runs, facilitating a more interconnected and efficient process.

If you are storing your data locally, you can find your request queue at the following location.

{APIFY\_LOCAL\_STORAGE\_DIR}/request\_queues/{QUEUE\_ID}/{ID}.json

The default request queue's ID is *default*. Each request in the queue is stored as a separate JSON file, where {ID} is a request ID.

To open a request queue, use the [Actor.openRequestQueue()](https://docs.apify.com/sdk/js/reference/class/Actor#openRequestQueue) method.

*// Import the JavaScript SDK into your project*  
import { Actor } from 'apify';  
  
await Actor.init();  
*// ...*  
  
*// Open the default request queue associated with*  
*// the Actor run*  
const queue = await Actor.openRequestQueue();  
  
*// Open the 'my-queue' request queue*  
const queueWithName = await Actor.openRequestQueue('my-queue');  
  
*// ...*  
await Actor.exit();

Once a queue is open, you can manage it using the following methods. Check out the RequestQueue class's [API reference](https://docs.apify.com/sdk/js/reference/class/RequestQueue) for the full list.

*// Import the JavaScript SDK into your project*  
import { Actor } from 'apify';  
  
await Actor.init();  
*// ...*  
  
const queue = await Actor.openRequestQueue();  
  
*// Enqueue requests*  
await queue.addRequests([{ url: 'http://example.com/aaa' }]);  
await queue.addRequests(['http://example.com/foo', 'http://example.com/bar'], {  
 forefront: true,  
});  
  
*// Get the next request from queue*  
const request1 = await queue.fetchNextRequest();  
const request2 = await queue.fetchNextRequest();  
  
*// Get a specific request*  
const specificRequest = await queue.getRequest('shi6Nh3bfs3');  
  
*// Reclaim a failed request back to the queue*  
*// and process it again*  
await queue.reclaimRequest(request2);  
  
*// Remove a queue*  
await queue.drop();  
  
*// ...*  
await Actor.exit();

Check out the [JavaScript SDK documentation](https://docs.apify.com/sdk/js/docs/guides/request-storage#request-queue) and the RequestQueue class's [API reference](https://docs.apify.com/sdk/js/reference/class/RequestQueue) for details on managing your request queues with the JavaScript SDK.

**Python SDK**[**​**](https://docs.apify.com/platform/storage/request-queue#python-sdk)

For Python [Actor](https://docs.apify.com/platform/actors) development, the [Python SDK](https://docs.apify.com/sdk/python/docs/concepts/storages#working-with-request-queues) the in essential. The request queue is represented by [RequestQueue](https://docs.apify.com/sdk/python/reference/class/RequestQueue) class. Utilize this class to determine whether your data is stored locally or in the Apify cloud. For managing your data, it provides the capability to [enqueue new URLs](https://docs.apify.com/sdk/python/reference/class/RequestQueue#add_requests), facilitating seamless integration and operation within your Actor.

Every Actor run is automatically connected to a default request queue, established specifically for that run upon the addition of the first request. If you're operating your Actors and choose to utilize this queue, it typically serves to store URLs for crawling in the respective Actor run, though its use is not mandatory. To extend functionality, you have the option to create named queue, which offer the flexibility to be shared among different Actors or across multiple Actor runs.

If you are storing your data locally, you can find your request queue at the following location.

{APIFY\_LOCAL\_STORAGE\_DIR}/request\_queues/{QUEUE\_ID}/{ID}.json

The default request queue's ID is *default*. Each request in the queue is stored as a separate JSON file, where {ID} is a request ID.

To *open a request queue*, use the [Actor.open\_request\_queue()](https://docs.apify.com/sdk/python/reference/class/Actor#open_request_queue) method.

from apify import Actor  
  
async def main():  
 async with Actor:  
 *# Open the default request queue associated with the Actor run*  
 queue = await Actor.open\_request\_queue()  
  
 *# Open the 'my-queue' request queue*  
 queue\_with\_name = await Actor.open\_request\_queue(name='my-queue')  
  
 *# ...*

Once a queue is open, you can manage it using the following methods. See the RequestQueue class's [API reference](https://docs.apify.com/sdk/python/reference/class/RequestQueue) for the full list.

from apify import Actor  
from apify.storages import RequestQueue  
  
async def main():  
 async with Actor:  
 queue: RequestQueue = await Actor.open\_request\_queue()  
  
 *# Enqueue requests*  
 await queue.add\_request(request={'url': 'http:#example.com/aaa'})  
 await queue.add\_request(request={'url': 'http:#example.com/foo'})  
 await queue.add\_request(request={'url': 'http:#example.com/bar'}, forefront=True)  
  
 *# Get the next requests from queue*  
 request1 = await queue.fetch\_next\_request()  
 request2 = await queue.fetch\_next\_request()  
  
 *# Get a specific request*  
 specific\_request = await queue.get\_request('shi6Nh3bfs3')  
  
 *# Reclaim a failed request back to the queue and process it again*  
 await queue.reclaim\_request(request2)  
  
 *# Remove a queue*  
 await queue.drop()

Check out the [Python SDK documentation](https://docs.apify.com/sdk/python/docs/concepts/storages#working-with-request-queues) and the RequestQueue class's [API reference](https://docs.apify.com/sdk/python/reference/class/RequestQueue) for details on managing your request queues with the Python SDK.

**Features**[**​**](https://docs.apify.com/platform/storage/request-queue#features)

Request queue is a storage type built with scraping in mind, enabling developers to write scraping logic efficiently and scalably. The Apify tooling, including [Crawlee](https://crawlee.dev/), [Apify SDK for JavaScript](https://docs.apify.com/sdk/js/), and [Apify SDK for Python](https://docs.apify.com/sdk/python/), incorporates all these features, enabling users to leverage them effortlessly without extra configuration.

In the following section, we will discuss each of the main features in depth.

**Persistence and retention**[**​**](https://docs.apify.com/platform/storage/request-queue#persistence-and-retention)

Request queues prioritize persistence, ensuring indefinite retention of your requests in named request queues, and for the data retention period in your subscription in unnamed request queues. This capability facilitates incremental crawling, where you can append new URLs to the queue and resume from where you stopped in subsequent Actor runs. Consider the scenario of scraping an e-commerce website with thousands of products. Incremental scraping allows you to scrape only the products added since the last product discovery.

In the following code example, we demonstrate how to use the Apify SDK and Crawlee to create an incremental crawler that saves the title of each new found page in Apify Docs to a dataset. By running this Actor multiple times, you can incrementally crawl the source website and save only pages added since the last crawl, as reusing a single request queue ensures that only URLs not yet visited are processed.

*// Basic example of incremental crawling with Crawlee.*  
import { Actor } from 'apify';  
import { CheerioCrawler, Dataset } from 'crawlee';  
  
interface Input {  
 startUrls: string[];  
 persistRquestQueueName: string;  
}  
  
await Actor.init();  
  
*// Structure of input is defined in input\_schema.json*  
const {  
 startUrls = ['https://docs.apify.com/'],  
 persistRequestQueueName = 'persist-request-queue',  
} = (await Actor.getInput<Input>()) ?? ({} as Input);  
  
*// Open or create request queue for incremental scrape.*  
*// By opening same request queue, the crawler will continue where it left off and skips already visited URLs.*  
const requestQueue = await Actor.openRequestQueue(persistRequestQueueName);  
  
const proxyConfiguration = await Actor.createProxyConfiguration();  
  
const crawler = new CheerioCrawler({  
 proxyConfiguration,  
 requestQueue, *// Pass incremental request queue to the crawler.*  
 requestHandler: async ({ enqueueLinks, request, $, log }) => {  
 log.info('enqueueing new URLs');  
 await enqueueLinks();  
  
 *// Extract title from the page.*  
 const title = $('title').text();  
 log.info(`New page with ${title}`, { url: request.loadedUrl });  
  
 *// Save the URL and title of the loaded page to the output dataset.*  
 await Dataset.pushData({ url: request.loadedUrl, title });  
 },  
});  
  
await crawler.run(startUrls);  
  
await Actor.exit();

**Batch operations**[**​**](https://docs.apify.com/platform/storage/request-queue#batch-operations)

Request queues support batch operations on requests to enqueue or retrieve multiple requests in bulk, to cut down on network latency and enable easier parallel processing of requests. You can find the batch operations in the [Apify API](https://docs.apify.com/api/v2#/reference/request-queues/batch-request-operations), as well in the Apify API client for [JavaScript](https://docs.apify.com/api/client/js/reference/class/RequestQueueClient#batchAddRequests) and [Python](https://docs.apify.com/api/client/python/reference/class/RequestQueueClient#batch_add_requests).

* **JavaScript**
* **Python**

const { ApifyClient } = require('apify-client');  
  
const client = new ApifyClient({  
 token: 'MY-APIFY-TOKEN',  
});  
  
const requestQueueClient = client.requestQueue('my-queue-id');  
  
*// Add multiple requests to the queue*  
await requestQueueClient.batchAddRequests([  
 {  
 url: 'http://example.com/foo',  
 uniqueKey: 'http://example.com/foo',  
 method: 'GET',  
 },  
 {  
 url: 'http://example.com/bar',  
 uniqueKey: 'http://example.com/bar',  
 method: 'GET',  
 },  
]);  
  
*// Remove multiple requests from the queue*  
await requestQueueClient.batchDeleteRequests([  
 { uniqueKey: 'http://example.com/foo' },  
 { uniqueKey: 'http://example.com/bar' },  
]);

**Distributivity**[**​**](https://docs.apify.com/platform/storage/request-queue#distributivity)

Request queue includes a locking mechanism to avoid concurrent processing of one request by multiple clients (for example Actor runs). You can lock a request so that no other clients receive it when they fetch the queue head, with an expiration period on the lock so that requests which fail processing are eventually unlocked and retried.

This feature is seamlessly integrated into Crawlee, requiring minimal extra setup. By default, requests are locked for the same duration as the timeout for processing requests in the crawler ([requestHandlerTimeoutSecs](https://crawlee.dev/api/next/basic-crawler/interface/BasicCrawlerOptions#requestHandlerTimeoutSecs)). If the Actor processing the request fails, the lock expires, and the request is processed again eventually. For more details, refer to the [Crawlee documentation](https://crawlee.dev/docs/next/experiments/experiments-request-locking).

In the following example, we demonstrate how you can use locking mechanisms to avoid concurrent processing of the same request across multiple Actor runs.

**info**

The lock mechanism works on the client level, as well as the run level, when running the Actor on the Apify platform.

This means you can unlock or prolong the lock the locked request only if:

* You are using the same client key, or
* The operation is being called from the same Actor run.
* **Actor 1**
* **Actor 2**

import { Actor, ApifyClient } from 'apify';  
  
await Actor.init();  
  
const client = new ApifyClient({  
 token: 'MY-APIFY-TOKEN',  
});  
  
*// Creates a new request queue.*  
const requestQueue = await client.requestQueues().getOrCreate('example-queue');  
  
*// Creates two clients with different keys for the same request queue.*  
const requestQueueClient = client.requestQueue(requestQueue.id, {  
 clientKey: 'requestqueueone',  
});  
  
*// Adds multiple requests to the queue.*  
await requestQueueClient.batchAddRequests([  
 {  
 url: 'http://example.com/foo',  
 uniqueKey: 'http://example.com/foo',  
 method: 'GET',  
 },  
 {  
 url: 'http://example.com/bar',  
 uniqueKey: 'http://example.com/bar',  
 method: 'GET',  
 },  
 {  
 url: 'http://example.com/baz',  
 uniqueKey: 'http://example.com/baz',  
 method: 'GET',  
 },  
 {  
 url: 'http://example.com/qux',  
 uniqueKey: 'http://example.com/qux',  
 method: 'GET',  
 },  
]);  
  
*// Locks the first two requests at the head of the queue.*  
const processingRequestsClientOne = await requestQueueClient.listAndLockHead(  
 {  
 limit: 2,  
 lockSecs: 120,  
 },  
);  
  
*// Checks when the lock will expire. The locked request will have a lockExpiresAt attribute.*  
const lockedRequest = processingRequestsClientOne.items[0];  
const lockedRequestDetail = await requestQueueClient.getRequest(  
 lockedRequest.id,  
);  
console.log(`Request locked until ${lockedRequestDetail?.lockExpiresAt}`);  
  
*// Prolongs the lock of the first request or unlocks it.*  
await requestQueueClient.prolongRequestLock(  
 lockedRequest.id,  
 { lockSecs: 120 },  
);  
await requestQueueClient.deleteRequestLock(  
 lockedRequest.id,  
);  
  
await Actor.exit();

A detailed tutorial on how to process one request queue with multiple Actor runs can be found in [Academy tutorials](https://docs.apify.com/academy/node-js/multiple-runs-scrape).

**Sharing**[**​**](https://docs.apify.com/platform/storage/request-queue#sharing)

You can grant [access rights](https://docs.apify.com/platform/collaboration) to your request queue through the **Share** button under the **Actions** menu. For more details check the [full list of permissions](https://docs.apify.com/platform/collaboration/list-of-permissions).

**Sharing request queues between runs**[**​**](https://docs.apify.com/platform/storage/request-queue#sharing-request-queues-between-runs)

You can access a request queue from any [Actor](https://docs.apify.com/platform/actors) or [task](https://docs.apify.com/platform/actors/running/tasks) run as long as you know its *name* or *ID*.

To access a request queue from another run using the [Apify SDK](https://docs.apify.com/sdk), open it using the same method like you would do with any other request queue.

* **JavaScript**
* **Python**

import { Actor } from 'apify';  
  
await Actor.init();  
  
const otherQueue = await Actor.openRequestQueue('old-queue');  
*// ...*  
  
await Actor.exit();

In the [JavaScript API client](https://docs.apify.com/api/client/js/reference/class/RequestQueueClient) as well as in [Python API client](https://docs.apify.com/api/client/python/reference/class/RequestQueueClient), you can access a request queue using its respective client. Once you've opened the request queue, you can use it in your crawler or add new requests like you would do with a queue from your current run.

* **JavaScript**
* **Python**

const otherQueueClient = apifyClient.requestQueue('jane-doe/old-queue');

The same applies for the [Apify API](https://docs.apify.com/platform/storage/request-queue#apify-api) - you can use [the same endpoints](https://docs.apify.com/platform/storage/request-queue#apify-api) as you would normally do.

Check out the [Storage overview](https://docs.apify.com/platform/storage/usage#sharing-storages-between-runs) for details on sharing storages between runs.

**Limits**[**​**](https://docs.apify.com/platform/storage/request-queue#limits)

* The maximum length for request queue name is 63 characters.

**Rate limiting**[**​**](https://docs.apify.com/platform/storage/request-queue#rate-limiting)

When managing request queues via [API](https://docs.apify.com/api/v2#/reference/request-queues/put-items), CRUD ([add](https://docs.apify.com/api/v2#/reference/request-queues/request-collection/add-request), [get](https://docs.apify.com/api/v2#/reference/request-queues/request-collection/get-request), [update](https://docs.apify.com/api/v2#/reference/request-queues/request-collection/update-request), [delete](https://docs.apify.com/api/v2#/reference/request-queues/request-collection/delete-request)) operation requests are limited to *200 requests per second* per request queue. This helps protect Apify servers from being overloaded.

**Proxy**

**Learn to anonymously access websites in scraping/automation jobs. Improve data outputs and efficiency of bots, and access websites from various geographies.**

[Apify Proxy](https://apify.com/proxy) allows you to change your IP address when web scraping to reduce the chance of being [blocked](https://docs.apify.com/academy/anti-scraping/techniques) because of your geographical location.

You can use proxies in your [Actors](https://docs.apify.com/platform/actors) or any other application that supports HTTP proxies. Apify Proxy monitors the health of your IP pool and intelligently rotates addresses to prevent IP address-based blocking.

You can view your proxy settings and password on the [Proxy](https://console.apify.com/proxy) page in Apify Console. For pricing information, visit [apify.com/pricing](https://apify.com/pricing).

**Quickstart**[**​**](https://docs.apify.com/platform/proxy#quickstart)

Usage of Apify Proxy means just a couple of lines of code, thanks to our [SDKs](https://docs.apify.com/sdk):

* **JavaScript SDK with PuppeteerCrawler**
* **Python SDK with requests**

import { Actor } from 'apify';  
import { PuppeteerCrawler } from 'crawlee';  
  
await Actor.init();  
  
const proxyConfiguration = await Actor.createProxyConfiguration();  
  
const crawler = new PuppeteerCrawler({  
 proxyConfiguration,  
 async requestHandler({ page }) {  
 console.log(await page.content());  
 },  
});  
  
await crawler.run(['https://proxy.apify.com/?format=json']);  
  
await Actor.exit();

**Usage**

**Learn how to configure and use Apify Proxy. See the required parameters such as the correct username and password.**

**Connection settings**[**​**](https://docs.apify.com/platform/proxy/usage#connection-settings)

To connect to Apify Proxy, you use the [HTTP proxy protocol](https://en.wikipedia.org/wiki/Proxy_server#Web_proxy_servers). This means that you need to configure your HTTP client to use the proxy server at the Apify Proxy hostname and provide it with your Apify Proxy password and the other parameters described below.

The full connection string has the following format:

http://<username>:<password>@<hostname>:<port>

**caution**

All usage of Apify Proxy with your password is charged towards your account. Do not share the password with untrusted parties or use it from insecure networks, as **the password is sent unencrypted** due to the HTTP protocol's [limitations](https://www.guru99.com/difference-http-vs-https.html).

**External connection**[**​**](https://docs.apify.com/platform/proxy/usage#external-connection)

If you want to connect to Apify Proxy from outside of the Apify Platform, you need to have a paid Apify plan (to prevent abuse). If you need to test Apify Proxy before you subscribe, please [contact our support](https://apify.com/contact).

| **Parameter** | **Value / explanation** |
| --- | --- |
| Hostname | proxy.apify.com |
| Port | 8000 |
| Username | Specifies the proxy parameters such as groups, [session](https://docs.apify.com/platform/proxy/usage#sessions) and location. See [username parameters](https://docs.apify.com/platform/proxy/usage#username-parameters) below for details. **Note**: this is not your Apify username. |
| Password | Apify Proxy password. Your password is displayed on the [Proxy](https://console.apify.com/proxy/groups) page in Apify Console. **Note**: this is not your Apify account password. |

**caution**

If you use these connection parameters for connecting to Apify Proxy from your Actors running on the Apify Platform, the connection will still be considered external, it will not work on the Free plan, and on paid plans you will be charged for external data transfer. Please use the connection parameters from the [Connection from Actors](https://docs.apify.com/platform/proxy/usage#connection-from-actors) section when using Apify Proxy from Actors.

Example connection string for external connections:

http://auto:apify\_proxy\_EaAFg6CFhc4eKk54Q1HbGDEiUTrk480uZv03@proxy.apify.com:8000

**Connection from Actors**[**​**](https://docs.apify.com/platform/proxy/usage#connection-from-actors)

If you want to connect to Apify Proxy from Actors running on the Apify Platform, the recommended way is to use built-in proxy configuration tools in the [Apify SDK JavaScript](https://docs.apify.com/sdk/js/docs/guides/proxy-management) or [Apify SDK Python](https://docs.apify.com/sdk/python/docs/concepts/proxy-management)

If you don't want to use these helpers, and want to connect to Apify Proxy manually, you can find the right configuration values in [environment variables](https://docs.apify.com/platform/actors/development/programming-interface/environment-variables) provided to the Actor. By using this configuration, you ensure that you connect to Apify Proxy directly through the Apify infrastructure, bypassing any external connection via the Internet, thereby improving the connection speed, and ensuring you don't pay for external data transfer.

| **Parameter** | **Source / explanation** |
| --- | --- |
| Hostname | APIFY\_PROXY\_HOSTNAME environment variable |
| Port | APIFY\_PROXY\_PORT environment variable |
| Username | Specifies the proxy parameters such as groups, [session](https://docs.apify.com/platform/proxy/usage#sessions) and location. See [username parameters](https://docs.apify.com/platform/proxy/usage#username-parameters) below for details. **Note**: this is not your Apify username. |
| Password | APIFY\_PROXY\_PASSWORD environment variable |

Example connection string creation:

const { APIFY\_PROXY\_HOSTNAME, APIFY\_PROXY\_PORT, APIFY\_PROXY\_PASSWORD } = process.env;  
const connectionString = `http://auto:${APIFY\_PROXY\_PASSWORD}@${APIFY\_PROXY\_HOSTNAME}:${APIFY\_PROXY\_PORT}`;

**Username parameters**[**​**](https://docs.apify.com/platform/proxy/usage#username-parameters)

The username field enables you to pass parameters like [**groups**](https://docs.apify.com/platform/proxy/usage#proxy-groups), [**session ID**](https://docs.apify.com/platform/proxy/usage#sessions) and **country** for your proxy connection.

For example, if you're using [datacenter proxies](https://docs.apify.com/platform/proxy/datacenter-proxy) and want to use the new\_job\_123 session using the SHADER group, the username will be:

groups-SHADER,session-new\_job\_123

The table below describes the available parameters.

| Parameter | Type | Description |
| --- | --- | --- |
| **groups** | Required | Set proxied requests to use servers from the selected groups: - groups-[group name] or auto when using datacenter proxies. - groups-RESIDENTIAL when using residential proxies. - groups-GOOGLE\_SERP when using Google SERP proxies. |
| **session** | Optional | If specified to session-new\_job\_123, for example, all proxied requests with the same session identifier are routed through the same IP address. If not specified, each proxied request is assigned a randomly picked least used IP address.  The session string can only contain numbers (0–9), letters (a-z or A-Z), dot (.), underscore (\_), a tilde (~). The maximum length is 50 characters.  Session management may work differently for residential and SERP proxies. Check relevant documentations for more details. |
| **country** | Optional | If specified, all proxied requests will use proxy servers from a selected country. Note that if there are no proxy servers from the specified country, the connection will fail. For example groups-SHADER,country-US uses proxies from the SHADER group located in the USA. By default, the proxy uses all available proxy servers from all countries. |

If you want to specify one parameter and not the others, just provide that parameter and omit the others. To use the default behavior (not specifying either groups, session, or country), set the username to auto, which serves as a default placeholder, because the proxy username cannot be empty.

**Code examples**[**​**](https://docs.apify.com/platform/proxy/usage#code-examples)

We have code examples for connecting to our proxy using the [Apify SDK](https://docs.apify.com/sdk) and [Crawlee](https://crawlee.dev/) and other libraries, as well as examples in PHP.

* [Datacenter proxy](https://docs.apify.com/platform/proxy/datacenter-proxy#examples)
* [Residential proxy](https://docs.apify.com/platform/proxy/residential-proxy#connecting-to-residential-proxy)
* [Google SERP proxy](https://docs.apify.com/platform/proxy/google-serp-proxy#examples)

For code examples related to proxy management in Apify SDK and Crawlee, see:

* [Apify SDK JavaScript](https://docs.apify.com/sdk/js/docs/guides/proxy-management)
* [Apify SDK Python](https://docs.apify.com/sdk/python/docs/concepts/proxy-management)
* [Crawlee](https://crawlee.dev/docs/guides/proxy-management)

**IP address rotation**[**​**](https://docs.apify.com/platform/proxy/usage#ip-address-rotation)

Web scrapers can rotate the IP addresses they use to access websites. They assign each request a different IP address, which makes it appear like they are all coming from different users. This greatly enhances performance and data throughout.

Depending on whether you use a [browser](https://apify.com/apify/web-scraper) or [HTTP requests](https://apify.com/apify/cheerio-scraper) for your scraping jobs, IP address rotation works differently.

* Browser—a different IP address is used for each browser.
* HTTP request—a different IP address is used for each request.

Use [sessions](https://docs.apify.com/platform/proxy/usage#sessions) to control how you rotate IP addresses. See our guide [Anti-scraping techniques](https://docs.apify.com/academy/anti-scraping/techniques) to learn more about IP address rotation and our findings on how blocking works.

**Sessions**[**​**](https://docs.apify.com/platform/proxy/usage#sessions)

Sessions allow you to use the same IP address for multiple connections. In cases where you need to keep the same session (e.g. when you need to log in to a website), it is best to keep the same proxy and so the IP address. On the other hand by switching the IP address, you can avoid being blocked by the website.

To set a new session, pass the session parameter in your [username](https://docs.apify.com/platform/proxy/usage#username-parameters) field when connecting to a proxy. This will serve as the session's ID and an IP address will be assigned to it. To [use that IP address in other requests](https://docs.apify.com/platform/proxy/datacenter-proxy#connecting-to-datacenter-proxies), pass that same session ID in the username field.

We recommend you to use [SessionPool](https://crawlee.dev/api/core/class/SessionPool) abstraction when managing sessions. The created session will then store information such as cookies and can be used to generate [browser fingerprints](https://docs.apify.com/academy/anti-scraping/mitigation/generating-fingerprints). You can also assign custom user data such as authorization tokens and specific headers. Sessions are available for [datacenter](https://docs.apify.com/platform/proxy/datacenter-proxy) and [residential](./ residential\_proxy.md#session-persistence) proxies. For datacenter proxies, a session persists for **26 hours** ([more info](https://docs.apify.com/platform/proxy/datacenter-proxy)). For residential proxies, it persists for **1 minute** ([more info](https://docs.apify.com/platform/proxy/residential-proxy#session-persistence)) but you can prolong the lifetime by regularly using the session. Google SERP proxies do not support sessions.

**Proxy groups**[**​**](https://docs.apify.com/platform/proxy/usage#proxy-groups)

You can see which proxy groups you have access to on the [Proxy page](https://console.apify.com/proxy/groups) in the Apify Console. To use a specific proxy group (or multiple groups), specify it in the username parameter.

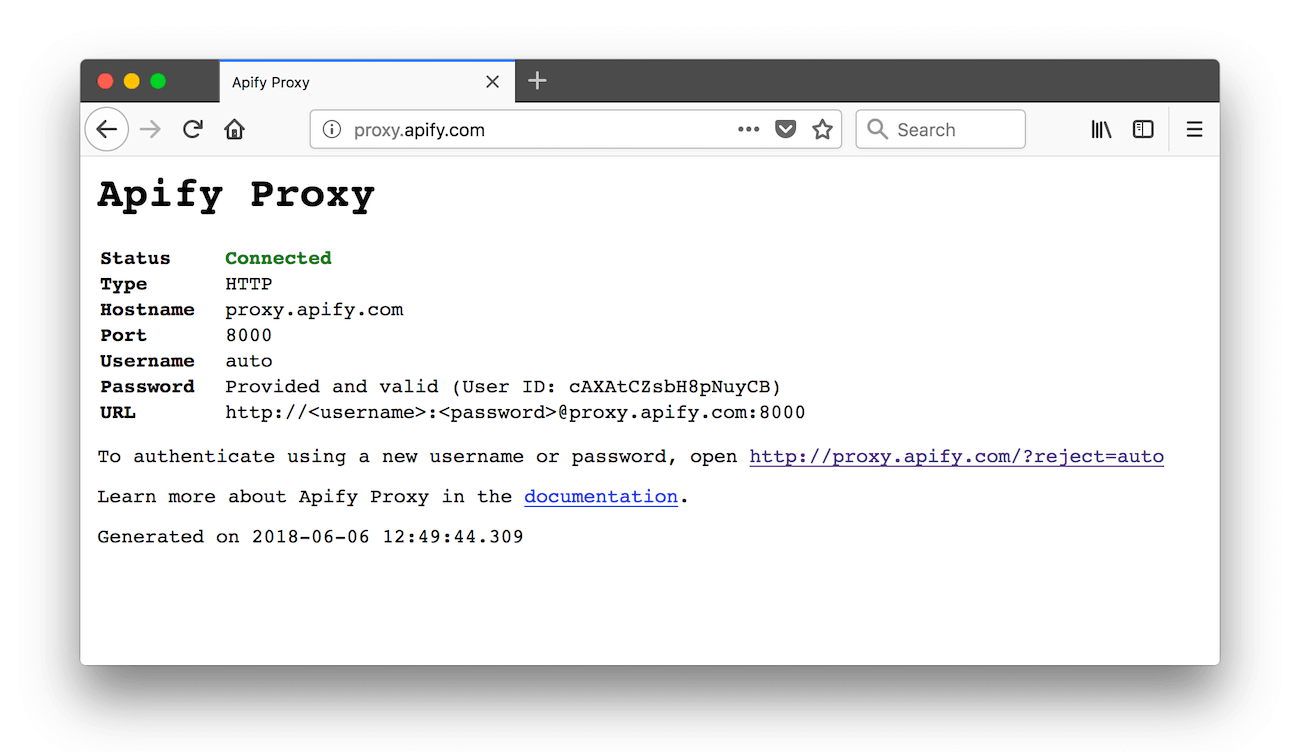
**Proxy IP addresses**[**​**](https://docs.apify.com/platform/proxy/usage#proxy-ip-addresses)

If you need to allow communication to apify.proxy.com, add the following IP addresses to your firewall rule or whitelist:

* 18.208.102.16
* 35.171.134.41

**Troubleshooting**[**​**](https://docs.apify.com/platform/proxy/usage#troubleshooting)

To view your connection status to [Apify Proxy](https://apify.com/proxy), open the URL below in the browser using the proxy. <http://proxy.apify.com/>. If the proxy connection is working, the page should look something like this:



To test that your requests are proxied and IP addresses are being [rotated](https://docs.apify.com/academy/anti-scraping/techniques) correctly, open the following API endpoint via the proxy. It shows information about the client IP address.

<https://api.apify.com/v2/browser-info/>

**A different approach to 502 Bad Gateway**[**​**](https://docs.apify.com/platform/proxy/usage#a-different-approach-to-502-bad-gateway)

Sometimes when the 502 status code is not comprehensive enough. Therefore, we have modified our server with 590-599 codes instead to provide more insight:

* 590 Non Successful: upstream responded with non-200 status code.
* 591 RESERVED: *this status code is reserved for further use.*
* 592 Status Code Out Of Range: upstream responded with status code different than 100–999.
* 593 Not Found: DNS lookup failed, indicating either [EAI\_NODATA](https://github.com/libuv/libuv/blob/cdbba74d7a756587a696fb3545051f9a525b85ac/include/uv.h#L82) or [EAI\_NONAME](https://github.com/libuv/libuv/blob/cdbba74d7a756587a696fb3545051f9a525b85ac/include/uv.h#L83).
* 594 Connection Refused: upstream refused connection.
* 595 Connection Reset: connection reset due to loss of connection or timeout.
* 596 Broken Pipe: trying to write on a closed socket.
* 597 Auth Failed: incorrect upstream credentials.
* 598 RESERVED: *this status code is reserved for further use.*
* 599 Upstream Error: generic upstream error.

The typical issues behind these codes are:

* 590 and 592 indicate an issue on the upstream side.
* 593 indicates an incorrect proxy-chain configuration.
* 594, 595 and 596 may occur due to connection loss.
* 597 indicates incorrect upstream credentials.
* 599 is a generic error, where the above is not applicable.