



fal api

Source: <https://fal.ai/docs/api-reference/cli/api>

Call any model on fal directly from the command line. Useful for quick testing without writing code.

Usage

```
fal api <model_id> [key=value ...]
```

Arguments:

Argument	Description
<code>model_id</code>	The model endpoint to call (e.g., <code>fal-ai/flux/schnell</code>)
<code>params</code>	Key-value pairs for the request body (e.g., <code>prompt="a sunset"</code>)

Examples

Generate an image:

```
fal api fal-ai/flux/schnell prompt="a cat wearing sunglasses"
```

The CLI shows real-time status updates (queued, in progress) and logs while the request is processing, then prints the result.

Nested parameters:

Use bracket notation for nested values:

```
fal api fal-ai/flux/schnell prompt="a sunset" image_size[width]:=1280 image_size[height]:=
```

Use `:=` for non-string values (numbers, booleans):

```
fal api fal-ai/flux/schnell prompt="a sunset" num_images:=4 seed:=42
```

Streaming endpoints:

If the model ID ends with `/stream`, the CLI streams output as it's generated:

```
fal api fal-ai/any-llm/stream prompt="What is the meaning of life?" model=google/gemini-flan
```

How It Works

- For regular endpoints: submits via the queue, polls for status with live log display, then prints the result
- For `/stream` endpoints: connects via streaming and prints output as it arrives
- Uses your configured `FAL_KEY` for authentication

`fal apps delete`

Source: <https://fal.ai/docs/api-reference/cli/apps/delete>

```
Usage: fal apps delete [-h] [--debug] [--pdb] [--cprofile] [--env ENV] app_name
```

Delete application.

Positional Arguments:

app_name Application name.

Options:

-h, --help show this help message and exit
--env ENV Target environment (defaults to main).

Examples:

```
fal apps delete my-app  
fal apps delete my-app --env staging
```

fal apps delete-rev

Source: <https://fal.ai/docs/api-reference/cli/apps/delete-rev>

```
Usage: fal apps delete-rev [-h] [--debug] [--pdb] [--cprofile] app_rev
```

Delete application revision.

Positional Arguments:

app_rev Application revision.

Options:

-h, --help show this **help** message and **exit**

fal apps list

Source: <https://fal.ai/docs/api-reference/cli/apps/list>

```
Usage: fal apps list [-h] [--debug] [--pdb] [--cprofile] [--env ENV]
```

List applications.

Options:

-h, --help show this **help** message and **exit**

--env ENV Target environment (defaults to main).

Examples:

```
fal apps list
```

```
fal apps list --env staging
```

fal apps list-rev

Source: <https://fal.ai/docs/api-reference/cli/apps/list-rev>

```
Usage: fal apps list-rev [-h] [--debug] [--pdb] [--cprofile] [--env ENV]
                        [app_name]
```

List application revisions.

Positional Arguments:

app_name Application name (optional).

Options:

-h, --help show this **help** message and **exit**
--env ENV Target environment (defaults to main).

fal apps rollout

Source: <https://fal.ai/docs/api-reference/cli/apps/rollout>

```
Usage: fal apps rollout [-h] [--debug] [--pdb] [--cprofile] [--team TEAM]
                        [--force] [--env ENV] app_name
```

Rollout application by restarting all active runners.

Positional Arguments:

app_name Application name.

Options:

-h, --help show this **help** message and **exit**
--team TEAM The team to use.
--force Force rollout by killing runners immediately instead of
 gracefully stopping them.
--env ENV Target environment (defaults to main).

Debug:

--debug Show verbose errors.
--pdb Start pdb on error.
--cprofile Show cProfile report.

When to Use

Use `fal apps rollout` when you need to restart all runners for an application without redeploying:

- **Environment variable changes:** Force runners to pick up updated secrets or environment variables
- **Bad state recovery:** Restart runners that may be in an unhealthy state
- **Memory cleanup:** Force garbage collection and memory cleanup across all runners
- **Configuration updates:** Apply changes that require a runner restart

Graceful vs Force Rollout

Graceful Rollout (Default)

```
fal apps rollout myapp
```

Gracefully stops all active runners, allowing them to finish processing current requests before shutting down. This is the recommended approach for most situations.

How it works:

1. Identifies all active runners for the application
2. Sends a graceful stop signal to each runner
3. Runners finish their current requests
4. New runners are automatically started by the auto-scaling system as needed

Force Rollout

```
fal apps rollout myapp --force
```

Immediately kills all active runners without waiting for current requests to complete. Use this when runners are unresponsive or when you need an immediate restart.

How it works:

1. Identifies all active runners for the application
2. Immediately terminates all runners
3. New runners are automatically started by the auto-scaling system

Force rollout will terminate in-flight requests, potentially causing errors for active users. Use the graceful rollout (without `--force`) unless absolutely necessary.

Examples

Rollout after updating a secret

```
# Update a secret
fal secrets set MY_API_KEY new_value

# Gracefully rollout to pick up the new secret
fal apps rollout myapp
```

Rollout in a specific environment

```
# Update a secret in staging
fal secrets set MY_API_KEY staging_value --env staging

# Rollout the staging environment
fal apps rollout myapp --env staging
```

Force rollout to recover from frozen runners

```
# Check runners status
fal apps runners myapp

# Force immediate restart if runners are unresponsive
fal apps rollout myapp --force
```

Difference from Redeployment

`fal apps rollout` is different from `fal deploy` :

- **`fal apps rollout`** : Restarts existing runners without changing the application code or configuration. Useful for applying environment changes or recovering from bad states.
- **`fal deploy`** : Creates a new revision of your application with updated code and configuration. Use this when you've made code changes.

See Also

- [fal deploy](#) - Deploy a new revision of your application
- [fal apps scale](#) - Adjust scaling parameters
- [fal apps runners](#) - View active runners for an application

`fal apps runners`

Source: <https://fal.ai/docs/api-reference/cli/apps/runners>

```
Usage: fal apps runners [-h] [--team TEAM] [--env ENV]
                        [--since SINCE]
                        [--state {all,idle,running,pending,setup,failure_delay,terminated}]
                        [--output {pretty,json}] [--json]
                        app_name
```

List application runners.

Positional Arguments:

app_name Application name.

Options:

-h, --help show this [help](#) message and [exit](#)

--team TEAM The team to use.

--env ENV Target environment (defaults to main).

--since SINCE Show terminated runners since the given time. Accepts '[now](#)', relative time, or a date in [YYYY-MM-DD](#) format.

--state {all,idle,running,pending,setup,failure_delay,terminated} [{all,idle,running,pending,setup,failure_delay,terminated}]

Filter by runner state(s). Choose one or more, or '[all](#)' (default)

Output:

--output {pretty,json} Modify the [command](#) output

--json Output [in](#) JSON [format](#) (same as --output json)

Examples:

```
fal apps runners my-app
fal apps runners my-app --env staging
```

The runner table displays the following columns:

- **Alias** -- Application name
- **Machine Type** -- Hardware type (e.g. `GPU-A100` , `GPU-H100`)
- **Runner ID** -- Unique runner identifier
- **In Flight Requests** -- Number of active requests
- **Expires In** -- Time until runner expires
- **Uptime** -- How long the runner has been running
- **Revision** -- Application revision
- **State** -- Current runner state

fal apps scale

Source: <https://fal.ai/docs/api-reference/cli/apps/scale>

```
Usage: fal apps scale [-h] [--debug] [--pdb] [--cprofile]
                        [--keep-alive KEEP_ALIVE]
                        [--max-multiplexing MAX_MULTIPLEXING]
                        [--max-concurrency MAX_CONCURRENCY]
                        [--min-concurrency MIN_CONCURRENCY]
                        [--request-timeout REQUEST_TIMEOUT]
                        [--startup-timeout STARTUP_TIMEOUT]
                        [--machine-types MACHINE_TYPES [MACHINE_TYPES ...]]
                        [--regions REGIONS [REGIONS ...]]
                        [--env ENV]
                        app_name
```

Scale application.

Positional Arguments:

app_name Application name.

Options:

-h, --help show this [help](#) message and [exit](#)

--keep-alive KEEP_ALIVE
 Keep alive (seconds).

--max-multiplexing MAX_MULTIPLEXING
 Maximum multiplexing

--max-concurrency MAX_CONCURRENCY
 Maximum concurrency.

--min-concurrency MIN_CONCURRENCY
 Minimum concurrency

--request-timeout REQUEST_TIMEOUT
 Request [timeout](#) (seconds).

--startup-timeout STARTUP_TIMEOUT
 Startup [timeout](#) (seconds).

--machine-types MACHINE_TYPES [MACHINE_TYPES ...]
 Machine types (pass several items to [set](#) multiple).

--regions REGIONS [REGIONS ...]
 Valid regions (pass several items to [set](#) multiple).

--env ENV Target environment (defaults to main).

****Note:****

Redeploying the application, by default, will not reset these settings, except for the code-

specific settings. See [Code-Specific Settings \(Reset on Deploy\)](#) for details.

fal apps set-rev

Source: <https://fal.ai/docs/api-reference/cli/apps/set-rev>

```
Usage: fal apps set-rev [-h] [--debug] [--pdb] [--cprofile]
                        [--team TEAM] [--auth {public,private,shared}]
                        [--strategy {recreate,rolling}] [--env ENV] app_name
                        app_rev
```

Set application to a particular revision.

Positional Arguments:

app_name	Application name.
app_rev	Application revision.

Options:

-h, --help	show this help message and exit
--team TEAM	The team to use.
--auth {public,private,shared}	Application authentication mode.
--strategy {recreate,rolling}	Deployment strategy.
--env ENV	Target environment (defaults to main). Can also be set via FAL_ENV

Debug:

--debug	Show verbose errors.
--pdb	Start pdb on error.
--cprofile	Show cProfile report.

fal auth

Source: <https://fal.ai/docs/api-reference/cli/auth>

```
fal auth [-h] [--debug] [--pdb] [--cprofile] command ...
```

Authenticate with fal.

Options:

-h, --help show this help message and exit

Commands:

command

login Log in a user.

logout Log out the currently logged-in user.

whoami Show the currently authenticated user.

Login

```
Usage: fal auth login [-h] [--debug] [--pdb] [--cprofile]
```

Log in a user.

Options:

-h, --help show this help message and exit

Logout

```
Usage: fal auth logout [-h] [--debug] [--pdb] [--cprofile]
```

Log out the currently logged-in user.

Options:

-h, --help show this help message and exit

Whoami

Usage: fal auth **whoami** [-h] [--debug] [--pdb] [--cprofile]

Show the currently authenticated user.

Options:

-h, --help show this **help** message and **exit**

Debug:

--debug Show verbose errors.

--pdb Start pdb on error.

--cprofile Show cProfile report.

fal create

Source: <https://fal.ai/docs/api-reference/cli/create>

Usage: fal create [-h] [--debug] [--pdb] [--cprofile] project_type

Create fal applications.

Positional Arguments:

project_type Type of project to create.

Options:

-h, --help show this **help** message and **exit**

fal deploy

Source: <https://fal.ai/docs/api-reference/cli/deploy>

Usage: fal deploy [-h] [--output {pretty,json}] [--json] [--team TEAM] [--app-name APP_NAME] [--auth AUTH] [--strategy {recreate,rolling}] [--no-scale] [--reset-scale] [--no-cache] [--env ENV] [app_ref]

Deploy a fal application. If no app reference is provided, the command will look for a python file in the current directory.

Positional Arguments:

app_ref Application reference. Either a file path or a file path and a function name.
File path example: path/to/myfile.py::MyApp
App name example: my-app (configure team in pyproject.toml)

Options:

-h, --help show this help message and exit
--team TEAM The team to use.
--app-name APP_NAME Application name to deploy with.
--auth AUTH Application authentication mode (private, public).
--strategy {recreate,rolling} Deployment strategy.
--no-scale Use the previous deployment of the application for scale settings.
--reset-scale Use the application code for scale settings.
--no-cache Do not use the cache for the environment build.
--env ENV Target environment (defaults to main).

Output:

--output {pretty,json} Modify the command output
--json Output in JSON format (same as --output json)

Examples:

fal deploy
fal deploy path/to/myfile.py
fal deploy path/to/myfile.py::MyApp
fal deploy path/to/myfile.py::MyApp --app-name myapp --auth public
fal deploy path/to/myfile.py::MyApp --env staging
fal deploy my-app

fal doctor

Source: <https://fal.ai/docs/api-reference/cli/doctor>

```
Usage: fal doctor [-h] [--debug] [--pdb] [--cprofile]
```

fal version and misc environment information.

Options:

-h, --help show this **help** message and **exit**

fal environments

Source: <https://fal.ai/docs/api-reference/cli/environments>

```
Usage: fal environments [-h] [--debug] [--pdb] [--cprofile] command ...
```

Manage fal environments.

Options:

-h, --help show this **help** message and **exit**

Commands:

command

list	List environments.
create	Create an environment.
delete	Delete an environment.

The ``environments`` command also has an alias ``envs`` for convenience:

```
fal envs list
```

List

```
Usage: fal environments list [-h] [--debug] [--pdb] [--cprofile]
                             [--output {pretty,json}] [--json]
```

List environments.

Options:

-h, --help show this [help](#) message and [exit](#)

Output:

--output {pretty,json}

Modify the [command](#) output

--json

Output [in](#) JSON [format](#) (same as --output json)

Examples:

```
fal environments list
```

```
fal envs list --output json
```


Create

```
Usage: fal environments create [-h] [--debug] [--pdb] [--cprofile]
                                [--description DESCRIPTION]
                                name
```

Create an environment.

Positional Arguments:

name	Environment name.
------	-------------------

Options:

-h, --help	show this help message and exit
--description DESCRIPTION	Environment description.

Examples:

```
fal environments create staging
fal envs create dev --description "Development environment"
```

Delete

```
Usage: fal environments delete [-h] [--debug] [--pdb] [--cprofile] [--yes]
                                name
```

Delete an environment.

Positional Arguments:

name Environment name.

Options:

-h, --help show this [help](#) message and [exit](#)
--yes Skip confirmation prompt.

Debug:

--debug Show verbose errors.
--pdb Start pdb on error.
--cprofile Show cProfile report.

Deleting an environment permanently removes all apps and secrets in that environment. You will be prompted to confirm by typing the environment name unless `--yes` is provided.

Using Environments with Other Commands

Once you've created environments, you can target them using the `--env` flag in other commands:

```
# Deploy to an environment
fal deploy path/to/myapp.py::MyApp --env staging

# Manage secrets per environment
fal secrets set API_KEY=value --env staging
fal secrets list --env staging

# Manage apps per environment
fal apps list --env staging
fal apps scale my-app --min-concurrency 1 --env staging
fal apps runners my-app --env staging
fal apps delete my-app --env staging

# Run functions with environment-specific secrets
fal run path/to/myapp.py::MyApp --env staging
```

fal files

Source: <https://fal.ai/docs/api-reference/cli/files>

```
fal files [-h] [--debug] [--pdb] [--cprofile] command ...
```

Manage fal files.

Options:

-h, --help show this help message and exit

Commands:

command

list (ls)

List files.

download Download files.

upload Upload files.

upload-url

Upload file from URL.

mv Move or rename a remote file or directory.

rm Recursively remove a remote file or directory.

List

```
fal files list [-h] [--debug] [--pdb] [--cprofile] [--team TEAM] [path]
```

List files.

Positional Arguments:

path The path to list

Options:

-h, --help show this [help](#) message and [exit](#)
--team TEAM The team to use.

Download

```
fal files download [-h] [--debug] [--pdb] [--cprofile] [--team TEAM]  
                    remote_path local_path
```

Download files.

Positional Arguments:

remote_path Remote path to download
local_path Local path to download to

Options:

-h, --help show this [help](#) message and [exit](#)
--team TEAM The team to use.

Upload

```
fal files upload [-h] [--debug] [--pdb] [--cprofile] [--team TEAM]
                  local_path remote_path
```

Upload files.

Positional Arguments:

local_path	Local path to upload
remote_path	Remote path to upload to

Options:

-h, --help	show this help message and exit
--team TEAM	The team to use.

Upload URL

```
fal files upload-url [-h] [--debug] [--pdb] [--cprofile] [--team TEAM]
                      url remote_path
```

Upload [file](#) from URL.

Positional Arguments:

url	URL to upload
remote_path	Remote path to upload to

Options:

-h, --help	show this help message and exit
--team TEAM	The team to use.

Move

```
fal files mv [-h] [--team TEAM] source destination
```

Move or `rename` a remote `file` or directory.

Positional Arguments:

<code>source</code>	Remote <code>source</code> path
<code>destination</code>	Remote destination path

Options:

<code>-h, --help</code>	show this <code>help</code> message and <code>exit</code>
<code>--team TEAM</code>	The team to use.

Remove

```
fal files rm [-h] [--team TEAM] path
```

Recursively remove a remote `file` or directory.

Positional Arguments:

<code>path</code>	Remote path
-------------------	-------------

Options:

<code>-h, --help</code>	show this <code>help</code> message and <code>exit</code>
<code>--team TEAM</code>	The team to use.

CLI Reference

Source: <https://fal.ai/docs/api-reference/cli/index>

Complete reference for the fal command-line interface

The fal CLI is the primary tool for deploying and managing your fal applications.

Installation

```
pip install fal
```

Authentication

```
fal auth login
```

Commands

Command	Description
fal auth	Authenticate with fal
fal deploy	Deploy an application
fal run	Run a function
fal apps	Manage applications
fal keys	Manage API keys
fal secrets	Manage secrets
fal files	Manage files in /data
fal queue	Manage queued requests
fal runners	Manage runners
fal doctor	Diagnose issues
fal create	Create a new project
fal profile	Manage profiles

Installation

Source: <https://fal.ai/docs/api-reference/cli/installation>

Install latest official version

```
pip install fal
```

Install upstream version

```
pip install git+https://github.com/fal-ai/fal#subdirectory=projects/fal
```

Install development version from a git revision

```
pip install git+https://github.com/fal-ai/fal@75fe22f19cf61c7b6488d919d9a8c4bcb3433b42#subdirectory=projects/fal
```

Install development version from a git tag

```
pip install git+https://github.com/fal-ai/fal@fal_v1.10.0#subdirectory=projects/fal
```

Install development version from a git branch

```
pip install git+https://github.com/fal-ai/fal@main#subdirectory=projects/fal
```

fal keys

Source: <https://fal.ai/docs/api-reference/cli/keys>


```
Usage: fal keys [-h] [--debug] [--pdb] [--cprofile] command ...
```

Manage fal keys.

Options:

-h, --help show this help message and exit

Commands:

command

create Create a key.

list List keys.

revoke Revoke key.

Create

```
Usage: fal keys create [-h] [--debug] [--pdb] [--cprofile] --scope {ADMIN,API}
                        [--desc DESC]
```

Create a key.

Options:

-h, --help show this help message and exit

--scope {ADMIN,API} The privilege scope of the key.

--desc DESC Key description (e.g. "My Test Key")

List

```
Usage: fal keys list [-h] [--debug] [--pdb] [--cprofile]
```

List keys.

Options:

-h, --help show this help message and exit

Revoke

```
Usage: fal keys revoke [-h] [--debug] [--pdb] [--cprofile] key_id
```

Revoke key.

Positional Arguments:

key_id Key ID.

Options:

-h, --help show this **help** message and **exit**

fal profile

Source: <https://fal.ai/docs/api-reference/cli/profile>

Managing Profiles

The `fal` CLI allows you to manage multiple profiles, making it easy to switch between different fal accounts. This is particularly useful if you have multiple environments or projects.

Adding a New Profile

To add a new profile, set it as the default and then add the key:

```
> fal profile set example
Default profile set to example.
No key set for profile. Use fal profile key to set a key.

> fal profile key
Enter the key: invalid
Invalid key. The key must be in the format key:value.
Enter the key: 112f05b4-6ee8-4d06-bdb1-7ba38789ef8e:954285993fa8e651dac37a03ea2efbc9
Key set for profile example.
```

Note:

The key used in the example above is no longer valid. 😊

Listing Profiles

To list all available profiles, use the `fal profile list` command:

```
› fal profile list
```

Default	Profile	Settings
	me	key
	comfy	key
*	example	key

Setting a Default Profile

To set a default profile, use the `fal profile set` command followed by the profile name:

```
› fal profile set comfy
Default profile set to comfy.

› fal profile list
```

Default	Profile	Settings
	me	key
*	comfy	key
	example	key

After setting the default profile, you can directly access the account information without specifying the profile name.

```
› fal app list
```

Name	Revision	Auth	Min Concurrency	Max Concurrency	Max Multiplexing	Keep Alive
my-app	11111111-2222-333...	shared	0	10	1	300

Deleting a Profile

To delete a profile, use the `fal profile delete` command followed by the profile name:

```
> fal profile delete example
Profile example deleted.
```

fal queue

Source: <https://fal.ai/docs/api-reference/cli/queue>

Manage application queues.

Usage: `fal queue [-h] command ...`

Manage application queues.

Options:

`-h, --help` show this `help` message and `exit`

Commands:

`command`

`size` Get queue size `for` an application.

`flush` Flush all pending requests `in` an application queue.

Size

Usage: fal queue size [-h] [--output {pretty,json}] [--json] [--team TEAM] app_name

Get queue size for an application.

Positional Arguments:

app_name Application name (do not prefix with owner).

Options:

-h, --help show this help message and exit

--team TEAM The team to use.

Output:

--output {pretty,json}

Modify the command output

--json Output in JSON format (same as --output json)

Flush

Usage: fal queue flush [-h] [--team TEAM] app_name

Flush all pending requests in an application queue.

Positional Arguments:

app_name Application name.

Options:

-h, --help show this help message and exit

--team TEAM The team to use.

fal run

Source: <https://fal.ai/docs/api-reference/cli/run>

```
Usage: fal run [-h] [--team TEAM] [--no-cache] [--app-name APP_NAME]
              [--auth AUTH] [--env ENV] func_ref
```

Run fal function.

Positional Arguments:

func_ref	Function reference. Configure team <code>in</code> pyproject.toml <code>for</code> app names.
----------	---

Options:

-h, --help	show this <code>help</code> message and <code>exit</code>
--team TEAM	The team to use.
--no-cache	Do not use the cache <code>for</code> the environment build.
--app-name APP_NAME	Application name to run with.
--auth AUTH	Application authentication mode (private, public), defaults to public
--env ENV	Target environment (defaults to main).

Examples:

```
fal run path/to/myfile.py::myfunc
fal run path/to/myfile.py::myfunc --env staging
fal run path/to/myfile.py::MyApp --auth private
```

Authentication Modes

The `--auth` flag controls who can access your app while it's running:

- **public** (default): Anyone can call your app without authentication. You pay for all usage.
- **private**: Only you (or your team) can call the app. Requires a valid API key.

By default, `fal run` uses `public` mode for easy testing during development.

fal runners

Source: <https://fal.ai/docs/api-reference/cli/runners>

Usage: fal runners [-h] **command** ...

Manage fal runners.

Options:

-h, --help show this **help** message and **exit**

Commands:

command

stop Stop a runner gracefully.

kill Kill a runner.

list List runners.

logs (log) Show logs **for** a runner.

shell Open an interactive shell **in** a runner.

List

Usage: fal runners list [-h] [--team TEAM] [--since SINCE]
[
--state {all,idle,running,pending,setup,failure_delay,terminated}
--output {pretty,json}] [--json]

List runners.

Options:

-h, --help show this **help** message and **exit**

--team TEAM The team to use.

--since SINCE Show terminated runners since the given time. Accepts '**now**', rela

--state {all,idle,running,pending,setup,failure_delay,terminated} [{all,idle,running,per
Filter by runner state(s). Choose one or more, or '**all**' (default)

Output:

--output {pretty,json}

Modify the **command** output

--json Output **in** JSON **format** (same as --output json)

The runner table displays the following columns:

- **Alias** — Application name
- **Machine Type** — Hardware type (e.g. GPU-A100 , GPU-H100)
- **Runner ID** — Unique runner identifier
- **In Flight Requests** — Number of active requests
- **Expires In** — Time until runner expires
- **Uptime** — How long the runner has been running
- **Revision** — Application revision
- **State** — Current runner state

Logs

```
Usage: fal runners logs [-h] [--output {pretty,json}] [--json] [--team TEAM]
                        [--search SEARCH] [--since SINCE] [--until UNTIL]
                        [--follow] [--lines LINES]
                        id
```

Show logs for a runner.

Positional Arguments:

id Runner ID.

Options:

-h, --help	show this help message and exit
--team TEAM	The team to use.
--search SEARCH	Search for string in logs.
--since SINCE	Show logs since the given time. Accepts 'now', relative like '30m'
--until UNTIL	Show logs until the given time. Accepts 'now', relative like '30m'
--follow, -f	Follow logs live. If --since is not specified, implies '--since 1m'
--lines, -n LINES	Only show latest N log lines. If '+' prefix is used, show oldest N

Output:

--output {pretty,json}	Modify the command output
--json	Output in JSON format (same as --output json)

Stop

Usage: fal runners stop [-h] [--team TEAM] *id*

Stop a runner gracefully.

Positional Arguments:

id Runner ID.

Options:

-h, --help show this *help* message and *exit*
--team TEAM The team to use.

Kill

Usage: fal runners *kill* [-h] [--team TEAM] *id*

Kill a runner.

Positional Arguments:

id Runner ID.

Options:

-h, --help show this *help* message and *exit*
--team TEAM The team to use.

Shell

```
Usage: fal runners shell [-h] [--team TEAM] id
```

Open an interactive shell session inside a running runner.

Positional Arguments:

`id` Runner ID.

Options:

`-h, --help` show this `help` message and `exit`
`--team TEAM` The team to use.

Use Cases

- **Debug running code:** Inspect the runtime environment of a live runner
- **Check dependencies:** Verify installed packages and their versions
- **Examine state:** Inspect files, environment variables, and runtime state
- **Troubleshoot issues:** Diagnose problems in production runners

Example

Connect to a running runner:

```
# Get runner ID from runners list
fal runners list

# Connect to the runner
fal runners shell runner_abc123xyz
```

Once connected, you have full shell access within the runner's container environment. Press `Ctrl+D` or type `exit` to disconnect.

fal secrets

Source: <https://fal.ai/docs/api-reference/cli/secrets>

```
Usage: fal secrets [-h] [--debug] [--pdb] [--cprofile] command ...
```

Manage fal secrets.

Options:

-h, --help show this help message and exit

Commands:

command

set Set a secret.

list List secrets.

unset Unset a secret.

Set

```
Usage: fal secrets set [-h] [--debug] [--pdb] [--cprofile] [--env ENV]
                        NAME=VALUE [NAME=VALUE ...]
```

Set a secret.

Positional Arguments:

NAME=VALUE Secret NAME=VALUE pairs.

Options:

-h, --help show this help message and exit

--env ENV Target environment (defaults to main).

Examples:

```
fal secrets set HF_TOKEN=hf_***
```

```
fal secrets set API_KEY=key123 --env staging
```

List

Usage: fal secrets list [-h] [--debug] [--pdb] [--cprofile] [--env ENV]

List secrets.

Options:

-h, --help show this [help](#) message and [exit](#)
--env ENV Target environment (defaults to main).

Examples:

fal secrets list
fal secrets list --env staging

Unset

Usage: fal secrets [unset](#) [-h] [--debug] [--pdb] [--cprofile] [--env ENV] NAME

Unset a secret.

Positional Arguments:

NAME Secret's name.

Options:

-h, --help show this [help](#) message and [exit](#)
--env ENV Target environment (defaults to main).

Debug:

--debug Show verbose errors.
--pdb Start pdb on error.
--cprofile Show cProfile report.

Examples:

fal secrets [unset](#) HF_TOKEN
fal secrets [unset](#) API_KEY --env staging

fal teams

Source: <https://fal.ai/docs/api-reference/cli/teams>

Manage and switch between teams you belong to. Useful when you're a member of multiple teams and need to deploy or manage apps under a specific team account.

fal teams list

List all teams you belong to:

```
fal teams list
```

Shows your personal account and all team accounts, indicating which is currently active.

fal teams set

Switch to a specific team:

```
fal teams set <account>
```

After switching, all CLI operations (`fal deploy` , `fal apps` , `fal secrets` , etc.) will run under the selected team account.

Arguments:

Argument	Description
<code>account</code>	The team name to switch to

Example:

```
# Switch to your company team
fal teams set my-company

# Deploy under that team
fal deploy my_app.py::MyApp
```

fal teams unset

Switch back to your personal account:

```
fal teams unset
```

You can also use `fal team` as a shorthand alias for `fal teams`.

Dart Client

Source: <https://fal.ai/docs/api-reference/client-libraries/dart/index>

fal client library for Flutter applications

The `fal_client` package provides a Dart interface for calling fal AI models in Flutter applications.

Installation

```
flutter pub add fal_client
```

Quick Start

```
import 'package:fal_client/fal_client.dart';

final fal = FalClient.withCredentials("YOUR_FAL_KEY");

final result = await fal.subscribe("fal-ai/flux/dev", input: {
  "prompt": "a cat",
  "seed": 6252023,
  "image_size": "landscape_4_3",
  "num_images": 4
});

print(result);
```

Supported Platforms

- Flutter (iOS, Android, Web, Desktop)
- Dart (standalone)

API Reference

Full API documentation on [pub.dev](https://fal.ai/docs/api-reference/) Source code and examples Simple Flutter app using fal image inference

Client Libraries

Source: <https://fal.ai/docs/api-reference/client-libraries/index>

Libraries for calling fal AI models from your applications

Use these libraries to call fal AI models from your applications. Available for multiple languages and platforms.

Python client for fal AI models JS/TS client for web and Node.js

Mobile & Other Platforms

iOS, macOS, tvOS, watchOS Android and JVM Flutter apps

auth

Source: <https://fal.ai/docs/api-reference/client-libraries/javascript/auth>

API reference for @fal-ai/client auth

Functions

getTemporaryAuthToken

```
async function getTemporaryAuthToken(app: string, config: RequiredConfig): Promise<string>
```

Get a token to connect to the realtime endpoint.

Parameter	Type	Description
app	string	-
config	RequiredConfig	-

Returns: Promise<string>

Types

TokenProvider

```
type TokenProvider = (app: string) => Promise<string>
```

A function that provides a temporary authentication token.

client

Source: <https://fal.ai/docs/api-reference/client-libraries/javascript/client>

API reference for @fal-ai/client client

Classes & Interfaces

FalClient

```
interface FalClient
```

The main client type, it provides access to simple API model usage, as well as access to the `queue` and `storage` APIs.

Name	Type	Description		

----- `queue`				
`QueueClient`		The queue client to interact with the queue API.	`realtime`	`RealtimeClient`
The realtime client to interact with the realtime API and receive updates in real-time.				
`storage`	`StorageClient`	The storage client to interact with the storage API.	`streaming`	
`StreamingClient` The streaming client to interact with the streaming API. `stream`				
`StreamingClient["stream"]` Calls a fal app that supports streaming and provides a streaming-capable object as a result, that can be used to get partial results through either				
`AsyncIterator` or through an event listener. ##### run				

```
run(endpointId: Id, options: RunOptions<InputType<Id>>): Promise<Result<OutputType<Id>>>
```

Runs a fal endpoint identified by its `endpointId` .

Parameter	Type	Description
<code>endpointId</code>	<code>Id</code>	The endpoint id, e.g. <code>fal-ai/fast-sdxl</code> .
<code>options</code>	<code>RunOptions<InputType<Id>></code>	The request options, including the input payload.

Returns: `Promise<Result<OutputType<Id>>>`

subscribe

```
subscribe(endpointId: Id, options: RunOptions<InputType<Id>> & QueueSubscribeOptions): Pro
```

Subscribes to updates for a specific request in the queue.

Parameter	Type	Description
<code>endpointId</code>	<code>Id</code>	- The ID of the API endpoint.
<code>options</code>	<code>RunOptions<InputType<Id>> & QueueSubscribeOptions</code>	- Options to configure how the request is run and how updates are received.

Returns: `Promise<Result<OutputType<Id>>>`

Functions

createFalClient

```
function createFalClient(userConfig?: Config): FalClient
```

Creates a new reference of the `FalClient` .

Parameter	Type	Description
<code>userConfig</code>	<code>Config</code>	Optional configuration to override the default settings.

Returns: `FalClient`

JavaScript Client

Source: <https://fal.ai/docs/api-reference/client-libraries/javascript/index>

API reference for `@fal-ai/client`

The `@fal-ai/client` package provides a TypeScript/JavaScript interface for calling fal AI models.

Installation

```
npm install @fal-ai/client
```

Quick Start

```
import { fal } from "@fal-ai/client";

const result = await fal.subscribe("fal-ai/flux/dev", {
  input: {
    prompt: "a cat wearing a hat",
    image_size: "landscape_4_3"
  },
  logs: true,
  onQueueUpdate: (status) => console.log(`Status: ${status.status}`)
});

console.log(result.data.images[0].url);
```

API Overview

Method	Description
<code>fal.run()</code>	Run a model synchronously
<code>fal.subscribe()</code>	Run via queue with status updates
<code>fal.stream()</code>	Stream partial results
<code>fal.queue.submit()</code>	Submit to queue, get request ID
<code>fal.queue.status()</code>	Check request status
<code>fal.queue.result()</code>	Get completed result
<code>fal.realtime.connect()</code>	Open realtime WebSocket connection
<code>fal.storage.upload()</code>	Upload file to fal CDN

API Reference

The following pages contain the auto-generated API reference for all public classes and

functions in the `@fal-ai/client` package.

middleware

Source: <https://fal.ai/docs/api-reference/client-libraries/javascript/middleware>

API reference for `@fal-ai/client` middleware

Functions

withMiddleware

```
function withMiddleware(middlewares: RequestMiddleware[]): RequestMiddleware
```

Setup a execution chain of middleware functions.

Parameter	Type	Description
<code>middlewares</code>	<code>RequestMiddleware[]</code>	one or more middleware functions.

Returns: `RequestMiddleware`

withProxy

```
function withProxy(config: RequestProxyConfig): RequestMiddleware
```

Parameter	Type	Description
<code>config</code>	<code>RequestProxyConfig</code>	-

Returns: `RequestMiddleware`

Types

RequestConfig

```
type RequestConfig = {  
  url: string;  
  method: string;  
  headers?: Record<string, string | string[]>;  
}
```

A request configuration object.

Note: This is a simplified version of the `RequestConfig` type from the `fetch` API. It contains only the properties that are relevant for the `fal` client. It also works around the fact that the `fetch` API `Request` does not support mutability, its clone method has critical limitations to our use case.

RequestMiddleware

```
type RequestMiddleware = (  
  request: RequestConfig,  
) => Promise<RequestConfig>
```

RequestProxyConfig

```
type RequestProxyConfig = {  
  targetUrl: string;  
}
```

queue

Source: <https://fal.ai/docs/api-reference/client-libraries/javascript/queue>

Classes & Interfaces

QueueClient

```
interface QueueClient
```

Represents a request queue with methods for submitting requests, checking their status, retrieving results, and subscribing to updates.

submit

```
submit(endpointId: Id, options: SubmitOptions<InputType<Id>>): Promise<InQueueQueueStatus>
```

Submits a request to the queue.

Parameter	Type	Description
endpointId	Id	- The ID of the function web endpoint.
options	SubmitOptions<InputType<Id>>	- Options to configure how the request is run.

Returns: Promise<InQueueQueueStatus>

status

```
status(endpointId: string, options: QueueStatusOptions): Promise<QueueStatus>
```

Retrieves the status of a specific request in the queue.

Parameter	Type	Description
endpointId	string	- The ID of the function web endpoint.

Parameter	Type	Description
<code>options</code>	<code>QueueStatusOptions</code>	- Options to configure how the request is run.

Returns: `Promise<QueueStatus>`

streamStatus

```
streamStatus(endpointId: string, options: QueueStatusStreamOptions): Promise<FalStream<unknown, QueueStatus>>
```

Subscribes to updates for a specific request in the queue using HTTP streaming events.

Parameter	Type	Description
<code>endpointId</code>	<code>string</code>	- The ID of the function web endpoint.
<code>options</code>	<code>QueueStatusStreamOptions</code>	- Options to configure how the request is run and how updates are received.

Returns: `Promise<FalStream<unknown, QueueStatus>>`

subscribeToStatus

```
subscribeToStatus(endpointId: string, options: QueueStatusSubscriptionOptions): Promise<CompletedQueueStatus>
```

Subscribes to updates for a specific request in the queue using polling or streaming.

See `options.mode` for more details.

Parameter	Type	Description
<code>endpointId</code>	<code>string</code>	- The ID of the function web endpoint.
<code>options</code>	<code>QueueStatusSubscriptionOptions</code>	- Options to configure how the request is run and how updates are received.

Returns: `Promise<CompletedQueueStatus>`

result

```
result(endpointId: Id, options: BaseQueueOptions): Promise<Result<OutputType<Id>>>
```

Retrieves the result of a specific request from the queue.

Parameter	Type	Description
endpointId	Id	- The ID of the function web endpoint.
options	BaseQueueOptions	- Options to configure how the request is run.

Returns: Promise<Result<OutputType<Id>>>

cancel

```
cancel(endpointId: string, options: BaseQueueOptions): Promise<void>
```

Cancels a request in the queue.

Parameter	Type	Description
endpointId	string	- The ID of the function web endpoint.
options	BaseQueueOptions	- Options to configure how the request is run and how updates are received.

Returns: Promise<void>

Types

QueuePriority

```
type QueuePriority = "low" | "normal"
```

QueueStatusSubscriptionOptions

```
type QueueStatusSubscriptionOptions = QueueStatusOptions &  
  QueueModeOptions &  
  Omit<QueueCommonSubscribeOptions, "onEnqueue" | "webhookUrl">
```

QueueSubscribeOptions

```
type QueueSubscribeOptions = QueueCommonSubscribeOptions &  
  QueueModeOptions
```

Options for subscribing to the request queue.

SubmitOptions

```

type SubmitOptions = RunOptions<Input> & {
  /**
   * The URL to send a webhook notification to when the request is completed.
   * @see WebHookResponse
   */
  webhookUrl?: string;

  /**
   * The priority of the request. It defaults to `normal`.
   * This will be sent as the `x-fal-queue-priority` header.
   *
   * @see QueuePriority
   */
  priority?: QueuePriority;

  /**
   * A hint for the runner to use when processing the request.
   * This will be sent as the `x-fal-runner-hint` header.
   */
  hint?: string;

  /**
   * Server-side request timeout in seconds. Limits total time spent waiting
   * before processing starts (includes queue wait, retries, and routing).
   * Does not apply once the application begins processing.
   *
   * This will be sent as the `x-fal-request-timeout` header.
   */
  startTimeout?: number;

  /**
   * Additional HTTP headers to include in the submit request.
   *
   * Note: `priority`, `hint`, `startTimeout`, and `objectLifecycle` will override the fo
   * - `x-fal-queue-priority`
   * - `x-fal-runner-hint`
   * - `x-fal-request-timeout`
   * - `x-fal-object-lifecycle-preference`
   */

```

```
headers?: Record<string, string>;  
}
```

Options for submitting a request to the queue.

QueueStatusOptions

```
type QueueStatusOptions = BaseQueueOptions & {  
  /**  
   * If `true`, the response will include the logs for the request.  
   * Defaults to `false`.  
   */  
  logs?: boolean;  
}
```

QueueStatusStreamOptions

```
type QueueStatusStreamOptions = QueueStatusOptions & {  
  /**  
   * The connection mode to use for streaming updates. It defaults to `server`.  
   * Set to `client` if your server proxy doesn't support streaming.  
   */  
  connectionMode?: StreamingConnectionMode;  
}
```

realtime

Source: <https://fal.ai/docs/api-reference/client-libraries/javascript/realtime>

API reference for @fal-ai/client realtime

Classes & Interfaces

RealtimeConnection

```
interface RealtimeConnection
```

A connection object that allows you to `send` request payloads to a realtime endpoint.

send

```
send(input: Input & Partial<WithRequestId>): void
```

Parameter	Type	Description
<code>input</code>	<code>Input & Partial<WithRequestId></code>	-

close

```
close(): void
```

RealtimeConnectionHandler

```
interface RealtimeConnectionHandler
```

Options for connecting to the realtime endpoint.

Name	Type	Description
<code>`connectionKey?`</code>	<code>`string`</code>	The connection key. This is used to reuse the same connection across multiple calls to <code>`connect`</code> . This is particularly useful in contexts where the connection is established as part of a component lifecycle (e.g. React) and the component is re-rendered

multiple times. | | `clientOnly?` | `boolean` | If `true`, the connection will only be established on the client side. This is useful for frameworks that reuse code for both server-side rendering and client-side rendering (e.g. Next.js). This is set to `true` by default when running on React in the server. Otherwise, it is set to `false`. Note that more SSR frameworks might be automatically detected in the future. In the meantime, you can set this to `true` when needed. | | `throttleInterval?` | `number` | The throttle duration in milliseconds. This is used to throttle the calls to the `send` function. Realtime apps usually react to user input, which can be very frequent (e.g. fast typing or mouse/drag movements). The default value is `128` milliseconds. | | `maxBuffering?` | `number` | Configures the maximum amount of frames to store in memory before starting to drop old ones for in favor of the newer ones. It must be between `1` and `60`. The recommended is `2`. The default is `undefined` so it can be determined by the app (normally is set to the recommended setting). | | `path?` | `string` | Optional path to append after the app id. Defaults to `/realtime`. | | `encodeMessage?` | `(input: any) => Uint8Array \| string` | Optional encoder for outgoing messages. Defaults to msgpack. Should return either a `Uint8Array` (binary) or string (text frame). | | `decodeMessage?` | `(data: any) => Promise \| any` | Optional decoder for incoming messages. Defaults to msgpack with JSON support for string payloads. | | `tokenProvider?` | `TokenProvider` | A custom token provider function. When provided, this function will be used to fetch authentication tokens instead of the default internal token fetching mechanism. This is useful when you want to fetch tokens through your own backend proxy. If not provided, the default `getTemporaryAuthToken` will be used. | | `tokenExpirationSeconds?` | `number` | The token expiration time in seconds. This is used to determine when to refresh the token. The token will be refreshed at 90% of this value. Only relevant when using a custom `tokenProvider`. If a custom `tokenProvider` is used without specifying this value, automatic token refresh will be disabled. | ##### onResult

```
onResult(result: Output & WithRequestId): void
```

Callback function that is called when a result is received.

Parameter	Type	Description
<code>result</code>	<code>Output & WithRequestId</code>	- The result of the request.

onError

```
onError(error: ApiError<any>): void
```

Callback function that is called when an error occurs.

Parameter	Type	Description
<code>error</code>	<code>ApiError<any></code>	- The error that occurred.

RealtimeClient

```
interface RealtimeClient
```

connect

```
connect(app: string, handler: RealtimeConnectionHandler<Output>): RealtimeConnection<Input>
```

Connect to the realtime endpoint. The default implementation uses WebSockets to connect to fal function endpoints that support WSS.

Parameter	Type	Description
<code>app</code>	<code>string</code>	the app alias or identifier.
<code>handler</code>	<code>RealtimeConnectionHandler<Output></code>	the connection handler.

Returns: `RealtimeConnection<Input>`

Functions

createRealtimeClient

```
function createRealtimeClient({ config, }: RealtimeClientDependencies): RealtimeClient
```

Parameter	Type	Description
<code>{ config, }</code>	<code>RealtimeClientDependencies</code>	-

Returns: RealtimeClient

response

Source: <https://fal.ai/docs/api-reference/client-libraries/javascript/response>

API reference for @fal-ai/client response

Classes & Interfaces

ApiError

```
interface ApiError
```

| Name | Type | Description | | :----- | :----- | :----- | | `status` | `number` | - | | `body` | `Body` | - | | `requestId` | `string` | - | | `timeoutType` | `string` | - |

ValidationError

```
interface ValidationError
```

getFieldErrors

```
getFieldErrors(field: string): ValidationErrorInfo[]
```

Parameter	Type	Description
field	string	-

Returns: ValidationErrorInfo[]

Functions

defaultResponseHandler

```
async function defaultResponseHandler(response: Response): Promise<Output>
```

Parameter	Type	Description
response	Response	-

Returns: Promise<Output>

resultResponseHandler

```
async function resultResponseHandler(response: Response): Promise<Result<Output>>
```

Parameter	Type	Description
response	Response	-

Returns: Promise<Result<Output>>

Types

ResponseHandler

```
type ResponseHandler = (response: Response) => Promise<Output>
```

ResponseHandlerCreator

```
type ResponseHandlerCreator = (  
  config: RequiredConfig,  
) => ResponseHandler<Output>
```

retry

Source: <https://fal.ai/docs/api-reference/client-libraries/javascript/retry>

API reference for @fal-ai/client retry

Classes & Interfaces

RetryMetrics

```
interface RetryMetrics
```

Retry metrics for tracking retry attempts

Name	Type	Description	:-----	:-----	:-----	`totalAttempts`	`number`	-
`totalDelay`	`number`	-	`lastError?`	`any`	-			

Functions

isRetryableError

```
function isRetryableError(error: any, retryableStatusCodes: number[]): boolean
```

Determines if an error is retryable based on the status code.
User-specified timeouts (504 with X-Fal-Request-Timeout-Type: user) are NOT retryable.

Parameter	Type	Description
<code>error</code>	<code>any</code>	-
<code>retryableStatusCodes</code>	<code>number[]</code>	-

Returns: `boolean`

calculateBackoffDelay

```
function calculateBackoffDelay(attempt: number, baseDelay: number, maxDelay: number, backoffMultiplier: number, enableJitter: boolean): number
```

Calculates the backoff delay for a given attempt using exponential backoff

Parameter	Type	Description
<code>attempt</code>	<code>number</code>	-
<code>baseDelay</code>	<code>number</code>	-
<code>maxDelay</code>	<code>number</code>	-
<code>backoffMultiplier</code>	<code>number</code>	-
<code>enableJitter</code>	<code>boolean</code>	-

Returns: `number`

executeWithRetry

```
async function executeWithRetry(operation: () => Promise<T>, options: RetryOptions, onRetry?: (error: Error) => void): Promise<T>
```

Executes an operation with retry logic and returns both result and metrics

Parameter	Type	Description
<code>operation</code>	<code>() => Promise<T></code>	-
<code>options</code>	<code>RetryOptions</code>	-

Parameter	Type	Description
<code>onRetry</code>	<code>(attempt: number, error: any, delay: number) => void</code>	-

Returns: `Promise<{ result: T; metrics: RetryMetrics }>`

Types

RetryOptions

```
type RetryOptions = {
  maxRetries: number;
  baseDelay: number;
  maxDelay: number;
  backoffMultiplier: number;
  retryableStatusCodes: number[];
  enableJitter: boolean;
}
```

storage

Source: <https://fal.ai/docs/api-reference/client-libraries/javascript/storage>

API reference for @fal-ai/client storage

Classes & Interfaces

StorageSettings

```
interface StorageSettings
```

Configuration for object lifecycle and storage behavior.

Parameter	Type	Description
lifecycle	StorageSettings	the lifecycle preference

Returns: number | undefined

buildObjectLifecycleHeaders

```
function buildObjectLifecycleHeaders(lifecycle: StorageSettings | undefined): Record<string, string>
```

Builds the headers for the Object Lifecycle preference to be used in API requests. This is used by the queue and run APIs to control the lifecycle of generated objects.

Parameter	Type	Description
lifecycle	StorageSettings undefined	the lifecycle preference

Returns: Record<string, string>

createStorageClient

```
function createStorageClient({ config, }: StorageClientDependencies): StorageClient
```

Parameter	Type	Description
{ config, }	StorageClientDependencies	-

Returns: StorageClient

Types

UploadOptions

```
type UploadOptions = {
  /**
   * Custom lifecycle configuration for the uploaded file.
   * This object will be sent as the X-Fal-Object-Lifecycle header.
   */
  lifecycle?: StorageSettings;
}
```

Options for uploading a file.

streaming

Source: <https://fal.ai/docs/api-reference/client-libraries/javascript/streaming>

API reference for @fal-ai/client streaming

Classes & Interfaces

FalStream

```
interface FalStream
```

The class representing a streaming response. With t

Name	Type	Description	-----	-----

----- `config` `RequiredConfig` -				
`endpointId`	`string`	-	`url`	`string` -
		`options`	`StreamOptions`	<div></div>
		`listeners`	`Map`	-
		`buffer`	`Output[]`	-
		`currentData`	`Output`	undefined` -

``lastEventTimestamp` | `any` | - || `streamClosed` | `any` | - || `_requestId` | `string` | `null` | - || `donePromise` | `Promise` | - || `abortController` | `any` | - || `start` | `any` | - || `handleResponse` | `any` | - || `handleError` | `any` | - || `on` | `any` | - || `emit` | `any` | - || `done` | `any`` | Gets a reference to the ``Promise`` that indicates whether the streaming is done or not. Developers should always call this in their apps to ensure the request is over. An alternative to this, is to use ``on('done')`` in case your application architecture works best with event listeners. || ``abort` | `any`` | Aborts the streaming request. **Note:** This method is noop in case the request is already done. | ##### ``[Symbol.asyncIterator]``

```
async [Symbol.asyncIterator](): any
```

Returns: `any`

StreamingClient

```
interface StreamingClient
```

The streaming client interface.

stream

```
stream(endpointId: Id, options: StreamOptions<InputType<Id>>): Promise<FalStream<InputType
```

Calls a fal app that supports streaming and provides a streaming-capable object as a result, that can be used to get partial results through either `AsyncIterator` or through an event listener.

Parameter	Type	Description
<code>endpointId</code>	<code>Id</code>	the endpoint id, e.g. <code>fal-ai/llavav15-13b</code> .
<code>options</code>	<code>StreamOptions<InputType<Id>></code>	the request options, including the input payload.

Returns: `Promise<FalStream<InputType<Id>, OutputType<Id>>>`

Functions

createStreamingClient

```
function createStreamingClient({ config, storage, }: StreamingClientDependencies): StreamingClient
```

Parameter	Type	Description
{ config, storage, }	StreamingClientDependencies	-

Returns: StreamingClient

Types

StreamingConnectionMode

```
type StreamingConnectionMode = "client" | "server"
```

StreamOptions

```
type StreamOptions = {
  /**
   * The endpoint URL. If not provided, it will be generated from the
   * `endpointId` and the `queryParams`.
   */
  readonly url?: string;

  /**
   * The API input payload.
   */
  readonly input?: Input;

  /**
   * The query parameters to be sent with the request.
   */
  readonly queryParams?: Record<string, string>;

  /**
   * The maximum time interval in milliseconds between stream chunks. Defaults to 15s.
   */
  readonly timeout?: number;

  /**
   * Whether it should auto-upload File-like types to fal's storage
   * or not.
   */
  readonly autoUpload?: boolean;

  /**
   * The HTTP method, defaults to `post`;
   */
  readonly method?: "get" | "post" | "put" | "delete" | string;

  /**
   * The content type the client accepts as response.
   * By default this is set to `text/event-stream`.
   */
  readonly accept?: string;
```

```

/**
 * The streaming connection mode. This is used to determine
 * whether the streaming will be done from the browser itself (client)
 * or through your own server, either when running on NodeJS or when
 * using a proxy that supports streaming.
 *
 * It defaults to `server`. Set to `client` if your server proxy doesn't
 * support streaming.
 */
readonly connectionMode?: StreamingConnectionMode;

/**
 * The signal to abort the request.
 */
readonly signal?: AbortSignal;

/**
 * A custom token provider function. Only used when `connectionMode` is `"client"`.
 * When provided, this function will be used to fetch authentication tokens
 * instead of the default internal token fetching mechanism.
 */
readonly tokenProvider?: TokenProvider;
}

```

The stream API options. It requires the API input and also offers configuration options.

types.client

Source: <https://fal.ai/docs/api-reference/client-libraries/javascript/types.client>

API reference for @fal-ai/client types.client

Types

EndpointType

```
type EndpointType = keyof EndpointTypeMap | (string & {})
```

InputType

```
type InputType = T extends keyof EndpointTypeMap  
  ? EndpointTypeMap[T]["input"]  
  : Record<string, any>
```

OutputType

```
type OutputType = T extends keyof EndpointTypeMap  
  ? EndpointTypeMap[T]["output"]  
  : any
```

types.common

Source: <https://fal.ai/docs/api-reference/client-libraries/javascript/types.common>

API reference for @fal-ai/client types.common

Classes & Interfaces

InQueueQueueStatus

```
interface InQueueQueueStatus
```

Name	Type	Description	:-----	:-----	:-----	`status`	`IN_QUEUE`
-	`queue_position`	`number`	-	-	-	-	-

InProgressQueueStatus

```
interface InProgressQueueStatus
```

| Name | Type | Description | | :----- | :----- | :----- | | `status` | `"IN_PROGRESS"` |
- | | `logs` | `RequestLog[]` | - |

CompletedQueueStatus

```
interface CompletedQueueStatus
```

| Name | Type | Description | | :----- | :----- | :----- | | `status` | `"COMPLETED"` | -
| | `logs` | `RequestLog[]` | - | | `metrics?` | `Metrics` | - |

Functions

isQueueStatus

```
function isQueueStatus(obj: any): obj is QueueStatus
```

Parameter	Type	Description
obj	any	-

Returns: obj is QueueStatus

isCompletedQueueStatus

```
function isCompletedQueueStatus(obj: any): obj is CompletedQueueStatus
```

Parameter	Type	Description
<code>obj</code>	<code>any</code>	-

Returns: `obj` is `CompletedQueueStatus`

Types

Result

```
type Result = {  
  data: T;  
  requestId: string;  
}
```

Represents an API result, containing the data,
the request ID and any other relevant information.

RunOptions

```
type RunOptions = {  
  /**  
   * The function input. It will be submitted either as query params  
   * or the body payload, depending on the `method`.  
   */  
  readonly input?: Input;  
  
  /**  
   * The HTTP method, defaults to `post`;  
   */  
  readonly method?: "get" | "post" | "put" | "delete" | string;  
  
  /**  
   * The abort signal to cancel the request.  
   */  
  readonly abortSignal?: AbortSignal;  
  
  /**  
   * Object lifecycle configuration for controlling how long generated objects  
   * (images, files, etc.) remain available before expiring.  
   *  
   * @see StorageSettings  
   * @see https://docs.fal.ai/model-apis/model-endpoints/queue#object-lifecycle  
   */  
  readonly storageSettings?: StorageSettings;  
  
  /**  
   * Server-side request timeout in seconds. Limits total time spent waiting  
   * before processing starts (includes queue wait, retries, and routing).  
   * Does not apply once the application begins processing.  
   *  
   * This will be sent as the `x-fal-request-timeout` header.  
   */  
  readonly startTimeout?: number;  
}
```

The function input and other configuration when running

the function, such as the HTTP method to use.

UrlOptions

```
type UrlOptions = {  
    /**  
     * If `true`, the function will use the queue to run the function  
     * asynchronously and return the result in a separate call. This  
     * influences how the URL is built.  
     */  
    readonly subdomain?: string;  
  
    /**  
     * The query parameters to include in the URL.  
     */  
    readonly query?: Record<string, string>;  
  
    /**  
     * The path to append to the function URL.  
     */  
    path?: string;  
}
```

RequestLog

```
type RequestLog = {  
    message: string;  
    level: "STDERR" | "STDOUT" | "ERROR" | "INFO" | "WARN" | "DEBUG";  
    source: "USER";  
    timestamp: string; // Using string to represent date-time format, but you could also use  
}
```

Metrics

```
type Metrics = {  
  inference_time: number | null;  
}
```

QueueStatus

```
type QueueStatus = | InProgressQueueStatus  
  | CompletedQueueStatus  
  | InQueueQueueStatus
```

ValidationErrorInfo

```
type ValidationErrorInfo = {  
  msg: string;  
  loc: Array<string | number>;  
  type: string;  
}
```

WebHookResponse

```
type WebHookResponse = | {  
    /** Indicates a successful response. */  
    status: "OK";  
    /** The payload of the response, structure determined by the Payload type. */  
    payload: Payload;  
    /** Error is never present in a successful response. */  
    error: never;  
    /** The unique identifier for the request. */  
    request_id: string;  
}  
| {  
    /** Indicates an unsuccessful response. */  
    status: "ERROR";  
    /** The payload of the response, structure determined by the Payload type. */  
    payload: Payload;  
    /** Description of the error that occurred. */  
    error: string;  
    /** The unique identifier for the request. */  
    request_id: string;  
}
```

Represents the response from a WebHook request.

This is a union type that varies based on the `status` property.

utils

Source: <https://fal.ai/docs/api-reference/client-libraries/javascript/utils>

API reference for @fal-ai/client utils

Functions

ensureEndpointIdFormat

```
function ensureEndpointIdFormat(id: string): string
```

Parameter	Type	Description
id	string	-

Returns: string

parseEndpointId

```
function parseEndpointId(id: string): EndpointId
```

Parameter	Type	Description
id	string	-

Returns: EndpointId

isValidUrl

```
function isValidUrl(url: string): any
```

Parameter	Type	Description
url	string	-

Returns: any

throttle

```
function throttle(func: T, limit: number, leading?: any): (...funcArgs: Parameters<T>) =>
```

Parameter	Type	Description
func	T	-
limit	number	-
leading	any	-

Returns: (...funcArgs: Parameters<T>) => ReturnType<T> | void

isReact

```
function isReact(): any
```

Not really the most optimal way to detect if we're running in React, but the idea here is that we can support multiple rendering engines (starting with React), with all their peculiarities, without having to add a dependency or creating custom integrations (e.g. custom hooks).

Yes, a bit of magic to make things works out-of-the-box.

Returns: any

isPlainObject

```
function isPlainObject(value: any): boolean
```

Check if a value is a plain object.

Parameter	Type	Description
value	any	- The value to check.

Returns: `boolean`

sleep

```
async function sleep(ms: number): Promise<void>
```

Utility function to sleep for a given number of milliseconds

Parameter	Type	Description
<code>ms</code>	<code>number</code>	-

Returns: `Promise<void>`

Types

EndpointId

```
type EndpointId = {  
  readonly owner: string;  
  readonly alias: string;  
  readonly path?: string;  
  readonly namespace?: EndpointNamespace;  
}
```

Kotlin / Java Client

Source: <https://fal.ai/docs/api-reference/client-libraries/kotlin/index>

fal client library for Android and JVM applications

The `fal-client` packages provide Kotlin and Java interfaces for calling fal AI models on Android and JVM platforms.

Installation

```
```groovy Gradle (Kotlin) theme={null} implementation 'ai.fal.client:fal-client-kotlin:0.7.1' ```
```

```
implementation 'ai.fal.client:fal-client:0.7.1'
```

```
<dependency>
 <groupId>ai.fal.client</groupId>
 <artifactId>fal-client-kotlin</artifactId>
 <version>0.7.1</version>
</dependency>
```

```
<dependency>
 <groupId>ai.fal.client</groupId>
 <artifactId>fal-client</artifactId>
 <version>0.7.1</version>
</dependency>
```

**\*\*Java Async Support\*\***

If your code relies on asynchronous operations via `CompletableFuture` or `Future`, use the `ai.fal.client:fal-client-async` artifact instead.

## Quick Start

```
```kotlin Kotlin theme={null} import ai.fal.client.kt
```

```
val fal = createFalClient()
```

```
val input = mapOf<String, Any>(
```

```
  "prompt" to "a cat",
```

```
  "seed" to 6252023,
```

```
  "image_size" to "landscape_4_3",
```

```
  "num_images" to 4
```

```
)
```

```
val result = fal.subscribe("fal-ai/flux/dev", input, options = SubscribeOptions(
```

```
logs = true
```



```

)) { update ->
if (update is QueueStatus.InProgress) {
println(update.logs)
}
}

```

```

```java Java theme={null}
import ai.fal.client.*;
import ai.fal.client.queue.*;

var fal = FalClient.withEnvCredentials();

var input = Map.of(
 "prompt", "a cat",
 "seed", 6252023,
 "image_size", "landscape_4_3",
 "num_images", 4
);
var result = fal.subscribe("fal-ai/flux/dev",
 SubscribeOptions.<JsonObject>builder()
 .input(input)
 .logs(true)
 .resultType(JsonObject.class)
 .onQueueUpdate(update -> {
 if (update instanceof QueueStatus.InProgress) {
 System.out.println(((QueueStatus.InProgress) update).getLogs());
 }
 })
 .build()
);

```

## Supported Platforms

- Android (API 21+)
- JVM (Java 11+)
- Kotlin Multiplatform (JVM target)

# API Reference

[JavaDoc documentation](#) [KDoc documentation](#) [Source code and examples](#)

## **fal\_client**

Source: [https://fal.ai/docs/api-reference/client-libraries/python/fal\\_client](https://fal.ai/docs/api-reference/client-libraries/python/fal_client)

API reference for fal\_client

```
from fal_client import (
 SyncClient,
 AsyncClient,
 RealtimeConnection,
 AsyncRealtimeConnection,
 Status,
 Queued,
 InProgress,
 Completed,
 SyncRequestHandle,
 AsyncRequestHandle,
 run,
 subscribe_async,
 subscribe,
 submit,
 stream,
 run_async,
 submit_async,
 stream_async,
 realtime,
 realtime_async,
 cancel,
 cancel_async,
 status,
 status_async,
 result,
 result_async,
 encode,
 encode_file,
 encode_image,
)
```

## Classes

### SyncClient

```
class fal_client.SyncClient
```

```
| Name | Type | Default | Description | | :----- | :----- | :----- | :----- | | `key` | `str` | None | None | - | | `default_timeout` | float | 120.0 | - | | Name | Type | Default | Description | | :----- | :----- | :----- | :----- | | `key` | `str` | None | None | - | | `default_timeout` | float | 120.0 | - | | Name | Type | Description | | :----- | :----- | :----- | | `_executor` | ThreadPoolExecutor | - | #### cancel
```

```
def cancel(self, application: 'str', request_id: 'str') -> 'None'
```

Parameter	Type	Default	Description
application	str	-	-
request_id	str	-	-

**Returns:** NoneType

## get\_handle

```
def get_handle(self, application: 'str', request_id: 'str') -> 'SyncRequestHandle'
```

Parameter	Type	Default	Description
application	str	-	-
request_id	str	-	-

**Returns:** SyncRequestHandle

## realtime

```
def realtime(self, application: 'str', *, use_jwt: 'bool' = True, path: 'str' = '/realtime')
```

Parameter	Type	Default	Description
application	str	-	-
use_jwt	bool	True	-
path	str	'/realtime'	-

Parameter	Type	Default	Description
<code>max_buffering</code>	<code>int   None</code>	<code>None</code>	-
<code>token_expiration</code>	<code>int</code>	<code>120</code>	-
<code>encode_message</code>	<code>Optional[Callable[Any, bytes]]</code>	<code>None</code>	-
<code>decode_message</code>	<code>Optional[Callable[bytes, Any]]</code>	<code>None</code>	-

**Returns:** `Iterator[RealtimeConnection]`

## result

```
def result(self, application: 'str', request_id: 'str') -> 'AnyJSON'
```

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>request_id</code>	<code>str</code>	-	-

**Returns:** `dict[str, Any]`

## run

```
def run(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '', timeout: 'Optional[int]'
```

Run an application with the given arguments (which will be JSON serialized).

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-
<code>path</code>	<code>str</code>	<code>''</code>	-
<code>timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Client-side HTTP timeout in seconds. Controls how long

Parameter	Type	Default	Description
			the HTTP client waits for a response. Defaults to the client's default_timeout.
<code>start_timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Server-side request timeout in seconds. Limits total time spent waiting before processing starts. Does not apply once the application begins processing.
<code>hint</code>	<code>str   None</code>	<code>None</code>	-
<code>headers</code>	<code>dict[str, str]</code>	<code>\{\}</code>	-

**Returns:** `dict[str, Any]`

## status

```
def status(self, application: 'str', request_id: 'str', *, with_logs: 'bool' = False) ->
```

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>request_id</code>	<code>str</code>	-	-
<code>with_logs</code>	<code>bool</code>	<code>False</code>	-

**Returns:** `Status`

## stream

```
def stream(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '/stream', ti
```

Stream the output of an application with the given arguments (which will be JSON serialized). This is only supported at a few select applications at the moment, so be sure to first consult

with the documentation of individual applications to see if this is supported.

The function will iterate over each event that is streamed from the server.

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-
<code>path</code>	<code>str</code>	<code>'/stream'</code>	-
<code>timeout</code>	<code>float   None</code>	<code>None</code>	-

**Returns:** `Iterator[dict[str, Any]]`

## submit

```
def submit(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '', hint: 'st
```

Submit an application with the given arguments (which will be JSON serialized).

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-
<code>path</code>	<code>str</code>	<code>''</code>	-
<code>hint</code>	<code>str   None</code>	<code>None</code>	-
<code>webhook_url</code>	<code>str   None</code>	<code>None</code>	-
<code>priority</code>	<code>Optional[Literal[normal, low]]</code>	<code>None</code>	-
<code>headers</code>	<code>dict[str, str]</code>	<code>\{\}</code>	-
<code>start_timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Server-side request timeout in seconds. Limits

Parameter	Type	Default	Description
			total time spent waiting before processing starts (includes queue wait, retries, and routing). Does not apply once the application begins processing.

**Returns:** `SyncRequestHandle`

## subscribe

```
def subscribe(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '', hint:
```

Subscribe to an application and wait for the result.

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-
<code>path</code>	<code>str</code>	<code>''</code>	-
<code>hint</code>	<code>str   None</code>	<code>None</code>	-
<code>with_logs</code>	<code>bool</code>	<code>False</code>	-
<code>on_enqueue</code>	<code>Optional[Callable[str, NoneType]]</code>	<code>None</code>	-
<code>on_queue_update</code>	<code>Optional[Callable[Status, NoneType]]</code>	<code>None</code>	-
<code>priority</code>	<code>Optional[Literal[normal, low]]</code>	<code>None</code>	-
<code>headers</code>	<code>dict[str, str]</code>	<code>\{\}</code>	-



Parameter	Type	Default	Description
<code>start_timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Server-side request timeout in seconds. Limits total time spent waiting before processing starts (includes queue wait, retries, and routing). Does not apply once the application begins processing.
<code>client_timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Client-side total timeout in seconds. Limits the total time spent waiting for the entire request to complete (including queue wait

Parameter	Type	Default	Description
			and processing). If not set, waits indefinitely.

**Returns:** `dict[str, Any]`

## upload

```
def upload(self, data: 'str | bytes', content_type: 'str', file_name: 'str | None' = None,
```

Upload the given data blob to the CDN and return the access URL. The content type should be specified as the second argument. Use `upload_file` or `upload_image` for convenience.

Parameter	Type
<code>data</code>	<code>str   bytes</code>
<code>content_type</code>	<code>str</code>
<code>file_name</code>	<code>str   None</code>
<code>repository</code>	<code>Optional[Literal[fal_v3, cdn, fal]]</code>
<code>fallback_repository</code>	<code>Literal[fal_v3, cdn, fal]   list[Literal[fal_v3, cdn, fal]]   None</code>

**Returns:** `str`

## upload\_file

```
def upload_file(self, path: 'os.PathLike', *, repository: 'UploadRepositoryId | None' = None,
```

Upload a file from the local filesystem to the CDN and return the access URL.

Parameter	Type
<code>path</code>	<code>PathLike</code>
<code>repository</code>	<code>Optional[Literal[fal_v3, cdn, fal]]</code>
<code>fallback_repository</code>	<code>Literal[fal_v3, cdn, fal]   list[Literal[fal_v3, cdn, fal]]   None</code>

**Returns:** `str`

## upload\_image

```
def upload_image(self, image: 'Image.Image', format: 'str' = 'jpeg', *, repository: 'UploadRepositoryId' = None, fallback_repository: 'UploadRepositoryId' = None) -> str
```

Upload a pillow image object to the CDN and return the access URL.

Parameter	Type	Default
<code>image</code>	<code>Image.Image</code>	-
<code>format</code>	<code>str</code>	<code>'jpeg'</code>
<code>repository</code>	<code>UploadRepositoryId   None</code>	<code>None</code>
<code>fallback_repository</code>	<code>UploadRepositoryId   list[UploadRepositoryId]   None</code>	<code>None</code>

## ws\_connect

```
def ws_connect(self, application: 'str', *, use_jwt: 'bool' = True, path: 'str' = '/', max_buffering: 'int' = None) -> str
```

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>use_jwt</code>	<code>bool</code>	<code>True</code>	-
<code>path</code>	<code>str</code>	<code>'/'</code>	-
<code>max_buffering</code>	<code>int   None</code>	<code>None</code>	-

Parameter	Type	Default	Description
token_expiration	int	120	-

## AsyncClient

```
class fal_client.AsyncClient
```

```
| Name | Type | Default | Description | | :----- | :----- | :----- | :----- | | `key` |
`str` \ `None` \ `None` | - | | `default_timeout` \ `float` \ `120.0` | - | | Name | Type | Default |
Description | | :----- | :----- | :----- | :----- | | `key` | `str` \ `None` \ `None` | - | |
`default_timeout` \ `float` \ `120.0` | - | ##### cancel
```

```
async def cancel(self, application: 'str', request_id: 'str') -> 'None'
```

Parameter	Type	Default	Description
application	str	-	-
request_id	str	-	-

**Returns:** `NoneType`

## get\_handle

```
def get_handle(self, application: 'str', request_id: 'str') -> 'AsyncRequestHandle'
```

Parameter	Type	Default	Description
application	str	-	-
request_id	str	-	-

**Returns:** `AsyncRequestHandle`

## realtime

```
def realtime(self, application: 'str', *, use_jwt: 'bool' = True, path: 'str' = '/realtime')
```

Parameter	Type	Default	Description
application	str	-	-
use_jwt	bool	True	-
path	str	'/realtime'	-
max_buffering	int   None	None	-
token_expiration	int	120	-
encode_message	Optional[Callable[Any, bytes]]	None	-
decode_message	Optional[Callable[bytes, Any]]	None	-

**Returns:** AsyncIterator[AsyncRealtimeConnection]

## result

```
async def result(self, application: 'str', request_id: 'str') -> 'AnyJSON'
```

Parameter	Type	Default	Description
application	str	-	-
request_id	str	-	-

**Returns:** dict[str, Any]

## run

```
async def run(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '', timeout:
```

Run an application with the given arguments (which will be JSON serialized). The path parameter can be used to specify a subpath when applicable. This method will return the

result of the inference call directly.

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-
<code>path</code>	<code>str</code>	<code>''</code>	-
<code>timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Client-side HTTP timeout in seconds. Controls how long the HTTP client waits for a response. Defaults to the client's <code>default_timeout</code> .
<code>start_timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Server-side request timeout in seconds. Limits total time spent waiting before processing starts. Does not apply once the application begins processing.
<code>hint</code>	<code>str   None</code>	<code>None</code>	-
<code>headers</code>	<code>dict[str, str]</code>	<code>\{\}</code>	-

**Returns:** `dict[str, Any]`

## status

```
async def status(self, application: 'str', request_id: 'str', *, with_logs: 'bool' = False
```

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>request_id</code>	<code>str</code>	-	-
<code>with_logs</code>	<code>bool</code>	<code>False</code>	-

**Returns:** `Status`

## stream

```
def stream(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '/stream', timeout: 'float | None' = None) -> AsyncIterator[dict[str, Any]]
```

Stream the output of an application with the given arguments (which will be JSON serialized). This is only supported at a few select applications at the moment, so be sure to first consult with the documentation of individual applications to see if this is supported.

The function will iterate over each event that is streamed from the server.

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-
<code>path</code>	<code>str</code>	<code>'/stream'</code>	-
<code>timeout</code>	<code>float   None</code>	<code>None</code>	-

**Returns:** `AsyncIterator[dict[str, Any]]`

## submit

```
async def submit(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '', timeout: 'float | None' = None) -> AsyncIterator[dict[str, Any]]
```

Submit an application with the given arguments (which will be JSON serialized). The path parameter can be used to specify a subpath when applicable. This method will return a handle to the request that can be used to check the status and retrieve the result of the inference call when it is done.

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-

Parameter	Type	Default	Description
<code>path</code>	<code>str</code>	<code>''</code>	-
<code>hint</code>	<code>str   None</code>	<code>None</code>	-
<code>webhook_url</code>	<code>str   None</code>	<code>None</code>	-
<code>priority</code>	<code>Optional[Literal[normal, low]]</code>	<code>None</code>	-
<code>headers</code>	<code>dict[str, str]</code>	<code>\{\}</code>	-
<code>start_timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Server-side request timeout in seconds. Limits total time spent waiting before processing starts (includes queue wait, retries, and routing). Does not apply once the application begins processing.

**Returns:** `AsyncRequestHandle`

## subscribe

```
async def subscribe(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '',
```

Subscribe to an application and wait for the result.

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-



Parameter	Type	Default	Description
path	str	''	-
hint	str   None	None	-
with_logs	bool	False	-
on_enqueue	Optional[Callable[str, NoneType]]	None	-
on_queue_update	Optional[Callable[Status, NoneType]]	None	-
priority	Optional[Literal[normal, low]]	None	-
headers	dict[str, str]	\{\}	-
start_timeout	int   float   NoneType	None	Server-side request timeout in seconds. Limits total time spent waiting before processing starts (includes queue wait, retries, and routing). Does not apply once the application begins processing.
client_timeout	int   float   NoneType	None	Client-side total

Parameter	Type	Default	Description
			timeout in seconds. Limits the total time spent waiting for the entire request to complete (including queue wait and processing). If not set, waits indefinitely.

**Returns:** `dict[str, Any]`

## upload

```
async def upload(self, data: 'str | bytes', content_type: 'str', file_name: 'str | None' =
```

Upload the given data blob to the CDN and return the access URL. The content type should be specified as the second argument. Use `upload_file` or `upload_image` for convenience.

Parameter	Type
<code>data</code>	<code>str   bytes</code>
<code>content_type</code>	<code>str</code>
<code>file_name</code>	<code>str   None</code>
<code>repository</code>	<code>Optional[Literal[fal_v3, cdn, fal]]</code>

Parameter	Type
<code>fallback_repository</code>	<code>Literal[fal_v3, cdn, fal]   list[Literal[fal_v3, cdn, fal]]   None</code>

**Returns:** `str`

### upload\_file

```
async def upload_file(self, path: 'os.PathLike', *, repository: 'UploadRepositoryId | None'
```

Upload a file from the local filesystem to the CDN and return the access URL.

Parameter	Type
<code>path</code>	<code>PathLike</code>
<code>repository</code>	<code>Optional[Literal[fal_v3, cdn, fal]]</code>
<code>fallback_repository</code>	<code>Literal[fal_v3, cdn, fal]   list[Literal[fal_v3, cdn, fal]]   None</code>

**Returns:** `str`

### upload\_image

```
async def upload_image(self, image: 'Image.Image', format: 'str' = 'jpeg', *, repository:
```

Upload a pillow image object to the CDN and return the access URL.

Parameter	Type	Default
<code>image</code>	<code>Image.Image</code>	-
<code>format</code>	<code>str</code>	<code>'jpeg'</code>
<code>repository</code>	<code>UploadRepositoryId   None</code>	<code>None</code>
<code>fallback_repository</code>	<code>UploadRepositoryId   list[UploadRepositoryId]   None</code>	<code>None</code>

## ws\_connect

```
def ws_connect(self, application: 'str', *, use_jwt: 'bool' = True, path: 'str' = '', max_
```

Parameter	Type	Default	Description
application	str	-	-
use_jwt	bool	True	-
path	str	''	-
max_buffering	int   None	None	-
token_expiration	int	120	-

## RealtimeConnection

```
class fal_client.RealtimeConnection
```

Synchronous realtime connection wrapper.

```
| Name | Type | Default | Description | | :----- | :----- | :----- |
:----- | | `_ws` | `Connection` | - | - | | `_encode_message` | `Callable[[Any], bytes]` \ | None` |
`None` | - | | `_decode_message` | `Callable[[bytes], Any]` \ | None` | `None` | - | ##### close
```

```
def close(self) -> 'None'
```

**Returns:** `NoneType`

## recv

```
def recv(self) -> 'dict[str, Any] | None'
```

**Returns:** `dict[str, Any] | None`

## send

```
def send(self, arguments: 'dict[str, Any]') -> 'None'
```

Parameter	Type	Default	Description
arguments	dict[str, Any]	-	-

**Returns:** NoneType

## AsyncRealtimeConnection

```
class fal_client.AsyncRealtimeConnection
```

Asynchronous realtime connection wrapper.

```
| Name | Type | Default | Description | | :----- | :----- | :----- |
| :----- | | _ws | 'WebSocketClientProtocol' | - | | | _encode_message | Callable[[Any],
bytes] \ | None | 'None' | - | | | _decode_message | Callable[[bytes], Any] \ | None | 'None' | - |
close
```

```
async def close(self) -> 'None'
```

**Returns:** NoneType

## recv

```
async def recv(self) -> 'dict[str, Any] | None'
```

**Returns:** dict[str, Any] | None

## send

```
async def send(self, arguments: 'dict[str, Any]') -> 'None'
```

Parameter	Type	Default	Description
<code>arguments</code>	<code>dict[str, Any]</code>	-	-

**Returns:** `NoneType`

## Status

```
class fal_client.Status
```

## Queued

```
class fal_client.Queued
```

Indicates the request is enqueued and waiting to be processed. The position field indicates the relative position in the queue (0-indexed).

**Inherits from:** `Status`

Name   Type   Default   Description	:-----   :----   :-----   :-----	`position`   `int`   -   -
Name   Type   Default   Description	:-----   :----   :-----   :-----	`position`   `int`   -
-		

## InProgress

```
class fal_client.InProgress
```

Indicates the request is currently being processed. If the status operation called with the `with_logs` parameter set to True, the logs field will be a list of log objects.

**Inherits from:** `Status`

Name   Type   Default   Description	:-----   :-----   :-----   :-----	`logs`   `list[dict[str, Any]] \ None`   -   -
Name   Type   Default   Description	:-----   :-----   :-----   :-----	`logs`   `list[dict[str, Any]] \ None`   -   -

# Completed

```
class fal_client.Completed
```

Indicates the request has been completed and the result can be gathered. The logs field will contain the logs if the status operation was called with the `with_logs` parameter set to True.

## Metrics

might contain the inference time, and other internal metadata (number of tokens processed, etc.).

**Inherits from:** Status

Name	Type	Default	Description
logs	list[dict[str, Any]]	None	metrics dict[str, Any]

## SyncRequestHandle

```
class fal_client.SyncRequestHandle
```

**Inherits from:** `BaseRequestHandle`

```
| Name | Type | Default | Description | | :----- | :----- | :----- | :----- | | `request_id` |
`str` | - | - | | `response_url` | `str` | - | - | | `status_url` | `str` | - | - | | `cancel_url` | `str` | - | - | | `client`
`Client` | - | - | | Name | Type | Default | Description | | :----- | :----- | :----- | :----- |
`client` | `httpx.Client` | - | - | #### cancel
```

```
def cancel(self) -> 'None'
```

Cancel the request.

**Returns:** `NoneType`

## from\_request\_id

```
def from_request_id(cls, client: 'httpx.Client', application: 'str', request_id: 'str') ->
```

Parameter	Type	Default	Description
client	Client	-	-
application	str	-	-
request_id	str	-	-

**Returns:** SyncRequestHandle

## get

```
def get(self) -> 'AnyJSON'
```

Wait till the request is completed and return the result of the inference call.

**Returns:** dict[str, Any]

## iter\_events

```
def iter_events(self, *, with_logs: 'bool' = False, interval: 'float' = 0.1) -> 'Iterator[
```

Continuously poll for the status of the request and yield it at each interval till the request is completed. If `with_logs` is True, logs will be included in the response.

Parameter	Type	Default	Description
with_logs	bool	False	-
interval	float	0.1	-

**Returns:** Iterator[Status]



# status

```
def status(self, *, with_logs: 'bool' = False) -> 'Status'
```

Returns the status of the request (which can be one of the following: Queued, InProgress, Completed). If `with_logs` is True, logs will be included for InProgress and Completed statuses.

Parameter	Type	Default	Description
<code>with_logs</code>	<code>bool</code>	<code>False</code>	-

Returns: `Status`

# AsyncRequestHandle

```
class fal_client.AsyncRequestHandle
```

Inherits from: `_BaseRequestHandle`

Name	Type	Default	Description						
<code>`request_id`</code>	<code>`str`</code>	-	-	-	-	-	-	-	-
<code>`response_url`</code>	<code>`str`</code>	-	-	-	-	-	-	-	-
<code>`status_url`</code>	<code>`str`</code>	-	-	-	-	-	-	-	-
<code>`cancel_url`</code>	<code>`str`</code>	-	-	-	-	-	-	-	-
<code>`client`</code>	<code>`AsyncClient`</code>	-	-	-	-	-	-	-	-
Name   Type   Default   Description     :-----   :-----   :-----   :-----									
:-----   :-----   :-----   :-----									
`client`   `httpx.AsyncClient`   -   -   ##### cancel									

```
async def cancel(self) -> 'None'
```

Cancel the request.

Returns: `NoneType`

# from\_request\_id

```
def from_request_id(cls, client: 'httpx.AsyncClient', application: 'str', request_id: 'st
```

Parameter	Type	Default	Description
<code>client</code>	<code>AsyncClient</code>	-	-
<code>application</code>	<code>str</code>	-	-
<code>request_id</code>	<code>str</code>	-	-

**Returns:** `AsyncRequestHandle`

## get

```
async def get(self) -> 'AnyJSON'
```

Wait till the request is completed and return the result.

**Returns:** `dict[str, Any]`

## iter\_events

```
def iter_events(self, *, with_logs: 'bool' = False, interval: 'float' = 0.1) -> 'AsyncIterator[Status]'
```

Continuously poll for the status of the request and yield it at each interval till the request is completed. If `with_logs` is True, logs will be included in the response.

Parameter	Type	Default	Description
<code>with_logs</code>	<code>bool</code>	<code>False</code>	-
<code>interval</code>	<code>float</code>	<code>0.1</code>	-

**Returns:** `AsyncIterator[Status]`

## status

```
async def status(self, *, with_logs: 'bool' = False) -> 'Status'
```

Returns the status of the request (which can be one of the following: Queued, InProgress, Completed). If `with_logs` is True, logs will be included

for InProgress and Completed statuses.

Parameter	Type	Default	Description
<code>with_logs</code>	<code>bool</code>	<code>False</code>	-

**Returns:** `Status`

---

## Functions

### run

```
def run(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '', timeout: 'Op
```

Run an application with the given arguments (which will be JSON serialized).

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-
<code>path</code>	<code>str</code>	<code>''</code>	-
<code>timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Client-side HTTP timeout in seconds. Controls how long the HTTP client waits for a response. Defaults to the client's <code>default_timeout</code> .
<code>start_timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Server-side request timeout in seconds. Limits total time spent waiting before processing starts. Does not apply once the application begins processing.

Parameter	Type	Default	Description
<code>hint</code>	<code>str   None</code>	<code>None</code>	-
<code>headers</code>	<code>dict[str, str]</code>	<code>\{\}</code>	-

**Returns:** `dict[str, Any]`

## subscribe\_async

```
async def subscribe_async(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' :
```

Subscribe to an application and wait for the result.

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-
<code>path</code>	<code>str</code>	<code>''</code>	-
<code>hint</code>	<code>str   None</code>	<code>None</code>	-
<code>with_logs</code>	<code>bool</code>	<code>False</code>	-
<code>on_enqueue</code>	<code>Optional[Callable[str, NoneType]]</code>	<code>None</code>	-
<code>on_queue_update</code>	<code>Optional[Callable[Status, NoneType]]</code>	<code>None</code>	-
<code>priority</code>	<code>Optional[Literal[normal, low]]</code>	<code>None</code>	-
<code>headers</code>	<code>dict[str, str]</code>	<code>\{\}</code>	-
<code>start_timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Server-side request timeout in seconds. Limits total time spent waiting

Parameter	Type	Default	Description
			before processing starts (includes queue wait, retries, and routing). Does not apply once the application begins processing.
<code>client_timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Client-side total timeout in seconds. Limits the total time spent waiting for the entire request to complete (including queue wait and processing). If not set, waits indefinitely.

**Returns:** `dict[str, Any]`

# subscribe

```
def subscribe(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '', hint:
```

Subscribe to an application and wait for the result.

Parameter	Type	Default	Description
application	str	-	-
arguments	dict[str, Any]	-	-
path	str	''	-
hint	str   None	None	-
with_logs	bool	False	-
on_enqueue	Optional[Callable[str, NoneType]]	None	-
on_queue_update	Optional[Callable[Status, NoneType]]	None	-
priority	Optional[Literal[normal, low]]	None	-
headers	dict[str, str]	\{\}	-
start_timeout	int   float   NoneType	None	Server-side request timeout in seconds. Limits total time spent waiting before processing starts (includes queue wait, retries, and routing).

Parameter	Type	Default	Description
			Does not apply once the application begins processing.
<code>client_timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Client-side total timeout in seconds. Limits the total time spent waiting for the entire request to complete (including queue wait and processing). If not set, waits indefinitely.

**Returns:** `dict[str, Any]`

## submit

```
def submit(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '', hint: 'str')
```

Submit an application with the given arguments (which will be JSON serialized).

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-
<code>path</code>	<code>str</code>	<code>''</code>	-
<code>hint</code>	<code>str   None</code>	<code>None</code>	-
<code>webhook_url</code>	<code>str   None</code>	<code>None</code>	-
<code>priority</code>	<code>Optional[Literal[normal, low]]</code>	<code>None</code>	-
<code>headers</code>	<code>dict[str, str]</code>	<code>\{\}</code>	-
<code>start_timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Server-side request timeout in seconds. Limits total time spent waiting before processing starts (includes queue wait, retries, and routing). Does not apply once the application begins processing.

**Returns:** `SyncRequestHandle`

## stream

```
def stream(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '/stream', timeout: 'float' = 10, hint: 'str | None' = None, headers: 'dict[str, str]' = {}, start_timeout: 'int | float | NoneType' = None) -> 'SyncRequestHandle'
```

Stream the output of an application with the given arguments (which will be JSON serialized). This is only supported at a few select applications at the moment, so be sure to first consult with the documentation of individual applications to see if this is supported.



The function will iterate over each event that is streamed from the server.

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-
<code>path</code>	<code>str</code>	<code> '/stream' </code>	-
<code>timeout</code>	<code>float   None</code>	<code>None</code>	-

**Returns:** `Iterator[dict[str, Any]]`

## run\_async

```
async def run_async(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '',
```

Run an application with the given arguments (which will be JSON serialized). The path parameter can be used to specify a subpath when applicable. This method will return the result of the inference call directly.

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-
<code>path</code>	<code>str</code>	<code>''</code>	-
<code>timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Client-side HTTP timeout in seconds. Controls how long the HTTP client waits for a response. Defaults to the client's default_timeout.
<code>start_timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Server-side request timeout in seconds. Limits total time spent waiting before

Parameter	Type	Default	Description
			processing starts. Does not apply once the application begins processing.
<code>hint</code>	<code>str   None</code>	<code>None</code>	-
<code>headers</code>	<code>dict[str, str]</code>	<code>\{\}</code>	-

**Returns:** `dict[str, Any]`

## submit\_async

```
async def submit_async(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '
```

Submit an application with the given arguments (which will be JSON serialized). The path parameter can be used to specify a subpath when applicable. This method will return a handle to the request that can be used to check the status and retrieve the result of the inference call when it is done.

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-
<code>path</code>	<code>str</code>	<code>''</code>	-
<code>hint</code>	<code>str   None</code>	<code>None</code>	-
<code>webhook_url</code>	<code>str   None</code>	<code>None</code>	-
<code>priority</code>	<code>Optional[Literal[normal, low]]</code>	<code>None</code>	-
<code>headers</code>	<code>dict[str, str]</code>	<code>\{\}</code>	-
<code>start_timeout</code>	<code>int   float   NoneType</code>	<code>None</code>	Server-side request timeout in seconds. Limits

Parameter	Type	Default	Description
			total time spent waiting before processing starts (includes queue wait, retries, and routing). Does not apply once the application begins processing.

**Returns:** `AsyncRequestHandle`

## stream\_async

```
def stream_async(self, application: 'str', arguments: 'AnyJSON', *, path: 'str' = '/stream')
```

Stream the output of an application with the given arguments (which will be JSON serialized). This is only supported at a few select applications at the moment, so be sure to first consult with the documentation of individual applications to see if this is supported.

The function will iterate over each event that is streamed from the server.

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-
<code>path</code>	<code>str</code>	<code> '/stream' </code>	-
<code>timeout</code>	<code>float   None</code>	<code>None</code>	-

**Returns:** `AsyncIterator[dict[str, Any]]`

## realtime

```
def realtime(self, application: 'str', *, use_jwt: 'bool' = True, path: 'str' = '/realtime')
```

Parameter	Type	Default	Description
application	str	-	-
use_jwt	bool	True	-
path	str	'/realtime'	-
max_buffering	int   None	None	-
token_expiration	int	120	-
encode_message	Optional[Callable[Any, bytes]]	None	-
decode_message	Optional[Callable[bytes, Any]]	None	-

**Returns:** Iterator[RealtimeConnection]

## realtime\_async

```
def realtime_async(self, application: 'str', *, use_jwt: 'bool' = True, path: 'str' = '/realtime')
```

Parameter	Type	Default	Description
application	str	-	-
use_jwt	bool	True	-
path	str	'/realtime'	-
max_buffering	int   None	None	-
token_expiration	int	120	-
encode_message	Optional[Callable[Any, bytes]]	None	-
decode_message	Optional[Callable[bytes, Any]]	None	-

**Returns:** AsyncIterator[AsyncRealtimeConnection]

## cancel

```
def cancel(self, application: 'str', request_id: 'str') -> 'None'
```

Parameter	Type	Default	Description
application	str	-	-
request_id	str	-	-

**Returns:** NoneType

## cancel\_async

```
async def cancel_async(self, application: 'str', request_id: 'str') -> 'None'
```

Parameter	Type	Default	Description
application	str	-	-
request_id	str	-	-

**Returns:** NoneType

## status

```
def status(self, application: 'str', request_id: 'str', *, with_logs: 'bool' = False) ->
```

Parameter	Type	Default	Description
application	str	-	-
request_id	str	-	-
with_logs	bool	False	-

**Returns:** `Status`

## status\_async

```
async def status_async(self, application: 'str', request_id: 'str', *, with_logs: 'bool' -
```

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>request_id</code>	<code>str</code>	-	-
<code>with_logs</code>	<code>bool</code>	<code>False</code>	-

**Returns:** `Status`

## result

```
def result(self, application: 'str', request_id: 'str') -> 'AnyJSON'
```

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>request_id</code>	<code>str</code>	-	-

**Returns:** `dict[str, Any]`

## result\_async

```
async def result_async(self, application: 'str', request_id: 'str') -> 'AnyJSON'
```

Parameter	Type	Default	Description
<code>application</code>	<code>str</code>	-	-
<code>request_id</code>	<code>str</code>	-	-

**Returns:** `dict[str, Any]`

## encode

```
def encode(data: 'str | bytes', content_type: 'str') -> 'str'
```

Encode the given data blob to a data URL with the specified content type.

Parameter	Type	Default	Description
<code>data</code>	<code>str   bytes</code>	-	-
<code>content_type</code>	<code>str</code>	-	-

**Returns:** `str`

## encode\_file

```
def encode_file(path: 'os.PathLike') -> 'str'
```

Encode a file from the local filesystem to a data URL with the inferred content type.

Parameter	Type	Default	Description
<code>path</code>	<code>PathLike</code>	-	-

**Returns:** `str`

## encode\_image

```
def encode_image(image: 'Image.Image', format: 'str' = 'jpeg') -> 'str'
```

Encode a pillow image object to a data URL with the specified format.

Parameter	Type	Default	Description
<code>image</code>	<code>Image.Image</code>	-	-

Parameter	Type	Default	Description
<code>format</code>	<code>str</code>	<code>'jpeg'</code>	-

# Python Client

Source: <https://fal.ai/docs/api-reference/client-libraries/python/index>

API reference for fal-client Python package

The `fal-client` package provides a Python interface for calling fal AI models.

## Installation

```
pip install fal-client
```

## Quick Start

```
import fal_client

result = fal_client.subscribe(
 "fal-ai/flux/dev",
 arguments={
 "prompt": "a cat wearing a hat",
 "image_size": "landscape_4_3"
 },
 with_logs=True,
 on_queue_update=lambda status: print(f"Status: {status}")
)

print(result["images"][0]["url"])
```



# API Reference

The following pages contain the auto-generated API reference for all public classes and functions in the `fal-client` package.

## Swift Client

Source: <https://fal.ai/docs/api-reference/client-libraries/swift/index>

fal client library for iOS, macOS, tvOS, and watchOS

The `fal-swift` package provides a native Swift interface for calling fal AI models on Apple platforms.

## Installation

Add the package via Swift Package Manager:

```
.package(url: "https://github.com/fal-ai/fal-swift.git", from: "0.5.6")
```

# Quick Start

```
import FalClient

let result = try await fal.subscribe(
 to: "fal-ai/flux/dev",
 input: [
 "prompt": "a cat",
 "seed": 6252023,
 "image_size": "landscape_4_3",
 "num_images": 4
],
 includeLogs: true
) { update in
 if case let .inProgress(logs) = update {
 print(logs)
 }
}
```

## Supported Platforms

- iOS 16+
- macOS 13+
- tvOS 16+
- watchOS 9+

## API Reference

Full API documentation on Swift Package Index Source code and examples

## Reference

Source: <https://fal.ai/docs/api-reference/index>

Complete API and SDK reference documentation for fal

Complete reference documentation for all fal SDKs and tools.

Libraries for calling fal AI models from your applications  
Command-line interface for deploying and managing fal applications  
Tools for building and deploying serverless AI applications

# Overview

Section	What it's for	When to use
<b>Client Libraries</b>	Call fal AI models and get results	You want to use fal's AI models in your app
<b>CLI Reference</b>	Deploy, manage, and monitor fal applications	You're working with fal from the command line
<b>Python SDK</b>	Build and deploy custom AI applications	You're deploying your own models on fal

# Authentication

Source: <https://fal.ai/docs/api-reference/platform-apis/authentication>

Platform APIs require API keys for secure access to your user or team's data.

# Generating API Keys

Navigate to the dashboard keys page and generate a key from the UI: [fal.ai/dashboard/keys](https://fal.ai/dashboard/keys)

# Scopes

Platform APIs may require different API key scopes.

[Learn more about key-based authentication and scopes.](#)

Most Platform APIs accept **API scope** keys. This scope is suitable for most use cases including model discovery, pricing, and analytics. Some Platform APIs require **Admin scope** keys for access to sensitive data. Check the specific Platform API documentation to see which scope is required. If you're unsure, start with an API scope key. You can always generate an Admin scope key later if needed. API keys should be kept secure and never exposed in client-side code or public repositories.

## Authentication Format

Include your API key in the `Authorization` header with the `Key` prefix:

```
Authorization: Key YOUR_API_KEY
```

For endpoints requiring Admin scope:

```
Authorization: Key YOUR_ADMIN_API_KEY
```

## Usage Examples

### cURL

Using an API scope key for model listing:

```
curl -X GET "https://api.fal.ai/v1/models?limit=10" \
-H "Authorization: Key YOUR_API_KEY"
```

Using an Admin scope key for usage data:

```
curl -X GET "https://api.fal.ai/v1/models/usage" \
-H "Authorization: Key YOUR_ADMIN_API_KEY"
```

### Python

Using an API scope key:

```
import requests

headers = {
 "Authorization": "Key YOUR_API_KEY"
}

response = requests.get(
 "https://api.fal.ai/v1/models",
 headers=headers,
 params={"limit": 10}
)

print(response.json())
```

## JavaScript

Using an API scope key:

```
const response = await fetch('https://api.fal.ai/v1/models?limit=10', {
 headers: {
 'Authorization': 'Key YOUR_API_KEY'
 }
});

const data = await response.json();
console.log(data);
```

## Best Practices

- Store API keys in environment variables
- Use the minimum required scope for your use case
- Rotate keys regularly
- Keep Admin API keys secure and never expose them client-side

# Troubleshooting

## 401 Unauthorized

- Verify your API key is correct
- Ensure the `Authorization` header includes the `Key` prefix
- Check that your API key hasn't been revoked

## 403 Forbidden

Your API key may not have the required scope for the endpoint. Check the endpoint documentation to determine if it requires an Admin scope key, and generate one from the dashboard if needed.

# Platform APIs for Accounts

Source: <https://fal.ai/docs/api-reference/platform-apis/for-accounts>

Programmatic access to account billing information, FOCUS-compliant cost reports, and model access controls

The **fal Platform APIs** provide programmatic access to account-level billing, cost reporting, and governance, including:

- **Billing information** - Retrieve account billing details and credit balances
- **FOCUS reports** - Download FinOps FOCUS-compliant billing reports for cost analysis and interoperability
- **Model access controls** - Export the resolved UI and API access state for each model in your organization

## Available Operations

The Platform APIs provide the following endpoints for account billing, reporting, and governance:

Retrieve billing information and credit balances for your account Download FOCUS-compliant billing reports from invoice or usage estimate data Export the resolved UI and API access state for each model in your organization FOCUS reports and model access controls are available to enterprise customers with the corresponding features enabled. Contact your account team or [support@fal.ai](mailto:support@fal.ai) to request access.

# Platform APIs for Compute

Source: <https://fal.ai/docs/api-reference/platform-apis/for-compute>

Programmatic access to compute instance management, lifecycle control, and monitoring

The **fal Platform APIs** provide programmatic access to platform management features for Compute, including:

- **Instance management** - Create, list, and delete compute instances
- **Instance details** - Retrieve detailed information about specific instances

## Available Operations

The Platform APIs provide the following endpoints for managing Compute instances:

List all compute instances with their current status and configuration Retrieve detailed information about a specific compute instance Create and provision a new compute instance Terminate and remove a compute instance These APIs are for **platform management** of Compute instances. For getting started with Compute, see the [Compute documentation](/docs/compute).

# Platform APIs for Keys

Source: <https://fal.ai/docs/api-reference/platform-apis/for-keys>

Programmatic access to API key management, creation, and deletion

The **fal Platform APIs** provide programmatic access to API key management, including:

- **List keys** - Retrieve all API keys associated with your account
- **Create keys** - Generate new API keys with custom aliases
- **Delete keys** - Revoke and remove existing API keys

## Available Operations

The Platform APIs provide the following endpoints for managing API keys:

Retrieve all API keys with pagination support  
Generate a new API key with a friendly alias  
Permanently revoke and delete an API key  
These APIs require **\*\*admin API key\*\*** authentication. For more information on authentication, see the [Authentication](/docs/reference/platform-apis/authentication) documentation.

## Platform APIs for Models

Source: <https://fal.ai/docs/api-reference/platform-apis/for-models>

Programmatic access to model metadata, pricing, usage tracking, and analytics

The **fal Platform APIs** provide programmatic access to platform management features for Model APIs, including:

- **Model metadata** - Search and discover available model endpoints with detailed information
- **Pricing information** - Retrieve real-time pricing and estimate costs
- **Usage tracking** - Access detailed usage line items with unit quantities and prices
- **Analytics** - Query time-bucketed metrics for request counts, success/error rates, and latency

## Available Operations

The Platform APIs provide the following endpoints for managing Model APIs:

Search and discover available model endpoints with metadata, categories, and capabilities  
Retrieve real-time pricing information for models  
Estimate costs for planned operations



Access detailed usage line items with unit quantities and prices Query time-bucketed metrics for requests, success rates, and latency List recent requests for a specific model endpoint with filters and pagination Delete IO payloads and CDN output files for a specific request These APIs are for **platform management** of Model APIs. For executing models and generating content, see the [Inference Methods](/docs/documentation/model-apis/inference) documentation.

# Platform APIs for Serverless

Source: <https://fal.ai/docs/api-reference/platform-apis/for-serverless>

Programmatic access to serverless apps metadata and analytics

## Available Operations

The Platform APIs provide the following endpoints for Serverless Apps:

### Files

List the contents of the root directory of your Serverless storage. List the contents of any nested directory by providing its path. Download any file from Serverless storage. Upload a file from a URL into Serverless storage. Upload a local file into Serverless storage.

### Requests

List recent requests for your serverless endpoints with filtering, sorting, and pagination.

### Logs

Query paginated logs with powerful label filters, time ranges, and search keywords. Stream live logs that match the provided filters using Server-Sent Events.

# Analytics

Query time-bucketed metrics across all inbound traffic to your apps, including request counts, success/error rates, and latency percentiles. Ideal for exporting to your own observability tools.

# Metrics

Read the current queue backlog for your serverless applications. Export app metrics (runners, queue size, concurrent requests, throughput, and latency) in Prometheus format for custom dashboards and monitoring.

# Platform APIs for Workflows

Source: <https://fal.ai/docs/api-reference/platform-apis/for-workflows>

Programmatic access to workflow metadata, listing, and details

The **fal Platform APIs** provide programmatic access to workflow management for the authenticated user, including:

- **List workflows** - Paginated list of your workflows with optional search and filtering
- **Get workflow details** - Retrieve a specific workflow by owner and name, including its full definition

# Available Operations

The Platform APIs provide the following endpoints for Workflows:

List workflows for the authenticated user with optional search and filtering by endpoint Get detailed information about a specific workflow, including its full definition

# Authentication

All Workflows endpoints require authentication. Include your API key in the `Authorization` header:

```
Authorization: Key YOUR_API_KEY
```

List workflows returns only workflows owned by the authenticated user. Get workflow details requires access to the workflow (e.g., it is yours or public).

These APIs are for **platform management** of Workflows (listing and reading). To run workflows, use the [Model APIs](/docs/model-apis) or workflow execution endpoints.

## Introduction to Platform APIs

Source: <https://fal.ai/docs/api-reference/platform-apis/index>

Platform APIs provide programmatic access to platform resources including model metadata, pricing information, usage tracking, and analytics.