Authenticate to the Cove API

> **About this sample**
>
> **Document type:** API authentication guide
> **Audience:** Merchant and agency developers
> **Purpose:** Describes how to generate API credentials, exchange them for OAuth 2.0 access tokens, and store both securely.
> **Note:** Company, product, and endpoint names are fictional. The technical content is representative of the published version.

When you make a request to the Cove API, you must provide an OAuth 2.0 *access token* in the header of the request. An access token is a set of short-lived credentials that signifies that Cove has given you permission to call the Cove APIs on a set of resources such as orders and delivery previews.

To get an access token, you first generate your API credentials on the Cove merchant console. You then exchange your API credentials for an access token by making a POST request to a Cove token endpoint.

Cove access tokens expire every 15 minutes. After your access token expires, you can request a new access token by using the same client permissions.

API credentials never expire and remain valid until you delete the credentials through the merchant console. Unlike access tokens, you do not need to periodically refresh API credentials. Create a single set of API credentials for each service component and use those credentials to generate access tokens as needed for your component's use case.

For a given generated `client_id` and `client_secret` pairing, you can generate multiple Cove access tokens. There is no limit on the number of generated access tokens and you do not need to wait for a previous token to expire before generating a new token. However, making multiple requests with the same `client_id` in rapid succession can cause throttling of requests, increased latency, and availability risks.

Granting API credentials full access can lead to an escalation of privilege risk. Instead, create a new set of API credentials for each service component. For example, if your service component manages Cove orders, create a set of API credentials with custom permissions for viewing and editing order data.

In addition, create new API credentials for any new service component you introduce, with permissions limited to the scope of the service component. By creating separate credentials for each component, you help prevent a compromised service from affecting your entire integration.

Some mutations and queries use a shopper's *Cove identity token*. With an identity token, you can get data related to the shopper, for example the shopper's location to help provide a more accurate delivery preview.

Step 1: Generate API credentials

You generate your API credentials by using the Cove merchant console.

To generate API credentials by using the merchant console

1. Sign in to the Cove merchant console as an owner/admin.
2. On the left, choose **Settings**.
3. Under **Settings**, choose **API credentials**.
4. Choose **Generate credentials**.
5. For **Credentials name**, enter a name that helps you identify the purpose of the credentials.
6. Choose the permissions that you want the API credentials to have. You can choose **Full Access** or **Custom**. We recommend that you adhere to the principle of least privilege and choose granular access depending on the functionality that you are building using the API credentials.
    - **Full access**: Allows edit and view access to all the listed permissions. Choosing this option automatically chooses all APIs in the list and automatically includes access to any future Cove APIs.
    - **Custom**: Allows you to select permissions from a list. If you choose this option, you must manually choose the specific APIs that you need.
7. Choose **Generate**.
   Cove generates your API credentials and then takes you to a page where you can download a file that contains the credentials. The file contains a client ID, client secret, target ID, and a list of permissions that the credentials have access to.
8. Download the credentials file to your computer.

    > **Important:** You **must** download the credentials file and save it securely. If you navigate away without downloading the credentials, you will not be able to download the credentials later.

9. Locate and open the downloaded credentials file on your computer.
   The file has the name that you chose for the credentials.

Step 2: Use API credentials to get an access token

After you get your API credentials, you use the API credentials to request a Cove access token.

> **Important**
> Cove access tokens allow you to call the Cove API. This is different from a shopper's *Cove identity token*. Cove generates identity tokens when a shopper signs in to a merchant store.

Cove access tokens cannot be revoked. You can use an access token for multiple API calls until the token expires. When you request a token from the `/token` endpoint, the call response includes an `expires_in` property that states the number of seconds the token remains valid. To help avoid throttling errors, continue to use the token until the token expires.

Deleting API credentials prevents components from generating access tokens, but doesn't revoke issued access tokens. For details, see Delete API credentials when no longer in use.

To get an access token for the Cove API

1. Make an HTTPS POST request to `https://api.cove.example.com/token` with the following fields in the body of the request.

   Request fields

Field	Description	Required?
`client_id`	The client ID of the API credentials that you downloaded in Generate API credentials.	Yes
`client_secret`	The client secret of the API credentials that you downloaded in Generate API credentials.	Yes
`grant_type`	OAuth 2.0 grant type. Use `client_credentials`.	Yes

   Example request

   In the following example request, replace the following placeholders:

   - `EXAMPLE_CLIENT_ID` with the `client_id` from the API credentials that you downloaded.
   - `EXAMPLE_CLIENT_SECRET` with the `client_secret` from the API credentials that you downloaded.

```
POST /token HTTP/1.1
Host: api.cove.example.com
Content-Type: application/x-www-form-urlencoded
x-api-version: $api_version

client_id=EXAMPLE_CLIENT_ID&client_secret=EXAMPLE_CLIENT_SECRET&grant_type=client_crede
ntials
```

```
curl --location --request POST 'https://api.cove.example.com/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--header 'x-api-version: $api_version' \
--data-urlencode 'client_id=EXAMPLE_CLIENT_ID' \
--data-urlencode 'client_secret=EXAMPLE_CLIENT_SECRET' \
--data-urlencode 'grant_type=client_credentials'
```

```
import requests
url = "https://api.cove.example.com/token"
payload='client_id=EXAMPLE_CLIENT_ID&client_secret=EXAMPLE_CLIENT_SECRET&grant_type=cli
ent_credentials'
headers = {
  'Content-Type': 'application/x-www-form-urlencoded',
  'x-api-version': '$api_version'
}
response = requests.request("POST", url, headers=headers, data=payload)
print(response.text)
```

```
const url = "https://api.cove.example.com/token";
const headers = {
  "Content-Type": "application/x-www-form-urlencoded",
  "x-api-version": "$api_version"
};
const
payload="client_id=EXAMPLE_CLIENT_ID&client_secret=EXAMPLE_CLIENT_SECRET&grant_type=cli
ent_credentials";

fetch(url, {
  method: "POST",
  headers: headers,
  body: payload
}).then(response => {
    return response.json();
}).then(json => {
  console.log(json);
});
```

2. Get the access token from the response, which contains the following fields.

Example successful response

```
{
    "access_token": "EXAMPLE_COVE_ACCESS_TOKEN",
    "expires_in": 885
}
```

Successful response fields

Field	Description
`access_token`	Token that you use to access the Cove API.
`expires_in`	Time until the token expires, in seconds.

Example failed response (HTTP 400)

```
{
    "message": "Content type is null or invalid. Ensure content type is: application/x-
www-form-urlencoded",
    "code": "InvalidContentType",
    "type": "ValidationError"
}
```

Failed response fields

Field	Description
`message`	Description of the cause of the error.
`code`	(Only present for HTTP 400 `InvalidParameterException` responses.) Code that further describes the cause of the HTTP 400 error. For a list of possible codes, see the table in the following section.
`type`	The exception type thrown by the service. Examples: `ValidationError` or `AccessDeniedError`.

Error codes

HTTP status code	Error type	Error code	Description
400	`ValidationError`	`InvalidContentType`	The `Content-Type` field in the request is missing or invalid. The `Content-Type` field must be `application/x-www-form-urlencoded`.
400	`ValidationError`	`NonDeserializableContent`	The request payload isn't in a format that the server can interpret.
400	`ValidationError`	`InvalidClientId`	The request payload doesn't contain a `client_id` field, or the specified `client_id` is incomplete, malformed, or invalid.
400	`ValidationError`	`InvalidClientSecret`	The request payload doesn't contain a `client_secret` field, or the specified `client_secret` is incomplete, malformed, or invalid.
400	`ValidationError`	`InvalidGrantType`	The request payload doesn't contain a `grant_type` field, or the specified `grant_type` is incomplete, malformed, or invalid. The `grant_type` must be `client_credentials`.
401	`AccessDeniedError`	N/A	The requested payload doesn't have permission to receive an access token.
429	`ThrottlingError`	N/A	The request was throttled by the service. Requests are throttled after a limit of 12 requests per second per `client_id` is reached.
500	`InternalServerError`	N/A	An internal error occurred. Try again.

Now that you have a Cove access token, you can call the Cove API.

Step 3: Cache API credentials

We recommend that any service component or process that fetches a Cove access token caches the token for as long as possible before fetching a new token. By caching access tokens and refreshing only when necessary, you can improve your system performance and reduce latency.

The response you receive when calling the `/token` endpoint to retrieve a new access token contains the field `expires_in` to indicate the number of seconds until the token expires. You can use the `expires_in` field to determine how long to cache the access token.

Consider the following approaches for caching, depending on your application design and access token usage.

In-memory caching

To improve overall system performance and reduce retrieval time, you can use in-memory caching to temporarily store access tokens within the fast-access memory of the service. Because the token is available within the service, in-memory caching reduces both latency and the number of network calls. In-memory caching also simplifies your implementation, as each service component manages its own token lifecycle.

Distributed caching

For accessing data quickly across multiple servers, you can use distributed caching. Distributed caching stores access tokens in a central location where all service components can retrieve them. Distributed caching reduces system load and the frequency of token generation requests, balancing local and centralized approaches. Common systems for distributed caching include Redis and Memcached, whether self-hosted or offered as a managed service by your cloud provider.

Asynchronous caching

If you need each service component to use a unique permission set, you can use asynchronous caching through a lightweight thread or service. The service manages token refreshing and caching, returning tokens when called by your service components.

Centralized token storage

To maximize control and consistency, you can use centralized token storage to provide service components with access tokens as needed. Managed key-value databases, relational databases, and object storage services are all examples of centralized services that can handle token storage.

Step 4: Encrypt API credentials at rest and in transit

To avoid exposing sensitive information to any entity that can inspect your application or components, encrypt stored `client_secret` values and Cove access tokens. Avoid hard-coding a `client_secret` or access token into any service component and never store a `client_secret` or access token as plain text.

In addition, encrypt any `client_secret` or access token in transit when interacting with public networks. For example, you can use an encryption protocol like Transport Layer Security. Encrypting these values in transit helps you prevent susceptibility to man-in-the-middle attacks.

Most cloud providers offer a managed secrets service for storing credentials and other secrets, and a key management service for encryption at rest. These approaches aren't comprehensive. Consider the unique security needs of your site and integrations.

Delete API credentials when no longer in use

Before you delete a set of API credentials, verify that there are no service components using the credentials by checking the usage of the associated `client_id`. To check the latest usage of a `client_id` for access token generation, contact support through the Cove merchant console. Include the `client_id` in your support request.

After you delete a set of API credentials, calling the `/token` endpoint using the associated `client_id` and `client_secret` generates an `AccessDeniedError`.