readme.md for @apiclient.xyz/cloudflare

> □□ A modern, elegant TypeScript client for the Cloudflare API with clean manager-based architecture

npm version License: MIT

Stop fighting with the Cloudflare API. This library gives you an intuitive, type-safe way to manage zones, DNS records, and Workers—all with clean, predictable method names and excellent IDE support.

□ Why This Library?

- □□ **Logical & Consistent**: Manager-based architecture with predictable method naming (`listZones()` , `listRecords()` , `listWorkers()`)
- □□ **Strongly Typed**: Full TypeScript support with excellent autocompletion
- □□ **Framework Integration**: Built-in `IConvenientDnsProvider` adapter for third-party modules
- ⚡ **Modern**: Uses async/await, ES modules, and the official Cloudflare SDK
- □□ **Reliable**: Comprehensive error handling and detailed logging
- □□ **Zero Config**: Works out of the box with just your API token

□□ Quick Start

```
npm install @apiclient.xyz/cloudflare
```

```
import { CloudflareAccount } from '@apiclient.xyz/cloudflare';

const cf = new CloudflareAccount('your-api-token');
```

```
// Manage DNS records with ease
await cf.recordManager.createRecord('api.example.com', 'A', '192.0.2.1');
await cf.recordManager.updateRecord('api.example.com', 'A', '203.0.113.1');

// Work with zones
const zones = await cf.zoneManager.listZones();
await cf.zoneManager.purgeZone('example.com');

// Deploy workers
const worker = await cf.workerManager.createWorker('my-api', workerScript);
await worker.setRoutes([{ zoneName: 'example.com', pattern: 'api.example.com/*' }]);
```

□□ Core Concepts

Managers: Your Gateway to Cloudflare

The library is organized around **managers**—specialized classes that handle specific Cloudflare resources:

- `recordManager` - DNS record operations
- `zoneManager` - Zone (domain) management
- `workerManager` - Cloudflare Workers deployment

Each manager provides consistent, predictable methods that feel natural to use.

□□ Complete Guide

Account Setup

```
import { CloudflareAccount } from '@apiclient.xyz/cloudflare';

const cf = new CloudflareAccount(process.env.CLOUDFLARE_API_TOKEN);

// If you have multiple accounts, select one
await cf.preselectAccountByName('Production Account');
```

□□ Zone Management

Zones are your domains in Cloudflare.

```
// List all zones
const zones = await cf.zoneManager.listZones();

// Find a specific zone
const zone = await cf.zoneManager.getZoneByName('example.com');

// Get zone ID for API calls
const zoneId = await cf.zoneManager.getZoneId('example.com');

// Create a new zone
await cf.zoneManager.createZone('newdomain.com');

// Purge entire zone cache
await cf.zoneManager.purgeZone('example.com');

// Work with zone objects
if (await zone.isActive()) {
  await zone.purgeCache();
  await zone.purgeUrls(['https://example.com/css/app.css']);

  // Development mode (bypasses cache for 3 hours)
  await zone.enableDevelopmentMode();
  await zone.disableDevelopmentMode();
}
```

□□ DNS Records

The `recordManager` gives you complete control over DNS records.

```
// List all records for a domain
const records = await cf.recordManager.listRecords('example.com');

// Create different record types
await cf.recordManager.createRecord('www.example.com', 'CNAME', 'example.com', 3600);
```

```
await cf.recordManager.createRecord('api.example.com', 'A', '192.0.2.1', 300);
await cf.recordManager.createRecord('_dmarc.example.com', 'TXT', 'v=DMARC1; p=reject', 3600);

// Get a specific record
const record = await cf.recordManager.getRecord('api.example.com', 'A');
console.log(record.content); // "192.0.2.1"

// Update or create (upsert behavior)
await cf.recordManager.updateRecord('api.example.com', 'A', '203.0.113.1');

// Delete a record
await cf.recordManager.deleteRecord('old.example.com', 'A');

// Clean up all records of a type (useful for wildcard cleanup)
await cf.recordManager.cleanRecords('example.com', 'TXT');
```

⚙ Cloudflare Workers

Deploy and manage serverless functions at the edge.

```
// Worker code
const workerScript = `
addEventListener('fetch', event => {
  event.respondWith(handleRequest(event.request));
});

async function handleRequest(request) {
  return new Response('Hello from the edge!', {
    headers: { 'content-type': 'text/plain' }
  });
}
`;

// Create/update a worker
const worker = await cf.workerManager.createWorker('my-api-worker', workerScript);

// List all workers
const workers = await cf.workerManager.listWorkers();
```

```
// Get existing worker
const existingWorker = await cf.workerManager.getWorker('my-api-worker');

// Set up routes
await worker.setRoutes([
  { zoneName: 'example.com', pattern: 'api.example.com/*' },
  { zoneName: 'example.com', pattern: 'example.com/api/*' }
]);

// Check worker routes
await worker.listRoutes();
console.log(worker.routes); // Array of routes

// Update worker script
await worker.updateScript(updatedWorkerScript);

// Delete worker
await worker.delete();
// or
await cf.workerManager.deleteWorker('my-api-worker');
```

□□ ACME DNS Challenges (Let's Encrypt)

Use the `ConvenientDnsProvider` adapter for certificate automation.

```
const dnsProvider = cf.getConvenientDnsProvider();

// Set DNS challenge
await dnsProvider.acmeSetDnsChallenge({
  hostName: '_acme-challenge.example.com',
  challenge: 'your-validation-token'
});

// Remove challenge after validation
await dnsProvider.acmeRemoveDnsChallenge({
  hostName: '_acme-challenge.example.com',
  challenge: 'your-validation-token'
});
```

□□ Third-Party Module Integration

If you're using a framework or library that expects `IConvenientDnsProvider`, use the adapter:

```
import { CloudflareAccount } from '@apiclient.xyz/cloudflare';

const cf = new CloudflareAccount(process.env.CLOUDFLARE_API_TOKEN);
const dnsProvider = cf.getConvenientDnsProvider();

// Now pass this to any library expecting IConvenientDnsProvider
await someFramework.setDnsProvider(dnsProvider);
```

□□ Real-World Example

Here's a complete example showing a typical deployment workflow:

```
import { CloudflareAccount } from '@apiclient.xyz/cloudflare';

async function deployApplication() {
  const cf = new CloudflareAccount(process.env.CLOUDFLARE_API_TOKEN);
  await cf.preselectAccountByName('Production');

  // 1. Set up DNS
  console.log('□□Configuring DNS...');
  await cf.recordManager.createRecord('app.example.com', 'A', '192.0.2.10');
  await cf.recordManager.createRecord('www.example.com', 'CNAME', 'example.com');

  // 2. Deploy API worker
  console.log('⚙□ Deploying API worker...');
  const apiWorker = await cf.workerManager.createWorker('api-handler', `
    addEventListener('fetch', event => {
      const response = new Response(JSON.stringify({ status: 'ok' }), {
        headers: { 'content-type': 'application/json' }
      });
      event.respondWith(response);
    });
  `);
```

```
  await apiWorker.setRoutes([
    { zoneName: 'example.com', pattern: 'api.example.com/*' }
  ]);

  // 3. Configure security headers worker
  console.log('□□□Setting up security...');
  const securityWorker = await cf.workerManager.createWorker('security-headers', `
    addEventListener('fetch', event => {
      event.respondWith(addSecurityHeaders(event.request));
    });

    async function addSecurityHeaders(request) {
      const response = await fetch(request);
      const newHeaders = new Headers(response.headers);
      newHeaders.set('X-Frame-Options', 'DENY');
      newHeaders.set('X-Content-Type-Options', 'nosniff');

      return new Response(response.body, {
        status: response.status,
        headers: newHeaders
      });
    }
  `);

  await securityWorker.setRoutes([
    { zoneName: 'example.com', pattern: 'example.com/*' }
  ]);

  // 4. Verify and purge cache
  console.log('□□Purging cache...');
  const zone = await cf.zoneManager.getZoneByName('example.com');
  await zone.purgeCache();

  console.log('□ Deployment complete!');
}

deployApplication().catch(console.error);
```

API Reference

CloudflareAccount

Main entry point for all operations.

```
class CloudflareAccount {
  constructor(apiToken: string)

  // Managers
  readonly recordManager: RecordManager
  readonly zoneManager: ZoneManager
  readonly workerManager: WorkerManager

  // Methods
  async preselectAccountByName(name: string): Promise<void>
  getConvenientDnsProvider(): ConvenientDnsProvider

  // Official Cloudflare SDK client (for advanced use)
  readonly apiAccount: Cloudflare
}
```

RecordManager

DNS record management with clean, predictable methods.

```
class RecordManager {
  async listRecords(domain: string): Promise<CloudflareRecord[]>
  async getRecord(domain: string, type: string): Promise<CloudflareRecord | undefined>
  async createRecord(domain: string, type: string, content: string, ttl?: number):
Promise<CloudflareRecord>
  async updateRecord(domain: string, type: string, content: string, ttl?: number):
Promise<CloudflareRecord>
  async deleteRecord(domain: string, type: string): Promise<void>
  async cleanRecords(domain: string, type: string): Promise<void>
}
```

ZoneManager

Zone (domain) management operations.

```
class ZoneManager {
  async listZones(name?: string): Promise<CloudflareZone[]>
  async getZoneById(id: string): Promise<CloudflareZone | undefined>
  async getZoneByName(name: string): Promise<CloudflareZone | undefined>
  async getZoneId(domain: string): Promise<string>
  async createZone(name: string): Promise<CloudflareZone | undefined>
  async deleteZone(id: string): Promise<boolean>
  async purgeZone(domain: string): Promise<void>
}
```

CloudflareZone

Zone instance with convenience methods.

```
class CloudflareZone {
  readonly id: string
  readonly name: string
  readonly status: string

  async purgeCache(): Promise<void>
  async purgeUrls(urls: string[]): Promise<void>
  async isActive(): Promise<boolean>
  async isUsingCloudflareNameservers(): Promise<boolean>
  async enableDevelopmentMode(): Promise<void>
  async disableDevelopmentMode(): Promise<void>
}
```

WorkerManager

Cloudflare Workers deployment and management.

```
class WorkerManager {
  async listWorkers(): Promise<WorkerScript[]>
  async getWorker(name: string): Promise<CloudflareWorker | undefined>
```

```
  async createWorker(name: string, script: string): Promise<CloudflareWorker>
  async deleteWorker(name: string): Promise<boolean>
}
```

CloudflareWorker

Worker instance for route management and updates.

```
class CloudflareWorker {
  readonly id: string
  readonly script: string
  readonly routes: WorkerRoute[]

  async listRoutes(): Promise<void>
  async setRoutes(routes: WorkerRouteDefinition[]): Promise<void>
  async updateScript(script: string): Promise<CloudflareWorker>
  async delete(): Promise<boolean>
}

interface WorkerRouteDefinition {
  zoneName: string
  pattern: string
}
```

ConvenientDnsProvider

Adapter for third-party modules requiring `IConvenientDnsProvider`.

```
class ConvenientDnsProvider implements IConvenientDnsProvider {
  async createRecord(domain: string, type: string, content: string, ttl?: number):
Promise<any>
  async updateRecord(domain: string, type: string, content: string, ttl?: number):
Promise<any>
  async removeRecord(domain: string, type: string): Promise<any>
  async getRecord(domain: string, type: string): Promise<any | undefined>
  async listRecords(domain: string): Promise<any[]>
  async cleanRecord(domain: string, type: string): Promise<void>
  async isDomainSupported(domain: string): Promise<boolean>
```

```
  async acmeSetDnsChallenge(challenge: DnsChallenge): Promise<void>
  async acmeRemoveDnsChallenge(challenge: DnsChallenge): Promise<void>
}
```

□□ Migration from 6.x

Version 7.0 introduces a cleaner architecture while maintaining **full backward compatibility**.

What Changed

□ **New manager-based API** - Clean, logical organization □ **Consistent naming** - All list operations use `list*()` □ **IConvenientDnsProvider support** - Better third-party integration ⚠ **Deprecated `convenience` namespace** - Still works, but use managers instead

Migration Examples

```
// □ Old (deprecated but still works)
await cf.convenience.createRecord('example.com', 'A', '1.2.3.4');
await cf.convenience.listZones();
await cf.workerManager.listWorkerScripts();
await worker.getRoutes();

// □ New (recommended)
await cf.recordManager.createRecord('example.com', 'A', '1.2.3.4');
await cf.zoneManager.listZones();
await cf.workerManager.listWorkers();
await worker.listRoutes();
```

No Breaking Changes: Your existing code will continue to work. The old `convenience` namespace methods now show deprecation warnings in TypeScript with clear migration paths.

□□ Development

```
# Install dependencies
pnpm install
```

```
# Build
pnpm build

# Run tests
pnpm test
```

□□ TypeScript Configuration

This library is written in TypeScript and provides complete type definitions. No `@types` package needed!

```
import { CloudflareAccount, CloudflareZone, CloudflareRecord } from
'@apiclient.xyz/cloudflare';

// Full type inference and autocompletion
const cf = new CloudflareAccount(token);
const zones: CloudflareZone[] = await cf.zoneManager.listZones();
const record: CloudflareRecord = await cf.recordManager.createRecord(/* ... */);
```

□□ FAQ

Q: Can I use this with the official Cloudflare SDK? A: Yes! The library wraps the official SDK. You can access it directly via `cfAccount.apiAccount` for advanced operations.

Q: Does this support Cloudflare Workers KV, R2, D1? A: Currently focuses on zones, DNS, and Workers. Additional features coming in future releases.

Q: How do I handle errors? A: All methods throw descriptive errors. Wrap calls in try/catch for error handling.

```
try {
  await cf.recordManager.createRecord('example.com', 'A', 'invalid-ip');
} catch (error) {
  console.error('Failed to create record:', error.message);
}
```

Q: Is this production-ready? A: Absolutely! Used in production environments. Comprehensive error handling and logging built-in.

License and Legal Information

This repository contains open-source code that is licensed under the MIT License. A copy of the MIT License can be found in the license file within this repository.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH and are not included within the scope of the MIT license granted herein. Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines, and any usage must be approved in writing by Task Venture Capital GmbH.

Company Information

Task Venture Capital GmbH Registered at District court Bremen HRB 35230 HB, Germany

For any legal inquiries or if you require further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #7
Created 2026-03-28 10:48:11 UTC by foss.global Team
Updated 2026-03-28 12:13:22 UTC by foss.global Team