readme.md for @apiclient.xyz/elasticsearch

> □□ **Enterprise-grade TypeScript client for Elasticsearch** — Type-safe, observable, and production-ready

A modern, fully-typed Elasticsearch client built for scale. Features fluent configuration, distributed transactions, intelligent bulk operations, advanced logging with Kibana compatibility, and comprehensive observability out of the box.

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

□ Features

Feature	Description
□□ **Fluent Configuration**	Builder pattern for type-safe, environment-aware configuration
□□ **Document Management**	Full CRUD with sessions, snapshots, and lifecycle management
□□ **Query Builder**	Type-safe DSL for complex queries and aggregations
□□ **Bulk Operations**	High-throughput indexing with backpressure and adaptive batching
□□ **KV Store**	Distributed key-value storage with TTL, caching, and compression

Feature	Description
□□ **Transactions**	ACID-like semantics with optimistic concurrency control
□□ **Schema Management**	Index templates, migrations, and schema validation
□□ **Logging**	Kibana-compatible log destination with ILM policies
□□ **Plugin System**	Extensible architecture with built-in retry, caching, rate limiting
□□ **Observability**	Prometheus metrics, distributed tracing, structured logging
⚡ **Circuit Breaker**	Automatic failure detection and recovery

□□ Installation

```
npm install @apiclient.xyz/elasticsearch
# or
pnpm add @apiclient.xyz/elasticsearch
```

□□ Quick Start

Configuration

```
import { createConfig, ElasticsearchConnectionManager, LogLevel } from
'@apiclient.xyz/elasticsearch';

// Fluent configuration builder
const config = createConfig()
  .fromEnv()  // Load from ELASTICSEARCH_URL, ELASTICSEARCH_USERNAME, etc.
  .nodes('http://localhost:9200')
  .basicAuth('elastic', 'changeme')
  .timeout(30000)
  .retries(3)
  .compression(true)
  .poolSize(10, 2)
  .logLevel(LogLevel.INFO)
  .enableMetrics(true)
```

```
  .enableTracing(true, {
    serviceName: 'my-service',
    serviceVersion: '1.0.0',
  })
  .build();

// Initialize connection with health monitoring
const connection = ElasticsearchConnectionManager.getInstance(config);
await connection.initialize();

console.log(`Health: ${connection.getHealthStatus()}`);
console.log(`Circuit: ${connection.getCircuitState()}`);
```

Document Management

```
import { DocumentManager } from '@apiclient.xyz/elasticsearch';

interface Product {
  name: string;
  price: number;
  category: string;
  inStock: boolean;
}

const products = new DocumentManager<Product>({
  index: 'products',
  connectionManager: connection,
  autoCreateIndex: true,
});

await products.initialize();

// CRUD operations
await products.create('prod-001', {
  name: 'Premium Widget',
  price: 99.99,
  category: 'widgets',
  inStock: true,
```

```
});

const product = await products.get('prod-001');
await products.update('prod-001', { price: 89.99 });
await products.delete('prod-001');

// Session-based batch operations with auto-cleanup
const result = await products.session({ cleanupStale: true })
  .start()
  .upsert('prod-001', { name: 'Widget A', price: 10, category: 'widgets', inStock: true })
  .upsert('prod-002', { name: 'Widget B', price: 20, category: 'widgets', inStock: false })
  .commit();

console.log(`Processed ${result.successful} documents in ${result.took}ms`);

// Iterate over all documents
for await (const doc of products.iterate()) {
  console.log(doc._source.name);
}

// Create analytics snapshot
const snapshot = await products.snapshot(async (iterator, previous) => {
  let total = 0, count = 0;
  for await (const doc of iterator) {
    total += doc._source.price;
    count++;
  }
  return { averagePrice: total / count, productCount: count };
});
```

Query Builder

```
import { createQuery } from '@apiclient.xyz/elasticsearch';

// Fluent query building
const results = await createQuery<Product>('products')
  .match('name', 'widget', { fuzziness: 'AUTO' })
  .range('price', { gte: 10, lte: 100 })
```

```
  .term('inStock', true)
  .sort('price', 'asc')
  .size(20)
  .from(0)
  .highlight({ fields: { name: {} } })
  .aggregations(agg => agg
    .terms('categories', 'category')
    .avg('avgPrice', 'price')
  )
  .execute();

console.log(`Found ${results.hits.total} products`);
console.log('Categories:', results.aggregations?.categories);
```

Bulk Operations

```
import { createBulkIndexer } from '@apiclient.xyz/elasticsearch';

const indexer = createBulkIndexer({
  flushThreshold: 1000,
  flushIntervalMs: 5000,
  maxConcurrent: 3,
  enableBackpressure: true,
  onProgress: (progress) => {
    console.log(`Progress: ${progress.processed}/${progress.total}
(${progress.successRate}%)`);
  },
});

await indexer.initialize();

// Queue operations
for (const item of largeDataset) {
  await indexer.index('products', item.id, item);
}

// Wait for completion
const stats = await indexer.flush();
```

```
console.log(`Indexed ${stats.successful} documents, ${stats.failed} failures`);
```

Key-Value Store

```
import { createKVStore } from '@apiclient.xyz/elasticsearch';

const kv = createKVStore<string>({
  index: 'app-config',
  enableCache: true,
  cacheMaxSize: 10000,
  defaultTTL: 3600, // 1 hour
  enableCompression: true,
});

await kv.initialize();

// Basic operations
await kv.set('api-key', 'sk-secret-key', { ttl: 86400 });
const key = await kv.get('api-key');

// Batch operations
await kv.mset([
  { key: 'config:a', value: 'value-a' },
  { key: 'config:b', value: 'value-b' },
]);

const values = await kv.mget(['config:a', 'config:b']);

// Scan with pattern matching
const scan = await kv.scan({ pattern: 'config:*', limit: 100 });
```

Transactions

```
import { createTransactionManager } from '@apiclient.xyz/elasticsearch';

const txManager = createTransactionManager({
  defaultIsolationLevel: 'read_committed',
```

```
  defaultLockingStrategy: 'optimistic',
  conflictResolution: 'retry',
  maxConcurrentTransactions: 100,
});

await txManager.initialize();

// ACID-like transaction
const tx = await txManager.begin({ autoRollback: true });

try {
  const account1 = await tx.read('accounts', 'acc-001');
  const account2 = await tx.read('accounts', 'acc-002');

  await tx.update('accounts', 'acc-001', { balance: account1.balance - 100 });
  await tx.update('accounts', 'acc-002', { balance: account2.balance + 100 });

  tx.savepoint('after-transfer');

  await tx.commit();
} catch (error) {
  await tx.rollback();
}
```

Logging Destination

```
import { createLogDestination, chainEnrichers, addHostInfo, addTimestamp } from
'@apiclient.xyz/elasticsearch';

const logger = createLogDestination({
  indexPrefix: 'app-logs',
  dataStream: true,
  ilmPolicy: {
    name: 'logs-policy',
    phases: {
      hot: { maxAge: '7d', maxSize: '50gb' },
      warm: { minAge: '7d' },
      delete: { minAge: '30d' },
```

```
    },
  },
  enricher: chainEnrichers(addHostInfo(), addTimestamp()),
  sampling: { strategy: 'rate', rate: 0.1 }, // 10% sampling
});

await logger.initialize();

await logger.log({
  level: 'info',
  message: 'User authenticated',
  context: { userId: '12345', service: 'auth' },
});

// Batch logging
await logger.logBatch([
  { level: 'debug', message: 'Request received' },
  { level: 'info', message: 'Processing complete' },
]);
```

Schema Management

```
import { createSchemaManager } from '@apiclient.xyz/elasticsearch';

const schema = createSchemaManager({ enableMigrationHistory: true });

await schema.initialize();

// Create index with schema
await schema.createIndex('products', {
  mappings: {
    properties: {
      name: { type: 'text', analyzer: 'standard' },
      price: { type: 'float' },
      category: { type: 'keyword' },
      createdAt: { type: 'date' },
    },
  },
```

```
  settings: {
    numberOfShards: 3,
    numberOfReplicas: 1,
  },
});

// Run migrations
await schema.migrate('products', [
  {
    version: '1.0.1',
    description: 'Add tags field',
    up: async (client, index) => {
      await client.indices.putMapping({
        index,
        properties: { tags: { type: 'keyword' } },
      });
    },
  },
]);

// Create index template
await schema.createIndexTemplate('products-template', {
  indexPatterns: ['products-*'],
  mappings: { /* ... */ },
});
```

Plugin System

```
import {
  createPluginManager,
  createRetryPlugin,
  createCachePlugin,
  createRateLimitPlugin,
} from '@apiclient.xyz/elasticsearch';

const plugins = createPluginManager({ enableMetrics: true });

// Built-in plugins
```

```
plugins.register(createRetryPlugin({ maxRetries: 3, backoffMs: 1000 }));
plugins.register(createCachePlugin({ maxSize: 1000, ttlMs: 60000 }));
plugins.register(createRateLimitPlugin({ maxRequests: 100, windowMs: 1000 }));

// Custom plugin
plugins.register({
  name: 'custom-logger',
  version: '1.0.0',
  hooks: {
    beforeRequest: async (ctx) => {
      console.log(`Request: ${ctx.operation} on ${ctx.index}`);
    },
    afterResponse: async (ctx, response) => {
      console.log(`Response: ${response.statusCode}`);
    },
  },
});
```

□□ Architecture

```
@apiclient.xyz/elasticsearch
├── core/
│   ├── config/          # Fluent configuration builder
│   ├── connection/      # Connection manager, circuit breaker, health checks
│   ├── errors/          # Typed errors and retry policies
│   ├── observability/   # Logger, metrics (Prometheus), tracing (OpenTelemetry)
│   └── plugins/         # Plugin manager and built-in plugins
└── domain/
    ├── documents/       # DocumentManager, sessions, snapshots
    ├── query/           # QueryBuilder, AggregationBuilder
    ├── bulk/            # BulkIndexer with backpressure
    ├── kv/              # Distributed KV store
    ├── transactions/    # ACID-like transaction support
    ├── logging/         # Log destination with ILM
    └── schema/          # Schema management and migrations
```

□□ Observability

Prometheus Metrics

```
import { defaultMetricsCollector } from '@apiclient.xyz/elasticsearch';

// Export metrics in Prometheus format
const metrics = defaultMetricsCollector.export();

// Available metrics:
// - elasticsearch_requests_total{operation, index, status}
// - elasticsearch_request_duration_seconds{operation, index}
// - elasticsearch_errors_total{operation, index, error_type}
// - elasticsearch_circuit_breaker_state{state}
// - elasticsearch_connection_pool_size
// - elasticsearch_bulk_operations_total{type, status}
```

Distributed Tracing

```
import { defaultTracingProvider } from '@apiclient.xyz/elasticsearch';

// OpenTelemetry-compatible tracing
const span = defaultTracingProvider.startSpan('custom-operation');
span.setAttributes({ 'custom.attribute': 'value' });
// ... operation
span.end();
```

□□ Authentication Methods

```
// Basic auth
createConfig().basicAuth('username', 'password')

// API key
createConfig().apiKeyAuth('api-key-value')
```

```
// Bearer token
createConfig().bearerAuth('jwt-token')

// Elastic Cloud
createConfig().cloudAuth('cloud-id', { apiKey: 'key' })

// Certificate
createConfig().auth({
  type: 'certificate',
  certPath: '/path/to/cert.pem',
  keyPath: '/path/to/key.pem',
})
```

⚡ Performance Tips

1. **Use bulk operations** for batch inserts — `BulkIndexer` handles batching, retries, and backpressure
2. **Enable compression** for large payloads — reduces network overhead
3. **Configure connection pooling** — `poolSize(max, min)` for optimal concurrency
4. **Leverage caching** — `KVStore` has built-in LRU/LFU caching
5. **Use sessions** for document sync — automatic cleanup of stale documents
6. **Enable circuit breaker** — prevents cascade failures during outages

□□ Compatibility

- **Elasticsearch**: 8.x, 9.x
- **Kibana**: 8.x, 9.x (for log visualization)
- **Node.js**: 18+
- **TypeScript**: 5.0+

License and Legal Information

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 or third parties, 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 or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

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

For any legal inquiries or 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 10:48:14 UTC by foss.global Team
Updated 2026-03-28 12:13:25 UTC by foss.global Team